# C3 Testability Strategy

December 12, 1988

CONVEX Computer Corporation

# Table of Contents

**1 C3 Test Strategy**
1.1 Introduction ..... 1-1
1.2 Intended Audience ..... 1-1
1.3 Contents of This Paper ..... 1-1

**2 Testing: Testability, Design, and Verification**
2.1 Testability ..... 2-1
2.2 Testing ..... 2-2

**3 Logic Testing**
3.1 What is Logic Testing? ..... 3-1
3.2 Faults and Fault Models ..... 3-2
3.3 Semiconductor Reliability and Failure Mechanisms ..... 3-9
3.4 Semiconductor Processes and Related Failure Mechanisms ..... 3-13

**4 Test Pattern Generation Methods and Algorithms**
4.1 Stuck-at Fault Algorithms for Combinational Circuits ..... 4-1
4.2 Testability Analysis ..... 4-3
4.3 Real Implementations ..... 4-3
4.4 Comparison of Stuck-at Fault Algorithms ..... 4-3
4.5 Stuck-at Test Generation for Sequential Circuits ..... 4-5
4.6 Delay Fault Algorithms ..... 4-5
4.7 Other Types of Fault Detection Algorithms ..... 4-8
4.8 Fault Coverage ..... 4-9
4.9 Tests for Memory Circuits ..... 4-9

**5 Design For Testability**
5.1 Scan Designs ..... 5-1
5.2 Self-Testing Circuits ..... 5-5

**6 Detection of Intermittent Failures**
6.1 Error Checking Circuits ..... 6-2

**7 C3 Testability Strategy**
7.1 C3 Testability Goals ..... 7-2
7.2 Design Rules ..... 7-3

**A Bibliography**

**B Glossary**

**C Stuck-at Fault Test Generation Algorithms**
Stuck-at Fault Algorithms ..... C-1

# List of Tables

3-1 Detectability of Varying Levels of Delay Defects ..... 3-4
3-2 Physical and Logical Faults in CMOS NAND Gate ..... 3-15
3-3 Physical and Logical Faults in CMOS NOR Gate ..... 3-15
3-4 VLSI Failure Mechanisms ..... 3-17
4-1 ISCAS Benchmark Circuits ..... 4-4
4-2 Performance Comparison of Stuck-at Fault Pattern Generators ..... 4-4
4-3 Delay Faults in ISCAS Circuits ..... 4-8
4-4 Stuck-at Fault Memory Algorithms ..... 4-10

4-5 Comparison of Coupling Fault Tests for Memory Circuits ..... 4-11
6-1 Error Detection Capability of Code Words ..... 6-2
6-2 Check Bits Required For Single Bit Error Correction ..... 6-2

# List of Figures

3-1 Gate Delays ..... 3-4
3-2 Crosspoint Fault in PLA ..... 3-6
3-3 Bridging Fault in TTL ..... 3-7
3-4 Stuck Open CMOS Fault ..... 3-8
3-5 CMOS NOR and NAND Gates ..... 3-14
4-1 Delay Fault Example ..... 4-7
5-1 LSSD Latch ..... 5-2
5-2 Stuck-at Testing Through Scan Rings ..... 5-4
5-3 Delay Fault Testing Through Scan Rings ..... 5-5
5-4 Self-Test Circuitry ..... 5-6
5-5 Self-Testing Ram Circuit ..... 5-8
6-1 Residue Code Circuit ..... 6-3
C-1 Singular Cover of Primative Logic Gates ..... C-1
C-2 D-Algorithm Example Circuit ..... C-3
C-3 Flowchart of D-Algorithm ..... C-4
C-4 Troublesome ECC Circuit ..... C-5
C-5 Flowchart of Podem Algorithm ..... C-6
C-6 Flowchart of FAN Algorithm ..... C-8

# 1

# C3 Test Strategy

## 1.1 Introduction

Convex is behind the state of the art in all facets of hardware test. Some of this is attributed to the relatively low volume and high cost of our systems. It is not feasible for Convex to install the type of diagnostic systems which IBM has installed, simply because it is not cost effective. However, a full explanation for Convex lagging the industry must include the company's lethargic attitude toward hardware test and development of hardware test technology. Since it can not be sold to a customer and does not show up immediately on the bottom line, less resources than necessary have been allocated to addressing an adequate solution to this problem. Test is a strategically vital technology for Convex to aggressively pursue and develop.

This paper details a proposed test strategy for the C3 system. Issues discussed cover: strategies for architectural verification, design correctness testing, manufacturing testing, and field service testing for the entire system. Design rules and guidelines for the C3 system are provided as they are the basis of the test strategy and implementation. Also included are materials that serve as a tutorial on test techniques and strategies that are in current use, or have emerged from research in the past decade. Appendix A contains a list of test technology source materials used in preparing this paper. A glossary is included in appendix B of this document to define the test-related terminology that is used throughout this document. Last, appendix C contains tutorial information and examples on some of the different stuck-at fault test generation algorithms.

The test technology strategy defined in this document is a leap forward for Convex. Any time an effort is made to move forward in any area of technology, there is an enormous amount of risk involved. Convex has operated in this mode for the past six years. Fear of failure can not paralyze us from doing the right thing. We must successfully implement this new test strategy now, lest we be overrun with our inability to manufacture and support increasingly complex systems in the future.

## 1.2 Intended Audience

This paper is intended for anyone in manufacturing, field service or engineering that is now, or will be involved in the C3 project. A successful implementation of the test strategy depends upon a cooperative effort among each of the involved groups. This cooperation includes the task of defining the problems from a Convex perspective so all departmental testing problems can be addressed without duplication of effort and with an increased probability of success. Hence, it is vital that each of the groups commit to supporting this test strategy and providing their input so that the tests and tools developed with this strategy can effectively be used by each group. Without the support of all of these groups, this strategy is doomed to failure.

## 1.3 Contents of This Paper

The paper is divided into seven chapters. Tutorial and introductory material is included in the front chapters to support the technical and philosophical portions of the strategy provided at the end. The contents of each chapter is described briefly below.

### 1.3.1 Chapter 2: Testing: Testability, Design, and Architecture Verification

This chapter discusses the importance of testability, design verification and architecture verification to Convex. Design verification is discussed with major emphasis placed on the simulation environment Architectural verification is presented from the computer industry view with emphasis on its important role in the development, manufacturing, and field service environments. Definitions of what testing, testability, design verification, and architectural verification are included in this chapter.

### 1.3.2 Chapter 3: Logic Testing

This chapter discusses logic testing. Included is a basic definition of logic testing, a discussion of faults and fault models and a discussion of semiconductor reliability and failure mechanisms. These concepts are tied together in a way to allow an understanding of what is trying to be accomplished by logic testing and how the requirements of the logic tests vary throughout the lifecycle of a product.

### 1.3.3 Chapter 4: Test Pattern Generation Methods and Algorithms

This chapter discusses test pattern generation to support logic testing. The most commonly used algorithms to support logic testing are also explained. A discussion of fault coverage as it relates to logic testing and a discussion of fault simulators is included. This chapter also discusses algorithms used for testing memory circuitry.

### 1.3.4 Chapter 5: Design For Testability

This chapter covers design for testability. Included in this chapter are discussions of circuit partitioning, scan designs and self-testing circuits. The concepts of self-testing combinatorial and sequential logic is covered as well as testing RAMS.

### 1.3.5 Chapter 6: Detection of Intermittent Failures

This chapter details the methods by which intermittent faults can be captured during the operation of a system. Several large systems manufacturers claim that intermittent failures account for over 90% of the actual failures that are seen in the field. Because of this, a successful strategy test can not ignore this important problem. Error detecting circuits will be discussed and system implications of the implementation of this type of circuitry will be identified.

### 1.3.6 Chapter 7: C3 Testability Strategy

This chapter defines the C3 diagnostic strategy relating to design testability, testability goals, and the design rules necessary to reach our goals.

# 2

# Testing: Testability, Design, and Verification

If anything can go wrong, it will; or so goes Murphy's Law. Basing a product on this premise, provokes an intense interest in being able to determine the correctness of a system design and the likehood the design will function properly. If the product fails to function, how is the problem determined? Answering this question requires a comprehensive approach to testing which provides two features: verifying that a product is correctly functioning and identifying the failing part of the product for replacement or repair.

Testing, in the general sense, means examining a product (the whole, as well as the parts) to ensure that it functions and exhibits the properties and capabilities that it was intended to provide. If the product does not function properly, then the 'testing' will discover the malfunction in the product hardware and locate the cause(s) which can then be eliminated. The concept of testing is simple, but its implementation to adequately test or not test a design is based directly on how well the product was designed for testing, or 'testability'. The following subsections will discuss testability and testing along with its subcomponents.

## 2.1 Testability

Testability simply means "capable of being tested". Circuit testability is defined as the ability to identify and diagnose failures in a circuit. Three major components of testability are: control, observation, and isolation. Control is the function of setting circuit conditions so stimuli can be applied to the circuit. Observation is the function of obtaining the circuit response from the application of the stimuli to allow evaluation of the circuit operation. Isolation is the function of partitioning a system so that control and observation are possible and more reliable [*86Tsui*].

If a circuit is highly testable, then it can be inferred that the circuit is highly controllable, observable, and can be isolated from other components in the system which makes the circuit more reliable.

### 2.1.1 Testability Costs

To view the cost of testability, one need only look at the cost of not being testable. The following questions have not been answered because of the limited research time. But as a reader, consider the questions in terms of your environment and evaluate what affect testability has on your work and performance.

Consider the cost of a product which is not ready for use when needed. What is the penalty Convex receives for not delivering a product on schedule? Consider the cost of the product not functioning properly when in use. What is the value of the machine down time to a customer and what view does the customer take of Convex? Consider the manufacturing cost involved with bringing up a system which in not testable. How much of Convex's resources are spent on repetitive manual product debugging? What impact does the repetitive debugging have on manufacturing personnel's attitudes? What manufacturing innovations were not addressed because of the labor intense function of manually debugging a system? What is the ultimate impact on Convex's bottom line for not designing testable systems? What is the cost to Convex

for approaching testability from a departmental view rather than from a corporate view?

## 2.2 Testing

Convex implements testing as verification of the original design and of the design's implementation. Using Convex's view of testing, a discussion of testing can be divided into two categories: design verification and architectural verification.

### 2.2.1 Design Verification

Design verification is testing which verifies that a design functions as intended. Historically, the advent of higher operating speed, higher density components, and shorter development cycles has caused the process of original design verification to evolve. When operating speeds where slow, designs were bread-boarded and verified. As the speeds increased, the designs were implemented and then made to work. But the continued increases in speeds, component package densities, and costs of the devices caused a new method of design verification to evolve: design simulation.

To discuss simulation and design verification, one must define some common terms. First, a behavioral model describes what a function does; not how it does it. Second, a structural model includes the information on the gate level interconnection of lower level components [*84Davi*]. Design verification using simulation can be described as follows. A design is defined and implemented in what is referred to as the behavioral model(s). The goal is to design the models so that when stimuli is applied to the models, they execute to produce expected results. When the models are completely verified, then an iterative process of replacing the behavioral models with their structural models (laying the actual gates) is performed and simulating of the hybrid models (a mix of behavioral and structural) is performed. The result is a gate level model of the system which has been simulated to verify its correctness. Next, timing verification is required to find maximum operational speeds, simulation to determine its dynamic behavior limitations (critical-path delays, possible hazards in circuit operation, and noise sensitivity), and simulation to estimate and evaluate the design's projected performance and testability [*86Tsui*].

### 2.2.2 Architectural Verification

No description of the stimuli used to verify the models was included in the previous discussion about design verification. In the case of a computer architecture, a set of tests (each test comprised of specific instruction code sequences) would have been the stimuli. This set of stimuli is referred to as architectural verifiers. Thus, the function of applying architectural verifiers in the design verification process is referred to as architectural verification.

Architectural verification exist in the development, manufacturing, and field service cycles. In the development cycle, architectural verification is used to verify that the actual design was indeed correct. In the manufacturing cycle, they are used to verify that the design represented in the schematics is the design which is implemented on the boards. In the field service environment architectural verification is used either to isolate a problem or verify a problem has been corrected. Historically, if each instruction is exhaustively tested along with exhaustively testing a machine's faulting and exception processing; the machine was considered 100% tested. Today this is still true for the manufacturing and field service environments. But simulation speed has changed the size of the set of architectural verifiers which development can realistically use.

A reduced or regression set of the architectural verifiers must be generated which allows adequate testing of the architecture within a reasonable amount of time. Before, a set of regression tests were put together via discussions with the designers. Several articles are appearing on characterizing the architecture of microprocessors and multiprocessors. For C3, it may be feasible that a method described in an article could be extrapolated to large system design. This avenue

is being investigated. Another issue arising from this is if an architecture can be modeled and its effective coverage calculated, then the regression tests could be of benefit to manufacturing and field service as a quick sanity check of the machine's operation.

Actual literature in this area is very limited, but a company's approach to architectural verification can be extracted from the way the machines are tested in the field. In the case of IBM, they build machines which are highly testable. A system is divided into many subsystems which are subdivided into functions. Within each function, they have included level sensitive scan devices, in-circuit checkers, and micro-code diagnostics. The approach is to verify the low level function for correct operation and then verify the function interfaces. As subsystems are verified, then the subsystem interfaces are verified. Thus the approach is to test and verify extensively the lowest level operations, and then hierarchically verify functional interfaces, subsystem interfaces, and the system interface. The result for IBM has been machines which have high reliability, and short mean time to repair. They have conquered the stuck-at and intermittent faults and are addressing transient faults. The costs involved with this approach are long development cycles and high machine cost.

Convex approaches the task from the other end. First, functional diagnostics are generated to exercise the major functions of a system from the service processor. Then a suite of exhaustive (supposedly) instructional tests are generated to exercise the architecture. If they pass, the architecture is verified; otherwise, the architecture problems are debugged using these instruction suites. In an initial prototype environment, this approach is adequate; however, in a manufacturing environment or field service environment, it is not adequate.

# 3

# Logic Testing

## 3.1 What is Logic Testing?

Logic testing verifies that the components of a logic circuit are functional. The goal of the test is to assure that there are no faults present in the system. To assure that there are no faults present, tests are designed to try to detect the presense or absence of these faults. The key to successful logic testing lies in the test engineer *identifying* faults that are likely to occur in the system and then designing tests that have high probability of detecting them.

In order to simplify the study of logic testing it is convenient to partition the set of logic devices into the following categories:

- Combinational logic circuits.
- Sequential logic circuits.
- Memory array circuits.
- Programmable Logic Arrays (PLAs).

The partitioning of logic circuits into these classes is particularly useful when considering testability and test generation. Each category has similar test generation and test application methods which requires different test strategies and may exhibit failure modes unique to that category.

### 3.1.1 Combinational Logic

A combinational logic circuit consists of an interconnected set of gates with no feedback loops. The output values of a combinational circuit at a given time depend only on the present inputs. Testing of purely combinational logic is fairly straightforward, with one method being to truth-table test the circuit. This is done by applying each of the possible $2^N$ input patterns, for an N-input circuit, and verifying the outputs. This method quickly becomes prohibitive for large circuits. To solve this problem, test pattern generators are used to generate an efficient set of test patterns aimed at detecting certain types of faults within the logic circuit. The types of faults targeted depend upon the technology from which the circuit is made. There are well-known algorithms in use that can generate high fault-coverage test patterns for certain types of assumed faults.

### 3.1.2 Sequential Logic

A sequential logic circuit contains feedback loops. The output values at a given time depend on the present inputs and also on inputs applied previously. Sequential circuits are much harder to test than combinational ones. Although several automatic test pattern generators for certain classes of sequential logic have been built, a test pattern generator that can develop tests for all types of sequential logical circuits has not yet been developed.

### 3.1.3 Memory Circuits

Memory circuits are special cases of sequential logic circuits and deserve special treatment because of the difficulty of testing them as a normal sequential logic machine. The number of states that a memory element can have is enormous, being $2^N$ where $N$ is the number of *bits* in the device! Memory devices are subject to different types of faults which are not normally tested for in the testing of sequential and combinational logic. In particular, memory circuits are prone to exhibit pattern-sensitive and coupling faults.

### 3.1.4 PLAs and PALS

Programmable logic arrays and PALS represent a special type of logic circuit. Although their main use is the implementation of sequential and combinational logic circuits, they deserve special treatment. The internal structure of these devices and the techniques involved in their manufacture causes them to be subject to a different class of faults than standard logic devices. In particular, crosspoint and bridging faults occur quite often in these devices.

## 3.2 Faults and Fault Models

A fault in a circuit is caused by a physical defect in one or more of the components contained in the circuit. Physical imperfections manifest themselves in an almost infinite variety of faults. However, research shows that certain types of semiconductor devices and processes are more prone to certain types of failure mechanisms than others. Successful test generation includes the process of determining which faults are of interest for the device(s) under test, and then generating tests that can detect the presence of these faults.

The use of models is ubiquitous throughout engineering. Test generation, like circuit design, relies on models. Models are useful and necessary in order to simplify extremely complex devices. Two types of models used in test generation are circuit models and fault models. Circuit models emulate the correct operation of the device when viewed from a macro-level. Examples are 'AND' gates, 'OR' gates, inverters, pass transistors, etc. Fault models are used to describe the operation of a circuit model in the presence of a physical imperfection.

Faults are classified as logical or parametric. A logical fault is one that causes the logic function of a circuit element or an input signal to be changed to some other logic function. A parametric fault alters the magnitude of a circuit parameter, causing a change in some factor such as circuit speed, current, or voltage. Circuit malfunctions associated with timing are due mainly to circuit delays and are called delay faults. Recurring faults that are present in some intervals of time and absent in others are intermittent faults. Nonrecurring faults are called transient faults. Faults that are always present and do not occur, disappear, or change their nature during testing are called permanent or solid faults.

Faults can be modeled either at the physical or the functional model. The delay and stuck-at faults are physical fault models which are used with gate and transistor level circuits. Another set of fault models which are used to model failures in more complex circuits are the functional fault models. Functional fault models are successfully used to model faults that occur in memory circuits, in particular: stuck cells, coupling and pattern-sensitive faults.

A study of the effects of physical failures at the circuit level has been made using a circuit level simulator such as SPICE. The correspondence between the physical failures and the circuit behavior can be understood, and from this understanding fault models can be created.

The majority of research in logic test generation has focused on two major types of fault models, the stuck-at fault and the delay fault. The reasons that these fault models are so wide-spread is that a large number of semiconductor manufacturing defects cause the circuit, on the macro-level,

to exhibit one of these fault types.

### 3.2.1 Delay Faults

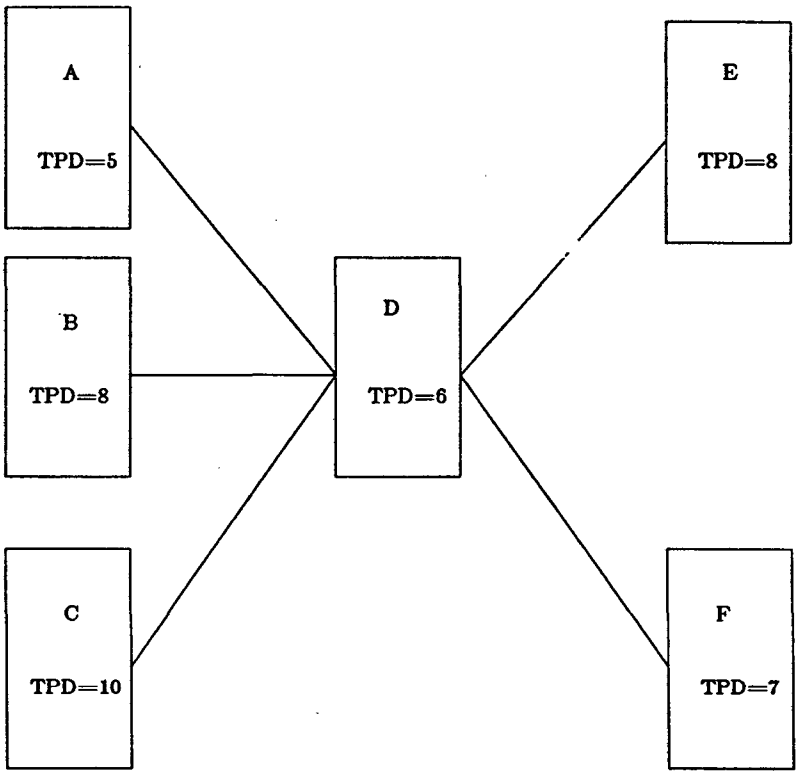
A delay fault exists in a system if a signal can not propagate through a combinational network in time to be clocked into the next set of latches. A delay fault occurs because the delay values of one or more gates in a combinational network exceed specifications.

In designs that are done with statistical timing analysis (such as the method used at Convex), there is a probability that a network may have delay faults yet have NO gates with delay values exceeding specifications! If a path happens to have all gates with high but valid delay values, the aggregate delay of the path may cause it to exceed the clocking interval. This network has a delay fault yet no gate has a delay value exceeding specifications. It should be noted that a design may be free of delay faults yet have many gates with delay values that exceed specifications.

