Reconvergent Fanout Analysis of Bounded Gate Delay Faults

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Hillary Grimes III

Certificate of Approval:

Charles E. Stroud
Professor
Electrical and Computer Engineering

Vishwani D. Agrawal, Chair
James J. Danaher Professor
Electrical and Computer Engineering

Victor P. Nelson
Professor
Electrical and Computer Engineering

George T. Flowers
Interim Dean
Graduate School

Reconvergent Fanout Analysis of Bounded Gate Delay Faults

Hillary Grimes III

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
August 9, 2008

Reconvergent Fanout Analysis of Bounded Gate Delay Faults

Hillary Grimes III

Permission is granted to Auburn University to make copies of this thesis at its discretion, upon the request of individuals or institutions and at their expense. The author reserves all publication rights.

_____

Signature of Author

_____

Date of Graduation

Vita

Hillary H. Grimes III, son of Mr. Hillary H. Grimes Jr. and Mrs. Rene P. Grimes, was born August 21, 1981, in Anniston, Alabama. He graduated with honors from Daphne High School in 1999. He attended Faulkner State Community College in Bay Minnette, Alabama, for two years, and graduated summa cum laude with an Associate of Science degree in Pre-Engineering. He then entered Auburn University in January, 2002, and graduated magna cum laude with a Bachelor of Science degree in Electrical and Computer Engineering in May, 2004.

Thesis Abstract

Reconvergent Fanout Analysis of Bounded Gate Delay Faults

Hillary Grimes III

Master of Science, August 9, 2008
(B.S., Auburn University, 2004)
(A.S., Faulkner State Community College, 2001)

86 Typed Pages

Directed by Vishwani Agrawal

To determine the quality that a set of gate delay tests provides for testing gate delay faults, gate delay fault simulation must determine the minimum size detectable for detected gate delay faults. The minimum size detectable is the minimum faulty gate delay that must be present for the test to detect the fault. Given two tests that detect the same fault, the test that detects the fault with a smaller size is considered a higher quality test for that fault. When bounded gate delays are used, gate delay fault simulation involves bounded delay simulation of the fault-free circuit and an evaluation of the faulty waveforms.

Whenever signals in a combinational circuit diverge from a fanout point and reconverge later, the inputs to the reconvergent gate are correlated. In conventional bounded delay simulation and gate delay fault simulation, these correlations are ignored. In this work, we present a method for adding reconvergent fanout analysis to bounded delay simulation and gate delay fault simulation. In bounded delay simulation, considering information about how signals are correlated due to reconvergent fanout provides more accurate evaluation of simulation waveforms than previous approaches that ignore this information. In gate delay fault simulation, this correlation information provides more accurate evaluation of

the minimum size detectable for detected gate delay faults. Results for gate delay fault simulation show that ignoring these correlations produces pessimistic calculations for the minimum sizes detectable.

Acknowledgments

I would like to gratefully acknowledge the assistance, support, and guidance from Dr. Vishwani D. Agrawal. I thank Dr. Victor P. Nelson and Dr. Charles E. Stroud for being on my committee. I thank Dr. Soumitra Bose for his valuable assistance and technical contribution. My sincere thanks to my parents for their encouragement, help, and support.

Style manual or journal used LaTeX: A Document Preparation System by Leslie Lamport (together with the style known as "aums").

Computer software used The document preparation package TeX (specifically LaTeX) together with the departmental style-file `aums.sty`. The images were generated using XFig.

Table of Contents

List of Figures xi

List of Tables xiii

1 Introduction 1
1.1 Problem Statement . . . 1
1.2 Contribution of Thesis . . . 2
1.3 Organization of Thesis . . . 2

2 Background 3
2.1 Delay Fault Modeling . . . 4
2.1.1 Transition Fault Model . . . 4
2.1.2 Path Delay Fault Model . . . 5
2.1.3 Gate Delay Fault Model . . . 6
2.1.4 Segment Delay Fault Model . . . 6
2.2 Static Timing Analysis . . . 7
2.2.1 Slack . . . 8
2.3 Bounded Delay Simulation . . . 8
2.3.1 Simplified Signal Waveforms . . . 9

3 Previous Work 11
3.1 Gate Delay Fault Simulation . . . 11
3.1.1 Fault-Free Circuit Simulation . . . 13
3.1.2 Fault Detection . . . 15
3.1.3 Detection Gap . . . 18
3.2 Reconvergent Fanout Analysis . . . 20
3.2.1 Bounded Delay Simulation . . . 21
3.2.2 Delay Fault Simulation . . . 23

4 Reconvergent Fanout Analysis in Detection Threshold Evaluation 25
4.1 Motivation . . . 25
4.2 Detection Threshold Evaluation . . . 28
4.2.1 Ambiguity Lists . . . 29
4.2.2 Faulty Waveforms . . . 29
4.3 Examples . . . 33
4.4 Results on ISCAS85 Benchmark Circuits . . . 35

5 Reconvergent Fanout Analysis in Fault-Free Circuit Simulation 38
5.1 Motivation . . . 38
5.2 Fault-Free Circuit Simulation . . . 40
5.3 Results on ISCAS85 Benchmark Circuits . . . 43

6 Conclusion 46
6.1 Future Work . . . 46
6.2 Other Applications . . . 47
6.3 An Important Observation . . . 48

Bibliography 49

Appendices 54

A Gate Delay Fault Simulation Program - User Information 55
A.1 Hierarchical Bench Format . . . 55
A.2 Simulator Options . . . 57
A.3 Example Simulation . . . 58

B Gate Delay Fault Simulation Program - Implementation 64
B.1 Data Structures . . . 64
B.2 Static Timing Analysis . . . 66
B.3 Bounded Delay Simulation . . . 68
B.4 Detection Threshold Evaluation . . . 69
B.5 Program Evaluation . . . 71

List of Figures

2.1 Min-Max Simulation Waveforms. 10

3.1 Detection Threshold. 12

3.2 Illustrating Correlation of Inputs to a Reconvergent Gate. 19

3.3 Min-Max Simulation Waveforms. 21

3.4 Min-Max Simulation Waveforms. 22

4.1 Incorrect Gate Delay Detection Threshold. 26

4.2 Incorrect Gate Delay Detection Threshold for XOR Circuit. 27

4.3 Ambiguity Lists. 33

5.1 Incorrect Detection Threshold. 39

A.1 Hierarchical Benchmark Description of c17. 56

A.2 Hierarchical Benchmark Description for a Half Adder. 56

A.3 Simulator Options. 57

A.4 Example Circuit. 59

A.5 Hierarchical Bench Netlist for Simulation Example. 60

A.6 Input Vector File for Simulation Example. 60

A.7 Fault Coverage and Detection Gap Reports. 61

A.8 Hazard Free Outputs for Simulation Example. 61

B.1 Circuit and Gate Data Structures. 65

B.2 Fault and Fault Size Data Structures. 65

B.3 Ambiguity List Data Structure. 66

B.4 Critical Delay Calculation. 67

B.5 Slack Calculation. 67

B.6 Bounded Delay Simulation. 69

B.7 Fault Simulation. 70

List of Tables

3.1 Incorrect Detection Threshold Calculations for XOR Circuit. 17

4.1 Incorrect Detection Threshold Calculations. 26

4.2 Incorrect Detection Threshold Calculations for XOR Circuit. 28

4.3 Corrected Detection Threshold Calculations. 34

4.4 Corrected Detection Threshold Calculations for XOR Circuit. 34

4.5 Detection Gap Results for 1,000 Random Vectors. 35

5.1 Incorrect Detection Threshold Calculation. 40

5.2 Corrected Detection Threshold Calculation. 42

5.3 Largest Output EA and LS Values for 10,000 Random Vectors. 43

5.4 Detection Gap Results for 1,000 Random Vectors. 44

B.1 Performance When Zero Delay Buffers Are Not Inserted on Fanout Branches. 72

B.2 Performance When Zero Delay Buffers Are Inserted on Fanout Branches. . 72

B.3 Performance With and Without Reconvergent Fanout Analysis. 73

Chapter 1

Introduction

As technology advances, manufactured VLSI devices are becoming more and more complex. As complexity of devices increases, the testing of devices to ensure correct operation becomes more and more difficult. Testing must not only ensure correct logical operation, but must also ensure correct timing operation.

Manufacturing process variations (variations in threshold voltage, channel length, etc.) and differences in operating environment (such as power supply and temperature variations) result in delay values within a manufactured device that vary. This uncertainty in gate and wire delays requires the delays within a circuit to be modeled as imprecise delays rather than a simple nominal delay value. One model that allows imprecise gate delays is the bounded gate delay model, where delays are lumped at gates, and a gate's delay is assumed to be somewhere between some minimum (lower bound) and maximum (upper bound) value. However, analysis techniques using the bounded gate delay model often produce results that are too pessimistic. As the uncertainty due to process variations increases, the need for more accurate analysis also increases.

1.1 Problem Statement

The problem solved in this thesis is: *A method to improve the accuracy for simulation of gate delay faults in the presence of reconvergent fanouts.*

1.2 Contribution of Thesis

We have analyzed the inaccuracy in bounded delay simulation and gate delay fault simulation when signal correlations due to reconvergent fanouts are ignored. A method for reconvergent fanout analysis during bounded delay simulation of the fault-free circuit is presented. The sizes of delay faults are an important parameter in determining the quality that a test set provides for gate delay faults, so another method is presented to accurately evaluate the minimum size detectable for detected gate delay faults in the presence of reconvergent fanouts.

One paper describing this work was presented at the *International Test Conference* 2007 [13] and another was presented at the *17th IEEE North Atlantic Test Workshop* 2008 [27].

1.3 Organization of Thesis

The thesis is organized as follows. In Chapter 2, we discuss a general background of delay fault modeling, static timing analysis, and min-max delay simulation. In Chapter 3, previous work in gate delay fault simulation and previous work in reconvergent fanout analysis during both bounded delay simulation and delay fault simulation are described. A method for reconvergent fanout analysis during detection threshold evaluation is presented in Chapter 4. In Chapter 5, a method for reconvergent fanout analysis in fault-free circuit simulation is presented to further improve the accuracy in computing the detection thresholds of detected gate delay faults. Conclusions and future work are discussed in Chapter 6.

Chapter 2

Background

The objective of delay testing is to ensure that a manufactured design meets its timing specifications. Defects that occur during fabrication of a VLSI device may increase circuit delays, causing an error if a transition is prevented from reaching an output within the clock period. Delay testing isolates those devices that will not operate within the designed timing constraints from those that operate as designed. In order to examine a circuit's timing operation, a delay test must propagate signal transitions through the design, which requires the application of vector pairs. The first vector of a delay test initializes the circuit and the second vector produces the required transitions [14, 37].

The delay of a signal transition in a combinational circuit has two main parts: switching delays and propagation delays. Switching delays are due to the input to output delay of gates. The switching delay of a gate is the time it takes for the gate's output to change after a change on an input. Propagation delays are due to the delay along wires. The propagation delay between gates is the time it takes a signal transition to travel from the output of a gate to the input of a fanout gate [14, 24]. For the bounded gate delay model, delays are lumped at gates, and a gate's delay is assumed to be somewhere between a minimum (lower bound) and maximum (upper bound) value.

2.1 Delay Fault Modeling

Physical defects such as imperfections in materials or missing contact windows that occur during device fabrication result in a difference between the chip's implemented hardware and it's intended design. A fault is a mathematical abstraction at the function level that represents physical defects and allows the development and application of analytical tools. A fault model should provide high confidence that faulty devices will be isolated from those that operate as indended by design [14, 37].

Some physical defects result in increased delays without causing an error in the device's logical function. If the increased circuit delay exceeds the intended clock period, the circuit will fail to operate as designed. Such defects that effect the performance of the device are modeled by a delay fault model. Three common fault models for delay faults are the transition fault model, the path delay fault model, and the gate delay fault model [14, 37].

2.1.1 Transition Fault Model

The transition fault model [18, 56, 63] is a qualitive delay fault model where circuit delays are often not even considered [37]. Each line in the circuit has two transition faults, a slow-to-rise fault and a slow-to-fall fault, and it is assumed that the delay fault only affects a single gate in the circuit. A slow-to-rise (fall) transition fault makes the falling (rising) signal on that line slow [14, 37]. The transition fault model is used to model large delay faults where the increase in delay due to the fault is assumed to be large enough to prevent the transition from reaching the circuit's outputs before the clock [14, 37]. To model small delay faults, two types of models are common, the path delay fault model and the gate delay fault model [13, 34, 52].

2.1.2 Path Delay Fault Model

The path delay fault [40, 54, 58] models small delay faults, and is usually considered to be closest to the ideal model for defects that increase circuit delays [37]. A path delay fault occurs when the cumulative propagation delay, rising or falling, of a combinational path exceeds a specified limit. A path is a connected chain of gates from primary input to primary output, between two clocked memory elements (flip-flops), or from a primary input to a clocked memory element. The length of a path is the sum of delays due to gates and interconnects along the path. The major disadvantage of the path delay fault model is that the number of paths can be very large in a circuit, and the number of path delay faults can be exponential in the number of gates [53]. Therefore, testing all path delay faults in a circuit is impractical. Often, only a subset of paths with a delay greater than a specified threshold are considered [14, 37].

Research continues on the selection of a relevant set of path delay faults that should be tested. In spite of sevral available results [44, 64], there is no accepted way of selecting paths. Because of their large number, path delay fault simulation of sequential circuits is a complex problem [8, 10, 11, 17]. For combinational circuits or for scan-mode sequential circuits, efficient methods of simulation [26] and diagnosis [47] have been reported.

Although several path delay test generation systems have been reported [23, 48], significant difficulties are encountered in this area. Tests generated with single-fault assumption can be invalidated in the presence of multiple faults. Such tests are known as *non-robust* [14, 25, 36]. Tests that cannot be invalidated by any arbitrary delays in the circuit are called *robust* [40]. In general, it is found that no robust tests exist for many path delay faults. Fortunately, it is possible to *validate* some, though not all, non-robust tests [19, 59].

Such tests are known as *validatable non-robust* (VNR) tests. The problem of delay test invalidation can also be reduced by finding tests that do not generate timing *hazards*. Hazards are timing ambiguities in the transient behavior of a signal. The use of hazard-free tests can improve the quality of delay testing and diagnosis [38, 57].

In view of the problems of path delay testing, as outlined above, we often derive pessimistic inferences about the speed of the circuit [9, 50]. A contribution of the research reported in this thsis is in reducing such pessimism.

2.1.3 Gate Delay Fault Model

The gate delay fault model assumes that a delay fault is lumped at a single gate [14, 37]. Unlike the transition fault model, the gate delay fault model is a quantitive fault model in which circuit delays are considered [37]. A gate delay fault is modeled as an increase in the input to output delay of a gate. The increase in delay due to a fault is called the size of the fault, which is an important parameter in determining the ability of a test to detect gate delay defects [14, 37].

The gate delay fault model has the advantages of the ability to model small delay defects and the number of gate delay faults in a circuit is linear in the number of gates [14, 37]. Because of the single gate delay fault assumption, however, the gate delay fault model has the disadvantage that tests derived for gate delay faults can fail to detect delay faults that are caused by the cumulative delay of multiple small delay defects [37].

2.1.4 Segment Delay Fault Model

A delay model that strikes a compromise between the path and gate delay models is the *segment delay model* [28]. In this model, faults on strings of gates within specified maximum

length (in number of gates) are considered. Thus, a unit length segment corresponds to the gate delay model and an infinite length segment corresponds to the path delay model. In practice, the segment length can be chosen from such considerations as spatial correlations among gates and analysis complexity.

2.2 Static Timing Analysis

Static timing analysis is a vector-independent analysis of the timing behavior of combinational paths through a circuit [3, 29, 60]. Static timing analysis is usually used in the design process before manufacturing to calculate the circuit's worse-case timing behavior and to identify the circuit's critical path [20, 62, 66]. The critical path is the longest delay combinational path in the circuit. The delay of the critical path determines the smallest clock period at which the circuit can function correctly [14, 34].

