ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

Department of Electrical and Computer Engineering

Identification of High-Risk Hardware Path-Delay Fault Locations and Evaluation of Their Impact

By

Tadele Basazenew Ayele

Advisor

Mr. Menor Tekeba

A Thesis Submitted to the School of Graduate Studies of Addis Ababa University in Partial Fulfillment of the Requirements for the Degree of Masters of Science in Computer Engineering

September, 2014

ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

Electrical and Computer Engineering Department

Identification of High-Risk Hardware Path-Delay Fault Locations and Evaluation of Their Impact

By

Tadele Basazenew Ayele

Approval by Board of Examiners:

Prof.H.Y.(Hayal) Kwon _____
Chairman, Department Graduate Committee Signature

Mr. Menor Tekeba _____
Advisor Signature

---------- _____
Internal Examiner Signature

---------- _____
External Examiner Signature

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification.

Tadele Basazenew Ayele _____
Name Signature

Addis Ababa, Ethiopia September, 2014
Place Date of Submission

This thesis has been submitted with my approval as a university advisor

Ato Menor Tekeba _____
Advisor Name Signature

Abstract

Ascertaining correct operation of digital logic circuits requires verification of functional behavior as well as correct operation at desired clock speed. The maximum allowable clock rate in a digital circuit is determined by the propagation delays of the combinational logic network between latches. If the delay of the manufactured network exceeds specifications due to some physical defects or process variations, non-confidential and possibly incorrect logic values may be latched in memory elements. In this thesis, we present novel and efficient model for path delay faults specifically for stack at fault in combinational logic circuits.

We propose new and efficient Model for delay fault analysis, test generation and fault simulation of path delay faults in combinational logic circuits. Then the new model was analyzed using reduced order binary decision diagram of the Colorado University Decision Diagram package. An approach for selecting critical paths along which testable path delay faults can exist is presented. The proposed method is particularly helpful on path intensive circuits (large number of paths). Critical paths are selected implicitly with the aid of a combination of decision diagrams.

Ideally, all the path delay faults of a circuit should be tested. However, a circuit may have a very large number of paths, making it impossible to target all the path delay faults explicitly during test generation or fault simulation. The large numbers of paths in practical circuits lead to the use of path selection, where only subsets of the path delay faults in circuits are targeted for test generation, in this case only high-risk paths. To reduce our efforts for finding test vectors, which in turn reduce testing memory and processor power and analyzing a circuit for its faults, we try to use reduced faults. Reduced faults can be obtained by eliminating redundant ones and ignoring some that do not occur often or by eliminating faults that have the same output effect by fault collapsing rules.The effectiveness of the approach is demonstrated on path intensive international symposium on circuits and systems (ISCAS'85) and International Transmission Company (ITC'99) benchmarks.

Keywords:-High-risk Paths, Delay Fault Model, ROBDD, Fault reduction

Acknowledgments

I would like to extend my sincere gratitude and appreciation to all those who have challenged, supported and encouraged me in the entire course of my study. First and foremost, I bow before the Lord Almighty with a grateful heart, for His blessings which have made me what I am.

I am deeply grateful to my thesis advisor, Mr. Menor Tekeba and would like to extend my gratitude for all the expert guidance, valuable professional advice and help throughout my thesis. To work with him was a great opportunity for me and has been a great learning experience. He has been a constant support throughout the various phases of the thesis.

I would like to express my sincere appreciation to all electrical and computer engineering staff who guided and extended their valuable knowledge and advice through the different phases of the seminars which helped me in my research work.

I would like to express my deepest and everlasting gratitude to my parents for their love, support and encouragement throughout my life. Their contribution towards my thesis is immeasurable.

Table of Contents Page

List of Figures Page

List of Tables Page

List of Acronyms and symbols

ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
BDD	Binary decision diagram
BLIF	Berkley Logic interchange format
BIST	Built in self-test
CAD	Computer-aided Design
CMOS	Complementary Metal-Oxide Semiconductors
CPT	Critical Path-Tracing
CUDD	Colorado University Decision Diagrams
CUT	Circuit under Test
DFT	Design for Testability
DSM	Deep Sub-Micron
DF	Detected Faults
DAG	Directed Acyclic Graph
DC	Direct Current
ECO	Engineering Change Orders
EDA	Electronic Design Automation
FC	Fault Coverage
FPGA	Field Programmable Gate Arrays
FUT	Fault-able Circuit Under-Test

GUT	Good-Circuit Under-Test
HDLs	Hardware Programming Languages
IE-Graph	Instruction Execution Graph
IEEE	Institute of Electrical and Electronics Engineers
ICs	Integrated Circuits
IJRET	International Journal of Research in En'ng &Tech
IO	Input-Output
ISCAS	International Symposium on Circuits and Systems
ITC	International Transmission Company
IP	Intellectual Property
LSSD	Level-Sensitive Scan Design
LUTs	Lookup Tables
MUX	Multiplexer
NF	Number of Faults
NP	Non-Deterministic Polynomial
OBDD	Ordered Binary Decision Diagrams
OBIST	Oscillation based built-in self-test
O (N)	Big Oh (polynomial complexity)
PCBS	Printed Circuit Boards
PDFM	Path Delay Fault Model
PDF	Path Delay Fault

PI	Primary Input
PLI	Programming Language Interface
PODEM	Path-Oriented Decision Making
PO	Primary Output
ROBDD	Reduced Ordered Binary Decision Diagrams
ROM	Read Only Memory
RTL	Register Transfer Level
SA0	Stuck-At zero
SA1	Stuck-At one
SAT	SATISFIABILITY (Boolean satisfiability problem)
SECT	Standard for Embedded Core Test
SEU	Single Event Upset
SOC	System On Chip
SOPC	System On a Programmable Chip
SPDF	Single Path Delay Fault
SRAM	Static Random Access Memory
SRLs	Shift Register Latches
STF	Slow To Fall
STR	Slow-To Rise
SSBDD	Structurally Sensitized Binary Decision Diagrams
TLM	Transaction level model
VLSI	Very Large Scale Integration
VPI	Verilog Programming Language Interface
ZBDD	Zero-Suppressed Binary Decision Diagrams

Chapter One

1. Introduction

1.1. Background

Extreme scaling practices in silicon technology are quickly leading to integrated circuit components with limited reliability, where phenomena such as early-transistor failures, gate-oxide wear out, and transient faults are becoming increasingly common. In order to overcome these issues and to develop robust design techniques for large-market silicon ICs, it is necessary to rely on accurate failure analysis frameworks which enable design houses to faithfully evaluate both the impact of a wide range of potential failures and the ability of candidate reliable mechanisms to overcome them. Unfortunately, while failure rates are already growing beyond economically viable limits, no fault analysis framework is yet available that is both accurate and can operate on a complex integrated system [1].

The reduction in feature size (according to Moore's law, figure 1.1 below) has also resulted in increased operating frequencies and clock speeds; for example, in 1971, the first microprocessor ran at a clock frequency of 108 KHz, while current commercially available microprocessors commonly run at several gigahertzes. The reduction in feature size increases the probability that a manufacturing defect in the IC will result in a faulty chip. A very small defect can easily result in a faulty transistor or interconnecting wire when the feature size is less than 100nm. Furthermore, it takes only one faulty transistor or wire to make the entire chip fail to function properly or at the required operating frequency. Yet, defects created during the manufacturing process are unavoidable, and, as a result, some number of ICs is expected to be faulty; therefore, testing is required to guarantee fault-free products, regardless of whether the product is a VLSI device or an electronic system composed of many VLSI devices. It is also necessary to test components at various stages during the manufacturing process. For example, in order to produce an electronic system, we must produce ICs, use these ICs to assemble printed circuit boards (PCBs), and then use the PCBs to assemble the system. There is a general agreement with the rule of ten(a common rule in VLSI design and verification), which says that the cost of detecting a faulty IC increases by an order of magnitude as we move through each stage of

manufacturing, from device level to board level to system level and finally to system operation in the field[2]

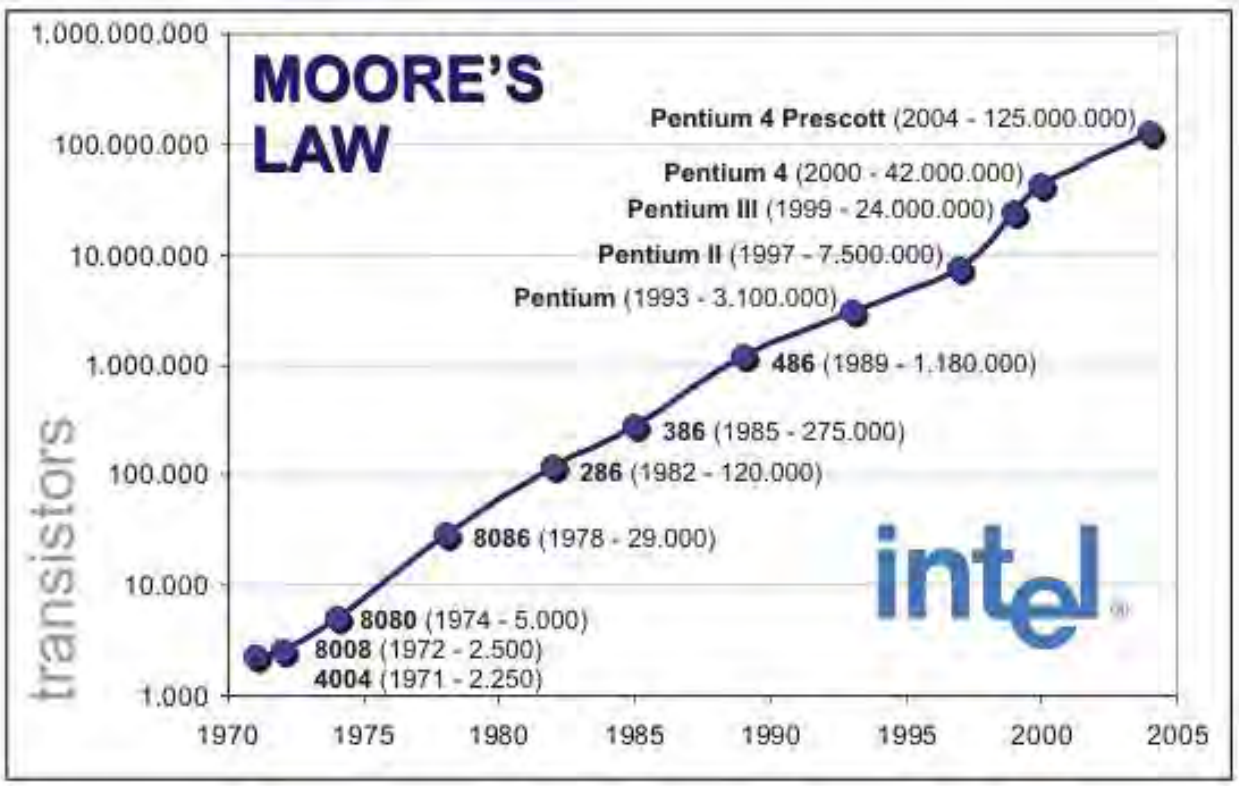


Figure 1.1:- VLSI complexity as per Moore's law [2] [3]

Electronic testing includes IC testing, PCB testing, and system testing at the various manufacturing stages and, in some cases, during system operation. Testing is used not only to find the fault-free devices, PCBs, and systems but also to improve production yield at the various stages of manufacturing by analyzing the cause of defects when faults are encountered. In some systems, periodic testing is performed to ensure fault-free system operation and to initiate repair procedures when faults are detected. Hence, VLSI testing is important to designers, product engineers, test engineers, managers, manufacturers, and end-users [1].

Fault-free operation of a logic circuit requires not only performing the logic function correctly but also propagating the correct logic signals along paths within a specified time limit. A delay fault causes excessive delay along a path such that the total propagation delay falls outside the specified limit. Delay faults have become more prevalent with decreasing feature sizes.

There are different delay fault models. In the gate-delay fault and the transition fault models, a delay fault occurs when the time interval taken for a transition from the gate input to its output exceeds its specified range. It should be noted that simultaneous transitions at inputs of a gate may change the gate delay significantly due to activation of multiple charge/discharge paths. The other model is path-delay fault, which considers the cumulative propagation delay along a signal path through the CUT—in other words, the sum of all gate delays along the path; therefore, the path-delay fault model is more practical for testing than the gate-delay fault (or the transition fault) model. A critical problem encountered when dealing with path-delay faults is the large number of possible paths in practical circuits. This number, in the worst case, is exponential for the number of lines in the circuit, and in most practical cases the number of paths in a circuit makes it impossible to enumerate all path-delay faults for the purpose of test generation or fault simulation [2][3][4].

As with transistor stuck-open faults, delay faults require an ordered pair of test vectors to sensitize a path through the logic circuit and to create a transition along that path in order to measure the path delay. For example, consider the circuit in Figure 1.2, where the fault-free delay associated with each gate is denoted by the integer value label on that gate. The two test vectors, v1and v2, shown in the figure.

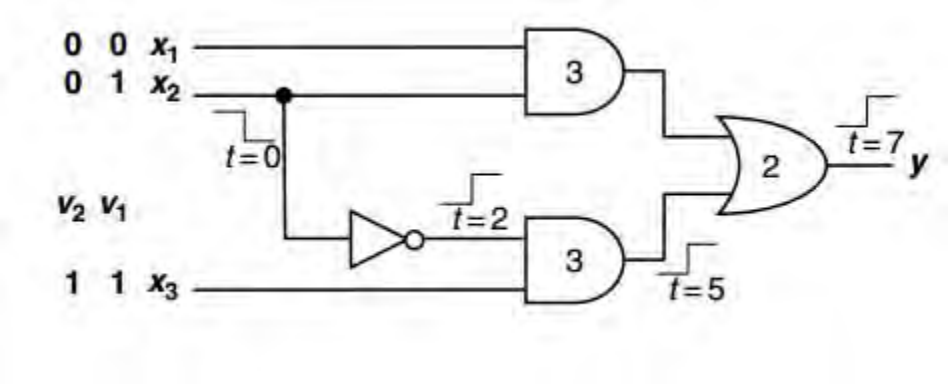


Figure 1.2 Path-delay Fault Test

are used to test the path delay from input x2, through the inverter and lower AND gate, to the output y. Assuming the transition between the two test vectors occurs at time t=0, the resulting transition propagates through the circuit with the fault-free delays shown at each node in the circuit such that we expect to see the transition at the output y at time t=7. A delay fault along

this path would create a transition at some later time t>7. Of course, this measurement could require a high-speed, high-precision test machine.

With decreasing feature sizes and increasing signal speeds, the problem of modeling gate delays becomes more difficult. As technologies approach the deep sub-micron region, the portion of delay contributed by gates reduces while the delay due to interconnect becomes dominant. This is because the interconnect lengths do not scale in proportion to the shrinking area of transistors that make up the gates. In addition, if the operating frequencies also increase with scaling, then the on-chip inductances can play a role in determining the interconnect delay for long wide wires, such as those in clock trees and buses. However, wire delays can be taken into account in the path-delay fault model based on the physical layout, as interconnections are included in paths. As a result, it is no longer true that a path delay is equal to the sum of all delays of gates along the path [4].