There are two common types of delay fault models that have been studied, the path fault and the gate delay fault. A path fault is a path of the combinational network between input and output latches for which a transition in the specified direction by setting of an input latch does not arrive at the path output in time for proper setting into an output latch. A gate delay fault is a gate defect that results in at least one path fault. A gate delay fault is similar to a DC stuck-at fault in that it is associated with a single gate, a path fault is associated with an entire path.

Whether or not a gate delay is detectable depends upon the path which is being tested. Consider the example shown in figure 3-1 [*85Smit*]. The blocks A-F are all some small pieces of combinational logic. Each block is labeled with its propagation delay value. For the following discussion it is assumed that this circuit is operating within a machine with a 25ns clock cycle time. The table shown in table 3-1 shows propagation delays along each path in the circuit for three different cases. The first case is for no faults, the second case is for a 2ns delay fault at D, the third case is for a 6ns delay fault at D.

## Figure 3–1: Gate Delays




## Table 3-1: Detectability of Varying Levels of Delay Defects

| Good Machine | | Delay Defect +2 at D | | Delay Defect +6 at D | |
|---|---|---|---|---|---|
| Path | Propagation Time | Propagation Time | Detectable? | Propagation Time Time | Detectable? |
| ADF | 18 | 20 | no | 24 | no |
| BDF | 21 | 23 | no | 27 | yes |
| CDF | 23 | 25 | ? | 29 | yes |
| ADE | 19 | 21 | no | 25 | ? |
| BDE | 22 | 24 | no | 28 | yes |
| CDE | 24 | 26 | yes | 30 | yes |

Table 3-1 illustrates that it is generally easier to detect a gate fault produced by a large delay defect than one produced by a small delay defect. The necessity for considering the size of a gate delay defect and the size of all gate delays makes the gate delay fault model less than optimal for direct modeling. Because of delay variation, it can not be assured that a gate delay defect of a certain size will produce a delay fault without prior knowledge of the delay values of all gates in each path through that gate for the particular machine. It is possible that the path with smallest total delay in one machine is not the path with the smallest total delay in another machine. In statistical based timing, the paths that are most likely to fail are those with the greatest delays in the design. These are the paths that should be selected for test. The delay values of all paths in

the design provides a tool for selecting a subset from all paths in the design that should be tested. One approach to test generation for delay faults corresponds to test generation for transition faults with the added complexity of seeking long sensitized paths. Test generation for delay faults is discussed in chapter 4.

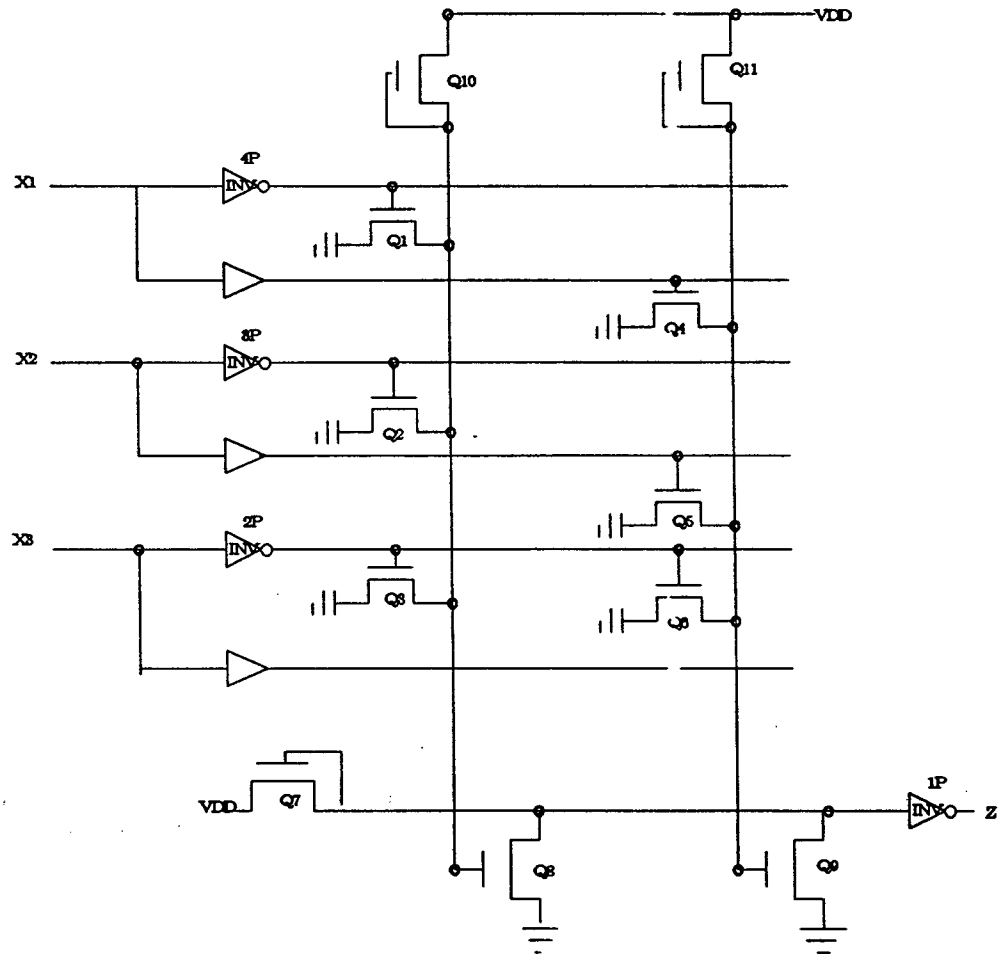
### 3.2.2 Stuck-at Faults

A large number of the physical defects that occur in semiconductor devices can be modeled by stuck at faults. A stuck-at failure is when a node is held constant at a logic level. For binary logic, there are stuck at 1 faults (s-a-1) and stuck-at 0 faults (s-a-0).

### 3.2.3 Crosspoint Faults

Crosspoint faults occur in PALS or PLAS when devices at the crosspoints of the array are located in the wrong place. This can occur with a device that exists at a crosspoint when it shouldn't and when a device does not exist at a crosspoint where it should. A crosspoint failure is shown in figure 3-2 where transistor Q3 is at a crosspoint where it should not exist. Note that the presense of transistor Q3 modifies the equation from $z = x_1x_2 + \overline{x}_1x_2x_3$ to the incorrect equation $z = x_1x_2x_3 + \overline{x}_1x_2x_3$ [*85Fuji*]

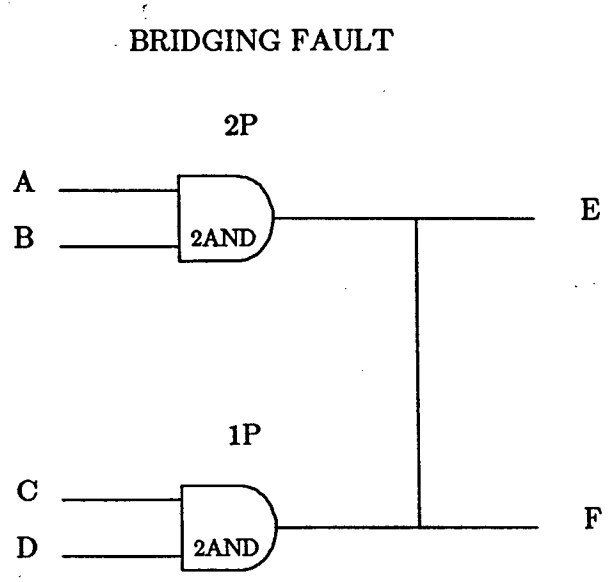
Figure 3–2: Crosspoint Fault in PLA




## 3.2.4 Bridging Faults

Faults in which two or more lines are shorted are called bridging faults. The effect of a bridging fault on the operation of the circuit depends upon circuit technology. A bridging fault on an ECL output causes the outputs to behave as a wired-or. For TTL, a bridging fault causes the outputs to behave in a wired-and fashion. In the figure below, there is a bridging fault at the outputs of gates A and B. This modifies the output equation from $E=AB, F=CD$ to $E=F=ABCD$. This is quite different from normal operation since inputs that are normally not considered in the evaluation of an output now cause the output to be affected.

**Figure 3–3: Bridging Fault in TTL**



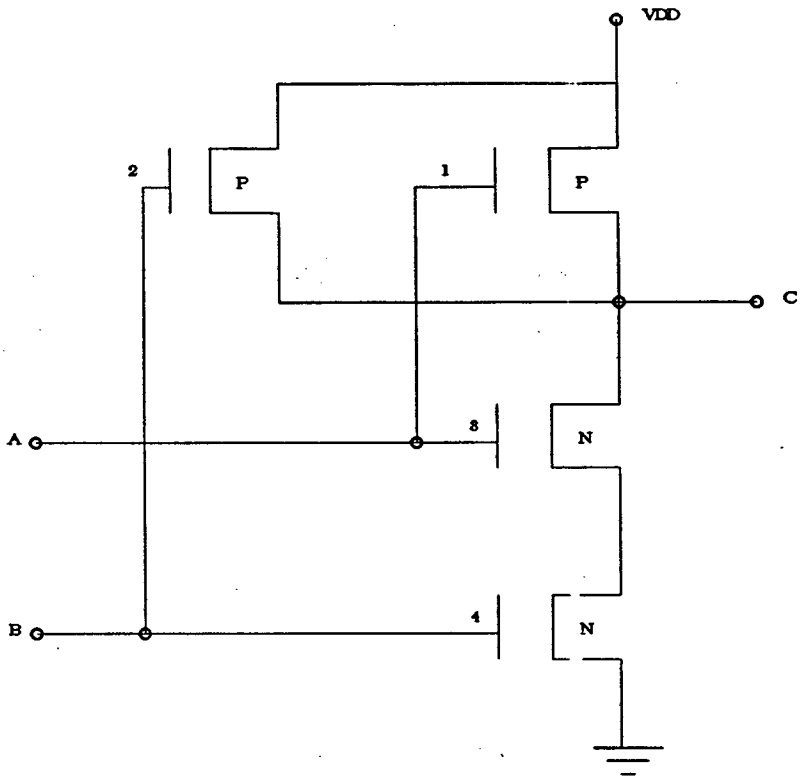

## 3.2.5 Other Fault Models

There are lots of different fault models besides the ones previously listed. Usually, these fault models are special cases for a particular type of device fabricated with a particular technology. Some of these are at least interesting, if not useful and are presented here.

### 3.2.5.1 CMOS Stuck-Open Fault

One of the commonly sited fault models in devices fabricated with CMOS technology is called the stuck open (s-op) fault. This fault describes what can occur in a CMOS circuit where there is a open in one of the input lines. The example shown in figure 3-4 is a CMOS two-input NAND gate [*85Fuji*]. The output C is low if A and B are high. There are 4 possible open faults (numbered 1-4 in figure 3-4). Fault number 1 occurs when there is a missing pull-up resistor on the P-channel A-input pull-up transistor. When this fault is present, a 0 on A and a 1 on B causes the output to go into a high-impedance state and retain the logic value of the previous output state. The length of time the state is retained is determined by the leakage current at the node. This is a case where a fault causes a combinational circuit to become sequential. Tests generated to determine stuck-at faults may miss the presence of this type of fault. The only way to detect this fault is with a pattern sequence that forces the output to 0 (A=1, B=1), and then follows with a pattern to force the output to 1 using the faulted transistor (A=0,B=1). Since the input pull-up is shorted, the output will remain low and the test will detect the fault.

Figure 3–4: Stuck Open CMOS Fault




### 3.2.5.2 Intermittent Faults

Intermittent faults are recurring temporary failures caused by component degradation or a poor system design. There have been several probabilistic fault models developed to describe this type of failure. The treatment of intermittent faults is covered in chapter 6.

### 3.2.5.3 Coupling Faults

Coupling faults are most often used to model failures that occur in memory circuits. The coupling fault is a functional level fault. Two memory cells are assumed to be coupled if a state change in one cell causes a state change in the other cell. The physical proximity of the two cells is usually a factor in the existence of this type of fault.

### 3.2.5.4 Pattern-sensitive Faults

Pattern-sensitive faults are primarily used to describe a functional level failure mode in memory devices. However, pattern-sensitive faults have also been used as a mechanism to explain intermittent faults [*86Cort*]. A fault that alters the state of a memory cell as a result of certain patterns of 0's, 1's, 0 to 1 transitions or 1 to 0 transitions in other cells is called a pattern-sensitive fault. Failure of a read or write operation on a cell when certain data patterns are stored in other cells is also a pattern-sensitive fault.

In order to make use of fault models, it is necessary to understand what physical failure modes need be be present in order to apply a given fault model. It is also necessary to understand which failure mechanisms apply to various semiconductor devices. By understanding the underlying semiconductor technology and failure modes, tests can be developed to detect the most likely failures for any given device. This strategy can be applied across a large system until a system-wide set of tests exist that test for the existence of the most likely faults. This strategy will not cover all possible faults, however it will have a high probability of detecting any fault and should be efficient in terms of test pattern generation.

## 3.3 Semiconductor Reliability and Failure Mechanisms

All electronic devices have a finite lifetime. It is important to understand the lifecycle of the electronic parts that we use in order to effectively develop t sts that can be used to test systems in manufacturing and the field. It is necessary to understand how parts fail over their lifetime, the failure mechanisms that dominate during any particular time period and the failure mode that can be detected by a test.

In order to develop an overall test and serviceability strategy for a system it is necessary to understand some components of reliability engineering. It is also necessary to understand the failure mechanisms that occur in the semiconductor devices that we use. These concepts are discussed below.

### 3.3.0.5 Reliability Engineering Concepts

The failure rate of a device is defined as the reciprocal of its time to failure. $\lambda = \frac{1}{t_i}$ where $t_i$ is the amount of time until the failure. If a set of N devices is put on life test at time t=0, and the devices die at times $t_1 t_2 t_3 \cdots t_N$, the total number of operating hours for the devices is

$$\sum_{r=1}^{N} t_r.$$

The average failure rate $\lambda_{av}$ can now be defined as the number of dead devices divided by the total operating hours.

$$\lambda_{av} = N / \sum_{r=1}^{N} t_r.$$

The mean time to failure , $t_{av}$ is

$$t_{av} = \sum_{r=1}^{N} tr/N.$$

A FIT is a unit used to measure failure rates. One device failure in $10^9$ hours of operation is termed one FIT. One FIT is approximately equal to 125,000 years. It is not possible to wait for such a long time to determine whether all of a large number of similar devices have failed and to record the time of individual failures (after all, we are not on a government contract!). Because of this, accelerated-stress life testing is used to determine reliability. If the degradation mechanism is known, it is often possible to test a small sample from a large population of devices under higher than normal stresses for short periods of time. From this data it is possible to extrapolate the device reliability under normal operation.

Examination of failure data over many years has led to the recognition of several periods of device failure. The first period is the *infant mortality period.* Early in the lifetime of a device there are a large number of failures, due to built-in weaknesses or defects. These early failures are attributed to *infant mortality* and lead to a decreasing failure rate with respect to time. Stress tests and various types of screening procedures are used to eliminate these potential failures.

During the middle period of the device lifetime fewer failures occur. In this region, sometimes called the "random failure region" there is a constant failure rate. This region is sometimes referred to as the useful-life region. The device characteristics are essentially constant and when a failure occurs, it is usually catastrophic.

As a device grows older, it begins to deteriorate at a faster rate. the failure rate increases linearly and many failures occur. This failure rate is called the "wear-out region" and is caused by material degradation due to high temperatures, electric fields and slow chemical reactions. Integrated circuits generally do not reach the wear-out region in normal operation.

Most devices and systems are subjected to a burn-in period. It is the intent of this burn-in period to cause any devices that will fail with an infant mortality type of failure to do so. Accelerated stress testing techniques are used in burn-in. Components and systems are subjected to temperature and voltage stresses in hopes of forcing these early-life failures to occur within a relatively short amount of time. A successful burn-in strategy requires an understanding of failure modes to determine the amount of stress to apply to the component under test.

Reliability screening selects from a collection of devices the ones that have superior reliability and rejects the ones that exhibit early failures. The military has three levels of reliability, class A, B and C. Class A devices should have a failure rate of 0.001%/1000 hours, class B 0.005%/1000 hours and Class C devices have failure rates of 0.05%/1000 hours. Commercial devices, those with the highest failure rates, may be as high as 0.1%/1000 hours. Since the high reliability (mil-spec) devices cost much more than the commercial versions, we need to be prepared for the implications of using lower reliability devices.

### 3.3.0.6 Failure Mechanisms

The physical or chemical process that causes a device to fail is called the failure mechanism. The detection mechanism is called the failure mode. To clarify these concepts with an example: if a gate fails a test because its output is stuck-at-one due to a microcrack in the metallization, the failure mode is stuck at one, the failure mechanism is microcrack in metallization.

Failure mechanisms for bipolar and CMOS integrated circuits can be divided roughly into three groups:

1) Chip-related failures, such as oxide defects, metallization defects and diffusion-related failures.
2) Assembly related problems such as chip mount, wire bonds or package failures.
3) Miscellaneous, undetermined or application-induced failures.

Several different failure mechanisms that fall into the proceeding categories are described below.

### 3.3.0.7 Electromigration

The mass transport of metal atoms by momentum exchange with conducting electrons is called "electromigration". This failure occurs in metal lines with high current densities and high temperatures. Metal atoms move toward the positive end of the conductor, with voids appearing in the other. eventually metal disappears from certain regions and an open-circuit occurs.

### 3.3.0.8 EOS/ESD

Electrical overstress(EOS) is caused by improper application or handling of a device. Electrostatic discharge (ESD) damage occurs when a high static charge is discharged through a device. Both of these failure mechanisms can cause either instantaneous failures or latent damage to a part. The ones that cause latent damage are the most troublesome.

### 3.3.0.9 Contamination

Mobile ions in semiconductor devices is an important infant mortality failure mechanism. These ions can cause shifts in threshold voltages in MOS devices and a change in carrier concentration in bipolar devices. Both device types will show performance degradation as a result.

### 3.3.0.10 Stress Relief Migration

This failure mechanism is another form of metal migration. The migration is due to atoms being transferred from areas of high stress in order to equalize the stresses. Whisker growth, and the formation of hillocks and voids in the metal have been observed due to this failure mechanism. Eventually this will lead to a short or an open in the metal.

### 3.3.0.11 Microcracks

This failure occurs where metallization passes over an oxide step. When the oxide step is high, there is a thinner metal line deposited. These thinner metal lines have a greater probability of failure than normal deposits under high current stress.

### 3.3.0.12 Wire-bond Failures

Gold bonding wire is usually attached to the metallization on the chip through a process of thermocompression. Failure of gold wire bonds to aluminum-metallized chips may be due to the formation of intermetallic compounds that lead to loss of strength and an increase in resistance.

### 3.3.0.13 Diffusion-related Failures

Nonuniform current-flow may occur within a device because of dopant diffusion-related causes. These may affect the base width, the emitter resistivity, the curvature of junctions, and other device parameters.

### 3.3.0.14 Oxide-related Failures

Contamination of oxide, during or after its growth directly affects its dielectric properties, particularly its breakdown strength. Other causes of failures include ion migration in the thermally grown oxide and along its surface.

Charge accumulation at the silicon surface (near a transistor or diode junction) is a cause in degradation of characteristics. This degradation of parameters is a common failure mode for silicon transistors and integrated circuits with aluminum contacts and thermally grown $SIO_2$ surface layers.

For a company such as Convex, we assume that the semiconductor parts that we receive have been tested and are free of any manufacturing induced defects. Therefore it is of primary interest that we understand the types of failures that can occur in these parts after they have been tested at the factory. Since we do not do incoming inspection testing of parts, a part can be damaged any time after it leaves the test handler at the semiconductor manufacturer. Some of these failures can be caused by improper handling, or in-circuit testing here at Convex, or through normal lifecycle failures by means of one of the cumulative failure mechanisms.

## 3.3.1 Manufacturing Defects

A different distribution of failure mechanisms will exist in the Convex manufacturing environment compared to the field environment. Defects caused by the board manufacturing process itself (mis-programmed parts, solder bridges, poor solder joints) do not occur in the field. Also, there will be catastrophic failures induced in semiconductor parts during the Convex manufacturing process. These failures can be caused by static damage, operation of systems at elevated parameters, and in-circuit testing.

### 3.3.1.1 In-Circuit Test and Component Reliability

There has been much written about the possibilities that in-circuit testing can cause a degraded lifetime of components. Most military and telecom companies do not use in-circuit test due to these fears. In-circuit test involves backdriving the components to allow the tester to test them as individual units. During backdrive, high currents in the output stage of the backdriven IC will cause localized heating in the active transistors and in the metal conductors. If this stress is great enough, it will cause an immediate catastrophic failure. If this occurs, the in-circuit tester, or some other step in the test process will detect it. Many failure mechanisms, such as electromigration, are cumulative. The stress of in-circuit test may aid the process causing the failure but will not push the device over the edge. This problem is potentially worse in an environment such as at Convex, where the same board may traverse through the functional test multiple times during it lifetime (once each time the board fails or is upreved).

