TESTING DIGITAL CIRCUITS FOR TIMING FAILURES BY OUTPUT WAVEFORM ANALYSIS

Piero Franco

CRC Technical Report No. 94-9
(CSL TR No. 94-637)
September 1994

CENTER FOR RELIABLE COMPUTING

Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University, Stanford, California 94305-4055

TESTING DIGITAL CIRCUITS FOR TIMING FAILURES BY OUTPUT WAVEFORM ANALYSIS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Piero Franco

July 1994

ABSTRACT

Delay testing is done to ensure that a digital circuit functions at the designed speed. Delay testing is complicated by test invalidation and fault detection size. Furthermore, we show that simple delay models are not sufficient to provoke the longest delay through a circuit. Even if all paths are robustly tested, path delay testing cannot guarantee that the circuit functions at the desired speed.

Output Waveform Analysis is a new approach for detecting timing failures in digital circuits. Unlike conventional testing where the circuit outputs are sampled, the waveform between samples is analyzed. The motivation is that delay changes affect the shape of the output waveform, and information can be extracted from the waveform to detect timing failures. This is especially useful as a Design-for-Testability technique for Built-In Self-Test or pseudo-random testing environments, where delay tests are difficult to apply and test invalidation is a problem.

Stability Checking is a simple form of Output Waveform Analysis. In a fault-free circuit, the outputs are expected to have reached the desired logic values by the time they are sampled, so delay faults can be detected by observing the outputs for any changes after the sampling time. Apart from traditional delay testing, Stability Checking is also useful for on-line or concurrent testing under certain timing restrictions. A padding algorithm was implemented to show that circuits can be efficiently modified to meet the required timing constraints.

By analyzing the output waveform before the sampling time, circuits with timing flaws can be detected even before the circuit fails. This is useful in high reliability applications as a screening technique that does not stress the circuit, and for wear-out prediction.

A symbolic waveform simulator has been implemented to show the benefits of the proposed Output Waveform Analysis techniques. Practical test architectures have been designed, and various waveform analyzers have been manufactured and tested. These include circuits implemented using the Stanford BiCMOS process, and a design implemented in a 25k gate Test Evaluation Chip Experiment.

ACKNOWLEDGMENTS

This dissertation describes a new approach for detecting timing failures in digital circuits performed at Stanford University. I am deeply grateful to my advisor, Prof. Edward J. McCluskey, without whose guidance and support this work would not have been possible. I am also grateful to Prof. Giovanni De Micheli, my associate advisor, and Prof. Dwight. G. Nishimura, my committee chairman, for reading my dissertation, and Prof. Oyekunle Olukotun for being the final member of my committee.

I would like to thank my colleagues at the Center for Reliable computing for their help: LaNae Avra, Hong Hao, Siyad Ma, Samy Makar, Samiha Mourad, Rob Norwood, Nirmal Saxena, Alice Tokarnia, and Nur Touba. I also thank Siegrid Munda for her administrative support, and often "helping me catch the DHL truck."

I would like to thank Dr. E. Eichelberger for helpful discussions in the early phase of this work.

I thank the Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, for their hospitality during my stay.

I would like to thank the many people who contributed to the Test Evaluation Chip Experiment. In particular, I would like to thank Robert Stokes for his work on the architectural and detailed design of the Test Chip, and Mike Sarvian for his help in writing and debugging the test program.

I wish to thank Mamma, Papà, Laura, Nonna and Pamela for their love, support, encouragement, understanding, and patience. I dedicate this dissertation to them.

I am grateful to the Chamber of Mines of South Africa for making it possible for me to come to Stanford, and supporting me for the first year.

This work was also supported in part by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization and administered through the Office of Naval Research under Contracts No. N00014-85-K-0600 and N00014-92-J-1782, in part by the National Science Foundation under Grants No. MIP-8709128 and No. MIP-9107760, and in part by Hughes Aircraft Co.

Major funding and support for the Test Evaluation Chip Experiment has been provided by Hughes Aircraft Co., LSI Logic, and Digital Testing Services.

TABLE OF CONTENTS

Chapter 4. On-Line Delay Testing

LIST OF TABLES

LIST OF ILLUSTRATIONS

Chapter 1

Introduction

1.1 BACKGROUND

The lessons learned from the US automotive industry in the 80s are clear. It has taken the industry one decade to recover from losses caused by failing to meet increased consumer product quality expectations. Similarly, increasingly higher quality is required for complex, high speed, digital systems. Not only does the cost of repair grow exponentially for each stage in the manufacturing cycle faulty components go undetected, but the use of systems in critical applications makes high quality and reliability essential.

Since manufacturing yields are typically much lower than the shipped product quality required, testing is done to detect failed components. Traditionally, testing digital circuits has consisted of generating tests to verify that signal nodes are not permanently stuck at one of the logic levels. This is known as *stuck-at* fault testing. Vectors are applied to the circuit under test, and the circuit outputs are sampled at the appropriate time and compared to the expected fault-free response.

Experience has shown that even complete coverage of the classic single stuck-at fault model is often not sufficient to achieve the required quality levels. This has resulted in the investigation of other fault models, including bridging, stuck-open, stuck-on, and timing faults. Timing problems, in particular, cannot be modeled as stuck-at faults. These defects are modeled as *AC faults* or *delay faults*. Which model is "best" is an open question, but it appears that to approach "zero defects" a combination of techniques will be most effective. Some form of timing-based testing is necessary to achieve high quality, as it is impossible to know if a circuit functions at the desired speed unless the longest delays are exercised. It is now standard practice in large high-performance computer companies to do some kind of delay fault testing.

Timing problems occur for two reasons. There are physical failure mechanisms that affect the performance of a system without changing the logic function, that need to be detected in order to achieve high quality. Another reason is that aggressive, statistical timing philosophies are often adopted to increase performance. Worst-case timing philosophy assumes that each component in a path has the worst-case delay, which can be very pessimistic and lead to slow designs. Using a statistical timing philosophy, most circuits will operate at the chosen speed, but a few will not even though they have no defects. Speed binning of components is often done, as a significant premium is paid for faster parts (microprocessors are a common example). Since these components do not have any defects that could be detected by other testing techniques, timing-based testing is necessary.

Delay testing is significantly more complex than stuck-at testing due to the added dimension of time, and a vast literature exists on detecting timing failures. Techniques for detecting timing failures are classified as "direct" or "indirect" in Chapter 2. Direct approaches include "at-speed testing", where design verification or other vectors are applied at system speed, or "delay fault testing", where vectors specifically targeted for the detection of delay faults are generated. Test invalidation by hazards is an important problem in delay testing.

Due to difficulties with detecting delay faults directly, indirect testing approaches have also been suggested. The defect cause rather than the resulting fault is targeted in the indirect testing approaches. One distinguishing characteristic of the indirect approaches is that the emphasis is generally shifted to the observation of the circuit response, rather than the application of patterns. The most well-known technique is quiescent current monitoring, or IDDQ testing. The indirect approaches are useful, but none can guarantee that a circuit functions at the specified speed since the longest delay through the circuit is not tested.

Although much research on delay testing has been done both at universities and in industry, there is not as yet a general solution used in practice for dealing with timing failures.

1.2 OUTPUT WAVEFORM ANALYSIS

This dissertation investigates a new approach for detecting timing failures in digital circuits. *Output Waveform Analysis* was first presented at the 1991 International Test Conference [Franco 91b].

Most work on delay testing has concentrated either on the choice of input test patterns to apply (test pattern generation), or modifying the circuit itself in order to make it easier to test (logic synthesis). Output Waveform Analysis, on the other hand, involves analyzing the output waveforms of the circuit under test to improve the delay fault coverage.

The motivation for Output Waveform Analysis is that, unlike catastrophic failures that simply have incorrect steady-state logic values at the circuit outputs, delay faults change the shape of the output waveforms of the circuit by moving the signal transitions in time. Therefore, since the output waveforms contain information about the circuit delays, instead of only latching the outputs at the sampling time, the output waveforms between samples are analyzed as well. Test patterns are applied as in conventional delay testing, with the addition of circuits that observe the output waveforms between samples. The waveform analyzers can be thought of as “mini-watchdogs” that check individual circuit outputs between samples.

Output Waveform Analysis is a combination of the direct and indirect delay testing approaches. It is direct in that delay faults are explicitly considered, yet is similar to the indirect approaches in that the focus is on the observation of the circuit output response. Output Waveform Analysis is a Design-for-Testability (DFT) technique, where extra resources are added to reduce the difficulty and improve the effectiveness of delay testing, and is suitable for Built-In Self-Test (BIST). Compatibility with BIST was considered important, as it is particularly difficult to detect timing failures in the field without access to complex automatic test equipment.

Although there are many waveform analysis functions of differing complexity and accuracy, they can be classified as shown in Fig. 1.1. The waveform before the sample is observed in *Pre-Sampling Waveform Analysis*, whereas the waveform after the sample is observed in *Post-Sampling Waveform Analysis*. Sampling (latching) the output waveform can be considered to be a special case of Output Waveform Analysis, where only a single point of the waveform is observed.

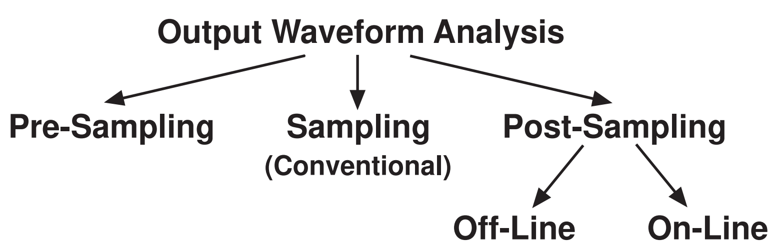


Figure 1.1. Output Waveform Analysis Classes

Applications of Output Waveform Analysis include:

1. Delay testing at various levels, e.g., wafer sort, final testing, or repair test.
2. On-line or concurrent checking.
3. Measuring propagation delays, process variations, and component wear-out.
4. Shortening or eliminating environmental stress screening (burn-in).

1.3 OVERVIEW OF DISSERTATION

This dissertation consists of the design and evaluation of various Design-for-Testability (DFT) techniques based on Output Waveform Analysis. Efficient practical implementations are described, and the benefits in terms of increased quality are shown.

Chapter 2 provides an overview of techniques for detecting timing failures. Early work on delay testing and the issues involved are discussed, before describing the newer delay testing approaches. Problems with currently-proposed approaches are shown, including inaccuracies in delay modeling that can limit the effectiveness of delay testing. For example, although it is generally agreed that two-pattern tests are necessary for delay testing, it is shown that three-pattern tests are actually needed for CMOS circuits. Chapter 2 concludes with a description of Output Waveform Analysis, and shows how it differs from the other approaches.

Different forms of Output Waveform Analysis are then described in more detail in separate chapters. Post-Sampling Waveform Analysis is described first in Chapter 3 as it is the simplest case. One important application of Post-sampling Waveform Analysis, on-line checking, is described in Chapter 4. Chapter 5 covers Pre-Sampling Waveform Analysis. The feasibility of each technique is shown by presenting efficient test architectures, circuit implementations of the waveform analyzers (including a few designs manufactured and tested using the Stanford BiCMOS process), and simulation results.

Chapter 6 provides a brief description of a Test Evaluation Chip Experiment currently underway, with the results so far. The purpose of this experiment is to compare the effectiveness of many different test techniques, and includes Post-Sampling Waveform Analysis. The Test Chip has also been used to estimate the severity of inaccurate delay modeling described in Chapter 2. The Test Chip is a 25k gate CMOS gate array.

Chapter 7 concludes the dissertation.

Evaluation of Output Waveform Analysis requires an accurate representation of the actual waveforms at the circuit outputs for different delay fault values. Existing timing simulators were found to be inconvenient, so an experimental symbolic waveform simulator was implemented. The delay of the faulty element is treated as a variable in the generation of the output waveform. The simulator, WSIM, is described in a Stanford CRC Technical Report [Franco 94c], which is included as Appendix I. The simulation results presented in this dissertation were derived using WSIM.

Appendix II is also a CRC Technical Report [Franco 94d], and contains a more complete description of the Test Evaluation Chip Experiment, including the design of the Test Chip and the test sets applied.

(Note: The Appendices are not included in this Technical Report.)

Chapter 2

Testing For Delay Faults

Early work on testing for delay faults is described in this chapter, followed by definitions and currently proposed approaches for detecting delay faults. Limitations of the different methods are noted, showing that the delay testing problem is not solved. It is shown that certain basic assumptions generally used in delay fault testing are not completely correct. Output Waveform Analysis is then described.

2.1 EARLY WORK IN DELAY FAULT TESTING

Some of the early developments related to delay fault testing are summarized in this section for a historical perspective, and to introduce the classic problems.

Digital circuits were originally tested with vectors designed to verify that the circuit performed the correct logic function. These vectors are often called *functional* or *design verification* vectors. However, primarily because of the difficulty in measuring the effectiveness of a functional test, [Eldred 59] proposed a structural test technique, which is now known as the classic *stuck-at* fault model.

From a synthesis perspective, [McCluskey 62] studied transients (hazards) in combinational circuits. An algorithm was presented based on a labeling that distinguishes between variables that reconverge after traveling different paths. These ideas were later used for work on robust path delay fault testability. Delays were modeled as propagation delay of gates and interconnect. The delay for a transition to propagate from the input to the output of a gate can (i) be different for different inputs, (ii) differ for rising and falling transitions, and (iii) depend on signals which are present at the other gate inputs. An algorithm is given in [Eichelberger 65] for the detection of hazards in combinational and sequential circuits.

One of the earliest papers to deal with delay faults is [Breuer 74b]. It was observed that there are physical failure mechanisms that change the circuit parameters and the timing of a circuit without causing stuck-at faults. These faults were termed *delay faults*. An element was considered to have a delay fault if its actual delay parameters are different from the designed parameters. Several parameters for gate delay were used. *Transport delay* is the time taken for a transition to propagate from the input to the output of a gate. Pulses smaller than the *inertial delay* [Breuer 76] of a gate are not propagated to the gate output. A range of possible delays was used to take ambiguity into account.

The main interest in [Breuer 74b] was that delay faults could cause hazards that *invalidate* tests for asynchronous circuits. Fundamental mode operation was assumed, where the input cannot change until the circuit stabilizes. Techniques for considering or eliminating hazards in test generation and fault simulation were presented (also [Breuer 74a]). The concept of test invalidation due to hazards was important from the start of delay fault testing.

One of the first papers to treat delay testing the way it is currently done is [Hsieh 77], which is one of six papers describing an LSI test system developed at IBM. It is pointed out that restrictions must be placed on a design, LSSD synchronous logic in this case. The hardware model assumed is shown in Fig. 2.1, with the combinational circuit-under-test (CUT) surrounded by registers. Output Waveform Analysis will be described within this framework, although it can be used in any application where useful information can be extracted from the shape of the waveform. Separate input and output clocks are shown in Fig. 2.1, but the system being checked could have a single clock or multiple clocks.

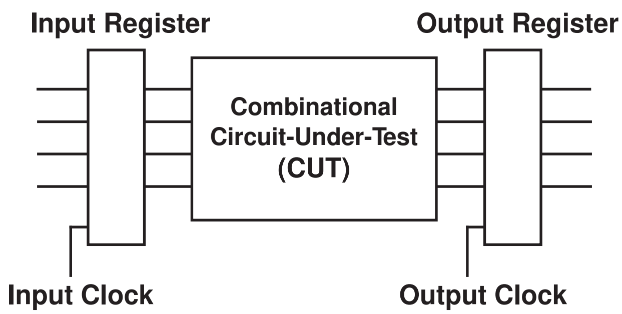


Figure 2.1. Hardware Model

The CUT inputs are either connected to register outputs or primary inputs, and the CUT outputs are either connected to register inputs or primary outputs. The longest propagation delay through the CUT in a fault-free circuit is less than the *cycle time* or *clock period*, T_C. The maximum clock rate is determined by timing analysis or timing verification [Hitchcock 82].

A single localized timing defect is assumed in [Hsieh 77], which causes one gate to operate slower than expected. This is now called the *gate delay fault* model. A delay test is performed by first initiating a transition at an input of the combinational logic, called the first timed action. The transition is propagated to one of the outputs, and the second timed action is the strobe that latches the CUT output at the specified time. The first work on delay fault simulation was reported in a companion paper [Storey 77]. Only delay faults greater than a certain size can be detected, unlike stuck-at faults that are simply detected or undetected.

Two vectors are needed to initiate the transitions necessary for detecting delay faults; the first vector is the value before the transition, and the second vector is the value after the transition. (Dynamic logic is an exception, as the precharge step acts as the first vector). Figure 2.2 shows a typical timing diagram for a delay test pattern pair <V_1,V_2>. The *initializing vector* <V_1> is applied and the transients in the circuit are allowed to settle, and then the *test vector* <V_2> is applied. After a timed interval equal to the cycle time T_C of the circuit, the CUT outputs are sampled and compared to the expected fault-free response to determine if the circuit is functioning correctly.

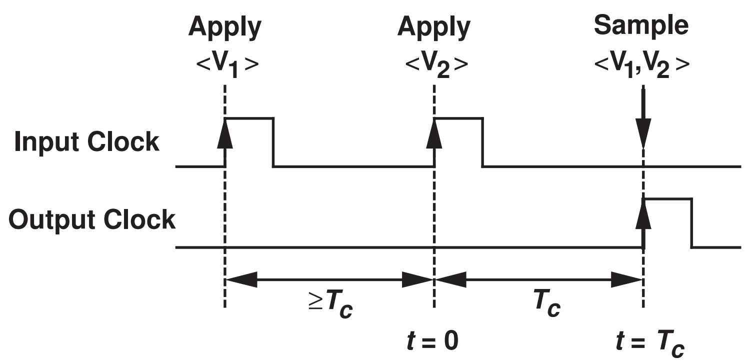
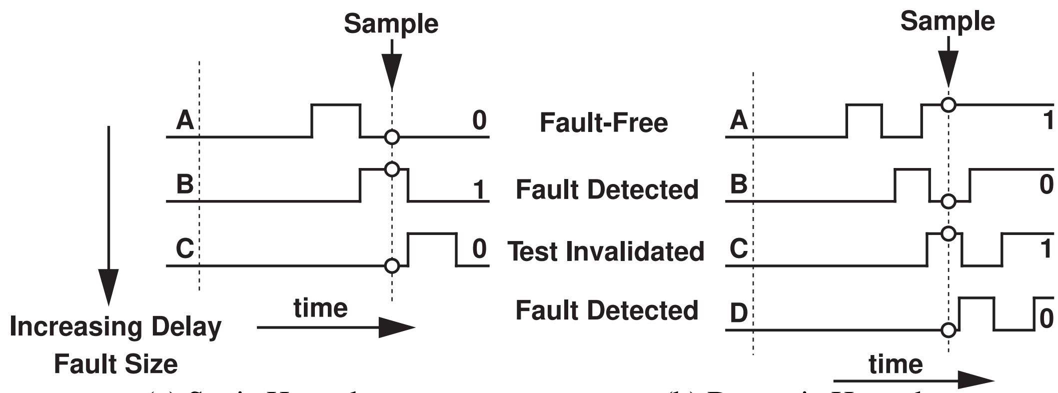


Figure 2.2. Waveforms for Delay Testing

The use of a "statistical" timing philosophy rather than a "worst-case" timing philosophy is explored in [Lesser 80]. The more aggressive timing philosophy takes advantage of statistical timing variations, and speeds up a circuit by using typical delays instead of worst-case delays to set the clock rate. Using statistical timing there are cases where the delay of every gate on a path is within specifications (i.e. no localized or gate delay fault), yet the cumulative delay along the path exceeds the cycle time. Since the objective of delay testing is to guarantee that the delay of the circuit falls within specification, distributed delay faults along paths through the circuit need to be considered. These timing defects are called *path delay faults*.

Test invalidation by hazards is a major consideration in delay fault testing. Essentially, a test for a delay fault can be invalidated by delays in other parts of the circuit. In general, a test that detects a delay fault of a certain size can miss faults that are both smaller or *larger*. For example, if a signal changes from 0 to 1 and back to 0 as shown in

Fig. 2.3(a), but is sampled before the pulse, the circuit will appear to be functioning correctly. Tests for delay faults can be invalidated by both function and logic hazards in the CUT. Although logic hazards can be removed by proper design, function hazards do not depend on the circuit implementation, and cannot be removed.



(a) Static Hazard (b) Dynamic Hazard

Figure 2.3. Test Invalidation by Hazards

A more concrete example of test invalidation is given below. Consider testing for a delay fault of size *d* at input P in the circuit in Fig. 2.4. The longest path through input P in the fault-free circuit is through the OR gate to output X, and has a propagation delay of 6 units. Assuming the cycle time $T_C = 7$, delay faults at P less than 1 unit are undetectable. For the given test pattern pair in Fig. 2.4, delay faults at input P between 1 and 3 units are detected at output X, while larger delay faults are undetected.

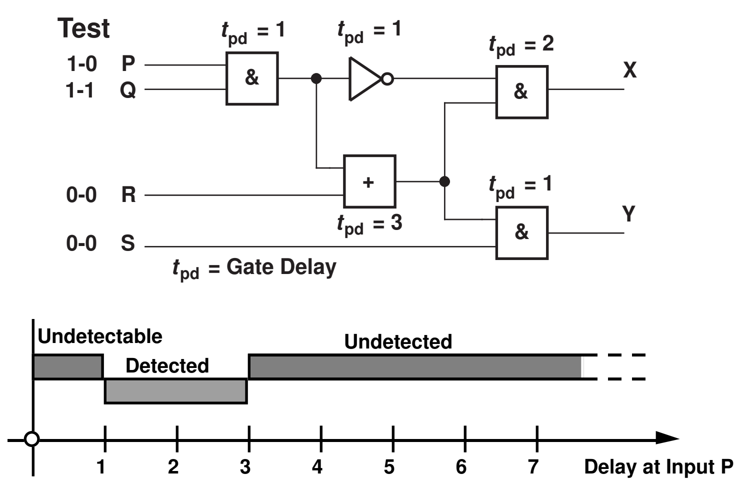


Figure 2.4. Delay Fault Detection can Depend on Size

Due to the possibility of test invalidation by hazards in stuck-open fault testing, the concept of "robust tests" was introduced by [Reddy 84]. Robust tests are tests that cannot be invalidated by delays in other parts of the circuit.

A six-valued algebra was presented in [Smith 85] for determining the path delay faults detected by an input vector pair. Faults were only considered detected if they are detected independent of the delays in the rest of the circuit. These tests were later called robust tests for delay faults [Lin 86] [Lin 87].

2.2 DELAY FAULT DEFINITIONS

Currently used definitions for delay faults are discussed in this section. Various definitions of delay faults have been used. Some of the differences are purely notational, while others depend on the assumptions made about the testing process. A timing failure is defined, followed by various definitions of delay faults.

Definition 1: Timing Failure

A component has a timing failure if the delay of the manufactured component is different from the designed delay.

The delay of a manufactured circuit could be either too long or too short. If the CUT output is not stable by the setup time of the latching element, then a *long path* delay fault has occurred. If the CUT output starts changing before the hold time of the latching element, then a *short path* delay fault has occurred. This is shown graphically in Fig. 2.5.

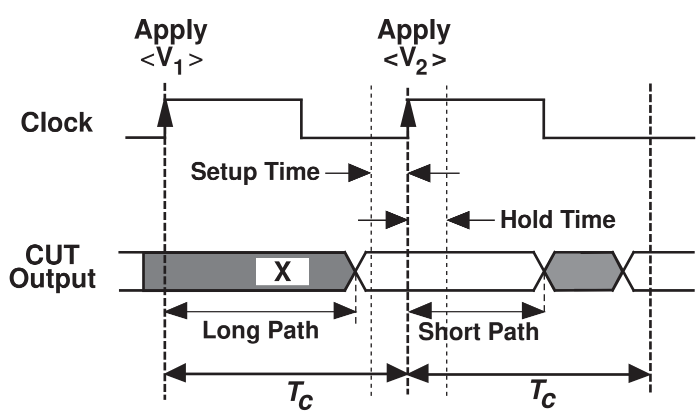


Figure 2.5. Short and Long Path Delay Faults

Although short paths are important and must be checked, most current work on delay testing focuses on long paths. Delay testing for long paths is be considered in this dissertation, although the techniques can be modified to detect short paths. Chapter 4 is the exception, where short paths need to be precisely controlled.

Delay faults can affect the propagation delay of both rising and falling transitions, or only single transitions. *Slow-to-rise* faults affect the rising transition, and *slow-to-fall* faults affect the falling transition.

Definition 2: Delay Fault

A circuit has a delay fault if it does not work at the cycle time T_C, but works at a slower speed. The goal of delay testing, then, is to guarantee that the circuit works at the designed speed (and lower speeds).

Definition 2 is too general to be practical, since it does not suggest a way to quantify the thoroughness of a particular test set. One approach would be to apply all possible input transitions to a circuit, and use that as a "reference test" for delay faults. A reference test is a test that would guarantee that the circuit was free of the faults under consideration. An exhaustive test where all input combinations are applied, for example, is a reference test for faults that change the logic function without causing sequential behavior (e.g. stuck-at faults). However, as shown in Section 2.4, some delay faults might need more than two patterns to be detected, so even applying all possible transitions might not be sufficient to fully test a circuit for delay faults.

Since definition 2 is not useful, two other definitions of delay faults are commonly used: the path delay model and the gate delay model. Before these definitions, false paths and timing slack are defined.

Definition 3: False Path

A path from the input to the output of a combinational circuit is a false path if it does not affect the operation of the circuit. This means that the path is not *sensitizable* under any timing conditions.

Definition 4: Slack

The slack of a path in a circuit is the difference between the cycle time and the propagation delay of the path [Hitchcock 82]. Similarly, the slack at a node is the difference between the cycle time and the propagation delay of the longest sensitizable path (i.e. excluding false paths) through the node.

Definition 5: Path Delay Fault

A circuit has a path delay fault if the propagation delay of at least one sensitizable path through the circuit exceeds the specified cycle time T_C.

Path delay faults can be either localized or distributed delay faults. The assumption is generally made that if the propagation of all paths is within specification, then the circuit is free of delay faults and will work at the designed speed. It is shown in Section 2.4, however, that path delay faults (as commonly defined) are only a subset of all delay faults.

Gate delay faults refer to localized timing failures in a circuit. These faults are usually modeled at gate inputs or outputs. Two distinct definitions are currently in use for gate delay faults, depending on the point of view taken. If a gate is considered individually, then there is a gate delay fault if the propagation delay of the gate exceeds its specifications. The problem with this definition is that it can be very difficult to perform this test when the gate is embedded in a circuit. The reason is that if there is slack at a node, then the sampling time will need to be changed to detect gate delay faults.

For example, assume that a gate has a specified delay of 1 ns and an actual delay of 4 ns, but the slack at the gate is 5 ns. It is not possible to determine that the gate has a timing problem without sampling the output early. Whether it is important to detect this failure, depends on the application. For high reliability applications it may be desirable to detect these failures, as they can be reliability detractors. *Non-cycle time* testing is sometimes done, where the pattern application is delayed [Iyengar 92], or the output is sampled early [Pramanick 89] [Mao 90a].

The definition for gate delay faults used in this dissertation is now more common, and is based on circuit operation at the specified speed. The term *delay flaw* is used to describe gates with excessive delays in circuits that work.

Definition 6: Gate Delay Fault

A circuit has a gate delay fault if a localized timing failure causes the propagation delay of at least one path through the circuit to exceed the specified cycle time T_C.

Definition 7: Transition Fault

A transition fault [Barzilai 83] or *gross delay fault* is a gate delay fault that is large enough to be tested using any path in the circuit through the fault site.

Definition 8: Delay Flaw

A circuit has a delay flaw if there is a timing failure but the circuit continues to work at the designed speed.

The notion of *fault detection size* [Pramanick 88] is used to quantify the size of gate delay faults that are detected by a test. The fault detection size of a fault is the smallest delay, such that all larger delays are detected. The smallest possible fault detection size for a gate delay fault is the slack at the fault site, which occurs when the fault is detected through the longest sensitizable path through the fault site.

One advantage of using Definition 6 for gate delay faults is that the term "fault" is associated with circuits that do not perform the intended function. Therefore gate delay faults are a subset of path delay faults, because only localized timing failures are considered.

Figure 2.6 shows the relationship between the different fault models. A complete test for transition faults is guaranteed to detect all stuck-at faults, for example. The reason is that the second vector in the transition fault test is a stuck-at vector for the corresponding node [Barzilai 83].

Stuck-At Fault ⊂ **Transition Fault** ⊂ **Gate Delay Fault** ⊂ **Path Delay Fault** ⊂ **Delay Fault**

Figure 2.6. Relation Between Delay Fault Models

The relationship between path delay faults and gate delay faults is similar to the relationship between multiple and single stuck-at faults. One difference is that multiple stuck-at faults are usually considered independent, whereas multiple timing changes are usually correlated as process variations tend to track across a chip.

2.3 REVIEW OF DELAY FAULT TESTING

Techniques for detecting delay faults are classified as either "direct" or "indirect" in this dissertation. Generally, direct techniques are explicitly based on delay faults, whereas indirect techniques target the underlying defect mechanisms.

2.3.1 Direct Approaches

Several alternatives have been proposed for modeling and testing for delay faults, ranging from simplified transition fault models, to synthesizing the circuit for delay fault testability. However all the methods involve tradeoffs, and many of the methods are not as yet used in practice due to their complexity. Some of the limitations of current delay testing methods that lead to this work are noted below.

A simple form of delay testing is to apply patterns to the CUT at system speed. This is called *at-speed* testing, and is usually only possible on fast ATE or using Built-In Self-Test. Although any vectors can be used, design verification vectors, pseudo-random, or weighted random vectors are probably most common. Pseudo-random testing has been successful for detecting transition faults [Waicukauski 87], but very long test lengths are needed to achieve high fault coverage of small delay faults [Savir 88]. The reason is that small delay faults need to be sensitized through long paths, and tests can be invalidated by delays in other parts of the circuit. Detecting each transition fault more than once has been suggested as a practical approach for improving the thoroughness of transition fault testing.

There are some subtle differences between at-speed testing and delay testing. The biggest difference is that for delay testing, after the initialization vector is applied, the circuit is assumed to have stabilized before the test vector is applied. Therefore "slow" and "fast" clocks are used for the test, which is not true in the case of at-speed testing. The disadvantage of at-speed testing is that certain initialization conditions cannot be assumed. For example, if a rising transition is propagated through a node that was expected to be low, but the node has not yet discharged fully, the transition will propagate faster than expected. On the other hand, dynamic issues such as ground bounce and capacitive coupling are better represented in at-speed testing than conventional delay testing. Some work has recently been done on at-speed delay tests [Pomeranz 92].

Process variations usually track across a die, and could cause distributed timing failures. This type of failure can be detected by using a ring oscillator as a process monitor to measure the overall speed of the die.

Algorithms have been developed for generating tests for both gate delay and path delay faults. Path delay testing is more thorough than gate delay testing, but the number of paths in a circuit can grow exponentially with the number of gates. There are non-enumerative techniques to estimate the path delay fault coverage [Pomeranz 94]. If distributed timing problems are expected to occur, testing using the gate delay fault model is not sufficient to determine if the circuit functions correctly, as the sum of distributed delays could exceed the cycle time on some paths through the circuit.

As noted earlier, test invalidation by delays in other parts of the circuit is a major concern in delay fault testing, and provided the motivation for the work in robust delay tests. This is particularly true for path delay fault testing where distributed delays are assumed, although non-robust tests are important for gate delay fault testing where only a single fault is assumed.

Hazard-free path delay tests cannot be invalidated by delays in other parts of the circuit, but these are only a subset of robust tests that cannot be invalidated [Lin 86] [Lin 87] [Park 87] [Savir 88]. A taxonomy of robust delay tests has been presented by [Reddy 87]. A general robust test can either contain hazards or be hazard-free, be single-path-propagating or multiple-path-propagating, and be single-input-changing or multiple-input-changing. (There can be paths in multi-level circuits that do not contain single-path-propagating robust tests, yet are robustly testable.)