The use of nanometer technologies increases cross-coupling capacitance and inductance between interconnects, leading to severe crosstalk effects that may result in improper functioning of a chip. Crosstalk effects can be separated to two categories: crosstalk glitches and crosstalk delays. A crosstalk glitch is a pulse that is provoked by coupling effects among interconnect lines. The magnitude of the glitch depends on the ratio of the coupling capacitance to the line-to-ground capacitance. When a transition signal is applied on a line that has a strong driver while stable signals are applied at other lines that have weaker drivers, the stable signals may experience coupling noise due to the transition of the stronger signal. Crosstalk delay is a signal delay that is provoked by the same coupling effects among inter-connects lines, but it may be produced even if line drivers are balanced but have large loads. Because cross talk causes a delay in addition to normal gate and inter-connects delay, it is difficult to estimate the true circuit delay, which may lead to severe signal delay problems. Conventional delay fault analysis may be invalid if these effects are not taken into consideration based on the physical layout. Several design techniques, including physical design and analysis tools, are being developed to help design for margin and minimization of crosstalk problems; however, it may be impossible to anticipate in advance the full range of process variations and manufacturing defects that may significantly aggravate the cross-coupling effects. Hence, there is a critical need to develop testing techniques for manufacturing defects that produce crosstalk effects [1].

To simplify path delay fault side effects that we tried to test for, we develop a fault model. The path delay fault model that we use is only a model to simplify analysis of a circuit for finding better test vectors and evaluating them based on buffer-inverter insertion depending on the fast and slow nature of the delay faults that are found from commercial simulators. Based on this model we define fault coverage as the percentage of faults that have been detected. For a given test vector, fault coverage determines its efficiency in the number of faults it can detect [5].

1.1.1.Fault Analysis

VLSI density is growing as per Moore's law and integrated circuit designs are accordingly becoming more and more complex [1-10]. As a result of this, VLSI testing has become expensive in terms of cost. Existing gate level fault simulation techniques exhibit poor performance standards when applied to such designs and are unsuitable for early testability analysis or fault simulations. Also test generation and fault simulation efforts in the post synthesis phase do not contribute to the improvement in the design. Therefore a design methodology for fault simulation at higher levels of abstraction is highly desired. Many high-level fault models and fault simulation techniques have been proposed. No single fault model is universally acceptable since no fault model has been developed so far that comprehensively covers all classes of circuits [7]. The smaller size makes IC chips more sensitive to fabrication variations and tolerance accumulations. Thus there is a growing demand for fault tolerance, which can be achieved not only by improving the reliability of the functional units, but also by an efficient fault detection, isolation and accommodation concept. Consequently, testing and fault diagnosis is becoming one of the major cost factors in the overall IC manufacturing expenses. It has been recognized as a valuable means to:

- Check system installation and configuration after maintenance activities,
- Ensure correct system functionality at start-up, and
- Avoid masking and accumulation of errors during operation.

In the context of fault diagnosis, a fault is understood as any kind of malfunction in the system that leads to an unacceptable anomaly in the overall system performance. A fault in a system can be very costly in terms of loss of production, equipment damage and economic setback. Faults are developed in a system due to normal wear and tear, design or manufacturing defects or

improper operation leading to stress beyond endurable limits. In many cases degradation in the performance of the system is sustained for some duration before it actually "fails". In many other cases a system continues to operate with a failed component resulting in degraded performance.

As silicon process technology pushes towards smaller technology sizes, device reliability is an emerging challenge for next-generation designs [11]. Silicon failure mechanisms such as early transistor failures, gate-oxide wear-out, manufacturing defects, and radiation-induced soft errors threaten the design's reliability and severely reduce the yield and lifetime of our products [12]. These reliability challenges are usually addressed either by conservative high-margin design techniques that avoid the manifestation of device failures during the product lifetime or by fault-tolerant techniques that detect failures and repair the system functionality in the field during operation [13]. Classic high-margin techniques for guaranteeing system reliability are rendered inadequate by extreme technology scaling and process variation. The result is a steady shift towards the adoption of fault-tolerant techniques into the design flow of modern computing systems [10].

Path delay fault model assumes that the logic function of the circuit is correct, but that the total delay in a path from inputs to outputs exceeds the allocated threshold and causes incorrect behavior. This model is used to mimic the effects of process variation or device degradation due to age-related wear outs.

Delay testing has become an important issue in the post production test due to the increased speed and continuously shrinking feature size of modern circuits. Several delay fault models have been proposed. Among them, the most accurate fault model is the Path Delay Fault Model (PDFM) [12]. This fault model captures small as well as large delay defects distributed along one path in the circuit. However, the number of paths in modern circuits is typically excessively large. Test generation for all paths is therefore not possible because of the long ATPG run time and the large number of generated tests. For that reason, usually only tests for critical paths are generated to ensure the correct timing behavior of signals on this path. Furthermore, tests for the PDFM are used for diagnostic reasons if the timing behavior of particular paths should be verified. High-quality tests are required especially for diagnosis.

Failures that cause logic circuits to malfunction at the desired clock rate and thus violate timing specifications are currently receiving much attention. Such failures are modeled as delay faults. They facilitate delay testing. The use of delay fault models in VLSI test generation is very likely to gain industry acceptance in the near future. In this paper, we are going to present delay fault models discuss their classifications and examine fault coverage metrics that have been proposed in the recent literature. A comparison between delay fault models, namely, gate delay, transition, path delay, line delay and segment delay faults, shows their benefits and limitations. Various classifications of the path delay fault model, which have received the most attention in recent years, are reviewed. We believe that an understanding of delay fault models is essential in today's VLSI design and test environment [7][11][12][13][14].

1.1.2.Delay Fault Models and Simulation For Fault-Tolerant Design

In engineering, fault-tolerant design is a design that enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails. The term is most commonly used to describe computer-based systems designed to continue more or less fully operational with, perhaps, a reduction in throughput or an increase in response time in the event of some partial failure. That is, the system as a whole is not stopped due to problems either in the hardware or the software. An example in another field is a motor vehicle designed so it will continue to be drivable if one of the tires is punctured. A structure is able to retain its integrity in the presence of damage due to causes such as fatigue, corrosion, manufacturing flaws, or impact [2].

Hardware fault tolerance is the most mature area in the general field of fault-tolerant computing. Many hardware fault-tolerance techniques have been developed and used in practice in critical applications ranging from telephone ex-changes to space missions. In the past, the main obstacle to a wide use of hardware fault tolerance has been the cost of the extra hardware required. With the continued reduction in the cost of hardware, this is no longer a significant drawback, and the use of hardware fault-tolerance techniques is expected to increase. However, other constraints, notably on power consumption, may continue to restrict the use of massive redundancy in many applications.

The single most important parameter used in the reliability analysis of hardware systems is the component failure rate, which is the rate at which an individual component suffers faults. This is the expected number of failures per unit time that a currently good component will suffer in a given future time interval. The failure rate depends on the current age of the component, any voltage or physical shocks that it suffers the ambient temperature, and the technology. The dependence on age is usually captured by what is known as the bathtub curve [2].

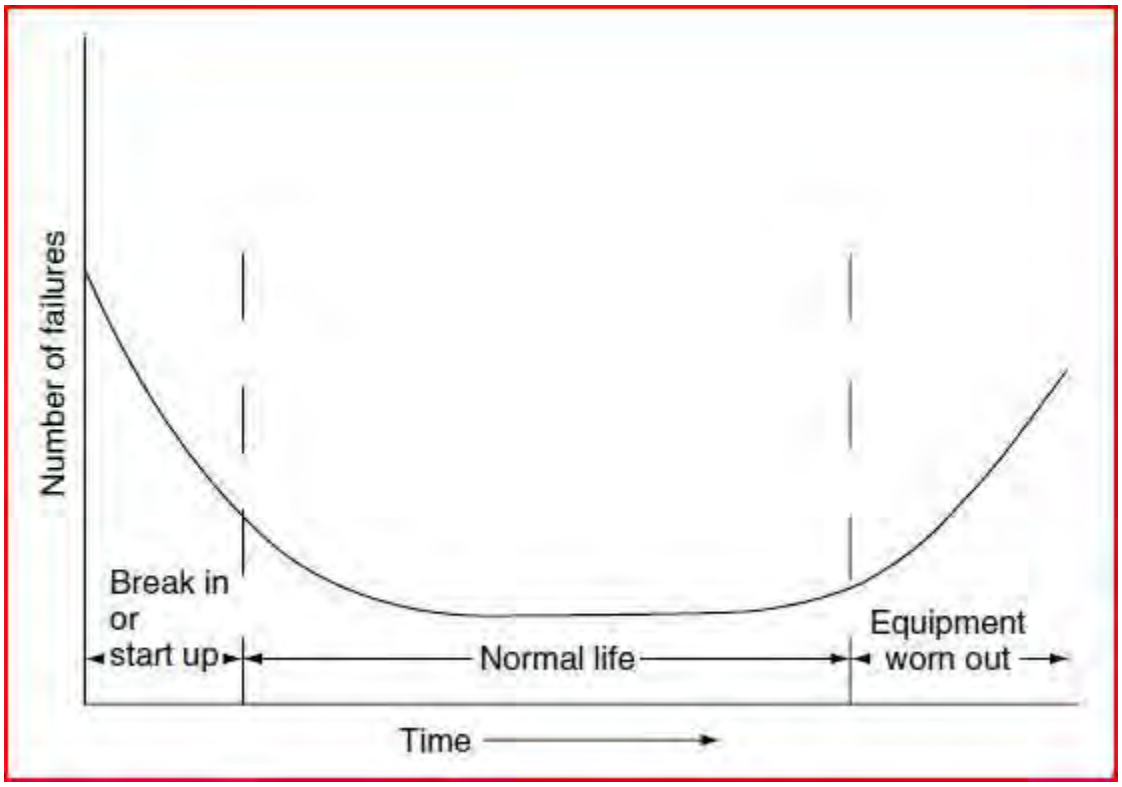


Figure1.3:- The bathtub Curve [2]

According to figure 1.3, when components are very young, their failure rate is quite high. This is due to the chance that some components with manufacturing defects slipped through manufacturing quality control and were released. As time goes on, these components are weeded out, and the component spends the bulk of its life showing a fairly constant failure rate. As it becomes very old, aging effects start to take over, and the failure rate rises again.

To design a fault tolerant system, we need an efficient test and verification methodology, model or algorithm to make our design as correct as we can. The verification techniques should be implemented at the different levels of abstraction of the design cycle.

1.2. Statement of Problem

Despite numerous efforts in improving path delay fault modeling techniques in digital circuitry using different HDLs and different FPGA modules; there is still a challenge in obtaining accurate result. Delay test in nano-scale VLSI circuits becomes more difficult with shrinking technology feature sizes and rising clock frequencies. Previous research of fault modeling on resistive spot defects, such as resistive opens and bridges in the interconnects and resistive shorts in devices, lacked an accurate fault model. As a result it was difficult to perform fault simulation and select the best vectors. Conventional methods to compute vibrational delay under process variation are either slow or inaccurate[1,6].

There are so many path delay fault analysis and identification tools and models presented in many papers [6][7][8][9][10]; but none of them are used the functionality of ROBDD for the path delay fault simulation and fault reduction together. Both fault reduction and ROBDD techniques are used to reduce simulation cost and processing overhead. Even though, the proposed methods and tools are low cost, they are not fast and efficient in terms of simulation performance and speed. To improve this trade off many researches have been conducted on hardware based path delay fault modeling and simulation. But none of these are on the high-risk path delay fault models. In this paper, we are going to present a path delay model that used to identify a high-risk path delay fault location and evaluating the impact of that fault location on the overall combinational hardware performance and functionality. The paper mainly focuses on creating path delay fault models and analysis to identify the critical paths in circuits (combinational) paths with the longest delays top 10-20%, and then injects path-delay faults only on those paths. We will then have to run a number of varied simulations to gather whether these faults place in those locations have impact at the overall system performance. To implement this paper we used the ordered binary decision diagram (OBDD) data structure's ability for hardware verification. On the other hand the paper also focuses on modeling delay faults by using buffers for slow model and invertors for fast model of the delay faults in digital ICs. Considering only high-risk paths out of very large number of interconnection lines, it is possible to reduce the test vector count and to increase test quality and also to reduce test vector coverage memory and computing power (CPU power) that is required by the system.

1.3. Objectives

1.3.1. General Objective

The general objective of this thesis is to model a path delay fault and to analyze and identify the critical paths in a circuit (combinational paths with the longest delays-top 10-20%), and evaluation of their impact and also to analyze the different types of fault models. That is, first we have designed a stuck at fault model and then using that model analysis its effect on different benchmark circuits by using the user defined threshold from 80% to 90% of the high-risk paths.

1.3.2. Specific Objectives

The specific objectives of this thesis are:

- To explore the impact of path delay faults in digital circuitry.
- Identify the major difficulties faced in studying and identifying path delay faults and try to come up with the solution.
- Designing viable path delay fault model and to use that model for analysis of high-risk path delay faults
- Analyze and evaluate the impact of different faults(stuck at, stuck open and bridge) on path delays
- Analyze the deferent fault models and evaluate their impact on digital circuitry.

1.4. Methodology and Scope

- The scope of this thesis is designing efficient critical Path delay Fault model and analysis of different benchmark circuits using proposed model. It is mainly focused on the identification of high-risk paths (combinational paths with the longest delays-top 10-20%) and evaluation of their impact than analyzing the whole set of paths in a certain digital circuitry due to its practical infeasibility, specifically combinational circuitry i.e not for sequential circuts.

  The methodologies that are going to be followed in this thesis are:

  1. Literature Review: Literature to be reviewed includes articles, journals, books any relevant resource.

2. Design Path delay fault Model with a better efficiency to analyze and evaluate their impact on modern VLSI fabrications.
3. Simulating the developed model using C++ and C for manipulation of ROBDD on top of the CUDD package and then analysis and interpretation of results obtained.

