Switch Level Test Generation For CMOS Circuits

Kuen-Jong Lee

CEng Technical Report 91-22

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Engineering)

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA. 90089-2562

August 19, 1991

UNIVERSITY OF SOUTHERN CALIFORNIA
THE GRADUATE SCHOOL
UNIVERSITY PARK
LOS ANGELES, CALIFORNIA 90089-4015

This dissertation, written by

Kuen-Jong Lee

under the direction of h.is....... *Dissertation Committee, and approved by all its members, has been presented to and accepted by The Graduate School, in partial fulfillment of requirements for the degree of*

DOCTOR OF PHILOSOPHY

Barbara Solomon

Dean of Graduate Studies

Date August 9, 1991

DISSERTATION COMMITTEE

Chairperson

Charles Lamski

Dedication

To my wife and my parents.

Acknowledgements

I am grateful to Professor Melvin Breuer for his inspiration and guidance during my dissertation work. I would also like to thank him for providing me access to an excellent working environment. During my years at USC I benefited greatly from interacting with many colleagues and friends. In particular I would like to mention Jung-Cheun Lien, Rajesh Gupta, Rajiv Gupta, Charles Njinda, Rajagopalan Srinivasan and Amitava Majumdar. I also wish to thank Professor Sarma Sastry and Professor Charles Lanski who served on my dissertation committee.

I would like to acknowledge the financial support provided by the Defense Advanced Research Projects Agency through Contract Numbers N00014-87-K-0861 monitored by the Office of Naval Research, and JFBI90092 monitored by the Federal Bureau of Investigation.

Finally I would like to thank my wife, Ling-Huei, without whose encouragement and love this dissertation would not be possible.

Contents

List Of Tables

List Of Figures

Abstract

Generating tests for CMOS circuits using a switch level model is both more realistic and difficult than using the conventional gate level model. In this dissertation a systematic approach to deal with the various problems associated with switch level test generation (SLTG) is presented. This approach divides the SLTG process for combinational circuits into three parts: circuit manipulation, fault analysis and a test generation framework.

The circuit manipulation part includes circuit partitioning, complex and primitive gate identification, level computation for circuit partitions, and signal flow direction assignment for transistors. This work can be applied to other VLSI design problems such as timing verification, logic simulation and design rule checking.

The faults considered include bridging, transistor stuck-on/open, breaking and stuck-at faults. An extensive analysis on IDDQ testing is given. A set of design and test rules are provided under which all irredundant single and multiple bridging faults can be detected using IDDQ testing. Test invalidation for stuck-open faults due to circuit delay and charge sharing is considered. A novel and efficient "robust" test generation algorithm is developed. Stuck-on and breaking faults are modeled as bridging and stuck-open faults, respectively.

A PODEM-based test generation framework is developed that consists of five major components: objective selection, backtracing, logic implication, backtracking and fault propagation. All algorithms developed can be shared by each individual test generator. Fundamental work such as multiple objective selection, search-based backtracing and incremental event-driven logic implication has been completed.

The problems associated with sequential circuit testing are also considered, with emphasis on the testing of scan registers. All possible bridging faults in a scan register are systematically analyzed. Based on this analysis a universal test sequence for CMOS scan registers is derived.

A switch level test generation system called SWiTEST has been implemented in C on a SUN Sparc workstation. Experimental results show that SWiTEST is quite efficient in both CPU time and memory size. For example "robust" test vectors for over 95 % of stuck-open faults in a circuit containing 15,396 transistors can be generated in less than 0.21 seconds per fault, using 2.556 M bytes of memory.

Chapter 1

Introduction

The objective of this study is to generate tests for various classes of faults in scan-based CMOS circuits modeled at the switch level. In this introductory chapter a brief description of CMOS circuits is given and the various test problems associated with CMOS circuits are discussed. An overview of this dissertation is given at the end of this chapter.

1.1 Advantages of CMOS Circuits

Designing VLSI circuits using CMOS technology has many advantages: the static power dissipation is almost zero; gate outputs settle at the voltage of *VDD* or *GND* in steady state; transmission gates pass both logic 0 and 1 with little signal loss; rise and fall times can easily be made to have about the same values; and the noise margin is large [1, 2]. These advantages make CMOS a popular technology. As the complexity of CMOS circuits increases, the probability of having physical defects in circuits also increases. These defects can change the behavior of a circuit and result in functional errors. To ensure proper circuit operation, these defects must be detected.

1.2 CMOS Circuit Representations

A CMOS circuit can be described at various *levels* including *functional* level, *gate* level, *switch* level and *circuit* level. At the functional level a circuit is either described by its functional behavior (such as truth tables) or is represented as functional blocks (such as multipliers, adders and multiplexers). This level provides the information about the circuit functions but does not describe the detailed circuit structure which is often required in testing.

Most previous work dealing with test generation employs the *gate-level* description, where a circuit is described using a number of primitive logic gates including NAND, NOR, AND, OR, inverters and sometimes exclusive-OR and exclusive-NOR gates. Though being able to describe most structural information of a circuit, this level cannot represent certain circuits such as pass transistors. With today's VLSI technology where unconventional "customer-designs" may be used to achieve some special objectives, the gate-level representation is often inadequate.

At the switch level a CMOS circuit consists of a number of N-type and P-type MOS transistors and a number of circuit nodes connecting these transistors. Each transistor is considered as an ON/OFF switch whose status depends on the logic value at its gate terminal. Since the basic unit of a circuit is a transistor, all the structural information of a circuit can be described.

Another level of description for a circuit is at the circuit level where many circuit parameters such as those used in SPICE simulation can be included. The problem with this level is that, even for a small circuit, unacceptable computation time is required to accomplish a much simpler task such as simulation, not to mention test generation.

From the practical point of view, the switch level is probably the best representation for CMOS circuits. This is especially true when testing is considered, as will be shown in this dissertation.

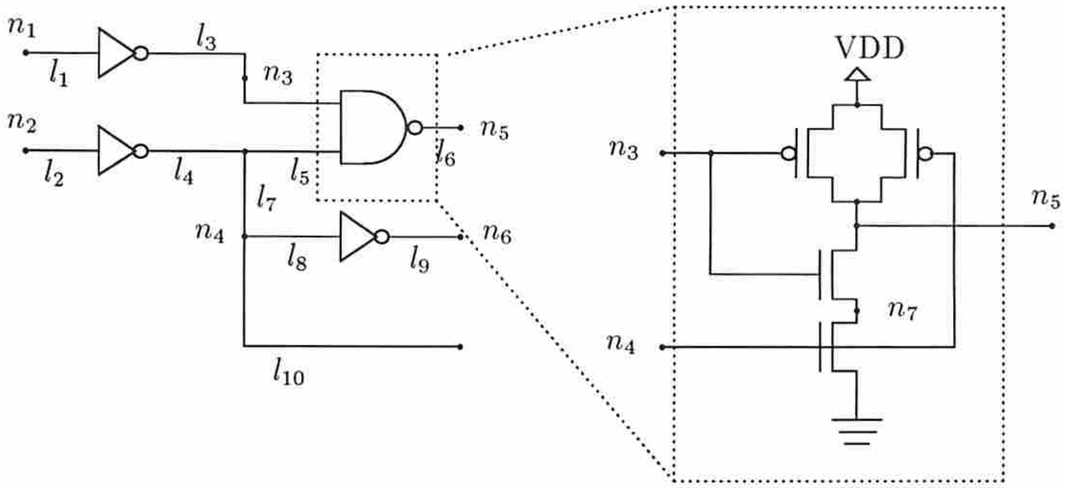
1.3 Physical Defects and Fault Models

Physical defects in a CMOS circuit are caused by many factors, including (1) faulty processes during manufacture such as overetching, poor contacts and inaccurate diffusion; and (2) environmental or electrical stress during operation such as electromigration, electrostatic discharge and avalanche breakdown. The effects of these physical defects on circuits have been studied extensively [3, 4, 5, 6, 7, 8]. It has been shown that these effects can be characterized using several *fault models*.

A fault model is an abstraction of how a fault mechanism may affect the functional operation of a device. With these models test patterns can be generated to specifically test for the fault mechanism. The reliability of a test highly depends on the accuracy and effectiveness of the fault models. Commonly used fault models for CMOS circuits include *line stuck-at faults, node stuck-at faults, transistor stuck-on/off faults, node-to-node bridging faults, line breaking faults* and *delay faults.* The definitions of *lines* and *nodes* are as follows. A *line* is a maximal segment of wire without a branch. A *node* is a collection of segments of connected wires which always have the same voltage value. Figure 1.1(a) illustrates a circuit at the gate level with 10 lines, $l_1, ..., l_{10}$, and 6 nodes, $n_1, ..., n_6$. Note that we often use a point to denote a node.

When a switch level circuit description is used, the number of lines increases dramatically. Figure 1.1(b) shows the NAND gate of Figure 1.1(a) at the switch level. There exist at least ten more lines inside the gate, but only three more nodes (*VDD*, *GND* and n_7) exist. Since *VDD* and *GND* are common to the entire circuit, only one additional node for this NAND gate needs to be considered.

The definitions of the various fault models are as follows. The *line stuck-at fault model*, which is the most commonly-used fault model applied to gate level circuit descriptions, assumes that the physical defect(s) cause a line to be permanently "stuck" at a logic high (stuck-at-1) or logic low (stuck-at-0) value.



(a) (b)

Figure 1.1: Definitions of nodes and lines

The *node stuck-at fault model* is the same as the line stuck-at fault model except that the fault is assumed to be on the node. The relations between various node and line stuck-at faults can be explained using Figure 1.1(a). Node n_4 stuck-at-1 implies lines l_4, l_5, l_7, l_8 and l_{10} are all stuck-at-1; line l_4 stuck-at-1 implies l_5, l_7, l_8 and l_{10} are stuck-at-1; l_7 stuck-at-1 implies l_8 and l_{10} are also stuck-at-1; l_5 stuck-at-1 does not imply any other line stuck-at-1.

The *transistor stuck fault model* assumes a transistor either permanently conducts (*stuck-on*) or does not conduct (*stuck-off*). A *transistor stuck-off* is also called *stuck-open*, which is the term used in this dissertation. A *bridging fault* between two nodes consists of an unwanted, zero-resistance line connecting these two nodes. The *line breaking fault model* assumes a line is permanently *broken*. *Delay faults* can be defined in two ways: *element delay* and *path delay faults*. The former assumes the transition time from an input to an output of an element (such as a transistor, a gate or a functional block) is larger than its specified upper bound, while the latter assumes the excess delay time is along a path (a series of transistors, gates or functional blocks). Note that all the above definitions assume the physical defects cause *permanent*, not *intermittent* faults. In this study only permanent faults are considered.

The fault models used depend on the level at which a circuit is tested. For example stuck-at and bridging fault models can be used at any level while stuck-on/off fault models are used only at the switch level. Many researchers have shown the inadequacy of gate level fault models for CMOS circuits [3, 4, 5, 6]. Thus more realistic fault models should be used if high-qualify tests are required.

Physically the defects in a CMOS circuit can be classified into two categories: device faults and connection faults. Device faults are the faults due to the defects in transistors, which can be modeled using stuck-open faults, stuck-on faults and bridging faults between device terminals. Connection faults include shorts and breaks on lines or nodes. Bridging, transistor stuck on/open and breaking faults can be considered as "physical" faults since they represent circuit defects in a manner that is very close to the actual physical structure of the circuit. The stuck-at and delay fault models, on the other hand, are more likely "logically" defined fault models. The stuck-at fault model assumes that when some defects occur in a circuit, the "logical" result is that some line or node will always have a logic high or low value while the delay fault model assumes that the defect causes the propagation time of "logic values" to fall out of a specific range.

In this study problems in generating tests for "physical" faults at the switch level are of major concern. These include analysis and development of test generators for transistor stuck-open faults, transistor stuck-on faults, bridging faults, and line breaking faults. To compare the performance of our switch level test generators with that of a gate level test generator, a switch level test generator for stuck-at faults is also implemented.

fundamental steps may be accomplished by one test vector or by a sequence of test vectors, depending on the "sequential behavior" of the circuit and the fault. It is possible for a combinational circuit to have sequential behavior when certain types of faults are present. For example it is well known that a CMOS stuck-open fault may result in some "memory" effect and thus two or more test patterns are required to detect such a fault.

Classical TG methods mainly concentrate on the detection of line stuck-at faults at the gate level. Even for this simple fault model, the TG problem is NP-complete [23]. Most problems encountered in detecting switch level faults can be shown to be NP-hard [24]. Thus heuristics must be employed to generate tests for large circuits. Test generation can be modeled as a search problem [25], in which an appropriate test pattern(s) for detecting a given fault is sought. Two factors are important for searching: effectiveness and efficiency. The former is concerned with the correctness of the solution. The latter deals with the computation time. In test generation these two factors correspond to whether the test pattern(s) really detects the given fault and how much computation is needed to generate the test pattern(s).

Many gate level TG algorithms have been developed and implemented, including the D-algorithm [15], PODEM [16], FAN [17], TOPS [19], Socrates [20] and the system described in [21]. These algorithms have been applied to some benchmark circuits containing up to tens of thousands of *primary gates*(AND, OR, NAND, NOR, Inverter gates, etc.) [26], and have been shown to be very efficient in generating tests for stuck-at faults. For example the system described in [21] is able to generate a complete test set for all stuck-at faults in a circuit containing more than 24,000 gates within 3 CPU minutes on an Applo DN3550 workstation.

1.4 Test Generation Problem

Test generation (TG) is the process of generating test vectors such that the application of these vectors to a circuit under test (CUT) enables the detection of faults [9]. TG can be achieved by three methods: random, exhaustive and deterministic. Random and exhaustive methods do not consider the functionality or structure of the CUT. The test lengths for these two methods are often large, even for a moderate size circuit, e.g., with 30 inputs. Deterministic TG methods construct a test pattern (or a sequence of test patterns) for a given fault by employing structural and/or functional information. Depending on whether or not the functionality of a circuit is considered, we may have algorithmic or functional TG. Test patterns can be generated manually or automatically (using a computer). Functional TG methods often involve high level knowledge; thus a manual method is usually better than an automatic one. Although many researchers have tried to apply artificial intelligence (AI) techniques to solve this problem [10, 11, 12, 13, 14], the practicability of automatic functional TG is still questionable. On the other hand, many automatic algorithmic TG methods have been successfully developed for gate level circuits [15, 16, 17, 18, 19, 20, 21]. Thus it is natural to consider this approach for switch level TG. In this study we focus on automatic and algorithmic TG methods. In the rest of this dissertation, test generation shall refer to an automatic algorithmic method.

Test generation for a fault consists of two fundamental steps: activate the fault and observe the fault effects. A fault can be activated by setting certain nodes to some specified logic values. Fault effects can be observed by monitoring the logic values on the primary output and/or by monitoring the current through the power supply. These two methods are referred to as *logic monitoring* and *current monitoring*, respectively. In the literature current monitoring is often called IDDQ testing [22]. Taking conventional methods for detecting line l *s-a-0* as an example, fault activation attempts to find a test vector for which l is set to 1 in the fault-free circuit but "stuck" at 0 in the faulty circuit, and fault observation attempts to propagate the fault effect (error) to a primary output. These two

1.5 Switch Level Test Generation

When only gate level stuck-at faults in a combinational circuit are of concern, many properties can be used to guide the search for test patterns: each gate is a primitive (minimal) element for the test generation process; the interface lines of a gate are unidirectional, i.e., each is either an input or an output of the gate; the relationship among gates are well defined, i.e., it is known exactly whether the output of a gate can affect another gate; and the distance between a gate and the closest primary input or primary output is easy to obtain. All these properties are not well defined when a circuit is described at the switch level. Since a MOS transistor is intrinsicly bidirectional, the signal flow direction (to be defined later) is usually unknown. A transistor can be used as the smallest unit of circuitry during test generation. However this often results in unacceptably large computation time.

Thus for switch level circuits, the following three problems must be addressed: *partition, direction*, and *level*. Partitioning is the process of dividing a circuit into subcircuits such that each subcircuit is easier to handle or has some special properties. The direction problem deals with assigning signal flow directions to transistors inside a subcircuit and the interfaces among subcircuits. The level problem deals with determining the relative positions of subcircuits in a circuit.

These problems along with new fault types, such as transistor stuck-on/off faults and bridging faults inside a gate, make switch level test generation a very complex process. Note that the term "switch level" is sometimes referred to as "transistor level" [27, 28]. In this dissertation these two terms are used interchangeably.

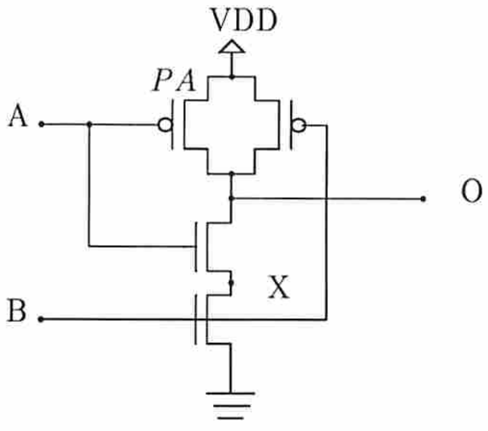


Figure 1.2: Sequential behavior of a stuck open fault

1.6 Special Properties of CMOS Circuits in Testing

There are several unique problems to be considered in testing a CMOS circuit. First, a stuck-open fault may force a combinational circuit to behave like a sequential one [29]. Referring to Figure 1.2, assume a stuck-open fault occurs on transistor PA. Let a sequence of two vectors $(A, B) = (1, 1)$ and $(A, B) = (0, 1)$ be applied to this circuit. The first vector sets the value of output O to 0. When the second vector is applied, in a fault-free circuit the output becomes 1, while in the faulty circuit the output remains at 0 since there is no conducting path from VDD to O. In general the state of output O of this faulty circuit depends on its previous state if the current input is $(A, B) = (0, 1)$. To detect such a fault a sequence of two test vectors is necessary. The first vector sets the faulty circuit to a required state, and the second one distinguishes the fault-free circuit from the faulty circuit.

A second problem deals with the detection of bridging faults (BFs). Conventional methods for detecting a BF adopt a *wired-AND* or *wired-OR* model, which assumes that when two nodes with complementary values are shorted, the resulting voltage on both nodes is either logic high (*wired-OR*) or logic low (*wired-AND*). This model has been successfully applied to TTL and ECL logic. However a BF in a CMOS circuit usually force both shorted nodes to take on an intermediate voltage value between *VDD* and *GND*. This value cannot always be interpreted as a logic high or logic low. As a result the wired-AND or wired-OR model is inadequate for use in the detection of BFs in CMOS circuits. Again consider Figure 1.2. Assume a BF occurs between node O and node X and no stuck-open fault on PA. When a vector $(A, B) = (0, 1)$ is applied, O and X are respectively connected to *VDD* and *GND* through conducting transistors. If the size of an P-type transistor is twice the size of a N-type transistor, then the resultant voltage at O will be close to *VDD*/2 which, depending on the physical parameter of the next stage, can be interpreted as either 1 or 0.

Another consideration is that many CMOS circuits are implemented using *fully complement MOS* gates (FCMOS). An FCMOS gate consists of a *P-net* and an *N-net* which contain only P-type transistors and N-type transistors, respectively. There exists a duality relation between the *P-net* and *N-net* which ensures that for a fault-free circuit a conducting path between the output node and *VDD* or *GND*, but not both, always exists during steady state. This duality property is very useful when developing tests for CMOS circuits. For example, to have a value 0 at the output of a FCMOS gate, the output must be connected to *GND* through one or more paths of conducting N-type transistors, and once a node is connected to *GND*, all paths connecting this node to *VDD* must be cut.

1.7 Invalidation of Test

As described before, a stuck-open fault may result in sequential circuit behavior and thus a two-pattern test is necessary: the first pattern is used to initialize the circuits and the second one is used to differentiate the faulty and fault-free circuits. Due to the effects of transient behavior and charge sharing, the state established by the first vector may be destroyed when the second vector is being applied, and thus the test may be *invalidated*. These effects can be classified into four categories, namely *transient strong charging*, *steady state charge sharing*, *transient charge sharing* and *static hazard*. Transient strong charging and static hazards are due to circuit delays, and static charge sharing is due to charge sharing only. Transient charge sharing is due to both circuit delays and charge sharing. These effects are described next.

1.7.1 Transient Strong Charging

Refer to Figure 1.3(a). Assume transistor PB is stuck-open. Consider a pair of test vectors $T_1 = (A, B, C) = (1, 0, 1)$ and $T_2 = (0, 0, 1)$. T_1 sets the output O to 0 in both fault-free and faulty circuits. T_2 is supposed to set O to 1 in the fault-free circuit and retain O at 0 in the faulty circuit. However during the transition from T_1 tp T_2, the control signal to PA may change before the control signal to $P\overline{A}$. Thus a transient state of input $(A, \overline{A}, B, C) = (0, 0, 0, 1)$ may exist. During this time both PA and $P\overline{A}$ are conducting and thus a conducting path from VDD to O exists. This results in node O being charged to 1 in both the fault-free and faulty circuits, and thus the test is invalidated. This *transient strong charging* is so named since the invalidation is caused by a *transient* conducting path from a *strong* power source to the output node.

1.7.2 Steady State Charge Sharing

Again consider transistor PB stuck-open in Figure 1.3(a), and the pair of test vectors $T_1 = (1, 0, 1)$ and $T_2 = (0, 0, 1)$. Assume for this case the control signal to transistor $P\overline{A}$ changes before the control signal to transistor PA. There is no transient strong charging from VDD to O. Consider node K which is inside the *P-net*. The value of K is set to 1 by T_1 since a conducting path from VDD to K through $P\overline{A}$ exists. When T_2 is applied to the faulty circuit, K is connected to O through transistor PA. Thus the charge between K and O will be shared. This may result in an intermediate value between VDD and GND and thus the test

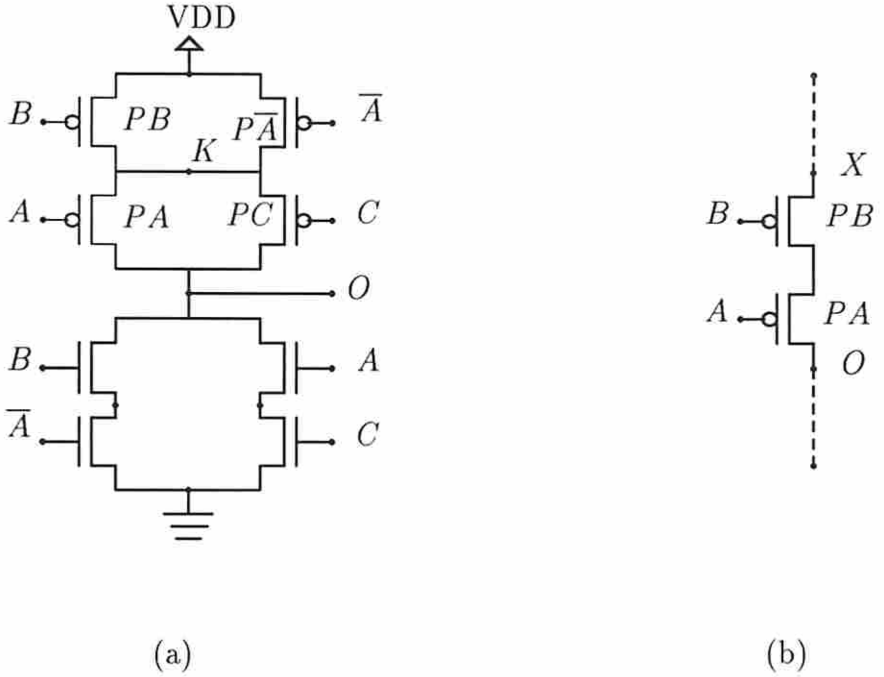


Figure 1.3: Invalidation of tests for stuck open faults

may be invalidated. Because this charge sharing occurs when both the control signals to PA and $P\overline{A}$ have changed to their final states, it is called *steady state charge sharing.*

1.7.3 Transient Charge Sharing

Another possibility for invalidation of a test is transient charge sharing. Consider Figure 1.3(b) where X and O are two nodes other than VDD or GND, and there are two transistors, PA and PB between them. Assume O is set to logic 0 by T_1, and is to be set to 1 in the fault-free circuit while retaining a 0 in the faulty circuit. If T_1 sets PA to conduct and PB to open, and T_2 sets PA to open and PB to conduct, then it is possible that during the transition both transistors are conducting. Thus transient charge sharing may occur between X and O if they have different logic values after T_1 is applied. Again, this may invalidate the test.

1.7.4 Static Hazard

The preceding three situations assume that the control signals to transistors (for both N-type and P-type) are hazard-free. If the circuit in Figure 1.3(a) is embedded in a large circuit, then it is possible to have static hazards at transistor inputs. For example the control signal to transistor PC may have a *static-1 hazard.* Again consider $T_1 = (1, 0, 1)$ and $T_2 = (0, 0, 1)$. If the input to PC has a static-1 hazard, then depending on the state of transistor $P\overline{A}$ when the hazard occurs, either a transient strong charging path from VDD through $P\overline{A}$ and PC to O is formed (if $P\overline{A}$ is conducting), or a transient charge sharing between K and O occurs (if $P\overline{A}$ is not conducting). In both cases the state established by T_1 is destroyed.

All the above situations must be carefully considered when generating tests for a stuck-open fault to ensure the detection of the fault. A test which can detect a fault under any circuit delay or charge sharing condition is called a *robust test.* In this study the generation of robust tests is considered.

1.8 Sequential Circuit Testing

Test generation for sequential circuits is an extremely difficult problem. The complexity of this problem can be reduced by using *design-for-test* techniques [30]. One popular design-for-test technique is to connect all the storage elements into a long *scan register.* Test vectors can then be scanned into the scan register and test responses can be scanned out for observation. This method, called *scan-based design* [31], reduces the sequential circuit test generation problem to one of generating tests for a combinational circuit. However, the scan register itself may have faults as well. Thus it is necessary to detect faults involving the scan register in order to guarantee the proper function of a scan-based sequential circuit.

1.9 Overview of the Dissertation

From the previous sections, it is clear that test generation at the switch level contains many complex problems. To systematically deal with these problems, the reminder of this dissertation is organized as follows.

Chapter 2 provides a background and surveys the previous work in this area. Due to the huge amount of research that has been carried on in this area, it is impossible to survey all previous work (e.g., more than 120 papers on stuck-open faults alone have been published). However we have tried to include most of the related and important work.

Chapter 3 discusses the overall strategies for dealing with the various problems associated with switch level test generation. These strategies can be divided into three parts: circuit manipulation, fault analysis and a test generation framework. We also present an overview of a switch level test generation system, called SWiTEST, which has been implemented using C on a SUN Sparc workstation.

Chapter 4 details the circuit manipulation part of this study, which includes circuit modeling, partitioning, complex and primitive gate identification, assigning levels to each circuit partition, and assigning signal flow direction to MOS transistors. Most work presented in this chapter can also be applied to other VLSI design problems such as timing verification, logic simulation and design rule checking. The signal flow direction assignment work will be described in great detail since it may be of interest to the readers who are not in the testing area.

Chapter 5 deals with the analysis of fault models. The faults analyzed include bridging faults, transistor stuck-on faults, transistor stuck-open faults and line breaking faults. The conditions for using IDDQ testing are explored. A set of design and test rules are provided which guarantee the proper use of IDDQ testing. It is shown that all single and multiple bridging faults in a circuit satisfying these rules can be detected using IDDQ testing. A new test generation method for stuck-open faults is developed. This method is quite efficient in solving the test invalidation problem. All four possible sources of invalidation described in Section 1.7 are considered. The line breaking and stuck-on faults are modeled as transistor stuck-open and bridging faults, respectively.

Chapter 6 describes the test generation framework. The framework is PODEM-based, thereby consisting of several major components such as objective selection, backtracing, logic implication, backtracking and fault propagation algorithms. All these algorithms consider both switch and gate level information and can be shared by any test generator developed for a specific type of fault. Some fundamental results such as backtracing at the switch level, efficient logic implication and objective selection via the help of signal flow direction information are presented.

Chapter 7 concentrates on problems associated with sequential circuit testing, with emphasis on the testing of a scan register. A systematic method is used to analyze all possible bridging faults in a scan register. Based on this analysis a universal test sequence for CMOS scan registers is derived.

Chapter 8 discusses the implementation issues and experimental results for SWiTEST. The SWiTEST system consists of three suites of software: preprocessor, test generation framework and individual fault manipulators. The preprocessor part reflects the work done in circuit manipulation as presented in Chapter 4. All the algorithms described in Chapter 6 are implemented which form the test generation framework. Three individual test generators (for bridging, stuck-open and stuck-at faults) are implemented. Experiment results for each type of fault are reported and discussed.

Chapter 9 concludes this dissertation and gives a list of future research topics.

Chapter 2

Background and Previous Work

This chapter provides the necessary background and surveys previous work on switch level test generation. The background emphasizes gate level test generation since most previous test generation efforts have been focused on this level and many concepts on this level will be used in our switch level test generation. The previous work covers a wide range of research, which includes circuit representation and partitioning, transistor signal flow direction assignment, switch level simulation, IDDQ testing, test generation for bridging, stuck-open and stuck-at faults, and testing for sequential circuits.

2.1 Test Generation at the Gate Level

Although the inadequacy of gate level test generation for testing CMOS circuits is well known many test concepts used at this level are still valuable for switch level testing. Thus a brief review of gate level test generation concepts and algorithms is given in this section.

Test generation can be modeled as a search problem. Depending on the search space employed, two major classes of algorithms, *D-algorithm like* [32, 15, 33] and *PODEM like* [16, 17, 19, 20, 21] can be defined. For the former the searching spaces contain all lines in the entire circuit, i.e., during test generation, the decision-making of a logic value assignment may occur on any line of the circuit. For the latter the decision is made on the primary inputs or some other lines whose values can always be justified. Due to this distinction the searching mechanisms are different. Next the D-algorithm and PODEM are described, followed by a brief discussion of their variations.

The four major components of the D-algorithm are *line justification*, *logic implication*, *backtracking* and *fault propagation*. Line justification refers to the process of satisfying the conditions for which a line can be assigned a specific logic value. If the assignment is impossible, it should return a conflict message so that *backtracking* can be triggered. Backtracking is a process of "undoing" a previous line value assignment to resolve any inconsistencies. Fault propagation attempts to propagate a fault effect to an observable primary output. When a fault effect is propagated through a gate, some conditions may be generated which need line justification. For example to propagate a fault effect from an input to the output of an OR gate, all other inputs of this gate must not have a value of 1. Logic implication propagates the effect of an assignment through a circuit. For example if one of the inputs of an AND gate is assigned a 0, then the output must be 0 too. Conversely if the output of an AND gate is to be assigned a value of 1 then all its input lines must have a value of 1. Inconsistencies may be found during this process, in which case backtracking is also necessary. These four processes (line justification, logic implication, fault propagation and backtracking) are repeated until a fault effect appears on a primary output and all line assignments have been justified, or until all possible input combinations fail to satisfy the requirement. In the first case the fault is detected by the resulting input combination, and in the second case the fault is undetectable.

The PODEM algorithm does not use the concept of line justification. It just repeatedly generates input assignments and checks whether the given fault is detected. A *binary decision tree* is used for this purpose. Each node in this tree represents a primary input and each child of a node represents a logic value assignment to the node. The most important task in PODEM is to find a promising input such that when a specific value is assigned to this input, the fault is likely to be detected. The main components of PODEM are *objective selection, backtracing, logic implication, fault propagation* and *backtracking.* An objective is a "local" goal to be achieved. For example the initial objective of detecting line l stuck-at-1 would be to set l to 0, and to propagate a fault from an input of a primitive gate to its output, the objective would be to set all other inputs to non-control values (i.e., 1 for an AND gate). Backtracing is a heuristic step which tries to find the most promising input assignment to achieve the selected objective. In the original PODEM algorithm the inverse parity of the backtracing path is used to determine the value to be assigned. Some measurements such as the distance between a line and its closest primary input or the controllability of the line can be used to determine the backtracing path. Logic implication used in PODEM is similar to the one used in D-algorithm, but is performed only when a new value is assigned to an input. Thus only forward implication is necessary. Backtracking is employed whenever it is identified that current assignment cannot lead to a test for the fault. Due to the "forward implication only" property of PODEM, the backtracking process is simply a forward implication starting from the input whose value is changed.

Two factors affect the efficiency of a test generation algorithm: the size of its search space and the ability to quickly discover potential inconsistencies. The advantage of the D-algorithm is that the values assigned to lines can be justified immediately and directly. The PODEM algorithm, however, has the advantage of smaller search space because only the values at the primary inputs need to be determined. Experiments have shown that in general PODEM is more efficient than the D-algorithm.

The 9-value algorithm [33] is similar to the D-algorithm. The difference between them is in the composite logic values they use. A composite logic value of a line consists of a pair of values, one representing the value in the fault-free circuit and the other in the faulty circuit [9]. A composite logic is denoted by u/v, where u and v are the fault-free and the faulty values, respectively. This notation not only simplifies the representation of logic values but also enables the concurrent manipulation of both the fault-free and faulty circuits. In the

D-algorithm, u/v may be one of $1/1$, $0/0$, $1/0(D)$, $0/1(\overline{D})$ and x/x, while for the 9-values algorithm, u/v may be one of $1/1$, $0/0$, $1/0(D)$, $0/1(\overline{D})$, x/x, $1/x$, $0/x$, $x/1$ and $x/0$, where x represents *don't care.*

PODEM like algorithms include PODEM [16], FAN [17], TOPS [19], Socrates [20, 34] and the system described in [21]. Among these algorithms, PODEM is the basic one. The others simply add more "intelligent" procedures to PODEM. For example FAN used a *multiple backtracing* mechanism to reduce the replicative backtracing efforts which may be required when backtracing through the same path several times. Also the concept of immediate assignment of *necessary* logic values is employed such that potential inconsistence can be identified quickly, thereby reducing the work of backtracking. The "intelligent" procedures used in FAN are further extended in Socrates where intensive preprocessing is carried out to record as many necessary assignments as possible.

It has been shown that the most time-consuming part of PODEM like algorithms is the backtracking process. Thus minimizing backtracking usually leads to better results. In [20] and [34] it has been shown that a near minimal number of backtracking steps has been achieved. In the implementation of [21] most concepts used in Socrates are employed and an extremely fast test generation system has been reported.

2.2 Transistor Circuit Representation and Partitioning

2.2.1 Gate Level Equivalent Model

Due to the success of gate level test generation, to detect faults in a circuit at the switch level some test generation algorithms use a preprocessing step to construct an equivalent gate level circuit [3, 35, 36, 37, 38, 39, 40]. Switch level faults are represented by gate level line stuck-at faults in the equivalent circuit. These

Finally, most techniques do not consider the possible invalidation effects when dealing with stuck-open faults. In [37] and [40] the invalidation due to circuit delay is considered, but the charge sharing effects are ignored.

2.2.2 Graph Representation and Circuit Partitioning

In contrast to the transistor-to-gate level transformation, many researchers have carried out the test generation process directly at the switch level. Here the basic idea is to consider (explicitly or implicitly) a switch level circuit as a directed or undirected graph. Two different relations between the graph and the circuit exist. The first one uses a tree to represent a circuit [41, 42, 27], where each terminal node of the tree is a control signal to a transistor and each internal node of the tree is a binary operation AND or OR, which represents a serial or parallel connection of transistors, respectively. The second relation treats each transistor as an edge in a graph, and each circuit node as a vertex in a graph [43, 44]. This correspondence captures the natural feature of the transistors and thus is used by most researchers.

One advantage of this second model is that it leads to a natural scheme for circuit partitioning. By breaking the control line to the gate terminal of each transistor, all nodes which are still connected to one another through the drains or sources of transistors form a *transistor group* [45]. This partitioning has the property that charge sharing can only occur inside a transistor group, and each interface (interconnect) line among the groups is unidirectional. Each transistor group is analogous to a logic gate in a gate level circuit, which is the smallest unit to be processed in a gate level test generator. In this thesis a formal definition for this representation will be given.

Because of the existence of pass transistors (or transmission gates), all logic gates which feed the same transmission gate network are assigned to the same group when the above partitioning scheme is used. This may lead to the loss of some information concerning the directionality of a logic gate. Thus a more detailed partitioning scheme may be necessary in order to obtain more information to aid the test generation process. In this study we will present an algorithm to further identify the complex and primitive gates inside a transistor group.

2.3 Transistor Signal Flow Direction

A MOS transistor is inherently a bidirectional device. Signals can propagate either from source to drain, or from drain to source when the transistor is in the ON state. The actual signal flow direction is determined by such factors as the capacitances of the source, drain, and the other nodes connected to them via ON transistors, and the conductances of these ON transistors. When a transistor is turned on, either its source or its drain terminal acts as the *source*, and the other as the *target* or *destination* of signal flow. The direction of a transistor is defined as the direction from signal source to signal destination.

In practice, most MOS transistors operate in a unidirectional mode, i.e., only one direction of signal flow through the transistor is feasible at all times. For example, in the fully complementary CMOS NOR gate shown in Figure 2.1 each transistor can be assigned a direction as shown in the figure. Nodes u and v are, respectively, the source and destination of signal flow through transistor PA.

For many computer-aided design and analysis tools, it has been found that if the direction of signal flow through transistors is known *a priori*, then both the computational effort and the accuracy of results can be considerably improved [46, 47, 48, 49]. Jouppi [46] used the transistor direction to guide the search for critical paths in a circuit to obtain accurate timing simulation data. He also used direction information for electrical design rule checking. Cirit [47] used direction information in calculating the controllabilities and observabilities of signal nets (or nodes). Chen *et al.* [48] have suggested using direction information to reduce

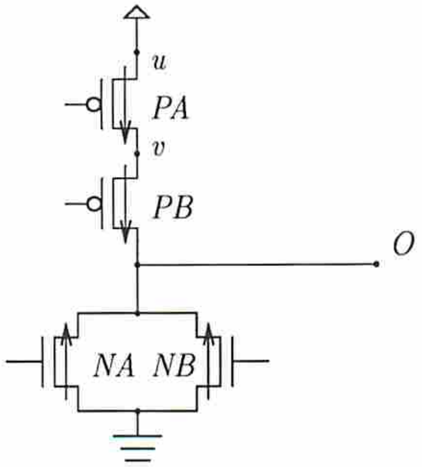


Figure 2.1: A NOR gate with transistor directions

the search time in switch level test generation. Barzilai *et al.* [49] used a similar concept to speed up switch level simulation.

Previous methods for assigning signal flow direction rely on rule-based systems [46, 47]. The most important rule-based system is probably the one developed by Jouppi [46] for timing simulation application. Eight *safe* rules and 5 *unsafe* rules are given. This system, though empirically shown to have good transistor coverage, has the following problems. First, the applications of rules rely on pattern matching techniques that are often time-consuming. Second, only local effects are considered. Thus some transistor directions that require global circuit information cannot be assigned. For example transistors PB, PC and PD in Figure 2.2(a) are unidirectional (as shown in the figure), but none of the rules proposed in [46] can assign these directions. Third, since the rules are verified through informal arguments rather than formal techniques, some "safe" rules may turn out to be unsafe in some special cases. For example Figure 2.2(b) shows a six transistors memory cell whose equivalent circuit diagram is given in Figure 2.2(c). Transistors A and B are both bidirectional but the system developed in [46] will identify them as unidirectional because, according to "safe" rule 6, all transistors fed by the output of an inverter are considered as unidirectional.

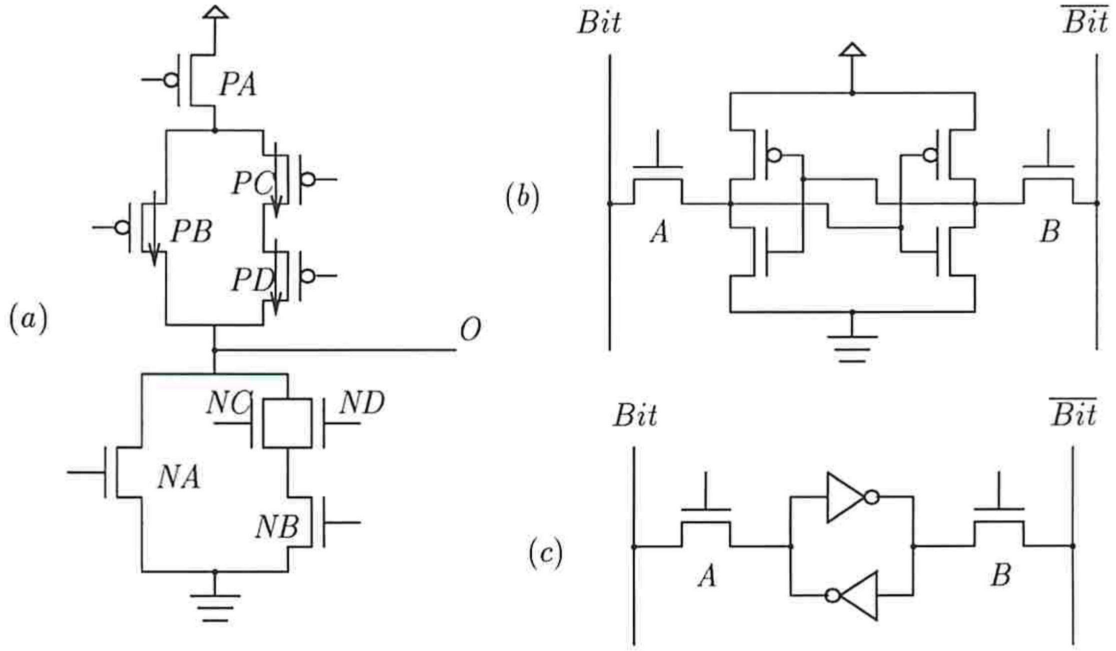


Figure 2.2: Problems associated with rule-based systems: (a) PB, PC, PD cannot be assigned a direction. (b)(c) Transistors A and B in a memory cell are assigned incorrect directions.

In [47] the direction rules of [46] are augmented based on the analysis of some special patterns such as transmission gate adders and Manchester carry chains. In addition Kirchoff's current rule is repeatedly applied to propagate the direction information throughout the circuit. This method requires that all transistors in the circuit be unidirectional, and thus cannot be applied to a general circuit. In this study we have developed a novel, graph-theoretic approach for assigning signal flow direction to transistors.

2.4 Switch Level Simulation

Logic implication is required during test generation. A major part of this process is logic simulation. Many switch level simulation algorithms exist. A nice survey in this area was done in [50]. In this section we shall only describe and discuss the MOSSIM II [51] and COSMOS [52, 53] systems.

As a switch level simulator, MOSSIM II models a MOS circuit as a network of nodes connected by ideal switches. Each switch corresponds to a transistor, and the state of a switch is determined by the node connected to the gate terminal of the transistor. Nodes can be either input nodes which include *VDD*, *GND* and all primary inputs, or storage nodes which are not input nodes. Ternary logic (0,1,X) is used for representing the state of a node or a switch. Node states 0, 1, and X represent low, high, and invalid (or uninitialized) voltages, and transistor states 0, 1, X represent open, closed, and indeterminate states. A novel mathematical analysis for ternary logic is used to prove the correctness of the model. Nodes are assigned different sizes (w, k_1, k_2, ...) to model the effects of their relative capacitances in charge sharing, and transistors are assigned different strengths (r_1, r_2, ...) to model the effects of their relative resistances in ratioed circuit. An input node has the size of w and a storage node has a size of k_i. The node sizes (or strengths) and switch strengths are totally ordered according to the relations $k_1 < k_2 < ... < r_1 < r_2 < ... < w$. A concept of *rooted paths* is used to compute the new state of the circuit given the old state and the current input. A rooted path p is a triple $< R(p), D(p), E(p) >$ consisting of a root (or initial) node $R(p)$, a destination node $D(p)$ and a sequence of switches $E(p)$ from $R(p)$ to $D(p)$. The strength of a path p is defined as the minimum of the size of $R(p)$ and the minimum switch strength in $E(p)$. The new state of a node v is determined by the states of the *root nodes* of the strongest paths to v. The equations for finding the strengths of paths are formulated as *fixed-point equations*. An algorithm of complexity $O(s + t)$ is then developed for solving these equations, where s is the total number of node and switch strengths and t is the number of transistors.

COSMOS [52, 53] uses a similar model as MOSSIM II. It also uses the concepts of node sizes, switch strengths, and ternary logic. The difference between COSMOS and MOSSIM II is in their solution algorithms. The correspondence between MOSSIM II and COSMOS is much like that of an interpreter and a compiler for programming languages. COSMOS converts the solution equations for a circuit into a C program which is then compiled into executable machine code. The work required for equation formulation is done only once as a preprocessing step. The simulation process is then carried out via the execution of object code.

COSMOS represents a ternary variable n using two boolean variables $n.1$ and $n.0$ such that $n = 0 \Longleftrightarrow (n.1 = 0, n.0 = 1)$, $n = 1 \Longleftrightarrow (n.1 = 1, n.0 = 0)$, and $n = X \Longleftrightarrow (n.1 = 1, n.0 = 1)$. The path equations can then be represented by symbolic boolean equations which contain only boolean operations, AND, OR and NOT. These equations are solved using Gaussian elimination to obtain a set of boolean formulas which contain $O(n^3)$ operations in the worst case. It has been shown that all but a limited set of dense pass transistor networks lead to formulas which contain $O(n)$ operations [52, 53]. These formulas can then be compiled into machine code for execution.

COSMOS is one or two orders of magnitude faster than MOSSIM II. However, this speed-up is due to extensive preprocessing. It is not clear how this methodology can be incorporated into a test generator where each fault gives a different faulty circuit. In MOSSIM II and COSMOS, faults can be represented by inserting *fault transistors* [54, 55] and *fault variables* [50], respectively. These modeling methods make the CUT more complex if many faults are processed concurrently. It is reported in [56] that 15 M bytes of memory is required to process a circuit containing 770 transistors using the method suggested in [50].

Another consideration is that the transient effect in circuits is ignored. Both COSMOS and MOSSIM II solve the path equations only for the steady state condition, and assume all input signals change at the same time. For a well-designed, fault-free circuit this generally causes no problem. However for a faulty circuit, where transient or hazard effects may affect the validation of the test, these simulators may fail to give accurate results.

It is stated in [51] that most transistors in CMOS circuits can be assigned the same strength. This statement is certainly applicable to COSMOS. However an analysis of bridging faults shows that even for an exhaustive test set, most bridging faults cannot be detected by COSMOS (so is MOSSIM II) if only one transistor strength is used [57]. This is because most bridging faults result in an "X" state for both shorted nodes.

Another disadvantage of MOSSIM II or COSMOS is the requirement of the total ordering of node sizes (strengths). From an electrical standpoint it makes sense to resolve charge sharing in favor of a node of greater capacitance. For example if a node u has eight or more times the capacitance of another node v, u should dominate the charge sharing [58]. However assigning strengths to nodes using a total ordering may encounter the following problem. Let $S(n)$ be the strength associated with node n. Consider three nodes n_1, n_2 and n_3 with relative capacitances 1, 4, and 16, respectively. Since the capacitance ratio between n_1 and n_2, and between n_2 and n_3 are both 1:4, the assignment of strengths satisfies the following equalities: $S(n_1) = S(n_2)$ and $S(n_2) = S(n_3)$. Now the total ordering requires that if $S(n_1) = S(n_2)$ and $S(n_2) = S(n_3)$, then $S(n_1) = S(n_3)$. Thus the same strength is assigned to all three nodes. This is undesired since if charge sharing only occurs between n_1 and n_3, then n_3 should dominate the result. Agrawal *et al.* [58] proposed a partial ordering method to solve this problem. However their method can only resolve the problem of charge sharing between two nodes. If more than 2 nodes can share charge, then the problem is not resolved. For example if charge sharing occurs among three nodes n_1, n_2 and n_3 which have relative capacitance values of 1, 1, and 8, and n_3 has a different logic value than n_1 and n_2, the resulting voltage should be "undetermined," while the partial ordering method will assign all three nodes the value of n_3.

2.5 IDDQ Testing

Classical methods for testing digital integrated circuits mainly use logic monitoring, i.e., a fault is detected by observing the logic values at the primary outputs of a circuit. This approach is often inadequate when a fault results in intermediate logic values.

Another feasible approach for detecting CMOS faults is by monitoring the current supplied by the power line [59, 60, 61]. This approach, often called IDDQ testing or *current supply monitoring (CSM)*, makes use of an important property of CMOS circuits, namely during steady state the current supplied by the power source should be relatively small. When a fault results in a conducting path between *VDD* and *GND*, the supply current becomes quite large. If this abnormal current can be measured, then the fault can be detected.

One important advantage of IDDQ testing over conventional logic monitoring is that no fault propagation is necessary. The fault effect is observed directly through the *VDD* or *GND* line. Thus a fault is detected as long as it results in a conducting path from *VDD* to *GND* during steady state. Therefore only the controllability, but not the observability of the circuit is relevant when IDDQ testing is used. This obviously reduces the test generation efforts dramatically.

Due to insufficient resolution, IDDQ testing may not be directly applied to a large circuit that normally consumes a large steady-state current. This difficulty can be overcome by using built-in current measurement devices [62, 63, 64, 65]. A new testing methodology called *built-in current testing* using *built-in current sensors (BICS)* has been described in [64, 65]. By partitioning a circuit into modules and using a separate *BICS* for each module, this methodology not only makes CSM applicable to large circuits but also facilitates on-line self-testing. Thus IDDQ testing appears to be a promising method for detecting some CMOS faults that may result in large current consumption.

A significant amount of research on IDDQ testing has been done recently [62, 63, 64, 65, 66, 67, 68]. However one fundamental problem associated with this method has not been thoroughly explored, namely to characterize those circuits for which IDDQ testing can be applied. In this study we shall derive a set of design and test rules under which the IDDQ testing can be used "safely."

2.6 Test Generation for Bridging Faults

The conventional approach for detecting bridging faults (BFs) is based on the wired-OR and/or wired-AND model which assumes that both shorted nodes will always have a logic 1 (wired-OR) or logic 0 (wired-AND) value [69]. The feasibility of this method has been examined by many researchers. In [70] the effect of BFs inside a logic gate is discussed. Both feedback and non-feedback BFs between two logic nodes (i.e., inputs or outputs to logic gates) are analyzed in [71]. A comparison of switch level and circuit level models of BFs is presented in [72]. These studies have shown the inadequacy of logic testing for CMOS BFs that may result in indeterminate logic levels. More work that shows that the wired-OR or wired-AND is unrealistic for CMOS circuits can be found in [73, 74, 75, 76].

Detecting bridging faults in CMOS circuits using IDDQ testing was introduced in [59, 60, 61], and developed further in [77, 71, 72, 73, 74]. Most of this work deals with bridging faults between gate inputs or outputs. Bridging faults inside a gate are usually ignored. Also none of the previous work has formally shown the feasibility of IDDQ testing in detecting bridging faults.

2.7 Test Generation for Stuck-open Faults

Tremendous efforts have been made to address the problem of stuck-open fault testing. It is reported that more than 120 papers dealing with this problem have been published [75]. Due to this huge amount of research results, it is impossible to give a complete survey here. In this section the discussion will focus primarily on the "true" switch level test generation. Methods using gate level equivalent circuits are not included here because of the problems described in Section 2.2.1.

Detecting a CMOS stuck-open fault is difficult because a CMOS combinational circuit may become sequential when a stuck-open fault occurs. In [78] a circuit design technique is proposed that guarantees no sequential behavior can occur due to stuck-open faults. This technique converts each CMOS gate to be either a pseudo-NMOS or a pseudo-PMOS gate during testing by inserting two additional transistors to each gate. Although this method enables the detection of stuck-open faults using single-vector tests, it requires high area overhead. In general two-vector tests are necessary for detecting stuck-open faults. The first vector, known as an *initialization vector*, establishes an initial circuit condition, and the second vector, known as a *test vector*, activates the fault effect so that it can be observed.

El-ziq has published several papers dealing with testing of MOS circuits [79, 80, 81, 82, 83]. In [79] he deals with the problem of test generation for CMOS circuits consisting only of primary gates (NAND, NOR, AND, OR). Two test generation algorithms are given. The first one is actually a fault simulator which identifies the stuck-opens faults that can be detected using an existing stuck-at test set. The second one is a test generation procedure using classical path sensitization techniques. This work has been extended to deal with circuits containing complex gates and transmission gates [80]. Problems dealing primarily with NMOS circuits are considered in [81, 82, 83]. These papers do not consider the problems of test invalidation due to circuit delay or charge sharing.

Chiang and Vranesic [84] use a planar graph to represent an NMOS complex gate. Algorithms in graph theory, such as finding cutsets and dual graphs are used in test generation. This approach is applied to FCMOS complex gates in [43]. Again no test invalidation is considered. Chandramouli [85] considers the problem of detecting stuck-open faults in a circuit consisting of primitive gates. He shows that all stuck-open faults in an *internal fanout-free* circuit (a circuit with no fanout on lines that are not primary inputs) can be detected by rearranging a complete stuck-at fault test set for that circuit. Circuit delay and charge sharing effect are again not considered.

