REPORT CSG-25 DECEMBER 1983

CSL COORDINATED SCIENCE LABORATORY
COMPUTER SYSTEMS GROUP

A COMPREHENSIVE FAULT MODEL FOR CONCURRENT ERROR DETECTION IN MOS CIRCUITS

DANIEL LEE HALPERIN

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS: None
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release, distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): CSG-25	5. MONITORING ORGANIZATION REPORT NUMBER(S): N/A
6a. NAME OF PERFORMING ORGANIZATION: Coordinated Science Lab. / University of Illinois; 6b. OFFICE SYMBOL (If applicable): N/A	7a. NAME OF MONITORING ORGANIZATION: Office of Naval Research
6c. ADDRESS (City, State and ZIP Code): 1101 W. Springfield Avenue / Urbana, Illinois 61801	7b. ADDRESS (City, State and ZIP Code): 2511 Jefferson Davis Highway / Arlington, Virginia 22202
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Naval Electronics Syst. Comm.; 8b. OFFICE SYMBOL (If applicable): N/A	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00039-80-C-0556
8c. ADDRESS (City, State and ZIP Code): 2511 Jefferson Davis Highway / Arlington, Virginia 22202	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
N/A	N/A	N/A	N/A

11. TITLE (Include Security Classification): A COMPREHENSIVE FAULT MODEL FOR CONCURRENT ERROR DECT. IN MOS CIRCUITS

12. PERSONAL AUTHOR(S): Daniel Lee Halperin

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	December 10, 1983	206

16. SUPPLEMENTARY NOTATION: N/A

COSATI CODES		
FIELD	GROUP	SUB. GR.

17a. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Concurrent Error Detection, Fault Models, Indeterminate Faults, MOS Circuits, Physical Failure Modes, Separable Codes, Ternary Algebra, Totally Self-Checking Circuits

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A comprehensive fault model is developed for concurrent error detection in MOS integrated circuits. This fault model is based on a thorough examination of physical failures in MOS integrated circuits. Models of MOS circuits are also developed which are used to determine the behavior of these circuits under failure. It is found from this analysis that many types of physical failures may result in logic signals that are not well-defined. In particular, it is shown that physical failures may lead to constant values that are neither logic 0 nor logic 1, timing failures, or oscillation. The concept of indeterminate faults is developed to describe the behavior of such failures. It is shown that most traditional fault models are unable to model the behavior of a circuit with an indeterminate fault correctly.

Ternary algebra is used to facilitate the analysis of circuits which receive indeterminate value inputs. Using ternary algebra, necessary conditions are developed for the propagation of indeterminate values through a circuit. It is shown that in (over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED X SAME AS RPT. OTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code) / 22c. OFFICE SYMBOL: NONE

FORM 1473, 82 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified

many cases, an indeterminate value can propagate through a circuit even when a Boolean value cannot propagate.

The methodology of totally self-checking systems is used to provide concurrent error detection. It is shown that the traditional definitions of the totally self-checking property are inappropriate for failures which include indeterminate faults. A new definition of the totally self-checking property is developed which is compatible with indeterminate faults. It is shown that under our fault models, duplication may be used to provide a totally self-checking implementation for any function. Procedures are developed to determine if a function has an implementation using a separable code which may provide concurrent error detection at a lower cost than duplication. Issues involved in the interconnection of several totally self-checking circuits are considered, as well as the requirements for checkers in systems which may experience indeterminate failures.

A COMPREHENSIVE FAULT MODEL FOR CONCURRENT ERROR DETECTION IN MOS CIRCUITS

BY

DANIEL LEE HALPERIN

B.S., University of Tennessee, 1978
M.S., University of Illinois, 1981

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1984

Urbana, Illinois

A COMPREHENSIVE FAULT MODEL FOR CONCURRENT ERROR DETECTION IN MOS CIRCUITS

Daniel Lee Halperin, Ph.D.
Department of Electrical Engineering
University of Illinois at Urbana-Champaign, 1984

A comprehensive fault model is developed for concurrent error detection in MOS integrated circuits. This fault model is based on a thorough examination of physical failures in MOS integrated circuits. Models of MOS circuits are also developed which are used to determine the behavior of these circuits under failure. It is found from this analysis that many types of physical failures may result in logic signals that are not well-defined. In particular, it is shown that physical failures may lead to constant values that are neither logic 0 nor logic 1, timing failures, or oscillation. The concept of indeterminate faults is developed to describe the behavior of such failures. It is shown that most traditional fault models are unable to model the behavior of a circuit with an indeterminate fault correctly.

Ternary algebra is used to facilitate the analysis of circuits which receive indeterminate value inputs. Using ternary algebra, necessary conditions are developed for the propagation of indeterminate values through a circuit. It is shown that in many cases, an indeterminate value can propagate through a circuit even when a Boolean value cannot propagate.

The methodology of totally self-checking systems is used to provide concurrent error detection. It is shown that the traditional definitions of the totally self-checking property are inappropriate for

failures which include indeterminate faults. A new definition of the totally self-checking property is developed which is compatible with indeterminate faults. It is shown that under our fault models*, duplication may be used to provide a totally self-checking implementation for any function. Procedures are developed to determine if a function has an implementation using a separable code which may provide concurrent error detection at a lower cost than duplication. Issues involved in the interconnection of several totally self-checking circuits are considered*, as well as the requirements for checkers in systems which may experience indeterminate failures.

Acknowledgment

The author wishes to express deepest thanks to his advisor, Dr. Ed Davidson, for his support, encouragement, and advice throughout this thesis project. He is deeply appreciative for the many long hours Dr. Davidson spent assisting with this thesis. The author would also like to thank Dr. Janak Patel, Dr. Jacob Abraham, and Dr. Dan Gajski for serving on his final committee. Finally, the author would like to thank his colleagues in the Computer Systems Group for their support, ideas, and friendship.

The author would like to dedicate this thesis to his parents, Joseph and Sita Halperin, and to his wife, Beverly. The support, sacrifices, and understanding provided by the author's family have made his education in general and this thesis in particular possible.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

vi

LIST OF FIGURES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1. Error Detection Strategies

As integration levels increase and more and more devices are placed on an integrated circuit, it becomes increasingly difficult to insure that a circuit and the system it is part of are operating properly. There are two basic approaches to this problem: off-line testing and concurrent error detection.

In off-line testing, the system is stopped periodically and a test procedure is performed. This test may be performed by the system itself, or an external tester may be used to stimulate the circuit and check its results. If the system successfully completes the test, then the assumption is made that the system is operating correctly. If the system fails the test, then the system is faulty. In this approach, since it is unknown exactly when the system failed, all computations performed since the last successful test procedure must be presumed erroneous.

The main advantage of using off-line testing is its simplicity. In most cases, only a very modest amount of additional on-chip hardware is required. Unfortunately, there are also many disadvantages. Because of the poor observability and controllability of VLSI circuits, it is very difficult to derive a test procedure that will completely test an entire

integrated circuit. Often it is necessary to add additional logic on the integrated circuit to increase its controllability and/or observability [13]. In addition, during the time the system is off-line for testing, it cannot perform any useful computation and thus system throughput is degraded. Since there is no way of pinpointing exactly when a failure has occurred, all results produced since the last successful testing procedure must be discarded. Alternatively some type of check-pointing scheme can be used. This approach involves saving enough of the system state and data so that all computations performed since the last successful test procedure can be repeated. The most serious drawback of off-line testing, however, is its inability to protect against intermittent errors. It has been reported [2] that between 90 and 98 percent of failures in computers are nonpermanent in nature. Off-line testing gives little if any protection against nonpermanent failures. Therefore, for any system in which we must immediately know when a failure has occurred (i.e., any type of real-time system) or for any system in which we expect a major fraction of errors to be intermittent, off-line testing is inadequate.

The second approach to this problem is concurrent error detection. In this approach, the system is divided up into one or more blocks called modules. The inputs (including both data and control vectors) and outputs of each module must be encoded with an appropriate code. Obviously, such encoding requires additional logic. These codes are selected so that when most failures occur, the result of a computation will either be correct or a non-codeword. Checkers are placed at the

output of each module. These checkers are used to detect non-codewords and thus indicate an error.

Concurrent error detection has several advantages. When an error occurs, the checkers immediately provide an error indication. With off-line testing, an error indication is only given after the off-line test procedure is performed. The lack of information concerning the precise time at which the failure occurred requires computations to be repeated. An immediate error indication eliminates the need to repeat computations. Protection is also provided against intermittent failures. If an intermittent failure results in an error, it will be detected. Therefore, concurrent error detection is well suited for real time systems and any system in which intermittent failures are a significant percentage of total failures.

The presence of checkers can greatly increase the observability of the circuit. If enough checkers are used, it is possible to completely or very nearly completely test a circuit simply by normal operation. Complete testing during normal operation prevents a buildup of undetected failures (the so-called "latent faults" problem). Since any concurrent error detection technique can only handle a limited number of failures, a buildup of latent faults can result in an error not being detected. If the checkers do not provide enough observability to detect all possible faults during normal operation, periodic testing must be used to detect any latent faults.

The major disadvantage of concurrent error detection is the additional logic required. The codes used for data and control vectors

require redundant bits. Extra logic is needed to process these bits. Additional logic is also needed for checkers. The logic which must be added to implement concurrent error detection can be significant. Depending on exactly which concurrent error detection scheme is used*, the additional logic required may be more than 100 percent of the original system. Whether this type of extra cost is justified is obviously an engineering judgment. It is possible to use only concurrent error detection for those parts of the system which are either judged most likely to fail or whose failure would be most serious. Depending on what portions of the circuit are protected, significant savings of hardware are possible. A technique has been developed recently for various arithmetic computations [3]. This technique employs time redundancy rather than logic redundancy. Although it is not applicable to all functions, time redundancy, where it is applicable, can provide concurrent error detection with only a very modest amount of additional logic but at the cost of additional time.

1.2. Fault Models

The purpose of a fault model is to describe the behavior of a physical failure in a manner that will allow us to predict the logical behavior of the failed system. Since in general, a physical failure affects the analog behavior of a circuit (i.e., gain, time constants, etc.) it may be very difficult to describe exactly how the failure will alter the logical behavior of the system. A fault model has three important attributes: accuracy, ease of analysis, and cost of fault tolerance.

If the fault model does not accurately describe the logical behavior of physical failures, then it is of little use. The quality of an error detection scheme is measured by the fraction of faults in the fault model which are detectable. Clearly, if the model does not accurately describe the behavior of physical failure, this measure is of little use.

Two factors contribute to the ease of analysis of a fault model: the number of faults which must be considered, and the complexity of the fault behavior. Any system which contains many thousands of logic elements will also have a large number of possible faults. The behavior described by the fault model must be simple enough to allow analysis of the system. For off-line testing, we must determine whether the test procedure will detect each fault. For concurrent error detection, we must insure that the encoding used will allow detection of an incorrect result. If the fault model is too complex, this analysis will be too difficult to perform and the fault model will be impractical. One technique which can greatly reduce the number of faults is fault collapsing. Fault collapsing can be done when two or more faults are indistinguishable. Fault collapsing makes it is possible to reduce significantly the number of faults which need to be considered.

Cost of fault tolerance is a very important consideration since it strongly affects system cost. For off-line testing, cost of fault tolerance determines how large the test procedure must be. It may also influence the complexity of the tester hardware. For concurrent error detection, cost of fault tolerance determines how much extra logic must

be added to the original system. Cost of fault tolerance is usually highly dependent on the exact nature of the error detection scheme and the target system.

The selection of a fault model requires a tradeoff between accuracy, ease of analysis, and cost of fault tolerance. Since these requirements are usually conflicting, the choice is never easy. In the past, a variety of fault models have been proposed.

1.2.1. Single Stuck-At Fault Model

The single stuck-at fault model assumes that any physical failure will cause one node (wire) of the circuit to become permanently either a logic 1 or a logic 0. This model is extremely easy to use and is by far the most common fault model in use. It was first proposed when logic elements were built from discrete devices and is generally accurate in describing the behavior of failures in such devices [4]. Unfortunately, its accuracy is much poorer for the highly integrated logic elements which make up most of today's systems.

1.2.2. Unidirectional Fault Model

The unidirectional fault model assumes that a failure causes any number of nodes in the circuit to be either stuck-at 1, or alternatively any number stuck-at 0. Smith [5] has shown that a unidirectional fault model implies the use of an unordered code (i.e., no codeword covers any other codeword) for concurrent error detection. He also showed that in most cases, concurrent error detection of unidirectional faults requires an inverterless implementation. Since nearly all logic families are

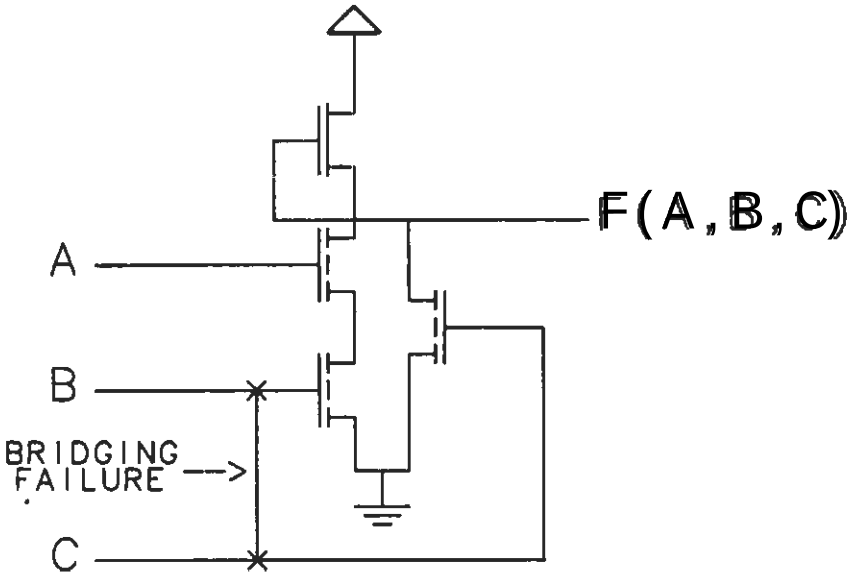
inherently inverting, this restriction severely limits the usefulness of this fault model.

A related fault model which is quite popular assumes that any physical failure results in a unidirectional error at the module's output. In general, inverter-free implementations are not required to allow concurrent error detection for such a system. An unordered code, however, is still required. This fault model has been very popular for various structured elements such as memories and programmed logic arrays. We will refer to this fault model as the unidirectional error fault model.

1.2.3. Bridging Fault Model

The bridging fault model assumes that a short between any two or more lines results in some sort of wired logic function. For NMOS and CMOS logic families, the wired logic operation is usually taken to be the AND operation. It is assumed that if any of the lines which are shorted together are a logic 0, then all the shorted lines will take on the value of a logic 0. If all lines have a value of logic 1, then the lines will retain a value of logic 1. Figure 1.1 shows an example of a bridging fault between two input lines resulting in a wired AND operation.

The behavior of a circuit under failure is much more complicated with this model than with the stuck-at fault model. A bridging fault results in an additional gate being added to the circuit. More importantly, a bridging fault can transform combinational logic into sequential logic. The bridging fault model is only useful for modeling shorts



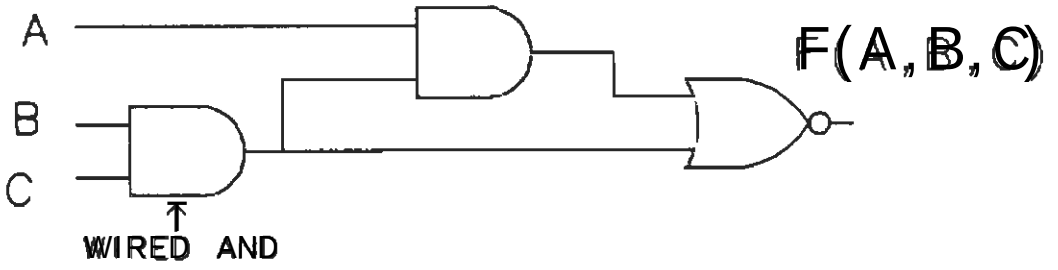


Figure 1.1. An Example of a Wired-AND Bridging Fault.

between lines. For this reason, it is usually combined with another fault model such as the stuck-at fault model.

1.2.4. Stuck-Open Fault Model

The stuck-open fault is peculiar to MOS logic families. A stuck-open fault results from a physical failure in which some node in the circuit is prevented from having a DC path to ground or power for certain input combinations.

Figure 1.2 shows an example of a stuck-open fault in a CMOS NAND circuit. Due to a physical failure, the pullup transistor corresponding to input A is permanently in the nonconducting state. Whenever input A is a logic 0 and input B is a logic 1, there is no DC path from the output node to either power or ground. The output node therefore remains at its previous value until the inputs are changed to re-establish a DC path to power or ground or until the charge leaks off the output node. The time required for a significant amount of charge to leak off the output node is usually much longer than the system clock period.

Static NMOS and PMOS gates are not subject to stuck-open faults. However, if pass transistors are utilized to implement certain logic functions, stuck-open faults can occur. Figure 1.3 shows an NMOS inverter whose input is loaded by a multiplexer with two pass transistors. The pass transistor corresponding to input A is permanently nonconducting due to a physical failure. If the control input is a logic 1, then there is no DC path from the gate of the inverter to power or ground (note that this path is normally provided by the gates that drive

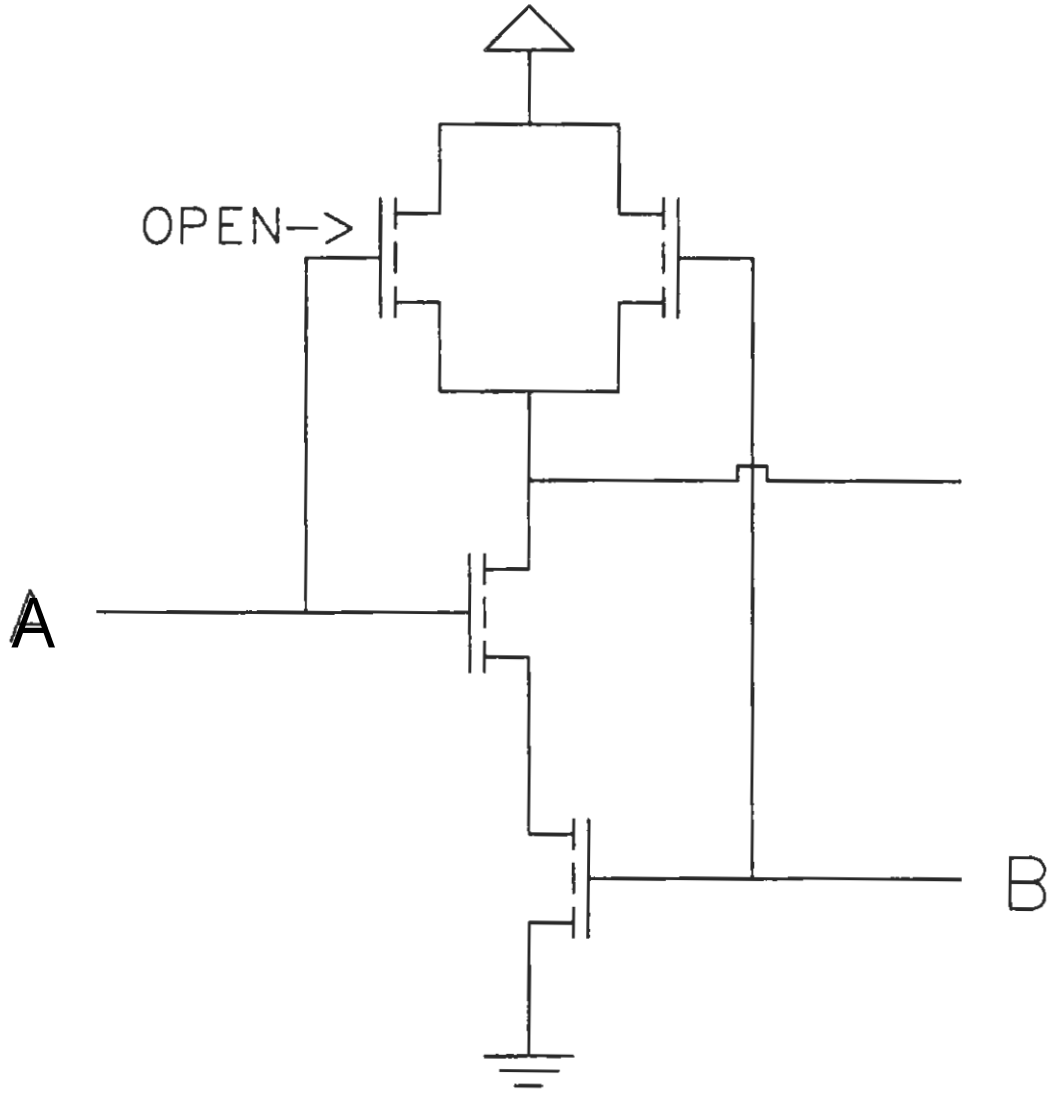


Figure 1.2. A CMOS NAND Gate with a Stuck-Open Fault.

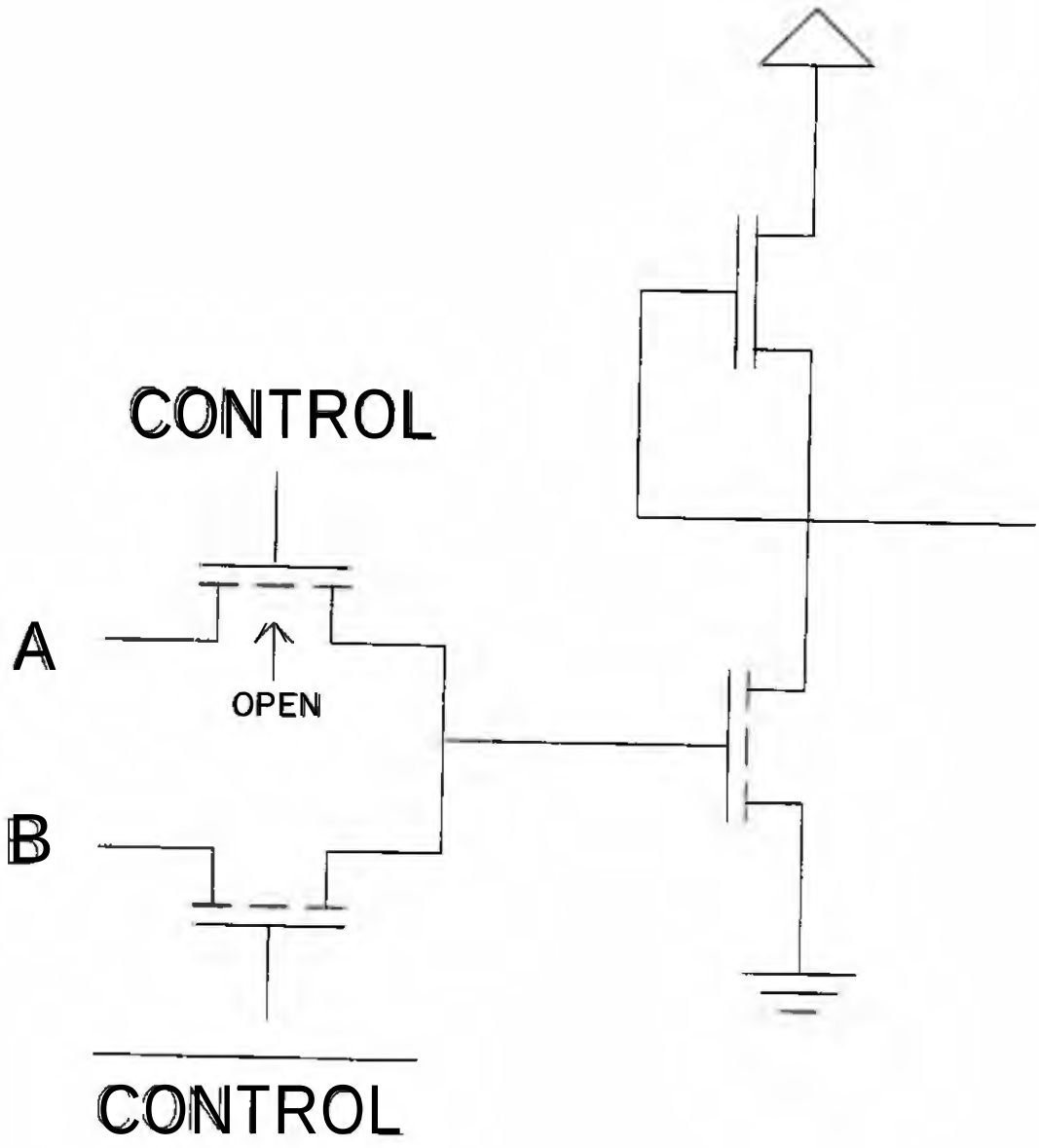


Figure 1.3. A NMOS Circuit with a Stuck-Open Fault.

inputs A and B). Once again, the output remains unchanged from its previous value.

For both CMOS and NMOS circuits, a stuck-open fault can transform a combinational circuit into a sequential circuit (under failure, the present output is a function of a previous input). Therefore, the stuck-open fault model suffers from the same deficiencies as the bridging fault model. It is difficult to use because of the possibility of sequential operation. It also needs to be combined with some other fault model since it can only model transistors that have permanently failed in a nonconducting state.

1.3. Overview of Research

The choice of a fault model is of crucial importance to any error detection scheme. Most of the fault models that have been proposed, were proposed long before the advent of large scale MOS integrated circuits. It is important that any fault model reflects the technology with which it is used.

We begin the presentation of our research in Chapter 2, by thoroughly reviewing the types of physical failures which are possible in present day MOS integrated circuits. We also examine the effects of scaling of device dimensions, voltages, and doping levels on the probability that a failure occurs.

In Chapter 3, models of several types of MOS inverters are developed. These models are used to study the effects of physical failures on MOS logic circuits.

In Chapter 4, fault models are defined based on the results from Chapters 2 and 3. The techniques required to analyze a circuit's concurrent error detection capabilities are also developed in Chapter 4. Finally, the hardware requirements of implementing concurrent error detection for our fault model are examined.

The purpose of this research is to find fault models for MOS circuits which are better than the traditional fault models. We have already defined the criteria for judging a fault model: accuracy, ease of analysis, and cost of fault tolerance. The results of Chapters 2 and 3 can be used to judge a fault model's accuracy for MOS logic circuits. The results of Chapter 4 are useful for judging the ease of analyses and the cost of fault tolerance for our fault model. The results of' this research show that fault models that are much more accurate than the traditional fault models are possible to use without sacrificing ease of analysis and cost of fault tolerance.

CHAPTER 2

Physical Failure Modes for MOS Integrated Circuits

In this chapter, we examine the various physical failure modes for MOS integrated circuits. We restrict our study to MOS circuit technologies because of its wider use in VLSI circuits.

One important consideration in analyzing failure mechanisms is the effect of future changes in technology. One such change is called scaling. Scaling is the process of reducing integrated circuit dimensions, doping, and voltages. Generally scaling results in denser integrated circuits that operate at a higher speed and consume less power. Most of the improvements in MOS integrated circuits over the past 15 years are due to scaling. There is every reason to believe that in the future, devices will continue to be scaled even further. As a result, effects which were unimportant in the past, will become of much greater concern.

The simplest scaling scheme is to reduce all dimensions, both horizontal and vertical, by a factor of K. Power supply voltages are also reduced by the same factor K while doping densities are increased by K. Because of this, the size of any device is reduced by a factor of K^2. Therefore, the number of devices which can be placed on an integrated circuit of a given size can be increased by a factor of K^2. The power consumed by a scaled device is also reduced by a factor of K^2 and the propagation delay is reduced by a factor of K. Current density in conductors, however, increases by a factor of K. This type of scaling is

referred to as constant field scaling since the magnitude of all electric fields remains approximately constant.

A majority of integrated circuits are designed to operate with a power supply of 5 volts. Since adding an additional power supply to a system is quite expensive, it is usually considered to be impractical to scale the power supply voltage. If all dimensions are reduced by a factor of K, but the power supply voltage is held constant, power per device increases by a factor of K while current density increases by a factor of K^3. If we take advantage of the fact that we can place K^2 times as many devices on an integrated circuit of the same size, total power increases by a factor of K^3 also. Clearly, as devices are scaled down, power and current density will be of concern. This type of scaling is referred to as constant voltage scaling. Figure 2.1 gives a summary of the scaling factors for both constant voltage and constant field scaling.

2.1. Interconnect Failures

Interconnect is that part of the circuit which connects transistors to other transistors and the input or output pads. Most MOS integrated circuit processes provide one or more levels of metal, a layer of polysilicon, and a layer of diffusion. All of these layers may be used for interconnect although polysilicon cannot be allowed to cross diffusion since an unwanted transistor will be formed. If, however, the process also provides for an enhancement transistor with a low enough threshold voltage, then the unwanted transistor may be made into an enhancement transistor. This allows polysilicon interconnect to cross

	CONSTANT FIELD	CONSTANT VOLTAGE
DIMENSIONS	K^{-1}	K^{-1}
VOLTAGE	K^{-1}	1
DOPING CONCENTRATION	K	K
NUMBER OF DEVICES	K^2	K^2
POWER PER DEVICE	K^{-2}	K
TOTAL POWER	1	K^3
CURRENT DENSITY	K	K^3
CONTACT RESISTANCE	K^2	K^2
NORMALIZED CONTACT VOLTAGE DROP	K	K^2

Figure 2.11 Summary of Scaling Factors.

diffusion interconnect, although the capacitance of the polysilicon line and the resistance of the diffusion line is significantly increased. This in turn increases the delay of a signal propagating on either line. The ability of polysilicon to cross diffusion is very important when only one layer of metal is available. As we will see in the next chapter, a sufficiently low enhancement transistor threshold voltage will have an impact on system performance.

2.1.1. Metal Interconnect Failures

It is well known that any metalization subjected to a high current density is susceptible to electromigration [6]. Electromigration typically occurs where there is a slight constriction in the conductor. The current density is highest at the constriction. The high current density causes metal ions to diffuse away from the constriction. This diffusion further narrows the conductor which in turn raises the current density and thus continuously accelerates the process. Eventually, the conductor fails. Lines subjected to DC current are most susceptible while lines subjected to AC current are essentially immune to electromigration. Nanosecond pulses of current (all pulses of the same polarity) two orders of magnitude higher than the DC case may be safely carried by metal conductors. CMOS and dynamic NMOS circuits which dissipate no static power are thus less likely to suffer electromigration failure.

A variety of factors affects the mean time to failure of a metal line. These include materials, grain size and orientation, and relative width and length of the conductor [7]. The most important factor, however, is current density. The mean time to failure for a line is given

by the formula:

$$MTTF = K_1 \cdot J^{-N} \exp(K_2/T)$$

where K_1 and K_2 are constants, J is the current density, N is a material dependent constant, and T is temperature. The value of N for aluminum is generally considered to be 2 (there is some disagreement on this point, see [7]). Therefore, the mean time to failure is inversely proportional to the current density squared and exponentially related to the reciprocal of temperature. For this reason, scaling will have an important (and unfortunately negative) impact on the reliability of metal conductors. If the power supply voltage is not scaled (constant voltage scaling), we have already stated that both power consumption and current density will increase by a factor of K^3 as the dimensions are scaled by a factor of K. Due to the current density alone, the mean time to failure for aluminum will scale down by a factor of K^6. Since we also increase the number of metal interconnects by a factor of K^2 by scaling, then the mean time to failure for the entire integrated circuit will decrease by an additional factor of K^2. Therefore, ignoring the effects of temperature, we can expect the mean time to failure of an entire integrated circuit to decrease by a factor of K^8 if aluminum metalization is used.

The temperature that an integrated circuit operates at is highly dependent on power consumption, packaging, and external cooling. Reducing the temperature by packaging improvements or adding external cooling tends to be expensive. Therefore, if power consumption is increased by a factor of K^3 during the scaling process, it is reasonable to expect at

least some increase in integrated circuit temperature. Since the relationship between temperature and conductor lifetime is exponential, relatively small increases in temperature will drastically reduce mean time to failure. One should note that if the power supply voltage is scaled along with the dimensions, the integrated circuit mean time to failure only decreases by a factor of K^4. Also, since total power remains constant, the integrated circuit temperature should also remain constant.

Accumulation of metal from electromigration presents another problem. This metal can form hillocks or whiskers [6]. Whisker formation tends to occur where there is a high electric field between conductors. The formation of hillocks and whiskers can result in shorting between adjacent metalization and cracking of the passivation level.

Ohmic contacts are formed where metalization must provide an electrical connection to a diffused area. Ohmic contacts ideally should produce no rectification or other asymmetry in the response to positive and negative waveforms. In addition, the resistance of contacts should be as low as possible. Ohmic contacts are used extensively in integrated circuits. Unfortunately, they appear to be a major problem area for future integrated circuits. Since the resistance of a contact is proportional to its area, the contact resistance will increase by a factor of K^2 during scaling. If supply voltage is reduced by the process of scaling, then normalized contact voltage drop (i.e., signal voltage divided by supply voltage) increases by a factor of K^2. If the power supply voltage remains constant, then normalized contact voltage

increases by a factor of K (see Figure 2.1). Any mask misalignment during processing will aggravate this situation since the effective area of the contact will be further reduced.

In addition to these scaling problems, a variety of effects due to electromigration can also lead to failures [6]. At fairly high temperatures, it is possible for silicon to leave the substrate and form an alloy with the aluminum. This depletion of silicon decreases the effective junction depth and thus makes it easier for spikes of the aluminum-silicon alloy to extend through the junction and into the substrate. This results in a short from the metal and diffusion to the substrate. It should be noted that junction depth is one of the dimensions which is reduced in the scaling process. Thus, scaling makes it easier for a spike to penetrate past the junction. The metalization of the contact is also susceptible to electromigration resulting in open contacts [8].

For most integrated circuit processes, metal forms the top layer. For this reason, the metalization will tend to be three dimensional as it crosses over features on lower layers. Any time metal has to go up or down steps on the surface of an integrated circuit, there is the possibility of either a break in the line, or a constriction. Obviously such a constriction is a prime site for electromigration to occur. A defect in the metalization mask can cause either a short or open in the metal interconnect depending on the defect.

Many of the metals used in integrated circuits are subject to corrosion (particularly aluminum) and accelerated electromigration from any

moisture or other contaminants [9]. Ideally, packaging should provide an almost impervious barrier to such contaminants. If the packaging should fail to perform this task, all metal on the integrated circuit is subject to failure.

2.1.2. Polysilicon Interconnect Failures

Polysilicon also appears to be vulnerable to electromigration [10]. The physical mechanism, however, seems to be somewhat different than for metal. In polysilicon, a high current density usually causes the dopant atoms to migrate rather than the silicon atoms. This migration results in areas with a lower concentration of dopant atoms. The resistance of polysilicon is very sensitive to doping levels. Therefore, the resistivity of the polysilicon increases in the areas where electromigration has left low concentrations of dopants. This leads to the formation of local hot spots which can further accelerate the electromigration process. Eventually, thermal runaway causes the line to fail. It should be noted that at extremely high temperatures which can occur with thermal runaway (temperatures greater than 1000 ^{0}C have been observed in polysilicon test structures [10]), silicon atoms start to migrate as well as dopant atoms. Usually silicon migration only occurs immediately before conductor failure.

It appears that electromigration becomes an important source of failures at current densities of 10^{6} A/cm^{2} [11] (approximately the same as for metal lines). Fortunately, when polysilicon is used as interconnect, it will seldom be subjected to DC current densities of this magnitude. It must be kept in mind, however, that just as for metal, current

density scales by a factor of K or K^3, depending on which scaling rules are used. Contacts between metal and polysilicon are subject to the same type of difficulties as the metal-diffusion contacts we have already discussed. Once again, since the currents will tend to be lower than for metal-diffusion contacts, there should be fewer problems.

2.1.3. Diffusion Interconnect Failures

When diffusion areas are formed on an integrated circuit, we are depending on a reverse biased pn junction to prevent the diffused area from shorting to either the substrate or other diffused areas. With a properly designed and manufactured integrated circuit, the breakdown voltage of the pn junctions is well above any voltage difference which the circuit will be subjected to. It is possible for various anomalies to result in significantly lower breakdown voltages. Possible causes include local crystal defects, changes in doping concentration, exposure to radiation and excessively shallow diffusions. Radiation can also increase the leakage current of a pn junction. Leakage current also has an exponential dependence on temperature. Regardless of the cause, if the pn junction should break down or if leakage current should become excessive, the diffusion area will become shorted to the substrate.

It is also possible for two closely spaced diffusion areas to become shorted together. This occurs if the depletion regions of the reverse biased pn junctions should happen to overlap. In this case, charge carriers in one diffusion area will be swept by any potential difference across the overlapped depletion regions, to the other diffusion area. The width of a depletion region is approximately

proportional to the reverse bias voltage. Therefore, the depletion regions of two adjacent diffusion areas are most likely to overlap when they are at their most positive voltage (most negative voltage for PMOS). In this case, however, both areas will be at the same potential. Such a short should have little effect except on circuits which are highly dependent on the relative capacitances of nodes (such as dynamic circuits). A more serious although less likely problem occurs if the two adjacent areas are at significantly different potentials. In this case, if the depletion regions overlap, it will be possible for significant currents to flow between the two areas. It is becoming more common to use a recessed field oxide. Recessed field oxide has several advantages which include lower capacitance and improved surface planarity. In addition, since a pn junction is only formed at the bottom of a diffusion area, it is virtually impossible for the depletion regions of two adjacent diffusion regions to overlap. If an insulating substrate is used, then isolation failures should not be an issue.

It is often the case that interconnect will run over the oxidized substrate between two diffusion areas. The result is a parasitic MOS transistor. The diffusion areas form the source and drain while the interconnect forms the gate. If the parasitic MOS transistor is allowed to turn on, an unwanted current flows between the two diffusion regions. In other words, the diffusion regions are shorted together by the parasitic transistor. To prevent this from happening, the field oxide is made thick enough to prevent the parasitic transistor from turning on. Similarly, the substrate under the field oxide is often implanted to make a channel even harder to form. If enough charge (due to

radiation*, mobile ions*, etc.) becomes trapped in the field oxide*, a channel may still form*, especially when the interconnect is at its most positive voltage (most negative voltage for PMOS) [12].