The literature in this area has not proven conclusively that there is a reduction in reliability of devices that have been exposed to in-circuit testing. It has shown that in-circuit testing does cause catastrophic failures. It has not been proven or disproven that in-circuit test has any measurable contribution to the cumulative failure mechanisms [*86Schn*].

## 3.3.2 Lifecycle Failures

In order to support a field service strategy, it is important that we understand the lifecycle failure modes of the parts that we are using in our systems. Using this information, a strategy can be defined, and actual numbers be given for MTBF, MTTR and system availability. The lifecycle models and test strategies must encompass more than just the individual parts, it must understand the system. "Grown" backplane opens and shorts are as real a problem in our system as electromigration problems in VLSI chips.

An electronic device may be thought of as a system consisting of very many particles. Experience indicates that most failure mechanisms in which transfer or rearrangement of particles occur require that an energy barrier be surmounted. The energy barrier can be modified with temperature, voltage, radiation, or other parameters.

In order to fail under stress, the device has to receive a fixed amount of degradation. Failure rates are calculated by subjecting the devices to accelerated stress (voltage, temperature, radiation) and fitting the failure results into a curve. If each failure mechanism can be isolated (which is not easy to do), then it is possible to come up with accurate curves that will describe the failure rate as a

function of time and the accelerated parameter. This is the concept of accelerated stress life testing.

By analyzing the failure mechanisms and the failure rates of semiconductor devices, it should be possible to determine which failures mechanisms will occur most frequently and what the failure modes will be. Knowing this information allows us to design tests which detect the majority of the failures that will occur. There is no sense in designing a test that will detect a relatively obscure failure mode before the tests for detecting the more common failure modes have been developed. Unfortunately, we have no information on the actual failure mechanisms of devices in Convex systems, we seldom even have the failure mode information. Since this information is not available, the approach that will be taken is to determine the types of failures that will occur for the types of semiconductor devices that will be used in C3, rank them in order of probability of occurrence, and design tests to detect the failures that are the most common.

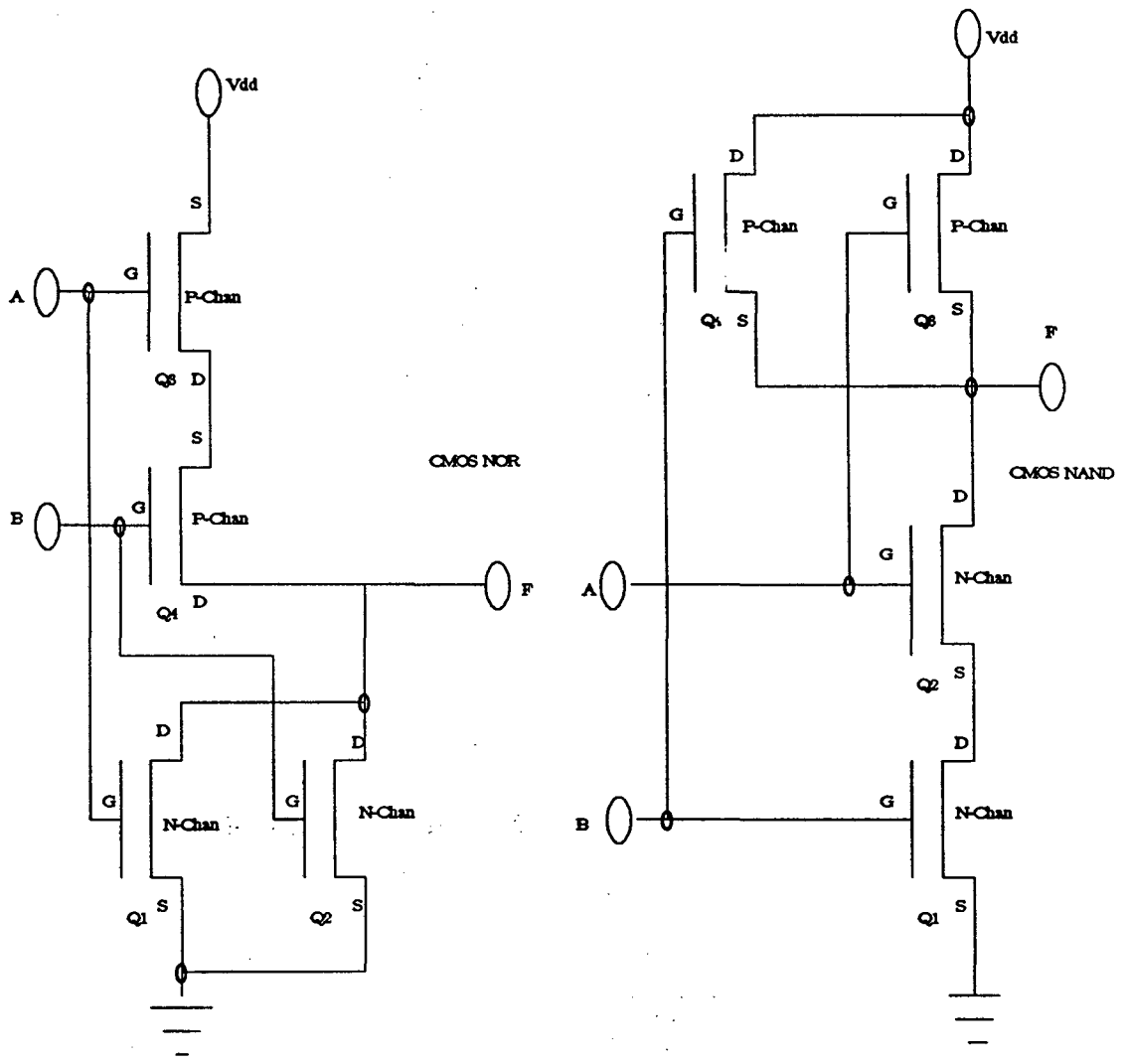
## 3.4 Semiconductor Processes and Related Failure Mechanisms

### 3.4.1 CMOS

Failures in CMOS circuits can be classified into shorts, opens, and circuit degradations. Shorts are due to oxide breakdown and metal bridging, and are caused by static discharge and time-dependent defects. Shorts and opens can also be caused by electromigration. Electromechanical corrosion can also produce shorts and opens. Degradations include threshold voltage shifts caused by ionic contamination, surface-charge spreading and the trapping of hot electrons in the gate oxide.

It has been estimated that at least 75% of the cases of failures are shorts and opens, the other 25% are composed of catastrophic failures and failures where the failure mechanism can not be determined.[*80Gala*] It is instructive to look at the mapping between some of these physical failures in a CMOS NAND and NOR gate and the logical fault produced. The following figures and tables demonstrate the correspondence between the physical failure and the logical fault [*87Aria*].

## Figure 3–5: CMOS NOR and NAND Gates



## Table 3-2: Physical and Logical Faults in CMOS NAND Gate

| Fault | Physical | Logical |
|---|---|---|
| 1 | Short G-D Q1 | B s-a-$\overline{B}$ |
| 2 | Short G-S Q1 | B s-a-0 |
| 3 | Short D-S Q1 | B s-a-1 |
| 4 | Short G-D Q2 | F s-a-A |
| 5 | Short G-S Q2 | F s-a-1 |
| 6 | Short D-S Q2 | A s-a-1 |
| 7 | Short G-D Q3 | F s-a-A |
| 8 | Short G-S Q3 | A s-a-1 |
| 9 | Short D-S Q3 | F s-a-1 |
| 10 | Short G-D Q4 | F s-a-B |
| 11 | Short G-S Q4 | B s-a-1 |
| 12 | Short D-S Q4 | F s-a- |
| 13 | Open G Q1 | Vdd s- )p |
| 14 | Open D Q1 | Vdd s-op |
| 15 | Open S Q1 | Vdd s-op |
| 16 | Open G Q2 | Vdd s-op |
| 17 | Open D Q2 | Vdd s-op |
| 18 | Open S Q2 | Vdd s-op |
| 19 | Open G Q3 | A s-op |
| 20 | Open D Q3 | A s-op |
| 21 | Open S Q3 | A s-op |
| 22 | Open G Q4 | B s-op |
| 23 | Open D Q4 | B s-op |
| 24 | Open S Q4 | B s-op |
| 25 | Input A s-a-0 | A s-a-0 |
| 26 | Output F s-a-0 | F s-a-0 |

## Table 3-3: Physical and Logical Faults in CMOS NOR Gate

| Fault | Physical | Logical |
|---|---|---|
| 1 | Short G-D Q1 | A s-a-$\overline{A}$ |
| 2 | Short G-S Q1 | A s-a-0 |
| 3 | Short D-S Q1 | F s-a-0 |
| 4 | Short G-D Q2 | B s-a-$\overline{B}$ |
| 5 | Short G-S Q2 | B s-a-0 |
| 6 | Short D-S Q2 | F s-a-0 |
| 7 | Short G-D Q3 | F s-a-0 |
| 8 | Short G-S Q3 | A s-a-1 |
| 9 | Short D-S A3 | No effect |
| 10 | Short G-D Q4 | B s-a-$\overline{B}$ |
| 11 | Short G-S Q4 | F s-a-0 |
| 12 | Short D-S Q4 | No effect |
| 13 | Open G Q1 | A s-op |
| 14 | Open D Q1 | A s-op |
| 15 | Open S Q1 | A s-op |
| 16 | Open G Q2 | B s-op |
| 17 | Open D Q2 | B s-op |
| 18 | Open S Q2 | B s-op |
| 19 | Open G Q3 | Vdd s-op |
| 20 | Open D Q3 | Vdd s-op |
| 21 | Open S Q3 | Vdd s-op |
| 22 | Open G Q4 | Vdd s-op |
| 23 | Open D Q4 | Vdd s-op |
| 24 | Open S Q4 | Vdd s-op |
| 25 | Input B s-a-1 | B s-a-1 |
| 26 | Output F s-a-1 | F s-a-1 |

As can be observed in the previous two examples, most faults can be modeled as stuck-at (s-a) or stuck-open (s-op) faults. The gate-to-drain shorts can not be modeled as either of these types of faults because high voltage and ground might exist at the output. The net result may be an intermediate voltage level. Consequently, gate-to-drain shorts can not be modeled using any existing classical fault models. The short couples two different gates and the output depends upon the nature of the short. If there is a short between a low driven input and Vdd, then low voltage will dominate at the output line, if there is a short between a high-driven input and the ground line, then a weak high voltage will probably exist at the output.

### 3.4.2 Bipolar Devices

Both ECL and TTL technologies come under this category. The available research in this area indicates that most of the failure mechanisms that effect CMOS also effect these devices. Because of the lack of the dielectric used to manufacture the gates in MOS devices, bipolar devices do not exhibit the same failure mechanisms. In particular, the stuck-op fault model is not related to any physical failure mechanism in bipolar devices. The failure mechanisms for Bipolar devices listed in order of occurrence [*87Amer*] are:

- EOS/ESD.
- Contamination.
- Corrosion.
- Electromigration.
- Stress relief migration.
- Diffusion defects.
- Microcracks.
- Insulating oxide breakdown.
- Radiation.
-

### 3.4.3 GaAs

Because of the newness of this technology for fabricating integrated circuits, there is not much available research on failure mechanisms that are particular to GaAs semiconductor devices. This is unfortunate for developing a test strategy for these components. However, several failure mechanisms inherent to the semiconductor process should still apply. The probabilities of these failure mechanisms will no doubt be different than in silicon based devices.

### 3.4.4 Summary of Failure Mechanisms

The following table has been adapted from [*87Fant*] and classifies the failure mechanisms of VLSI devices. The table shows the location of the defect within the device, the defect, the failure mechanism, the failure mode and the time (or place) where the failure occurs. It is instructive to understand the classification of device failures by failure mechanism. The basic process is described in [*87Fant*]. It is by no means a trivial process. Although failure mechanism information would be helpful in determining causes and appropriate remedies we as a company can probably not afford to gather this information for each failed device.

## Table 3-4: VLSI Failure Mechanisms

| LOCATION | DEFECT | FAILURE MECHANISM | FAILURE MODE | WHEN* |
|---|---|---|---|---|
| Substrate | Crystal Defect Near Junction | High generation-recombination | Refresh time degradation in RAMS | P/I |
| | Crystal Defect Near Junction | Low-resistance paths | Leakage/short circuit | P/I |
| | Crystal Defect Near Junction | Secondary breakdown | Short/open circuit | F |
| | None | Alpha particles | Soft errors | F |
| Thin Oxides | Traps and charges at $SI-SIO_2$ interface | | Leakage, Instabilities/Degradation | P |
| | Oxide Traps | Hot carrier injection and trapping | MOS Threshold and transconductance degradation | F |
| Thin and Thick Oxides | None | High E-field breakd· wn(ESD) | leakage/short circuit | F/I/P |
| | Pin-holes | Low E-field breakdown | Leakage/short circuit | P/I |
| | Other defects/contaminants | Time dependent breakdown | Leakage/short circuit | I/F |
| | Contaminants | Surface depletion/inversion | Leakage/characteristic instabilities and degradation | P/I/F |
| Contacts | Crystal defects/ Non-saturated Al | metal-si interdiffusion | Leakage/short circuit | P/F |
| | Oxidized surface | | Open circuit/floating conductors | P/I/F |
| | None | Electromigration | Junction leakage / Short circuit / Open metal | F |
| Conductors | Si nodules/ High setups/Scratches | Joule melting/Electromigration | Open circuit | P/I/F |
| | None | Electromigration | Open circuit | F |
| | None | Overstress | Short circuit/Open circuit | I/F |
| | Contaminants | Corrosion | Open circuit(Al)/Short circuit(Au) | F |
| Multilayer Conductors | Al hillocks | Stress relief/Electromigration | short circuit | P/F |
| Chip-package connection | Poor wire bond<br>Excess wire binding | Thermal fatigue/Mechanical stress<br>Oxide under pad break | Open circuit<br>Short circuit | I/F<br>P/I |
| | Excess wire binding | Au-Al interdiffusion | Open circuit | F |
| | Thermal coefficient mismatch between wires and package | Thermal fatique | Open circuit | F |
| | Shallow angle of wires with substrate | | short circuit | P |
| | Defective die-attach | Poor thermal conductivity | Characteristic degradation | I/F |
| | Defective die-attach | Thermal runaway | Short/open circuit | F |

*P: Semiconductor Device Production
I: Incoming inspection
F: Field Failures

Although all of the devices used in our systems can not be classified as VLSI devices, the failure mechanisms apply for all silicon-based semiconductor devices, although the frequency of occurrence of various defects is directly related to the circuit geometries. Many of these defect mechanisms will certainly occur in GaAs devices, in particular the ones relating to contacts, conductors, and chip-package connections.

Some of these failure mechanisms occur in MOS types circuits and not in bipolar or GaAs circuits, in particular those located in the thin and thick oxides. These failure mechanisms would therefore apply to CMOS gate arrays of the C3 and the BiCMOS gate arrays of Javelin. It is of interest to note that a significant portion of these failures can occur in the field (77%). It is also significant to note that at least 80% of these failures could be detected by stuck-at fault tests, the other 20% *may* be detected by tests for stuck-at, delay, or stuck-op faults. Package failure rates, particularly failures in solder and plug-in connections, far exceed the failure rates of logic gates. From this analysis, it appears that the first order of

attack should be stuck-at types of faults, with secondary emphasis placed on stuck-op (for CMOS), and delay faults. The actual test strategies are described in chapter 7.

# 4

# Test Pattern Generation Methods and Algorithms

This chapter discusses the specifics of automatic test pattern generation schemes for gate-level faults in combinational and sequential logic circuits. Included is a discussion of various algorithms for testing memory circuits. A test for logic circuits consists of one or more patterns applied at the primary inputs of the circuit under test. Results are observed at the primary outputs of the circuit. A test detects a fault if the output observed at the primary output is different from the output that would be observed in a fault-free circuit.

The primary goal associated with pattern generation is to generate a set of test patterns that assure a high fault coverage for the faults under consideration. Secondary goals are efficient use of computer resources for generating the patterns and a compact set of patterns to minimize test execution time and reduce storage requirements.

## 4.1 Stuck-at Fault Algorithms for Combinational Circuits

As was described in Chapter 3, a large number of physical defects in semiconductors can be modeled as a stuck-at fault in a logic circuit. This section will examine several different types of pattern generators that generate tests for stuck-at faults. A detailed discussion of these algorithms is presented in appendix C.

Most stuck-at fault test pattern generators manipulate a topological gate-level description of a circuit. Also, almost all test pattern generators assume only a single fault exists at a time in the circuit under test. For a circuit with N lines there are at most $2*N$ possible single stuck-at faults. For multiple faults, the number of possible faults increases to $3^{(N-1)}$. A circuit with 100 lines would contain approximately $5*10^{47}$ faults! Luckily it has been shown that test patterns that have a large single fault coverage also have large multiple fault coverage.

The history of test pattern generators for stuck-at faults goes back almost 30 years. Some of the algorithms have survived the test of time and have been implemented in proprietary test pattern generators as well as commercial ones. Most test pattern generation algorithms in use today have been developed from ideas first presented in the D-algorithm, the PODEM algorithm or the FAN algorithm. Each of these algorithms is based upon some path sensitization approach. In order to understand these algorithms it is important to understand the basic methods involved in the process of a path-sensitizing test pattern generator. The following four steps are central to most of the gate-level test generators.

1) *Fault Excitation*: This involves establishing a logic value at the fault location opposite to that produced by the fault.
2) *Fault Effect Propagation*: This involves moving the effect of the fault closer to a primary output.
3) *Line-value Justification*: The process of justifying an internal node value by backing up towards the primary input. For example having the output of an OR gate at 1 is justified by setting either input to a 1.
4) *Line-value Implication*: The process of working in the forward direction to assign values

implied by previously assigned values.

Both implication and justification can run into consistency problems when an implied value has already been set to another value or when a value being justified runs into a dead-end. Usually these two steps (2 and 3) involve decisions. There are several paths to propagate a fault towards an output, there are also several ways to justify a given line. When an inconsistency occurs, an effective algorithm must be able to backtrack through previous decisions and remember the deferred choices. The differences in the various algorithms have to do with the heuristics used for making a choice or for selecting a deferred choice. The most efficient algorithms are the ones that have to backtrack the least.

### 4.1.1 D-Algorithm

The D-algorithm was first described in a paper in 1967 [*67Roth*]. It was shown that this algorithm would create a test for any non-redundant stuck-at fault. It was the first published algorithm that could do this. It is important to understand the D-algorithm due to the fact that many other papers published in this field make reference to this algorithm. The references are used to help describe new algorithms since a large deal of the test generation vocabulary was developed along with the D-algorithm. The performance of the D-algorithm is used as a basis upon which other test pattern generation algorithms justify their existence.

A study done by [*75Ibar*] shows that test generation for combinational circuits belongs to the class of problems called NP-complete, strongly suggesting that no test-generation algorithm with a polynomial time complexity is likely to exist. By examining the details of the D-algorithm it can be seen how time consuming it would be for large combinational circuits where a large amount of backtracking were required. The major areas of improving this algorithm have been in the areas of better decision making to reduce the number of backtracks. However, for all of these algorithms, when a test does not exist, the entire search space must be exhausted before the non-existence of a test can be determined. Because of this, these ATPGs are usually run with a timeout limit to abort searching for a given fault after a certain amount of CPU run time. Thus, fault coverage will drop because of a poor/inefficient algorithm. Even though these algorithms can in principle find a pattern for a fault if a pattern exists, practically they can not.

### 4.1.2 PODEM

A test-generation algorithm called PODEM (Path Oriented DEcision Making) was reported in [*81Goel*]. This algorithm was shown to be more efficient than the D-algorithm. Like the D-algorithm, the PODEM algorithm is complete in the sense that it will generate a test for a stuck fault if a test exists.

Several comparisons have been made between DALG and PODEM. Actual performance varies depending upon the actual circuit that is being processed. PODEM has been shown to have from 2-35X the performance (measured in processing time) ove. DALG. The greatest performance increases occur in circuits where DALG is forced to do much backtracking.

The performance of the PODEM algorithm depends greatly on the initial assignments of the primary inputs. Several researchers have employed various levels of heuristics to help guide the PODEM algorithm through correct assignments of primary inputs. Some of these include [*88Calh*] and [*85Brgl*]. This technique has proven to increase the performance of PODEM substantially.

### 4.1.3 FAN

Another test pattern generation algorithm was described in [*83Fuji*] called FAN (fan-out-oriented test generation algorithm). This algorithm has proven to be more efficient than PODEM. The FAN algorithm uses some advanced heuristics to reduce the amount of backtracking required during the test generation algorithm.

## 4.2 Testability Analysis

Testability is the property of a circuit that makes it easy to test. The primary objective of testability analysis is to detect potential testability problems before they occur. To be of any use, it is necessary for the testability algorithms to be more efficient than the actual process of test generation and fault simulation.