Static timing analysis is included in the standard design flow of VLSI circuits and is supported by commercial tools [6]. Its applications have been proposed for timing optimization of synchronous sequential circuits at the functional clocking [43] and physical design [22] levels. Though the analysis formally verifies the correctness of the timing behavior of the design, it does not eliminate the need for delay testing. This is because no fabrication process is perfect and errors and variations are introduced, requiring every manufactured device to be tested.

A simple form of timing analysis may use fixed (or nominal) delays for gates. Delay variations are anayzed by using either a statistical delay model [1, 5, 35, 42, 49, 62] or a bounded delay model [12, 13, 16, 39, 55, 61]. In the research reported in this thesis we use the bounded delay model.

2.2.1 Slack

Static timing analysis is used to calculate the slacks of paths through the circuit. For a path P with delay d, the slack of P is the quantity $T_c - d$, where T_c is the clock period. Slacks are defined for a gate G as $T_c - D_G$, where D_G is the delay of the longest delay path from input to output through G [32, 33, 34, 49, 51, 52]. For bounded gate delays, Iyengar *et al.* [34] distinguish between two different calculations for the slack at a gate, the slack for the design process and the slack for testing. For the calculation of slacks during the design process, maximum gate delays are used when calculating the delay of the longest delay path through gate G. The reason is that timing analysis during design has to ensure that all outputs are stable before the clock, even if the delay for every gate is at it's maximum value. In testing, minimum gate delays are used to calculate the longest path delay through gate G because testing should guarantee detection of delay faults. A test to detect a delay fault has to ensure the fault will be detected without being masked by gate delays that are decreased due to delay variations.

2.3 Bounded Delay Simulation

Delay simulation is a dynamic timing analysis that analyzes the timing behavior of a circuit for a set of inputs and signal transitions. During bounded delay simulation of a vector pair, two vectors, V_1 and V_2, are applied to the circuit's inputs, and signal timing is analyzed for the pair. After Vector V_1 is applied, the circuit is assumed to have stabalized before V_2 is applied. Each gate G has two logic values, $IV(G)$ and $FV(G)$. $IV(G)$ is the initial value, which is the output logic value for gate G after vector V_1 is applied, and $FV(G)$ is the final value, which is the output logic value after vector V_2 is applied [37, 32, 34, 51, 52].

2.3.1 Simplified Signal Waveforms

The timing of a signal at the output of gate G can be represented using simplified signal waveforms descibed in [15], where signal timing is represented using two quantities, $EA(G)$ and $LS(G)$. $EA(G)$ is the earliest arrival time for the output of gate G after vector V_2 is applied, and $LS(G)$ is the latest stabalization time for the gate's output after vector V_2 is applied. The waveform at the output of a gate G is at $IV(G)$ before time $EA(G)$, and at $FV(G)$ after time $LS(G)$. Between times $EA(G)$ and $LS(G)$, gate G has an ambiguous (unknown) value (X). During this ambiguity region, the waveform at the output of G can have any number of pulses [13, 32, 34, 37, 51, 52].

Figure 2.1 illustrates bounded delay simulation waveforms. Minimum and maximum gate delays are shown beside each gate. The output of gate C is at logic 1 before time $t = 1$, and logic 0 after time $t = 3$. Due to process variations, the signal can change any time between times 1 and 3. Between times 1 and 3, the output of C is ambiguous (X). Similarly, the output of gate E changes from 0 to 1 sometime between times 2 and 5. The output of E has the unknown value X in the ambiguous region between times 2 and 5.

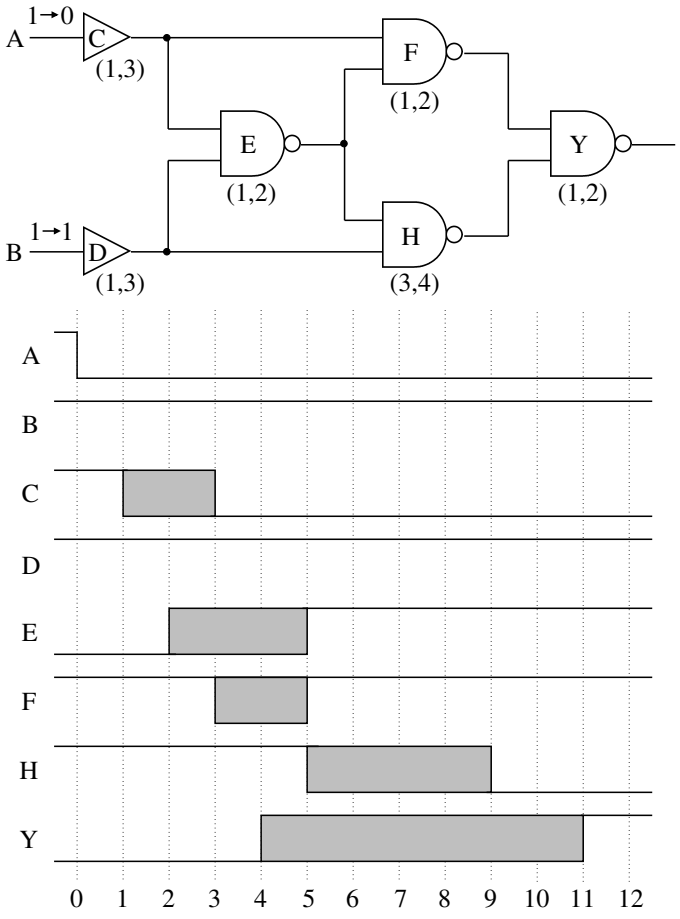

Figure 2.1: Min-Max Simulation Waveforms.

Chapter 3

Previous Work

The first section in this chapter describes previous work on computing the sizes of detected gate delay faults during gate delay fault simulation. The second section describes previous work for reconvergent fanout analysis in both bounded delay simulation and delay fault simulation.

3.1 Gate Delay Fault Simulation

Given a set of vectors, gate delay fault simulation is used to determine the quality the set of test vectors provides for detecting gate delay faults in the circuit [12, 34, 37, 52]. In order to detect a gate delay fault, the test must both place a transition at the fault site and propagate its effect to an observation point. Therefore, testing a gate delay fault requires two vectors: the first to set the initial value of the transition at the fault site, and the second to both make the transition to the final value at the site and propagate the fault's effect to an observation point. For a rising (falling) transition, the first vector places a logic 0 (logic 1) at the fault site, and the second vector is a stuck-at-0 (stuck-at-1) test for the same fault site [34, 52].

To determine the quality of a set of tests, gate delay fault simulation determines both how many faults are detected and their minimum size detectable. The minimum size detectable is the minimum faulty delay that must be present for the test to detect the fault [21, 34, 52]. Given two tests that detect the same fault, the test that detects the fault with a smaller size is considered a higher quality test for that fault. The minimum

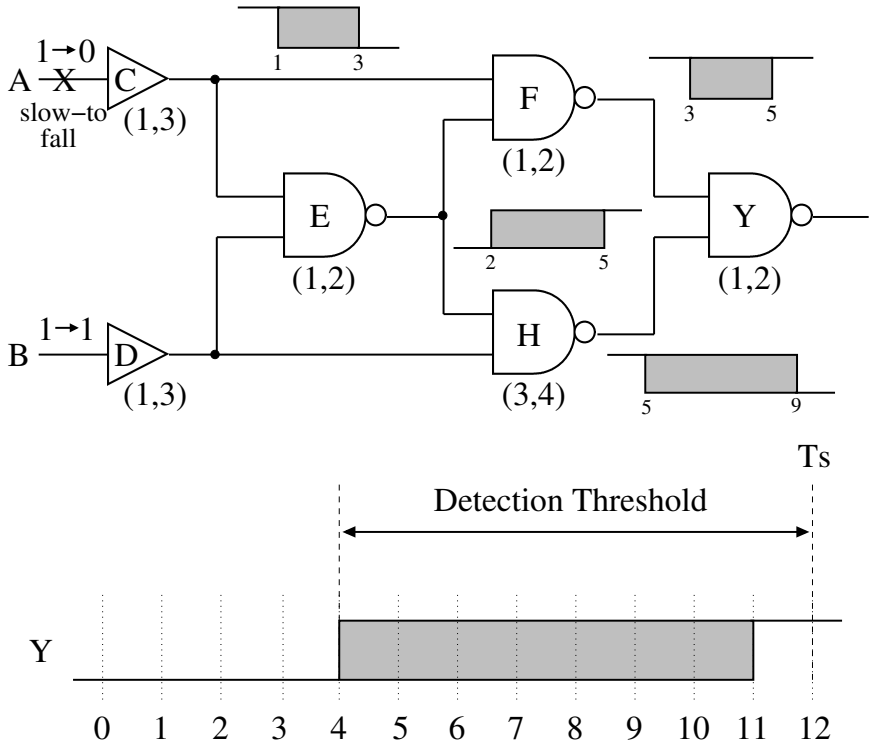

Figure 3.1: Detection Threshold.

size gate delay fault that is guaranteed to be detected by a test is defined as the detection threshold [34, 37]. A test is guaranteed to detect a gate delay fault if the size of the delay fault δ is greater than the fault's detection threshold for that test. The size of a delay fault at a gate G is defined in [34] as the amount of faulty delay added to the earliest arrival time of G ($EA(G)$). When a gate delay fault of size δ is present at G, the output of G transitions after time $EA(G) + \delta$ [34, 37].

Figure 3.1 illustrates the detection threshold of a slow-to-fall gate delay fault on input A in the example circuit of Figure 2.1. A slow-to-fall fault at A of size δ shifts the earliest arrival time of A by δ time units. A delay of $EA(A)$ by δ time units delays the earliest

arrival times for gates C, E, F, H, and Y by δ time units. For the fault to be detected, the size of the fault should be large enough to shift the output at Y far enough such that the initial value of Y $IV(Y)$ is sampled at the sample clock. If output Y was sampled at $T_s = 12$, a fault at A of size larger than 8 ($\delta > 8$) is required to guarantee that $IV(Y)$ is sampled at time 12. Therefore, the detection threshold for the slow-to-fall fault at A in Figure 3.1 is 8.

Iyengar *et al.* [34] describe a method to compute the detection thresholds of detected gate delay faults. In this method, for every gate delay fault test (vector pair V_1 and V_2), bounded delay simulation computes the simplified signal waveform values IV, FV, EA, and LS to summarize gate output waveforms for the fault-free circuit, and gate delay fault simulation computes information about circuit waveforms under the influence of a fault. This information about faulty waveforms is compared to fault-free waveforms at circuit outputs to determine the detection threshold of detected gate delay faults.

3.1.1 Fault-Free Circuit Simulation

The calculations for EA and LS to represent fault-free waveform timing are described in [15, 32, 33, 34]. For a vector pair, EA and LS for each primary input, PI, is initialized, and EA and LS for each gate is evaluated in one forward pass over the circuit. For a primary input that has a transition, $IV(PI) \neq FV(PI)$, EA and LS are both initialized to the time at which the stimulus is applied to the input. If a primary input does not have a transition, $IV(PI) = FV(PI)$, it's waveform is at a steady logic value, which is represented by setting EA and LS to:

$$EA(PI) = \infty \qquad LS(PI) = -\infty$$

EA and LS are evaluated for all other gates using EA and LS values at the gate's inputs. For all inputs (i) to a gate G whose initial value ($IV(i)$) is a controlling input logic value for G, EA is calculated as:

$$EA(G) = max\{EA(i)\} + minDelay(G)$$

If no input to G has an initial value that is a controlling input logic value for G, then for all inputs (i):

$$EA(G) = min\{EA(i)\} + minDelay(G)$$

For all inputs to G whose final value ($FV(i)$) is a controlling input logic value for G, LS is calculated as:

$$LS(G) = min\{LS(i)\} + maxDelay(G)$$

If no input has a final value that is a controlling input logic value for G, then for all inputs:

$$LS(G) = max\{LS(i)\} + maxDelay(G)$$

If these calculations result in $EA(G) > LS(G)$, the output of G is at a steady logic value. If this occurs, EA and LS are set to:

$$EA(G) = \infty \qquad LS(G) = -\infty$$

to represent a steady logic value on the output of gate G.

3.1.2 Fault Detection

For each gate delay fault simulated, each signal in the faulty circuit has a fault propagating value, FPV, which is the signal's value in the presence of a stuck-at fault at the fault site such that the fault site is stuck-at it's initial value [32, 34]. Therefore, simulation of a gate delay fault at gate G at the fault site, $FPV(G)$ is the initial value of G:

$$FPV(G) = IV(G)$$

At a gate G outside the cone of influence of the faulty gate, the waveforms are unaffected by the fault. Gate G lies outside the cone of influence of the faulty gate if no directed path exists from the output of the faulty gate to an input to gate G. In this case, FPV is set to the gate's final value:

$$FPV(G) = FV(G)$$

At a gate G inside the cone of influence, the FPV is evaluated using boolean logic on the FPVs of its inputs. Gate G lies inside the cone of influence if a directed path from the output of the faulty gate to an input to gate G does exist. Propagating the FPVs propagates the fault effects through the circuit, and can be used to determine whether or not a delay fault of any size can be detected by the test. If at an output, the FPV differs from the output's final value, the fault is considered detected by that test, and the fault's detection threshold determines the size required for detection [32, 34, 37].

To determine the detection threshold for a given delay fault f, a reference fault size, $\rho(G)$, and two reference times, $RTa(G)$ and $RTb(G)$, are evaluated for each gate G, which provide timing information about the faulty waveforms. Certain assertions are made about

the signal waveform at G. The logic value at a gate G for a given delay fault of size δ is at $FPV(G)$ between times $RTa(G)$ and $RTb(G) + \delta$, provided $\delta > \rho(G)$. If the size of the fault is less than the reference size, the fault has no effect at the output of G, and the waveform does not differ from the fault-free waveform [32, 34, 37, 52]. For a gate G, at the fault site, $FPV(G)$ is at $IV(G)$ for the time between $-\infty$ and $EA(G)$:

$$\rho(G) = 0 \qquad RTa(G) = -\infty \qquad RTb(G) = EA(G)$$

and for a gate G, outside the cone of influence of the fault, $FPV(G)$ is at $FV(G)$ between the times $LS(G)$ to ∞:

$$\rho(G) = 0 \qquad RTa(G) = LS(G) \qquad RTb(G) = \infty$$

For a gate G, inside the cone of influence of the fault site, if all inputs (i) have sensitizing fault propagating values $FPV(i)$, $RTa(G)$ and $RTb(G)$ are calculated as follows:

$$RTa(G) = max\{RTa(i)\} + maxDelay(G)$$
$$RTb(G) = min\{RTb(i)\} + minDelay(G)$$

To calculate the reference size, $\rho(G)$, the value ω is calculated, which represents the minimum delay at some input to G that overcomes the inertia of G so as to observe the delay at the output of G:

$$\omega = max\{0,\ max\{RTa(i)\} + maxDelay(G) - min\{RTb(i)\}\}$$

The reference size for gate G is then:

$$\rho(G) = max\{max\{\rho(i)\},\ \omega\}$$

For a gate G, inside the cone of influence of the fault site, if G has inputs with controlling fault propagation values, an input i is chosen such that $FPV(i)$ is a controlling input value for gate G. $RTa(G)$ and $RTb(G)$ are then calculated as follows:

$$RTa(G) = RTa(i) + maxDelay(G)$$

$$RTb(G) = RTb(i) + minDelay(G)$$

$$\rho(G) = max\{max\{\rho(i)\},\ RTa(i) + maxDelay(G) - RTb(i)\}$$

The detection threshold for a delay fault f ($DT(f)$) at a circuit output z is then:

$$DT(f) = max\{\rho(z),\ T_s(z) - RTb(z)\}$$

Table 3.1: Incorrect Detection Threshold Calculations for XOR Circuit.

signal	EA	LS	FPV	$\rho(s)$	RTa	RTb
A	0	0	1	0	$-\infty$	0
B	∞	$-\infty$	1	0	$-\infty$	∞
C	1	3	1	0	$-\infty$	1
D	∞	$-\infty$	1	0	$-\infty$	∞
E	2	5	0	0	$-\infty$	2
F	3	5	1	0	$-\infty$	3
H	5	9	1	0	$-\infty$	5
Y	4	11	0	0	$-\infty$	4

where T_s is the sample time of the output. The minimum size fault detectable is then the minimum detection threshold for all outputs and all tests [12, 32, 34, 37, 52].