2.1.4. Dielectric Failures

In MOS integrated circuits*, silicon dioxide (SiO_2) is the most common dielectric*, although silicon nitride (Si_3N_4) is also used occasionally. The dielectric material is used for two important purposes: insulation and protection.

The dielectric must separate any two conducting layers from each other. One very important use of a dielectric is in the gate oxide which insulates a transistor's channel from its gate electrode. Almost all MOS circuits depend on the extremely high gate impedance of a MOS transistor. The smallest pinhole in the gate oxide can result in a short from the gate electrode to either the source diffusion*, channel*, or drain diffusion (depending on where the pinhole is). Gate electrodes which are connected to Input/Output pins are of particular concern. Simply handling an integrated circuit will subject the pins to electrostatic discharge. Three sources of electrostatic discharge as reported in [13] are:

(1) A charged person touches a device and discharges the stored charge to or through the device to ground.

(2) The device itself*, acting as one plate of a capacitor*, can store charge. Upon contact with an effective ground the discharge pulse can create damage.3

(3) An electrostatic field is always associated with charged objects. Under particular circumstances*, a device inserted in this field can have a potential induced across an oxide that creates breakdown.

Clearly, electrostatic discharge is not limited to situations where the device is being handled. It may also occur while the integrated circuit is in use. An electrostatic discharge can easily generate a potential difference of 1000 or more volts [13]. Due to the high input impedance of a MOS transistor, there is no way for the static charge to leave the gate electrode. Since the gate oxide typically has a breakdown voltage on the order of 100 volts or less, electrostatic discharge leads to breakdown of the gate oxide. Since the gate oxide thickness is typically reduced during the scaling process, it is reasonable to expect gate oxide breakdown to occur at even lower voltages. For silicon dioxide, this breakdown is permanent resulting in either a resistive short or a diode short between the gate and source, drain, or channel. The type of short is determined by whether the gate is of the same type or opposite type as the material it is shorted to [14]. If both materials are of the same type, the short will be resistive. If they are of opposite types, the short will be a diode.

Due to the susceptibility of MOS to electrostatic discharge, it is standard practice to use protective circuits on all Input/Output pins. Many different circuits have been proposed, but they typically use two diodes (or the functional equivalent). These diodes are biased so that any time the pin voltage goes significantly outside the range of ground to power supply, one of the two diodes conducts providing a path for charge to leak off the gate. Even though such circuits lower the probability that electrostatic discharge will destroy a transistor, they do not provide complete protection.

Studies have examined the susceptibility of gate oxide, both with and without protection circuits, to electrostatic discharge[14,15]. In both cases, the failure mechanism appears to be cumulative. That is, the more stress the oxide has been exposed to in the past, the higher the failure rate.

As we have previously discussed, electromigration may result in the accumulation of metal which can crack dielectric layers. Another possible source of failure is due to differences in the coefficient of expansion of the dielectric and substrate or interconnect.

Usually, one of the last steps in fabrication before dicing and packaging is covering the integrated circuit with a thick layer of dielectric material. This layer is called the passivation layer and along with the other packaging is responsible for protecting the integrated circuit both mechanically and chemically. It must protect the surface of the integrated circuit from scratches during the packaging procedure and seal out any moisture or other chemicals which could cause corrosion of the metalization. In addition, it must prevent ions from diffusing close to the substrate. Any such ions can change the threshold of a transistor or allow the substrate under the field oxide to invert. If metal interconnect crosses any two diffusion areas, A parasitic MOSFET is formed. Normally this transistor will be off. If the substrate under the field oxide should invert, then the MOSFET is turned on and the two diffusion regions are now shorted together by the parasitic MOSFET.

2.2. Transistor Failures

Transistors are responsible for providing the switching action which allows a circuit to implement a Boolean function. There are a variety of parameters which control the operation of a transistor. Any change in these parameters affects the ability of circuits to perform a desired switching operation. If a transistor is allowed to break down, uncontrolled currents will flow through the transistor. This also leads to circuit failure.

2.2.1. Parameter Shift Failures

The two most important parameters of a MOS transistor are threshold voltage and transconductance. The threshold voltage is the gate to source voltage which causes an enhancement mode transistor to go from the nonconducting state to the conducting state. The transconductance is a measure of how much the transistor's conductance changes due to a change in the gate to source voltage. Transconductance is defined as the partial derivative of drain current with respect to gate to source voltage. Both of these parameters are of great importance to the transient and steady state responses of MOS logic circuits.

An important source of parameter shifts in a MOS transistor is hot electron injection. Electrons in a high electric field can be accelerated to a very high velocity. Because of the direction of the electric field in the area of the channel pinch-off region, any hot electrons generated in this area will be directed toward the gate oxide. Some of these electrons will have a sufficient energy to overcome the potential barrier between the silicon and silicon dioxide. Of these

electrons, a fraction will be trapped in the oxide as the remaining electrons proceed to the gate electrode. Whether or not an electron will enter the oxide and the fraction of such electrons that become trapped depends on a variety of factors. Such factors include temperature, electrode potentials, doping levels, and device dimensions [16,17]. The buildup of negative charge in the gate oxide will eventually cause a shift in both threshold voltage and transconductance [18,19]. Scaling will increase the likelihood of hot electron failures. If constant voltage scaling is used, the higher electric fields will increase the number of electrons injected into the oxide. If constant field scaling is used, circuits will be more sensitive to parameter shifts.

Mobile ions can be introduced during processing or by a packaging failure. These ions will move in response to electric fields which will result in threshold voltage and transconductance varying with age. Moisture in the passivation layer has been found to cause similar results [20].

Another cause of parameter shifts is exposure to ionizing radiation. Such radiation can be of many different forms including X-rays, alpha particles, cosmic rays, and high energy sub-atomic particles such as electrons, protons, and neutrons. The effects of such radiation includes damage to the crystal lattice, photo currents, and most importantly, the accumulation of static charge in the oxide [21,22]. This charge leads to threshold voltage shifts and decreases in transconductance. It has also been shown [23] that radiation can increase the

noise level in transistors long before any shift in threshold voltage or transconductance is observable.

2.2.2. Breakdown Failures

MOS integrated circuits are subject to a variety of breakdown mechanisms. The drain of a MOS transistor forms a reverse biased junction with the channel. One limit to maximum power supply voltage is the breakdown voltage of the drain channel junction. Another type of breakdown is punch-through. Punch-through occurs when the drain depletion region extends all the way across the channel to the source depletion region. Punch-through results in a large uncontrolled current flowing between drain and source.

Another source of failures is due to parasitic bipolar transistors. A NMOS transistor has a parasitic lateral npn bipolar transistor. The collector and emitter are formed by the source and drain areas while the base is made up of the channel. A substrate current caused by impact ionization will eventually lead to a voltage drop between the substrate and source. This drop forward-biases the emitter-base junction of the parasitic bipolar transistor, which turns on the bipolar transistor causing drain breakdown at a much lower voltage. Short channel devices aggravate the situation. The shorter the channel, the more efficient the bipolar transistor will be due to the thinner base. It has been reported [24] that trapped charge in the gate oxide can make the bipolar transistor easier to turn on. Since the current flow due to bipolar action increases hot electron injection, this is a regenerative process.

Latchup is a similar, although more serious problem, that can occur in bulk CMOS circuits. An n-tub CMOS process results in a lateral npn parasitic bipolar transistor (as in the NMOS case) and a vertical pnp parasitic bipolar transistor. Together, these two transistors form a npnp semiconductor controlled rectifier. If the product of the two parasitic bipolar transistor's current gains exceed 1, then a transient pulse or exposure to radiation may result in the semiconductor controlled rectifier turning on. This results in a large current flowing from power to ground. If this current is large enough, the circuit may be damaged. A thorough discussion of the transient conditions necessary for latchup is given in [25].

2.3. Radiation-Induced Soft Failures

Soft failures are random non-recurring errors. These errors are caused by radiation striking integrated circuits and generating electron-hole pairs. The failure rate will depend on the amount of radiation striking the integrated circuit at any given time. By contrast, the radiation failure modes discussed previously depend on the total dose the integrated circuit has received. The higher the dose, the more the circuit is damaged. Soft errors, however, are caused by excess carriers, not damage to the device. Since dynamic devices are non-restoring, they are most susceptible to soft errors but static circuits may also be affected by high radiation environments.

The two major causes of soft errors are alpha particles and cosmic rays. The alpha particles are due to small amounts of radioactive material (usually uranium or thorium) in the packaging. The radioactive

material emits high energy alpha particles. If these particles are generated close enough to the surface of the integrated circuit, they will enter the substrate and generate electron-hole pairs which can then be collected by a reverse-biased junction. A thorough discussion of electron-hole pair generation and subsequent collection is given in [26]. Information in dynamic circuits is represented by charge stored on a node. Therefore, excess carriers generated by ionizing radiation can erase information stored in the circuit. In the scaling process, the amount of charge used to store information is reduced. An error only occurs if the amount of excess charge generated is at least of the same order as the amount of charge used to store information. Therefore, the scaling process will make circuits more susceptible to soft errors. Steps can be taken to protect a circuit from alpha particles [27]. Unfortunately, it is very difficult to shield an integrated circuit from cosmic rays.

CHAPTER 3

Behavior of Failed Circuits

In order to develop an accurate fault model, it is necessary to have a good understanding of the behavior of circuits that have failed. In addition, if it is possible for a failed circuit to produce an output which is not a valid logic value, then we must also have an understanding of the behavior of a good circuit given such invalid logic values as inputs. In this chapter, we develop an understanding of both circumstances.

3.1. Summary of Failure Mechanisms

In Chapter 2, we arrived at the following list of possible failure mechanisms:

(1) Interconnect failures: Opens in metal and polysilicon lines due to electromigration. Shorts between metal lines due to electromigration. Shorts between diffusion lines due to junction failure and parasitic field transistors. Shorts between diffusion lines due to junction failure and parasitic field transistors. Shorts between diffusion contacts and substrate due to spike formation. Open polysilicon contacts due to electromigration. Shorts between diffusion and substrate due to junction failure. Shorts between metal or polysilicon and other interconnect layers (including transistor channels) due to dielectric failure.

(2) Transistor failures: Parameter shifts due to hot electron injection, radiation exposure, and exposure to contaminants. Increased noise due to radiation exposure. Drain breakdown due to junction failure and parasitic bipolar transistors. Latchup due to parasitic bipolar transistors.[3]

(3) Soft failures: Soft failures due to ionizing radiation and other environmental sources of interference.

Two prior research studies have evaluated the likelihood of particular failure mechanisms for integrated circuits. Galiay et al. studied the failures of a 4-bit microprocessor [28]. The microprocessor was fabricated using a metal gate PMOS process. Failed microprocessors were examined under an optical and scanning electron microscope. The microprocessors were also probed directly. The study found the following distribution of failures:

Short between metallization	39%
Open metallization	14%
Short between diffusions	14%
Open diffusion	6%
Short between metallization and substrate	2%
Inobservable [sic]	10%
Insignificant	15%

The failures labeled inobservable [sic] were those failures which resulted in incorrect behavior but for which no physical failure could be found. Insignificant failures were those failures which resulted from "large imperfections" such as a scratch across the entire integrated circuit. Galiay et al. felt that such failures were insignificant since they should be easily detected by almost any test sequence.

Another study by Banerjee [4] was based on Texas Instruments' experience with MOS circuit failures. Failures are listed as either device failures or interconnect failures. The following failures were listed, divided into groups based on their likelihood of occurrence:

Most likely:
- Device failures:
  - Gate to drain short
  - Gate to source short
- Interconnect failures:
  - Short between diffusion lines

Moderately likely:
    Device failures:
        Drain contact open
        Source contact open
    Interconnect failures:
        Aluminum-polysilicon crossover broken

Least likely:
    Device failures:
        Gate to substrate short
        Floating gate
    Interconnect failures:
        Short between aluminum lines

From these two studies and the results of Chapter 2, it appears that interconnect failures will be a major failure mechanism. The Galiay et al. study attributed all significant observable faults to interconnect failures. If we enlarge the concept of interconnect failures to include all failures that result in an open or short, then all the failures mentioned in the Banerjee study are also interconnect failures. The idea of classifying transistor failures that result in opens or shorts as interconnect failures is quite reasonable for MOS circuits. MOS transistors are formed by one level of interconnect (polysilicon) crossing over another layer of interconnect (diffusion) [29]. For this reason, the transistor itself may simply be considered another type of interconnect.

Reviewing our summary of physical failure mechanisms listed at the beginning of the chapter reveals that all interconnect and transistor failures with the exception of parameters shifts and noise result in either opens or shorts. It must be kept in mind, however, that many of the failures (especially transistor failures) result in resistive shorts whose impedance depends on the voltage of various nodes in the vicinity

of the failure. Radiation-induced soft errors have no correspondence to shorts or opens. Nevertheless, it is possible to model the effect of such an error as a transient short. The short creates a "wire" which carries the current that flows due to excess carriers generated by the radiation.

From the above discussion, it is possible to account for nearly all of the listed physical failures by considering only opens or shorts. A short results when a failure causes an anomalous impedance to occur between two nodes. This impedance may depend on the voltages of neighboring nodes (as is the case for many transistor failures) and may also be time dependent (as is the case for radiation-induced soft errors). An open results when a failure causes an anomalous impedance to occur in series with an existing element between two nodes. This impedance may not be infinite since many failures (such as electromigration) tend to occur gradually. As was the case for shorts, the open impedance may be voltage and time dependent. The only failures which we haven't accounted for by our enlarged class of interconnect failures are parameter shifts and noise. Parameter shifts of transistors will affect both the steady state and transient performance of a circuit. These effects are due to changes in the conductance of a transistor with a given bias. If the conductance of the channel increases, we may model this failure as an impedance placed in parallel with the channel. If the conductance of the channel decreases, we may model this as an impedance in series with the channel. These two situations correspond to our definition of a short and open, respectively.

Now that we have classified failures as being either interconnect failures or noise, we are ready to study the effects that physical failures have on the behavior of various circuits. We begin by modeling transistors and the basic circuits used to process digital signals. We then use these models to study the behavior of such circuits under physical failure.

3.2. Circuit Models

The basic building block for MOS circuits is the MOS transistor. Figure 3.1 shows the symbols we use for enhancement and depletion transistors. The MOS transistor is a four terminal device. The four terminals are drain, gate, source, and body. For proper operation, the body terminal of all n channel transistors must be connected to the most negative voltage in the integrated circuit. A p channel transistor must have its body connected to the most positive voltage. Unless the body terminal is pertinent to the discussion, it will be ignored.

The exact relationship between drain current i_{ds}, and the voltages of the four terminals is quite complex. The MOS transistor has three regions of operation. In the off region, the drain current is approximately zero. In the nonsaturated region, the drain current increases as the drain to source voltage increases. Operation in the nonsaturated region is often approximated by replacing the channel of the transistor with a resistor. In the saturated region drain current is roughly independent of the drain to source voltage. Saturation is sometimes approximated as a current source between the drain and source terminals. A simplified model which is accurate enough for a variety of purposes

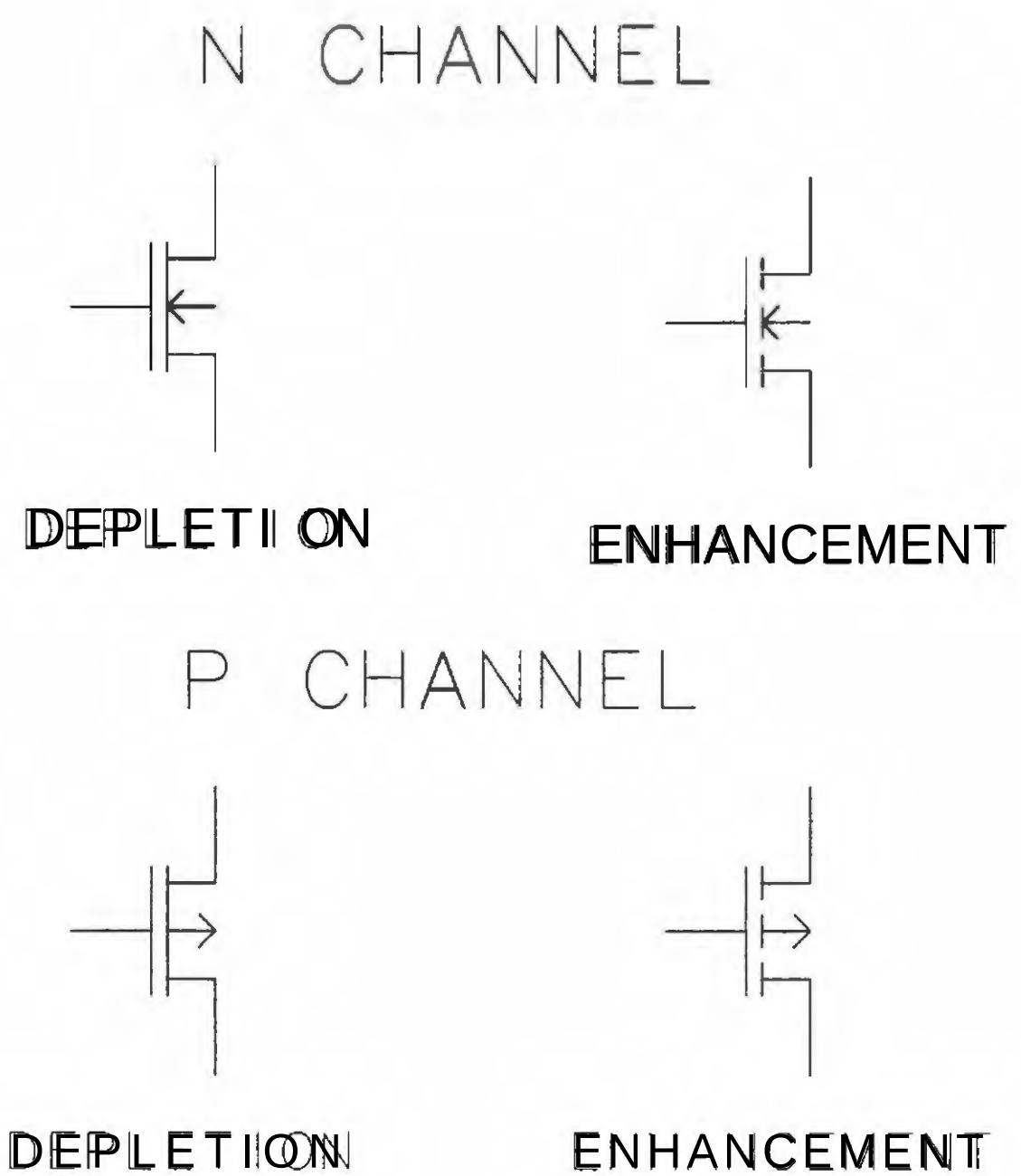


Figure 3.1. MOS Transistor Symbols.

gives the following equations:

$$i_{ds} = \begin{cases} 0 & \pm V_{gs} < \pm V_{th} \quad \text{(off)} \\ \pm\beta\left[(V_{gs} - V_{th})V_{ds} - \frac{V_{ds}^2}{2}\right] & \pm V_{gs} \geq \pm V_{th},\ \pm V_{gd} \geq \pm V_{th} \quad \text{(nonsat.)} \\ \pm\frac{\beta}{2}(V_{gs} - V_{th})^2 & \pm V_{gd} \leq \pm V_{th} \leq \pm V_{gs} \quad \text{(sat.)} \end{cases}$$

The voltages are defined in Figure 3.2. β is a constant which depends on processing parameters and the geometry of the device. β is equal to $\mu C_{ox} W/L$ where μ is the mobility of the charge carriers, C_{ox} is the gate oxide capacitance per unit area, and W and L represent the width and length of the channel, respectively. In the above equations, where the sign is $\pm$, the plus signs are for n channel devices while the minus signs are for p channel devices. If the threshold voltage V_{th} is greater than zero, then an n channel transistor is operating in the enhancement mode while a p channel transistor is operating in the depletion mode. For a negative threshold, an n channel transistor is in the depletion mode while a p channel transistor is in the enhancement mode. The MOS transistor is symmetric with respect to its drain and source terminals. It is customary to assign the drain and source terminals by their voltages. For an n channel device, the drain voltage is greater than the source voltage. For a p channel device, the source voltage is greater than the drain voltage.

This model fails to take into account several factors. In particular, if the transistor is saturated, then the model predicts that i_{ds} will be constant with respect to V_{ds}. This is approximately true for long channel devices. For shorter channel devices, an effect known as

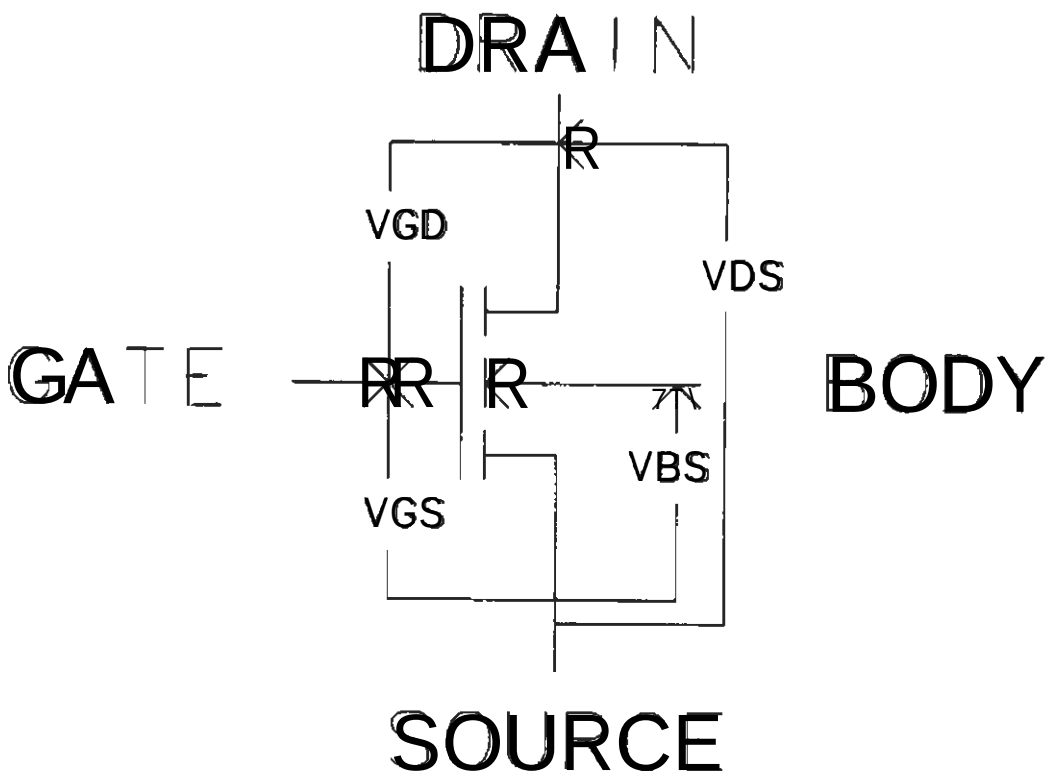


Figure 3.2. Definitions of Voltages and Polarities.

channel length modulation occurs [30]. Channel length modulation causes i_{ds} to increase slightly as V_{ds} increases. The shorter the channel, the more pronounced the effect. The simplified model also fails to account for the influence of V_{bs} on drain current [30]. This is the so-called body effect. If the body to source voltage is relatively large, then the change in threshold voltage is approximately proportional to the square root of the body to source voltage. A change in threshold voltage causes drain current to vary.

We use a small signal model of the saturated transistor in those situations where these effects are important. The model we use is basically the same as the model developed in [31]. Drain current is a function of V_{gs}, V_{ds}, and V_{bs}. We assume that the transistor is at some operating point represented by $\bar{V}_{gs}$, $\bar{V}_{ds}$, and $\bar{V}_{bs}$. The drain current of the transistor at this operating point is defined to be I. We may now use the Taylor's series to represent the drain current:

$$i_{ds} = I + \left.\frac{\partial i_{ds}}{\partial V_{gs}}\right|_{\bar{V}_{gs},\,\bar{V}_{ds},\,\bar{V}_{bs}} (V_{gs} - \bar{V}_{gs}) + \left.\frac{\partial i_{ds}}{\partial V_{ds}}\right|_{\bar{V}_{gs},\,\bar{V}_{ds},\,\bar{V}_{bs}} (V_{ds} - \bar{V}_{ds})$$

$$+ \left.\frac{\partial i_{ds}}{\partial V_{bs}}\right|_{\bar{V}_{gs},\,\bar{V}_{ds},\,\bar{V}_{bs}} (V_{bs} - \bar{V}_{bs}) + \ldots$$

Following standard convention, we define g_m (transconductance), g_d, and g_{mb} as follows:

$$g_m \equiv \left.\frac{\partial i_{ds}}{\partial V_{gs}}\right|_{\bar{V}_{gs},\,\bar{V}_{ds},\,\bar{V}_{bs}} = \frac{2I}{\bar{V}_{gs} - V_{th}}$$

$$g_d \equiv \left.\frac{\partial i_{ds}}{\partial V_{ds}}\right|_{\bar{V}_{gs},\,\bar{V}_{ds},\,\bar{V}_{bs}} = \frac{\chi I}{1 + \chi \bar{V}_{ds}}$$

$$g_{mb} \equiv \left.\frac{\partial i_{ds}}{\partial V_{bs}}\right|_{\bar{V}_{gs},\ \bar{V}_{ds}} \equiv \frac{\gamma I}{(\bar{V}_{gs} - V_{th})(2\phi_f - \bar{V}_{bs})^{1/2}}$$

χ is the channel length modulation parameter. Its value is given by the formula::

$$\chi = \frac{1}{LV_{ds}}\sqrt{\frac{2\varepsilon_0\varepsilon_{si}}{qN_{sub}}[V_{ds} - (\bar{V}_{gs} - V_{th})]}$$

where N_{sub} represents the substrate doping concentration. In the expression for g_{mb}, ϕ_f is the Fermi level of the substrate and γ is the bulk threshold parameter and is given by the formula,,

$$\gamma = \frac{\sqrt{2\varepsilon_{si}qN_{sub}}}{C_{ox}}$$

For more information on the derivation of g_m, g_d, and g_{mb}, see [31]. The significance of the various device parameters is discussed in [30]. The Taylor series expansion of i_{ds} can now be rewritten as*

$$i_{ds} = I + g_m(V_{gs} - \bar{V}_{gs}) + g_d(V_{ds} - \bar{V}_{ds}) + g_{mb}(V_{bs} - \bar{V}_{bs}) + \ldots$$

For a very small change from the operating point,, we can ignore the higher order terms in the expansion giving us

$$i_{ds} = I + g_m(V_{gs} - \bar{V}_{gs}) + g_d(V_{ds} - \bar{V}_{ds}) + g_{mb}(V_{bs} - \bar{V}_{bs})$$

3.2.1. Static NMOS Inverter Model

Figure 3.3 shows the circuit diagram for a standard NMOS inverter using a depletion load transistor. One of the attributes of this circuit we are interested in is the input-output transfer characteristics. In particular,, we are interested in the gain of the inverter at its transition point. The transition point occurs when the voltage at the

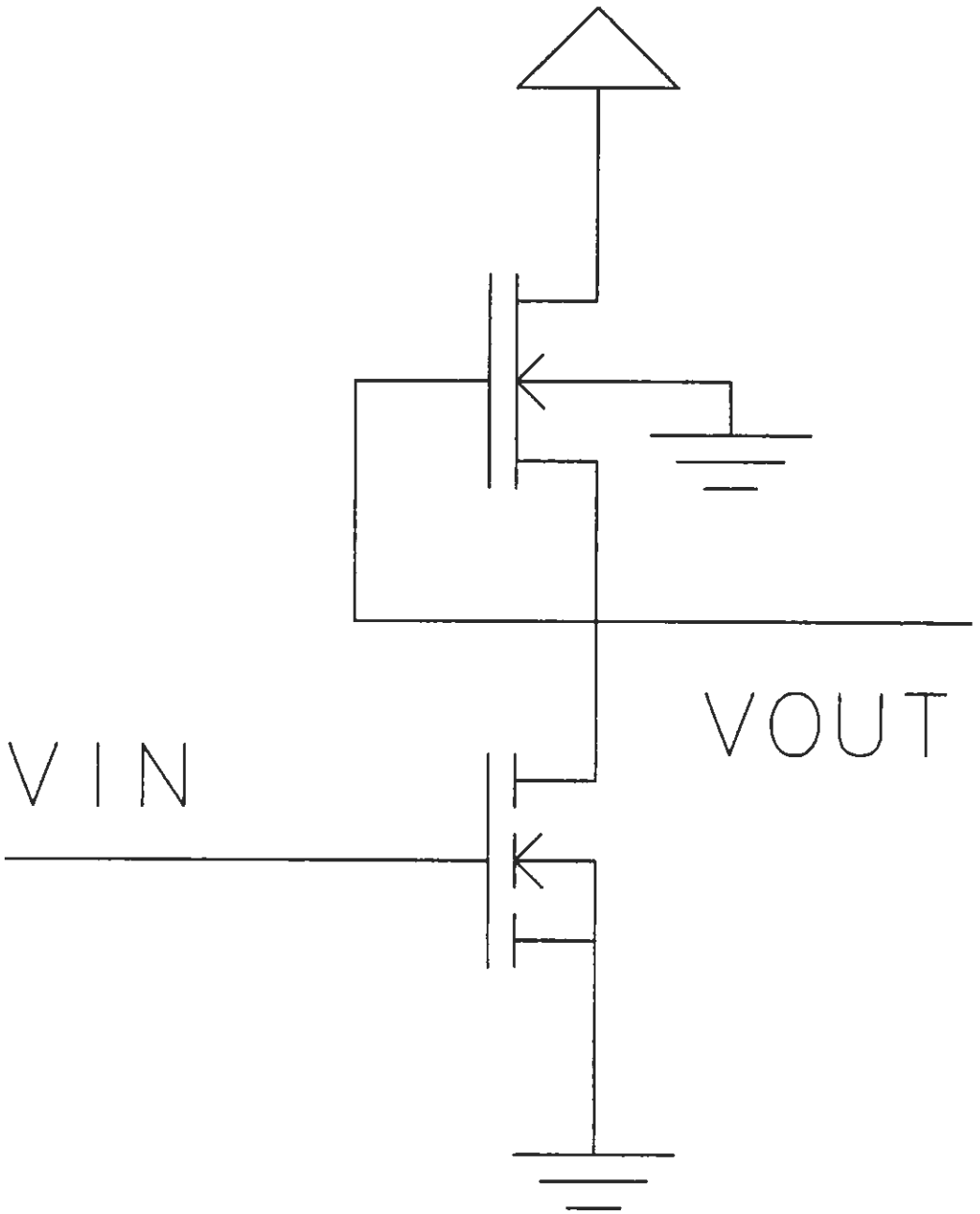


Figure 3.3. NMOS Inverter Circuit.

output of the inverter, V_{out}, is equal to V_{in}, the voltage at its inputs. The gain is the derivative of V_{out} with respect to V_{in}.

It is quite easy to find the steady state transfer characteristics of an inverter by equating the drain to source current of the load transistor with the drain to source current of the driver transistor. If the inverter is at its transition point, then the voltage at the gate of the driver transistor must be equal to the voltage at the drain of the driver transistor. This equality implies that V_{gd} of the driver transistor is zero. Since the driver transistor is an enhancement transistor, the driver transistor is saturated. Depending on the parameter and geometry of the transistors as well as the supply voltage, V_{dd}, the load transistor may be either saturated or nonsaturated. It can be shown that the load transistor is saturated at the transition point when

$$V_{thL} \leq \frac{V_{dd}}{1 + \beta_r^{1/2}}$$

β_r is the ratio of β_D to β_L. If the threshold voltage of the enhancement transistor is low enough to allow polysilicon to cross diffusion, then the load transistor must be nonsaturated at the transition point.

If the load is nonsaturated, then the equation for V_{out} is

$$V_{out} = (V_{thL} + V_{dd}) + [V_{thL}^2 - \beta_r(V_{in} - V_{thD})^2]^{1/2}$$

By taking the derivative of V_{out} with respect to V_{in}, we find that the gain A is:

$$\frac{d(V_{out})}{d(V_{in})} = A = \frac{-\beta_r (V_{in} - V_{th_D})}{[V_{th_L}^2 + \beta_r (V_{in} - V_{th_D})^2]^{1/2}}$$

or

$$A = \frac{-\beta_r (V_{in} - V_{th_D})}{V_{out} - V_{th_L} - V_{dd}}$$

Recognizing that at the transition point $V_{in} \equiv V_{out}$, we find that the gain at the transition point A^* is:

$$A^* = -\beta_r \frac{V_{in}^* - V_{th_D}}{V_{in}^* - (V_{th_L} + V_{dd})}$$

where V_{in}^* is the input voltage at the transition point. Its value may be found from the following formula:

$$V_{in}^* \equiv \frac{1}{\beta_r + 1}\{\beta_r V_{th_D} + V_{dd} + V_{th_L} + [V_{th_L}^2 + 2\beta_r (V_{dd}(V_{th_D} - V_{th_L})$$

$$+ V_{th_D} V_{th_L}) - \beta_r (V_{dd}^2 + V_{th_D}^2)]^{1/2}\}$$

If we attempt to substitute V_{in}^* into the equation for A, we find that the relationship between A and β_r is quite complex. An approximation for V_{in}^* given in [29] is:

$$V_{in}^* \approx V_{th_D} - \frac{V_{th_L}}{\beta_r^{1/2}}$$

Substituting this value into the expression for A^*, we find

$$A^* \approx \frac{V_{th_L} \beta_r^{1/2}}{V_{th_D} - V_{th_L}(1 + \beta_r^{-1/2}) - V_{dd}}$$

From this expression, we can see that for large β_r, A^* is approximately

proportional to $\beta_r^{1/2}$, while for small β_r, A^* is approximately proportional to β_r. Therefore, to achieve a large value for A^*, β_r must be as large as possible. Scaling will have little effect on A^*.

If we assume the load transistor is saturated at the transition point and equate the currents through the load and driver transistors, we find

$$V_{in} = V_{th_D} - \frac{V_{th_L}}{\beta_r^{1/2}}$$

This equation implies that at the transition point, V_{out} is not dependent on V_{in}. In other words, A^* is infinite. This anomaly is due to the fact that in our simplified transistor model, when a transistor is saturated, its current is independent of V_{ds}. The model also entirely ignores the body effect. By using the simplified transistor model when both transistors are saturated, we have implied that the currents through the transistors are totally independent of V_{out}. The dependence of V_{out} on a saturated transistor's drain current is fairly small. When one of the two transistors is nonsaturated, ignoring the effect of V_{out} on the saturated transistor's drain current only results in a small error. When both transistors are saturated, however, the error becomes quite large, and we are forced to use the small signal model of the transistor.

Using the small signal model, if we equate the currents, we find

$$I_L + g_{dsL}(V_{dd} - V_{out} - \bar{V}_{dsL}) + g_{mbL}(-V_{out} - \bar{V}_{bsL})$$

$$= I_D + g_{mD}(V_{in} - \bar{V}_{gsD}) + g_{dsD}(V_{out} - \bar{V}_{dsD})$$

Note that the load current lacks a g_m term since $V_{gsL} = 0$ and the driver current lacks a g_{mb} term since $V_{bsD} = 0$. If we solve for V_{out}, we get

$$V_{out} = -\frac{1}{g_{dL} + g_{mbL} + g_{dD}}[g_{mbL}\bar{V}_{bsL} - g_{dL}(V_{dd} - \bar{V}_{dsL})$$

$$- g_{dD}\bar{V}_{dsD} + g_{mD}(V_{in} - \bar{V}_{gsD})]$$

To find A^*, we can take the derivative of V_{out} with respect to V_{in} giving us

$$A^* = \frac{-g_{mD}}{g_{dL} + g_{mbL} + g_{dD}}$$

For devices with moderately long channel lengths ($L > 10^{-5}$m), one typically finds that

$$g_{mD} >> g_{mbL} \approx g_{dD} > g_{dL}$$

This relation allows one to build inverters of reasonably high gain (gains between 5 and 20 are typical). Due to the complexity of the expressions for g_m, g_d, and g_{mb}, it is difficult to predict the precise behavior of A^* during the scaling process. A careful analysis shows that depending on the scaling rules used, some of the terms in A^* increase and others decrease -- all at varying rates. In general, it appears that A^* is pretty much invariant to scaling although it may decrease slightly if constant voltage scaling is used. If it is necessary to have an inverter with a very high value of A^*, the best one can do is to use very long channel devices. This strategy minimizes the

values of g_{ds_L} and g_{ds_D}. The gain will still be limited by g_{mb_L} which is not a function of channel length. Although making the channels longer increases the gain, it also decreases the circuit's density (each transistor requires more area) and in general decreases the circuit's speed of operation. As we will later see, speed of operation is severely limited if the value of β_r is large. For this reason, inverters with saturated loads are preferred over inverters with nonsaturated loads when large values of A^* are required.

Another parameter of interest is the propagation delay of an inverter. It is shown in [6] that the propagation delay of an inverter, τ_d is approximately

$$\tau_d \simeq \frac{4C_L(V_m - V_{th})}{3g_m V_m}$$

where V_m is the voltage swing and C_L is the capacitance of the load on the output of the inverter. If we make the simplifying assumption that $V_m \gg V_{th}$, then we can write

$$\tau_d \simeq \frac{4C_L}{3g_m}$$

Actually, this equation is only valid for the output switching from a logic 1 to a logic 0. It also ignores the fact that the driver transistor must not only sink the current flowing from the discharging load capacitance but also the current sourced by the load.

An alternate approach may be taken where an on transistor is modeled as a resistor. Glasser [32] develops a Thevenin equivalent circuit for an inverter. The Thevenin equivalent circuit is formed by two

resistors, two voltage sources, and two switches. The switches open and close represent the transistors turning on and off. Each resistor represents the resistance of one of the two transistors which make up an inverter. Hoyte [33] implemented a simulator based on a resistive model of the transistor. Hoyte claimed the simulator had an accuracy in the range of 10 to 15 percent compared to an accuracy range of 5 to 10 percent for the SPICE circuit simulator. Mead and Conway [29] also used a resistive model for delay calculations of MOSFET circuits.