The basic idea behind testability analysis programs is to assign a controllability and observability number to every node in a circuit. These values correspond to the effort to control or observe each node, the higher the value, the greater the effort. Since test pattern generation is the process of controlling and observing nodes, nodes with high values are more difficult to generate tests for than others. By looking at the values throughout the circuit, the relative testability of the circuit can be determined. Although several of these programs exist, their actual usage is limited. Some of these algorithms are incorporated into some of the more advanced test pattern generator programs. Examples of these types of algorithms are given in [*80Gold*] and [*85Seth*]. The details will not be given here except to say that they use testability analysis algorithms to preprocess a circuit to determine a testability figure of merit for each node in the circuit. Testability for paths are then calculated during run time, these values are used to help guide the program in the path selection process to proceed down paths that are the most testable. This reduces the amount of backtracking required by these algorithms since they will tend to make a correct choice the first try.

## 4.3 Real Implementations

Deterministic test pattern generation is very expensive. Most all test pattern generation schemes utilize a fault simulator to determine which faults are covered by any given test pattern. The usual approach is to generate a fault list for a circuit then apply a random or heuristically determined set of patterns to the inputs and perform fault simulation. Faults that are uncovered are removed from the fault list. When the coverage reaches around 80% or when a certain number of patterns have been generated, the pattern generators implement one of the deterministic test pattern generation schemes to develop tests for the remaining faults.

## 4.4 Comparison of Stuck-at Fault Algorithms

A paper was presented at ISCAS in 1985 [*85Brgl*] that included a set of benchmarks, these benchmarks have come to be known as the ISCAS '85 benchmarks. These circuits vary in size and complexity and have been used as benchmarks for test pattern generation, routing and simulation. The table below summarizes the circuits used in the benchmark.

## Table 4-1: ISCAS Benchmark Circuits

| Circuit Name | Function | Total Gates | Inputs | Outputs |
|---|---|---|---|---|
| C432 | Priority Decoder | 160 | 36 | 7 |
| C499 | ECC | 202 | 41 | 32 |
| C880 | ALU/Control | 383 | 60 | 26 |
| C1355 | ECC | 546 | 41 | 32 |
| C1908 | ECC | 880 | 33 | 25 |
| C2670 | ALU | 1193 | 233 | 140 |
| C3540 | ALU | 1669 | 50 | 22 |
| C5315 | ALU/Selector | 2307 | 178 | 123 |
| C6288 | 16bit Multiplier | 2406 | 32 | 32 |
| C7522 | ALU | 3512 | 207 | 108 |

Note: faults are collapsed faults based on fault equivalencing.

Table 4-2 compares the performance of several different gate level stuck-at test pattern generator approaches.

## Table 4-2: Performance Comparison of Stuck-at Fault Pattern Generators

| Circuit | AIDSTG | | | PODEM-10 | | | FAN-10 | | | SOCRATES | | | DALGSPLIT | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Coverage | Patterns | Time[1] | Coverage | Patterns | Time[2] | Coverage | Patterns | Time[2] | Coverage | Patterns | Time[3] | Coverage | Patterns | Time[4] |
| C432 | 99.09% | 55 | 70 | 91.8% | 62 | 1.9 | 93.7% | 75 | 1.5 | 99.24% | 58 | 8.7 | 98.47% | 516 | 88.7 |
| C499 | 99.29% | 62 | 101 | 99.4% | 122 | 7.9 | 97.2% | 115 | 12.6 | 98.94% | 68 | 8.1 | 98.9% | 750 | 58.5 |
| C880 | 100% | 71 | 107 | 100% | 88 | 1.8 | 100% | 70 | 1.8 | 100% | 60 | 5.7 | 100% | 942 | 29.5 |
| C1355 | 99.04% | 109 | 301 | 99.8% | 141 | 9.1 | 97.8% | 117 | 9.0 | 99.49% | 88 | 21.9 | 91.85% | 1458 | 451.4 |
| C1908 | 99.59% | 132 | 533 | 99.8% | 170 | 9.2 | 99.8% | 152 | 9.4 | 99.82% | 125 | 53.1 | 95.26% | 1760 | 576.9 |
| C2670 | 99.25% | 170 | 809 | 94.6% | 157 | 13.8 | 95.7% | 165 | 10.6 | 95.49% | 127 | 69.8 | 95.3% | 2618 | 830.3 |
| C3540 | 95.90% | 183 | 1898 | 95.8% | 209 | 27.6 | 95.8% | 208 | 21.8 | 98.00% | 171 | 62.0 | 94.2% | 3229 | 722.6 |
| C5315 | 99.21% | 155 | 934 | 98.8% | 188 | 24.0 | 98.9% | 202 | 20.1 | 98.90% | 148 | 85.8 | 98.8% | 5291 | 749.4 |
| C6288 | 99.45% | 49 | 802 | 99.8% | 88 | 68.7 | 99.4% | 26 | 67.7 | 99.86% | 88 | 114.8 | 98.87% | 7479 | 4559.8 |
| C7552 | 98.26% | 249 | 2121 | 99.8% | 288 | 78.8 | 98.2% | 294 | 80.8 | 98.26% | 231 | 294.8 | 95.9% | 7229 | 2782.84 |

1: Time in seconds measured on a SUN workstation (68010)
2: Time in seconds measured on NEC ACOS-1000 (15 mips)
3: Time in seconds measured on an Apollo DN-3000
4: Time in seconds measured on a Convex C-1 (4Mips)

AIDSTG is a commercial product from Gateway Design Automation company. The actual algorithms used inside this program are not known.

PODEM-10 is a FORTRAN implementation of the PODEM algorithm with the backtrack limit set to 10. FAN-10 is an implementation of the FAN algorithm with the backtrack limit set to 10. Either of these algorithms could generate tests for all of the detectable faults given enough time.

Faults that are not tested due to excessive backtracks are called aborted faults. Both of these results are based on deterministically generating test patterns then fault grading and removing all faults detected by each subsequent test pattern.

SOCRATES is a system developed at Siemens and is described in [*87Sarf*] it uses a modified FAN approach along with improved heuristics. Its performance is quite impressive considering the host machine.

The DALG-Split is an implementation of the D-algorithm that uses a 9-V logic system [*88Chen*]. Unlike the other algorithms in this chart, this is the only one where a test is generated for each target fault with no intervening fault simulation step to remove previously detected faults. Thus the large number of patterns and large amount of time required for test generation. From looking at the results of these programs, several conclusions can be made.

1) It is possible to generate high fault coverage patterns in reasonable amounts of computer time if the circuits are kept to a reasonable size.
2) The best results are seen with a combination of random and deterministic pattern generation.
3) It is essential to have a fault simulator integrated with a pattern generator.
4) FAN and its descendants have the best performance for these benchmarks.

## 4.5 Stuck-at Test Generation for Sequential Circuits

Test generation for sequential circuits is much more difficult than for combinatorial circuits. Tests for sequential circuits require a series of vectors. In addition, it is necessary to force the sequential circuit into a known state to be able to perform the test. For asynchronous circuits, the problem is even more difficult due to races, hazards or oscillations that are induced during the fault propagation process. No one has solved the problem of automatic test pattern generation for sequential circuits in the general case. A theoretically complete solution that would terminate only after excessive amounts of computer time is not of much use. Also, approaches that are dependent upon the recognition of special cases are not very practical. Both of these approaches have been proposed and implemented. It can be said that the computation requirements to generate adequate tests for sequential circuits is orders of magnitude more than for combinational circuits. We should avoid having to perform ATPG for sequential circuits if at all possible by employing one of the design for test approaches presented in chapter 5.

## 4.6 Delay Fault Algorithms

Test generation for delay faults has been performed using the path delay and the gate delay fault models described in chapter 3. The gate delay fault model throws some complexities into the choosing of not only the location of the faults but the magnitude of the faults. In gate delay fault simulation an excessive delay of 1 nanosecond at some point in the circuit is not the same as a delay of 2 nanoseconds at the same point. One may be detectable, the other not. Either of these faults may be considered according to the underlying reasons for the existence of one of the defects that may cause these faults. This makes fault coverage calculation for delay tests based on the gate delay fault very difficult to determine

With the path delay fault model, a path is declared free of delay faults if and only if it propagates a signal within the operational system clock interval regardless of delay defects on the propagation path. This approach makes generating meaningful fault coverage estimations easier. However, the number of paths can grow exponentially with the number of gates in a circuit, whereas the number of gate delays grows linearly with the circuit. This causes path delay fault test generation and grading to be a very expensive process.

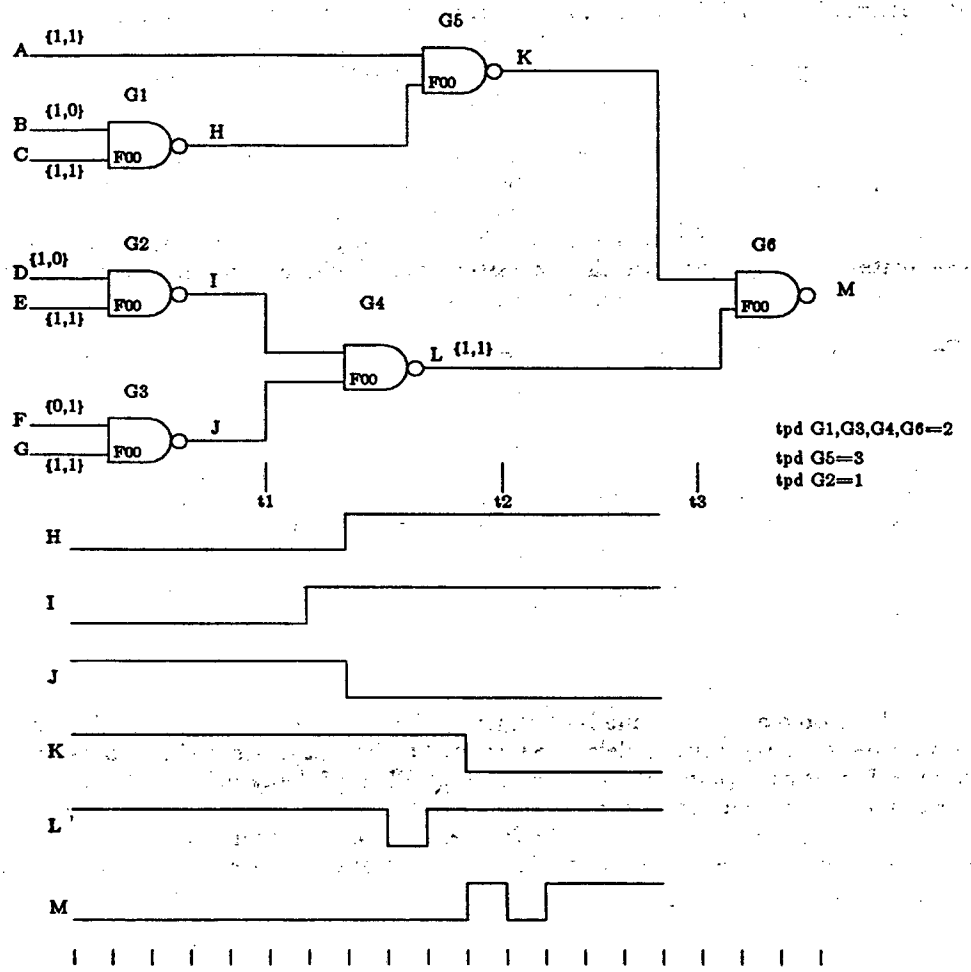
Whichever approach is taken, a delay fault is tested by a pattern pair. The basic approach is to make a sensitized path from input to output. A gate delay fault is sometimes referred to as a transition fault. A test for a transition fault must create the appropriate transition at the point of the fault and propagate it to the output. This requires two patterns, the initialization pattern and the transition propagation pattern. The transition propagation pattern is identical to the pattern that detects the corresponding DC stuck at fault. The transition initialization pattern is the pattern that detect the inverse of the corresponding DC fault.

Figure 4-1 shows a simple circuit made up of NAND gates. Assume that it is desired to test path BHKM for a delay fault. In order to guarantee that this path is free of delay faults, it is necessary to show that both a rising transition and a falling transition originating at input B and propagating along the path can reach point M before the clock. The transition generated at an input will propagate to the output, but this transition may include a glitch that will invalidate the test. Closer examination of the process will show why.

The nominal propagation delay of each of these gates is 2ns. For this particular instance of the circuit there is a delay defect at gate 5 changing it to have tpd=3ns, there is also a fast gate at gate 2 causing it to have a tpd of 1ns. Suppose the two patterns (vectors time sequenced left to right) {1,1,1,1,1,0,1} and {1,0,1,0,1,1,1} were applied to inputs A-G respectively. As can be seen in the timing diagram in figure 4-1, there is a glitch on line L caused by the fast propagation delay at gate 2 that propagates to the output. If the output strobe is set at 6ns, it will see the value at M to be a 1 even though a gate in the path under test has a delay defect. This is a case of a delay test improperly declaring a faulty path good. This example illustrates the difficulty of actually generating useful delay tests. Just because a test generates a rising and falling transition at each output based on each input says nothing about the actual paths tested. Even if the test was improved to propagate a transition on each path does not say anything about uncovering actual delay defects on these paths.

In order to avoid the problem of occasionally declaring a bad circuit as being good, it is necessary to perform a hazard-free delay test. A delay test is said to be hazard-free if no spikes can occur on the tested path during the application of the test regardless of delay values. Another term for this type of test is a robust delay test. A robust delay test is a test that detects an excessive path delay for the path passing through the fault site *independent* of the other path delays in a circuit. A two-pattern test should be developed such that they initiate a desired state transition at the input to the path of interest and propagate it along the path such that the state transitions at the outputs of on-path gates occur only after the desired transition at its on-path inputs has propagated to the output.

## Figure 4–1: Delay Fault Example




Methods have been proposed in [*85Smit*] and [*87Redd*] to generate tests for path delay faults. Since the tests were aimed at path delay faults the following rules apply:

1) Paths are tested for path faults no matter how large or small the delays of individual gates are.
2) Whether the path fault is caused by a single gate or by multiple gates is irrelevant.
3) An improper value set into the latch at the output of a tested path does not assure that the path contains a path fault because numerous simultaneously tested paths may end at the same latch.
4) Testability can be determined without delay simulation.

These algorithms attempt to find robust tests for paths. This is done by preassigning values down a given path and then implementing a justification process similar to PODEM. One of these papers [*Redd*] detailed results of trying to find robust delay tests for all of the maximally long paths in the ISCAS '85 benchmark circuits. The results are shown in table 4-3.

**Table 4-3: Delay Faults in ISCAS Circuits**

| Circuit | Number of Max Paths | Tested | Dropped | No Test |
|---|---|---|---|---|
| c432 | 2916 | 0 | 0 | 200 |
| c499 | 12288 | 0 | 200 | 0 |
| c880 | 9 | 6 | 0 | 3 |
| c1355 | >10000 | 0 | 0 | 200 |
| c1908 | 33 | 0 | 0 | 33 |
| c2670 | 16 | 0 | 0 | 16 |
| c3540 | 12 | 0 | 0 | 12 |
| c5315 | 16 | 0 | 0 | 16 |
| c6288 | >10000 | 0 | 0 | 200 |
| c7522 | 2 | 0 | 0 | 2 |

For the circuits with more than 200 maximum delay paths, the number was set to 200. The numbers in the dropped column represent cases where the test generator aborted due to excessive backtracking. It can be seen that very few paths can be tested with robust tests. For most of these cases, the reason for non-existence of a test was due to the lack of the ability to sensitize the chosen path. Even though a given circuit path is not sensitizable, excessive delays along such paths may still cause erroneous outputs for some circuit input transitions. A path can be considered tested with a non-robust pattern if all other paths that are convergent with the path under test have been tested by robust delay tests.

As can be seen from the proceeding discussion, testing for delay faults in a circuit is not straight forward, it requires much more thought than simply wiggling lines on the inputs and outputs of a circuit.

## 4.7 Other Types of Fault Detection Algorithms

As detailed in chapter 3, there are several types of faults that can not be modeled with the stuck-at or the delay fault. Included are the CMOS stuck-op, bridging, and crosspoint faults. This section describes the options available for generating patterns for these types of faults.

### 4.7.1 CMOS Stuck-op

A paper has been written [87Aria] which details how to handle stuck-op faults through a circuit transformation process that when used with a stuck-at type of fault simulator will generate test patterns that detect the presence of these types of faults. Thus having a test pattern generator for stuck-at faults can cover the stuck-op fault condition.

### 4.7.2 Tests for PLAs

Tests for PLAs need to include the ability to detect the crosspoint fault that was described in chapter 3. Thus the test for a PLA should have the crosspoint patterns generated first. Next these patterns should be fault-graded to determine which stuck-at faults are not covered. Finally a stuck-at fault generator should generate the remaining tests.

The test for a PLA crosspoint fault is the verification of every crosspoint element in the array. The procedure is to select or de-select a word line and then sensitize a path to each output. A pattern generator that can correctly handle PLAs must understand more information than a topological description of the decomposed circuit, it must understand the makeup of the PLA. Crosspoint faults are most common after programming the device, however crosspoint faults can also occur at a later time when a previously blown device reappears in the crosspoint.

## 4.8 Fault Coverage

The concept of fault coverage has been discussed throughout this chapter. The definition of fault coverage is the number of modeled faults that are detected by the test pattern set. In order to determine fault coverage, a fault grader or fault simulator is necessary. A fault simulator will take a set of faults and a set of input vectors and find out which faults are detected by the input vectors. Most work in fault simulators has been directed towards detecting stuck-at faults.

There are three major architectures of fault simulators, the parallel fault simulator [*75Thom*], the deductive fault simulator [*72Arms*] and the concurrent fault simulator [*74Ulri*]. Each of these approaches has performance variations depending upon the topology of the circuit being simulated.

The parallel fault simulator gets its name from the fact that it simulates the state of the good machine and the state of N-1 other machines at the same time. N is typically the word width of the computer that is running the simulation. The operations on the primatives in the simulator usually can be done in one or two instructions on the computer, hence the simulator processes a large number of gates and faults in a short amount of time. However, this fault simulator always evaluates every gate, even when it is not necessary. This causes there to be many more evaluations than necessary. As the size of the circuit being simulated increases, the efficiency of this algorithm decreases because of the ratio of active to inactive gates decreasing.

The deductive fault simulator was created to reduce the total number of passes through the circuit. This is done by attaching a fault list to each node and processing the circuit in one pass. This is an improvement over the parallel fault simulator. Although the actual rate of gate evaluations is much lower, there are less evaluations to perform.

The concurrent fault simulator takes advantage of the fact that the faulted circuit differs only slightly from the non-faulted circuit, thus only differences in the circuits are propagated. Like the deductive fault simulator, the concurrent simulator reduces the number of evaluations that need to be performed.

There are two basic methods that can be used in fault grading. The first method is the deterministic method. In this method all possible faults of the selected type are simulated. The coverage is the ratio of detected to undetected faults. The second method is statistical of which there are two common models. The first picks a random sample of the total number of faults and simulates them. If the sample space is assumed uniformly distributed and if the sample size is large enough to be relevant, then the fault coverage numbers can be inferred. The second model counts transitions on input/output pins in the circuit and tries to infer the actual fault coverage based on transitions. This method is the least reliable. Although it can point to areas that are not exercised during the application of tests, it gives no true indication if the areas exercised are in fact testing faults.

The deterministic fault simulation method is the only method that can be used in conjunction with a pattern generator since the pattern generator must have the exact location and types of faults input. This input allows the pattern generator to reduce its search space.

## 4.9 Tests for Memory Circuits

As stated in chapter 3, memory circuits are subject to different faults from other logic circuits, in particular pattern sensitive faults. Most tests for memory circuits are aimed at the functional level failures of the memory device. This is in contrast to the tests to detect gate-level failures described previously in this chapter. Functionally a RAM consists of memory cells, an address decoder, a memory address register, memory data register, and read/write logic (sense amps, write drivers, etc). There are three fault models of practical interest in testing RAMS:

1) Stuck-at faults.

2) Coupling faults.
3) Pattern sensitive faults.

These fault models were described in chapter 3. Testing for stuck-at faults is the easiest. Testing for coupling faults is more difficult but such tests will also detect all stuck-at faults. Testing for pattern sensitive faults is very expensive and complicated.

Several algorithms exist for the testing of stuck-at faults throughout a RAM chip. The table below [*83Abad*] compares these algorithms, the faults they detect and their complexity ( n is the number of cells in the device).

## Table 4-4: Stuck-at Fault Memory Algorithms

| Algorithm | Complexity | Stuck at Fault Coverage |
|---|---|---|
| *MSCAN* | 4n | Covers memory array faults, doesn't cover decoder or MAR faults. |
| *ATS* | 4n | Covers all for RAMS with wired-OR logic behavior and noncreative decoder design. |
| *Complemented ATS* | 4n | Same as ATS but for RAMs with wired-AND logic behavior. |
| *ATS+* | 13n/3 | Covers all for RAMS with either wired logic behavior and noncreative decoder design |
| *MATS* | 4n | Covers all for RAMS with wired-OR logic behavior and arbitrary decoder design. |
| *Complemented MATS* | 4n | Same as MATS for RAMS with wired-AND logic behavior. |
| *MATS+* | 5n-2 | Covers all for either wired logic behavior and arbitrary decoder design. |

Noncreative decoder design is one where a stuck at fault does not create a new address to be accessed without also accessing the desired address.