1.5. Contribution Of The Thesis

- Since silicon fault modeling is an open area of research, we propose new and efficient delay model using buffer insertion technique for delay test analysis and fault simulation of path delay faults in combinational logic circuits and analysis it using ROBDD of the tool in [1,14].
- An approach for selecting high-risk paths along which testable path delay faults using user defined threshold (80% and 90% of the maximum delay) can exist is presented. The proposed method is particularly helpful on path intensive circuits. Critical paths are selected implicitly with the aid of a combination of decision diagrams. The effectiveness of the approach is demonstrated on path intensive ISCAS'85 and IT C'99 benchmarks.
- It has been previously shown that in order to guarantee the temporal correctness of a circuit, only the primitive path delay fault set needs to be tested. However, as in the case of the traditional and simpler path delay fault model, the number of possible faults can be exponential to the circuit size and, therefore, it is only practical to consider the set of high risk path delay faults. This work defines critical primitive path delay faults and identify them using reduced order binary decision diagrams (ROBDD). We report the number of critical primitive path delay faults for criticality threshold under the bounded delay model. The results indicate that only a small, but still necessary, number of multiple (primitive) faults, which escape testing under the singly testable fault criterion, must be considered in order to guarantee the timing correctness of the circuit.
- The other contribution of the thesis is the use fault reduction to reduce test overhead. To reduce our efforts for finding test vectors and analyzing a circuit for its faults, we try to use reduced faults that we deal with by eliminating redundant ones, and, perhaps, ignoring some that do not occur often. Reducing number of faults can be incorporated in our fault model, i.e., using a simple model consisting of very few fault types. In addition, reducing can be done by eliminating faults that have the same output effect; this process is referred to as fault collapsing. The fault collapsing scheme indirectly helps to reduce

the test vector amount (test size) and this also helps to reduce the CPU and memory overhead.

1.6. Thesis Organization

The remainder of the thesis is organized as follows:

- Chapter two is a literature review part that discusses about some of state of the art path delay fault analysis, test generation and simulation methods in VLSI design in general.
- The third chapter presents the available path delay fault models and their properties, shortcomings and strong sides. This chapter presents a brief discussion on the available delay fault models and one test generation method for single stuck-at fault model. It also presents the proposed delay model based on buffer insertion technique at the RTL level and the simulation output of the faulty and fault free version of a sample ISCAS'85 c17 benchmark circuit.
- Similarly, chapter four provides a detailed discussion of the path delay fault simulation using Verilog programming language interface and some methods of delay fault simulation like critical path tracing. And it also presents binary decision diagram as a light weight data structure (used to reduce the CPU time and processing memory) to simulate and analyze hardware faults in general and delay faults in particular. This chapter presents the equivalent checking between two hardware designs described as a text file like BLIF format and combinational equivalence checking.
- Chapter five presents the results of the delay fault simulation on ISCAS'85 benchmark circuits BDD techniques and the performance improvement as compared to other related works.
- Finally, chapter six concludes the thesis and presents ideas for future work.

Chapter Two

2. Literature Review

There are a number of researches that had been done on path delay fault analysis. These research works has addressed different dimensions in the locating path delay faults and analyzing their effects. The works that are presented are divided in to different sections as described below.

2.1. Decision Diagram Based Methods

The current interest in BDDs from the theoretical computer science community has largely been motivated by the practical importance and success of BDDs in formal hardware verification. Conversely, the growing industrial interest in formal hardware verification has largely been inspired by the effectiveness of BDD-based techniques in finding real bugs in practical, large-scale designs.

Delay fault test method that is based on complex data structures is the near-critical path analysis: a tool for parallel program optimization in [7]. They proposed that program activity graphs (PAGs) can be constructed from time stamped traces of appropriate execution events. Information about the activities on the k longest execution paths is useful in the analysis of parallel program performance. In their paper, four algorithms for finding the near-critical paths of PAGs are compared. A framework for using the near-critical path information is also described. The framework includes statistical summaries and visualization capabilities that build upon the foundation of existing performance analysis tools. Within the framework, guidance is provided by the maximum benefit metric, which uses near-critical path data to predict the maximum overall performance improvement that may be realized by optimizing particular critical path activities. The shortcoming of the paper is that, it uses lower bounds as target of faults.

The other paper presented in [9] presents exact path delay fault coverage with fundamental zero-suppressed BDD operations. They formulate the path delay fault grading problem as a combinatorial problem that amounts to storing and manipulating sets on a special type of Binary Decision Diagrams (BDD), called zero-suppressed binary decision diagrams (ZBDD). The zero-

suppressed BDD is a canonical data structure inherently having the property of representing combinational sets very compactly. A simple modification of the basic scheme allows them to increase significantly the storage capability of the data structure with minimal loss in the fault coverage. Experimental results on the ISCAS'85 benchmarks show considerable improvement over all existing techniques for exact PDF grading. The proposed methodology is simple; it consists of a polynomial number of increasingly efficient ZBDD based operations and can handle very large test sets that grade very large number of faults. But this paper doesn't use ROBDD and fault reduction.

The other data structure based testing approach proposed in [10] clarifies static variable ordering in ZBDDs for path delay fault coverage calculation. They states that Zero-suppressed Binary Decision Diagrams (ZBDDs) are data structures that represent sets efficiently and they have recently been suggested for use in non-enumerative path delay fault (PDF) coverage calculations. Many heuristics have been proposed to order variables (representing primary inputs) in ZBDDs to avoid size explosion; however, in ZBDD-based PDF coverage calculations, the variables represent the nets in a circuit, not the circuit primary inputs. This fact motivates them to investigate new ordering strategies since the number of nets in a circuit is relatively large as compared to the number of primary inputs. Several new static ordering heuristics are proposed based on structural properties of the circuit undergoing PDF coverage calculations and are evaluated. The experimental results show that the new heuristics they propose greatly reduce the size of the ZBDDs. This paper didn't try the efficiency of ROBDD.

The embedded software based self-testing for SOC design approach proposed in [11] indicate the programmable cores are used for on-chip test generation, measurement, response analysis and even diagnosis. After the programmable core on a System-on–Chip (SoC) has been self-tested, it can be reused for testing on-chip buses, interfaces and other non-programmable cores. The advantages of this methodology include at-speed testing, low design-for-testability overhead and application of functional patterns in the functional environment. They had proposed that First, the microprocessor tests itself by executing a set of instructions. Next, the processor can be used for testing the bus and other non-programmable IP cores in the SoC. In order to support the self-testing methodology, the IP core has a test wrapper around it.

The software base approach proposed in [13] states the programmable cores on SoCs can perform on-chip test generation, measurement, response analysis, and even diagnosis. This software-based approach to self-testing enables at-speed testing and incurs low DFT overhead. The work also presents to apply at-speed tests to detect timing-related faults; existing structural BIST must resolve various complex timing issues related to multiple clock domains, multiple frequencies, and test clock skews that are unique in test mode. A new paradigm, embedded software-based self-testing, could alleviate the problems of both external testers and structural BIST. In their strategy, sometimes called functional self-testing, the SoC's programmable cores first under-go self-test by running an automatically synthesized test program that achieves high fault coverage. Next, the programmable core functions as a pattern generator and response analyzer to test on-chip buses, interfaces between cores, and even other cores, including digital, mixed signal, and analog components. In addition, functional information can guide diagnostic self-test program synthesis.

The method proposed in [18] proposes delay fault testing of IP-based designs via symbolic path modeling. Delay testing of designs that contain intellectual property (IP) cores is challenging. They propose a method that can test paths traversing both IP cores and user-defined blocks. It employs a highly efficient BDD-based path modeling method and an associated ATPG technique. Experimental results show that it can robustly test selected paths without using extra logic, and protects the intellectual property.

The decision diagram based approach proposed in [19] elaborates identification of critical primitive path delay faults without path enumeration. Their work defines critical primitive path delay faults and presents an exact algorithm to identify them, using zero-suppressed binary decision diagrams and newly introduced operators necessary for handling multiple path delay faults. They report the number of critical primitive path delay faults for various criticality thresholds under the bounded delay model. Their result indicate that only a small but still necessary, number of multiple (primitive) faults, which escape testing under the singly testable fault criterion, must be considered in order to guarantee the timing correctness of the circuit.

The other work presented in [20] reveals binary decision diagrams and beyond: enabling technologies for formal verification. Ordered Binary Decision Diagrams (OBDDs) have found widespread use in CAD applications such as formal verification, logic synthesis, and test

generation. OBDDs represent Boolean functions in a form that is both canonical and compact for many practical cases. They can be generated and manipulated by efficient graph algorithms. Researchers have found that many tasks can be expressed as series of operations on Boolean functions, making them candidates for OBDD-based methods. The success of OBDDs has inspired efforts to improve their efficiency and to expand their range of applicability. Techniques have been discovered to make the representation more compact and to represent other classes of functions. This has led to improved performance on existing OBDD applications, as well as enabled new classes of problems to be solved. Their work provides an overview of the state of the art in graph-based function representations. They focus on several recent advances of particular importance for formal verification and other CAD applications.

2.2 Path delay Fault Models and Techniques for SOC Based Systems

With rapid strides in Semiconductor processing technologies, the density of transistors on the die is increasing in line with Moore's law which in turn is increasing the complexity of the whole SoC design. With manufacturing yield and time-to-market schedules crucial for an SoC(System on Chip), it is important to select verification and analysis solutions that offer the best possible performance, while minimizing iteration time and data volume.

The method proposed in [21] focuses on testing the path delay faults of a microprocessor using its instruction set. They propose that if a test can be applied in the normal operations of a circuit, they refer to it as a functional test. A structural test for a path is a test which can detect a fault on the path through the (enhanced or standard) full-scan chain. They propose a methodology that can identify a tight superset of the set of functionally testable paths of a microprocessor. Existence of delay defects on the functionally untestable paths will not cause any chip failure unless the delay defect causes the path delay to exceed twice the clock period. They stated that if a delay defect causes the path delay to exceed twice the clock period, it will most likely is caught by tests for transition faults, functionally untestable faults do not need to be targeted for path delay fault testing. Elimination of functionally untestable paths can significantly help identifying achievable path delay fault coverage and also reduce the computation effort of test generation. To identify the functionally testable paths in a microprocessor, they consider the data path logic

and the controller of the microprocessor separately. Path classification is performed by extracting a set of constraints capturing correlations among input/output (IO) signals and registers/flip-flops of the data path logic and the controller. Their experimental results show that a large percentage of the total number of paths is functionally untestable.

The other work in [22] proposed software-based delay fault testing of processor cores. Software-based self-testing is a promising approach for the testing of processor cores which are embedded inside a System-on-a-Chip (SoC), as it can apply test vectors in functional mode using its instruction set. Their work presents software based self-testing methodology for delay fault testing. Delay faults will affect the circuit functionality only when it can be activated in functional mode. A systematic approach for the generation of test vectors, which are applicable in functional mode, is presented. A graph theoretic model (represented by IE-Graph) is developed in order to model the data path. A finite state machine model is used for the controller. These models are used for constraint extraction under which a test can be applied in functional mode. Their approach uses instruction set architecture, RT level description along with gate level net list for test generation.

Mariusz Wegrzyn, Franc Novak, Anton Biasizzo [23] proposed functional testing of processor cores in FPGA-based applications. Embedded processor cores, which are widely used in SRAM-based FPGA applications, are candidates for SEU (Single Event Upset)-induced faults and need to be tested occasionally during system exploitation. Verifying a processor core is a difficult task, due to its complexity and the lack of user knowledge about the core-implementation details. In user applications, processor cores are normally tested by executing some kind of functional test in which the individual processor's instructions are tested with a set of deterministic test patterns, and the results are then compared with the stored reference values. For practical reasons the number of test patterns and corresponding results is usually small, which inherently leads to low fault coverage. They develop a concept that combines the whole instruction-set test into a compact test sequence, which can then be repeated with different input test patterns. That improves the fault coverage considerably with no additional memory requirements. Their proposed approach of functionally testing processor cores produces compact test sequences that are suitable for built-in self-test implementations. The test sequence, organized in a sensitive data path, can be repeated several times, each time with a different input test pattern, which increases

the probability of detecting faults. Relatively high initial fault coverage can be obtained if the bijective rule is applied at the level of sub-sequences of the instruction sequence. In some cases, fault coverage can be improved by interchanging the order of the sub-sequences. The concept of a data-sensitive path with a bijective property can be formalized, which opens up the possibility of an automatic test-sequence generation. Modeling the faults in an HDL description and the corresponding fault-injection process is another issue that can be further elaborated. In particular, a well-structured HDL description allows an algorithmic identification of the parts of the code that, when properly modified, model SEU-induced faults. Again, the fault-injection process can be made automatic for individual classes of faults. An evaluation case study of a functional test for the Pico Blaze processor core was performed on a generalized fault model, including both stuck-at faults and functional faults in LUTs, which are more difficult to detect. The achieved fault coverage confirms the efficiency of the proposed approach.

The work presented in [25] presents another methodology for at-speed testing of SOC ICs. Their work discusses the aspects and associated requirements of design and implementation of at-speed scan testing. It also demonstrates some important vector generation and implementation procedures based on a real design. They present an innovative method of scan pattern timing creation based on the results from Static Timing Analysis. They also describe the usage of a clock control module on J750 tester, which creates fast clock by combining two tester channels with high edge placement accuracy. These methods allow a short test pattern preparation time and the use of low-cost test equipment, while providing the high quality at-speed testing. High application frequencies, automated timing closure, new deep submicron (DSM) effects (dominance of wire delay, crosstalk, etc.), and finally the decrease of the absolute slack on the critical paths require testing of the IC at application frequency. On the other hand, the test cost must be kept on a minimum involving low cost tester solutions. Additionally, due to the short time-to-market window, all the tasks must be fulfilled just or even before the first prototypes arrive from the wafer fab. They present a possible solution to this challenge: scan based ATPG testing at application speed (at-speed).

2.3 BIST Based Methods

Due to the high cost of failure, verification and testing now account for more than half of the total life time cost of an integrated circuit (IC). Increasing emphasis needs to be placed on finding design errors and physical faults as early as possible in the life of a digital system, new algorithms need to be devised to create tests for logic circuits, and more attention should be paid to synthesis for test and on-line testing. On-line testing requires embedding logic that continuously checks the system for correct operation. Built-in self-test (BIST) is a technique that modifies the IC by embedding test mechanisms directly into it. BIST is often used to detect faults before the system is shipped and is potentially a very efficient way to implement on-line testing. The work proposed in [26], recommend a fault diagnoses structure for revealing of any software or hardware or permanent failures in the embedded read only memories. BIST controller, along with row selector and column selector is designed to meet necessities of at speed test thus enabling detection of timing defects. The projected approach offers a simple test flow and does not require intensive communications between a BIST controller and a tester. The system rests on partitioning of rows and columns of the memory array by employing low cost test logic. It is intended to meet requirements of at-speed test thus enabling detection of timing defects. It reduces the diagnostic data that needs to be scanned out during ROM test such that the minimum information to recover the failure data is preserved, and the time to unload the data is minimized. The presented approach allows an uninterrupted collection and processing of test responses at the system speed. This has been achieved by using low-cost on-chip selection mechanisms, which are instrumental in very accurate and time efficient identification of failing rows, columns, and single memory cells. In particular, the scheme employs the original designs of row and column selectors with phase shifters controlling the way the address space is traversed. Furthermore, the new combined selection logic allows the scheme to collect test results in parallel (leading to shorter test time) without compromising quality of diagnosis. Results of experiments performed on several memory arrays for randomly generated failures clearly confirm high accuracy of diagnosis of the scheme provided the signature registers and the proposed selection logic are properly tuned to guarantee a desired diagnostic resolution.