In the resistive model of a MOSFET, the channel resistance is assumed to be proportional to the length to width ratio of the transistor. This model in turn implies that the resistance of a transistor is also inversely proportional to β. We are now able to estimate the time for both rising and falling transitions. Let us define V_{hi} to be the highest voltage the circuit output is able to obtain, and V_{lo} to be the lowest voltage the circuit output is able to obtain. V^0 is the highest voltage that other gates will reliably interpret as a logic 0, while V^1 is the lowest voltage that other gates will reliably interpret as a logic 1. V_m for a circuit is the difference between V_{hi} and V_{lo}. Finally, let us define the following

$$Z = \frac{R_L}{R_D}\Big|_{V_{in} = V^1}$$

$$Z' = \frac{R_D}{R_L}\Big|_{V_{in} = V^0}$$

$$\alpha = \frac{Z + 1}{Z}$$

$$a' = \frac{Z'+1}{Z'1}$$

Figure 3.4 shows the resistive model for a NMOS inverter. Writing the node equation for $V_{out}(t)$, we get

$$\frac{V_{dd} - V_{out}(t)}{R_L} = C_L \frac{dV_{out}(t)}{dt} + \frac{V_{out}(t)}{R_D}$$

We are interested in solving this differential equation for two sets of initial conditions. One set is for a falling transition while the other is for a rising transition. Solving for the falling transition, we get

$$V_{out}(t) = V_{lo} + V_m e^{-\frac{a't}{R_L C_L}}$$

Solving for the rising transition, we get

$$V_{out}(t) = V_{hi} + V_m e^{-\frac{a't}{R_L C_L}}$$

Inspection of Figure 3.4 shows that the load and driver transistors form a voltage divider. Therefore, the voltage limits are

$$V_{lo} = \frac{V_{dd}}{Z + 1}$$

and

$$V_{hi} = \frac{V_{dd}Z'}{Z' + 1}$$

This information may be used to solve for V_m.

$$V_m = V_{dd}\left[\frac{Z'}{Z' + 1} - \frac{1}{Z + 1}\right]$$

Figure 3.5 is a graph of V_{lo} and V_{hi} versus Z and Z', respectively. As

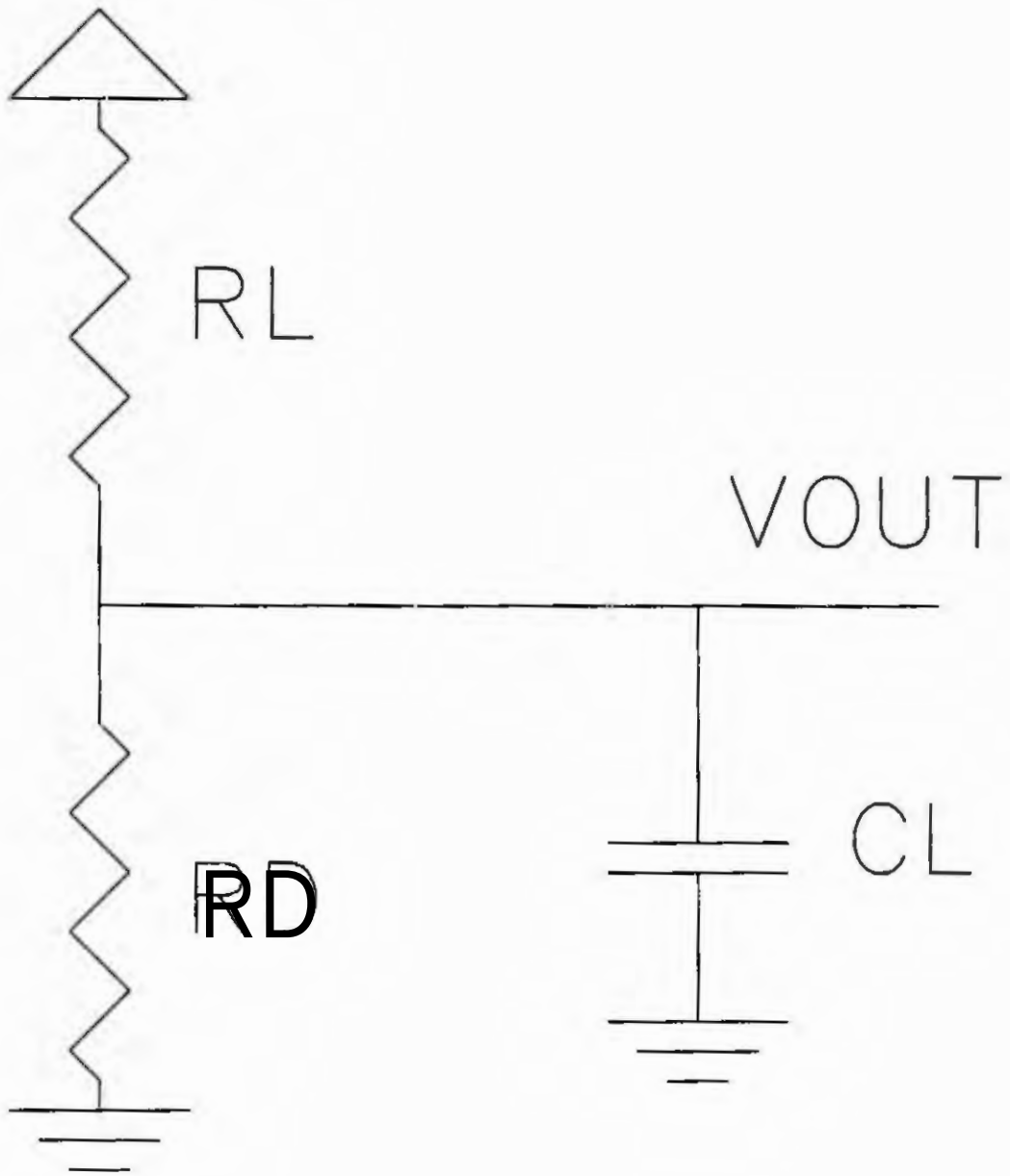


Figure 34. Resistive Model of an Inverter.

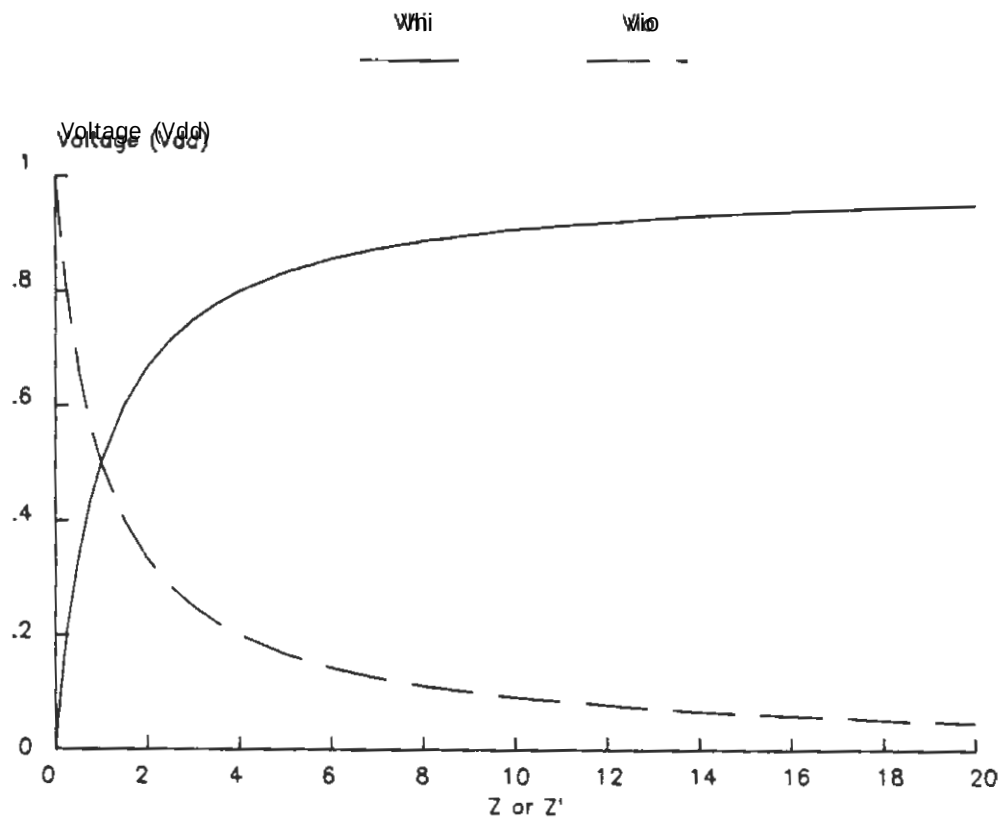


Figure 35. Voltage Limits vs. Z.

Z and Z' are increased, V_{lo} decreases and V_{hi} increases until V_{lo} and V_{hi} eventually approach the ground and power supply voltages. Since V_m is defined as the difference between V_{hi} and V_{lo}, large values of Z and Z' will maximize V_m. For the NMOS inverter, Z is proportional to β_r. In order to make Z large enough to provide proper separation between logic levels, it is necessary to make β_D greater than β_L. This is usually done by having the W/L ratio of the driver transistor much larger than the W/L ratio of the load transistor. Unfortunately, this restriction requires extra area. Z' is infinite for an NMOS inverter since the driver transistor is off during the rising transition.

We are now in a position to calculate the switching time, τ_{SW}. The switching time is the time taken to switch between V^0 and V^1. To calculate the rising switching time, set the equation for $V_{out}(t)$ (rising transition) equal to V^1 and solve for t. This value of t is τ_{SW_r}. The falling switching time, τ_{SW_f}, is found by setting the equation for $V_{out}(t)$ (falling transition) equal to V^0, and once again solving for t. The following values for τ_{SW_f} and τ_{SW_r} are thus obtained

$$\tau_{SW_f} = -\frac{R_D C_L}{a} \ln\left[a \frac{V^0}{V_{dd}} - \frac{1}{Z}\right]$$

$$\tau_{SW_r} = -\frac{R_D C_L}{a} \ln\left[a\left(1 - \frac{V^1}{V_{dd}}\right)\right]$$

The average switching time, $\tau_{SW_{ave}}$, is the average of the rising and falling switching times. It is given by the following formula

$$\tau_{SW_{ave}} = -\frac{R_D C_L}{2}\left\{\frac{1}{a}\ln\left[a\frac{V^0}{V_{dd}} - \frac{1}{Z}\right] + Z\ln\left[a\left(1 - \frac{V^1}{V_{dd}}\right)\right]\right\}$$

For small values of Z, the average switching time is dominated by the

falling transition. For large values of Z, the average switching time is dominated by the rising transition. Figure 3.6 shows a graph of average switching time (in units of RC_L) as a function of Z. In the graph, it is assumed that the ratio of V^0 to V_{dd} is 0.4, while the ratio of V^1 to V_{dd} is assumed to be 0.6. Notice how the average switching time rapidly approaches ∞ as V_{lo} approaches V^0. For this example, the minimum average switching time occurs when Z is approximately 2. Although a value of Z = 2 may optimize the average switching time, such a low value is usually unacceptable due to the resulting inverter's low gain and low noise margin. Therefore, a larger Z ratio is typically used.

In this section, we have dealt with a NMOS inverter. The analysis of a PMOS inverter is identical. Equations for gain, voltage limits, output voltage, and switching time are all the same except that the sign of supply and threshold voltages must be changed to be appropriate for PMOS devices.

3.2.2. Static CMOS Inverter Model

Figure 3.7 shows the circuit model for a CMOS inverter. The load transistor is a p channel MOS transistor while the driver transistor is an n channel MOS transistor. At the transition point, both transistors have a V_{ds} of 0 volts. Therefore, since V_{th_D} is positive and V_{th_L} is negative, both transistors are saturated. If we attempt to use the simplified transistor model we would once again arrive at the result that A^* is infinite. For this reason, we immediately proceed to the small signal model. Equating currents, we find that

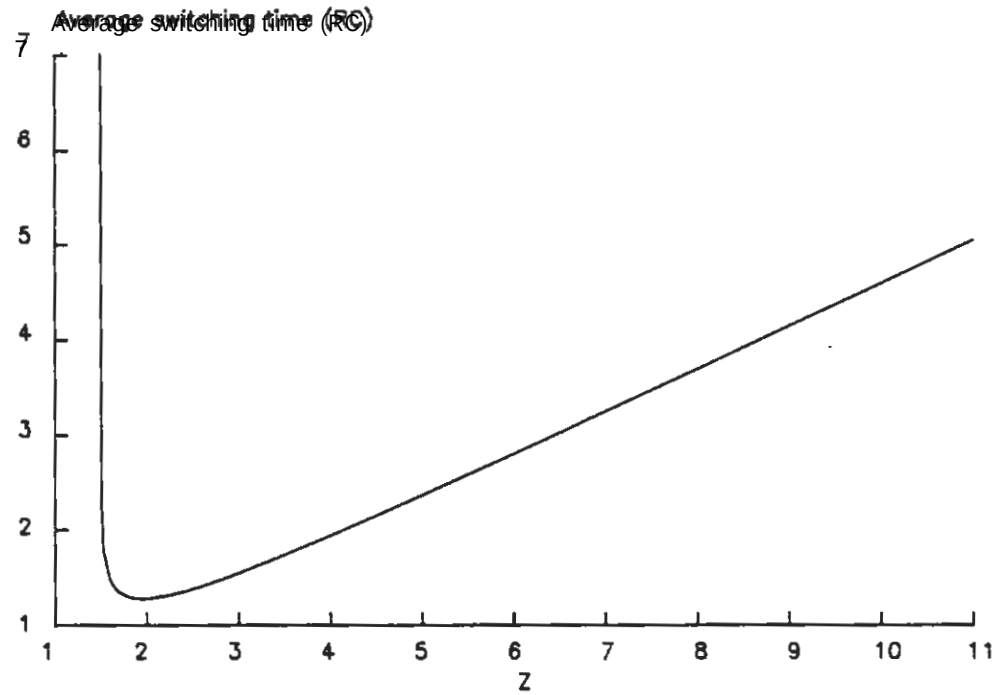


Figure 3.6. Average Switching Time vs. Z Ratio.

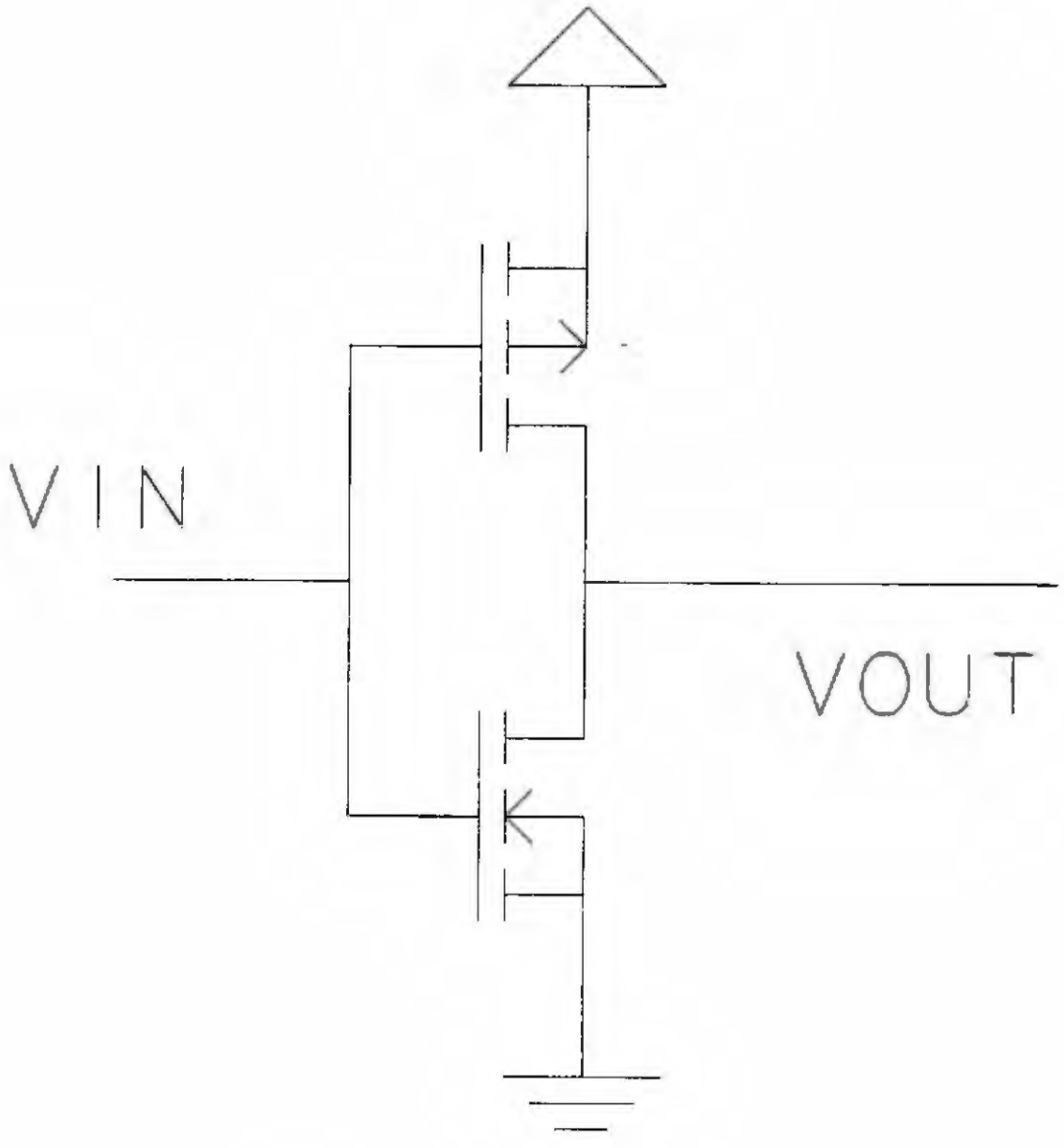


Figure 3.7. CMOS Inverter Circuit.

$$I_D = g_{m_D}(V_{in} - \bar{V}_{gs_D}) + g_{d_D}(V_{out} - \bar{V}_{ds_D})$$

$$= -I_L = g_{m_L}(V_{in} - V_{dd} - \bar{V}_{gs_L}) - g_{d_L}(V_{out} - V_{dd} - \bar{V}_{ds_L})$$

The load current terms are all negative since the drain to source current of the load transistor flows in a direction opposite to the drain to source current of the driver transistor. Also there is no body effect term for either transistor since the body to source voltage is always 0 for both transistors. This equation can be solved for V_{out} giving us

$$V_{out} = \frac{1}{g_{d_D} + g_{d_L}}\left[g_{d_D}\bar{V}_{ds_D} + g_{d_L}(V_{dd} + \bar{V}_{ds_L}) - g_{m_L}(V_{in} - V_{dd} - \bar{V}_{gs_L})\right.$$

$$\left. - g_{m_D}(V_{in} - \bar{V}_{gs_D})\right]$$

To find A^*, we take the derivative of V_{out} with respect to V_{in} giving

$$A^* = -\frac{g_{m_L} + g_{m_D}}{g_{d_D} + g_{d_L}}$$

In many situations, the load and driver transistors are designed to have identical characteristics so that the circuit response will be symmetrical and $V^*_{in} = V_{dd}/2$. In such situations, $g_{m_D} = g_{d_L}$. If this is the case, then A^* simply becomes:

$$A^* = -\frac{g_{m_D}}{g_{d_D}}$$

The value of A^* is only dependent on the values of g_m and g_d of the two transistors. For this reason, CMOS inverters can be built with higher gain than NMOS inverters. By making the channels very long, g_d can be made quite small. For driver transistors of the same size, the gain of

a CMOS inverter is typically 3 to 4 times greater. During the scaling process, A^* decreases slightly regardless of whether constant voltage or constant field scaling is used.

If the response of a CMOS inverter is to be symmetric, then $\beta_r \sim 1$. This implies that the rising and falling propagation delays are roughly equal. Also, except when the inverter is near its transition point, only one of the two transistors is on. Because of this, V_m spans the full range from 0 to V_{dd}. The expression for τ_d given in [6] applies to this case giving us

$$\tau_d \sim \frac{4C_L}{3g_m}$$

where the value of g_m corresponds to the on transistor. The equations derived for the NMOS inverter delay and output voltage also apply to the CMOS inverter. In this case, both Z and Z' are infinite.

1.2.3. Dynamic NMOS Inverter Model

In order to reduce power consumption and increase packing density, dynamic circuits are becoming quite popular. Since dynamic logic is typically ratioless, it usually requires much less area than equivalent static logic. More importantly, dynamic circuits have very low power consumption. The only power consumed is that required to charge and discharge nodes. Dynamic circuits are fundamentally different than static circuits. In a dynamic circuit, information is represented by the presence or absence of charge on a node. A dynamic circuit processes information by charging and discharging nodes, and transferring charge from one node to another. The most important difference

between dynamic and static circuits is that static circuits are restoring. If an external force disrupts the operation of a static circuit, the static circuit opposes the disruption. A dynamic circuit is not able to oppose a disruption. Dynamic circuits are very sensitive to charge leakage, changes in device parameters, and clock skew. They are also sensitive to ionizing radiation which can erase the charge stored on a node. For these reasons, dynamic circuits might be a poor choice where high reliability is a necessity. On the other hand, since the power consumption is low (and thus circuit temperature is low) and the currents tend to be pulses rather than constant (and thus electromigration is less likely), dynamic circuits may offer advantages for long term reliability.

A great variety of dynamic circuits exist [34]. Dynamic circuits range from bootstrap drivers which can drive large capacitive loads to dynamic CMOS circuits which can implement very complex logic functions. The circuit we examine is perhaps the simplest dynamic circuit, the two phase ratioless shift register. Figure 3.8 shows the circuit diagram for a 1 bit section of the shift register. The circuit uses two nonoverlapping clocks, ϕ_1 and ϕ_2. An inspection of the circuit shows that ϕ_1 and ϕ_2 serve the function of both power and ground and that there is no way for a static current to flow. The circuit samples V_{in} while ϕ_1 is high. When ϕ_2 goes low, node 1 is the complement of V_{in}'s value when ϕ_1 was high. Node 1 is sampled while ϕ_2 is high. When ϕ_2 goes low, V_{out} becomes the complement of the value of node 1 when ϕ_2 was high. Therefore, when ϕ_1 goes high, V_{out} has the same value as V_{in} on the

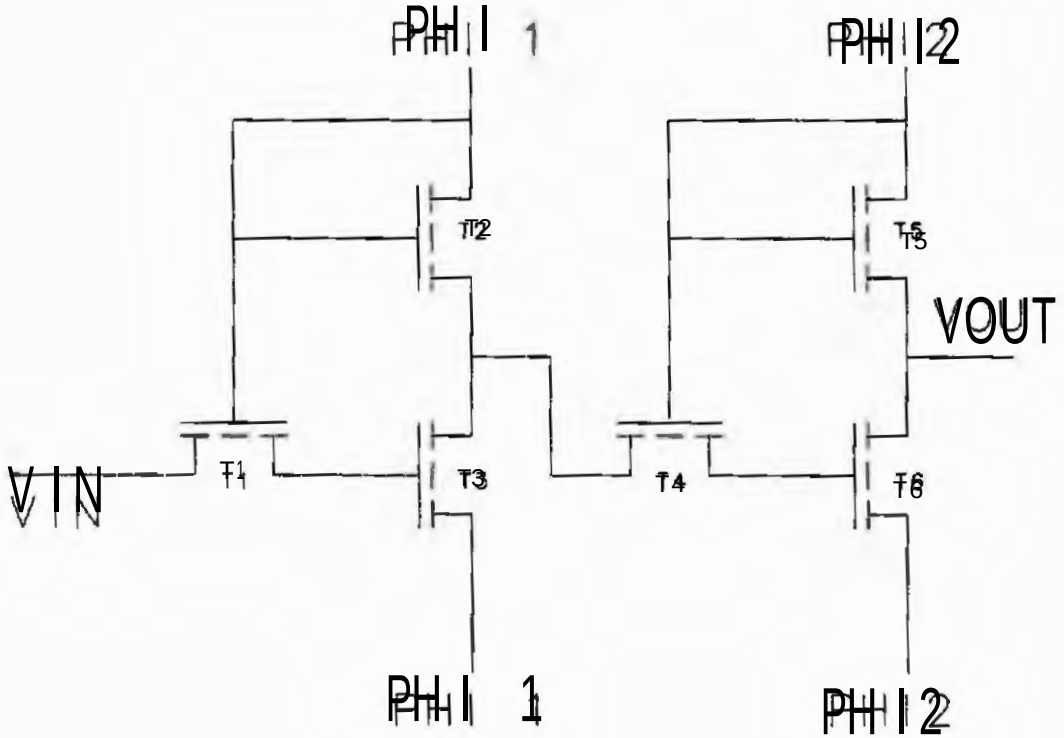


Figure 3.8. Dynamic Shift Register.

previous clock pulse. In other words, while ϕ_1 is high, V_{out} is a delayed version of V_{in}.

The transistors in the circuit can be broken into two groups, the inverter transistors which make up the inverter and the sampling transistors, T_1 and T_4, which couple together the stages. The inverter transistors (T_2, T_3, T_5, and T_6) are grouped into pairs that form inverters.

The function of the inverter transistors is to charge and discharge the inverter's output node. Charging of the output node is primarily performed by the load transistor while ϕ is high. If the gate of the driver transistor is high while ϕ is high, the driver transistor also assists in charging the node. The output node is discharged by the driver transistor while ϕ is low but only if the gate of the driver transistor is high. The load transistor is off whenever ϕ is low.

The sampling transistors serve the purpose of coupling the output node of one inverter to the input node of the next inverter. The gate of a sampling transistor is always connected to one of the two clock signals. When the gate goes high, an inverter is able to sample the output of the preceding inverter. When the output node of the preceding inverter is low, then the input node of the current inverter is discharged. The discharge path is through the sampling transistor and driver transistor of the preceding inverter. If the output node of the preceding inverter is high, then some of the charge already stored on the output node is transferred to the input node. Due to the charge being split between two nodes, the voltage after sampling at the output

node is less than it was before sampling. The input node voltage after sampling is always less than the output node voltage it sampled. The output node capacitance must be much greater than the input node capacitance, otherwise the input node may never be charged to a satisfactory level.

In order for the circuit to operate properly, the clock pulses ϕ_1 and ϕ_2 must be high long enough to charge both the input and output node and discharge the input node. The output node is charged up by a saturated transistor. Using the formula given in [29] for charging a capacitance through a saturated transistor gives

$$V_{out}(t) = V_{dd} - V_{th} - \frac{C_{out}}{\beta t}$$

From this equation, we see that V_{out} will never be charged above $V_{dd} - V_{th}$. Figure 3.9 shows the resistive model of the coupling transistor. We can use this model to calculate the time required to charge the input node through the sampling transistor. The loop equation is

$$-C_{out}\frac{dV_{out}(t)}{dt} + \frac{V_{out}(t) - V_{in}(t)}{R} + C_{in}\frac{dV_{in}(t)}{dt} = 0$$

Solving for $V_{in}(t)$, we find

$$V_{in}(t) = V_m \frac{C_{out}}{C_{in} + C_{out}}\left[1 - e^{-t/RC_{out}}\right]$$

In this circuit, $V_m = V_{dd} - V_{th}$ since the output node will never be charged past this point.

From these equations, we can calculate the time required to charge the input and output nodes. Notice that both equations depend on C_{out}.

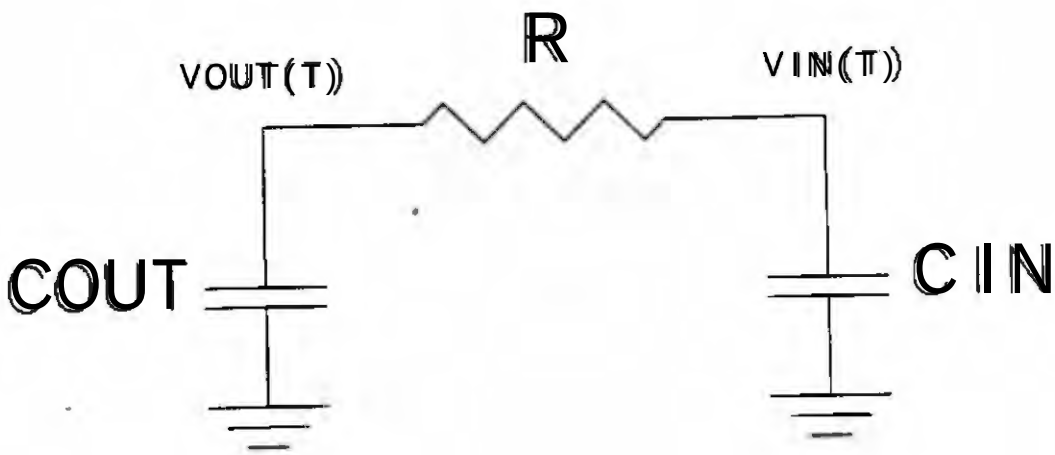


Figure 3.9. Resistive Model of Coupling Transistor.

The equation for V_{in} depends not only on C_{out}, but also the relative sizes of C_{in} and C_{out}.

3.3. Response of Failed Circuits

We have discussed the type of failures that may occur in MOS circuits in Chapter 2. We have developed models of MOS circuits in the previous section. In this section, we use these models to predict the response of failed circuits.

3.3.1. Response of Circuits with Shorts

Figure 3.10 shows an NMOS and a CMOS inverter. If we ignore the power and ground nodes, we see that each type of inverter contains two nodes, an input node and an output node. Therefore the possible shorts that are internal to an inverter are

(1) Input node shorted to power or ground

(2) Output node shorted to power or ground

(3) Input node shorted to output node

(4) Power shorted to ground

If the input node is shorted to power or ground, we may model this as the output node of the previous inverter being shorted to power or ground. We therefore only need to consider three cases.

If the output node is shorted to power or ground, then we have the impedance of the short in parallel with the impedance of the transistor. If the impedance of the short is much less than the impedance of the transistor, then the output will be stuck-at 1 or stuck-at 0, depending on whether the short is to power or ground. If the impedance of the

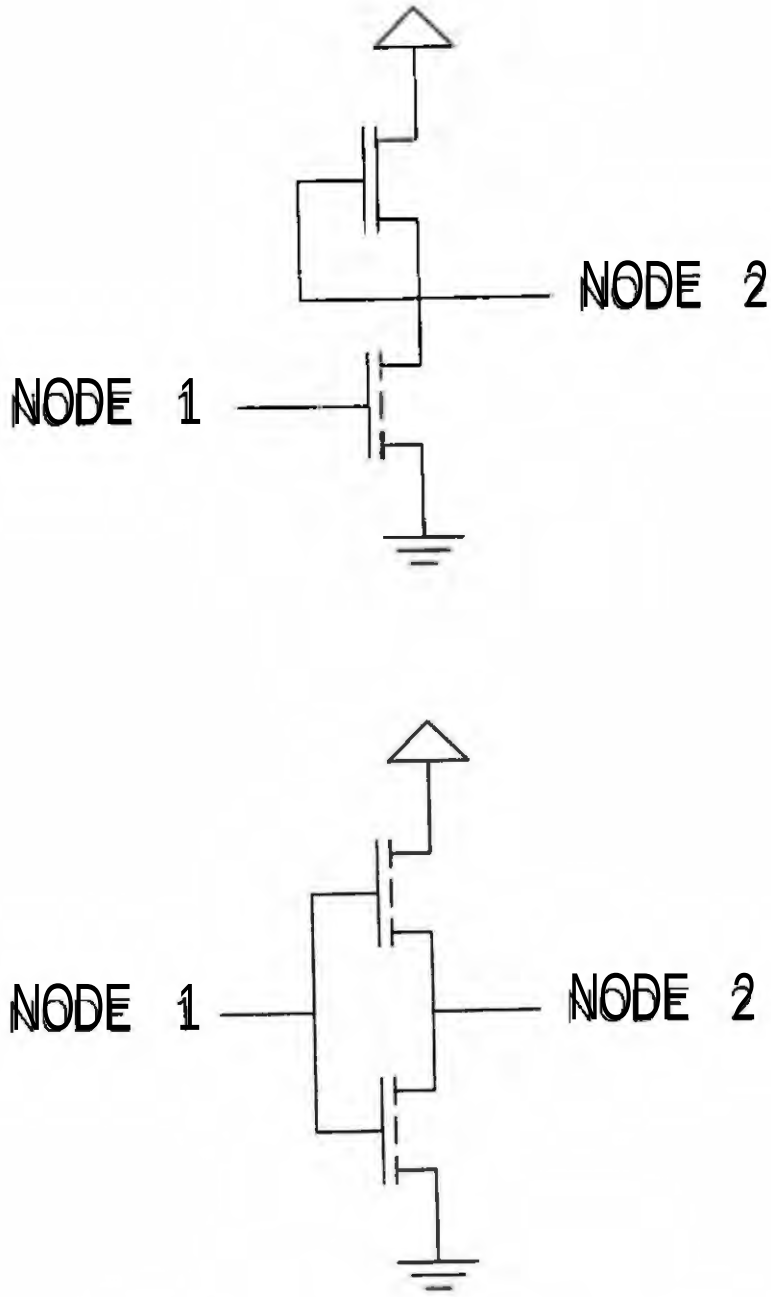


Figure 3.10. NMOS and CMOS Inverters.

short is much greater than the impedance of the transistor, then the short will have no effect on the operation of the inverter. A more interesting situation occurs if the impedances of the short and transistor are of the same order of magnitude. The impedance of the shorted transistor can be replaced with the parallel combination of the transistor impedance and the short impedance. If the short occurs in an NMOS inverter between the output node and power, then the value of Z decreases while the value of Z' increases. These new values for Z and Z' may be used with the equations already derived for inverters. The decreased value of Z causes an increase in V_{lo} and the rising transition switching time. If the short occurs between the output node and ground, the value of Z increases while the value of Z' decreases. In this case, V_{hi} decreases while the rising transition switching time increases. For a CMOS inverter, an output node to power or ground short causes Z or Z', respectively, to be reduced to a specific finite value, whereas under no failure they can be treated as effectively infinite. This decrease in Z or Z' will either increase the falling transition time and increase V_{lo} or increase the rising transition time and reduce V_{hi}. In addition, the CMOS gate now dissipates static power.

To summarize, a short from the output node to ground decreases the falling transition switching time, increases the rising transition switching time, and reduces V_{hi}. A short from the output node to power decreases the rising transition switching time, increases the falling transition switching time, and raises V_{lo}.

Figure 3.11 shows the situation that exists when the output node is shorted to the input node. By recognizing that the short and the driver transistor's gate together form a distributed RC network, we see that the circuit is of the same form as a phase-shift oscillator. The inverter forms the inverting amplifier while the short and driver transistor's gate together form the phase-shift network which serves to feed a delayed version of the inverter's output back into its input. The frequency of oscillation, ω_0 is given in [35] as:

$$\omega_0 = \frac{2}{RC}$$

where R is the resistance of the phase-shift network and C is its capacitance. The conditions necessary for oscillation are studied in [36], where it is shown that the gain of the amplifier must be less than -29 for sustained oscillation. From our discussion of A^*, it is fairly unlikely that an NMOS inverter would have the required gain for sustained oscillations. This value of gain is not unreasonable for a CMOS inverter, especially one that was deliberately designed for high gain. In order for an inverter to have high gain, it must be operating near its transition point. If the input to the inverter is driven to either a logic 0, or a logic 1, the inverter will not be able to oscillate. There are three conditions where an inverter of sufficient gain has the potential to oscillate:

(1) The circuit driving the failed inverter is not capable of driving the failed inverter's input a significant distance from its transition point. It is much harder to drive such a failed inverter than a good inverter.

(2) The failed inverter's input is coupled by a pass transistor to the previous stage. Any time the pass

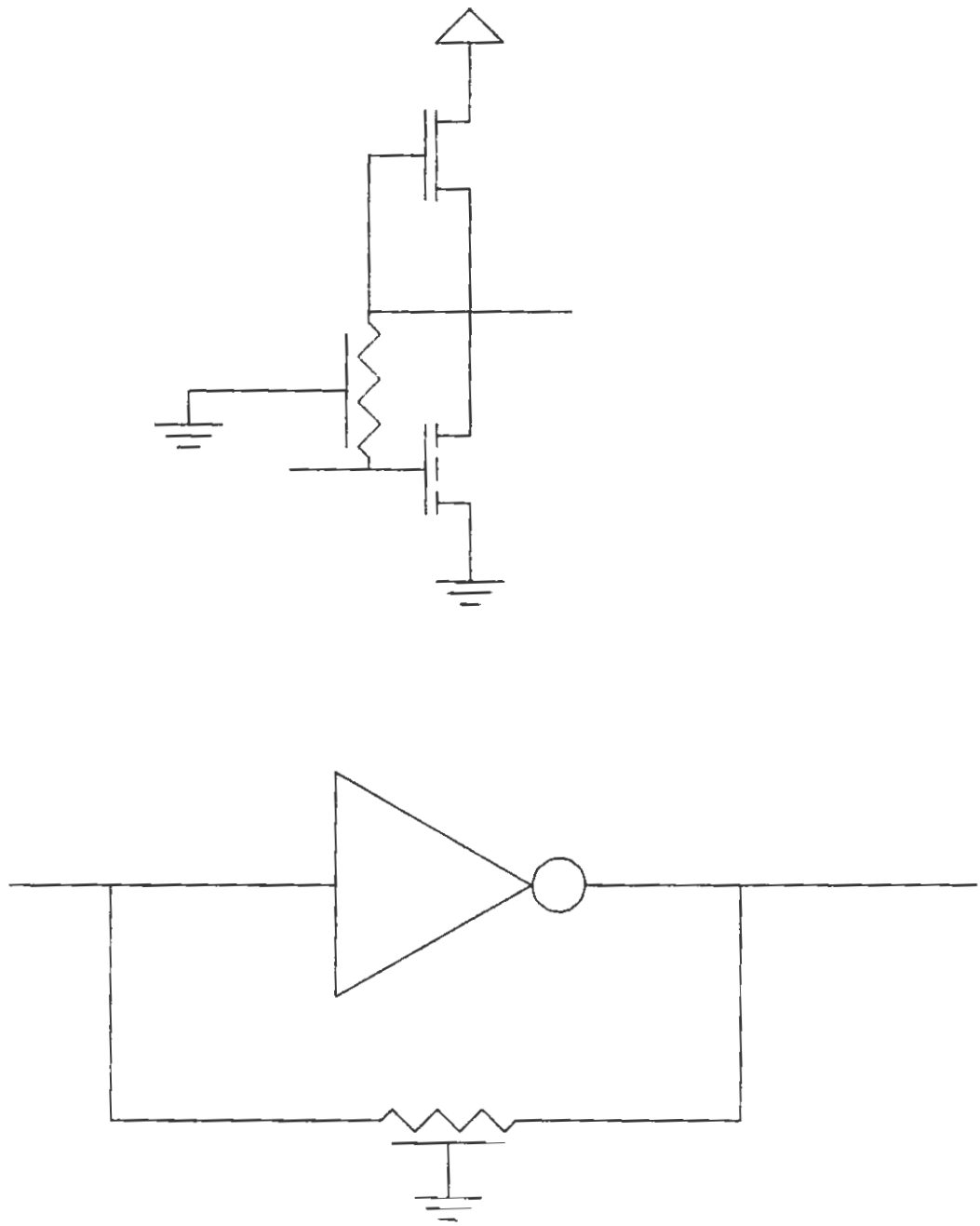


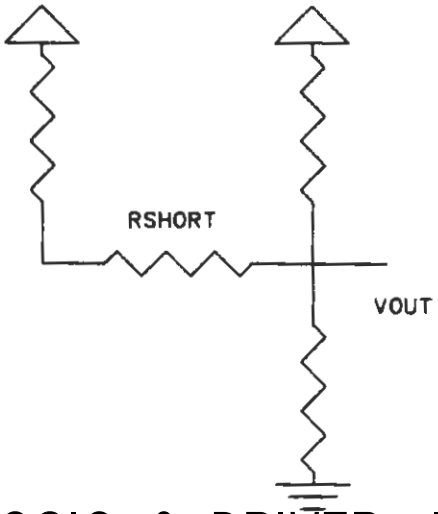
Figure 3.11. Output Node to Input Node Short.

transistor is off, the failed circuit may begin to oscillate.