From the table it is seen that the most flexible algorithm for the detection of stuck-at faults in memories is the MATS+ algorithm. It is important to have this type of test in a memory system. It is also important to define the set of coupling faults and pattern sensitive faults that are of interest. Coupling faults in the strictest sense are most likely to occur within a single RAM chip. It is important to treat the memory system as a collection of memory chips and know where to test for coupling faults and where not to test for coupling faults.

There have been several algorithms for testing for coupling faults, the most well known being GALPAT. A table of the various algorithms is given below, for details see the original works by the authors.

## Table 4-5: Comparison of Coupling Fault Tests for Memory Circuits

| Name | Complexity | Stuck-at Coverage | Coupling Coverage |
|---|---|---|---|
| *Column Bar test* | 4n | All but decoder or MAR faults. | Covers coupling faults between adjacent columns only |
| *Marching 1's and 0's* | 14n | ALL | Does not cover all single coupling faults. |
| *GALPAT* | $4n^2+2n$ | ALL | Covers most coupling faults |
| *Modified GALPAT* | $4n^2+4n$ | ALL | ALL |
| *TA Alg.*[1] | $8n\log_2 n$ | ALL | ALL |
| *NTA Alg.*[2] | $30n$ | ALL | ALL |
| *SR AlgA.*[3] | $14n$ | All if coupling faults are not present | All if stuck-at faults are not present |
| *ST AlgB.*[4] | $16n$ | Decoder faults are covered if coupling faults are not present. | All if decoder multiple access faults are not present |

1:[*77That*]
2:[*78Nair*]
3:[*81SukR*]
4:[*81SukR*]

Although the GALPAT is the best known, it is quite inefficient relative to some of the other algorithms.

As the density of cells in memory chips increases, pattern-sensitive faults (PSFs) become the predominant fault mode. An algorithm developed for PSFs where the RAM is defined as a sequential machine with $2^n$ states and $3n$ inputs. A checking sequence for this machine of length $((3n^2+2n)2^n)$ is capable of detecting any PSF in the memory. This algorithm is computationally prohibitive for any useful memory size. In order to deal with this problem, the concept of a neighborhood was developed. In dealing with PSFs and neighborhoods, it is assumed that only the nearest k cells can affect a given cell n. This cuts down on the search space for each cell to not include the entire memory array. The popular neighborhood sizes have been 5 (the base cell and 4 nearest neighbors) and 9 (the base cell and 8 nearest neighbors)

Two algorithms have been describe that have complexities of $((k+5)2^{k-1}n)$ and $((3k+2)2^k n)$ where k is the neighborhood size and n the number of cells in the memory. These types of tests are applied on the individual RAMS by the semiconductor manufacturers. It is difficult to apply these tests at the system level. The main reason is that the logical addresses of cell locations have little to do with the physical locations. Sometimes there is an algorithmic way of determining this, other times only a table driven approach can be used. This means that tests that would work for one manufacturer's device would be meaningless for another's.

# 5

# Design For Testability

This chapter describes various design concepts that can make a hardware system more testable. Most of these concepts have been implemented in large computer systems. The benefits and costs of each of these design methods is discussed.

## 5.1 Scan Designs

There are a number of different scan design disciplines that all fall under the term scan design. Among these are level sensitive scan design (LSSD), scan path, scan set, and random access scan. There are two main requirements in a scan design. The first is that there is sufficient control over the clocking of these memory elements in order to prevent races, and the second is that it is possible to load and unload the memory elements in an independent manner.

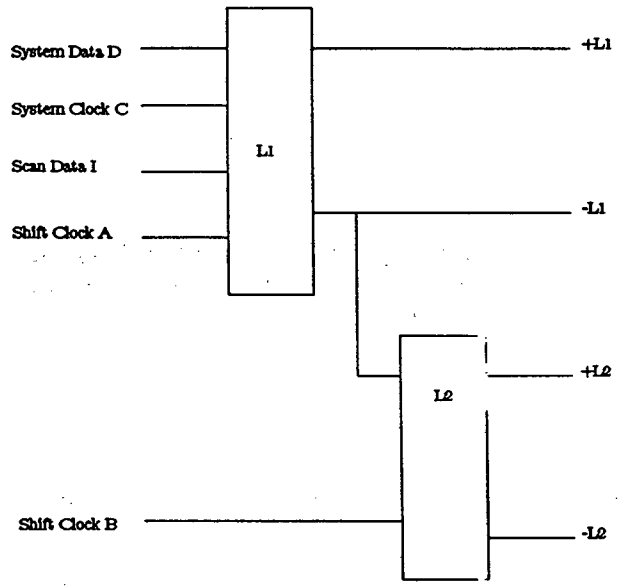
All scan design disciplines are an attempt to reduce the problem of test generation to one for combinational circuits. Tests have been generated for most of the fault models described in chapter 3 for combinational circuits using scan approaches.

### 5.1.1 Level Sensitive Scan Design

Level sensitive scan design is a technique invented by IBM and implemented in several of their computer systems[*77Eich*]. LSSD imposes a clocked structure on all memory elements and forms the elements into shift-register latches, making it possible to shift values into and out of the elements by way of the scan path. The storage elements are implemented as clocked DC latches (latches where the stored data can not be changed while the clock inputs are off). A sample LSSD latch is shown in figure 5-1. The shifting of the registers are controlled by two clocks, A and B. These two clocks must be non-overlapping to ensure correct operation of the chain. The design rules that must be adhered to in order to implement LSSD are:

1) *All internal storage is in clocked DC latches.*
2) *The latches are controlled by two or more nonoverlapping clocks.* Running the system clocks in a nonoverlapping mode eliminates any system dependency on minimum circuit delays since a fast circuit can not create a system malfunction.
3) *Latch X can feed latch Y if and only if the clock that feeds X is not the same as the clock that feeds Y, and if the clocks are nonoverlapping.* This rule assures that the data at the input to a latch will not change while the latch clock is on.
4) *All latches are contained in a shift register latch.*
5) *Either L1 or L2 outputs can be used for system functions, but both together may not be used in the same logic structure.* This is to assure test coverage since it would not be possible to force L1 and L2 to different values.
6) *SRL clock inputs must be controlled from PIs so that when the clock PIs are off, the SRL clock inputs are off.*

Figure 5–1: LSSD Latch




The benefits associated with a design based on LSSD are

1) Testing of a complex circuit is reduced to testing a set of combinational logic circuits
2) The design is almost independent of any delay characteristics of the logic circuits, especially minimum logic delays.

It has been estimated that this approach adds from 4% to 20% in logic overhead. There is a minimal performance penalty.

## 5.1.2 Scan Path

Scan path [*75Funa*] is an approach similar to LSSD except for the implementation of the storage elements being D-type flip flops. This is the way that scan rings have been implemented in both the C1 and C2. This approach does not pose as stringent a set of design rules on the clock circuitry. This approach has smaller logic overhead than the LSSD approach, about 4-10%. This is because the basic scan latch is simpler. There is a slight performance degradation due to the multiplexor on the input of the latch.

## 5.1.3 Other Scan Methods

Some less used scan methodologies are described in this section. These approaches are less rigorous than the other scan approaches presented.

### 5.1.3.1 Random Access Scan

Random access scan is another design for testability method[*80Ando*]. In this method instead of using a shift register to connect the circuits, each latch is individually addressable in a fashion similar to that used in a RAM. The logic overhead is high, as is the additional routing necessary to route the outputs to the appropriate multiplexors. This approach is interesting, but not very practical for a system of our size.

#### 5.1.3.2 Scan Set

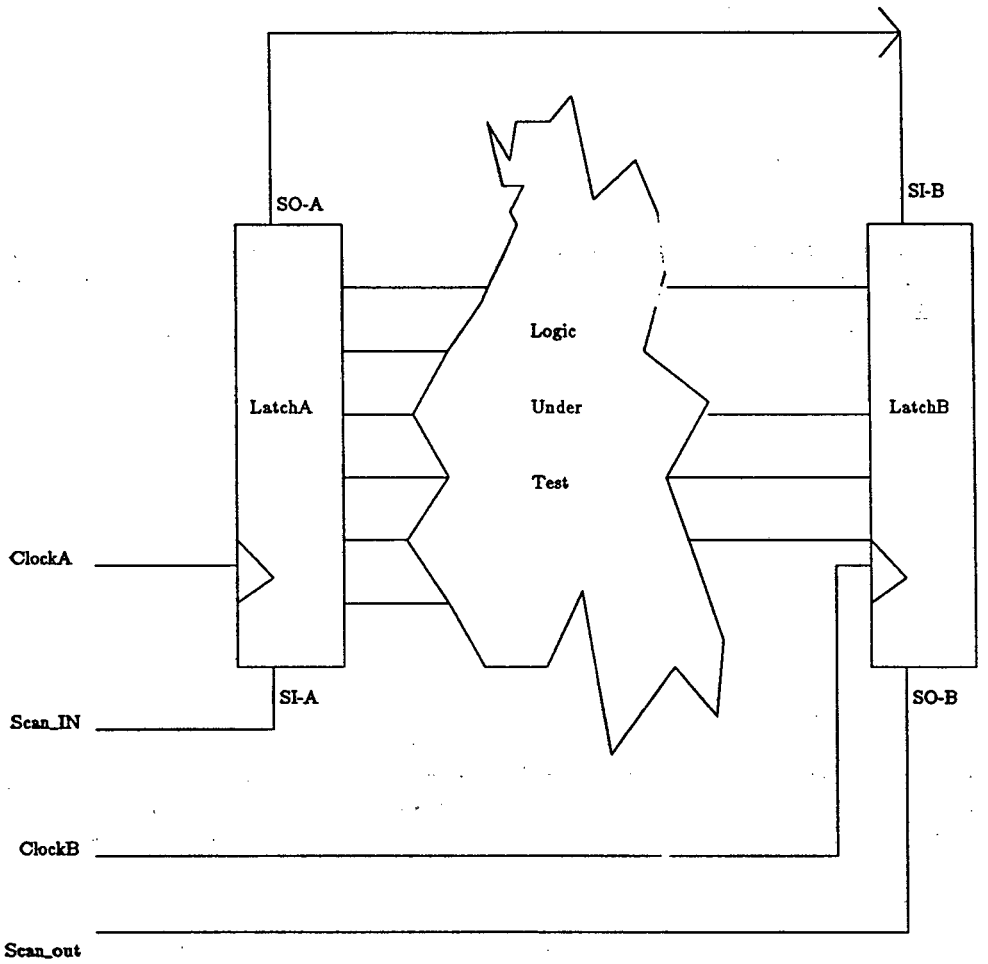
The scan-set design approach [*77Stew*] is a partial scan-structured design, not as thorough as the scan path or LSSD approaches. This approach appends scannable registers to the original sequential circuits for the purpose of either scanning selected circuit outputs or setting values into a circuit or both. Test generation for this approach is much more complex than for some of the other approaches. The C2 was designed partially with this approach and partially with the scan path approach (with the violations of many of the clocking requirements of either of these methods).

### 5.1.4 Advantages of Scan-based Testing

The test pattern generators described in chapter 4 are readily applied to the portions of circuits between two scan rings. This approach has been used successfully for years in large systems. It is readily seen that this approach will test for the existence of static faults (stuck-at, crosspoint, bridging). However, delay faults have also been tested using scan rings. A modified latch has been designed [*87Merc*] and implemented that allows the generation of the two patterns required for the delay test. The literature [*87Bard*] also describes using normal LSSD or scan-set latches to perform propagation delay testing of a logic circuit through the previous combinational stages (similar to the CAST system).

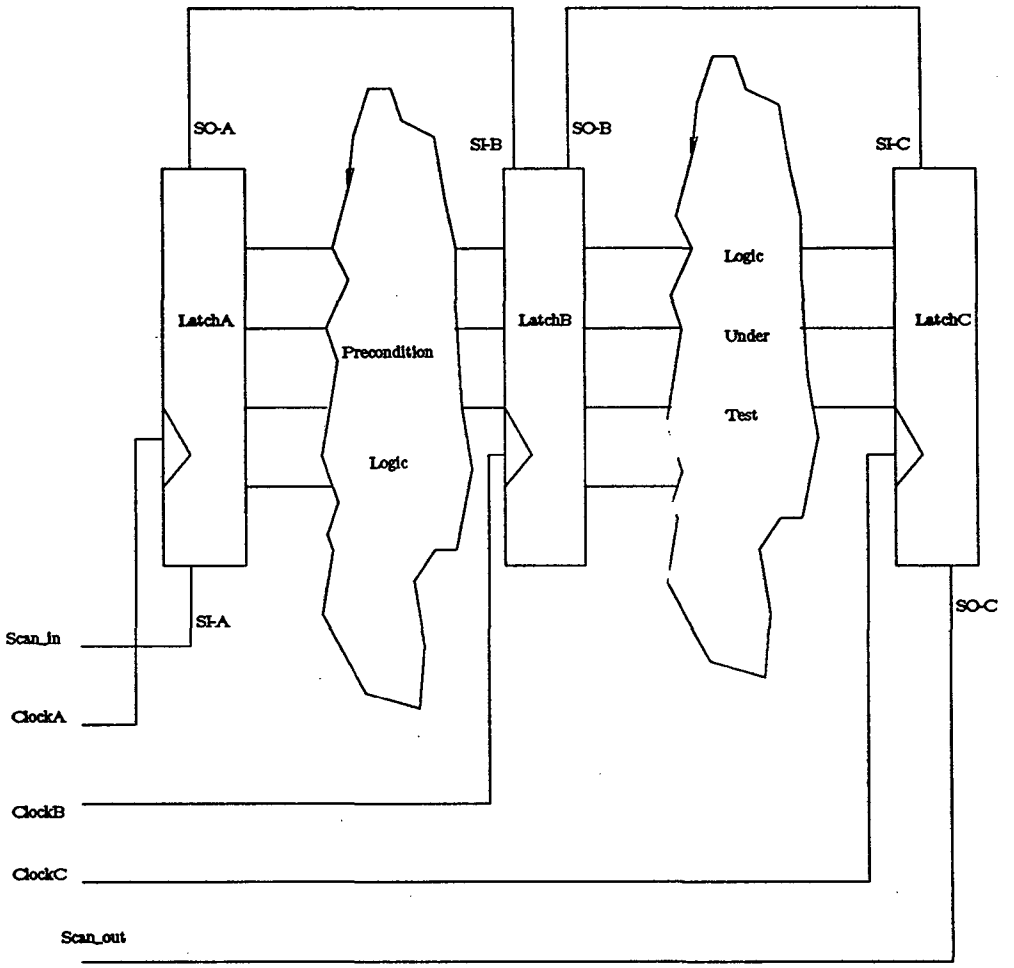
In order to use the scan latches effectively for testing the logic, there are certain clocking requirements. To test for stuck-at faults it must be possible to scan a pattern into a scan ring, then issue one clock and have all latches on the ring clocked in phase. This is necessary to avoid a race condition on the inputs to the latches that are on the outputs of the circuit under test. This example in figure 5-2. In order to test the logic block between the two scan latches and avoid a race condition, the outputs of A must be stable and propagated to the inputs of Latch B prior to latching B. If latchA were on a different clock edge than latchB, it would be necessary to give B a latching edge without giving one to A.

## Figure 5–2: Stuck-at Testing Through Scan Rings




In order to have the ability to perform a delay test through the scan rings it is necessary that the above example be extended to two levels of latches. Figure 5-3 illustrates the clock requirements of a delay test. In order to test for a delay fault in the logic under test it is necessary to scan an initialization pattern into latch B, then apply two clocks. The first clock latches the transition propagation pattern in latch B through the precondition logic, the second clock will latch the results into latch C. For this to be an effective test, the time difference between the clocks at latches B and C must be the same as when the system is used. If latch B is on half-clock during normal system operation, it must be able to be clocked that way during the test.

**Figure 5–3: Delay Fault Testing Through Scan Rings**



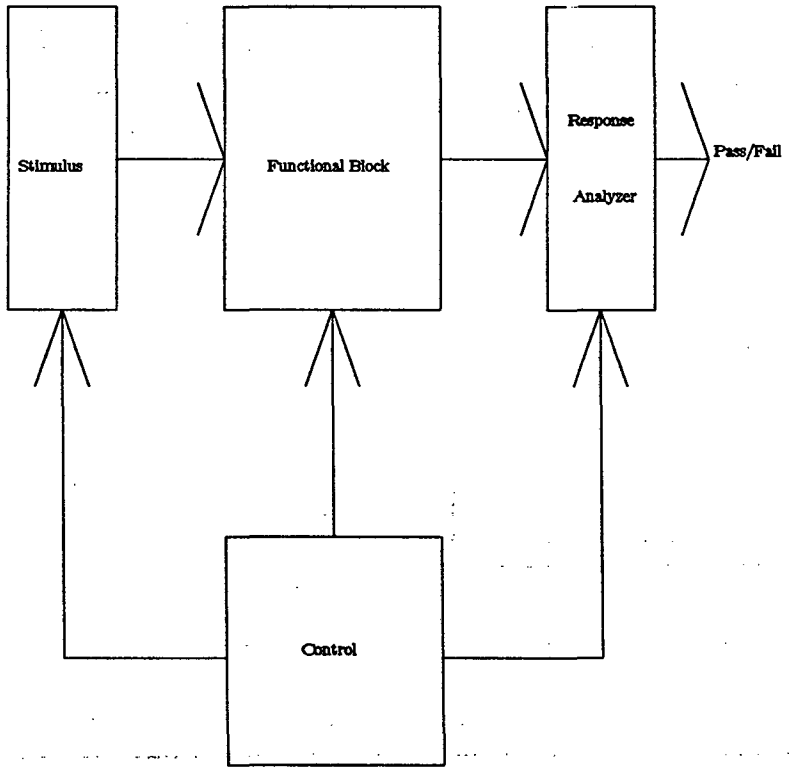

As discussed in chapter 4, it is quite difficult to actually generate delay tests for specified paths in a circuit. The method shown in figure 5-3 is not optimal for delay fault testing due to the fact that a previous combinatorial logic block must be used to se the transition propagation pattern. It is possible that certain input combinations can not be presented to latchA to set up these patterns. This make the ability to generate delay fault tests (robust or non-robust) not possible for all paths. It is this observation that has driven the development of double latches for use in delay fault testing.

## 5.2 Self-Testing Circuits

Circuits can be designed with the ability to test themselves. Built-in test can be either nonconcurrent (an offline test using functional or structural information) or concurrent (online test using either information redundancy, hardware redundancy or both). This section will talk about the offline tests, concurrent testing is discussed in chapter 6.

Self-test circuitry usually consists of some type of stimulus generator, a response analyzer and some control circuitry as shown in figure 5-2. In general, the response analyzer performs some sort of data compression on the responses before a comparison is made. The data compression causes a loss of information and therefore creates some probability of a bad circuit being declared good. These probabilities can be made quite low by proper design of the testing circuitry. Consequently the fault detection of these circuits can be quite high.

**Figure 5–4: Self-Test Circuitry**




One of the big advantages of built-in tests is that they can be made to execute very fast ( at system speed). It also reduces the problem of pattern generation, although the problem of fault simulation is equivalent. The overhead of the self-testing circuitry can be quite high depending upon the methods used.

Self test circuits can use any of the following techniques:

1) Exhaustive testing.
2) Pseudorandom testing.
3) Functional testing
4) Prestored testing.

The exhaustive test simply applies $2^N$ patterns for a circuit with N inputs. Not very elegant but if N is small relative to the clock speed, the test time for this could still be quite low. Partitioning strategies can be deployed to break larger circuits into circuits that are more manageable.

The pseudorandom tests apply some subset of the patterns to the circuit. A linear feedback shift register is usually the choice for the pattern generator. The use of these techniques requires that the patterns either be fault graded or that probabilistic methods be used to determine coverage. A great deal of research has occurred in this area, an good source of additional material is [*Bard*]. By appropriately designing the pseudorandom pattern generator, the error compression circuit and the circuit under test, fault coverage can exceed 95%.

The functional test is a test that tests the functionality of the circuit. The most widespread use of this circuit is built-in test circuitry for RAMS. These circuits test the functions of the RAMS through a algorithmic process.

Stored pattern testing applies a set of patterns that have been precomputed and stored in a ROM. This allows for a standard pattern generator to generate the test patterns so that the set can be small in size and high in fault coverage. The disadvantage is the complexity and the cost of the extra circuitry.