The work in [27] proposes online BIST for embedded systems. Makers of embedded systems take many measures to ensure safety and reliability throughout the lifetime of products

incorporating the systems. Here, they consider techniques for identifying faults during normal operation of the product—that is, on-line-testing techniques. They evaluate them on the basis of error coverage, error latency, space redundancy, and time redundancy. Since embedded systems are frequently components of mobile products, they are exposed to vibration and other environmental stresses that can cause them to fail. Embedded systems in automotive applications are exposed to extremely harsh environments, even beyond those experienced by most portable devices. The authors focus on digital hardware testing, including techniques by which hardware tests itself, built-in self-test (BIST). Nevertheless, they consider the role of software in detecting, diagnosing, and handling hardware faults.

The other work presented in [28] presents a complete and versatile online test solution based on reconfigurable test architecture. Reconfigurable test architecture works alongside the controllers for online concurrent fault detection. The output vectors of the controllers are concurrently monitored and any fault present is detected in a few cycles from the sensitization of the fault. The architecture is then reprogrammed to a similar set of diagnostic hardware to locate a sub block which is the cause for the fault. The same architecture is then reprogrammed to replace the faulty block thereby completing repair. The test architecture is designed based on configurable logic blocks. They proposed that since fault detection, diagnosis and repair are completed online with one test hardware, the effective hardware overhead is negligible and the system can resume its function within a brief period. The applicability of the architecture is demonstrated for the control blocks in OC8051. They also proposed that diagnosis has been almost completely offline, since online diagnosis is expensive and the online diagnostic resolution achieved has been very low. Moreover, there is nothing much one can do after online diagnosis because repair of logic blocks has again been an almost impossible task. One method of diagnosis is using external hardware or a reconfigurable FPGA connected to the chip, which is a very long and tedious process. The paper also presents a test repair scheme i.e. once a faulty sub block is identified, the corresponding switch matrices are programmed to isolate the sub block from the controller and the inputs are rerouted to the test architecture. They proposes that to support the self-testing methodology, the IP core lies inside a test wrapper containing test support logic to control scan chain shifting, buffers to store scan data and support at-speed test, and so on. In this example, the on-chip bus is a shared bus to which the arbiter controls access. They proposed that tests patterns could be applied to the I/O of a core in parallel. Parallel access has a high routing penalty and

wiring overhead; serial access has a higher test time and might intro-duce arbitrary switching during the scan operation. Parallel access' routing penalty and wiring overhead increase as designs become denser. A reasonable addition of test logic over-head can improve penalties associated with serial access without wiring overhead of parallel access method; therefore, serial access method is used for test pattern application. Not all test data can be loaded via the scan-in operation. Therefore, from a test perspective, the core ports can be divided into ports that should be controlled in parallel—called parallel test ports (such as test clock and other control signals)—and ports that can be accessed in series, or serial test ports(such as functional data ports). If necessary, functional data ports can be added to parallel test ports and accessed accordingly.

The method proposed in [29] presents at-speed testing strategies for high-performance microprocessors. In a synchronous sequential circuit, the cause of a timing failure can be excessive propagation delay through combinational logic connected to a storage element—either a flip-flop or a latch. (For ease of presentation in this article, they refer to these storage elements as flip-flops.) If a logic net-work's output is not stable by the flip-flop's setup time, the network violates the long path timing constraint. This is called a setup time violation, the focus of this article. Alternatively, if the propagation delay through combinational logic is too short, circuit outputs can start changing before the flip-flop's hold time, violating the short path timing constraint. This is called a hold time violation, and it too can cause a timing failure. Unfortunately, design constraints typically prevent data collection across a full spectrum of frequencies. Most designs have a minimum operating frequency, often determined by the on-chip phase-locked loop circuit. Designs operating at gigahertz speeds might not be operable or testable at frequencies below several hundred megahertz. They generally categorize units failing tests at a design's minimum operating frequency as speed insensitive. This is somewhat arbitrary, and it may be that some failing parts could have passed an even slower test were one possible. However, they propose that if they must draw a line between delay defects and gross defects, the design's minimum operating frequency is a reasonable choice.

2.4 Other Delay Fault Analysis Methods on VLSI

The core based testing method of VLSI circuits proposed by H.J. Vermaak and H.G. Kerkhoff [30] states the increased usage of embedded cores necessitates a core-based test strategy in which cores are also tested separately. The IEEE P1500 proposed standard for Embedded Core Test (SECT) is a standard under development which aim is to improve the testing of core-based system chips. This work deals with the enhancement of the Test Wrapper and Wrapper Cells to provide a structure to be able to test embedded cores for delay faults. This approach allows delay fault testing of cores by using the digital oscillation test method and the help of the enhanced elements while staying compliant to the P1500 standard. They stated that design methodologies for VLSI system-on-a-chip (SOC) circuits based on reusable predesigned circuits, which are variously called intellectual property (IP) circuits or cores are becoming very popular. Verifying that such designs meet their performance objectives is difficult due to the hidden implementation details of the IP circuits. Delay fault testing of these cores is therefore difficult as access to these cores is not always available. The testing of cores is receiving a lot of interest nowadays due to the popularity of SOCs. The IC design community is divided into two groups: the core providers and the core users. The providers supply these cores as fault-free blocks to the user who is only concerned with the design, manufacturing and testing of his own system. The cores are provided as soft, firm or hard-core, not manufactured and therefore not tested. This leaves the testing of the core to the user after manufacturing the system. The user would therefore require assistance from the provider by delivering pre-defined tests with the core. In this environment the IEEE P1500 SECT was developed. It intends to facilitate the testing of embedded cores as separate entities. The IEEE P1500 does not cover the core's internal test methods or SOC test integration and optimization. The IEEE P1500 is targeted at testing "black-box" third party cores. The implementation details of the cores are hidden from the core user and it is mandatory that the core providers deliver tests with these cores to ensure that testing of these cores can be carried out.

The other work proposed in [31] is the oscillation test methodology for built-in analog circuits. Their article aims to describe the fundamentals of analog and digital testing methods to analyze the difficulties of analog testing and to develop an approach to test the analog components in a

mixed signal circuit environment. Oscillation based, built-in self-test methodology for testing analog components in mixed-signal circuits, in particular, is discussed. A major advantage of the OBIST method is that it does not require any complex response analyzers and test vector generators which are costly Furthermore, since the oscillation frequency is considered to be digital it can be easily interfaced to test techniques dedicated to the digital part of the circuit under test (CUT). OBIST techniques show promise in detecting faults in mixed signal circuits and requires little modification of the CUT to improve the fault coverage. Extensive simulation results on some sample analog benchmark circuits are described in Spice format.

The level-sensitive scan design test method proposed in [32] presents a technology-independent test synthesis tool extends the basic level-sensitive scan design (LSSD) boundary scan methodology. It reuses functional storage elements wherever possible and introduces minimal test logic overhead and delay. Embedded memories and reusable cores have become common because they reduce the design time to market for complex systems. Scan design is the most commonly practiced approach to enhancing design testability. This approach lets design storage elements be configured into shift registers during test mode, thereby enhancing the logic's controllability and observability. The LSSD boundary scan approach uses the LSSD test protocol. In LSSD, storage elements are designed as shift register latches (SRLs). A shift register is designed either as a double-latch configuration for functional flip-flops or as a single-latch configuration for transparent latches.

Another paper proposed in [33], proposes a method for testing a circuit in order to improve defect coverage of delays due to resistive open and close. The proposed method uses a signature analysis and a scan design to detect small delay defects. This measures the delay of the explicitly sensitized paths with the resolution of the on-chip variable clock generator. The proposed scan design measures the small delay in short measurement time by delay measurement technique and extra latches for storing the test vectors. By evaluating with Rohm 0.18-μm process shows that the measurement time is 67.8% reduced compared with that of the delay measurement with standard scan design and the area overhead is larger than that of the delay measurement architecture using standard scan design.

The other paper proposed in [34] proposed functional test generation for delay faults in combinational circuits. They propose a functional fault model for delay faults in combinational

circuits and describe a functional test generation procedure based on this model. The proposed method is most suitable when a gate-level description of the circuit-under-test, necessary for employing existing gate-level delay fault test generators, is not available. It is also suitable for generating tests in early design stages of a circuit, before a gate-level implementation is selected. It can also potentially be employed to supplement conventional test generators for gate-level circuits to reduce the cost of branch and bound strategies. A parameter called Δ is used to control the number of functional faults targeted and thus the number of tests generated. If Δ is unlimited, the functional test set detects every robustly testable path delay fault in any gate-level implementation of the given function. An appropriate sub-set of tests can be selected once the implementation is known. The test sets generated for various values of Δ are fault simulated on gate-level realizations to demonstrate their effectiveness.

Even if all the works reviewed here are exhaustive studies on path delay fault modeling, test generation and analysis, but none of them use buffer insertion technique of delay modeling, ROBDD for delay fault analysis and the reduced fault and reduced test vector idea in to consideration. This paper, "Identification of High-Risk Path-Delay Hardware Faults Locations and Evaluation of Their Impact" primarily focuses on the verification of high risk path delay faults of combinational circuits. And this work also aims to reduce processing power needed for path delay fault simulation using reduced ordered binary decision diagrams and fault reduction of any kind i.e. it may be reduction of equivalent faults, gate oriented or line oriented fault reduction of the stuck at fault model.

2.5 Summary

In this chapter we made a review on some notable and recent research work done on path delay fault modeling, test and simulation for a variety of cases. Presenting all proposed methods on this paper is beyond a scope. This is due to the task to model and present very effective path delay fault model, analysis them using tools and developing algorithms is the most active research area and researchers have been publishing papers in alarming rate. Because of the availability of the Moore's law trend on VLSI fabrication from then till now, currently there is no universally accepted delay fault model or analysis framework that represents all available hardware defects that cause delay.

Chapter Three

3. Delay Fault Modeling

3.1. Introduction to Fault and Defect Modeling

A model of a physical object or model of a phenomenon is a representation of the object or phenomenon that is used for the specific purpose of analyzing the behavior of the object, studying the phenomenon, or studying the effect of the object or phenomenon on its environment or surroundings. A computer simulation program uses a model. The information we obtain from running a simulation program depends on the model that is used for simulation. For example, a model of a circuit can be developed for predicting its temperature radiation, its logical behavior, or its behavior when the circuit becomes faulty [5].

Just as the binary logic value system, containing 0 and 1, is used as a simplified model for complex line values in digital systems, a simplified value system is needed to model faults on circuit lines. Such a fault model should be simple and be able to facilitate analysis of faulty behavior of a digital system [35].

This chapter discusses fault models and issues that are related to fault analysis and simulation of a circuit that is faulty. We try to set a solid background in understanding faults and circuit fault models, as this background is crucial in understanding the materials in the rest of this paper. In this chapter we present the proposed delay fault model to analysis the behavior of combinational circuits and its faulty counterpart.

Definition of Terms:

A defect in an electronic system refers to a flaw in the actual hardware. A fault on the other hand, is a representation of a defect and is used in computer programs for analyzing defects in electronic components [1].

An error is caused by a defect, and it happens when a defect in hardware causes a line or a gate output to have a wrong value. In a computer simulation program, an error is defined as an observed fault.

A failure occurs when a defect causes misbehavior in the functionality of a system that cannot be reversed or recovered. In a computer simulation program, change in the intended functionality of a system due to an existing fault is referred to as a failure. In order to be able to predict the behavior of a defective system, or generate tests to find the defective parts, or in general, for any type of computer analysis of defects, a good fault model is needed.

A fault model is one that has a close correspondence with the actual defects, it is easy to represent in a computer program, and is as brief as possible for an optimized computer processing time [3].

3.2. Delay Fault Models

A delay in a circuit is defined as a failure for signals to propagate from a source node to a destination node within the specified time. If a typical source node is a sequential element like a flip-flop, and a typical destination node is also a flip-flop, then it is easy to see that the "specified time" is the time between clock pulses. The first clock pulse, typically called the launch pulse or release pulse, begins the propagation of the signals through connecting logic to the destination. The second clock pulse, called the capture pulse, defines the time at which these signals should have completed their function, and arrived at the destination sequential element [36].

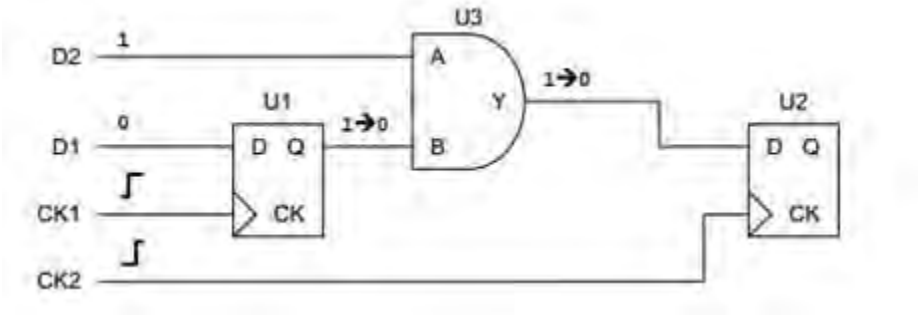


Figure 3.1 depicts a typical group of logic for illustration of PDF [16]

In Figure 3.1 CK1 is called the launch clock. U1/Q is called the launch node. It is, in this case, initialized to one (1). This one along with any other initial constraints is sometimes referred to as V1 or T1. It is the initial state of the signals involved in the test prior to the launch clock. D1 is sensitized to cause a transition to occur at U1/upon arrival of the launch clock, CK1. D1, then, is set to apply V2 or T2. After the launch clock is applied, an at-speed time, or delay, is made

before the application of the capture clock, CK2. The transition arrives at the capture node, U2/D. If the zero (0), in this case, is captured into U2 by CK2, then the test passes. If a one (1), in this case, is captured into U2, then the test fails. Note that for the transition to propagate through the combinational logic, U3, a one (1) must be held on U3/A by setting D2 to one (1). U3/A is called an off-path or side input. These are signals that may serve to sensitize the path of interest, but are not actually on the path of interest [12].

A passing test indicates that there is no defect causing a delay larger than the cycle time of the path. A failure of the test indicates a defect resides along the path. These defects are modeled as faults. A slow-to-rise (STR) fault is a failure for a low-to-high transition to propagate from the launch node to the capture node in the specified time. A slow-to-fall (STF) fault is detected if the high-to-low transition fails to propagate between the launch and capture clocks.

Different delay fault models are considered: transition fault model, gate delay fault model, path delay fault model, segment delay fault model and line delay fault model. It is assumed that each gate can have an arbitrary fall (rise) delay from each input to the output pin. Also, interconnect are assumed to have arbitrary rise (fall) delays. Since the gate pin-to-pin delays and the interconnect delays can be combined together. Transition, gate and line delay models are used for representing delay faults lumped at gates while the path and segment delay model address faults that are distributed over several gates. The advantages and disadvantages of each model are discussed.