Table 3.1 show the calculations for EA, LS, fault propagating values, and the three reference quantity calculations when evaluating the detection threshold of a slow-to-fall fault on input A in the example circuit of Figure 3.1. If the output Y were sampled at time $T_s = 12$, then the detection threshold would be:

$$DT(A) = T_s(Y) - RTb(Y)$$

$$DT(A) = 12 - 4 = 8$$

3.1.3 Detection Gap

The detection gap described by Iyengar *et al.* [34] provides a way to relate the detection threshold of a detected gate delay fault to the slack at the fault site. The detection gap for a detected gate delay fault ($gap(G)$) is:

$$gap(G) = DT(G) - slack(G)$$

where $slack(G)$ is the slack for testing purposes defined in [34], which is the sum of all minimum gate delays along the longest delay path through gate G. If a vector pair detects a fault at G such that $gap(G) = 0$, the smallest possible delay fault at G has been detected. If $gap(G) > 0$, there is a possibility that there exists a better test to detect a gate delay fault at G with a smaller detection threshold [27, 34].

The smaller the detection gaps are for a set of vectors, the better quality that set provides for detecting gate delay faults. Suppose a gate delay fault is detected at a gate by

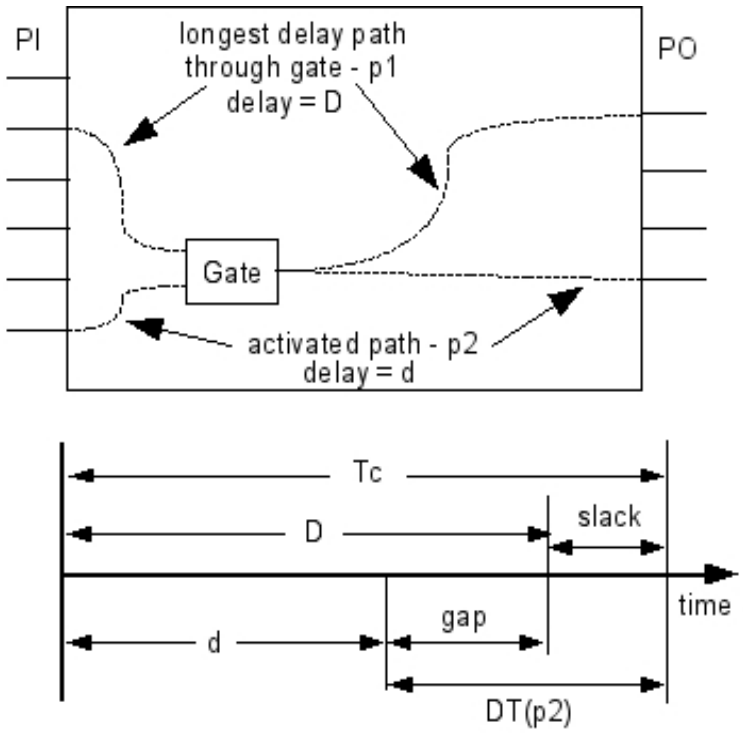

Figure 3.2: Illustrating Correlation of Inputs to a Reconvergent Gate.

activating a path $p2$, which is shorter than the longest path, $p1$, through that gate as shown in Figure 3.2. Path delays D and d in Figure 3.2 are the sums of all minimum gate delays along paths $p1$ and $p2$ that pass through the gate. For a clock period T_c, $T_c \geq D \geq d$, and the slack for the gate is $Tc - D$, where D is the delay of the longest delay path, $p1$. If the fault is detected through the shorter path $p2$, then the detection gap is $DT(p2) - slack$, which is larger than 0 by the amount $D - d$. Ideally we would like to detect the smallest gate delay fault possible, so a better test would detect the delay fault through path $p1$. If detected through $p1$, the detection gap would be $DT(p1) - slack$, which is 0 because $DT(p1) = slack$.

3.2 Reconvergent Fanout Analysis

When signals produced by a common fanout point reconverge, the inputs to the reconvergent gate are correlated [7, 12, 13, 16, 27, 30, 39, 41, 61]. Consider the fanout at the output of gate C in Figure 3.3, which reconverges at gate F. Figure 3.4 illustrates the correlation of both inputs to gate F (signals C and D). Suppose signal C changes at time x, which is somewhere between times 1 and 3. Since gate C's output is an input to gate E, the signal at E follows the signal at C in time, and δ cannot be smaller than the minimum delay of gate E, which is 1. Therefore, the output of E cannot change before time $x + minDelay(E)$, or $x + 1$.

Conventional bounded delay simulation ignores correlations of signals at inputs to reconvergent gates, and is known to produce pessimistic results [12, 13, 16, 27, 30, 31, 39, 41, 61]. In Figure 3.3, the inputs to gate F are correlated, and the input from C (top input) transitions to a controlling input logic value for gate F before the input from E (bottom input) transitions away from a controlling input value for F. Therefore, there is always a controlling logic value on at least one input to gate F. Conventional min-max delay simulation produces an ambiguity region at the output of F between times 3 and 5, which can never occur in an actual circuit. Propagating its effect to the output at gate Y would produce extra ambiguity between times 4 to 6. Removing, or suppressing, the erroneously produced static hazard results in the signal at output Y transitioning from a 0 to a 1 at some time between 6 and 11, instead of times 4 and 11 from conventional bounded delay simulation.

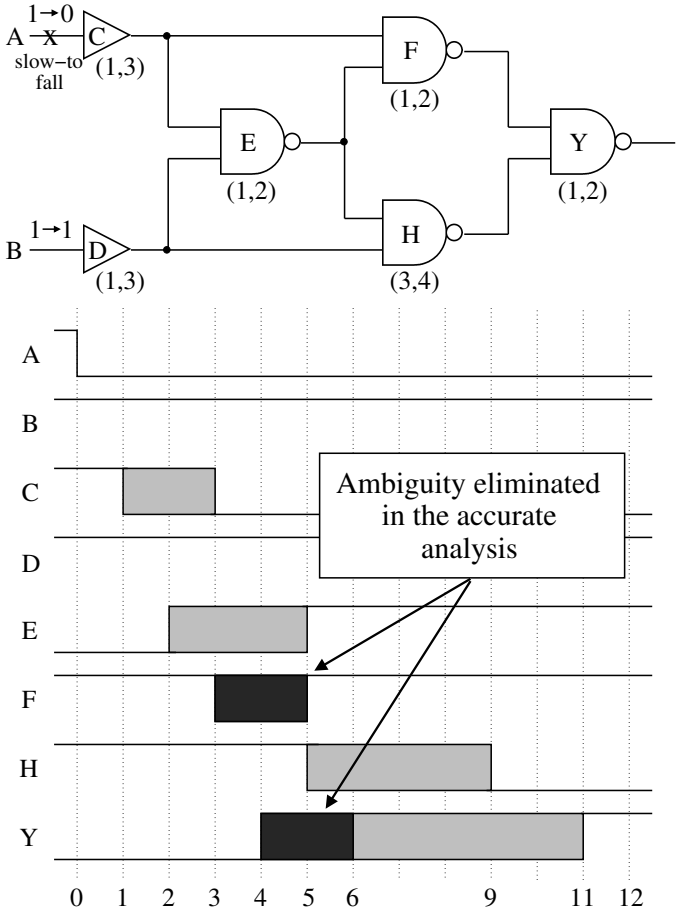

Figure 3.3: Min-Max Simulation Waveforms.

3.2.1 Bounded Delay Simulation

Chakraborty *et al.* [16] present an algorithm for accurate timing analysis in the presence of reconvergent fanouts. In this method, gates are evaluated using a thirteen valued waveform algebra. If a gate output is evaluated to be a constant value, it is guaranteed that for all possible gate input transition orderings, the output remains at a constant value. If the gate's output is not evaluated to be a constant value, the gate is further analyzed to

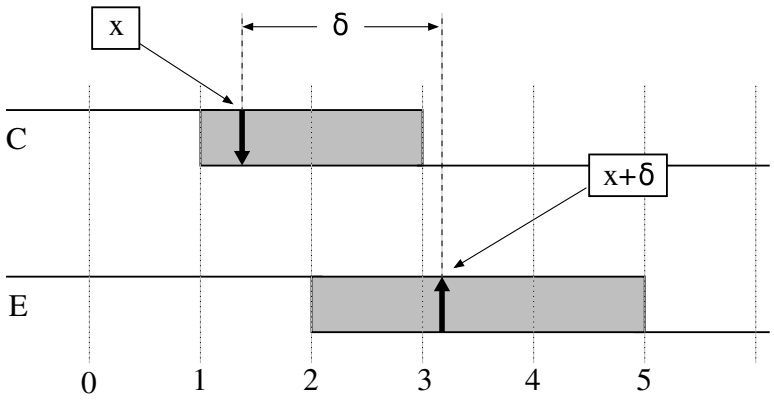

Figure 3.4: Min-Max Simulation Waveforms.

determine whether or not a hazard occurring at the output is masked. This analysis tries to determine for every input p that is different from input q if the transition on p is ordered relative to the transition on q. If a transition on p is ordered relative to a transition on q, the time at which the transition occurs on q depends on the time at which the transition occurs on q. Any ordering information detected is used to determine if hazards occurring at the gate's output should be masked [16]. Although this method results in an accurate timing analysis, its complexity would prevent it's use in logic simulation [12, 13].

An event-driven symbolic min-max simulator is described by Linderman and Leeser [41] that eliminates common ambiguity due to reconvergent fanout. In the event graph, delays to a reconvergent gate from events back to the last common event (event from the common fanout point) are used to determine the common ambiguity to eliminate [41]. This method produces less pessimistic results in the presence of reconvergent fanouts, but it's worse-case complexity is exponential in circuit size [16]. Time-symbolic simulation [31] and coded time-symbolic simulation [30] are two methods that also produce accurate results in the

presence of reconvergent fanouts, but their worse-case complexity is also exponential in circuit size [16].

Lalgudi *et al.* [39] present an event-driven min-max delay simulator on the MARS [2] parallel computer that removes the common ambiguity between events at the inputs to a reconvergent gate before the gate is evaluated. To determine the common ambiguity between events, the event data structure used has an "ambiguity descriptor list" that contains fanout stem IDs that are originators of ambiguity. For each fanout stem ID, modified ambiguity information is stored. The modified ambiguity information is needed when determining common ambiguity of two events at the inputs to a reconvergent gate because both events are modified differently as they propagate along two different paths from the fanout stem to the reconvergent gate. Both events may have been modified such that any common ambiguity has been eliminated. The ambiguity descriptor lists of events can grow very large, so the simulator limits the list's maximum size, which may result in pessimism [39].

3.2.2 Delay Fault Simulation

A path delay fault simulation algorithm using bounded gate delays that considers correlations between inputs to reconvergent gates is described by Bose and Agrawal [12]. In this approach, each gate has an "ambiguity list" that is propagated with events during event-driven simulation. Ambiguity list propagation is similar to fault list propagation of concurrent fault simulation. When a reconvergent gate is evaluated, the ambiguity lists of fanin gates provide information about how the reconvergent gate's inputs are correlated due to a common fanout point. If this information is such that ambiguity due to a static hazard at the output of the reconvergent gate cannot occur, the hazard is suppressed [12, 13].

Each element of the ambiguity list at gate G contains:

- Gate ID of a fanout point that causes an ambiguity region at G
- Minimum delay from the fanout to G
- Maximum delay from the fanout to G

During simulation, if the current value of a gate G is a Boolean value (logic 0 or logic 1), the ambiguity list at G is empty. If the current value of G is ambiguous (X), then the ambiguity list at G may not be empty [13].

When an event arrives at gate G and the output of G is evaluated to be ambiguous (X) for the current simulation time, every fanout point (element) in the ambiguity lists of fanin gates to G is examined to evaluate the ambiguity list at G's output. If a fanout point f is found at more than one input to gate G, then G is a reconvergent gate with common fanout f. When this occurs, if a fanout point has a minimum delay to an input transitioning from a controlling to a sensitizing logic value that is larger than the maximum delay to an input that is transitioning from a sensitizing to controlling value, the hazard at the output of the reconvergent gate is suppressed, the output is set to be stable, and the hazard list is set to *null*. If hazard suppression does not occur, then the fanouts at the gate's input hazard lists are updated with the gate's delay values and propagated to the output hazard list. Results presented for the ISCAS85 combinational benchmark circuits show error as high as 20% when measuring the critical path delay using bounded delay simulation [12, 13].

Chapter 4

Reconvergent Fanout Analysis in Detection Threshold Evaluation

This chapter presents a method to determine the detection threshold of detected faults during gate delay fault simulation that is more accurate than previous approaches. This method adds a reconvergent fanout analysis technique to the detection threshold evaluation method presented by Iyengar *et al.* [34]. The analysis technique presented in this chapter has appeared in recent papers [13, 27].

4.1 Motivation

Consider the circuit in Figure 4.1 which shows bounded delay simulation waveforms for a test to detect a slow-to-fall gate delay fault on input B. Input A is held constant while B undergoes a falling transition. Min-Max bounds for all gate delays are shown beneath each gate in the figure. The worse-case delay for the maximum delays shown is 7 time units, so in this example, the sample period is set to $T_s = 8$. If the output Y is sampled at time 8, the value sampled at Y will always be at $FV(Y)$ for the fault-free circuit.

Table 4.1 shows earliest arrival (EA) and latest stabilization (LS) times, fault propagation values (FPV), reference fault sizes ($\rho(s)$), and reference times (RTa and RTb) calculated using the detection threshold evaluation method presented by Iyengar *et al.* [34]. Using these reference quantity calculations, the slow-to-fall gate delay fault at gate B is detected with a detection threshold of $T_s - RTb(Y) = (8 - 4) = 4$.

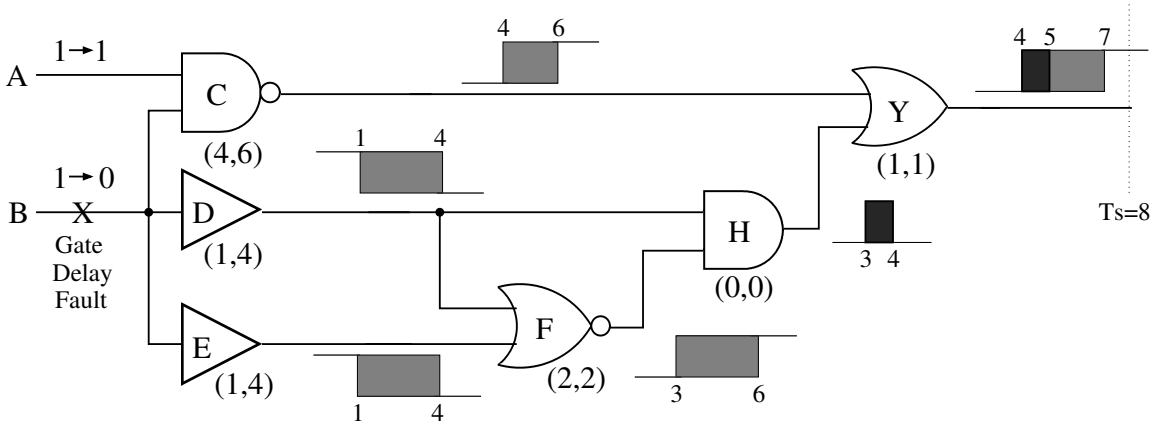

Figure 4.1: Incorrect Gate Delay Detection Threshold.