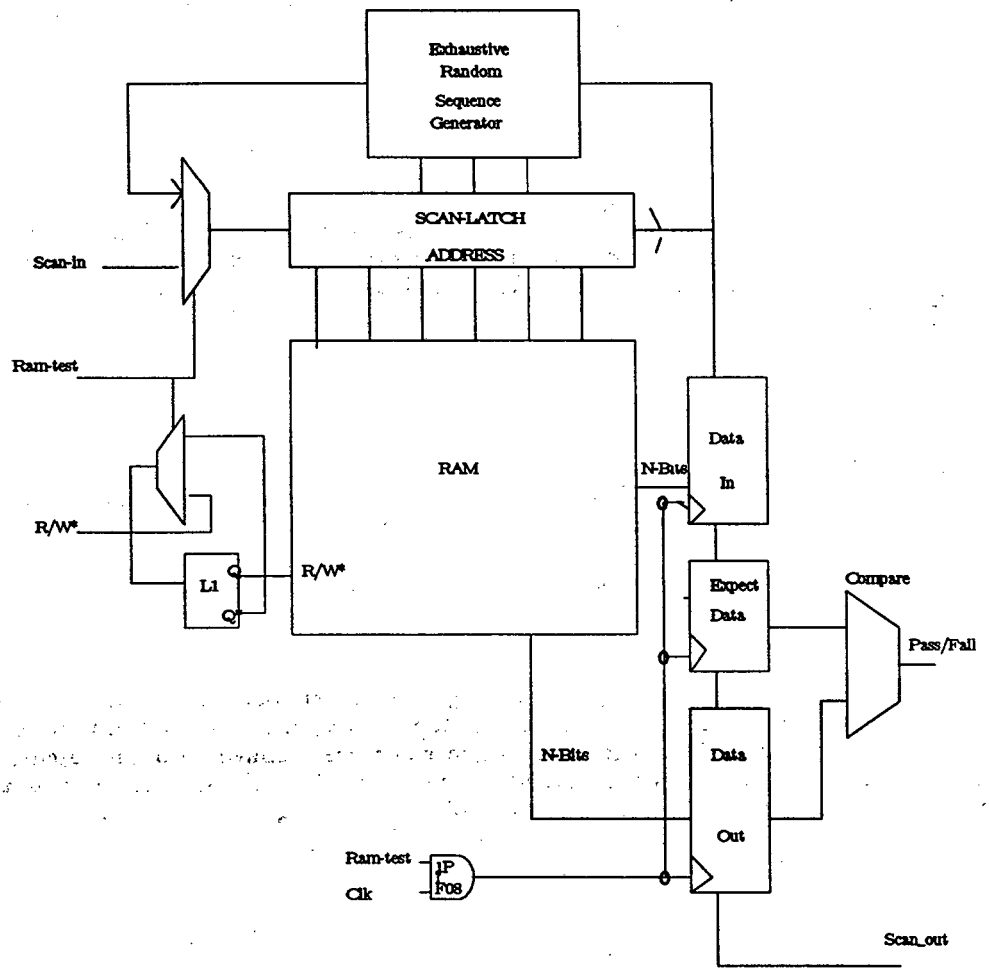
### 5.2.1 Self-Testing Memory Circuits

Systems with RAMS that are embedded in control logic (caches, register files, etc), are particularly difficult to test. Sometimes it is virtually impossible to generate adequate patterns or to provide any level of isolation when testing embedded memory devices. It is difficult to differentiate a failure in the memory from the surrounding logic. It is possible to put address, data and control lines on a scan chain to allow scan pattern testing the memory device. This method is feasible for small memories, but not for large ones. Also, this method does not allow any at-speed testing of the memory circuits.

A solution to this problem is in designing self-testing memory circuits. An example of a self-testing memory circuit is shown in figure 5-5. To test the memory a data-in pattern, an expected data pattern, and a starting seed for the address generator are scanned into the latches. The Ram-test mode bit is then set. Clocks are applied to the circuit. The circuit will perform a write and then a read at each address. The comparator will detect failing sequences. The comparator would need to have a disable bit in the scan ring. Also latch L1 would be on the scan ring so that the initial rd/write state could be set.

This circuit has minimal logic overhead for a circuit already designed with scan latches. The overhead being the sequence generator (M XOR gates and 1 M-input NAND gate where M is the number of address bits), the N-bit wide comparator, the two 2-bit wide multiplexors and the expect-data latch. The basic read/write cycle is required to happen once for each change in the address. This type of circuit can detect 99% of the stuck at cells and decoder faults in the memory.

## Figure 5–5: Self-Testing Ram Circuit



# 6

# Detection of Intermittent Failures

A test strategy would not be complete without a plan for dealing with the temporary failures that will exist in a complex computer system. Studies have been done that show that temporary failures occur at least 10 times more often than permanent failures in large digital systems. Although offline testing for temporary failures has been studied, most effective system applications have methods to detect temporary failures concurrently with normal system operation.

There are two main types of temporary failures, transient and intermittent failures. A **transient failure** is a nonrecurring temporary failure caused by radiation, power supply fluctuation, etc. An **intermittent failure** is a recurring temporary failure caused by component degradation or a poor system design.

It is important in the design of the system that intermittent failures can be located and repaired. The inability to find intermittent failures is a great burden on the field service organization and is very detrimental to customer good-will.

Experimental results reported by [*86Cort*] reveal that intermittent faults (IFs) show pattern sensitivity. Unlike the classical stuck-at faults, the occurrence of pattern-sensitive faults depends not only on the logic value at the fault site but also on the logic values at other lines in the circuit. Pattern-sensitivity problems are well known in memory chips. It is not possible to test an entire system for pattern sensitive failures, there is not enough time remaining in the universe.

A study done on Sperry-Univac computers has tracked intermittent failures that later become hard-failures. These failures were classified into two main types of faults:

1) Metal-related open and short circuits.
2) Marginal operation or violations of operating margins.

Several of these intermittent problems were actually caused by design errors where the fault was actually a delay fault on a seldom used path. However, most were not attributed to design flaws.

The biggest problem with trying to detect an intermittent failure by running diagnostics is that the failure may not occur during diagnostic execution. If the fault is pattern sensitive, it is highly unlikely that any reasonable set of diagnostics is going to detect the failure. One way to try to force an intermittent failure to occur during diagnostics is to margin the system power supplies or clocks. This is an effort to stress the device and hopefully cause an intermittent failure to become more pronounced (this is precisely the method used in [*86Cort*]to study intermittent failures). This increases the probability of detection only slightly. Diagnostics that are designed to detect delay and stuck-at types of faults will have a much lower fault coverage of pattern sensitive faults. Because of the difficulty of designing diagnostics that can detect all intermittent faults, it is necessary to have a test strategy that includes hardware detection methods as well as software isolation and tracking methods to adequately solve this problem.

## 6.1 Error Checking Circuits

Error checking circuits detect illegal combinations of bit patterns on the outputs of circuits. This is only possible if the outputs of a circuit can take on only specific values. In a circuit which has output configurations which do not occur during fault-free operation the outputs which do occur are called code words and the unallowable configurations are called non-code words. The Hamming distance *d* of a code is the minimum number of bits in which two code words differ. The error detecting and correcting capability of a code as a function of *d* is shown below.

### Table 6-1: Error Detection Capability of Code Words

| Hamming Distance(d) | Capability |
|---|---|
| 1 | none |
| 2 | 1 bit error detect, 0 bit error correct |
| 3 | 2 bit error detect, 1 bit error correct |
| E+1 | E-error detect, E/2 error correct |

The parity detector has *d=2* and thus can detect a single bit error. A general set of parity check codes can be designed to perform single bit correction. The number of extra check bits that must tag along with the information bits to provide this error correction is a function of the number of information bits. The equation is $2^c \geq i + c + 1$, where c is the number of check bits and i is the number of information bits. The table below shows the number of check bits required for single bit correction as a function of the number of information bits.

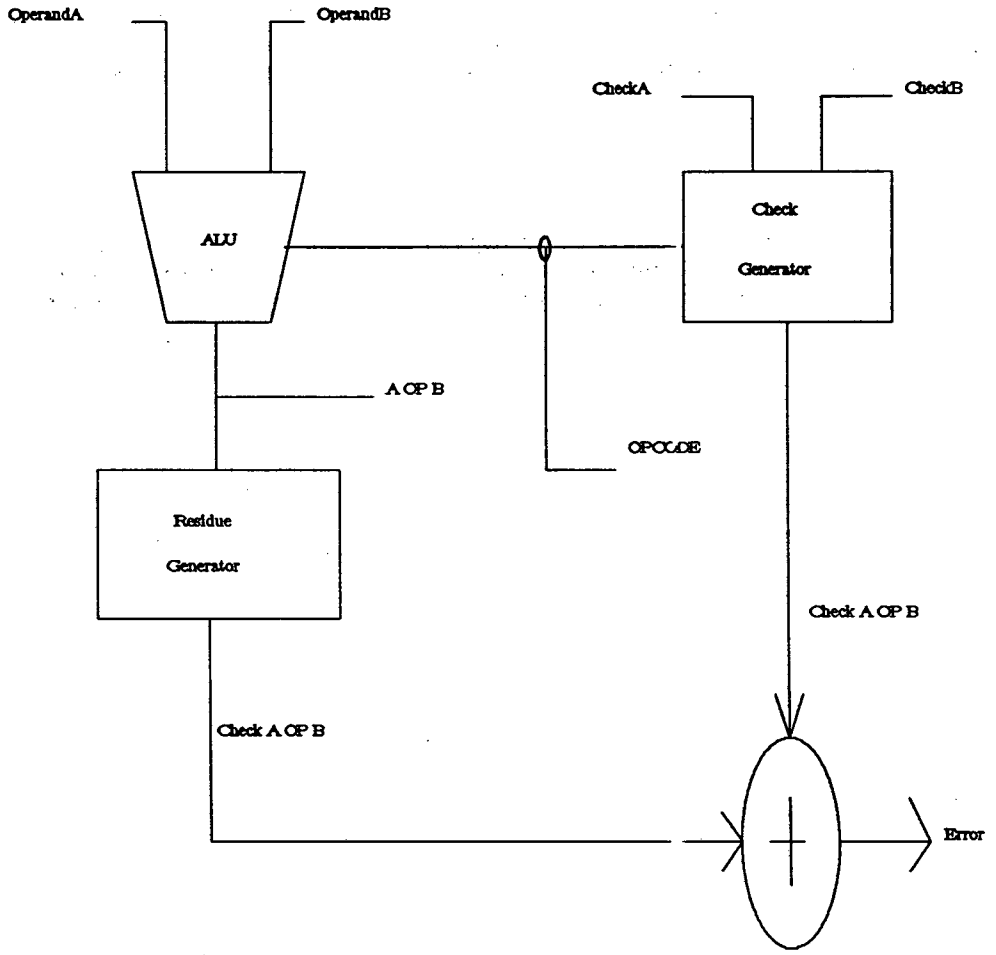
### Table 6-2: Check Bits Required For Single Bit Error Correction

| Information Bits | Check Bits |
|---|---|
| 1 | 2 |
| 4 | 3 |
| 11 | 4 |
| 26 | 5 |
| 57 | 6 |
| 120 | 7 |

As can be seen from table 6-2, the proportion of checkbits to information bits decreases as the number of information bits increases. However, the complexity of the circuitry necessary to generate the checkbits increases directly with the number of information bits.

For error detection in transmission busses or storage elements, parity check codes are adequate. However, to check for errors in an arithmetic unit, parity codes are inadequate. Another class of codes, called residue codes, has been applied for the operations of addition, multiplication and subtraction. These codes can determine the check bits of the result from the check bits of the operands. An example of this type of circuit is shown in figure 6-3. The ALU works on the operands A and B, the check generator operates on the check bits of A and B. After the result is obtained from the ALU, it is placed through a residue circuit. The results of the residue circuit are compared to the output of the code generator. It can be seen that these circuits have much overhead in both performance and circuit size. The performance problem can be overcome with the error detection lagging the operation of the machine. When an error is detected, the machine is rolled back to its state at the point where the error was detected. The machine is then restarted. This rollback requires much more overhead to allow the saving of all of the necessary machine state.

## Figure 6–1: Residue Code Circuit




Convex has used parity generators and checkers on most major system busses and all memory except main system memory. Main system memory has implemented a code with a hamming distance of 3 giving single correct and multiple detect with an overhead of 6 check bits per 32 bit word. This strategy implies that errors only occur outside of devices during transmission from point to point except in the case of memories, where errors only occur inside the device. For a system with a large amount of the total circuitry implemented in gate arrays, this strategy does not seem valid. Observation of C2 crashes and hangs compared to hard errors would also cast some doubts on this strategy.

Some companies have used error correcting hardware to a much greater extent than Convex. By carrying the concept to the extreme a fault tolerant system is built. Our main interest at Convex is not in fault tolerancy but in the detection and isolation of intermittent failures.

A useful model for this type of system would be the IBM3081 computer[*82Tend*],[*82Boss*]. This system was designed with the ability to detect and isolate intermittent failures. The basic approach taken consisted of the following steps

1) Partition the design.

2) Place hardware error detection circuitry at strategic points.
3) Understand in software all of the possible contributors to any error detection circuit.
4) Be able to examine the status of all of the possible contributors to any error detection circuit.
5) Have software that will isolate boards/modules based on the error detected and the status of all contributors.
6) Have software to log intermittent failures for tracking purposes and to allow better recommendations the next time a failure occurs.

All of these things can be implemented to some degree in a Convex system. We probably can not afford adding residue circuits to our ALUs and such, but we probably can use parity circuits intelligently and design state machines that trap invalid states. It is important that software systems be developed concurrent with the hardware design in order to assure that intermittent faults can be detected and isolated.

# 7

# C3 Testability Strategy

Using the information presented in the preceding chapters, the next step is to develop the C3 testability strategy. However, before we address the actual strategy, let's regress and identify the advantages of a highly testable system.

A system which is highly designed for testing can be categorized by the following advantages:

- Shorter test program development time
- Lower test program development cost
- Lower production costs
- Higher system reliability
- Reduced difficulty in system testing
- Lower system field repair costs

Before these points are discussed in more detail, it is worth noting that the last four advantages continue to affect the system throughout the machine's operation. This alone is sufficient to justify an endeavor to improve the testability of the machine.

*Shorter test program development times*: For complex systems with testability problems, it takes more time to complete the development of the tests, and the tests do not exercise the hardware as completely. For severe cases, the test development may not be ready when simulation and prototype checkout begins. Systems designed with testability in mind, however, allow more comprehensive tests to be written and these tests will be easier to write since the hardware is amenable to testing.

*Lower test program development cost*: The longer it takes to develop a test program, the more the development costs. It is not just a question of the man-hours expended, it is also the computer time for development, debug, and verification. Another hidden issue is the man-hours verses productivity. A machine which is testable requires the test engineer to 'jump through fewer hoops' to determine a method for testing a function as opposed to a machine which is not testable.

*Lower production test costs*: There are two ways a testable system results in reduced production costs: production test execution time and fault detection. Difficulty in testing requires the execution of very long architectural verifiers, which translates into testing functional failures from a system level rather than from a function level. In difficult systems to test, the actual effectiveness of the testing would be compromised; hence a greater chance of faulty devices would be missed. Research has indicated that without component fault isolation the cost of detecting a fault will increase by an order of magnitude for every increase in the level of system hierarchy. This shows that any compromise in the quality of the test programs results in higher production costs since a greater number of failures are not detected until the systems are running under the operating system.

*Higher system reliability*: Another characteristic which suffers as a result of any compromise to the testing effectiveness is the reliability of the system. Systems which are difficult to test may result in failures not being detected and the systems being certified as operational. Subsequent system problems are notoriously difficult to diagnose and similar problems arise with 'marginal' devices when inadequate AC or DC parameter testing is performed. These problems will continue to plague a system throughout its operating life.

*Reduced difficulty in system testing*: A system which is difficult to partition for isolation, will inherently be difficult to test. Designing a highly partitioned system, allows systematic testing of isolated functions. This allows the functional interfaces to be verified, which directly reduces the difficulty of testing a system.

*Lower system field repair costs*: Poor testability leads to systems which are difficult to test and less reliable. Such systems attract higher field repair costs throughout their operational lives as opposed to those of an easily testable design.

To summarize, the lack of testability translates into higher operational costs and less reliable systems. Testability is a major objective in the production of any new product and must be addressed as one.

## 7.1 C3 Testability Goals

The design of a testable system begins with a defined set of objectives. The following is a list of testability goals which will make a machine highly testable and cost effective to produce and maintain.

- **Testing the system for 98% stuck-at faults via scan.**
  - Stuck-at fault testing at this level allows manufacturing to run scan based tests to determine if a board has been correctly manufactured and allows field service a method of checking for hard system failures. This approach provides both manufacturing and field service with an easier and more efficient method of determine hard system failures.
- **At-speed, delay fault testing of all testable maximum level paths in a system.**
  - At-speed, delay fault testing allows isolation of the delay failures in those maximum level paths which are testable. The word 'testable' is added to the goal to identify a testing problem. Even though a two pattern sequence can be generated for a path, the path is not delay fault testable if the transition propagation pattern can not be applied to the network due to combinational logic proceeding the path being tested.
- **Continuity checks for shorts and opens in the backplanes and crossbar.**
  - With the level of scan being provided, the backplanes and crossbar are viewed as other boards in the system which allows quick isolation of backplane problems. Unfortunately, this testing requires the support of the system boards since no scan rings reside directly on the backplane. The crossbar will not have this requirement.
- **Complete scan testing of the diagnostic subsystem, crossbar, one CPU bay, and the IO bay in less than one hour.**
  - Testing this configuration in less than one hour provides a substantial cost savings regarding the time required to bring up and troubleshoot a system.
- **Hot mock up running scan based testing to bring up board sets.**
  - Providing scan testing for hot mock up provides a consistent testing approach for all boards in the system. This provision allows faster failure isolation and throughput.
- **Isolate hardware detected intermittent failures to the board and function, and**

**identify the source of the error.**

- Effective intermittent error isolation allows quick isolation of those functions being monitored by the hardware(e.g. Parity, ECC, etc.). This provides manufacturing and field service with information necessary to locate and repair parts that exhibit intermittent failures.

- **Mean time to diagnose of less than two hours.**

  - The field service engineer shall diagnose the system failure within two hours after arriving at a customer's site. This will not only strengthen our customer's confidence in our system but it is also an impressive marketing issue.

## 7.2 Design Rules

To support the testability goals it is necessary to enforce some design rules upon the C3 system. Without implementing these rules the above goals are unachievable, or are achievable at much higher costs.

- **All Storage Elements That Are Not Memory Devices Must be Scannable.**

  - This aids in meeting the goal of scan-based stuck-at fault testing for 98% of the failures. It also allows implementing delay fault testing.

- **All Embedded Memory Elements Contain Built-In Test Capabilities.**

  - This allows the goal of 98% stuck-at fault testing for the memory circuitry to be met it also allows at-speed pattern testing of these memory elements. Without self-test circuitry it is impossible to provide adequate fault isolation of faults in these memory elements.

- **All New Board Designs Will Have At Least 1Kbyte of EEPROM.**

  - The large EEPROMS will store board failure information. This information will be used by the failure and diagnostic tracking system.

- **All Boards Reset To A Consistent State When A Reset Signal Is Asserted.**

  - It is necessary for boards to be reset into a state where they will not affect the operation of clocks in any chassis.

- **It Will Be Possible To Isolate a Chassis From The Crossbar For The Purpose Of Testing.**

  - It is necessary to isolate a chassis during scan testing such that it does not interfere with the correct operation of other chassis or the crossbar. The chassis under test should also be isolated such that it is not interfered with by the operation of any other chassis or the crossbar.

- **It Will Be Possible To Control All Clock Signals Within Any Chassis Independently Of Clock Signals In All Other Chassis.**

  - This is needed to test a chassis while the rest of a system is in the run state.

- **All Memory Devices Contain Parity.**

- This allows for detecting transient memory errors that occur during system execution.

- **Dynamic Ram Circuitry Contains EDC Circuitry.**

  - This allows for detecting and correcting transient memory errors that occur during system execution.

- **All Signals That Cross Boards Are Latched On The Sending And Receiving Sides.**

  - This allows backplane continuity tests on a loaded system to be performed. It also allows scan-based tests to be applied to a board independent of the other boards in the system.

- **All Latches Can Be Clocked On The Same Clock Edge For Testing.**

  - This enables stuck-at testing from the scan rings without race conditions. This is also necessary for delay fault testing by being able to change the actual clock rate to be either 1, 2 or 3X.

- **All Non-Control Signals That Cross Boards Are Parity Encoded**

  - This enables the detection and isolation of transient errors during system execution. For parity detectors that are detecting parity for a multiple-source signal, it is necessary to latch the status of the driving signal to allow isolation of the faulty unit.

- **All State Machines Are Implemented To Detect Invalid Inputs**

  - Circuits are able to detect invalid inputs for each state and indicate a hard error condition. This allows for detecting and isolating transient errors during system execution.

- **All Tag Information Is Accessible And All Cache Implementations Support The Disabling Of The Caches As Well As Forcing Cache Hits.**

  - This allows cache control and consistency testing from the architectural verifiers.

- **All Boards That Are Not Accessible Via Scan Rings Shall Implement Self-testing Hardware.**

  - This includes all new CCUS and VME controller boards as well as system monitor boards. It is imperative that ALL boards in a system be testable. Although we can not affect the VME controllers or CCUS we already have, we should no longer accept untestable designs in any portion of the system. The system is only as strong as its weakest link.