3.2.1 Transition Fault Model

Transition fault model [37] assumes that the delay fault affects only one gate in the circuit. There are two transition faults associated with each gate a slow-to-rise fault and a slow-to-fall fault. It is assumed that in the Fault free circuit each gate has some nominal delay. Delay faults result in an increase or decrease of this delay. Under transition fault model, the extra delay caused by the fault is assumed to be large enough to prevent the transition from reaching any primary output at the time of observation. In other words, the delay fault can be observed independent of whether the transition propagates through a long or a short path to any primary output.

3.2.2 Gate Delay Fault Model

Gate delay fault model [38][39] assumes that the delay fault is lumped at one gate in the circuit. However, unlike the transition model gate delay fault model does not assume that the increased delay will affect the performance independent of the propagation path through the fault site. It is assumed that long paths through the fault site might cause performance degradation. Gate delay fault model is a quantitative model since it takes into account the circuit delays. The delays of the gates are represented as intervals. The gate delay fault model has the following characteristics.

1. The delay through a gate depends on the logic values applied to the gate.
2. Multiple copies of a gate have different delays due to manufacturing variations.
3. A gate has some inertia in responding to changes at its inputs. Transients of short duration at the gate inputs get altered out from the response at the output.

Taking the timing into consideration when deriving tests for gate delay faults, allows application of some tests that would otherwise not be considered. The limitations of the gate delay fault model are similar to those for the transition fault model. Namely because of the single gate delay fault assumption a test may fail to detect delay faults that are result of the sum of several small delay defects. The main advantage of this model is that the number of faults is linear in the number of gates in the circuit [38].

3.2.3. Path Delay Fault Model

Under path delay fault model [39] a combinational circuit is considered faulty if the delay of any of its paths exceeds a specified limit. A path is defined as an ordered set of gates {g0, g1…, gn} where g0 and gn are a primary input and primary output, respectively. Also, gate gi is an input to gate gi+1where ($0 \leq i \leq n-1$). A delay defect on a path can be observed by propagating a transition through the path. Therefore a path delay fault specification consists of a physical path and a transition that will be applied at the beginning of the path. The delay or length of the path represents the sum of the delays of the gates and interconnections on that path.

Tests for the path delay fault model can detect small distributed delay defects caused by statistical process variations. A major limitation of this fault model is that the number of paths in the circuit can be very large (possibly exponential in the number of gates). For this reason testing all path delay faults in the circuit is not practical. Two strategies are commonly used for selecting the set of path delay faults for testing.

One is to select a minimal set of paths such that for each signals in the circuit the longest path containing those selected set of paths is selected for testing [7]. The other is to select all paths with expected delays greater than the specified threshold. The reason behind selecting the longest paths is that the delay defects on shorter paths might not be large enough to affect the circuit performance. Also if the defects on short paths are large and could affect the performance, one expects that such defects would be detected by other tests (e.g. at speed tests and gate delay tests) that precede the path delay fault testing. This strategy might work for circuits whose paths have very different delays so that there are a small percentage of long paths. However, often in performance optimized designs almost all paths have long delays and in these circuits not even all longest paths can be tested[9][34].

The Path Delay Fault (PDF) model has been long considered as the most accurate one among the various delay fault models, due to its ability to detect both lumped as well as small distributed delay defects. The PDF set consists of single as well as multiple faults (a fault that is not singly detectable may be detected when tested as part of a multiple fault). A lot of effort has been devoted in the past on the quality of the generated tests for PDFs. Single PDFs are tested using robust tests, if these exist, otherwise non-robust tests are used. Multiple faults (also referred to as functionally sensitized faults in the literature) are tested using non-robust tests [6]. The primitive PDF model, which is a refinement of the traditional PDF model, limits significantly the number of multiple faults needed to be tested (a non-primitive fault is always covered by a corresponding primitive one) and, besides in delay testing, it has also been shown to apply in timing verification and timing analysis [10], [40].

3.2.4. Line Delay Fault Model

Line delay fault model [39] tests a rising (falling) delay fault on a given signal (line) in the circuit. The fault is propagated through the longest sensitizable path passing through the given

line. Similar to transition and gate delay fault models, line delay fault model assumes a single delay fault. Therefore, the number of faults equals twice the number of lines in the circuit. Sensitizing the longest path through the target line allows detecting the delay fault of the smallest size on the target line. In general, a test will cover several line delay faults. Therefore, this fault model can also detect some distributed delay defects on the propagation paths. However, since only one propagation path through each line is considered, this model can fail to detect some distributed defects.

Table 3.1 Delay test generation methods and fault models [16].

Method	Definition	Pros	Cons	Use
Functional	"Handmade " pattern set exercising the true function of a component	-uses real function -May leverage existing simulation pattern sets	-Difficult to construct -Difficult to measure coverage -May not integrate well with ATE	-speed binning -minimum specification determination and validation
Path delay[41]	Pattern exercising a specific circuit node	-path of transition well defined -good for distributed / small defect detection	-Exponential fault list growth	-speed binning -process monitoring -I/O characterization
Transition(Gate, Line) Delay[41]	Pattern exercising a specific circuit node	-Good for gross defect detection -Easier test generation -Linear fault list growth -Geographically diverse coverage	- May not address subtle delay defects	-Broad defect coverage - Process Monitoring
Segment Delay [3]	Pattern exercising a smaller segment of a path	-More linear growth of fault list compared to path - Easier test Generation	- No tool automation support - Benefit not well studied	-Impractical solution without automation

3.2.5. Slow to Fall and Slow to Rise Delay Faults(Stuck-at Faults)

Stuck at faults are stuck-at-0 and stuck-at-1 which we will refer to as SA0 and SA1 or slow to rise (STR) and slow to fall (STF) faults respectively. These faults account for many transistor level faults and can model other fault types [1].

The stuck-at fault is a logical fault model that has been used successfully for decades. A stuck-at fault affects the state of logic signals on lines in a logic circuit, including primary inputs (PIs), primary outputs (POs), internal gate inputs and outputs, fan-out stems (sources), and fan-out branches. A stuck-at fault transforms the correct value on the faulty signal line to appear to be stuck at a constant logic value, either logic 0 or logic 1, referred to as stuck-at-0 (SA0) or stuck-at-1 (SA1), respectively. Consider the example circuit shown in Figure 3.2, where the eight signal lines representing potential fault sites are labeled alphabetically. There are 16 (2×8) possible faulty locations under the single-fault assumption. It should be noted that, rather than a direct short to a logic 0 or logic 1 value, the stuck-at fault is emulated by disconnection of the source for the signal and connection to a constant logic 0 or 1 value. SA0 on fan-out branch line d behaves differently from SA0 on fan-out branch line e, while the single SA0 fault on the fan-out source line b behaves as if both fan-out branches line d and line e are SA0.

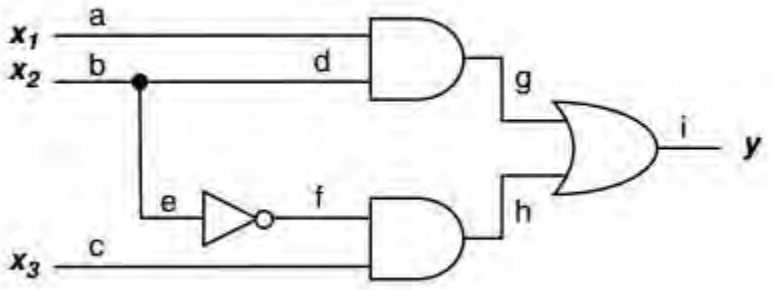


Figure 3.2 Example circuit to demonstrate a stuck at fault

In physical testing of circuits, we assume that circuits are tested frequent enough that faults are detected and identified as they occur, before we run into multiple fault situations. Although this

assumption may be unrealistic for complete systems, it is a realistic situation for sub modules of a system tested independently.

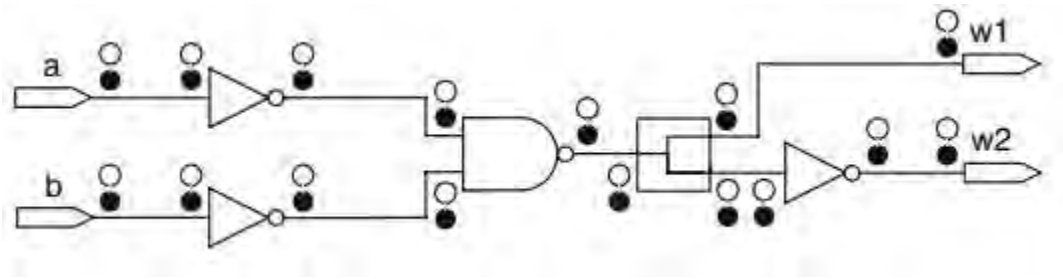


Figure 3.3 Stuck at faults in s gate level circuit [5]

Where o= is stuck at zero (slow to rise fault)

•= stuck at one (slow to fall fault)

Table 3.2:- Stuck-at faults in s gate-level circuit

	Gate	Ports	Faults	Total faults
Inverter	3	2	2	12
PI	2	1	2	4
PO	2	1	2	4
Fan-out	1	3	2	6
NAND	1	3	2	6

Total fault sites= 16

Total faults = 32

3.3. Detecting Single Stuck-at Faults

For detecting a fault, finding Boolean function of the faulty circuit and comparing it with the good circuit function, applies to functional faults and structural faults of any type. However, a simpler method exists for detecting single stuck at faults that we explain here.

A stuck at fault to be detected must be activated and then propagated. To activate a fault, primary input values must be adjusted such that the faulty line receives a value different than its faulty value. Once a fault is activated, input values must be adjusted such that the fault effect can

propagate to a primary output. A fault effect on a line propagates to a primary output when toggling the value of line also toggles the output [42].

As an example, consider the SA1 fault on I_4 of Fig. 3.4. In order to activate this fault, a 0 must reach this line (I_4= 0/1, 0: good value, 1: faulty value). This can be achieved by s = 1. Now that I_4:SA1 is activated; it must be propagated to the output. For this to happen, I_5 must become 1, so that faulty value of l_4 propagates to the output of G_2. Thus a = 1.

This makes I_7= 1/0. For l_7 output of $G_2$to propagate to l_9, and thus w, I_8must be 1; otherwise, l_8= 0 forces l_9= 1 regardless of value of l_7. Making I_8= 1 can be achieved by a 0 on b. This makes I_9= 0/1, which shows that w becomes 0 if there is no fault in the circuit, and it becomes 1 in presence of I_4:SA1. The input combination abs = 101 detect I_4:SA1.

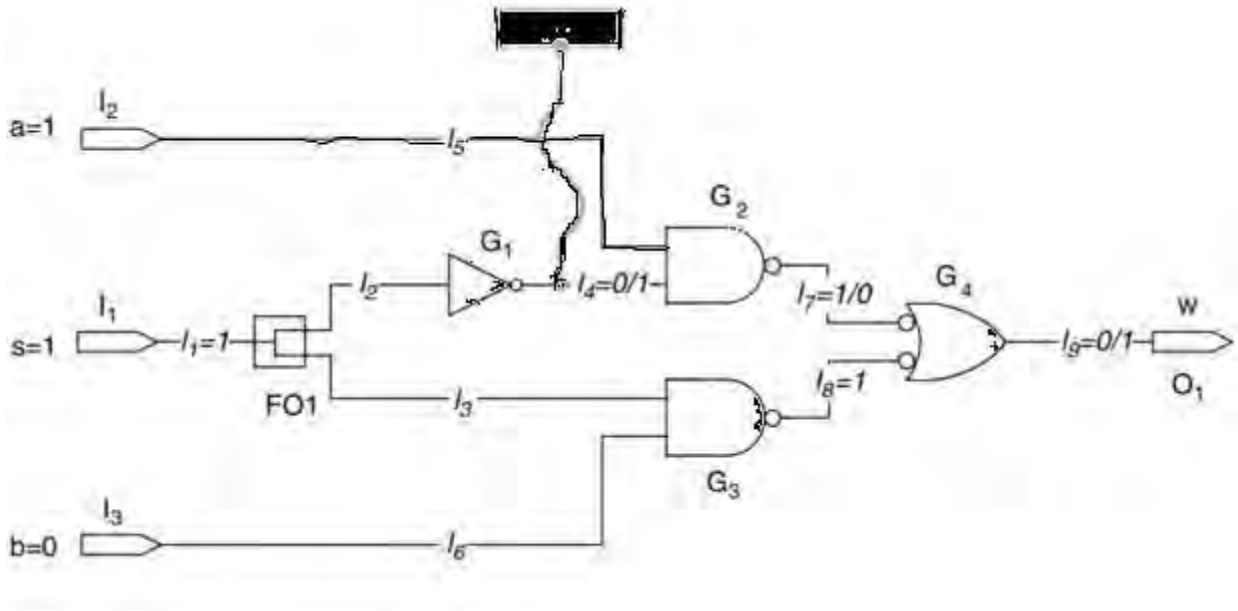


Figure 3.4 Detecting stuck-at faults [1]

3.3.1. Boolean Difference as a Test Vector Generator

Consider the circuit shown in Figure 3.5. Let the target fault be the stuck-at-0 fault on primary input y. In the example circuit shown in Figure 3.5, the faulty circuit part is y stuck at 0.Note that the circuit output can be expressed as a Boolean formula:

$$f = xy + y\hat{}z$$

Let f^ be the faulty circuit with the fault y/0 present. In other words,

$$f\hat{} = f(y = 0)$$

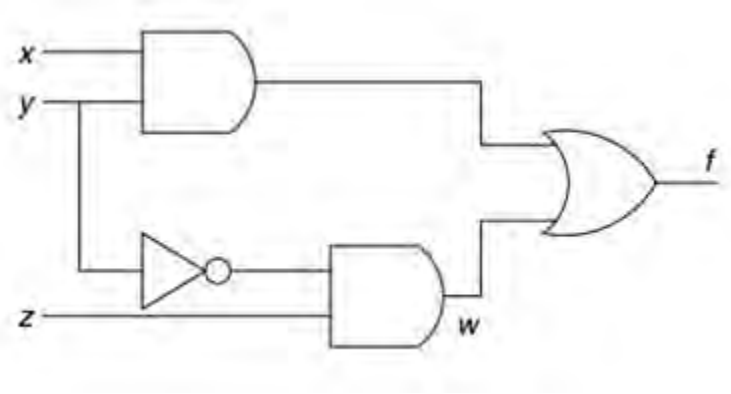


Figure 3.5 Example circuit to illustrate the concept of Boolean difference

In order to distinguish the faulty circuit f from the fault-free counterpart f, any Input vector that can make f⊕f^=1 would suffice. Furthermore, as the aim is test generation, the target fault must be excited. In this example, the logic value on primary input y must be logic 1 to excite the fault y/0. Putting these two conditions together, the following equation is obtained:

$$y.f(y = 0) \oplus \mathrm{f}(\mathrm{y} = 0) = 1 \quad ---------- \quad 3.1$$

Note that f(y=1) ⊕f(y=0) indicates the exclusive-or operation on the two functions f(y=1) and f(y=0); it evaluates to logic 1 if and only if the two functions evaluate to opposing values. In terms of ATPG, this is synonymous to propagating the fault effect at node y to the primary output f. Therefore, any input vector on primary inputs x, y, and z that can satisfy Equation (3.1) is a valid test vector for fault y/0:

$$y \cdot f(y=1) \oplus f(y=0) = y \cdot (x \oplus z)$$

$$= y \cdot (xz\hat{} + x\hat{}z)$$

$$= xyz\hat{} + x\hat{}yz$$

In this running example, the two vectors xyz= {110, 011} are candidate test vectors for fault y/0.Formally, f(y=1) ⊕f(y=0) is called the Boolean difference of with respect to y and is often written as:

$$\frac{df}{dy} = f(y = 1) \oplus f(y = 0)$$

In general, if f is a function of $x_1, x_2, \ldots\ldots\ldots, x_n$, then

$$\frac{df}{dxi} = f(x1, x2, \ldots\ldots, xi, \ldots\ldots\ldots, xn) \oplus f(x1, x2, \ldots\ldots xi' \ldots\ldots xn) \quad ---------3..2$$