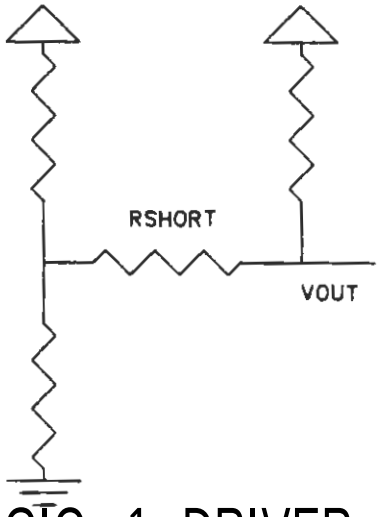
(3) The circuit driving the failed inverter switches the failed inverter's input. As the input moves through the transition region, it may oscillate until the driving circuit is capable of forcing the input a significant distance from its transition point. As mentioned in (1), this takes longer than it would for a good inverter.

If the gain of the inverter is insufficient to sustain oscillation, the result of an input to output short is to shift the inverter's logic levels and increase its switching time. The exact nature of these shifts is dependent on the impedance of the short, the value of V_{in}, the impedances of the load and driver transistors, and the impedances of the load and driver transistors driving the failed inverter. If the impedance of the short is very large (at least a factor of 10 larger than the transistor's impedance), it will have little or no effect on the circuit. As the short impedance becomes smaller, the difference between V_{in} and V_{out} becomes smaller and smaller. For an impedance of 0, V_{in} equals V_{out}. Depending on the impedances of the driving inverter, the failed inverter, and the short, V_{out} will range anywhere from ground to the supply voltage. A situation of particular interest occurs if the driving inverter and failed inverter are identical and the short resistance is small. Figure 3.12 shows the resistive models for the failed inverter including the output stage of the driving inverter. Two cases are shown, namely a logic 0 and logic 1 input to the driving inverter.

If the input to the driving inverter is a logic 0, then we effectively have the parallel combination of the load transistors of both



(A) LOGIC 0 DRIVER INPUT



(B) LOGIC 1 DRIVER INPUT

Figure 32. Resistive Model of Failed Inverter.

inverters trying to pull V_{out} high while the driver transistor of the failed inverter tries to pull V_{out} low. By setting the load currents equal to the driver currents, we find that

$$V_{out} = V_{thD} + V_{thL}\left[\frac{2\beta_r}{\beta_r}\right]^{1/2}$$

In deriving this equation, we have assumed that the load transistor is saturated. As this equation shows, the output, (which should be a logic 1), is significantly lower than V_{dd}.

If the input to the driving inverter is a logic 1, then the load transistors of both inverters try to pull V_{out} high, while the driver transistors of both inverters will be trying to pull the output low. If we assume the current through the failed inverter's driver transistor is very small, then the steady state value of V_{out} is the same as an inverter which has a value of Z which is half of the original inverter's value of Z. Therefore, the value of V_{hi} is lowered while the value of V_{lo} is raised. When the input to the driving inverter is a logic 1, it is possible for V_{out} to become greater than V_{out} when the input to the driving inverter is a logic 0. Furthermore, the speed of operation of the failed circuit is reduced considerably. This reduction is due to both the degraded values of V_{hi} and V_{lo} and the fact that the failed inverter output must drive the load capacitances of both inverters.

One interesting variation occurs if the input node is shorted to the output node and, simultaneously, the connection from the previous stage is open circuited. As long as the impedance of the open circuit to the previous stage is very large, the input and output node of the failed inverter charges to V_{in}^{*} regardless of the impedance of the short.

There are two likely ways to get a simultaneous open to the previous stage and short from input node to output node. One way is for the gate of an inverter's driver transistor to be coupled to the previous stage with a pass transistor. Whenever the pass transistor is off, the open-short condition would exist. A second way to get a simultaneous open-short would be for metal migration to cause an open. The accumulated metal could then form a short.

A short from power to ground can have catastrophic consequences. If the impedance of the short is very small, then the voltage difference between the power and ground lines would become quite small. In this case, the output of all circuits supplied by these power and ground lines would be unpredictable. In order for the power and ground line voltages to change appreciably, there would have to be a large current flowing through the short. Electromigration and/or ohmic heating of the short and power and ground lines would lead to one or more of these lines almost instantly failing (most likely the short) which would allow the power and ground lines to return to their original values. If the impedance is large enough not to reduce power supply voltage significantly, the short should have little effect; at least for the short run. The short increases the power dissipated from the integrated circuit and thus raises the temperature locally. It may also encourage electromigration to occur along power or ground lines which must now carry heavier currents than they were designed for. A power-ground short in a CMOS circuit due to latchup may be able to sustain heavy currents for a long period of time before the latched CMOS device or a power or ground line fails.

We have now studied all possible internal shorts in NMOS and CMOS inverters. An NMOS NOR gate behaves in a similar manner for internal shorts. NMOS NAND gates and CMOS gates have a structure of stacked transistors. In such a stack, the drain of one transistor is connected to the source of the next transistor. The first transistor in the stack has its source connected to the power or ground node. The drain of the top transistor in the stack is connected to the output node. A drain to source short of any of the transistors in the stack may be analyzed by the same procedures as those used for the NMOS and CMOS inverters. The most difficult situation to analyze occurs when a gate to drain short occurs. The analysis is basically the same as for the inverter except that more transistors must be considered. The results will be the same; voltage levels and speed of operation will be degraded.

Dynamic circuits are much more susceptible to shorts than static logic circuits. Dynamic logic depends on the ability to store charge on the stray capacitance of nodes. Any short, whether to another node, a clock signal, or the substrate (ground), will allow charge to leak on or off the node. If enough charge enters or leaves a node, the information stored there is destroyed. An RC time constant determines the time required to charge or discharge a shorted node, where R is the resistance of the short and C is the node capacitance. If the RC time constant is much longer than the clock pulse, the circuit should be unaffected. If RC is of the same order of magnitude as the clock pulse or smaller, the short will be able to alter the voltage of a node significantly. If the short is almost able to completely charge or discharge a node during one clock pulse, the node will appear to be stuck-at 1 or

stuck-at 0 depending on whether the short is charging or discharging the node. If the short is unable to charge or discharge the node completely during a clock pulse, but is still able to alter the node voltage significantly, then the circuit may or may not operate correctly. This situation is somewhat analogous to the shifting of V_{lo} and V_{hi} in static circuits. The most critical determinate of maximum clock speed for this circuit is the time taken to charge and discharge the input node of the inverter. A short occurring at either the input or output nodes, significantly increases the time required to perform these operations.

In addition to internal shorts, it is also possible for external shorts to occur between inverters. We again treat external shorts as if they occur between output nodes. Let us first assume that the short does not introduce feedback. That is, neither of the shorted outputs is a function of the other. If both outputs are the same value, the behavior of the two outputs is generally unaffected. If the outputs have complementary values, several possibilities may occur. If the impedances of the short and one of the inverters are much less than the impedances of the other inverter, then the inverter with the larger impedances will follow the output of the other inverter. If the impedances of both inverters are similar, then the exact behavior will depend primarily on the impedance of the short. The two load transistors, coupled by the impedance of the short will be trying to pull both output nodes high while one of the driver transistors will be trying to pull the output nodes low. See Figure 3.13. The voltage at nodes 1 and 2 will be:

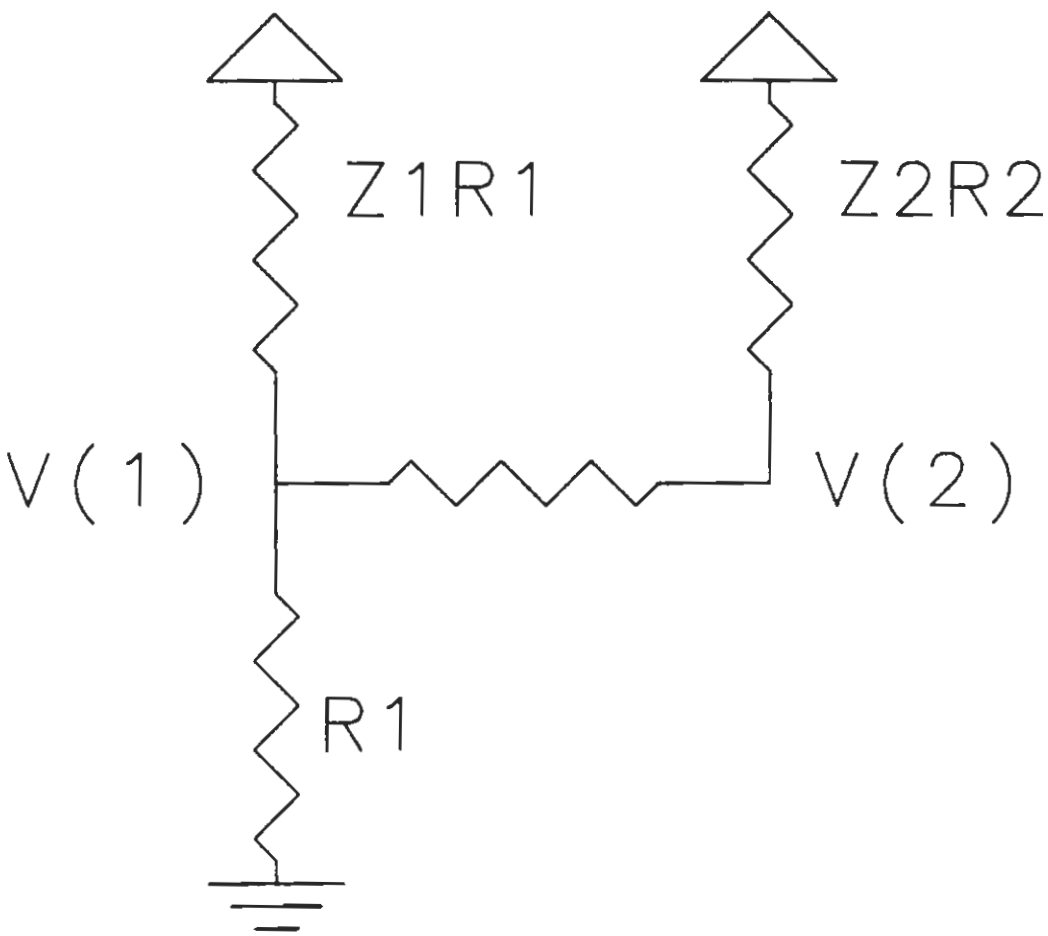


Figure 3.13. Resistive Model of Two Outputs Shorted Together.

$$V(1) \equiv V_{dd} \frac{Z_1R_1 + Z_2R_2 + R_{short}}{Z_1Z_2R_2 + Z_1R_{short} + Z_1R_1 + Z_2R_2 + R_{short}}$$

$$V(2) \equiv V_{dd} \frac{R_{short}}{Z_2R_2 + R_{short}} + V(1) \frac{Z_2R_2}{Z_2R_2 + R_{short}}$$

If the two inverters are identical, then

$$V(1) = V(2) = \frac{V_{dd}}{\frac{Z}{Z} + 1}$$

In this case, the effective value of Z has been reduced by one-half. In addition to degrading the steady state output values, a short also reduces the speed of a falling transition since both inverters' load capacitors have to be discharged by one driver transistor. As mentioned in Chapter 1, many people use a wired AND operation assumption to model shorts between outputs. An examination of the equations for V(1) and V(2) shows the wired AND operation assumption is only justified if the value of the short resistance is small and R_1 and R_2 are both much smaller than either Z_1R_1 or Z_2R_2.

If a short occurs between two output nodes where one of the outputs is a function of the other, then we have feedback. If this feedback loop includes an odd number of inversions, oscillation is possible. Let us define the looped inverter to be the inverter whose output is a function of the other inverter's output (that is the inverter inside the feedback loop). We refer to the other inverter as the unlooped inverter. In order for oscillation to occur, the looped inverter must have a driver transistor with much lower impedance than that of the unlooped inverter's load. An algorithm is given in [37] to predict

whether or not feedback bridging faults will lead to oscillation.* The algorithm is useful for determining whether or not a complicated circuit will oscillate with a given input vector. Unfortunately this work is based on the wired AND or wired OR assumptions which may not be applicable. If the inverter which is out of the feedback loop has much lower impedances than the inverter in the feedback loop, then the inverter in the feedback loop's output will follow the other inverter's output.

If the feedback loop encloses an even number of inversions, the circuit will generally not oscillate. The inverters inside the loop will form a latch circuit. If the unlooped inverter has lower impedances than the looped inverter to which it is shorted, then the latch will change state each time the input to the unlooped inverter changes. If the looped inverter has the lower impedances, then the output of both inverters will appear to be stuck-at 0 or 1, depending on what value is stored in the latch. Under very unusual circumstances, it is possible for the latch circuit to exhibit metastable behavior. This behavior is discussed later in Section 3.4.2.

3.3.2. Response of Circuits with Opens

Opens that occur in series with transistor channels are very easy to analyze. For all three types of circuits we have studied, it is only necessary to replace the open transistor with the series combination of the channel resistance and the open resistance. This new resistance may

---

*Questions have been raised about several of the theorems in this paper (see [38]). The disputed theorems all concern the detection and location of bridging faults, not the conditions necessary for oscillation.

now be used in the equations we have already derived for switching speed and voltage limits for each of the circuit types. If the open occurs in the driver transistor, Z is decreased and Z' is increased. If the open occurs in the load transistor, Z is increased and Z' is decreased. If the resistance of the open is very large (i.e., much greater than the resistance of a transistor), the output of the inverter will either be stuck-at 0 or stuck-at 1, depending on whether the open is in series with the load or driver transistor, respectively. As discussed in Chapter 1, high resistance opens in CMOS NAND or NOR gates and NMOS gates fed with pass transistor logic, result in stuck-open type faults. If a high resistance short occurs in NMOS NAND or NOR gates, either one of the imputs appears to be stuck-at 0 (driver transistor open), or the output appears to be stuck-at 0 (load transistor open). In the dynamic circuit, high resistance opens cause the output node to appear to be either stuck-at 1 (driver transistor open), or stuck at 0 (load transistor open). A high resistance open of the coupling transistor in general leads to unpredictable behavior.

Low resistance opens in series with the gate terminal of a transistor significantly reduce the speed of NMOS and CMOS inverters. For an NMOS inverter, the capacitance of the driver transistor's gate must be charged though the open. In a CMOS inverter, two cases are possible. If the short affects both driver and load transistors, then the capacitance of both transistors must be charged through the open. If the short affects only one of the transistors, then the shorted transistor turns off and on more slowly than the other transistor. An open gate terminal to the depletion load transistor of an NMOS inverter has very

little effect on circuit operation [4]. The primary reason the gate-to-source connection has so little effect on circuit operation is due to capacitive feed-through from the source terminal to the gate terminal. A parasitic capacitance exists between the gate and source of a transistor. Any rapid change at the source terminal is coupled to the gate terminal. It is difficult to predict the circuit behavior if a depletion transistor's gate should open completely. If no signal levels change on the chip for a long period of time, the charge will eventually leak off the gate [28]. Charge leakage causes an n channel device to be off and a p channel device to be on. This analysis, however, fails to account for capacitive feed-through. Any transistor whose drain or source is connected to a clock or other rapidly changing signal will experience capacitive feed-through to the gate. As a result, the gate voltage will be constantly changing. Whether or not the gate voltage ever gets above (below for a p channel device) the threshold voltage will depend on the particular details of the circuit. Since a large percentage of the transistors in dynamic circuits have a source or drain connected to a clock signal, capacitive feed-through is an important factor. If the gate is connected to a long interconnection, and the open occurs at the end of the interconnection away from the gate, then the interconnection will act as an antenna collecting all the noise and other signals in the vicinity. This essentially random signal drives the inverter which in turn amplifies it and distributes it to other circuits.

3.3.3. Response of Circuits to Noise

During normal operation, an integrated circuit is constantly exposed to noise. This noise is of two types, random noise due to various physical processes (we call this physical noise) and capacitive or inductive coupling of signals as well as any external electrical disturbances (we call this coupling noise). The most common types of physical noise are thermal noise, shot noise, and quantum noise [39]. These types of noise are usually modeled as an independent random white Gaussian process. Wallmark [40] has developed a statistical model for capacitive and inductive coupling. For large circuits, especially those consisting of a large percentage of random logic, he shows that the coupling noise may also be considered as another random noise source. He also treats device variations (random fluctuations in geometric and process parameters) in a similar fashion. Under Wallmark's assumptions, the total rms voltage due to all sources of noise is three to four times the value of physical noise alone.

For proper operation, the circuit must be designed to work correctly in the presence of noise. Although it is impossible to make a circuit totally immune to noise, it is possible to make a circuit relatively insensitive to noise. Usually this is done by making the absolute value of the gain of a circuit small both for values of V_{in} close to logic 1 and for those close to logic 0, while the absolute value of the gain at the transition point is made as large as possible. The absolute value of the gain for V_{in} equal to a logic 0 or 1 must be less than 1. Otherwise, noise is amplified rather than suppressed. Ideally,

the gain at these points should be close to zero. The absolute value of the gain at the transition point should be as large as possible to provide a sharp transition from a logic 0 to a logic 1. Other techniques for maximizing noise immunity are using a large supply voltage and making $V_{in}^{\bullet}$ close to the midpoint of the voltage swing.

Long-term exposure to radiation and hot electron injection increases a circuit's susceptibility to noise. As mentioned in Chapter 2, noise levels in transistors increase after exposure to radiation. In addition, radiation exposure and hot electron injection cause shifts in the threshold voltages and a decrease in transconductance. These parameter shifts may result in values of V_{lo} and V_{hi} closer to the transition point. As V_{lo} and V_{hi} move closer to the transition point, the circuits fed by an affected gate tend to amplify the noise to a greater extent. A reduction in transconductance also tends to reduce the gain of the inverter. Lower gain also reduces an inverter's noise immunity.

For well-designed circuits operating normally, the effect of noise should be soft errors very similar to radiation-induced soft errors. On very rare occasions, a noise spike may be large enough to change the value of an output. As is the case for radiation-induced soft errors, dynamic circuitry is more susceptible than static circuitry.

3.4. Response of Good Circuits to the Output of a Failed Circuit

As shown in the last section, there are a variety of ways a circuit may behave under failure. In some cases, the output of a circuit is a legal logic value although it may not be the correct one, e.g., outputs may exhibit stuck-at or stuck-open behavior. In these circumstances, we

know the response of good circuits which must process the failed circuit's output. The output from the failed circuit is a legal logic value and is processed just as any legal logic values from good circuits would be processed.

However, many of the failures that we have examined may result in outputs which are not legal logic values. Under a variety of failures, it is possible to produce a steady state output which is between V^0 and V^1 (undefined constant logic value). Another possibility is a timing failure. Synchronous systems are designed so that all signals are steady when a clock pulse or edge occurs. A timing error may violate this constraint. A related type of failure is oscillation. When oscillation occurs, the steady signal constraint is once again violated. All three of these types of failures have one important attribute in common; circuits which process these signals are unable to interpret them reliably as being either a logic 0 or logic 1.

3.4.1. Metastable Operation

During normal operation, a system undergoing a state transition shifts from one stable state to another. Unfortunately, under certain circumstances, it is possible for the system to be left in a metastable state. A system is at equilibrium when it is in either a stable or metastable state. In a stable state, a small disruption will cause the system to react in a manner which restores the system to its original state. The larger the disruption, the larger the restoring force until, for a disruption which is large enough, the system changes state. In a metastable state, if a disruption is applied, the system will react by

forcing itself further from its metastable equilibrium condition toward some stable state. Eventually, the system will come to rest in a stable state. Unfortunately, the system may remain in a metastable state for an unbounded time period.

As an example of such a system, consider a bistable element. Such an element can store one bit of information. A power or energy function is associated with any such element. For an inverted pendulum or other mechanical bistable element, this associated function is the system's potential energy. For a flip-flop, this associated function is called the dissipative function (see [29] for a discussion of the dissipative function). A stable state is represented by a local minimum in the element's associated function. A metastable state is represented by a local maximum. A bistable element must have two local minima corresponding to its two stable states. For any continuous function, however, between any two local minima, there must also exist at least one local maximum. Therefore, between any two stable states, there must always be a metastable state.*

A certain amount of energy or power (depending on the memory element) is required to switch the state of a flip-flop. If the input sig-

---

*By flip-flop, we mean a static restoring memory element. We do not use the term flip-flop for a dynamic memory element where information is stored as charge on a transistor. A dynamic memory element has a constant dissipative function. Such an element has an infinite number of stable states. Any disruption to such an element, no matter how small, will simply move the element to another of the infinitely many stable states. Due to the flatness of the dissipative function, the element has no restoring or nonrestoring response to a disruption. In a dynamic memory element, there is no distinction between stable and metastable states.

nal does not have quite enough power or energy to complete the flip-flop's transition, the flip-flop may be left in a metastable state. Such a pulse is called a runt pulse. A runt pulse lacks the duration and/or amplitude required to change the flip-flop state reliably. In a properly designed system, there are only two ways that the system will be left in a metastable state: a synchronization failure, or a component failure. In both cases, a runt pulse is presented to a flip-flop.

Synchronization failures result when a synchronous system must accept a nonsynchronous input. Such an input may change at any time with respect to the system clock. For a synchronous system to work properly, all inputs must be stable before the clock pulse arrives. In order to accomplish this, the asynchronous signal is usually presented first to a clocked flip-flop. Unfortunately, as we have already shown, the flip-flop has a metastable state. If the asynchronous signal should change during a very small window with respect to the clock, the flip-flop may be left in a metastable state. Such a situation is referred to as a synchronization failure. Notice that such a synchronization failure can occur without any part of the circuit experiencing a physical failure.

Several suggestions have been proposed to prevent synchronization failures. One approach is to design an asynchronous network to perform the synchronization function. One such network is a time-bound arbiter. Unger [41] has developed a technique for designing asynchronous networks including time-bound arbiters. Unfortunately, Unger's technique depends on the use of a device called an inertial delay. There is some question

as to the realizability of an inertial delay. Marino [42] has investigated three proposed inertial delay designs and has demonstrated that they are all unreliable. In addition, Strom [43] has shown that a time-bound arbiter and an inertial delay are equally realizable since an inertial delay may be built from a time-bound arbiter and vice versa.

In a more general study, Marino [44] has proposed an extremely general model for any system that exhibits sequential behavior. The only restriction imposed by this model is that the system is nonanticipatory. Using this model, Marino has shown that unless certain relationships between the inputs can be guaranteed, metastable operation is unavoidable.

Several techniques have been proposed to eliminate synchronization failures. The only proposed technique which will prevent synchronization failure was first suggested by Chaney et al. [45]. This method uses flip-flops to synchronize the asynchronous inputs. Instead of attempting to prevent the input flip-flops from entering a metastable state, circuitry is included to detect a metastable state. If this circuitry detects a metastable state, the clock signal is delayed until the metastable state is resolved. Such an approach is clearly not satisfactory for all applications, since an adjustable frequency clock is required. In addition, the maximum clock period is unbounded since the time for a flip-flop to exit from its metastable state is also unbounded. A practical compromise is to reduce the probability of synchronization failure below some "acceptable" level [46,47,48].

Another source of metastable operation is component failure. If a failure occurs, the timing and/or voltage levels of signals produced by the failed components may present a runt pulse to a flip-flop. Oscillation at the input of a flip-flop can also cause runt pulses and thus metastable operation. Regardless of whether the failure is a synchronization failure or a component failure, metastable operation is caused by a runt pulse being presented to a flip-flop.

Researchers have found two modes of metastable behavior in flip-flops [49,50]. In one mode of behavior, the outputs of the flip-flop remain for some time at a level between V^0 and V^1. In the second mode of behavior, the outputs of the flip-flop oscillate. In both cases, other gates receiving the outputs of the flip-flop will be unable to interpret the flip-flop's state reliably. Some gates may interpret the state as a logic 0, while others may interpret the state as a logic 1. Still others may themselves produce an illegal logic output.

A variety of researchers have examined the probability of failure due to metastable operation [29,46,50,51]. Unfortunately, in these prior studies, only synchronization failure is considered as a cause of metastable operation. Generally, a synchronizing flip-flop is considered for processing an synchronous input occurring at some average frequency f. In [29], it is estimated that metastable operation will occur if the asynchronous input changes within a window of width $0.1\tau_{SW_{ave}}$. The value of $\tau_{SW_{ave}}$ is that of the cross-coupled gates which form the flip-flop. Therefore, metastable operation occurs with a probability of $0.1f\tau_{SW_{ave}}$ per synchronization event. Clearly, the faster

the gate used in the synchronizer flip-flop, the lower the probability of the flip-flop entering a metastable operation. On the other hand, this advantage is lost if a faster synchronizer is forced to synchronize more events (i.e., f is higher).

The probability of a synchronizer leaving a metastable state before some time t, is usually modeled as a Poisson process with rate ρ [29,51]. Under a number of simplifying assumptions, it can be shown that [51,46]

$$\rho = \frac{A^*}{2\tau_{SW_{ave}}}$$

Therefore, for everything else equal, the lower the value of $\tau_{SW_{ave}}$ (i.e., the faster the flip-flop) and the higher the value of A^*, the lower the probability of failure from synchronization failure. The probability of a synchronizer being in a metastable state at time t, p(t) is

$$p(t) = 0.1 f \tau_{SW_{ave}} e^{-\frac{A^* t}{2\tau_{SW_{ave}}}}$$

These equations should be used with caution. They were derived under a number of simplifying assumptions. Lacroix et al. [52] found a three order of magnitude difference in the average length of metastable operation for 7475 D latches from different vendors. It is quite doubtful that the gain-bandwidth product would vary enough to account for this difference. Pechoucek [50] found that from a random sample of 74S74 flip-flops, the flip-flops with the smallest delay exhibited longer

average length of metastable operation than those flip-flops with a larger delay.

If we make the assumption that during a timing failure, the added delay modulo the clock period is uniformly distributed, then the same equations derived for synchronization failure will also apply to timing failures. This assumption is somewhat tenuous. One could reasonably expect the actual probability of metastable operation due to a timing failure to be several times higher than that predicted by the synchronization failure analysis. Since most device failures affect both timing and logic levels, it is very difficult to estimate the probability of metastable operation due to a component failure. If, however, the component failure does not occur in the flip-flop itself, then the average length of metastable operation should be the same regardless of what caused the metasable operation in the first place. Based on the equations, the gain-bandwidth product of the flip-flop gates should be as large as possible in order to minimize the average time of metastable operation. If NMOS gates with nonsaturated loads are used, then the gain of the flip-flop gates will be limited. To increase a gate's gain, we must increase the value of β_r. Since Z is proportional to β_r, any increase of β_r also increases Z. For large Z, $\tau_{SW_{ave}}$ is proportional to Z. Therefore, if β_r is large, the gain-bandwidth product actually decreases by a factor of approximately $\beta_r^{1/2}$ as β_r is increased. Better synchronizers can be built using NMOS with a saturated load or CMOS. Both of these types of gates will have inherently larger gains than the nonsaturated NMOS gates. Any attempt, however, to increase the gain-bandwidth product of a saturated NMOS or CMOS gate by increasing channel

length is futile. As the channel length is increased, gain increases, but switching time decreases due to increased resistance and capacitance. The implications of scaling on metastable operation is examined in [51]. If the number of devices are increased by the scaling process and clock speeds are increased as gate propagation delays decrease, the average length of metastable operation is roughly invariant.

3.4.2. Response of Combinational Logic

Combinational logic is only susceptible to timing errors if it is part of a sequential machine or if it must produce its output in some bounded period of time. Unfortunately, nearly all cases of practical interest are included in this case. In addition, combinational logic is susceptible to oscillation and illegal constant logic values.

If the combinational logic is part of a synchronous sequential machine, then a timing failure may result in incorrect behavior. In order for a synchronous sequential machine to operate properly, it is necessary for all outputs from the combinational logic to have reached their steady state values before the arrival of the next clock pulse. In the event of a timing failure, one or more of the outputs may be in the process of changing at the same time that the clock pulse arrives. In this case, it is not possible to predict whether the affected combinational logic outputs will be interpreted as a logic 0 or a logic 1. Any outputs which are delayed may be incorrectly interpreted. Although asynchronous sequential machines do not depend on all internal signals settling before a clock pulse arrives, they are still susceptible to timing failures. A large class of asynchronous circuits have essential

hazards which cannot be eliminated*. In these circuits, excessive delays in part of the circuit may result in an erroneous state transition. Furthermore, asynchronous sequential circuits are usually designed under the assumption that after an input changes, all signals in the circuit settle before further input changes occur. Timing failures can lead to the violation of this assumption.

Any node which oscillates will cause other nodes which are sensitized to it to oscillate also (the conditions required for sensitization are discussed in Chapter 4). If the outputs of a combinational logic block are sensitized to an oscillating node, then they may be interpreted as either a logic 1 or a logic 0 by following logic.

If the input of a gate is at a voltage close to its transition point (i.e., an illegal constant logic value), then its output voltage may also be close to its transition point. Similarly, other inverters which receive this inverter's output as their input may have their output voltages close to their transition points. Consider a string of n identical inverters where the first inverter's input is at its transition point. The first several inverters will have output voltages which are close to the transition point. Any noise present in the system will tend to force the inverter's output voltage away from its transition point. Intuitively, inverters at the beginning of the string would be expected to have a relatively high probability of being close to the transition point. Inverters further down the string would be expected to have a much lower probability of being close to the transition point due to each inverter's amplification. Inverters at some distance from

the beginning of the string would be expected to oscillate between V_{hi} and V_{lo}.

We now develop a simplified model in order to determine the approximate probability that a given inverter's output is greater than V^1 or less than V^0. Figure 3.14 shows a string of inverters and the simplified model which we use. Each inverter is modeled as an ideal finite gain, finite bandwidth amplifier. Each amplifier has a transfer function $H(\omega)$. At the input of each amplifier is a summing point where noise from a noise source is added to the output from the previous amplifier. Each noise source is assumed to be a Gaussian white noise source and each source is assumed to be statistically independent of every other source. For the sake of convenience, the transition point is taken to be zero while V^0 is assumed to be negative and V^1 is assumed to be positive.

Referring to Figure 3.14, the response at point y due to some noise source i ($1 \leq i \leq n$) is

$$Y_i(\omega) = [H(\omega)]^{n+1-i} W_i(\omega)$$

The power spectrum density of y_i [39] is

$$S_{y_i} = |H(\omega)|^{2(n+1-i)} S_{w_i} df$$

while the variance is

$$VAR_{y_i} \equiv \int_{-\infty}^{\infty} |H(\omega)|^{2(n+1-i)} S_{w_i} df$$

The central limit theorem states that when several independent random variables are summed, the variance of the sum is the sum of the vari-

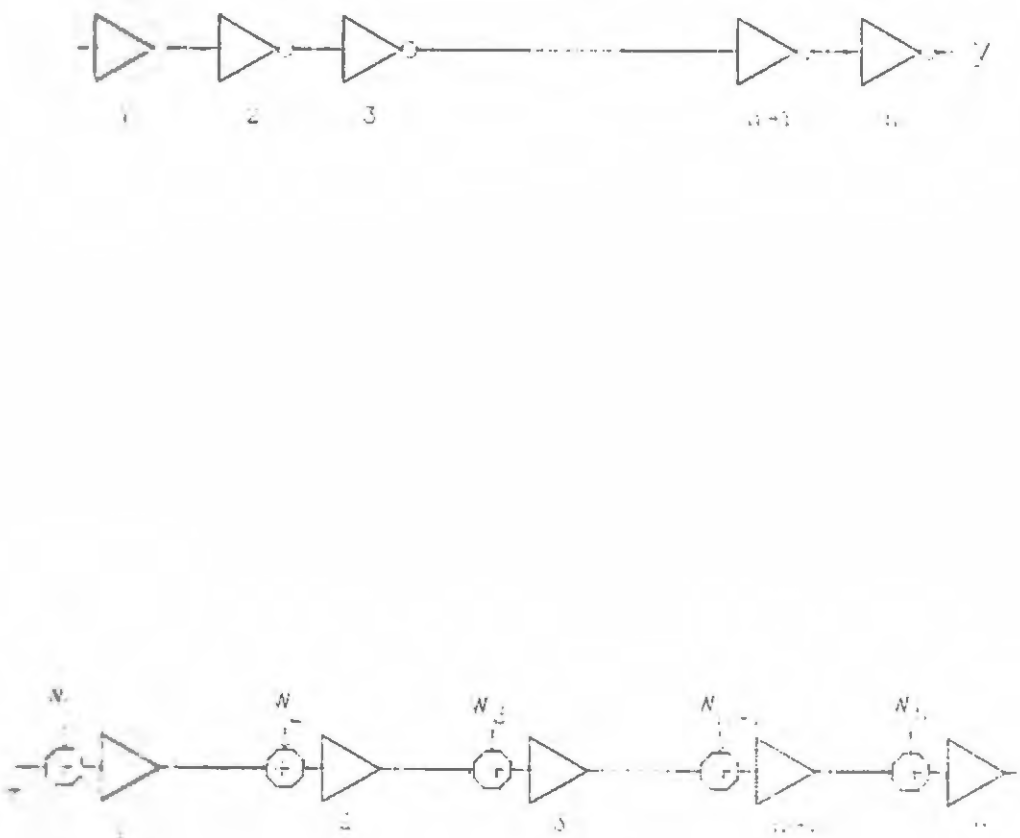


Figure 3.14. Model of Inverter String.

ances [39]. Therefore, the variance due to all the noise sources is

$$VAR_y \equiv \sum_{i=1}^{n} \int_{-\infty}^{\infty} \|H(\omega)\|^{2(n+1-i)} S_{w_i} df$$

We assume that every source has identical statistics (i.e., $S_{w_1} = S_{w_2} = \cdots = S_{w_n} = S_w = N_0/2$). The variance is now

$$VAR_y = \frac{N_0}{2} \sum_{i=1}^{n} \int_{-\infty}^{\infty} \|H(f\omega)\|^{2(n+1-i)} df$$

If the amplifiers are assumed to have a single pole, then

$$H(j\omega) \equiv \frac{A}{1 + j\omega\gamma}$$

and

$$\|H(\omega)\|^2 \equiv \frac{A^2}{1 + j\omega^2\gamma^2}$$

where A is the gain of the amplifier and γ is the reciprocal of the amplifier's cut-off frequency. Using the fact that $\omega = 2\pi f$, the expression for variance can be rewritten as

$$VAR_y \equiv \frac{N_0}{2\pi} \sum_{i=1}^{n} A^{2(n+1-i)} \int_{0}^{\infty} \left[\frac{1}{1 + \omega^2\gamma^2} \right]^{n+1-i} d\omega$$

Using a table of integrals and simplifying, the variance is

$$VAR_y \equiv \frac{N_0}{2\gamma\pi^{1/2}} \sum_{i=1}^{n} \frac{A^{2(n-i+1)}}{2n + 2i - 1} \left[\frac{\text{gamma}(n - i + 1.5)}{\text{gamma}(n - i + 1)} \right]$$

In order to avoid the gamma function, we developed the following approximation

$$\frac{\text{gamma}(x + .5)}{\text{gamma}(x)} \approx \left[x - .25 + \frac{1}{(2500x)^{1/2}}\right]^{1/2}$$

For integer values of x between 1 and 70, the error is less than 1 percent for this approximation. If we substitute this approximation into the equation for VAR_y, we find

$$VAR_y \sim \frac{N_0}{2\gamma\pi^{1/2}} \sum_{i=1}^{n} \frac{2^{2(n-i+1)}}{2n+2-i} \left[n - i + 0.75 + [2500(n - 1 + 1)]^{1/2} \right]^{1/2}$$

Let P^* be the probability that $y \leq -a$ or $y \geq a$. In other words, P^* is the probability that y is at least a distance of a from the transition point. Since y has a mean of zero, it is easy to show that

$$P^*(a, N_0, \gamma, A, n) = 1 - \text{erf}(a[2VAR_y]^{-1/2})$$

where erf() is the error function. P^* is most strongly dependent on the inverter's gain, A. This dependence is due to the fact that the variance of y is proportional to $A^{2(n+1)}$. As n becomes large, this factor will increase rapidly as A increases. Figure 3.15 is a graph of P^* vs n as A is varied from 2 to 100. The value of γ is based on a SPICE simulation of an inverter designed using Mead-Conway [29] design rules for a 5 micron process. a was arbitrarily chosen to be 1 volt while N_0 was roughly equal to the thermal noise present at room temperature. As pointed out in [40], the total noise in the circuit will probably be several times this value. The SPICE simulation of the Mead-Conway inverter had a value of A equal to 2.27. The graph shows that P^* is very close to zero for small values of n. When n increases beyond a certain value, P^* increase rapidly until it becomes almost equal to one.