Table 4.1: Incorrect Detection Threshold Calculations.

signal	EA	LS	FPV	$\rho(s)$	RTa	RTb
A	∞	$-\infty$	1	0	$-\infty$	∞
B	0	0	1	0	$-\infty$	0
C	4	6	0	0	$-\infty$	4
D	1	4	1	0	$-\infty$	1
E	1	4	1	0	$-\infty$	1
F	3	6	0	0	$-\infty$	3
H	3	4	0	0	$-\infty$	3
Y	4	7	0	0	$-\infty$	4

From Figure 4.1 we see that the earliest arrival time for the rising transition at the output of Y is 4. To guarantee detection, the size of the fault at B must be large enough such that it's effect will shift the waveform at Y by at least 4 time units. A shift in the waveform at Y of 4 time units or greater will guarantee that the output Y is at $IV(Y)$ when the output is sampled at time 8, so a fault at gate B with size 4 or greater is detected.

Notice, however, that the fanout at the output of gate D reconverges at gate H. Because the top input (from D) is guaranteed to transition to a controlling input logic value (0) for

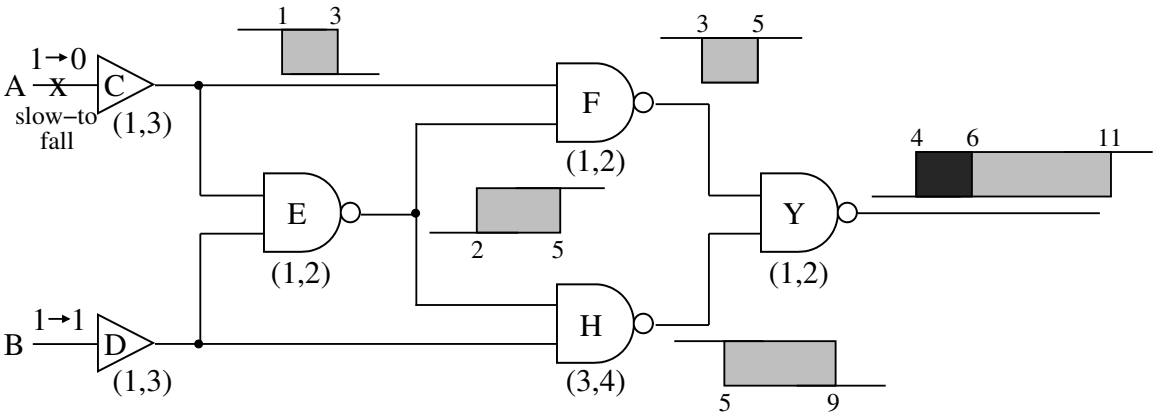

Figure 4.2: Incorrect Gate Delay Detection Threshold for XOR Circuit.

H before the bottom input (from F) transitions away from a controlling input value, there is always a controlling logic value on some input to gate H, and the output is stable at logic 0. The incorrect hazard at the output of gate H between times 3 and 4 results in the ambiguity at gate Y between times 4 and 5. A more accurate analysis would show that a slow-to-fall gate delay fault at gate B would only need to shift the output waveform at Y by 3 time units to guarantee detection, and the correct detection threshold should be evaluated as 3 instead of 4. This is because for the fault-free circuit, the ambiguous region between times 4 and 5 at the output of gate Y is guaranteed to be at $IV(Y)$ in the accurate analysis.

A similar error can be seen in the circuit of Figure 4.2. Table 4.2 shows the calculations for evaluating the detection threshold of a slow-to-fall gate delay fault at input gate A. Because both inputs to gate F are correlated in such a way that there is always a controlling logic value (0) at at least one input to F, the hazard produced between times 3 and 5 is incorrect. Using the Iyengar *et al.* [34] method, and a sample period of $T_s = 12$, the

Table 4.2: Incorrect Detection Threshold Calculations for XOR Circuit.

signal	EA	LS	FPV	$\rho(s)$	RTa	RTb
A	0	0	1	0	$-\infty$	0
B	∞	$-\infty$	1	0	$-\infty$	∞
C	1	3	1	0	$-\infty$	1
D	∞	$-\infty$	0	0	$-\infty$	∞
E	2	5	0	0	$-\infty$	2
F	3	5	1	0	$-\infty$	3
H	5	9	1	0	$-\infty$	5
Y	4	11	0	0	$-\infty$	4

detection threshold is evaluated as $T_s - RTb(Y) = (12 - 4) = 8$. A more careful analysis would show that fault sizes greater than 6 would be sufficient to shift the output at Y to guarantee detection. Here, an accurate analysis should calculate the detection threshold of the slow-to-fall fault at A as 6 instead of 8.

4.2 Detection Threshold Evaluation

This section describes how reconvergent fanout analysis is added to the method described in [34] for a more accurate detection threshold evaluation of detected gate delay faults. The method described here is similar to that in [34] in that gate delays are specified by their upper and lower bounds, and the stimulus to the circuit consists of a vector pair, V_1 and V_2. It is assumed the circuit has stabilized after the first vector is applied, before applying the second vector. The time frame of reference, $t = 0$, is assumed to be the instant the second vector is applied to the primary inputs. For each gate G in the fault-free circuit, values to represent the simplified signal waveforms, $IV(G)$, $FV(G)$, $EA(G)$, and $LS(G)$, are evaluated as in [15, 32, 34, 37].

For gate G, the fault propagating value, $FPV(G)$ is identical to the logic value of G when V_2 is applied to the circuit with the corresponding stuck-at fault at G. For a slow-to-rise (slow-to-fall) delay fault at a gate output, the corresponding stuck-at fault is the stuck-at-0 (stuck-at-1) fault at the same fault site. As described in [34], the fault propagating value for a gate G at the fault site is the initial value of G, $FPV(G) = IV(G)$, and the fault propagating value for a gate G outside the cone of influence of the fault site is the final value of G, $FPV(G) = FV(G)$. For a gate G inside the logic cone of the fault site, $FPV(G)$ is evaluated using fault propagating values at G's fanin gates and the usual rules of Boolean logic [34].

4.2.1 Ambiguity Lists

To provide information about the correlations between signals, each gate G has an ambiguity list similar to those described by Bose and Agrawal [12], where each element consists of the following:

- Gate IDs of fanout points (f) that lie in the forward cone of the fault site.
- $d(G, f)$: Minimum delay to G from f.
- $D(G, f)$: Maximum delay to G from f.

Unlike the ambiguity lists evaluated during logic simulation in the algorithm presented in [12], these lists are evaluated for each gate inside the cone of influence of the fault [13, 27].

4.2.2 Faulty Waveforms

For a given delay fault f of size δ, the reference fault size, $\rho(G)$, and the two reference times, $RTa(G)$ and $RTb(G)$, are evaluated for each gate G, such that the logic value at G

is guaranteed to be at $FPV(G)$ between the time interval $RTa(G)$ to $RTb(G)+\delta$, provided the size of delay fault f exceeds $\rho(G)$. For a gate at the fault site and for gates that lie outside the downcone of the fault site, the three reference quantities are evaluated as in [34]. For a gate G outside the cone of influence of the fault, G is at it's fault propagating value for the time between $LS(G)$ and ∞:

$$\rho(G) = 0 \qquad RTa(G) = LS(G) \qquad RTb(G) = +\infty$$

For a gate G at the fault site, G is at it's fault propagating value for the time between $-\infty$ and $EA(G)$:

$$\rho(G) = 0 \qquad RTa(G) = -\infty \qquad RTb(G) = EA(G)$$

If the fault site also happens to be a fanout stem, an ambiguity list is also created at the fault site with the gate ID of fanout point G and delays $d(G,G) = 0$ and $D(G,G) = 0$ [13, 27].

Reference quantities for gates inside the downcone of the fault site are evaluated along with ambiguity lists. The ambiguity lists at the gate's inputs are used to determine the ambiguity list at the output. For a gate G inside the cone of influence of the fault, if all inputs (i) have sensitizing fault propagating values, $RTa(G)$ and $RTb(G)$ are calculated identically as in [34]:

$$RTa(G) = max\{RTa(i)\} + maxDelay(G)$$

$$RTb(G) = min\{RTb(i)\} + minDelay(G)$$

Calculating the reference size is also identical to that described in [34]:

$$\omega = max\{0,\ max\{RTa(i)\} + maxDelay(G) - min\{RTb(i)\}\}$$

$$\rho(G) = max\{max\{\rho(i)\},\ \omega\}$$

The ambiguity list at the output of gate G is updated by adding an element for every fanout element found at the inputs (i) of G. The minimum and maximum delays of ambiguity list elements, $d(G, f)$ and $D(G, f)$, are updated using the minimum and maximum delays of G:

$$d(G, f) = min\{d(i, f)\} + minDelay(G)$$

$$D(G, f) = max\{D(i, f)\} + maxDelay(G)$$

For a gate G inside the cone of influence of the fault site, if inputs to G have both sensitizing and controlling fault propagating values, the following quantities are evaluated for all fanout points (f) of the hazard lists at inputs to G:

$$min_{DV}(f) = max\{d(M, f)\}$$

$$max_{SV}(f) = max\{D(m, f)\}$$

where M is an input to G with a controlling FPV and m is an input to G with a sensitizing FPV. The quantity $min_{DV}(f)$ represents the largest minimum delay from the common fanout point f to an input to G with a FPV that is a controlling input logic value for G. The quantity $max_{SV}(f)$ represents the largest maximum delay from f to an input to G with FPV that is a sensitizing input value for gate G [13, 27].

If no common fanout point f results in $min_{DV}(f) \geq max_{SV}(f)$, hazard suppression does not occur, and the reference quantities at the output of G are calculated using an input i with a dominating FPV:

$$\rho(G) = max\{\rho(i),\ RTa(i) + maxDelay(G) - RTb(i)\}$$

$$RTa(G) = RTa(i) + maxDelay(G)$$

$$RTb(G) = RTb(i) + minDelay(G)$$

Input i is chosen such that $\rho(i)$ is minimum among all such inputs of G. The ambiguity list at G is updated similar to before, adding an element to G's ambiguity list for every fanout point f appearing at G's inputs:

$$d(G, f) = min\{d(i, f)\} + minDelay(G)$$

$$D(G, f) = max\{D(i, f)\} + maxDelay(G)$$

If, however, a common fanout point at the inputs to G results in $min_{DV}(f) \geq max_{SV}(f)$, then hazard cancellation occurs. Here, fault effects cannot be propagated through gate G. The ambiguity list at the output of G is set to the empty (*null*) list and the following reference values are used at the output:

$$\rho(G) = 0 \qquad RTa(G) = -\infty \qquad RTb(G) = \infty$$

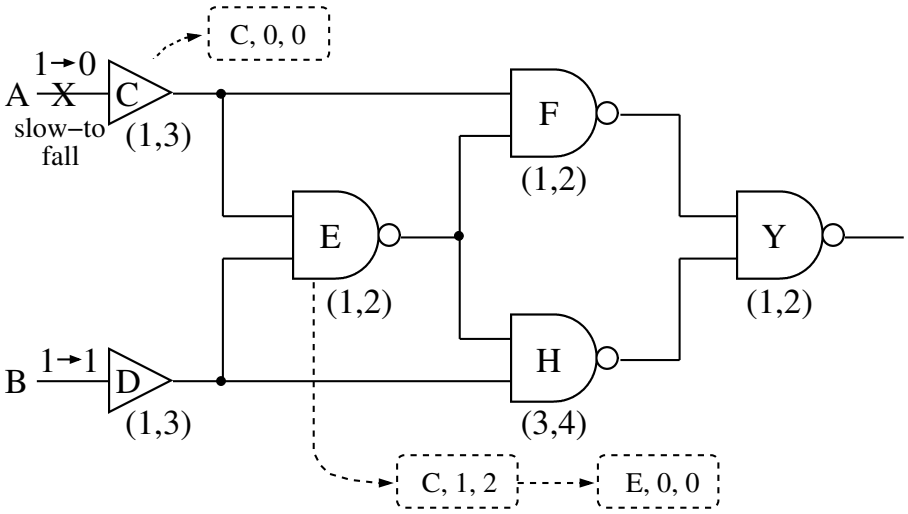

Figure 4.3: Ambiguity Lists.

4.3 Examples

Figure 4.3 shows the ambiguity lists for the example of Figure 4.2 when gate F is about to be evaluated. Since inputs to F have both sensitizing and controlling fault propagating values, min_{DV} and max_{SV} are calculated for all fanout points at the inputs to F. For the common fanout point C, the following quantities are evaluated:

$$minDV(C) = 1 \qquad maxSV(C) = 0$$

Since $minDV(C) \geq maxSV(C)$, the hazard is suppressed, and the output of gate F is stable. The ambiguity list at F is set to *null*, and the reference quantities are set to: $\rho(G) = 0$, $RTa(G) = -\infty$, and $RTb(G) = \infty$.

Tables 4.3 and 4.4 show the results obtained when this algorithm is used for the examples shown in both Figures 4.1 and 4.2. The second column shows fault propagating values,

Table 4.3: Corrected Detection Threshold Calculations.

signal	FPV	$\rho(s)$	RTa	RTb	Hazard List
A	1	0	$-\infty$	∞	$null$
B	1	0	$-\infty$	0	(B,0,0)
C	0	0	$-\infty$	4	(B,4,6)
D	1	0	$-\infty$	1	(B,1,4),(D,0,0)
E	1	0	$-\infty$	1	(B,1,4)
F	0	0	$-\infty$	3	(B,3,6),(D,2,2)
H	0	0	$-\infty$	∞	$null$
Y	0	0	$-\infty$	5	(B,5,7)

Table 4.4: Corrected Detection Threshold Calculations for XOR Circuit.

signal	FPV	$\rho(s)$	RTa	RTb	Hazard List
A	1	0	$-\infty$	0	$null$
B	1	0	$-\infty$	∞	$null$
C	1	0	$-\infty$	1	(C,0,0)
D	1	0	$-\infty$	∞	$null$
E	0	0	$-\infty$	2	(C,1,2),(E,0,0)
F	1	0	$-\infty$	∞	$null$
H	1	0	$-\infty$	5	(C,4,6),(E,3,4)
Y	0	0	$-\infty$	6	(C,5,8),(E,4,6)

and columns three through five show the three reference quantities for each signal. The last column shows the ambiguity lists at all gate outputs. For the example of Figure 4.1, the output sampling time was assumed to be $t = 8$. The correct detection threshold $(8 - RTb(Y))$ is evaluated as 3, while the original algorithm in Table 4.1 evaluates the threshold to be 4. For Figure 4.2, assuming an output sampling period of $T_s = 12$, the erroneous detection threshold is calculated as 8 $(12 - RTb(Y))$. This threshold is now correctly evaluated as 6.

Table 4.5: Detection Gap Results for 1,000 Random Vectors.

	Without Reconvergent Fanout Analysis		With Reconvergent Fanout Analysis	
	Average Detection Gap	Faults Detected with Gap ≤ 3.5	Average Detection Gap	Faults Detected with Gap ≤ 3.5
c432	72.4	8.80%	68.1	10.80%
c499	31.1	11.44%	29.2	18.20%
c880	21.3	40.18%	21.3	40.18%
c1355	33.4	10.90%	31.8	17.21%
c1908	48.1	15.94%	47.9	15.94%
c2670	34.2	29.70%	33.9	29.70%
c3540	43.9	20.45%	41.9	21.15%
c5315	28.3	36.48%	28.2	36.64%
c6288	395.2	0.45%	379.3	0.45%
c7552	36.3	11.95%	36.1	11.98%

4.4 Results on ISCAS85 Benchmark Circuits

Table 4.5 shows results obtained for ISCAS85 combinational benchmark circuits using the gate delay fault simulation method presented in this chapter. Each circuit was simulated for 1,000 random vectors, and for each vector, the fault-free waveforms were calculated using the method described by Iyengar *et al.* [34]. The minimum and maximum delays for each gate were set using a simple wireload delay model [46, 65], where delay bounds for each gate are set to $(Nom \times n) \pm Tol$. Nom is a nominal delay value, n is the number of fanouts, and Tol is the tolerance to set minimum and maximum delays around the nominal value. The min-max delays for a gate are then set to:

$$maxDelay = (Nom \times n) + Tol$$

$$minDelay = (Nom \times n) - Tol$$

For these results, a nominal delay of 3.5 time units and a tolerance of ±14% was used, which would set the default min-max delays of a gate with a single fanout to approximately 3 and 4. A static timing analysis calculated the longest path delay as the sum of maximum gate delays along the longest delay path from input to output. The sample period T_s was then set to 1 time unit above the longest path delay.