Robust tests are desirable, but unlike the case for stuck-at faults, irredundancy does not guarantee robust delay fault testability. In fact, it has been found that robust tests do not exist for most faults in the multi-level ISCAS'85 [Brglez 85] combinational circuits tested [Kundu 88a]. Techniques have been presented for synthesizing robustly delay-fault-testable multi-level logic circuits [Kundu 88a] [Roy 89] [Devadas 90a,b] [Pramanick 90a,b]. The first method for guaranteeing 100% hazard-free robust fault coverage was based on repeated Shannon Decomposition [Kundu 88a], which resulted in a high area overhead. Algebraic factorization-based synthesis techniques have been proposed [Devadas 90a,b] [Pramanick 90a,b], since algebraic factorization preserves robust path delay fault testability.

In order to reduce the area required to make a circuit completely robustly path delay fault testable, validatable non-robust tests have been proposed [Reddy 87] [Devadas 92]. Essentially, under certain simplifying assumptions, a non-robust test can be considered robust if the delays in other parts of the circuit that could invalidate the test have already been robustly tested.

"Non-robust" tests can be generated for paths that do not have robust tests. However, it has been shown that there are non-redundant paths that may affect the circuit's timing, for which no non-robust tests exist either. These paths have been called "non-robust untestable" [Cheng 93]. A path that is statically sensitizable is non-robust testable, whereas a path that is only functionally sensitizable is non-robust untestable.

It was assumed in the above discussion that any sequence of two vectors could be applied to the combinational CUT. For sequential circuits, however, this is not necessarily the case. For stuck-at faults, scan design is used to reduce the sequential testing problem to a combinational testing problem, by making all CUT inputs controllable and CUT outputs observable. However, there are scan-path correlations that place restrictions on consecutive vectors, so not all two-pattern tests can be applied.

One solution to this problem is to used an "enhanced" scan chain that can store two values, but this is very expensive. Another solution is to use a "skewed-load" test [Savir 92], where the second pattern is shifted one bit from the first. It has been found that arranging latches to improve the input ordering can make a significant difference in the attainable fault coverage [Mao 90b] [Patil 92]. Synthesis of delay fault testable sequential circuits has been explored, using partial enhanced scan [Cheng 91], or state encoding [Pramanick 93], for example.

2.3.2 Indirect Approaches

The above approaches for detecting delay faults can be considered direct approaches, as delay faults are tested explicitly. There are also indirect testing approaches, which target the underlying defects. Indirect methods generally try to provoke faults by changing the testing environment, or increase observability by using alternative observation strategies. While these methods were not necessarily developed for delay fault testing, their ability to detect delay faults has been investigated due to difficulties with the direct delay testing approaches.

The most common indirect testing approach is quiescent current monitoring, or IDDQ testing [Hawkins 89] [Levi 81]. CMOS circuits draw very little static power, so any defect that causes a current path in the circuit can be detected by monitoring the quiescent current. Common examples are defects such as bridges and gate oxide shorts. Some delay faults can be detected, since certain defects that cause delay faults also increase the quiescent current.

Although IDDQ testing is a very successful technique for detecting shorts, it is not a complete method, since only some of the failure mechanisms that cause delay faults are detected. For example, RC interconnect delay, which can be significant in CMOS, cannot be detected. IDDQ testing is also more difficult in technologies (e.g., TTL, ECL) where the quiescent supply current is inherently high.

Other, more exotic, indirect techniques have also been proposed. Embedded testing provides massive observability [Gheewala 89]. Very-Low-Voltage Testing [Hao 93] tries to provoke weak parts to fail. Transient current testing is similar to IDDQ testing, but the dynamic current is measured [Dorey 90]. Static or transient current noise testing has also been proposed, based on the assumption that faulty parts will have more noise [Dorey 90].

All the indirect methods provide some measure of delay fault coverage; this might be sufficient for certain applications, but the coverage in incomplete, as parts that pass the tests cannot be guaranteed to operate at the designed speed. Speed binning, for example is difficult to do accurately with the indirect approaches since the longest delays through the circuit are not tested.

2.4 INACCURATE DELAY MODELING

This section focuses on the effect of inaccurate delay modeling on delay fault testing. This work was presented at the 1994 VLSI Test Symposium [Franco 94b].

The delay properties of gates have been greatly simplified in delay test generation. As fault models become more realistic, their complexity increases. It is clear that simple delay models do not accurately represent the delay properties of gates -- after all, the purpose of models is to hide the circuit complexity. The important question is not how accurate the models are, but whether the models are *adequate* for generating tests that detect defective circuits with reasonable cost.

It is shown that accurate delay models are needed for effective delay fault testing. This is particularly important for large timing-optimized circuits with many paths. The reason is that there are too many paths to test in general, so only the longest or *critical* paths are often tested. Therefore it becomes important to choose the actual longest paths.

Limitations of the path delay fault model are shown. Even the assumption that two-pattern tests are sufficient for delay testing is shown to have limitations.

Modeling gate delay is described first, and the problem with using a simple delay model is then shown by simulating a 2-input gate. An experiment is described and possible solutions to the problem are discussed.

2.4.1 Modeling Gate Delay

Stuck-open and bridging fault models are more complex than stuck-at fault models. Furthermore, work has been reported on making these models more accurate. Tests for stuck-open faults have been extended to take into account both hazards [Reddy 84] and charge sharing [Barzilai 86]. Similarly, wired AND or OR bridging faults have been extended by the voting-model [Aiken 88] and biased voting [Maxwell 93]. Complex behavior such as "pattern dependence" has been described [Hao 91].

Work on delay fault testing has concentrated on the generation of delay tests and making circuits robustly path delay fault testable, but the underlying model assumed in test generation, however, is fairly simple. The normal assumption is that the propagation delay of a gate can be different for each input, rising and falling delays can be different, and is affected by output loading due to interconnect and input capacitance of the following gates.

In reality, the propagation delay also depends on parasitic capacitance and dynamic factors such as capacitive coupling and ground bounce. A classification of propagation delay models of increasing accuracy is presented in Table 2.1, for a transition at an input of the gate to propagate to the gate output. This classification is an extension of the list presented in [McCluskey 62].

Table 2.1. Delay Fault Models

Level	Propagation delay dependence
0	None -- (zero delay model)
1	None -- (unit delay model)
2	Gate type
3a	Level 2 & loading
3b	Level 2 & rising or falling transition at input
3c	Level 2 & input
4	Level 2 & input, transition at input, loading
5	Level 4 & state of other inputs
6	Level 5 & transitions at other inputs
7	Level 6 & state of gate
8+	More complex...

The three Level 3 models are of the same complexity and grouped together. The commonly used model for delay testing corresponds to Level 4. It is shown below that Level 7 behavior exists even in simple 2-input gates.

There are generally too many paths to test, so the longest paths are chosen for path delay testing. Testing multiple paths together was proposed in [Pramanick 91], and recent results show that, on average, five paths can be tested per vector [Bose 93] [Saxena 93]. This reduction helps, but is not enough to make testing all paths feasible for many large circuits. For example, there are 9.89×10^{19} structural paths in the C6288 circuit in the ISCAS'85 benchmark suite. In fact, for one of the outputs, there are 6.44×10^{10} paths with the maximum delay (using the Level 1 delay model).

In timing-optimized circuits, there will be many paths close to the maximum delay [Williams 91] [Park 91]. This means that even if there is a small error in the delay modeling, it could turn out that few of the actual longest paths are tested, reducing the effectiveness of the test. After going to the expense of path delay testing, it is a waste to then test the wrong paths! The situation is even worse for gate delay testing, as only one of the many paths through each gate is tested.

2.4.2 Three-Pattern Delay Tests

It is generally agreed that for static logic, delay testing requires a two-pattern test $<V_1,V_2>$. The *initializing vector* $<V_1>$ is applied, and enough time is allowed for the transients in the circuit to settle. Then the *test vector* $<V_2>$ is applied, and after a timed period equal to the *clock cycle*, T_C, the circuit outputs are sampled. These two-pattern tests do not provoke the longest delay, however, as the example below shows.

The simplest 2-input gate is used as the first example. Since realistic values are needed for parasitic capacitance, the 2-input NAND2 gate was laid out using MAGIC, based on the cell design in the CMOS3 standard cell library [Heinbuch 88]. The circuit was extracted and simulated using SPICE. Figure 2.7 shows the simulation setup; the inverters were also laid out.

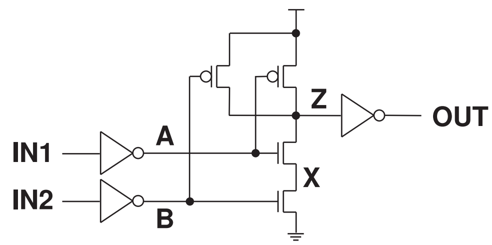


Figure 2.7. NAND2 Gate Simulated

The delay for a rising transition at the NAND2 output depends on whether one or both pull-up transistors are active. The worst-case delay occurs when only one transistor is on. However, to avoid test invalidation, only one transistor is turned on at a time in delay testing (need the non-controlling value on the other input), so there are no problems for rising transitions. The output falling transition (Z: 1→0) is more complex, however. Consider input rising transitions at the NAND2 gate inputs A and B. Based on the SPICE simulations, the delays are:

Delay of rising transition in A:	(A,B): (0,1) → (1,1) = 0.796 ns
Delay of rising transition in B:	(A,B): (1,0) → (1,1) = 0.808 ns
Delay of rising transitions in A and B:	(A,B): (0,0) → (1,1) > 1 ns

This simple example shows that double input changes must be considered; the worst-case delay gate delay is almost 32% greater than the worst single-input-change delay. This is of importance for pseudo-exhaustive adjacency testing [Craig 85], for example, where vectors with single input changes are produced.

With both inputs changing, the gate has state. When the both A and B are 0, node **X** in Fig. 2.7 could be either high or low. Therefore there are two possibilities:

Precharge **X** low:
(A,B): (0,1) → (0,0) → (1,1): Delay of transition (0,0) → (1,1) = 1.003 ns

Precharge **X** high:
(A,B): (1,0) → (0,0) → (1,1): Delay of transition (0,0) → (1,1) = 1.065 ns

The example shows that the state of the gate before the test must be taken into account (Level 7 delay model), as one of the possibilities has a 6.2% greater delay than the other. This difference might seem small, but this is for a simple gate, and every gate in the circuit can exhibit this type of behavior. In timing-optimized circuits with many paths, it is possible that only paths close to the length of the longest path are tested, so a 6.2% inaccuracy is significant. *Three-pattern delay tests* are required to set up the state of the gate and then launch an input transition.

The physical causes for complex delay behavior are based on parasitic capacitance, and include charge sharing, body effect, and bootstrapping [Weste 85]. The behavior observed above is due to charge sharing between the output node of the NAND2 and node **X**. It is not possible to eliminate this behavior, although it becomes relatively less severe if there is a large capacitance connected to the output node. Figure 2.8 shows another phenomenon observed in the 2-input NAND2 gate. The voltage at node **X** drops below ground due to bootstrapping, then rises to 1V before settling at 0V. This behavior is common in more complex gates with a number of internal nodes.

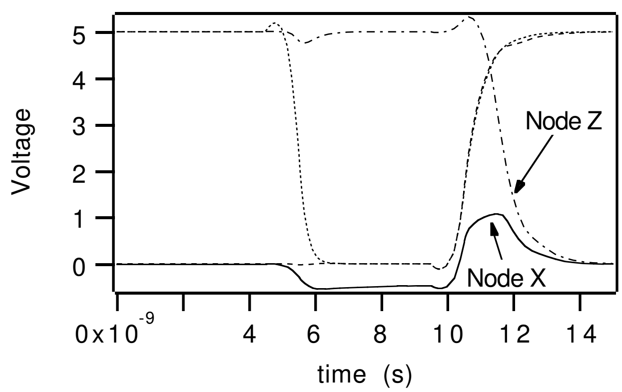


Figure 2.8. Simulation of (A,B): (0,1) → (0,0) → (1,1)

The SPICE simulations described above were repeated for a 3-input NAND gate. Once again, the NAND3 from the CMOS3 library was laid out and the circuit parameters were extracted. A summary is presented in Table 2.2. The 3-input NAND has a 7.2% propagation delay difference between having the two internal nodes charged or discharged.

Table 2.2. 3-input NAND3 Delays

Transitions	Delay (ns)
(0,1,1) → (1,1,1)	2.596
(1,0,1) → (1,1,1)	2.671
(1,1,0) → (1,1,1)	2.864
(0,1,1) → (0,0,0) → (1,1,1)	3.415
(1,1,0) → (0,0,0) → (1,1,1)	3.660

A complete path delay test for the NAND2 gate, for example, will not necessarily include the sequence (1,0) → (0,0) → (1,1), or even the pair (0,0) → (1,1). This means that the longest propagation delay through the gate will not be exercised by using the path delay model. Therefore path delay faults are only a subset of all delay faults, and even if all paths in a circuit are tested for delay faults, the circuit might still not operate at the desired speed.

2.4.3 Need For Three-Pattern Tests

The significance of the behavior described above depends on how common it is, since it is not always possible to change both inputs at exactly the same time for gates embedded in a circuit. In this section, it is first shown that the inputs do not have to change at exactly the same time, and then it is shown that multiple input changes are possible for a large fraction of the gates in the combinational benchmark circuits investigated.

Figure 2.9 shows that the inputs don't have to change at exactly the same time to cause an increase in the propagation delay of the gate. The vertical axis is the combined delay of the input inverters and NAND2 gate. This was done as the inverter delay also changes slightly depending on the state of the NAND2 gate.

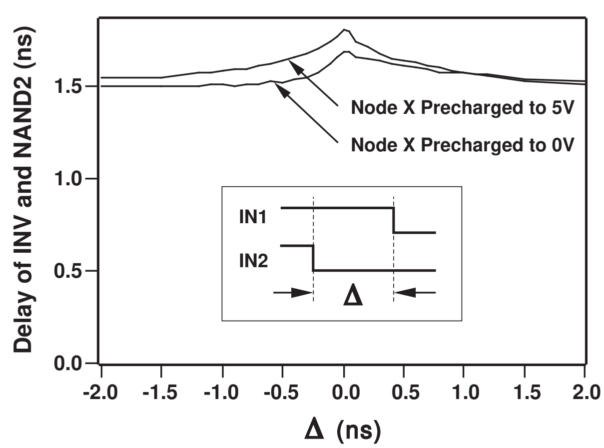


Figure 2.9. Delay of INV and NAND2 as a Function of Shift in Input Transitions

The horizontal axis is the difference between the fall time of inputs IN1 and IN2 to the circuit. The latest transition at the circuit inputs is at time 0. Node **X** is charged to 5V in the top curve, and discharged in the bottom curve. The dynamic effect of multiple input transitions is still visible even if there is one gate delay (approx. 1 ns) between transitions.

In order to estimate the likelihood of having both inputs to a gate changing together, the ISCAS'85 circuits were analyzed. For every multi-input gate in a circuit, the longest path to each input was computed, and the fraction of gates with the same longest path for two inputs was recorded. This is useful because when testing the longest path through a gate, it is desirable to have a transition at the other gate inputs. This analysis is only an approximation, as false paths are not considered, and the two long paths might not be sensitizable together. Table 2.3 shows that up to half the gates can have the same longest path for two inputs. The Level 2 delay model was used.

Table 2.3. Percentage of Gates with Equal Longest Paths for Two Inputs

Circuit	Multi-Input Gates	Percentage of Gates
ALU181	53	50.94%
c432	120	5.00%
c499	162	45.68%
c880	294	29.25%
c1355	474	37.55%
c1908	441	20.86%
c2670	676	24.85%
c3540	956	18.10%
c5315	1413	21.30%
c6288	2384	34.61%
c7552	2102	16.13%

2.4.4 Experiment

The problem of determining the modeling accuracy required for delay fault testing can also be approached experimentally. Ideally, the following experiment would be performed: Choose a combinational circuit that is robustly path-delay-fault testable, and apply two test sets to the circuit, measuring the propagation delay for each delay test vector pair applied. The first test set is a complete robust path delay fault test set generated using the Level 4 delay model, and the second test set is a reference delay test. The reference test should be very thorough; at least a super-exhaustive ($2^n(2^n$-1)) test if possible. By comparing the maximum propagation delay through the circuit by both the robust test and the reference test, it can be seen whether the Level 4 modeling of delay is adequate to provoke the longest path through the circuit. If not, then a more accurate delay model might be necessary.

Unfortunately, such an experiment is difficult to do and has not been reported in the literature. There is some data from an experiment on the effect of high IDDQ on circuit reliability [Hao 93]. This data is not conclusive, but does give some indication of the problem. More conclusive data is presented in Chapter 6, based on measurements taken from the Test Evaluation Chip Experiment.

The circuits used in [Hao 93] were 148, 74AC138 static CMOS 3-bit decoders (with 3 select inputs) from National Semiconductor, that passed all production tests except the IDDQ test. This is a mature process, and the data below was taken after 168 hours of burn-in.

The delay test set consisted of 232 two-pattern tests. A subset of 68 two-pattern tests was selected which is a complete robust path delay test for the decoder. (Most of the robust tests were single-input-changing, and the Level 4 model was used.) The propagation delay for each test pair was measured on a Sentry 21 tester. Figure 2.10 shows the maximum delay for the complete delay test set ("All Tests") and the robust subset of the test ("Robust Tests").

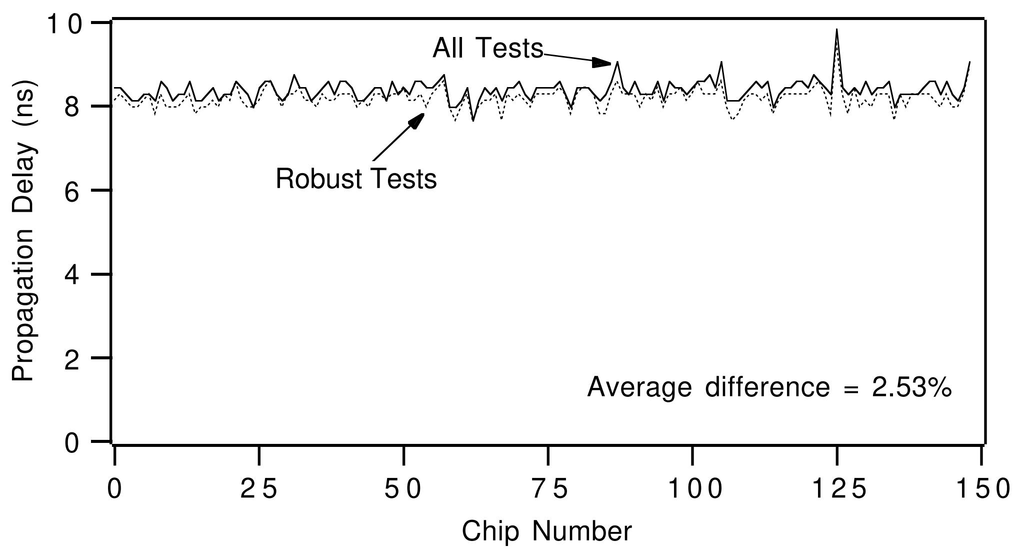


Figure 2.10. Graph of Maximum Propagation Delay of All Tests and Robust Tests
© IEEE [Franco 94b]

The robust delay tests underestimate the maximum delay by 2.53% on average for this experiment. This seems significant, given that the circuit is only 2-level with no reconvergent fanout, and the I/O buffers probably account for a substantial fraction of the delay. Furthermore, the reference test was only 232 two-pattern tests, whereas it should have been at least 4,032 to test every transition, or longer for three-pattern tests.

The worst-case possible accuracy of the tester must be taken into account. One of the chips was tested three times at the beginning and end of the test to check the repeatability of the tester. Results of the average difference in delay between the first and second, and second and third tests are shown below. (There is also warming of the chip to consider, as the IDDQ portion of the test is fairly long). The tester repeatability was within 0.2% except for one case where it was 1.1%.

	from 1 to 2	from 2 to 3
beginning	0.23%	0.16%
end	1.10%	-0.04%

2.4.5 Possible Solutions

There are several approaches for dealing with the limitations of currently-used delay models. The simplest approach is to try to avoid the problem by estimating the worst-case error introduced by the simple delay model. Extra tolerance is then added between the test speed and actual system operating speed, beyond the normal tolerance allowed for variations in operating conditions and processing factors.

Another approach is to use many patterns that provoke high node activity in the circuit (e.g. weighted-random) in an attempt to sensitize the longest delay through the circuit.

A more direct approach is to test multiple paths together. Testing multiple paths is useful as a first step since by increasing the activity in the circuit, it is more likely to excite the longest delay through the circuit. Delay test pattern generators can be constrained to try to generate multiple input changes for all gates along a path, and only relax the constraints if the conditions cannot be met. Figure 2.11 shows part of a CUT being tested. The side inputs to the path-under-test must change for the maximum possible delay. In general, it is not possible to change all the side inputs, so the number of side inputs that change at the same time as the path-under-test must be maximized. The previous patterns applied can also be inspected to determine the state of the gates along the selected paths.

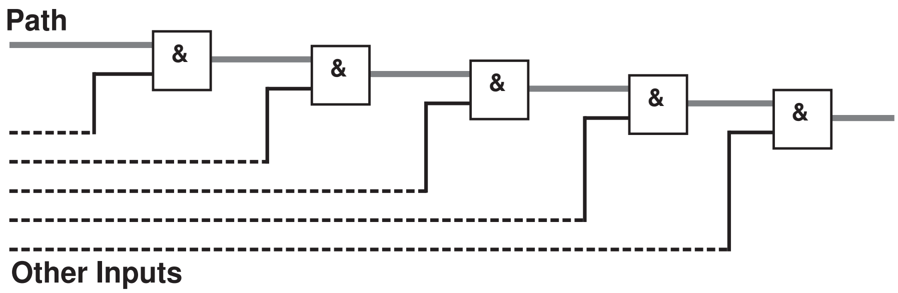


Figure 2.11. Path Under Test

As shown below, Output Waveform Analysis can be used in conjunction with the improved test sets suggested.

2.5 OUTPUT WAVEFORM ANALYSIS

Both direct and indirect techniques for detecting delay faults were summarized in Section 2.3, and limitations were discussed. In general, the direct approaches focus on the application of particular patterns and timing, whereas the indirect approaches are more focused on the observation of the response of the circuit. It has been shown in the previous section that even in a best-case scenario where there are not too many paths to test, and all paths are robustly path delay fault testable, modeling inaccuracies can still limit the effectiveness of a delay test. In practice, the situation is worse since not all paths will be tested, robust tests might not exist, and there are often restrictions on the patterns applied (e.g., pseudo-random patterns during BIST).

Output Waveform Analysis, the technique proposed in this dissertation, is a combination of direct and indirect approaches. As noted in Chapter 1, it is direct in that delay faults are explicitly considered, yet is similar to the indirect approaches in that the focus is on the observation of the circuit output response.

Conventional delay testing is complicated by the fact that only a single sample of the output waveform is taken. Instead of only sampling the output waveform, we propose to look at the output waveform between samples, so any changes can be used to detect delay faults. The premise is that there is significant information in the output waveform between samples to justify the added complexity of looking at the waveform.

The justification for the method is that, unlike catastrophic failures that simply have incorrect steady-state logic values at the circuit outputs, delay faults change the shape of the output waveforms of the circuit by moving the signal transitions in time. Therefore, since the output waveforms contain information about the circuit delays, instead of only latching the outputs at the sampling time, the output waveforms between samples are analyzed as well.

We propose to perform delay testing by analyzing the output response "continuously". Test patterns are applied with the same timing as conventional delay testing, and the CUT output is sampled as in conventional delay testing. Circuits are added to monitor the output waveform between samples. It is not feasible to store the entire waveform between samples, so some scheme for compacting the information between cycles is necessary. There are two classes of waveform analysis functions, depending on whether the output waveform before or after the sampling time is analyzed. These are illustrated in Fig. 2.12, and will be called *Pre-Sampling Waveform Analysis* and *Post-Sampling Waveform Analysis* respectively.

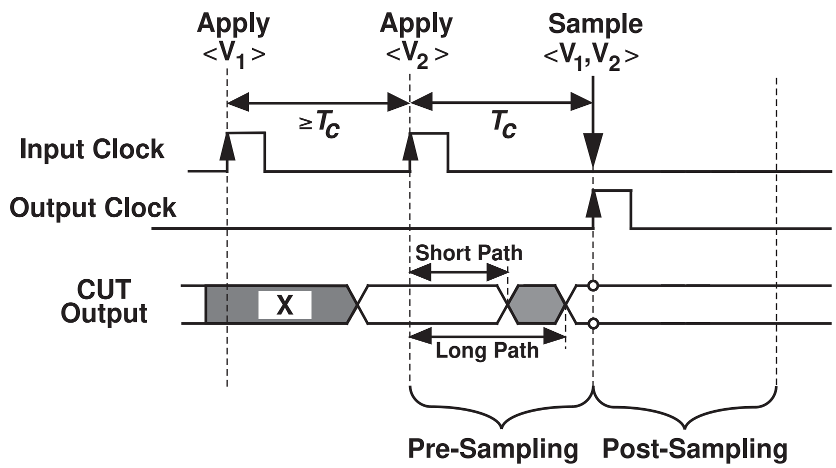


Figure 2.12. Output Waveform Analysis

Output Waveform Analysis attempts to overcome some of the limitations and difficulties of traditional delay fault testing, particularly for BIST and on-line delay testing applications. Post-Sampling Waveform Analysis is the simpler method, and reduces invalidation of tests for delay faults due to hazards. Pre-Sampling Waveform Analysis is more complex, but the method can also be used to detect delay flaws and screen weak parts that do not yet have delay faults. This is useful for environmental stress screening and predicting component wear-out in the field.

The rest of this dissertation will describe different forms of Output Waveform Analysis, and show that the techniques are useful and practical. Post-Sampling Waveform Analysis is described in Chapter 3, and its feasibility for on-line checking is discussed in Chapter 4. Pre-Sampling Waveform Analysis is covered in Chapter 5.

Chapter 3

Post-Sampling Waveform Analysis

Delay testing by Post-Sampling Waveform Analysis is described in this chapter. The test mode architecture and circuits for performing the waveform analysis are presented to show that the method is feasible, and examples are given. Combining Post-Sampling Waveform Analysis and conventional Built-In Self-Test (BIST) is also discussed. Chapter 4 covers on-line delay testing, an important application of Post-Sampling Waveform Analysis.

3.1 DESCRIPTION

Output Waveform Analysis is based on the premise that the waveforms at the circuit outputs are different for faulty and fault-free circuits, and this information can be used to detect timing failures. In Post-Sampling Waveform Analysis, the output waveform *after* the sampling time is observed.

If the CUT is operating as designed, all pathlengths are shorter than the cycle time, so the outputs will be stable by the time they are sampled and latched. If the CUT has an error due to a delay fault, however, then at least one output must not have settled at the correct logic value by the sampling time (waveforms B-F in Fig. 3.1). This faulty output then changes to the correct value after the sampling time. Therefore, delay faults are detected in Post-Sampling Waveform Analysis by continuously observing the CUT outputs for any changes after the sampling time.

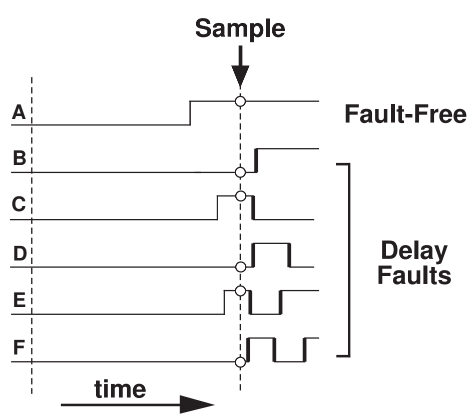


Figure 3.1. Waveforms with Delay Faults

Note that if the output waveform is unstable at the sampling time, the sampled value may or may not be correct. If the sampled value is incorrect (waveforms B, C, and F in Fig. 3.1), then the delay fault can be detected from the sample alone, whereas if the sample is correct, the conventional delay test has been invalidated (waveforms D and E in Fig. 3.1).

Observing the output waveform for any changes after the sampling time will be called *Stability Checking*. It is the "best" Post-Sampling Waveform Analysis technique in some sense, since no information in the output waveform is lost. *Final-Value Checking* is another technique that can be useful for on-line delay testing described in the next chapter. Pre-Sampling Waveform Analysis is more complex, on the other hand, and there are many useful waveform analysis techniques.

Figure 3.2 shows the timing diagram for Stability Checking. The initializing vector $\langle V_1 \rangle$ and test vector $\langle V_2 \rangle$ are applied as in conventional testing. The interval from $t=T_C$ to $t=T_C+T_{Stab}$ during which signals are checked for stability, is the *checking period*. The vector $\langle V_3 \rangle$ following $\langle V_2 \rangle$ is not applied during the checking period for $\langle V_1,V_2 \rangle$, since the output of a good circuit must remain stable after the sampling time, and changes due to $\langle V_3 \rangle$ would be erroneously flagged as delay faults.

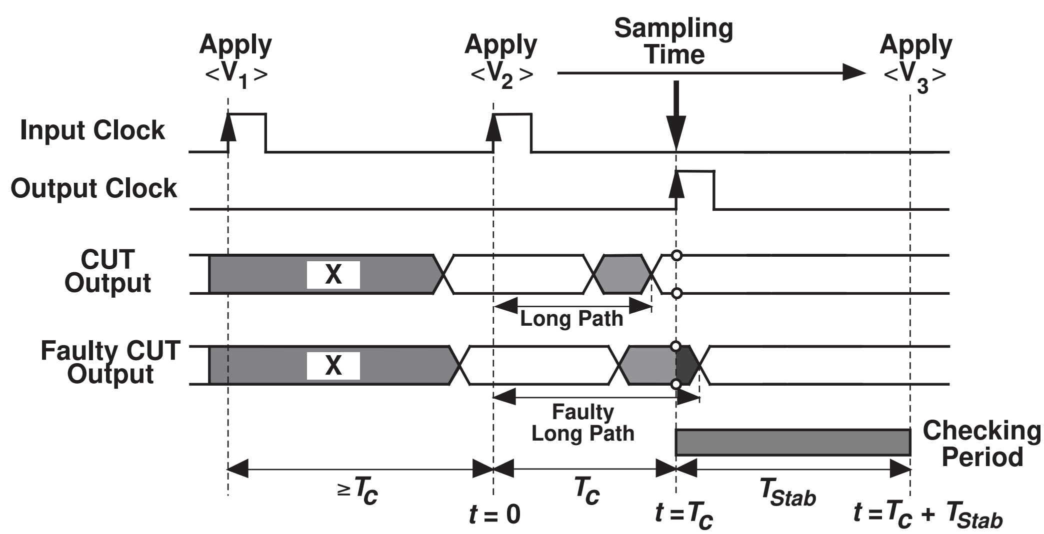


Figure 3.2. Stability Checking Waveforms

A faulty CUT output where the propagation delay of the longest path exceeds the cycle time is shown in Fig. 3.2. Changes during the checking period are detected by *stability checkers* that observe the output waveform.

The duration of the checking period depends on the largest delay fault under consideration, as delay faults longer than the duration of the checking period are not guaranteed to be detected using Stability Checking. For example, if the fault-free delay of a path is close to the cycle time T_C, and the path has a delay fault larger than the checking period, then the output will only start changing after the end of the checking period and thus not be detected. (Gross delay faults [Savir 88] larger than the clock cycle might be detected during the next checking period, but one cannot rely on this.)

Digital automatic test equipment (ATE) can be used for stability checking by using window strobe or window comparison [Parker 87], but only the chip I/O pins can be checked in this way. Since Built-In Self-Test is an important application of Stability Checking, and many internal signals need to be checked, the focus here will be on efficient stability checker designs that can be incorporated on-chip.