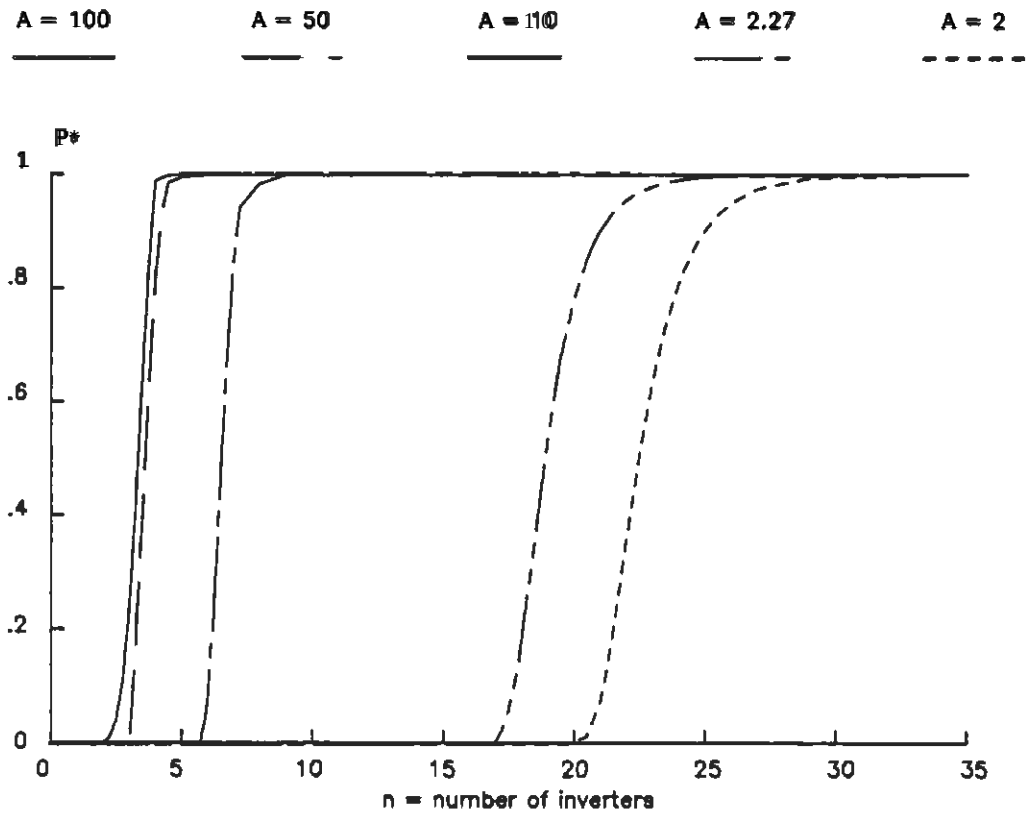


Figure 3.15. Probability of $|y| \geq a$ vs. Number of Inverters for Small Noise.

The value at which P^* begins its rapid rise is highly dependent on A. The larger the value of A, the sooner P^* begins its rapid increase. The slope during this increase is also larger for larger values of A.

In Figure 3.15, we assume a very low value for N_0. Figure 3.16 is another graph of P^* as A is varied from 2 to 100. The values of a and γ remain the same in Figure 3.16, but the value of N_0 is increased by a factor of 10 from the value used to derive Figure 3.15. The value of N_0 used for Figure 3.16 is probably much larger than the actual total noise in a circuit.

The graph of Figure 3.16 is very similar in shape to the graph of Figure 3.15. The only appreciable difference is that the graph of Figure 3.16 is shifted roughly one inverter to the left with respect to the graph of Figure 3.15. In other words, the effect of increasing the value of N_0 by a factor of 10 is approximately the same as the effect of adding one more inverter to the chain. By examining the equation for VAR_y, it is apparent that an increase in A, N_0, and n leads to an increase in the value of VAR_y and thus P^*. Likewise, increases in a and γ decreases the value of both VAR_y and P^*. In order to maximize the value of P^*, circuits should be designed to maximize gain and bandwidth. Note that gain is more important than bandwidth in maximizing P^*.

It is important to consider the behavior of a node when its voltage leaves the range of $\pm$ a. As long as the node voltage is small, the response of each inverter is approximately linear. The input to the first inverter is white Gaussian noise. The output of the first inverter will be colored Gaussian noise. The frequency components that

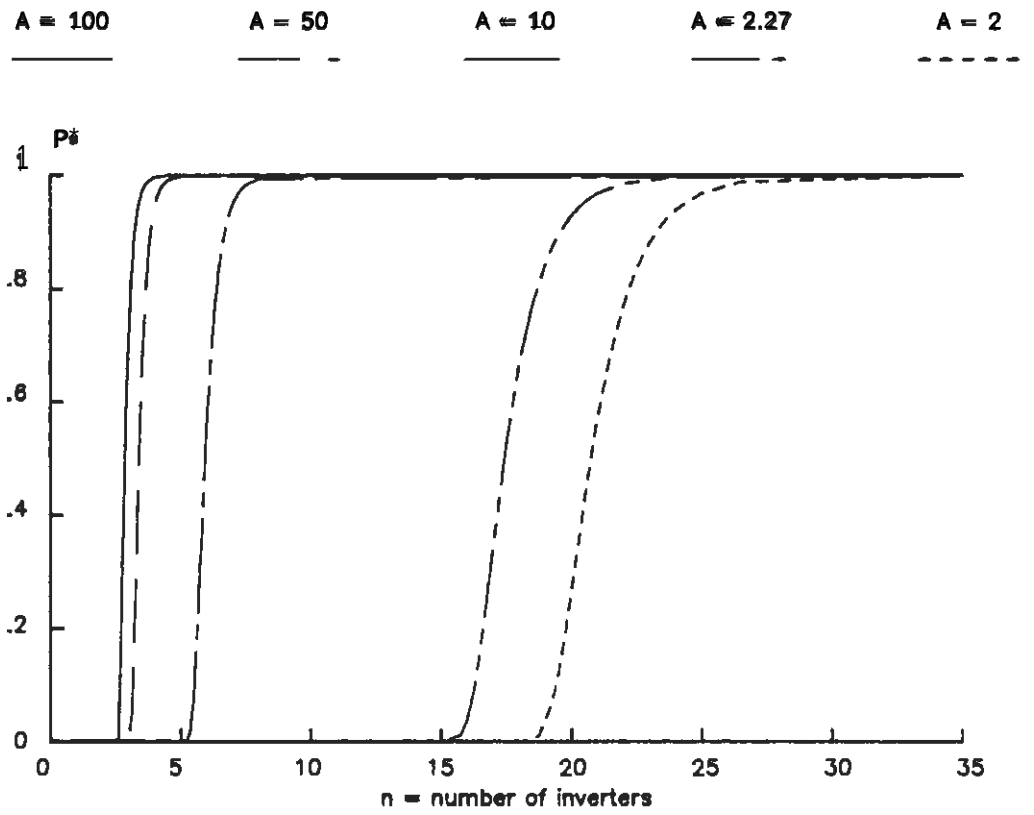


Figure 3.16. Probability of $|y| \geq a$ vs. Number of Inverters for Large Noise.

are removed from the output noise are those frequencies that are too high for the inverter to respond to. The colored Gaussian noise from the output of the first inverter is added to white Gaussian noise and then input to the second inverter. The output of the second inverter is once again colored Gaussian noise. This process is repeated as we progress down the chain of inverters. Finally, the colored Gaussian noise at the input of one of the inverter is large enough so that the linear response assumption is no longer valid. Since the noise from the previous inverter is so large, we may neglect the white noise which is being injected at this node. Therefore, this inverter is being driven by a fairly large colored Gaussian signal. By large, we mean that the signal is large enough to saturate the inverter. The frequencies present in the Gaussian signal are low enough for the inverter to respond to. Therefore, the output of this inverter will be a "clipped" version of its input. Since the colored Gaussian noise is a zero mean process, the inverter outputs will oscillate.

In this analysis, it is assumed that the input to the first inverter in the string is exactly at zero (i.e., its transition point). It is more likely that there will be some small offset, ϵ, from 0. The effect of such a DC offset is to change the mean of the input signal from 0 to ϵ. Likewise, the output signal's mean is changed from 0 to ϵA. In general, the output from the ith inverter has a mean value of ϵA^i. Since the signal at y no longer has a mean of zero, the probability that $y \geq a$ is no longer the same as the probability that $y \leq -a$. Due to the symmetry of the problem, we may, without loss of generality,

assume that a is positive. In this case, the response at y is identical to our previous analysis except that εA^n is added to the signal.

Let y' be the original signal at y (i.e., the value at y if $\varepsilon = 0$). Therefore, $y = y' + \varepsilon A^n$. The original value of P^* was defined to be the probability that $y \leq -a$ plus the probability that $y \geq a$. Since the value at y is the sum of the original signal at y and εA^n. Therefore, P^* is the probability that $y' \leq -a - \varepsilon A^n$ plus the probability that $y' \geq a - \varepsilon A^n$. It is easy to show that

$$P^* = \frac{1}{2}[1 - \text{erf}[(a - \varepsilon A^n)(2\text{VAR}_{y'})^{-1/2}] + 1 - \text{erf}[(-a - \varepsilon A^n)(2\text{VAR}_{y'})^{-1/2}]$$

As εA^n becomes large, then

$$1 - \text{erf}[(a - \varepsilon A^n)(2\text{VAR}_{y'})^{-1/2}] \gg 1 - \text{erf}[(-a - \varepsilon A^n)(2\text{VAR}_{y'})^{-1/2}$$

Therefore, for large values of εA^n, an approximation for P^* is

$$P^*(a, N_\theta, \gamma, A, n, \varepsilon) \sim \frac{1}{2}[1 - \text{erf}[(a - \varepsilon A^n)(2\text{VAR}_{y'})^{-1/2}]$$

As the value of e becomes larger, the probability that y is outside the range of -a to a increases.

This analysis demonstrates that if an illegal constant logic value occurs at a node, then other nodes that are sensitized to the illegal value node may either oscillate or also have an illegal logic value. The more levels of logic between the nodes, the greater the probability of oscillation.

In most cases, the output of a chain of gates drives the input of a flip-flop. If a failure has occurred so that the input of one of the gates is forced to its transition point, the flip-flop may enter a meta-stable state. If the output of the last gate in the chain is still

close to its transition point, then the probability that the flip-flop enters a metastable state is relatively high. The flip-flop only enters a metastable state if its input is in the vicinity of the transition point. If the input is oscillating with a very small amplitude (i.e., the input is near the transition point), then the probability of entering a metastable state is much higher than if the input is oscillating between the voltage limits of the circuit. The most effective way to minimize the probability of a metastable state is to keep the input to the inverter as far away from the transition point as possible. Therefore, the higher the probability that the output from the last gate in a chain is a legal logic value, the lower the probability of metastable operation. From Figure 3.15, the probability of a legal logic value approaches 1 as the chain length becomes longer. This would seem to imply that as the chain length becomes long, the last gate's output spends a smaller and smaller percentage of its time in a region near the transition point. If this is true, then as the chain becomes long, the probability that the flip-flop enters metastable operation approaches zero. Unfortunately, in our analysis, we have neglected the fact that a gate has a finite slew rate (i.e., the output of a gate can only change at some maximum rate). As the gate output oscillates back and forth from one voltage to the other, it takes a finite time to switch from one logic level to the other. Therefore, the gate output must spend some nonzero time in the region of the transition point.

We have already calculated the switching time for an inverter. In the section on metastable operation, we estimated that there is window width of approximately $.1\tau_{SW_{ave}}$ during which time the flip-flop can

enter a metastable state. When the output is oscillating between the voltage limits of the circuit, it should typically be switching at a speed fairly close to $\tau_{SW_{ave}}$. Therefore, when the probability of a legal logic value is very close to 1, the probability that the flip-flop enters a metastable state should be approximately 0.1. On the other hand, when the probability of of a legal logic value is very low, the probability that the flip-flop enters a metastable state is quite high. Therefore, it is important that the probability of having a legal logic value is as high as possible.

In summary, when a component failure occurs in combinational logic, three types of illegal logic values may result: timing failures, oscillation, and illegal constant logic values. Synchronization failure may also result in an illegal logic value. For the rest of this manuscript, we restrict our scope to synchronous sequential systems. These systems consist of blocks of combinational logic followed by some type of clocked bistable elements. If static flip-flops are used, then it has been shown that when sensitized to an illegal logic value in the combinational logic, these flip-flops may either assume a legal (although possibly incorrect) logic value or an illegal logic value (by entering and remaining in a metastable state). If the output of one or more flip-flops assumes an illegal logic value, then these illegal logic values are presented to the combinational logic block following the latches. Although it is possible for illegal logic values to propagate through many blocks of combinational logic, it is unlikely since properly designed flip-flops have a high probability of leaving a metastable state well within one clock period. Obviously, the longer the system

clock period is with respect to the combinational delay, the lower the probability that an illegal logic value propagates through more than one block of combinational logic. If dynamic latches are used, the probability that an illegal logic value propagates through several combinational blocks is much higher, since dynamic latches do not attempt to resolve an illegal logic value to a legal logic value. For this reason, static flip-flops are to be preferred over dynamic latches.

Many of the physical failure modes in Chapter 2 result in a gradual degradation of switching speed and inverter gain. Such failures include hot electron injection, exposure to ionizing radiation, and electromigration. From analyzing the probability of metastable operation and illegal constant logic level propagation along a chain of inverters, it is clear that the degradation of gate performance has a very negative influence on the circuits ability to react to undefined logic values. The average length of metastable operation is proportional to the gain-bandwidth product. The probability of producing a legal logic value from a chain of inverters, is a very strong function of gain and is also influenced by the bandwidth of the inverters. As circuits degrade, timing becomes more critical since all gates in the circuit become slower, but not necessarily by the same amount. In addition, the lowering of the gain coupled with the decrease in bandwidth makes flip-flops more likely to enter a metastable state and more likely to stay in the metastable state for a longer time.

CHAPTER 4

Concurrent Error Detection of Physical Failures

In the last chapter, we developed an understanding of how circuits behave when they fail. The behavior of good circuits which must process the outputs of failed circuits was also discussed. We are now in a position to develop concurrent error detection schemes for physical failures.

4.1. Indeterminate Faults

The analysis presented in Chapter 3, demonstrated the diverse ways in which a digital circuit may behave when it fails. Most of the classical faults are only capable of accurately modeling a subset of all failures. Physical failures which result in timing failures, oscillations, or illegal logic levels are very poorly modeled by the classical fault models. Synchronization failures also result in circuit behavior which is not well modeled by the classical models. These failures all result in circuit outputs which cannot be reliably interpreted as either a logic 0 or a logic 1 and are hence called indeterminate values.

If all but one input to an AND (or NAND) gate is a logic 1 while the remaining input is an indeterminate value, then it is possible for an indeterminate value to appear at the output of the AND gate. If, however, at least one input to an AND gate is a logic 0, then any indeterminate input present at any other inputs does not propagate to

the output of the gate. Instead, the output of the gate is a logic 0. Similarly, for an OR (or NOR) gate, if all but one input is a logic 0, then an indeterminate value may propagate. If one or more inputs to the OR gate is a logic 1, then its output is a logic 1 and the indeterminate value does not propagate. An indeterminate value input to an inverter may always be propagated to its output.

When an indeterminate input to a gate may be propagated to the gate output, we say that the output of the gate is sensitized to the input with the indeterminate value. Therefore, AND and NAND gates are sensitized to an indeterminate value when all inputs other than the input (or inputs) with an indeterminate value have logic 1 values. OR and NOR gates are sensitized to an indeterminate input when all inputs other than the input (or inputs) with an indeterminate value have logic 0 values. Inverters are always sensitized to an indeterminate value.

It is important to realize that simply because a gate is sensitized to an indeterminate value which is occurring at one of its inputs does not necessarily insure that the gate output is an indeterminate value. In this case, the output may be either a legal logic 0, an indeterminate value, or a legal logic 1. If the indeterminate input happens to be due to oscillation, then the output of a sensitized gate is usually an indeterminate value. If the indeterminate input is either an illegal constant logic value or a timing failure, then the value assumed by the sensitized output depends on such factors as the gate's delay, the gate input's transition point, and the noise which is present in the circuit at that instant. For this reason, the response of circuits with

indeterminate value inputs is, in general, nondeterministic. Effects of indeterminate errors may not be repeatable and a signal which is fanned-out may be interpreted differently at distinct destinations.

Indeterminate faults are a very general type of fault. Most of this chapter is based on the following hypothesis

> Hypothesis: An indeterminate value at a node is the most general type of single node failure.

This hypothesis is due to the fact that when an indeterminate value occurs, it may be subsequently interpreted as either a logic 0, an indeterminate value, or a logic 1. Therefore, indeterminate failures are able to represent not only the nondeterministic behavior but also the deterministic behavior of many classical faults. In this sense, stuck-at faults, stuck-open faults, and any other fault which forces a node to a legal logic value are only special cases of indeterminate failures.

4.1.1. Ternary Algebra

In order to analyze digital systems which operate on indeterminate values, it is helpful to have an appropriate algebra. Since such an algebra must deal with an alphabet of three distinct values, Boolean algebra is clearly inadequate. A ternary algebra [53] however has the three required levels. The three values may be represented as {0, u, 1} with the property that $0 \leq u \leq 1$. When a signal undergoes a transition from a voltage which is less than V^0 to a voltage greater than V^1, then the ternary algebra models this transition as the sequence 0 -> u -> 1 [54]. Likewise, a negative transition is modeled by the sequence 1 -> u

-> 0. In order for the algebra to be useful, there must be a mapping between Boolean functions and ternary functions. The ternary functions MIN, MAX, and INV may be defined as

$$Y = MIN[x_1, \ldots, x_n] \leq x_i \text{ for } (1 \leq i \leq n) \text{ and } Y \in \{x_1, \ldots, x_n\}$$

$$Y = MAX[x_1, \ldots, x_n] \geq x_i \text{ for } (1 \leq i \leq n) \text{ and } Y \in \{x_1, \ldots, x_n\}$$

$$INV[x_i] = \bar{x}_i$$

where $x_1, \ldots, x_n$ represents the n ternary inputs to the functions and 1 - u is defined to be u. Figure 4.1 gives the truth tables for these ternary functions for two inputs. An examination of these functions when the inputs are all either 0 or 1 shows that MIN is the ternary equivalent of AND, MAX is the ternary equivalent of OR, and INV is the ternary equivalent of NOT [53,54].

If MIN, MAX, and INV are substituted for AND, OR, and NOT, then many of the laws of Boolean algebra are also valid for ternary algebra. These laws include idempotency, commutativity, absorption, associativity, distributivity, involution, and De Morgan's law [53]. The complementation law, however, does not extend to ternary algebra. That is;

$$MIN[x_i, INV[x_i]] \neq 0$$

and

$$MAX[x_i, INV[x_i]] \neq 1$$

The ternary value u will be used to represent two cases. The value u indicates either that a signal has an indeterminate value or that the

MIN	0	u	1
0	0	0	0
u	0	u	u
1	0	u	1

MAX	0	u	1
0	0	u	1
u	u	u	1
1	1	1	1

INV	
0	1
u	u
1	0

Figure 4.11 Ternary Algebra Truth Tables

value is a usually unknown* (but legal) Boolean value. Therefore, a u may represent a logic 0, an indeterminate value, or a logic 1. This is useful since if a gate is sensitized to an input which is an indeterminate value, its output may be either an indeterminate value or a legal Boolean value. It is therefore possible to use the ternary algebra to determine whether or not a gate is sensitized to a particular input with a given input vector. The gate is replaced with its ternary equivalent (i.e., a MIN gate replaces an AND gate, a MAX gate replaces an OR gate, and an INV gate replaces a NOT gate). The value of the particular input is set to u while all other inputs are set to the values given in the input vector. If the gate output is u, then the gate is sensitized to the input. If the gate output is a 0 or 1, then the gate is not sensitized.

The concept of sensitization may be defined for any combinational function. A function is sensitized to a particular node (or nodes) of the circuit if under a given input vector and an indeterminate value at the particular node (or nodes), an indeterminate value may occur at the function output. In order to determine if the combinational function is sensitized to a particular set of nodes under a given input vector, the gates in the function must be first transformed into their ternary equivalents. The input vector is then applied to the ternary function. Finally, the particular set of nodes are set to the value u. If the output of the function is u, the function is sensitized to the set of

---

*That is, the value is usually unknown a priori. We discuss in Section 4.1.2 why a static hazard may make the Boolean value predictable.

nodes under the given input vector. It should be noted that sensitization is always defined with respect to some input vector.

The concept of sensitization developed here for indeterminate logic values is analogous to the concept of path sensitization for stuck-at faults [55]. A node is said to be path sensitized to an output with respect to an input vector if a change in the Boolean logic value at the node results in a change in the Boolean logic value at the output. The method of Boolean differences [56] can be used to determine whether or not an output is path sensitized with respect to a particular node with a given input vector. Let y be the output of f, some Boolean function, and $\bar{X} = x_1, \ldots, x_n$ be the input vector. Let q be some node in the circuit which implements f. Then q must be some function of X which we shall call g. Therefore, $q = g(X)$ and $y = f(X,q)$, and y is path sensitized to q if and only if

$$\frac{df(X,q)}{dq}\Big|_{\bar{X}} = f(\bar{X},0) \oplus f(\bar{X},1)\Big|_{\bar{X}} = 1$$

Otherwise, y is not path sensitized to q. The Boolean difference being 1 implies that, for the given assignment of values to X, any change of the Boolean logic value at q results in a change of the Boolean logic value at the function's output.

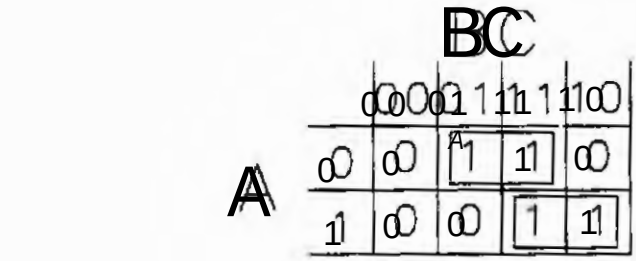
The Boolean difference may be computed for ternary functions as well as for Boolean functions. If the inputs to a ternary function are Boolean values, the ternary function and a Boolean equivalent of the ternary function produce the same result. Therefore, the Boolean difference of a ternary function may be computed by finding the Boolean difference of the ternary function's Boolean equivalent.

Theorem 1: If a node is path sensitized to a second node with a given input vector, then it must also be sensitized to an indeterminate failure at the second node with the same input vector.

Proof: Consider the ternary model of the circuit. For Boolean inputs, the behavior of the ternary model must be identical to the behavior of the Boolean equivalent. Let node a be path sensitized to node b for a given input vector. Since node a is path sensitized to node b, when the value of node b is set to 0, then node a will assume value d and when the value of node b is set to 1, then node a will assume the value $\bar{d}$, where d is 0 or 1. If the value of node b is set to u, then the circuit may interpret the value of node b as either 0 or 1. Therefore, the value of node a may be either 0 or 1 when node b is set to u and node a is path sensitized to node b. Consequently, when a node is path sensitized to a second node, it must also be sensitized to an indeterminate failure at the second node.

4.1.2. The Effects of Hazards on Sensitization

In the last section, it was shown that when the Boolean difference of a function is equal to 1, with respect to some node, the output is sensitized to an indeterminate value at that node. The converse of this statement is not, however, true. That is, if the Boolean difference of a function with respect to some node and input vector is equal to zero, then the function may still be sensitized to an indeterminate value on the node. As an example, consider Figure 4.2. If inputs B and C to the



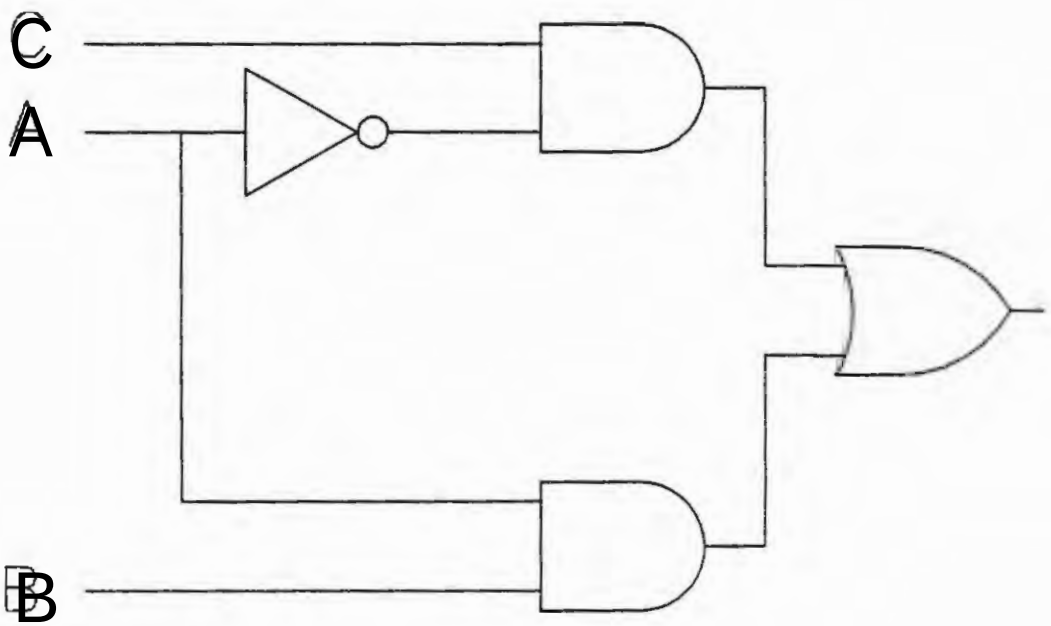


Figure 4.22. Example of a Static Hazard.

function are both a logic 1, then the output should be a logic 1, regardless of the value at the A input. Therefore

$$\frac{df}{dA}\Big|_{B=1,\ C=1} = 0$$

This Boolean difference implies that the output of function f is not path sensitized to input A when B = C = 1. If we consider the equivalent ternary function, however, we see that if input A assumes a value of u, while inputs B and C both have values of 1, then the output is u. Therefore, the output is indeed sensitized to A. This discrepancy between path sensitization for a Boolean value and sensitization for an indeterminate value, is a direct consequence of the fact that the complementation law is not valid for the ternary algebra.

By examining the Karnaugh map of the function, it is evident that a static hazard [56] exists for a transition of input A while B = C = 1. Any time a static hazard exists with respect to an input transition, then the Boolean difference with respect to the input must be zero.

A static hazard occurs when there is reconvergent fan-out along two or more paths, and at least one of the paths has a different inversion parity than the other paths. If x is the input whose transition causes a static hazard, then the reconvergent fan-out either results in $x + \bar{x}$ or $x \cdot \bar{x}$ depending on whether the reconvergence occurs at an OR gate or an AND gate respectively. The complementation law of Boolean algebra guarantees that $x + \bar{x} = 1$ and $x \cdot \bar{x} = 0$, which in turn implies that the Boolean difference with respect to x must be zero. Therefore, the output is not path sensitized to x.

If we use the ternary model, then MAX[x, INV[x]] ≠ 1 and MIN[x, INV[x]] ≠ 0. In particular, if x = u, then MAX[x, INV[x]] = MIN[x, INV[x]] = u. Consequently, the output of the reconvergence gate is sensitized to an indeterminate value at x. If the output of the function is sensitized to one or more of the outputs of the reconvergence gate,* then the function is sensitized to an indeterminate value at input x. By modifying a function, it is always possible to remove a static hazard. In the example of Figure 4.2, the static hazard may be removed by adding the product term B • C to the sum-of-products implementation. This term is redundant but serves to remove the static hazard by desensitizing the reconvergent fan-out from the output of the function. When B = C = 1, the term B • C is 1. The reconvergence gate is an OR gate. An input of 1 from this product term thus forces the OR gate output to a 1 value. This new implementation of the function no longer has its output sensitized to an indeterminate value at input A when B = C = 1.

From this analysis, it is clear that a static hazard implies sensitization. This fact is not surprising since ternary algebra has long been used to detect the presence of hazards in digital circuits [54, 57].

Eichelberger [54] has extended the concept of static hazards to hazards that occur during multiple input transitions. He calls a hazard due to a transition at p of the inputs, a p-variable logic hazard. Let

---

*It is possible for the reconvergence gate to be the same gate as the output gate.

$$X_1 = (x_1, \ldots, x_p, x_{p+1}, \ldots, x_n)$$

and

$$X_2 = (\bar{x}_1, \ldots, \bar{x}_p, x_{p+1}, \ldots, x_n)$$

where X_1 and X_2 represent input vectors of some combinational Boolean function f. Function f is said to have a p-variable logic hazard for a transition from input vector X_1 to input vector X_2 (p variables change in this transition) if and only if

(1) $f(X_1) = f(X_2)$,

(2) all of the 2^p values specified for f in the sub-cube $(x_{p+1}, \ldots, x_n)$ are the same, and

(3) during the input change from X_1 to X_2, a spurious pulse may be present at the output.

For the special case of p = 1, the p-variable logic hazard is identical to a static hazard. Eichelberger shows that by modifying the implementation of a function, it is possible to remove all p-variable logic hazards. In many cases, this will require the addition of redundant logic just as it did for static hazards.

From the above definition of a p-variable logic hazard, it is apparent that whenever a p-variable logic hazard exists, the output of the function is sensitized to indeterminate values at the p variables. This sensitization occurs despite the fact that all 2^p values of f in the sub-cube $(x_{p+1}, \ldots, x_n)$ are the same. Clearly, if these values are not the same, the function must be sensitized.

4.2. Concurrent Error Detection

The goal of concurrent error detection is to detect errors during the normal operation of the system. Ideally, the concurrent error detection scheme should guarantee the detection of all possible errors. A class of circuits has been defined [5] that under a number of assumptions achieves the goal of total error detection. These circuits are designed so that under the proper assumptions, any error results in a non-codeword output from the circuit and the first incorrect output is a non-codeword output. Such circuits are called totally self-checking circuits.

4.2.1. Totally Self-Checking Circuits

Let the input code space be all input vectors that can be applied to a circuit under normal (i.e., fault-free) operation. All output vectors not in the code space are non-codewords. The following definitions are paraphrased from [5]:

> Self-Testing: A circuit is said to be self-testing if for every fault in the fault model, there is at least one sequence of codeword inputs which produces a non-codeword output.
>
> Fault-Secure: A circuit is said to be fault-secure if for every fault in the fault model, the circuit either produces the correct output or a non-codeword output for the entire input code space.
>
> Totally Self-Checking: A circuit is said to be totally self-checking if it is self-testing and fault-secure.
>
> Code Disjoint: A circuit is said to be code disjoint if all non-codeword inputs produce non-codeword outputs.

The following assumptions are made about the operation of a totally self-checking circuit:

(1) Only failures which are modeled by the fault model occur.

(2) Failures occur one at a time with some minimum time interval, τ, between each failure.

(3) The inputs to the circuit are applied often enough to insure that during any time period of length τ, enough inputs are applied to the circuits to test the circuit completely. This assumption is referred to as the testability assumption.

In practice, the period of time between failures is a random variable and is often modeled as a Poisson process. It is possible for two failures to occur in a period of time much less than τ although this is unlikely. If the circuit is completely tested in any time period τ, then τ can be made sufficiently small to insure that the probability that a second failure occurs before the first failure is detected, is low. Note that in the testability assumption, completely testing the circuit, refers only to testing for those faults which are testable.

Under these assumptions, a totally self-checking circuit is able to detect any failure. This fact is guaranteed by the self-testing property and the three assumptions. The self-testing property is necessary to prevent the buildup of undetectable latent faults. The fault-secure property, assures that for any fault from the fault model, all incorrect outputs are non-codewords. We are therefore assured of meeting the goal that the first incorrect output is a non-codeword. This property is referred to as the totally self-checking goal [58]. The totally self-checking property is more restrictive than it needs to be since there

are circuits which are not totally self-checking but which still satisfy the totally self-checking goal. Consider a sequence of faults from the fault model. As each subsequent fault in the sequence occurs*, the behavior of the circuit is modified to reflect the effects of all the faults in the sequence. A sequence of faults is said to be detectable if at least one codeword input produces an incorrect result. The following definition is paraphrased from [58]:

> Strongly Fault-Secure: A circuit is said to be strongly fault-secure if for all possible sequences of faults*, as each fault in the sequence occurs*, the first fault in the sequence which causes the sequence to be detectable only produces correct outputs or non-codeword outputs.

Strongly fault-secure networks are the largest class of networks which achieve the totally self-checking goal [58]. Totally self-checking networks are a subset of strongly fault-secure networks. The fault secure property is a necessary (but not sufficient) condition for a circuit to be strongly fault-secure. If a circuit is strongly fault-secure*, then each failure that occurs must either be detectable or transform the circuit into a new circuit which is also fault-secure until a detectable failure occurs. Totally self-checking and strongly fault-secure circuits are generically referred to as totally self-checking. A distinction between the two will only be made when it is relevant to the discussion.

Typically*, the totally self-checking properties are only considered for combinational circuits. This restriction is made to insure that the testability assumption is met. If the circuit is combinational*, then by applying the input code space to the circuit, all stuck-at faults which

are detectable will be detected. If the circuit is sequential, then a specific sequence of input codewords must be applied in order to assure the detection of all detectable faults. Therefore, for a combinational circuit, it is only necessary during any time period, τ, to apply at most the entire input code space to the circuit to satisfy the testability assumption. Since the inputs to a circuit are in general unknown, it is very difficult to insure that the testability assumption is satisfied during normal operation. The most obvious solution is to use periodic off-line testing to test the circuit completely. If off-line testing is used, then sequential circuits can be tested by performing the tests in the proper sequence to test all faults.* We have worded our definitions of the self-testing property so that it may apply to sequential as well as combinational logic. If the self-testing property is only specified for combinational logic, then any fault which causes sequential behavior, automatically prevents the circuit from satisfying the self-testing property. For this reason, the self-testing property is defined for both sequential and combinational logic even though the circuits we consider are combinational. If the combinational circuit has some of its outputs fed back as inputs, it may be analyzed as a combinational circuit for the purpose of determining whether it satisfies the self-checking property.

Figure 4.3 shows a typical totally self-checking module. The module is made up of a totally self-checking circuit which performs the

*Generating tests for sequential circuits is significantly more difficult than for combinational circuits.

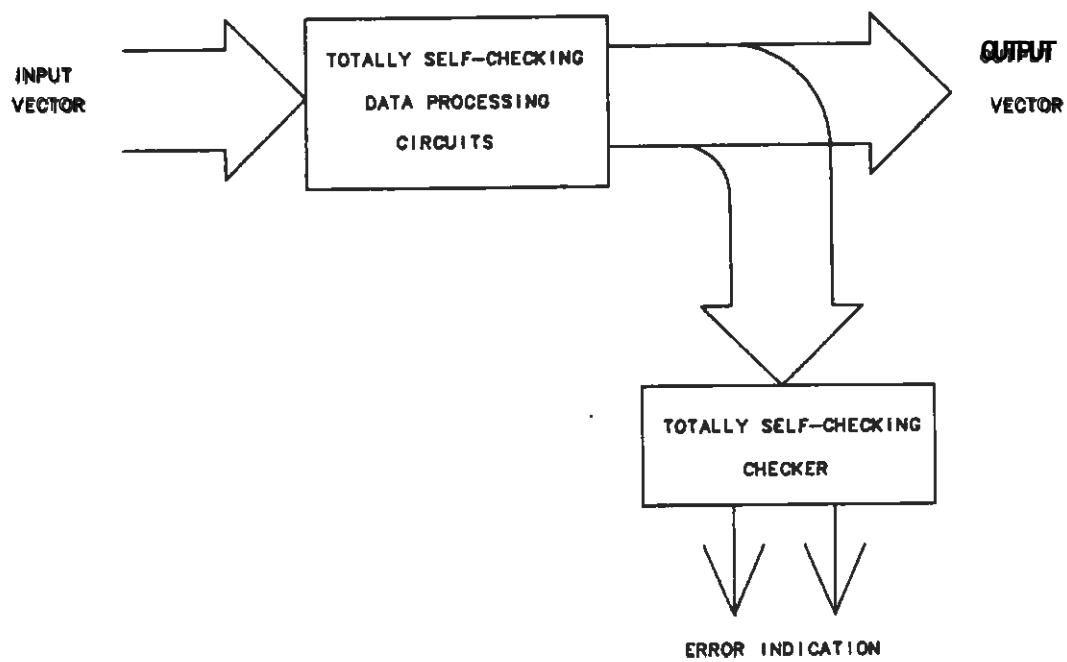


Figure 4.3. Totally Self-Checking Module.

desired data processing. The inputs and outputs must be encoded in an appropriate codes. The outputs from the circuit are examined by a totally self-checking checker. A totally self-checking checker must itself be both totally self-checking and code disjoint. The code disjoint property is required since the whole purpose of the checker is to indicate when it receives a non-codeword input from the data processing circuit. The checker does this by producing a non-codeword on the error indication lines. The code disjoint property assures that if a non-codeword is produced by the processing circuit, then the checker indicates the fact by producing a non-codeword. The checker must be totally self-checking to insure that any failure in the checker is detected before it can cause the checker to miss detecting a non-codeword output from the processing circuit. The checker is therefore able to detect faults in the data processing circuit as well as in itself. The error indication lines are usually encoded using a 1-out-of-2 code. A minimum of two lines are required for the error indication. This requirement prevents the failure of one checker output line from causing the error indication to appear permanently good.

In many cases, it is advantageous to build a totally self-checking system by connecting together several smaller totally self-checking circuits. If all circuits have their output checked by totally self-checking checkers, there are no additional restrictions which are necessary. If it is desired to connect two circuits together without checking the output from the first circuit, then the second circuit must be code disjoint. The code disjoint property assures that if a non-codeword output is produced by the first circuit, then the second

circuit also produces a non-codeword output. The non-codeword from the second totally self-checking circuit is detected by the checker and thus the fault in the first circuit is detected.