For the data presented in columns two and three of Table 4.5, signal correlations due to reconvergent fanouts were ignored. For each vector, and for each detected gate delay fault, faulty waveforms were evaluated using the method presented by Iyengar *et al.* [34] to evaluate the detection threshold. For the data in columns four and five, ambiguity lists were evaluated during detection threshold evaluation as described in this chapter. In each case, for each detected fault, the smallest detection threshold over all simulated vector pairs was stored. The slack for each gate was calculated during the static timing analysis step as the difference between T_s and the sum of all minimum gate delays along the longest delay path through that gate. Detection gaps were then calculated to relate the detection thresholds to the slacks. The smaller the detection gaps are for detected gate delay faults, the better quality the vector set provides for gate delay testing.

Columns two and four show the average detection gap for all detected gate delay faults to illustrate the pessimism when signal correlations are ignored. When reconvergent fanout analysis is used, the average detection gap is smaller. Columns three and five show the fault coverage of detected gate delay faults with a detection gap that is less than or equal to the nominal gate delay used, which is 3.5. This is to allow faults to be counted as detected if they are either detected through the longest path through the gate, or if they are detected through a path which is less than the longest path by only one gate delay.

When reconvergent fanout analysis is used, more gate delay faults are detected with smaller detection gaps.

Chapter 5

Reconvergent Fanout Analysis in Fault-Free Circuit Simulation

This chapter presents a method that adds reconvergent fanout analysis to the bounded delay simulation used in simulating a fault-free circuit. The reconvergent fanout analysis technique, similar to that presented by Bose and Agrawal [12], is used to more accurately calculate the fault-free waveform timing values EA and LS when reconvergent fanouts are present. The technique presented in this chapter has appeared in a recent paper [27].

5.1 Motivation

Consider the circuit in Figure 5.1, which shows fault-free circuit simulation waveforms for a test to detect a slow-to-rise gate delay fault at gate J. Table 5.1 shows EA, LS, FPV, and the three reference quantity calculations for evaluating the detection threshold of the fault at J. For a sample period of $T_s = 14$, the detection threshold is evaluated as $T_s - RTb(Y) = (14 - 5) = 9$.

At reconvergent gate H, the top input (from D) is guaranteed to transition to a logic 0 at least 1 time unit before the bottom input (from F) transitions away from logic 0. Due to this correlation, there is always a logic 0 on at least one input to NAND gate H, so the hazard between times 3 and 5 at the output of H cannot occur. A correct analysis would set the fault-free timing values as:

$$EA(H) = \infty \qquad LS(H) = -\infty$$

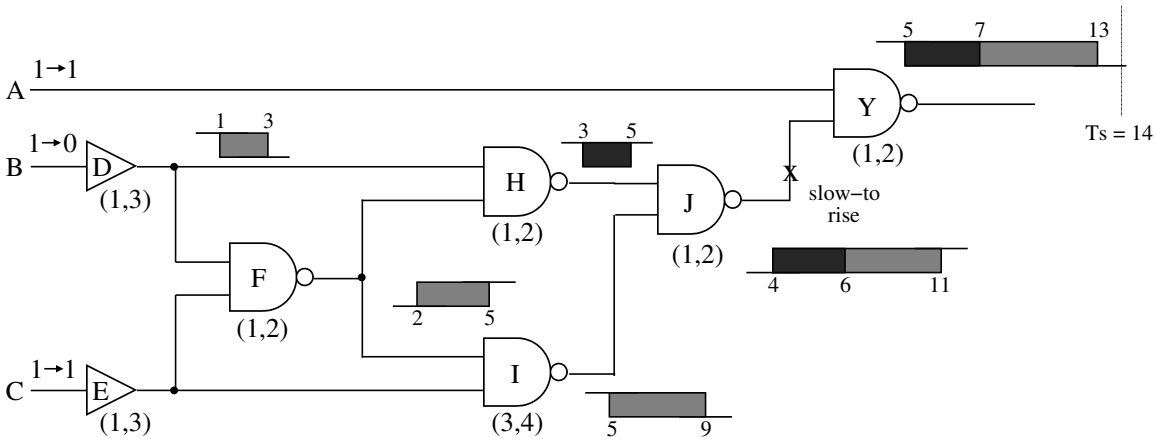

Figure 5.1: Incorrect Detection Threshold.

which would result in the following timing values for output Y:

$$EA(Y) = 7 \qquad LS(Y) = 13$$

Because gate J is the fault site, $RTb(J)$ is set to $EA(J)$, which is 4 time units in Table 5.1. If the incorrect hazard at H were suppressed, $RTb(J) = EA(J) = 6$ instead of 4. This change would propagate to output Y by changing $RTb(Y) = 5$ to the correct value of $RTb(Y) = 7$. By adding reconvergent fanout analysis to fault-free circuit simulation, the detection threshold for the fault at J would be correctly evaluated as $T_s - RTb(Y) = (14 - 7) = 7$ instead of the pessimistic value of 9.

Notice that for this example, propagating ambiguity lists in the downcone of the fault site during faulty circuit calculations, without considering reconvergent fanout analysis in the calculation of $EA(H)$ and $LS(H)$, results in a pessimistic detection threshold evaluation. To accurately evaluate detection thresholds of detected gate delay faults, signal correlations

Table 5.1: Incorrect Detection Threshold Calculation.

signal	EA	LS	FPV	$\rho(s)$	RTa	RTb
A	∞	$-\infty$	1	0	$-\infty$	∞
B	0	0	0	0	0	∞
C	∞	$-\infty$	1	0	$-\infty$	∞
D	1	3	0	0	3	∞
E	∞	$-\infty$	1	0	$-\infty$	∞
F	2	5	1	0	5	∞
H	3	5	1	0	5	∞
I	5	9	0	0	9	∞
J	4	11	0	0	$-\infty$	4
Y	5	13	1	0	$-\infty$	5

due to reconvergent fanouts should be considered during both fault-free circuit simulation and faulty waveform calculations.

5.2 Fault-Free Circuit Simulation

To propagate correlation information during fault-free circuit simulation, we propagate ambiguity lists similar to those in [12, 13]. Each ambiguity list element at a gate G contains a gate ID of a fanout point f effecting G, the minimum delay from f to G, and the maximum delay from f to G:

- Gate IDs of fanout points (f) that lie in the forward cone of the fault site.
- $d(G, f)$: Minimum delay to G from f.
- $D(G, f)$: Maximum delay to G from f.

When $EA(G)$ and $LS(G)$ are evaluated, the ambiguity list at G is evaluated. Ambiguity lists at the inputs to G are used to determine the ambiguity list at the output [27].

If inputs at a reconvergent gate are correlated due to a common fanout point, a ambiguity list element for that fanout point appears at all correlated inputs of G [12, 13, 27]. If fanout point f appears in the ambiguity list of more than one input to G, then f is reconverging at G. If correlated inputs to G are transitioning both to and away from a controlling input logic value for gate G, the following quantities are evaluated:

$$min_{DV}(f) = max\{d(M, f)\}$$

$$max_{SV}(f) = max\{D(m, f)\}$$

where M is an input to G that is transitioning away from a controlling logic value, and m is an input to G that is transitioning to a controlling input logic value. Here, $min_{DV}(f)$ represents the largest delay from a common fanout point f to an input (M) of gate G, where $IV(M)$ is a controlling input logic value for G. The quantity $max_{SV}(f)$ represents the largest delay from f to an input (m) of G where $FV(m)$ is a controlling input value for gate G [27].

If $min_{DV}(f) \geq max_{SV}(f)$, then the correlated inputs to G are such that the transition to a controlling input value for G on input m is guaranteed to occur before M transitions away from a controlling input value for G. In this case, there is always a controlling logic value at some input to G, so hazard suppression occurs in which the ambiguity list at the output of G is set to *null*, and the following fault-free timing values are used:

$$EA(G) = \infty \qquad LS(G) = -\infty$$

Table 5.2: Corrected Detection Threshold Calculation.

signal	EA	LS	FPV	$\rho(s)$	RTa	RTb
A	∞	$-\infty$	1	0	$-\infty$	∞
B	0	0	0	0	0	∞
C	∞	$-\infty$	1	0	$-\infty$	∞
D	1	3	0	0	3	∞
E	∞	$-\infty$	1	0	$-\infty$	∞
F	2	5	1	0	5	∞
H	∞	$-\infty$	1	0	$-\infty$	∞
I	5	9	0	0	9	∞
J	7	11	0	0	$-\infty$	7
Y	8	13	1	0	$-\infty$	8

If no fanout point results in $min_{DV}(f) \geq max_{SV}(f)$, hazard suppression does not occur, and $EA(G)$ and $LS(G)$ are evaluated as in [34]. In this case, an ambiguity list element is added to G's ambiguity list for every fanout element found in the ambiguity lists at the inputs (i) to G, and the minimum and maximum delays of ambiguity list elements are updated using the minimum and maximum delays of G:

$$d(G, f) = min\{d(i, f)\} + minDelay(G)$$

$$D(G, f) = max\{D(i, f)\} + maxDelay(G)$$

Table 5.2 shows the fault-free and faulty waveform calculations for the example of Figure 5.1, when ambiguity lists are propagated during both fault-free and faulty simulations. $RTb(Y)$ is now calculated at 8, which results in a correct detection threshold evaluation of 6.

Table 5.3: Largest Output EA and LS Values for 10,000 Random Vectors.

	Without Reconvergent Fanout Analysis		With Reconvergent Fanout Analysis	
	Largest EA	Largest LS	Largest EA	Largest LS
c3540	195.7	339.2	222.7	335.2
c5315	150.5	339.2	171.6	323.2
c6288	292.0	933.7	424.4	825.9
c7552	165.5	339.2	198.7	339.2

5.3 Results on ISCAS85 Benchmark Circuits

Results on ISCAS85 combinational benchmark circuits for randomly generated input vectors are shown in Table 5.3 and Table 5.4. A simple wireload delay model is used for minimum and maximum gate delays, where delay bounds for each gate are set to $(Nom \times n) \pm Tol$. Nom is a nominal delay value, n is the number of fanouts, and Tol is the tolerance to set minimum and maximum delays around the nominal value. The min-max delays for a gate are then set to:

$$maxDelay = (Nom \times n) + Tol$$

$$minDelay = (Nom \times n) - Tol$$

For these results, a nominal delay of 3.5 time units and a tolerance of ±14% was used, which would set the default min-max delays of a gate with a single fanout to approximately 3 and 4. A static timing analysis step calculated the longest path delay and the slacks for each gate. The sample period T_s was chosen to be 1 time unit more than the longest path delay.

Table 5.4: Detection Gap Results for 1,000 Random Vectors.

	Without Reconvergent Fanout Analysis		With Reconvergent Fanout Analysis	
	Average Detection Gap	Faults Detected with Gap ≤ 3.5	Average Detection Gap	Faults Detected with Gap ≤ 3.5
c432	72.4	8.80%	66.8	11.00%
c499	31.1	11.44%	25.4	20.54%
c880	21.3	40.18%	20.3	40.18%
c1355	33.4	10.90%	27.8	19.42%
c1908	48.1	15.94%	45.6	16.37%
c2670	34.2	29.70%	31.6	31.31%
c3540	43.9	20.45%	35.3	21.90%
c5315	28.3	36.48%	25.2	37.73%
c6288	395.2	0.45%	352.5	0.57%
c7552	36.3	11.95%	30.3	13.05%

Results on fault-free timing analysis (EA and LS) for a few larger combinational benchmark circuits for a set of 10,000 random vectors are shown in Table 5.3. These results show the difference seen at circuit outputs when reconvergent fanout analysis is used. The second and third columns show the largest EA and largest LS value at a circuit's output for all vectors without using reconvergent fanout analysis. Columns four and five show the same results when signal correlations are used, and hazards that cannot occur are suppressed. The last two columns show the difference in these calculations between those with reconvergent fanout analysis and those without reconvergent fanout analysis. The result of using information about correlated signals at reconvergent gates during simulation generally results in larger EA values and smaller LS values at outputs, and is more apparent for circuits that contain a large number of reconvergent fanout such as in the array multiplier circuit c6288.

Table 5.4 shows the results when reconvergent fanout analysis is used during both fault-free timing analysis and detection threshold evaluation. Using reconvergent fanout analysis on all gates of the fault-free circuit in addition to gates inside the cone of influence during fault simulation is the primary difference between the method presented in this chapter and the method presented in Chapter 4. The detection gap is used to display the data in Table 5.4. The smaller the detection gaps are for detected gate delay faults, the better quality the vector set provides for gate delay testing. Columns two and three of Table 5.4 show the detection gap results without using reconvergent fanout analysis. Column two shows the average detection gap for all detected gate delay faults, and column three shows the fault coverage of detected gate delay faults with a detection gap that is less than or equal to the nominal gate delay used, which is 3.5. This is to allow faults to be counted as detected if they are either detected through the longest path through the gate, or if they are detected through a path which is less than the longest path by only one gate delay. Columns four and five of Table 5.4 show detection gap results when reconvergent fanout analysis was used during both fault-free timing analysis and detection threshold calculation. Column four shows the average detection threshold for all detected faults and column five shows the fault coverage of detected faults with a detection gap less than or equal to 3.5. The data shown in Table 5.4 illustrates the pessimism when signal correlations are ignored in gate delay fault simulation. When reconvergent fanout analysis is used, the average detection gap is smaller and more faults are detected with smaller detection gaps.

Chapter 6

Conclusion

We have presented a method for reconvergent fanout analysis in gate delay fault simulation in Chapter 4. This method considers signal correlations due to reconvergent fanouts when evaluating the detection thresholds of detected gate delay faults. The use of signal correlations results in less pessimistic detection threshold calculations. In Chapter 5, we presented a method for reconvergent fanout analysis in bounded delay simulation that considers signal correlations due to reconvergent fanouts when evaluating the fault-free waveform of a reconvergent gate. This method results in a more accurate fault-free circuit simulation when bounded gate delays are used to model imprecise circuit delays.

Since the quantities propagated inside the cone of influence of the fault site during detection threshold evaluation are initialized at the fault site and outside the downcone of the fault site using the waveforms from the fault-free circuit, reconvergent fanout analysis should be used during both fault-free circuit simulation and detection threshold evaluation for accurate gate delay fault simulation in the presence of reconvergent fanouts.

6.1 Future Work

During simulation, the ambiguity lists that are propagated to provide information about signal correlations can grow quite large. The efficiency of the described reconvergent fanout analysis technique can be affected significantly since these lists are evaluated at every gate in the circuit during bounded delay simulation of the fault-free circuit, and for all gates

inside the cone of influence of every fault simulated. The efficiency of the list propagation algorithms described in Chapter 4 and Chapter 5 can be improved.

6.2 Other Applications

The bounded delay simulation algorithms developed in this research has found two other applications. The first of these is in identifying hazard-free tests. Such tests consist of pairs of vectors, which when applied to combinational logic produce no hazards (transients with multiple transitions) at primary outputs. These are used for timing calibration of scan-based circuits [57]. For a vector-pair to be effective, its hazard-free behavior should be preserved in spite of the manufacturing variations in delays. The use of min-max delay model and the ambiguity list simulation algorithms have allowed the identification of many more hazard-free tests than was possible with pessimistic zero-delay simulators [45].

The second application is in a tool that determines the dynamic power consumption of a digital CMOS circuit. Dynamic power of a CMOS logic circuit is directly related to the signal transitions. However, the actual number of transitions is strongly influenced the the delay-dependent transient behavior of signals. A relevant problem is that of estimating the minimum, maximum and average values of dynamic power consumed when gate delays are affected by process variations. The use of the bounded delay model and the ambiguity delay simulation has allowed the development of new power analysis algorithms [4]. These algorithms require much less computation compared to the conventional Monte Carlo approach.