The function of the stability checkers is described in Table 3.1. If any change occurs in the circuit outputs during the checking period, an ERROR signal is set. The checkers need to be reset on startup, and after every cycle if dynamic logic is used or the test is retried.

Table 3.1. Functional Description of Stability Checker

```
For each output of CUT, Di:
   if (any change in Di)
       AND (Checking Period = 1)
         then set ERRORi := 1

Before Start of Next Checking Period:
   set ERRORi := 0
```

The rest of this chapter covers the implementation of Stability Checking, including the test mode architecture and design of several stability checkers. Some experimental results concerning hazards are then given.

Some of the main features of Stability Checking are:

1. Stability Checking is independent of the test vectors used, and can improve the delay fault coverage for any set of vectors. The greatest improvement over traditional testing is in situations where the input patterns cannot be controlled and test invalidation by hazards is a problem, such as pseudo-random testing.

2. Since multiple transitions at the outputs are detected, tests that were originally not robust for delay faults cannot be invalidated by pulses appearing in the output waveform.

3. There is no latency in detecting faults, because as soon as a change is observed during the checking period, it is known that the circuit is faulty. This is usually not possible in conventional BIST techniques, where the actual circuit response is compacted and compared to the expected response at the end of the test.

3.2 IMPLEMENTATION

This section is not intended to cover all possible implementations of Stability Checking, but rather to show that it is practical to implement Stability Checking, even for BIST applications.

Design considerations are discussed first, followed by test architectures that provide the correct clocks and control signals to perform the Stability Checking, and the design of the stability checkers themselves. Integrating Stability Checking with other BIST testing is described in Section 3.3.

3.2.1 Design Considerations

Implementing Stability Checking involves applying vectors with the correct timing, providing a signal to mark the checking period, and collecting the error response from the individual stability checkers. It is also necessary to reset the stability checkers, and in some implementations, provide a Test Mode signal.

Clocking: The clocking needs to be modified to ensure that new vectors are not applied to the CUT during the checking period. This is easily achieved in two-phase double latch designs [McCluskey 86], since the input and output clocks shown in Fig. 2.1 in the previous chapter can be independently controlled. In single clock designs, the same clock is used for applying vectors and observing the response. Either the test vector source is modified to hold $<V_2>$ or produce the sequence $<V_1,V_2,V_2>$, or the clock itself is slowed down.

Checking Period: The checking period signal must be precisely controlled with respect to the clock. One solution is to distribute the Checking Period signal as another clock. A better solution is to define the Checking Period as a function of the clock.

Test Mode Signal: Depending on the design of the checkers and the clocking scheme used, a signal might be necessary to put the circuit in delay-test mode. If BILBO-type registers are used in a design, then an unused control combination can be used.

ERROR Signal: The $ERROR_i$ output of the stability checker signals the detection of a delay fault. Depending on the design of the checkers, it could be a pulse or latched output. The individual $ERROR_i$ signals need to be combined to produce a global ERROR signal. Depending on the diagnostic resolution desired, a single ERROR signal can be generated by ORing all the $ERROR_i$ signals, or only certain $ERROR_i$ signals are grouped together. The extreme is to latch each individual $ERROR_i$ signal.

Reset Signal: It is necessary to reset the analyzer during startup, and if the test is to be continued after a fault has been detected. It might be desirable to repeat the test after an error to determine if the error is temporary or permanent. Dynamic stability checkers need to be reset before every vector to ensure that voltages are not degraded.

3.2.2 Timing Diagrams

It is now shown how the above requirements can be met without significant hardware overhead for the two most common clocking schemes for synchronous sequential circuits. The first is single-phase edge triggered flip-flop design, and the second is two-phase double latch design [McCluskey 86].

Flip-flop Designs

In this test architecture, the system clock has been slowed down rather than modifying the pattern generator. During testing, the system clock is used to define the checking period, as well as to apply patterns and sample outputs. The system clock is held high for T_C, and is held low for the duration of the checking period, T_{Stab}. This implementation is only possible in cases where the system clock can be controlled.

The duration of the checking period can be chosen to simplify the clock modifications needed. One solution is to make the checking period equal to the normal cycle time, i.e. T_{Stab}=T_C. Now the system clock just needs to be divided by two during testing. This can be done either on-chip or off-chip with a simple circuit.

The timing diagram for stability checking with a single phase clock is shown in Fig. 3.3. The first vector <V_1> in the test pair is applied at the rising edge of the system clock at time ***A***, and the second vector <V_2> is applied at time ***B***. The checking period is defined by the system clock being low, and starts at time ***C*** and ends at time ***D***. No new inputs are applied to the CUT during the checking period, as required.

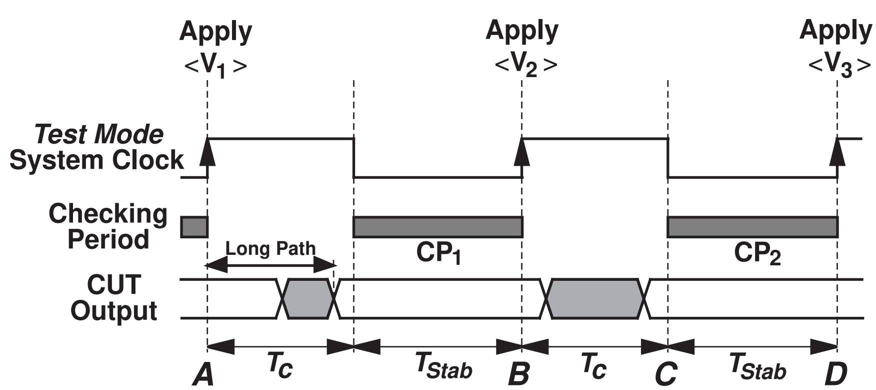


Figure 3.3. Timing Waveforms for Flip-Flop Designs

Note that the second vector in the test, <V_2>, is the initialization vector for the next pair of patterns. The tests are pipelined by overlapping the checking period for one test with the application of the first pattern of the following test. For example, the checking period CP_1 is overlapped with the application of the two-pattern test <V_1,V_2>.

No changes in the clocking of the registers are required during testing. Using the clock to define the checking period both reduces the cost of stability checking since no extra checking period signal is necessary, and reduces skew in the checking period since the clock distribution is well controlled. If there is clock skew, the checking period is also skewed, so that stability checking starts at the right time relative to the sampled output. The beginning of the checking period needs to be accurately synchronized with the flip-flip setup time for effective Stability Checking (as we found out from the Test Evaluation Chip Experiment described in Chapter 6).

Two-phase Double Latch Designs

The timing requirement is easy to implement in two-phase double latch designs, since there are already two independent clocks in the circuit that control the application of patterns (C2, with L2 latch), and the sampling of CUT outputs (C1, with L1 latch). The block diagram is shown in Fig. 3.4, and the relative timing of the clocks in test mode is shown in Fig. 3.5. During testing, the desired checking period is achieved by delaying the application of C2.

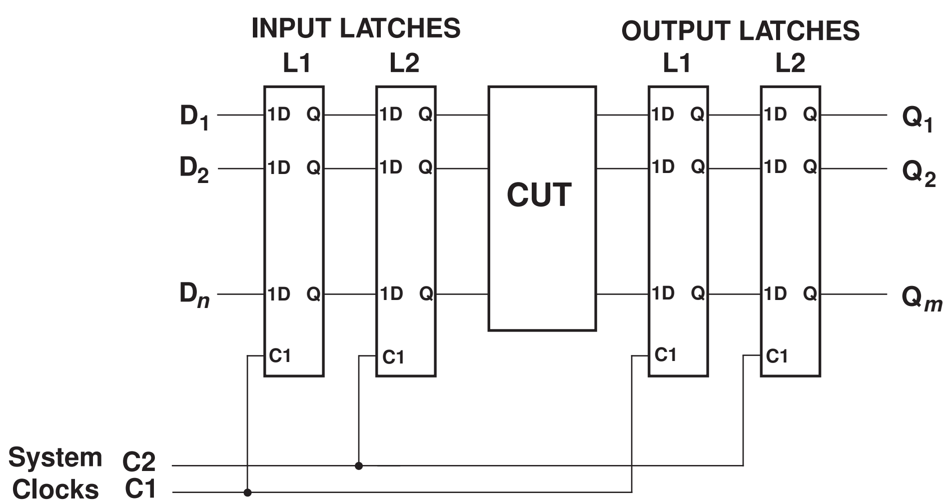


Figure 3.4. Block Diagram for Two-Phase Double Latch Designs

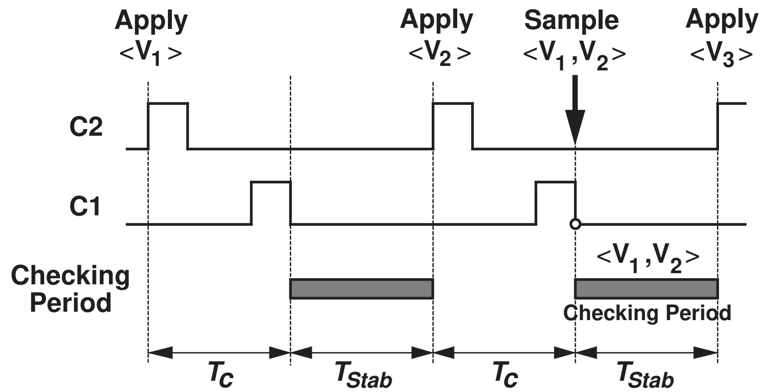


Figure 3.5. Test Mode Timing Waveforms for Two-phase Designs

The checking period for double latch designs is not simply one of the clocks as in the flip-flop design, but it can be derived from C1 and C2 using the fundamental mode circuit shown in Fig. 3.6. The checking period starts at the falling edge of C1, and ends at the rising edge of C2.

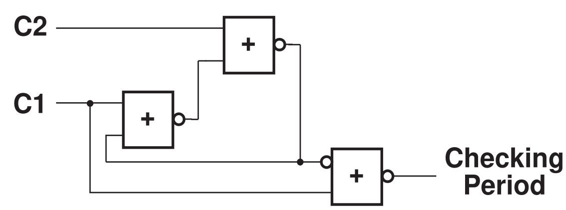


Figure 3.6. Generating Checking Period for Two-phase Designs
© IEEE [Franco 91b]

3.2.3 Architecture for Stability Checking

The block diagram for Stability Checking is shown in Fig. 3.7. Single clock designs will be used in the following discussions, as there are fewer constraints if the application of patterns and the output sampling can be controlled separately, making the implementation of Stability Checking simpler. The CUT output is named **D**, and the sampled CUT output is named **Q**, due to their relation to the system flip-flop. The checking period is denoted by **CP**.

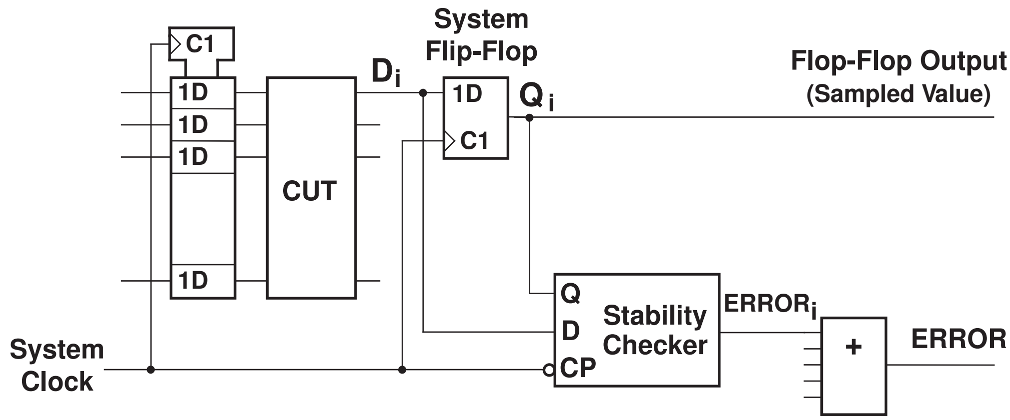


Figure 3.7. Block Diagram of Stability Checking Architecture

3.2.4 Stability Checker Implementations

Various stability checker designs are presented in this section. The designs have been simulated using SPICE, and two of the designs have been manufactured using the Stanford BiCMOS process. One design has been used in the Test Evaluation Chip Experiment described in Chapter 6. Each Test Chip contains 216 Stability Checkers.

Since stability checkers could be added to every flip-flop in a design, it is important to minimize their implementation cost. CMOS circuits are considered in this work. There are three main considerations in the design of stability checkers:

1. Static or Dynamic Checkers. Since delay testing is intrinsically based on timing, dynamic checkers can be designed. The advantage is that dynamic checkers have a lower hardware overhead than static checkers, but operation is more sensitive to circuit parameters.

2. Pulse or Level Error Signal. Checkers that produce a pulse when a stability error occurs are simpler than level output checkers, but the pulse must be latched eventually.

3. Independent or Combined with system flip-flop. It is possible to design checkers that share logic with the system flip-flop. The main advantage of this is the reduced area overhead. One of the disadvantages of sharing logic is that the flip-flop value can be corrupted if dynamic logic is used. The flip-flop can get corrupted during delay testing, but not during normal operation.

Intuitive stability checker designs are presented first, followed by designs using formal methods, and finally other designs. The checkers were simulated to find the smallest pulses that could be detected. All designs were found to have roughly the same performance, but the smaller dynamic designs were more sensitive to circuit parameters. Optimization of transistor sizes to improve performance was not done.

3.2.4.1 Intuitive Designs

An intuitive stability checker design is to compare the value of the CUT output D during the checking period, to its value at the beginning of the checking period, which is the sampled value Q. This is shown in Fig. 3.8, and involves XORing D and Q, and setting a latch if the signals differ during the checking period. The checker is reset at the end of the checking period. It is not necessary to have an SR latch for every signal being checked, reducing the area overhead.

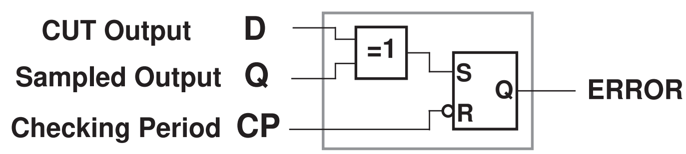


Figure 3.8. Conceptual Implementation of Stability Checker

One difficulty with this approach is that the propagation delay of the system flip-flop must be taken into account. D and Q can only be compared once Q is valid, so small delay faults will not be detected using the XOR. In fact, to check for setup violations, D should be checked for any changes starting one setup time before the rising edge of the system clock to ensure that the correct value is latched.

Figure 3.9 shows another intuitive design. If D is both 0 and 1 during the checking period, then there must have been a stability checking error. An efficient design of a 1s detector is shown in Fig. 3.8(b). Node Z is low if D=1 during the checking period. This design was used in [van Brakel 92]. The 0s detector is similar.

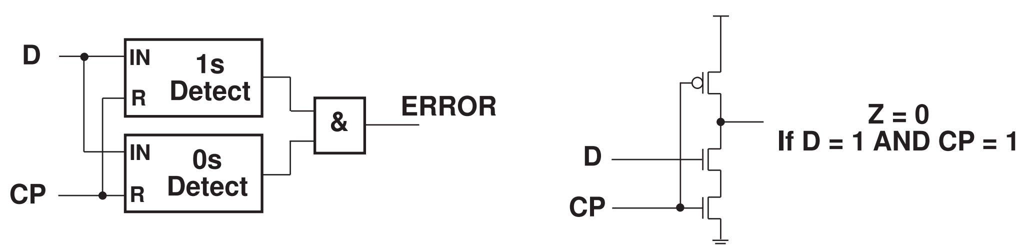


Figure 3.9. (a) Intuitive Design, (b) 1s Detector Implementation

3.2.4.2 Formal Design -- Level Output

Figure 3.10 shows a fundamental mode static logic stability checker design. This circuit was designed directly from the flow table [McCluskey 86] describing the circuit functionality. The output ERROR signal is driven high if there is any change in D while the checking period CP=1. The operation of the circuit is as follows. Assume D=0 when the checking period is inactive, CP=0. We have Y_1=0 and Y_2=1, so ERROR=0. Now the checking period starts, CP→1. If D rises, then Y_1→1 and ERROR→1. The circuit is symmetrical for detecting falling edges in D.

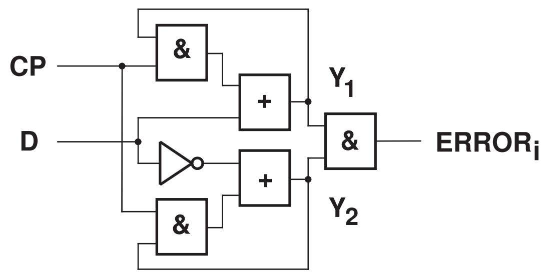


Figure 3.10. Gate-Level Stability Checker Design

Note that in this design, as in others that need the complement of the CUT output, D′ can often be taken from the system flip-flop, reducing the area overhead. For example, inverters I1, I2 or I3 in the flip-flop shown in Fig. 3.11 could be used. This also reduces loading on the CUT output.

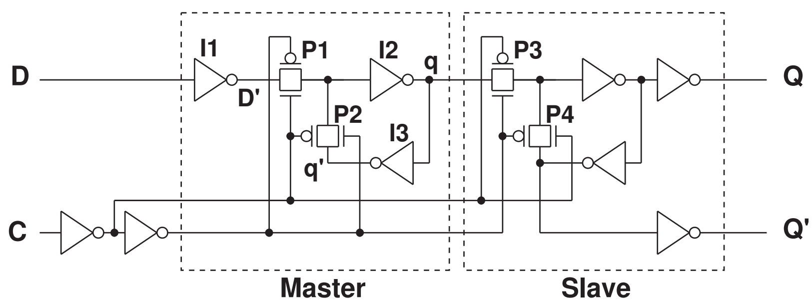


Figure 3.11. Master-Slave D Flip-Flop (FD1 [LSI 91])

The stability checker design shown in Fig 3.10 was implemented with CMOS NAND gates using the Stanford BiCMOS process. The layout is shown in Fig. 3.12. This is also the stability checker design used in the Test Evaluation Chip Experiment described in Chapter 6. Both implementations have been tested and operate correctly.

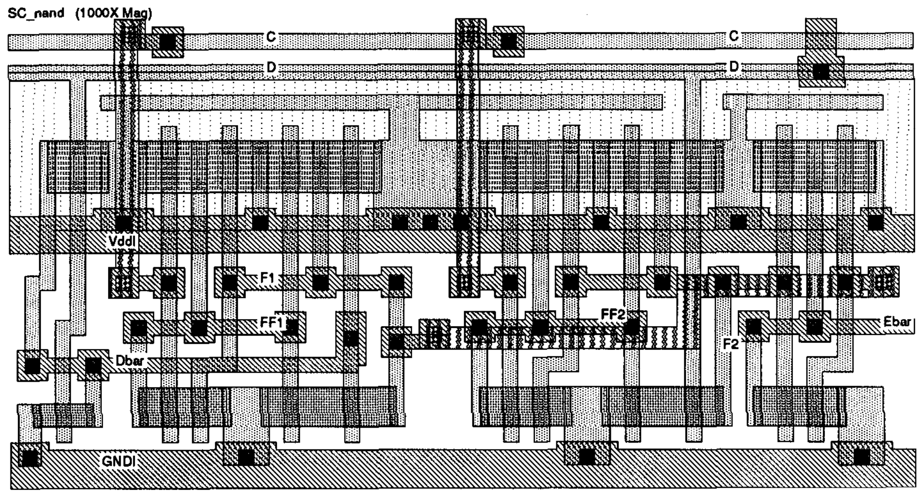


Figure 3.12. NAND Layout of Stability Checker Design in Fig. 3.10

3.2.4.3 Formal Design -- Pulse Output

The flow table approach can also be used to design a pulse-mode circuit. In this case, there is a pulse at the output of the stability checker if D changes during the checking period. This circuit has two states, and the state variable Y_1 is a delayed copy of the CUT output D. Stability errors are found by XORing D with Y_1, as shown in Fig. 3.13. Note that the implementation is very similar to the conceptual design in Fig. 3.8.

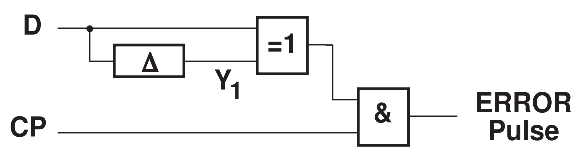


Figure 3.13. Pulse Output Stability Checker Design

3.2.4.4 Ad-hoc Designs

A dynamic logic ad-hoc stability checker design is shown in Fig. 3.14. Assume that D=1 before the beginning of the checking period. Node X is precharged to 5V. When CP=1, node C1 is discharged to 0V. Now if D goes to 0 during the checking period, then the voltage at Node X is reduced due to charge sharing with Node C1. As long as the resultant voltage is lower than the threshold of the output inverter, the error is detected.

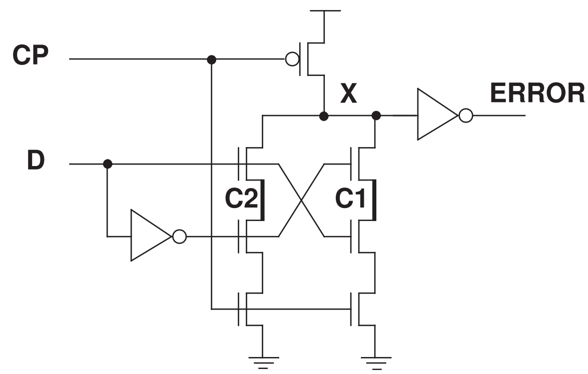


Figure 3.14. Dynamic Stability Checker

This design is very sensitive to the circuit parameters, particularly the parasitic capacitance at nodes C1 and C2. Figure 3.15 shows the circuit layout that was manufactured using the Stanford BiCMOS process, including the extra capacitance needed for the circuit to function correctly.

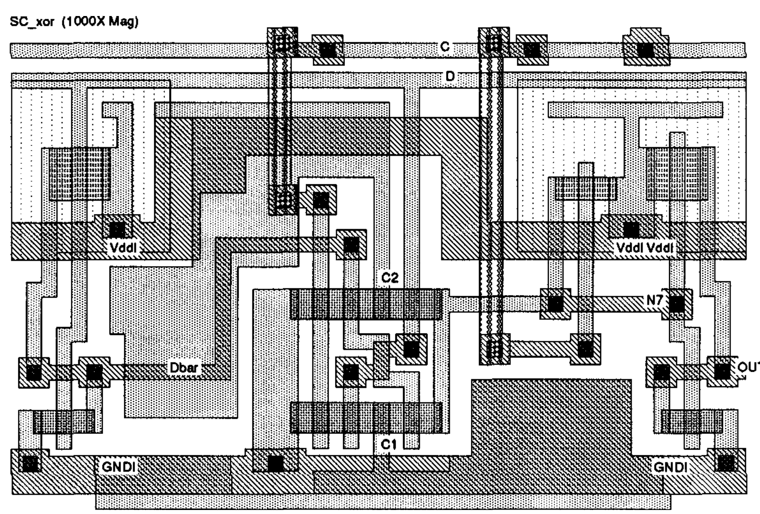


Figure 3.15. Layout for Dynamic Stability Checker in Fig. 3.14

Figure 3.16 shows a similar design using fewer transistors. Voltage levels are degraded in this design since one-transistor pass gates are used.

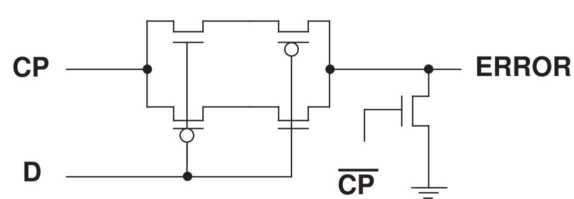


Figure 3.16. Efficient XOR Checker Design

3.2.4.5 Switching (Short-Circuit) Current Design

Figure 3.17 shows a design of a stability checker that uses the transient switching current in CMOS inverters to detect signal changes. While the checking period is inactive, node Y in Fig. 3.17 is kept at V_{DD} by the precharging transistor, so the inverter operates normally. Once the checking period signal rises, the bus is left floating at a little above V_{DD} (due to bootstrapping -- the gate-to-drain capacitor pulls the Precharged node high when CP=1 [Weste 85]). If the output of the CUT, D, does not change, the inverter will draw negligible static current and node Y will remain high. If D changes, the current flow while the inverter changes state will partially discharge the bus, which can be detected with a sense amplifier. SPICE simulations were performed to verify the operation of the circuit.

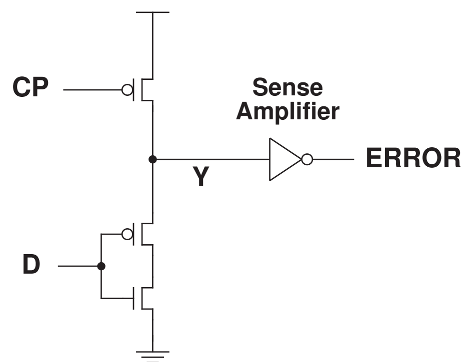


Figure 3.17. Switching Current-Type Stability Checker

The inverter switching current is small, so a sense amplifier is needed to detect the small voltage swing on node Y. A possibly better design is charge node Y to just a little above the threshold of the sense amplifier (leaving enough noise immunity), so that small voltage drops can be detected. In this way, an ordinary inverter can be used as the sense amplifier, and it might be possible to connect more than one CUT output to node Y.

3.2.4.6 Bridging Current Design

The previous design used the transient switching current in CMOS circuits to discharge a node and detect the error. It is also possible to design a circuit which bridges two nodes together. As long as D does not change, the two nodes have the same value, but of D changes, then the nodes will have opposite values, shorting the precharged node to ground. This approach is used in a stability checker design presented in the next chapter.

3.3 COMPATIBILITY WITH SAMPLING AND BIST

Stability checking needs to be integrated with other BIST techniques since stuck-type faults are not detected. For example, if a CUT output is always low, the stability checker will never produce an error. The other BIST techniques provide stuck-at coverage, while Stability Checking improves the coverage of delay faults.

The Stability Checking test can be done at the same time as the conventional BIST test to reduce test time. If the tests are done separately, then test logic can be shared. For example, the system flip-flops can be used to store stability errors, which can be scanned out for improved diagnosis.

Designs combining Stability Checking and BIST are shown, followed by a brief discussion of error masking due to aliasing in BIST designs.

3.3.1 Combined Stability Checking and BIST

The normal clock waveforms were modified significantly for implementing Stability Checking, so it needs to be determined if it is still possible to latch the CUT outputs and do Stability Checking together.

The timing waveform in Fig. 3.3 for flip-flop designs is repeated in Fig. 3.18. Note that the CUT outputs cannot be latched one cycle time after the application of the input vector. For example, after the vector <V_2> is applied, the CUT outputs are normally latched at time ***C***. There are two problems with doing this. First, added logic is needed as the flip-flops are assumed to be positive edge-triggered during normal operation. Second, if the flip-flop is clocked, then new inputs will be applied to the combinational logic following the flip-flop. If this combinational logic is also being tested, new inputs must not be applied during the checking period, otherwise output changes due to those inputs would be flagged as delay faults erroneously. In fact, the flip-flop at the output of the CUT could feed one of the inputs of the same CUT; this is known as "self-adjacency".

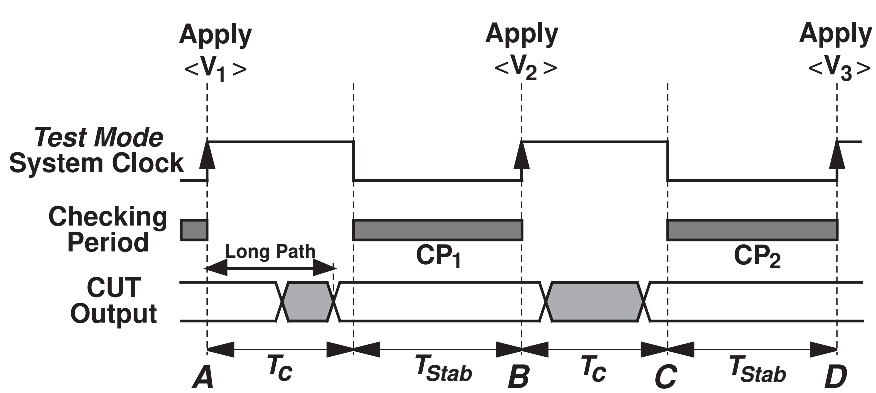


Figure 3.18. Timing Waveforms for Flip-Flop Designs

It is possible to do Stability Checking and latch the output at the same time, however, by taking advantage of the fact that the sampled value at the end of the checking period is the same as the sampled value at the beginning of the checking period, if there are no delay faults. Therefore, although the value at ***C***, should be latched, the value at ***D*** can be used. If there are stuck-type faults present, then the values at ***C*** and ***D*** are the same, so there is no error loss. If ***C*** and ***D*** are not the same, the stability checker will detect the difference.

Several BIST techniques have been proposed. Built-in Logic Block Observer (BILBO) and Circular BIST are two examples. Each BILBO register can function as a normal register, a scan chain, a pseudo-random test pattern generator, or signature analyzer [Benowitz 75]. All the registers are connected into one large combined test pattern generator and signature analyzer in Circular BIST.

The stability checker outputs can be ORed independently of the BIST technique used, or logic can be shared. Figure 3.19 shows a modified BILBO cell. BILBO registers usually have an extra reset Test Mode that is not necessary. Four modes are possible using two Test Mode signals, but the reset can also be performed using the scan mode, so this mode can be used to load the output of the stability checkers directly into the system flip-flops. No new Test Mode signals are needed. The stability checking results can then be scanned out in the usual way.

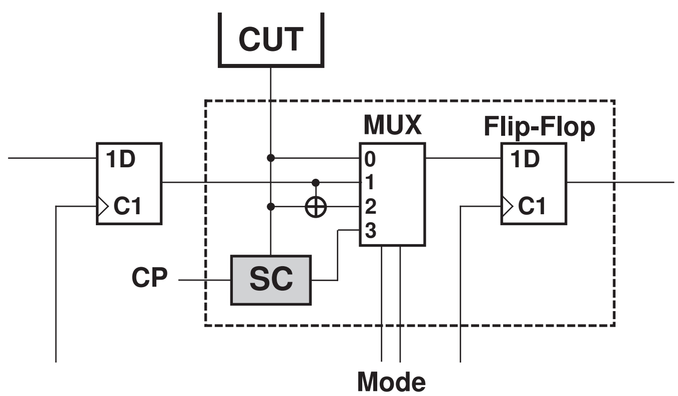


Figure 3.19. Modified BILBO Cell

The above approach is useful if diagnosis of the Stability Checking error is necessary, but increases test time. Another approach is to XOR the output of the stability checker with the output of the circuit, and collect the stability checking errors together with the normal signature.

3.3.2 Aliasing

The biggest uncertainty in BIST is the error masking in the response compaction. There is a certain probability that the compacted faulty response will be equal to the fault-free compacted response. This phenomenon is called aliasing. Extensive fault simulations can give an idea of the aliasing probability for particular fault models, and probabilistic approaches to computing the aliasing probability have been presented. I did some work with Dr. Nirmal Saxena on simple upper bounds on the aliasing probability for serial signature analysis [Saxena 91b] [Saxena 91c] [Saxena 92], but this will not be reported here. The motivation for this work was choosing test lengths and linear feedback shift register (LFSR) feedback polynomials to minimize the aliasing probability. The conclusion was that although the aliasing probability approaches the well-known asymptotic value of 2^{-k} for a k bit signature register, for short test lengths L, the aliasing probability is essentially bounded by $1/L$.

There is one result presented in [Franco 91a] that is of interest for delay testing. This work investigated the aliasing probability for faults with very low fault detection probabilities. Delay faults are harder to detect than stuck-at faults, and can have very low fault detection probabilities [Savir 88]. The *fault detection probability* of a delay fault is the probability that a randomly-selected vector pair will detect the fault. The main result is that for certain non-primitive polynomials and faults with low detectabilities, the asymptotic aliasing probability of 2^{-k} is not reached even for very long test lengths. The notion of "practically infinite" test lengths was used to resolve the apparent contradiction between this result and the asymptotic result.