The definitions given above, are the traditional definitions for totally self-checking systems. These definitions are quite adequate when traditional fault models are used. When failures cause indeterminate values to occur, the circuit behavior is no longer deterministic. If the output of a circuit with a given fault and input vector could be one of several different output vectors (some of which may be codewords and some of which may be non-codewords), then the self-testing property is not satisfied. Therefore, if the traditional definitions of totally self-checking were retained, it would not be possible to construct totally self-checking systems which include indeterminate value faults in their fault model.

In order to allow the construction of totally self-checking circuits for fault models allowing indeterminate value faults, new definitions are required. We, therefore, propose the following

> Potential Codeword: Let A be a ternary logic vector containing i elements assigned the value u. It is possible to construct 2^i distinct Boolean vectors by replacing all u values with a logic 0 value or a logic 1 value in all possible combinations. Vector A is said to be a potential codeword if exactly one of the 2^i Boolean vectors is a codeword. The Boolean vector which is a codeword is called the corresponding codeword of the potential codeword. (Any vector which is neither a codeword nor a potential codeword is said to be a non-codeword.)

> Self-Testing: A circuit is said to be self-testing if for every fault in the fault model, there is at least one sequence of codeword inputs which produces either a non-codeword output or a potential codeword output.

Fault-Secure: A circuit is said to be fault-secure if for every fault in the fault model, the circuit either produces the correct output, a non-codeword output, or a potential codeword output whose corresponding codeword is the correct output for the entire input code space.

Totally Self-Checking: A circuit is said to be totally self-checking if it is self-testing and fault secure.

Strongly Fault-Secure: A circuit is said to be strongly fault-secure if for all possible sequences of faults from the fault model, as each fault in the sequence occurs, the first fault in the sequence which causes the sequence to be detectable, only produces the correct output, a non-codeword output, or a potential codeword output whose corresponding codeword is the correct output for the entire input code space.

The new definitions explicitly allow for the presence of indeterminate values in vectors. In the remainder of this thesis we use these definitions rather than the traditional definitions.

4.2.2. Checker Strategy

Checkers designed for traditional types of faults are only designed to work properly with legal logic values. In general, digital circuits are unable to react in a reliable manner to indeterminate values (i.e., the circuit response to indeterminate values is nondeterministic). A variety of checker strategies is possible when indeterminate values may occur in output vectors. One strategy is to include additional circuitry in the checker portion of the circuit. The purpose of this additional circuitry (which we refer to as indeterminate detection circuitry), is to detect the occurrence of indeterminate logic values in the output vector. If an error is present in the output vector, then either the original checker and/or the additional indeterminate detection cir-

cuitry detects it. The original checker circuitry detects any erroneous bits in the output vectors which are incorrect, but legal, logic values. The checker circuitry may or may not detect the presence of any indeterminate values. The indeterminate detection circuitry is designed to detect the occurrence of indeterminate values in the output vector. Therefore, the checker together with the indeterminate detection circuitry is able to defect all erroneous output vectors.

There are several problems with this strategy. First of all, indeterminate values may be of several different forms. Circuits which are capable of detecting all types of indeterminate values are inherently quite complex. To make the problem even more difficult, the indeterminate detection circuitry is inherently analog. The circuitry must be capable of measuring voltage amplitudes and determining accurately when high frequency transitions on one line occur in relationship to another line (most likely the clock). To fabricate such circuits with an integrated circuit process that is optimized for digital circuits, further complicates this problem.

The most serious problem with the indeterminate detection circuitry is the need to make it part of a totally self-checking system. The concept of totally self-checking is defined for digital systems where it is meaningful to discuss the encoding of input and output vectors. Therefore, it is doubtful that indeterminate detection circuitry can be built which is totally self-checking.

At the very least, is should be possible to test the indeterminate detection circuitry under normal operation. Testing of analog circuitry

is considerably more complicated and inherently different from testing digital circuitry [59]. It is altogether unclear how to go about testing analog circuits as complicated as the indeterminate detection circuitry during normal system operation of a digital circuit. Unless a scheme can be developed to test the indeterminate detection circuitry, the overall system reliability is seriously compromised since it is now possible for a series of failures to lead to an undetected error. Therefore, cost and reliability concerns make the strategy of directly detecting indeterminate values unattractive.

An alternative that eliminates the cost objection is possible if all lines which are to be checked by the checker come directly from the output of clocked flip-flops. Recall that the output of a flip-flop is either a a legal (but possibly incorrect) logic value from the flip-flop or the flip-flop is in a metastable state. Therefore, the problem of detecting indeterminate values has been reduced to the problem of detecting metastable operation. As discussed in Section 3.4.1, circuits capable of detecting metastable states do exist. A circuit given by Stucki and Cox [47] is shown in Figure 4.4. This circuit is an exclusive NOR gate and is intended for implementation using MOSFETs. The true and complemented outputs from the flip-flop are the inputs to this circuit. The MOSFET exclusive NOR gate is being used as an analog comparator to compare the voltage difference between the flip-flop's Q and $\overline{Q}$ outputs. When a metastable condition occurs in the flip-flop, the true and complemented outputs are at approximately the same voltage. An examination of Figure 4.4 shows that any time $|V_Q - V_{\overline{Q}}| \leq V_{th}$ (V_{th} of the two enhancement mode transistors), the output is high. An

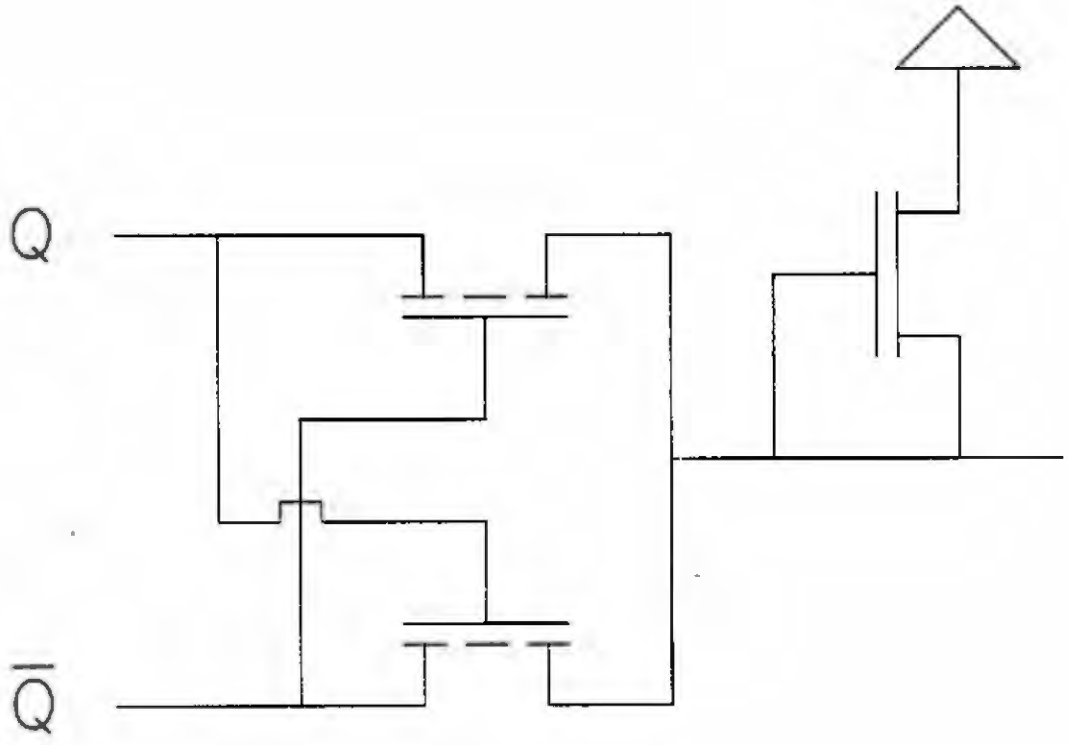


Figure 4.4. Metastable Detection Circuit.

indeterminate detection circuit can be built using the exclusive NOR circuit and a circuit which samples the exclusive NOR's output at the appropriate time in relation to the system's clock.

This circuit still suffers from the same reliability issues which were raised earlier for the analog indeterminate detection circuitry. There is simply no way to test the operation of the exclusive NOR circuits during normal system operation. An additional problem with this circuit is that it only detects indeterminate values as long as the flip-flop is operating properly. A failure in the flip-flop could also render the exclusive NOR circuit unable to detect indeterminate values reliably.

An alternative checker strategy is to make no attempt to detect indeterminate values. Instead, the assumption is made that all indeterminate values will eventually become legal logic values as they propagate through the system. Errors are detected when they finally manifest themselves as incorrect, but legal, logic values. It is also possible for all indeterminate values all to become correct and legal logic values. In this case, no error is indicated and the system has produced the correct output.

The success of this strategy is dependent on the assumption that indeterminate values eventually become legal logic values. For systems which are constructed with blocks of combinational circuitry sandwiched between clocked flip-flops, this is a very reasonable assumption. Our analysis in Section 3.4 showed that under normal circumstances, the probability that the output of a flip-flop is other than a legal logic

value is low. If an indeterminate value from one block of combinational logic is presented to a clocked flip-flop, the probability of the indeterminate value propagating into the next block is therefore low. The probability that it propagates through another clocked flip-flop into a following combinational logic block is even lower.

If the failure occurs in the flip-flop itself, an indeterminate value may be produced but subsequent flip-flops should eventually prevent continued propagation of the indeterminate value throughout the system. The major shortcoming of this assumption is the possibility of indeterminate values being generated in the proximity of the system outputs. If these system outputs are connected to another system which is designed to be tolerant of indeterminate valde inputs, then the other system is able to respond in some appropriate manner to the indeterminate values. Otherwise, a serious failure can occur. In general, the only solution to this problem is to make the other systems fault-tolerant with respect to indeterminate values since a failure in the lines which connects the two systems may also result in indeterminate values.

The strategy we use is the second one (i.e., no attempt is made to detect indeterminate logic values). This strategy does not require costly analog detection circuitry. It also does not result in an untestable design. The checker circuitry required for this strategy is entirely digital.

It should be pointed out that both strategies suffer from a testing problem. The definition of self-testing requires that a faulty circuit

must produce either a non-codeword or a potential codeword output for some sequence of codeword inputs. If the faulty circuit produces a non-codeword output, then the fault is detected. If however, the fault produces a potential codeword output, then the fault may or may not be detected. There are three cases that must be considered. One case occurs if the potential codeword's u values are all interpreted as legal Boolean values which also happen to be correct. In this case, neither strategy would detect the fault. Another case occurs if at least one of the potential codeword's u values is an indeterminate value. All u values which are not indeterminate values must be legal and correct logic values. In this case, the strategy which relies on indeterminate detection circuitry detects the fault but the strategy we use may not detect the fault. Finally, there is the case where at least one of the potential codeword's u values is an incorrect Boolean value. In this case, both strategies detect the fault.

Therefore, testing is a serious concern. Regardless of which strategy is used, there is no assurance that a given sequence detects a fault that produces indeterminate values. One of the assumptions that was made earlier was that there is some minimum time interval, τ, during which the circuit is completely tested. In order to prevent the buildup of latent faults in this case, the time interval, t, may have to be reduced considerably.

In many ways, the testing problem for indeterminate failures is quite similar to the testing problem for intermittent failures. In fact, most indeterminate faults can alternatively be considered to be

intermittent. Depending on how the circuit responds to an indeterminate fault, an error may or may not be produced when one or more outputs are sensitized to the fault. Therefore, the error produced by an indeterminate fault may certainly be viewed as being intermittent. Techniques for the detection of intermittent failures are discussed in [60, 61]. These techniques are intended for off-line testing. Nevertheless, for those situations where off-line testing is used to help satisfy the testability assumption, these techniques could be used to increase the off-line testing effectiveness and/or reduce the off-line test length.

4.3. CED and a Simplified Indeterminate Fault Model

The properties of indeterminate faults have been established. In addition, we have established the conditions which we require of our systems in order for them to implement concurrent error detection. We are now ready to propose a fault model that incorporates indeterminate-type faults.

4.3.1. Fault Model Assumptions

The simplified indeterminate fault model assumes that any physical failure causes a single node in the circuit to become a ternary u value. This fault model excludes some (but not all) bridging-type failures. Our analysis of Chapter 3 shows that, in general, each line which is shorted to another line may have an indeterminate value. Thus, if we wish to model the most general case, then each line which is shorted to another line has a ternary u value on it. In a few cases, it may be possible to model two lines shorted together with only a single ternary

u value. Figure 4.5 shows examples of two different bridging faults. Fault 1 is disallowed by the simplified indeterminate fault model since lines X and Y must both be considered to have u values on them. Fault 2 is allowed by the simplified indeterminate fault model since indeterminate values on both inputs X and Y are indistinguishable from an indeterminate value at the output of the gate. Therefore, fault 2 may be modeled as a u value on line A. This fault model does not consider the effect of failures on certain global signals such as ground, power, and clocks. Such failures may affect the entire circuit or very large sections of it. If protection must be provided against failure of these global lines, then the circuit must be designed so that a global line failure results in a non-codeword. This type of design usually requires at least a redundant copy of each such global signal.

From Theorem 1, stuck-at fault (or any other type of failure which causes a single line to become a legal logic value) propagation automatically is considered by this fault model. It should be pointed out that using the ternary u value for legal logic values may result in misleading results if there are hazards in the circuit. In the presence of hazards, the ternary model may predict that an output or outputs are sensitized to a value of u at a node. If the value of u is a legal logic value and the Boolean difference with respect to the node is zero, then the path is not sensitized. Therefore, there are cases where an indeterminate value propagate even though a legal Boolean value does not propagate.

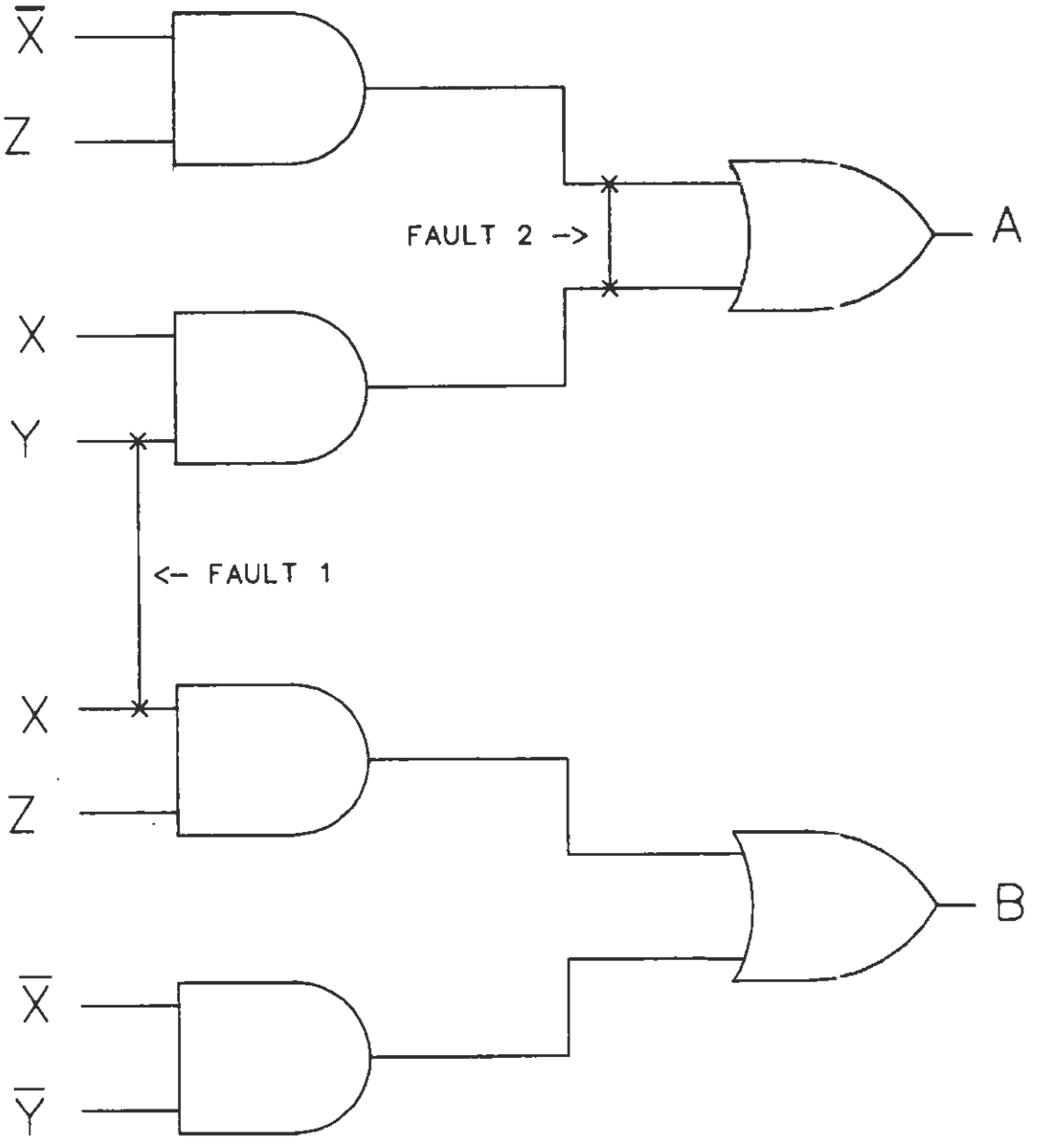


Figure 4.5. Two Types of Bridging Faults.

When determining whether a circuit satisfies the fault-secure property, we are interested in which faults are sensitized to the outputs for a given input. If a fault is propagated to an output, the code used in the circuit must be able to detect this fault. If a fault always results in a legal logic value, then the ternary model may predict that the output is sensitized, even though the output is not path-sensitized to the fault. Legal logic values may only propagate to an output when the output is path sensitized to the fault. Therefore, the ternary model is pessimistic for legal logic values when determining whether or not a circuit satisfies the fault-secure property.

On the other hand, when determining whether or not a circuit satisfies the self-testing property, it is desirable to propagate as many faults as possible to the outputs as this is the only way in which a fault may be detected. Consequently, the ternary model is optimistic for legal logic values when determining whether or not a circuit satisfies the self-testing property. For this reason, the indeterminate fault model is a poor choice if failures cause only legal logic values. On the other hand, in situations where both indeterminate values and legal logic values are caused by failures, then the indeterminate fault model is a good choice, since it handles both indeterminate and legal logic values. In situations where failures may cause both indeterminate values and legal logic values, the simplified indeterminate fault model should be used to determine whether or not a circuit satisfies the fault-secure property. However, the single stuck-at fault model should be used to determine whether or not a circuit satisfies the self-testing property.

In order to study the implications of using the simplified indeterminate fault model, we assume that all indeterminate values become legal logic values by the time they reach the circuit's output. This assumption is required to insure that a failure in a previous circuit does not result in several indeterminate values appearing on the inputs of a circuit. By placing a checker at the output of every circuit, this assumption implies that any error in the first circuit is detected before the error reaches the second circuit.

When using the simplified indeterminate fault model, it is not necessary to consider all possible faults. Many possible faults do not need to be considered, since consideration of certain faults, takes into account all the effects of other faults.

Theorem 2: For any switching function, an implementation exists in which all failures allowed by the simplified indeterminate fault model may be modeled as a single ternary u value on a single input line or output line.

Proof: Figure 4.6 demonstrates a manner in which any switching function may be implemented. Each output is generated by its own independent block of logic. Clearly, any modeled failure which affects an input line may be modeled by a ternary u value on the failed input line. Any modeled failure which occurs in one of the blocks of logic may be modeled conservatively as a ternary u value on the output line from the failed logic block. Therefore, for this implementation, all failures allowed by the simplified indeterminate fault model may be modeled by a single ternary u value on an input or output line.

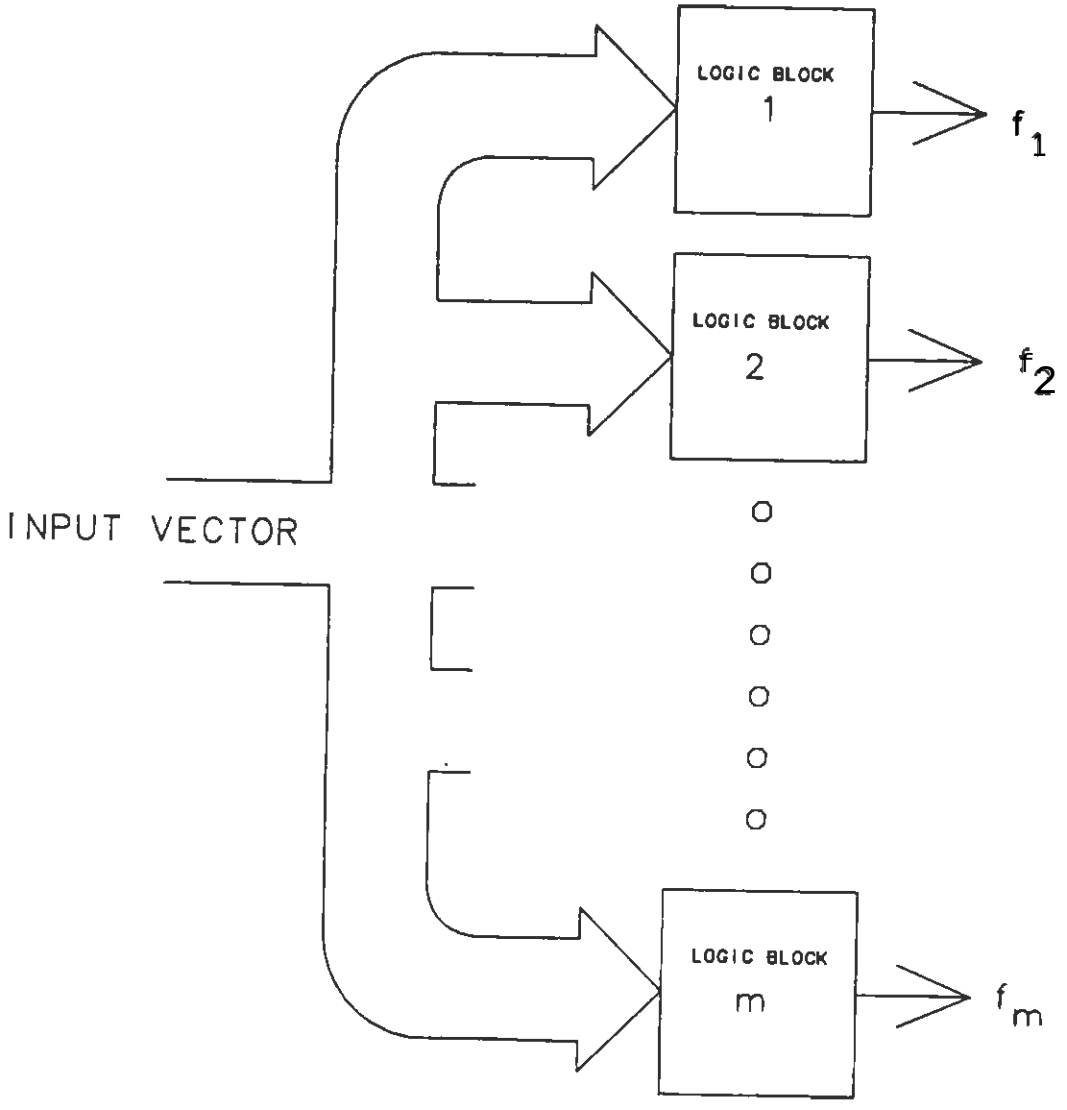


Figure 4.6. Possible Circuit Implementation.

As long as circuits are implemented in the form of Figure 4.6, the number of faults which must be considered is significantly reduced. Unfortunately, the implementation of Figure 4.6 is seldom the most efficient implementation of a switching function. By using shared logic to produce two or more outputs, the total amount of logic may be significantly reduced. Figure 4.7 shows a 4-input, 2-output circuit. Clearly, this implementation is not of the same form as shown in Figure 4.6. In Figure 4.7, the product term labeled m is used in generating both outputs. If it were desired to implement this function in the form shown in Figure 4.6, then an additional 3-input AND gate would be required. If $W = X = Y = Z = 0$, then both outputs are 0. Furthermore, neither output is sensitized to any of the 4 inputs. Both outputs are, however, sensitized to the output of the gate labeled m. Therefore, a failure resulting in a ternary u value at the output of gate m cannot be modeled as any single input or output having a ternary u value. In general, a fault which is sensitized to two or more outputs and at the same time is not sensitized to any inputs, cannot be modeled as a single input or output having a ternary u value.

4.3.2. Separable Codes

Codes used in totally self-checking circuits can be divided into two broad classes: separable codes and non-separable codes.

A separable code consists of two parts: the data vector and the check vector. The data portion of the codeword merely consists of the unencoded data. The check vector consists of redundant information. Therefore, decoding a separable code simply requires stripping the check

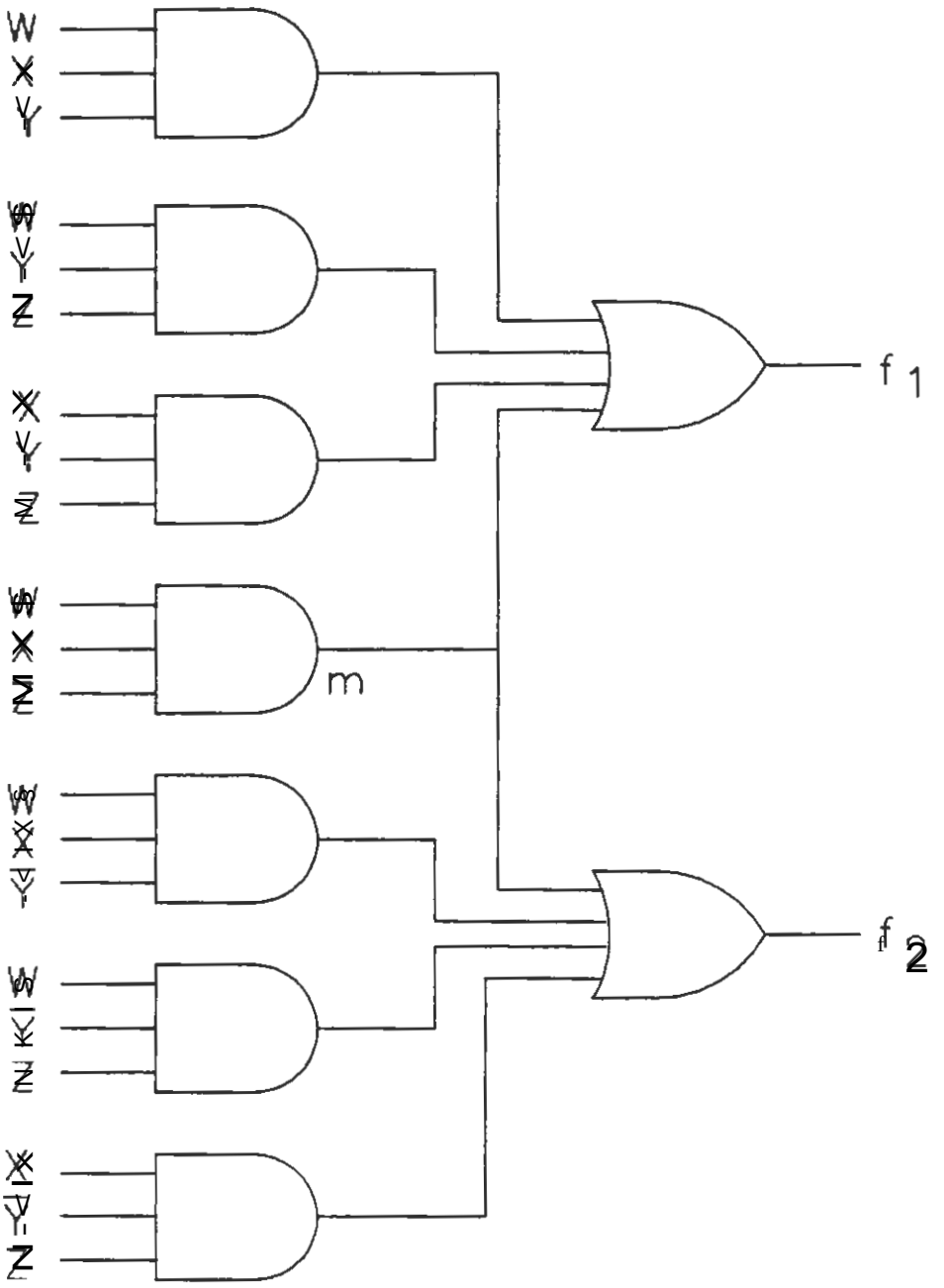


Figure 4.7. Circuit Implementation with Shared Logic.

vector from the codeword. Common examples of separable codes include parity codes and two-rail codes. Any code which does not satisfy the definition of a separable code is considered to be non-separable. A common non-separable code is the k-out-of-n code.

We restrict our attention exclusively to separable codes. Separable codes are usually much easier to implement than non-separable codes. With a separable code, the data portion of the code is no harder to implement than a non-encoded version of the same function since the data portion is not altered by the encoding into a separable code. The simplicity of the encoding can often be a significant advantage. The designer is usually able to use a knowledge of the function and its properties to determine an efficient implementation. By encoding into a non-separable code, a function typically becomes more complicated in a manner that often obscures the original function. More than likely, this new function will be harder to implement than the original unencoded function. A separable code may be implemented with two independent relatively small circuits while the implementation of a non-separable code requires one larger circuit. In terms of switching speed, two smaller circuits in parallel as in a separable code are usually preferable to one larger circuit as in a non-separable code. This is especially true in structured elements such as PLAs. Therefore, a separable implementation may be faster than a non-separable implementation. Finally, the analysis of separable codes is usually easier than for non-separable codes. Because of the advantages that separable codes offer over non-separable codes, non-separable codes are not considered further.

There are a variety of possible separable codes which may be useful in totally self-checking circuits. In addition, there are a variety of implementations for each separable code. One important class of separable code implementations is functional duplication. A circuit is said to employ functional duplication if

(1) The circuit uses a separable code.

(2) The circuit layout is such that the data and check portions of the circuit are physically disjoint.

(3) There is a bijective (one-to-one and onto) mapping between the data vector and check vector of the entire output code space.

Note that condition (3) does not require the checker circuit to be an exact copy of the data circuit for functional duplication. Since the circuit uses a separable code and the data and check portions of the circuit are physically separated, then any failure modeled by the simplified indeterminate fault model affects either the check portion or the data portion, but not both. This fact leads us to the following theorem:

Theorem 3: Any switching function has a functional duplication implementation. Furthermore, any functional duplication implementation of a switching function, satisfies the totally self-checking goal.

Proof: First we prove the existence part of the theorem. Consider any arbitrary switching function f. It is possible to construct a circuit C that implements the function f. Consider a circuit C' which is formed by two distinct and physically dis-

joint copies of C. One of the copies of C represents the data portion while the other copy represents the check portion of C'. Clearly C' employs a separable code where there is a bijective mapping between the check and data vectors of the code. Furthermore, we have specified that C' is constructed with disjoint data and check circuitry. Therefore, circuit C' is a functional duplication implementation of switching function f.

From the definition of functional duplication, any modeled failure affects at most one portion of a functionally duplicated circuit's output. If a failure occurs in the data portion, then the check vector is always correct while the data vector may or may not be correct. Likewise, if a failure occurs in the check portion, the data vector is correct while the check vector may or may not be correct. The bijective property of functional duplication assures that any failure that causes either the data vector or the check vector (but not both) to be incorrect is detectable.

If a failure occurs which is undetectable, then the circuit is transformed into a new circuit which still employs functional duplication and still implements the same function as the original circuit. The next modeled failure which occurs is either detectable, in which case it is detected when the first non-codeword output is produced, or it is undetectable and the circuit is once again transformed into a new functional duplication circuit which continues to implement the original switching

function. This process is continued until a detectable failure occurs and a non-codeword output results. Therefore, the first incorrect output is a non-codeword and the circuit thus satisfies the totally self-checking goal.

Corollary 3: If a functional duplication implementation contains no redundant logic (with respect to the input code space), then it is totally self-checking with respect to the simplified indeterminate fault model.

Proof: From Theorem 3, we know that the first incorrect output from a functional duplication circuit must be a non-codeword. Therefore, the circuit satisfies the fault-secure property. Since the circuit contains no redundant logic, any modeled failure which occurs must be detectable. Since the circuit satisfies both the fault-secure and self-testing property, then it must also satisfy the totally self-checking property. Therefore, the circuit is totally self-checking with respect to the simplified fault model.

From Theorem 3 and Corollary 3, we know that a totally self-checking implementation exists for any switching function. Unfortunately, functional duplication requires roughly a 100 percent increase in both area and power dissipation. When the additional circuitry for the checkers is included, this increase is significantly above 100 percent. Therefore, a question, which we now examine, is: under what circumstances do totally self-checking implementations exist that are more economical than functional duplication?

4.3.3. Finding Economical Totally Self-Checking Implementations

To determine which of several implementations of a given function is most economical, it is usually necessary to layout each implementation. The area of the circuit may then be determined and a circuit simulator such as SPICE may be used to estimate the circuit's power consumption. The simulator may also be used to determine the speed of the circuit. Although this technique assures that we always use the most economical of the several implementations, it is not in general practical. For any given switching function, there is a large number of implementations which must be considered. The average circuit designer's productivity in industry may be as low as 5 - 10 transistors per day [62]. Even when structured designs and extensive design automation software is used, the designer productivity may still be less than 40 transistors per day [63]. For large integrated circuits containing hundreds-of-thousands of transistors, implementing a large number of alternative designs is quite clearly not practical. Instead, we consider the cost of an implementation to be completely determined by the number of bits the circuit must process. Under this assumption, the cost of an implementation depends solely on the code it uses. By making this assumption, we are shifting the problem from finding the most economical implementation to finding the most economical code. Since there are many possible implementations of a given code, once a code has been selected, a good implementation of the code must still be determined. It is usually much easier to determine which implementation of one code is more economical than to determine which implementation among different possible codes is more economical. It does not always follow

that if code A is more economical than code B, then a "good" implementation of code A is more economical than a "good" implementation of code B. Nevertheless, it is reasonable to expect this usually to be true. We therefore, restrict our attention to finding economical codes.

Since all codes that we consider are separable, all codes for a given function have the same number of data bits. Therefore, when considering the relative economy of several codes, only the number of check bits for each code needs to be considered. We define the cost of a code to be the number of check bits in the code. The code with the lowest cost is considered to be the most economical.

Theorem 3 guarantees that a functional duplication implementation exists for any desired switching function. If N is the number of distinct output codewords of a switching function, then the most economical code which may be used in a functional duplication implementation has a cost C^* given by

$$C^* = \lceil \log_2(N) \rceil$$

Any code with a cost less than C^* cannot satisfy the bijection requirement for functional duplication. Therefore, we are interested in finding codes for a given function which have a cost less than C^* but still have an implementation that satisfies the totally self-checking goal.

In searching for codes more economical than functional duplication, we concentrate on the fault-secure property. There are several reasons for this. The fault-secure property is a necessary condition for both the totally self-checking property and the strongly fault-secure property. Therefore, a circuit must be fault-secure if it is to satisfy

the totally self-checking goal. For circuits without hazards, it is possible to determine whether a code for a given function is fault-secure without knowing the details of the implementation except that it is of the form of Figure 4.6. On the other hand, whether a function satisfies the self-checking property is strongly dependent on the implementation. The procedure we use to search for codes with a cost less than C^*, but which still satisfies the fault-secure property, only depends on the switching function that we desire to encode. Since a hazard-free implementation of the form shown in Figure 4.6 always exists for any code, this procedure may be used to find whether a fault-secure implementation exists with a cost less than C^*.

The search procedure we propose is now demonstrated by an example. Figure 4.8 shows the truth table for a full adder circuit with inputs X, Y, and Z, and outputs C and S. The question we wish to answer is whether or not a fault-secure implementation exists with a cost less than C^*. For this circuit, there must be 4 output codewords. Therefore, $C^* = 2$. Each codeword consists of a data vector and a check vector. Codewords are formed by combining each code vector with certain data vectors. Not all combinations are allowed. The allowed combinations are the codewords while the disallowed combinations are non-codewords.

In order for the check vector of the code to require fewer bits than C^*, there must be some check vectors which may be combined with more than one data vector to form codewords. We make the assumption that for all codewords, a given data vector only has a single possible

X	Y	Z	CS
0	0	0	00
0	0	1	01
0	1	0	01
0	1	1	10
1	0	0	01
1	0	1	10
1	1	0	10
1	1	1	11

Figure 4.8 Full Adder Example.

check vector. In other words, the data vector of a codeword implies the check vector of a codeword. Violating this assumption never reduces the cost of a code since by violating the assumption, we are increasing the number of distinct check vectors which the code must include. The main reason for this assumption is that it means we only need to consider failures in the data portion of the function.

In order for the fault-secure property to be satisfied, a failure must either be undetectable (in which case the correct output is always produced) or detectable for some input (in which case the circuit must produce a non-codeword output). When a failure occurs in the data portion of the circuit, then the code must be designed so that it is never possible for the data vector of one codeword to be transformed into the data vector of another codeword. If this transformation is allowed to happen, then the failure has caused an incorrect codeword output to occur and the circuit is thus not fault-secure. If a failure occurs in the check portion of the circuit, the check vector is either correct (in which case the output codeword is also correct), or the check vector is incorrect. We have assumed that the data vector implies the check vector. Therefore, if the check vector is incorrect, then the output from the circuit is a non-codeword. Consequently, failures in the check portion of the circuit automatically satisfy the fault-secure property.

Returning to our example of Figure 4.8, we must find a code which requires fewer than 2 check bits. The code must be selected such that if a fault in the data portion of a circuit can cause the data vector of one codeword to change to the data vector of another codeword, then the

two codewords must have different check vectors. The first step is to determine the effect of all faults on the data portion of the circuit's outputs. By Theorem 2, we only need to consider failures on the data portion of the circuit's inputs and outputs. Let us first consider the effect of a fault on the input.

Figure 4.9 shows the result for the full adder example of Figure 4.8. The first 4 columns of the table of Figure 4.9 repeats the truth table from Figure 4.8. The next 3 columns of the table show the effect of failures on lines X, Y, and Z, respectively. Each row of the table represents one of the possible input conditions. If a data bit, C or S, retains its original value when a given input is changed, that data bit retains its original value in the column under the given input. If a data bit changes its value when a given input is changed, the data bit takes on the value of u in the column under the given input.