# A

# Bibliography


[66Arms] D. B. Armstrong, "On Finding a Nearly Minmal Set of Fault Detection Tests for Combinational Logic Nets", *IEEE Transactions on Electronic Computers*, Vol. EC-15, Number 1, February 1966, pp. 66-73.

[67Roth] J. Paul Roth, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits", *IEEE Transactions on Electronic Computers*, Vol. EC-16, Number 5, October 1967, pp. 567-580.

[75Frie] Arthur D. Friedman, *Logical Design of Digital Systems.* Rockville, MD: Computer Science Press, 1975.

[75Funa] S. Funatsu, N. Wakatsuki and T Arima, "Test Generation Systems in Japan, " *12th Design Automation Symposium*, June 1975, pp. 114-112.

[76Breu] Melvin A. Breuer and Arthur D. Friedman, *Diagnosis & Reliable Design of Digital Systems.* Rockville, MD: Computer Science Press, 1976.

[77Eich] E. B. Eichelberger and T. W. Williams, "*A Logic Design Structure For LSI Testability*", *Proceedings of the 14th Design Automation Conference*, 1977, pp 462-468.

[77That] S. M. Thatte and J. A. Abraham, "Testing of Semiconductor Random Access Memories", *Proceedings 1977 International Conference on Fault-Tolerant Computing*, pp. 81-87.

[78Nair] R. Nair, S. M. thatte and J. A. Abraham, "Efficient Algorithms for Testing Semiconductor Random-Access Memories", *IEEE Transactions on Computer*, VOL C-27, NO. 6, June 1978, pp. 572-576.

[79Glas] Arthur B. Glaser and Gerald E. Subak-Sharpe, *Integrated Circuit Engineering. Design, Fabrication, and Applications*, Reading Mass: Addison-Wesley, 1979.

[80Gala] J. Galaiy, Y. Crouzet, and M. Vergniault, "Physical Versus Logical Fault Models of MOS LSI Circuits: Impact on Their Testability", *IEEE Transactions on Computers*, VOL C-26, June 1980, pp. 527-531.

[80Gold] L. A. Goldstein, E. J. Thigpen, "SCOAP: Sandia Controllability/Observability Analysis Program", *Proceedings 1980 Design Automation Conference*, July 1980, pp. 190-196.

[80Roth] Paul Roth, *Computer Logic, Testing and Verification*, Potomac, Maryland:Computer Science Press, 1980.

[81Prab] Goel Prabhakar, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", *IEEE Transactions on Computers*, VOL. C-30, NO. 3, March 1981, pp. 215-222.

[81SukR] D. S. Suk and S. M. Reddy, "A March Test for Functinal Faults in Semiconductor Random Access Memories", *IEEE Transactions on Computers*, VOL. C-20, NO 12, December 1981, pp. 982-985.

[82 Boss] D. C. Bossen and M. Y. Hsiao, "Model for Transient and Permanent Error-Detection and Fault-Isolation Coverage", *IBM Journal of Research and Development*, VOL 26, NO. 1,

January 1982, pp. 67-88.

[82Tend] Nandakumar N. Tendolkar and Robert L. Swann, "Automated diagnostics Methodology for the IBM 3081 Processor Complex", *IBM Journal of Research and Development*, VOL 26, NO. 1, January 1982 pp. 78-88.

[83Fuji] Hideo Fujiwara and Takeshi Shimono, "On the Acceleration of Test Generation Algorithms", *IEEE Transactions on Computers*, VOL. C-32 NO. 12, December 1983, pp. 1137-1144.

[84Bozo] Sied Bozorgui-Nesbat and Edward J. McCluskey, "Lower Overhead Design For Testability of Programmable Logic Arrays", *Proceedings IEEE 1984 International Test Conference*, October 1984, pp. 856-865.

[84Brgl] Franc Brglez, Philip Pownall, and Robert Hum, "Application of Testability Analysis: From ATPG to Critical Path Tracing", *Proceedings IEEE 1984 International Test Conference*, October 1984, pp. 705-712.

[84Capl] Stephen Caplow, "Conquering Testability Problems by Combining In-Circuit and Functional Techniques", *Proceedings IEEE 1984 International Test Conference*, October 1984, pp. 581-588.

[84Cord] Vincent A. Cordi, "4381's Error-Detection Fault-Isolation Speeds Repairs", *Computer Systems Equipment Design*, November 1984, pp. 23-29.

[84Dand] Ramaswami Dandapani, Janak H. Patel, and Jacob A. Abraham, "Design of Test Pattern Generators for Built-in Test", *Proceedings IEEE 1984 International Test Conference*, October 1984, pp. 315-319.

[84Davi] Scott Davidson, "Fault Simulation at the Architectural Level", 1984 International Test Conference, October 1984, pp. 669-679.

[84Gern] M. Gerner and H. Nertinger, "Scan Path in CMOS Semicustom LSI Chips", *Proceedings IEEE 1984 International Test Conference*, October 1984,pp. 834-841.

[84Hugh] Joseph L. A. Hughes and Edward J. McCluskey, "An Analysis of the Multiple Fault Detection Capabilities of Single Stuck-at Fault Test Sets", *Proceedings IEEE 1984 International Test Conference*, October 1984, pp. 52-58.

[84Kino] Kozo Kinoshita and Kewal K. Saluja, "Built-in Testing of Memory Using On-Chip Compact Testing Scheme", *Proceedings IEEE 1984 International Test Conference*, October 1984, pp. 271-281.

[84Mala] Yashwant K. Malaiya and Shoubao Yang, "The Coverage Problem for Random Testing", *Proceedings IEEE 1984 International Test Conference*, October 1984, pp. 237-245.

[84Mant] Sirdhar R. Mathani and Sudhakar M. Reddy, "On CMOS Totally Self-Checking circuits", *Proceedings IEEE 1984 International Test Conference*, October 1984, pp. 271-281.

[84Sun] Zuxi Sun and Laung-Terng Wang, "Self-Testing of Embedded Rams", *Proceedings 1984 International Test Conference*, October 1984, pp. 148-156.

[84Star] C. W. Starke, "Built-in Test for CMOS Circuits", *Proceedings 1984 International Test Conference*, October 1984, pp. 309-314.

[84Yash] Yashwant K. Malaiya and Shoubao Yang, "The Coverage Problem for Random Testing", *Proceedings 1984 International Test Conference*, October 1984, pp. 237-245.

[85Seth] Sharad C. Seth and Lilu Pan, "PREDICT-Probabilistic Estimation of Digital Circuit

Testability", *Proceedings 1985 International Symposium on Fault-Tolerant Computing*, 1985, pp. 220-225.

[85Brgl] Franc Brglez, Phillip Pownall and Robert Hum, "Accelerated ATPG and Fault Grading VIA Testability Analysis", ISCAS 1985, pp. 695-698.

[85Dall] A. Dallman, G. Menzel, R. Weyl and F. Fox, "Failure Analysis of ECL Memories by Means of Voltage Contrast Measurements and Advanced Preparation Techniques", *Proceedings 1985 International Reliability Physics Symposium*, March 1985, pp. 224-227.

[85Fuji] Hideo Fujiwara, *Logic Testing and Design for Testability*, Cambridge, Mass: MIT Press, 1985.

[85Smit] Gorden L. Smith, "Model for Delay Faults Based Upon Paths", *Proceedings 1985 International Test Conference*, November 1985, pp. 342-349.

[86Abad] Magdy S. Abadir and Melvin A. Breuer, 'Scan Path with Look Ahead Shifting (SPLASH)", *Proceedings 1986 International Test Conference*. September 1986, pp. 696-704.

[86Cort] Mario L. Cortes and Edward J. McCluskey, "An Experiment on Intermittent-Failure Mechanisms", *Proceedings 1986 International Test Conference*, September 1986, pp. 435-442.

[86Davi] Scott Davidson and James L. Lewandowski, "ESIM/AFS - A Concurrent Architectural Level Fault Simulator", *Proceedings 1986 International Test Conference*, September 1986, pp. 375-383.

[86Hugh] Joseph L. A. Hughes and Edward J. McCluskey, "Multiple Stuck-at Fault Coverage of Single Stuck-at Fault Test Sets", *Proceedings IEEE 1986 International Test Conference*, September 1986, pp. 368-374.

[86Koep] S. Koeppe, "Modeling and Simulation of Delay Faults in CMOS Logic Circuits", *Proceedings 1986 International Test Conference*, September 1986, pp. 530-536.

[86Schn] Birger Schneider, Gert Jorgensen and Mogens Bo Christensen, "The Effects of Backdrive Stressing Fast IC Technologies", *Proceedings 1986 International Test Conference*, September 1986, pp. 452-464.

[86Tsui] Frank F. Tsui *LSI/VLSI Testability Design*. New York,NY: McGraw-Hill, 1986.

[86Waic] J. A. Waicukauski, E. Lindbloom, B. Rosen and V. Iyengar, "Transition Fault Simulation by Parallel Pattern Single Fault Propagation", *Proceedings 1986 International Test Conference*, September 1986, pp.542-549.

[86Wu] D. M. Wu, C. E. Radke, and J. P. Roth, "Statistical AC Test Coverage", *Proceedings 1986 International Test Conference*, September 1986, pp. 538-541.

[86Yau] Chi W. Yau, "Concurrent Test Generation Using AI Techniques", *Proceedings 1986 International Test Conference, September 1986*, pp. 722-731.

[87Amer] E. A. Amerasekera and D. S. Campbell, *Failure Mechanisms in Semiconductor Devices*,Poughkeepsie, New York:John Wiley & Sons, 1987.

[87Aria] S.A. Al-Arian, and D. P. Agrawal ,"Physical Failures and Fault Models of CMOS Circuits", *IEEE Transactions on Circuits and Systems*, Vol. CAS-34, NO. 3, Mar 1987, pp. 269-279.

[87Bard] Paul H. Bardell, William H. McAnney and Jacob Savir, *Built-in Test for VLSI: Pseudorandom Techniques*, Poughkeepsie, New York:John Wiley & Sons, 1987.

[87Fant] Fausto Fantini and Massimo Vanzi, "VLSI Failure Mechanisms", *Proceedings VLSI and Computers*, 1987 pp. 937-943.

[87Merc] M. Ray Mercer and Eun Sei Park, "Robust and Nonrobust Tests for Path Delay Faults in a Combinational Circuit", *Proceedings 1987 International Test Conference*, September 1987, pp. 1027-1024.

[87Nany] Takashi Nany and Hendrik A. Goosen, "Effect of Byzantine Hardware Faults on Concurrent Error Checking", *IEEE International Conference on Computer Aided Design*, November 1987, pp. 242-245.

[87Redd] S. M. Reddy, S. Patil and C. J. Lin, "An Automatic Test Pattern Generator for the Detection of Path Delay Faults", *IEEE International Conference on Computer Aided Design*, November 1987, pp. 284-287.

[87Rama] Ramachandra P. Junda and Bharat D. Rathi, "Improving Memory Subsystem Availability Using BIST", *IEEE International Conference on Computer Aided Design*, November 1987, pp. 340-343.

[87Sarf] Thomas M. Sarfert, Michael H. Schulz and Erwin Trischler, "SOCRATES: A highly Efficient Automatic Test Pattern Generation System", *IEEE International Test Conference Proceedings*, September, 1987, pp. 1016-1024.

[88Agra] Vishwani D. Agrawal, Kwang-Ting Cheng, Prathima Agrawal, "CONTEST: A Concurrent Test Generator for Sequential Circuits", *25th ACM/IEEE Design Automation Conference*, 1988, pp. 84-89.

[88Chen] Wu-Tung Cheng, "Split Circuit Model For Test Generation", *25th ACM/IEEE Design Automation Conference*, 1988, pp. 86-101. [88Glob] C. Thomas Glober and M. Ray Mercer, "A Method of Delay Fault Test Generation", *25th ACM/IEEE Design Automation Conference*, 1988, pp. 90-95.

[88Calh] John Calhoun, David Bryan and Franc Brglez, "Automatic Test Pattern Generation for Scan-Based Digital Logic", MCNC Technical Report TR87-17.

[88Mao] Weiwei Mao and Michael D. Ciletti, "DYTEST: A self-learning Algorithm Using Dynamic Testability Measures to Accelerate Test Generation", *25th ACM/IEEE Design Automation Conference*, 1988, pp. 591-596.

# B

# Glossary

**Backward Line Justification**: The process of finding a primary input pattern (test pattern) that will realize all of the necessary gate input values on sensitized paths.
**Behavioral Model**: A model that describes what a function does, not how it does it.
**Bridging Faults**: A fault that occurs when two lines are shorted.
**Combinatorial Logic**: A combinational or combinatorial logic circuit consists of an interconnected set of gates with no feedback loops. The output values of a combinatorial circuit at a given time depend only on the present inputs.
**Crosspoint Faults**: A fault that occurs in a PLA or PAL where a device at a crosspoint exists when it is not supposed to or does not exist when it is supposed to.
**Delay Faults**: A fault that exists in a system if a signal can not propagate through a combinatorial network in time to be clocked into the next set of latches.
**Equivalent Faults**: a set of faults that cause a circuit to malfunction in precisely the same way.
**Failure Mechanism**: The physical or chemical process that causes a device to fail.
**Failure Mode**: The means by which a failure is detected.
**Failure Rate**: The reciprocal of a device's time to failure.
**FIT**: A unit of measuring failure rate. One device failure in 10^9 operating hours is termed one FIT.
**Forward Fault Sensitization**: The process of of driving the sensitized value at a fault site forward to a primary output to be observed.
**Gate Delay Fault**: A gate delay fault is a gate defect that results in at least one path fault (see path fault).
**Hamming Distance**: The minimum number of bits in which any two binary words differ.
**Infant Mortality**: When used in the context of reliability engineering this term refers to device failures that within a relatively short time.
**Intermittent faults**: A recurring temporary failure caused by component degradation or a poor system design.
**Logic fault**:A fault that causes the logic function of a circuit element or an input signal to be changed to some other logic function.
**Nonrobust Delay Test**: A test which detects path delay faults under the assumption that only the paths passing through a given delay fault site can cause an excessive path delay.
**Parametric fault**:A fault that alters the magnitude of a circuit parameter, causing a change in some factor such as circuit speed, current or voltage.
**Path Fault**: A path of the combinatorial network between input and output latches for which a transition in the specified direction by setting of an input latch does not arrive at the path output in time for proper setting into an output latch.
**Permanent faults**: Faults that are always present and do not occur, disappear, or change their nature during testing.
**Primary Input**:A line which is not fed by any other line in the given circuit.
**Primary Output**:A line whose signal is accessible to the exterior of the given circuit.
**Robust Delay Test**: A test that detects an excessive path delay for the path passing through the fault site independent of the other path delays in a circuit.
**Sequential Logic**:A sequential logic circuit contains feedback loops. The output values at a given time depend not only on the present inputs but also on inputs applied previously.
**Solid faults**: See permanent fault.
**Structural Model**:A model that includes implementation specific information. This model describes how a function is implemented as well as what it does.
**Stuck-at Faults**: A fault that causes a circuit node to be permanently stuck at a logic value.
**Testing**: the examination of a product (the whole, as well as all of the its parts) to ensure that it

functions and exhibits the properties and capabilities that it was originally intended to provide.
**Testability**: a measure of the ability to design to be tested to accurately and quickly determine a hardware failure.
**Transient Fault**: a nonrecurring temporary failure caused by radiation, power supply fluctuation, etc.

# C

# Stuck-at Fault Test Generation Algorithms

## Stuck-at Fault Algorithms

This appendix describes several of the stuck-at fault test generation algorithms in detail. The D, PODEM, FAN and 9-V algorithms are described.

## D-Algorithm

The D-algorithm is based upon a 5-valued logic system with the values 0, 1, X, D and $\bar{D}$. The value D represents the value 1 in a functioning circuit and a 0 in a faulty circuit. The value $\bar{D}$ represents the value 0 in a functioning circuit and a 1 in a faulty circuit. For a gate $G$ which implements a logic function $f$, the *singular cover* of $G$ is a set of prime implicants $f$ and $\bar{f}$. Each prime implicant of the singular cover is called a *singular cube*. For example, consider a two input AND with inputs a and b, and output f. The prime implicant of $f$ is $a \cdot b$ and the prime implicants of $\bar{f}$ are $\bar{a} + \bar{b}$. Likewise for a two input OR gate, the prime implicants of $f$ are $a + b$ and the prime implicant of $\bar{f}$ is $\bar{a} \cdot \bar{b}$. The singular cover of these two gates are shown below.

**Figure C–1: Singular Cover of Primative Logic Gates**

| Singular Cover of AND Gate | | | Singular Cover of OR Gate | | |
|---|---|---|---|---|---|
| a | b | f | a | b | f |
| 1 | 1 | 1 | 1 | X | 1 |
| 0 | X | 0 | X | 1 | 1 |
| X | 0 | 0 | 0 | 0 | 0 |

A primitive D-cube of a fault (pdcf) is used to express tests for a fault in terms of the input and output lines of the faulty gate. As a simple case, consider an AND gate with an output s-a-0 fault. To test the fault it is necessary to set the output to 1, and thus the inputs to 1. Then the output will have a value of 1 in the functional circuit and a 0 in the faulty circuit. Thus the primitive D cube of this fault is: 1 1 D. This represents two cubes and the discrepancy D for discrepancy. A pdcf is formed from an input pattern to the circuit for which the correct circuit has the value 1 and the incorrect, 0. It is the goal of a test to be able to drive the D value where it can be observed on a primary output.

A simple circuit to illustrate the D-algorithm is shown in figure C-2. This example will be stepped through, the flowchart of the D-algorithm is shown in figure C-3. It is helpful to refer to this flowchart during the following discussion.

For the circuit in figure C-2 it is desired to generate a test for line 6 s-a-1. We initialize the test cube ($tc^0$) to all unknowns, then select a pdcf for this gate. Since it is a s-a-1 fault, we would like to force the output to zero. The pdcf is therefore:

| 2 | 3 | 6 |
|---|---|---|
| 1 | 1 | $\overline{D}$ |

In this case, there is only one pdcf, however for line 6 s-a-0, there are two pdcfs possible:

| 2 | 3 | 6 |
|---|---|---|
| 0 | X | D |
| X | 0 | D |

When there is a choice, the D-algorithm picks the first one and remembers that there was a choice made. If a dead end is reached the algorithm backtracks to make an alternate choice. The first step after the selection of the pdcf is to intersect the test cube with the pdcf selected. Which results in

| 2 | 3 | 6 |
|---|---|---|
| 1 | 1 | $\overline{D}$ |

The implication step is performed next, in this case there is no implication to perform since the current assignments of lines 2, 3 and 6 do not imply any other logic values. Since no D or $\overline{D}$ is on an output, the algorithm continues.