Agrawal [44] has developed a D-algorithm based switch level test generation algorithm. D-drive (fault propagation), forward and backward implications, as well as two pattern tests for stuck-open faults are described. It is not clear how this algorithm can be applied to a circuit containing pass transistors. The invalidation of tests for stuck-open faults is considered, but only as a post examination process, i.e, no consistency checking is done until a pair of test vectors are generated.

Reddy *et al.* [28] and Chen *et al.* [48] both present a test generation algorithm based on extending Bryant's MOSSIM II simulation model. In Bryant's model [51] an uninitialized state and a conflict (or undetermined) state are both represented by the state X. This can cause some confusion on when to trigger backtracking during test generation. Reddy *et al.* extends the model by introducing a larger state set so that state ambiguity can be eliminated. An IMPLY process of linear complexity is used to perform logic implication based on the larger state set. The test generation procedure is again based on the D-algorithm. An ordered triplet $l_1l_2l_3$ is used to represent the states of a transistor or a node during the initialization, transition, and test time frames. This notation simplifies the examination of test invalidation. However no program implementation issues have been reported. Thus it is not clear how the various D-cubes used in the D-algorithm are generated. Also test validation checking appears to be done after a pair of test vectors have been generated. Chen *et al.* [48] use a similar state set to represent the states of nodes and transistors. They employ the

PODEM test generation strategy. Their backtracing process works in both space and time using a single transistor as the basic unit of circuitry. A time domain expansion is performed only when no further space domain expansion is possible. Though this algorithm aims at generating tests for any sequential circuit, it is not complete as it assumes fault effect can be propagated from the cite of the fault to a primary output in one time frame. This algorithm was implemented in the C language. A typical run shows that for a 238 transistor circuit, the average time to generate a test is around 100 CPU seconds on a Pyramid 90X minicomputer. This performance does not seem impressive when compared with gate level test generators. As discussed in [48], many enhancements are possible. Unfortunately no further work has been reported.

Najm [86] describes a D-algorithm based test generator which is capable of generating robust tests that cannot be invalidated by any circuit delay or charge sharing effect. However this algorithm has the danger of diagnosing a fault-free circuit to be faulty when a test requires electrical charge to be established and held in the fault-free circuit. This is because certain approximations involved in the switch level simulator may fail to predict the charge loss. Another problem with this method is that it requires the explicit enumeration of all possible subsets of the set of nodes in a transistor group in order to deal with the charge sharing problems. This results in exponential computational complexity. Experiments indicate that to generate a test for a four-bit adder with about 200 transistors requires a total time of from less than one second to about one minute on a VAX 11/780.

A universal test set for CMOS stuck-open faults is given by Gupta and Jha [87]. They use a concept similar to that of Akers [88] and Reddy [89] who derive universal test sets for gate level circuits. The major problem for these methods is the serious constraint imposed on the circuit, namely all the paths starting from the primary inputs and meeting at the same gate must have the same inversion parity.

Some more work [56, 90, 91] dealing with test generation for stuck-open faults in sequential circuits will be discussed in Section 2.9.

Because of the complexity of switch level test generation, the design of testable CMOS combinational circuits is an important research area. Here a circuit is realized according to some design rules so that tests can be more easily generated for the circuit. However to date all proposed testable designs can only realize circuits using primary gates (AND, OR, NOR, NAND, etc.) or using only a large single complex gate [92, 93, 94, 95, 96, 97]. Using only primary gates has the disadvantage of increasing the total layout area and realizing a circuit using a complex gate is not practical for large circuits. Design for test may simplify the test generation problem, but does not eliminate it. Thus in this study we assume that designs do not adhere to the various design for test technologies which have been suggested.

Another possible solution for the stuck-open fault problem is to lay out circuits under some guidelines. This is a relatively new research area. In [94] several layout rules are suggested which may reduce the possibility of stuck-open faults. Unfortunately this method cannot be used if the layout of a circuit is fixed or only allows for minor modifications. Two other layout techniques are described in [75], namely, 1) simply widen the metal where possible, and 2) use parallel *VDD* (*GND*) paths to the source connections, called substrate-ties. These two techniques are more practical as they try to prevent the occurrence of stuck-open faults with only slight modification to the layout. Two recent papers use layout information to generate more realistic fault sets [74, 98]. Because actual physical defects are highly related to circuit layout, these methods provide an improvement in the quality of testing over conventional methods. In this study it is shown that the layout information can be very helpful when dealing with the invalidation problem due to charge sharing.

2.8 Test Generation for Stuck-at Faults

Robinson [99, 100] presents a test generation algorithm for switch level node stuck-at faults based on the D-algorithm. Here a CMOS combinational circuit is partitioned into a set of subcircuits via the help of test engineers. The program then generates various D-cubes for each subcircuit and constructs the test vectors using a 9-value logic based D-algorithm. This method does not consider sequential faults in combinational circuits, e.g., a stuck-at 0 on the gate terminal of an N-type transistor is equivalent to a stuck-open fault. Also the requirement for users to partition the circuit makes this method less attractive.

Cho and Bryant [56] develop a complete switch level ATPG for all stuck-at faults at the I/O of *transistor groups*. No stuck-at faults inside a transistor group is considered. Also this method can only deal with a very small circuit as described before.

2.9 Sequential Circuit Testing

Recently some work has been done in the area of test generation for a generic sequential circuit described at the switch level [56, 90, 91]. However from the reported experimental results, all of this work either takes large computation time [90, 91] or requires excessive memory space [56]. Thus they can be applied to small circuits only. Also these algorithms rely solely on logic monitoring. No IDDQ testing is employed. In [91] possible extension to IDDQ testing is mentioned. However since the proposed algorithm may rely on charge sharing or charge retention to detect a fault, it is possible that a fault-free circuit will be identified as faulty. Details of this fact will be described in Section 5.1.

The problem of test generation for sequential circuits can be reduced to the problem of test generation for combinational circuits by using a scan-based design. However faults may also occur inside the scan registers. These faults must be detected to guarantee the proper functionality of scan-based systems. A test procedure for scan registers is described in [101]. Two test sequences, 010 and 01100, are applied to the scan register and the scan outputs are observed. This technique may not detect some bridging faults (BFs) that occur between some specific nodes in the scan path. To achieve high fault coverage an extensive analysis on all possible BFs is necessary.

Chapter 3

Strategies and System Overview

From the discussions in Chapters 1 and 2 it is obvious that a significant amount of research has been conducted for switch level test generation but many problems still remain unsolved. In this chapter a general overview of this study is given. First the basic considerations that lead to the final decision on the scope of the switch level test generation system will be discussed. Based on these basic considerations, an outline of the approaches used in this study is then given. The approaches can be divided into three parts: circuit manipulation, fault analysis and a test generation framework. These three parts will be described in details in the next three chapters respectively. An overview of the test generation system, SWiTEST, will be given at the end of this chapter.

3.1 Basic Considerations

3.1.1 Test Generation Algorithms

Due to the success of gate level test generation, it is natural to consider making use of the gate level test generation concepts to deal with the switch level test generation problem. A comparison of existing gate-level test generation algorithms was made and it was concluded that PODEM like algorithms are more efficient than the D-algorithm for switch-level test generation.

This conclusion is drawn not only because gate-level experiments have shown the superiority of PODEM over the D-algorithm, but also because the nature of switch-level circuits generally makes it difficult to define some major concepts used in the D-algorithm such as propagation D-cubes and primitive D-cubes of faults [32]. In addition, the D-algorithm requires backward propagation of signals (line justification and implication), while PODEM employs only forward implication. Furthermore, it was found that the backtracing and logic implication, which are two major procedures in PODEM, can be efficiently implemented at the switch level.

Another reason to select PODEM is due to the evolution of gate level test generation. The D-algorithm was developed in 1966 [32]. At that time many people thought that the test generation problem was well solved, even though the D-algorithm was quite inefficient for large circuits. It took almost 15 years before the PODEM algorithm was conceived in the early 80's. After PODEM was published [16], it was found that not only is PODEM more efficient than the D-algorithm, but also it has provided a very nice framework for test generation and many improvements are still possible. Since the publication of the PODEM algorithm, every few years a new and more efficient test generation algorithm has been developed [17, 18, 19, 20, 21]. These algorithms are PODEM-based, i.e., they all make use of the basic concepts of PODEM such as objective selection, decision tree, backtracing and logic implication.

It is believed that a PODEM-based switch level test generation framework will be most useful and expandable. Thus the present implementation focuses on how to modify the gate level PODEM algorithm into a switch level test generation framework.

3.1.2 Circuits and Faults Under Consideration

Many different design styles for CMOS circuits exist, such as precharged logic (e.g., Domino logic), cascade voltage switch logic, and FCMOS gate design [1]. Various fault models can be used in each design style. The goal of a test generator may be to

1. generate test for any possible fault in all styles of circuits,
2. generate test for a specific fault model in all styles of circuits,
3. generate test for any possible fault in one style of circuits, or
4. generate test for one specific fault model in one style of circuits.

A test generator which attempts to detect all possible faults in all styles of circuits is impractical. A test generator aimed at generating tests for only one fault model is not a stand alone system since it requires extra effort to detect other faults. Thus a test generator which can detect as many faults as possible for a specific design style seems more durable and useful. Theoretically any circuit function can be implemented using any design style. Thus from the testing point of view it is quite reasonable to place some restrictions on circuits so that certain special circuit properties can be used during test generation. The restrictions on circuits in this study, which can also be considered as design or test rules, will be discussed in detail where appropriate.

The fault models analyzed in this study include the following.

- All bridging faults, which include both single and multiple BFs. A bridging fault can occur between any pair of circuit nodes.
- All stuck-on faults, which include both single and multiple stuck-on faults.
- All single stuck-open faults.
- All single line breaking faults.

All single and multiple bridging and stuck-on faults are considered while only single stuck-open and line-breaking are considered. This is because by employing IDDQ testing, multiple bridging and stuck-on faults can be detected using single fault tests, but the same argument does not hold for stuck-open and breaking faults.

All faults are assumed to be permanent faults. Though stuck-at faults are not analyzed in this study, a test generator for stuck-at faults is still implemented so that a comparison between our test generation system and the existing gate level test generation systems can be made.

3.1.3 Fault Effect Observation Methods

As described before, for the bridging faults that result in intermediate logic values, the only way to guarantee their detection is by using IDDQ testing. Thus IDDQ testing is necessary if all bridging faults are to be detected. On the other hand, since IDDQ testing cannot detect faults that do not result in large current, logic monitoring is still required. In this study both logic and current monitoring will be used to enhance the detectability of faults.

3.2 Outline of Approaches

The approaches used in this study can be divided into three parts: circuit manipulation, fault analysis and a test generation framework. The basic idea of circuit manipulation is to extract useful information from switch level circuit descriptions such that the concepts used for gate level test generation can be extended to the switch level. The fault analysis part aims at discovering the most efficient way to deal with each fault model. It also tries to reduce the number of faults that need to be considered. The objective of the test generation framework part is to identify the most important components that can be shared by each individual test generator for each fault model. Once these components are implemented, the test generation efforts can be focused on the specific properties of each fault model. Also it will be much easier to extend the test generation system to extra fault models that are not considered in this study, e.g., delay faults.

3.2.1 Circuit Manipulation

3.2.1.1 Circuit Representation and Partitioning

Comparing gate and switch level circuits, it is easy to find that the basic components of gate and switch level circuits are primitive gates and transistors, respectively. Assume each gate consists of, say, 6 transistors, then using a gate to switch level transformation the problem size becomes 6 times larger if transistors are used as the basic units during test generation. Also at the gate level the I/O relations between any two gates are well defined, while at the switch level each transistor is a bidirectional device. To make efficient use of PODEM concepts such as backtracing, transistors must be clustered into groups so that during test generation each group can be considered as a basic unit and the I/O relations among groups is well defined. A circuit partitioning scheme will be used to solve these two problems.

3.2.1.2 Complex Gate Identification

Switch level circuits basically consist of complex gates and pass transistors, where complex gates are made up of a P- and an N-type transistor networks and one output node. It is clear that to have a logic 1 (0) at the output of a complex gate, a conducting path of P(N)-type transistors must be turned on. Thus the inputs affecting the output must have the complementary logic values of the output. This information will help in both backtracing and logic implication during test generation.

Moreover, many complex gates are in fact primitive gates which include NAND gates, NOR gates and inverters. The input/output of a primitive gate is even more precise than a generic complex gate, e.g., to have a logic 1 at the output of an NOR gate, all inputs must be 0. Thus the test generation efforts can be further reduced. In this study information about complex gates as well as primitive gates will be extracted.

3.2.1.3 Transistor Group Leveling

The gate level PODEM algorithm requires information about "how easy" it is to set a circuit line to a specific logic value, or "how far" a circuit line is from its closest primary input or output. This information is very useful for determining the next objective to be achieved for backtracing and fault propagation. At the switch level a similar concept based on the partitioning of circuits can be used.

3.2.1.4 Transistor Signal Flow Direction Assignment

The partitioning scheme used in this study only identifies the I/O relation among partitions. The signal flow direction inside a partition is yet to be defined. This information is also useful for determining the objective to select and the direction to backtrace. A novel algorithmic method to deal with this problem will be presented.

3.2.2 Fault Analysis

3.2.2.1 IDDQ Testing

IDDQ testing has been identified as an efficient and effective test method, but no formal proof for its effectiveness exists. In this study a set of design and test rules under which IDDQ testing can be "safely" used are derived. The strategies to deal with circuits which do not satisfy these rules are also discussed.

3.2.2.2 Bridging Faults and Stuck-on Faults

IDDQ testing will be used to detect bridging faults in this study. Several new theoretic results have been obtained which include 1) all irredundant single bridging faults in a circuit satisfying a given set of design and test rules can be detected by using IDDQ testing; 2) if a test detects a single bridging fault f, then it also detects all multiple bridging faults which contain f; and 3) if any of the rules is removed, then some circuits exist for which IDDQ testing cannot give correct results. Results 1) and 2) illustrate the sufficiency of the set of rules for using IDDQ testing, while 3) shows the "minimality" of the set of rules.

A stuck-on fault is modeled as a bridging fault between the source and drain terminals of the faulty transistor.

3.2.2.3 Stuck-open Faults and Breaking Faults

The difficulty in detecting CMOS stuck-open faults is due to the possible invalidation of a test, either caused by charge sharing or circuit delay. A very efficient test generation procedure to generate stuck-open tests that cannot be invalidated by any circuit delay is developed. For the charge sharing problem an extensive analysis has been conducted which shows that this problem can be solved by using IDDQ testing. Finally an algorithm that can generate "robust" tests for stuck-open faults is given.

Once the test generation for stuck-open faults is done, the detection of line breaking faults can be dealt with by modeling these faults as transistor stuck-open faults.

3.2.2.4 Sequential Circuit Testing

In this study it is assumed that scan-based sequential circuits are used. All possible bridging faults in a scan register are analyzed and a test sequence is derived for each fault. A universal test sequence for all bridging faults has been derived based on the information for each individual fault.

3.2.3 PODEM-based Test Generation Framework

3.2.3.1 Objective Selection

The goal of objective selection at the switch level is the same as that at the gate level: to select an objective that is likely to lead to the discovery of a valid test. The actual implementation of objective selection at the switch level is harder due to the lack of I/O relations among transistors and signal flow information of each transistor. Thus the information obtained in the circuit manipulation part of this study becomes very important.

Another problem is that different faults may require different objectives. For example the objectives of detecting a bridging fault between two nodes would simply be to set these two nodes to complementary logic values, while the objectives of detecting a stuck-open fault could be much more complex. The strategy for this problem, in addition to making use of the information obtained in the circuit manipulation part, is to use an "objective array" in which a list of objectives are kept. Each objective in this array is then selected and processed until all objectives in the array are satisfied.

3.2.3.2 Backtracing

Similar to objective selection, backtracing at the switch level also faces the problem of the lack of I/O relations among transistors and the signal flow information of each transistor. Thus the information obtained from the circuit manipulation part is also very useful here.

Switch level backtracing does have two problems that do not exist at the gate level, namely at the switch level backtracing may fail to find an unassigned primary input, and loops inside a partition may exist. Thus the concept of gate level backtracing cannot be directly used at the switch level. A new method to deal with these problems will be presented.

3.2.3.3 Logic Implication

Logic implication for PODEM is basically a good circuit simulation. This is true for either gate or switch level. Existing switch level simulation may be used to accomplish the logic implication process during test generation. This, however, is inefficient because of the nature of the PODEM algorithm that assigns only one primary input at a time. In this study an event-driven, incremental algorithm will be presented. The worst case complexity of this algorithm is linear in the number of transistors inside the transistor group to be evaluated.

3.2.3.4 Backtracking

At the gate level backtracking can be done simply by re-evaluating each gate that is affected by the new-assigned primary input because the I/O relation of a gate is well specified. It is not necessary to record which primary input assignment causes the status change of each internal node of a circuit. This is not true at the switch level as will be explained in Chapter 6. To efficiently deal with this problem in terms of both memory usage and CPU time, some "elegant" program tricks in C have been developed.

3.2.3.5 Fault Propagation

The concepts used in fault propagation at the switch level is again similar to that used at the gate level. The information obtained during circuit manipulation also plays an important role. The difference between gate and switch level fault propagations is that the fault effects at the switch level always "fork" and "merge," i.e., the fault effects from one node to another node always go through two paths, one through P-transistors and the other through N-transistors. Details will be described in Chapter 6.

3.3 System Overview

A switch level test generation system, called SWiTEST, has been implemented using the C programming language on a SUN Sparc workstation running SunOs 4.1 (a UNIX-based operating system). The block diagram of this system is illustrated in Figure 3.1.

The algorithms developed for circuit manipulation are used as the preprocessor of the system. Data obtained are stored in files so that they can be repeatedly used. The box containing "others" is for future extension, e.g., to implement the concepts used in FAN or Socrates. The algorithms for the test generation framework are implemented as modular routines. Each of these routines can be accessed by each individual fault manipulator. Three individual test generators, for bridging, stuck-open and stuck-at faults, are implemented. The stuck-on faults are considered as bridging faults. In the future the system can be extended to include other faults such as delay faults and can be tied to a fault simulator so that a complete *automatic test pattern generation* system (ATPG) can be developed.

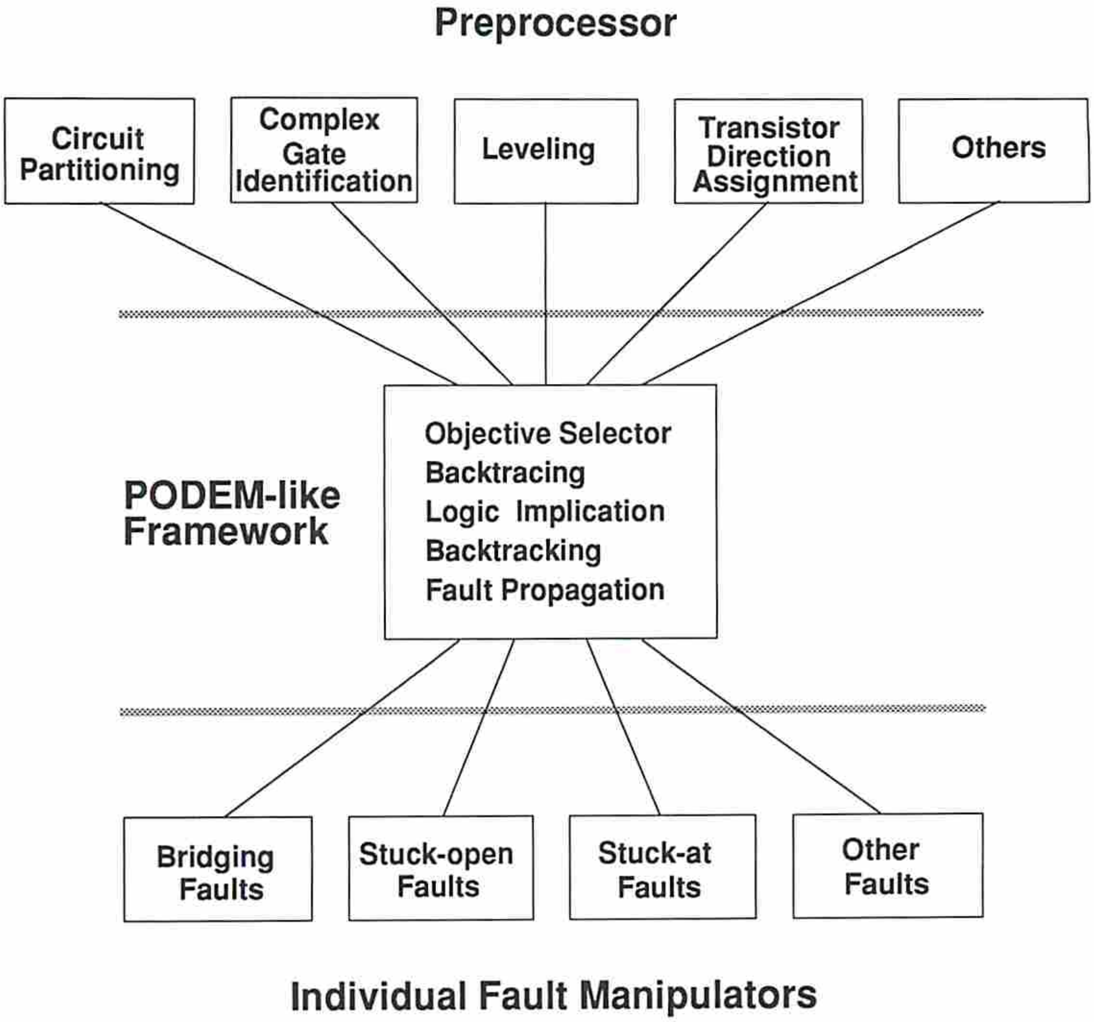


Figure 3.1: Overview of SWiTEST

Chapter 4

Circuit Manipulation

In this chapter various manipulations on switch level circuits are described. Though the original objective of these manipulations is for test generation, they are also useful in many other CAD fields such as circuit verification, timing analysis, logic simulation and design rule checking. The algorithms to be described include CMOS circuit modeling and partitioning, complex gate identification, level assignment to each circuit partition, and transistor signal flow direction assignment. The work of assigning signal flow direction to MOS transistors is particularly novel and unique. Thus it will be described in great detail.

4.1 CMOS Circuit Modeling and Partitioning

As described in Section 1.2, a switch level CMOS circuit consists of a number of N- and P-type transistors and a number of nodes connecting these transistors. A commonly-used partitioning method for MOS circuits [45] as described below is adopted in this study.

A MOS circuit is mapped into an undirected graph whose edges correspond to transistor channels and whose vertices (or nodes) correspond to nodes in the circuit. The *VDD* and *GND* circuit nodes are then replicated so that each transistor connected to *VDD* or *GND* has its own copy of a *VDD* or *GND* node in the graph. These procedures naturally partition the graph model of a circuit into a number of disjoint "channel-connected" components. By merging all *VDD* and *GND* nodes in a connected component into a "VDD-node" and a "GND-node," respectively, a *transistor group* (TG) is formed. Figure 4.1 shows a circuit and its TGs. Note that a primary input may form a TG by itself, e.g., B and C. The TG containing the nodes that control the pass transistors, i.e., are connected to the gate terminals of the pass transistors, are not shown.

The relationship between two TGs can be defined by the interconnections through the gate terminals of transistors in the original circuit. Due to the property that the gate voltage controls the conducting status of the channel but not vice versa, each interconnection between two TGs is unidirectional and can be considered as an input or output of the corresponding TGs. A transistor group *TA* is said to have *direct control* on another transistor group *TB* if one of the outputs of *TA* is one of the inputs of *TB*. If there exists a sequence of transistor groups $T_1, T_2, ..., T_n$, $n \geq 2$ such that $TA = T_1$, $TB = T_n$, and T_{i-1} has direct control on T_i for $i = 2, ..., n$, then it is said that *TA* can *control TB*. For example, in Figure 4.1 TG_1 has direct control on TG_2 (through O_1) and TG_2 has direct control on TG_3 (through O_3), hence TG_1 can control TG_3. Two TGs are said to be *related* if one of them can control the other. If neither of them can control the other, they are *unrelated*. If both can control each other, then a *control loop* exists.

The following procedure is used for the partitioning scheme described above. This procedure is based on a non-recursive *depth-first search* algorithm [102]. Step 2 is used to identify all primary input nodes that are not connected to the drain or source of any transistor. Each of these nodes forms a *trivial* transistor group. Step 3 identifies each *non-trivial* transistor group with the help of a *first-in-last-out stack* S which is used for recording the nodes that have been visited but not yet expanded. When a search branch encounters the *VDD*, *GND* or a used node, it stops. After procedure **Partition** terminates each trivial TG contains a PI node and each non-trivial TG contains transistors that are

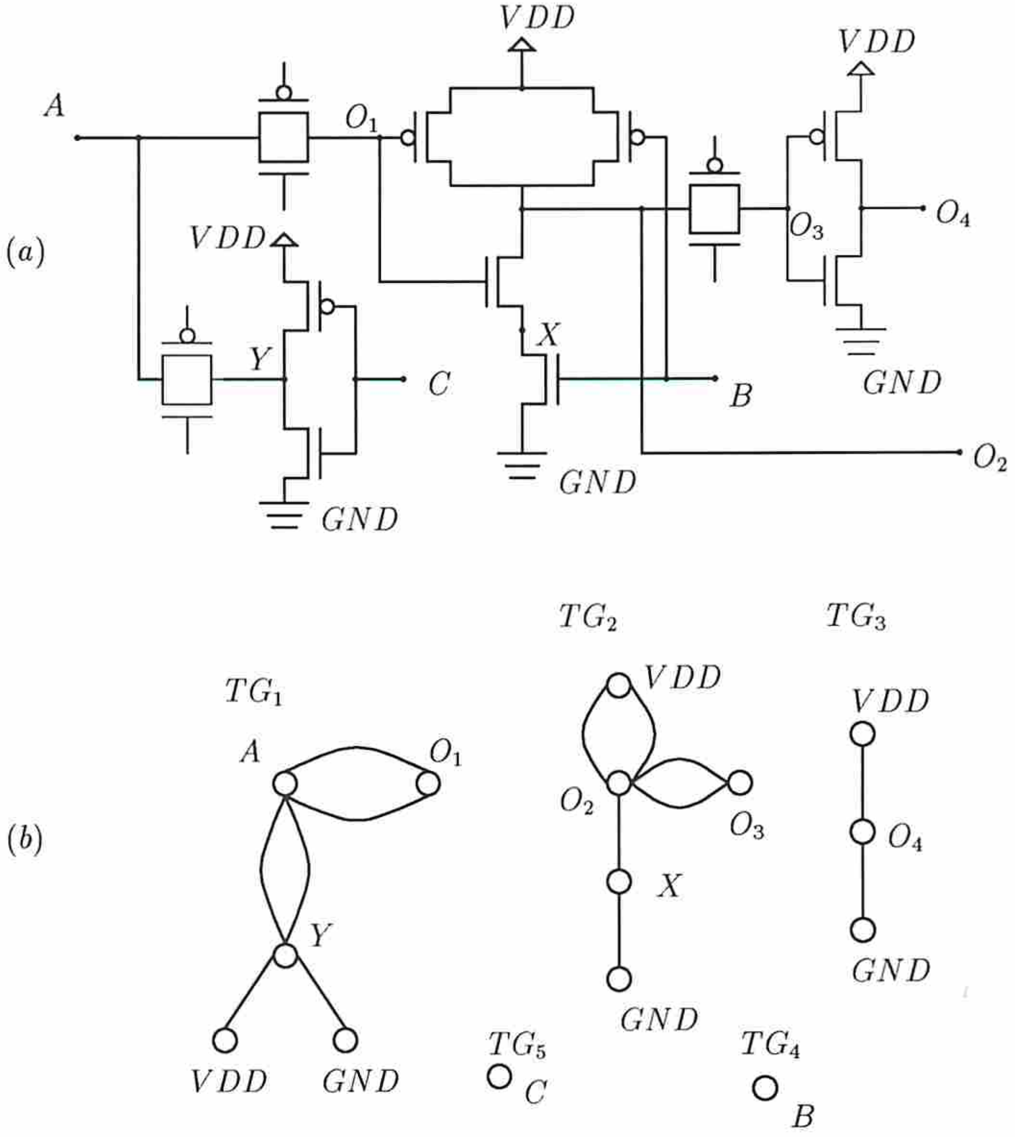


Figure 4.1: Partitioning of a CMOS circuit: (a) original circuit (b) partitioning of the circuit

"channel-connected." Charge sharing can occur only between nodes in the same TG.

Procedure **Partition**$(C = (N, T, I))$
Inputs: C is the circuit to be partitioned which is described by its set of nodes N, set of transistors T and set of primary inputs I. Each node v in N is associated with a list of transistors connected to v. Each transistor is associated with its drain, source and gate nodes.
Outputs: Each primary input not connected to any transistor drain or source becomes a trivial transistor group. All other transistors are clustered into channel-connected components.

```
1. Mark all nodes in $N$ and transistors in $T$ as unused.
2. $i = 0$;
   For each primary input node $v$ in $I$, if no transistor drain or source is connected to $v$, do:
   (a) $i = i + 1$;
   (b) Mark $v$ as used;
   (c) $TTG_i = \{v\}$.
3. $j = 0$;
   While there exist unused transistors, do:
   (a) $j = j + 1$;
   (b) $TG_j = \Phi$ (empty set);
   (c) Get one unused transistor $t$. Let its source and drain nodes be $a$ and $b$;
       if $(a \neq GND)$ and $(a \neq VDD)$, do:
         i. mark $a$ as used;
         ii. push $a$ into node stack $S$;
       else:
         i. mark $b$ as used;
         ii. push $b$ into node stack $S$;
   (d) while ($S$ is not empty), do:
         i. pop a node $v$ from $S$;
```

```
ii. for each unused transistor $t$ connected to $v$, do
    a. $TG_j = TG_j \cup \{t\}$;
    b. mark $t$ as used;
    c. let the other end node of $t$ be $u$.
       if $u$ is unused and $u \neq VDD$ or $GND$, mark $u$ as used and
       push $u$ into stack $S$;
```

End **Partition**

4.2 Complex and Primitive Gate Identification

Partitioning divides a circuit into a number of disjoint groups such that each of the interface lines for a group is unidirectional. This enables logic implication, fault propagation and backtracing procedures used during test generation to be directed, and hence more efficient. The partitioning method described in Section 4.1 guarantees that any two nodes which can be connected to each other through conducting transistors are in the same group. This, however, may not establish the direction of all the lines in the circuit.

For the purpose of test generation it is possible to identify more unidirectional lines in the circuit. For example, the output line of a primitive or complex gate should never be treated as an "input" to this gate. Thus if complex gates and primitive gates inside transistor groups can be identified, then the I/O relation of these gates can be used to speed up the switch level test generation. In practice many transistor groups are actually complex gates or even primitive gates, therefore the identification of these gates can potentially reduce the switch level test generation complexity to an extent that is close to the gate level test generation complexity.

The following procedure identifies all complex and primitive gates inside a nontrivial transistor group. The input $TG = (V, E)$ can be easily constructed from each TG_j obtained in procedure **Partition**. The *breadth-first search* mechanism used in the procedure is similar to the depth-first search mechanism used in procedure **Partition**. The only difference is that the first-in-last-out stack used in depth-first-search is replaced by a *first-in-first-out* queue in breadth-first search. A *critical* node in the procedure is defined as a node which connects both N-type and P-type transistors.

Procedure **GateIden**($TG = (V, E), CV$)
Inputs: TG is a nontrivial transistor group. CV contains all critical nodes in TG.
Outputs: All complex and primitive gates in TG.

1. Starting with the *VDD* node, do a breadth-first search through P-type transistors with each search branch stopping at a traversed node or at a *critical node*. When a branch b stops at a traversed node v that is in a different branch from b, the subgraph that has been traversed by b (connected to b without going through *VDD*) is merged with the branch containing v. When all branches stop, each branch containing exactly one critical node in CV forms a *pull up network* (*PU*). If a PU consists of only one path from *VDD* to the critical node, then mark this PU as a pull up part of a NOR gate. If all transistors in a PU connect both *VDD* and the critical node, then mark this PU as a pull up part of a NAND gate.

2. Starting with the *GND* node, do a similar breadth-first search as in Step 1 through N-type transistors. This step forms a set of *pull down networks* (*PDs*). If a PD has only one path from *GND* to its critical node, then mark this PD as a pull down part of a NAND gate. If all transistors in a PD connect both *GND* and the critical node, then mark this PD as a pull down part of a NOR gate.

3. For each visited critical node, if both of its corresponding PU and PD exist, then form a complex gate as the union of the corresponding PU and PD. If the following conditions are satisfied then mark this gate as an inverter.

   (a) Both PU and PD of the complex gate contain only one transistor.

   (b) The nodes connected to the gate terminals of these two transistors are the same.

   Else if the following conditions are satisfied, then mark the complex gate as NOR (NAND) gate.

   (a) The PU and PD have the same number of transistors.

(b) Both PU and PD are marked as a part of a NOR (NAND) gate.

(c) Each node connected to the gate terminal of a transistor in PU is also connected to the gate terminal of a transistor in PD, and vice versa.

End **GateIden**

4.3 Level Assignment of Transistor Groups

During test generation it is often required to know the distance between a node and the closest primary input or output. To provide this information at the switch level, the following method is used.

A directed graph DG is formed whose vertices correspond to the transistor groups and whose edges correspond to the interface lines among transistor groups. Each trivial transistor group is also a vertex. Each edge in DG is directed since each interface line among transistor groups in the circuit is unidirectional. The *distance* between two vertices in the graph is defined as the minimum number of edges required to traverse from one vertex to the other vertex. The following procedure computes the distance between any vertex and its closest primary input vertex, where a primary input vertex corresponds to a transistor group containing a primary input node. A similar procedure can be used to assign the distance between transistor group and its closest primary output.

Procedure **Level**($DG = (V, E)$)
Inputs: DG is a directed graph with each vertex corresponding to a transistor group and each edge corresponding to an interface line between two groups.
Outputs: A level to each transistor group.

1. Add one source vertex s to DG and add new edges from the source vertex to each primary input vertex.

2. Starting with s do a breadth-first search and assign the *breadth-first search level* to each vertex, where the breadth-first search level is defined as follows:

(a) The level of s is 0.

(b) When traversing through an edge from a vertex u to a vertex v, if v has not been assigned a level and the level of u is i, then assign level $i+1$ to vertex v.

End **Level**

After assigning levels to all transistor groups, each transistor and node inside a transistor group are assigned the same level as the transistor group to indicate the distance between individual node or transistor and its closest primary input or output nodes.

4.4 Transistor Signal Flow Direction Assignment

4.4.1 Introduction

In this section a novel approach to the signal flow direction assignment problem (DAP) is presented. The DAP is modeled as a *two-paths problem (TPP)* in an undirected graph, called the ST-graph. A general TPP is defined as follows: *Given 4 nodes s_1, t_1, s_2, t_2 in an undirected graph G, determine whether there exist two vertex-disjoint paths in G, one from s_1 to t_1 and the other from s_2 to t_2.* If there exists an edge e between t_1 and s_2, then the TPP is equivalent to the problem of determining the existence of a simple path from s_1 to t_2 which traverses e from t_1 to s_2. Existing algorithms for the TPP and the deficiencies in applying them directly to the DAP will be discussed in Section 4.4.2.

Instead of using existing algorithms for a general TPP, a sequence of efficient algorithms to deal with the TPP in an ST-graph have been developed. The algorithms take into account the following facts: 1) in practice, most transistors are designed to be unidirectional; and 2) most circuits have a series-parallel structure. For example, to implement a function $f = \overline{C_1 + C_2 + \ldots + C_n}$, where each $C_i = l_{i_1} l_{i_2} \ldots l_{i_{im}}$ is a product term of input literals, the pull-down network will be constructed by connecting n C_i networks in parallel, where each C_i network is formed by serially connected transistors controlled by $l_{i_1}, l_{i_2}, \ldots, l_{i_{im}}$. It will be shown that all transistors in a circuit constructed in a series-parallel manner can be identified as unidirectional in linear time by the first algorithm, called *PS-reduction*, if the number of transistors connected to each circuit node is bounded by a constant.

Even for a circuit that is not constructed in this manner, it is still possible that its corresponding ST-graph has the series-parallel property, and thus the direction assignment problem for this circuit can be solved in linear time. For a circuit whose corresponding ST-graph is not a series-parallel graph, the algorithm still manages to assign directions to most transistors in linear time. In addition, for all these circuits the ST-graphs will be reduced to much smaller graphs, thereby simplifying the task of solving the TPP for the remaining transistors.

If the reduced graph obtained by applying PS-reduction to the original ST-graph contains more than one edge, further processing is needed. The second algorithm uses a divide and conquer approach to recursively partition the reduced graph into smaller components and identify a set of special nodes called *local articulation points*. It is shown that all the transistors connected to these special nodes are unidirectional. This step, called the *LAP-identification* (*local articulation point identification*), takes $O(e_1 n_1)$ time, where e_1 and n_1 are the numbers of edges and vertices in the reduced graph, respectively.

The third algorithm, called *AE-cut*, is then used for each remaining transistor to identify more unidirectional transistors. AE-cut works on the components identified by LAP-identification. It runs in $O(e_2)$ time, where e_2 is the number of edges in the smallest component containing the edge.

It is possible that the above algorithms may fail to assign directions to certain unidirectional and bidirectional transistors. Two theorems presented in this section make it possible to use the direction information obtained by the above algorithms to assign direction to some of these transistors. These theorems lead to two simple yet powerful procedures, called *neighbor-triggered assignment*, which can identify most bidirectional transistors immediately.

For the remaining transistors, it is viable to use a computationally expensive algorithm because, having used the algorithms just mentioned, (1) the graph to which the TPP to be applied is considerably simpler than the original ST-graph, and (2) only a few unassigned transistors rather than all the transistors in the circuit need be dealt with. Several approaches for dealing with these transistors will be discussed.

The algorithms presented in this section can be applied to all *static* circuits and most *non-static* circuits. A design criteria is given which, when met, guarantees the applicability of these algorithms to non-static circuits.

All the techniques mentioned so far are based solely on the structural information of a circuit. It is found that for some circuits such as a barrel shifter, all the above techniques will fail to identify some unidirectional transistors because no semantics are considered. We have developed another technique to overcome this problem. This technique can solve the direction assignment problem in barrel shifters as well as many circuits whose transistor directions rely on circuit semantics.

4.4.2 Existing TPP Algorithms

Polynomial time algorithms for the TPP have been proposed in the literature [103, 104, 105]. In [103] the emphasis is on the derivation of the worst case complexity. A complex analysis is given to show that the TPP can be solved in $O(ne)$ time for each edge in the graph, where n and e are the numbers of nodes and edges, respectively. This analysis involves six successive reductions each of which has the form "One may assume the graph has a property X, otherwise the problem is either already known to be solvable or can be transformed into a solvable one."

Another algorithm for the TPP has been proposed in [104]. This algorithm requires a procedure that repeatedly examines whether there exists a subgraph $G' = (V', E')$ of $G = (V, E)$ such that (1) G' is connected, (2) V' does not contain s_1, t_1, s_2, t_2, and (3) $|\delta(V')| \leq 3$, where $\delta(V') = \{x | (x, v) \in E, x \notin V', v \in V'\}$. In addition this algorithm requires a modified planarity check on a transformed graph in order to determine the existence of two vertex-disjoint paths. Though easier to implement, this algorithm is more computationally intensive than the one described in [103].

In [105] an $O(ne)$ algorithm is developed which can solve the TPP for all edges in an undirected graph. This algorithm is probably the most efficient algorithm that can be developed since any further improvement implies a better algorithm for a general TPP exists. However this algorithm again is quite difficult to implement. In this dissertation the emphasis is on the direction assignment for practical circuits. Therefore circuit properties are taken into account and efficient and easy-to-implement algorithms are developed.

4.4.3 Circuit Model and Problem Formulation

For ease of exposition, it is initially assumed that the circuits under consideration are static, i.e., the operation of circuits does not rely on charge retension or charge sharing effects. Thus, whenever a node affects the circuit operation, it must be connected to a power source (*VDD*, *GND*, or a primary input) through a path consisting of conducting transistors. The extensions of this method to non-static circuits will be discussed in Section 4.4.8.

A circuit is modeled as follows. After the partitioning procedure described in Section 4.1, for each non-trivial TG (i.e., a TG containing at least one transistor), the sets of *input* and *output nodes* are defined as follows. The input node set contains nodes that act as the source of signal flow to the corresponding TG. For static circuits an input node is either *VDD*, *GND* or a primary input connected to the drain or source of a transistor in the TG. The output node set contains nodes that act as the destination of signal flow. An output node is either a primary output or a node which controls (i.e., is connected to) the gate terminal of some transistor. In Figure 4.1, A is a bidirectional port of the circuit and thus is both an input and an output node of TG_1; O_1 is an output node of TG_1; O_2 and O_3 are output nodes of TG_2; and O_4 is an output node of TG_3. Both *VDD* and *GND* are input nodes to all non-trivial TGs.

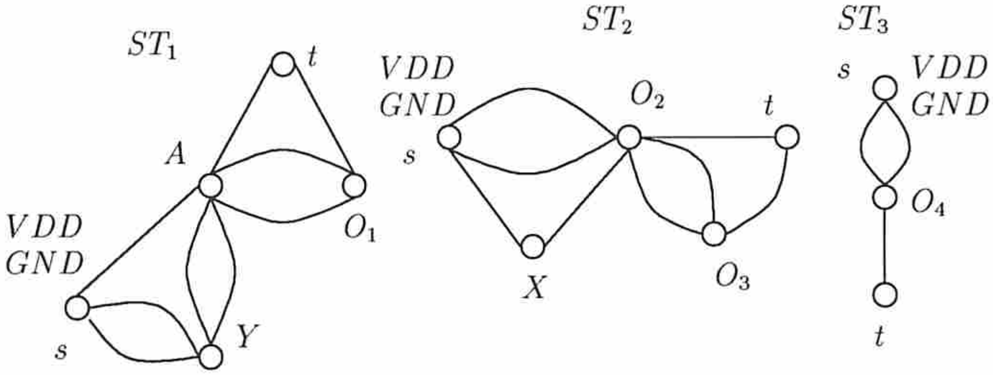


Figure 4.2: The ST-graph for TG_1, TG_2 and TG_3 of Figure 4.1.

4.4.3.1 ST-graph and Direction of Edges

Throughout this section, all graphs can be multiple graphs, i.e., two distinct nodes can have more than one edge between them. However a self loop, i.e., an edge whose two end nodes are the same, is not allowed.

An ST-graph can be defined as follows.

Definition 1 [*ST-graph*] *An undirected graph G is an ST-graph if it contains two distinct nodes s and t, and each edge in G is on at least one simple path between s and t.*

This definition characterizes some properties of an ST-graph: it must contain at least two nodes; it must be connected; by removing node s or t, but not both, it is still connected; the removal of a node can divide an ST-graph into at most two subgraphs, and s and t must be in two different subgraphs if the graph is disconnected. By this definition it is easy to see that a tree with one vertex having degree greater than or equal to 3 cannot be an ST-graph. We next describe how to construct an ST-graph for a transistor group.

Let TG_i be a nontrivial transistor group. Let I and O be its input and output node sets, $B = I \cap O$ be the set of its "bidirectional i/o" nodes, and $I' = I - O$ and $O' = O - I$ be the sets of its "input only" and "output only" nodes, respectively. An ST-graph ST_i for TG_i is constructed as follows.

1. Remove all transistors (edges) whose source and drain are both in I'.
2. Merge nodes in I' into a new node s.
3. Add a new node t.
4. Connect each node in O' to t by a new edge.
5. Connect each node in B to s and to t by two new edges.
6. Remove all edges that are not on any simple path between s and t.

It is clear that the above procedure will generate an ST-graph for any nontrivial transistor group. Figure 4.2 shows the ST-graphs of TG_1, TG_2 and TG_3 of Figure 4.1(b). An intuitive rationale for this construction procedure is as follows. For any static circuit, during steady state operation a node in I' (an input only node) always acts as an "active" node while a node in O' always acts as a "passive" node (driven by an input node). In the above procedure, any transistor connecting two active nodes is removed because such a transistor is either the result of a design error or is redundant. After removing these transistors, all nodes in I' are merged into one node s because any transistor connected to these nodes must be unidirectional (outward from these nodes). For nodes in O', the situation is different. It is not uncommon for two nodes in O' to be connected by a transistor (e.g., nodes O_2 and O_3 in Figure 4.2). To preserve these transistors while retaining only one destination node as required by the TPP, an additional destination node t is introduced into the ST-graph. When a node is both an input and output node, two edges are added such that its bidirectionality can be preserved. Finally all edges that are not on any simple path between s and t are removed because the transistors corresponding to these edges are obviously redundant.

It will be assumed that for each ST-graph constructed from a transistor graph, the degree of each node except s and t is bounded by some constant C. This is true for most circuits because of the underlying technological considerations. The degrees of s and t are not restricted because they represent a collection of nodes rather than any single "physical" node.

A formal definition of the direction of an edge in an ST-graph is given next.

Definition 2 [*Direction*] *An edge (u, v) in an ST-graph can be assigned a direction from u to v if and only if there exists two simple vertex-disjoint paths, one between s and u and the other between v and t. (u, v) is bidirectional if and only if both directions from u to v and from v to u can be assigned. If only one direction can be assigned, then the edge is unidirectional.*

This definition captures the essential nature of signal flow direction of a transistor in a static circuit where each output node of a transistor group is driven by an input node through a path of conducting transistors. For an edge (u, v) in an ST-graph to have a direction from u to v, it is essential that there exist two vertex-disjoint paths, one between s and u and the other between v and t.

The correct construction of an ST-graph is critical to the direction assignment. Figure 4.3 shows how an erroneous direction may be assigned if all input only nodes were not merged into one node. Figure 4.3(a) is a TG with two input nodes I_1 and I_2 and one output node O. Clearly both (I_1, x) and (I_2, x) are unidirectional. Figure 4.3(b) shows the situation where instead of merging I_1 and I_2 into one node, a new node s and two new edges, (s, I_1) and (s, I_2), were added. Because of the existence of the vertex-disjoint paths s to I_1 and x to t, and s to x and I_1 to t, edge (I_1, x) would be assigned as bidirectional according to Definition 1. A similar argument holds for the edge (I_2, x). Figure 4.3(c) is the correct ST-graph in which all edges can be assigned a correct direction.

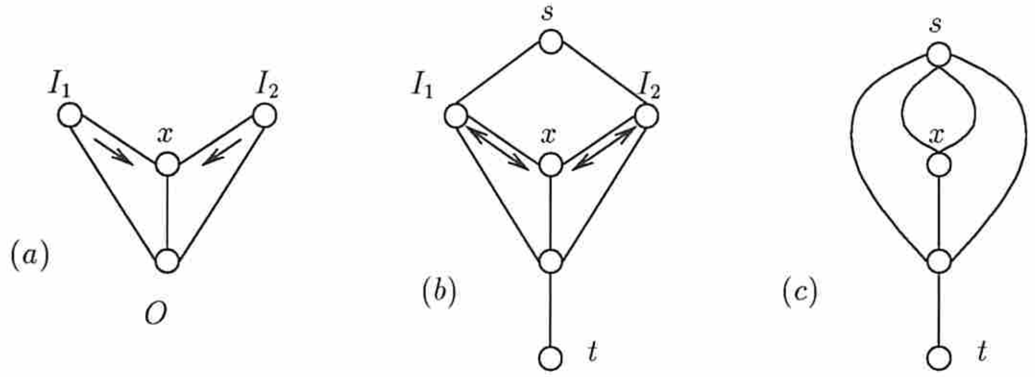


Figure 4.3: The importance of merging input nodes while forming an ST-graph

4.4.3.2 Notation

The following notation and symbols are used throughout this section.

(G, s, t): An ST-graph G with source s and destination t. Sometimes only G is used if no confusion is possible because of the context.

(u, v): An edge of an ST-graph with u and v as its end nodes.

$u \rightharpoonup v$ or $v \leftharpoonup u$: (u, v) can be assigned a direction from u to v; the other direction (from v to u) is unknown.

$u \rightarrow v$ or $v \leftarrow u$: (u, v) can be assigned a direction from u to v but the other direction is impossible.

$u \leftrightarrow v$ or $v \leftrightarrow u$: (u, v) is bidirectional.

$x \Longleftrightarrow y$: There exists a simple path between nodes x and y. $x \Longleftrightarrow x$ is always true.

$x \overset{p}{\Longleftrightarrow} y$: These exists a simple path p between nodes x and y. The length of p, denoted by $|p|$, is the number of edges in p.

$(x \Longleftrightarrow u, v \Longleftrightarrow y)$: There exist two vertex-disjoint paths, one between x and u and the other between v and y.

4.4.4 PS-reduction

This subsection describes the PS-reduction algorithm which, in general, reduces the problem size significantly. PS-reduction consists of two basic operations: P-reduction and S-reduction. It will be proved that the resulting graphs obtained from an ST-graph through different PS-reductions are isomorphic. A linear time algorithm for PS-reduction is then described. The relationship between the edge directions in the original graph and the "reduced" graph obtained after PS-reduction are established. In Section 4.4.5 we shall prove a theorem which shows that an ST-graph can be reduced to a single edge if and only if all edges in the ST-graph are unidirectional.

PS-reduction is formally defined as follows.

Definition 3 [*S-reduction*] *A node v in an ST-graph (G, s, t) is S-reducible if $v \notin \{s, t\}$ and $deg(v) = 2$. Let the two neighbors of v be u and x. The S-reduction on v (or on (u, v) and (v, x)) in G is the operation of replacing (u, v) and (v, x) with one new edge (u, x), and removing v from G (see Figure 4.4).*

Definition 4 [*P-reduction*] *If two edges e_1 and e_2 of an ST-graph (G, s, t) have the same two end nodes u and v, then these two edges are P-reducible in G; the P-reduction on e_1 and e_2 in G is the operation of replacing these two edges with one new edge e_3 which has the same two end nodes (see Figure 4.5).*

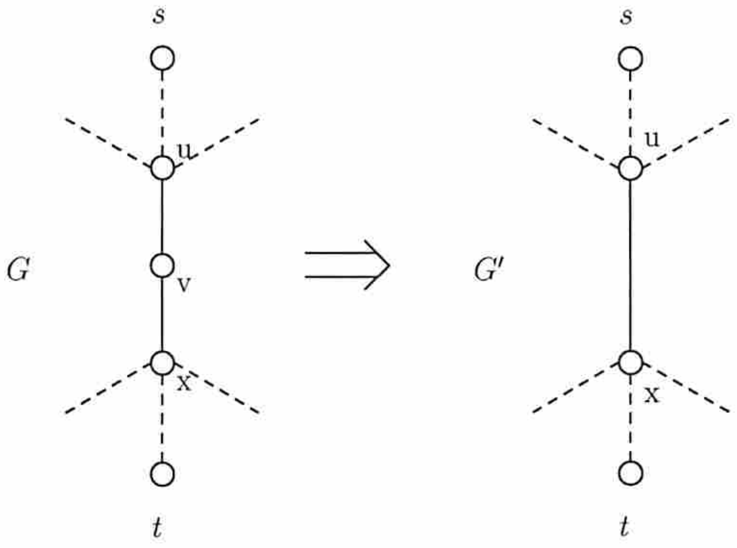


Figure 4.4: S-reduction

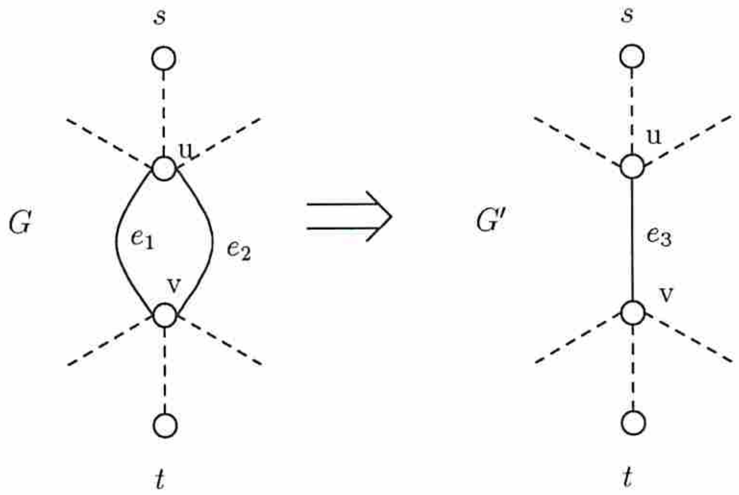


Figure 4.5: P-reduction

the closer an AP is to t, the earlier it is identified. Let $EAP_0 = t$, $EAP_i = AP_i$ for $i = 1, \ldots, n-1$, and $EAP_n = s$. Then each pair of EAP_i and $EAP_{i+1}, i = 0, \ldots, n-1$ are consecutive.