6.3 An Important Observation

In this work, we found that adding reconvergent fanout analysis to gate delay fault simulation always reduced the detection threshold. This may be due to a possible fact that considering signal correlations from reconverging fanouts cannot cause ambiguity intervals to increase. The reconvergent fanout analysis technique described in this thesis removes ambiguity at the output of a gate when correlations between the gate's inputs guarantee the output to be stable, so ambiguity should only decrease when reconvergent fanout analysis is used. Although we have not proven that the detection threshold cannot increase when signal correlations are considered, we were not able to construct a contradicting example in which the use of reconvergent fanout analysis would result in larger detection thresholds. We currently view this result as a conjecture and hope that it will draw the attention of future researchers.

Bibliography

[1] A. Agarwal, D. Blaauw, and V. Zolotov, "Statistical Timing Analysis for Intra-Die Process Variations with Spatial Correlations," in *Proc. International Conference on Computer Aided Design*, Nov. 2003, pp. 900–907.

[2] P. Agrawal, W. J. Dally, W. C. Fischer, H. V. Jagadish, A. S. Krishnakumar, and R. Tutundjian, "MARS: A Multiprocessor Based Programmable Accelerator," *IEEE Design & Test of Computers*, vol. 4, no. 5, pp. 1–10, Oct. 1987.

[3] V. D. Agrawal, "Synchronous Path Analysis in MOS Circuit Simulator," in *Proc. 19th Design Automation Conf.*, June 1982, pp. 629–635.

[4] J. D. Alexander and V. D. Agrawal, "Dynamic Power Estimation with Bounded Delay Model of Process Variation." Unpublished Manuscript, May 2008.

[5] S. Bhardwaj, S. Vrudhula, and D. Blaauw, "Probability Distribution of Signal Arrival Times Using Bayesian Networks," *IEEE Trans. Computer Aided Design*, vol. 24, no. 11, pp. 1784–1794, Nov. 2005.

[6] H. Bhatnagar, *Advanced ASIC Chip Sysnthesis using Synopsys Design Compiler and Primetime*. Boston: Springer, 1999.

[7] J. W. Bierbauer, J. A. Eiseman, F. A. Fazal, and J. J. Kulikowski, "System Simulation with MIDAS," *AT&T Technical Journal*, vol. 70, no. 1, pp. 36–51, Jan. 1991.

[8] S. Bose, P. Agrawal, and V. D. Agrawal, "A Path Delay Fault Simulator for Sequential Circuits," in *Proc. 6th International Conference on VLSI Design*, Jan. 1993, pp. 269–274.

[9] S. Bose, P. Agrawal, and V. D. Agrawal, "Delay Fault Testability Evaluation Through Timing Simulation," in *Proc. Third IEEE Great Lakes Symposium on VLSI*, Mar. 1993, pp. 18–21.

[10] S. Bose, P. Agrawal, and V. D. Agrawal, "A Rated-Clock Test Method for Path Delay Faults," *IEEE Trans. VLSI Systems*, vol. 6, no. 2, pp. 323–331, June 1998.

[11] S. Bose, P. Agrawal, and V. D. Agrawal, "Deriving Logic Systems for Path Delay Test Generation," *IEEE Trans. Computers*, vol. 47, no. 8, pp. 829–846, Aug. 1998.

[12] S. Bose and V. D. Agrawal, "Delay Test Quality Evaluation using Bounded Gate Delays," in *Proc. 25th IEEE VLSI Test Symposium*, May 2007, pp. 23–28.

[13] S. Bose, H. Grimes, and V. D. Agrawal, "Delay Fault Simulation with Bounded Gate Delay Model," in *Proc. IEEE International Test Conference*, Sept. 2007. Paper 26.3.

[14] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Boston: Springer, 2000.

[15] J. L. Carter, V. S. Iyengar, and B. K. Rosen, "Efficient Test Coverage Determination for Delay Faults," in *Proc. IEEE International Test Conference*, Sept. 1987, pp. 418–427.

[16] S. Chakraborty, D. L. Dill, and K. Y. Yun, "Min-Max Timing Analysis and an Application to Asynchronous Circuits," *Proc. IEEE*, vol. 87, no. 2, pp. 332–346, Feb. 1999.

[17] T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell, "Path Delay Fault Simulation of Sequential Circuits," *IEEE Trans. VLSI Systems*, vol. 8, pp. 223–228, Apr. 2000.

[18] K. T. Cheng, "Transition Fault Testing for Sequential Circuits," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 12, no. 12, pp. 1971–1983, Dec. 1993.

[19] S. Devadas and K. Kuetzer, "Validatable Nonrobust Delay Fault Testable Circuits Via Logic Synthesis," *IEEE Trans. Computer Aided Design*, vol. 11, no. 12, pp. 1559–1573, Dec. 1992.

[20] A. Devgan and C. Kashyap, "Block-based Static Timing Analysis with Uncertainty," in *Proc. International Conference on Computer Aided Design*, Nov. 2003, pp. 607–614.

[21] D. Dumas, P. Girard, C. Landrault, and S. Pravossoudovitch, "An Implicit Delay Fault Simulation Method with Approximate Detection Threshold Calculation," in *Proc. International Test Conference*, Oct. 1993, pp. 705–713.

[22] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel, "Chip Layout Optimization using Critical Path Weighting," in *Proc. 21st Design Automation Conf.*, June 1984, pp. 133–136.

[23] F. Fink, K. Fuchs, and M. H. Schulz, "Dynamite: An Efficient Automatic Test Pattern Generation System for Path Delay Faults," *IEEE Trans. Computer Aided Design*, vol. 10, no. 10, pp. 1323–1335, Oct. 1991.

[24] A. Gattiker, S. Nassif, R. Dinakar, and C. Long, "Timing Yield Estimation from Static Timing Analysis," in *Proc. International Symposium on Quality Electronic Design*, Mar. 2001, pp. 437–442.

[25] M. A. Gharaybeh, M. L. Bushnell, and V. D. Agrawal, "Classification and Test Generation for Path-Delay Faults Using Single Stuck-at Fault Tests," *J. Electronic Testing: Theory and Applications*, vol. 11, no. 1, pp. 55–67, Aug. 1997.

[26] M. A. Gharaybeh, M. L. Bushnell, and V. D. Agrawal, "The Path-Status Graph with Application to Delay Fault Simulation," *IEEE Trans. CAD*, vol. 17, no. 4, pp. 324–332, Apr. 1998.

[27] H. Grimes and V. D. Agrawal, "Analyzing Reconvergent Fanouts in Gate Delay Fault Simulation," in *Proc. 17th IEEE North Atlantic Test Workshop*, May 2008, pp. 98–103.

[28] K. Heragu, J. H. Patel, and V. D. Agrawal, "Segment Delay Faults: A New Fault Model," in *Proc. 14th IEEE VLSI Test Symp.*, Apr. 1996, pp. 32–39.

[29] R. B. Hitchcock, "Timing Verification and the Timing Analysis Program," in *Proc. 19th Design Automation Conf.*, June 1982, pp. 594–604.

[30] N. Ishiura, Y. Deguchi, and S. Yajima, "Coded Time-Symbolic Simulation Using Shared Binary Decision Diagrams," in *Proc. Design Automation Conference*, June 1990, pp. 130–135.

[31] N. Ishiura, M. Takahashi, and S. Yajima, "Time-Symbolic Simulation for Accurate Timing Verification," in *Proc. Design Automation Conference*, June 1989, pp. 497–502.

[32] V. S. Iyengar, B. K. Rosen, and I. Spillinger, "Delay Test Generation 1 - Concepts and Coverage Metrics," in *Proc. IEEE International Test Conference*, Sept. 1988, pp. 857–866.

[33] V. S. Iyengar, B. K. Rosen, and I. Spillinger, "Delay Test Generation 2 - Algebra and Algorithms," in *Proc. IEEE International Test Conference*, Sept. 1988, pp. 867–876.

[34] V. S. Iyengar, B. K. Rosen, and J. A. Waicukauski, "On Computing the Sizes of Detected Delay Faults," *IEEE Trans. Computer Aided Design*, vol. 9, no. 3, pp. 299–312, Mar. 1990.

[35] H. F. Jyu, S. Malik, S. Devadas, and K. W. Kuetzer, "Statistical Timing Analysis of Combinational Logic Circuits," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 1, no. 2, pp. 126–137, June 1993.

[36] H. Konuk, "On Invalidation Mechanisms for Non-Robust Delay Tests," in *Proc. International Test Conference*, Oct. 2000, pp. 393–399.

[37] A. Krstić and K. T. Cheng, *Delay Fault Testing for VLSI Circuits*. Boston: Springer, 1998.

[38] B. Kruseman, A. K. Majhi, G. Gronthoud, and S. Eichenberger, "On Hazard-Free Patterns for Fine-delay Fault Testing," in *Proc. IEEE International Test Conference*, Sept. 2004, pp. 213–222.

[39] K. N. Lalgudi, D. Bhattacharya, and P. Agrawal, "Architecture of a Min-Max Simulator on MARS," in *Proc. International Conference on VLSI Design*, Jan. 1993, pp. 246–249.

[40] C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits," *IEEE Trans. Computer Aided Design*, vol. 6, no. 5, pp. 694–703, Sept. 1987.

[41] M. Linderman and M. Leeser, "Simulation of Digital Circuits in the Presence of Uncertainty," in *Proc. International Conference on Computer Aided Design*, Nov. 1994, pp. 248–251.

[42] J. J. Liou, A. Krstić, L. C. Wang, and K. T. Cheng, "False Path Aware Statistical Timing Analysis and Efficient Path Selection for Delay Testing and Timing Validation," in *Proc. Design Automation Conference*, June 2002, pp. 566–569.

[43] N. Maheshwari and S. S. Sapatnekar, *Timing Analysis and Optimization of Sequential Circuits*. Boston: Springer, 1999.

[44] A. K. Majhi, V. D. Agrawal, J. Jacob, and L. M. Patnaik, "Line Coverage of Path Delay Faults," *IEEE Trans. VLSI Systems*, vol. 8, pp. 610–614, Oct. 2000.

[45] S. N. Menon, "Personal Communication." 2008.

[46] W. Nebel and J. Mermet, *Low Power Design in Deep Submicron Electronics*. Boston: Springer, 1997.

[47] S. Padmanabhan and S. Tragoudas, "An Implicit Path Delay Fault Diagnosis Methodology," *IEEE Trans. Computer Aided Design*, vol. 22, no. 10, pp. 1399–1408, Oct. 2003.

[48] E. S. Park and M. R. Mercer, "An Efficient Delay Test Generation System for Combinational Logic Circuits," *IEEE Trans. Computer Aided Design*, vol. 11, no. 7, pp. 926–938, July 1992.

[49] E. S. Park, M. R. Mercer, and T. W. Williams, "A Statistical Model for Delay-Fault Testing," *IEEE Design & Test of Computers*, vol. 6, no. 1, pp. 45–55, Feb. 1989.

[50] Q. Peng, V. D. Agrawal, and J. Savir, "On the Guaranteed Failing and Working Frequencies in Path Delay Fault Analysis," in *Proc. 16th IEEE Instrumentation and Mesurement Technology*, volume 3, Mar. 1999, pp. 1794–1799.

[51] A. K. Pramanick and S. M. Reddy, "On the Detection of Delay Faults," in *Proc. IEEE International Test Conference*, Sept. 1988, pp. 845–856.

[52] A. K. Pramanick and S. M. Reddy, "On the Fault Coverage of Gate Delay Fault Detecting Tests," *IEEE Trans. Computer Aided Design*, vol. 16, no. 1, pp. 78–94, Jan. 1997.

[53] W. Qiu and D. M. H. Walker, "Testing the Path Delay Faults of ISCAS85 Circuit c6288," in *Proc. International Workshop Microprocessor Test Verification*, May 2003, pp. 19–24.

[54] S. M. Reddy, C. J. Lin, and S. Patil, "An Automatic Test Pattern Generator for Path Delay Faults," in *Proc. International Conference on Computer Aided Design*, Nov. 1987, pp. 284–287.

[55] S. Roy, P. P. Chakrabarti, and P. Dasgupta, "Bounded Delay Timing Analysis using Boolean Satisfiability," in *Proc. 20th International Conference on VLSI Design*, Jan. 2007, pp. 295–302.

[56] M. H. Schulz and F. Brglez, "Accelerated Transition Fault Simulation," in *Proc. 26th Design Automation Conference*, June 1987, pp. 237–243.

[57] A. D. Singh and G. Xu, "Output Hazard-Free Transition Tests for Silicon Calibrated Scan Based Delay Testing," in *Proc. 24th IEEE VLSI Test Symp.*, Apr. 2006, pp. 349–357.

[58] G. L. Smith, "Model for Delay Faults Based Upon Paths," in *Proc. International Test Conference*, Oct. 1985, pp. 342–349.

[59] M. K. Srinivas, M. L. Bushnell, and V. D. Agrawal, "Flags and Algebra for Sequential Circuit VNR Path Delay Fault Test Generation," in *Proc. 10th International Conf. on VLSI Design*, Jan. 1997, pp. 88–94.

[60] T. G. Szymanski, "LEADOUT: A Static Timing Analyzer for MOS Circuits," in *Proc. International Conf. Computer-Aided Design*, Nov. 1986, pp. 130–133.

[61] E. G. Ulrich, K. P. Lentz, S. Demba, and R. Razdan, "Concurrent Min-Max Simulation," in *Proc. Design Automation Conference*, June 1991, pp. 554–557.

[62] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan, "First Order Incremental Block Based Statistical Timing Analysis," in *Proc. Design Automation Conference*, June 2004, pp. 331–336.

[63] J. A. Waicukauski, E. Lindbloom, B. Rosen, and V. Iyengar, "Transition Fault Simulation," *IEEE Design & Test of Computers*, vol. 4, no. 2, pp. 32–38, Apr. 1987.

[64] K. Yang, L. C. Wang, K. T. Cheng, and S. Kundu, "On Statistical Correlation Based Path Selection for Timing Validation," in *Proc. IEEE International Symp. VLSI Design, Automation and Test*, Apr. 2005, pp. 8–11.

[65] G. K. Yeap, *Practical Low Power Digital VLSI Design.* Boston: Springer, 1998.

[66] L. Zhang, W. Chen, Y. Hu, and C. C. Chen, "Statistical Static Timing Analysis With Conditional Linear MAX/MIN Approximation and Extended Canonical Timing Model," *IEEE Trans. Computer Aided Design*, vol. 25, no. 6, pp. 1182–1191, June 2006.

Appendices

Appendix A

Gate Delay Fault Simulation Program - User Information

User information for the gate delay fault simulation program used for the results presented in Chapter 4 and Chapter 5 is described in this appendix. The program is implemented in the C programming language to run in a UNIX/LINUX environment. The GNU C compiler GCC 4.2 (*http://gcc.gnu.org/*) was used in this work to compile the source code (simulator.c) into an executable named "simulate". The simulator is then invoked with the following command:

simulate *cktName*.bench [options]

The circuit netlist is specified in the *cktName*.bench file in a hierarchical bench format, which is discussed in the next section. The simulator flattens the netlist, stores the circuit in memory, and simulates the circuit for a set of input vectors. Simulation is setup by specifying the various simulator options discussed in Section A.2. Section A.3 describes an example simulation.

A.1 Hierarchical Bench Format

The circuit description is specified in a hierarchical bench format, where each hierarchical block, *blockName*, is separated into its own section with a beginning "BLOCK *blockName*" statement and an ending "END" statement. The description of each block is in the regular bench format used to describe the ISCAS85 combinational benchmark circuits, where the following Boolean gate types are supported: NAND, AND, OR, NOR, NOT, and BUFF.

To simulate ISCAS85 benchmark circuits, the circuit's bench netlist has to be modified by adding "BLOCK *circuitName*" to the beginning of the netlist and "END" to the end of the netlist. For example, circuit c17 would be modified by adding the first and last bold face lines shown in Figure A.1.