In other words, we are interested in whether or not the data bit is sensitized to a fault on the input. If a change in the input causes a change in the data bit output, then the data bit is sensitized to the input. If the circuit contains a static hazard, then a data bit output may be sensitized to an input even though a change in an input does not cause the data bit to change. An implementation which is free of static hazards exists for any function although it may require redundant logic. Therefore, we only need to consider whether a input change leads to an output change to determine sensitization. We may determine sensitization by inspection from the truth table as we have done here or more formally, by the Boolean difference method. If the Boolean difference

FUNCTION				OUTPUT FOR INDETERMINATE FAULT IN			OUTPUT MAP		
X	Y	Z	CS	X CS	Y CS	Z CS			
0	0	0	00	0u	0u	0u	0	1	1,2
0	0	1	01	uu	uu	0u	1	0,2,3	0,3
0	1	0	01	uu	0u	uu	1	0,2,3	0,3
0	1	1	10	1u	uu	uu	2	0,1,3	0,3
1	0	0	01	0u	uu	uu	1	0,2,3	0,3
1	0	1	10	uu	1u	uu	2	0,1,3	0,3
1	1	0	10	uu	uu	1u	2	0,1,3	0,3
1	1	1	11	1u	1u	1u	3	2	1,2

Correct Output	Possible Faulty Outputs
0	1,2
1	0,2,3
2	0,1,3
3	1,2

Figure 4.9 Fault Behavior of Full Adder*.

is 1, then the output data bit is sensitized to the input and the output data bit is set to u. If the Boolean difference is 0, then the data bit retains its original value. It must be emphasized that using the Boolean difference method (or equivalently determining from the truth table whether or not an input change causes an output change)# is only valid because of the assumption that the implementation has no static hazards.

In Figure 4.9, on the first row, all columns have a value of 0u. This implies that if the input of the circuit is X = Y = Z = 0, then a single indeterminate fault on any of the three inputs results in output bit C remaining 0. Output bit S has a value of u meaning it may take on a value of either 0 or 1. Likewise, the second row of the table corresponds to X = Y = 0, Z = 1. In this case, a single indeterminate fault on either input X or input Y results in output bits C and S both having a value of u. A single indeterminate fault on input Z results in output bit C having a value of 0 and output S having a value of u. A simple indeterminate fault on either input X or input Y can result in output bits C and S being either 0 or 1. Other rows are similarly constructed.

The three columns under the heading "OUTPUT MAP," represent the correct output, all outputs that may be produced if a single indeterminate fault occurs on any one of the inputs, and all outputs that may occur due to a single indeterminate fault on any one of the outputs. Note that the output vector CS is treated as an unsigned two-bit binary

number. To determine the possible faulty outputs, all u's in the output vector are replaced by all possible combinations of 0's and 1's.

We have now considered the effect of a ternary u value on any of the circuit inputs. It is still necessary to consider the effect of a ternary u value on any of the circuit outputs. Output faults may be considered by taking each of the possible output vectors one at a time and complementing each of the output bits. For example, if the correct output is 0, an output fault may result in an output vector of 1 or 2. If the correct output vector is 1, then an output fault may result in an output vector of 0 or 3. If the correct output vector is 2, then an output fault may also result in an output vector of 0 or 3. Finally, if the correct output vector is 3, then an output fault may result in an output vector of 1 or 2.

The list at the bottom of Figure 4.9, gives a summary of all errors due to any single indeterminate fault on an input or output line. Below the table, the fault behavior is summarized. Each of the four possible correct outputs are listed along with the faulty outputs that they may be changed into. From the summary, we see that any time the correct output vector is 0, then with a fault on one of the inputs or outputs, we may get an output vector of either 0, 1, or 2. When the correct output vector is 1 or 2, we may get any output vector. When the correct output vector is 3, we may get an output vector of 1, 2, or 3.

It is now necessary to assign a check vector to each of the possible data vectors. In order to keep the cost of the code as low as possible, it is desirable to assign as many of the data vectors to a single

check vector as possible. On the other hand, two data vectors cannot be assigned to the same check vector if a fault may transform one data vector into the other data vector.

A set of data vectors is said to be compatible if no member of the set may be transformed by a fault into another member of the set. The problem is to determine the fewest sets of compatible data vectors such that each data vector occurs in exactly one set. Each set of compatible data vectors is assigned a unique check vector.

In Figure 4.10, a merger diagram is drawn as a graphical aid to determine the fewest sets of compatible data vectors. The merger diagram has a node for each data output vector. An arc is drawn between each pair of compatible data output vectors. A set of nodes is compatible if and only if every node in the set is connected by an arc to every other node in the set. From the merger diagram, we see that at least three sets of data vectors are required. Data vectors 0 and 3 form a compatible set since a correct data vector of 0 can never be changed by a fault to 3 and a correct data vector 3 can never be changed by a fault to 0. On the other hand, data vectors 1 and 2 must each be in a set by themselves since a fault may change these data vectors to any of the other possible data vectors. Since at least three sets of data vectors are required, there must be three distinct check vectors. Therefore, no code exists with a cost less than 2. Since C^* for this function is 2, no code exists which is more economical than the most economical functional duplication code.

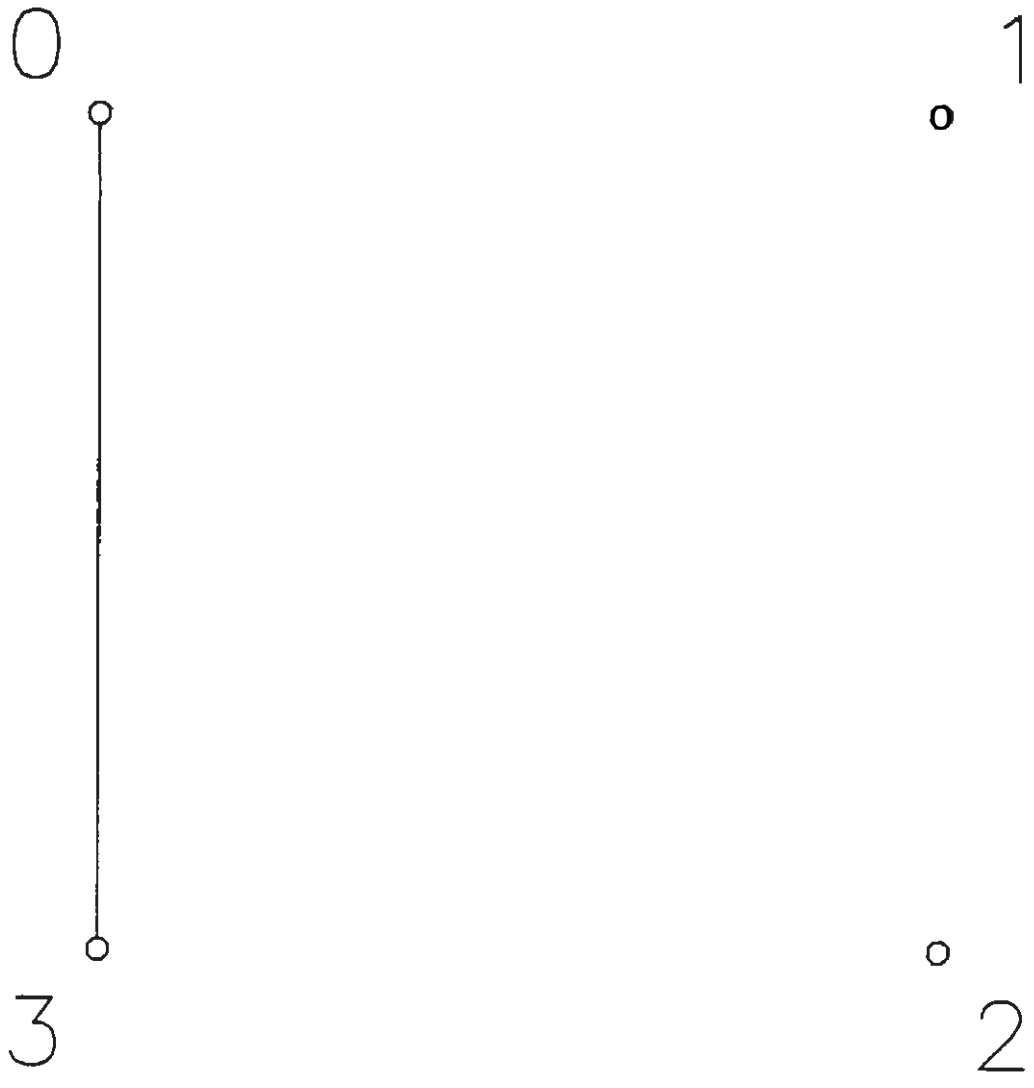


Figure 4.10. Merger Diagram for Full Adder Example.

The procedure for finding codes that are both more economical than functional duplication and have implementations that are fault-secure with respect to the simplified indeterminate fault model may now be summarized as follows:

(1) Construct a truth table for the desired switching function. This function is implemented by the data portion of the circuit.

(2) For each possible data input vector, determine the possible incorrect data output vectors that may result from a fault on a single input.

(3) Summarize the results from step 2 to obtain a list of each correct data output vector and the incorrect data output vectors that may result from a fault on an input.

(4) Update the list from step 3 to include the effects of faults on output lines.

(5) Determine the minimum number of sets of compatible data output vectors required so that each output vector is included in exactly one set.

(6) The minimum number of check bits is the smallest integer which is greater than or equal to the $\log_2$ of the minimum number of compatible sets.

When the minimum number of sets of compatible data vectors has been found, then each set of data vectors must be assigned a unique check vector. This assignment is completely arbitrary as it has no effect on either the fault secure property or the cost of the code. Therefore, the assignment may be done so as to minimize the cost of the implementation of the disjoint check bit generation logic.

In the example of Figure 4.8, it is possible for all data input and output vectors to occur. In other cases, it is possible that one or

more of the input or output vectors cannot occur in normal operation. In such a case, this procedure may still be used. Any data input vectors which are not used should be left out of the truth table. Any data output vectors which do not occur during normal operation may also be ignored since the checker may be designed to recognize any unused data output vector.

This procedure may be used to detect whether or not a code exists which has a cost lower than the most economical functional duplication code. Furthermore, if such a code is found by the procedure to exist, then there is always a fault-secure implementation of the code. Since the fault-secure property is necessary for all circuits that meet the totally self-checking goal, then if a function is found to have no code more economical than functional duplication for meeting the fault-secure property, the function also does not have a more economical implementation which satisfies the totally self-checking goal.

We have assumed that the implementation has no static hazards. In many cases, this requires the addition of redundant logic. This logic has important implications if the desire is for the implementation to satisfy the totally self-checking goal. If redundant circuitry is added, then the circuit cannot satisfy the totally self-checking property. However, the circuit might not satisfy the strongly fault-secure property. For such fault-secure circuits which are not strongly fault-secure, fault detection cannot be guaranteed for some fault sequences.

The procedure we have proposed does not take into consideration any static hazards which may exist in the implementation when examining

whether a fault may propagate from an input to an output. In some cases, a static hazard does not destroy the fault-secure property. Each static hazard in the circuit allows a fault to propagate from an input to an output. Such a static hazard causes a correct output vector to be transformed to another incorrect output vector. In some cases, this transformation occurs for some other input vector, regardless of whether the static hazard exists. In this case, the static hazard does not affect the fault-secure property of the implementation. In other cases, the static hazard causes the transformation from a correct output vector into an incorrect output vector that does not otherwise occur. If this transformation causes one of the sets of compatible output vectors to become incompatible, then the static hazard must be removed from the implementation.

Redundant logic is often required to remove a static hazard. Occasionally, the situation arises in which the only way a code may be implemented so that it satisfies the fault-secure property is to add redundant logic to the implementation. In this situation, the implementation is fault-secure, but it is not either totally self-checking or strongly fault-secure. The redundant logic which is added to remove the static hazard is not testable. Therefore, if the redundant logic fails, then the static hazard exists once again, but it is impossible to test for all failures in redundant logic. Since the redundant logic was added to make the circuit fault-secure, then the failure of this redundant logic causes the circuit not to be fault-secure. We now have a situation where a failure has occurred that cannot be detected by testing. In addition, the circuit is no longer fault-secure so that the

next fault that occurs may cause an incorrect codeword output. This is an example where the first incorrect output is a codeword output. Therefore, such a circuit does not satisfy the totally self-checking goal. From this argument, we see that if the desire is to build circuits which have the strongly fault-secure (or totally self-checking) property, then redundant logic should not be used to remove static hazards.

In many instances, it may be desirable to use implementations which are not of the form of Figure 4.6. Often by sharing logic among several outputs, the amount of logic required for an implementation is significantly reduced. In many cases, sharing logic between several outputs results only in faults which can be modeled as a single fault on an input or output. In other cases, the shared logic causes faults which cannot be modeled as a single fault on an input or an output, but nevertheless, no sets of compatible output vectors become incompatible due to the sharing of logic. In both of these cases, the sharing of logic does not affect the fault-secureness of the implementation. In other cases, sharing of logic results in compatible sets of output vectors becoming incompatible. In cases where one or more sets of output vectors become incompatible, the resulting implementation is not fault-secure.

Simulators are usually the most practical method of evaluating the effect of static hazards and sharing logic on an implementation. Ternary simulators are quite straightforward to implement [57]. If the circuit implementation is of the same form as Figure 4.6 (no shared

logic)# then the simulator can be used to determine when a u value on an input causes a u value on an output. The simulator can determine when a u value propagates from input to output of an implementation,, regardless of whether static hazards exist. A ternary simulator may also be used to study the effect on an implementation of sharing logic among its outputs.

The procedure we have outlined in this section can be used to search for codes that are more economical than the most economical functional duplication code. If a functional duplication code is found to be the most economical code# then it may be implemented without any concern about static hazards or the sharing of logic between outputs in the implementation. If another code is found to be more economical,, then it may be implemented in the form of Figure 4.6 and a ternary simulator may be used to check for the presence and effect of static hazards. If an implementation that shares logic among the outputs is desired (i.e.# the implementation is not of the form of Figure 4.6),, then the simulator may also be used to determine the effect of shared logic. For non-functional duplication codes,, any static hazard or sharing of logic which causes sets of compatible output vectors to become incompatible must be removed or else the code must be modified so that the sets of outputs are split into smaller outputs. Obviously removing a static hazard or using separate logic to calculate each output requires extra logic. Modifying a code by splitting sets of compatible outputs,, may also require additional logic. It should be noted that for functional duplication codes,, hazards and shared logic are not a concern. The bijective property requires that each set of compatible output vectors

have only one member. Therefore, for functional duplication, the sets of data vectors always remain compatible.

The full adder example is a case where no code more economical than functional duplication exists. There are other functions, however, where codes more economical than functional duplication do exist. Figure 4.11 shows the truth table and fault behavior of a two-bit, vector AND function. From the list of input and output fault behavior, it is clear that only two sets of compatible output vectors are required, {0,3} and {1,2}. For any such vector bitwise function, regardless of the length of the input vectors, a fault under the simplified indeterminate fault model may only affect at most one output bit. Therefore, any possible erroneous vector will be distance 1 away from the correct output vector. To detect such errors, it is only necessary for every output codeword to be at least distance 2 away from every other output codeword. A one-bit parity code is an excellent choice for such a code. Consequently, for any bitwise vector operation, fault-secure operation with respect to the simplified indeterminate fault model may be provided at a cost of only one check bit.

4.3.4. Check Vector Generation

If the circuit we wish to design accepts unencoded inputs, then the generation of the check vector presents no particular difficulties. The unencoded input vector is fanned-out to both the data and check portions of the circuit. Since the entire input vector is available to the check portion of the circuit, the generation of the output check vector is straightforward. Unfortunately, if there is a fault on an input line

FUNCTION					OUTPUT FOR INDETERMINATE FAULT IN				OUTPUT MAP	
X_1	X_0	Y_1	Y_0	S	X_1 S	X_0 S	Y_1 S	Y_0 S		
0	0	0	0	00	00	00	00	00	0	1,2
0	0	0	1	00	00	0u	00	00	0	1,2
0	0	1	0	00	u0	00	00	00	0	1,2
0	0	1	1	00	u0	0u	00	00	0	1,2
0	1	0	0	00	00	00	00	0u	0	1,2
0	1	0	1	01	01	0u	01	0u	1	0,3
0	1	1	0	00	u0	00	00	0u	0	1,2
0	1	1	1	01	u1	0u	01	0u	1	0,3
1	0	0	0	00	00	00	u0	00	0	2
1	0	0	1	00	00	0u	u0	00	0	1,2
1	0	1	0	10	u0	10	u0	10	2	0,3
1	0	1	1	10	u0	1u	u0	10	2	0,3
1	1	0	0	00	00	00	u0	0u	0	1,2
1	1	0	1	01	01	0u	u1	0u	1	0,3
1	1	1	0	10	u0	10	u0	1u	2	0,3
1	1	1	1	11	u1	1u	u1	1u	3	1,2

Figure 4.11 Vector AND Example.

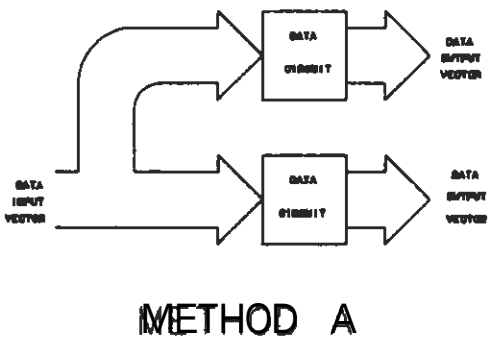
before the input vector is fanned-out to the data and check portions of the circuit*, then the fault may cause an undetected error since the incorrect value is passed to both the data and check portions of the circuit.*

The generation of the check vector becomes more complicated if the circuit is part of a larger totally self-checking system. In this case, the check vector must be generated in such a manner that no modeled failure on an input violates the fault-secure property.

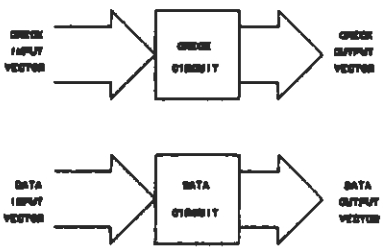
Figure 4.12 demonstrates three possible ways of generating the check vector that are compatible with the philosophy of separable codes. Method A of Figure 4.12 has the advantage of being very simple. In this method, the data input vector is used by both the data and check portions of the circuit. Unfortunately, as we have just shown, this method cannot protect against input line faults. Method B is also very simple. In this method, the data input vector is used to calculate the data output vector and the check input vector is used to calculate the check output vector. Note that the check output vector of the previous function forms the check input vector of this function. The drawback to this method is that there may not be enough information in the check input vector to calculate the check output vector. It should be noted that the bijective property of functional duplication guarantees that method B may always be used with functional duplication. Method C is more complicated than either method A or method B. In method C, the data input vector is used to calculate the data output vector and the

---

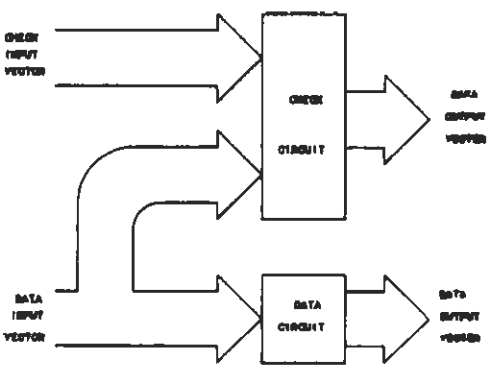
*We are assuming that none of the data bits is redundant.



METHOD A



METHOD B



METHOD C

Figure 4.12. Three Methods of Check Vector Generation.

data input and check input vectors are used to calculate the check output vector. Since the check circuitry must process both the data input and check input vectors, method C generally requires more logic than either method A or method B. For this reason, method B is preferred whenever it is feasible.

One of the advantages of using separable codes is that, in general, single faults only affect either the data vector or the check vector, but not both. If method C is used for generating the check vector then a single fault may affect both the data and check portions of the circuit. Ideally, if method C is used to generate the check vector, then it would be desirable to design the check portion of the circuit so that no single failure on one of the data input bits causes both the data and check portions of the circuit to produce erroneous output vectors. If both the data and check output vectors may be in error, it is very difficult to determine whether the circuit violates the fault-secure property. In some cases, it may be possible to design both the data and check circuits so that even when both data and check vectors are incorrect, the output vector is a non-codeword. In general, this goal is very difficult to achieve since we must now consider the effect of faults in the data input vector on the check output vector. The primary reasons for choosing separable codes is to simplify the analysis and design of the circuit. If faults are allowed to cause errors in both the data output vector and the check output vector, then in order to insure that the circuit is fault-secure, we must consider the data circuit and the check circuit together.

Up to this point, only the function performed by the data circuit needed to be considered. We were able to ignore the details of the check circuit because errors were not allowed to occur in both the data output vector and the check output vector. If simultaneous errors were allowed in both the data output vector and the check output vector, then the functions performed by the data circuit and the check circuit must be considered when finding codes that satisfy the fault-secure property. However, the function that the check circuit performs depends on the code selected. In this case, a code has to be assumed, and then, the data circuit and check circuit together as a unit may be tested to determine if the entire circuit satisfies the fault-secure property. The additional analysis required by this process negates any advantage that separable codes have over non-separable codes in terms of ease of analysis. In addition, the code used and the function performed by the preceeding circuit, which produces the inputs for this circuit, must also be considered when evaluating the fault-secureness of this circuit. This requirement serves to complicate the design process further. For the sake of simplicity, we assume that simultaneous incorrect data and check vectors imply a de jure violation of the fault-secure property. This assumption will be referred to as the disjoint error assumption.

In many cases method B is not applicable. It is important to know whether or not method C is universally applicable so that it may be used when method B cannot be used.

Theorem 4: Method C may be used to provide a fault-secure implementation with respect to the simplified indeterminate fault model of the check portion of the circuit provided that the

Hamming distance between any two data input vectors in a compatible set is at least 3.*

Proof: We prove this theorem by describing a method C implementation that satisfies the theorem. Assume that the check portion of the circuit has no static hazards. As we have already discussed, any switching function has a static hazard free implementation. Recall that added redundancy if any, does not jeopardize the fault-secure property. If method B is sufficient to provide a fault secure implementation, then clearly this theorem is true (i.e., simply use the method B implementation and have the data input vector ignored by the check generation circuitry).

If method B is not sufficient, then there must be at least one instance where the same check input vector is used by the check circuit to calculate two different check output vectors. In this case, the information contained in the data input vector must be used to help calculate the check output vector. Let us call any two such data input vectors D_1 and D_2. Let their corresponding check input vector be C_{i1}, and their check output vectors be C_{o1} and C_{o2}, respectively. Since data input vectors D_1 and D_2 have the same check vector C_{i1}, they must belong to the same set of compatible data vectors. By the distance 3 res-

*Under any circumstances, the Hamming distance between any two compatible data vectors must be at least 2. Otherwise, a fault on an output line could transform one member of a compatible set into another member of the same set. This property violates the definition of a compatible set.

triction of the theorem, D_1 and D_2 must differ in at least three bits. Let S be the set of bit positions in the data input vector which are different in D_1 and D_2. Since the minimum Hamming distance is 3, set S must have at least 3 members.

When a fault exists, the output may be either the correct codeword or a non-codeword. Precisely which non-codeword that is produced is not important. Therefore, as long as an incorrect codeword is not produced, we may design the circuit to behave in any manner we wish when an unused input occurs. Since a single fault on the data input bits in S corresponds to unused input vectors, we are free to assign these in any way we find convenient as long as an incorrect codeword is not produced.

In order to insure that the check generation circuit satisfies the fault-secure property, the check circuit must be designed so that if the correct check output vector is C_{o1}, then for any single bit change in D_1 of bits in set S, the check output vector is still C_{o1}. This requirement follows from the disjoint error assumption. Likewise, if the correct check output vector is C_{o2}, then for any single change in D_2 of a bit in set S, the check output vector must remain C_{o2}. The check function may always be defined in this manner, since all pairs of data input vectors in the same compatibility set have a Hamming distance of at least 3.

Since the check circuit has no static hazards, when C_{i1} and

either D_1 or D_2 is applied to the check circuit*, no check output bit is sensitized to any one of the bits in set S. Similarly given C_{i1} and either D_1 or D_2 as inputs*, no check output can be sensitized to a data input bit which is not in set S, due to the disjoint error assumption. The distance 3 restriction in the theorem statement assures that such a circuit is feasible.

Therefore, any fault on a single data input bit results in the correct check output vector and either the correct or the incorrect data output vector. If the data output vector is correct, then the correct codeword is produced. If an incorrect data output vector is produced, then it is not compatible with the correct check output vector. Therefore, the circuit is fault-secure.

Theorem 4 shows that method C may always be used provided that the minimum Hamming distance in any compatible set is at least 3. In the proof, it is required that the check circuits be designed so that in those cases where there is insufficient information in the check input vector to calculate the data input vector, a single fault on one of the data input bits would not change the check output vector. By making this requirement, we are insuring that a fault on one of the data input bits does not cause the check output vector to be incorrect. If we do not design the check circuit in this manner, then a fault on one of the data input bits may cause both the data and check vectors to be incorrect. This in turn may lead to an incorrect codeword output and thus a violation of the fault-secure property.

In order to satisfy the fault-secure property under the disjoint error assumption, the check circuitry must be designed so that no single fault on one of the data input bits changes the check output vector. Unfortunately, this type of check circuit creates a testability problem if the desire is to implement a circuit which satisfies the totally self-checking goal.

Theorem 5: If a circuit cannot be implemented using method B, then no implementation using method C satisfies the totally self-checking goal with respect to the simplified indeterminate fault model.

Proof: If the checker circuit is implemented so that a single fault on one of the data input bits may change the check output vector, then a single failure on one of the data input lines can result in an incorrect data output and check output vector. By the disjoint error assumption, the circuit is not fault-secure. Therefore, the circuit cannot satisfy the totally self-checking goal.

Consider an implementation using method C where no single bit input fault alters the check output vector. We now prove the the theorem by constructing a sequence of faults on the data input bits for which the implementation violates the self-checking goal. Since the data portion of the circuit may contain redundancy, we consider faults only on irredundant data input bits, i.e., each such fault will affect the data output vector for some data input vector. Let the first fault in the sequence occur on one of the data input bits after they have been

fanned out so that the fault only affects the check circuit. This fault is undetectable. Let the next fault occur occur on another data input bit. If this fault causes the check output vector to change, then let the fault occur before the data input bits are fanned out so that it affects both the data and check portions of the circuit. Otherwise, let the fault occur after the data input bit is fanned out so that it affects only the check portion of the circuit and is therefore undetectable. Continue this process until a fault finally causes an incorrect check output vector. Note that such a fault must eventually be encountered since otherwise all data input bits would be redundant in the checker portion of the circuit and we would have a method B implementation contrary to the theorem hypothesis. We now have a sequence of undetectable faults followed by a data input bit fault that alters the check output vector. This last fault must also alter the data output vector for some choice of input vector. It therefore causes an incorrect data output vector and an incorrect check output vector. From the disjoint error assumption, the circuit is not fault-secure for this sequence of faults. Therefore, the circuit cannot satisfy the totally self-checking goal.

From this discussion, several conclusions can be drawn. Method A must be used if the circuit receives unencoded inputs. Method A, however, does not protect against failures that occur on data inputs. When the circuit receives encoded inputs, method B is the method of choice.

Method B is relatively simple to implement and when feasible, always provides a fault-secure implementation. Unfortunately, in some cases, there is not enough input information in the check input vector to compute the check output vector. In such cases, method B cannot be used. Method C usually requires more logic than either method A or method B. Method C provides a fault-secure implementation provided that the minimum distance of all compatible data input sets is at least 3. Unfortunately, we have shown in Theorem 5, that if method B is not feasible for a given function and input encoding, then no method C implementation can satisfy the totally self-checking goal. Therefore, if the desire is to construct circuits which satisfy the totally self-checking goal, then only method B merits further consideration.

In Theorem 4, we required that the Hamming distance between any two data input vectors be at least 3. This requirement is actually more restrictive than necessary. In particular, if a compatible set of data input vectors all produce data outputs which are all in the same set of compatible output vectors, then it is unnecessary to use the data input vector to calculate the check output vectors. In this case, the check input vector implies the check output vector. Consequently, for this check input vector, none of the check output bits is a function of any of the data input bits. If the check circuit has no static hazards, then none of the check output bits is sensitized to any of the data input bits. Therefore, it is only necessary that those compatible data input vectors which may produce data output vectors in different compatible output sets must have a minimum Hamming distance greater than 2.

Figure 4.12 demonstrates three different methods of generating the check output vector. A fourth method exists where both the check output and data output vectors are calculated using both the check input and the data input vectors. This method is not considered since it violates the spirit of a separable implementation. One of the advantages of a separable implementation is that the data portion of the circuit is unchanged by the coding function. If the data output vector were computed from both the data input and the check input vectors*, the data portion of the circuit would be changed.

4.4. CED Under a General Single Failure Indeterminate Fault Model

The simplified indeterminate fault model is adequate for describing failures that only affect a single line. Unfortunately*, the simplified indeterminate fault model fails to take into account the behavior of bridging failures. For this reason*, we propose a new fault model which includes bridging failures.

4.4.1* Fault Model Assumptions and Properties

The general single failure indeterminate fault model assumes that any physical failure that causes a short between two nodes causes the value on the two nodes to become ternary u values.* Any physical failure which affects a single node causes the value on the node to become a ternary u value.

---

*Only bridging failures between two nodes are considered. The probability that a single failure causes more than 2 lines to become shorted is quite low.

Clearly, the general single-failure indeterminate fault model and the simplified indeterminate fault model are identical for physical failures that affect only a single node. The difference is that the general single-failure indeterminate fault model is also able to model failures which cause two nodes to become shorted together. We assume that a bridging failure always causes both nodes to assume a ternary u value. It can be argued that if both lines have the same Boolean value, the short has no effect. In most cases, this is true. For some types of circuits which are very sensitive to changes in circuit parameters (i.e., certain classes of dynamic circuits), a short between two nodes may definitely affect circuit operation, even when they would have the same Boolean value under no fault. For other classes of circuits it is also possible to make the assumption that both nodes assume a u value only when the nodes have different Boolean logic values under no failure. In this case, any time both lines have the same value, we still must consider the effect of single faults at each node.

Most of the theorems and procedures which were developed in Section 4.3 for the simplified indeterminate fault model have an analog for the general single-failure indeterminate fault model. When considering a theorem for the general single-failure indeterminate fault model, which is analogous to a theorem we have considered for the simplified indeterminate fault model, we use a "*" after the theorem's number to indicate that the theorem applies to the general single-failure indeterminate fault model.

When more than one variable of a logic function may be an indeterminate value, the Boolean difference is no longer satisfactory for determining whether an output is sensitized to n values on several inputs. Consider

$$X = (x_1, \ldots, x_p, x_{p+1}, \ldots, x_n)$$

where X represents an input vector to some combinational function f. If function f has no p-variable logic hazards, then the output of f is sensitized to ternary n values on $(x_1, \ldots, x_p)$ if and only if there exists both 1's and 0's specified for f within the 2^p cells of the sub-cube $(x_{p+1}, \ldots, x_n)$. When a p-variable logic hazard exists, then the output of f is sensitized, even if the 2^p cells of the sub-cube $(x_{p+1}, \ldots, x_n)$ are specified as all 1's or all 0's.

In the general single-failure indeterminate fault model, we assume that any two nodes in the circuit can be shorted together. In practice, only lines which are in close proximity to one another can become shorted. Unfortunately, unless the circuit layout is available, there is no way of knowing which lines are near each other. For this reason, we assume that with one exception, any line in the circuit may become shorted to any other line in the circuit. The one exception concerns shorts between the data and check portions of the circuit. We assume that the circuit is designed so that no shorts can occur between the check circuit and data circuit. Presumably, a design rule can be specified so that if two lines are separated by some distance, no short can occur between the two nodes. This restriction insures that no single short will cause an error to occur in both the data and check output

vectors. In many cases, a circuit layout is such that inputs and outputs are on opposite sides of the circuit. If this is the case, the probability of a short between an input node and an output nodes is very low. We assume that input-output shorts may occur. If enough information is known about the layout, it may be desirable to assume that input-output shorts do not occur. All of the results of this section may be easily modified if desired for the assumption that input-output shorts do not occur.

We are now ready to begin reconsideration of the theorems which we have already developed for the simplified indeterminate fault model.

Hypothesis*: Indeterminate values at a pair of nodes is the most general model for a bridging fault.

Theorem 2*: For any switching function, an implementation exists in which all failures allowed by the general single-failure indeterminate fault model may be modeled as ternary u values on one or two input lines, ternary u values on one or two output lines, or ternary u values on a single input line and a single output line.

Proof: If one of the two u values behaves as the correct logic value, then this situation is equivalent to a single u value on an input or output. In the proof of Theorem 2, we showed that a single u value on an input or output could model all failures in the simplified indeterminate fault model. Therefore, we only need to consider bridging failures. Assume the function is implemented in the form of Figure 4.6, i.e., no output bits share logic. Clearly, any failure which causes a short between two input lines may be modeled as a pair of indeterminate values on

the two shorted input lines. A short between two nodes within the logic for one output bit only affects that output. This condition may be represented as a single ternary u value on the affected output. Any short that occurs between two nodes associated with two distinct output bits, can at most affect the two output bits. Thus, such faults may be modeled as a pair of ternary u values on these outputs. A short that occurs between an input node and an output node may be modeled as the as a ternary u value on the affected input and a ternary u value on the affected output. Therefore, for this implementation, all failures allowed by the general single-failure indeterminate fault model may be modeled as ternary u values on at most two input lines, two output lines, or one of each.

Just as was the case for the simplified indeterminate fault model, Theorem 2* significantly reduces the number of faults which must be considered for implementations of the form of Figure 4.6. If shorts between an input node and an output node are not being considered, then only pairs of ternary u values on input nodes and pairs of ternary u values on output nodes need to be considered.

Theorem 3*: Functional duplication provides an implementation which satisfies the totally self-checking goal with respect to the general single-failure indeterminate fault model for any switching function.

Proof: Based on the assumption that the circuit can be designed such that no node in the data portion of the circuit can be

shorted to a node in the check portion of the circuit, the proof is identical to the proof for Theorem 3.

Corollary 1*: If a functional duplication implementation contains no redundant logic (when only the input code space may be applied), then it is totally self-checking.

Proof: The proof is identical to the proof for Corollary 1.

4.4.2. Economical Implementations for the General Indeterminate Fault Model

Once again, we are now left with the question of when, if ever, an implementation exists which is cheaper than functional duplication. The procedure that we developed for the simplified indeterminate fault model is directly applicable to the general single-failure indeterminate fault model. The only difference is that we must consider faults on a pair of input and output lines rather than single faults.

As an example of searching for a more economical code, consider a four-input, three-output function. The inputs consist of two 2-bit numbers, $X = x_1x_0$ and $Y = y_1y_0$. The output $S = s_2s_1s_0$ is the sum of X and Y. Figure 4.13 shows the truth table for the function and the result of failures on all pairs of inputs. It is assumed the function is implemented without any 2-variable logic hazards so that the sensitized bits may be determined from the truth table. By only considering faults on the inputs, we have the situation where any of the correct output vectors, except 0, can be transformed to any other output vector. When 0 is the correct output vector, then any output vector may result

FUNCTION					OUTPUT FOR INDETERMINATE BRIDGING FAULT IN					
x_1	x_0	y_1	y_0	S	$x_1 x_0$ S	$x_1 y_1$ S	$x_1 y_0$ S	$x_0 y_1$ S	$w_0 p_0$ S	$w_1 p_0$ S
0	0	0	0	000	0uu	uu0	0uu	0uu	0uu	0uu
0	0	0	1	001	uuu	uu1	0uu	uuu	0uu	0uu
0	0	1	0	010	uuu	uu0	uuu	0uu	uuu	0uu
0	0	1	1	011	uuu	uu1	uuu	uuu	uuu	0uu
0	1	0	0	001	0uu	uu1	uuu	0uu	0uu	uuu
0	1	0	1	010	uuu	uu0	uuu	uuu	0uu	uuu
0	1	1	0	011	uuu	uu1	uuu	0uu	uuu	uuu
0	1	1	1	100	uuu	uu0	uuu	uuu	uuu	uuu
1	0	0	0	010	0uu	uu0	0uu	uuu	uuu	uuu
1	0	0	1	011	uuu	uu1	0uu	uuu	uuu	uuu
1	0	1	0	100	uuu	uu0	uuu	uuu	uuu	uuu
1	0	1	1	101	uuu	uu1	uuu	uuu	uuu	uuu
1	1	0	0	011	0uu	uu1	uuu	uuu	uuu	uuu
1	1	0	1	100	uuu	uu0	uuu	uuu	uuu	uuu
1	1	1	0	101	uuu	uu1	uuu	uuu	uuu	uuu
1	1	1	1	110	uuu	uu0	uuu	uuu	uuu	uuu

Figure 4.13 Two-Bit Adder Example.

except 5 and 7. Since output vector 7 is never a legal output vector, it may be ignored. When output faults are considered, it is possible for the correct output vector 0 to be transformed into output vector 5. Therefore, any of the correct output vectors can be transformed by a modeled failure into any of the other legal output vectors. Clearly, functional duplication is the cheapest code for this example if the general single-fault indeterminate fault model is used.

Figure 4.14 shows the behavior of the outputs under input-output shorts. In order to consider the effects of input-output shorts, it is necessary to consider a ternary u value on one input node and one output node simultaneously. The procedure of Section 4.3.3 considers the effect of ternary u values on all single input nodes. The first four columns of Figure 4.14 show the effect of faults on the input nodes. If an indeterminate fault simultaneously occurs on an output node, then the resulting output vector may be altered in at most one additional bit position. Therefore, the output vector resulting from a ternary u value at both an input node and an output node, as shown in the output map of Figure 4.14, is either:

(1) the correct output vector

(2) one of the incorrect output vectors that can result from a fault on an input node

(3) other output vectors which are a Hamming distance of 1 from one of the output vectors in (1) or (2).