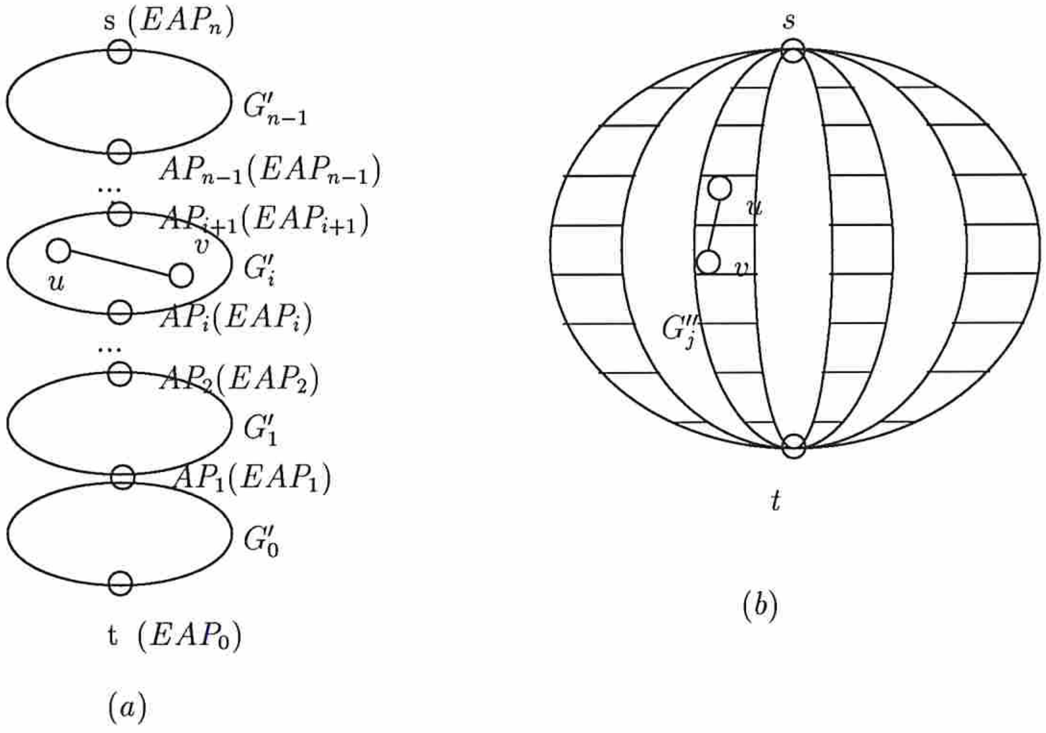


Figure 4.11: (a) Biconnected components and articulation points, and (b) slices

Definition 7 [*Biconnected component (BCC)*] *A BCC of an ST-graph G is a connected subgraph G' of G such that G' contains at least one edge, and for any two distinct edges e_1, e_2 in G, if e_1 is in G', then e_2 is also in G' if and only if there exists a simple loop (i.e., a loop that contains no preper sub-loops) in G that contains both e_1 and e_2.*

By definition, each pair of consecutive EAPs defines a BCC and each BCC is by itself an ST-graph with the two EAPs as its source and destination. A BCC is called *induced* by two consecutive EAPs if the BCC contains these two nodes. For example in Figure 4.11(a), each G'_i is induced by EAP_i and $EAP_{i+1}, i =$

$0, \ldots, n-1$. One can think of articulation points as defining "vertical" partitions of an ST-graph. The next definition, *slices*, defines the "horizon" partitions of an ST-graph.

Definition 8 [*Slice* (SLC)] *A SLC of an ST-graph* (G, s, t) *is a connected subgraph* G'' *of* G *such that* G'' *contains at least one edge, and for any two distinct edges* e_1, e_2 *in* G, *if* e_1 *is in* G'', *then* e_2 *is also in* G'' *if and only if there exists a simple path (i.e., a path in which no nodes is visited twice) in* G *that contains both* e_1 *and* e_2, *but not* s *or* t *except as end nodes.*

Figure 4.11(b) shows the SLCs of an ST-graph. Like a BCC, each SLC of an ST-graph (G, s, t) is itself an ST-graph with s and t as source and destination, respectively.

An ST-graph can either be "vertically" or "horizontally" divided, but not both. Since each BCC is an ST-graph, it may be horizontally divided into several SLCs. Each of these SLCs can then be horizontally divided into several BCCs, and again each of these BCCs can be divided into SLCs, and so on. This procedure can continue until no further partitioning is possible. The following recursive definitions formalize these structures.

Definition 9 [*Local biconnected component* $(LBCC)$] *A LBCC of an ST-graph* G *is a BCC of* G *or is a BCC of a LSLC (defined below) of* G.

Definition 10 [*Local slice* $(LSLC)$] *A LSLC of an ST-graph* G *is a SLC of* G *or is a SLC of a LBCC of* G.

Definition 11 [*Local articulation point* $((LAP)$] *A LAP of an ST-graph* G *is a source or destination node of a LBCC of* G.

Definition 12 [*Indivisible ST-graph*] *An ST-graph is indivisible if it contains only one LSLC.*

Figure 4.10 illustrates the above definitions. Figure 4.10(a) shows an ST-graph G with an articulation point D. D divides G into two ST-graphs (or BCCs) G'_1 and G'_2 as shown in Figure 4.10(b). Node D becomes both the destination node of G'_1 and the source node (denoted as D') of G'_2. G'_1 can be further divided into three SLCs G''_{11}, G''_{12}, and G''_{13} as shown in Figure 4.10(c). While G''_{12} is indivisible, G''_{11} and G''_{13} can each be divided into 4 BCCs as shown in 4.10(d). Each of these BCCs are indivisible. All the nodes that are denoted by double circles are LAPs of G.

The following three lemmas are used to prove Theorem 4, which guarantees that each edge connected to a LAP is unidirectional.

Lemma 6 *All edges connected to an EAP of an ST-graph (G, s, t) are unidirectional.*

Proof: Edges connected to s and t are clearly unidirectional with edge directions away from s and towards t. Thus only APs need be considered. Let u be an AP of G as shown in Figure 4.12. u divides G into two parts, one contains s and the other contains t. Without loss of generality consider an edge (u, v) in the part containing t. Obviously there exists two disjoint paths, one between s and u and the other between v and t. Thus $u \rightharpoonup v$ can be assigned. However any two paths p_1 and p_2 such that $s \overset{p_1}{\Longleftrightarrow} v$ and $u \overset{p_2}{\Longleftrightarrow} t$ will not be vertex-disjoint since both of them must contain u. Hence $u \rightarrow v$ is established. □

Lemma 7 *Let $EAP_i, i = 0, \ldots, n$ be the EAPs of an ST-graph (G, s, t) and G'_i be the BCC induced by EAP_i and EAP_{i+1}. For any edge (u, v) in G'_i, the direction of (u, v) in (G'_i, EAP_{i+1}, EAP_i) is the same as the direction of (u, v) in (G, s, t).*

Proof: Refer to Figure 4.11(a). Clearly $(s \Longleftrightarrow u, v \Longleftrightarrow t)$ in (G, s, t) is equivalent to $(EAP_{i+1} \Longleftrightarrow u, v \Longleftrightarrow EAP_i)$ in (G'_i, EAP_{i+1}, EAP_i) since any path from s to u (v to t) can be divided into two paths, $s \Longleftrightarrow EAP_{i+1}$ and $EAP_{i+1} \Longleftrightarrow u$

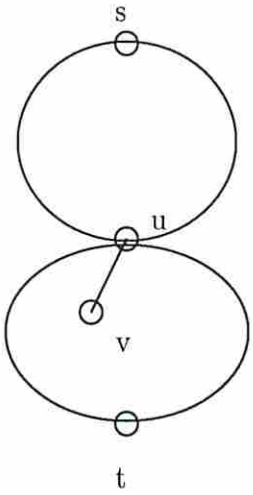


Figure 4.12: Unidirectionality of edges connected to articulation points

($v \Longleftrightarrow EAP_i$ and $EAP_i \Longleftrightarrow t$). The path $s \Longleftrightarrow EAP_{i+1}$ ($EAP_i \Longleftrightarrow t$) always exists and is independent of any path from EAP_{i+1} to u (from v to EAP_i). Similarly ($s \Longleftrightarrow v, u \Longleftrightarrow t$) in (G, s, t) is equivalent to ($EAP_{i+1} \Longleftrightarrow v, u \Longleftrightarrow EAP_i$) in (G'_i, EAP_{i+1}, EAP_i). □

Lemma 8 *Let $G''_1, G''_2, \ldots, G''_k$ be the SLCs of an ST-graph (G, s, t), and (u, v) be in G''_j. The direction of (u, v) in (G, s, t) is the same as the direction of (u, v) in (G''_j, s, t).*

Proof: Refer to Figure 4.11(b). This lemma holds since ($s \Longleftrightarrow u, v \Longleftrightarrow t$) in (G, s, t) is equivalent to ($s \Longleftrightarrow u, v \Longleftrightarrow t$) in (G''_j, s, t), and ($s \Longleftrightarrow v, u \Longleftrightarrow t$) in (G, s, t) is equivalent to ($s \Longleftrightarrow v, u \Longleftrightarrow t$) in (G''_j, s, t). □

Theorem 4 *Each edge connected to a local articulation point of an ST-graph is unidirectional.*

Proof: By induction based on Lemmas 6, 7 and 8. □

Theorem 4 establishes the importance of LAPs. In Figure 4.13 a procedure for identifying all the LAPs and assigning direction to the edges connected to LAPs is given.

```
Procedure LAP_identification($G, s, t$)
Inputs: An reduced ST-graph G with source $s$ and destination $t$.
Outputs: The directions of edges connected to LAPs.
{   $EAP_0 = t$
    Start with $s$, do depth-first search on $G$ to find the sequence of
        extended articulation points $EAP_1, EAP_2, \ldots, EAP_{n-1}, EAP_n(= s)$.
    For each BCC $G'_i$ induced by $EAP_{i+1}$ and $EAP_i$, $i = 0, 1, \ldots, n-1$, do
    {   Split $G'_i$ into $G''_1, G''_2, \ldots, G''_m$ where each $G''_j = (V''_j, E''_j)$ is a SLC of $G'_i$
        If $m = 1$ assign directions to edges connected to $EAP_i$ and $EAP_{i+1}$ in $G'_i$,
        else for each $G''_j$ do
            if $|E''_j| = 1$ then assign the direction $EAP_{i+1} \rightarrow EAP_i$ to this edge,
            else do LAP_identification($G''_j, EAP_{i+1}, EAP_i$) }
}
```

Figure 4.13: Local articulation point identification

LAP-identification recursively divides an ST-graph into smaller and smaller ST-graphs until no further division is possible. All the EAPs of G are first found using a depth-first search algorithm [102]. Each BCC induced by two consecutive EAPs is then split into slices. If only one slice exists, then the edges connected to the EAPs are assigned an appropriate direction. Otherwise for each trivial slice (i.e., a slice containing only one edge) a direction is assigned, and for each non-trivial slice LAP_identification is invoked again. This procedure continues until all local slices become indivisible.

The exact complexity of procedure **LAP-identification** is clearly problem dependent. Here only the worst case complexity is considered. Let the *level* of a LAP v in an ST-graph $G = (V, E)$, denoted as $l(v)$, be the minimum number of recursive calls to the procedure LAP_identification that are required to mark v as a LAP. The LAP level of G, denoted by $L(G)$, is defined as $\max_{v \in V} l(v)$.

For example in Figure 4.10, $l(s) = l(t) = 0$, $l(D) = 1$, $l(A) = l(B) = l(C) = 2$ and L(G)=2. The complexity of procedure **LAP-identification** is bounded by $O(|E| \cdot L(G))$. In the worst case $L(G) = O(|V|)$ since there are at most $|V|$ recursive calls. Thus the worst case complexity of **LAP-identification** is $O(|E||V|)$. Since this procedure is performed on a reduced graph that has considerably fewer edges than the original ST-graphs, the expected complexity is much less than that obtained by directly solving the TPP for the original ST-graph.

PS-reduction in conjunction with the LAP-identification procedure provides a fairly useful direction assignment technique. Their utility is enhanced by the fact that most transistors operate in a unidirectional mode. In particular it will be proved in Theorem 5 that all the edges in an ST-graph are unidirectional if and only if PS-reduction can reduce the graph to a single edge and thereby assign direction to all edges. First a *completely reducible ST-graph* is defined and a lemma to show an important property of a *partially reducible graph* is given.

Definition 13 [*Completely and partially reducible ST-graphs*] *An ST-graph is completely reducible if and only if its reduced graph contains only one edge, otherwise it is a partially reducible graph.*

Lemma 9 *An ST-graph is not completely reducible if and only if the graph shown in Figure 4.14(a) is embedded in it such that there exist two vertex disjoint paths $s \Longleftrightarrow A$ and $D \Longleftrightarrow t$; i.e., an ST-graph is not completely reducible if and only if it contains 4 distinct nodes A, B, C, D (A may be s and D may be t) with $B, C \notin \{s, t\}, A \neq t, D \neq s$, and 7 paths $s \Longleftrightarrow A$, $A \Longleftrightarrow B$, $A \Longleftrightarrow C$, $B \Longleftrightarrow C$, $B \Longleftrightarrow D$, $C \Longleftrightarrow D$, $D \Longleftrightarrow t$, such that these paths are mutually vertex-disjoint except at the ends (e.g., $B \Longleftrightarrow C$ and $B \Longleftrightarrow D$ share node B) (see Figure 4.14(b)).*

Proof: Let G be the reduced graph of a partially reducible ST-graph. First, it will be proved that there exists an indivisible LSLC in G which contains

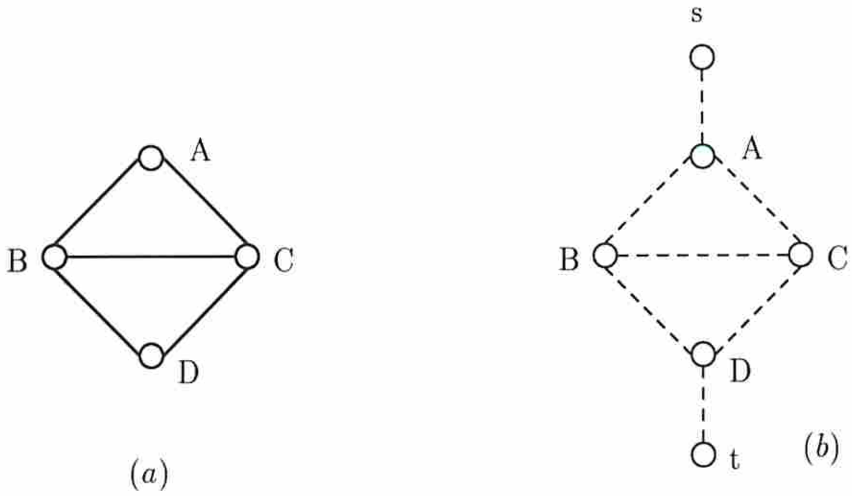


Figure 4.14: (a) Graph structure embedded in every partially reducible ST-graph, and (b) the seven paths used in Lemma 9

more than one edge. Assume every LBCC in G contains only one edge. Let v be one of the LAPs with the largest level. If $v = s$ or $v = t$ then the largest level of LAPs is 0. This, however, is a contradiction as the reduced graph would contain only one edge while G is not completely reducible. Now consider the case when $v \neq s$ and $v \neq t$. Let (G'', s', t') be the smallest LSLC of G for which v is an AP, and let u and w be two EAPs of G'' which immediately precedes and follows v respectively, as shown in Figure 4.15. Let G'_1 and G'_2 be the LBCCs induced by u, v and v, w, respectively. Since v has the largest level, no SLC of G'_1 is further dividable. Thus each SLC of G'_1 contains at most one edge. This implies that G'_1 contains only one edge because G is a reduced graph. Similarly G'_2 contains only one edge. Thus v is S-reducible. This contradicts the fact that G is a reduced graph. The contradiction is due to the assumption that each LSLC of G contains only one edge. Thus there exists at least one indivisible LSLC that contains more than one edge.

Let K be such an indivisible LSLC. Refer to Figure 4.16. Let the source and destination nodes of K be s_k and t_k respectively. s_k must have at least two

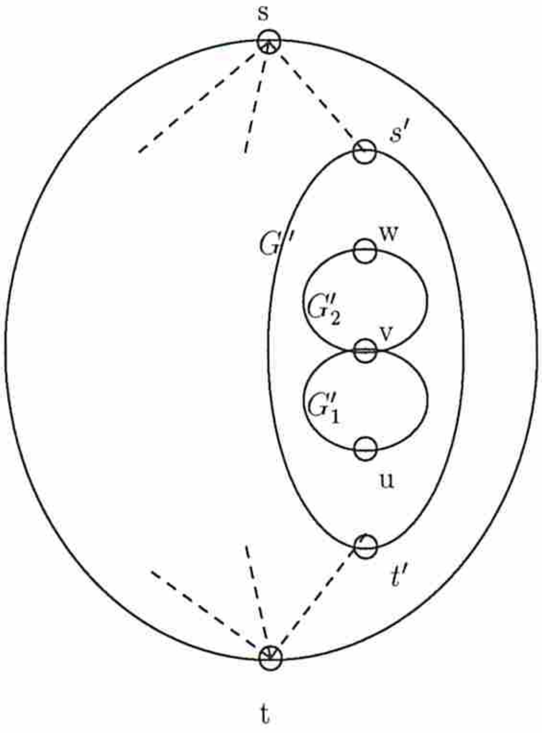


Figure 4.15: The smallest LSLC containing a LAP with the maximum level

neighbors in K, otherwise the only neighbor of s_k in K is a LAP and hence K is divisible. Let a and b be two neighbors of s_k in K. Since K is indivisible, there must exist one path, say p, from a to b which does not contain s_k or t_k. This is because if all paths from a to b pass through either s_k or t_k, then K can be divided into two slices, one contains a and the other contains b. Furthermore since K is indivisible, no node on p can be a LAP. Thus there must exist two distinct nodes, say c and d on p such that $c \overset{p_1}{\Longleftrightarrow} t_k$ and $d \overset{p_2}{\Longleftrightarrow} t_k$ and p_1, p_2 contain no nodes on p except c and d, respectively (c or d may be the same as a or b). Let e be the node on both p_1 and p_2 which is closest to c. This node must exist since p_1 and p_2 intersect at least at t_k. Now let $s_k = A$, $c = B$, $d = C$, $e = D$ then these 4 nodes satisfy the requirement in this lemma. □

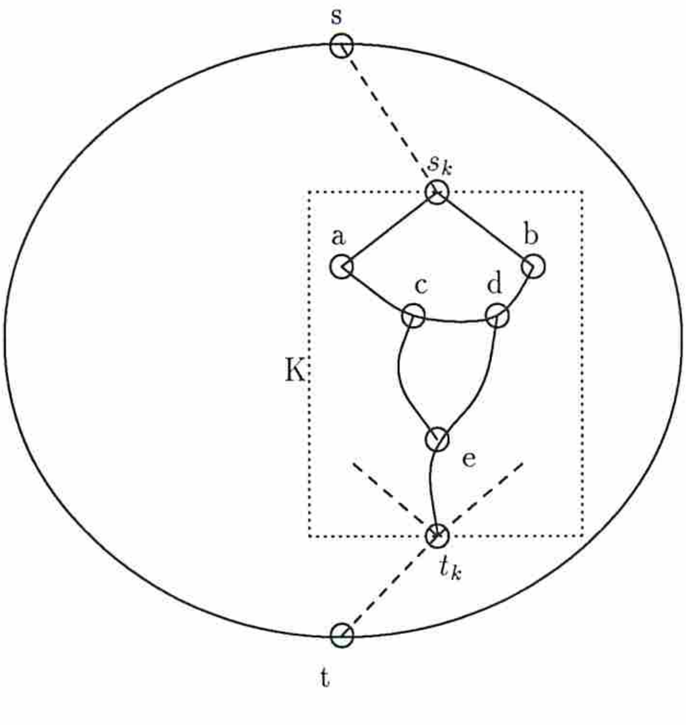


Figure 4.16: Proof of Lemma 9 and Theorem 5

Theorem 5 *All edges in an ST-graph are unidirectional if and only if the graph is completely reducible.*

Proof:

If part: By Theorem 3 if an ST-graph G is completely reducible then all edges in G are unidirectional.

Only if part: From Lemma 9, it is easy to see that every edge on the path $B \Longleftrightarrow C$ is bidirectional. □

Theorem 5 provides a fundamental result on the reducibility of an ST-graph. Much like the well-known conditions for the non-planarity of a graph, it provides the minimum sub-structure that stops an ST-graph from being reduced to an edge via a PS-reduction.

4.4.5.2 AE-cut

PS-reduction and **LAP-identification** assign direction to most but not necessarily all unidirectional transistors. For example, edges 5 and 6 in Figure 4.10 are actually unidirectional but remain unassigned. An additional algorithm that may be used to assign direction is described next. The definition *AE-cut* is first given and it will be shown that an edge in an AE-cut is unidirectional.

Definition 14 [*AE-cut*] *An AE-cut consists of an edge e and a node z in an ST-graph (G, s, t) such that if e is removed from G, z becomes an articulation point which separates s and t.*

Theorem 6 *Any edge in an AE-cut of an ST-graph is unidirectional.*

Proof: Consider Figure 4.17 where (x, y) is such an edge and z is the articulation point when (x, y) is removed. Without loss of generality, assume x and s are in the same connected component and y and t are in the other connected component of the graph when both (x, y) and z are removed. It is impossible to have two vertex-disjoint paths, one from s to y and the other from x to t because each

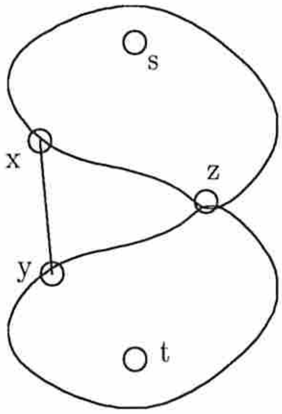


Figure 4.17: Unidirectional transistor in an AE-cut

path from s to y must go through x or z, and in either case all paths from t to x are "blocked." Thus the direction from y to x is impossible. □

For an edge in an ST-graph with $|E|$ edges, a depth-first search can find whether it is in an AE-cut in $O(|E|)$ time. By virtue of Lemmas 7 and 8 and the fact that an LBCC or a LSLC is also an ST-graph, the AE-cut algorithm only has to be applied to the smallest subgraphs that cannot be further divided using the LAP-identification procedure. Thus $|E|$ can be very small as compared to the number of edges in the original ST-graph. For example, edge 5 and node H in Figure 4.10(d) form an AE-cut of the ST-graph induced by A' and B. Thus edge 5 is unidirectional and can be assigned $E \rightarrow G$. Similarly edge 6 can be assigned $F \rightarrow H$.

4.4.5.3 Edges with Directed Neighbors

It is possible to assign directions to more edges if their neighbors have been assigned directions via the algorithms discussed so far. Consider for example the ST-graph shown in Figure 4.10(d). No direction has been assigned to edges 4, 34, 7, 12, 19, 24. In fact these edges are all bidirectional. Consider edge 34. By definition two disjoint paths $A' \Longleftrightarrow E$ and $H \Longleftrightarrow B$ to ensure $E \rightharpoonup H$, and two disjoint paths $A' \rightharpoonup H$ and $E \Longleftrightarrow B$ to ensure $H \rightharpoonup E$ must be found. The following theorem, however, enables the immediate assignment of directions to some edges that were not identified earlier.

Theorem 7 *After PS-reduction, LAP-identification and the AE-cut algorithms, if an edge (u, v) satisfies the following conditions, then the direction of $u \rightharpoonup v$ can be assigned.*

1. *(u, v) has not been assigned a direction.*
2. *There exists a neighbor w of u such that $w \rightarrow u$ has been assigned.*
3. *There exists a neighbor x of v such that $v \rightarrow x$ has been assigned.*

Proof: Since (u, v) has not been assigned a direction, neither u nor v is a LAP. Thus w, u, v and x must be in the same indivisible LSLC, say (G', s', t'). $v \rightarrow x$ can be assigned only due to (1) x is a LAP (x may be the same as t), or (2) (v, x) is in an AE-cut of G'. If x is a LAP it must be the same as t'. Similar statements hold for (w, u). Thus only two mutually disjoint cases need be considered: (1) both w and x are LAPs, or (2) at least one of w and x is not a LAP.

Case 1: $w = s', x = t'$: Since $(s' \Longleftrightarrow u, v \Longleftrightarrow t')$ in G' is true, thus $u \rightharpoonup v$ in G' follows. By Lemma 8, $u \rightharpoonup v$ in G is also true.

Case 2: $w \neq s'$ or $x \neq t'$: Without loss of generality, assume $x \neq t'$. Refer to Figure 4.18(a). Let y be the AP and G'_1 and G'_2 be the two parts of G' separated by y when (v, x) is removed. There must exist a path, say p_1, from x to t' in G'_2 that does not go through y, otherwise G'_2 will be separated into two disjoint parts by y and so will G' (see Figure 4.18(b)). This implies that y is an articulation point of G' , which contradicts the assumption that G' is indivisible. There must also exist a path, say p_2, from s' to u in G'_1 which does not pass through v in G'_1,

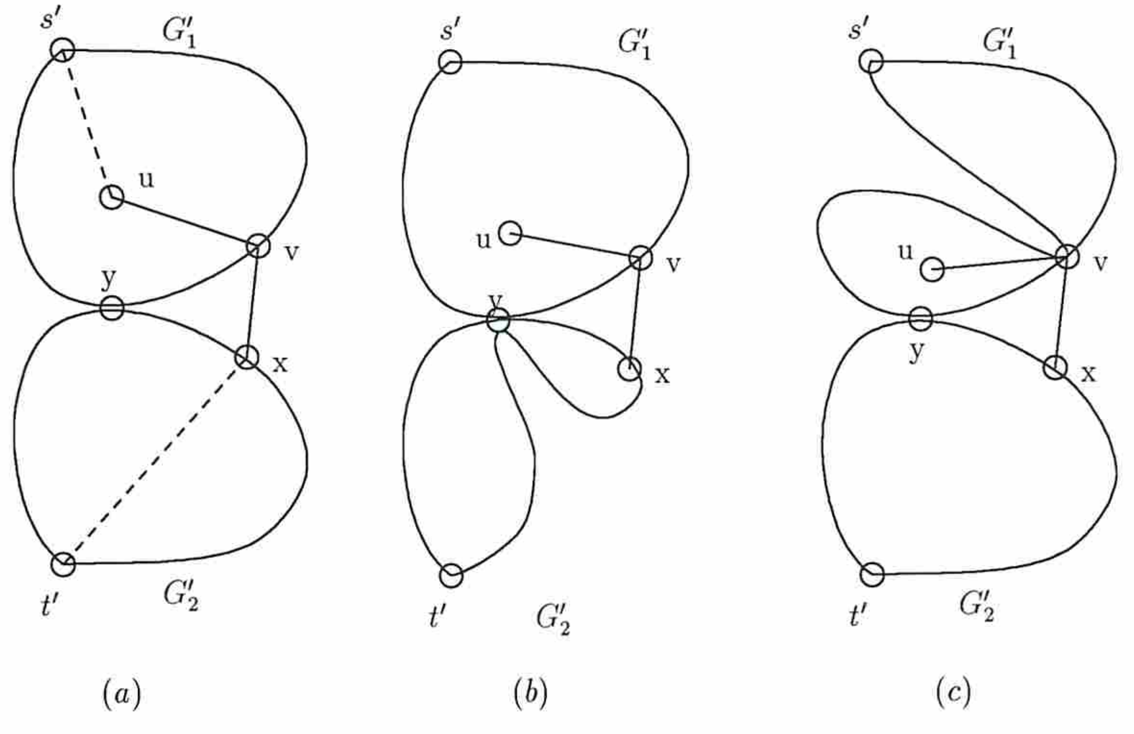


Figure 4.18: Assigning direction to (u, v) — Case 2

otherwise v will separate G_1' into two disjoint parts making it an AP of G' (see Figure 4.18(c)). Clearly p_1 and p_2 are vertex-disjoint and the theorem follows. □

By virtue of this theorem, all the remaining unassigned edges in Figure 4.10 can be immediately identified as bidirectional. Since in general most transistors in real circuits are unidirectional, the probability that Theorem 7 can be applied to an unassigned edge is quite high. It should also be pointed out that despite its simplicity, this theorem is by no means trivial. The following apparent "counter example" will highlight the difficulty.

Refer to Figure 4.19 where $w \rightarrow u$ and $v \rightarrow x$ have been assigned. Because of Theorem 7, it appears that one may assign $u \rightarrow v$. However this is not true as only $v \rightarrow u$ is possible. This "contradiction" arises because in this example the direction of (u, v) would have been assigned while **PS-reduction**, **LAP-identification** or the AE-cut algorithm are executed. Thus condition 1 of Theorem 7 is not satisfied.

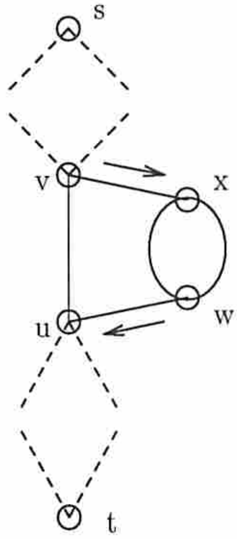


Figure 4.19: An apparent "counter" example to Theorem 6

It can be verified that when proving Case 2 of Theorem 7, only condition 2 or condition 3 of the Theorem, but not necessarily both, is required. In other words, if one of the two edges (w, u) and (v, x) has been assigned a direction ($w \rightarrow u$ or $v \rightarrow x$) by the AE-cut algorithm, then no matter whether the direction of the other edge has been assigned or not, $u \rightharpoonup v$ can be assigned. The following theorem formally states this fact.

Theorem 8 *After PS-reduction, LAP-identification and the AE-cut algorithm, if an edge (v, x) was assigned a direction $v \rightarrow x$ by the AE-cut algorithm, then any unassigned edge (u, v) can be assigned a direction $u \rightharpoonup v$ and any unassigned edge (x, y) can be assigned a direction $x \rightharpoonup y$.*

4.4.6 Remaining Edges

For most real circuits that have been studied, all the transistors could be assigned a direction using the procedures discussed so far. However, in some pathological cases unassigned edges may still exist after all these procedures have been executed. These remaining edges can be dealt with in several ways.

One can simply regard the unassigned edges as bidirectional (as was done in the case of rule-based systems [46]). Clearly, the over-all result will be pessimistic but not wrong. Or one can determine the existence of two disjoint paths via exhaustive search. Since it is possible to efficiently count the number of paths between two nodes—though enumerating them is computationally intensive—one may wish to first count the number of paths from s to u and v to t before deciding how to proceed. If the product of these two counts is not very large, one can determine the existence of two disjoint paths via exhaustive search. An advantage of this approach is that it can use the direction information obtained by earlier procedures to guide the search for disjoint paths and prune the search space. Another approach might be to use an existing algorithm for the TPP. Because the problem size has become much smaller, it is likely that the largest indivisible LSLC containing unassigned edges is quite manageable.

4.4.7 Recovery of Reduced Edges

After all edges in a reduced graph have been assigned directions, it is necessary to assign directions to the edges that have been reduced by PS-reduction. The recovery procedure is described in Figure 4.20. The basic idea is to keep a last-in-first-out stack that contains information about each reduction, viz., the type of the reduction and the edges involved. The procedure is self-explanatory.

4.4.8 Extension to Non-static Circuits

So far it is assumed that the circuits under consideration are static. In this subsection the applicability of the methods to non-static circuits is discussed. A non-static circuit is defined as a circuit whose operation relies on charge retention or charge sharing effects. Two types of non-static circuits will be discussed: *non-static logic gates* and *non-static memory cells.*

4.4.8.1 Non-static Logic Gates

According to the classification used in [1], there are mainly 4 types of non-static circuits: (1) dynamic logic, (2) clocked CMOS logic (C^2MOS), (3) CMOS domino logic, and (4) cascade voltage switch logic (CVSL). The basic structure of these logic gates are shown in Figure 4.21(a)-(d). Detailed description of their operations is beyond the scope of this study. In this study only those properties of these circuits that affect the signal flow directions of transistors are examined.

In general a non-static logic gate operates in two phases. During the precharge phase the precharge node is connected to a power source (usually *VDD*). During the evaluation phase the output node (which may or may not be the precharge node itself [1]) of the gate is either connected to an opposite

Modifications to the PS-reduction algorithm for recovery:

```
after line 2 (Initialization):
    stack_index = 0;

after line 8 (when executing an S-reduction):
    stack_index = stack_index + 1;
    stack[stack_index].type = $'S'$;
    stack[stack_index].e1 = $(u, v)$;
    stack[stack_index].e2 = $(v, x)$;

after line 13 (when executing a P-reduction):
    for $i = 2$ to $k$, do {
        stack_index = stack_index + 1;
        stack[stack_index].type = $'P'$;
        stack[stack_index].e1 = $e_1$;
        stack[stack_index].e2 = $e_i$; }
```

After completing all direction assignment:

```
Procedure Recovery
Inputs: A stack containing the information about PS-reduction.
Outputs: The directions of edges that have been reduced.
{   for $i$ = stack_index to 1, do {
    if (stack[stack_index].type = $'S'$) then
        $(u, v)$ = stack[stack_index].e1;
        $(v, x)$ = stack[stack_index].e2;
        dir = direction of $(u, x)$;
        $E' = (E' \cup (u, v) \cup (v, x)) - (u, x)$; $V' = V' \cup \{v\}$;
        direction of $(u, v)$ = direction of $(v, x)$ = dir;
    else
        $(u, v)$ = stack[stack_index].e1;
        $(u', v')$ = stack[stack_index].e2;
        $E' = E' \cup (u', v')$;
        if $(u' = u)$ then direction of $(u', v')$ = direction of $(u, v)$
                  else direction of $(u', v')$ = reverse direction of $(u, v)$ }
}
```

Figure 4.20: The "recovery" and direction assignment of the reduced edges

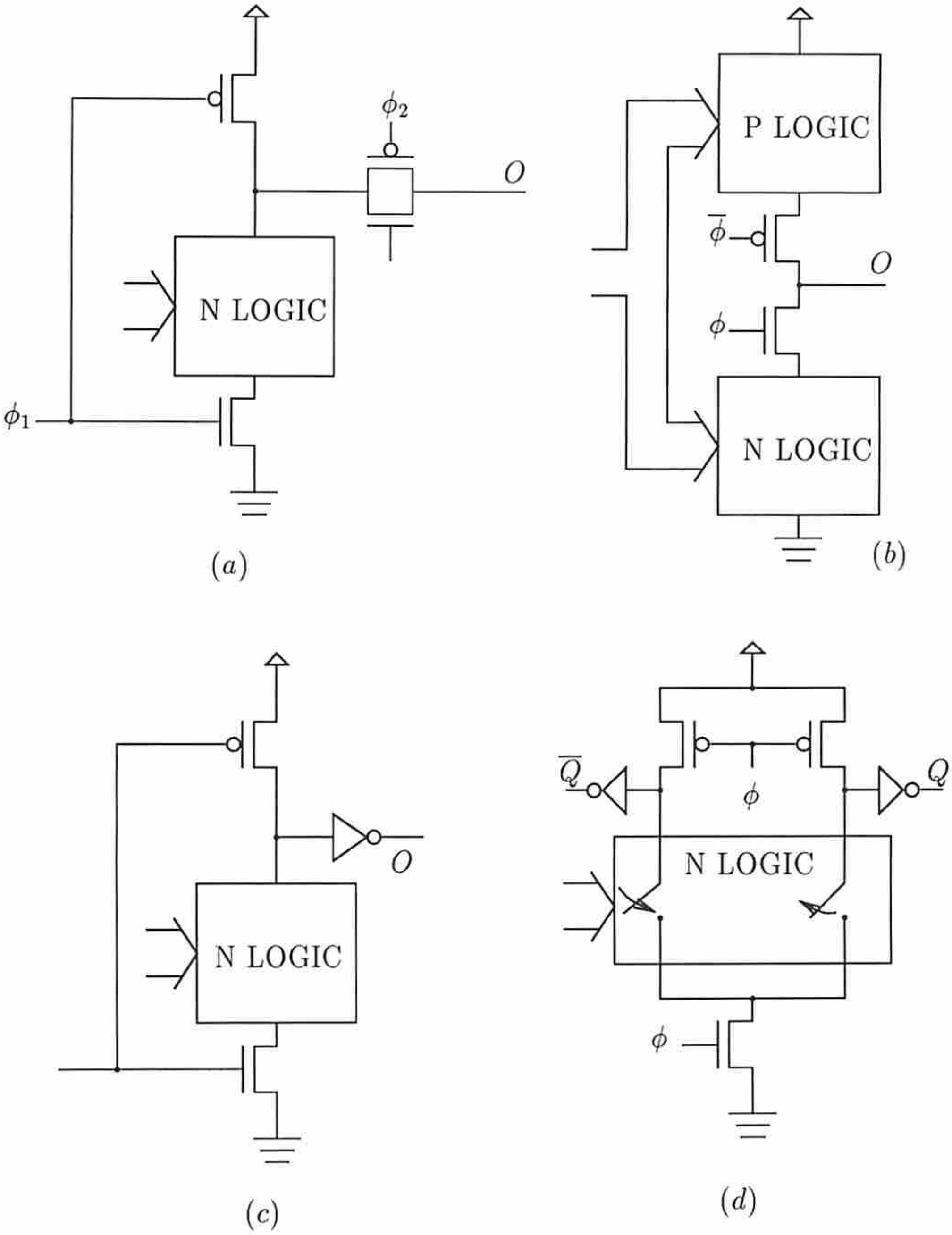


Figure 4.21: Non-static logic gates: (a) dynamic logic, (b) clocked CMOS logic, (c) domino logic, and (d) cascade voltage switch logic

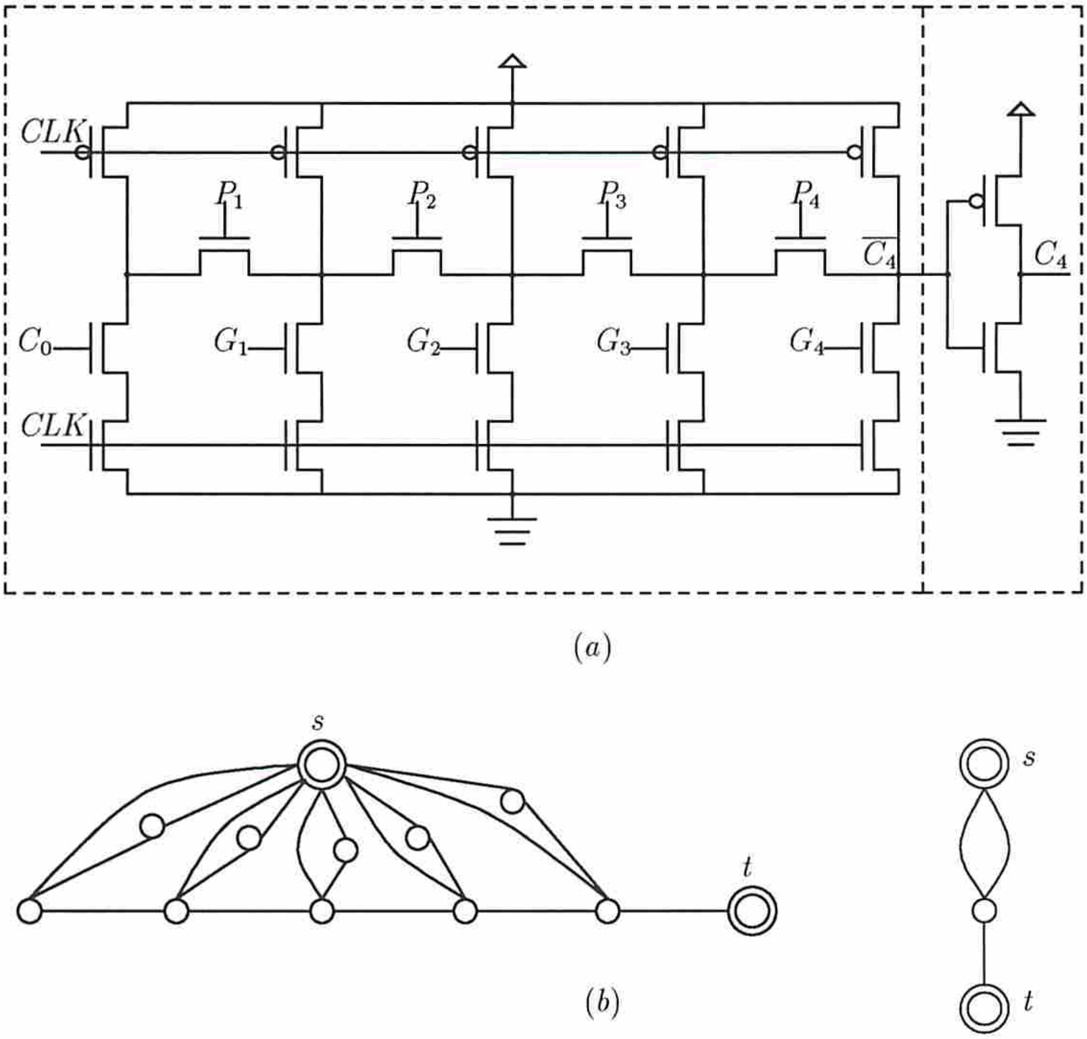
power source (usually *GND*) or is dominated by the precharge node, depending on whether a conducting path exists from the opposite power source to the precharge node.

Most precharge logic circuits share a common property: *any precharge path and any evaluation path (any path that connects the precharge node to a power source that is complementary to the power source for precharge) are disjoint except for sharing the precharge node(s)* [1]. Thus if a precharge node is not the output node of a gate, then it can be considered as an intermediate node of signal flow. If the precharge node is the same as the output node, then it is the signal flow destination. In both cases the signal flow destination is always the output node and thus the presented algorithms are applicable. For example all circuits shown in Figure 4.21 can be processed using these algorithms.

A special circuit can be used to illustrate the above arguments. Figure 4.22(a) shows a dynamic Manchester carry chain [1] that satisfies the criteria that precharge and evaluation paths are disjoint except for sharing the precharge nodes. For the left transistor group, the only signal flow destination is node $\overline{C_4}$. The two corresponding ST-graphs are shown in Figure 4.22(b). Both can be reduced to a single edge by PS-reduction. Thus all transistors can be assigned a proper direction by PS-reduction. This circuit also illustrates an interesting point mentioned earlier, viz., the original circuit graph need not be a series-parallel graph in order to obtain a series-parallel ST-graph.

4.4.8.2 Non-static Memory Cells

A typical dynamic D flip-flop design is shown in Figure 4.23 where A and B are two storage nodes. Since A and B are both connected to the gate terminal of the inverters, they are the output nodes of the corresponding transistor groups. It can be seen that A and B only serve as the signal flow destination. Thus the



(a)

(b)

Figure 4.22: A Manchester carry chain and its ST-graphs

presented direction assignment algorithms assign correct directions, as shown in Figure 4.23, to the pass transistors.

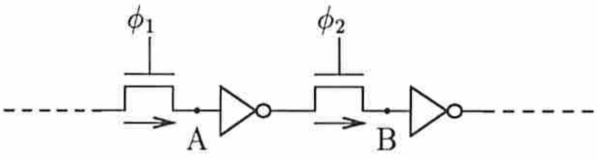


Figure 4.23: A dynamic Flip-Flop

In some designs a storage node can serve both as a source and a destination of signal flow. Consider, for example, the circuit in Figure 4.24(a) which shows a typical design of a dynamic RAM. The source and drain of transistor M are connected to form a capacitor. The value of node v can be read or written through the pass transistor PA. The reading process relies on the charge sharing between nodes v and u and the fact that v dominates u if u is not driven by any other source. While writing, u is controlled by a strong power source and the voltage value at u can overwrite the value at v. Since v can serve as both a signal flow source and destination, PA is used as a bidirectional transistor in this design. In order to correctly model this circuit two new edges incident on v are added when constructing the ST-graph. One edge connects v to s and the other connects v to t, as shown in Figure 4.24(b). Note that v is not merged into s but is just connected to s via an extra edge. The presented algorithms will now identify transistor PA as bidirectional. In general for a non-static circuit that relies on charge sharing, any storage node that dominates the charge sharing effect (e.g., v in the above example) can be modeled in this manner.

4.4.9 Experimental Results

The algorithms presented in this section have been implemented in C on a SUN 3/60. The experimental results for some sample circuits are shown in Table

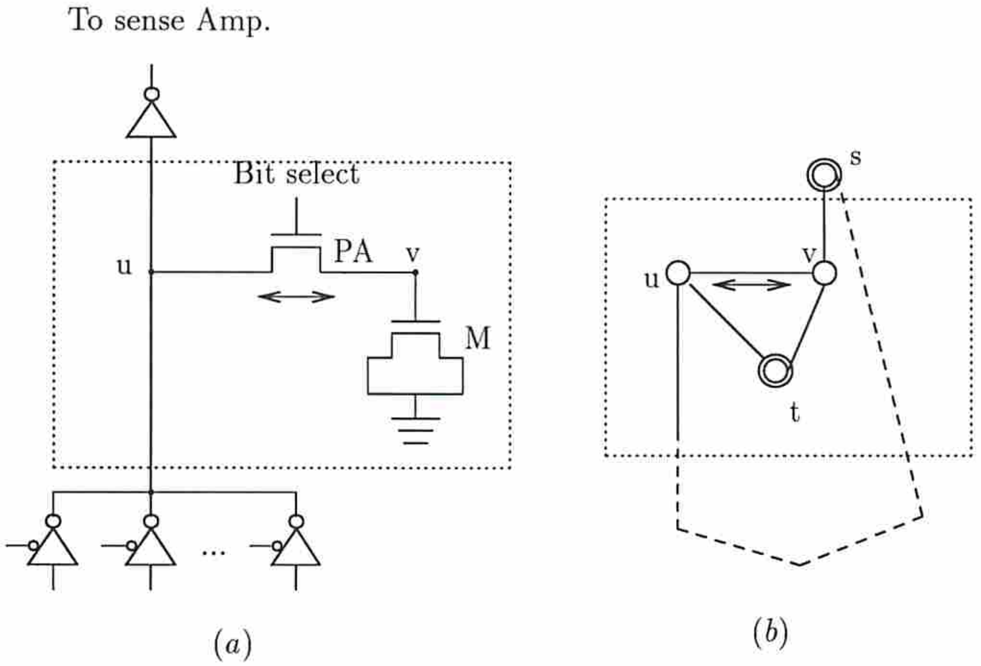


Figure 4.24: A dynamic RAM cell

4.1. The first 5 circuits (starting with a "c") are switch-level descriptions of the ISCAS-85 benchmark circuits [26]. *s1* and *nmul* are an 8-bit square/square-root processor and a 8 × 8 multiplier, respectively. The reduced ST-graph of *f13* was shown in Figure 4.10. *mem1k* is a 2-dimensional 1k bit memory chip that contains 32 dynamic input/output latches, sense amplifiers, row decoders and column decoders, and 1024 static memory cells (shown in Figure 2.2(b)(c))[106].

The results of the ISCAS-85 benchmark, *s1* and *nmul* circuits demonstrate the gain of performance due to the linearity of the PS-reduction algorithm. All transistors in these circuits were correctly identified as unidirectional by PS-reduction.

There are 27 unidirectional and 8 bidirectional transistors in *f13* (each of the edges 29, 30 and 31 corresponds to a transmission gate consisting of a

Circuit	# trans.	# uni	# bi	cpu sec.
c880	1802	1802	0	0.717
c1908	3446	3446	0	1.367
c2670	5668	5668	0	2.400
c3540	7504	7504	0	3.233
c7552	15396	15396	0	6.350
s1	1215	1215	0	0.483
nmul	4828	4828	0	1.833
f13	35	27	8	0.033
mem1k	7752	5608	2144	7.833

Table 4.1: Result of the direction assignment

P-FET and an N-FET; edges 32 and 33 were added when constructing the ST-graph because Y and Z are output nodes of the circuit). PS-reduction and LAP-identification identified 25 and AE-cut identified 2 unidirectional transistors. The 8 bidirectional transistors were all identified by the neighbor-triggered assignment algorithm.

In *mem1k* each memory cell or input latch has two bidirectional transistors and the sense amplifier has a bidirectional transistor. All these were correctly identified. The computation time for this circuit is higher than for the other circuits because 1) the degree (> 32) of the node on each Bit (or $\overline{Bit}$) line is much higher than that of any node in any other circuit that was considered, and 2) this circuit contains many bidirectional transistors.

From the table it is clear that the performance of the presented algorithms is much better than that of the rule-based system described in [46] , which typically resolve about 20,000 transistors per VAX 11/780 CPU minute.

It is worth mentioning that the algorithms, when applied to the circuit *nmul*, found that several inverters did not have an output node, i.e., the TGs corresponding to these inverters did not have a destination node as their outputs were floating. It turned out that these inverters were redundant, a situation that occurs quite often in circuits designed using standard cells. Thus, to a limited extent, the programs can also aid in circuit-debugging.

4.4.10 Semantics Consideration

One limitation of the method described in this section is that it is based solely on the circuit structure and does not consider circuit semantics. It is possible that a path from s to t contains transistors that cannot be turned on at the same time. Consider, for example, the circuit shown in Figure 4.25. The existence of paths p_1 and p_2 does not guarantee the direction $X \rightharpoonup Y$ because transistors T_1, T_2 and T_3 cannot be turned on simultaneously. This problem, in general, is very difficult to solve for two reasons. First, it is difficult to derive the boolean function which encodes the conditions that cause a transistor to turn on or off purely from a structural description of the circuit. Second, even if the activation condition for each transistor were known, determining whether all transistors in a path can be turned on simultaneously is equivalent to solving the general satisfiability problem [107], a well-known NP-complete problem.

Fortunately since the goal of direction assignment algorithms is to identify as many unidirectional transistors as possible, the above problem does not invalidate the direction assignment results. By disregarding the semantics of circuits the results obtained may be *pessimistic*, but never *wrong*, i.e., it only results in some unidirectional transistors being assigned as bidirectional, but never the other way. Furthermore once a transistor is identified as unidirectional by the algorithms, the assigned direction must be correct because the other direction is impossible.

For some circuits it is possible to consider the semantics to a limited extent. In the following a special "rule" that has been found very useful is described. By using this rule along with the techniques described in this section a well-known

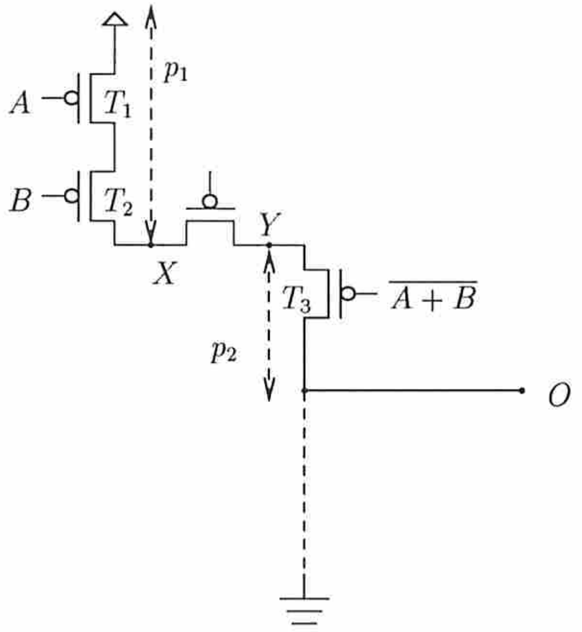


Figure 4.25: An example showing the effect of circuit semantics on TPP

difficult problem, namely assigning directions to transistors in a barrel-shifter, can be solved. Next we use a 4 to 4 barrel shifter shown in Figure 4.26(a) as an example to illustrate this method.

The circuit graph of this barrel shifter is shown in Figure 4.26(b) and consists of a 4 to 4 bipartite complete graph with 4 bus lines (nodes) on the top and 4 output lines on the bottom. Each edge is controlled by a shift line (not shown in Figure 4.26(b)). Depending on how the circuit is implemented either the 4 bus lines can be considered as primary inputs and can be merged into one source node S when constructing the ST-graph, or the 4 bus lines are driven by other gates and therefore are internal nodes rather than input nodes. The ST-graphs for these two cases are shown in Figure 4.26(c) and 4.26(d), respectively. For Figure 4.26(c) it is easy to identify all transistors as unidirectional. For Figure 4.26(d), since a new S node must be added, none of the transistors can be identified as unidirectional if only structural information is used.