Since the simulator does not support XOR gates, circuits that contain XORs (such as c432 and c499) must have an XOR block added to their hierarchy. For example, in the half adder description in Figure A.2, the NAND configuration for an XOR gate (defined in the XOR block) would be inserted for the "Sum" output. For the statement "Sum=XOR(in1,in2)", Sum corresponds to the XOR block's Y output, and $in1$ and $in2$ correspond to the XOR block's A and B inputs.

Min-max gate delays may also be specified for standard gates by adding these values to that line in the netlist after a colon. For example, the line "Cout = AND(in1,in2) : 2 3" would set the minimum delay to 2 and the maximum delay to 3 for the AND gate of the "Cout" output in the half adder circuit description of Figure A.2.

```
BLOCK c17
 INPUT(1)
 INPUT(2)
 INPUT(3)
 INPUT(6)
 INPUT(7)
 OUTPUT(22)
 OUTPUT(23)
 10 = NAND(1, 3)
 11 = NAND(3, 6)
 16 = NAND(2, 11)
 19 = NAND(11, 7)
 22 = NAND(10, 16)
 23 = NAND(16, 19)
END
```

Figure A.1: Hierarchical Benchmark Description of c17.

```
BLOCK XOR
 INPUT(A)
 INPUT(B)
 OUTPUT(Y)
 X1 = NAND(A,B)
 X2 = NAND(X1,A)
 X3 = NAND(X1,B)
 Y = NAND(X2,X3)
END

BLOCK HalfAdder
 INPUT(in1)
 INPUT(in2)
 OUTPUT(Sum)
 OUTPUT(Cout)
 Sum = XOR(in1,in2)
 Cout = AND(in1,in2)
END
```

Figure A.2: Hierarchical Benchmark Description for a Half Adder.

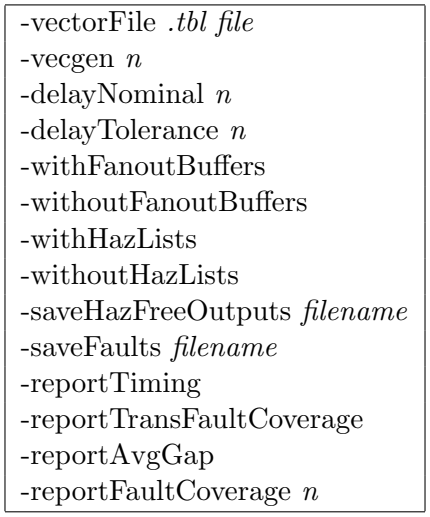

Figure A.3: Simulator Options.

A.2 Simulator Options

An option list is shown in Figure A.3, which determines the type of simulation performed, the input vectors to simulate, how gate delays are set, and how to report simulation results.

The "-vectorFile $filename$" option specifies the input vector file containing the set of input vectors to be simulated. The vector file contains a list of the circuit's primary input and primary output names, followed by a list of vectors, where each vector is specified as a single line. The input and output values for each vector are specified in the same order as they were listed at the top of the vector file. If a vector file is not specified using this option, the vector file defaults to *circuitName*.tbl for a netlist specified in *circuitName*.bench.

A set of n random vectors can be generated for the circuit if the "-vecgen n" option is specified. Logic simulation determines the circuit's output response for each generated vector, and the simulator creates the vector file specified with the "-vectorFile $filename$" option. With this option specified, neither bounded delay simulation nor gate delay fault simulation are performed.

Gate delays that are not specified in the netlist file are set using a simple wireload delay model, where delay bounds for each gate are set to $(Nom \times n) \pm Tol$. Nom is a nominal delay value, n is the number of fanouts, and Tol is the tolerance to set minimum and maximum delays around the nominal value. The min-max delays are then set to:

$$maxdel(G) = (Nom \times n) + Tol$$
$$mindel(G) = (Nom \times n) - Tol$$

The "-delayNominal n" and "-delayTolerance n" options set the Nom and Tol values. If these options are not specified, the wireload delay values default to $Nom = 3.5$ and $Tol = 14\%$.

The "-withHazLists" option tells the simulator to use reconvergent fanout analysis and propagate ambiguity lists during both fault-free circuit simulation and gate delay fault simulation as described in Chapter 5. If the "-withoutHazLists" option is specified, ambiguity lists will not be used at all during simulation. Ambiguity lists are not used by default, so to use reconvergent fanout analysis during simulation, the "-withHazLists" option must be specified.

If the "-saveHazFreeOutputs $filename$" option is specified, each input vector pair is stored in $filename$ along with an indication of which outputs were found to be hazard free for simulation of that vector pair. An example vector pair written to this file for a circuit with 5 primary inputs and 2 primary outputs would be:

```
01101
10100 01
```

These lines indicate that for the vector pair $V_1 = 01101$ and $V_2 = 10100$, the circuit's first primary output listed in the input vector file is not hazard free ('0'), and the second primary output listed is hazard free ('1').

The "-saveFaults $filename$" option tells the simulator to save fault detection information to a file specified by $filename$. The total number of faults and the number of detected faults are listed first, then detected and undetected gate delay faults are listed. Each fault in the list of detected faults specifies the gate name at the fault site, the fault type (slow-to-rise or slow-to-fall), and the best detection threshold and detection gap for detection of that fault for the simulated input vectors. The list of undetected faults gives the gate name and fault type (slow-to-rise or slow-to-fall) for all faults that are not detected by the vector set.

If the "-reportTiming" option is specified, the simulator reports the sample period T_s, the largest EA seen at any primary output, and the largest LS seen at any primary output for the vector set. The "-reportTransFaultCoverage" option reports the fault coverage of gate delay faults for the vector set if fault sizes are ignored. The "-reportFaultCoverage n" option prints the fault coverage of gate delay faults that are detected with a detection gap $\leq n$. If the "-reportAvgGap" option is specified, the simulator reports the average detection gap of gate delay faults that are detected by the vector set.

A.3 Example Simulation

This section describes an example simulation for the circuit shown in Figure A.4 using input vectors specified by the vector file shown in Figure A.6. The input vectors were generated randomly using the command:

```
simulate example.bench -vecgen 10
```

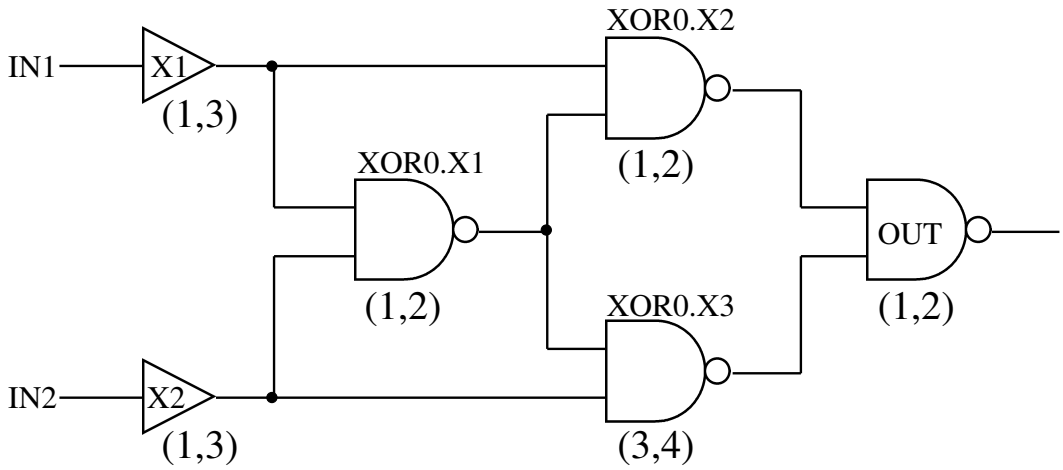

Figure A.4: Example Circuit.

where the circuit netlist is described in the file "example.bench" in the hierarchical bench format. The following command performs gate delay fault simulation on the example circuit using 10 randomly generated vectors:

```
simulate example.bench -withHazLists -withoutFanoutBuffers -reportTiming
        -reportTransFaultCoverage -reportAvgGap -reportFaultCoverage 1
        -saveHazFreeOutputs example.hazFree -saveFaults example.faults
```

The simulator first read in the netlist and flattened the hierarchy by inserting the XOR block for the *OUT* output in the example circuit. Gate names in the flattened netlist are those shown in Figure A.4. Since gate delays are not specified in the netlist (Figure A.5) and the "-delayNominal" and "-delayTolerance" options were not specified in the command above, the delay bounds for a gate with n fanouts default to $3.5n \pm 14\%$. The "-useHazLists" option is specified to perform reconvergent fanout analysis during both bounded gate delay simulation of the fault-free circuit and gate delay fault simulation of the vector set. The simulator does not insert zero delay fanout buffers because of the "-withoutFanoutBuffers" option. Since the "-vectorFile" option is not specified, the input vector file defaults to "example.tbl".

The options "-reportTiming", "-reportTransFaultCoverage", "-reportAvgGap" and "-reportFaultCoverage 1" produced the timing report, and the fault coverage and detection gap reports shown in Figure A.7. The timing information reported shows the largest EA value seen at the primary output for simulation of all ten vectors is 5, the largest LS value seen at the primary output for simulation of the ten vectors is 11, and the sample period used by the simulator was set to 12. The fault coverage and detection gap reports show

```
BLOCK XOR
 INPUT(A)
 INPUT(B)
 OUTPUT(Y)
 X1 = NAND(A, B) : 1 2
 X2 = NAND(X1, A) : 1 2
 X3 = NAND(X1, B) : 3 4
 Y = NAND(X2, X3) : 1 2
END

BLOCK EXAMPLE-CKT
 INPUT(IN1)
 INPUT(IN2)
 OUTPUT(OUT)
 X1 = BUFF(IN1) : 1 3
 X2 = BUFF(IN2) : 1 3
 OUT = XOR(X1, X2)
END
```

Figure A.5: Hierarchical Bench Netlist for Simulation Example.

```
PI A
PI B
PO Y

101
000
011
011
101
000
110
000
011
011
```

Figure A.6: Input Vector File for Simulation Example.

```
Output Timing Information:
largest EA: 5.0
largest LS: 11.0
sampleTime: 12.0

Number Transition Faults Detected = 14
Total Number Transition Faults = 16
Transition Fault Coverage: 87.50%

Avg. detection Gap for all detected Fautls: 1.9

Number of faults detected with less than 1.0 detection gap: 0
```

Figure A.7: Fault Coverage and Detection Gap Reports.

```
10
00 0
00
01 0
01
10 1
10
00 0
00
11 0
11
00 1
00
01 0
```

Figure A.8: Hazard Free Outputs for Simulation Example.

87.5% transition fault coverage, no faults detected with a detection gap less than 1.0, and an average detection gap of 1.9.

The "-saveHazFreeOutputs example.hazFree" option produced the output file shown in Figure A.8. The input vector pairs simulated are listed in the first column. The second column indicates which outputs were hazard free for the simulation of that vector pair. A '1' for an output in the second column indicates a hazard free output and a '0' indicates that output was not hazard free for the vector pair.

The "-saveFaults example.faults" option produced a file containing fault information, which is shown below. This file contains two lists of faults, one list for gate delay faults that are detected by the vector set, listed along with their detection threshold and detection gap. The second list shows which faults are undetected by the vector set. The slow-to-rise gate delay faults on gates $XOR0.X1$ and $XOR0.X3$ are not detected by the simulated input vectors.

Fault Information File for Example Simulation:

```
Number Faults Detected = 14
Total Number Faults = 16
Transition Fault Coverage: 87.50%

******* List of Detected Faults *******
GATE: IN1: slow-to-rise fault
Detection Threshold: 9.0
Detection Gap: 3.0
GATE: IN1: slow-to-fall fault
Detection Threshold: 9.0
Detection Gap: 3.0
GATE: IN2: slow-to-rise fault
Detection Threshold: 7.0
Detection Gap: 1.0
GATE: IN2: slow-to-fall fault
Detection Threshold: 8.0
Detection Gap: 2.0
GATE: X1: slow-to-rise fault
Detection Threshold: 9.0
Detection Gap: 3.0
GATE: X1: slow-to-fall fault
Detection Threshold: 9.0
Detection Gap: 3.0
GATE: X2: slow-to-rise fault
Detection Threshold: 7.0
```

```
Detection Gap: 1.0
GATE: X2: slow-to-fall fault
Detection Threshold: 8.0
Detection Gap: 2.0
GATE: XOR0.X1: slow-to-fall fault
Detection Threshold: 8.0
Detection Gap: 2.0
GATE: XOR0.X2: slow-to-rise fault
Detection Threshold: 9.0
Detection Gap: 1.0
GATE: XOR0.X2: slow-to-fall fault
Detection Threshold: 9.0
Detection Gap: 1.0
GATE: XOR0.X3: slow-to-fall fault
Detection Threshold: 7.0
Detection Gap: 1.0
GATE: OUT: slow-to-rise fault
Detection Threshold: 7.0
Detection Gap: 1.0
GATE: OUT: slow-to-fall fault
Detection Threshold: 9.0
Detection Gap: 3.0

******* List of Undetected Faults *******
GATE: XOR0.X1: slow-to-rise fault not detected
GATE: XOR0.X3: slow-to-rise fault not detected
```

Appendix B

Gate Delay Fault Simulation Program - Implementation

This appendix describes the implementation of the gate delay fault simulation program used for the results presented in Chapter 4 and Chapter 5. The program is implemented in the C programming language to run in a UNIX/LINUX environment. Execution begins by reading the circuit netlist description in a hierarchical bench format, flattening the hierarchy, and storing the circuit into memory using the various data structures. Simulation of the circuit involves the following three main steps:

- Static Timing Analysis
- Bounded Delay Simulation
- Detection Threshold Evaluation

The static timing analysis step is performed to analyze circuit delays in a vector independent manner. For each input vector pair, the bounded delay simulation step is performed to determine information about signal waveforms at each gate in the fault-free circuit. The detection threshold evaluation step propagates fault effects through the circuit for each gate delay fault that is activated by the vector pair, determines if the activated fault is detected. If the fault is detected, the detection threshold is evaluated. A slow-to-rise (slow-to-fall) gate delay fault is activated if the output of the gate at the fault site has a rising (falling) transition for that vector pair. Both slow-to-rise and slow-to-fall delay faults are considered at each fault site. Fault sites for gate delay faults in a combinational circuit are at all gate outputs and at all fanout branches [34]. The simulator can insert zero delay buffers on all fanout branches in the circuit to model all fault sites as faults on gate outputs.

B.1 Data Structures

Two data structures are used to store the circuit in memory, the Circuit and Gate data structures shown in Figure B.1. Each gate is stored with a unique ID, a $Type$, a minimum ($mindel$) and maximum delay ($maxdel$), an initial value (IV), a final value (FV), an earliest arrival time (EA), a latest stabilization time (LS), and two arrays, $fanin$ and $fanout$. The gate's $Type$ is set to indicate one of the following Boolean gate types: NAND, AND, OR, NOR, NOT, and BUFF. The output waveform for the gate in the fault-free circuit is represented by it's IV, FV, EA, and LS values. To store information about how gates are connected in the circuit, the $fanin$ array for each gate contains a pointer to each fanin gate and the $fanout$ array for each gate contains a pointer to each fanout gate.

The Circuit data structure contains three arrays, PI, PO, and $gateArray$. The $gateArray$ contains an element for each gate in the circuit. The gate's ID value is it's

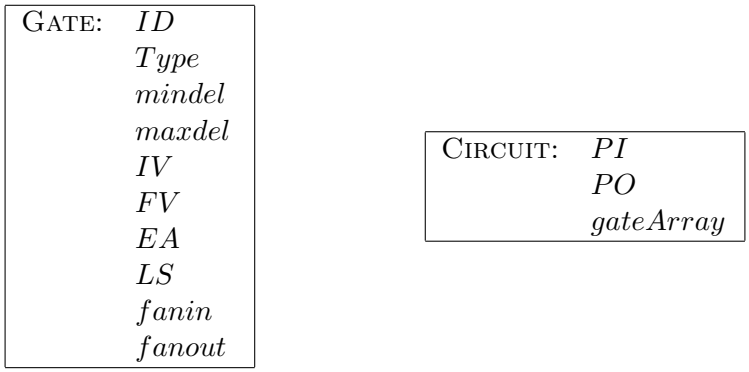

Figure B.1: CIRCUIT and GATE Data Structures.