In terms of test generation, for any target fault on some fault α /v, the set of all vectors that can propagate the fault-effect to the primary output f is then those vectors that can satisfy:

$$\frac{dy}{d\alpha} = 1$$

(Note that this is independent of the polarity of the fault, whether it is stuck-at-0 or stuck-at-1.) Next, the constraint that the fault must be excited, set to value, must be added. Subsequently, the set of test vectors that can detect the fault becomes all those input values that can satisfy the following equation:

$$(a = v^{-}).\frac{df}{da} = 1 \quad ----------3.3$$

Consider the same circuit shown in Figure 3.5 again. Suppose the target fault is w/0. The same analysis can be performed for this new fault. The set of test vectors that can detect w/0 is simply:

$$w.\frac{dy}{dx} = 1$$

$$=> w.\big(f(w = 1) \oplus f(w = 0)\big) = 1$$

$$w.(1 \oplus xy) = 1$$

$$w.\big(xy(bar)\big) = 1$$

$$w.(x\text{^}+y\text{^}) = 1$$

$$w.x\text{^} + wy\text{^} = 1$$

Now, w can be expanded from the circuit shown in the figure to be w=y^·z. Plugging this into the equation above gives us:

$$=>w.x\hat{} + w\,y\hat{} = 1$$

$$=>y\hat{}zx\hat{} + y\hat{}zy\hat{} = 1$$

$$=>x\hat{}y\hat{}z + y\hat{}z = 1$$

$$=> y'z = 1$$

Therefore, the set of vectors that can detect w/0 is {001,101}.

If the target fault is untestable, it would be impossible to satisfy Equation 3.3. Consider the circuit shown in Figure 3.6. Suppose the target fault is z/0. Then the set of vectors that can detect z/0 are those that can satisfy:

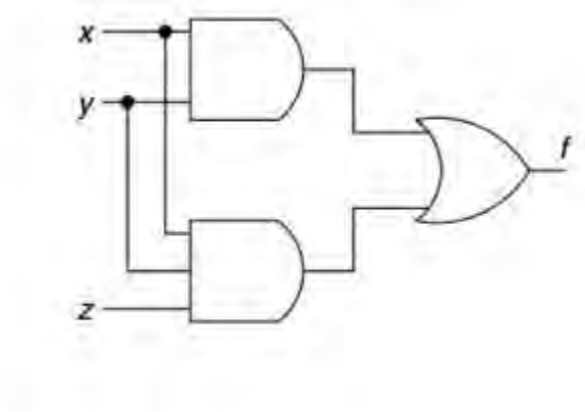


Figure 3.6 Example for an untestable Fault

$$z.\frac{df}{dz} = 1$$

$$z =>.\big(f(z=1) \oplus f(z=0)\big) = 1$$

$$=> z.(xy \oplus xy) = 1$$

$$=>z.0 = 1 = false => untestable$$

In other words, there exist no input vectors that can satisfy $z.\frac{df}{dz} = 1$, indicating that the fault z/0 is untestable.

But in this paper we have used the already generated test vector found in the ISCAS -85 bench mark circuit. In this bench mark each model circuit has its own test vectors. But for model verification, we have used a deterministic test vector application, which is a method of test vector generation by hand at the specified locations where test is conducted.

3.4. The proposed Delay Fault Model

A **fault model** is an engineering model of something that could go wrong in the construction or operation of a piece of equipment. From the model, the designer or user can then predict the consequences of this particular fault. Fault models can be used in almost all branches of engineering. As already stated above at chapter two, fault modeling and analysis is an open area of research [43].

Why fault models? I/O function tests are inadequate for manufacturing (functionality versus component and interconnect testing).Real defects (often mechanical) are too numerous and often not analyzable. A fault model identifies targets for testing i.e. model faults that are most likely to occur. Fault model makes analysis possible i.e. associate specific defects with specific test patterns. And lastly, fault model makes effectiveness measurable by experiments i.e. fault coverage can be computed for specific test patterns to reflect its effectiveness

The most common model used for logical fault is the single stuck-at fault (SSF). In this a fault in a logic gate gives a favorable outcome in one of its inputs or the output being fixed to either a logic 0(stuck-at-0) or a logic 1(stuck-at-1).

In this paper, we propose a stuck at delay fault model using buffer insertion technique. A buffer is a gate, typically two serially connected inverters, that regenerates a signal without changing functionality. Ideally adding a buffer in to a circuit line doesn't change the circuit's functionality.

Buffers can [43]:

i. Improve timing delays either by speeding up the circuit or by serving as delay elements and
ii. Modify transition times to improve signal integrity and coupling-

induced delay variation

Using this concept in mind, we propose the following delay fault model for combinational circuits. Let's consider one of the ISCAS'85 benchmark circuit c17 as an example to present the good circuit model and the faulty circuit model.

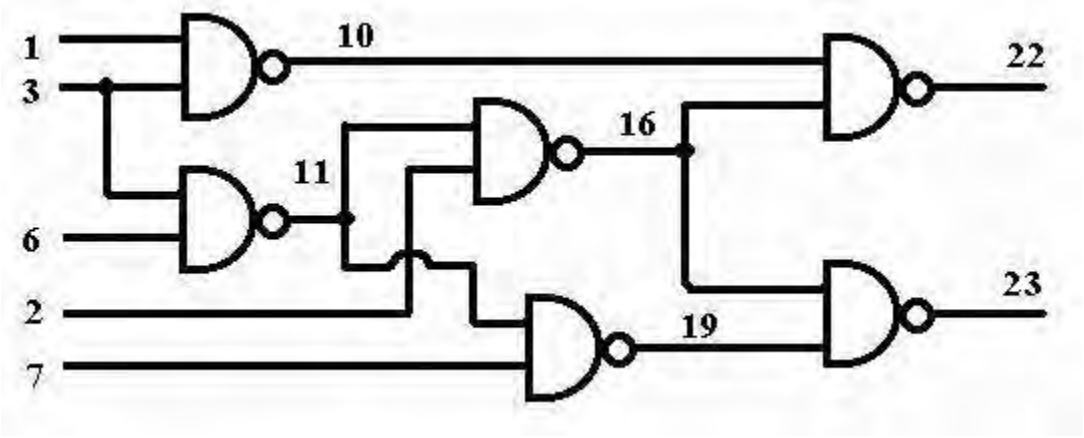


Figure 3.7 A gate level schematic of the c17 benchmark circuit (GUT)

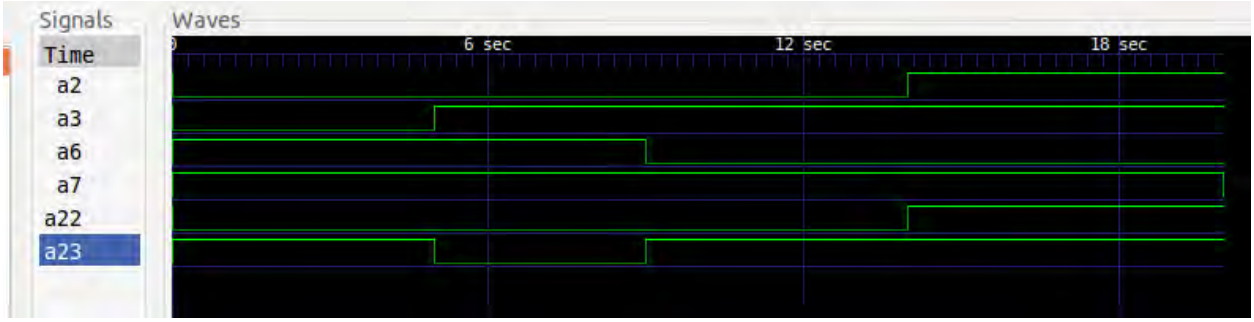


Figure 3.8 A graphical depiction of the simulation output of figure 3.7 which is fault free

The above RTL description is a fault free module. Faulty model is created such that the functionality will remain same as the fault free module but there is a delay in change in value i.e 1/0 transition delay which definitely produces a faulty output. This is done by inserting the buffer for the specified ports.

The following pseudo code shows fault insertion

Module Definition;

Interface Definition;

If (interface)

Interface=BUF; //apply Buf to the Gate Input

Interface =Gate;

While (interface=Gateoutput)

Interface=BUFFR;

Gateoutput=Interface; //assign the buffer to the gate output

While (endmodule)

Interface = Test;

Gateoutput=Interface //apply a test vector for stuck at fault location

end

Figure 3.9: Fault Injection Procedure for a stuck at fault application

To these faulty and fault free modules fault simulation is performed with the reduced number of test patterns for each of the faults. The outputs obtained in each case of the faulty circuits are compared with the output of the good circuit to determine which faults are detected, finally to obtain the fault coverage. A single pattern can detect many faults or a single fault. Many patterns can detect many faults or a single fault. The challenge in testing is to obtain a minimal number of test patterns which guarantees high fault coverage of a circuit with known set of faults. In order for an input vector X to detect a fault h s-a-D, D = 0, 1 the input X must cause the signal h in the normal (fault-free) circuit to take the value D^. Activate: Specify inputs so as to generate the appropriate value (0 for s-a-1, 1 for s-a-0) at the site of the fault. This test application can be manual (using deterministic test generation) or we can use Boolean difference (automatic test generation using ATPG) as described in section 3.3.1.And the output wave form is shown in figure 3.11 below. As we can observe from the graphs in figures 3.9 and 3.11, the difference in the two graphs is discernible. From fault free graph of 3.9, at the 0 clock time, the inputs a1,a3,a6,a2,a7 are 1,0,11,10,01 respectively and output a22=0, output a23=1. in contrast to this at the same clock cycle(zero), of faulty graph of figure 3.11, inputs a1,a3,a6,a2,a7 are all the same as above but the outputs a22_faulty=0, output a23_faulty=x. This shows that in the presence of delay fault even if there is a similar input condition we will get a faulty output. All the graph parts are analyzed in the same way.

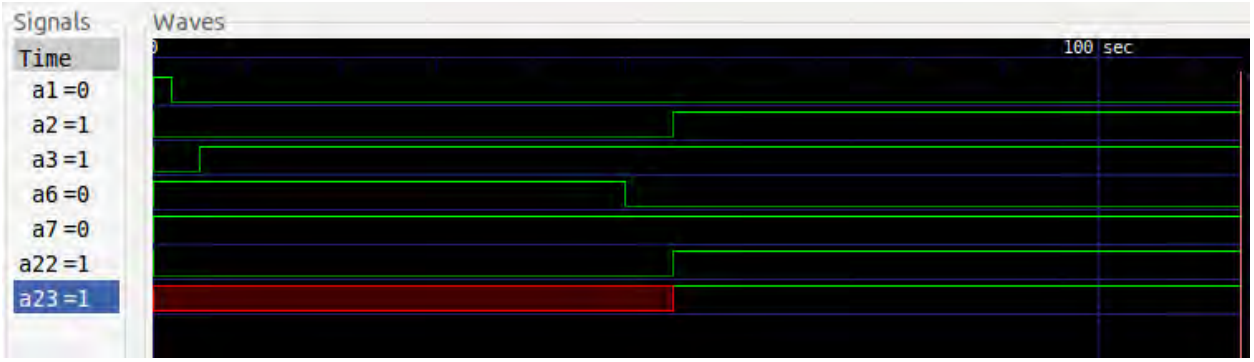


Figure 3.10: A faulty Circuit output of figure 3.7 using the algorithm in figure 3.9

To summarize the proposed delay fault model, we are going to present the fault free and faulty output of the C17 circuit in table 3.3 below.

Table 3.3: The functional simulation output of the GUT and FUT of the proposed model

Fault free circuit						Faulty circuit								
Time	a1	a3	a6	a2	a7	out put		a1	a3	a6	a2	a7	out put	
						a22	a23						a22	a23
0	1	0	1	0	1	0	1	1	0	1	0	1	0	X
2	0	0	1	0	1	0	1	0	0	1	0	1	0	X
5	0	1	1	0	1	0	0	0	1	1	0	1	0	X
9	0	1	0	0	1	0	1	0	1	0	0	1	0	X
14	0	1	0	1	1	1	1	0	1	0	1	1	1	1
20	0	1	0	1	0	1	1	0	1	0	1	0	1	1
120	0	1	0	1	1	1	1	0	1	0	1	0	1	1
240	0	1	0	1	0	1	1	0	1	0	1	0	1	1

From table 3.3, we can conclude that anybody who has a Verilog knowhow can understand the effect of the proposed delay fault model by applying on the different combinational circuits available.

Chapter Four

4. Delay Fault Simulation Methodologies

4.1. Introduction

Simulating a faulty model of a circuit is called fault simulation. This process is used by test and design engineers; it is the most used test method, and perhaps is one of the most computationally intensive test applications. This chapter is on fault simulation methodologies that are common in VLSI design and test [5].