For example, consider a 100 bit signature analyzer with a test length of 10^6, and a fault with a very low fault detection probability. The asymptotic aliasing probability is miniscule. However, the aliasing probability could be as high as 1% if a simple feedback connection is used where the last stage is fed back into the first stage of the signature analyzer.

3.4 TESTING STABILITY CHECKERS

The stability checking circuits need to be tested after manufacture. If the CUT is free of delay faults and is operated at system speed, the checkers will never signal errors so they cannot be tested. Test signals can be added to test the stability checkers; this was done for the Test Evaluation Chip Experiment described in Chapter 6. Another possible testing strategy is to induce delay "faults" by increasing the clock frequency above the maximum specified operating speed of the CUT, so the outputs will be unstable during the checking period. This is sufficient if the outputs of individual stability checkers can be observed, but care must be taken if the stability checker outputs are ORed together, otherwise error masking can occur. Test pattern generation is needed to find input vector pairs such that only one output is unstable at a time, so the corresponding stability checker can be tested. These are called single-path-propagating delay tests, using the classification presented in Chapter 2. Test generation time should not be an issue, as tests are only needed for CUT outputs, and not all internal nodes.

3.5 RESULTS

Examples have been given to show the advantages of delay testing by Stability Checking, and circuits for the analyzers have been designed to show that the technique is feasible. Experimental results are now given to show the benefit of using Stability Checking for delay testing.

The benefit of Stability Checking for delay testing depends on the presence of hazards in the CUT response: if all the outputs have single transitions, the technique will only detect faults that are also detected by sampling the output. Even if outputs have single transitions, Stability Checking can still be useful in cases where low latency is required, as one does not have to wait until the end of the test to know if there were delay faults.

To determine the number of hazard pulses in typical designs, the ALU181 and ISCAS’85 [Brglez 85] benchmark combinational circuits were simulated with pseudo-random input patterns, and the output waveforms were analyzed.

For the ALU181, the test patterns were also applied to a 74LS181 chip using a Tektronix DAS9200 ATE, and the response was observed on a high speed oscilloscope. A high correlation between the actual circuit hazard transitions and the simulation results was found, despite the fact that the actual gate delays were not known for the chip. A typical waveform is shown in Fig. 3.20. The 1-hazards at clock cycles 1, 3 and 4 were observed both on the ATE and the Verilog simulations.

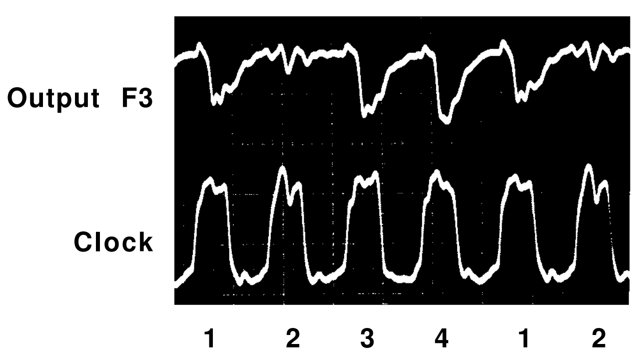


Figure 3.20. Hazards in 74LS181
© IEEE [Franco 91b]

The waveform simulator described in [Franco 94c] was used to find the distribution of the number of transitions at the outputs of the ISCAS'85 circuits. Table 3.2 shows the average number of transitions per output node per input vector.

Table 3.2. Average Number of Transitions per Node per Vector

Circuit	Output Transitions
ALU181	0.613
c432	1.131
c499	0.526
c880	0.527
c1355	0.536
c1908	0.886
c2670	0.513
c3540	1.370
c5315	0.634
c6288	31.660
c7552	1.081

Multiple transitions were observed for most circuits. The C6288 benchmark circuit has particularly many transitions due to its rich reconvergent fanout structure. Figure 3.21. shows some output waveforms for the C6288, for a single vector pair at the inputs.

The distribution of the number of transitions at the outputs of the ISCAS'85 circuits are shown in Fig. 3.22. The distributions are based on counting the number of output transitions at each output for 120 random vectors. Some of the circuits have multiple transitions a small fraction of the time, while other have a reasonable fraction of multiple output transitions. C6288 is very different from the rest; there was even one output with 80 transitions. Zero transitions means that the output was stable, one transition means that there was a single change in the output, and higher numbers indicate the presence of hazards pulses.

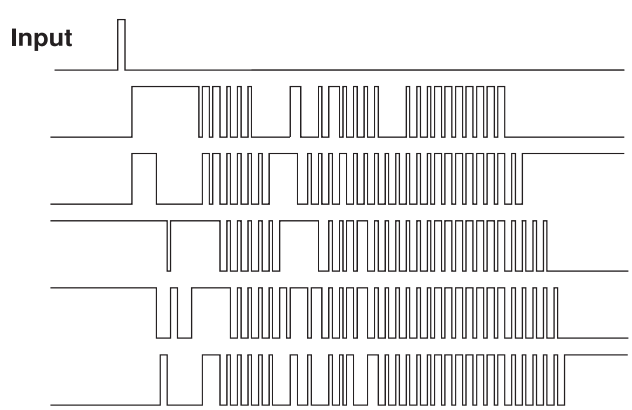


Figure 3.21. Hazards in C6288 Outputs

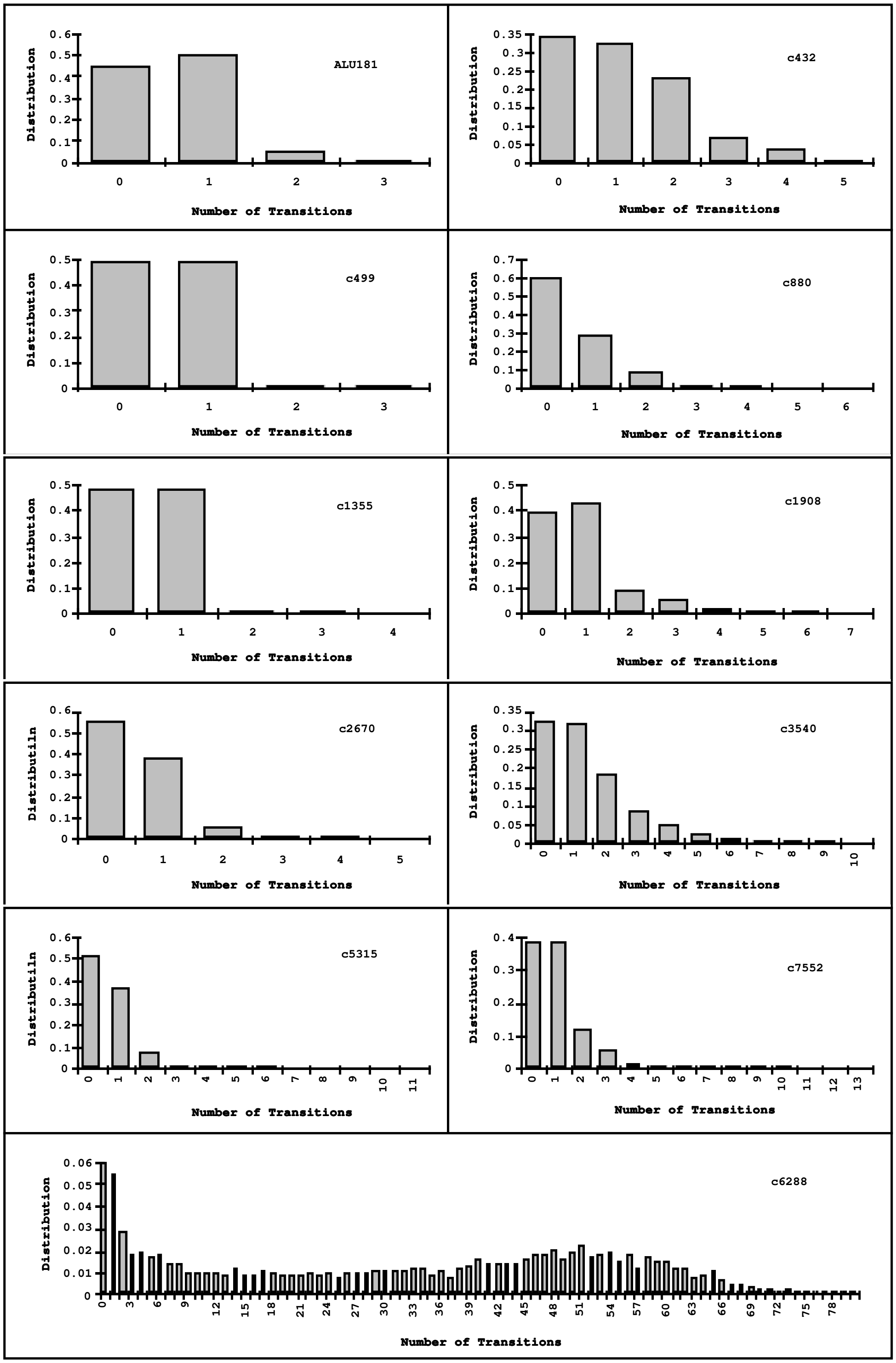


Figure 3.22. Distribution of Number of Output Transitions

3.6 CONCLUSION

A new technique for delay fault testing was introduced in this chapter. Delay faults are detected in Stability Checking by observing the output waveforms of the CUT for any changes after the sampling time. Test architectures and efficient stability checker implementations have been shown. Two design were manufactured using the Stanford BiCMOS process, and work correctly. The design in Sec 3.2.4.2 was included in the Test Evaluation Chip Experiment, and also works correctly.

Stability Checking is suitable for BIST, where delay testing is particularly difficult due to test invalidation by hazards. Simulations were performed to determine the expected number of hazard pulses at circuit outputs with pseudo-random inputs applied. For the ALU181, the simulations were verified on an ATE.

Multiple transitions were found at the circuit outputs, but it should be noted that even if there are only single transitions, Stability Checking can still be useful. If there are only single transitions, then delay faults will also be detected by the sampled value, but in Stability Checking there is no latency or possibility or aliasing.

Although the Stability Checking has been described as an off-line testing technique, it can also be used for on-line delay testing. This is the subject of the next chapter.

Chapter 4

On-Line Delay Testing

This chapter describes how Stability Checking can be used for on-line checking. This work was presented as a CRC Technical Report [Franco 93], and at the 1994 VLSI Test Symposium [Franco 94a].

4.1 INTRODUCTION

Circuits are tested after manufacture to determine if they operate as designed. Due to environmental influences and finite reliability, testing also needs to be done to verify that a circuit that worked when manufactured continues to work correctly. To ensure data integrity in critical applications, the circuit needs to be constantly monitored to ensure that correct outputs are produced. In these cases, testing is done concurrently with normal system operation, and is called *on-line* or *concurrent* checking.

Sources of error include circuit reliability failures, as well as external transient disturbances such as capacitive coupling, radiation, supply voltage changes, and environmental changes such as temperature and humidity. Some form of redundancy is used to detect these errors.

Hardware redundancy is a widely used technique for on-line checking. Two common approaches are duplication, where two copies of the circuit are compared, and parity checking, where an overall parity bit is generated [Johnson 89]. The overhead for parity checking is much smaller than for duplication if the relationship between outputs is simple (e.g., bus), but it can be as high as for duplication for general circuits. Other approaches using Berger codes and groups of parity bits have been explored [Jha 91] [De 92], and result in lower hardware overhead than duplication for some circuits.

A new general technique for on-line checking of digital systems is proposed in this chapter. Unlike other techniques which are generally totally self-checking with respect to stuck-at faults (which are known to have limitations for CMOS circuits), the proposed technique targets the expected failures in CMOS circuits. It is shown that, under certain conditions, Stability Checking can be used for on-line checking. Stability Checking was described in Chapter 3, and is based on the fact that in a fault-free circuit, the outputs are expected to have reached the desired logic values by the time they are sampled, so delay faults can be detected by observing the outputs for any changes after the sampling time. The stability checkers are "mini-watchdogs" that check if a computation completes in the specified time within each clock cycle, analogous to system-level watchdog timers [Connet 72] that check if a task completes within a specified time interval.

The advantage is lower hardware overhead than duplication while detecting most common CMOS reliability failures [Woods 86], as well as many transient failures.

Some errors will not be detected by on-line Stability Checking, but this is a limitation of all checking techniques. Different approaches trade off the class of errors detected for area or speed overhead. Parity checking, for example, does not detect many classes of errors but is still useful in many applications. On-line Stability Checking detects errors caused by most common reliability failures and transients, assuming the circuit is initially free of functional faults, at a fraction of the cost of duplication. A *functional fault* is a fault that changes the logic function of the circuit. Most CMOS reliability failure mechanisms are not instantaneous, but rather "wear out" and degrade performance before causing functional faults. Sudden functional faults are not guaranteed to be detected, but these are expected to be rare.

On-line Stability Checking is also suitable for aggressively clocked systems that could have marginal timing problems in certain environmental conditions, for example.

This chapter is divided into three parts. First, it will be shown that on-line Stability Checking is *feasible*, by showing timing diagrams and stability checker designs. Next, on-line Stability Checking is shown to be *useful*, by evaluating its performance and comparing it to other on-line checking techniques. It is then shown that on-line Stability Checking is *practical* by presenting an algorithm for modifying circuits to meet timing constraints. Experimental results for benchmark circuits are given, and extensions, including software run-time checking and VHDL synthesis, are described in Sec. 4.6.

4.2 ON-LINE STABILITY CHECKING

The requirement for using Stability Checking on-line is given in this section. The hardware model used is fully synchronous design, with the combinational circuit-under-test (CUT) surrounded by edge-triggered registers, as used in Chapter 3. On-line Stability Checking is also possible for other clocking schemes.

4.2.1 Off-line Delay Testing by Stability Checking

Stability Checking was proposed as a different way of improving the thoroughness of delay fault testing. The greatest improvement over traditional testing is achieved in situations where the input patterns cannot be controlled, such as pseudo-random testing, for example. This is also the case for on-line checking, as the signals during normal operation are used as the test vectors.

Figure 4.1 shows a typical timing diagram for off-line Stability Checking. In off-line testing, the vector <V_3> following <V_2> is not applied during the checking period for <V_1,V_2>, since output changes due to <V_3> would be erroneously flagged as delay faults.

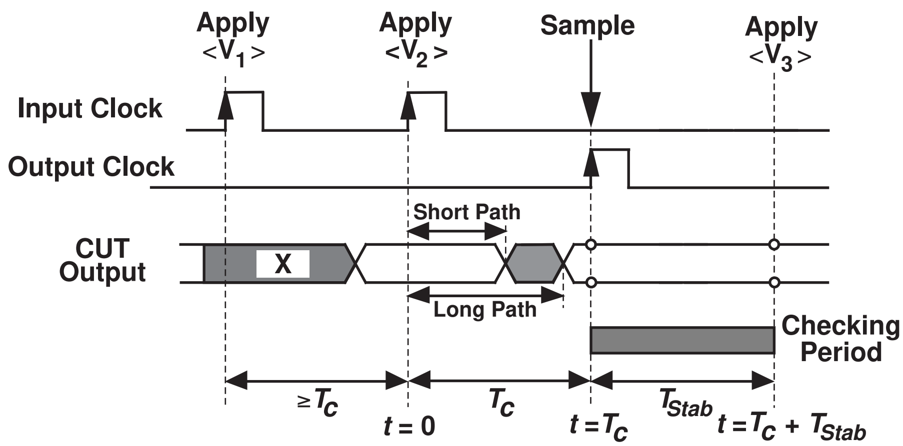


Figure 4.1. Off-Line Stability Checking (similar to Fig. 3.2)
© IEEE [Franco 94a]

Delay faults shorter than the duration of the checking period are detected when they cause errors using Stability Checking. The reason is that the fault free value will be available by the end of checking period, so if there are no changes during the checking period (stability errors), then the sampled value must be correct. This property makes on-line Stability Checking possible, since the CUT output need not be compared to another signal to determine if it is correct.

4.2.2 Reduced On-Line Checking Period

The difficulty in implementing Stability Checking on-line is that the vector <V_3> following <V_2> cannot be delayed without impacting the performance of the circuit. This is resolved by making the duration of the checking period duration less than the propagation delay of the shortest path in the circuit, T_{short}. This often requires modifications to the CUT, as discussed in Sec. 4.5. In this case, although vector <V_3> is applied at time $t = T_C$ in Fig. 4.1, it will not affect the output until after the end of the checking period. Reducing the checking period is the only restriction necessary for implementing on-line Stability Checking, i.e.:

$$T_{Stab} < T_{short} \text{ (Best-case)} \qquad (4.1)$$

The restriction in equation (4.1) is shown graphically in Fig. 4.2. The worst-case longest path in the circuit must be shorter than the cycle time less the worst setup time for the flip-flop, D to Q propagation delay for the flip-flip, and clock skew [Weste 93].

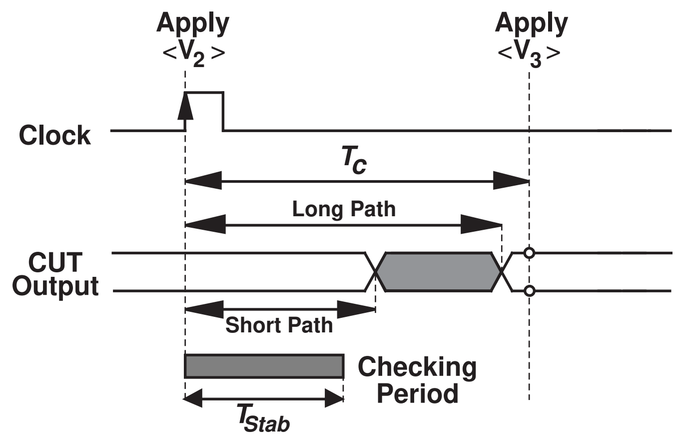


Figure 4.2. Checking Period Restriction

Different operating conditions and timing skew must be taken into account in computing the best-case short path through the circuit. For example, typical values for delay dependence are a 6% speed increase for a 0.25V supply increase, and an 8% speed increase per 25°C junction temperature decrease [LSI 91]. Delay ranges due to process variations can be large for different wafers. Process variations within one chip, however, are small. (It would be very difficult to control clock skew if this were not the case.) Therefore if speed binning is done, process variations are taken care of. Another approach is to generate the checking period signal on chip, so that it tracks process variations.

Note that, unlike off-line Stability Checking, hazards are not a factor in on-line Stability Checking. Whether the output has a hazard-free transition or multiple transitions, any change in the output waveform when the waveform should be stable is detected.

4.3 IMPLEMENTING ON-LINE STABILITY CHECKING

The requirements for implementing on-line Stability Checking are similar to implementing off-line Stability Checking:

- Providing a signal to mark the checking period,
- Collecting the error signals from the different flip-flops,
- Implementation of the stability checker itself.

The architecture will be discussed first, followed by possible stability checker implementations.

4.3.1 Architecture for Stability Checking

Figure 4.3 shows a block diagram of the on-line Stability Checking architecture, with a stability checker added to each CUT output in a design that will be checked. The problem of distributing the checking period signals is resolved by using the system clock to mark the checking period as in off-line Stability Checking, except that the checking period is now when the clock is high. Using the clock as the checking period both reduces the cost of Stability Checking since no extra signals are required to indicate when the CUT outputs should be checked, and reduces skew in the checking period since the clock signal distribution is typically well controlled. The duty cycle of the system clock is adjusted so that the time, T_{Stab}, the system clock remains high defines the checking period.

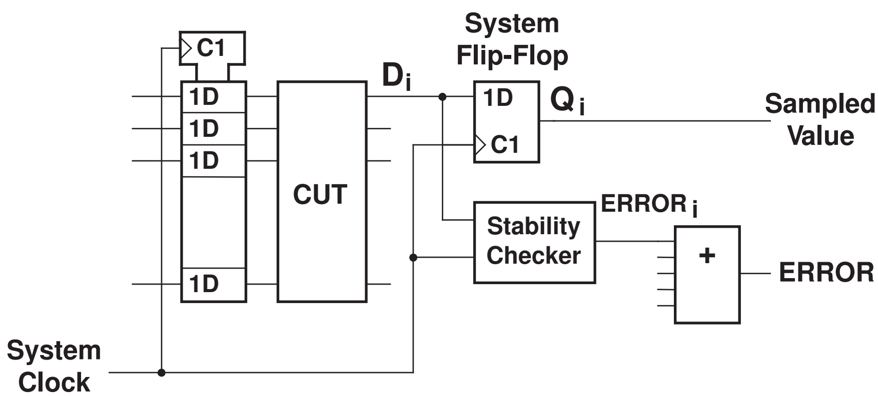


Figure 4.3. Block Diagram of On-Line Stability Checking Architecture

The timing diagram for a single-clock, edge-triggered design is shown in Fig. 4.4. The checking period CP_1 for vector <V_1> starts after the application of <V_2>, and ends before the CUT outputs can be affected by input <V_2>.

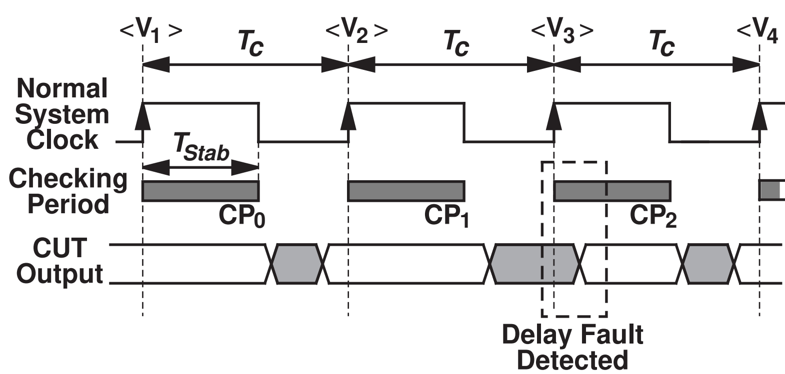


Figure 4.4. Timing Waveforms for On-Line Stability Checking
© IEEE [Franco 94a]

The duty cycle of the system clock determines the checking period, but the duty cycle of the system clock cannot usually be adjusted arbitrarily. Very small or very large duty cycles are undesirable, due to minimum clock pulsewidth restrictions. This is not expected to be a problem here, as the target checking period duration is roughly half the clock cycle. The problem of generating a clock with the required duty cycle remains, however. This problem is also encountered in off-line Stability Checking if the clock and checking period signals are shared, but off-line, there are two separate modes for normal operation and delay testing. On-line, there is only the normal operating mode.

The duty cycle of the clock can either be set externally, by delay elements, by dividing a higher frequency, or by an on-chip phase-locked loop (PLL) [Weste 93]. For the frequency divider or PLL, only ratios of small integers are practical for the duty cycle (e.g. 1/3, 2/5, 1/2, etc.). The fraction closest to the desired checking period can be selected for the circuit.

There are many ways to collect the individual $ERROR_i$ signals, depending on the level of diagnosability required. For many applications it is sufficient to localize the error board or chip, and a global error indication can be produced by ORing all the individual $ERROR_i$ signals. Figure 4.5 shows one implementation, where the ERROR signals for different registers are cascaded, minimizing the wiring overhead, depending on the layout of the registers. More detailed information might be useful, however, for failure mode analysis of No Trouble Found boards [Cortner 87]. The extreme case is to latch each $ERROR_i$ signal and scan the result.

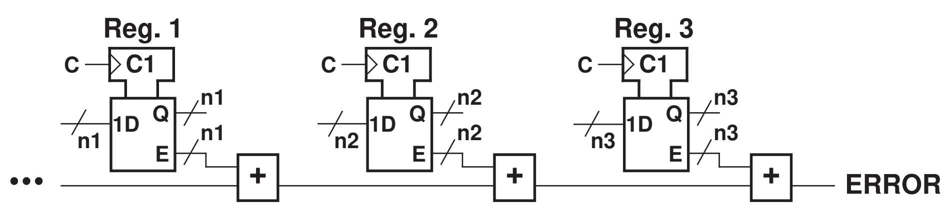


Figure 4.5. Cascaded ERROR Signal Collection

4.3.2 Stability Checker Design

The function of a stability checker is to produce an error if there are any changes in the CUT output during the checking period. The stability checkers designed in Chapter 3 can be used for on-line Stability Checking. Another design is shown below that combines the stability checker and flip-flop and is actually smaller than the original flip-flop. A combined stability checker and scan flip-flop is then shown.

Dynamic Stability Checker Design Combined with Flip Flop

A dynamic CMOS [Weste 85] combined flip-flop and stability checker is shown below. This design exploits the fact that CMOS circuits draw very little static power. Instead of supplying power directly to inverters in the flip-flop master, power is supplied by precharging and then floating an internal node during the checking period. If the CUT output D does not change during the checking period, the precharged node will remain high. The transient short-circuit current from changes in D will discharge the monitored precharged node.

Figure 4.6 shows a typical CMOS master-slave flip-flop [LSI 91]. The function of transmission gate P1 is to decouple D′ and q when the clock is high. While the clock is low, transmission gate P1 conducts, and the value at D′ propagates to the input, q′, of transmission gate P2. Transmission gate P2 is turned on, latching q′. The critical observation is that since the checking period is defined by the system clock, D does not change in a fault-free circuit while the clock is high, and so P1 can be removed.

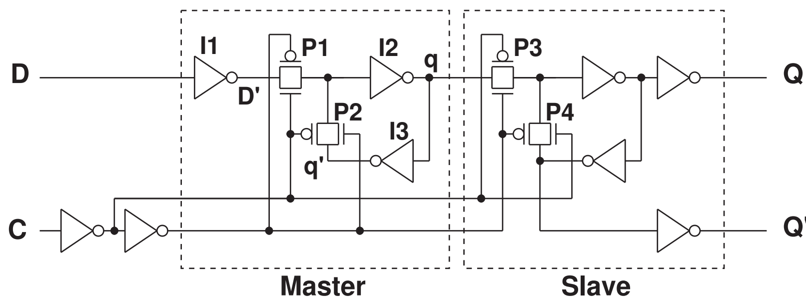


Figure 4.6. Master-Slave D Flip-Flop (FD1 [LSI 91])

In the presence of a delay fault, D will change while the clock is high. The modified flip-flop master is shown in Fig. 4.7. Transmission gate P1 is removed, and inverters I1, I2 and I3 are powered by the precharged node $ERROR_i'$.

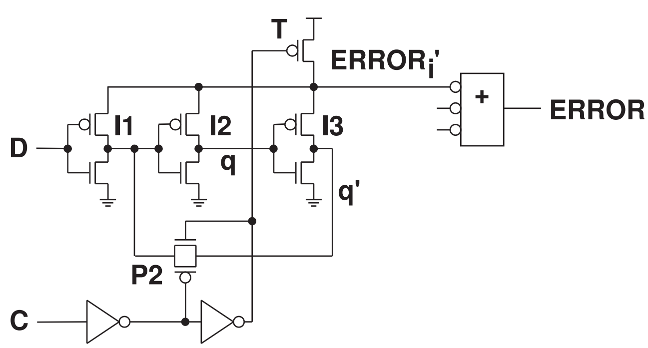


Figure 4.7. Modified Master of Flip-Flop with Stability Checking

$ERROR_i'$ is kept at V_{DD} by transistor T while the system clock is low, so inverters I1, I2 and I3 operate normally. When the system clock rises, $ERROR_i'$ is left floating close to V_{DD}. If D does not change, I1, I2 and I3 draw negligible static current and $ERROR_i'$ remains high. If D changes, there are two types of current paths that will discharge $ERROR_i'$, detecting the fault. The first component is the transient short-circuit switching current of the inverters (this was used in the stability checker design in Sec. 3.2.4.5). The second component is due to the finite propagation delay of inverters I2 and I3 in the loop. When D changes, the outputs of I1 and I3 will have opposite values and be tied together, momentarily discharging $ERROR_i'$. For a 0→1 (1→0) transition in D, there is a path from $ERROR_i'$ through the P channel of I3 (I1) and the N channel of I1 (I3), to ground.

SPICE simulations show that the modified flip-flop functions as described. The waveforms in Fig. 4.8 show that erroneous rising (time 90) and falling (time 170) transitions in D during the checking period are detected. Single transitions were simulated for D, since if a single transition is detected, then multiple transitions will also be detected. Several transitions in D while the clock is low don't affect the ERROR output.

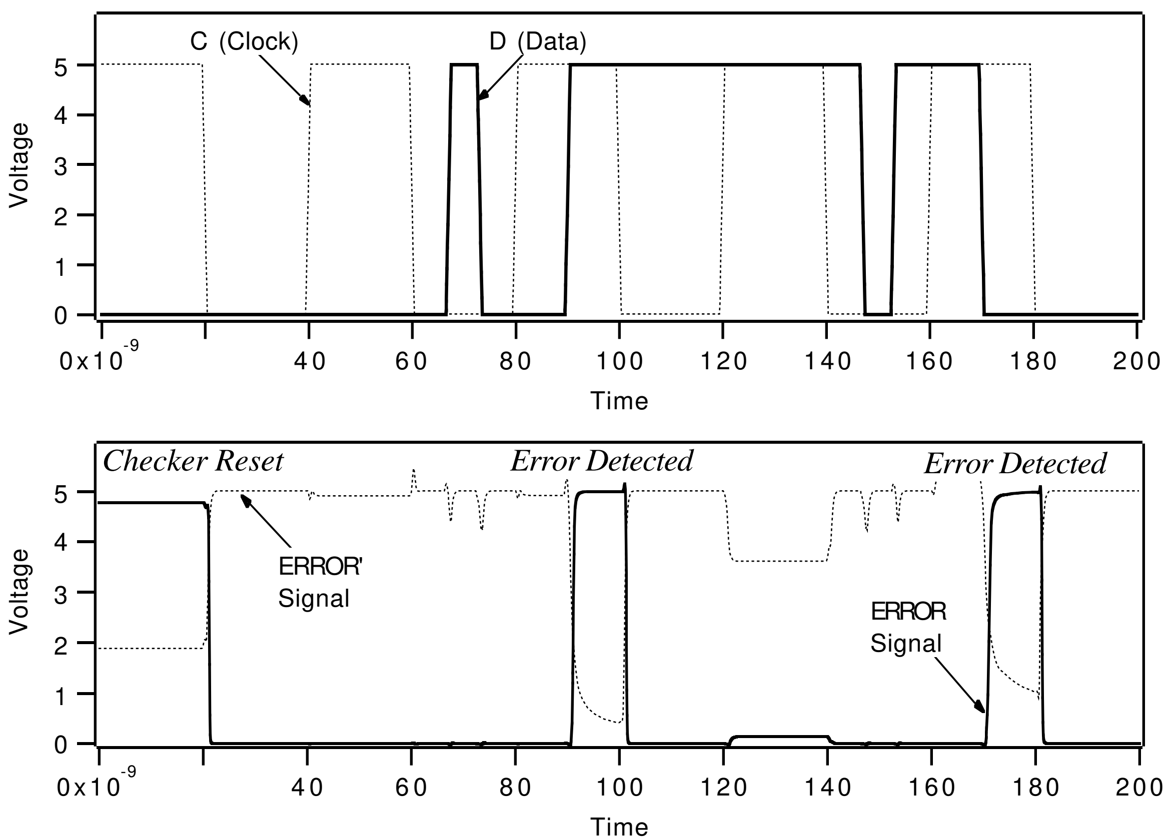


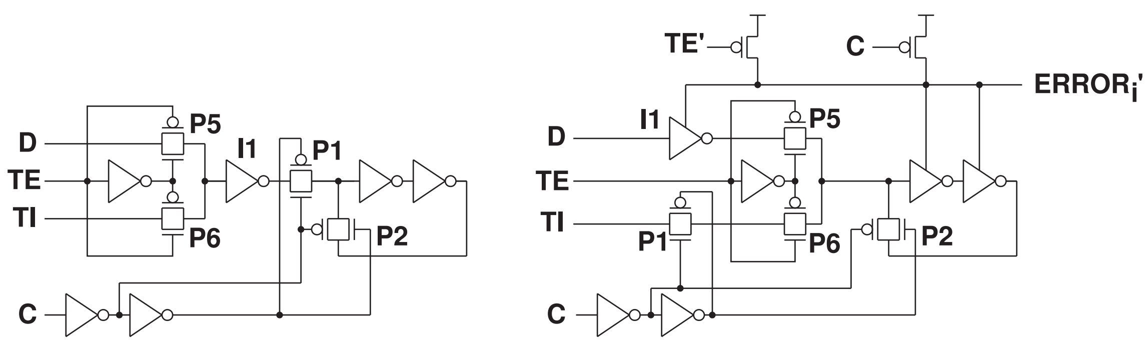
Figure 4.8. Spice Simulations for Modified Stability Checking Flip-Flop

Another benefit of the modified flip-flop is that, since a transmission gate is removed, the setup time is reduced. The setup time for the simulated circuit was about 2/3 of the setup time of the original flip-flop. Sharing logic between the flip-flop and stability checker also makes it is easier to synchronize the start of the checking period with the flip-flop setup time, by using internal signals in the flip-flop.