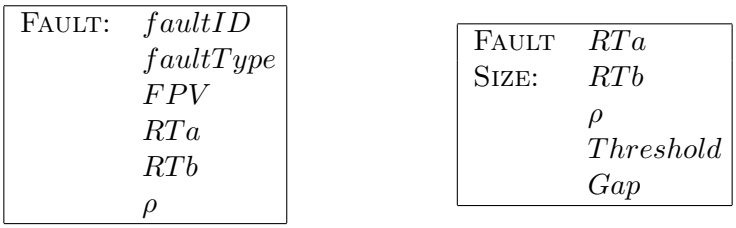

Figure B.2: FAULT and FAULT SIZE Data Structures.

index in *gateArray*, which is a unique identifier for each gate in the circuit. The PI and PO arrays contain pointers to the primary input and primary output gates that are stored in *gateArray*. The gates in *gateArray* are stored in increasing level order. A level-order simulation simply loops through the circuit's *gateArray* to visit all gates in increasing level order, so that when a gate is simulated, all of the gate's fanins have already been processed.

Two data structures are used to store fault information, the FAULT and FAULT SIZE data structures shown in Figure B.2. During fault simulation of each activated fault, the simulator maintains a list of FAULT data structures for each gate in the circuit. Each element of a gate's FAULT list contains a $faultID$, $faultType$, FPV, RTa, RTb, and ρ. The $faultID$ is equal to the ID of the gate at the fault site, and the $faultType$ indicates whether a slow-to-rise or a slow-to-fall delay fault is being simulated. FPV, RTa, RTb, and ρ store information about the faulty waveform at the gate. FPV stores the fault propagating value and RTa, RTb, and ρ provide timing information about faulty waveforms as described by Iyengar *et al.* [34] and discussed in Chapter 3.

To store fault detection information globally for all simulated vectors, the simulator maintains two FAULT SIZE data structures for each gate in the circuit, one for the slow-to-rise fault, and one for the slow-to-fall fault at each gate. The FAULT SIZE element for a detected gate delay fault stores the best RTa, RTb, ρ, detection threshold ($Threshold$),

AMBIGUITY LIST:	$fanoutID$
	$d(G, f)$
	$D(G, f)$

Figure B.3: AMBIGUITY LIST Data Structure.

and detection gap (Gap) information for all vector pairs that have been simulated. If the fault is not detected by the simulated vectors, that fault's FAULT SIZE element is *null*.

The simulator maintains a list of AMBIGUITY LIST data structures, shown in Figure B.3, for each gate G in the circuit, which provides information used for the reconvergent fanout analysis technique described in Chapter 4 and Chapter 5. The $fanoutID$ of an element in a gate's AMBIGUITY LIST is the gate ID of a fanout point f effecting gate G. The element's $d(G, f)$ and $D(G, f)$ values store the minimum and maximum delays from fanout f to gate G.

B.2 Static Timing Analysis

The static timing analysis step is performed to calculate the critical delay of the circuit and the slack for every gate in the circuit. The critical delay of the circuit is the maximum delay possible from primary input to primary output for any possible input vector pair if all gate delays are within their specified min-max delay range. The critical delay ($CritDelay$) is calculated in one forward pass over the circuit's gates in level order, as shown in Figure B.4. At each gate G, delay values are maintained ($delay(G)$), which are initialized to 0 for primary input gates. For all other gates (G), these delay values are evaluated by lines 5-9 as the maximum delay value previously calculated at fanin gates to G plus the maximum delay of gate G ($maxdel(G)$). After all gates have been evaluated, the critical delay for the circuit is the maximum delay value calculated at a primary output gate (lines 12-15). The sample period T_s is then set to one delay unit longer than the circuit's critical delay: $T_s = CritDelay + 1$.

The slack for each gate G is calculated as described by Iyengar *et al.* [34] and discussed in Chapter 3:

$$slack(G) = T_s - LMPD(G)$$

where $LMPD(G)$ is the longest minimum path delay through gate G, which is the sum of all *minimum* gate delays along the longest delay path through G. The algorithm to calculate slacks involves two passes over the circuit, and is shown in Figure B.5. The first pass is in increasing level order (forward pass) and the second is in decreasing level order (backward pass). During the forward pass, delay values are maintained for each gate G ($forwdDel(G)$), which are initialized to 0 for primary input gates. For all other gates (G), these delay values are evaluated by lines 5-9 as the maximum delay value previously

```
1: for all gates (G) in increasing level order do
2:   if G is a primary input then
3:     delay(G) = 0
4:   else
5:     delay(G) = −∞
6:     for all fanin gates G_fanin to gate G do
7:       delay(G) = max{delay(G_fanin), delay(G)}
8:     end for
9:     delay(G) = delay(G) + maxdel(G)
10:  end if
11: end for
12: CritDelay = −∞
13: for all primary output gates (G) do
14:   CritDelay = max{delay(G), CritDelay}
15: end for
```

Figure B.4: Critical Delay Calculation.

```
1: for all gates (G) in increasing level order do
2:   if G is a primary input then
3:     forwdDel(G) = 0
4:   else
5:     forwdDel(G) = −∞
6:     for all fanin gates G_fanin to gate G do
7:       forwdDel(G) = max{forwdDel(G_fanin), forwdDel(G)}
8:     end for
9:     forwdDel(G) = forwdDel(G) + mindel(G)
10:  end if
11: end for
12: for all gates (G) in decreasing level order do
13:   if G is a primary output then
14:     backDel(G) = 0
15:   else
16:     backDel(G) = −∞
17:     for all fanout gates G_fanout to gate G do
18:       backDel(G) = max{[backDel(G_fanout) + mindel(G_fanout)], backDel(G)}
19:     end for
20:   end if
21:   LMPD(G) = forwdDel(G) + backDel(G)
22:   slack(G) = T_s − LMPD(G)
23: end for
```

Figure B.5: Slack Calculation.

calculated at fanin gates to G plus the minimum delay of gate G ($mindel(G)$). During the backward pass, the delay values maintained for each gate G ($backDel(G)$) are initialized to 0 at primary output gates, and evaluated by lines 16-19 as the maximum delay value previously calculated at fanout gates of G plus the minimum delay of the fanout gate. The longest minimum path delay through each gate G ($LMPD(G)$) is then evaluated for each gate during the backward pass at line 21, and the slack for G ($slack(G)$) is evaluated at line 22.

B.3 Bounded Delay Simulation

Bounded delay simulation is performed for each vector pair (V_1 and V_2) to determine the signal waveforms at each gate in the fault-free circuit. The initial logic values of primary inputs are initialized using vector V_1, and final logic values of primary inputs are initialized using vector V_2. Initial (IV) and final values (FV) for all other gates are evaluated in one forward level order pass over the circuit. A gate's initial (final) value is evaluated using the initial (final) values of it's fanin gates and the rules of Boolean logic. The earliest arrival (EA) and latest stabilization (LS) times are then initialized for primary inputs, and evaluated at all other gates in a second forward level order pass over the circuit as described by Iyengar *et al.* [34] and discussed in Chapter 3.

Figure B.6 shows the algorithm for bounded delay simulation after initial and final logic values have been evaluated at every gate. EA and LS for primary inputs are initialized in lines 1-10 as described in [34] and discussed in Chapter 3. If hazard suppression is enabled with the "-withHazLists" option, and the primary input is also a fanout stem, an ambiguity list element is inserted at that gate G with the G's ID as the $fanoutID$ and minimum and maximum delays of zero: $d(G, G) = 0$ and $d(G, G) = 0$.

Lines 11-21 visit all gates in a forward pass. The function $evalGateTiming(G)$ at line 15 evaluates EA and LS for gate G using the EA and LS values of fanin gates as described in [34] and discussed in Chapter 3. Lines 14-16 filter out hazards that are not large enough to overcome the gate's inertia, where the inertia of a gate is the gate's minimum delay.

If hazard suppression is enabled, the function $evalAmbiguityList(G)$ (line 18) evaluates the ambiguity list at gate G as described in Chapter 5. If gate G is a reconvergent gate, and inputs to G are such that hazard suppression occurs, $evalAmbiguityList(G)$ sets the ambiguity list at the output of G to *null* and sets the output of G to a stable logic value:

$$EA(G) = \infty \qquad LS(G) = -\infty$$

If hazard suppression does not occur, $evalAmbiguityList(G)$ adds an ambiguity list element to G's ambiguity list for every fanout element found in the ambiguity lists of fanin gates (G_{fanin}) to G, and updates the minimum and maximum delays of ambiguity list

```
1: for all primary input gates G do
2:   if IV(G) ≠ FV(G) then
3:     EA(G) = 0 and LS(G) = 0
4:   else
5:     EA(G) = ∞ and LS(G) = −∞
6:     if G is a fanout stem and hazard suppression is enabled then
7:       insertAmbiguityList(G)
8:     end if
9:   end if
10: end for
11: for all gates (G) in increasing level order do
12:   if G is not a primary input then
13:     evalGateTiming(G)
14:     if [LS(G) − EA(G)] < mindel(G) then
15:       EA(G) = ∞ and LS(G) = −∞
16:     end if
17:     if hazard suppression is enabled then
18:       evalAmbiguityList(G)
19:     end if
20:   end if
21: end for
```

Figure B.6: Bounded Delay Simulation.

elements using the minimum and maximum delays of gate G:

$$d(G, f) = min\{d(G_{fanin}, f)\} + mindel(G)$$
$$D(G, f) = max\{D(G_{fanin}, f)\} + maxdel(G)$$

B.4 Detection Threshold Evaluation

The detection threshold evaluation step performs fault simulation to propagate fault effects through the circuit for each gate delay fault that is activated by the input vector pair. Fault propagation begins by initializing the FAULT data structure's reference quantities, FPV, RTa, RTb, and ρ for the gate at the fault site and for gates that lie outside the cone of influence of the fault as described by Iyengar *et al.* [34] and discussed in Chapter 3. If hazard suppression is enabled with the "-withHazLists" option, and the fault site is a fanout stem, an ambiguity list is also created at the fault site with the gate ID of fanout point G and zero delays: $d(G, G) = 0$ and $D(G, G) = 0$ [13, 27]. The FAULT data structure reference quantities at gates inside the cone of influence of the fault site are evaluated during fault propagation as described in [34] and discussed in Chapter 3.

```
 1: for all gates (G) do
 2:   if IV(G) ≠ FV(G) then
 3:     if G is a fanout stem and hazard suppression is enabled then
 4:       insertAmbiguityList(G)
 5:     end if
 6:     for all fanout gates G_out of gate G do
 7:       enqueue(G_out)
 8:     end for
 9:     while queue is not empty do
10:       G_q = dequeue()
11:       evalRefVals(G_q)
12:       if hazard suppression is enabled then
13:         evalAmbiguityList(G_q)
14:       end if
15:       for all fanout gates G_out of gate G_q do
16:         enqueue(G_out)
17:       end for
18:     end while
19:     updateGlobalFaultInfo()
20:   end if
21: end for
```

Figure B.7: Fault Simulation.

The fault propagation algorithm is shown in Figure B.7. Reference quantities for gates inside the downcone of the fault site (G_q) are evaluated as described in [34] and discussed in Chapter 3 by the $evalRefVals(G_q)$ function at line 11. If hazard suppression is enabled, the $evalAmbiguityList(G_q)$ function at line 13 evaluates the ambiguity list for gate G_q as described in Chapter 4. If hazard suppression does not occur for gate G_q, $evalAmbiguityList(G_q)$ updates the ambiguity list at G_q by adding an element for every fanout element found in the ambiguity lists of fanin gates to G_q. The minimum and maximum delays of ambiguity list elements, $d(G_q, f)$ and $D(G_q, f)$, are updated using the minimum and maximum delays of G_q:

$$d(G_q, f) = min\{d(G_{fanin}, f)\} + mindel(G_q)$$
$$D(G_q, f) = max\{D(G_{fanin}, f)\} + maxdel(G_q)$$

If G_q is a reconvergent gate, and inputs to G_q are such that hazard suppression does occur, $evalAmbiguityList(G_q)$ updates the reference quantities at G_q to:

$$\rho(G_q) = 0$$
$$RTa(G_q) = -\infty$$
$$RTb(G_q) = \infty$$

and sets the ambiguity list at the output of G_q to the empty ($null$) list.

The detection threshold ($Threshold$) and detection gap (Gap) for a detected gate delay fault at output z is then calculated as described in [34] and discussed in Chapter 3 ($Threshold = max\{\rho(z), T_s - RTb(z)\}$ and $Gap(G) = Threshold(G) - slack(G)$). The $updateGlobalFaultInfo()$ function at line 19 updates the slow-to-rise and slow-to-fall FAULT SIZE data structures to store the best RTa, RTb, ρ, detection threshold ($Threshold$), and detection gap (Gap) information for all input vector pairs that have been simulated.

B.5 Program Evaluation

CPU execution times for the ISCAS85 benchmark circuits are shown in Table B.1 and Table B.2 for gate delay fault simulation of 1,000 input vectors without propagating ambiguity lists. For the data in Table B.1, zero delay fanout buffers were not inserted on fanout branches. For the data in Table B.2, zero delay fanout buffers were inserted on all fanout branches, increasing the number of gates for each circuit. In both tables, columns two, three, and four show the number of primary inputs, primary outputs, and gates for each circuit, and column five shows the execution time on a 2 GHz Intel Pentium processor. By inserting zero delay buffers on all fanout branches, the number of gates for each circuit increases by approximately a factor of two, which results in an increase in CPU execution time by a factor of about four.

Table B.3 shows a separate set of execution times for simulation of the ISCAS85 benchmark circuits for 1,000 input vectors with and without the ambiguity list propagation technique described in Chapter 5. For the data Table B.3, zero delay fanout buffers were inserted on all fanout branches in each circuit. Column five shows the performance when the ambiguity list propagation technique is not used, and column six shows the performance when ambiguity lists are propagated during both bounded delay simulation of the fault-free circuit and gate delay fault simulation. Propagating ambiguity lists during simulation of both the fault-free and faulty circuit increases CPU execution time by a factor of about 1.8.

Table B.1: Performance When Zero Delay Buffers Are Not Inserted on Fanout Branches.

	Number Primary Inputs	Number Primary Outputs	Number of Gates	CPU Execution Time (sec.)
c432	36	7	250	5
c499	41	32	555	23
c880	60	26	443	4
c1355	41	32	587	31
c1908	33	25	913	42
c2670	233	140	1426	31
c3540	50	22	1719	72
c5315	178	123	2485	84
c6288	32	32	2448	267
c7552	207	108	3719	201

Table B.2: Performance When Zero Delay Buffers Are Inserted on Fanout Branches.

	Number Primary Inputs	Number Primary Outputs	Number of Gates	CPU Execution Time (sec.)
c432	36	7	558	16
c499	41	32	1323	82
c880	60	26	880	15
c1355	41	32	1355	101
c1908	33	25	1908	120
c2670	233	140	2670	102
c3540	50	22	3540	253
c5315	178	123	5315	370
c6288	32	32	6288	1460
c7552	207	108	7552	1005

Table B.3: Performance With and Without Reconvergent Fanout Analysis.

	Number Primary Inputs	Number Primary Outputs	Number Of Gates	CPU Time (sec.) Without Reconvergent Fanout Analysis	CPU Time (sec.) With Reconvergent Fanout Analysis
c432	36	7	558	16	30
c499	41	32	1323	82	136
c880	60	26	880	15	24
c1355	41	32	1355	101	154
c1908	33	25	1908	120	222
c2670	233	140	2670	102	158
c3540	50	22	3540	253	495
c5315	178	123	5315	370	636
c6288	32	32	6288	1460	4316
c7552	207	108	7552	1005	1516