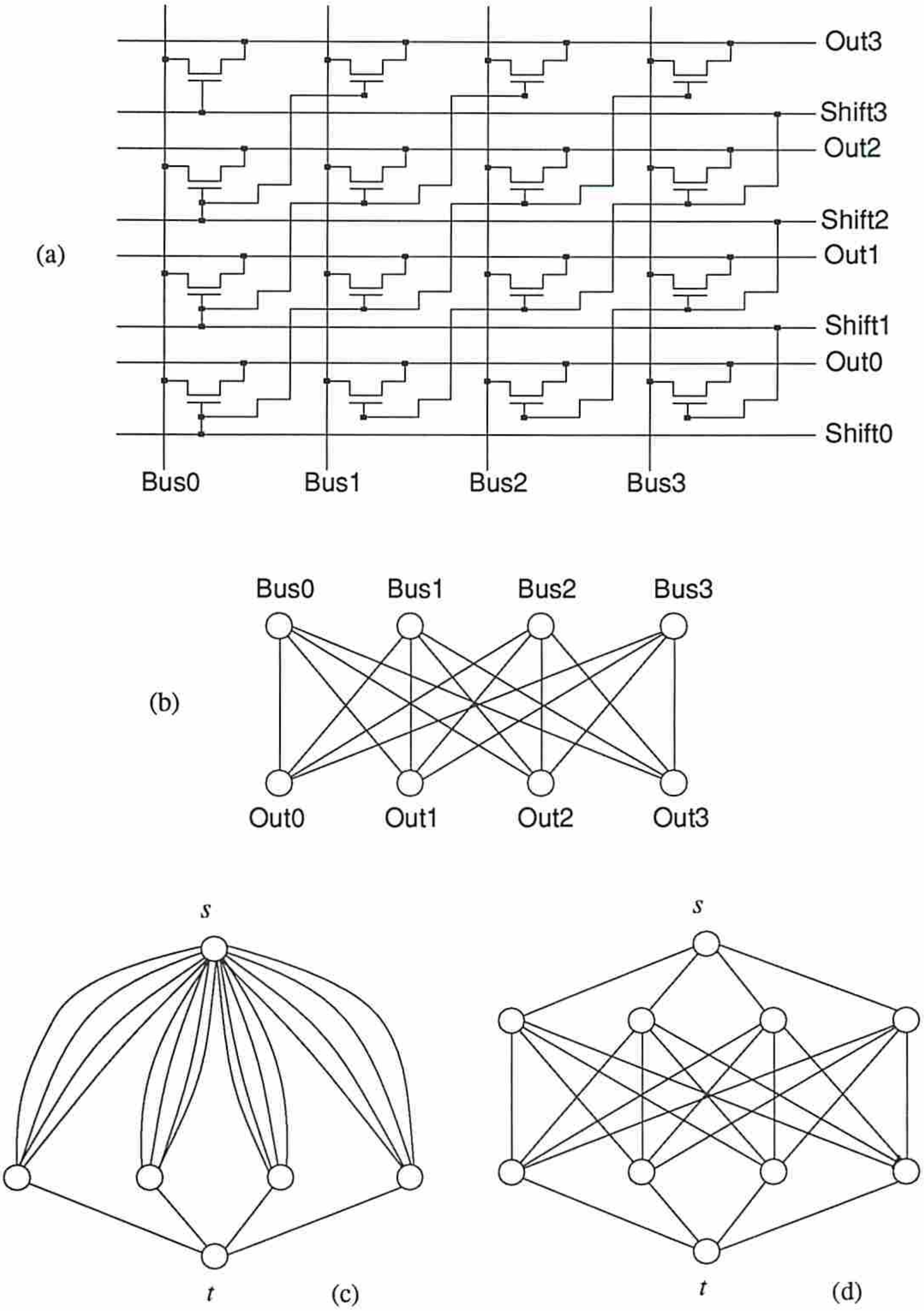


Figure 4.26: Direction assignment for a barrel shifter

Definition 5 *[PS-reduction, reduced graphs] A PS-reduction on an ST-graph G is a sequence of P-reductions and S-reductions on G such that no further P-reduction or S-reduction can be applied to the resulting graph. The resulting graph is called a reduced ST-graph of G.*

From the above definitions it is easy to verify that all ST-graphs corresponding to non-trivial TGs in Figure 4.2 can be reduced to a single edge.

Since each P- or S-reduction reduces the number of edges in the graph by one, uniform termination is guaranteed. Next it will be proved that any two different PS-reductions will reduce an ST-graph into isomorphic graphs, which establishes the desired unique termination property. To prove this, the following lemmas are needed.

Lemma 1 *Let G' be the reduced graph obtained from G via a PS-reduction PS. For each $e_i = (u_i, v_i)$ in G', there exists a corresponding subgraph G_i in G such that G_i is itself an ST-graph with v_i and u_i as its source and destination nodes. In addition, all G_i's are mutually edge-disjoint.*

Proof: Each edge $e_i = (u_i, v_i)$ in G' is an ST-graph by definition. Consider the reverse procedure of PS which recovers G from G'. To undo one S-reduction that created an edge $e = (u, v)$, a node x is added which splits e into two edges (u, x) and (x, v); to undo a P-reduction that created (u, v), one more parallel edge (u, v) is simply added. Clearly, these two operations simply transform an ST-graph into another one with the same source and destination nodes, and any two ST-graphs recovered from two edges in G' must be edge-disjoint (see Figure 4.6. Thus, the lemma follows by induction. □

Lemma 2 *Let G' be the reduced graph of G, and v be any node of G' (also of G). Then no S-reduction in* any *PS-reduction on G can be applied to v.*

Proof: If v is s or t then it cannot be S-reduced by definition. Thus it can be assumed $v \neq s$ and $v \neq t$. Since v cannot be further reduced in G', $deg(v) \geq 3$

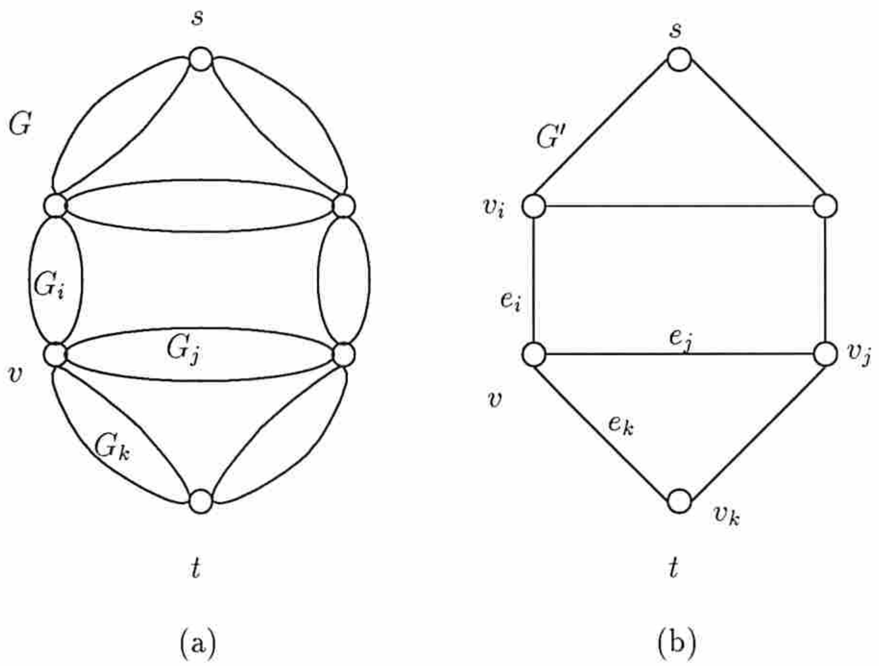


Figure 4.6: An ST-graph (a) and its reduced graph (b)

(see Figure 4.6). There must exist three distinct edges in G', say $e_i = (v, v_i)$, $e_j = (v, v_j)$ and $e_k = (v, v_k)$, such that they all have v as one end node (Figure 4.6(b)). From the previous lemma, e_i, e_j and e_k in G' correspond to three edge-disjoint subgraphs G_i, G_j and G_k in G. For v to become S-reducible, all the edges in one of these subgraphs must be completely reduced (removed) to make $deg(v) = 2$. Thus one of v_i, v_j and v_k must be S-reduced. It can be proved by induction on the total number of P- and S-reductions that none of v_i, v_j or v_k can be S-reduced by any sequence of P- or S-reductions. □

Lemma 3 *If there exists a PS-reduction that reduces an ST-graph G to a single edge, then any PS-reduction on G will reduce it to a single edge.*

Proof: This lemma can be proved by induction on $|E|$, the number of edges in the original ST-graph. The lemma obviously holds when $|E| = 1$. Assume it also holds when $|E| = n > 1$. Let PS_1 be a PS-reduction that reduces a graph with $|E| = n + 1$ into an edge (u, v). PS_1 must contain n P-reductions and

S-reductions in total. Consider the two edges e_1 and e_2 in the graph just before the last reduction of PS_1. The original graph must contain two subgraphs G_1 and G_2 (either in "parallel" or in "series") such that G_1 and G_2 can be reduced to e_1 and e_2, respectively. Obviously all reductions of PS_1 involving edges in G_1 are independent of those involving edges in G_2. By the induction hypothesis, any PS-reduction on G_1 (or G_2) reduces it to one edge. Thus any PS-reduction on G that first reduces G_1 and G_2 individually to one edge will also reduces G to a single edge.

Now consider a PS-reduction that does not first reduces G_1 and G_2 individually to one edge, i.e., it contains one or more reductions that involve one edge from G_1^d and another from G_2^d, where G_1^d and G_2^d are obtained from G_1 and G_2. Let R be the first such reduction in this PS-reduction. R has the effect of removing one edge from G_1^d and has no effect on G_2^d or vice versa. Without loss of generality assume R removes an edge e_1 from G_1^d. If R is an S-reduction then Figure 4.7(a) and (b) show this situation right before and after R is executed. It is clear that R cannot affect the applicability of any reduction on any edge in $G_1^{d'}$ (the graph obtained by removing e_1 from G_1^d) and G_2^d. Thus by induction, G_2^d and $G_1^{d'}$ (which both have less edges than $|E|$) can both be reduced to one edge and hence the lemma holds. Similar arguments show that the lemma also holds when R is a P-reduction. □

Theorem 1 *All reduced graphs of an ST-graph are isomorphic.*

Proof: Let G' be a reduced graph obtained from an ST-graph G, and $e_i, i = 1, 2, \ldots, m$ be the edges of G' which are reduced from the subgraphs $G_i, i = 1, 2, \ldots, m$ of G, respectively. By Lemma 2, no node in G' can be reduced. In addition, every node in G' is also in G. By Lemma 3 all edges in a G_i can be reduced to one edge by any PS-reduction on G_i. Since no node in G' can be reduced, no PS-reduction on G can involve two edges from two distinct $G_i s$. Thus an arbitrary PS-reduction on G is equivalent to some permutation of individual reductions on $G_i s$. Since reductions on G_i are independent of those on G_j, $i \neq j$,

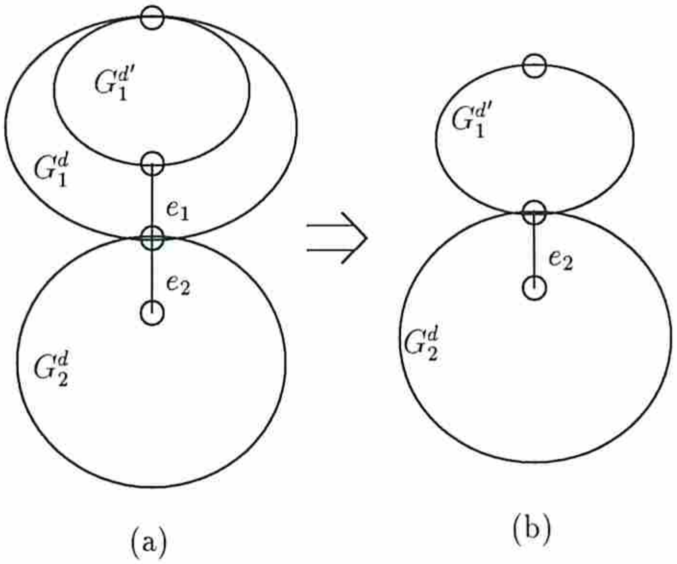


Figure 4.7: Proof of Lemma 3, where the S-reduction on e_1 and e_2 does not affect $G_1^{d'}$ and G_2^d

an arbitrary PS-reduction will independently reduce each G_i to a single edge. Thus all reduced graphs are isomorphic to G'. □

This theorem guarantees that different PS-reductions lead to the same result, i.e., the result is independent of the order in which individual P- and S-reductions are applied. By virtue of this theorem one way to obtain the reduced graph of an ST-graph is by iteratively checking whether any nodes or edges are S- or P-reducible. If either reduction can be applied, it is executed. The process continues with the new graph until no further reduction is possible. This method, however, is inefficient as it requires several passes over the graph. In Figure 4.8 a linear time algorithm that accomplishes the same task with only one pass over the graph is presented.

In this algorithm, $G(V, E)$ is the original ST-graph, U' is a list of nodes that need further processing, and V' and E' are temporary sets of nodes and edges during processing. At the end of PS-reduction V' and E' represent the final reduced graph.

```
Algorithm PS-reduction
Inputs: An ST-graph $G = (V, E)$ with source $s$ and destination $t$.
Outputs: The reduced graph of $G$.
Procedure PS_reduction($G, V, E, s, t$)
{  1)  $V' = V$; $U'$ = List of nodes in $V - \{s, t\}$; $E' = E$;
   2)  Mark all nodes in $U'$ as unreduced
   3)  while ( $U' \neq null$) do
   4)  { remove up to the first unreduced node $v$ from $U'$,
           if no such $v$ exists, go to (7)
   5)    else { if $deg(v) = 2$ then S_reduction($v$)
   6)           else for each neighbor $u$ of $v$ do
                if Parallel($u, v, p\_list$) then P_reduction($u, v, p\_list$);}}
   7)  Remove from $E'$ all but one edge whose two end nodes are $s$ and $t$.
}
Procedure S_reduction($v$)
{  8)  Mark $v$ as reduced; Let $u, x$ be such that $(u, v), (v, x) \in E'$
   9)  $E' = (E' - (u, v) - (v, x)) \cup (u, x)$; $V' = V' - \{v\}$;
  10)  if Parallel($u, x, p\_list$) then P_reduction($u, x, p\_list$)
  11)  else if ($u \neq s$ and $u \neq t$ and $deg(u) = 2$) S_reduction($u$)
  12)  else if ($x \neq s$ and $x \neq t$ and $deg(x) = 2$) S_reduction($x$)
}
Procedure P_reduction($u, v, p\_list$)
{ 13)  Let $p\_list = \{e_1, e_2, \ldots, e_k\}$
  14)  $E' = E' - e_2 - e_3 - \ldots - e_k$
  15)  $deg(u) = deg(u) - k + 1$; $deg(v) = deg(v) - k + 1$
  16)  if ($u \neq s$ and $u \neq t$ and $deg(u) = 2$) then S_reduction($u$)
  17)  else if ($v \neq s$ and $v \neq t$ and $deg(v) = 2$) then S_reduction($v$)
}
Function Parallel($u, v, p\_list$): Boolean
{ 18)  If ($u = s$ and $v = t$) or ($u = t$ and $v = s$) then return(false)
  19)  else
  20)  {    if ($u \neq s$ and $u \neq t$) then
  21)           for each edge $e$ connected to $u$, if the other end
  22)           node of $e$ is $v$ then put this edge in $p\_list$
  23)       else for each edge $e$ connected to $v$, if the other end
  24)           node of $e$ is $u$ then put this edge in $p\_list$
  25)       return($|p\_list| > 1$) }
}
```

Figure 4.8: PS-reduction algorithm

The algorithm is *event driven* with P- and S-reductions driving each other as long as possible. Procedure PS_reduction examines each node in the list U' to initiate a P- or an S-reduction. Whenever a node is examined it is removed from U' (line 4). Due to the event driven nature of the algorithm, a node may already have been marked as reduced when it is examined. In this case the node is simply removed. Thus it is guaranteed that each node is examined at most once (as a v in lines 4-6).

An S-reduction can be performed on a node v if 1) it is not s or t and 2) it has only two neighbors. If v is S-reducible, the procedure S-reduction marks it as *reduced*, removes it from V' and replaces the two edges connected to it with an edge which connects its two neighbors. After reducing a node, the procedure S-reduction initiates another P- or S-reduction if possible.

The function Parallel(u, v, p_list) determines whether a P-reduction should be applied to the edges connecting u and v. It returns a true value only if 1) $\{u, v\} \neq \{s, t\}$ and 2) the number of edges whose two end nodes are u and v is greater than 1. *p_list* is used to stored these edges. Because the degrees of s and t are not restricted, all edges which connect both s and t are not examined until the final step of PS-reduction (line 7). If only one of u and v is s or t, then only the edges connected to the non-s (or non-t) node are examined. These two steps are necessary to guarantee the linearity of the overall PS-reduction algorithm.

Procedure P-reduction(u, v, p_list) removes all but one edge in *p_list* and reduces the degrees of u and v appropriately. It also initiates another S-reduction when possible.

The correctness and linearity of this algorithm is proved in the following theorem.

Theorem 2 *The reduced graph of any ST-graph* $G = (V, E)$ *can be obtained using algorithm* **PS-reduction** *in* $O(|E|)$ *time.*

Proof of correctness: The correctness of the algorithm be proved by considering the possible sources of P-reductions and S-reductions. Two edges can be

P-reducible because: (1) they are P-reducible in the original graph, or (2) one or both of these two edges are created by an S-reduction. Note that an edge created in a P-reduction cannot be involved in another P-reduction immediately since all edges in parallel with this edge have just been removed. Both categories of P-reductions can be processed either in procedure PS_reduction or in procedure S_reduction. Note that nodes s and t are not explicitly examined in procedure PS_reduction since any P-reduction involving either s or t, but not both, will be processed when the neighbors of s and t are examined, and the P-reduction involving both s and t is processed after all other reductions have been executed. A node can undergo an S-reduction because: (1) it is S-reducible in the original graph, (2) it becomes S-reducible due to a P-reduction, and (3) it is S-reducible and an S-reduction was just executed on one of its neighbors. All these cases are taken care of by procedure **PS_reduction**.

Proof of linearity: Consider procedure PS_reduction first. The *while* loop can be executed at most $|V| - 2$ times since each node is examined at most once. Line 4 contributes at most $|V| - 2$ time steps for the *while* loop. If the condition of line 5 is *true*, then an S-reduction is executed. In each iteration of the while loop, line 6 is checked a constant number of times since it is assumed that the degree of each node (except s and t) is bounded by a constant. Line 7 contributes at most $O(|E|)$ time steps. Now consider the total time taken by procedures S_ and P_reduction. Since each type of reduction eliminates one edge, there are at most $|E| - 1$ S- and P-reductions. Hence procedures S_ and P_reduction can be executed at most $|E| - 1$ times. Since each of these procedures takes constant time, the total time taken by them is at most $O(|E|)$. □

Next the relations between the directions of edges in an ST-graph and its reduced graph are discussed.

Lemma 4 *If G' is the ST-graph obtained by executing an S-reduction on v in (G, s, t), and u and x are the two neighbors of v in G, then $u \rightharpoonup x$ in G' if and only if $u \rightharpoonup v$ and $v \rightharpoonup x$ in G, and $u \leftharpoondown x$ in G' if and only if $u \leftharpoondown v$ and $v \leftharpoondown x$ in G.*

Proof: Refer to Figure 4.4. It is obvious that the following three conditions are equivalent (s and t may be interchanged).

1. $(s \Longleftrightarrow u,\ v \Longleftrightarrow t)$ in G.
2. $(s \Longleftrightarrow v,\ x \Longleftrightarrow t)$ in G.
3. $(s \Longleftrightarrow u,\ x \Longleftrightarrow t)$ in G'.

Thus by Definition 2 the lemma follows. □

Lemma 5 *If G' is the ST-graph obtained after a P-reduction on e_1 and e_2 in (G, s, t), and e_3 is the new edge, then e_1 and e_2 in G have the same direction as e_3 has in G'.*

Proof: Refer to Figure 4.5. Since the following two conditions are equivalent (u and v are the two end nodes of e_1, e_2 and e_3; s and t may be interchanged), the lemma follows directly from Definition 2.

1. $(s \Longleftrightarrow u, v \Longleftrightarrow t)$ in G.
2. $(s \Longleftrightarrow u, v \Longleftrightarrow t)$ in G' □

Theorem 3 *Let G' be the graph obtained by executing a PS-reduction on an ST-graph G. If a subgraph of G, say G_1, is reduced to an edge e in G' by this PS-reduction, then all edges in G_1 are unidirectional in G if and only if e is unidirectional in G'.*

Proof: By induction based on Lemmas 4 and 5. □

After PS-reduction, all edges that are connected to s or t in the reduced graph can be assigned as unidirectional. By applying Theorem 3 the directions of all edges (in the original graph) that have been removed during the PS-reduction can be determined. The "recovery" of these edges can be achieved by maintaining a first-in-last-out stack at the time of PS-reduction. This procedure will be discussed after finishing all the other direction assignment algorithms.

Example: Figure 4.9 shows a static carry lookahead gate [1] and its corresponding ST-graph. It is easy to see that this ST-graph can be reduced to a single edge. Thus each transistor in this circuit is unidirectional. □

4.4.5 Manipulating the Reduced Graph

If an ST-graph cannot be reduced to one edge by PS-reduction, further processing is needed to assign directions to all edges. This subsection describes a sequence of procedures for manipulating the reduced graphs.

4.4.5.1 Local Articulation Points

Consider the reduced ST-graph shown in Figure 4.10(a). Only the edges that are connected to s or t (i.e., edges 1, 16, 28, 32, 33) can be assigned as unidirectional by PS-reduction. Many other unidirectional edges exist in this graph. In fact all edges connected to the nodes with double circles are unidirectional. A systematic method to find all these unidirectional edges is given next. First *articulation points* and some related terms in an ST-graph are defined.

Definition 6 *[Articulation point (AP), extended articulation point (EAP) and consecutive EAPs] An AP of an ST-graph is a node whose removal disconnects s and t. An EAP of an ST-graph is either s, t or an AP of this graph. Two EAPs are consecutive if every path between these two nodes contains no other EAP.*

Using a depth-first search algorithm [102] starting with s, all APs in an ST-graph can be identified in $O(|E|)$ time. Since the removal of any AP disconnects s and t, all paths from s to t must pass through all APs. Thus the depth-first search algorithm identifies all APs in the order shown in Figure 4.11(a), where

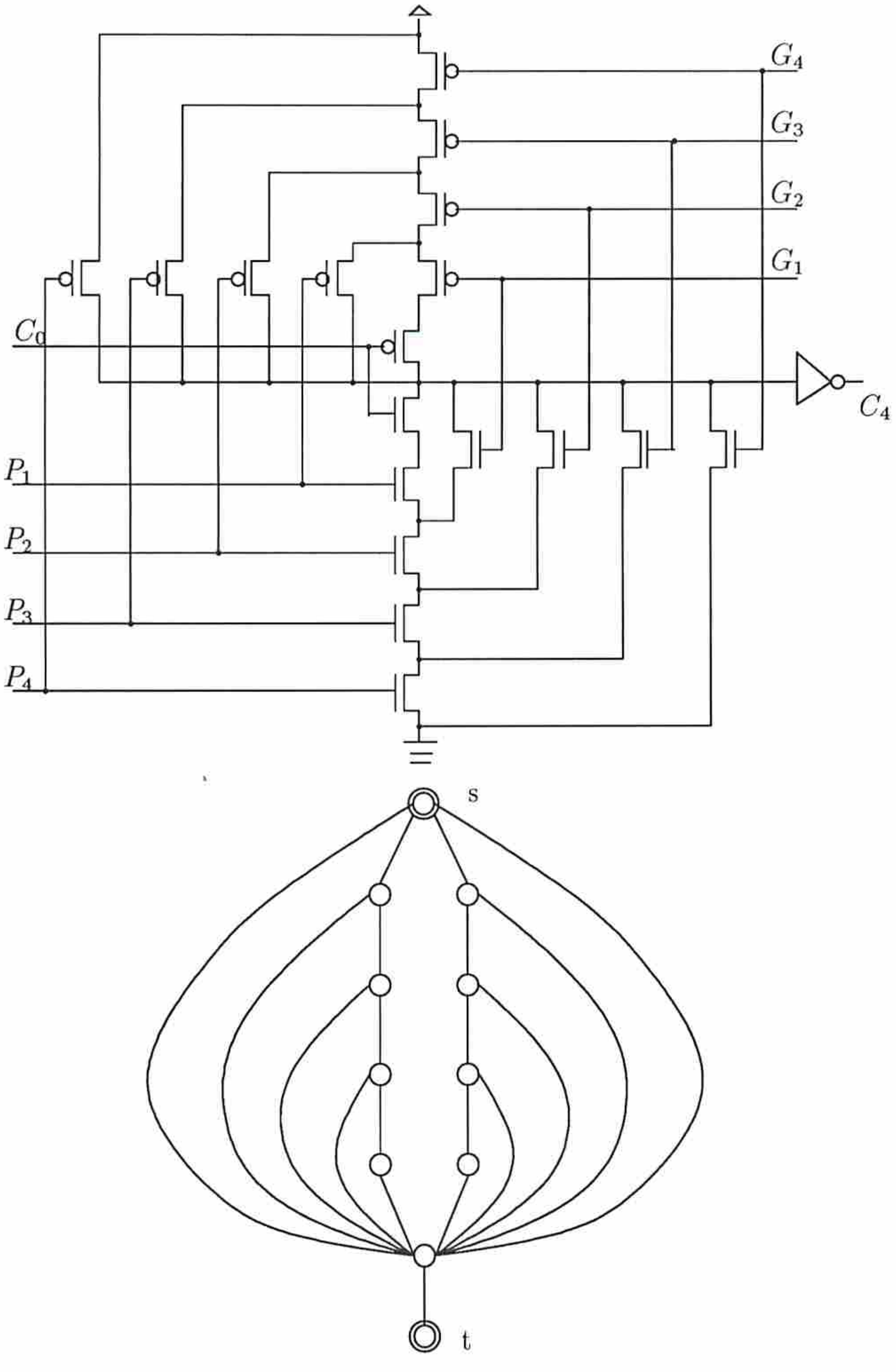


Figure 4.9: A static carry lookahead gate and its corresponding ST-graph

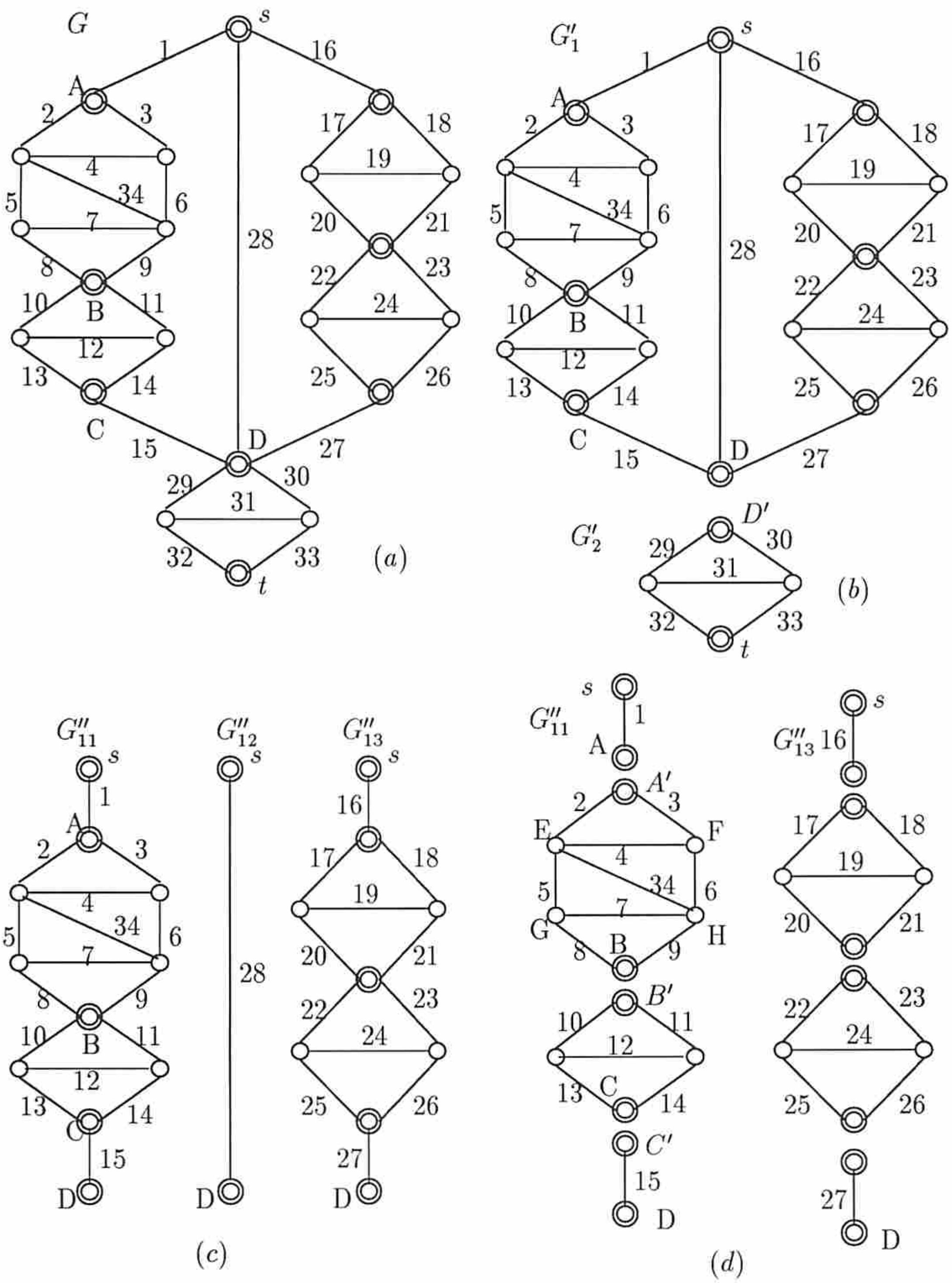


Figure 4.10: Local articulation points

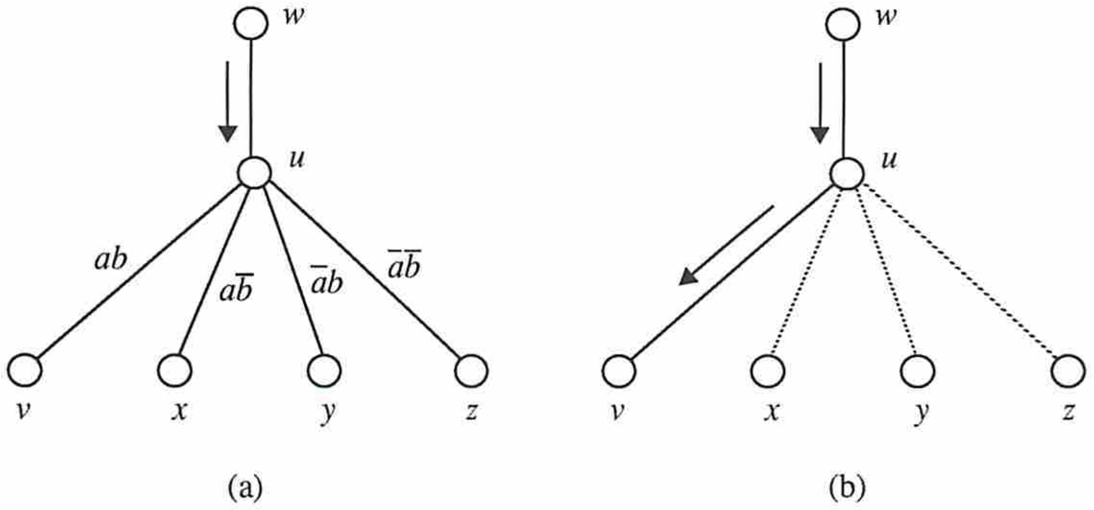


Figure 4.27: Direction assignment using implication

To make use of the semantic information, the following procedures can be applied after all direction assignment procedures using structural information are executed. For each unassigned transistor $t_1 = (u, v)$ do the following: Assign a logic value to the gate terminal of t_1 so as to turn it on. Imply this logic assignment throughout the circuit. This implication will turn off some transistors. If all non-off transistors except t_1 that are connected to u are incoming to (outgoing from) u, then t can be assigned as undirectional $u \rightarrow v (v \rightarrow u)$.

Example: Refer to Figure 4.27. Assume $w \rightarrow u$ has been assigned and (u, v), (u, x), (u, y), (u, z) are unassigned as shown in Figure 4.27(a). When turning on ab, the other three transistors must be turned off due to logic implication. Therefore $u \rightarrow v$ can be assigned as shown in Figure 4.27(b). Similarly $u \rightarrow x$, $u \rightarrow y$, $u \rightarrow z$ can be assigned.

Logic implication throughout the circuit can be done by using a method similar to those used in some test generation algorithms such as FAN and Socrates. Referring again to the ST-graph shown in Figure 4.26(d), all edges will be assigned as unidirectional since each edge connected to s is unidirectional and turning on one shift line implies that all other shift lines be turned off. The implication should be easy to carried out because the control to the shift lines of a barrel shifter is usually implemented by a demultiplexer for which logic implication can be easily done.

4.4.11 Summary

A new method for assigning directions to MOS transistors is presented. This method is based on formal graph theoretic results rather than *ad hoc* techniques. A new circuit model and a sequence of efficient algorithms are given. All static circuits and most non-static circuits can be processed using this method. By slightly modifying the model, the direction problem for some special dynamic circuits can also be solved. The experimental results show the superiority of this method over a pure rule-based system in both accuracy and computation time. Finally a special rule can be used along with the algorithms presented so that some direction assignments that require circuit semantics can be determined.

Chapter 5

Fault Analysis

This chapter deals with various fault models for CMOS circuits. An extensive analysis on IDDQ testing is given and a set of design and test rules are presented. Under these rules it is guaranteed that IDDQ testing can be safely used to detect all irredundant bridging faults. The invalidation problem that occurs when using two-pattern tests for stuck-open faults is analyzed. IDDQ testing is shown to be an effective way of solving the invalidation problem that occurs due to charge sharing. An efficient algorithm that can generate tests that cannot be invalidated by any transient effect is described.

5.1 IDDQ Testing and Bridging Faults

5.1.1 Introduction

IDDQ testing, or current supply monitoring (CSM), is an efficient and effective method for detecting CMOS bridging faults (BFs). The applicability of this technique, however, requires careful examination. Intuitively if a test vector can set two circuit nodes to complementary logic values in the fault-free circuit then a BF between these two nodes should be detected. Also a test that detects a single

BF should detect all multiple BFs that contain this single BF. Unfortunately, depending on the circuit under test, these two conclusions may be incorrect. In this section several examples are used to show the problems that may arise when using CSM. From these examples the importance of carefully analyzing the applicability of CSM becomes obvious.

A set of design and test rules for CMOS circuits are then presented. It is shown that if any one of these rules is removed, then there exist circuits for which CSM will not give correct results. It is then formally shown that if circuits are designed satisfying these rules, then all single irredundant BFs, i.e., BFs that do not affect the logic function of the circuit, can be detected by CSM, and if a test vector detects a single BF, it also detects every multiple BF that contains this single BF.

The rules presented provide a guideline for using CSM. This result, however, does not imply that CSM should be discarded when encountering a circuit that does not satisfy all the proposed rules. It will be shown that there exist circuits that do not satisfy all these rules, but due to certain other circuit properties, CSM is still applicable. For a circuit to which CSM cannot be directly applied, by adopting some strategies such as circuit partitioning, CSM is still applicable to a large portion of the circuit. There do exist some circuits for which CSM is not effective. The properties of these circuits that cause problems will be analyzed.

5.1.2 Examples Showing the Limitations of CSM

In this subsection four examples are presented to illustrate some of the limitations of CSM. Throughout this section the notation (x, y) is used to denote the BF between nodes x and y.

Example 1: This example shows that a circuit may be identified as faulty by CSM even if no fault exists. Refer to Figure 5.1. The MUX output $O = O_1$ if

$Sel = 1$, and $O = O_2$ if $Sel = 0$. When $ABC = 100$, both w and z are isolated from VDD and GND. Since $B = 0$, charge sharing between z and w occurs and the resulting voltage value v depends on their previous states. It is possible that v may be greater than the threshold voltage of transistor N but not large enough to cut off the transistor P [75]. Thus both P and N may conduct resulting in a large current through the inverter. Therefore the circuit may be identified as faulty when using CSM. However if the circuit is designed such that $Sel = 0$ whenever $ABC = 100$, the circuit output is still correct if O_2 is correct. This type of problem may occur when implementing a circuit whose I/O relation is not completely specified, e.g., when there exist *don't care* terms in the Karnaugh map of the circuit. □

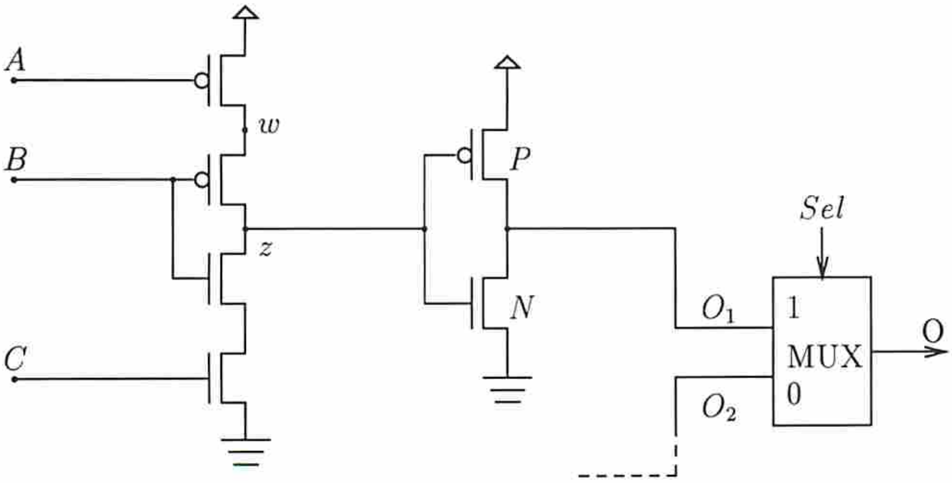


Figure 5.1: A good circuit that may be identified as a faulty circuit

Example 2: This example shows that a test that can set up two conducting paths in a fault-free circuit, one from VDD to x and the other from GND to y, does not necessarily detect the BF between x and y using CSM. Consider the circuit shown in Figure 5.2(a) that contains a bridging fault (x, y). Without (x, y), x and y can be set to complementary values by setting nodes a and b to complementary values during ϕ_1, and propagating these values to x and y, respectively, during $\overline{\phi_1}$. The equivalent circuit of Figure 5.2(a) during $\overline{\phi_1}$ is shown in Figure 5.2(b). Because of (x, y), the two loops containing x and y are connected together. Since there is no "external" control to these two loops, once the transition from ϕ_1 to $\overline{\phi_1}$ is made, x and y may reach the same stable state (either 0 or 1) and no large current can be observed during steady state. Thus the fault (x, y) cannot be detected by CSM. □

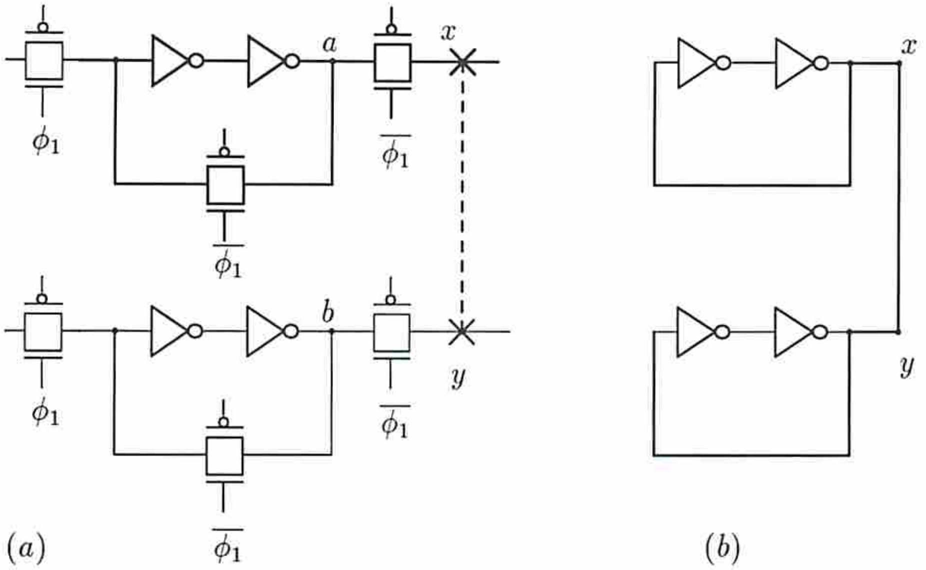


Figure 5.2: A BF that cannot be detected by just setting two shorted nodes to complementary values

Example 3: This example shows that in a sequential circuit the effect of a single BF may be masked by the existence of another BF. Referring to Figure 5.3, to detect the single BF (x, y), x and y must be set to 1 and 0, respectively. This requires A, B, C to be set to $0, 1, 0$, respectively. C can be set to 0 during $\overline{\phi_1}$ if z was set to 0 during ϕ_1. Now assume another BF (x, z) exists in the circuit. z could still be set to 0 if A was 1 during ϕ_1. When entering $\overline{\phi_1}$, a feedback loop that contains x, z, w, C and y is formed. If A and B are now 0 and 1, respectively, the final state of this feedback loop will be as follows: x, y and z are charged to 1, w becomes 0, and C becomes 1. Thus no large current is consumed during steady state since no conducting path exists between *VDD* and *GND*. □

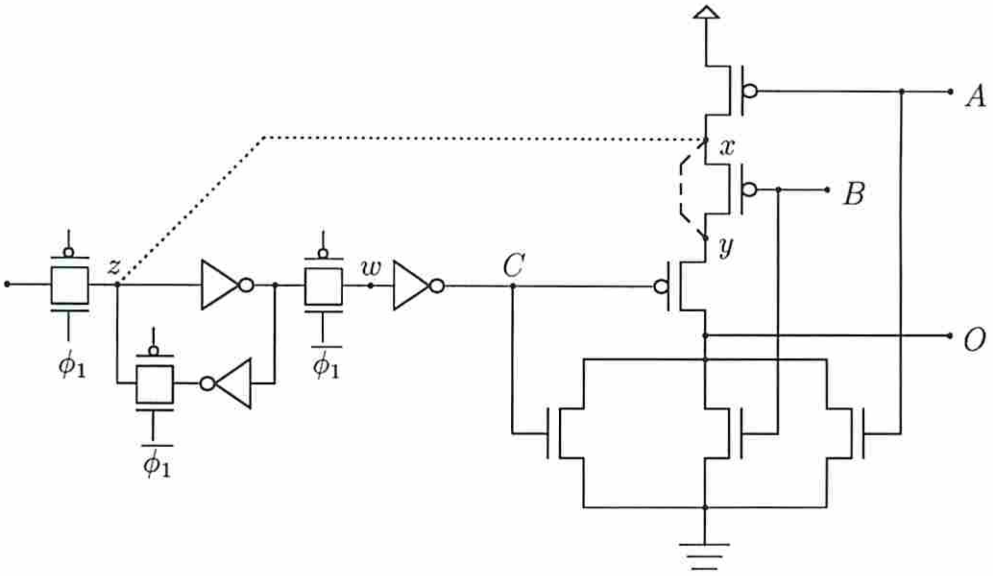


Figure 5.3: A BF that is masked by another BF in a sequential circuit

Example 4: This example shows that 1) the detection of a BF in a combinational circuit may rely on charge retention and hence require a two-vector test, and 2) even in a combinational circuit the detection of a single BF may be invalidated by multiple BFs. Consider the circuit shown in Figure 5.4. To detect the BF (x, y) both z and A must be set to 0 ($\overline{A}$ must be 1). However when A is set to 0, z cannot be connected to *GND*. Thus the detection of (x, y) must rely on the charge retention on z. Two vectors $ABC = 111, 010$ may be used to detect (x, y) where $ABC = 111$ sets z to 0 and $ABC = 010$ retains z at 0 and sets up a conducting path from y to *GND*. Now assume another BF (w, z) exists in the circuit. z is still set to 0 when $ABC = 111$. However when the second vector $ABC = 010$ is applied, z is charged to 1 due to (w, z) and thus BF (x, y) cannot be detected. □

From the above examples it follows that detecting BFs using CSM needs careful consideration. Next a set of design and test rules under which CSM can be "safely" used will be presented.

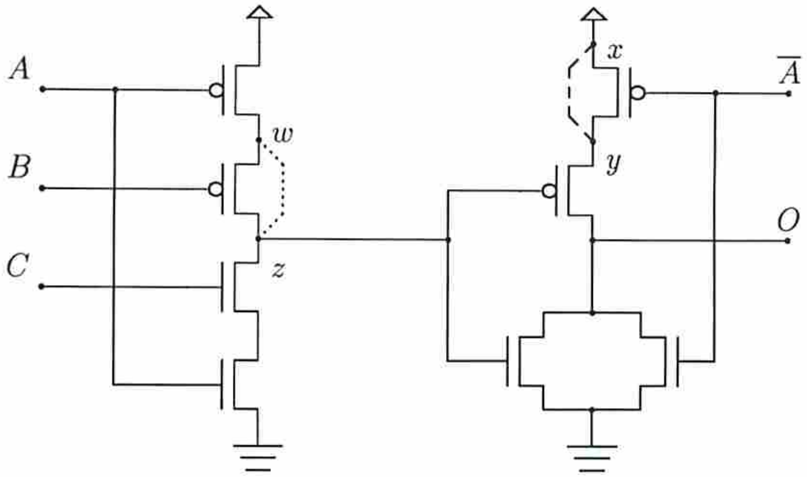


Figure 5.4: A BF that is masked by another BF in a combinational circuit

5.1.3 Design and Test Rules for IDDQ Testing

In this subsection a set of design and test rules to ensure the proper use of CSM are presented. The circuit model for these rules was described in Section 4.1.1.

A1 The gate and drain (or source) node of a transistor cannot be in the same transistor group.

A2 During steady state operation, there must be no conducting path from *VDD* to *GND*.

A3 During steady state operation, each output of a transistor group must be connected to *VDD* or *GND* through a path of conducting transistors.

A4 There are no control loops among *TGs*.

A5 The bulk (or well) of an n-type (p-type) transistor is connected to *GND* *(VDD)*.

A6 During testing, each primary input is controlled by a strong power source whose current is also monitored.

The rationale of these rules is as follows. A1 is made to exclude the possibility of "self-control" inside a transistor group. A2 is a common attribute of CMOS circuits. A3 ensures that a circuit's normal operation does not rely on any "charge sharing" or "charge retention" effects. This rule excludes the possibility of identifying a good circuit as faulty using CSM. A4 assumes no feedback exists in the circuit. A5 ensures that a BF will not cause *anomalous reverse conduction* that may occur at the drain-bulk junction when the bulk is connected to the source [72]. This effect will be explained in Section 5.1.4. A6 makes sure that if a primary input is involved in a BF, it cannot change state without consuming a large steady state current, and this abnormal current can be detected. The reasons for these rules will become clearer later.

Among these rules, A1-A5 are design rules for CMOS circuits and A6 is a rule that should be followed during testing. For brevity in later discussions, rather than using the phrase "a circuit satisfies A1—A5 and the test environment A6 exists for testing this circuit," it is simply said that "a circuit satisfies A1—A6." Some typical circuits that satisfy A1—A6 are (1) all fully complementary primary gates (NAND, NOR and inverters), (2) all fully complementary complex gates, (3) all fully complementary pass transistor networks (e.g., multiplexers), (4) any acyclic combination of 1, 2 and 3, and (5) the combinational part (CP) of any sequential circuit if the CP satisfies 1, 2, 3 or 4, and all inputs to the CP are controlled by strong power sources that cannot change values (due to faults) without consuming an excessive current.

5.1.4 Minimality of the Set of Rules

In this subsection circuit examples are used to show that the set of rules A1—A6 is *minimal* in the sense that if any rule is removed then circuits exist for which CSM cannot give correct results.

A1: Consider the circuit shown in Figure 5.5 that contains two transistor groups TG_1 and TG_2. TG_2 violates A1 because the gate and drain nodes of the P-type and N-type transistors of the inverters driving y and z are in the same group. A2 is satisfied because no *VDD-GND* connection exists during steady state operation (the two pass transistors connected to O are controlled by complementary values). A4 is also satisfied because the loop containing y and z is an "internal loop" rather than a control loop. It is easy to verify that all other rules are/can be satisfied.

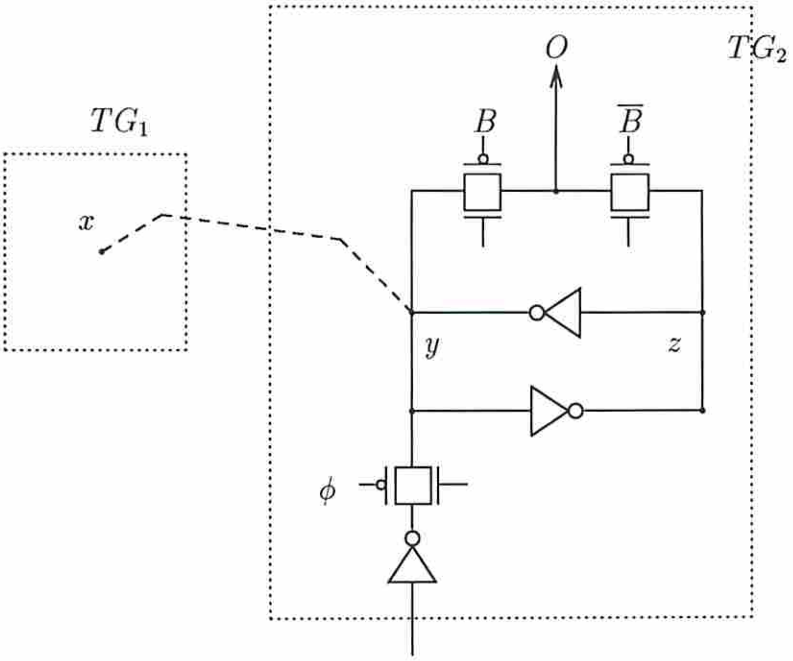


Figure 5.5: A circuit that violates only A1

Assume TG_1 satisfies A1—A6 and there exists a BF between a node x in TG_1 and a node y in TG_2. To detect this BF, x and y must be set to complementary logic values. If x and y become stable only when $\phi = 1$, then due to the internal loop that contains y, during steady state y will reach the same logic value as x and no excess current can be observed. Therefore CSM will fail to detect this fault.

A2: This rule is essential since otherwise an excess current will exist in a fault-free circuit.

A3 and A4: The circuits shown in Figure 5.1 and Figure 5.2 only violate A3 and A4, respectively.

A5: Consider the circuit shown in Figure 5.6(a) which contains a BF (x, y). In a fault-free circuit if $x = 0$ then y must be 0 and if $x = 1$ then either $y = 1$ or y is floating. Therefore no test can set x and y to complementary logic values and BF (x, y) is considered as redundant.

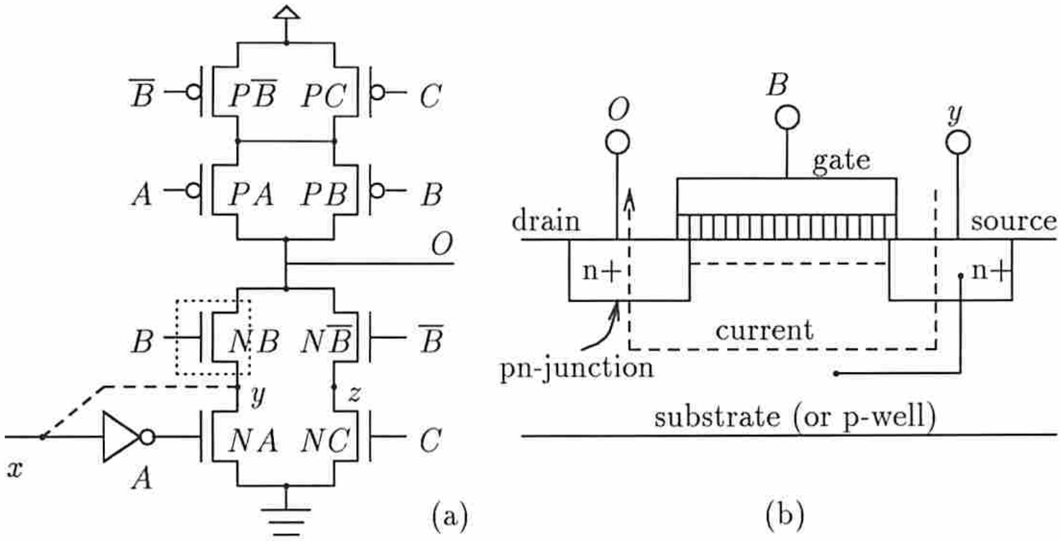


Figure 5.6: Anomalous reverse conduction

Now assume the substrate of transistor NB is connected to its source node rather than GND as shown in Figure 5.6(b). When a vector $xBC = 101$ is applied, node y and the substrate of NB will be set to 1 due to (x, y). Thus the pn-junction at the drain node (O) will be forward biased because O is connected to GND through transistors $N\overline{B}$ and NC and an anomalous reverse conducting path [72] forms from node x, through node y, the substrate of NB, node O, node z, to *GND*. This results in an excess current. However classical test generation procedures will not generate a test for this fault since it is considered as redundant.