Stability Checker Design Combined with Scan Flip Flop

Since scan design is common, a scan-compatible flip-flop based on the design in Fig. 4.7 is shown in Fig. 4.9. Inverters I1, I2 and I3 are powered by precharged node $ERROR_i'$ as in Fig. 4.7.

Two modifications are necessary to the flip-flop. Transmission gate P1 could only be removed from the D input of the flip flop, because it satisfied the restriction in equation (4.1). The scan input (TI), however, could be a very short path from the previous flip-flop, so the input transmission gate is necessary. Furthermore, in scan mode, the short TI input will toggle the flip-flop while the clock is high (during the checking period). Therefore an extra transistor is needed to keep the precharged node high, to keep the flip-flop operating correctly during scan.



(a) Conventional Scan (b) Combined Scan and Stability Checking

Figure 4.9. Combined Scan and Stability Checking Flip-flop Master

Typical transistor counts for different flip-flops are given in Table 4.1. The combined flip-flop and stability checker is smaller than the original flip-flop since a transmission gate was removed (this flip-flop cannot be used in designs that do not satisfy equation (4.1)). The routing overhead for Stability Checking should be lower than for full scan because no extra scan clock or scan-mode signal is needed. The combined scan flip-flip and stability checker is only 6.25% larger than the scan flip-flop.

Table 4.1. Comparison of Transistor Counts

Flip-Flop	26
Flip-Flop & Stability Checking	25
Scan Flip-Flop	32
Scan Flip-Flop & Stability Checking	34

4.4 PERFORMANCE EVALUATION

In this section, the error detecting capability of on-line Stability Checking is evaluated. Limitations are noted, followed by a description of common failure modes and transient errors. The performance is then compared to other on-line checking techniques.

4.4.1 Limitations

The biggest limitation of on-line Stability Checking is that functional faults are not guaranteed to be detected. Output stuck-at faults, for example, will not be detected since the output is always stable. (Off-line, this is not a problem as the fault-free sampled value is known.) If a sudden functional failure does occur, it will only be detected if one of the CUT outputs changes during the checking period. The probability of detection can be small, as it depends on when the defect occurs, and the number of outputs to which it propagates. The probability of detection is bounded by the fraction of the clock cycle covered by the checking period, i.e.:

$$\Pr\{\text{detect sudden functional failure}\} \le \frac{T_{Stab}}{T_c} \qquad (4.2)$$

For a delay fault of size d, the fault-free output is available by time $T_c + d$ at the latest. Therefore, any resulting error is guaranteed to be detected if the output is checked for stability between T_c and $T_c + d$. Therefore, errors due to delay faults smaller than the checking period are always detected:

$$\Pr\{\text{detect error due to delay fault } d < T_{Stab}\} = 1 \qquad (4.3)$$

4.4.2 Common Failure Modes

Although functional faults are not guaranteed to be detected, most common reliability failures in CMOS VLSI circuits first manifest themselves as small delay faults, which become progressively larger over a period of time, until a functional fault results.

The rate at which the delay increases is such that the transition to functional fault occurs over many clock cycles. Therefore, *if periodic off-line testing is performed to ensure that no functional faults exist at the start of operation*, most new defects are expected to be detected before they cause functional faults. Therefore, it is appropriate to detect reliability failures as delay changes.

The properties of the most common reliability failure mechanisms in CMOS VLSI [Woods 86] are briefly described below.

Gate Oxide Shorts

Gate oxide shorts can be the dominant failure mechanism in some CMOS processes [Hawkins 85][Hawkins 86]. Pin-holes in the oxide form a resistive path between the gate and the source, channel or drain of a transistor. Leakage current can cause time-dependent breakdown of the oxide [Hawkins 86] [Yamabe 85], and increased propagation delay is common, as observed in [Hawkins 86] and as predicted by the circuit-level analysis in [Hao 91].

Hot Carrier Effects

Some of the carriers in the channel of an MOS transistor can gain enough energy to be injected and remain trapped in the gate oxide, changing the transistor's threshold voltage and transconductance. As more carriers become trapped over time, the propagation delay of the logic gate continues to increase [Hu 85]. Propagation delay versus time graphs are shown in [Hu 89].

Electromigration

As atoms are moved along a wire or a contact, either voids or hillocks can occur. The current density is greater in the constricted portion, so accelerated electromigration will cause further narrowing. Electromigration causing voids increases the wire resistance and RC time constant, resulting in an increasingly larger delay before an open circuit occurs. Hillocks can cause a bridging fault. During build-up, the change in coupling capacitance and eventual resistance between bridging lines will affect the timing of the CUT.

Transients

Temporary external disturbances such as power supply variations or electromagnetic interference can cause transient errors in the CUT signals, and need to be considered for on-line testing. If the duration of the transient is less than the checking period, then it will be detected by the stability checker if it caused an error. The reason is that if the sampled value is incorrect, the CUT output will switch back to the correct output within the checking period.

$$\Pr\{\text{detect error due to transient shorter than } T_{Stab}\} = 1 \tag{4.4}$$

If the duration of the transient is greater than the checking period, errors might not be detected, depending on how the output waveform is affected. Increasing the duration of the checking period increases the probability that a transient will be detected. The probability of detecting a long transient is approximately T_{Stab}/T_C at each affected output. Assuming that the CUT output is forced to a constant value for the duration of the transient (worst case), then the probability of detecting a long transient at each affected output is approximately:

$$\begin{aligned} \Pr\{\text{detect start of transient longer than } T_{Stab}\} &= \frac{T_{Stab}}{T_c} \Pr\{\text{Output at opposite level}\} \\ &\approx \frac{T_{Stab}}{2T_c} \end{aligned}$$

$$\begin{aligned} \Pr\{\text{detect transient longer than } T_{Stab}\} &= \Pr\{\text{Detect start or end of transient}\} \\ &= \frac{T_{Stab}}{2T_c} + \frac{T_{Stab}}{2T_c} - \frac{T_{Stab}}{2T_c}\frac{T_{Stab}}{2T_c} \\ &= \frac{T_{Stab}}{T_c} - \left(\frac{T_{Stab}}{2T_c}\right)^2 \qquad (4.5) \end{aligned}$$

Single event upsets (e.g. high energy particles) that cause state changes can be detected either directly in the flip-flop that toggles, or if the transition propagates through the combinational logic and causes a stability checking error in another flip-flop.

4.4.3 Performance Comparison with Other Techniques

In the same way that parity schemes rely on single errors being much more common than multiple errors, on-line Stability Checking relies on reliability failures appearing as delay changes first. As long as this is the case, it has almost the same performance as duplication with a significantly lower hardware cost.

Hardware duplication [Johnson 89] does not detect some common-mode problems that could be detected by Stability Checking. Temperature or power supply variations, for example, might affect both copies of the CUT in the same way, whereas excessive path delays due to the variations would be detectable by on-line Stability Checking.

Stability Checking can also be combined with duplication. Two copies of the circuit are compared, and when they differ, the circuit without stability checking errors is assumed to have the correct response. Once the failed circuit is removed, the remaining circuit still has Stability Checking. This approach seems to provide a level of fault tolerance similar to triple modular redundancy (TMR).

A simple form of time redundancy is to repeat a computation and compare results. Transient errors are detected by this technique, but unless precautions are taken, the same permanent error will appear in both computations. (One solution is recomputing with shifted operands [Johnson 89], but this only works for certain data-paths.) In comparison, on-line Stability Checking does not have the performance overhead, and there is no masking due to repeated errors.

The main benefit over parity checkers is that on-line Stability Checking can be used with any design, whereas parity prediction can be as expensive as duplication in general. (Circuits with a single output are a simple example.) A further benefit is that since each output is checked independently, there is no masking of multiple errors such as can occur in parity schemes. This makes it possible to diagnose the incorrect output if necessary.

4.5 PADDING SHORT PATHS

The performance of on-line Stability Checking depends on the duration of the checking period, and consequently the shortest path in the CUT. There is a tradeoff between longer checking periods that guarantee the detection of larger delay faults and increase the probability of detecting untargeted faults, and placing more severe restrictions on the CUT pathlengths. Most benchmark circuits investigated did not meet pathlength restrictions, so techniques for increasing the delay of short paths are presented in this section.

Since some paths are too fast, ideally, smaller gates with less drive can be used. Unfortunately this is not possible in all cases, since often there will be short and long paths through a gate, so a slower gate cannot be used. In general, extra padding elements are needed to increase the delay of short paths. *Padding* refers to the addition of extra delay in short paths in a circuit to meet timing requirements.

Eliminating short paths is more complicated for CMOS than for other technologies, due to the severity of CMOS *pattern dependent* delays. When there are parallel transistors, the drive strength of the gate depends on how many transistors are conducting. The delay for the 1111→1110 transition in a 4-input NAND gate, for example, could be about four times the delay for the 1111→0000 transition. Pattern dependent delays are used in the examples below.

It is shown that the hardware overhead to achieve reasonable checking periods (say $\frac{1}{2}T_c$) is generally small compared to duplication. The reasons are:

- Timing optimized designs have few short paths,
- Gate delay is a strong function of area.

These points are discussed below, before describing a simple algorithm for padding short paths and presenting experimental results in Sec. 4.5.4.

4.5.1 Timing-Optimized Circuits

The cost of padding short paths depends on the distribution of pathlengths in the CUT. The focus will be on timing optimized designs, as the emphasis is now often on performance over minimizing area. It has been shown that timing-optimized circuits approach the condition where all paths have the same maximum delay [Williams 91] [Park 91], reducing the need for padding elements. Shortening the longest path affects the padding of short paths in two ways. The timing optimizations tend to equalize path delays, increasing the length of short paths. Furthermore, short paths need less padding to reach a given fraction of a reduced longest path. (In fact, even for the ISCAS'85 circuits, it has been found that there are few very short paths [Cheng 92].)

4.5.2 Custom Cells for Padding Short Paths

For circuits where padding is necessary, the overhead for eliminating short paths can be reduced if customized gates are designed. A custom or standard cell design is assumed; the cost for a gate array design will probably be larger since the transistor geometries are fixed. The area of padding elements grows slowly as a function of delay because the delay depends on the active transistor area, which covers a small fraction of the cell area. As an example, the delay and area of various buffers in a LSI Logic standard cell library [LSI 91] is shown in Fig. 4.10. A cell five times the size of the LSI inverter has 34 times the delay.

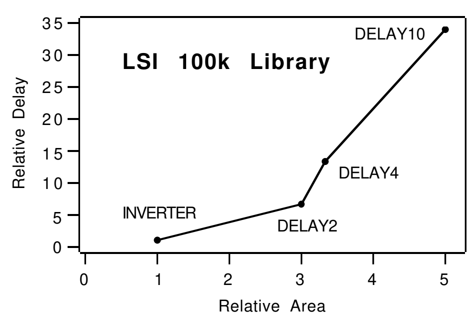


Figure 4.10. Delay versus Area for LSI Standard Cells

The cells in Fig. 4.10 are useful when large delays are needed for padding, but it was found that cells with many intermediate values of delay were also necessary. The idea is to have multiple cells with slightly different delays, and then choose the optimal one to tune pathlengths. Designing multiple cells with similar footprints also simplifies post-routing replacement of cells to further improve pathlength tuning. There are two kinds of padding elements, shown in Fig. 4.11. If there is timing slack at a gate, then a slower gate can be used. If only some of the fanout branches have slack, then a separate padding element must be inserted.

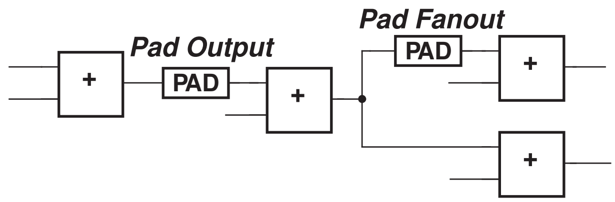


Figure 4.11. Two Types of Padding Elements

As examples of both types of padding elements, the buffer and 2-input NAND gate in the CMOS3 standard cell library [Heinbuch 88] were modified. The CMOS3 library was used, as the cell layouts are available. The cells were modified using MAGIC, and SPICE simulations were performed on the extracted layouts. The input capacitance and drive strength of the cells remained virtually the same. Figure 4.12 shows that a buffer with 2.8 times the delay of the original buffer is only 14% larger, and a NAND gate 2.4 times the delay of the original NAND is only 17% larger. (Incidentally, the pattern dependent delay for both inputs falling is 56% of the maximum delay for the NAND2 gate.) There is a limit to the delay of cells designed by resizing transistors, as the signals eventually change very slowly and are prone to noise. These cells are used in the examples in the next sections.

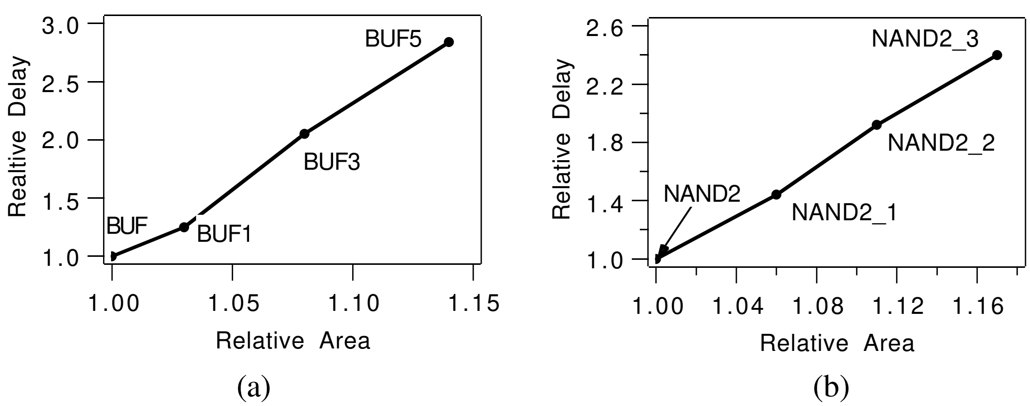
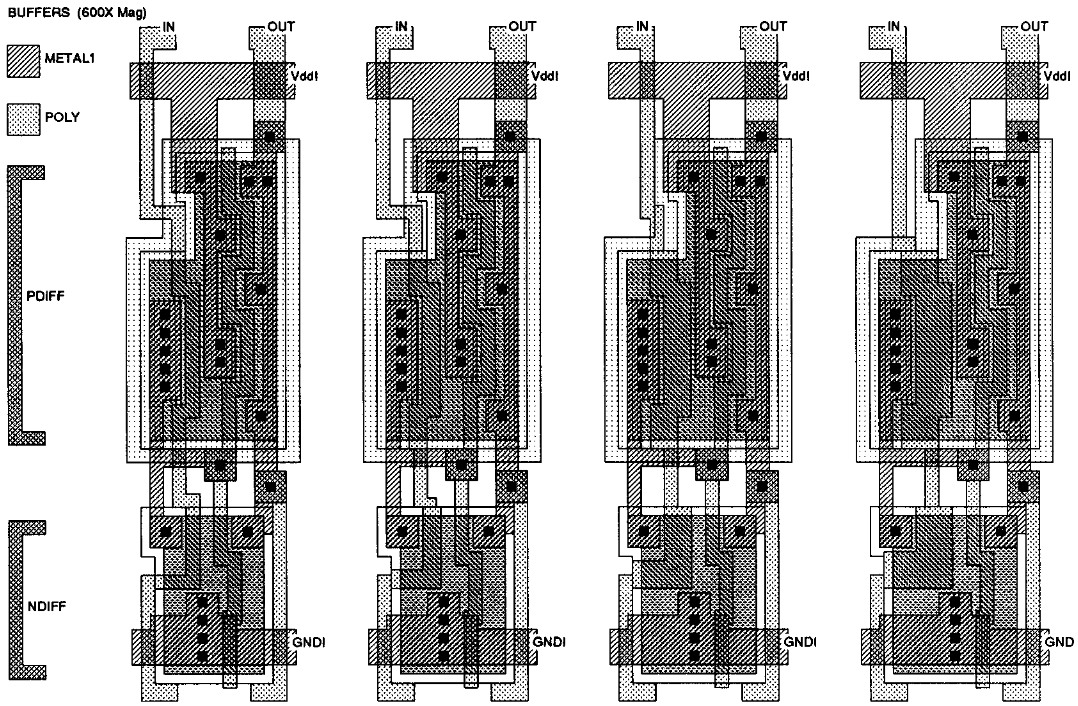


Figure 4.12. (a) CMOS3 Buffer and Derivatives, (b) CMOS3 NAND2 and Derivatives © IEEE [Franco 94a]

The MAGIC layout of the buffers is shown in Fig. 4.13 below. The new buffers are formed by increasing the length of the transistors in the first inverter.



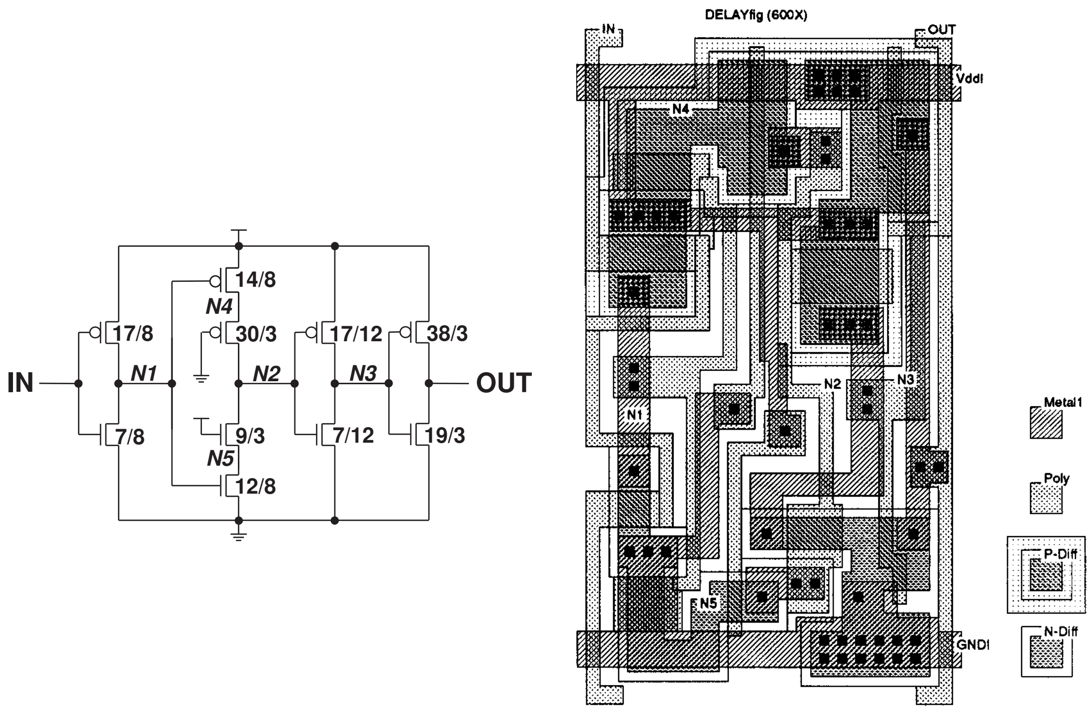
BUF BUF1 BUF3 BUF5

Figure 4.13. The CMOS3 Buffer, and 3 Derivatives

For cases where large buffers are required, the DELAY cell in the CMOS3 library can be efficiently tuned. The circuit diagram and layout of the DELAY cell are shown in Fig. 4.14, and it can be seen that there is scope for resizing the transistors without increasing the area of the cell.

4.5.3 Padding Example Using Logic Synthesis Tool

All circuits need to meet minimum pathlength restrictions to prevent flip-flop hold time violations, so in principle, logic synthesis tools could be used for padding short paths by specifying long hold times for flip-flops. In this section, the short paths in the ALU181 are padded using the logic synthesis tool Synopsys [Synopsys 91] as an example. Unlike MIS [Brayton 87] where the synthesis steps are controlled by a script, Synopsys has a compile function that does both logic synthesis and technology mapping. Multiple iterations of the compile function were performed with Boolean optimization turned on and the mapping effort set to “high”, to get a highly optimized design.



Circuit Layout

Figure 4.14. DELAY Cell in CMOS3 Library

The ALU181 was first synthesized for minimum delay, and mapped to a subset of the CMOS3 library consisting of 2 and 3-input NAND and NOR gates, buffers and inverters. NAND and NOR gates were used for a worst-case analysis, since they have more severe pattern dependent delay characteristics than either AND or OR gates (these gates have an output inverter that does not have pattern dependent delay). The results are shown in Table 4.2. There are two short path columns: $T_{short}*$ was computed without taking CMOS pattern dependent delays into account, and is significantly longer than the conservative worst-case minimum path, T_{short}.

Iteration 4 was selected as the "best" timing-optimized circuit, as it is significantly smaller than Iteration 5, and only a little slower. The circuit in Iteration 4 was then synthesized with both maximum and minimum pathlength constraints. The maximum delay was kept the same to avoid any performance penalty, and the target minimum delay was 32 units. The target library was extended to include the modified gates discussed in the previous section, and shown in Fig. 4.12. (The modified gates would have no effect on the minimum delay circuits, since they are both slower and larger than the original gates.)

Table 4.2. Synopsys Results for ALU181

Constraint		Gate Area	Wire Area	Total Area	T_{long}	$\frac{T_{short}*}{T_{long}}$	$\frac{T_{short}}{T_{long}}$
Min. Delay	Iter. 1	4416	1164	5580	71.52	9.8%	7.3%
	Iter. 2	4212	1107	5319	62.67	11.2%	8.4%
	Iter. 3	4044	1059	5103	55.51	12.7%	9.5%
	Iter. 4	**3804**	**1014**	**4818**	**55.72**	**12.6%**	**9.4%**
	Iter. 5	4596	1126	5722	53.93	18.2%	12.4%
Padded (32,56)		**4461**	**1070**	**5731**	**55.99**	**53.6%**	**41.4%**

For this example, the area overhead for increasing the minimum delay from 9.4% to 41.4% of the longest path is 19%. The path lengths in the minimum-delay timing-optimized (Iteration 4) and padded implementations of the ALU181 are shown graphically in Fig. 4.15. There is a vertical bar starting from the minimum delay and ending at the maximum delay for each of the eight outputs.

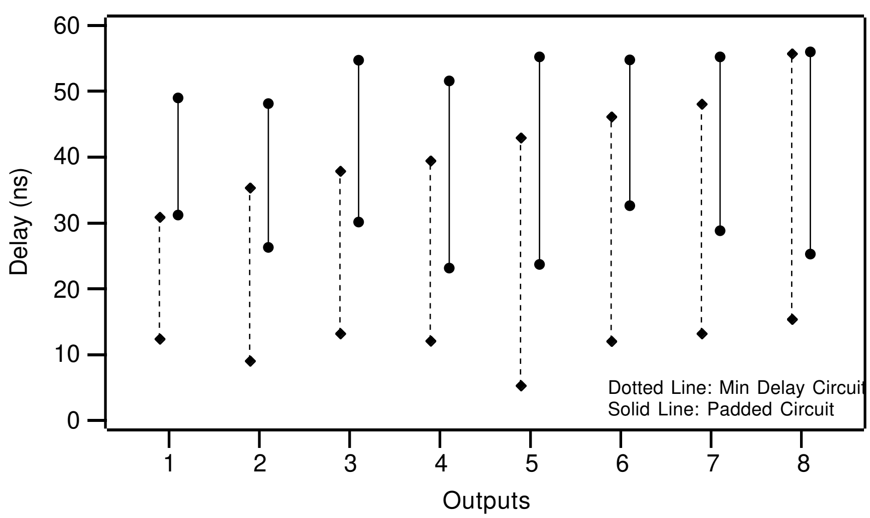


Figure 4.15. Graphical Representation of Path Lengths in ALU181

The minimum delay could not be increased beyond 41% without affecting the longest path in the circuit (Output 8). This limitation was even more severe for some of the other benchmark circuits investigated.

It seems that Synopsys did not make full use of the modified cells for padding. For example, it was found that the slower gates shown in Fig. 4.12 were not included in the final netlist, even though they would reduce the area overhead. The solution adopted was to use a target library with incorrect areas for these gates, in order to guide the logic synthesis. By specifying the slower gates to have less area than the faster gates, and then optimizing for minimum delay, the slower gates were always used whenever possible. The circuit was then linked to the correct library to compute the actual circuit area.

Even though it appears as though Synopsys cannot be used in its present form to pad short paths enough for on-line stability checking, the example illustrates a few points:

- CMOS pattern dependent delays should be used in the timing analysis since they alter the short path delays significantly,
- Timing-optimizations tend to increase the ratio of shortest to longest paths,
- The total area overhead is less than gate area overhead since wiring is relatively constant.

4.5.4 Algorithm for Padding Short Paths

Since synthesis tools are not optimized for very long minimum path constraints, and sufficient padding was not possible for some of the benchmark circuits investigated without affecting the longest path in the circuit, an algorithm specifically for padding short paths is presented in this section.

Padding algorithms to meet minimum path constraints have been implemented for wave pipelining, where all paths are required to have the same length [Wong 89] [Shenoy 93]. Considerations specific to equalizing path lengths in CMOS circuits for wave pipelining are described in [Klass 90] [Klass 92] [Gray 91]. Typically, buffers are first inserted using either graph-based techniques [Wong 89], or circuit levelization [Klass 90]. The number of padding elements is then minimized using linear programming, or non-linear programming if loading is taken into account. It should be noted that padding of short paths is *always* possible, as long as buffers with suitable delays can be inserted [Shenoy 93].

Since non-linear programming problems are computationally expensive to solve, a simpler algorithm is presented in Table 4.3. This algorithm does not affect the longest path in the circuit. The simplifying assumption made is that the maximum allowable delay is added to each chosen node. This means that `Pad_Short_Paths` is a greedy algorithm, as each node is padded at most once.

A figure of merit is computed for each node, based on the slack at the node and the desired added delay necessary to eliminate short path problems. The node with the highest figure of merit is then padded, and the process is repeated until the shortest path is the desired fraction of the longest path. The performance of the algorithm depends on the choice of nodes to pad. After experimenting, the following three figures of merit were used for the benchmark circuits below. (The absolute value is used in `merit_2` as $\mathtt{desired}_i$ is negative if there is extra delay at the node.)

$$\text{merit\_1} = \text{slack}_i * \text{desired}_i$$

$$\text{merit\_2} = \text{slack}_i * \text{desired}_i * |\text{desired}_i|$$

$$\text{merit\_3} = \min(\text{slack}_i, \text{desired}_i)$$

Table 4.3. Greedy Padding Algorithm

```
Procedure Pad_Short_Paths;
   Add_fanout_pseudo_gates;
   Repeat until Tshort  ≥ Target_Short_Path {
     For each node i do {
        slacki = TC - (longest path from inputs to node)
                    - (longest path from node to outputs);
        desiredi = Target_Short_Path
                   - (shortest path from inputs to node)
                   - (shortest path from node to outputs);
        meriti = f(desiredi, slacki, gate_typei);
     }
     Choose i to maximize meriti;
     Pad node i by min(desiredi, slacki);
   }
}
```

The algorithm also has the following refinements:

- Different rising and falling delays, gate strengths, and fanout loading are taken into account.
- CMOS pattern dependent delays are used. The delay used for the 1111→1110 transition in a 4-input NAND gate, for example, is four times the 1111→0000 delay. The longest and shortest delay are used to compute the longest and shortest pathlengths respectively.
- A scaling factor is used to distinguish real gates from fanout pseudo-gates. The reason is that it is more efficient to pad real gate outputs rather than fanout branches whenever possible. The figure of merit is divided by an experimentally derived heuristic value *h* for fanout pseudo-gates.
- Added delays on the different fanout branches of a stem are combined whenever possible.
- After the padding, some nodes have too much added delay. The extra delay was removed using the heuristic:

  $$\text{merit\_remove} = \max(-\text{desired}_i, \text{added\_delay}_i)$$

  This reduced the area overhead figures by 2-3% for the benchmarks tried.

Table 4.4 shows some benchmark results using `Pad_Short_Paths`. The "s" circuits are from the ISCAS'89 sequential benchmark suite. The target is to ensure that the shortest path, T_{short}, is at least 60% of the longest path, T_{long}. The area overhead is estimated based the cells designed in Sec. 4.5.2. Interpolation is used to approximate the area for cells that have not been designed. SIS [Brayton 90] was used for both logic optimizations and technology mapping. The standard script for minimizing delay improved the performance of 6 benchmark circuits.

Table 4.4. Results for Greedy Padding Algorithm © IEEE [Franco 94a]

Circuit	Initial Circuit Parameters					Area overhead for $T_{short} \geq 60\%\ T_{long}$			
	SIS Opt.	Flip Flops	Gate Area	T_{long}	$\frac{T_{short}}{T_{long}}$	merit_1 (gate area)	merit_2 (gate area)	merit_3 (gate area)	Total Area Overhead
alu181	No	8	4,500	49.24	22.4%	11.7%	11.9%	10.5%	11.9%
s208	No	3	3,444	57.76	7.2%	52.3%	51.2%	45.1%	41.7%
s298	Yes	4	4,368	36.57	10.3%	23.0%	22.8%	23.7%	21.8%
s344	Yes	15	5,676	51.56	7.0%	50.1%	48.7%	44.6%	37.8%
s349	Yes	15	5,676	51.56	7.0%	50.1%	48.7%	44.6%	37.8%
s382	No	21	6,036	42.37	7.3%	30.0%	30.1%	27.7%	25.4%
s386	Yes	6	6,888	46.99	6.6%	28.0%	27.8%	27.1%	26.2%
s420	No	16	6,780	120.1	3.5%	143.4%	136.8%	112.4%	89.7%
s444	Yes	31	6,792	51.35	6.0%	36.2%	34.5%	33.8%	29.8%
s510	Yes	6	8,844	46.29	19.4%	16.3%	16.5%	17.3%	16.5%

The first three area overhead figures in Table 4.4 are the gate overhead to meet the timing requirements for the different heuristics. The Total Area Overhead column is based on the best heuristic for each circuit, and takes into account both the combined flip-flops and stability checkers, as well as the cost of ORing the outputs of the stability checkers. The overhead is low for most circuits, except for s420 which is close to duplication (100% + checker). This is the slowest circuit by far. The results represent a conservative upper bound since the algorithm is not optimal, and wiring area is considered. The example in Sec. 4.5.3 shows that estimated wiring area increased more slowly than gate area.

Short path problems can also be reduced as an integral part of logic synthesis by placing constraints on the different steps of the synthesis process. Promising optimizations seem to be factoring and decomposition of Boolean functions [Brayton 87] [Brayton 90]. The aim is to factor the function F as F = GH + R. Restricting the sizes of the different factors would eliminate a potential short path problem that occurs if R is much simpler than GH. Similar restrictions can be placed on candidate nodes for resubstitution. Developing such an algorithm is beyond the scope of this work.