One can use fault simulation for test data generation, test set evaluation, circuit testability evaluation, providing information for testers, finding faults in a circuit, diagnostics, and many other applications. In all such applications, a faulty model of the circuit being analyzed is made, and it is simulated for a test vector or a complete test set. The level at which fault simulation is done depends on the level of the circuit being simulated and the level at which faults are injected in the circuit. Most fault simulations deal with gate-level circuits and stuck-at structural gate-level faults.[3]

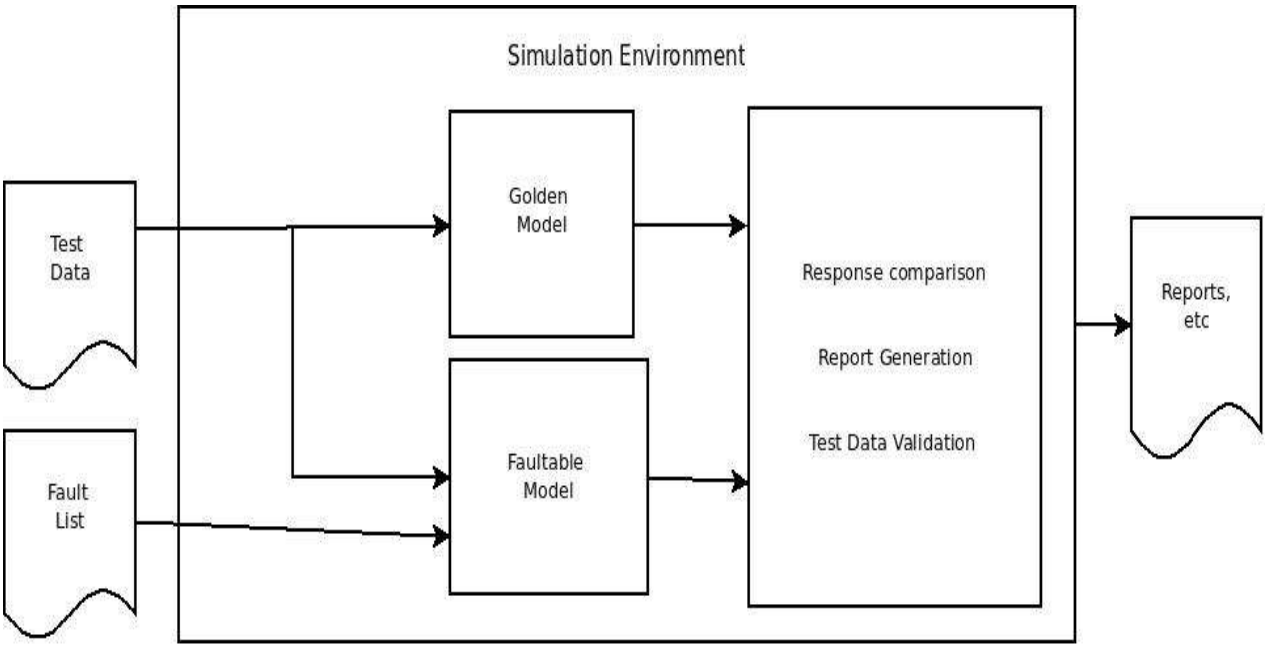


Figure 4.1 Fault simulation process

In this environment, the good circuit model (golden model), and the fault-able model are instantiated and a setup for comparing and reporting their responses is provided. The golden model can be a good net-list or a behavioral description of the circuit being simulated. The fault-able model is at the gate-level and it is in the net-list form. There are provisions in this net-list for making circuit lines faulty, and thus creating various faulty models [41].

The inputs to the simulation environments are test data and fault list. Faults from the fault list input of the simulation environment are read and by the use of the fault injection process injected into the fault-able model to make various faulty models. Test Data (or test set) is read by the simulation environment and applied to the golden and faulty circuit models. For single stuck-at fault model, faults are injected one at a time. With each new fault injection, a new faulty model is created. Each faulty model is compared with the golden model for test vectors in the test set.

As mentioned, the simulation environment is also responsible for collecting simulation results and reporting discrepancies and generating other required reports. The exact tasks performed here determine the application for which fault simulation is being performed. One such application may be identifying test vectors in the test set that can detect a certain number of faults.

4.2. Delay Fault Simulation Requirements

As with the above discussion, information that a fault simulation environment requirements [44] are:

- Circuit net-list with provisions to become faulty (for gate level simulation) ·
- Good circuit net-list or behavioral description(behavioral simulation) ·
- A file containing test data ·
- A file containing fault list ·

A fault simulator produces:

- Report files
- Flags and messages

Let's explain them one by one:

A. **Gate-level simulation:** Since the golden and faulty circuit models are usually available at the gate level, a fault simulation environment should have gate-level simulation capability.

B. **Behavioral Simulation:** While the faulty model is being simulated at the gate-level for stuck-at gate-level faults, the golden model is simulated for its functionality only. Therefore, a fault simulation environment with behavioral simulation capability can save processing time by simulating the behavioral description of the golden model instead of its gate-level net-list.

C. **Reading Data Files**: Capability to read text inputs is required in a fault simulation environment for reading test data and list of faults from external data files. Since in many instances, test data and fault lists are generated by different programs, capability to read different formats of data by the fault simulator may be needed. This eliminates the need for extra format conversion programs.

D. **Fault Injection Capability:** What distinguishes a fault simulation program from a standard logic simulator is the capability of fault injection in a fault simulator. To create a faulty model, a mechanism for injecting faults in the net-list of circuit being simulated is needed. This capability can be internal to the fault simulator or a function added to a standard logic simulator.

E. **Writing Report Files:** Producing output and report files noting various comparison results and simulation reports is the direct result of a fault simulation run. A fault simulation environment must be capable of writing external data files for creating such reports. As in the case of input files, having formatting capabilities to produce reports readable by other test programs is an added advantage.

Fault simulation can be done by using a dedicated fault simulator. Alternatively, some fault simulation techniques can be implemented within standard logic simulators. This may require extra functions and procedures added to the logic simulator which is usually not as efficient as dedicated fault simulation programs.

4.3. Performing Delay Fault Simulation Using Verilog Programming Language Interface

Fault simulation processing listed "A" through "E" above are executed by the Verilog test-bench shown in Fig. 4.5A.The first part of the test-bench declares wires and variables for data application and test control. The golden-under-test (GUT) and fault-able-under-test (FUT) modules are instantiated at the beginning of the test-bench. GUT is the behavioral module of Fig. 4.3 and FUT is that shown in Fig. 4.2. These instantiations have the same set of inputs (a_i, b_i, s_i), and have different output signals (woG, woF) to be compared. If delays are to be considered the net-list module, mux2to1, should be instantiated for the golden model instead of mux2to1B [1].

```
modulemux2to1 ( inputa, b, s, outputw );

wirel1, l2, l3, l4, l5, l6, l7, l8, l9;

pin I1 (s,l1);

pin I2 (a,l5);

pin I3 (b,l6);

pout O1 (l9,w);

fanout_n #(2,1,1) FO1 (l1,{l2,l3});

notg #(1,1) G1 (l4, l2);

nand_n #(2,1,1) G2 (l7, {l5, l4});

nand_n #(2,1,1) G3 (l8, {l6, l3});

nand_n #(2,1,1) G4 (l9, {l7, l8});

endmodule
```

figure 4.2 Fault-able net-list (gate level net-list)

```
modulemux2to1B ( inputa, b, s, outputw );
```

assignw = ~s ? a : b; endmodule

Figure. 4.3: Behavioral description of the multiplexer (Good circuit model)

The **initial** block shown in Fig. 4.4 is responsible for reading tests and faults, injecting faults in FUT and analyzing the outputs. Near the beginning of this statement, the *Mux.flt* file is opened using Verilog $**fopen** system task. This is followed by a **while** loop that uses Verilog's $**fscanf** (formatted scan from a file) to read wireName and stuckAtVal from the fault list. **wireName** is a string and *stuckAtVal* is an integer. The $InjectFault function, that comes after $***fscanf*** injects the *stuckAtVal* value into the FUT wire identified by the wireName hierarchical name

moduleTester ();

regai=0, bi=0, si=0;

wirewoG, woF;

mux2to1B GUT (ai, bi, si, woG);

mux2to1B FUT (ai, bi, si, woF);

initialbegin

endmodule

Fig. 4.4 Fault injection and removal

moduleTester ();

reg ai=0, bi=0, si=0;

wire woG, woF;

reg detected;

integertestFile, faultFile, status;

reg[2:0] testVector;

*reg[8*60:1] wireName;*

```
regstuckAtVal;

while( ! $feof(faultFile))begin

detected = 1'b0;

status = $fscanf(faultFile,"%s@%b\n",wireName, stuckAtVal);

$InjectFault( wireName , stuckAtVal);

testFile = $fopen("Mux.tst", "r");

while((!$feof(testFile))&(detected == 0)) begin

$stop; end// end of initial

Endmodule
```

Figure 4.5 A. Test-bench performing fault simulations

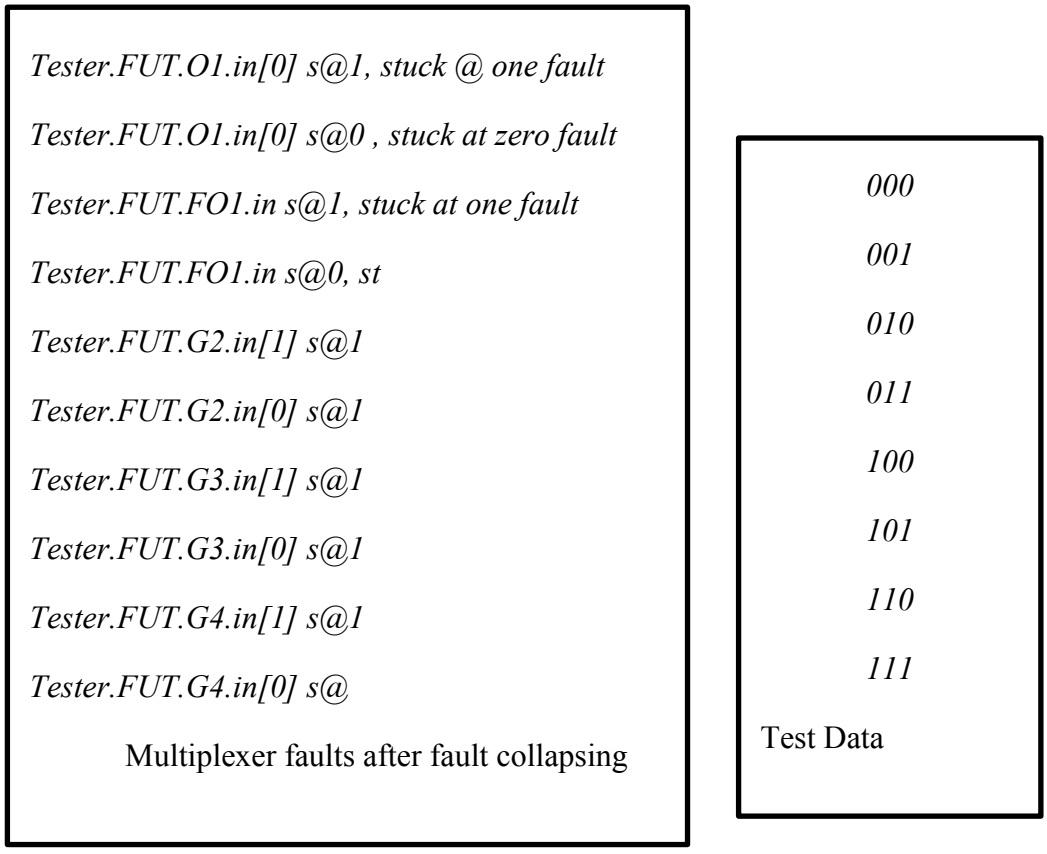


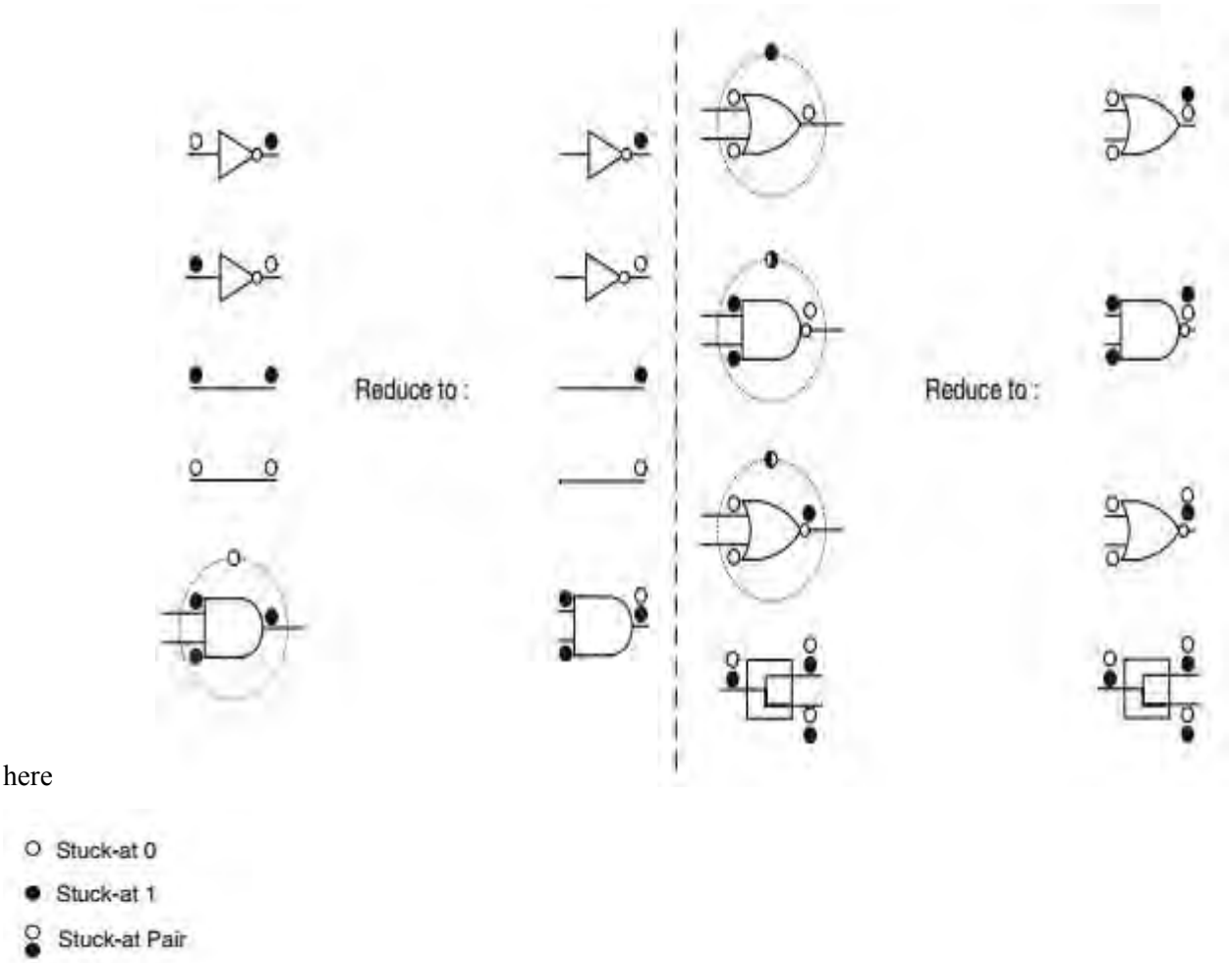
Figure 4.5 B multiplexer faults after fault Collapsing and test vector

Recall that the FUT instance name is used in the hierarchical names in the fault list sample of Fig. 4.5B. This fault injection causes the FUT instance to become a faulty model of the multiplexer. Now that we have a faulty model, the test file is opened in a similar fashion to the fault list file, and in a while loop, test data are read from it. This is the inner while loop in Fig. 4.5 A. As shown, each test vector read from the test file is named as test Vector. The 30 ns delay at the beginning of this while loop is needed for its next iterations. After applying test Vector to circuit inputs, a 60 ns delay allows the propagation of values through the faulty and golden models. Following this, an if statement compares golden and faulty outputs.