A6: If this rule is not satisfied, then a bridging fault that involves a primary input may not be detected. Consider the circuit shown in Figure 5.7 that contains a BF (x, y) where y is a primary input driven by an external device. If $x = 1$ and $y = 0$ are set then there exists a large current flowing from *VDD* through x, y to the driver of y. Since the current on the driver may not be monitored, the fault may not be detected.

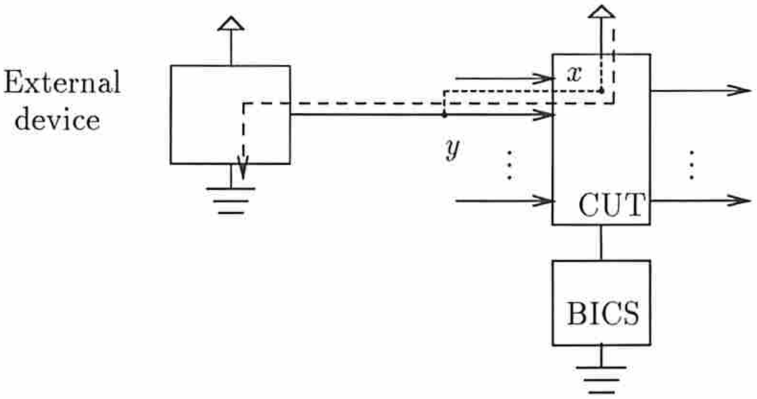


Figure 5.7: Current flowing through PI may not be detected

5.1.5 Sufficiency of the Rules

In this subsection it is formally shown that if all the proposed rules are satisfied then each irredundant single or multiple BF can be detected by a single test vector. Though intuitively the theorems hold, carefulness must be taken to make sure that each step of the proofs is correct. Also it is useful to see how rules A1—A6 play a role in these proofs. The following basic approach is used in the proofs: *when an appropriate test vector is applied, if due to a fault there exists one or more paths from VDD to GND such that at any time at least one of these* *paths is conducting, then the fault is detected using CSM.* This approach ensures that CSM can detect a BF only if there exists conducting path(s) between *VDD* and *GND all the time*, not just *temporarily.*

Without loss of generality the following discussion assumes that a test vector for BF (x, y) will set x and y to 1 and 0, respectively, in a fault-free circuit.

Theorem 9 *A single BF in a circuit satisfying A1—A6 is either detected using CSM or is redundant.*

Proof: Three different cases of single BFs exist in circuits satisfying A1—A6, namely (1) BFs inside a TG, (2) BFs between two unrelated TGs, and (3) BFs between two related TGs. For each case it will be proved that (a) if an input vector T can be found that simultaneously sets up a conducting path P_1 from *VDD* to x and another conducting path P_2 from *GND* to y in a fault-free circuit, then the fault (x, y) can be detected by using CSM, and (b) if no such input vector exists, then (x, y) is redundant.

First assume such a T can be found for each case.

Case 1: Refer to Figure 5.8(a). When T is applied and (x, y) is present, the voltages of circuit nodes on P_1 and/or P_2 will be different from those in the fault-free circuit since x and y will have the same voltage level. If these new voltage levels result in the opening of either P_1 or P_2, then the short between *VDD* and *GND* may not be observed. However, this is impossible since all the transistors on both P_1 and P_2 are controlled by signals from other *TGs* (assumption A1), none of which can be controlled by nodes on P_1 or P_2 (assumption A4). Thus P_1 and P_2 are always conducting and (x, y) can be detected by CSM.

Case 2: Refer to Figure 5.8(b). Similar to Case 1, where paths P_1 and P_2 cannot be broken.

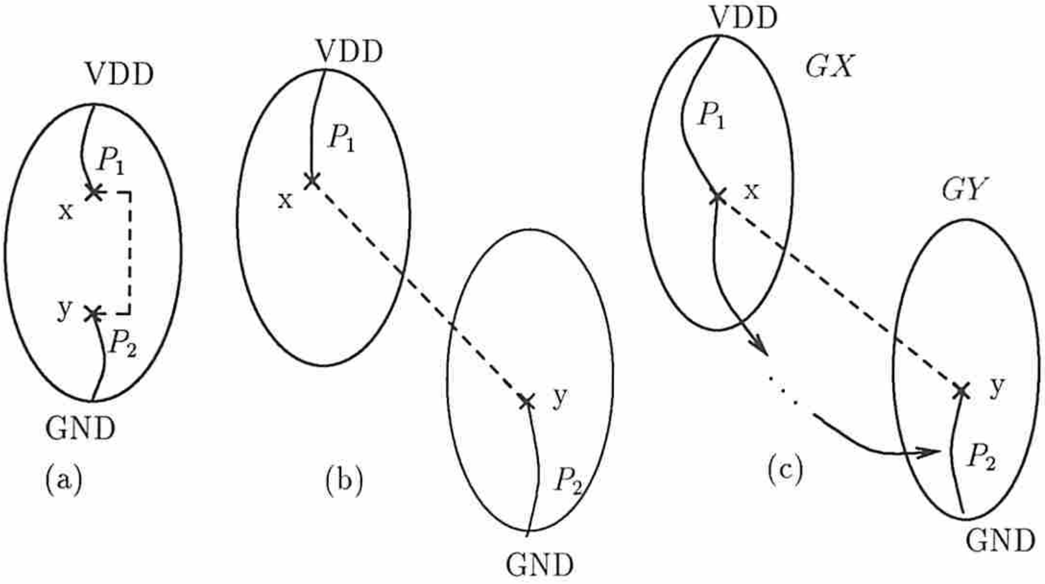


Figure 5.8: (a) BF inside a TG (b) BF in two unrelated TGs (c) BF between two related TGs

Case 3: Let GX and GY be the two TGs containing x and y, respectively. Since there is no loop containing GX and GYin the fault-free circuit (assumption A4), only one of them can control the other. Without loss of generality, assume GX can control GY as shown in Figure 5.8(c). When T is applied, if in the faulty circuits the new voltages along P_1 and P_2 does not affect any output of GX, or they only affect those outputs of GX that do not affect P_2, then both P_1 and P_2 are always conducting and the fault can be detected. Now assume the new voltages affect some outputs of GX and they in turn affect some transistors on P_2. Let $\hat{O}$ be a set consisting of such output nodes. Each element in $\hat{O}$ must be connected to x through some conducting path. Now if no large current through P_1 ever occurs, P_2 must have been broken since P_1 cannot be broken. But in this case node x and all nodes in $\hat{O}$ will charge to their fault-free values. This will cause path P_2 to once again become conducting. The net effect is a circuit oscillation. But an oscillation means that at any instant of time at least one TG is consuming a large current and the fault can be detected. Note that in the above proofs, if x or y is a primary input then P_1 or P_2 may exist in the external current measurement device whose current is also monitored (assumption A6).

If no input vector T exists that can set x and y to complementary values in the fault-free circuit, then for any input vector, x and y are either at the same state (0, 1, or *floating*) or one of them is *floating*. For the former case, the bridging fault is obviously redundant. For the latter, the floating node will be set to the same value as the other node. Since there is no *anomalous reverse conduction* effect in the circuit (assumption A5) and a floating node cannot affect the outputs of a transistor group (assumption A3), this fault cannot affect the logic function of a circuit and thus is redundant. □

Theorem 10 *If T is a test vector for a single BF f_1, then T is also a test vector for every multiple BF that contains f_1.*

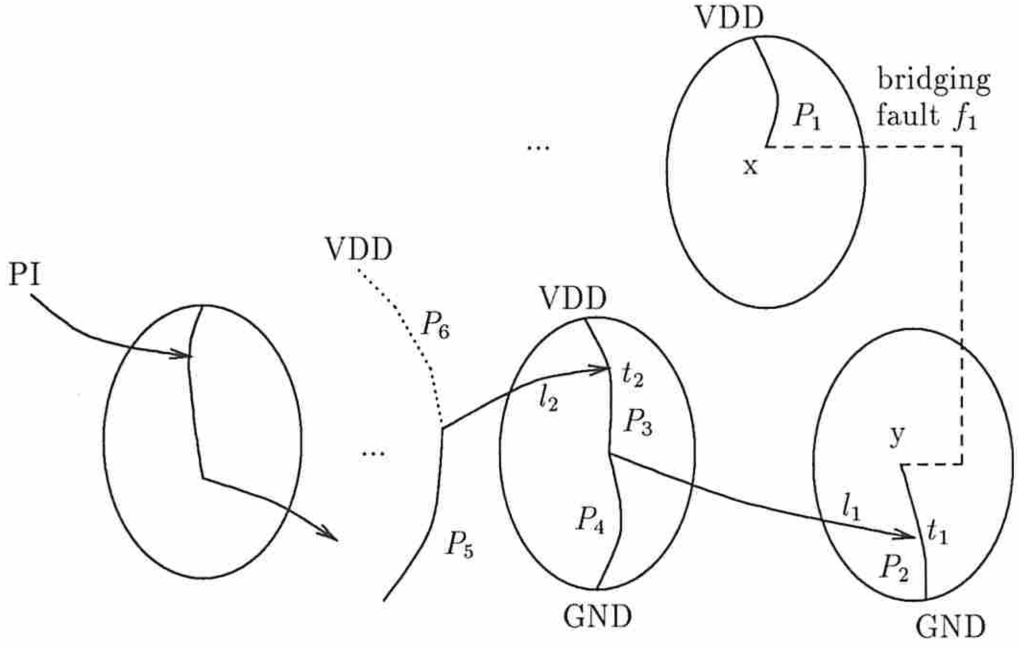


Figure 5.9: Detecting multiple bridging faults

Proof: Refer to Figure 5.9 where the bridging fault f_1 occurs between x and y. Again assume T sets up conducting paths P_1 from *VDD* to x and P_2 from *GND* to y. Similar to Theorem 1, if any oscillation occurs in the circuit, then CSM will detect the fault. Thus it may be assumed that no oscillation occurs. If during steady state no large current flows through P_1 or P_2, then at least one transistor on P_1 or P_2 must be switched off. Without loss of generality assume transistor t_1 on P_2 is off. Let the input line to t_1 be l_1. l_1 must be an output of another transistor group (assumption A1) and T must set up another conducting path, say P_3, from a power source to l_1 (assumption A3) in the fault-free circuit. Now the value of l_1 has been changed due to faults in the circuit. If P_3 is not switched off, then another path, say P_4, from l_1 to an opposite power source must have been formed, and the resulting voltage value of l_1 is under the threshold value of t_1. But if this is the case, a conducting path containing P_3 and P_4 from *VDD* to *GND* must be formed and the faults can be detected. If P_3 is switched off, the value of one input, say l_2, to a transistor on P_3 must be the complement of its fault-free value. By a similar argument, either a conducting path, say P_5, set up by T from a power source to l_2 in the fault-free circuit is not cut but another conducting path, say P_6, from l_2 to an opposite power source is formed, or P_5 is cut. In the former case, the fault is detected. In the latter case, the above process may be continued. Since there is no control loop in the fault-free circuit (assumption A4), eventually either a conducting path from *VDD* to *GND* is found or a primary input is reached. Since a primary input is controlled by a strong power source whose current is monitored (assumption A6) the fault can be detected. □

Theorems 9 and 10 imply that all multiple BFs that contain at least one irredundant single BF are detectable by using CSM, and a test set that detects all single irredundant BFs is sufficient for detecting all such multiple BFs.

5.1.6 Circuits not Satisfying A1—A6

It has been shown that A1—A6 is a minimal set of rules required to ensure the detection of all irredundant BFs using CSM and there exist some circuits that do not satisfy all of these rules. In this subsection several problems that may arise when each of these rules is released are examined and, if possible, some strategies to eliminate these problems are proposed.

The circuits to be discussed include (1) an Exclusive OR gate shown in Figure 5.10, (2) BiCMOS circuits, (3) domino logic, (4) synchronous sequential circuits and (5) circuits implemented by *Silicon on Insulator (SOI)* technology, which are typical circuits that do not satisfy A1, ..., A5, respectively. A6 is a rule for the test environment. However when dealing with sequential circuits, A6 can also be considered as a design rule for the output gates of the storage elements.

5.1.6.1 Release of A1 (Drain (or Source) and Gate Are Not in the Same TG)

Without this rule the logic value of a node in an "internal" loop of a faulty TG (i.e., a TG involved in a BF) may stabilize at an incorrect logic value without consuming an excess steady state current. If it can be guaranteed that this situation will not occur, then A1 can be released. For example in the Exclusive-OR gate shown in Figure 5.10, though the drains (or sources) and gates of transistors T_1, T_2, T_3 and T_4 are in the same TG, CSM can still be applied. Next a general method to identify this type of circuits is presented. First the following definitions are needed.

Input-only node: A node v is an *input-only node* to a transistor group TG_1 if and only if 1) v is in TG_1, and v is a primary input node but not an output node

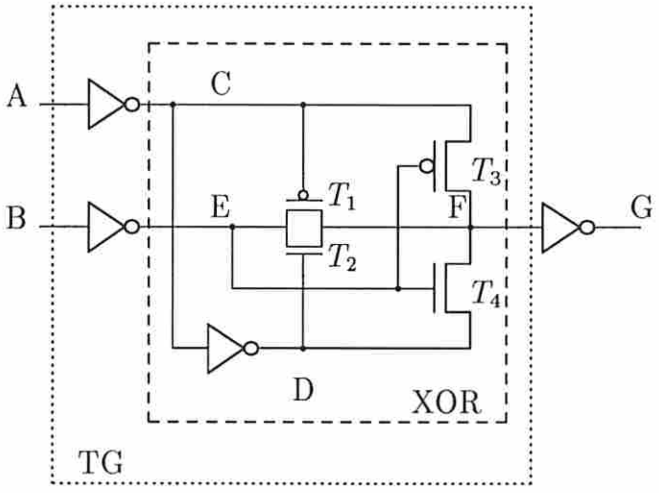


Figure 5.10: A circuit that violates A1 but CSM can still apply

of TG_1, or 2) v is not in TG_1, but v is an output node of another TG and v feeds the gate of some transistor in TG_1.

Strong-node (S-node): A node is an *S-node* of a TG if and only if 1) it is either *VDD*, *GND* or an input-only node to this TG, or 2) it is always connected to some S-node of this TG through some conducting transistor controlled by an S-node of this TG.

Changeable node: A node v is called *changeable* if and only if there exists an input vector and some bridging fault(s) such that the logic values of v in a faulty and fault-free circuit are different but there exists no excess steady state current in the faulty circuit.

For the circuit shown in Figure 5.10 A and B are both input-only nodes to this TG and thus are S-nodes. C and E are also S-nodes because they are connected to either *VDD* or *GND* through conducting transistors controlled by A and B, respectively. D is an S-node because C is, and F is an S-node because C, D and E are S-nodes. Thus all nodes in Figure 5.10 are S-nodes. For the circuit in Figure 5.5, y and z are controlled by each other. According to the "if and only if" property of the definition of S-nodes, neither of them can be an S-node.

Lemma 10 *For a circuit G that satisfies A1-A6, the output nodes of each TG in G are not changeable.*

Proof: Since A4 is satisfied, no control loop exists in G. Thus the *levels* of all transistor groups can be assigned such that if a TG, say TG_1, can control another TG, say TG_2, then the level of TG_1 is less than that of TG_2. The output nodes of TGs at the first level are not changeable because each output node must be connected to a power source through a number (≥ 0) of conducting transistors controlled by primary inputs. By induction it is easy to prove, level by level, that the output nodes of each TG in G are not changeable. □

Lemma 11 *Let G be a circuit that satisfies rules A2-A6. An S-node of any TG in G is not changeable if all input nodes to this TG are not changeable.*

Proof: Let C be an S-node of a transistor group TG_0. The lemma obviously holds if C is *VDD*, *GND* or an input node to TG_0. Thus it may be assumed that C is not *VDD*, *GND* or an input-only node to TG_0. Since C is an S-node of TG_0, for any input vector there always exists some S-node A of TG_0 and some conducting transistor T_B that is controlled by an S-node B of TG_0. This is shown in Figure 5.11. By induction it may be assumed that A and B are not changeable. Therefore transistor T_B must be conducting if no excess current exist. Thus C cannot be disconnected from A due to a fault and hence C is not changeable. □

By using the next rule the following result is obtained.

R1: All nodes in a transistor group are S-nodes of this TG.

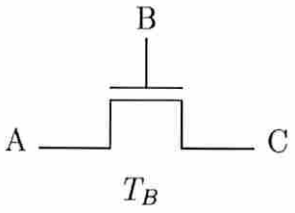


Figure 5.11: Circuit used in proof of Lemma 2

Theorem 11 *If a circuit G satisfies A2-A6, and either A1 or R1 is satisfied for each TG in G, then all irredundant BFs in G can be detected by CSM.*

Proof: The theorem follows from Lemmas 10 and 11 based on the results of Theorems 9 and 10. □

It is interesting to note that there exist circuits that satisfy A1 but not R1, and vice versa. Figure 5.10 indicates a circuit that satisfies R1 but not A1. A circuit that satisfies A1 but not R1 is shown in Figure 5.12 where node x is not an S-node because when input $AB = 11$, x is not connected to any S-node.

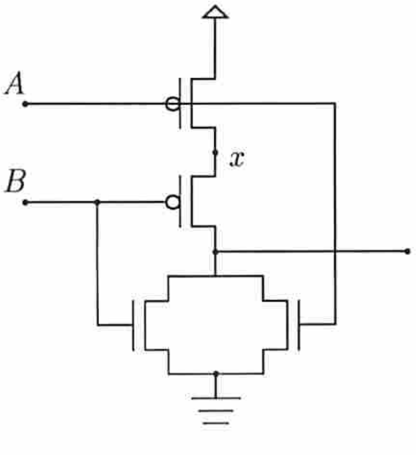


Figure 5.12: A circuit that satisfies A1 but not R1

5.1.6.2 Release of A2 (no Conducting Path from *VDD* to *GND*)

In general this rule must be followed because CSM cannot be applied to circuits that consume large steady state current such as BiCMOS circuits. However one can partition a circuit into two parts BX and BY; BX containing all subcircuits satisfying A2 and BY containing those that do not, as illustrate in Figure 5.13. By using BICSs only for BX, BFs *inside* BX, i.e., BFs involving nodes in BX but not the inputs to BX, can still be detected. If the inputs to BX are monitored then BFs involving them can also be detected.

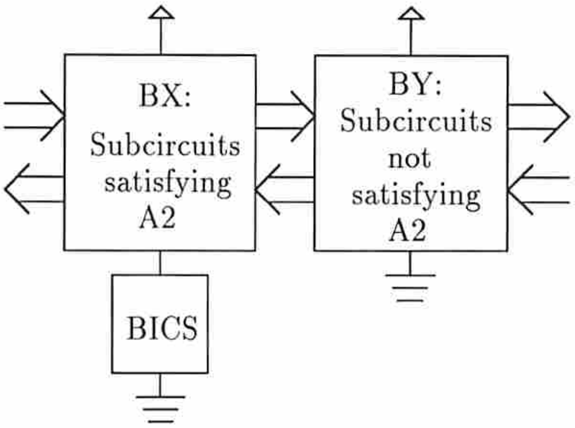


Figure 5.13: Partitioning of a circuit: one satisfies A2 and the other not

For example a BiCMOS circuit can be partitioned into CMOS circuitry and bipolar circuitry. BICSs can be used to detect BFs inside the CMOS circuitry. Since most bipolar circuitry in a BiCMOS circuit are used for interface, e.g., I/O drivers, one can still test a large portion of the circuit using CSM. A similar concept can be applied to circuits implemented using "mixed" technology such as CMOS/pseudo-NMOS or CMOS/pseudo-PMOS.

5.1.6.3 Release of A3 (no Floating Output Nodes)

All precharge logic circuits may contain floating output nodes during steady state as explained next. The operation of a precharge logic circuit can be divided into two phases: precharge and evaluation phases. During the precharge phase, a precharge node is charged to some logic value (usually 1), and during the evaluation phase, depending on the inputs, either the precharge node retains its precharged value or it is driven by an opposite power source (usually *GND*). For the former case, the precharge node is neither connected to *VDD* nor to *GND*. Thus A3 is not satisfied.

Consider the domino logic gate that consists of a precharge stage and an inverter shown in Figure 5.14(a). For any BF (x, y) inside one evaluation block, during $\phi = 0(1)$ neither x nor y can be connected to *GND (VDD)*. Thus x and y cannot be connected to complementary power sources at the same time and hence (x, y) cannot be detected using CSM. This is also true if x and y are in two distinct evaluation blocks. The only BFs that can be detected by CSM are those involve *VDD*, *GND* or the output nodes of the inverters.

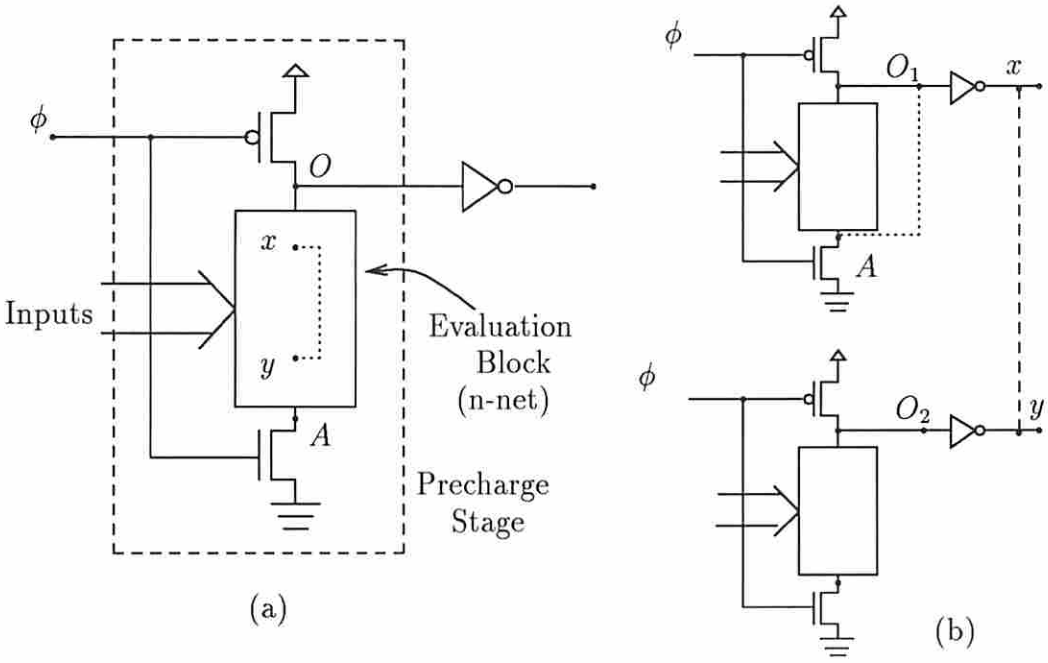


Figure 5.14: A domino logic gate

Another problem is due to the charge sharing effect at the precharge nodes. In Figure 5.14(a) if during $\phi = 1$ there is no conducting path from O to A, then O is a floating node and should retain its logic value 1. However due to charge sharing between O and some nodes inside the evaluation block, the voltage value

v at O may be less than $VDD(5V)$. If v is larger than $4V$, no problem exists. If v is less than $4V$ but larger than $VDD/2$, due to the high noise immunity of CMOS circuits it is likely that the logic value at the output of the inverter is still at 0. Therefore logically no faults exist. Because of this when a designer designs domino logic, he may allow the resulting voltages at O after charge sharing to be at, say $3.5V$. This voltage will turn on both N- and P-type transistors of the inverter resulting in a large current.

Yet another problem is that the effect of one fault may be masked by the existence of another fault. For example in Figure 5.14(b), to detect the fault (x, y), O_1 and O_2 must be set to complementary values. Assume O_1 and O_2 are set to 1 and 0, respectively. Due to another BF (O_1, A), the value at O_1 will be 0 during $\phi = 1$ and thus no excess current exists.

In summary the release of A3 may result in the following problems: (1) many undetectable BFs may exist if only CSM is used; (2) a large current may flow through a gate driven by a floating node (or the precharge node); and (3) the detection of a single fault may be masked by the existence of other faults. These problems are not easy to eliminate. The undetectability of BFs in precharged logic is an intrinsic problem due to the "precharge" property of these circuits, and appears to invalidate the use of CSM. The charge sharing problem may be alleviated by using a larger precharge node (in terms of capacitance). This, however, can result in performance degradation of the circuit. It seems that the fault masking problem always exists and cannot be avoided.

From the above discussions it can be concluded that CSM is not an effective method for detecting BFs in precharge logic. However if a circuit can be partitioned into non-precharge and precharge portions, then CSM can still be used for the non-precharge portions.

5.1.6.4 Release of A4 (no Control Loops)

Violating this rule may result in a feedback loop in which nodes may stabilize at incorrect logic values without consuming excess current. However, to have this erroneous situation two conditions must be satisfied. First, the feedback loop must be a "real" loop, i.e., it must be a *logically active* loop, not just a *physical* loop. Consider the circuit shown in Figure 5.15 that contains two nodes A and B. When $\phi = 1(0)$, the data at node $A(B)$ can be propagated to node $B(A)$. There exists a physical loop in this circuit. However, since the two pass transistors T_1 and T_2 cannot be turned on at the same time, no logically active loop exists.

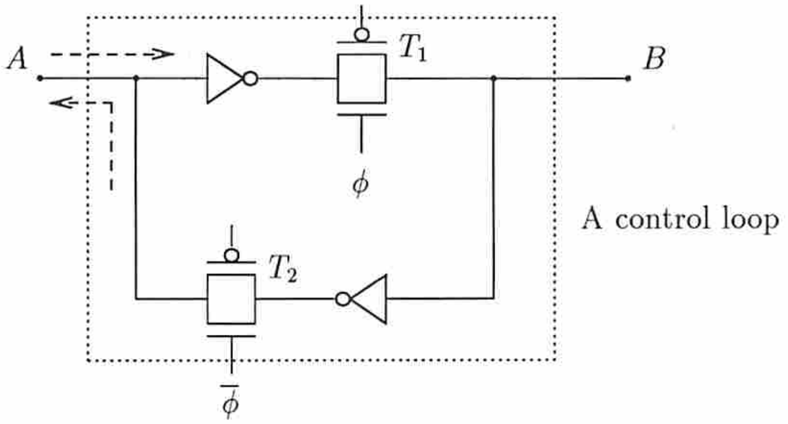


Figure 5.15: A physical loop that is not a logical loop

The second condition is that there exists no external control to the loop so that the logic values in the loop may change without consuming an excess steady state current. Consider the Huffman model of a sequential circuit shown in Figure 5.16(a). In the following discussions master-slave flip-flops are used as storage elements. Similar arguments can be applied to circuits using other types of storage elements. Figure 5.16(b) shows that both the master and slave latches contain a control loop (indicated by dotted boxes). The control clock is shown in Figure 5.16(c), which also shows when to measure the current. During $CLK = 0$ the slave level has an external control but not the master level as explained next.

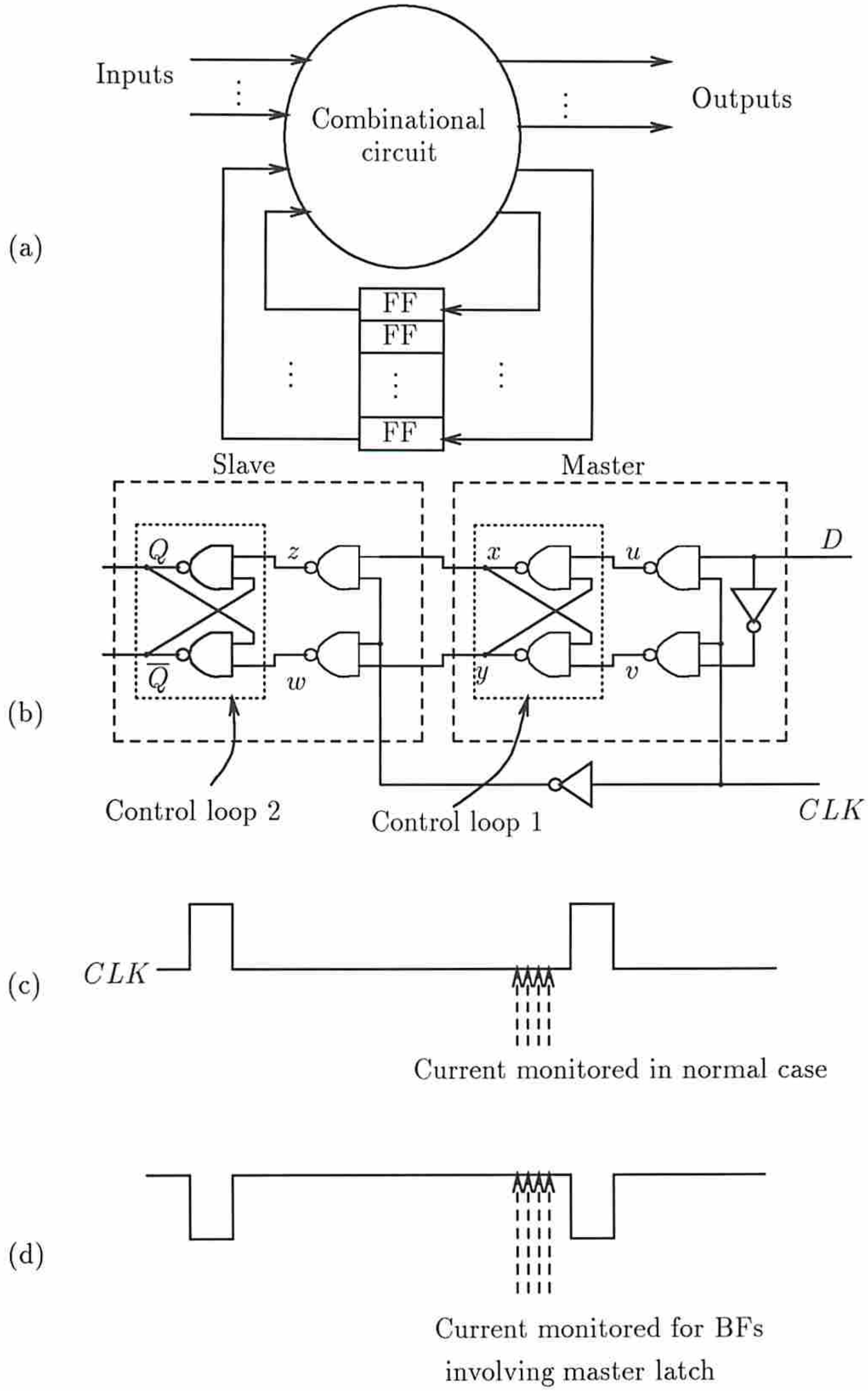


Figure 5.16: A sequential circuit with master-slave flip-flops

When $CLK = 0$, $u = v = 1$. Thus control loop 1 is not controlled by u or v, i.e., the value of $x(y)$ may stabilize at either 1(0) or 0(1). Any BF to either x or y but not both can change the state of control loop 1 without consuming an excess steady state current. For control loop 2 the situation is different. When $CLK = 0$ one of z and w must have a logic value 0 and thus the control loop 2 is under the control of z or w. The state of this loop cannot be changed without consuming an excess current.

Thus the problem is on the BFs involving x or y but not both. Two strategies can be used to deal with this problem. First, the possibility of having such BFs can be reduced by a careful layout. This can be done because both x and y are nodes within a FF. Another approach is using the clock as shown in Figure 5.16(d). One can then measure the current near the end of $CLK = 1$.

It is important to note that the "global loops" (FFs—combinational part—FFs) do not logically exist during steady state. Thus BFs involving either x or y but not both are the only faults that may not be detected by CSM. This observation can be generalized to all synchronous sequential circuits since during steady state all global loops should not logically exist otherwise races in the circuit will occur.

5.1.6.5 Release of A5 (Substrate Connected to VDD or GND)

A5 requires that the substrate or well of a N-(P-)type transistor be connected to $VDD(GND)$ so that no anomalous reverse conductance (ARC) effect can occur. In general this is a feasible requirement. However if some well must be connected to the source node of a transistor, e.g., to reduce body effects, then to use CSM a more complex test generation strategy must be adopted. This test generation procedure must be able to construct a test vector that can detect a fault that creates an ARC effect. For example for the circuit in Figure 5.6, the test generator may produce $xBC = 101$ as a test vector for the BF (x, y).

The above discussion focuses on circuits implemented using bulk-silicon technology in which the substrate or well can be connected to either *VDD* (or *GND*) or a source node. Figure 5.17 show a different technology, namely *Silicon on Insulator (SOI)* [108]. SOI uses a layer of insulating material (e.g., oxide) on top of a substrate (such as Si). The channel of a MOSFET is isolated from the substrate, as shown in Figure 5.17, where a circuit node containing a channel is called a *body node.* A body node can be either connected to a source node, left to float, or connected to a power source. Therefore rule A5 may be violated.

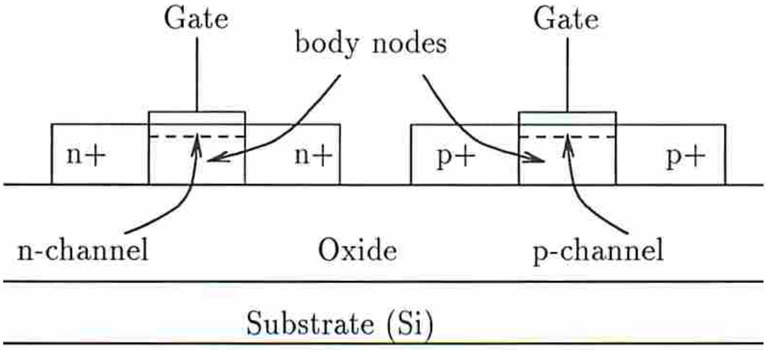


Figure 5.17: Circuit implemented by Silicon on Insulator technologies

If all body nodes are floating or connected to a power source, then there is no ARC effect and CSM is applicable. In some applications it may be desired to connect the body node to a source node in order to eliminate the *kink* effect [109, 110]. The ARC problem then needs to be addressed by using a more advanced test generation algorithm.

5.1.6.6 Release of A6 (PI Current Monitored)

As shown in Figure 5.7, without this rule some BFs involving primary inputs of a combinational circuit may not be detected. However a more careful examination of Figure 5.7 reveals that if x and y can be set to 0 and 1, respectively, then the fault can be detected because there will be an excess of current through the BICS on the *GND* line of the circuit, as shown in Figure 5.18(a). This observation can be generalized as the following: if BICSs are used to monitor the current through the *GND(VDD)* line, then even without using external current monitors for PIs, one can still detect a BF involving a PI and an internal node by setting the PI to 1(0) and the other node to 0(1). This approach, however, may not be feasible for all BFs involving PIs because there may exist some faults whose detections require some PI to be set to 0(1).

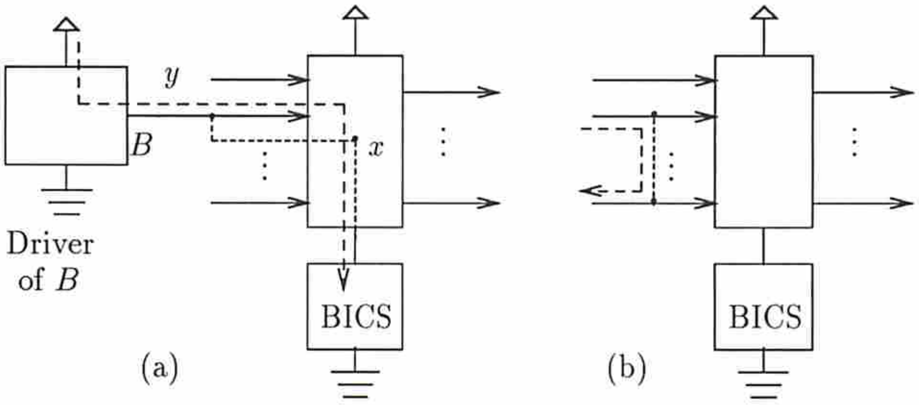


Figure 5.18: Detecting BFs involving PIs: (a) a fault detected by CSM and (b) a fault not detected by CSM

Another problem is that the above method cannot be used to detect a BF that involving two PIs. Refer to Figure 5.18(b). Since both shorted nodes are connected to external power sources, the fault cannot be detected if the current through PIs is not monitored. Fortunately since the problems due to the release of A6 are caused by BFs involving PIs, one feasible strategy to alleviate these problems is to use a careful layout to reduce the possibility of having BFs on PIs.

Next consider the Huffman model of a sequential circuit described in Figure 5.16(a). As mentioned in Section 5.1.3, CSM can be used to detect BFs in the combinational part of a sequential circuit if the inputs to the combinational part are controlled by strong power sources whose currents are monitored. The inputs to the combinational parts can be divided into two parts: primary inputs and the outputs of the storage elements. We shall only discuss the latter here since the discussion for PIs is the same as that for a combinational circuit. From the discussion in Section 5.1.6.4, we know the outputs of the storage elements are controlled by strong power sources. Now the problem is how to monitor the current of these power sources.

A built-in current sensor can be used to monitor the *VDD* or *GND* line of the combinational part of a sequential circuit. To monitor the current of an input driven by a gate G_1 of a storage element, as shown in Figure 5.19(a), either the *VDD* or the *GND* line of G_1 must be monitored. The site (*VDD* or *GND*) to be monitored must be the same as that for the combinational part. Figure 5.19(b) shows why this is necessary.

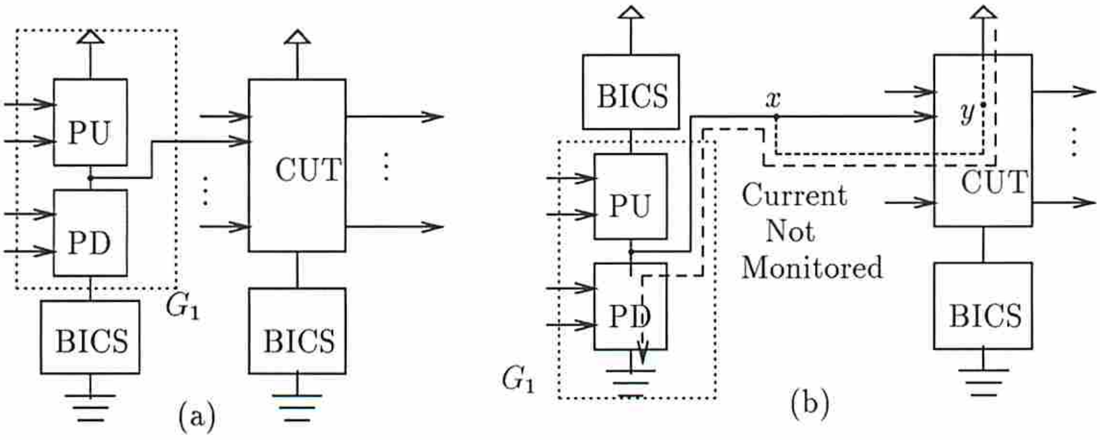


Figure 5.19: Monitoring the current at the inputs

The various strategies and/or suggestions presented in this section are summarized in Table 5.1.

5.1.7 Summary

One major advantage of using CSM to detect CMOS BFs is that once a fault is activated, the fault can be detected without observing the logic values at any primary output. Thus the test generation process is simplified as no fault effect propagation is required. The fault simulation process is also quite simple since

Release of rules	Problems	Strategies/suggestions
A1: Source and gate in the same TG	"Internal loop" in a TG	• Identify S-nodes • Use R1 to replace A1
A2: Conducting path from VDD to GND	Excess current in good circuits	• Partition the circuit
A3: Floating nodes	• Undetectable faults • Charge sharing • Fault masking	• Do not release this rule • Partition the circuit
A4: Control loops	Erroneous steady state logic values	• Identify active loops • Identify external control • Careful layout
A5: Substrate or body node not connect to VDD or GND	Anomalous reverse conduction	• Careful test generator • Leave body nodes floating or connect them to power sources
A6: PI not monitored	BFs involving PIs not detected	• Careful test generation • Careful layout

Table 5.1: Summary of strategies for circuits not satisfying each rule

a bridging fault between any two nodes that are respectively connected to *VDD* and *GND* can be easily identified as detected. These advantages, however, can be achieved only when CSM is carefully analyzed.

In this section the applicability of CSM has been extensively analyzed. In particular a set of design and test rules that guarantee the detection of all possible irredundant BFs are presented. For those circuits that do not satisfy these rules, problems that can arise are discussed and various strategies to deal with these problems are proposed. The results can serve as a general guideline for using CSM: to determine whether CSM can be applied to a circuit one first checks whether the circuit satisfies rules A1—A6. If it does, then CSM can be used safely. Otherwise depending on which rule is not satisfied, various strategies can be applied.

5.2 Stuck-open Faults

5.2.1 Introduction

This section deals with generating tests for stuck-open faults. It is assumed that two-pattern tests are used. The invalidation problem due to circuit delay and charge sharing are both considered. A necessary and sufficient condition for a test to be valid under any circuit delay is given. This condition enables the development of an efficient test generation algorithm. The charge sharing problem is analyzed at the circuit level, i.e., the capacitances of various circuit nodes are estimated such that the effect of charge sharing can be characterized. It is shown that the charge sharing problem can be easily solved by employing IDDQ testing. Finally a procedure for generating **robust** tests, i.e., tests that cannot be invalidated by any circuit delay or charge sharing, is given.

A two-pattern test consists of two input vectors. The first vector, T_1, is the initialization vector and the second vector, T_2, is the test vector. A transient vector, T_d, which is used to describe the transient input status, will be defined later. The circuit model described in Section 4.1 is used and the rules presented in Section 5.1.3 are followed. For simplicity in this section it is assumed that each transistor group has only one output. The techniques described can be easily modified and applied to circuits containing multiple output transistor groups.

5.2.2 Invalidation due to Circuit Delay

5.2.2.1 Strategy

To detect a stuck-open fault, the initialization vector T_1 must set the output of the faulty transistor group to a certain value and the test vector T_2 propagates the fault effect to a primary output. Since in general T_1 is much easier to obtain than T_2, in most of the previous work T_2 is generated first, followed by a T_1 that guarantees no invalidation occurs [28, 48]. This method requires a validation checking procedure during each step of the generation of T_1, and thus can be very inefficient.

In [40] a new strategy is taken. Instead of generating T_2 before T_1, an intermediate vector called T_{INT} is first generated. T_1 and T_2 are then generated by adding more input assignments to T_{INT}. T_{INT} has an important characteristic, namely the invalidation condition due to circuit delay has been explicitly considered, thus as long as T_{INT} is obtained, no invalidation checking is necessary when further generating T_1 and T_2. This method apparently would lead to better performance of test generation.

Unfortunately the work in [40] requires transforming a switch level circuit to a gate level equivalent circuit, and thus has the problems described in Section 2.2.1. Also to derive T_{INT} in the gate level equivalent circuit, a sensitization path from the fault site to the line corresponding to the output node of the faulty transistor group must exist such that all the lines on the path are free of static hazard under faulty condition [40]. This makes the problem even more complex. Another problem of [40] is that no actual implementation is reported.

In this study a similar concept to [40], i.e., deriving the intermediate vector first, and then generating T_1 and T_2, will be employed. The difference between this work and the one in [40] is that a true switch level model will be used, i.e., no gate level equivalent circuit is necessary. To guarantee that no invalidation problem exists after the intermediate vector is generated, a different approach is necessary for switch level test generation. The approach to be presented requires the development of an algorithm to find all possible cutsets for an undirected graph. Next a method for enumerating all possible cutsets is described.

5.2.2.2 Enumeration of All Cutsets

Given a connected and undirected graph G and two distinct vertices s and t in G, a *cutset* that separates s and t is defined as a set of edges C that has the following properties: 1) the removal of edges in C disconnects s and t, and 2) for any edge e in C, the removal of edges $C - \{e\}$ cannot disconnect s from t. A cutset separating s and t is sometimes called an *st-cutset* [111].

The problem of enumerating all st-cutsets is one of the most fundamental problems in graph theory [112]. In [111] two algorithms (Algorithms 2 and 3 in the paper) to enumerate all the cutsets in $O(|V|+|E|)$ time per cutset are presented. These two algorithms are the most efficient algorithms known to date. However they are both recursive and will generate all possible cutsets when invoked. Since for the test generation propose only one cutset is needed at a time, it is necessary to modify these algorithms so that they become callable routines and each time the algorithms are invoked, only one cutset is returned.

In this study Algorithm 2 [111] is adopted because it is easier to modify it to suit our test generation requirement. Two major modifications have been made: 1) the recursive mechanism of the algorithm has been replaced by a non-recursive one, and 2) the algorithm has been modified such that *multiple entrance* to the routine becomes possible. Making the algorithm non-recursive has the advantage that both memory space and computation time are reduced. The multiple entrance property of a routine refers to the property that allows for repeated calls to the routine, and actions in the routine depend on the status when the routine is called. Another minor modification is that for the purpose of detecting a stuck-open fault, a cutset is not desired if it does not contain the faulty transistor. Thus whenever a cutset is generated, it is necessary to check whether it is valid or not.

After these modifications, each time the routine is called it returns a new, valid cutset. This eliminates the necessity for storing all possible cutsets in memory as when a recursive algorithm is used. Also as long as a cutset leads to the generation of a valid stuck-open test, no further generation of other cutsets is necessary. Thus the computation time is also reduced.

Next a procedure based on the modified algorithm is presented. Since many lemmas are required to fully understand the algorithm, the interested reader should refer to the original paper [111].

Procedure **Find-cutset**$(G = (V, E), a, b, s, t)$;
Inputs: V and E are the sets of vertices and edges, respectively; a and b are the two end nodes of the faulty transistor; s and t are the two nodes to be separated. For P-(N-)network s is $VDD(GND)$. t is the output node of the transistor group.
Outputs: A set of vertices such that the edges that have exactly one end node in this set form a cutset, and edge (a, b) is in the cutset.

```
1. if (first_call) do the following, else go to Step 4;
    (a) $S = \{s\}, T = \{t\}$;
    (b) put $(S, T)$ in queue;
    (c) if $((a = s)$ or $(b = s))$ return$(S)$; else goto Step 2;
2. if (queue is empty) return("no more cutsets");
   else do:
    (a) get and delete a pair $(S, T)$ from queue;
    (b) $T' = \{\}$;
    (c) Candidates = $\{v|$ there exists an edge $(u, v)$ such that $u \in S$, $v \notin S \cup T$,
        and $v$ is not an articulation point in $G - S\}$;
3. if (Candidates is empty) go to Step 2;
   else do:
    (a) select and delete one vertex $v$ from Candidates;
    (b) put $(S \cup \{v\}, T \cup T')$ in queue;
    (c) if ( ( $(a \in S)$ and $(b \in G - S)$ ) or ( $(b \in S)$ and $(a \in G - S)$ ) )
        return$(S \cup \{v\})$; else goto Step 4;
4. $T' = T' \cup \{v\}$; goto Step 3;
```

End **Find-cutset**

If a new, valid cutset is found, Procedure **Find-cutset** returns a set of vertices. The cutset consists of the edges that have exactly one end node in this set of vertices.

5.2.2.3 Necessary and Sufficient Condition

In this subsection the necessary and sufficient condition for a pair of test vectors $< T_1, T_2 >$ to be valid under any circuit delay is given.

Let TG_f be the transistor group that contains the faulty transistor t_f. Without loss of generality, assume that t_f is a P-type transistor. Also for simplicity assume that t_f is in a pull-up network of a complex gate. If t_f is a part of a pass transistor, then the t_f stuck-open fault is obviously redundant. Let the output node of TG_f be t and the VDD node be s.

To detect t_f stuck-open, T_1 must set t to 0 and T_2 must set t to 1 in the good circuit and retain t at 0 in the faulty circuit. To retain t at 0 in the faulty circuit no transient path from s to t can exist during the transition from T_1 to T_2. Thus if a cutset that contains t_f can be found and each transistor except t_f in this cutset can be turned off during the transition, then no transient path exists from s to t and the test cannot be invalidated. This condition is in fact necessary and sufficient as is formally described next.

First let the binary operator "$\wedge$" on two vectors be defined as follows. Let $T_d = T_1 \wedge T_2$ where $T_d(i) = 0$ if $T_1(i) = T_2(i) = 0$, $T_d(i) = 1$ if $T_1(i) = T_2(i) = 1$ and $T_d(i) = X$ otherwise, and the index i denotes the i-th bit of a vector.

In the following theorem, the sentence "a vector *holds off* a transistor" means that when the vector is applied to the circuit under test using the simulation procedure described in Section 6.3, the transistor is turned off.

Theorem 12 *Let $< T_1, T_2 >$ be a test for a transistor t_f stuck-open when no circuit delay is considered. Let t be the output node of the transistor group containing t_f and let s be the power source that is connected to t in the good circuit when T_2 is applied.*

If $T_d = T_1 \wedge T_2$, then "$< T_1, T_2 >$ is a valid test for t_f under any circuit delay" if and only if "there exists a cutset C that separates s and t, C contains the faulty transistor t_f, and T_d holds off all transistors in $C - \{t_f\}$."

The proof of this theorem will be presented in the next chapter after the simulation procedure is discussed.

5.2.2.4 An Efficient Test Generation Procedure

With the results of the previous subsection, a test generation procedure for stuck-open faults can be implemented as described next.

Procedure **TG_SOP**(t_f);
Inputs: A circuit with a stuck-open transistor t_f.
Outputs: A pair of test vectors for t_f if it is detectable; otherwise reports t_f is undetectable.

1. Find a new cutset C that contains the faulty transistor. If no such cutset exists, return("No valid test").
2. Find a new vector T_d that holds off all edges in $C - \{t_f\}$ with the following constraints.
    (a) T_d should not set t to any logic value.
    (b) T_d should not turn off t_f in the good circuit.

    If no such T_d exists, go to Step 1.
3. Find a new T_1 based on T_d, i.e., copy all input assignments on T_d to T_1 and continue assigning values to more inputs until t is set to the required value v or it is found that t cannot be set to v. If successful, go to Step 4. Otherwise go to Step 2.
4. Find a new T_2 based on T_d, i.e., copy all input assignments of T_d to T_2 and continue assigning values to more inputs until the condition for T_2 is satisfied or it is found that the conditions cannot be satisfied. The conditions for T_2 are:
    (a) t_f is turned on in the good circuit, and
    (b) a fault effect can be observed at a primary output.

    If the above conditions are satisfied, return a test $< T_1, T_2 >$; otherwise go to Step 3.

End **TG-SOP**

5.2.3 Invalidation due to Charge Sharing

5.1 The invalidation of a stuck-open test due to charge sharing is difficult to solve using conventional test strategies. This subsection presents a layout-driven method to characterize this problem and shows that by monitoring the current supply, the charge sharing problem becomes much easier to solve. The results reveal that by slightly modifying the layout of a circuit the charge sharing problem can be eliminated.

5.2.3.1 Problem and Strategy

The circuit shown in Figure 5.20 will be used throughout this subsection. The invalidation problem due to charge sharing in this circuit can be explained as follows. Assume that transistor PA is stuck-open. To detect this fault two vectors must be applied: the first one sets node O_1 to 0 for both the fault-free and faulty circuits; the second vector differentiates the output logic levels at O_1 in the fault-free and faulty circuits. Consider a two vector test $< T_1, T_2 >=< 1111, 0001 >$. When $ABCD = 1111$ is applied, O_1 is set to 0 and nodes x, y and O_1 are isolated from each other. When $ABCD = 0001$ is applied, O_1 should retain its value at 0 in a faulty circuit and change to 1 in the fault-free circuit. However since x, y and O_1 are now connected through conducting transistors, charge sharing among them occurs. In this subsection it will be shown that due to this charge sharing the logic value at O_1 in a faulty circuit may become higher than the threshold of an N-type transistor ($\approx 1V$) and thus there is no guarantee that this value will be interpreted as a logic low by the next stage (or gate).