Other CAD tools such as routing tools can also be used to increase delays of short paths, since slower interconnect lines with more vias can be assigned to these paths.

4.6 EXTENSIONS

On-line Stability Checking is also possible for other clocking schemes, such as two-phase double latch designs [McCluskey 86], for example. The biggest difference is that the checking period must be derived from both clock phases, as discussed in Sec. 3.2.2.

Other extensions of on-line Stability Checking are described in this section. Applications in other areas such as software checking and VHDL synthesis are also discussed.

4.6.1 Stability Checking versus Final-Value Checking

Consider a CUT output that is correct at the sampling time, then has transitions, before settling to the correct value again. This situation would be detected as an error by Stability Checking, and could be considered a "false alarm". If one is only concerned with errors due to small delay faults, then false alarms can be reduced by comparing the sampled value at the beginning of the checking period to the value at the end of the checking period, instead of looking for changes in the output throughout the checking period. This will be called *final-value checking*. Final-value checking is similar to time redundancy, except that instead of a performance penalty, consecutive operations are overlapped in time. This is only possible because of the timing restrictions placed on the CUT. (This is the same concept as wave pipelining.)

Stability Checking detects at least as many errors as final-value checking, since if the values at the start and end of the checking period are different, there must have been at least one transition during the checking period. Stability Checking also has a greater probability of false alarm, as the sampled value could be correct, and yet be followed by transitions.

The reliability requirements and operating environment of the system will determine which method is more suitable, depending on whether the increased probability of detecting untargeted faults by Stability Checking outweigh the effect of increased false alarms.

The checkers for final-value checking are similar to those for Stability Checking.

4.6.2 Multiple Checking Periods

A single checking period has been used in the above discussions. In principle, however, a different duration checking period could be used for each output, although it is probably not practical to have more than a few distinct checking periods. In this way, much larger checking periods could be used for some outputs, since they are not limited by the global shortest path. All the checking periods start at the same time, so could be generated using clock choppers [Wagner 88] near the corresponding CUT. For example, a longer checking period could be used for an ALU than a barrel shifter.

If all paths to an output are short, it is possible to start the checking period before the clock edge. This increases the probability of detecting untargeted faults, but will detect delay changes before they produce system errors. Delay changes that don't affect the operation of the circuit are called delay *flaws*, and are generally not tested. This can be used as a process monitor to predict system degradation before failure, or help in detecting intermittent faults.

4.6.3 Self-Timed Clock Frequency

There are two ways to test the stability checkers without adding extra signals. The clock frequency can be increased to induce delay "faults", or the duty cycle of the clock can be increased to violate equation (4.1) and cause "short path" faults. This can also be used to optimize the performance of a design, by tuning the clock frequency and duty cycle to match individual CUT characteristics. It is conceivable that for applications where repair is not possible, the clock frequency can be dynamically reduced when timing errors occur, in an attempt to "fall back" to a reduced performance state before failing completely.

4.6.4 Software Stability Checking

An interesting application of Stability Checking seems to be run-time software checking. The software equivalent of checking a signal for stability is checking if a variable is modified. Variables are tagged to indicate when their values can be changed. Compiler analysis tools (reachability, liveness) can be used to determine when variable assignments are possible. If a runtime error occurs and a write is attempted in the incorrect sequence, the error can be detected. Note that this is more than just scoping [Aho 85], since scoping rules are checked at compile time and only offer protection for error-free program execution. A simple mechanism for detecting errors is to cause a system trap or access violation, by considering the variable to belong to another process when it cannot be modified.

This is similar to using watchdog processors for control flow checking [Mahmood 88], but it can be implemented entirely in software on any system. Control flow errors are detected when a write is attempted on a variable unexpectedly. Bus errors that corrupt the write address can also be detected.

Not every variable needs to be checked, although increasing the number of checked variables increases the error coverage at the expense of performance overhead. It is likely that only state variables will be checked, and not all local variables.

For example, if a variable is going to be modified in a certain block after a branch, writing could be activated for that variable just before the branch. If there is a control flow error, and the variable assignment is not reached from the correct branch, writing to the variable would cause an error. Writing to a variable could automatically disable further writing to that variable. In cases where a variable is assigned in many places in the program, temporary variables can be used, and each checked individually.

4.6.5 VHDL Synthesis

Stability Checking has been described in the context of detecting delay faults in this dissertation. There are other applications, however, where stability checkers can be used, as signal changes are sometimes used to trigger actions during normal system operation. An unclocked memory, for example, can be built such that when there is a change in one of the address lines, the new output value is produced. Another interesting application of on-line Stability Checking appears to be VHDL synthesis.

Although VHDL was originally intended as a simulation language, synthesis from VHDL descriptions has drawn increased attention. Process statements are one of the basic building blocks of VHDL descriptions. A process is activated whenever any of the signals in its sensitivity list changes [IEEE 88], which is conceptually similar to Stability Checking. The actual implementation depends on the body of the process statement, as shown by the examples in Table 4.5. Assuming A and B are input signals, and Z is an output signal, Process_1 corresponds to an AND gate, and Process_2 corresponds to a positive edge-triggered flip-flop. Process_3 is more complicated, however, and cannot be implemented using a standard logic element.

Table 4.5. VHDL Process Statements

Process_1 (A,B)
  begin
    Z <= A AND B;
  end process;
	Process_2 (A)
	  begin
	    if A = 1 then
	      Z <= B;
	    end if;
	  end process;
		Process_3 (A)
		  begin
		    Z <= A AND B;
		  end process;

Complex VHDL descriptions consist of multiple processes that are synchronized by signal changes. One proposed technique for synthesizing multi-process descriptions is to synthesize each process independently, and then synthesize the control and synchronization logic. Stability checkers could be used to detect any changes in signals during system operation to determine when processes need to be activated.

4.7 CONCLUSION

A new technique for on-line checking of digital systems has been proposed. By targeting the expected failure modes in CMOS VLSI, on-line Stability Checking achieves high coverage of most common wear-out failure mechanisms and transient errors in CMOS circuits at a fraction of the cost of duplication. On-line stability checkers behave like "mini-watchdogs" that detect signal changes at unexpected times.

The main limitation of on-line Stability Checking is that catastrophic functional faults are not guaranteed to be detected. As in all testing, there is only a certain probability of detecting real defects. Parity checking, for example, does not detect many classes of errors, yet it is widely used. In the same way, there are classes of applications where on-line Stability Checking provides the best cost/performance tradeoff.

Typical applications are aggressively clocked, optimized, high speed systems, which are becoming increasingly common. The aggressive clocking makes marginal timing problems more important, and the optimized, pipelined designs help to equalize path delays and reduce the cost of Stability Checking.

A large cell library is necessary for meeting pathlength restrictions efficiently. However, this places no burden on the designer, as the padding can be automated as part of logic synthesis or technology mapping.

Chapter 5

Pre-Sampling Waveform Analysis

This chapter describes Pre-Sampling Waveform Analysis, the second class of Output Waveform Analysis techniques. The advantages of Pre-Sampling Waveform Analysis are discussed, and different waveform analysis techniques are mentioned. One waveform analysis technique is then described in detail in the rest of the chapter. Examples are given, and circuits for performing the waveform analysis are shown.

5.1 DESCRIPTION

As shown in Fig. 5.1, the output waveform of the CUT between the application of the second pattern in the test pair $<V_2>$, and the sampling time, is analyzed in Pre-Sampling Waveform Analysis. The objective of Pre-Sampling waveform analysis is to use the information in this part of the waveform to infer the delays in the circuit.

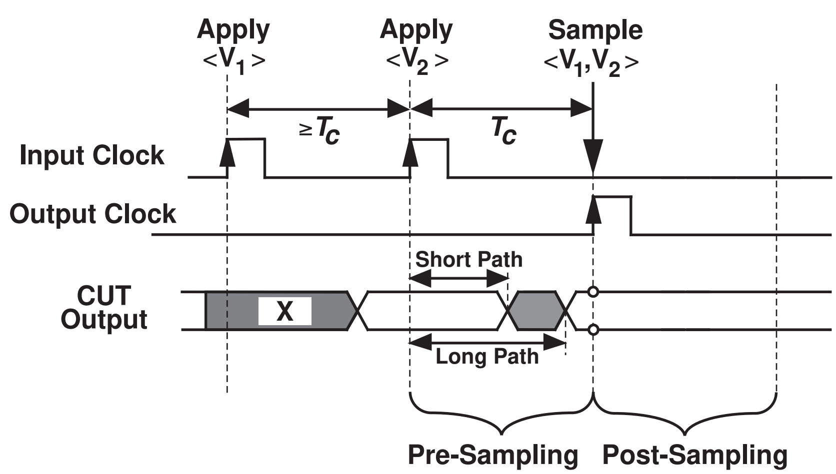


Figure 5.1. Output Waveform Analysis

The biggest difference between Pre-Sampling Waveform Analysis and Post-Sampling Waveform Analysis described in the previous chapters, is that in Pre-Sampling Waveform Analysis both the faulty and fault-free waveforms can have transitions. Therefore waveform analysis techniques must be found that distinguish the two responses, rather than only detecting transitions. Information is extracted from the output waveform of the CUT, which must be compared to the corresponding information from the fault-free response. This makes Pre-Sampling Waveform Analysis more complex than Post-Sampling Waveform Analysis. This complexity is offset against the capability to detect delay flaws, as well as delay faults.

5.1.1 Delay Flaws

One of the main benefits of Pre-Sampling Waveform Analysis is that, by analyzing the output waveform before the sampling time, timing failures that do not cause delay faults can be detected. These timing failures were called *delay flaws* in Chapter 2.

Delay flaws do not affect the output waveform after the sampling time, so cannot be detected from the sampled value or by Post-Sampling Waveform Analysis.

At first, it might seem that there is no need to detect delay flaws that do not cause incorrect operation. This is true in many applications. However, if the delay of a part is different from the expected value, this is an indication that the part has not been manufactured correctly or it has degraded, and the part might not have the required reliability for certain applications. As a simple example, consider a pace-maker circuit. If the delay of an inverter in the circuit should be 10 ns but is actually 70 ns, and the pace-maker still works correctly, would you want that circuit?

Pre-Sampling Waveform Analysis can be used as a reliability screen both during manufacturing test and in the field. During manufacturing test, costly environmental stress screening can be reduced by first weeding out weak parts that have delay flaws, or by detecting parts with abnormal delay characteristics before the part fails during burn-in test.

As discussed in the previous chapter, most common CMOS reliability failure mechanisms change the delays in a circuit before causing catastrophic (stuck-type) faults. Therefore, by periodically monitoring a system in the field, potential reliability failures can be discovered before the system fails. This can be used for preventative maintenance, for example.

5.1.2 Delay Faults

Pre-Sampling Waveform Analysis can also be used to detect delay faults. One benefit of Pre-Sampling Waveform Analysis is that delay faults do not have to be sensitized through the longest path to be detected. This makes detecting small delay faults much easier, especially for pseudo-random tests.

Throughout this chapter, different Pre-Sampling Waveform Analysis techniques are compared to conventional delay testing. Localized or gate delay faults were used, and the fault simulations were done with the waveform simulator described in [Franco 94c].

For gate delay faults, the fault coverage depends on both the location of the delay fault, and the magnitude of the delay fault detected [Park 88]. If the distribution of delay faults is known, then a statistical fault coverage measure has been proposed [Park 88].

No assumption is made about the delay fault distribution in the fault coverages computed below. Usually five delay faults with different values were injected at each gate output. All the delay faults injected were greater than the slack (structural) at each node, otherwise the added delay would not be detectable by conventional delay testing, making the comparison more difficult.

5.1.3 Waveform Analysis Functions

Stability Checking and Final-Value Checking were the most useful Post-Sampling Waveform Analysis techniques, but there are many techniques for analyzing the output waveform of the CUT before the sampling time.

One extreme is to try to store as much of the waveform as possible. This can be done by sampling the waveform many times during the clock period. As long as the waveform is sampled at more than twice the highest frequency component in the waveform, the original waveform can be recovered. This technique is generally only feasible on a very high speed ATE.

The other extreme is to narrow the area of interest to a single point, and sample the output waveform once before the normal sampling time. Moving the sampling time has been described in [Pramanick 89] [Mao 90a], where the sampling time is adjusted for each vector. One problem with this approach is that it is difficult to precisely control the sampling time for each vector.

Counting the number of transitions in the output waveform was considered as another possible waveform analysis function. In the presence of a delay fault, the transitions in the output waveform will move in time. Therefore it is reasonable to expect that the number of transitions between $<V_2>$ and the sampling time will change. Figure 5.2 shows the fault coverage for both sampling the output (conventional testing) and counting the number of transitions in the output waveform. The smallest delay fault injected at each node was the slack at the node plus one gate delay, and the largest was the slack plus five gate delays. The number of transitions for each vector and output was compared to the fault-free circuit. The simulation shows that counting transitions does detect delay faults, although the benefit does not justify the complexity of the method. Circuits for counting multiple transitions between cycles are complex, and are usually not practical, particularly for BIST applications.

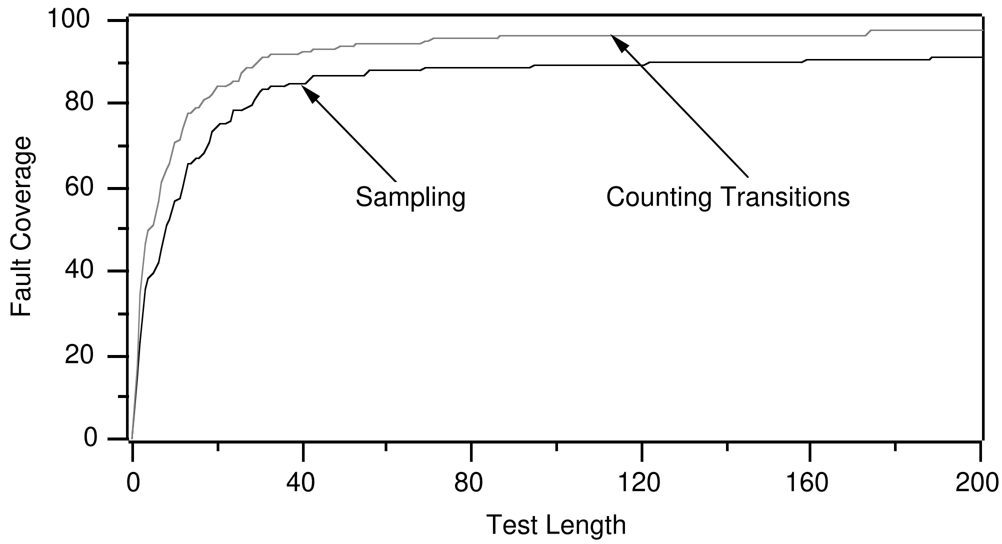


Figure 5.2. Fault Coverage for ALU181 for Sampling and Counting Transitions

The waveform analysis technique that was eventually chosen is the analog *integral* of the output waveform. Unlike the other techniques mentioned above, this technique is feasible even for Built-in Self-Test environments. Integration is described using examples in the next section, and implementations are shown in Section 5.3.

There is a loss of information in any waveform analysis function that compacts the information in the output waveform, so masking can occur. This means that although the faulty and fault-free waveforms may be different, the computed waveform analysis functions can be the same. Techniques for reducing the masking or aliasing probability for integration are mentioned below.

5.2 INTEGRATION

Different forms of integration are discussed below with examples.

5.2.1 Integration Over Whole Cycle

The simplest integration function is to integrate the output waveform of the CUT $x(t)$ from the application of $<V_2>$ to the sampling time when the output is latched:

$$I = \int_0^{T_C} x(t)dt \tag{5.1}$$

Note that the integral in equation (5.1) is equivalent to the *average value* of the waveform over the interval. (The average value of the waveform is actually I/T_C, so the two measures only differ by a scaling factor.)

A fault is considered detected if the fault-free integral I_{FF}, and the faulty integral I_F, differ by more than a certain amount, as shown in equation (5.2). This amount depends on RES, the resolution of the integrator and quantizing effects, as integral values that are very close cannot be distinguished. There is a tradeoff between the resolution of the integrator, and the complexity of the integrator or the time taken to compute the integral. Circuit tolerances also need to be taken into account unless relative integrals are compared. This is discussed in the implementation section.

$$|I_F - I_{FF}| \geq \frac{T_C}{RES} \tag{5.2}$$

The usefulness of integration as a pre-sampling waveform analysis function is best shown by example. The circuit in [Pramanick 89] is used for the two examples below. Fault detection size is considered in the first example, and delay flaws are considered in the second example.

Example 1: Input P Slow-to-Fall

Consider a pair of test patterns with a 1 to 0 transition at P, and Q=R=S=1, which produce a hazard-free 0 to 1 transition at the CUT output X, for the circuit in Fig. 5.3. In [Pramanick 89] it was shown that although the slack at input P is 1 unit, the fault detection size is 2 units for robust delay tests. It was also proposed to move the sampling time from 7 to 6, in order to reduce the fault detection size to the slack. Using the integral of the output, the same result can be achieved without moving the sampling time.

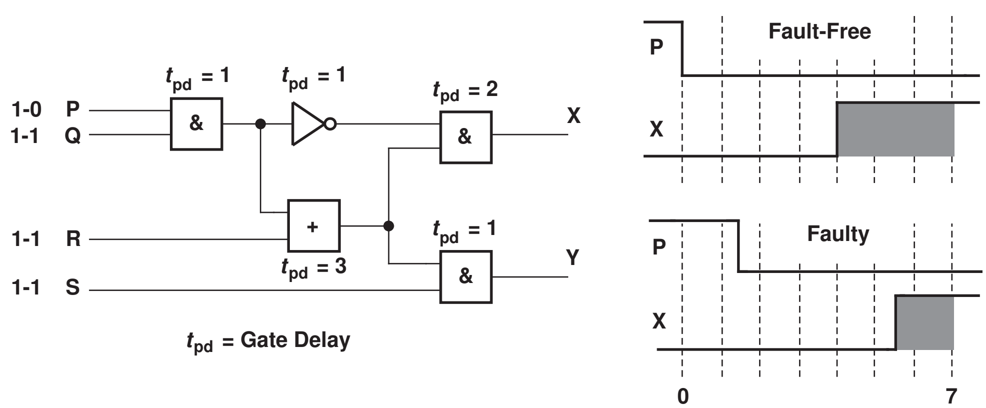


Figure 5.3. Slow-to-Fall Fault at Input P

Output waveforms for both the fault-free circuit and a 1.5 unit slow-to-fall fault at node P are shown in Fig. 5.3. The fault-free integral is 3 units, and the faulty integral is

1.5 units. Therefore not only is the delay fault detected by this pair of patterns, but one can infer precisely when the transition in the output occurred. The integral at output X is 3-d for a slow-to-fall fault of size d at node P. Note that the delay fault was detected even though it was not sensitized through the longest path.

All 64 pairs of patterns for inputs Q, R and S were simulated for the same fault, and the results are shown below. Therefore even if integral differences of 0.5 cannot be detected due to the resolution of the integrator, almost half the possible pattern pairs detect the fault.

# Pattern Pairs	28	8	4	24
Diff. in Integral	0.0	0.5	1.0	1.5

Example 2: Inverter Delay

The delay in the inverter in Fig. 5.4 is considered as a second example. Since the slack at the inverter is 3 units, delay faults less than 3 units in the inverter will not cause circuit malfunction and will not be detected with conventional delay testing (no delay faults). Here we show that using integration, delays less than the slack of a node can be detected.

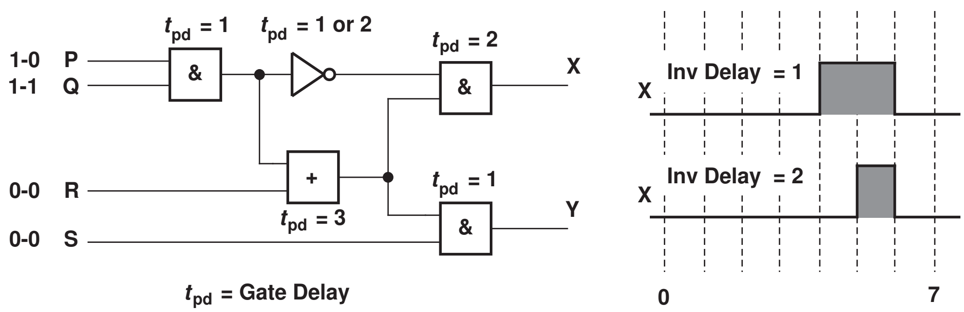


Figure 5.4. Inverter Delay

Consider the waveform at output X shown in Fig. 5.4 for delays of 1 and 2 units in the inverter, for a 1 to 0 transition at P and Q=1, R=S=0. The integral value at the output is 2 and 1 respectively, so the change in the inverter delay can be detected at the output. For inverter delays t_{pd} less than 3 units, the integral at X is 3 - t_{pd} for the given test pattern pair.

All 256 possible pattern pairs were simulated for the fault-free circuit and the faulty circuit with a delay of 2 units in the inverter. The delay flaw was detected by 37.5% of the pattern pairs using integration, even though there is no delay fault.

These examples show that the output waveform between samples contains information about the delay of the circuit. Therefore using integration as a Pre-Sampling Waveform Analysis function, delay flaws undetectable with conventional delay testing schemes can be observed if desired.

Although somewhat simplified, the above examples indicate what is achievable with integration as a waveform analysis function. Both examples are cases that are difficult to deal with using conventional delay testing, but have high detectabilities using Pre-Sampling Waveform Analysis.

5.2.2 Integration Over Part of Cycle

The output waveform was integrated over the complete cycle in the above examples, but more generally, the output waveform can be integrated over any part of the cycle, as shown in equation (5.3). The advantage of reducing the *integrating period* is that small delay changes can be detected without increasing the resolution of the integrator. This helps to reduce aliasing of faulty and fault-free integrals. The disadvantage is that delay changes that only affect the output waveform beyond the integrating period are not detected. The most useful integrating period depends on the CUT. For example, if the cycle time is much longer than the longest path, then it is better to integrate the output waveforms in the first part of the cycle. In most cases, however, it is better to integrate in the last part of the cycle when the outputs have started changing.

$$I(\alpha,\beta) = \int_{\alpha}^{\beta} x(t)dt \tag{5.3}$$

Assuming that the waveform over a fraction f of the cycle is integrated, the modified condition for detecting the fault is given in equation (5.4).

$$\left|I_F - I_{FF}\right| \geq \frac{f\,T_C}{RES} \tag{5.4}$$

The benefit of integrating over part of the cycle is shown in Fig. 5.5. Five detectable delay faults were injected at each gate output. The size of the delay faults was slack plus 5 to slack plus 25. (The two-input NAND delay is 20.)

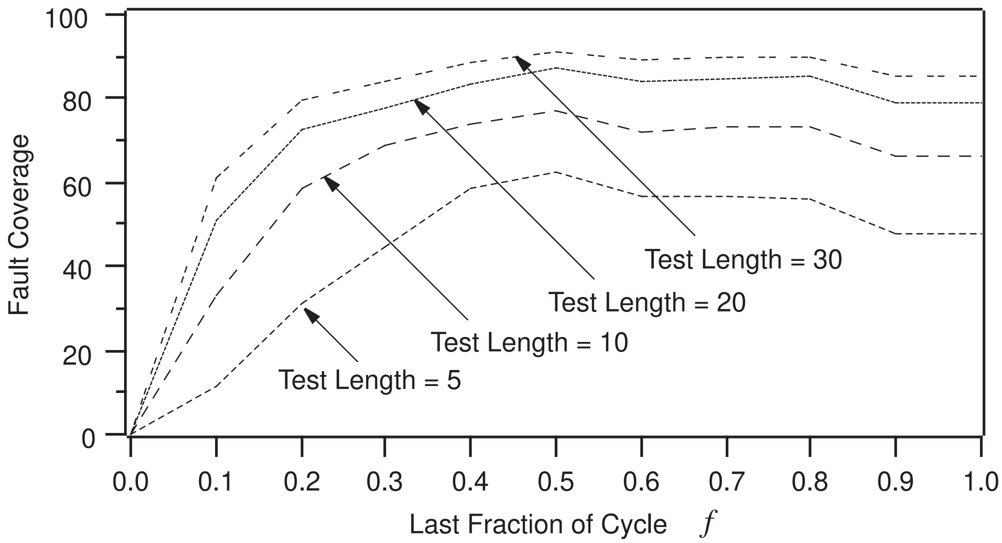


Figure 5.5. Fault Coverage for ALU181 Integrating Over Last Part of Cycle

Figure 5.5 shows the fault coverage for various test lengths as a function of the fraction of the cycle that is integrated. The last part of the cycle was integrated, as the cycle time was chosen close to the maximum delay through the circuit. The resolution of the integral was $RES=10$.

As expected, the coverage starts at 0 if the integrating period is very small. For this circuit, an integrating period of half the cycle is best, and increasing the integrating period further results in loss of coverage since the smallest delay faults do not meet the minimum detectability criterion in equation (5.4).

5.2.3 Enhanced Integration

The integrand in the above cases has always been the output waveform $x(t)$, although any function $g(x(t),t)$ can be used. For example, by integrating $t\,x(t)$, changes in the later part of the cycle are weighted more heavily that changes in the early part of the cycle. This places more importance on changes near the sampling time.

Using an enhanced integrating function is another way of reducing the possibility of aliasing. For example, consider a 1-pulse starting at A and ending at B, during the checking period. This is represented as $x(t) = (A\ B)$, using the waveform notation in [Franco 94c]. The integral value, given by equation (5.5), depends only on the thickness of the pulse, B-A. Therefore, delay changes that move the pulse without changing the width of the pulse are not detected.

$$\int_{\alpha}^{\beta}(A\ B)dt = B - A \qquad if \quad \alpha \le A \le B \le \beta \tag{5.5}$$

Equation (5.6) shows the integral if the integrand is $t\,x(t)$. In this case, the integral depends on the pulsewidth, as well as the position of the pulse, making aliasing more difficult.

$$\int_{\alpha}^{\beta}(A\ B)t\,dt = (B - A)\left(\frac{A + B}{2}\right) \qquad if \quad \alpha \le A \le B \le \beta \tag{5.6}$$

5.2.4 Fault Coverage Examples

Three fault coverage examples are given in this section. The resolution was *RES*=10 and *RES*=20 when integrating over the complete cycle, and *RES*=10 when integrating over the last half of the cycle. Faults of size slack plus one gate delay to slack plus five gate delays were injected at each gate output.

Figure 5.6 shows the fault coverage for the ALU181. Integration reached 100% fault coverage with 28 pseudo-random vectors, whereas sampling the output reached about 90% after 200 vectors. For this circuit, there was no increase in coverage by using the higher value of *RES* or integrating over the last half of the cycle. The reason is that the cycle time is 180 units, and the smallest injected fault was 20 units, so there were no fault coverage losses due to aliasing. This is not true for the larger circuits simulated below.

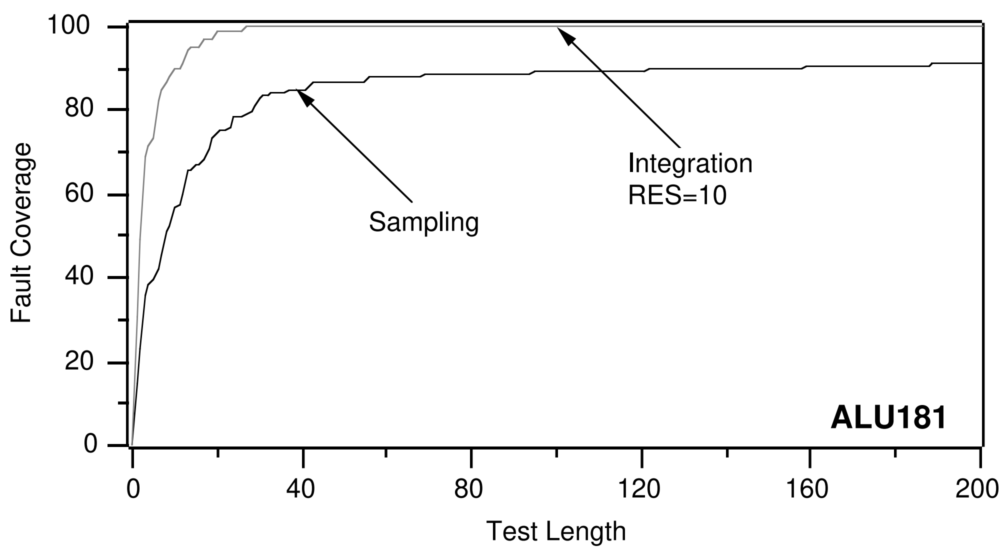


Figure 5.6. Fault Coverage for ALU181

The second and third examples are from the ISCAS'85 combinational benchmarks. The fault coverage for c432 is shown in Fig. 5.7. For this circuit, the coverage using sampling increases slowly, and is less than 70% after 200 pseudo-random vectors.

There is a difference in fault coverage between the different integration simulations in this case. The fault coverage reaches 100% after 67 vectors using a resolution of 20, but does not reach 100% within 200 vectors for the other two integrals. Integrating over the last half of the cycle with a resolution of 10 is almost as effective as integrating over the whole cycle with a resolution of 20.

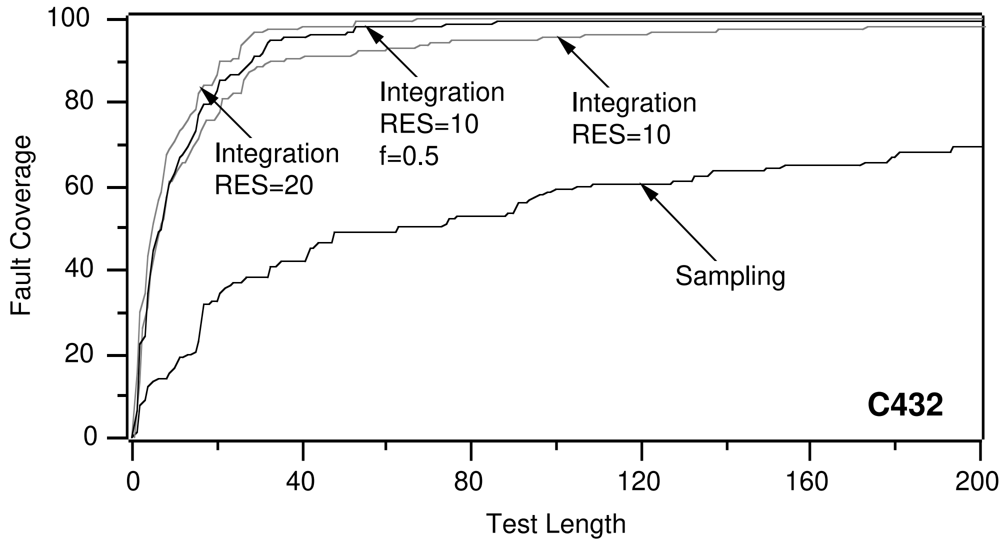


Figure 5.7. Fault Coverage for c432

The fault coverage for c499 is shown in Fig. 5.8. Sampling and integration seem to track each other for this circuit. After 200 patterns, the fault coverage by sampling the output is 5% lower than integration with a resolution of 10, which is in turn 7% lower than the other two integrations.