The process of moving the D value towards the primary output is called D-drive. D-drive consists of selecting a gate in the D-frontier and intersecting the current test cube with the pdc of the selected gate. The set of all gates whose output values are unspecified but whose input has some signal D or $\overline{D}$ is called the *D-frontier*. At this point in the example, gates 5 and 6 are on the D-frontier of the test cube $tc^0$. Again a choice must be made, here the algorithm will arbitrarily choose gate 5. The pdc of this gate is

| 1 | 6 | 9 |
|---|---|---|
| 1 | $\overline{D}$ | $D$ |

The D-intersection is therefore:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $tc^0$ | X | 1 | 1 | X | X | $\overline{D}$ | X | X | X | X | X | X |
| $pdc^1$ | 1 | | | | | $\overline{D}$ | | D | | | | |
| $tc^1$ | 1 | 1 | 1 | X | X | $\overline{D}$ | X | X | D | X | X | X |

Implication applied at this step will assign a 0 to line 5 since 1 and 3 are both 1, also line 8 can be assigned a 1 since line 5 is a zero. So the current test cube is

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $tc^3$ | 1 | 1 | 1 | X | 0 | $\overline{D}$ | X | 1 | D | X | X | X |

Note that the test cube number gets incremented each time it takes on a new value. The D-frontier is now gates 6 and 8. If gate 8 is selected for D drive we will get the following $tc^4$ by intersecting the pdc of gate 8 with $tc^3$.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $pdcgate8$ | | | | | | | | 1 | D | 1 | 1 | $\overline{D}$ |
| $tc^4$ | 1 | 1 | 1 | X | 0 | $\overline{D}$ | | 1 | D | 1 | 1 | $\overline{D}$ |

Again implication is performed. Line 11 being set to a 1 forces line 7 to be a 0 since line 3 has already been fixed at 1. Line 10 being set to 1 causes line 4 to be set to a 0 since line 10 is already at $\overline{D}$. The forcing of line 7 to a zero implies that lines 2 and 4 are 1's, however this is inconsistent since line 4 has been assigned the value 0. Since there is an inconsistency the algorithm must backtrack to the last decision point. This was in selecting which gate to select on the D-frontier. Since gate 8 was selected last time, gate 6 will be selected this time. The new value of $tc^4$ is calculated by intersecting $tc^3$ with the pdc of line gate 6.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $pdcgate6$ | | | | 1 | | $\overline{D}$ | | | | D | | |
| $tc^4$ | 1 | 1 | 1 | X | 0 | $\overline{D}$ | | 1 | D | 1 | 1 | $\overline{D}$ |

Again, implication must be performed. The ones on lines 2 and 4 imply a 0 on line 7. The 0 on line 7 implies a 1 on line 11. Since 9 and 10 are D and 8 and 11 are 1, the value $\overline{D}$ appears on

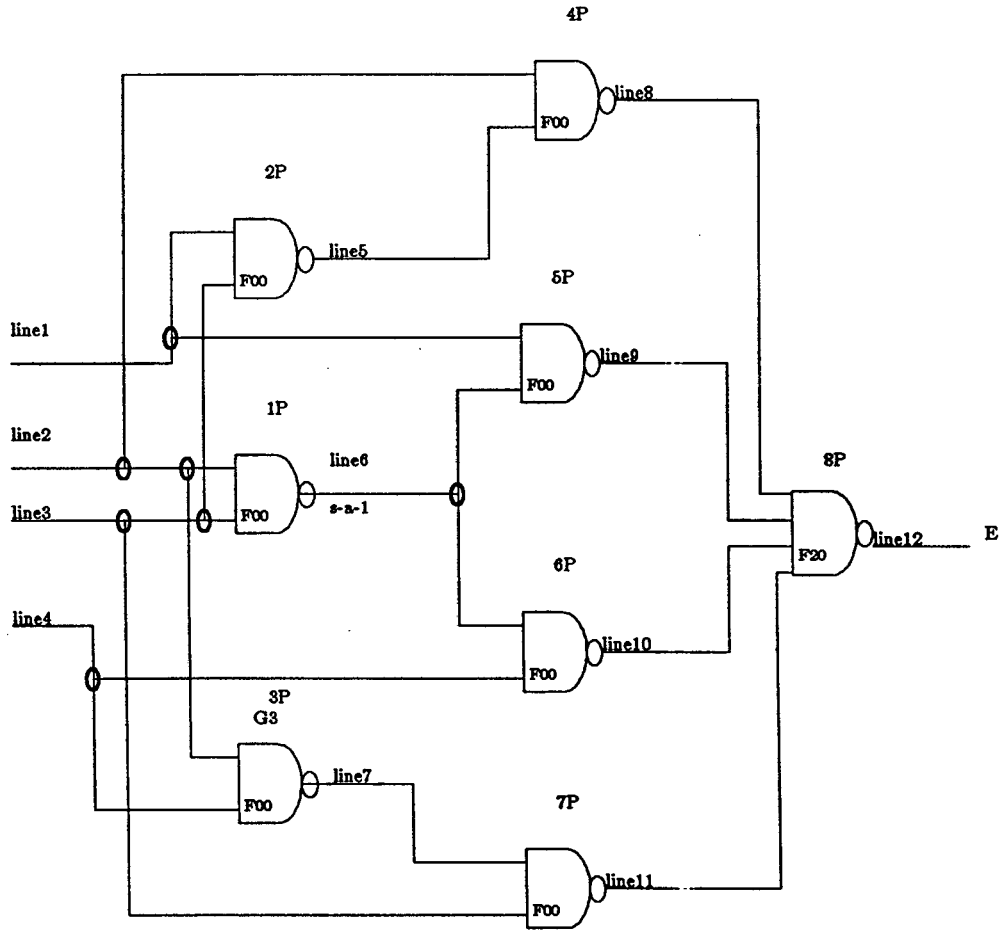
primary output 12.

Once a fault signal reaches an output it is necessary to perform the process of line justification. Line justification is necessary to specify input values on all gates that have outputs already specified. In this case, no line justification is necessary, the test is generated with the following cube:

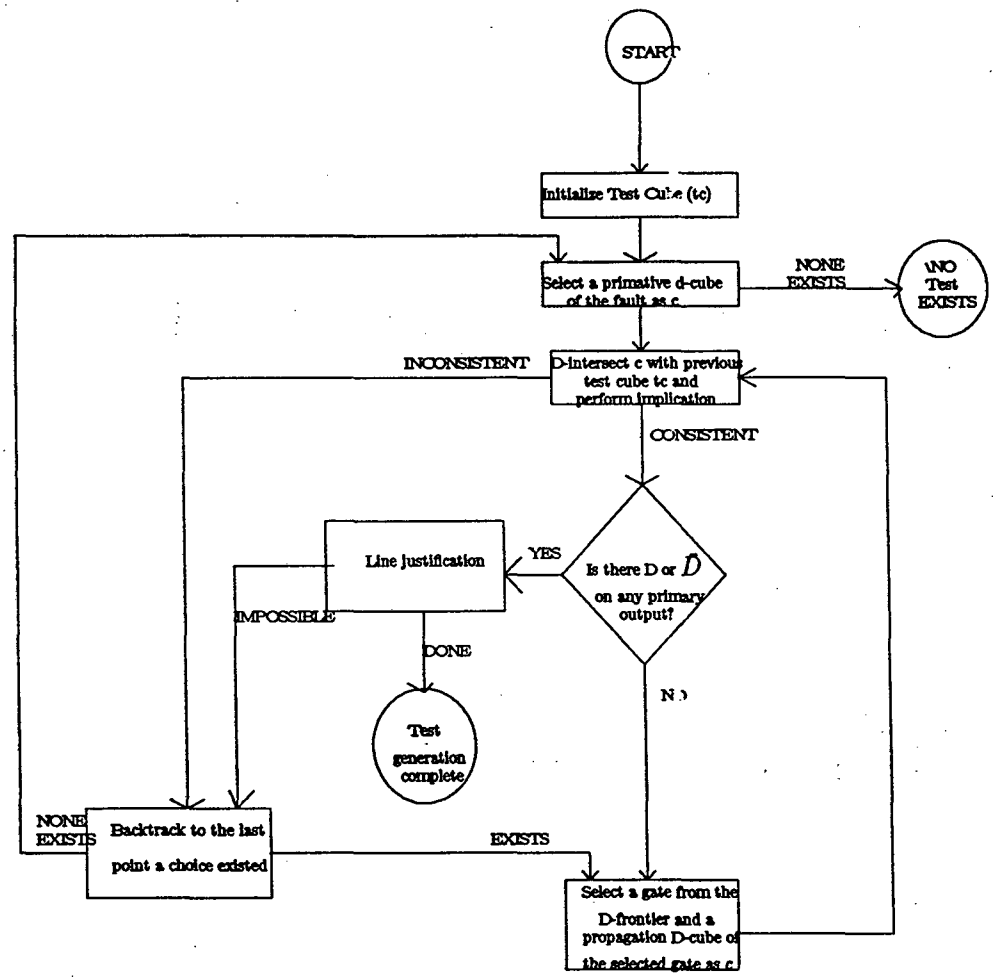
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | $\overline{D}$ | 0 | 1 | D | D | 1 | $\overline{D}$ |

The line justification step can is also a decision making process with backtracking involved if dead ends are reached. If line justification is impossible, then the D-algorithm must backtrack and try to find another D-drive step to get the faulty signal to the output. Through an implicit enumeration process, all alternatives at each decision node are examined until a test is found.

## Figure C–2: D-Algorithm Example Circuit



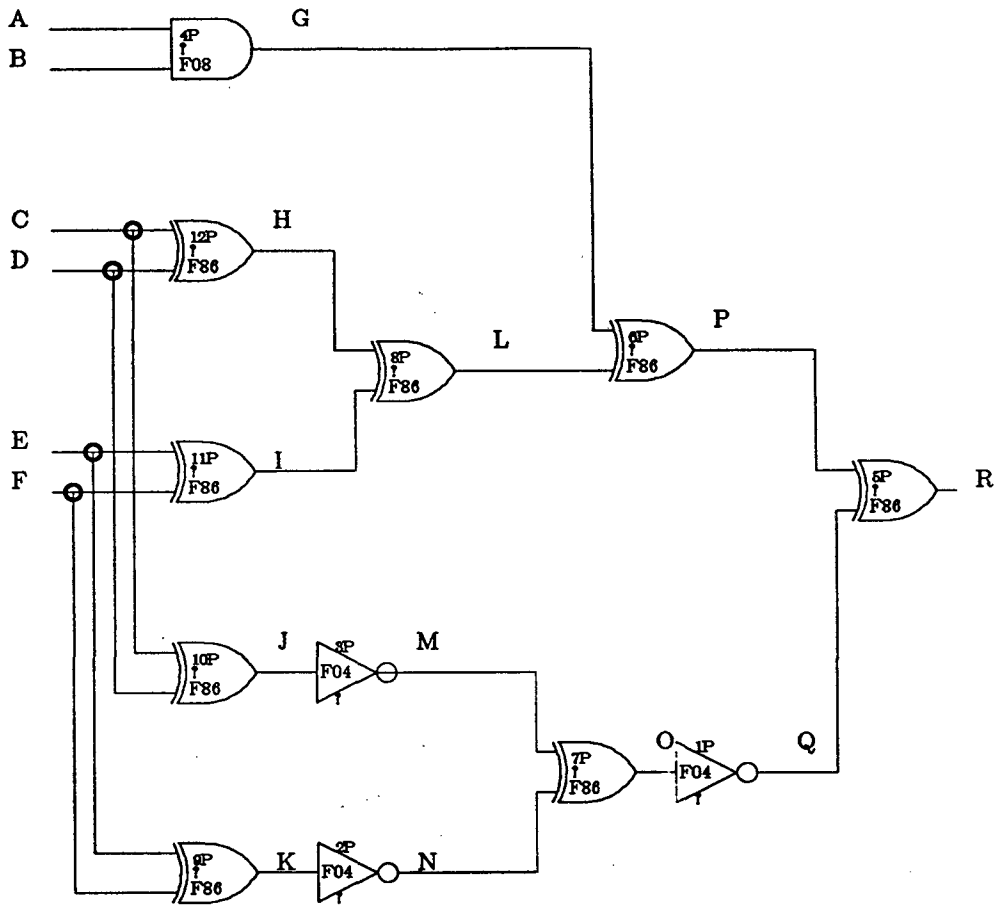
## Figure C–3: Flowchart of D-Algorithm




# PODEM Algorithm

The impetus for the PODEM algorithm was the difficulty that the D-algorithm had dealing with troublesome ECC type circuits like the one shown in figure C-4. This is due to the number of XOR gates that have reconvergence.

## Figure C–4: Troublesome ECC Circuit

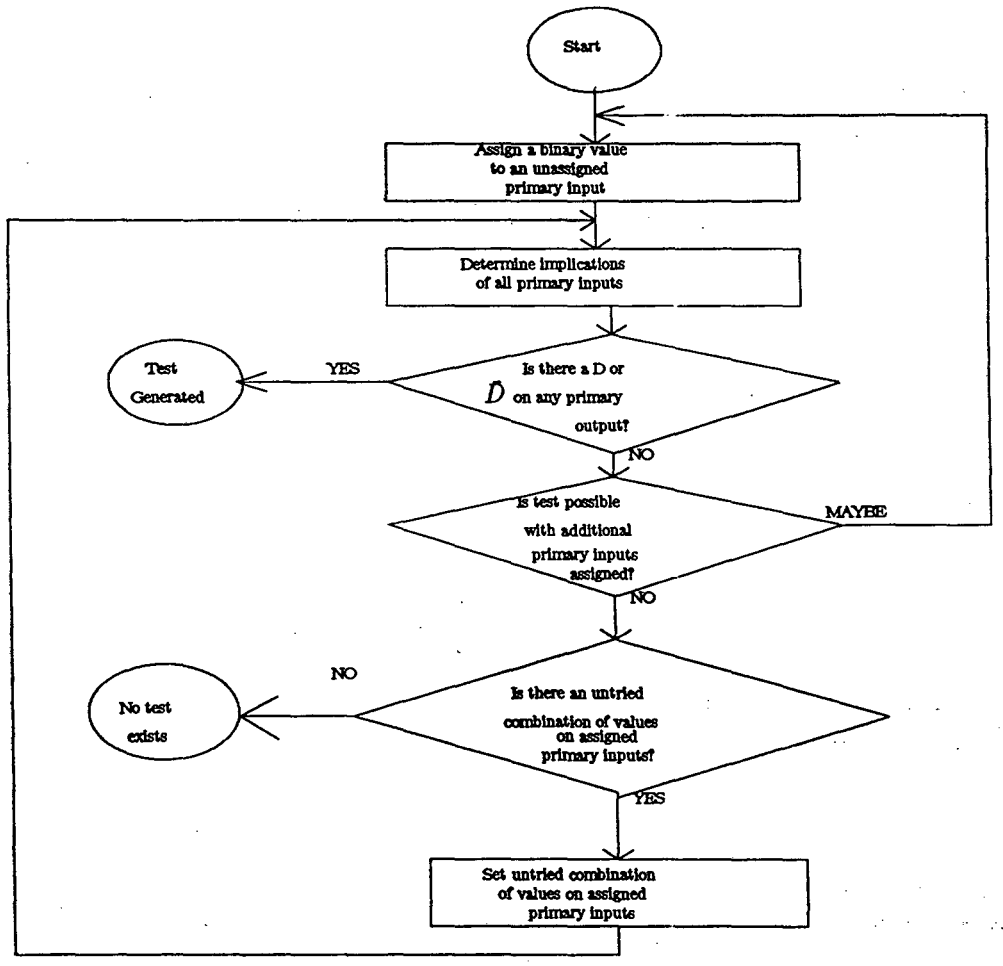



If the D-algorithm (DALG) were used to generate a test pattern to detect a s-a-0 fault at G, it would choose the primative D-cube of the fault, A=B=1 and G=D. To propagate the faulty signal to the primary output R, D-drive operations are performed by D-intersecting the test cube with the propagation D-cubes of gates along the path. After D-drive, L=1,P=$\overline{D}$, Q=1 and R=D could be obtained. Next, DALG begins to justify lines L and Q. However, since lines L and Q realize complementary functions with respect to each other, no justification is possible for the assignments of L=1 and Q=1. Thus, DALG must enumerate input values exhaustively until it determines that there is no way to justify these values. In this process, DALG creates a decision tree and examines all alternatives at each decision node. DALG will backtrack many times until it finally reaches the decision L=1 and Q=1. However, in establishing the absence of the justification, DALG must enumerate $2^3$ primary input values ($2^*$ if 2n were the number of external inputs to the XOR tree) before it can correct that bad decision made on Q, that is change the assignment on Q from 1 to 0. Consider what would happen for parity trees with up to 72 external inputs (as in real ECC circuits). For some faults the number of primary input values that must be enumerated can be of the order of $2^{36}$. It can be seen that performance of DALG would be dismal under these conditions.

Since the assignment of values is allowed to internal lines in DALG, more than one choice is available at each internal line or gate and backtracking could occur at each gate. In contrast, the PODEM algorithm allows assigning values only to primary inputs. The values assigned to the primary inputs are then propagated toward internal lines by implication. In the PODEM

algorithm, backtracking can occur only at the primary inputs. A high-level flowchart of PODEM is shown in figure C-5.

**Figure C–5: Flowchart of Podem Algorithm**



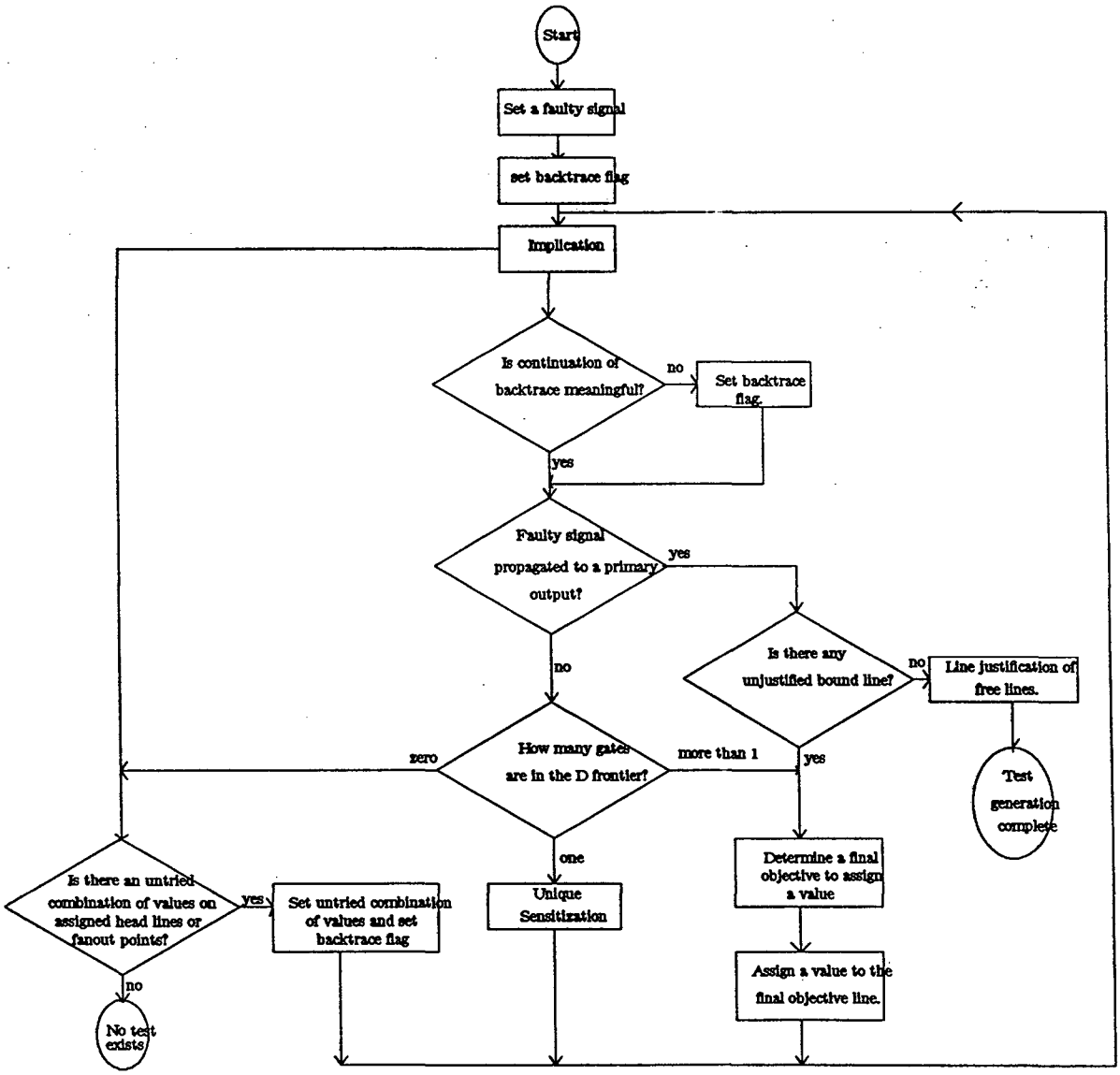

## FAN Algorithm

In order to accelerate an algorithm for test generation, it is necessary to reduce the number of backtracks and to reduce the amount of processing time required to perform a backtrack. The FAN algorithm pays special attention to fan-out points. The algorithm preprocesses the topology of a circuit and indicates areas where there are fanout-free areas. When assigning values, the FAN algorithm will defer the justification of a value into a fanout-free area because it is known that this can be done without backtracking. There is no sense doing this before it is known whether or not the current assignments are valid. Like DALG, as many signal values as possible are determined by implication after any given assignment, therefore there are chances of multiple backtracking. FAN is designed to handle the multiple backtracking. The basic strategy and heuristics employed by FAN are:

- In each step of the algorithm, determine as many signal values as possible that can be uniquely implied.
- Assign a faulty signal D or $\overline{D}$ that is uniquely determined or implied by the fault in question.
- When the D-frontier consists of a single gate, apply a unique sensitization.
- Stop the backtrace at a head line (beginning of a fanout-free region), and postpone the line justification for the head line to later.
- Concurrent backtracing of more than one path is more efficient than backtracing along a single path.
- In the multiple backtrace, if an objective at a fanout point p has a contradictory requirement, stop the backtrace so as to assign a value to the fanout point.

A high level flowchart of FAN is shown in figure C-6.

## Figure C–6: Flowchart of FAN Algorithm




## Nine-valued D-Algorithm

The original D-algorithm was based on a five value calculus. There have also been several papers written regarding test generation using a nine value calculus The table below shows the various values used in the five and nine valued calculus.

| 5-V | 9-V |
|---|---|
| 0 | 0/0 |
| 1 | 1/1 |
| D | 1/0 |
| $\overline{D}$ | 0/1 |
| X | X/X |
| | 0/X |
| | X/0 |
| | 1/X |
| | X/1 |

There are several advantages to the nine-value model. The five valued D-algorithm has to explicitly find all of the sensitized paths needed which is called multiple path sensitization. The nine-valued D-algorithm can use single path sensitization. The linear search space for single path sensitization is much smaller than the exponential search space for multiple path sensitization. This allows the search time to be reduced drastically.