This problem is difficult to solve using conventional testing techniques which only consider circuit structure information and only monitor the logic response. Charge sharing certainly results in analog circuit behavior and thus unless the capacitance information of the circuit is available or can be characterized, the charge sharing problem would be almost impossible to solve. Just analyzing

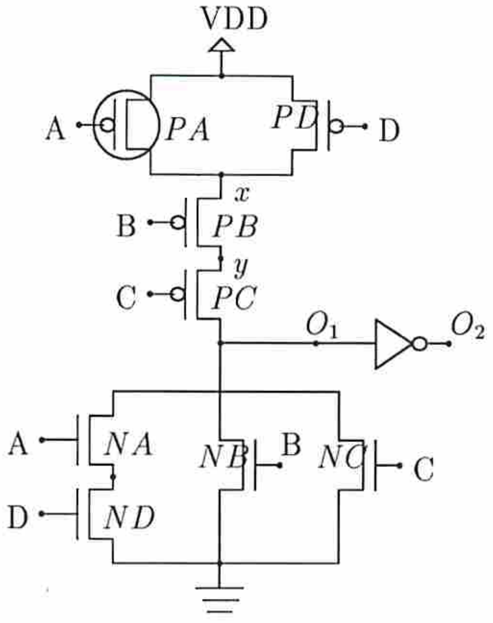


Figure 5.20: Invalidation of tests due to charge sharing

this problem is difficult because most circuit simulators currently in use, including SPICE, cannot precisely predict the results of charge sharing [113], not to mention simulators which use only switch-level models.

Because of the problems discussed above, most previous research dealing with CMOS stuck-open faults either totally ignores the charge sharing problem [35, 92, 44, 97, 114, 40] or only considers the worst case, i.e, if charge sharing is possible then the test is considered as invalid [48, 115, 86]. Neither of these two approaches leads to a satisfactory solution. A new approach to this problem is thus necessary if high quality tests are to be obtained.

The use of CSM for the detection of CMOS short faults has been discussed. It can be seen that the only requirement for CSM to detect a fault is that when an appropriate test vector is applied, a large current must exist due to the fault. Thus CSM is not limited to the detection of short faults only. It has been shown that some faults caused by an open conductor also result in large currents and thus CSM is also applicable to these faults [73, 75]. For example in [75] it is reported that after a long duration (e.g., 100 seconds) a floating node caused by a stuck-open fault may drift to an intermediate voltage that can turn on both P- and N-type transistors. However within a normal clock period, a floating node may retain its previous voltage value or may share its charge with other nodes connected to it. The latter provides a clue to deal with the charge sharing problem in testing stuck-open faults.

In this subsection the charge sharing problem is analyzed based on the layout of a circuit. First a description of how to estimate the capacitances of various components in a circuit and how charge sharing may indeed invalidate a test is provided. This establishes the fact that if the charge sharing effect is totally ignored, the resulting tests may not be reliable. It is then shown that if the worst case situation is assumed, then the detectability of faults may be dramatically reduced. The capability of CSM in solving the charge sharing problem is then examined. It is shown that by employing CSM in addition to conventional logic monitoring, the detectability for stuck-open faults that may result in charge sharing is greatly enhanced. In most cases, this problem can be easily solved. For those few cases where charge sharing may indeed invalidate a test, a fault may still be detected by slightly modifying the circuit layout.

5.2.3.2 Estimation of Capacitance

The circuit shown in Figure 5.21 will be used to illustrate the estimation of capacitances. The estimates presented in this section are based on a 4 μm technology. The capacitance values for more advanced processes will be given later. Figure 5.21(a) shows the layout of the circuit, where rectangles F, E, A and D are polysilicon areas, B and C are diffusion areas, G and F are metal, and H is a metal-poly contact. The area under rectangle A is a transistor. The size of each rectangle is indicated. All units are in $\mu m (10^{-6} m)$. The circuit diagram of this layout is indicated in Figure 5.21(b) (assume this transistor is n-type). Figure 5.21(c) shows an approximate 3-dimensional picture of this circuit.

The layout of a circuit can be classified into three basic areas: gate area, diffusion area and routing area [1]. In Figure 5.21(a), rectangle A belongs to the

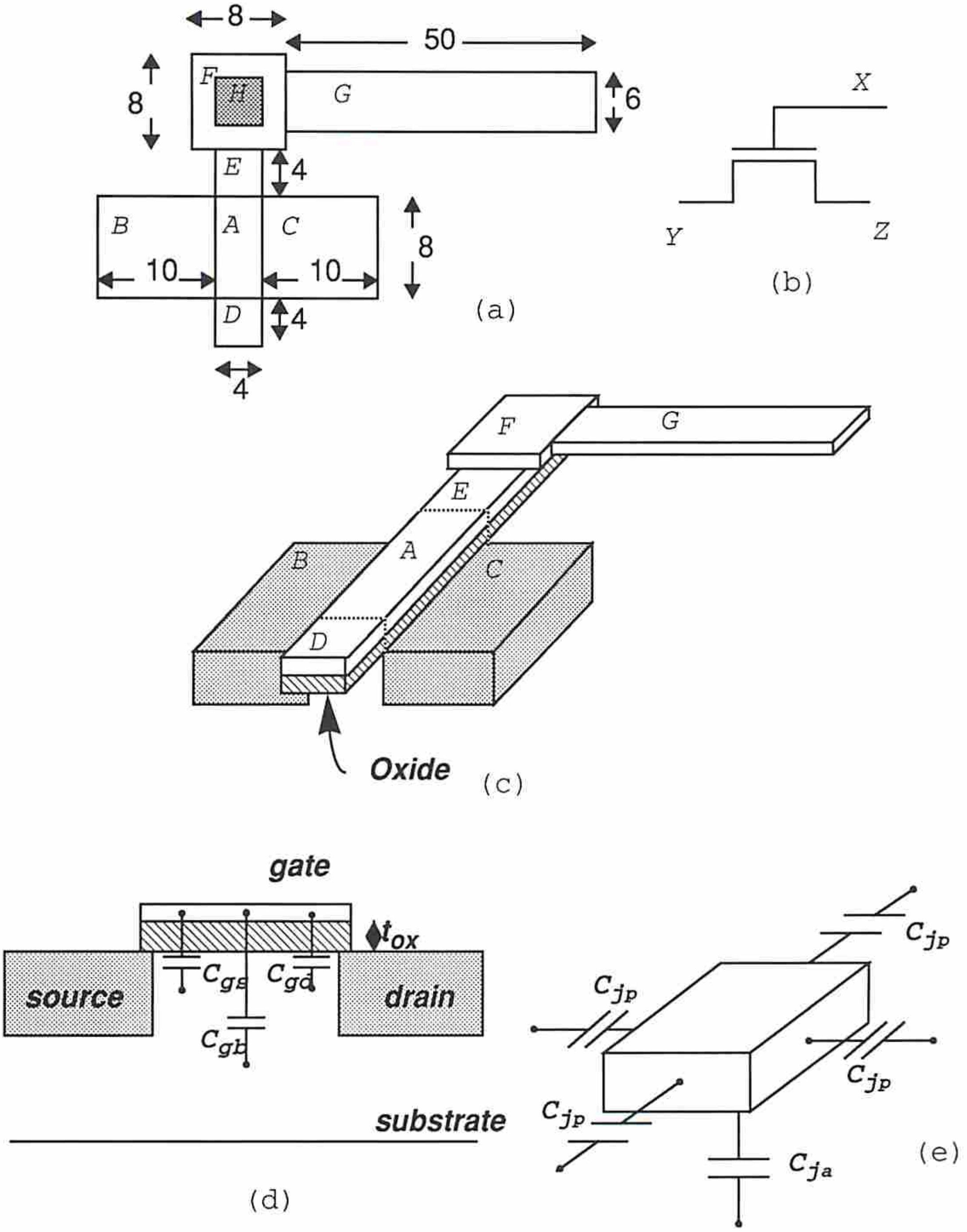


Figure 5.21: Capacitance estimation of various components

gate area, B and C are in diffusion area, and the remaining rectangles belong to the routing area.

Gate Capacitance

The capacitance C_g of a gate area is the sum of the gate to source capacitance C_{gs}, gate to drain capacitance C_{gd} and gate to bulk (or substrate) capacitance C_{gb} as illustrated in Figure 5.21(d). The total gate capacitance C_g is approximately [1]:

$$C_g = C_{gs} + C_{gb} + C_{gd} \approx \left(\frac{\varepsilon_o \varepsilon_{SiO_2}}{t_{ox}}\right) \cdot A_g \tag{5.1}$$

where ε_o = permittivity of free space = $8.85 \times 10^{-12} F/m$, ε_{SiO_2} = relative permittivity of silicon dioxide = 3.9, t_{ox} is the thickness of thin-oxide, and A_g is the area of the gate. Let $t_{ox} = 1000 \times 10^{-10} m$. Then the gate capacitance of the circuit in Figure 5.21 can be approximated as (in this section a subscript x under character C denotes the capacitance of region or node x):

$$\begin{aligned} C_A &\approx (3.9 \times 8.85 \times 10^{-12}/1000 \times 10^{-10}) \\ &\quad \times 8 \times 4 \times 10^{-12} \\ &= 11.0 \times 10^{-15} F \end{aligned}$$

Diffusion Capacitance

All diffusion regions have a capacitance to substrate (or well) that depends on the voltage between the diffusion regions and the substrate. This capacitance, denoted as C_d, is proportional to the total diffusion to substrate junction area that consists of one "base" area and 4 "sidewall" (or peripheral) areas as shown in Figure 5.21(e). The base capacitance, denoted as C_{j_a}, is proportional to the area of the diffusion and has a unit of F/m^2. When assuming constant depth diffusion, the periphery capacitance C_{j_p} can be characterized by a unit-length

Para.	Typical value	Comments
C_g	$4.0 \sim 5.0 \times 10^{-4}\ F/m^2$	Gate
C_p	$0.4 \sim 0.6 \times 10^{-4}\ F/m^2$	Poly over field
C_{m_p}	$0.4 \sim 0.6 \times 10^{-4}\ F/m^2$	Metal over poly
C_{m_f}	$0.15 \sim 0.3 \times 10^{-4}\ F/m^2$	Metal over field
C_{m_d}	$0.8 \sim 1.0 \times 10^{-4}\ F/m^2$	Metal over diffusion
C_{ja_n}	$0.8 \sim 1.0 \times 10^{-4}\ F/m^2$	n-diffusion base
C_{jp_n}	$7.0 \sim 9.0 \times 10^{-10}\ F/m$	n-diffusion periphery
C_{ja_p}	$0.8 \sim 1.0 \times 10^{-4}\ F/m^2$	p-diffusion base
C_{jp_p}	$6.0 \sim 8.0 \times 10^{-10}\ F/m$	p-diffusion periphery
C_{m2_f}	$0.1 \sim 0.15 \times 10^{-4}\ F/m^2$	Metal 2 over field
C_{m2_p}	$0.2 \sim 0.3 \times 10^{-4}\ F/m^2$	Metal 2 over poly
C_{m21}	$0.3 \sim 0.5 \times 10^{-4}\ F/m^2$	Metal 2 to metal 1

Table 5.2: Typical capacitance values in a $4\mu m$ process [1]

capacitance F/m. Thus the capacitance of a diffusion region with area A_d and perimeter P_d can be estimated as

$$C_d = C_{j_a} \cdot A_d + C_{j_p} \cdot P_d \tag{5.2}$$

Both C_{j_a} and C_{j_p} depend on the voltage across the diffusion to substrate junction. Typical values for a 4 μm CMOS process are displayed in Table 5.2. When the diffusion area is scaled down, the effects of peripheral capacitance become more significant. For our example in Figure 5.21, the capacitance of C_B or C_C is approximately

$$\begin{aligned} C_B \approx C_C &\approx 1 \times 10^{-4} \times (10 \times 8 \times 10^{-12}) \\ &\quad +8 \times 10^{-10} \times (20 + 16) \times 10^{-6} \\ &= 80 \times 10^{-16} + 288 \times 10^{-16} \\ &= 36.8 \times 10^{-15} F \end{aligned}$$

Routing Capacitance

Routing capacitances between two layers can be approximated using a parallel plate model with a compensation factor, or

$$C = K \cdot \frac{\varepsilon}{t} A \tag{5.3}$$

where ε is the permittivity of the insulating material between the two layers, t is the thickness of the insulator, and A is the area of the plates. K is a factor that takes into account the fringing effect due to other routing area. Interlayer capacitance (such as metal-poly capacitance) is also enhanced by fringing. The typical value of K is $1.5 \sim 3$ [1]. Various routing capacitances are also given in Table 5.2.

Using the values in Table 5.2, the capacitances of rectangles G, F, E and D in Figure 5.21 can be computed as:

$$\begin{aligned}
C_G &= 50 \times 6 \times 10^{-12} \times 0.3 \times 10^{-4} = 9.0 \times 10^{-15} F \\
C_F &= 8 \times 8 \times 10^{-12} \times 0.6 \times 10^{-4} = 3.84 \times 10^{-15} F \\
C_D &= C_E = 4 \times 4 \times 10^{-12} \times 0.6 \times 10^{-4} \\
&= 0.96 \times 10^{-15} F
\end{aligned}$$

The capacitance of each "node" shown in Figure 5.21(b) can be estimated as: $C_X = C_G + C_F + C_E + C_A + C_D \approx (9.0+3.84+0.96+11.0+0.96) \times 10^{-15} F = 25.8 \times 10^{-15} F$; $C_Y \approx C_Z \approx 36.8 \times 10^{-15} F$. This example shows that the capacitances of a gate node and the drain (or source) node are quite close. This implies that charge sharing may be a problem. Next the circuit example given in Figure 5.20 is used to verify this effect.

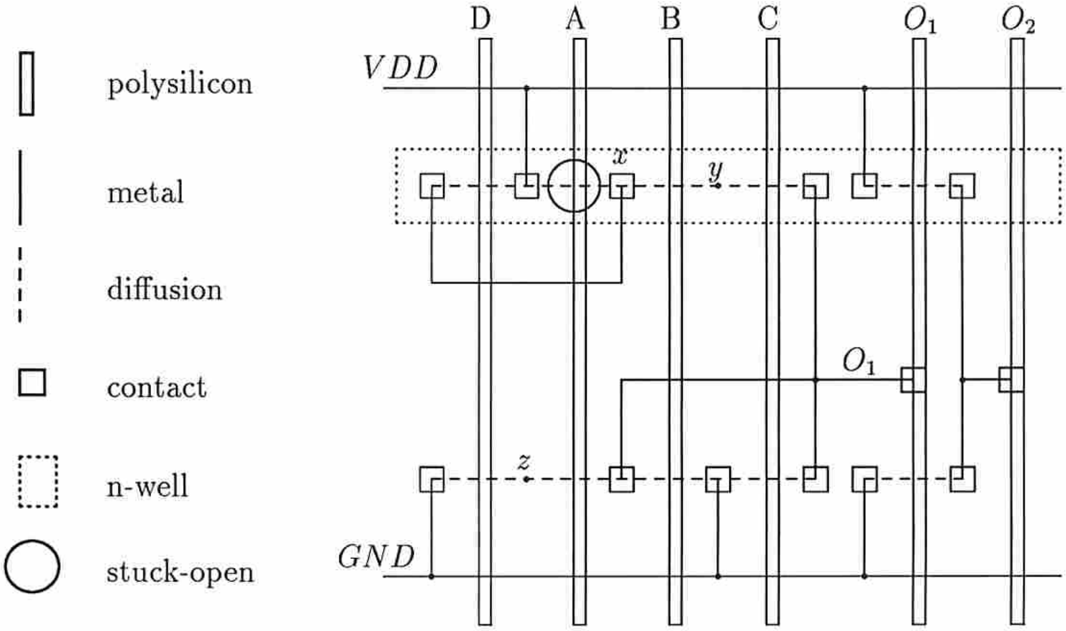


Figure 5.22: One possible layout of Figure 5.20

5.2.3.3 Invalidation of Tests

Figure 5.22 shows one possible layout of the circuit shown in Figure 5.20. Assume $ABCD = 1111$ is the initialization vector. As described before, when the test vector $ABCD = 0001$ is applied, charge sharing among x, y and O_1 occurs. Since only O_1 is set to 0 by the initialization vector, both x and y may have a voltage of $5V$ before charge sharing. Thus the final voltage at O_1 may be $V_{O_1} = (0 \times C_{O_1} + 5 \times (C_x + C_y))/(C_{O_1} + C_x + C_y)$[1]. From an actual layout corresponding to the stick diagram of Figure 5.22 the area of node x plus the area of node y is approximately the same as the area of O_1, and the resulting voltage on V_{O_1} after charge sharing will no longer be below the threshold voltage of an n-transistor, and the test may be invalidated.

All the above estimations are based on a $4\mu m$ process. Similar arguments, however, can be applied to more advanced processes. For example, in [116] the gate capacitance has been shown to be about the same as the diffusion capacitance

[1] See Appendix of this chapter

Parameter	$4\mu m$	$2\mu m$	$1.2\mu m$	Unit
T_{ox}	1000	405	205	$10^{-10}m$
C_{j_a}	0.9	4.19	4.84	$10^{-4}F/m^2$
C_{j_p}	8	2.83	1.27	$10^{-10}F/m$
A_g	8×4	4×2	3.6×1.2	$10^{-12}m^2$
A_d	8×10	4×5	3.6×3.6	$10^{-12}m^2$
P_d	18×2	9×2	7.2×2	$10^{-6}m$
C_g	11.0	6.8	7.3	$10^{-15}F$
C_d	36.8	13.48	8.1	$10^{-15}F$

Table 5.3: The comparison of C_g and C_d in 4, 2, 1.2 μm technologies

for a typical $1.75\mu m$ process. In Table 5.3 the values of the gate and diffusion capacitances for a N-type transistor in three different technologies are given. The data for $2\mu m$ and $1.2\mu m$ are based on typical technologies used in MOSIS. The routing capacitances are not given because they are relatively small compared with gate and diffusion capacitances. From the table it is clear that gate and diffusion capacitances differ by a factor of about 3 or less. Although all the estimations are approximations, the above discussion clearly implies that charge sharing may indeed invalidate a two-pattern vector. Thus completely ignoring this effect may affect the quality of the test.

Next the problem that can result from assuming the worst case situation is examined. The worst case situation refer to the case that whenever charge sharing is possible, the test is considered invalidated.

5.2.3.4 Worst Case Consideration

When an input vector is applied to a circuit a node in the circuit may be either 1) connected to *VDD* through conducting transistors, 2) connected to *GND* through conducting transistors or 3) isolated from *VDD* and *GND*. The detection of stuck-open faults relies on the charge retention on the output node, say O, of the faulty gate (or transistor group). Assume O is set to a logic value v by the initialization vector. If the worst case situation of charge sharing is considered, then all nodes that can be connected to O during the transition from the first vector to the second vector must also have a logic value v in a faulty circuit. Otherwise the test may be invalidated. This implies that all these nodes must be set to v before the test vector (second vector) is applied. If only two-vector tests are allowed, the first vector may not be able to set up all these initial conditions. Next a more formal description about this situation is given. For simplicity it is assumed that each transistor group has only one output. The notation used in the following discussion is defined below.

i: the index of the phase of testing; $i = 1, t, 2$ represent the initialization, transition and test phase, respectively.

t_f: the faulty transistor.

G_f: the transistor group that contains t_f.

OUT: the output node of G_f.

T_1, T_2: the initialization and test vectors.

E_i: for $i = 1$ or 2, the set of conducting transistors in G_f when T_i is applied; $E_t = E_1 \cup E_2$.

S_i^n: the set of nodes in the faulty circuit that are connected to node n if all transistors in E_i are conducting; $n \in \{VDD, GND, OUT\}$.

By these definitions S_2^{GND} is the set of nodes that are connected to GND when T_2 is applied, and S_d^{OUT} is the set of nodes that may invalidate the two-vector test. Note that E_1 and E_2 depends on T_1 and T_2, respectively, while E_t is defined as $E_1 \cup E_2$ so as to consider the worst case charge sharing situation.

Without loss of generality, assume T_1 sets OUT to 0. The following facts can be derived.

Fact 1: $S_1^{OUT} = S_1^{GND}$

Proof: When T_1 is applied, there is a conducting path from GND to OUT. □

Fact 2: $S_d^{OUT} \supseteq (S_1^{OUT} \cup S_2^{OUT})$, but $S_d^{OUT} = (S_1^{OUT} \cup S_2^{OUT})$ is not always true.

Proof: Since transistors that are conducting during the initialization or test phase are assumed conducting during the transition phase ($E_t = E_1 \cup E_2$), all nodes connected to OUT during the initialization or test phases must be connected to OUT during the transition phase. The fact that $S_t^{OUT} = (S_1^{OUT} \cup S_2^{OUT})$ is not always true is illustrated in Figure 5.23, where $X \notin S_1^{OUT}$ or S_2^{OUT} but $X \in S_t^{OUT}$. □

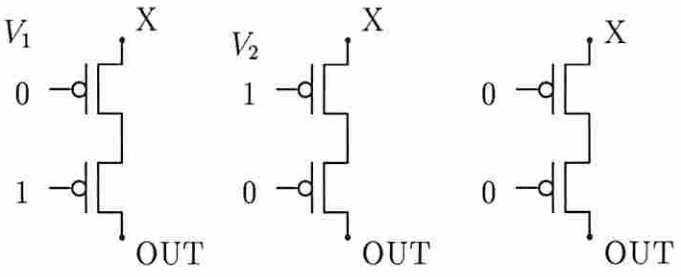


Figure 5.23: Example of $S_t^{OUT} \neq (S_1^{OUT} \cup S_2^{OUT})$

Fact 3: If $< T_1, T_2 >$ is a robust test under any charge sharing situation, then $S_t^{OUT} \subseteq S_1^{OUT}$.

Proof : If there exists a node in S_t^{OUT} but not in S_1^{OUT}, then by Fact 1 this node cannot be precharged to 0 by T_1. Thus this node may invalidate the test. □

Fact 4: If $< T_1, T_2 >$ is a robust test under any charge sharing situation, then $S_1^{GND} = S_1^{OUT} = S_t^{OUT} \supseteq S_2^{OUT}$

Proof: Direct result of Facts 1, 2 and 3. □

Fact 4 formally describes a special property of T_1: it must precharge all nodes that are connected to the output node when T_2 is applied. Considering the fact that T_1 must also set a path from GND to OUT, it is expected that the number of qualified vectors for T_1 is dramatically reduced when this worst case charge sharing condition is considered. The following example gives a clearer picture of this problem.

Example: Again consider the test for the PA stuck-open fault in Figure 5.20. The test vector T_2 must be $ABCD = 0001$ so that a faulty circuit can be distinguished from a fault-free circuit. For T_1, D must be 1, otherwise due to various circuit delays the test may be invalided if T_2 set B, C to 0 before it sets D to 1. If charge sharing is not considered, then there are 7 possible combinations of A, B, C (except $ABC = 000$) that can set the output node to 0. Each of these 7 vectors along with T_2 forms a test that cannot be invalidated by any circuit delays. However, if the worst case charge sharing situation is considered, then both x and y must be precharged to 0. Only one initialization vector, namely $ABCD = 1001$, can set up this condition. Thus the number of qualified initialization vectors is reduced from 7 to 1. If the circuit in Figure 5.20 is embedded in a large circuit, the probability of setting $ABCD = 1001$ becomes much smaller. Thus a test generator that takes only the worst case conditions into account may report this fault as undetectable. □

In general if only the worst case condition is considered when dealing with the charge sharing problem, the number of valid initialization vectors is reduced. This also reduces the detectability of the stuck-open faults. There is no efficient solution to this problem if only conventional logic monitoring and two-vector tests are used. Some researchers have tried to solve this problem by using multiple vector (more than 2) tests such that before the test vector is applied, all nodes that may be connected to the output node are precharged to the required values [48, 86]. This method has several deficiencies: the generation of such multiple vector tests require more CPU time and memory space; even when multiple vectors are allowed, it may be impossible to simultaneously precharge these nodes to the same value; finally test application time may increase because of the increased number of vectors.

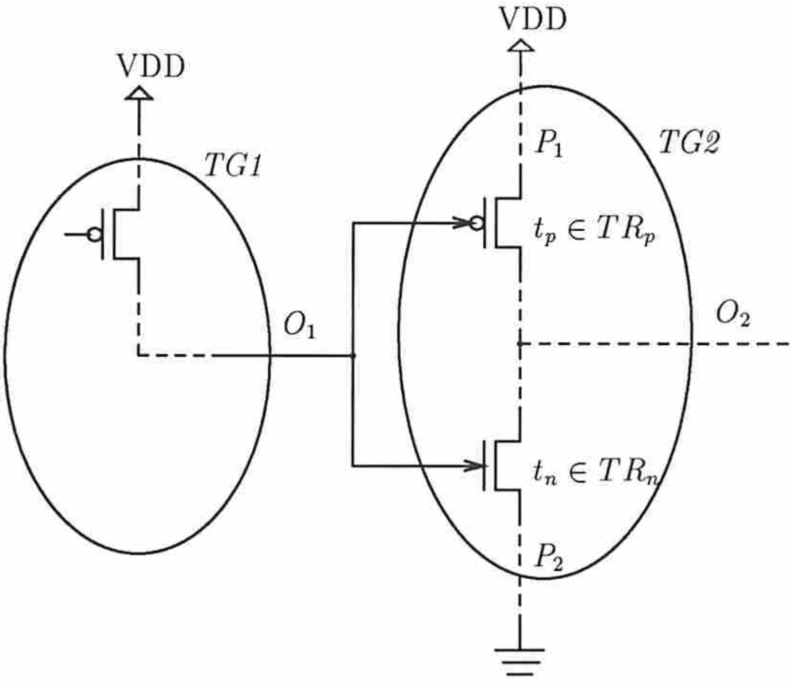


Figure 5.24: Detection of stuck-open faults under charge sharing

Next it will be shown that by employing the CSM method, a test that is valid under any circuit delay will remain valid under most charge sharing situation. The fault either will still be detected by logic monitoring or the charge sharing effects will be detected by CSM.

5.2.3.5 Using CSM to Solve the Charge Sharing Problem

Refer to Figure 5.24. Assume a stuck-open fault occurs in transistor group *TG1*. Let T_1 and T_2 be a valid test for this fault under any circuit delay using logic monitoring. When T_2 is applied, there must be a sensitized path P from one output, say O_1, of *TG1* to some primary output.

If O_1 itself is a primary output, then any charge sharing effect can be observed directly. Now assume that O_1 is not a primary output. Without loss of generality assume when T_2 is applied, O_1 has a value 1 in the fault-free circuit and 0 in the faulty circuit if no charge sharing exists. O_1 must feed another transistor group *TG2* that has an output, say O_2 on the sensitization path P, and the value of O_2 is totally dependent on the value of O_1. Again without loss of generality it may be assumed that $O_2 = 1$ if $O_1 = 0$ and $O_2 = 0$ if $O_1 = 1$. This implies that if $O_1 = 0$, there must exist a path, say P_1, of conducting transistors from *VDD* to O_2. Similarly if $O_1 = 1$, there must exist a conducting path, say P_2, from *GND* to O_2. Let TR_p and TR_n be, respectively, the sets of transistors on P_1 and P_2 which are controlled by O_1. If there is no charge sharing, each transistor in TR_p will be conducting while those in TR_n will be non-conducting in the faulty circuit.

Now consider the problem due to charge sharing. Let v be the voltage value of O_1 after charge sharing. If v is less than the threshold voltage of n-type transistors ($\approx 1V$), then charge sharing has no effect on the circuit and the fault is detected by logic monitoring. If v is greater than the threshold voltage of n-type transistors but not large enough to turn off p-type transistors ($\approx 4V$), then both transistors in TR_n and TR_p will conduct. Since *TG2* is on the sensitized path, the value of O_2 is determined by the conducting conditions of TR_p and TR_n. This implies both paths P_1 and P_2 are conducting, and O_2 must be connected to both *VDD* and *GND*. Therefore by monitoring the current supply, the fault effect is detected. The last case is when v is greater than about $4V$ and a p-type transistor is off. In this case the test is invalidated. However to have this situation v must change from $0V$ to more than $4V$. Let $S = S_1 \cup S_0$ be the set of nodes which can share charge with O_1, where S_0 are the set of nodes which have been precharged to 0, and $S_1 = S - S_0$ (all the reminding nodes). To invalidate the test the total capacitance of nodes in S_1, denoted as C_{S_1}, must be at least 4 times that of nodes in S_0 (C_{S_0}). This is very unlikely as can be seen from the previous estimation of capacitances, especially when *TG1* has a small number of transistors.

Next the improvements that can be achieved by using CSM is examined. Without CSM, the ratio of C_{S_1} over C_{S_0} should be lower than 1/4. If CSM is used along with the logic level monitoring, this ratio can be as high as 4/1. If an *improvement factor* (IF) is defined as the ratio of the above two ratios (with and without CSM), then $IF = (4/1) \;/\; (1/4) = 16$. Such a high improvement factor clearly shows the advantage of using CSM to deal with the charge sharing problem. Also because of this high improvement factor, the CSM test methodology is still valid even if the estimates of capacitances are not very precise. For the extreme case where $C_{S_1} > 4C_{S_0}$, one may modify the layout to force $C_{S_1} < 4C_{S_0}$. The degree of this modification is obviously much less than that required for the case where CSM is not used.

5.2.3.6 An Algorithm for Generating Robust Tests

An algorithm for generating robust tests is given below. Again without loss of generality it is assumed that T_1 sets OUT to 0.

Algorithm **ROBUST-TG-SOP**
Inputs: A circuit containing a stuck-open transistor. Each node in the circuit is associated with a capacitance.
Outputs: A pair of robust test vectors for the stuck-open transistors and the required modifications of the layout if the fault is detectable; otherwise reports the fault is undetectable.

1. Estimate the capacitance of each node in the circuit based on the layout information.

2. Find a new test $< T_1, T_2 >$ that cannot be invalided by circuit delays using the procedure **TG-SOP** described in the previous section. If this fails, then report the fault as undetectable and exit.

3. Derive S_1^{OUT}, S_2^{OUT} and S_t^{OUT} from T_1 and T_2. Derive S_1 and S_0 from S_1^{OUT}, S_2^{OUT} and S_t^{OUT}. If $C_{S_1} < 4C_{S_0}$ then report $< T_1, T_2 >$ as a test and exit. Otherwise if the layout is allowed to change then go to Step 4, else go to Step 2.

4. Try to increase the size of nodes in S_0 or decrease the size of nodes in S_1 such that $C_{S_1} < 4C_{S_0}$. If this can be done then report this layout change, report $< T_1, T_2 >$ as a test, and exit, else go to Step 2.

End **Robust-TG-SOP**

In Step 3, the derivation of S_1^{OUT}, S_2^{OUT} and S_t^{OUT} can be done by simulation. A simulator that, for a given input vector, is capable of identifying all nodes connected to the OUT node is sufficient. The most conservative estimate for S_0 is $S_0 = S_1^{OUT} \cap S_2^{OUT}$ because only the nodes in S_1^{OUT} are guaranteed to be precharged and, of these nodes, only the nodes which are also in S_2^{OUT} are guaranteed to be involved in the charge sharing. $S_1 = S_t^{OUT} - S_0 - \{GND\}$ must be used because every node in S_t^{OUT} may be involved in the charge sharing, and only the nodes in S_0 and $\{GND\}$ definitely have the value of 0.

Note that the effects of charge sharing are checked only as a postprocess after T_1 and T_2 have been generated. This will speed up the test generation process since it can be expected that the probability of test invalidation due to charge sharing is very small. The amount of backtracking due to charge sharing should be much smaller than that found in Step 2.

5.2.3.7 Summary

The charge sharing problem associated with the detection of CMOS stuck-open faults has been analyzed. It has been shown that this problem cannot be ignored if high quality tests are required. If the worst case condition is assumed and only conventional logic monitoring is used the detectability of stuck-open faults may be dramatically reduced. However, by employing CSM and layout information it has been shown that this problem can be easily solved. Although the estimations made in this section are based on a $4\mu m$ process, similar results are applicable to more advanced processes. Through the use of CSM the very high improvement factor that has been obtained can easily offset the error caused by imprecise estimations of capacitance.

5.3 Line Breaking Faults

Once the test generation algorithm for stuck-open faults is developed, a line breaking fault can be tested by modeling it as a transistor stuck-open fault as described below.

Assume the two end nodes of the line under consideration are a and b, then a breaking fault on this line can be modeled by replacing the line with a transistor t such that the drain and source terminals of t are a and b, respectively. In a good circuit transistor t is always turned on so that no break between a and b exists, while in the faulty circuit transistor t is always off and thus the break effect can be modeled.

Appendix

In Section 5.2.3.3 the result of charge sharing is given by the formula $V_{O_1} = (0 \times C_{O_1} + 5 \times (C_x + C_y))/(C_{O_1} + C_x + C_y)$. In this appendix a more detailed derivation of this formula is given. For simplicity assume only two nodes, a and b, are involved in charge sharing, where a is an output node of a CMOS gate and b is an internal node in the pull-up side of the same gate, as shown in Figure 5.25(a). The implementation of this circuit is shown in Figure 5.25(b). The bulk (or n-well) of transistors PA and PB must be connected to VDD. Let b and a be precharged to $5V$ and $0V$, respectively. The capacitance on node a can be considered as consisting of two parts: the capacitance C_{a_1} whose other side is connected to GND, and the p-diffusion capacitance C_{a_2} whose other side is connected to VDD. The former charge consists of gate capacitance, routing capacitance and n-diffusion capacitance.

Before charge sharing, i.e., before PA is turned on, the equivalent circuit of this circuit is shown in Figure 5.25(c). There are some charge accumulated in C_{a_2}. When PA is turned on, the equivalent circuit is shown in Figure 5.25(d). Some positive charge on the upper side of C_b will flow to the upper sides of C_{a_1} and C_{a_2}, and/or some negative charge on the upper sides of C_{a_1} and C_{a_2} will flow

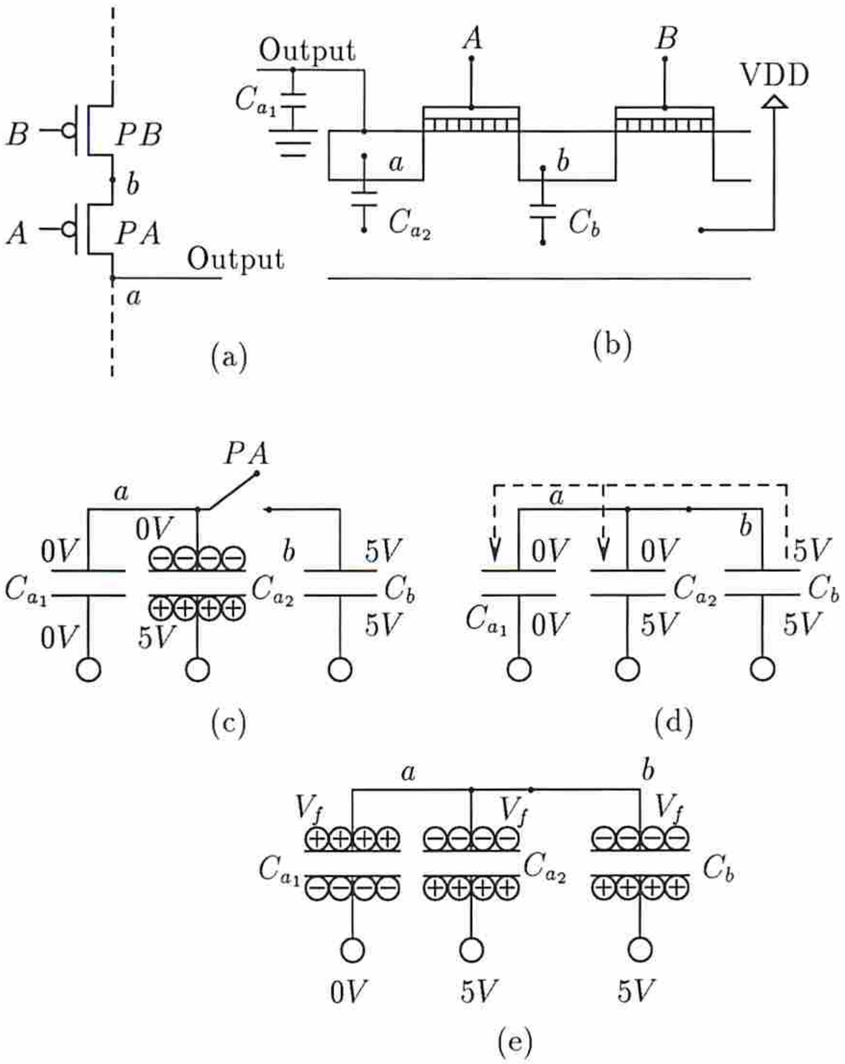


Figure 5.25: Detailed analysis of charge sharing

to the upper side of C_b. Thus the final result of this charge redistribution will be as shown in Figure 5.25(e). According to the conservation law of charge, the total change of charge in all capacitances must be zero. Thus the final voltage value V_f on both nodes a and b must satisfy the equation

$$\int_0^{V_f} C_{a_1}(v)dv + \int_0^{V_f} C_{a_2}(v)dv + \int_5^{V_f} C_b(v)dv = 0$$

where both C_{a_1}, C_{a_2} and C_b are functions of the voltages across the capacitors. For simplicity it may be assumed that C_{a_1}, C_{a_2} and C_b are all constant. Thus the above equation becomes

$$C_{a_1}(V_f - 0) + C_{a_2}(V_f - 0) + C_b(V_f - 5) = 0$$

or

$$V_f = \frac{5 \times C_b}{C_{a_1} + C_{a_2} + C_b}.$$

A simplier way for determining the final voltage V_f is by equating the total charges before and after turning on PA. First consider the case before turning on PA. Since both sides of capacitance C_b have the same voltage value ($5V$), the total charge Q_b on C_b is 0 ($Q_b = C_b \times V_{C_b}$). Similar $Q_{a_1} = C_{a_1} \times V_{C_{a_1}} = 0$. The charge on C_{a_2} is $Q_{a_2} = C_{a_2} \times V_{C_{a_2}} = -5C_{a_2}$. After turning on PA, the voltage on both a and b becomes V_f. Thus $V_{C_{a_1}} = V_f - 0$, $V_{C_{a_2}} = V_f - 5$ and $V_{C_b} = V_f - 5$. By equating the total charge, the following equation is obtained.

$$\begin{aligned} -5 \times C_{a_2} &= (V_f - 0) \times C_{a_1} \\ &\quad +(V_f - 5) \times C_{a_2} + (V_f - 5) \times C_b. \end{aligned}$$

Again, the same result is obtained.

$$V_f = \frac{5 \times C_b}{C_{a_1} + C_{a_2} + C_b}.$$

Chapter 6

Test Generation Framework

In this chapter a switch level test generation framework is presented. The basic idea for developing such a framework is as follows. At the switch level many different fault models exist. If test generators for each individual fault model were developed independently, a lot of effort would be repeated. On the other hand if the major test generation components that will be shared by each individual fault model can be identified and implemented, then to deal with each individual fault model, it suffices to implement the necessary interfaces between these components. The framework to be presented is based on the PODEM algorithm. It contains 5 major components: objective selection, backtracing, logic implication, fault propagation and backtracking.

6.1 Objective Selection

Objective selection is the process of selecting a local goal such that when the goal is achieved, a test for the fault is likely to be identified. In the original PODEM algorithm, only one objective is to be achieved at a time, that is, only one line (or node) is to be assigned a desired value. When dealing with switch level faults, it is possible that more than one objective are to be achieved simultaneously. For example the initial objectives of detecting a bridging fault are to set the two shorted nodes to complementary values. Thus two objectives are needed. Information obtained in circuit manipulation such as transistor directions can be used to determined which objective is better. Another objective selection problem occurs when generating a test for a stuck-open fault. It is necessary to turn off all but the faulty transistor in the cutset. Thus multiple objectives are possible. In this section the above two problems will be addressed, namely dealing with two objectives for bridging faults and multiple objectives for stuck-open faults.

There are of course other problems associated with objective selection. For example during backtracing or fault propagation, it is necessary to select a temporary objective to achieve. To determine what the next temporary objective should be at the switch level is not trivial. However these problems are best discussed together with the processes requiring them, e.g., backtracing or fault propagation.

6.1.1 Initial Objectives for Bridging Fault Testing

To detect a BF (x, y), two sets of initial objectives are possible, i.e., either $x = 1, y = 0$ or $x = 0, y = 1$. The correct selection of the set of objectives may have a large impact on test generation performance since it is quite possible that only one set of the objectives can be achieved. For example for the BF (x, y) in Figure 6.1 only $(x = 1, y = 0)$ is achievable. If $(x = 0, y = 1)$ is selected it is inevitable that all the efforts spent to achieve this goal would be wasted.

Though it is important to identify which set of objectives is better, it is impractical and unnecessary to obtain an exact solution. Experiences acquired from experiments with SWiTEST indicate that the following guidelines in general give quite satisfactory results.

1. If one of x and y is *VDD* or *GND*, immediately assign the correct logic value to this node, and the only objective is to assign the complementary logic value to the other node.

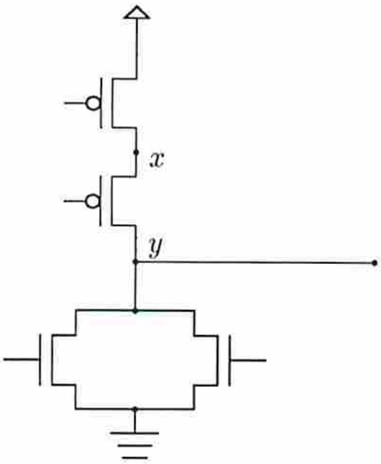


Figure 6.1: Selection of initial objectives: $(x = 1, y = 0)$ is achievable while $(x = 0, y = 1)$ is not

2. If x (y) and y (x) are in P- and N-network, respectively, then the objectives are $x = 1$ and $y = 0$ $(x = 0$ and $y = 1)$.

3. If x (y) is in a P-network and y (x) is an output of a transistor group, then the objectives are $x = 1$ and $y = 0$ $(y = 1$ and $x = 0)$. Similarly if x (y) is in a N-network and y (x) is an output of a transistor group, then the objectives are $x = 0$ and $y = 1$ $(y = 0$ and $x = 1)$.

4. If x and y are two end nodes of a transistor t_f (e.g., x and y in Figure 6.1; this bridging fault is equivalent to a transistor stuck-on fault), then select objectives based on the signal flow direction information of the transistor t_f as illustrated in Table 6.1.

5. If x (y) is an output node of a NAND gate, select $x = 1, y = 0$ $(y = 1, x = 0)$. If x (y) is an output node of a NOR gate, select $x = 0, y = 1$ $(y = 0, x = 1)$.

The above rules should be applied in the order indicated. The rationale for these rules is self-explanatory. If all the above conditions fail, then randomly assign $(x = 1, y = 0)$ or $(x = 0, y = 1)$. If during test generation it is found that one node has been assigned an opposite logic value to its objective value then complement the original objective.

Type of t_f	Direction of t_f	Objectives to select
P	$x \rightarrow y$	$(x = 1, y = 0)$
N	$x \rightarrow y$	$(x = 0, y = 1)$
P	$y \rightarrow x$	$(x = 0, y = 1)$
N	$y \rightarrow x$	$(x = 1, y = 0)$

Table 6.1: Objective selection according to signal flow direction information

6.1.2 Objective Array

Because multiple objectives may exist, it is necessary to use some data structure to store these objectives. In this study an *objective array* is used for this purpose. Each element in this array stores one objective to be achieved. The objective of the first element is first selected and achieved, followed by the next one, and so on. When all objectives in the array are achieved, either the test generation procedure is completed (e.g., detecting BFs using CSM) or the next stage of test generation should start (e.g., the cutset condition for a stuck-open fault is satisfied and next T_1 and T_2 are to be identified).

If during test generation backtracking occurs, the previously achieved objectives must be examined to see whether some of them are affected by the backtracking. Two methods can be used to deal with this problem. The first method is just to check all previously achieved objectives to determine which objectives need be reevaluated. The second method is to record the *first* objective that is achieved by each PI assignment (PIA). This is illustrated in Figure 6.2. PIA 1 achieves objectives a and d, PIA 2 achieves b, PIA 3 achieves c and f, and PIA 4 achieves e. Thus the positions 1, 2, 3 and 5 are recorded for PIAs 1, 2, 3 and 4, respectively. Now if it is found that objective g cannot be achieved, then backtracking on PIA 4 must be performed. By retrieving the number 5 that is associated with the PIA 4, it can be immediately identified that the next objective to be achieved is e and all objectives preceding this objective are not affected.

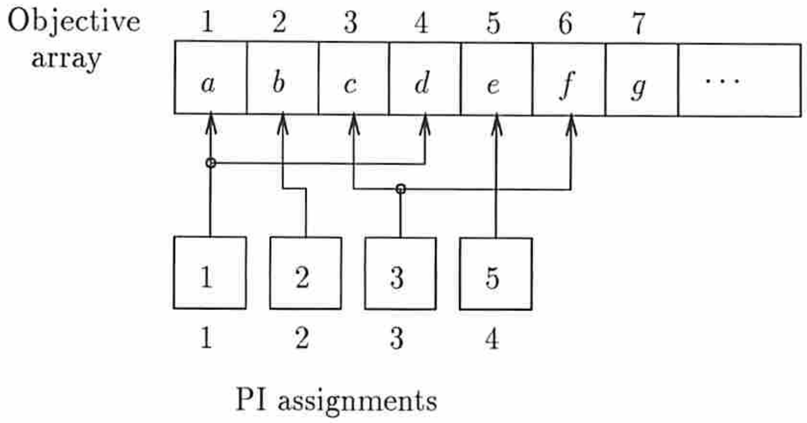


Figure 6.2: Recording the first objective that is achieved by each PI assignment

6.2 Backtracing

Once an objective is identified, the next step is backtracing, i.e., to find a promising PI assignment to achieve the objective. Backtracing always succeeds in the original PODEM algorithm, i.e., there always exist some unspecified primary input(s) so that by assigning logic values to these PIs the status of the objective line (or node) can be changed. This is because for a logic gate G with an unspecified output, there must exist some unspecified input(s) to G and assignments of values to these inputs will give the gate output a specified logic value. For those unspecified input(s) of G assume they are fed by a set of gates GS, then there in turn exist some unspecified input(s) to the gates in GS. Continuing this procedure for a combinational circuit where no loops exist, it is always possible to find a set of unspecified primary inputs whose assignments can change the unspecified status of the objective line. This, however, is not always true at the switch level as explained below.

Refer to Figure 6.3(a). Let the objective be "set x to 1." If due to previous assignments transistors t_1 and t_2 have both been turned off, then there would be no further PI assignments that can set x to 1 unless a backtracking is conducted. This example shows that backtracing at the switch level may fail, i.e., it is not always possible to change the unspecified status of a node by assigning values to more primary inputs.

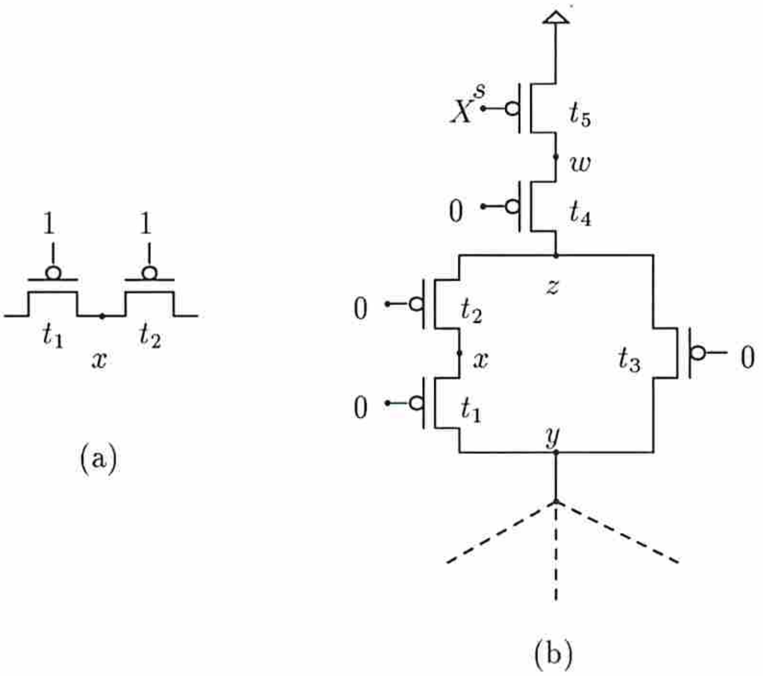


Figure 6.3: Problems at the switch level backtracing: (a) no further assignment can give x a specified logic value, and (b) a loop may exist at the switch level

There exists another problem that must be addressed during backtracing. Consider Figure 6.3(b). Assume the objective is to set y to 1 and transistors t_1, t_2, t_3 and t_4 have been set to ON. A backtracing process may figure out that to set y to 1, a temporary objective of setting x to 1 is a promising one. This in turn leads to another temporary objective $z = 1$. Now to set z to 1, there are two possibilities: going through t_3 or going through t_4. If t_4 is selected, then the next temporary objective will be setting w to 1 and the objective $y = 1$ is likely to be achieved. However if t_3 is selected, then the next temporary objective will be setting y to 1, which is the original objective. In the gate level PODEM algorithm, it is assumed that no loops exist in the circuit and thus the backtracing is a "memoryless" process, i.e., when a new temporary objective is determined, the old objective is simply discarded. If the same mechanism is used for the switch level test generation, then when a "switch level loop" such as the above one exists ($y \rightarrow x \rightarrow z \rightarrow y$), the test generation procedure may enter an infinite loop.

To solve the above two problems while still preserving the advantage of using gate level information, a sophisticated backtracing algorithm, **Backtracing**, is developed. The input to **Backtracing** is an objective to achieve which is given to the algorithm by the second argument "Obj." **Backtracing** tries to identify a promising primary input assignment. When it succeeds, it returns a "TRUE" logic value and the PI assignment is stored in the first argument "PI-obj." If it fails, then the algorithm returns "FALSE." "Current-obj" in the algorithm is used to store each temporary objective. To achieve each temporary objective, the algorithm first checks whether it is an output node of an inverter, NOR or NAND gate. If it is, then the selection of the next temporary objective is similar to that used in the gate level PODEM algorithm. The *hardest* and *easiest* input to a gate is determined by the level information obtained during circuit manipulation.

If the objective node is not an output of a logic gate, then a procedure **Tran-backtracing** is called. This procedure is a true switch level backtracing algorithm. It employs a *hybrid depth-first/breadth-first search* mechanism using two search queues Q_1 and Q_2. This search mechanism guarantees that each node in the transistor group containing the original objective can be used as a temporary objective at most once. It is hybrid because during searching, some nodes will be put at the beginning of the search queues while some will be put at the end of the queues. Whether an objective is to put in Q_1 or Q_2 depends on the likeness of objective to be successful. Information about the type, on/off status, and signal flow direction of each transistor determined this likeness. The objectives in Q_1 are always processed before the objectives in Q_2. By this manner, the temporary objectives that are likely to be successful will be processed first. The **Backtracing** algorithm is given next.

Boolean Algorithm **Backtracing**(PI-obj, Obj)
Inputs: The initial objective "Obj" that contains an objective node "Obj.node" and an objective value "Obj.value."

Outputs: If backtracing succeeds, reports the primary input and the value to be assinged via "PI-obj"; otherwise reports backtracing failed.

```
1. Current-obj.node = Obj.node;
   Current-obj.value = Obj.value;

2. If (Current-obj.node is a PI) do
   (a) PI-obj.node = Current-obj.node;
       PI-obj.value = Current-obj.value;
   (b) return(TRUE);
   else do:
   (a) If (Current-obj.node is the output of an inverter) do
         i. Current-obj.node = input of the inverter;
            Current-obj.value = 1− Current-obj.value;
        ii. Go to Step 2;
   (b) If (Current-obj.node is the output of an NAND gate) do
         i. If (Current-obj.value is 0) Current-obj.node = Hardest input to
            the gate;
            else Current-obj.node = Easiest input to the gate;
        ii. Current-obj.value = 1− Current-obj.value;
       iii. Go to Step 2;
   (c) If (Current-obj.node is the output of an NOR gate) do
         i. If (Current-obj.value is 1) Current-obj.node = Hardest input to
            the gate;
            else Current-obj.node = Easiest input to the gate;
        ii. Current-obj.value = 1− Current-obj.value;
       iii. Go to Step 2;
   (d) Call Tran-backtracing(Current-obj);
       If (Current-obj is not empty) go to Step 2; else return(FALSE);
```

End **Backtracing**

Procedure **Tran-backtracing**(Current-obj)

1. Start with Current-obj.node, do a hybrid depth-first/breadth-first search through each transistor whose status is ON or unknown in the same transistor group. Let $t_v = (u, v)$ be the transistor to be visited next, u be the node currently being visited, and v be the other node of t_v. Also let g be the gate terminal of t_v and Q_1 and Q_2 be the search queues.