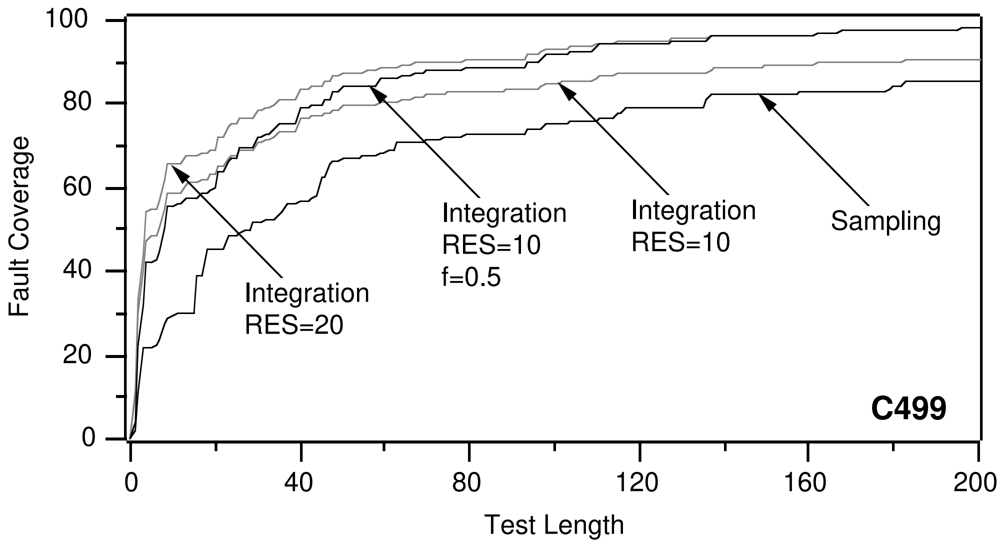


Figure 5.8. Fault Coverage for c499

Detecting delay faults more than once has been suggested as a heuristic try to minimize the possibility of test invalidation by hazards [Waicukauski 87]. The distribution of the number of times faults were detected were computed in [Franco 91b] for the ALU181. The test length was 25 pseudo-random vectors, and faults of sizes ranging from the slack plus 20 to slack plus 40 were injected. The distributions in Fig. 5.9 show that faults are detected significantly more times using integration than sampling. (Note that the areas of the two graphs are not the same, since the fault coverage is different.)

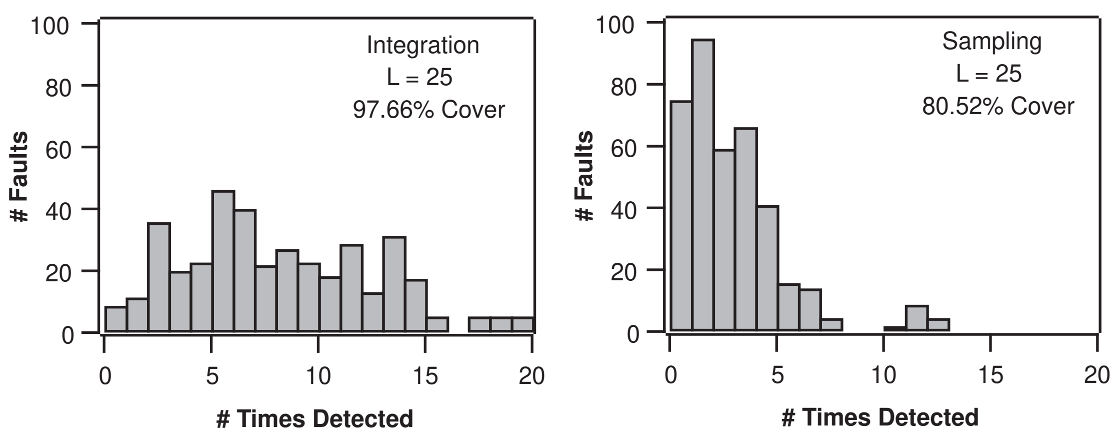


Figure 5.9. Distribution of Number of Patterns that Detect a Fault
© IEEE 1991 [Franco 91b]

5.3 IMPLEMENTATION

Design considerations and integrator implementations that are simple enough to be used on-chip is described in this section.

5.3.1 Design Considerations

The design considerations for implementing integration are similar to those discussed for implementing Stability Checking. In both cases the period when the waveform must be analyzed needs to be defined, the waveform analyzers need to be reset, and the results must be collected.

Since Pre-Sampling Waveform Analysis is more complex than Post-Sampling Waveform Analysis, it is generally not be possible to place integrators wherever needed in a design. It might be better to have fewer, higher resolution, integrators on a chip, and multiplex the signals that need to be checked to the integrators.

An alternative solution multiplexing signals is to use a grid structure as in CrossCheck [Gheewala 89] to access the required nodes. (It should be noted that one of the claimed advantages of CrossCheck is its ability to test for realistic CMOS faults such as stuck-open, shorts, noise margins etc., since analog measurements can be made of the internal nodes.) Only the CUT outputs need to be observed, unlike CrossCheck where internal nodes in the CUTs are also observed. The delay of the testing interconnect is a constant, and is taken into account in the integral value.

It is also possible to multiplex signals to the chip's primary outputs, so that external integrators can be used. The advantage of having the integration performed externally (on the ATE, for example) is that high precision integrators can be used which can be calibrated for absolute measurements.

This solution is not suitable for BIST applications, however, so on-chip integrator designs were investigated. Figure 5.10 shows a conceptual integrator design. An operational amplifier can be used to compute the analog integral, which is then digitized by an analog-to-digital (A/D) converter.

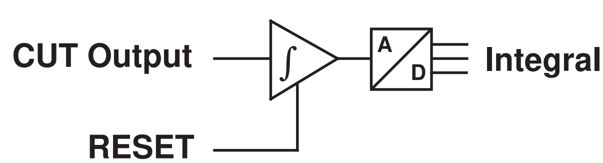


Figure 5.10. Conceptual Integrator Design

It is not practical, however, to put operational amplifiers and A/D converters on chip since these are analog components and most digital fabrication processes would not be suitable, apart from the area overhead. The increasing use of mixed-signal devices could change this in the future. The designs below use only conventional N and P channel transistors.

It is expected to be difficult to calibrate the on-chip integrators for absolute measurements, but relative measurements can still be made. If periodic measurements are made, for example, the integral values for the same test at different times can be compared, so only a relative measurement is required.

For a test set, the relationships between the integrals for different patterns are known, and this can be used to determine if the integrator is in error or there is a delay fault in the circuit. If the integrator is functioning correctly, it should produce the minimum integral if the CUT output is kept at 0, and the maximum integral if the CUT output is kept at 1.

The integration function is performed by charging a node using RC delays, with the final value on the node dependent on the fraction of the integrating period that the CUT output was high. There are different ways to quantize the integral value, depending on the overhead and performance requirements. This is also the case for conventional A/D's, with fast "flash" A/D's or slower, smaller, successive approximation A/D's. Both parallel and serial implementations are discussed below.

5.3.2 Parallel Implementation

Figure 5.11 shows an example of a parallel integrator and analog-to-digital converter. Circuits with different RC constants are fed by the CUT output. The circuit with the longest RC constant that gets charged determines the integral. A priority encoder can be used if a binary representation of the integral is needed.

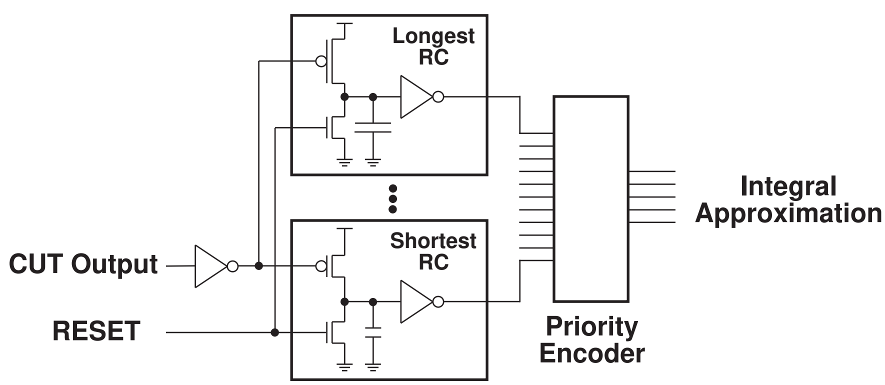


Figure 5.11. Parallel Integrator

5.4.3 Serial Implementation

Serial integrators can be designed by using the CUT output to charge a node, and then measuring the discharge time of the node. Figure 5.12 shows a possible implementation. Node INT is charged during the integrating period whenever the CUT output is high. The final voltage at node INT is proportional to the average value of the output waveform. Note that node INT is reset to approximately 2.5 volts by the RESET signal. This is necessary otherwise small integral values will not charge node INT to beyond the threshold of the output inverter.

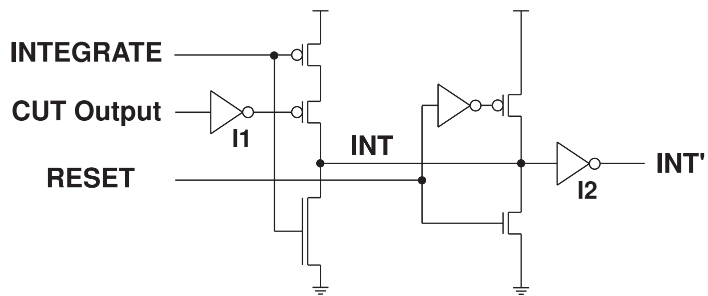


Figure 5.12. Serial Integrator

The design in Fig. 5.12 was manufactured using the Stanford BiCMOS process, and the layout is shown in Fig. 5.13. This circuit was tested, and the operation was partially verified. The reason that only a partial evaluation was possible is that the speed of the ATE used (Tektronix DAS9200) was not high enough to generate pulses that covered only a fraction of the intended cycle time for the designed circuit.

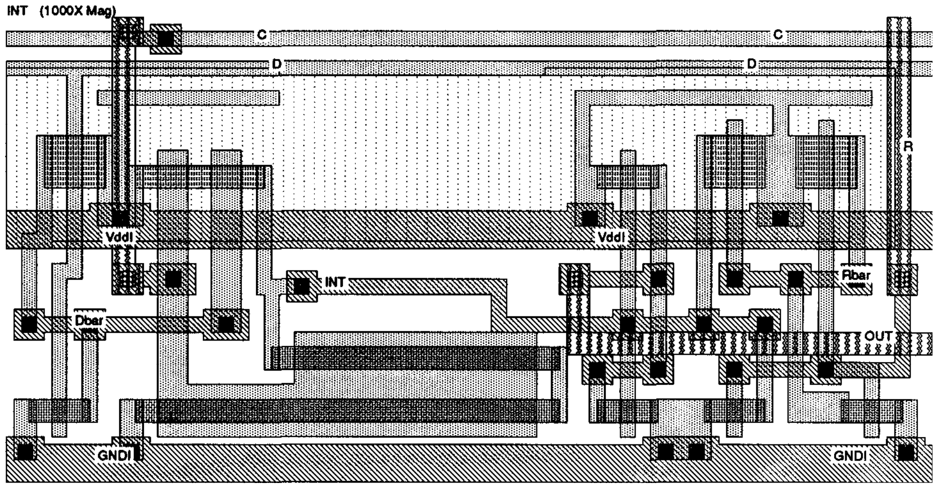


Figure 5.13. Layout for Integrator in Fig. 5.12

The resolution of the serial integrator can be increased by discharging node INV over multiple cycles. The integral value is computed by counting the number of cycles it takes for node INV to discharge. The discharging transistor in the layout in Fig 5.13, for example, is very weak.

Two ways measure the time it takes discharge node INV are shown in Figs. 5.14 and 5.15. The output of the integrator can be used to enable a counter. The counter is then clocked a sufficient number of times (for the maximum integral), but it stops counting once the node INV drops to below the input threshold of the inverter. Another approach is to connect the integrator output to the first flip-flop in a scan chain. As the scan out operation is performed, 1s will be scanned out until the node discharges.

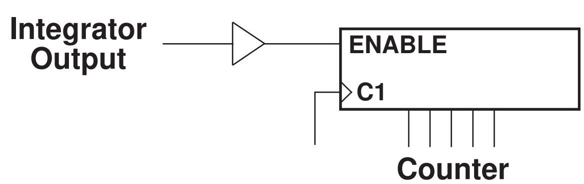


Figure 5.14. Measuring Serial Integral Using Counter

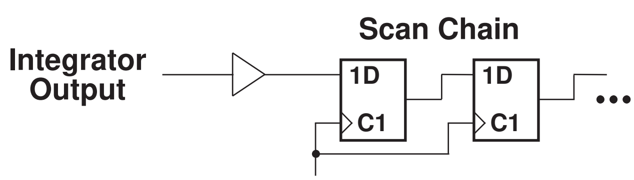


Figure 5.15. Measuring Serial Integral Using Scan Chain

5.4 CONCLUSION

One of the main advantages of Pre-Sampling Waveform Analysis is that it is possible to detect delay flaws, although the hardware overhead is significantly greater than for Post-Sampling Waveform Analysis, and the response compaction is more difficult. The two methods represent different tradeoffs, and Pre-Sampling Waveform Analysis is useful for high reliability systems, where delay flaws can be reliability detractors and must be detected, even if there are no delay faults.

Different Pre-Sampling Waveform Analysis techniques were discussed, and the analog integral of the output waveform was found to be useful, and feasible to implement on-chip. Different forms of integration were presented. The simplest form is equivalent to computing the average value of the waveform. Enhanced integration functions can reduce the probability of aliasing and increase the effective resolution of the integral.

Fault simulation results show that integration is also effective for detecting small delay faults, as the faults do not have to be sensitized through the longest paths to be detected.

Chapter 6

Test Chip Experiment

6.1 OVERVIEW OF EXPERIMENT

The Center for Reliable Computing has participated in a Test Evaluation Chip Experiment over the last two years. A *Test Chip* has been designed and manufactured to compare different testing techniques for combinational or full-scan circuits. The motivation is that it is difficult to determine the effectiveness of the many test techniques that have been proposed without experimental data.

This experiment is a collaboration with several industrial partners. The Test Chip architecture was designed in conjunction with Hughes Aircraft Corporation, where most of the detailed design was done. Over 5,000 Test Chips have been fabricated at LSI Logic, and the wafers are being tested at Digital Testing Services. We are also thankful to many others who have provided test sets or expertise to this project. Our involvement has been in the architectural design, the generation and collection of test sets, writing the ATE test program, and analyzing the experimental results.

This is an on-going experiment, and production testing has just started. The experiment is briefly reviewed in this chapter, and the current experimental results are presented. A more detailed description of the Test Chip and the test sets applied is included in CRC Technical Report 94-5 [Franco 94d]. The final results will also be presented as a CRC Technical Report.

6.1.2 Tests Applied

One of the main distinguishing features of this experiment, is that many different test techniques are being investigated. Generally, previous experiments have only compared few test techniques (recent experiments are summarized in Table 1 in [Franco 94d]). The test sets applied include design verification, exhaustive, pseudo-random, weighted random and deterministically-generated vectors. Tests have been generated for stuck-at faults, transition faults, delay faults and IDDQ testing. The Test Chip is also being tested using the CrossCheck methodology [Gheewala 89], and includes an investigation of the aliasing behavior in both serial and parallel signature analysis. Delay testing by Stability Checking and Very-Low-Voltage Testing [Hao 93] are also investigated.

Apart from investigating Stability Checking, the experiment is also been used to validate the delay modeling issues described in Chapter 2.

6.1.2 Test Chip

The Test Chip is a 25k gate CMOS gate array, manufactured using LSI Logic's LFT150K FasTest array series, and has 96 I/O pins. The Test Chip includes five types of combinational circuits-under-test (CUT), as well as support circuitry for applying patterns to the CUTs, and analyzing the CUT response. There are four copies of each of the five CUTs. The support circuitry takes up approximately half of the Test Chip area.

The basic Test Chip architecture is shown in Fig. 6.1. Both external vectors from the ATE and pseudo-random vectors can be applied to the CUTs through the common data source. External and internally generated clocks are used to investigate different timing modes.

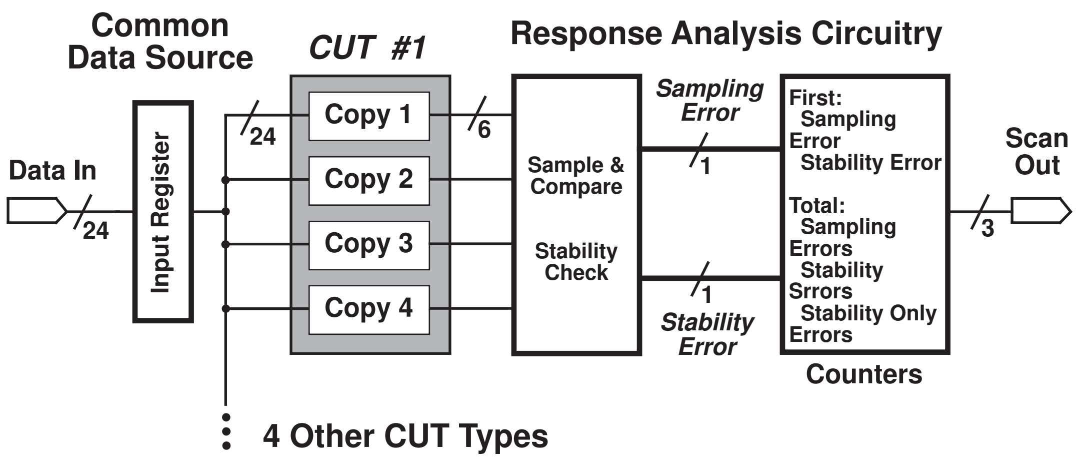


Figure 6.1. Test Chip Architecture

The CUTs are representative of data-path and control logic, as well as different design styles. There are two multipliers and three control logic circuits. The three control logic circuits perform the same function, but have been designed with different constraints. There is a "standard" implementation, which uses the complete cell library, an implementation that uses only elementary gates, and a robustly path-delay-fault testable implementation.

The corresponding outputs of the four copies of each CUT are compared on-chip to determine if any errors have occurred. All CUT outputs have stability checkers. There are 216 Stability Checkers per Test Chip, using the 5 gate NAND design described in Sec. 3.2.4.2. For each test, counters in the response analysis circuitry record the total number of sampling and stability checking errors, as well as the first-fail vector for each case.

6.2 EXPERIMENTAL RESULTS

As described in [Franco 94d], only die that pass the gross parametric tests and support circuitry tests will be used for this experiment. Thus far, the CUTs on 207 die have been tested. The tests described in Sec. 6.2.3 and 6.2.4 are not part of the main experiment, and were done to investigate the delay modeling issued brought up in Chapter 2.

6.2.1 Test Comparisons

The number of CUTs that have been tested is only a small fraction of the total number that will be tested, and clear trends have not yet emerged. It was found, however, that many die had Stability Checking errors for the aggressive clocking rate for one of the CUTs. The clock rate was increased slowly to find the first sampling and Stability Checking error, and it was found that the Stability Checkers started checking the output waveform too early. This verified that the Stability Checker design implemented works correctly, but unfortunately makes direct comparison between the different techniques more difficult. These Stability Checkers are prone to “false alarms” if the CUT outputs change just before the sampling time.

Excluding the Stability errors discussed above, thus far, 10 die have failed at least some of the tests applied. Of the 10 die, there are five with “interesting” behavior. For example: one die failed the boolean and IDDQ tests, but passed the CrossCheck tests; another die failed the CrossCheck tests and boolean tests, but passed the IDDQ test; one die failed the CrossCheck and IDDQ tests, but only failed the Very-Low-Voltage boolean and Stability Checking tests.

6.2.2 Chip Speed Measurement

The Test Chip includes a ring oscillator and five delay lines in different parts of the die. These delay can be used as a process monitor to measure the overall speed of each die. Figure 6.2 shows the result for the six delay measurements on the 207 die. It can be seen that there is not much variation between die, and delays on each die tend to track each other.

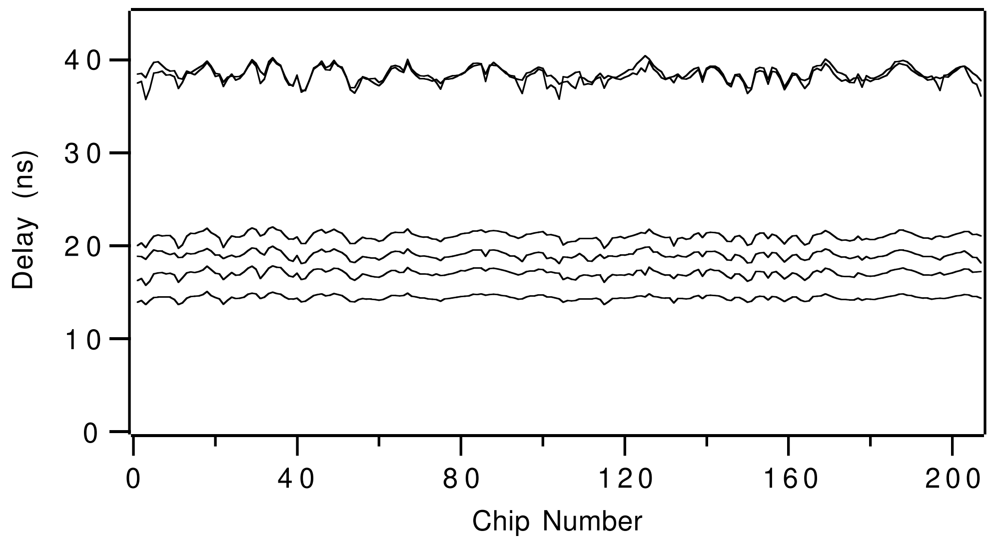


Figure 6.2. Delay Line Measurements

This data is encouraging for both on-line Stability Checking and Pre-Sampling Waveform Analysis by integration. For on-line Stability Checking, the smaller the process variation, the larger the checking period can be made. For integration, small process variations mean that the integrators can be fairly accurate, even though they are not calibrated.

6.2.3 "ROB" CUT Propagation Delay Measurements

Modeling of delay was discussed in Chapter 2, and limitations of the path delay fault model were suggested. Accurate propagation delay measurements have been taken for the robustly path-delay-fault testable CUT and one of the multipliers, to investigate the effect of inaccurate modeling of gate delay in practice. Figure 6.2 shows the response analysis circuitry and clocking mode for this test.

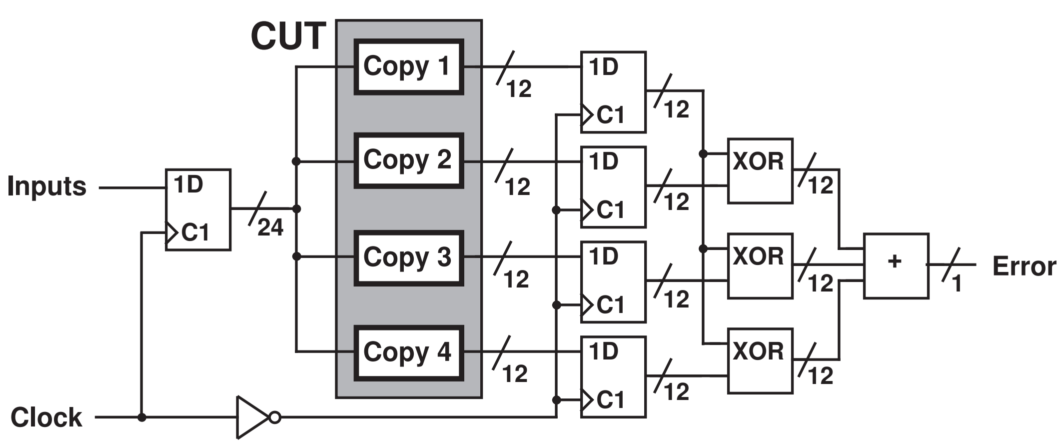


Figure 6.3. Test Setup

Patterns are applied to the CUT on the rising edge of the clock, and the CUT outputs are sampled on the falling edge of the clock. The advantage of this type of clocking is that the only timing-critical pin on the ATE is the clock, so that skew between tester pins does not have to be taken into account.

The duty cycle of the clock was decreased in 25 ps steps until the CUT started failing. The tests applied to the robustly path-delay-fault testable CUT are shown in Table 6.1. These tests are a subset of the tests applied when testing each die, but take much longer than the main test suite, since the tests are repeated many times.

Table 6.1. Tests Applied to Robust CUT

Test	Condition	Length	Output Strobes
Single-Stuck-Fault		262	262
Critical Path	X→0	400	200
Critical Path-R	X→random	400	200
Critical Path-R2	X→ran, Sample all	400	400
Gate Delay	X→0	516	308
Gate Delay-R	X→random	516	516
Robust	X→0	7,068	3,534
Robust-R	X→random	7,068	3,534
Robust-R2	X→ran, Sample all	7,068	7,068
Non-Robust-a		884	884
Non-Robust-b		4,136	4,136
Exhaustive		16.8M	16.8M

The critical path test only tested the 100 longest paths for rising and falling transitions. The paths were chosen using an inaccurate gate delay model. The robust test tested every path in the CUT robustly. For the critical path, gate delay, and robust tests, the test generators left unused inputs at X. The tests were repeated with 0s assigned to the X's, as well as 0 of 1 randomly assigned to the X's. All vectors were sampled, as well as every second vector as is normally done for delay testing.

Figure 6.4 shows the results for one die, for the robustly path-delay-fault testable circuit. The Exhaustive test fails at the slowest cycle time, which means that the other test sets do not provoke the longest delay through the circuit. (The actual longest delay could be worst than determined by the exhaustive test.) In particular, the longest delay through the circuit is not exercised using the robust test.

(Approximately 100 measurements were remade to check the repeatability of the ATE, and all measurements were within 25 ps, except one that differed by 50 ps, so there were no ATE consistency problems.)

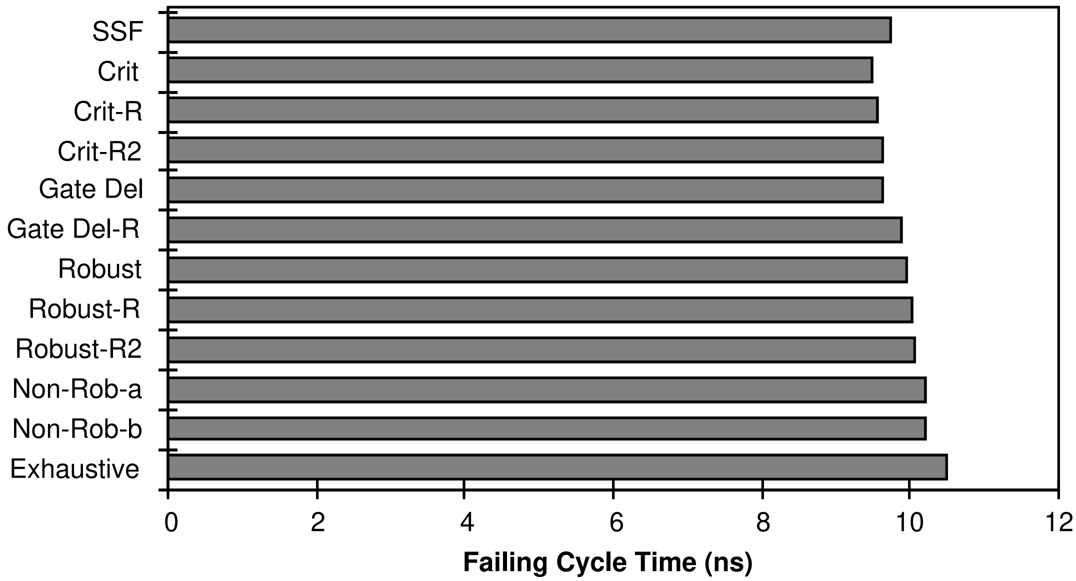


Figure 6.4. Robust Circuit Results for 1 Die

Figure 6.5 shows the average results for the 10 die tested. For each die, the test sets were measured relative to the exhaustive test set, and the percentage difference relative to the exhaustive test was plotted. For example, the single-stuck-fault test needed to be run between 6% and 7.2% faster than the exhaustive test to fail the 10 die tested. The critical path test was the worst. This shows the danger in testing only a small fraction of the paths in the circuit, and using an inaccurate model to choose the paths.

For both the gate delay test and the robust test, there was a noticeable improvement by replacing X's randomly with 0 or 1 rather than only 0. The longest delays through none of the 10 die were exercised with any of the tests.

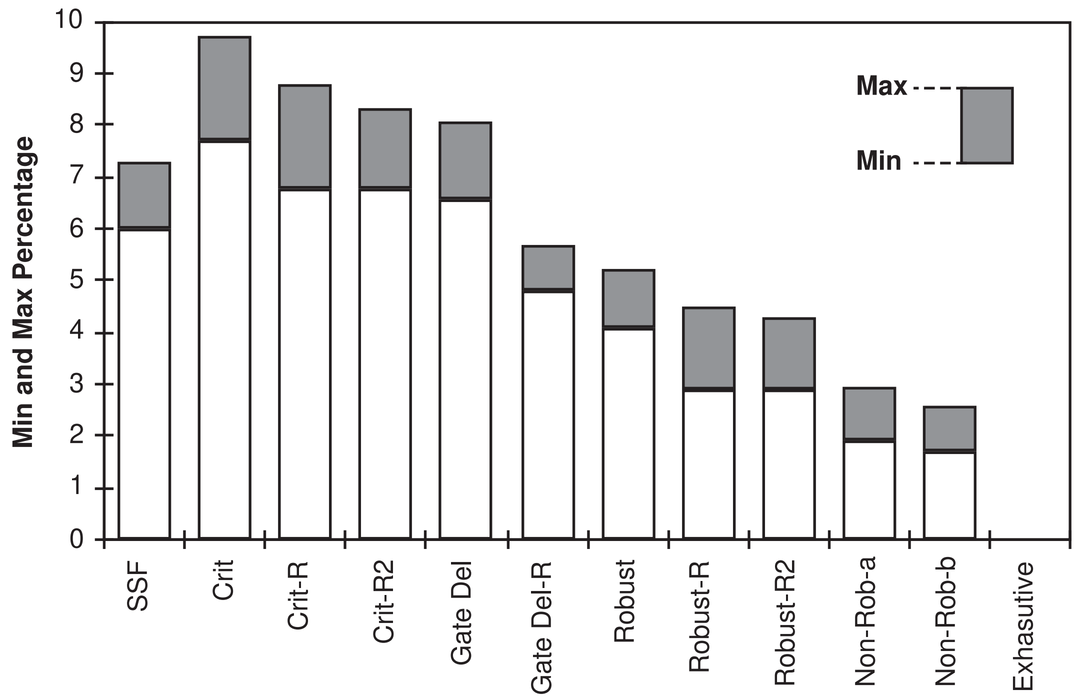


Figure 6.5. Robust Circuit Results for 10 Die

6.2.4 "MULT6SQ" CUT Propagation Delay Measurements

A similar test was done for one of the multipliers. This CUT consists of two cascaded multipliers, and only has 12 inputs, making the super-exhaustive test possible. There are 7×10^{15} structural paths in this circuit, and test pattern generation for all paths was not possible. The tests applied are shown in Table 6.2.

Table 6.2. Tests Applied to 6x6 Multiplier

Test	Condition	Length	Output Strobes
Single-Stuck-Fault		22	22
Transition Fault		304	152
Critical Path	X→0	1,692	846
Critical Path-R	X→ran	1,692	846
Critical Path-R2	X→ran, Sample all	1,692	1,692
Gate Delay	X→0	976	488
Gate Delay-R	X→ran	976	516
Exh-pr	From LFSR	4,096	4,096
Exh-gray	Single Trans.	4,096	4,096
Exh-max. Trans.	Maximal Trans.	4,096	4,096
Super Exhaustive		16.8M	16.8M

Figure 6.6 shows the experimental results for the multiplier circuit. The propagation delays on four die were measured, and the maximum and minimum values relative to the "super-exhaustive" test were plotted.

As in Fig. 6.5, the longest delay is not exercised by any of the shorter tests. For this circuit, the single-stuck, transition, and critical path tests were very similar. Three exhaustive tests were also applied. The first was generated with a primitive polynomial, the second is a gray code with single bit transitions between vectors, and the third test maximizes the number of transitions between vectors (either n or n-1, for an n-bit vector). The gray code performs very poorly, and the circuit must be clocked at least 22% faster than the worst-case delay to detect the delay fault. The waveform simulator described in [Franco 94c] was used to compute the node activity for the three exhaustive tests, and as expected, the activity for the gray code was significantly lower than for the other two tests (12% compared to 24% for the pseudo-random, and 33% for the maximal-transition test).