The procedure for finding codes that are more economical than functional duplication and that have implementations that are fault-secure

FUNCTION					OUTPUT FOR INDETERMINATE FAULT IN				OUTPUT MAP	
x_1	x_0	y_1	y_0	S	x_1 S	x_0 S	y_1 S	y_0 S		
0	0	0	0	000	0u0	00u	0u0	00u	0	1,2
0	0	0	1	001	0u1	0uu	0u1	00u	1	0,2,3
0	0	1	0	010	uu0	01u	0u0	01u	2	0,3,4,6
0	0	1	1	011	uu1	uuu	0u1	0u1	3	0,1,2,4,5,6,7
0	1	0	0	001	0u1	00u	0u1	0uu	1	0,2,3
0	1	0	1	010	uu0	0uu	uu0	0uu	2	0,1,3,4,6
0	1	1	0	011	uu1	01u	0u1	uuu	3	0,1,2,4,5,6,7
0	1	1	1	100	1u0	uuu	uu0	uuu	4	0,1,2,3,5,6,7
1	0	0	0	010	0u0	01u	uu0	01u	2	0,3,4,6
1	0	0	1	011	0u1	1 uuu	uu1	01u	3	0,1,2,4,5,6,7
1	0	1	0	100	uu0	1 10u	1 uu0	1 10u	4	0,2,4,5,6
1	0	1	1	101	uu1	1 1uu	1 uu1	1 10u	5	1,3,4,6,7
1	1	0	0	011	0u1	1 01u	1 uu1	1 uuu	3	0,1,2,4,5,6,7
1	1	0	1	100	uu0	1 uuu	1 uu1	1 uuu	4	0,1,2,3,5,6,7
1	1	1	0	101	uu1	1 10u	1 1uu	1 1uu	5	1,2,3,4,6,7
1	1	1	1	110	1u0	1 1uu	1 1uu	1 1uu	6	4,5,7

Figure 4.14 Behavior of Input-Output Faults in Two-Bit Adder.

with respect to the general single-failure indeterminate fault model may now be summarized as follows:

(1) Construct a truth table for the desired switching function. This function is implemented by the data portion of the circuit.

(2) For each possible data input vector, determine the possible incorrect data output vectors that may result from a fault on a pair of inputs.

(3) Summarize the results from step 2 to obtain a list of each correct data output vector and the incorrect data output vectors that may result from a pair of input faults.

(4) Update the list from step 3 to include the effects of faults on a pair of output lines.

(5) Update the list from step 4 to include the effects of faults on an input line and an output line simultaneously.

(6) Determine the minimum number of sets of compatible data output vectors so that each output vector is included in exactly one set.

(7) The minimum number of check bits is the smallest integer which is greater than or equal to the $\log_2$ of the minimum number of compatible sets.

So far, we assumed that any two nodes in the data portion of the circuit may become shorted together. If it is known a priori that two particular inputs cannot become shorted, then this fault need not be considered when determining the effects of faults on output vectors. Likewise, if it is known that two outputs (and the logic which computes these outputs) cannot be shorted together or that an input node cannot be shorted to an output node, then these faults do not have to be

considered either. It is only necessary to consider the effects of faults which may actually occur.

This procedure is based on the assumption that any short between two nodes results in a ternary u value on both nodes regardless of what the original logic values of the shorted nodes would be under no fault. For some types of circuits, particularly static circuits, this assumption is overly pessimistic. For such circuits, a more reasonable assumption is that a bridging failure between two nodes, causes a ternary u value at the node only if the original values at the nodes are different. The procedure for finding more economical codes, can easily be modified to work with this assumption. The only difference is that if the two nodes have the same value, then the short has no effect on circuit operation. In those cases where the failed nodes have different values, the above procedure is unchanged. In the remaining cases where the nodes have the same value, the effect of a single ternary u value on each of the two nodes individually must be considered (i.e., the effect of a single ternary u value needs to be considered for each input vector only for nodes whose bridging faults have no effect).

4.4.3. Check Vector Generation

The general single-failure indeterminate fault model presents the same problems for check vector generation as in the simplified indeterminate fault model. The three methods presented in Figure 4.12 are still possible candidates for generating the check vector. Method A is the method to use when the circuit receives unencoded input data. Method B is the method to use when there is enough information in the

check input vector to calculate the check output vector. When the check input vector contains insufficient information, method C must be used.

Theorem 4*: Method C may always be used to provide a fault-secure implementation of the check portion of the circuit with respect to the general single-failure indeterminate fault model provided that the Hamming distance between any two data input vectors in a compatible set is at least 5.

Proof: The proof is identical to the proof for Theorem 4 except that the check circuit must be specified so that no pair of faults on the data input lines causes the check output vector to change. This requirement can always be met when the minimum Hamming distance between any two data input vectors in a compatible set is at least 5.

Theorem 5*: If a circuit cannot be implemented using method B, then no implementation using method C satisfies the totally self-checking goal with respect to the general single-failure indeterminate fault model.

Proof: The proof is identical to the proof for Theorem 5 except that we must consider a pair of faults on data input lines.

4.5. Checker Requirements

It was stated in Section 4.2.1 that a totally self-checking checker must be both totally self-checking and code disjoint. As pointed out by Smith [5], it is not actually necessary for the checker to satisfy the fault-secure property. A checker which is self-testing and code disjoint also operates satisfactorily. The fault-secure property is not necessary since what is important is whether the output from the circuit

being checked is a codeword or a non-codeword. If a circuit is totally self-checking and code disjoint, then as long as the checker is operating properly, it will always produce a non-codeword output if the output from the circuit being checked is a non-codeword. When the checker fails, then the totally self-checking property insures that there is some test to detect the failure. As long as all modeled failures are testable, it is not necessary for the checker output vector to be the correct codeword output under all possible faults and checker input vectors. Therefore, the checker does not need to be fault-secure.

Checkers for indeterminate faults are much easier to design if they do not have to satisfy the fault-secure property. We have assumed that checkers are unable to detect indeterminate values. Therefore, a checker cannot be code disjoint with respect to indeterminate failures. Checkers should, however, be code disjoint with respect to vectors which contain only Boolean values. If the input to the checker is a potential-codeword, then the precise response of the checker becomes non-deterministic.

In our design methodology, checker input vectors come from the outputs of the flip-flops which separate blocks of combinational logic. If a failure occurs inside the block of logic, then the flip-flops should with very high probability have a legal logic value output. The probability that more than one flip-flop passes an indeterminate input through to its output should thus be negligible with respect to the probability that some multiple or other unmodeled failure occurs. However, we do need to be concerned about the checker receiving an indeterminate

value, for example if one of the flip-flops should fail. Thus a checker may experience three different conditions when a failure occurs: the checker may receive a non-codeword with only Boolean values, the checker may receive a potential codeword that has exactly one indeterminate value (two indeterminate values due to a short if the single-failure indeterminate fault model is used), or the checker may receive the correct codeword. The first case should be detected by the checker, the next case is compatible with the requirements for concurrent error detection in the next block of combinational logic, and the last case involves no error.

If the checker is code disjoint with respect to Boolean values and self-testing with respect to indeterminate faults occurring within itself, it is able to respond appropriately to all three of the situations that may occur. Note that iff one of the checker's inputs is indeterminate, then the failure may or may not be detected. If it is not detected, the next block of combinational logic which accepts the output of this circuit as its input, may receive one (two if the general single-failure indeterminate fault model is used) incorrect input bits.

Any checker which is acceptable for the single stuck-at fault model is also acceptable for the simplified indeterminate fault model. Since the checker is self-testing with respect to single stuck-at faults it must also be self-testing with respect to simplified indeterminate fault model faults. The checker is also code disjoint for all vectors which only contain Boolean values. This fact implies that any non-codeword input vector results in a non-codeword output vector. If the input vec-

tor is a potential codeword, then at least one output must be sensitized to all of the input vector bits. If we consider all possible Boolean vectors that can be constructed by replacing indeterminate values in the potential codeword by Boolean values, exactly one of these is a codeword (the potential codeword's corresponding codeword). If the potential codeword applied to the checker's input is the corresponding codeword, then the checker output vector must be a codeword. When the other Boolean vectors (which are non-codewords) are applied, then the checker output vector must be a non-codeword. Therefore, at least one output bit of the checker must be sensitized to the checker input vector bits. Therefore, if the checker is adequate for single stuck-at faults, it is also adequate for the simplified indeterminate fault model. This result is quite important since a variety of checkers for different codes have been designed under the single stuck-at fault assumption. Techniques for designing checkers are discussed in [5].

Checker design is more complicated for the general single-failure indeterminate fault model. A line of reasoning similar to that used for the simplified indeterminate fault model may be used to show that a checker which is adequate for a double stuck-at fault model is also adequate for the general single-failure indeterminate fault model. Unfortunately, checker designs for a double stuck-at fault model are not well-known. One possible solution is to use duplicate checkers that are designed for single stuck-at faults. The duplicate checkers would be placed so as to prevent a short between nodes in two distinct checkers. In addition, the checker inputs should be buffered before going to each checker so that a bridging fault in one of the checkers will not affect

the other checker through the checker input lines. With such an approach, the outputs of at most one checker will be erroneous.

Consider a totally self-checking system constructed from a number of smaller totally self-checking modules. If a checker is placed at the output of each module, then instead of having one set of encoded lines which indicate the presence of an error, there are several sets of error indication lines (one set from each of the checkers). It is possible to use one global checker which checks the outputs of all the other checkers. The output of the global checker produces one set of error indication lines which indicate if an error has occurred anywhere in the system.

If a failure occurs in one of the flip-flops which separates two blocks of combinational logic, it is possible for a simple global checker scheme to fail. In particular, the checker which receives the output of the failed flip-flop may have an indeterminate value input (two indeterminate value inputs if the general single-failure indeterminate fault model is used). The next combinational block which receives the output of the failed flip-flop may also receive an indeterminate value input (two indeterminate value inputs if the general single-failure indeterminate fault model is used) and in response, produce a non-codeword output. If this situation occurs, the global checker may receive both an indeterminate value (from the checker of the first block of logic) and a non-codeword input (from the checker of the second block of logic). We assume as above, that there is a negligible probability that an indeterminate input can propagate through an entire

logic block and its output flip-flops. If the second logic block produces a Boolean-valued non-codeword*, then the global checker must indicate an error if the system is to operate appropriately.

However, because of the presence of an indeterminate value output from the first checker, it is possible for a codeword to be produced by the global checker due to the fact that the indeterminate value can propagate through one or more stages of the global checker. In such a case, it is possible that no error would be indicated by the global checker even though the second block checker output indicates an error. To make this possibility extremely unlikely, flip-flops should separate the outputs of the block checkers from the inputs of the global checkers. In this manner, any indeterminate values produced by one of the block checkers should become legal logic values before they are presented to the global checker.

CHAPTER 5

Conclusion

5.1. Evaluation of Fault Model

Two models for concurrent error detection are defined in Chapter 4: the simplified indeterminate fault model and the general single failure indeterminate fault model. These fault models are based on indeterminate-type faults. We are now in a position to evaluate these fault models by comparing them to the traditional fault models using the criteria proposed in Chapter 1.

5.1.1. Fault Model Accuracy

The indeterminate-type fault is based on the analyses of Chapters 2 and 3. This analysis showed that when MOS logic circuits fail, they may produce outputs that are not legal logic values. Traditional fault models rely on faults that may be represented using Boolean algebra (i.e., stuck-at faults, wired logic faults, etc.). Unfortunately, these traditional models are not able to represent many of the types of anomalous behavior that we have discussed in Chapter 3.

Historically, faults and tests for faults have been divided into two broad classes: logical (or static) and parametric (or dynamic). Logical faults are defined by Breuer and Friedman [64] as those faults that change the logical behavior of some element or signal. Parametric faults are considered to be those faults that cannot be modeled as

logical faults. There is quite a bit of ambiguity in such a classification of faults. Beh et all. [65] define static quality as

> the occurrence of defects that if present would most certainly cause a circuit failure in all systems applications if exercised.

Dynamic quality is defined as

> the occurrence of defects that if present may possibly cause a circuit failure in some or all system applications if exercised.

Perhaps the most reasonable definition is that logical (static) faults alter DC behavior while dynamic (parametric) faults alter behavior of the circuits at higher clock rates. From such definitions, it is hard to state definitively that a specific failure results in one type of fault or the other. In fact# it is difficult to state that a given behavior should be classified as a logical (static) or parametric (dynamic) fault. Clearly# most of the traditional fault models are intended to address logical fault types.

When concurrent error detection is incorporated in a system, the goal is to detect errors when they occur. Whether a fault is logical or parametric is of little concern to the end user who has paid a substantial premium for the concurrent error detection capability. To the end user, it is only important that the system detect errors in a timely fashion.

Many of the traditional fault models are special cases of the indeterminate fault models of Chapter 4. The single stuck-at fault model and stuck-open fault model are special cases of both the simplified indeterminate fault model and the general single failure

indeterminate fault model. In addition, the bridging fault model is a special case of the general single failure indeterminate fault model. The only traditional fault models that are not covered by one of the indeterminate fault models of Chapter 4, are the unidirectional fault model and the unidirectional error fault model. The unidirectional fault models are intended to cover two distinct type of failures: failures of certain global signal lines, and device and line failures.

The first type of failure is usually catastrophic, such as the complete failure of an integrated circuit's power line. If an entire integrated circuit loses its power, all outputs drift rather quickly to a logic 0 and remain at a logic 0 until power is restored. Such a failure clearly results in both a unidirectional fault and a unidirectional error. If a ground line fails instead of a power line, it is more difficult to predict precisely how the integrated circuit outputs respond. If the integrated circuit is static NMOS, then the outputs certainly would all become logic 1's. If the integrated circuit is CMOS, then probably one or more of the circuit's outputs would be indeterminate. Therefore, for a global failure of the ground lines in CMOS logic, the unidirectional fault model is of questionable validity. The indeterminate fault models are unable to model the effects of a global signal failure. If it is desirable to protect against such global power and ground failures (or any other failure causing a unidirectional error), then a <u>two-rail</u> implementation may always be used. A two-rail implementation consists of the original circuit that becomes the data portion of the circuit and the Boolean dual [56] of the original circuit that forms the check portion of the circuit. Such a two-rail

implementation always exists, and furthermore, it satisfies the functional duplication property. Since a two-rail code is unordered, it may be used to detect the occurrence of any unidirectional error. In some cases, codes more economical than functional duplication may also be unordered. In this situation, the more economical code also detects all unidirectional errors. If it is only necessary to detect the situation where a power or ground failure causes all outputs to become all logic 0's or all logic 1's, then it is only necessary that the all 0's output vector and all 1's output vector not be legal codewords. This requirement is significantly less restrictive than requiring an unordered code. By carefully assigning the check vectors to the data vectors, it is always possible to make the all 0's output vector and the all 1's output vector be non-codewords as long as the check output vector contains more than one bit.

The second type of failure that unidirectional fault models are intended to cover, is the single failure of a device or line. Usually, this is done for structured elements. For instance, Banerjee [4] shows that under certain restrictions, failures in a PLA or decoder result in a unidirectional error at the device's output. From the hypotheses of Chapter 4, any such failures are modeled by indeterminate faults.

Therefore, all traditional fault models, except the unidirectional fault model and the unidirectional error fault model, are special cases of the indeterminate fault models. The indeterminate fault models are also applicable to unidirectional errors caused by the failure of a single line or device. In addition, many of the codes that are derived

using the indeterminate fault models also protect against all unidirectional errors including those caused by global power and ground failures. As mentioned in Chapter 1, Smith [5] has shown that the unidirectional fault model requires that an implementation be built with noninverting gates. This makes the unidirectional fault model useless for most MOS circuits.

With the possible exception of some type of unidirectional faults, all traditional fault models are merely special cases of our indeterminate fault models. Therefore, the indeterminate fault models should be more accurate. In addition to the logical type of faults modeled by the traditional models, the indeterminate fault models are also able to account for parametric-type faults that are beyond the ability of traditional models to describe. These parametric faults include timing failures and oscillations. The biggest limitation of the indeterminate fault models is their inability to model the behavior of multiple device failures. In many cases, the behavior of such multiple failures will map into one of the modeled faults. If a functional duplication code is used, then the circuit is protected against an arbitrary number of failures of any type as long as these failures only affect either the data portion or check portion of the circuit, but not both simultaneously.

5.1.2. Ease of Analysis

The second criterion discussed in Chapter 1 is ease of analysis. The indeterminate fault model is a very easy fault model to work with. This is primarily due to the fact that the fault model is comprehensive

for many types of physical failures. The simplified indeterminate fault model accurately represents the behavior of all failures modeled by the simplified indeterminate fault model as well as shorts between nodes. Therefore, these fault models do not need to be combined with other fault models to account for the behavior of all single failures accurately. The traditional fault models often must be combined with other fault models in order to cover certain types of physical failures.

If an implementation in the form of Figure 4.6 is acceptable, then only faults on circuit inputs or outputs need to be considered for the indeterminate fault models. With the traditional fault models, it is generally necessary to consider faults on all nodes of the circuit, not just inputs and outputs. Typically, a circuit has many more nodes than inputs and outputs. Therefore, the number of faults that must be considered is greatly reduced. It is true that for many of the traditional fault models, especially the stuck-at fault model, many faults are indistinguishable from other faults. However, even after collapsing the fault model, there are usually significantly more stuck-at faults that must be considered than simplified indeterminate faults.

The difficulty in using the indeterminate fault models lies in the fact that a ternary algebra must be used rather than Boolean algebra. Fortunately, the rules of ternary algebra are very similar to the rules of Boolean algebra. Furthermore, when the inputs to a ternary function are restricted to 0 and 1 values, then the function's behavior may be described using Boolean algebra. Thus, perhaps with the exception of

unfamiliarity, ternary algebra is no more difficult to use than Boolean algebra.

In general, the indeterminate fault models should provide good ease of analysis. Both indeterminate fault models are comprehensive models and only require that a limited number of faults be considered. Although ternary algebra is required in order to analyze circuits with indeterminate faults, this should provide no real difficulty.

5.1.3. Cost of Fault Tolerance

The cost of fault tolerance for any fault model is highly dependent on the target system. Some systems naturally lend themselves more readily to concurrent error detection than others. Furthermore, there are a variety of costs involved in utilizing any fault-tolerance scheme. Such costs include: power cost, size cost, speed cost, and most importantly, monetary cost. Clearly, a variety of tradeoffs exist between each of these costs. Usually one is most concerned with the tradeoff between monetary cost and the other types of costs.

When attempting to implement a concurrent error detection scheme, one is faced with two basic choices: whether to implement the entire system as one single totally self-checking circuit or to divide the system into several smaller totally self-checking circuits that are interconnected to perform the same function.

A variety of tradeoffs are involved in this decision. All other things being equal, the smaller the blocks of logic checked by a checker, the better the logic block's observability, and hence, the

easier the block is to test. Therefore, breaking the system into several smaller circuits is generally advantageous in regard to increasing the system's testability. It is also usually easier to analyze small circuits as opposed to large circuits. Thus, it is usually easier to find totally self-checking implementations if the system is broken into a number of smaller parts. It is not always obvious how to partition a system into a number of smaller parts in order to maximize testability and minimize the difficulty in finding a totally self-checking implementation of the system. In general, it is often desirable to partition the system into its functional parts such as adders, register banks, busses, etc. Such a partition usually allows an efficient implementation.

An alternative to partitioning is to implement the entire system as one totally self-checking circuit. In general, this approach results in poorer testability, possibly more logic (and thus higher power consumption), and larger system size. However, for large and very large scale integrated circuits, this has an important advantage. By duplicating standard off-the-shelf circuits, totally self-checking circuits can be built quite cheaply. Due to the high development cost and relatively low manufacturing cost, the price of a very large scale integrated circuit is a strong function of the number of identical circuits manufactured [34]. Typically, the demand for systems with concurrent error detection is smaller than the demand for the same or a similar system without concurrent error detection. If custom integrated circuits must be designed, then the monetary cost of a system with concurrent error detection is much greater than the monetary cost of a similar system

built by duplication with off-the-shelf parts. Even if a custom integrated circuit must be designed, duplication still simplifies the design process since very little analysis is required. Unless power consumption and/or size is an overriding consideration, then any time a very large scale integrated circuit already exists that performs the desired function, the best way to gain concurrent error detection is simply to use two copies of the existing integrated circuit.

Intel's iAPX 432 family [66] uses this approach so that the same set of integrated circuits may be used for those applications that require concurrent error detection and those applications that do not require concurrent error detection (i.e., those where the benefits of concurrent error detection are outweighed by its cost). Each of the integrated circuits in the iAPX 432 family are designed so that each output pin may also serve as an equality checker. One pin is devoted to "programming" the chip to be a master (circuit operates normally), or a checker. All pins on the master and checker integrated circuits except the programming pin and the error indication pin are wired together. Any discrepancy between the logical values of the integrated circuits' outputs are indicated by the checker circuit. The only errors that are not detected by this scheme are the failure of certain of the global signal lines. Many of the issues involved in protecting against global signal failures are discussed in [67].

In almost all cases, the cost of fault tolerance with an indeterminate fault model will be greater than or equal to the cost using a traditional fault model. This is due to the fact that except for the

unidirectional fault model (which is not applicable to logic constructed from inverting circuits) and in some cases the unidirectional error model. all the traditional fault models are only special cases of the indeterminate fault models. Any implementation that provides concurrent error detection for indeterminate fault models will also provide concurrent error detection for the traditional fault models (except the unidirectional fault models). Therefore, the cost of fault tolerance with the traditional fault models will always be less than or equal to the cost of fault tolerance for the indeterminate fault models.

As we have mentioned above, duplication (whether at the intra-integrated circuit level or the inter-integrated circuit level) has many advantages, especially for systems that are produced in low numbers. In many cases, duplication will be used regardless of the fault model. Therefore, as a practical matter, the cost of fault tolerance in most cases is roughly the same whether indeterminate fault models are used or one of the traditional fault models is used.

5.2. Summary

In Chapter 2, typical physical failure models are reviewed. Three broad classes of physical failures are considered: interconnect failures, transistor failures, and radiation-induced soft failures. Interconnect failures result in shorts and opens in the lines that link the transistors. Transistor failures are caused by a shift in device parameters and device breakdown. Radiation-induced soft failures are transient, non-recurring upsets of a node or nodes in the circuit caused by high energy radiation generating charge carriers in the integrated

circuit. Circuits become more susceptible to all three of these types of failures as devices are scaled.

In Chapter 3, the effects of these failures on integrated circuits are studied. It is found that nearly all of these failures may be modeled as resistive shorts or opens in a circuit. Models are developed for static NMOS, static CMOS, and dynamic NMOS inverters. These models are used to predict the behavior of inverters under failure. It is shown that when physical failures occur, the logic levels of the inverter output may degrade, the inverter switching speed may decrease and under some circumstances, the inverter output may oscillate. Integrated circuits are also constantly exposed to the effects of random noise which may cause soft failures, similar in nature to radiation-induced soft failures. Thus, the analysis of Chapter 3 shows that physical failures, in general, may cause the output of a failed circuit to assume a value that is logically undefined.

The behavior of good circuits with logically undefined inputs is examined. It is also shown that a flip-flop may undergo metastable operation when its inputs are undefined logic values. When a flip-flop is in a metastable state, its outputs are generally illegal logic values. Since clocked flip-flops are commonly used to separate blocks of combinational logic, the effect of circuit parameters on the probability of entering a metastable state and average length of metastable operation is studied. It is shown that high gain and high bandwidth are important to minimize the effects of metastable operation.

In Chapter 4, concurrent error detection for errors caused by physical failures is discussed. Indeterminate faults are used to represent the undefined logic values that may occur due to physical failures. It is shown that indeterminate faults may also be used to represent the behavior of any failure that forces a single node of the circuit to a legal logic value. A ternary algebra is used to describe the behavior of logic gates with indeterminate fault inputs. By using the ternary algebra, it is shown that static hazards and p-variable logic hazards will sensitize an output to an indeterminate fault, even when the output is not a function of the faulted node.

The traditional definitions for fault-secure, self-testing, and totally self-checking are discussed. It is shown that due to the non-deterministic behavior of indeterminate faults, these definitions are inappropriate for systems that are subject to physical failures that may cause indeterminate faults. New definitions of fault-secure, self testing, totally self-checking, and strongly fault secure are given that are compatible with indeterminate faults.

Two fault models are introduced that are based on indeterminate faults. The concept of functional duplication is introduced. It is shown that a functional duplication implementation, that satisfies the totally self-checking goal, exists for any switching function. Procedures are also discussed for each of the fault models to find any codes that may exist for a function that are less costly than functional duplication. The problem of generating the check output vectors when

the circuit in question is part of a larger totally self-checking system is also examined.

5.3. Suggestions for Future Research

One of the major detriments to systems that are totally self-checking with respect to the indeterminate fault model is the testing problem. Further research into methods that generate efficient and effective tests for indeterminate faults is necessary in order to improve the concurrent error detection capabilities of such systems. As mentioned in Chapter 4, testability techniques for intermittent failures appears to be a very promising foundation for developing such techniques for indeterminate faults.

It would be desirable to extend the research of Chapter 4 to cover a broader range of possible circuits and implementations. Since sequential networks are such an important class of circuits, it is imperative to study them explicitly and develop the requirements for providing them with concurrent error detection capability. Non-separable codes should also be examined to determine if such codes might provide more economical implementations of certain functions than separable codes.

The algorithms presented in Chapter 4 to search for codes more economical than functional duplication are straightforward to apply. Unfortunately, for functions with a large number of inputs and outputs, these procedures may become quite unwieldy to apply. For this reason, new search algorithms should be developed to find such codes more efficiently.

One of the major difficulties in using the general single-failure indeterminate fault model is the problem of designing appropriate checkers. Therefore, designs of checkers for the general single-failure indeterminate fault model should be studied.

References

[1] T. W. Williams and K. P. Parker, "Design for testability - a survey," Proc. IEEE, vol. 71, no. 1, Jan. 1983, pp. 98-112.

[2] S. R. McConnel, D. P. Siewiorek, and M. M. Tsao, "The measurement and analysis of transient errors in digital computer systems," IEEE Int. Symp. Fault-Tolerant Computing. Los Angeles, 1979, pp. 67-70.

[3] J. H. Patel and L. Y. Fung, "Concurrent error detection in ALU's by recomputing with shifted operands," IEEE Trans. Computers, vol. C-31, no. 7, July 1982, pp. 589-595.

[4] P. Banerjee, "A model for simulating physical failures in MOS VLSI circuits," Report CSG-13, Coordinated Science Laboratory, University of Illinois, Urbana, IL, 1982.

[5] J. E. Smith, "The design of totally self-checking combinational circuits," Report R-737, Coordinated Science Laboratory, University of Illinois, Urbana, IL, 1976.

[6] D. F. Barbe, Very large scale integration (VLSI) Fundamentals and Applications. Berlin: Springer-Verlag, 1980.

[7] S. Vaidya, D. B. Fraser, and A. K. Shinha, "Electromigration resistance of fine-line Al for VLSI applications," Proc. Int. Reliability Physics, 1980, pp. 165-170.

[8] P. A. Gargini, C. Tseng, and M. H. Woods, "Elimination of silicon electromigration in contacts by the use of an interposed barrier metal," Proc. Int. Reliability Physics, 1982, pp. 66-76.

[9] G. DiGiacomo, "Metal migration (Ag, Cu, Pb) in encapsulated modules and time-to-fail model as a function of the environment and package properties," Proc. Int. Reliability Physics, 1982, pp. 27-33.

[10] J. R. Lloyd, G. S. Hopper, and W. B. Roush, "In situ IR observation of electromigration induced damage in heavily doped polycrystalline silicon resistors," Proc. Int. Reliability Physics. 1982, pp. 47-49.

[11] M. R. Polcari, J. R. Lloyd, and S. Cvikevich, "Electromigration failure in heavily doped polycrystalline silicon," Proc. Int. Reliability Physics, 1980, pp. 178-185.

[12] H. C. Potter and D. R. Reber, "A study of surface charge induced inversion failure of junction isolated monolithic silicon integrated circuits," Proc. Int. Reliability Physics, 1976, pp. 11-17.

[13] B. A. Unger, "Electrostatic discharge failures of semiconductor devices," Proc. Int. Reliability Physics, 1981, pp. 193-199.

[14] A. R. Hart, J. Smyth, and Stan Gorski, "Predicting ESD related reliability effects," Proc. Int. Reliability Physics, 1982, pp. 233-237.

[15] E. S. Anolick, "Screening of time-dependent dielectric breakdowns," Proc. Int. Reliability Physics, 1982, pp. 238-243.

[16] B. Evzent, "Hot electron injection efficiency in IGFET structures," Proc. Int. Reliability Physics, 1977, pp. 1-4.

[17] B. Eitan and D. Frohman-Bentchkowsky, "Hot-electron injection into the oxide in n-channel MOS devices," IEEE Trans. Electron. Devices. vol. ED-28, no. 3, pp. 328-340, March 1981.

[18] P. K. Chaudhari, "Leakage-induced hot carrier instability in phosphorus-doped SiO_2 gate IGFET devices," Proc. Int. Reliability Physics, 1977, pp. 5-9.

[19] S. A. Abbas and R. C. Dockerty, "Hot electron induced degradation of n-channel IGFETs," Proc. Int. Reliability Physics, 1976, pp. 38-41.

[20] M. Nojori and T. Ishihara, "Secondary slow trapping - a new moisture induced instability phenomenon in scaled CMOS devices," Proc. Int. Reliability Physics, 1982, pp. 113-121.

[21] J. P. Mitchell and D. K. Wilson, "Surface effects of radiation on semiconductor devices," Bell Systems Technical Journal, vol. XLVI, no. 1, Jan. 1967, pp. 1-80.

[22] E. H. Snow, A. S. Grove, and D. J. Fitzerald, "Effects of ionizing radiation on oxidized sillicon surfaces and planar devices," Proc. IEEE, vol. 55, no. 7, July 1967, pp. 1168-1185.

[23] I. N. Krishnan and T. M. Chen, "G-R noise and microscopic defects in irradiated junction field effect transistors," Solid-State Electron., vol. 20, Nov. 1977, pp. 897-906.

[24] S. A. Abbas and E. E. Davidson, "Reliability implications of hot electron generation and parasitic bipolar action in an IGFET device," Proc. Int. Reliability Physics, 1976, pp. 18-22.

[25] R. R. Troutman and H. P. Zappe, "A Transient analysis of latch-up in Bulk CMOS," IEEE Trans. Electron. Devices, vol. ED-30, no. 2, Feb. 1983, pp. 170-179.

[26] C. M. Hsieh, P. C. Murley, and R. R. O'Brien, "Dynamics of charge collection from alpha-particle tracks in integrated circuits," Proc. Int. Reliability Physics, 1981, pp. 38-42.

[27] M. L. White, J. W. Serpiello, K. M. Striny, and W. Rosenzweig, "The use of silicone RTV rubber for alpha particle protection on silicon integrated circuits," Proc. Int. Reliability Physics, 1981, pp. 43-47.

[28] J. Galiay, Y. Crouzet, and M. Vergniult, "Physical versus logical fault models MOS LSI circuits: impact on their testability," IEEE Trans. Computers, vol. C-29, no. 6, June 1980, pp. 527-531.

[29] C. Mead and L. Conway, Introduction to VLSI systems. Reading: Addison-Wesley Publishing, 1980.

[30] B. G. Streetman, Solid state electronic devices. Englewood Cliffs: Prentice-Hall, 1980.

[31] R. D. Davis, "Design and analysis of an NMOS operational amplifier with depletion loads," Report R-857, Coordinated Science Laboratory, University of Illinois, Urbana, IL, 1979.

[32] L. A. Glasser, "The analog behavior of digital integrated circuits," Design Automation Conf., 1981, pp. 603-612.

[33] L. P. J. Hoyte, "Automated calculation of device sizes for digital IC designs," M.S. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1982.

[34] S. Muroga, VLSI system design. New York: John Wiley and Sons, 1982.

[35] D. J. Hamilton and W. G. Howard, Basic integrated circuit engineering. Reading: Addison-Wesley, 1979.

[36] L. Strauss, Wave generation and shaping. New York: McGraw-Hill, 1970.

[37] M. Karpovsky and S. Y. H. Su, "Detection and location of input and feedback bridging faults among input and output lines," IEEE Trans. Computers, vol. C-29, no. 6, June 1980, pp. 523-527.

[38] T. Yamada and T. Nanya, "Comments on 'Detection and location of input and feedback bridging faults among input and output lines'," IEEE Trans. Computers, vol. C-32, no. 5, May 1983, pp. 511-512.

[39] R. E. Ziemer and W. H. Tranter, Principles of communications. Boston: Houghton Mifflin, 1976.

[40] J. T. Wallmark, "Noise spikes in digital VLSI circuits," IEEE Trans. Electron. Devices, vol. ED-29, no. 3, March 1982, pp. 451-458.

[41] S. H. Unger, "Asynchronous sequential switching circuits with unrestricted input changes," IEEE Trans. Computers, vol. C-20, no. 12, Dec. 1971, pp. 1437-1444.

[42] L. R. Marino, "The effect of asynchronous inputs on sequential network reliability," IEEE Trans. Computers, vol. C-26, no. 11, Nov. 1977, pp. 1082-1090.

[43] B. I. Strom, "Proof of the equivalent realizability of a time-bound arbiter and a runt-free inertial delay," Sixth Annual Symp. Computer Architecture, 1979, pp. 178-181.

[44] L. R. Marino, "General theory of metastable operation," IEEE Trans. Computers, vol. C-30, no. 2, Feb. 1981, pp. 107-115.

[45] T. J. Chaney, S. M. Ornstein, and W. M. Littlefield, "Beware the synchronizer," IEEE Compcon, 1972, pp. 317-319.

[46] I. Catt, "Time loss through gating of asynchronous logic signal pulses," IEEE Trans. Computers, vol. C-15, no. 2, Feb. 1966, pp. 108-111.

[47] M. J. Stucki and J. R. Cox, "Synchronization strategies," Cal-Tech Conf. VLSI, 1979, pp. 375-393.

[48] P. A. Stoll, "How to avoid synchronization problems," VLSI Design. Nov./Dec. 1982, pp. 56-59.

[49] T. J. Chaney and C. E. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," IEEE Trans. Computers, vol. C-22, no. 4, April 1973, pp. 421-422.

[50] M. Pechoucek, "Anomalous response times of input synchronizers," IEEE Trans. Computers, vol. C-25, no. 2, Feb. 1976, pp. 133-139.

[51] T. J. Chaney and F. U. Rosenberg, "Characterization and scaling of MOS flip flop performance in synchronizer applications," Cal-Tech Conf. VLSI, 1979, pp. 357-374.

[52] G. Lacroix, P. Marchegay, and G. Piel, "Comments on "The Anomalous behavior of flip-flops in synchronizer circuits'," IEEE Trans. Computers, vol. C-31, no. 1, Jan. 1982, pp. 77-78.

[53] D. E. Muller, "Treatment of transition signals in electronic switching circuits by algebraic methods," IRE Trans. Electronic Computers, vol. EC-8, no. 3, Sept. 1959, p. 401.

[54] E. B. Eichelberger, "Hazard detection in combinational and sequential switching circuits," IBM Journal, vol. 9, no. 2, March 1965, pp. 90-99.

[55] D. B. Armstong, "On finding a nearly minimal set of fault detection tests for combinational logic nets," IEEE Trans. Electron. Computers, vol. EC-15, no. 1, Feb. 1966, pp. 66-73.

[56] Z. Kohavi, Switching and finite automata theory. New York: McGraw-Hill, 1978.

[57] J. S. Jephson, R. P. McGuarrie, and R. E. Vogelsberg, "A three-value computer design verification system," IBM Syst. Journal, vol. 8, no. 3, 1969, pp. 178-188.

[58] J. E. Smith and G. Metze, "Strongly fault secure logic networks," IEEE Trans. Computers, vol. C-27, no. 6, June 1978, pp. 491-499.

[59] P. Duhamel and J. C. Rault, "Automatic test generation techniques for analog circuits and systems: a review," IEEE Trans. Circuits and Systems, vol. CAS-26, no. 7, July 1979, pp. 411-440.

[60] S. Kamal and C. V. Page, "Intermittent faults: a model and a detection procedure," IEEE Trans. Computers, vol. C-33, no. 7, July 1974, pp. 713-719.

[61] J. Savir, "Testing for single intermittent failures in combinational circuits by maximizing the probability of fault detection," Report 145, Center for Reliable Computing, Stanford University, Palo Alto, CA, 1977.

[62] J. W. Beyers, L. J. Dohse, J. P. Fucetala, R. L. Kochis, C. G. Lob, G. L. Taylor, and E. R. Zeller, "A 32-bit VLSI CPU chip," IEEE Journal Solid-State Circuits, vol. SC-16, no. 5, Oct. 1981, pp. 537-542.

[63] W. W. Lattin, J. A. Bayliss, D. L. Budde, J. R. Rattner, and W. S. Richardson, "A methodology for VLSI chip design," Lambda, Second Quarter 1981, pp. 34-44.

[64] M. A. Breuer and A. D. Friedman, Diagnosis and reliable design of digital systems. Rockville: Computer Science Press, 1976.

[65] C. C. Beh, K. H. Arya, C. E. Radke, and K. E. Torku, "Do stuck fault models reflect manufacturing defects?," Int. Test Conf., Philadelphia, 1982.

[66] R. Grappel and J. Hemenway, "Understand the newest processor to avoid future shock," EDN, vol. 26, no. 9, April 29, 1981, pp. 129-136.

[67] R. M. Sedmak and H. L. Bergot, "Fault tolerance of a general purpose computer implemented by very large scale integration," IEEE Trans. Computers, vol. C-29, no. 6, June 1980, pp. 492-500.

Vita

Daniel Lee Halperin was born January 23, 1957 in Oak Ridge, Tennessee. He graduated first in his class in the College of Engineering at the University of Tennessee in 1978 with a B.S. in Electrical Engineering. He received his M.S. and Ph.D. in Electrical Engineering in 1981 and 1984, respectively, from the University of Illinois. While at the University of Tennessee, he was inducted into the Phi Eta Sigma, Eta Kappa Nu, Tau Beta Pi, and Phi Kappa Phi honorary societies. While pursuing his graduate studies at the University of Illinois, he was a member of the Computer Systems Group of the Coordinated Science Laboratory. He is currently employed by Hewlett-Packard as a member of the engineering staff in the System Technology Operation at Fort Collins, Colorado.