   The action when visiting t_v is described in Table 6.2, where

   Sta(t_v) is the status of transistor t_v, Obj(u) is the objective value at node u, Val(v) is the current value of node v, Typ(t_v) is the type of t_v (P-transistor or N-transistor), Dir(t_v) is the signal flow direction of t_v,

   V_{gon} is the logic value of g that turns on transistor t_v,

   [g,V] is the objective of setting g to V,

   BQ_i and EQ_i are the beginning and the end of $Q_i, i = 1, 2$, respectively.

2. Return an empty Current objective;

End **Tran-backtracing**

	Sta(t_v)	Obj(u)	Val(v)	Typ(t_v)	Dir(t_v)	Action
1	X	1 (0)	1 (0)	—	—	Return ([g, V_{gon}])
2	X	1 (0)	X	P (N)	$v \to u$	If v unvisited, put v at EQ_1
3	X	1 (0)	X	Else		If v unvisited, put v at EQ_2
4	ON	1 (0)	X	P (N)	$v \to u$	If v unvisited, put v at BQ_1
5	ON	1 (0)	X	Else		If v unvisited, put v at BQ_2

Table 6.2: Actions to take during backtracing

Consider the example shown in Figure 6.3(a). Since no transistor with ON or unknown status is connected to x, **Tran-backtracing** can immediately return an empty Current-obj to Algorithm **Backtracing** which in turn returns "FALSE" to the program calling it.

Next consider the example shown in Figure 6.3(b). Assume an objective of setting x to 1 is to be achieved. The direction assignment described in Chapter 4 will assign a downward direction to all transistors in Figure 6.3(b). Since both

t_1 and t_2 are ON, z and y will be put in the beginnings of Q_1 and Q_2 according to rows 4 and 5 of Table 6.2, respectively. Thus z will be the next node to be processed. Since t_4 is ON, w will be put in the beginning of Q_1. No action will be taken on t_3 since it is ON and y is already in Q_2. Thus the next objective is to set w to 1. Procedure **Tran-backtracing** will then return $[s, 0]$ as the Current-obj to Algorithm **Backtracing** according to the row 1 in Table 6.2. Note that though Q_2 still contains y, it is not necessary to further process this node.

It is clear that by using **Backtracing** and **Tran-backtracing**, both problems described previously, namely "no more PI assignment" and "switch level loops," can be solved. Also because the direction information and two queues are used , the search process in **Tran-backtracing** is guided and can exit as early as possible, thus improving the efficiency of test generation.

6.3 Logic Implication

During the initialization step of the PODEM algorithm each primary input is set to a *don't care* (or X) state. As test generation progresses, new values are assigned to the primary inputs and the effects of the assignments are propagated to other parts of the circuit. Define a node as a **1-node (0-node)** if it is *VDD* (*GND*) or is a primary input with a logic value 1 (0). Since the output of a transistor group can be set to 1 (0) only when a conducting path from some 1-node (0-node) to that node is established, it is not necessary to perform logic implication on a group if it is known that such a path cannot be set up. Based on this observation, an efficient algorithm called **Implication** has been developed.

Implication is a recursive algorithm. It is basically a "depth-first search" starting with the node N that is to be assigned a logic value V. When invoked as **Implication**(N, V), it first saves the old value of N before assigning the new value V to N. This is necessary for the backtracking purpose as will be described later. In the **Implication** algorithm n.value and n.old-value are the current and old logic values of node n, respectively. All unassigned nodes that are connected to N through conducting transistors are then assigned a value of V. When the status of the gate terminal of a transistor is changed such that the transistor is turned on, the status of the drain and source nodes of the transistor are examined. If one of them, say v, is X and the other has a specified logic value, then v will be assigned the logic value of the other node and **Implication** continues on v. If it is found that two nodes with complementary logic values are connected, the program reports a message which indicates something may be wrong (e.g., the circuit under test does not satisfy rule A2 given in Section 5.1.3).

```
Algorithm Implication($N$, $V$)
Inputs: A node $N$ and a logic value $V$ to be assigned to node $N$. The value of
node $N$ is represented as $N$.value.
Outputs: Logic assignments to nodes that are affected by the assignment of $V$ to
$N$.

  1. $N$.old-value = $N$.value;
     $N$.value = $V$;

  2. For each node $u$ that is connected to $N$ through a conducting transistor,
     do:

       If $u$.value = $X$ do Implication($u$, $V$);
           else if $u$.value = $(1 - V)$ then print ("A conducting path from VDD
           to GND exists") and exit;

  3. For each transistor $t$ controlled by $N$, if $t$ is turned on by assigning $V$ to
     $N$, do the following:

       Let $t_s$ and $t_d$ be the source and drain nodes of $t$ respectively.

       If ($t_s$.value = $X$ and $t_d$.value $\neq X$) do Implication($t_s$, $t_d$.value);
           else if ($t_d$.value = $X$ and $t_s$.value $\neq X$) do Implication($t_d$, $t_s$.value);
           else if ($t_d$.value $\neq$ $t_s$.value) print ("A conducting path from VDD to
           GND exists") and exit;

End Implication
```

The advantage of **Implication** is that it is an *incremental, event-driven* algorithm. Logic simulation is performed only through conducting transistors. The worst case time complexity of **Implication** is linear with the number of transistors in the circuit. In general the complexity will be much less than linear time since the depth-first search will only search through conducting transistors.

It is interesting to note that **Implication** only assigns "strong" logic values to circuit nodes, i.e., when a logic value 1 (0) is assigned to a node v, v must be connected to a 1-node (0-node) through conducting transistors, and the gate terminal of each of these conducting transistors must in turn be connected to a power source or a primary input. This property has some impacts on test generation as is described next.

First consider the example shown in Figure 6.4(a) where A is assigned a logic value 1 and $B = X$. Since either $B = 1$ or $B = 0$ will make $C = 1$, apparently the assignment $A = 1$ should imply $C = 1$. **Implication** cannot identify this situation since both inputs to the gate driving C are X. This problem, however, does not affect the generation of a test. Assume in Figure 6.4(a) a test for C stuck-at 0 is to be generated. Though a test generator employing **Implication** cannot identify $A = 1$ as a test vector, it will either identify $AB = 10$ or $AB = 11$ as a test vector. Since the goal of test generation is to identify only one test vector for a given fault, either $AB = 10$ or $AB = 11$ is as good as $A = 1$.

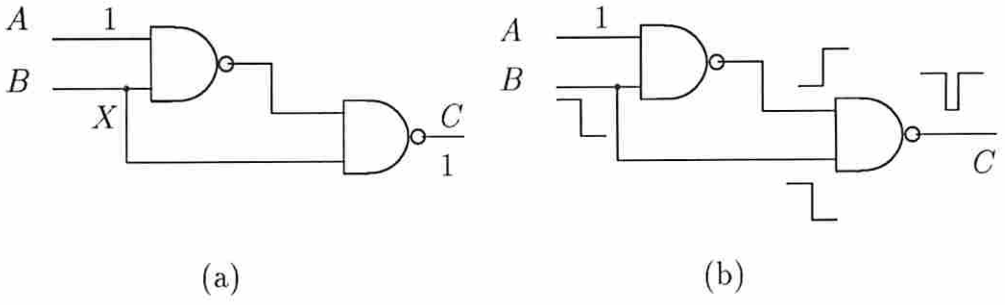


Figure 6.4: (a) $A = 1$ implies $C = 1$ (b) hazard condition

Now consider the case where a two pattern test $< T_1, T_2 >$ is to be used. If $T_1 = AB = 11$ and $T_2 = AB = 10$, then there may exist a static 1-hazard at

C during the transition as shown in Figure 6.4(b). Recall that in Section 5.2.2 we have presented a test generation procedure which generates T_d first and then T_1 and T_2 based on T_d. If when T_d is generated, a simulation algorithm which identifies line C as 1 given $AB = 1X$ is used, then the hazard condition at C will not be identified when generating T_1 and T_2. Thus the resulting test is not necessarily "robust."

With the above observation in mind, next the necessary and sufficient condition given in Theorem 12 will be proved. The proof requires the following definitions and Lemmas.

Lemma 12 *If a circuit node is assigned a logic value 1 (0) using* **Implication** *then it must be connected to a 1-node (0-node) through a path of conducting transistors, and the gate terminal of each of these transistors is assigned a logic value by* **Implication** *which turns on the transistor.*

Proof: By the definition of **Implication.** □

Definition 15 (Cover): *A test vector T_a covers another vector T_b if the following conditions are satisfied, where i indicates the i-th bit in a test vector and each bit of a vector is either 0, 1 or X.*

1. *If $T_b(i) = 1$ then $T_a(i) = 1$.*
2. *If $T_b(i) = 0$ then $T_a(i) = 0$.*

Definition 16 (Hold on/off): *A transistor t is held on (off) by a test vector T if when T is applied the gate terminal of t is assigned a logic value that turns t on (off) using the* **Implication** *algorithm.*

Lemma 13 *T_1 holds on (off) a P-type transistor t with gate terminal g if and only if there exists a path of transistors from some 0-node (1-node) to g such that each transistor on this path is held on by T_1. A similar lemma holds for N-type transistors.*

Proof: By induction and the definition of hold on (off). □

Lemma 14 *If $T_d = T_1 \wedge T_2$ as defined in Section 5.2.2, then both T_1 and T_2 cover T_d.*

Proof: By definitions. □

Lemma 15 *If T_a covers T_b, and T_b holds off (on) a transistor t in a circuit, then T_a also holds t off (on).*

Proof: If T_b holds t off (on), then the status of t cannot be changed by adding more PI assignments to the don't care bits of T_b. □

Lemma 16 *If both T_1 and T_2 hold off (on) a transistor t but $T_d = T_1 \wedge T_2$ does not hold t off (on), then there exists a potential static hazard at the gate terminal of t during the transition from T_1 to T_2.*

Proof: Without loss of generality assume t is a P-type transistor. Let the gate terminal of t be g. Since T_d cannot hold t off (on), there exists no path from any 1-node (0-node) to g such that each transistor on this path is held on by T_d. If there exists no path from any 1-node (0-node) to g such that all transistors on this path are held on by both T_1 and T_2, then T_1 and T_2 must hold off (on) t through different paths. Therefore during transition from T_1 to T_2, it is possible that no conducting path exists from any 1-node (0-node) to g. This implies that a conducting path from some 0-node (1-node) to g is possible. Thus a potential static hazard exists at g.

Now assume there exists a path P from some 1-node (0-node) to g such that all transistors on P are held on by both T_1 and T_2. T_d cannot hold on all transistors on P otherwise T_d will also hold off (on) transistor t. Therefore by induction at least one transistor on P will have a potential static hazard at its gate terminal during the transition from T_1 to T_2. This implies that it is possible that P can be temporarily turned off during the transition, thereby resulting in a static hazard at the gate terminal of t. □

Now we are ready to prove Theorem 12 in Section 5.2.2 which is rephrased below with node t being replaced by node g since t is used as a transistor in this section.

Theorem 13 *Let $< T_1, T_2 >$ be a test for a transistor t_f stuck-open when no circuit delay is considered. Let g be the output node of the transistor group containing t_f and let s be the power source that is connected to g in a good circuit when T_2 is applied.*

If $T_d = T_1 \wedge T_2$, then "$< T_1, T_2 >$ is a valid test for t_f under any circuit delay" if and only if "there exists a cutset C that separates s and t, C contains the faulty transistor t_f, and T_d holds off all transistors in $C - \{t_f\}$."

Proof: If there exists such a cutset C then obviously the test cannot be invalidated because any transient vector from T_1 to T_2 must cover T_d thereby holding off all transistors in $C - \{t_f\}$. If such a C does not exist, then there must exist a path from s to g such that this path does not contain t_f and each transistor is not held off by T_d. Thus during the transition from T_1 to T_2 each transistor on this path is either turned on or has a hazard (Lemma 16). Thus the test may be invalidated. □

6.4 Backtracking

Backtracking in the gate level PODEM algorithm is simply a forward implication starting with the primary input whose value is just changed. This is feasible because at the gate level each circuit line (or node) is completely specified by the inputs to the gate that drives the line (or node). Figure 6.5 shows this effect where the primary input A is to be changed from 0 to 1. The figure shows that by simply reevaluating each gate that is affected, the effects of a PI assignment change can be easily propagated throughout the circuit. It is not necessary to keep track of the information about which PI assignment causes the status change of any internal line (or node) of a circuit.

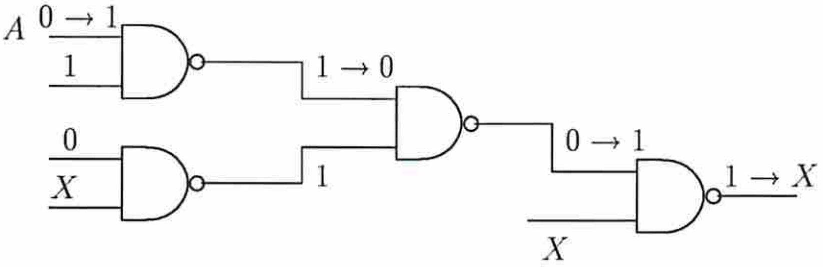


Figure 6.5: Backtracking at gate level can be accomplished by logic simulation

Next consider the above situation at the switch level and the circuit shown in Figure 6.6. In Figure 6.6(a) A is assigned 0. This results in x being assigned a value of 1. Assume after some steps of test generation, B is assigned a value of 0 as shown in Figure 6.6(b), which results in the assignment of value 1 to y. Assume later it is found that $AB = 00$ cannot lead to a test, then backtracking must be performed and a value of 1 must be assigned to B as shown in Figure 6.6(c). This will turn transistor T_B back to off. If the information that x drives y to 1 due to $B = 0$ was not recorded, then unless a new simulation process on the entire transistor group that contains x and y is done, it is impossible to go back to the status before $B = 0$ was assigned.

Since in general a switch level simulation through an entire transistor group is much more time-consuming than a gate level simulation, in this study the information about which PI assignment makes the status change of each node is recorded so that when a backtracking occurs the previous status of each affected node can be restored immediately. The additional memory for this method is a pointer for each node. The value change information can be recorded using the bit-pattern property of C language. Thus no other additional memory is required since many unused bits exist for each node.

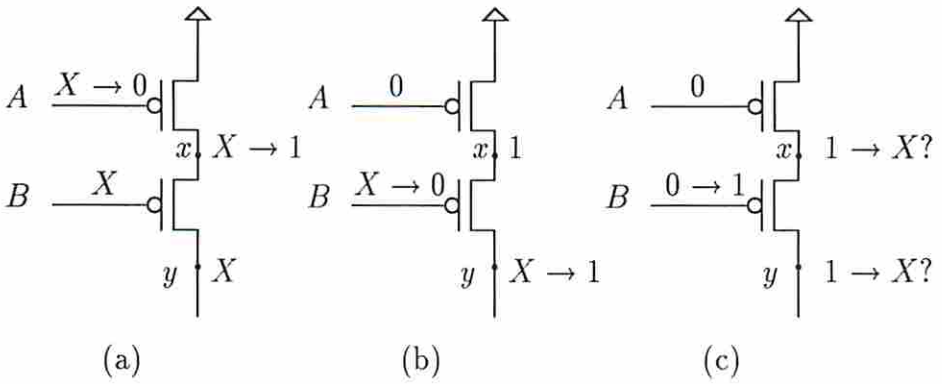


Figure 6.6: Difficulty in switch level backtracking

6.5 Fault Propagation

When dealing with bridging fault using CSM, it is not necessary to propagate the fault effect to some primary outputs. However for stuck-open or stuck-at faults, fault propagation is necessary.

Fault propagation at the switch level can be divided into two stages. The first one is to propagate the fault effect from the fault site to the output of the faulty transistor group, and the second one is to propagate fault effect from the output of the faulty transistor group to the output of another transistor group and then to the output of yet another transistor group, such that eventually the fault effect can be observed at some primary output.

The propagation of fault effects to the output g of the faulty transistor group can be done by simply setting objectives that will give g different logic values in the fault-free and the faulty circuits, and then try to backtrace and assign PI values to achieve these objectives. Similarly, to propagate a fault effect from the input to the output of a transistor group, simply try to find some PI assignments such that the output will have different logic values in the fault-free and faulty circuit.

The fault propagation mechanism used in this study is similar to that used in the PODEM algorithm. To determine a transistor group through which to propagate an error, the level information, i.e., the distance between a transistor group and its closest primary output, is used. The objective is to select transistor groups in such a way as to reach a primary output in a minimal number of steps. In the PODEM algorithm both the fault-free and faulty circuits are processed at the same time using the D notation [16, 32], while in our switch level test generator, the fault-free and faulty circuits are processed separated. Thus the objective of setting the output of a transistor group to different values in the fault-free and faulty circuits is actually two objectives, one for each circuit. During fault propagation, if a node has different values in the fault-free and faulty circuits (including 1 and X, and 0 and X), then it is considered to be in the D-frontier [32].

When one primary input is assigned a new value, the implication must be performed on the fault-free and faulty circuits separately, thereby apparently requiring twice the computation time compared with the original PODEM algorithm. However at the switch level if a node has complementary logic values in the fault-free and faulty circuits, the node turns on different types of transistors. Refer to Figure 6.7(a) where the fault effect is to be propagated from u to v. Assume u has values 1 and 0 in fault-free and faulty circuits, respectively. In the fault-free circuit only the N-network is processed using **Implication** since $u = 1$ cannot turn on any transistor in the P-network. Similarly in the faulty circuit only the P-network is processed as $u = 0$.

Now consider the method of processing the fault-free and faulty circuits at the same time. To simplify the analysis consider Figure 6.7(b). The value of v will become $1/x$ due to the P-type transistor and $x/0$ due to the N-type transistor. By using the original PODEM algorithm where 5-valued logic $(0, 1, X, D, \overline{D})$ is employed, both $1/x$ and $x/0$ are considered as X. Thus fault propagation fails. This problem can be overcome by using 9-valued logic [33]. However using 9-valued logic the P-network and N-network must both be processed at the switch level, thus computation time is not saved. It may even be more time-consuming since it requires operations on 9-valued logic compared to three-valued logic when the fault-free and faulty circuits are processed separately.

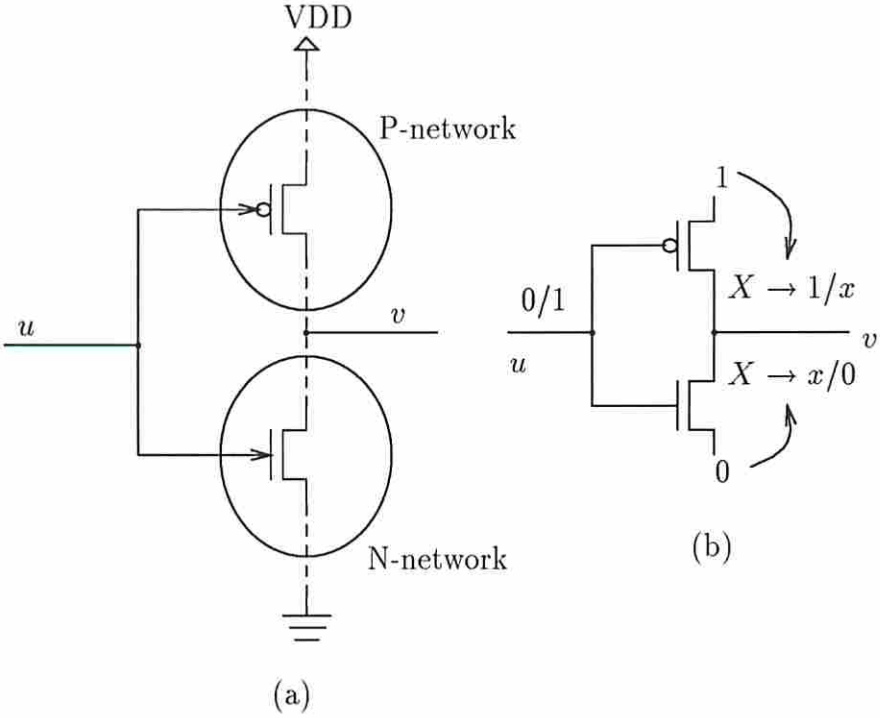


Figure 6.7: Comparison of processing fault-free and faulty circuits separately and simultaneously

Another advantage of separating the fault-free and faulty circuits is illustrated in Figure 6.8, where the fault effect at u is to be propagated to v. If both the fault-free and faulty circuits are processed at the same time using the D notation, the input $1X1$ does not propagate the fault effect to v, as shown in Figure 6.8(a). By separating the fault-free and faulty circuits and using **Implication**, Figure 6.8(b) and (c) show that v will have different logic values in the fault-free and faulty circuits and thus the fault effect has been propagated from u to v.

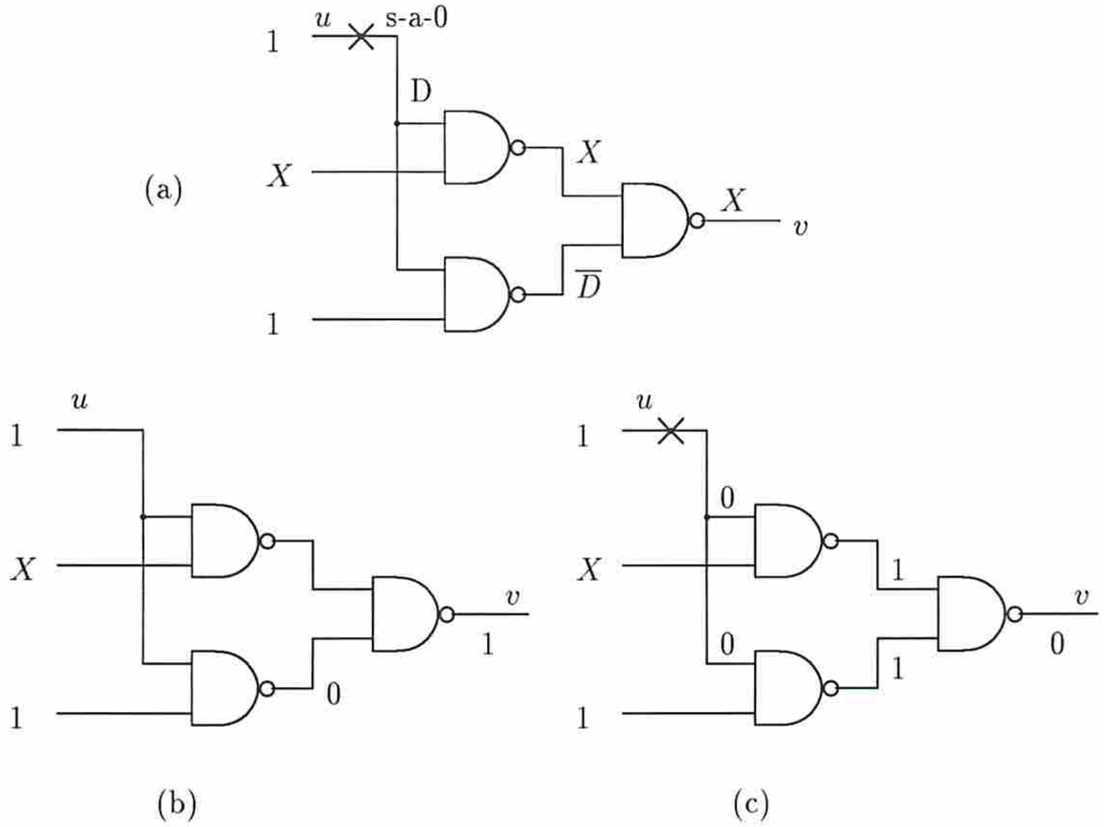


Figure 6.8: Advantage of separating fault-free and faulty circuits

Chapter 7

Scan-Based Sequential Circuit Testing

So far the emphasis in this dissertation has been on test generation for combinational circuits. Generating tests for sequential circuits is a harder problem. Sequential circuit testing can be simplified by employing a scan-based design [30], such as LSSD [117]. One problem with this technique is that defects may also occur within the scan registers. Thus it is essential to detect faults in scan registers to guarantee the proper functionality of a scan-based system. In this chapter a systematic method for analyzing all possible faults within the scan path of a scan-based CMOS circuit is given. The analysis shows that both logic and current monitoring are necessary in order to detect all irredundant faults. A universal test sequence is derived based on the analysis of single bridging faults. This sequence also detects all irredundant stuck-at and stuck-open faults.

7.1 CMOS Scan Registers & Test Sequences

Scan registers can be implemented in several ways [1]. The discussion in this chapter is based on the implementation shown in Figure 7.1. The analysis can easily be applied to other types of scan registers.

Figure 7.1(a) and (b) are the switch- and gate-level descriptions of a *scan cell* which consists of two inverters and two pass transistors. Figure 7.1(c) shows

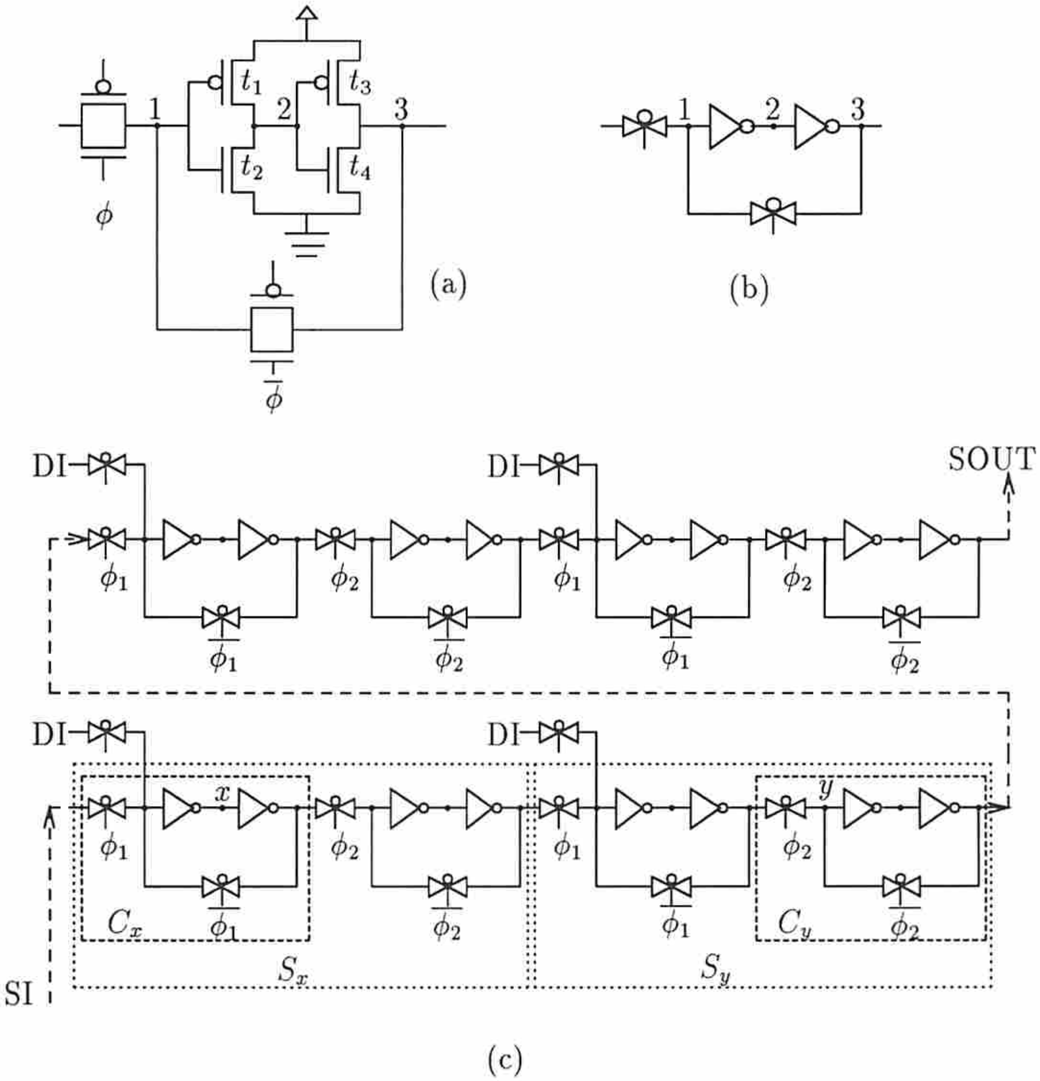


Figure 7.1: A scan path, storage elements and scan cells

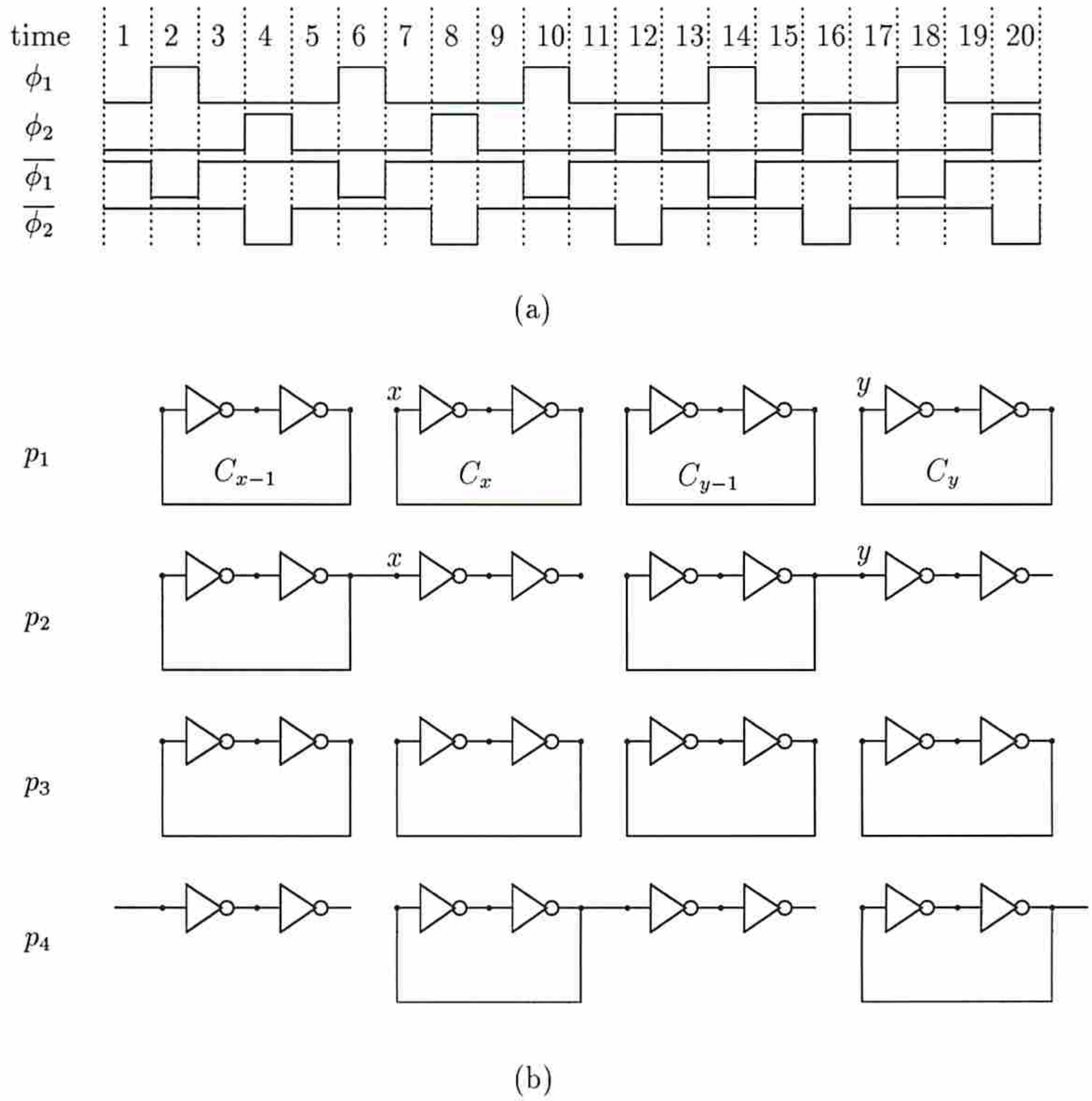
the connection between cells. A *storage element* consists of two scan cells. The notation C_a (S_a) denotes a scan cell (storage element) that contains node a.

In this chapter only BFs between nodes within the scan path are considered. Also BFs which involve *VDD*, *GND* or any clock signal are assumed to be easily detected and will not be considered. Thus in each scan cell three nodes (nodes 1, 2 and 3 in Figure 7.1(a) and (b)) may be shorted (bridged) to other nodes. A scan cell C_a is said to have a value v, denoted by $C_a = v$, if the value of its node 1 is v. For example $C_x = 0$ implies that the first, second and third nodes of scan cell C_x have values 0, 1 and 0, respectively.

Two-phase clocking (ϕ_1 and ϕ_2) on the scan path is assumed. Each clock cycle consists of 4 subcycles: $p_1 = (\phi_1 = 0, \phi_2 = 0)$, $p_2 = (\phi_1 = 1, \phi_2 = 0)$, $p_3 = (\phi_1 = 0, \phi_2 = 0)$ and $p_4 = (\phi_1 = 0, \phi_2 = 1)$ as shown in Figure 7.2(a). Figure 7.2(b) shows the circuit configuration of the scan path during each subcycle. A scan cell always forms a feedback loop during p_1 and p_3. It also forms a loop either during p_2 or p_4, but not both. For example in Figure 7.2(b), C_x forms a loop during p_1, p_3 and p_4. The notations C_{a-1} and C_{a+1} denote the scan cells which immediately precede and follow C_a, respectively. During p_2, x is actually in the feedback loop of cell C_{x-1}.

For each node a, two functions, $n(a)$, $l(a)$, are defined. $n(a)$ is the number of scan cells between the scan input and the scan cell containing node a; $l(a)$ is the location of a (1, 2 or 3) in the scan cell. Without loss of generality it is assumed that a bridging fault occurs between nodes x and y, where x is closer to the scan input (SI) than y is. A *distance function* D between x and y is defined as $D(x, y) = n(y) - n(x)$. For example in Figure 7.1, $l(x) = 2$, $l(y) = 1$ and $D(x, y) = 3$.

A BF on the scan path is detected by scanning a test sequence through the scan path and observing the scanned-out data or monitoring the current supply. For CSM, the fault is detected when the appropriate part of the test sequence is scanned through the fault site. It is not necessary to scan out the test data. For some BF which cannot be detected by CSM, it is still possible to detect the



(a)

(b)

Figure 7.2: Timing of the scan path

fault without fully scanning out the test data. The notation $ud(k)v$ will be used to denote a test sequence, where v and u are the first and last bit to be scanned in, and $d(k)$ is a sequence of k *don't care* bits, $k \geq 0$. A bit is underlined if it must be scanned out and observed. For example, $1d(3)\underline{0}$ denotes a sequence of bits $1 \times \times \times 0$ and only the first bit (0) need be scanned out and observed. The complement value of v is denoted by $\bar{v}$.

7.2 Effects of BFs

This section analyzes the effects of BFs. The necessity of monitoring currents and observing scan data in detecting BFs is illustrated. Some fault effects which have not been treated in the literature are discussed.

The necessity of CSM is obvious. A BF may result in a circuit oscillation where the logic value of each shorted node cannot be determined. The only way to guarantee the detection of this fault is by using CSM. The necessity of observing scan data needs some justification. Consider a BF between node $1(x)$ of C_x and node $1(y)$ of C_y in Figure 7.2. Assume during p_1, $C_{x-1} = 1$ and $C_{y-1} = 0$. When entering p_2, the BF results in a *clash* between the feedback loops in C_{x-1} and C_{y-1}. No external control for these two loops exist during p_2. Thus both loops (cells) may reach a steady state value where either both are logic high or logic low. There will be no large current consumption during steady state. To detect such a fault the test data must be scanned out and observed.

The above clash situation may occur when two shorted loops become *isolated* from their input lines due to the clock setting. The final value on both loops depends on circuit parameters (e.g., the physical size of transistors). Four possible results exist when two cells C_x and C_y clash: 1) cell C_x dominates, 2) cell C_y dominates, 3) logic high dominates, and 4) logic low dominates. It is assumed that the result of a clash for a particular BF is time-invariant. For example, if C_x dominates C_y in a clash due to a BF f_1 at a certain time, then C_x always dominates C_y in any clash between C_x and C_y due to f_1. However for a different

BF between C_x and C_y (there are 9 different BFs between C_x and C_y), say f_2, the dominance relation may be different, e.g., C_y may dominate C_x for f_2. For two different clashes their dominance results are assumed to be independent.

Now consider the detection of BFs that may result in clashes. The BF between x and y in Figure 7.2 can be detected by a test sequence containing a transition from 0 to 1 or 1 to 0. However, not all BFs are so easy to detect. Consider a fault between x and y such that $D(x, y) = 8$ and $l(x) = l(y) = 1$. There are 3 storage elements between x and y. Thus unless a test sequence containing $1\times\times\times0$ or $0\times\times\times1$ is used, the fault cannot be detected. The test sequence 010 and 01100 proposed in [101] cannot detect such a fault. One may think the problem is still easy and the only requirement is to scan two different values to two shorted nodes such that the clash results in a value change at one node. By scanning out the changed value, the fault can be detected. Unfortunately, this is not always true due to two reasons. First, to have a clash two complementary values must be scanned to the two shorted nodes. However before arriving at the shorted nodes, the test data may have been changed. Secondly, even if this data successfully arrives at the shorted nodes and a clash occurs, the test data still must be scanned out and observed. During the scan out process the test data may again be modified to its original value and thus the fault effect is masked. Consider BF F14 in Figure 7.3. A detailed description of faults in Figure 7.3 will be given later. For now only F14 is considered. Assume a 1 and a 0 have been successfully scanned to the second node of C_y and the third node of C_{x-1}, respectively. There is a clash between these two cells during the next p_2. Assume logic high dominates in this clash such that the value at x is changed to 1. To observe this effect, this value must be scanned out. But during the scan out process this 1 must pass through C_y and sets y to 0. If at this time the value of x happens to be 1, then the clash between C_y and C_{x-1} occurs again. Since 1 dominates this clash, the value of y will be changed to 1 and the fault effect is masked.

An even more complex case is F13, where two clashes between two different pairs of cells (C_{x-1} and C_y during p_2, and C_x and C_{y-1} during p_4) can occur. It is clear that a systematic analysis is necessary to guarantee the detection of all faults.

7.3 Detailed Examination of BFs

In this section a systematic method for analyzing all possible BFs is presented. All BFs are classified into 4 major categories according to the value of $D(x, y)$. Under each category, each fault is further classified using the values of $l(x)$ and $l(y)$. It is shown that only one type of fault may be redundant ($D(x, y) = 0$, $(l(x), l(y)) = (1, 3)$). Without loss of generality, in the following discussion the pass transistors in C_x are controlled by ϕ_1 and $\overline{\phi_1}$. The results can be applied to the case where the pass transistors in C_x are controlled by ϕ_2 and $\overline{\phi_2}$ by interchanging p_1 with p_3, and p_2 with p_4. All discussions refer to Figure 7.3.

7.3.1 $D(x, y) = 0$ — BFs in the Same Scan Cell

F1: $(l(x), l(y)) = (1, 2)$ or $(2, 3)$. This fault causes a short between the input and output of an inverter and thus is detected by CSM.

F2: $l(x) = 1$, $l(y) = 3$. This fault forces C_y to have a permanent loop. Thus if C_{x-1} and C_y had different values at p_1, then a clash between them occurs during p_2. If C_y dominates this clash, a sequence $b_2b_1 = \underline{v\bar{v}}$ ($\bar{v}$ is the first bit to be scanned in) can detect this fault as described next. At some p_1, b_1 is in C_y and C_{y+1}, and b_2 is in C_{x-1}. If b_1 has been changed to v at this time due to the BF, then C_{y+1} must have the same value v and this value cannot be further changed during the scan out process. Thus the fault can be detected by observing b_1. If b_1 is not changed, then since b_2 cannot be changed before arriving at C_{x-1}, there must be a clash between C_{x-1} and C_y during p_2. Since C_y dominates the clash, the value in C_y cannot be changed to v as in the fault-free circuit. Thus during p_3, C_y still has a value of $\bar{v}$ and hence the value of b_2 is not propagated through C_y. This fault effect cannot be masked during further scan operation and the

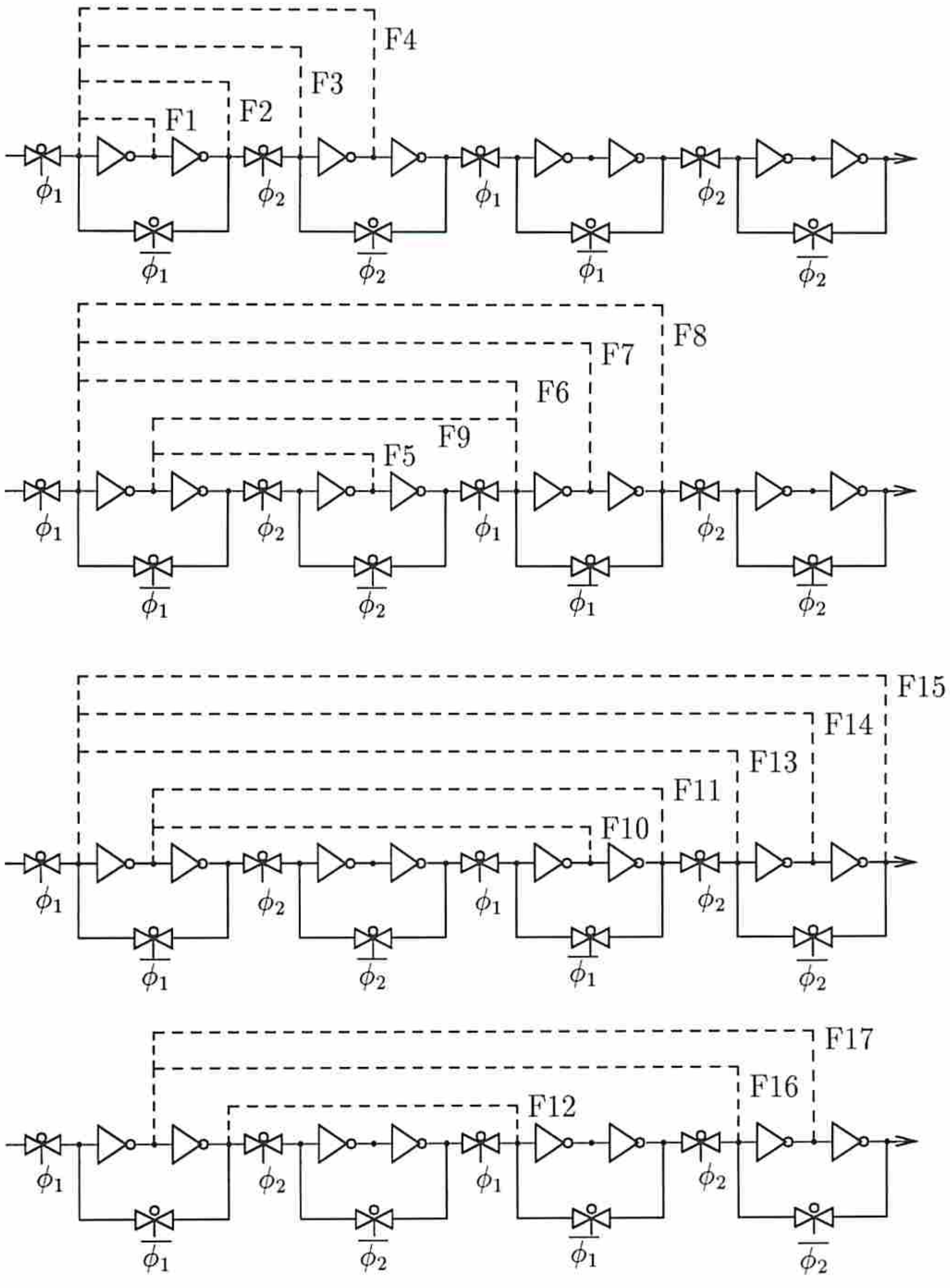


Figure 7.3: Bridging faults on a scan path

fault will be detected by observing b_2. Thus $b_2b_1 = \underline{v\bar{v}}$ can detect this fault if C_y always dominates the clash. If logic high (1) always dominates the clash, then the sequence $b_2b_1 = 1\underline{0}$ will detect this fault because b_2 cannot be changed before arriving at C_{x-1}, and B_1 will be changed to 1 if a clash occurs during some p_2. This 1 cannot be further changed and thus the fault can be detected by observing b_1. Similarly if 0 always dominates the clash, then $b_2b_1 = 0\underline{1}$ can detect the fault. Finally if C_{x-1} always dominates the clash, then during p_2 the value of C_{x-1} is propagated to C_y in both the faulty and fault-free circuits. Thus this fault is redundant. Obviously this fault cannot affect the logic function of the scan path. As will become clear later, this is the only redundant fault among all possible BFs. In summary this fault is either redundant or is detected by a test sequence that contains $\underline{10}$ and $\underline{01}$.

7.3.2 $D(x, y) = 1$ — BFs Between Two Adjacent Scan Cells

F3: $(l(x), l(y)) = (1, 1)$ or $(1, 3)$. A clash between C_{x-1} and C_y occurs during p_2 if such a fault exists. If C_y dominates the clash, a sequence $b_2b_1 = \underline{v\bar{v}}$ can detect this fault as described next. If the value of b_1 becomes v when b_1 is in C_y and b_2 is in C_x and C_{x-1}, b_1 cannot be further changed during the scan out operation. If b_1 is not changed, b_2 will be changed to $\bar{v}$ by the clash and cannot be further changed thereafter. If C_{x-1} dominates the clash, then $b_2b_1 = v\underline{\bar{v}}$ can detect the fault since the value of b_1 will be changed to v during the clash and this fault effect cannot be masked. If 1(0) always dominates, then $b_2b_1 = 1\underline{0}(0\underline{1})$ detects this fault since the 0(1) bit will be changed to 1(0) and can be observed. In summary, a sequence containing $\underline{01}$ and $\underline{10}$ detects this fault.

F4: $(l(x), l(y)) = (1, 2)$, $(3, 2)$, $(2, 1)$ or $(2, 3)$. These faults are detected by CSM during p_4.

F5: $(l(x), l(y)) = (2,2)$, $(3,1)$ or $(3,3)$: These faults are detected by $b_2b_1 = v\underline{\bar{v}}$ since the value of b_1 will be changed to v during some p_2 and this change cannot be masked thereafter.

7.3.3 $D(x,y) = 2k$, $k \geq 1$

F6: $l(x) = l(y) = 1$. A clash occurs between C_{x-1} and C_{y-1} during p_2 if these cells had different values during p_1. If C_{y-1} dominates the clash, this fault can be detected by the sequence $b_{k+1}d(k-1)b_1 = \underline{v}d(k-1)\underline{\bar{v}}$ since either b_1 is changed to v when it passes through C_{x-1}, or b_{k+1} is changed to $\bar{v}$ if b_1 is not changed. If C_{x-1} dominates, then $b_{k+1}d(k-1)b_1 = vd(k-1)\underline{\bar{v}}$ can detect the fault since b_1 will be changed to v when passing through C_{y-1}. If $1(0)$ dominates, then $1d(k-1)\underline{0}$ $(0d(k-1)\underline{1})$ detects this fault. In summary a sequence containing $\underline{1}d(k-1)\underline{0}$ and $\underline{0}d(k-1)\underline{1}$ detects this fault.

F7: $l(x) = 1$, $l(y) = 2$. If this fault is present, the value of x is dominated by the value of y during p_2, i.e., node x takes the value of node y, since y is controlled by an inverter which is not in a feedback loop. Thus the sequence $b_{k+1}d(k-1)b_1 = \underline{v}d(k-1)\underline{v}$ detects this fault since either b_1 is changed to $\bar{v}$ before arriving at C_y, or b_1 is not changed but b_{k+1} is changed to $\bar{v}$.

F8: $l(x) = 1$, $l(y) = 3$. Similar to **F7**, the test sequence must contain $\underline{v}d(k-1)\underline{\bar{v}}$.

F9: $l(x) = 2$, $l(y) = 1$. During p_2 the value of y is changed to that of x. Thus $b_{k+1}d(k-1)b_1 = vd(k-1)\underline{v}$ detects this fault.

F10: $(l(x), l(y)) = (2,2)$ or $(3,3)$. During p_2, x and y are both driven by an inverter which is not in a feedback loop. Thus by scanning in $b_{k+1}d(k-1)b_1 = vd(k-1)\underline{\bar{v}}$ the fault can be detected by CSM if b_1 is not changed when it arrives at C_{y-1}. If b_1 is changed, then it cannot be further changed and thus the fault effect can be observed at the scan output.

F11: $(l(x), l(y)) = (2,3)$ or $(3,2)$. Similar to **F10**; the test sequence must contain $b_{k+1}d(k-1)b_1 = vd(k-1)\underline{v}$.

F12: $l(x) = 3,\ l(y) = 1$. Similar to **F9**; the sequence $b_{k+1}d(k-1)b_1 = vd(k-1)\underline{\bar{v}}$ detects this fault.

7.3.4 $D(x, y) = 2k + 1,\ k \geq 1$

F13: $l(x) = l(y) = 1$. Two possible clashes can occur when this fault is present: the clash between C_{x-1} and C_y during p_2, and the clash between C_x and C_{y-1} during p_4. The former is referred to as *Clash 1*, and the latter as *Clash 2*. If C_{x-1} dominates *Clash 1*, then a sequence $b_{k+2}d(k)b_1 = vd(k)\underline{\bar{v}}$ can detect the fault no matter what the result of *Clash 2* is, since the value of b_1 will become v after passing through C_y. If 1 (0) dominates *Clash 1*, then $1d(k)\underline{0}$ ($0d(k)\underline{1}$) detects the fault as the first bit will be changed to a 1(0). If C_y dominates *Clash 1*, then depending on the result of *Clash 2*, four subcases exist. If C_x dominates *Clash 2*, then the sequence $b_2b_1 = \underline{v\bar{v}}$ can detect this fault as explained next. During some p_1, b_1 is in C_y and C_{y-1}, and b_2 is in C_{y-2}. If the value of b_1 has been changed to v at this time, then this value cannot be further changed (because C_y dominates *Clash 1*) and the fault can be detected by observing b_1. Thus assume b_1 is not changed. When entering p_2, *Clash 1* occurs and the value of C_{x-1} will be changed to $\bar{v}$ no matter what its previous value is since C_y dominates this clash. Also the value of C_x will become $\bar{v}$ since it is controlled by C_{x-1} during p_2. The value of b_2 will also arrive at C_{y-1} during p_2. When entering the next p_4, *Clash 2* occurs between C_x and C_{y-1}. Since C_x dominates this clash, the value of b_2 will become $\bar{v}$. This value cannot be further changed since C_y dominates *Clash 1*, which is the only way that b_2 can be changed again. Thus if C_x dominates *Clash 2*, a sequence $\underline{v\bar{v}}$ can detect this fault. Now consider the case where C_{y-1} dominates *Clash 2*. This fault can be detected by a sequence $b_{k+1}d(k-1)b_1 = \underline{v}d(k-1)\underline{\bar{v}}$. During some p_3, b_1 is in C_{y-1} and C_{y-2}, and b_{k+1} is in C_x. If b_1 has been changed to v, then b_1 cannot be further changed. Thus the fault can be detected by observing b_1. If b_1 has not been changed, then during the next p_4, *Clash 2* occurs and b_{k+1} will have a value of $\bar{v}$. This value cannot be further changed and thus the fault is detected. If 1(0) always dominates *Clash 2*, then $b_2b_1 = \underline{01}$ ($\underline{10}$) can detect this fault since either b_1 or b_2 will change its value and the fault effect can propagate to scan output. In summary this fault is detected by a test sequence containing $1d(k)\underline{0}$, $0d(k)\underline{1}$, $\underline{v}d(k-1)\underline{\bar{v}}$, $\underline{01}$ and $\underline{10}$.