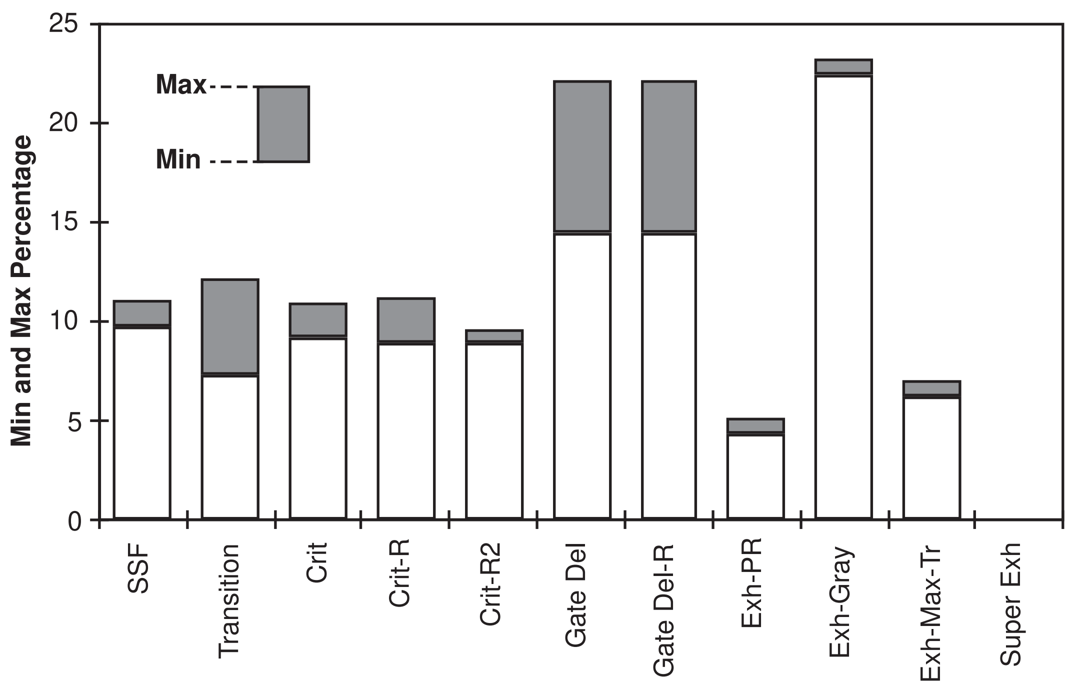


Figure 6.6. Multiplier Circuit Results

The propagation delay measurements in this Chapter show that the longest delays in circuit are not exercised by using test sets generated using a simple delay model. Even if all paths are robustly testable, and there are not too many paths to test, the complete robust test does not guarantee that the circuit functions at the designed speed.

Chapter 7
Concluding Remarks

7.1 CONTRIBUTIONS

A new approach for detecting timing failures in digital circuits has been described in this dissertation. Output Waveform Analysis is different from other techniques in that information is extracted from the output waveforms between samples, instead of just using the sampled values. Although the circuit-under-test is digital, one is not restricted to digital techniques for observing the circuit output response. The intuitive justification for Output Waveform Analysis is that timing failures will change the shape of the output waveform, and therefore the whole waveform contains information about the delays in the circuit.

Output Waveform Analysis techniques can be classified into Pre-Sampling Waveform Analysis and Post-Sampling Waveform Analysis, depending on whether the output waveform is observed before or after the sample.

Post-Sampling Waveform Analysis is very simple to implement, and is suitable for delay testing at various levels, from wafer sort, to Built-In Self-Test in the field, as well as for on-line or concurrent checking.

Pre-sampling Waveform Analysis is more complex, and is suited to applications where delay flaws need to be detected due to reliability considerations. Propagation delays can be measured with Pre-Sampling Waveform Analysis, which can be used to predict reliability failures in the field or shorten environmental stress screening.

Output Waveform Analysis is independent of the test patterns applied, and can be used with any vectors.

Incidentally, there is an interesting connection between the Pre-Sampling and Post-Sampling Waveform Analysis functions used here. Mathematically, a stable waveform is described as the derivative equal to zero. So while integration was chosen as the Pre-Sampling Waveform Analysis function, differentiation was used as the Post-Sampling Waveform Analysis function.

It was shown that various Output Waveform Analysis techniques are practical by presenting test architectures, circuit implementations, experimental results. Three waveform analyzers were implemented and tested using the Stanford BiCMOS process, and one Stability Checker design has been included in a Test Evaluation Chip Experiment.

An algorithm was implemented for padding short paths for on-line Stability Checking, and it was found to be efficient as long as modified cells were available for padding. A symbolic waveform simulator was also implemented to investigate different Output Waveform Analysis techniques.

Work has also been done on the effect of delay modeling of on delay fault testing, and it was shown that three-pattern delay tests are needed to exercise the longest delays in CMOS circuits. Path delay faults, as commonly defined, are only a subset of all delay faults.

7.2 FUTURE

Looking at the future, it seems very likely that timing failures will become even more important as systems become more complex. The emergence of standards (bus designs, PC architectures, image and video compression, etc.), for example, means that systems from different vendors will perform essentially the same function. This will make aggressive clocking to improve cost/performance very attractive.

The waveform analyzers presented in this dissertation have been designed with standard digital components. With the recent increase in mixed-signal designs, it is becoming more common to mix digital and analog functions on a chip. This makes it possible to design analog waveform analyzers, and implement techniques that are difficult to do in purely digital systems.

A natural result of timing optimizations is the equalization of long and short paths. In the limit, all paths in a circuit will have similar delays. In this case, Post-Sampling Waveform Analysis might be sufficient, as any timing failure will cause a delay fault, and change the output waveform after the sampling time.

There is scope for future work in several directions. Other Pre-Sampling Waveform Analysis functions can be investigated, and test pattern generation algorithms specifically for Output Waveform Analysis could be derived.

Limitations of the path delay fault model, and inaccurate delay modeling in general, have been discussed. Both spice simulations of extracted circuit layouts, and propagation delay measurements done on the Test Evaluation Chip Experiment show that the simple delay models are not sufficient to exercise the longest delays in CMOS circuits. As feature sizes shrink and short-channel devices are used, it will become more challenging to predict the timing behavior of circuits.

The waveform simulator described in [Franco 94c] was implemented to investigate Output Waveform Analysis, because existing timing simulators were not convenient. The simulator is written in LISP, and no optimizations were done, so it is perhaps surprising that its speed was similar to Verilog-XL for the benchmark circuits investigated. The essential feature was the waveform representation chosen, and it might be worth investigating this further.

The Test Evaluation Chip Experiment is currently underway, and although less than 5% of the die have been tested, some interesting results have been observed. Once the experiment is complete, the results might suggest further areas of research.

REFERENCES

[Acken 88] Acken, J.M., "Deriving Accurate Fault Models," *Technical Report CSL-TR-88-365*, Computer Systems Laboratory, Stanford University, 1988.

[Aho 85] Aho, A.V., R. Sethi, and J.D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, MA, 1985.

[Barzilai 83] Barzilai, Z., and B.K. Rosen, "Comparison of AC Self-Testing Procedures," *Proc. 1983 Int. Test Conf.*, Philadelphia, PA, pp. 89-94, Oct. 18-20, 1983.

[Barzilai 86] Barzilai, Z, et. al., "Efficient Fault Simulation of CMOS Circuits with Accurate Models," *Proc. 1986 Int Test Conf.*, Washington, DC, pp. 520-529, Sep. 8-11, 1986.

[Benowitz 75] Benowitz, N., D.F. Calhoun, G.E. Alderson, J.E. Bauer, and C.T. Joeckel, "An Advanced Fault Isolation System for Digital Logic," *IEEE Trans. Computers*, Vol. C-24, No. 5, pp. 489-497, May 1975.

[Bose 93] Bose, S., P. Agrawal, and V.D. Agrawal, "Generation of Compact Delay Tests by Multiple Path Activation," *Proc. 1993 Int. Test Conf.*, Baltimore, MD, pp. 714-723, Oct. 17-21, 1993.

[Brayton 87] Brayton, R.K., R. Rudell, A.L. Sangiovanni-Vincentelli, and A.R. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Trans. CAD*, Vol. 6, pp. 1062-1081, Nov. 1987.

[Brayton 90] Brayton, R.K., G.D. Hachtel, and A.L. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," *Proc. IEEE*, Vol. 78, No. 2, pp. 264-300, Feb. 1990.

[Breuer 74a] Breuer, M. A., and R. L. Harrison, "Procedures for Eliminating Static and Dynamic Hazards in Test Generation," *IEEE Trans. Computers*, Vol. C-23, pp. 1069-1078, Oct. 1974.

[Breuer 74b] Breuer, M. A., "The Effects of Races, Delays, and Delay Faults on Test Generation," *IEEE Trans. Computers*, Vol. C-23, pp. 1078-1092, Oct. 1974.

[Breuer 76] Breuer, M. A., and A. D. Friedman, *Diagnosis & Reliable Design of Digital Systems*, Computer Science Press, Woodland Hills, CA, 1976.

[Brglez 85] Brglez, F., and H. Fujiwara, "Neutral Netlist of Ten Combinational Benchmark Circuits and a Target Translator in FORTRAN," *Proc. Int. Symp. on Circuits and Systems* (Special session on ATPG and fault simulation), June 1985.

[Bula 90] Bula, O., et. al., "Gross delay defect evaluation for a CMOS logic design system product," *IBM J. Res. Develop.*, Vol. 34, No. 2/3, pp. 325-338, Mar./May 1990.

[Cheng 91] Cheng, K.-T., S. Devadas, and K. Keutzer, "A Partial Enhanced-Scan Approach to Robust Delay-Fault Test Generation for Sequential Circuits," *Proc. Int. Test Conf.*, Nashville, TN, pp. 403-410, Oct. 29-Nov. 1, 1991.

[Cheng 92] Cheng, S. W., H.-C. Cheng, and D. H. C. Du, "The role of Long and Short Paths in Circuit Performance Optimization," *Proc. 29th Design Automation Conf.*, Anaheim, CA, pp. 543-548, 1992.

[Cheng 93] Cheng, K.-T., and H.-C. Chen, "Delay Testing For Non-Robust Untestable Circuits," *Proc. 1993 Int. Test Conf.*, Baltimore, MD, pp. 954-961, Oct. 17-21, 1993.

[Connet 72] Connet, J.R., E.J. Pasternak, and B.D. Wagner, “Software defenses in real time control systems,” *FTCS-2*, Newton, MA, pp. 94-99, June 19-21, 1972.

[Cortner 87] Cortner, J.M., *Digital Test Engineering*, Wiley, New York, NY, 1987.

[Craig 85] Craig, G.L., and C.R. Kime, “Pseudo-Exhaustive Adjacency Testing: A BIST Approach for Stuck-Open Faults,” *Proc. 1995 Int. Test Conf.*, Philadelphia, PA, pp. 126-137, Nov. 19-21, 1985.

[De 92] De, K., C. Wu, and P. Banerjee, “Reliability Driven Logic Synthesis of Multilevel Circuits,” *Proc. Int. Symposium on Circuits and Systems (ISCAS)*, pp. 1105-1108, 1992.

[Devadas 90a] Devadas, S., and K. Keutzer, “Synthesis and Optimization Procedures for Robustly Delay-Fault Testable Combinational Logic Circuits,” *Proc. 27th Design Automation Conf.*, pp. 221-227, June 1990.

[Devadas 90b] Devadas, S. and K. Keutzer, “Design of Integrated Circuits Fully Testable for Delay-Faults and Multifaults,” *Proc. 1990 Int. Test Conf.*, Washington, DC, pp. 284-293, Sep. 10-12, 1990.

[Devadas 92] Devadas, S. and K. Keutzer, “Validatable Nonrobust Delay-Fault Testable Circuits Via Logic Synthesis,” *IEEE Trans. CAD*, Vol. 11, No. 12, pp. 1559-1573, Dec. 1992.

[Dorey 90] Dorey, A.P., B.K. Jones, A.M.D. Richardson, and Y.Z. Xu, *Rapid Reliability Assessment of VLSICs*, Plenum Press, New York, 1990.

[Eichelberger 65] Eichelberger, E. B., “Hazard detection in combinational and sequential switching circuits,” *IBM J. Res. Develop.*, Vol. 9, pp. 90-99, Mar. 1965.

[Eichelberger 77] Eichelberger, E.B., and T. W. Williams, “A Logic Design Structure for LSI Testability,” *Proc. 14th Design Automation Conf.*, pp. 462-468, June 1977.

[Eldred 59] Eldred, R. D., "Test Routines Based on Symbolic Logical Statements," *J. Assoc. Comput. Mach.*, Vol. 28, No. 6, pp. 2-8, 1959.

[Franco 91a] Franco, P., N.R. Saxena, and E.J. McCluskey, “Relating Aliasing in Signature Analysis to Test Length and Register Design,” *Proc. ISCAS’91*, Singapore, pp. 1889-1892, June 11-14, 1991.

[Franco 91b] Franco, P., and E.J. McCluskey, “Delay Testing of Digital Circuits by Output Waveform Analysis,” *Proc. 1991 Int. Test Conf.*, Nashville, TN, pp. 798-807, Oct. 26-30, 1991.

[Franco 93] Franco, P., and E.J. McCluskey, “On-Line Delay Testing of Digital Circuits,” *Center for Reliable Computing Technical Report, No. 93-7* Stanford University, Nov. 1993.

[Franco 94a] Franco, P., and E.J. McCluskey, “On-Line Delay Testing of Digital Circuits,” *Proc. 12th IEEE VLSI Test Sym.,* Cherry Hill, NJ, pp. 167-173, Apr. 25-28, 1994.

[Franco 94b] Franco, P., and E.J. McCluskey, “Three-Pattern Tests for Delay Faults,” *Proc. 12th IEEE VLSI Test Sym.,* Cherry Hill, NJ, pp. 452-456, Apr. 25-28, 1994.

[Franco 94c] Franco, P., and E.J. McCluskey, “WSIM: A Symbolic Waveform Simulator,” *Center for Reliable Computing Technical Report,* No. 94-4*,* June 1994.

[Franco 94d] Franco, P., R.L. Stokes, W.D. Farwell, and E.J. McCluskey, “An Experimental Chip to Evaluate Test Techniques, Part 1: Description of Experiment,” *Center for Reliable Computing Technical Report,* No. 94-5*,* June 1994.

[Gheewala 89] Gheewala, T., “CrossCheck: A Cell Based VLSI Testability Solution,” *Proc. 26th Design Automation Conf.*, pp. 706-709, June 1989.

[Gray 91] Gray, C.T., et. al., “A High Speed CMOS FIFO Using Wave Pipelining,” *Technical Report* NCSU-VLSI-91-01, North Carolina State University, Jan. 1991.

[Hao 91] Hao, H., and E.J. McCluskey, “‘Resistive Shorts’ within CMOS Gates,” *Proc. 1991 Int. Test Conf.*, Nashville, TN, pp. 292-301, Oct. 26-30, 1991.

[Hao 93] Hao, H., and E.J. McCluskey, “Very-Low-Voltage Testing for Weak CMOS Logic ISs,” *Proc. 1993 Int. Test Conf.*, Baltimore, MD, pp. 275-284, Oct. 17-21, 1993.

[Hawkins 85] Hawkins, C.F., and J.M. Soden, “Electrical Characteristics and Testing Considerations for Gate Oxide Shorts in CMOS ICs,” *Proc. 1985 Int. Test Conf.*, Philadelphia, PA, pp. 544-555, Nov. 19-21, 1985.

[Hawkins 86] Hawkins, C.F., and J.M. Soden, “Reliability and Electrical Properties of Gate Oxide Shorts in CMOS ICs,” *Proc. 1986 Int. Test Conf.*, Washington, DC, pp. 443-451, Sep. 8-11, 1986.

[Hawkins 89] Hawkins, C.F., J.M. Soden, R.R Fritzemeier, and L.K. Horning, “Quiescent Power Supply Current measurement for CMOS IC Defect Detection,” *IEEE Trans. on Industrial Electronics*, Vol. 36, No. 2, pp. 211-218, May 1989.

[Heinbuch 88] Heinbuch, D.V., Editor, *CMOS3 Cell Library*, Addison-Wesley, Reading, MA, 1988.

[Hitchcock 82] Hitchcock, R.B., G.L. Smith, and D.D. Cheng, “Timing Analysis of Computer Hardware,” *IBM J. Res. Develop.,* Vol. 26, No. 1, pp. 100-105, Jan. 1982.

[Hsieh 77] E.P. Hsieh, R.A. Rasmussen, L.J. Vidunas and W.T. Davis, “Delay Test Generation,” *Proc. 14th Design Automation Conf.*, pp. 486-491, June 1977.

[Hu 85] Hu, C., et.al., “Hot-Electron-Induced MOSFET Degradation -- Model, Monitor, and Improvement,” *IEEE Trans. on Elec. Dev.*, Vol. ED-32, No. 2, pp. 375-385, Feb. 1985.

[Hu 89] Hu, C., “Reliability Issues of MOS and Bipolar ICs,” *Proc. IEEE Int. Conf. Comput. Design*, pp. 438-442, 1989.

[IEEE 88] IEEE Standard 1076-1987, “IEEE Standard VHDL Language Reference Manual,” *IEEE Standards Board*, 345 East 47th Street, New York, NY 10017, 1988.

[Ishiura 89] Ishiura, N., M. Takahashi, and S. Yajima, “Time-Symbolic Simulation for Accurate Timing Verification of Asynchronous Behavior of Logic Circuits,” *Proc. 26th Design Automation Conf.*, pp. 497-502, June 1989.

[Ishiura 90] Ishiura, N., Y. Deguchi, and S. Yajima, “Coded Time-Symbolic Simulation Using Shared Binary Decision Diagram,” *Proc. 27th Design Automation Conf.*, pp. 130-135, June 1990.

[Iyengar 88a] Iyengar, V.S., B.K. Rosen, and I. Spillinger, “Delay Test Generation 1 -- Concepts and Coverage Metrics,” *Proc. 1988 Int. Test Conf.*, Washington, DC, pp. 857-866, Sep. 12-14, 1988.

[Iyengar 88b] Iyengar, V.S., B.K. Rosen, and I. Spillinger, “Delay Test Generation 2 -- Algebra and Algorithms,” *Proc. 1988 Int. Test Conf.*, Washington, DC, pp. 867-876, Sep. 12-14, 1988.

[Iyengar 92] Iyengar, V.S., and G. Vijayan, "Optimized Test Application Timing for AC Test," *IEEE Trans. CAD*, Vol. 11, No. 11, pp. 1439-1449, Nov. 1992.

[Jha 91] Jha, N.K., and S.-J. Wang, "Design and Synthesis of Self-Checking VLSI Circuits and Systems," *Proc. Int. Conf. Computer Design (ICCD)*, pp. 578-581, 1991.

[Johnson 89] Johnson, B.W., *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley, MA, 1989.

[Klass 90] Klass, F., and J.M. Mulder, "CMOS Implementation of Wave Pipelining," *Technical Report: 1-68340-44(1990)02*, Delft University of Technology, Dec. 1990.

[Klass 92] Klass, F., and J.M. Mulder, "Use of CMOS Technology in Wave Pipelining," *Proc. 5th IEEE Conf. on VLSI Design*, Bangalore, India, pp. 303-306, Jan. 1992.

[Konemann 79] Konemann, B., J. Mucha, and G. Zwiehoff, "Built-in Logic Block Observation Technique," *Dig. 1979 IEEE Test Conf.*, pp. 37-41, Oct. 1979.

[Kundu 88a] Kundu, S., and S.M. Reddy, "On The Design of Robust Testable CMOS Combinational Logic Circuits," *Proc. 18th Int. Fault Tolerant Comp. Symp.*, Tokyo, Japan, pp. 220-225, June 27-30, 1988.

[Kundu 88b] Kundu, S., and S.M. Reddy, and N.K. Jha, "On The Design of Robust Multiple Fault Testable CMOS Combinational Logic Circuits," *Proc. IEEE Int. Conf. on CAD.*, pp. 240-243, Nov. 1988.

[Lesser 80] Lesser, J.D., and J.J Shedletsky, "An Experimental Delay Test Generator for LSI Logic," *IEEE Trans. Computers*, Vol. C-29, No. 3, pp. 235-248, Mar. 1980.

[Levi 81] Levi, M.W., "CMOS Is Most Testable," *Proc. 1981 Int. Test Conf.*, pp. 217-220, 1981.

[Lin 86] Lin, C.J., and S.M Reddy, "On Delay Fault Testing in Logic Circuits," *Proc. ICCAD*, pp. 148-151, Nov. 1986.

[Lin 87] Lin, C.J., and S.M Reddy, "On Delay Fault Testing in Logic Circuits," *IEEE Trans. CAD*, Vol. CAD-6, No. 5, pp. 694-703, Sep. 1987.

[LSI 90] LSI Logic, *0.7-Micron Array-Based Products Databook*, Apr. 1990.

[LSI 91] LSI Logic, *1.0-Micron Cell-Based Products Databook*, LCB007 Cell-Based ASICs, Feb. 1991.

[Mahmood 88] Mahmood, A., and E.J. McCluskey, "Concurrent Error Detection Using Watchdog Processors -- A Survey," *IEEE Trans. Comp.*, Vol. 27, No. 2, pp. 160-174, Feb. 1988.

[Mao 90a] Mao, W.-W., and M.D. Ciletti, "A Variable Observation Time Method for Testing Delay Faults," *Proc. 27th Design Automation Conf.*, pp. 728-731, June 1990.

[Mao 90b] Mao, W.-W., and M.D. Ciletti, "Arrangement of Latches in Scan-Path Design to Improve Delay Fault Coverage," *Proc. 1990 Int. Test Conf.*, Washington, DC, pp. 387-393, Sep. 10-12, 1990.

[Maxwell 93] Maxwell, P.C. and R.C. Aitken, "Biased Voting: A Method for Simulating CMOS Bridging Faults in the Presence of Variable Gate Logic Thresholds," *Proc. 1993 Int. Test Conf.*, Baltimore, MD, pp. 63-72, Oct. 17-21, 1993.

[McCluskey 62] McCluskey, E.J., "Transients in Combinational Logic Circuits," in R.H. Wilcox and W.C. Mann (eds.), *Redundancy Techniques for Computing Systems*, Spartan Book, Washington, D.C. pp. 9-46, 1962.

[McCluskey 86] McCluskey, E.J., *Logic Design Principles with Emphasis on Testable Semicustom Circuits*, Prentice-Hall, 1986.

[Park 87] Park, E.S., and M.R. Mercer, "Robust and Nonrobust Tests for Path Delay Faults in a Combinational Circuit," *Proc. 1987 Int. Test Conf.*, Washington, DC, pp. 1027-1034, Sep. 1-3, 1987.

[Park 88] Park, E.S., M.R. Mercer, and T. W. Williams, "Statistical Delay Fault Coverage and Defect Level for Delay Faults," *Proc. 1988 Int. Test. Conf.*, Washington, DC, pp. 492-498, Sep. 12-14, 1988.

[Park 91] Park, E.S., B. Underwood, T. W. Williams, and M.R. Mercer, "Delay Testing Quality in Timing-Optimized Designs," *Proc. 1991 Int. Test Conf.*, Nashville, TN, pp. 897-905, Oct. 26-30, 1991.

[Parker 87] Parker, K.P., *Integrating Design and Test: using CAE tools for ATE programming*, IEEE Computer Society Press, 1987.

[Patil 92] Patil, S., and J. Savir, "Skewed-Load Transition Test: Part II, Coverage," *Proc. 1992 Int. Test Conf.*, Baltimore, MD, pp. 714-722, Sep. 20-24, 1992.

[Pomeranz 91] Pomeranz, I., and S. M. Reddy, "Achieving Complete Delay Fault Testability by Adding Extra Inputs," *Proc. Int. Test Conf.*, Nashville, TN, pp. 273-282, Oct. 29-Nov. 1, 1991.

[Pomeranz 92] Pomeranz, I., and S. M. Reddy, "At-Speed Delay Testing of Synchronous Sequential Circuits," *Proc. 29th ACM/IEEE Des. Autom. Conf.*, pp. 177-181, 1992.

[Pomeranz 94] Pomeranz, I., and S.M. Reddy, "An Efficient Nonenumerative Method to Estimate the Path Delay Fault Coverage in Combinational Circuits," *IEEE Trans. CAD*, Vol. 13, No. 2, pp. 240-250, Feb. 1994.

[Pramanick 88] Pramanick, A.K., and S.M. Reddy, "On the Detection of Delay Faults," *Proc. 1988 Int. Test Conf.*, Washington, DC, pp. 845-856, Sep. 12-14, 1988.

[Pramanick 89] Pramanick, A.K., and S.M. Reddy, "On the Computation of the Ranges of Detected Delay Fault Sizes," *Proc. Int. Conf. on Comp. Aided Design*, pp. 126-129, Nov. 1989.

[Pramanick 90a] Pramanick, A.K., S.M. Reddy, and S. Sengupta, "Synthesis of Combinational Logic Circuits for Path Delay Fault Testability," *Proc. Int. Symp. on Circuits and Systems*, pp. 3105-3108, May 1990.

[Pramanick 90b] Pramanick, A.K., and S.M. Reddy, "On the Design of Path Delay Fault Testable Combinational Circuits," *Proc. 1990 Int. Fault-Tolerant Computing Symp.*, pp. 374-381, June 1990.

[Pramanick 91] Pramanick, A.K., and S.M. Reddy, "On Multiple Path Propagating Tests for Path Delay Faults," *Proc. 1991 Int. Test Conf.*, Nashville, TN, pp. 393-402, Oct. 29-Nov. 1, 1991.

[Pramanick 93] Pramanick, A.K., and S. Kundu, "Design of Scan-Based Path Delay Testable Sequential Circuits," *Proc. 1993 Int. Test Conf.*, Baltimore, MD, pp. 962-971, Oct. 17-21, 1993.

[Reddy 84] Reddy, S.M., M.K. Reddy and V.D. Agrawal, "Robust Tests for Stuck-Open Faults in CMOS Combinational Logic Circuits," *14th FTCS*, pp. 44-49, 1984.

[Reddy 87] Reddy, S.M., C.J. Lin, and S. Patil, "An Automatic Test Pattern Generator for the Detection of Path Delay Faults," *Proc. Int. Conf. on Comp. Aided Design*, pp. 284-287, Nov. 1987.

[Roy 89] Roy, K., J.A. Abraham, K. De, and S. Lusky, "Synthesis of Delay Fault Testable Combinational Logic," *Proc. Int. Conf. on Comp. Aided Design*, pp. 418-421, Nov. 1989.

[Savir 88] Savir, J, and W.H. McAnney, "Random Pattern Testability of Delay Faults," *IEEE Trans. Computers*, Vol. 37, No. 3, pp. 291-300, Mar. 1988.

[Savir 92] Savir, J, "Skewed-Load Transition Test: Part I, Calculus," *Proc. 1992 Int. Test Conf.*, Baltimore, MD, pp. 705-713, Sep. 20-24, 1992.

[Saxena 90] Saxena, N.R., E.J. McCluskey, and P. Franco, "Bounds on Signature Analysis Aliasing for Random Testing," *Center for Reliable Computing Technical Report*, No. 90-11, Stanford University, Dec. 1990.

[Saxena 91a] Saxena, N.R., E.J. McCluskey, and P. Franco, "Refined Bounds on Signature Analysis Aliasing for Random Testing," *Center for Reliable Computing Technical Report*, No. 91-2, Stanford University, Feb. 1991.

[Saxena 91b] Saxena, N.R., P. Franco, and E.J. McCluskey, "Bounds on Signature Analysis Aliasing for Random Testing," *Dig. 21st Annu. Int. Symp. Fault-Tolerant Comput.* (FTCS-21), Montreal, Canada, pp. 104-111, June 25-27, 1991.

[Saxena 91c] Saxena, N.R., P. Franco, and E.J. McCluskey, "Refined Bounds on Signature Analysis Aliasing for Random Testing," *Proc. 1991 Int. Test Conf.*, Nashville, TN, pp. 798-807, Oct. 26-30, 1991.

[Saxena 92] Saxena, N.R., P. Franco, and E.J. McCluskey, "Simple Bounds on Signature Analysis Aliasing for Random Testing," *Special Issue on Fault-Tolerant Computing, IEEE Trans. Comput.*, pp. 638-645, May 1992.

[Saxena 93] Saxena, L., and D.K. Pradhan, "A Method to Derive Compact Test Sets for Path Delay Faults in Combinational Circuits," *Proc. 1993 Int. Test Conf.*, Baltimore, MD, pp. 724-733, Oct. 17-21, 1993.

[Shedletsky 78] Shedletsky, J.J., "Delay Testing LSI Logic," *Dig. 8th Annu. Int. Symp. Fault-Tolerant Comput. (FTCS-8)*, pp. 159-164, June 1978.

[Shenoy 93] Shenoy, N.V., R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "Minimum Padding to Satisfy Short Path Constraints," *Proc. Int. Conf. Computer Aided Design*, Santa Clara, CA, pp. 156-161, Nov. 7-11, 1993.

[Shoji 92] Shoji, M., *Theory of CMOS Digital Circuits and Circuit Failures*, Princeton, NJ, Princeton University Press, 1992.

[Smith 85] Smith, G.L., "Model for Delay Faults Based Upon Paths," *Proc. 1985 Int. Test Conf.*, Philadelphia, PA, pp. 342-349, Nov. 19-21 1985.

[Storey 77] Storey, T.M., and J.W. Barry, "Delay Test Simulation," *Proc. 14th Design Automation Conf.*, pp. 492-494, June 1977.

[Synopsys 91] Synopsys, *Design Compiler Reference Manual*, Version 2.0, May 1991.

[van Brakel 92] van Brakel, G., Y. Xing, and H.G. Kerkhoff, "Scan Cell Design for Enhanced Delay Fault Testability," *Proc. Fifth Annual IEEE Int. ASIC Conf. and Exhibit*, New York, pp. 372-375, 1992.

[Vierhaus 93] Vierhaus, H.T., W. Meyer, and U. Glaser, "CMOS Bridges and Resistive Transistor Faults: IDDQ versus Delay Effects," *Proc. 1993 Int. Test Conf.*, Baltimore, MD, pp. 83-91, Oct. 17-21, 1993.

[Wagner 88] Wagner, K.D., "Clock System Design," *IEEE Des. & Test of Comp.*, pp. 9-27, Oct. 1988.

[Waicukauski 87] Waicukauski, J.A., E. Lindbloom, B.K. Rosen, and V.S. Iyengar, "Transition Fault Simulation," *IEEE Design & Test*, pp. 32-38, Apr. 1987.

[Weste 85] Weste, N.H.E, and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, 1985.

[Weste 93] Weste, N.H.E, and K. Eshraghian, *Principles of CMOS VLSI Design A Systems Perspective*, Second Edition, Addison-Wesley, 1993.

[Williams 91] Williams, T.W., B. Underwood, and M.R. Mercer, "The Interdependence between Delay Optimization of Synthesized Networks and Testing," *Proc. 28th Design Automation Conf.*, San Francisco, CA, pp. 87-92, 1991.

[Wong 89] Wong, D., G. De Micheli, and M. Flynn, "Inserting Active Delay Elements to Achieve Wave Pipelining," *Proc. Int. Conf. Computer Aided Design*, Santa Clara, CA, pp. 270-273, Nov. 6-9, 1989.

[Woods 86] Woods, M.H., "MOS VLSI Reliability and Yield Trends," *Proc. IEEE*, Vol. 74, No. 12, Dec. 1986.

[Yamabe 85] Yamabe, K., and K. Taniguchi, "Time-Dependent Dielectric Breakdown of Thin Thermally Grown SiO_2 Films," *IEEE Trans. on Electron Devices*, Vol. ED-32, No. 2, pp. 423-428, Feb. 1985.