F14: $l(x) = 1$, $l(y) = 2$. A clash between C_{x-1} and C_y occurs during p_2. If C_y dominates, then the fault is detected by a sequence $b_{k+1}d(k-1)b_1 = \underline{v}d(k-1)\underline{v}$ as explained next. During some p_3, b_1 is in C_{y-1} and C_{y-2}, and b_{k+1} is in C_x and C_{x-1}. If b_1 has been changed, then it cannot be further changed and the fault is detectable by scanning out and observing b_1. If b_1 is not changed, then during p_4, C_{x+1} and C_x will be dominated by C_y and b_{k+1} will have a value of $\bar{v}$. This value cannot be further changed and thus the fault is detected by observing b_{k+1}. If C_{x-1} dominates the clash, then a sequence $b_{k+2}d(k)b_1 = vd(k)\underline{v}$ can detect this fault since during some p_2, C_{x-1} will dominate C_y and C_{y+1}. Thus the value of b_{k+2} will dominate the value of b_1 such that C_{y+1} becomes $\bar{v}$ and the fault can be detected. If $1(0)$ dominates the clash, then it can be shown that a sequence $1d(k)\underline{1}$ ($0d(k)\underline{0}$) detects this fault using similar arguments. In summary a test sequence containing $\underline{v}d(k-1)\underline{v}$, $1d(k)\underline{1}$ and $0d(k)\underline{0}$ detects this fault.

F15: $l(x) = 1$, $l(y) = 3$. Similar to **F14**; a sequence containing $\underline{v}d(k-1)\underline{\bar{v}}$, $1d(k)\underline{0}$ and $0d(k)\underline{1}$ detects this fault.

F16: $(l(x), l(y)) = (2,1)$, $(2,3)$ or $(3,2)$. These faults are detected by a sequence $b_{k+2}d(k)b_1 = vd(k)\underline{v}$ since during some p_2, C_y is controlled by an inverter of C_x which in turn is controlled by C_{x-1}. Thus the value of b_1 will be changed to $\bar{v}$ and is observable at the scan output.

F17: $(l(x), l(y)) = (2,2)$, $(3,1)$ or $(3,3)$. Similar to **F16**; the sequence $vd(k)\underline{\bar{v}}$ detects this fault.

7.4 A Universal Test Sequence for All Faults

The results obtained in the previous section are summarized in Table 7.1. If a test sequence contains all sequences shown, then it must detect all irredundant

BFs. Using the notation $1^n(0^n)$ to represent a sequence of n 1s (0s), the following results can be obtained.

Fault	$D(x,y)$	$(l(x),l(y))$	Test method/sequence
F1	0	(1,2) or (2,3)	CSM
F2	0	(1,3)	Redun. or detected by $\underline{10}$ and $\underline{01}$
F3	1	(1,1) or (1,3)	$\underline{01}$ and $\underline{10}$
F4	1	(1,2),(3,2),(2,1) or (2,3)	CSM
F5	1	(2,2),(3,1) or (3,3)	$v\underline{\bar{v}}$
F6	$2k$	(1,1)	$\underline{0}d(k-1)\underline{1}$ and $\underline{1}d(k-1)\underline{0}$
F7	$2k$	(1,2)	$\underline{v}d(k-1)\underline{v}$
F8	$2k$	(1,3)	$\underline{v}d(k-1)\underline{\bar{v}}$
F9	$2k$	(2,1)	$vd(k-1)\underline{v}$
F10	$2k$	(2,2) or (3,3)	$vd(k-1)\underline{\bar{v}}$ and CSM
F11	$2k$	(2,3) or (3,2)	$vd(k-1)\underline{v}$ and CSM
F12	$2k$	(3,1)	$vd(k-1)\underline{\bar{v}}$
F13	$2k+1$	(1,1)	$1d(k)\underline{0}$, $0d(k)\underline{1}$, $\underline{v}d(k-1)\underline{\bar{v}}$, $\underline{01}$, $\underline{10}$
F14	$2k+1$	(1,2)	$\underline{v}d(k-1)\underline{v}$, $1d(k)\underline{1}$ and $0d(k)\underline{0}$
F15	$2k+1$	(1,3)	$\underline{v}d(k-1)\underline{\bar{v}}$, $1d(k)\underline{0}$ and $0d(k)\underline{1}$
F16	$2k+1$	(2,1),(2,3) or (3,2)	$vd(k)\underline{v}$
F17	$2k+1$	(2,2),(3,1) or (3,3)	$vd(k)\underline{\bar{v}}$

Table 7.1: Summary of tests for each BF

Theorem 14 *The test sequence $TS = 0^n\underline{01^{n+1}0}$ detects all irredundant BFs in a scan path consisting of n storage elements.*

Proof: The proof follows by verifying that all sequences in Table 7.1 are subsequences of TS. □

Note that it is not necessary to scan out all test data in TS to detect all irredundant faults. Only the first $n+3$ bits need to be scanned out and observed. The last n bits can serve as reset data (0) for the scan path. Furthermore, TS is very easy to generate. The external test controller need only store one piece of

Fault	$(l(x), l(y))$	Subcycle	Oscillation	Current value
F1	(1,2)	p_2	no	8.1×10^{-5}
F1	(2,3)	p_2	no	8.1×10^{-5}
F4	(1,2)	p_4	no	1.3×10^{-4}
F4	(3,2)	p_4	no	9.1×10^{-5}
F4	(2,1)	p_4	no	8.1×10^{-5}
F4	(2,3)	p_4	yes	$8.0 \times 10^{-5} \sim 1.65 \times 10^{-4}$
F10	(2,2)	p_2	no	1.8×10^{-4}
F10	(3,3)	p_2	no	1.5×10^{-4}
F11	(2,3)	p_2	no	1.6×10^{-4}
F11	(3,2)	p_2	no	1.6×10^{-4}

Table 7.2: Current values when CSM is used

data, i.e., n, in order to generate TS. Since n must be available because it is the length of the scan path, there is no storage overhead for generating TS.

Four types of faults are detected by CSM. The current values (in amperes) obtained by SPICE simulations for these faults are given in Table 7.2. The third column shows when the current is measured. The fourth column shows whether or not an oscillation occurs. Among all BFs, only one fault (F4, $(l(x), l(y)) = (2, 3)$) results in oscillation. The average current value for this fault is of the same order as for the other faults.

7.5 Other Faults

After deriving the test sequence TS for all possible BFs, all stuck-at and stuck-open faults are examined against TS. It is easy to verify that all stuck-at faults are also detected by TS because if a stuck-at fault occurs in a scan cell C_a, then the scanned data can never reach the scan cells following C_a.

The problem of detecting stuck-open faults is a little more complex. Refer to Figure 7.1(a). If a stuck-open fault occurs on a transistor that is part of a transmission gate, then the fault is redundant. If the fault occurs on transistor t_1 (t_2), then node 2 of the faulty cell will never be charged to 1, and thus the fault is equivalent to a stuck-at 0 (1) fault at node 2. Now consider the case where t_3 (t_4) is stuck-open. When $\phi = 1$, if node 1 is driven to 0 (1), then no fault effect can be observed. However if node 1 is driven to 1 (0), then this value cannot be propagated to node 3. When ϕ becomes 0 and $\overline{\phi}$ becomes 1, charge sharing occurs between node 1 and node 3 due to the transmission gate. If after charge sharing the voltage v of of both nodes 1 and 3 is between 1 and 4 V then both transistor t_1 and t_2 will conduct and the fault can be detected using CSM. If $v > 4V$ ($v < 1V$) then the fault is redundant since the information at node 1 can be propagated to node 3. If $v < 1V(v > 4V)$, then the value of node 3 is 0 (1) and the fault can be detected by scanning out the test data.

7.6 Summary

All possible bridging faults in the scan path of a CMOS scan-based design have been analyzed. Through an extensive analysis it is possible to derive a universal test sequence that guarantees 100% fault coverage for the scan path. The derived test sequence requires little overhead to generate. SPICE simulations show that CSM is an effective method for detecting some BFs. All stuck-at faults and irredundant stuck-open faults are also detected by the derived test sequence.

Chapter 8

Implementation & Experimental Results

This chapter describes the implementation issues of the SWiTEST system and presents experimental results. The working environment and the organization of SWiTEST are first described, followed by an I/O description of the system. Experimental results on 9 circuits including six ISCAS-85 benchmark circuits [26] are presented and discussed.

8.1 SWiTEST Test Generation System

8.1.1 Working Environment

SWiTEST is implemented in the C language on a SUN Sparc workstation running SunOS Release 4.1. The memory required for using this system is circuit dependent because most data structures that are circuit dependent are calculated and allocated in the initialization procedures using the dynamic allocation facility of the C language. By this fashion it is not necessary to declare fixed array sizes for most data structures. This has the following advantage.

In a circuit each node may connect or control a different number of transistors. If a fixed number is given, then if there exists one node that controls

30 transistors (this is a case in some benchmark circuits), at least 30 connections/controls must be allocated to each node. This is obvious a waste since most nodes have only a few connections/controls. On the other hand, if all memory were dynamically allocated only when required, then much CPU time would be spent in memory management since dynamic allocation is a computationally intensive process (e.g., garbage collection). Therefore figuring out how much memory is required and allocating the required memory at one time in the beginning of test generation improves both CPU time and memory usage.

Experiments show that for the largest ISCAS-85 benchmark circuit, namely `c7552`, which contains 15,396 transistors, the memory requirement is 2.556M bytes.

8.1.2 Organization

The block diagram of SWiTEST has been shown in Figure 3.1. The three parts of this system are described below.

Part 1: This part is a preprocessor which includes the following routines.

- A routine for partitioning the circuit into transistor groups.
- A routine for identifying complex and primitive gates.
- A routine for assigning levels to transistor groups.
- A routine for assigning directions to transistors.

Part 2: This part is the framework of the test generator. It includes the following routines.

- An objective selector.
- A backtracing routine.
- A switch-level simulator.

- A backtracking mechanism.
- A fault propagation routine.

Part 3: This part deals with the individual faults. It includes the following routines.

- Test generation procedures for all bridging faults and stuck-on faults.
- Test generation procedures for all stuck-open faults.
- Test generation procedures for all node stuck-at faults.

Most of the implementation of these routines follow the descriptions in Chapters 4, 5 and 6. One exception is that the routine for stuck-open faults does not check for possible invalidation due to charge sharing. From the analysis of Section 5.1, it is clear that if IDDQ testing is used the possibility of test invalidation due to charge sharing is very small. Thus ignoring charge sharing should have little effect on the quality of a test.

8.1.3 Use of SWiTEST

Two main programs, `prep` and `switest`, are implemented that include all routines described. `prep` is used to preprocess a circuit and create the information needed for test generation. The input to `prep` is a circuit file with a SPICE-like format. In addition to the flat SPICE description of a circuit, two additional cards `.INPUTS` and `.OUTPUTS` must be added to the input file to provide information on primary inputs and outputs.

`prep` generates two files: `*.lst` and `*.tg`. where "*" is the file name of the input circuit. A `*.lst` file contains the information about nodes and transistors in the circuit, which includes among other things the numbers of nodes and transistors, numbers of transistors connected to each node, the drain, source and gate nodes of each transistor, the signal flow direction of each transistor. A *.tg file contains the information about each transistor group and gate, and includes among other things a list of the transistors and nodes in a transistor group or gate, the level information of each group or gate, and the I/O relation among groups and gates.

After generating *.lst and *.tg files, switest can be invoked to generate test vectors for faults. The faults to be detected are provided in "fault files" with names "*.flt." Each fault file contains a list of faults each of which can be a stuck-open, bridging, stuck-on or stuck-at fault. The results of test generation are displayed on screen, which can be easily saved to a file using the "redirect" property of UNIX.

8.2 Experimental Results and Discussions

Nine circuits have been used for experiments. parity is a 4-bit parity generator and cadder is a combinational full adder. These two circuits are described in pages 334 and 312 of [1], respectively. c60 is a circuit that contains some redundant stuck-at, stuck-on and stuck-open faults. All others are ISCAS-85 benchmark circuits. Originally c60 and all benchmark circuits are described at the gate level. A translator called g2s has been implemented to convert a gate level description to a switch level description. The translation replaces each gate by a pair of N- and P-nets. It also extracts the required I/O information from the gate level description.

All transistor stuck-open and stuck-on faults in each circuit are considered. In the experiments all stuck-on faults are selected rather than all bridging faults because the number of all possible bridging faults is too large for a large circuit. Each stuck-on fault is actually considered as a bridging fault between the source and drain of the faulty transistor. It is assumed that IDDQ testing is used. Thus for a stuck-on fault the test generator simply sets the source and drain of the faulty transistor to complementary logic values. The system can also deal with a bridging fault between any two nodes. For stuck-open faults, test invalidation due to charge sharing is not considered. Programs for generating tests for any node stuck-at fault are also implemented. To compare SWiTEST to gate level generators, experiments are performed only on the stuck-at faults at the outputs of transistor groups. Three programs, `songen,` `sopgen` and `sangen`, are implemented to help generate the files containing all stuck-on, stuck-open and stuck-at faults under consideration, respectively.

`switest` is applied independently to each stuck-on, stuck-open and stuck-at faults in each circuit. The experimental results are shown in Table 8.1-8.4. Each of Tables 8.1, 8.2 and 8.3 contains information on three experiments, one for each of the three fault models. The differences between these three tables are the numbers of backtrackings allowed, which are set to 10, 100 and 1000, respectively. Each column of these tables is defined as follows.

circuit : name of the circuit under test.

faults : number of faults under test.

f-det : number of faults detected within the limited number of backtrackings.

f-und : number of faults identified as undetectable within the limited number of backtrackings.

%f-det : percentage of detected faults among all faults.

total time : total CPU time used.

sec/flt : average time spent on each fault.

All faults in the first two circuits, `cadder` and `parity` are detected within 10 backtrackings. For `c60` all detectable faults are detected and all undetectable faults are so identified within 100 backtrackings. All the stuck-open and stuck-at faults of `c880` and the stuck-at faults of `c1355` are also detected. For all other faults or circuits, there exists faults that cannot be detected or identified as undetectable within the limited backtrackings. This is expected since it is well known that the PODEM algorithm cannot efficiently identify undetectable faults even at the gate level.

The total test generation time is dependent on the size of circuits. However it appears that the average test generation time for each type of fault does not totally depend on the size of circuits. For example when the number of backtrackings is set to 100, the average test generation times for `c1355`, `c1908` and `c2670` are about the same. The average test generation time for `c3540` is even larger than that for `c7552` for all three fault models when the number of backtrackings is set to 1000. This is due to the fact that `c3540` has a larger percentage of aborted faults than `c7552`.

Table 8.4 gives the relation between the size of the required memory and the size of a circuit. It also summarizes the percentages of detected faults when the backtracking number is set to 1000. The required memory size is clearly linear with circuit size and is much less than that described in [56] which requires 15M bytes to deal with only 770 transistors.

As expected, the percentage of detected stuck-open faults is in general smaller than for the other two types of faults. However the difference is not significant. For `c880` and `c1908`, the percentages of detected stuck-open faults are higher than those of detected stuck-on faults using IDDQ testing. Considering that tests for stuck-open faults require two vectors and have the problem of invalidation, the experimental results have illustrated that the robust detection of stuck-open faults using PODEM based algorithms is achievable.

The experimental results clearly show that `switest` can deal with much larger circuits than any existing switch level test generators [48, 86, 56, 90, 91]. It is also much more efficient in terms of CPU time and memory usage.

It is difficult to compare the performance of `switest` with those of gate level test generation systems in a formal way due to the difference of available hardware resources and actual implementation. A "rough" comparison is described next between `switest` and the system described in [21], which is probably the fastest gate level test generation system currently available. When the backtracking number is set to 1000, the average time for `switest` to deal with a stuck-at fault for `c7552` is 0.334 seconds. In [21], it takes 41.5 seconds to generate 380 test vectors that detect all detectable stuck-at faults. The 41.5 seconds include fault simulation time. Recent results have shown that fault simulation time and test generation time are usually approximately the same [20]. Therefore the average time to generate a test for a given fault is about $(41.5/2)/380 = 0.0546$ seconds, which is within an order of magnitude of 0.334 seconds.

The above comparison is very rough. However it does provide some interesting observation. The number of backtrackings in [21] is close to zero due to many heuristics employed [20]. For `switest` some faults are aborted after 1000 backtrackings. These faults clearly waste a major portion of cpu time. Therefore we can expect that if similar heuristics used in [21] can be employed in `switest`, the performance of switch level test generation for stuck-at faults can be compatible to that at the gate level. Since the results on `switest` show that there is no significant performance difference between detecting stuck-at, stuck-on and stuck-open faults, it can be concluded that switch level test generation can be done at a performance close to that at the gate level using a PODEM based algorithm.

(a) Stuck-on faults, number of backtrackings = 10

circuit	faults	f-det	f-und	%f-det	total time	sec/flt
cadder	28	28	0	100.00	0.017	0.001
parity	32	32	0	100.00	0.050	0.002
c60	110	107	0	97.27	0.117	0.001
c880	1802	1596	0	88.568	13.567	0.008
c1355	2308	2074	0	89.86	38.033	0.016
c1908	3446	3006	0	87.23	84.967	0.019
c2670	5668	5185	19	91.48	113.117	0.020
c3540	7504	6897	0	91.91	272.283	0.036
c7552	15396	14060	16	91.32	794.150	0.052

(b) Stuck-open faults, number of backtrackings = 10

circuit	faults	f-det	f-und	%f-det	total time	sec/flt
cadder	28	28	0	100.00	0.050	0.002
parity	32	32	0	100.00	0.050	0.002
c60	110	102	0	92.73	0.267	0.002
c880	1802	1802	0	100.00	25.200	0.014
c1355	2308	2050	48	88.82	132.900	0.058
c1908	3446	3303	0	95.85	203.517	0.059
c2670	5668	5286	0	93.26	311.400	0.055
c3540	7504	6507	0	86.71	951.967	0.127
c7552	15396	13999	0	90.93	2557.217	0.166

(c) Stuck-at faults, number of backtrackings = 10

circuit	faults	f-det	f-und	%f-det	total time	sec/flt
cadder	10	10	0	100.00	0.017	0.002
parity	16	16	0	100.00	0.017	0.001
c60	86	84	0	97.67	0.133	0.002
c880	1178	1178	0	100.00	12.250	0.010
c1355	1290	1270	0	98.45	65.250	0.051
c1908	2226	2166	0	97.30	117.683	0.053
c2670	4088	3881	0	94.94	177.167	0.043
c3540	5020	4418	0	88.01	562.467	0.112
c7552	10330	9766	0	94.54	1481.700	0.143

Table 8.1: Experimental results, number of backtrackings = 10

(a) Stuck-on faults, number of backtrackings = 100

circuit	faults	f-det	f-und	%f-det	total time	sec/flt
cadder	28	28	0	100.00	0.017	0.001
parity	32	32	0	100.00	0.033	0.001
c60	110	108	2	98.18	0.167	0.002
c880	1802	1666	0	92.45	34.650	0.019
c1355	2308	2104	0	91.16	72.750	0.032
c1908	3446	3094	0	89.79	158.483	0.046
c2670	5668	5427	29	95.75	192.217	0.034
c3540	7504	7132	8	95.04	527.350	0.070
c7552	15396	14779	26	95.99	1159.950	0.075

(b) Stuck-open faults, number of backtrackings = 100

circuit	faults	f-det	f-und	%f-det	total time	sec/flt
cadder	28	28	0	100.00	0.050	0.002
parity	32	32	0	100.00	0.050	0.002
c60	110	102	8	92.73	0.400	0.004
c880	1802	1802	0	100.00	24.950	0.014
c1355	2308	2050	64	88.82	173.800	0.075
c1908	3446	3370	0	97.79	247.500	0.072
c2670	5668	5339	37	94.20	458.383	0.081
c3540	7504	6738	0	89.79	2009.500	0.268
c7552	15396	14662	0	95.23	3207.950	0.208

(c) Stuck-at faults, number of backtrackings = 100

circuit	faults	f-det	f-und	%f-det	total time	sec/flt
cadder	10	10	0	100.00	0.017	0.002
parity	16	16	0	100.00	0.017	0.001
c60	86	84	2	97.67	0.200	0.002
c880	1178	1178	0	100.00	11.983	0.010
c1355	1290	1270	0	98.45	70.900	0.055
c1908	2226	2194	0	98.56	139.083	0.062
c2670	4088	3917	12	95.82	257.117	0.063
c3540	5020	4573	0	91.10	1293.667	0.258
c7552	10330	10017	0	96.70	2627.400	0.254

Table 8.2: Experimental results, number of backtrackings = 100

(a) Stuck-on faults, number of backtrackings = 1000

circuit	faults	f-det	f-und	%f-det	total time	sec/flt
cadder	28	28	0	100.00	0.033	0.001
parity	32	32	0	100.00	0.033	0.001
c60	110	108	2	98.18	0.167	0.002
c880	1802	1751	0	97.17	126.933	0.070
c1355	2308	2126	0	92.11	414.950	0.180
c1908	3446	3132	0	90.89	948.433	0.275
c2670	5668	5555	29	98.01	483.233	0.085
c3540	7504	7325	12	97.61	1927.150	0.257
c7552	15396	15100	26	98.08	2673.083	0.173

(b) Stuck-open faults, number of backtrackings = 1000

circuit	faults	f-det	f-und	%f-det	total time	sec/flt
cadder	28	28	0	100.00	0.050	0.002
parity	32	32	0	100.00	0.050	0.002
c60	110	102	8	92.73	0.417	0.004
c880	1802	1802	0	100.00	24.867	0.014
c1355	2308	2068	72	89.60	573.500	0.248
c1908	3446	3393	0	98.46	558.267	0.162
c2670	5668	5381	48	94.94	1457.233	0.257
c3540	7504	6888	0	91.79	8826.550	1.176
c7552	15396	14762	0	95.88	7263.917	0.472

(c) Stuck-at faults, number of backtrackings = 1000

circuit	faults	f-det	f-und	%f-det	total time	sec/flt
cadder	10	10	0	100.00	0.017	0.001
parity	16	16	0	100.00	0.033	0.002
c60	86	84	2	97.67	0.217	0.003
c880	1178	1178	0	100.00	12.200	0.010
c1355	1290	1290	0	100.00	80.617	0.062
c1908	2226	2204	0	99.01	306.533	0.138
c2670	4088	3955	16	96.75	895.417	0.219
c3540	5020	4697	0	93.57	5858.317	1.167
c7552	10330	10061	0	97.40	3451.817	0.334

Table 8.3: Experimental results, number of backtrackings = 1000

circuit	# trans	memory	% f-det (# bkk = 1000)		
		(K bytes)	s-on	s-op	s-at
cadder	28	128	100.00	100.00	100.00
parity	32	132	100.00	100.00	100.00
c60	110	140	98.18	92.73	97.67
c880	1802	400	97.17	100.00	100.00
c1355	2308	468	92.11	89.60	100.00
c1908	3446	656	90.89	98.46	99.01
c2670	5668	1044	98.01	94.94	96.75
c3540	7504	1300	97.61	91.79	93.57
c7552	15396	2556	98.08	95.88	97.40

Table 8.4: Memory usage and % of faults detected

Chapter 9

Conclusion

Automatic test generation for CMOS circuits at the switch level is an important and complex problem. This dissertation has attempted to solve many issues associated with this problem. Some of the results have been published in [118, 119, 120, 121, 122, 123, 124, 125, 126]. In this chapter a summary of work described in this dissertation is given and a list of future research topics is provided.

9.1 Summary of the Dissertation

The contributions of this dissertation can be summarized as follows.

9.1.1 Circuit Manipulation

- A circuit partitioning mechanism has been used to divide a CMOS circuit into transistor groups that not only makes it unnecessary to use transistors as the basic unit during test generation, but also defines the I/O relationship among the partitions.

- A complex and primitive gate identification routine has been employed so that whenever possible the switch level test generator can employ the much more efficient gate-level test generation techniques.

- The levelization of circuit partitions provides the information of distances from a partition to its closest primary input and output. This information directs objective selection during backtracing and fault propagation and results in more efficient test generation.

- A new method for assigning signal flow directions to transistors is developed. This method is based on solid mathematical analysis and thus can achieve more accurate results. This method employs several efficient algorithms that outperform all previous methods. One technique that makes use of circuit semantics is also presented which can solve the direction assignment problem in a barrel shifter.

9.1.2 Fault Analysis

- An extensive analysis on the applicability of IDDQ testing on detecting CMOS bridging faults has been carried out. A set of design and test rules are identified which guarantee the detection of all signal and multiple irredundant bridging faults using single vector tests. The problems arisen when each rule is released are analyzed and various strategies are developed to deal with circuits that do not satisfy each rule.

- Test invalidation problems of stuck-open faults due to both circuit delay and charge sharing are analyzed. Through a detailed capacitance analysis it is shown that previous methods dealing with the charge sharing problem are not sufficient. It is also shown that by simply employing IDDQ testing this problem becomes much easier to solve. An efficient algorithm to generate tests that cannot be invalidated by any circuit delay is developed. This algorithm does not require the circuit to be modeled at the gate level, and as long as a pair of vectors are identified, no further test invalidation checking is necessary.

9.1.3 Test Generation Framework

- An objective selection routine that takes into account transistor directions and multiple objectives has been implemented.

- A hybrid backtracing algorithm is implemented. Whenever possible, gate level backtracing is used. Switch level backtracing is much more complex. Two problems that have never been addressed in previous work are identified and solved, namely that backtracing may either fail or enter an infinite loop. The transistor direction information is also used to reduce the backtracing time.

- A new logic implication method is developed that makes use of the full complement property of CMOS circuits. This method is both event-driven and incremental, and is well suitable for use in a PODEM-based test generation algorithm. The average computational time for evaluating a status change of a transistor group is less than linear time.

- A problem of backtracking that does not exist at the gate level but exists at the switch level is identified. This problem results in the necessity of recording the logic value assignment to each node. An efficient data structure is used to store this information.

- Fault propagation at the switch level is analyzed and implemented. It is shown that by separately processing the fault-free and faulty circuits, both effectiveness and efficiency of fault propagation are improved.

9.1.4 Implementation of Individual Test Generators

- Three individual test generators, for bridging (including stuck-on), stuck-open and stuck-at faults, have been implemented. A stuck-on fault is considered as a bridging faults. Experimental results on stuck-on, stuck-open and stuck-at faults are reported. The results imply that switch level test generation can be done in CPU time compatible to that required for gate level test generation. This is true even for stuck-open faults which have previously been considered as much more difficult to detect due to the test invalidation problem. The inefficiency of SWiTEST in identifying redundant faults is not a surprise since even a gate level PODEM cannot deal with this problem well. It is expected that more heuristics such as those used in [23, 19, 20, 21] will be very helpful in alleviating this inefficiency.

9.1.5 Scan Register Testing

- All bridging faults in a specific scan register have been analyzed. The analysis shows that to achieve 100% fault coverage both logic and current must be monitored. A universal test sequence which guarantees the detection of all irredundant bridging, stuck-at and stuck-open faults is identified. The approach is applicable to other scan cell designs.

9.2 Future Work

Though many issues on switch level test generation have been addressed in this dissertation, to develop a highly efficient automatic test pattern generation (ATPG) system at this level still requires more efforts. In this section three areas of future work are discussed, namely 1) improvements on SWiTEST, (2) development of a complete switch level ATPG system, and (3) switch level test generation for more general circuits. The first one is to improve the work done in this dissertation. The second one is an extension of the current work to include fault simulation, fault dropping, and fault coverage analysis, so that a complete ATPG becomes possible. The third one is to deal with the most general problems in test generation, including testing of dynamic and sequential circuits.

9.2.1 Improvements on SWiTEST

Most of the work done for SWiTEST is based on the consideration of implementing the PODEM algorithm at the switch level. Recent developments in gate level test generation have shown that many heuristics can be added to a PODEM-based test generator to enhance its performance. The improvements to SWiTEST can be divided into three parts: preprocessor, test generation framework and individual test generators.

Preprocessor

Concepts such as *Conflict-free assignment* [17], *immediate dominators* [19] and *static learning* [20] can be implemented to the preprocess. The level information of partitions can be replaced by controllability and observability of a circuit node. The implication rule which uses semantic information for assigning signal flow directions to transistors should be implemented.

Test Generation Framework

The concept of *multiple backtracing* used in FAN [17] can be used to improve the backtracing process. *Dynamic learning* of Socrates [20] as well as the information obtained during preprocess can be used to identify more logic implications so as to reduce the possibility of backtracking or to identify redundant faults as early as possible. If a fault is aborted, a *decision strategy switching* [127] may lead to the detection of the fault.

Fault Analysis

SWiTEST assumes that IDDQ testing is used for detecting bridging faults. This may not be true in some cases. Thus a test generation routine for bridging faults using only logic monitoring should be included. Other faults such as delay faults should also be detected. Before doing test generation for these faults, an extensive fault analysis should be conducted to deal with problems such as the following.

- Can theses fault be detected by IDDQ testing?
- Can these faults be converted to other fault models?
- What are the most efficient ways to deal with these faults?

9.2.2 Switch Level ATPG

SWiTEST is not a complete ATPG system since it only generates test vectors for given faults. It should be tied to a fault simulator and a fault dropping routine so that it can perform a complete ATPG process, i.e., given a circuit, identify a reasonable set of test vectors that detect all irredundant faults or achieve a required fault coverage. To do this the following issues must be addressed.

Definition of Fault Coverage

At the gate level the definition of fault coverage is simple because it assumes the well-defined single stuck-at fault model. At the switch level there are many different fault models. Therefore what is an actual fault coverage becomes a problem. It is possible to define a fault coverage for each fault model, but it would be more convenient to define a unified fault coverage for a circuit. However this is difficult because the possibilities of occurrence of different faults may be different. For example it has been argued that the possibility of having stuck-open faults in a circuit is quite small [128]. Thus how to define a unified fault coverage is still a fundamental problem in switch level ATPG.

Switch Level Fault Simulation

Fault simulation for bridging faults using IDDQ testing is simple because a fault is detected when the two shorted nodes have complementary logic values. However if IDDQ testing is not used or the fault model is not the bridging fault one, then the fault simulation problem is more difficult. Some work on switch level fault simulation has been done recently [129]. However this work can only be applied to circuits containing primitive gates and the test invalidation problem is ignored. More research is required before a practical switch level fault simulator can be implemented.

9.2.3 Test Generation for More General Circuits

Non-static Circuits

The test generation algorithms presented in this dissertation can be applied to all static circuits. To deal with non-static circuits, many algorithms need be modified. For example the **Implication** algorithm presented in Chapter 6 cannot be directly applied because it does not model charge retention in a good circuit. Also it has been pointed out that IDDQ testing is not appropriate for non-static circuits. As non-static circuits become more popular in high performance application specific integrated circuits, test generation for non-static circuits will become more important.

Scan-based Sequential Circuits

In this dissertation a universal test sequence is derived that can detect all irredundant faults in one type of scan register. This sequence should also be applicable to other type of registers. However a systematic method to classify all possible scan registers and analyze all faults in each register will be needed before the actual fault coverage can be calculated.

Generic Sequential Circuits

Test generation for sequential circuits at the gate level is currently a very active research area. Due to the development of fast computers and better heuristics and algorithms, sequential test generation is no longer an impractical task for large circuits. It can be expected that, just as for combinational circuits, switch level test generation for sequential circuit will become an important research topic in the near future.

Reference List

[1] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design.* Addison-Wesley, Mass., 1985.

[2] J. Millman. *Microelectronics.* McGraw-Hill Book Co., New York, 1981.

[3] R.L. Wadsack. Fault modeling and logic simulation of CMOS and MOS integrated circuits. *The Bell System Technical Journal*, pages 1449–1474, May-June 1978.

[4] J. Galiay, Y. Crouzet, and M. Vergniault. Physical versus logical fault models MOS LSI circuits: impact on their testability. *IEEE Trans. on Computers*, C-29(6):527–531, June 1980.

[5] C. Timoc, M. Buehler, T. Griswold, C. Pina, F. Scott, and L. Hess. Logical models of physical failures. In *Proc. Int'l. Test Conf.*, pages 546–553, 1983.

[6] R. Chandramouli and H. Sucar. Defect analysis and fault modeling in MOS technology. In *Proc. Int'l. Test Conf.*, pages 313–321, 1985.

[7] J.P. Shen, W. Maly, and F.J. Ferguson. Inductive fault analysis of MOS integrated circuits. *IEEE Design & Test of Computers*, pages 13–26, Dec. 1985.

[8] F.J. Ferguson and J.P. Shen. Extraction and simulation of realistic CMOS faults using inductive fault analysis. In *Proc. Int'l. Test Conf.*, pages 475–484, 1988.

[9] M.A. Breuer and A.D. Friedman. *Diagnosis & Reliable Design of Digital Systems.* Computer Science Press, Inc., 1976.

[10] G.D. Robinson. HITEST–intelligent test generation. In *Proc. Int'l. Test Conf.*, pages 311–323, 1983.

[11] M.H. Shirley. Generating test by exploiting designed behavior. *Proc. of AAAI*, pages 884–890, Aug. 1986.

[12] B. Krishnamurthy. Hierarchical test generation: Can AI help? In *Proc. Int'l. Test Conf.*, pages 694–700, 1987.

[13] N. Singh. *An Artificial Intelligence Approach to Test Generation.* Kluwer Academic Publishers, Norwell, MA, 1987.

[14] M.A. Breuer, R. Gupta, R. Gupta, K.J. Lee, and J.C. Lien. Knowledge-based systems for test and diagnosis. In *IFIP Workshop on KBS for Test & Diagnosis*, Grenoble, France, Sept. 1988.

[15] J.P. Roth, W.G. Bouricius, and P.R. Schneider. Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits. *IEEE Trans. on Electronic Computers*, EC-16(5):567–580, Oct. 1967.

[16] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Trans. on Computerns*, C-30(3):215–222, Mar. 1981.

[17] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Trans. on Computers*, C-32(12):1137–1144, Dec. 1983.

[18] M. Abramovici, J.J. Kulikowski, P.R. Menon, and D.T. Miller. Test generation in LAMP2: system overview. In *Proc. Int'l. Test Conf.*, pages 45–56, 1985.

[19] T. Kirkland and R. Mercer. A topological search algorithm for ATPG. In *Proc. Design Automation Conf.*, pages 502–508, 1987.

[20] M.H. Schulz and E. Auth. Advanced automatic test pattern generation and redundancy identification techniques. In *Proc. Int'l. Symp. on Fault Tolerant Computing*, pages 30–35, 1988.

[21] J.A. Waicukauski, P.A. Shupe, D.J. Giramma, and A. Matin. ATPG for ultra-large structured designs. In *Proc. Int'l. Test Conf.*, pages 44–51, Sept. 1990.

[22] C.F. Hawkins, J.M. Soden, R.R. Fritzemeier, and L.K. Horning. Quiescent power supply current measurement for CMOS IC defect detection. *IEEE Trans. on Industrial Electronics*, pages 211–219, May 1989.

[23] H. Fujiwara. *Logic Testing and Design for Testability.* The MIT Press, Massachusetts, 1985.

[24] F.N. Najm and I. Hajj. The complexity of test generation at the transistor level. Technical Report UILU-ENG-87-2280, DAC-9, Univ. of Illinois at Urbana-Champaign, Dec. 1987.

[25] E. Rich. *Artificial Intelligence.* McGraw-Hill Book Company, New York, 1983.

[26] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In *Proc. Int'l. Symp. on Circuits and Systems*, 1985.

[27] H.C. Shih and J.A. Abraham. Transistor-level test generation for physical failures in CMOS circuits. In *Proc. Design Automation Conf.*, pages 243–249, 1986.

[28] M.K. Reddy, S.M. Reddy, and P. Agrawal. Transistor level test generation for MOS circuits. In *Proc. Design Automation Conf.*, pages 825–828, 1985.

[29] R.L. Wadsack. Fault coverage in digital integrated circuits. *The Bell System Technical Journal*, pages 1475–1488, May-June 1978.

[30] T.W. Williams and K.P. Parker. Design for testability—A survey. *The Proc. of the IEEE*, pages 98–112, Jan. 1983.

[31] E.J. McCluskey. A survey of design for testability scan techniques. *VLSI Systems Design*, pages 38–61, Dec. 1984.

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

[33] C.W. Cha, W.E. Donath, and F. Ozguner. 9-V algorithm for test generation of combinational logic circuits. *IEEE Trans. on Computers*, C-27(3):193–200, Mar. 1978.

[34] M.H. Schulz, E. Trischler, and T.M. Sarfert. SOCRATES: a highly efficient automatic test pattern generation system. In *Proc. Int'l. Test Conf.*, pages 1016–1026, 1987.

[35] S.K. Jain and V.D. Agrawal. Test generation for MOS circuits using D-algorithm. In *Proc. Design Automation Conf.*, pages 64–70, 1983.

[36] S.M. Reddy, V.D. Agrawal, and S.K. Jain. A gate level model for CMOS combinational logic circuits with application to fault detection. In *Proc. Design Automation Conf.*, pages 504–509, 1984.

[37] S.M. Reddy and M.K. Reddy. Robust tests for stuck-open faults in CMOS combinational logic circuits. In *Proc. Int'l. Symp. on Fault Tolerant Computing*, pages 44–49, 1984.

[38] S.A. AlArian and D.P. Agrawal. CMOS fault testing: multiple faults in combinational circuits; single fault in sequential circuits. In *Proc. Int'l. Test Conf.*, pages 218–223, 1984.

[39] J.P. Roth, V.G. Oklobdzija, and J.F. Beetem. Test generation for FET switching circuits. In *Proc. Int'l. Test Conf.*, pages 59–62, 1984.

[40] J.F. Wang, T.Y. Kuo, and J.Y. Lee. A new approach to derive robust tests for stuck-open faults in CMOS combinational logic circuits. In *Proc. Design Automation Conf.*, pages 726–729, June 1989.

[41] P. Lamloureux and V.K. Agrawal. Non-stuck-at fault detection in NMOS circuits by region analysis. In *Proc. Int'l. Test Conf.*, pages 129–137, 1983.

[42] C.Y. Lo, H.N. Nham, and A.K. Bose. A data structure for MOS circuits. In *Proc. Design Automation Conf.*, pages 619–624, 1983.

[43] K.W. Chiang and Z.G. Vranesic. On fault detection in CMOS logic networks. In *Proc. Design Automation Conf.*, pages 50–56, 1983.

[44] P. Agrawal. Test generation at switch level. In *Proc. Int'l. Conf. on Computer-Aided Design*, pages 128–130, 1984.

[45] R.E. Bryant. An algorithm for MOS LSI logic simulation. *LAMBDA*, pages 46–53, Fourth Quarter 1980.

[46] N.P. Jouppi. Derivation of signal flow direction in MOS VLSI. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(3):480–490, May 1987.

[47] M.A. Cirit. Switch level random pattern testability analysis. In *Proc. Design Automation Conf.*, pages 587–590, 1988.

[48] H.H. Chen, R.G. Mathews, and J.A. Newkirk. An algorithm to generate tests for MOS circuits at the switch level. In *Proc. Int'l. Test Conf.*, pages 304–312, 1985.

[49] Z. Barzilai, D.K. Breece, L.M. Huisman, V.S. Iyengar, and G.M. Silberman. SLS–A fast switch level simulator. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, CAD-7(8):838–849, Aug. 1988.

[50] R.E. Bryant. A survey of switch-level algorithms. *IEEE Design & Test of Computers*, pages 26–40, Aug. 1987.

[51] R.E. Bryant. A switch-level model and simulator for MOS digital systems. *IEEE Trans. on Computers*, C-33(2):160–177, Feb. 1984.

[52] R.E. Bryant. Algorithmic aspects of symbolic switch network analysis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(4):618–633, July 1987.

[53] R.E. Bryant. Boolean analysis of MOS circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(4):634–649, July 1987.

[54] M. Schuster. Switch-level fault simulation of MOS digital circuits. Master's thesis, California Institute of Technology, May 1984.

[55] M.D. Schuster and R.E. Bryant. Concurrent fault simulation of MOS digital circuits. In *1984 M.I.T. Conf. on Advanced Research in VLSI*, pages 129–138, Jan. 1984.

[56] K. Cho and R.E. Bryant. Test pattern generation for sequential MOS circuits by symbolic fault simulation. In *Proc. Design Automation Conf.*, pages 418–428, 1989.

[57] S. Jan. Applying inductive fault analysis and symbolic boolean analysis in the extraction, analysis and testing of bridging faults. Technical Report SRC Tech. Rep. T88099, Carnegie-Mellon Univ., Sept. 1988.

[58] P. Agrawal, S.H. Robinson, and T.G. Szymanski. Automatic modeling of switch-level networks using partial orders. In *Proc. Int'l. Conf. on Computer-Aided Design*, pages 350–353, 1988.

[59] M.W. Levi. CMOS is most testable. In *Proc. Int'l. Test Conf.*, pages 217–220, 1981.

[60] Y.K. Malaiya and S.Y. Su. A new fault model and testing technique for CMOS devices. In *Proc. Int'l. Test Conf.*, pages 25–34, 1982.

[61] J.M. Acken. Testing for bridging faults (shorts) in CMOS circuits. In *Proc. Design Automation Conf.*, pages 717–718, 1983.

[62] J.M. Soden and C.F. Hawkins. Test considerations for gate oxide shorts in CMOS ICs. *IEEE Design & Test of Computers*, 3(4):56–64, Aug. 1986.

[63] C. Crapuchettes. Testing CMOS Idd on large devices. In *Proc. Int'l. Test Conf.*, pages 310–315, 1987.

[64] W. Maly (editor). Built-in current testing — Part I. Technical Report CMUCAD-88-27, Carnegie-Mellon Univ., June 1988.

[65] P. Nigh and W. Maly. A self-testing ALU using built-in current sensing. In *Proc. Custom Integrated Circuits Conf.*, pages 22.1.1–22.1.4, 1989.

[66] R.R. Fritzemeier, J.M. Soden, R.K.Treece, and C.F. Hawkins. Increased CMOS IC stuck-at fault coverage with reduced IDDQ testing sets. In *Proc. Int'l. Test Conf.*, pages 427–435, 1990.

[67] F.J. Ferguson, M. Tayler, and T. Larrabee. Testing for parametric faults in static CMOS circuits. In *Proc. Int'l. Test Conf.*, pages 436–443, 1990.

[68] T.M. Storey and W. Maly. CMOS bridging faults detection. In *Proc. Int'l. Test Conf.*, pages 842–851, Sept. 1990.

[69] K.C.Y. Mei. Bridging and stuck-at faults. *IEEE Trans. on Computers*, C-23(7):720–727, July 1974.

[70] P. Banerjee and J. Abraham. Generating tests for physical failures in MOS logic circuits. In *Proc. Int'l. Test Conf.*, pages 554–559, 1983.

[71] Y.K. Malaiya, A.P. Jayasumana, and R. Rajsuman. A detailed examination of bridging faults. In *Proc. Int'l. Test Conf.*, pages 78–81, 1986.

[72] R. Rajsuman, Y.K. Malaiya, and A.P. Jayasumana. On accuracy of switch-level modeling of bridging faults in complex gates. In *Proc. Design Automation Conf.*, pages 244–250, 1987.

[73] W. Maly, P.K. Nag, and P. Nigh. Testing oriented analysis of CMOS ICs with opens. In *Proc. Int'l. Conf. on Computer-Aided Design*, pages 344–347, 1988.

[74] P. Nigh and W. Maly. Layout-driven test generation. In *Int'l. Conf. on Computer-Aided Design*, pages 154–157, Nov. 1989.

[75] J.M. Soden, R.K. Treece, M.R. Tailor, and C.F. Hawkins. CMOS IC stuck-open fault electrical effects and design consideration. In *Proc. Int'l. Test Conf.*, pages 423–430, Aug. 1989.

[76] S.D. Millman, E.J. McCluskey, and J.M. Acken. Diagnosing CMOS bridging faults with stuck-at fault dictionaries. In *Proc. Int'l. Test Conf.*, pages 860–870, Sept. 1990.

[77] Y.K. Malaiya. Testing stuck-on faults in CMOS integrated circuits. In *Int'l. Conf. on Computer-Aided-Design*, pages 248–250, 1984.

[78] R. Rajsuman, A.P. Jayasumana, and Y.K. Malaiya. CMOS stuck-open fault detection using single test patterns. In *Proc. Design Automation Conf.*, pages 714–717, June 1989.

[79] Y.M. El-ziq and R.J. Cloutier. Functional level test generation for stuck-open faults in CMOS VLSI. In *Proc. Int'l. Test Conf.*, pages 536–546, 1981.

[80] Y.M. El-ziq. Automatic test generation for stuck-open faults in CMOS VLSI. In *Proc. Design Automation Conf.*, pages 347–354, 1981.

[81] Y.M. El-ziq and S.Y.H. Su. Fault diagnosis of MOS combinational networks. *IEEE Trans. on Computers*, C-31(2):129–139, Feb. 1982.

[82] Y.M. El-ziq. Classifying, testing and eliminating VLSI MOS failures. *VLSI Design*, pages 30–35, Sept. 1983.

[83] Y.M. El-ziq. Failure analysis and test generation for VLSI physical defects. In *Proc. Custom Integrated Circuit Conf.*, pages 300–303, 1983.

[84] K.W. Chiang and Z.G. Vranesic. Test generation for MOS complex gate networks. In *Proc. Int'l. Symp. on Fault Tolerant Computing*, pages 149–157, 1982.

[85] R. Chandramouli. On testing stuck-open faults. In *Proc. Int'l. Symp. on Fault Tolerant Computing*, pages 258–265, 1983.

[86] F.N. Najm. Switch-level test generation for MOS VLSI circuits. Master's thesis, Univ. of Illinois at Urbana-Champaign, Aug. 1986.

[87] G. Gupta and N.K. Jha. A universal test set for CMOS circuits. *IEEE Trans. on Computer-Aided Design of Intergrated Circuits and Systems*, CAD-7(5):590–597, May 1988.

[88] S.B. Akers. Universal test sets for logical networks. *IEEE Trans. on Computers*, C-22(9):835–839, Sept. 1973.

[89] S.M. Reddy. Complete test sets for logical networks. *IEEE Trans. on Computers*, C-22(11):1016–1020, Nov. 1973.

[90] C.H. Chen and J.A. Abraham. Mixed-level sequential test generation using a nine-valued relaxation algorithm. In *Int'l Conf. on Computer-Aided Design*, pages 230–233, Nov. 1990.

[91] M.K. Reddy. *Testable CMOS digital design and switch-level test generation for MOS digital circuits.* PhD thesis, Univ. of Iowa, Iowa City, 1991.

[92] N.K. Jha and J.A. Abraham. Testable CMOS logic circuits under dynamic behavior. In *Proc. of Int'l. Conf. on Computer-Aided Design*, pages 131–133, 1984.

[93] S.M. Reddy and M.K. Reddy. Testable realizations for FET stuck-open faults in CMOS combinational logic circuits. *IEEE Trans. on Computers*, C-35(8):742–754, Aug. 1986.

[94] S. Koeppe. Optimal layout to avoid CMOS stuck-open faults. In *Proc. Design Automation Conf.*, pages 829–835, 1987.

[95] S.D. Sherlekar and P.S. Subramanian. Conditionally robust two-pattern tests and CMOS design for testability. *IEEE Trans. on Computer-Aided Design of Intergrated Circuits and Systems*, C-37(3):325–332, Mar. 1988.

[96] S. Kundu and S.M. Reddy. Robust tests for parity trees. In *Int'l. Test Conf.*, pages 680–687, 1988.

[97] S. Kundu and S.M. Reddy. On the design of robust testable CMOS combinational logic circuits. In *Proc. Int'l. Symp. on Fault Tolerant Computing*, pages 220–225, 1988.

[98] R.J. Evans and W.R. Moore. Layout based testing of CMOS logic. Technical Report Alvey–CAD042–OX3–16, Dept. of Engineering Science, University of Oxford, UK, December 1989.

[99] S.H. Robinson and J.P. Shen. Towards a switch-level test generation problem. In *Proc. Int'l. Conf. on Computer-Aided Design*, pages 39–41, 1985.

[100] S.H. Robinson. Switch-level automatic test pattern generation. Master Thesis CMUCAD-87-43, Carnegie-Mellon Univ., Sept. 1987.

[101] P. Goel and M.T. McMahon. Electronic chip-in-place test. In *Proc. Int'l. Test Conf.*, pages 83–90, 1982.

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

[103] Y. Shiloach. A polynomial solution to the undirected two paths problem. *Journal of the ACM*, 27(3):445–456, July 1980.

[104] P.D. Seymour. Disjoint paths in graphs. *Discrete Mathematics*, 29(1980) 293-309.

[105] B. Mishra. An efficient algorithm to find all 'bidirectional' edges of an undirected graph. In *Annual Symp. on Foundations of Computer Science*, pages 207–216, 1984.

[106] S.L. Lu. Internal memo of MOSIS project. 1988.

[107] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[108] H.W. Law, Jr. A.F. Tasch, and R.F. Pinizzotto. Silicon-on-insulator for VLSI and VHSIC. In N.G. Einspruch, editor, *VLSI Electronics, Microstructure science*, chapter 1, pages 1–54. Academic Press, New York, 1982.

[109] J.R. Davis, A.E. Glaccum, K. Reeson, and P.L. Hemment. Improved subthreshold characteristics of n-channel SOI transistors. *IEEE Electron Device Letters*, EDL-7(10):570–572, Oct. 1986.

[110] T.W. Houston. Considerations for the design of an SRAM with SOI technology. *IEEE Circuits & Devices Magazine*, pages 8–10, Nov. 1987.

[111] S. Tsukiyama, H. Ariyoshi, and I. Shirakawa. Algorithm to enumerate all the cutsets in $o(|v| + |e|)$ time per cutset. In *Proc. Int'l. Symp. on Cirucits and Systems*, pages 645–648, 1979.

[112] H. Ariyoshi. Cutset graph and systematic generation of separating sets. *IEEE Trans. Circuit Theory*, pages 233–240, March 1972.

[113] M.A. Cirit. The Meyer model revisited: Why is charge not conserved? *IEEE Trans. on Computer-Aided Design of Integrated Circuits and System*, CAD-8(10):1033–1037, Oct. 1989.

[114] S. Kundu, S.M. Reddy, and N.K. Jha. On the design of robust multiple fault detectable CMOS combinational logic circuits. In *Int'l. Conf. on Computer-Aided Design*, pages 240–243, 1988.

[115] Z. Barzilai, J.L. Carter, V.S. Iyengar, I. Nair, B.K. Rosen, J. Rutledge, and G.M. Silberman. Efficieny fault simulation of CMOS circuits with accurate models. In *Proc. Int'l. Test Conf.*, pages 520–529, 1986.

[116] A. Shoji. *CMOS digital circuit technology.* Prentice Hall, New Jersey, 1988.

[117] E.B. Eichelberger and T.W. Williams. A logic design structure for LSI testability. In *Design Automation Conf.*, pages 462–468, 1977.

[118] K.J. Lee and M.A. Breuer. Detailed analysis of bridging faults in CMOS scan registers. Technical Report CENG-89-05, Univ. of Southern California, Jan. 1989.

[119] K.J. Lee and M.A. Breuer. Detecting multiple bridging faults in CMOS combinational circuits. Technical Report CRI-88-62, Univ. of Southern California, Jan. 1989.

[120] K.J. Lee and M.A. Breuer. A universal test sequence for CMOS scan registers. In *Proc. Custom Integrated Circuit Conf.*, pages 28.5.1–28.5.4, May 1990.

[121] K.J. Lee and M.A. Breuer. On detecting single and multiple bridging faults in CMOS circuits using the current supply monitoring method. In *Proc. Int'l. Symp. on Circuits and Systems*, pages 5–8, May 1990.

[122] K.J. Lee and M.A. Breuer. Assigning signal flow direction to MOS transistors. Technical Report CENG-90-09, Univ. of Southern California, Los Angeles, California, Feb. 1990.

[123] K.J. Lee and M.A. Breuer. On the charge sharing problem in CMOS stuck-open fault testing. In *Proc. Int'l. Test Conf.*, pages 417–426, Sept. 1990.

[124] K.J. Lee and M.A. Breuer. A new method for assinging signal flow directions to MOS traisistors. In *Proc. Intl' Conf. on Computer-Aided Design*, pages 492–495, Nov. 1990.

[125] K.J. Lee and M.A. Breuer. Design & test rules for CMOS circuits to facilitate IDDQ testing to detect bridging faults. *To be published in IEEE Trans. on Computer-Aided Design on Integrated Circuits and Systems*, 1991.

[126] K.J. Lee and M.A. Breuer. Constraints for using IDDQ testing to detect CMOS bridging faults. In *IEEE VLSI Test Symposium*, pages 303–308, May 1991.

[127] H.B. Min and W.A. Rogers. Search strategy switching: an alternative to increased backtracking. In *Proc. Int'l. Test Conf.*, pages 803–811, 1989.

[128] B.W. Woodhall, B.D. Newton, and A.G. Sammuli. Empirical results on undetcted CMOS stuck-open failures. In *Peoc. Int'l. Test Conf.*, pages 166–170, 1987.

[129] M. Favalli, P. Olivo, M. Damiani, and B. Ricco. Fault simulation of unconventional faults in CMOS circuits. *IEEE Trans. on Computer-Aided Design for Integrated Circuits & Systems*, CAD-10(5):677–682, May 1991.