Outputs woG and woF are different when a test has been applied that detects fault injected in FUT. Making fault reports, counting the number of detected faults, saving tests that detect a fault, and other functions that fault simulation is being done in this if block. In the above example, it is only reported the detected fault, the input value, and the time that fault is detected. After the while loop applying all tests in Mux.tst file to a given faulty model, $Remove Fault removes the fault, allows a settling time required by the PLI function, and goes into the next iteration of injecting a new fault. The fault simulation report will be explained and given in the coming chapters and sections.

4.4. Stack at Fault Dropping (Collapsing)

A term that is often used in relation with fault simulation is fault dropping. This term refers to dropping a fault from further processing (or simulation), once the fault is found to be detected. Fault dropping is done for reducing simulation time by skipping extra testing for the fault [5].



here

Figure4.6. Rules for Local Fault Collapsing [5][16]

According to Fig. 4.6 and 4.7, figure 3.4(from chapter 3) is analyzed as follows. The G3 NAND gate requires a SA1 on each of its inputs, and SA0 and SA1 on its output. This means that the SA0 faults on l3 and l6 inputs of G3 are not needed and can be removed. As for the output of G3, the required SA0 and SA1 faults are already placed on l_8 at the input of G_4. Before going forward, the inverter that is at the same distance to circuit inputs as G_3 must be treated. Gate G_2 SA0 and SA1 faults move to the output of this gate and become SA1 and SA0. These faults move on I_4 close to G2 and collapse into faults already at this location. We then follow this procedure for G2, which leads to keeping a SA1 on its I5 input, a SA1 on its I_4 input, and SA0 and SA1 on its I_7 output. As before, output faults move to the inputs of the next gate. The last

level in this circuit is that of G4. Considering that this is also a NAND, we remove SA0 faults at its inputs and keep the SA0 and SA1 faults at its output.

Another type of fault collapsing that can further reduce the number of faults we deal with is called dominance fault collapsing. Fault f1 is said to dominate fault f2 if all tests that detect f2 also detect f1, but detecting f1 does not necessarily mean that f2 is also detected [24]. Dominance fault collapsing removes dominated output faults and only keeps gate input faults. Figure 4.7 shows dominance rules for basic gate structures.

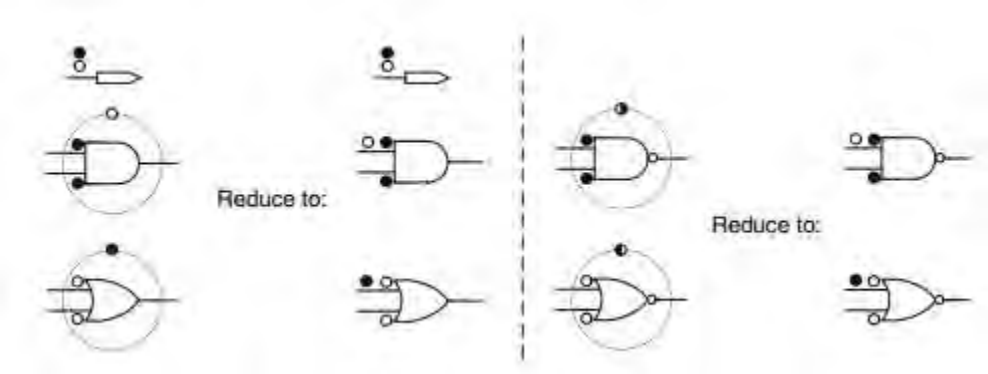


Figure 4.7 Dominance fault collapsing example [5]

The process of selecting one fault from each equivalence set is called fault collapsing. The set of selected faults is known as the equivalence collapsed set. The relative size of the equivalence collapsed set with respect to the set of all faults is the collapse ratio:

$$collapse\ ration = \frac{|set\ of\ collapsed\ faults|}{|set\ of\ all\ faults|} \quad \text{---------- 4.1}$$

Table 4.1 Equivalence fault collapsing for stuck-at faults in benchmark circuits [45]

Circuit Name	NO. of Gates	NO. of Inputs	No. of outputs	Number of Faults		
				All	Collapsed	Collapse Ratio
C432	160	36	7	864	524	0.61
C499	202	41	32	998	758	0.76
C880	383	60	26	1760	968	0.55
C1355	546	41	32	2710	1606	0.59
C1909	880	33	25	3816	2041	0.54
C2670	1193	233	140	5340	2943	0.55
C3540	1669	50	22	7080	3651	0.52
C5315	2307	178	123	10630	5663	0.53
C7552	3513	207	108	15104	8084	0.54
S27	10	4	1	52	32	0.62
S9234	5597	19	22	10572	3862	0.37
S38584	19257	22	278	78854	36303	0.47

As we can see from the table above, larger circuits allow more fault collapsing as indicated by smaller collapse ratios. Circuits with large fan-ins also reduce the collapse ratio. An example is the s9234 circuit. The circuit c499 contains a large number of exclusive-OR modules. These are not primitive Boolean gates and do not allow fault collapsing among input and output faults. As a result, the collapse ratio (0.76) is high. The Boolean gate implementation of c499 is the circuit c1355, which allows greater collapsing.

4.5 Critical Path Tracing Fault Simulation methods

A fault simulation with a different approach than the one discussed in the previous sections is critical path tracing (CPT). The method discussed so far was fault-oriented, which means that faults are injected and tests are applied to check for their detection. In CPT, for a given test vector, the circuit is traced from outputs to inputs, and faults that can be detected are identified [5], [33] and [46]

We present definitions regarding critical paths before discussing the implementation of CPT fault simulation.

Critical Lines:- With a set of logic values at a gate's input and output, an input line is critical if the output is critical and toggling the input toggles the output value. Figure 4.8 shows several critical and noncritical gate inputs. Lines driving primary outputs are always critical.

Critical Path: Critical path is a continuous sequence of critical lines starting from a line in a circuit leading to a primary output, from a critical path.

Critical Path Faults: Stuck-at faults along a critical path, with fault values that are complement of those of the critical lines of the path, can be detected by the test vector causing the critical path.

Fault simulation based on CPT is simple, has fewer computations than most method, and requires a simple data structure for its implementation. Some of the disadvantages of this method include the issue with re-convergent fan-out. When we reach such a structure, the simple problem of tracing gates from output to input of a circuit turns into finding cones of convergence and performing simulation. This problem becomes more critical with nested re-convergent fan-outs. Some works on approximations and solutions of this problem have been presented in the literature [14] and [40].

The biggest price we are paying for the simple implementation of CPT fault simulation is that this method only works for combinational circuits. Other disadvantages are the requirement for a special simulation engine, and not considering collapsed fault lists.

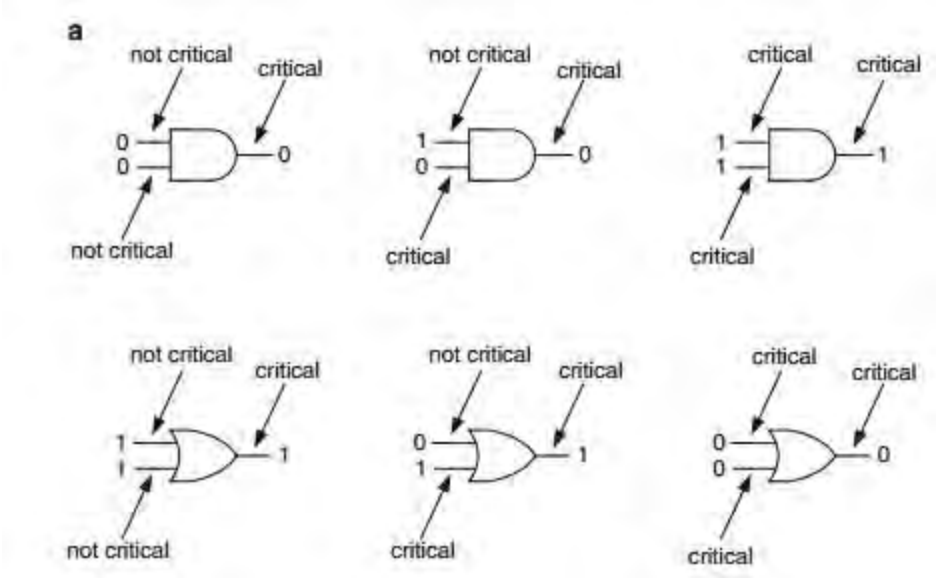


Figure 4.8 Critical values [1]

Given test T, ntest vectors, t1: n;

Given fault list F, mfaults, f1: m; f0 no fault;

For I in 1 to nLoop – every tin T

Propagate ti;

Find all STR/STF faults;

Lists faults detected by ti;

End for;

Figure 4.9 Critical Path Tracing Algorithm [39]

For a test set, fault coverage (or test coverage) is the ratio of detected faults over total faults in a circuit [4]. Total faults are often those obtained after fault collapsing. For test set T,

$$FC = \frac{Detected}{total\ faults}$$ ----------4.2

For a given test set, fault simulation is used as a stand-alone program to calculate fault coverage to measure how good the test set is. For a test set being developed, fault simulation is used to measure the coverage of an existing test set, to stop generating more tests when enough coverage has been obtained. Another area of test generation where fault coverage is important is the evaluation of deterministic tests. Such tests are generally generated without regard to circuit and gate delays. Because of this, due to actual gate delays, some tests may not detect faults that they have been targeted for. For this reason, performing fault simulation of gate-level circuits with gate and wire delays gives a realistic measure of faults covered by the generated tests [5].

Figure 4.10 shows the procedure for fault coverage calculation. We start with a given test set and a fault list. We assume a good circuit model and a fault-able model are instantiated and being compared (GUT and FUT). Injecting faults of the fault list into FUT makes our faulty circuits.

The procedure consists of two nested loops. The outer loop considers every fault in the fault list, and the inner loop applies tests in the test set to the faulty circuit until one detects the fault. In the inner loop, the outputs of the good circuit and faulty circuits are compared. The two outputs differ when test t detects fault f. In that case, the number of detected faults is incremented, the rest of tests are skipped, and the next fault is considered (this implements fault dropping).

When completed, every fault has been examined for detection by the test set. The first test vector of the test set that detects a fault causes the fault to be marked as detected, and the number of detected faults to be incremented. When all faults are considered, fault-coverage is calculated by the ratio of the detected faults, DF, over the original number of faults, NF.

Calculation of fault coverage for sequential circuits is only different from combinational circuits fault coverage in the application of reset after injection of each fault, and the application of clock after assigning a test vector to the circuit inputs.

Given Test Set T, n test vectors, t1: n;

Given Fault List F, m faults, f1: m; f0 no fault;

For j in 1 to m loop —- every f in F

Inject f_j;

For i in 1 to n loop —- every t in T

While f_j is not detected begin

Simulate faulty circuit;

Increment DF if f_j is detected;

End while;

End for;

Remove f_j;

End for;

Record DF, detected fault in F;

Calculate %FC based on m and DF;

Figure 4.10 Fault coverage calculations algorithm [39]

4.6. Path Delay Fault Simulation Using BDD Methodology

In this paper, we use this method for path delay fault simulation methodology. Ordered binary decision diagrams (OBDDs) are used in applications for circuit verification and equivalent checking especially for path delay fault simulation as we have seen it in literatures above. Although the underlying model of decision diagrams (or synonymously branching programs) was already studied by Lee and Akers in the 1950s and 1970s, these representations have not been used in serious applications for a long time. Through time due to improvements in OBDDs, has invaded nearly all areas of computer-aided VLSI design [9].

These facts have been established by the following valuable properties of OBDDs:

- Reduced OBDDs provide a canonical representation of switching functions
- Reduced OBDDs can be manipulated efficiently

For many practically important switching functions, the corresponding OBDD representations are quite small.

4.6.1.Overview of Binary Decision Diagrams

Definition :- Let π be a total order on the set of variables $x_1,\ldots\ldots,x_n$. An ordered Binary decision diagram with respect to the variable order π is a directed acyclic graph with exactly one root which satisfies the following properties [4]:

- There are exactly two nodes without outgoing edges. These two nodes are labeled by the constants 1 and 0, respectively and are called sinks.
- Each non-sink node is labeled by a variable x_i, and has two outgoing edges, which are labeled by 1 and 0, respectively. These edges are called the 1-edge and the 0-edge, respectively.
- The order, in which the variables appear on a path in the graph is consistent with the variable order π, i.e., for each node determined by the 1-edge is denoted high (v), and the successor node determined by the 0-edge is denoted by low (v).

Binary Decision Diagrams (BDDs) are one of the biggest breakthroughs in CAD in the last decade. BDDs are a canonical and efficient way to represent and manipulate Boolean functions and have been successfully used in numerous CAD applications. The design and synthesis of digital circuits has gained importance in both the industrial and academic worlds. Timed circuits are a class of asynchronous circuits that incorporate explicit timing information in the specification. This information is used throughout the synthesis procedure to optimize the design. Meeting timing requirements is an important constraint imposed on highly integrated circuits, and the verification of timing of a circuit before manufacturing is one of the critical tasks to be solved by CAD tools. Several algorithms to analyze and speed up gate level timing simulation have been proposed in various research papers. Some propose gate delay analysis and others path

delay analysis. Just as in case of asynchronous circuits, delay analysis of synchronous circuits is also becoming a region of active research [10].

A binary decision diagram is a directed acyclic graph in which every node represents some Boolean function. There are exactly two outgoing edges for every non - terminal node. There are only two terminal nodes representing the constants 1 and 0.

4.6.2.Delay Faults and Their Analysis Using OBDD

A delay fault is a fault that causes incorrect data to be latched into a memory element or appear at an output. The maximum allowable path delay in a synchronous circuit is determined by its clock rate. If the delay on a path of a manufactured circuit exceeds the time period of the clock, erroneous values may be latched at the output. The objective of the delay testing is to ensure that the manufactured circuit operates correctly at the functional clock rate [47].

Two common delay faults models are the gate and line/path delay fault models. These have been investigated for test generation and fault simulation. The path delay model is capable of modeling distributed failures resulting due to statistical variation in the manufacturing process. It is thus very critical for circuit designing using statistical timing analysis. The main problem with path delay fault model is that there can be an exponential number of path faults in a circuit. Thus making it practically impossible to enumerate all paths for the purpose of test generation and fault simulation [17].

A combinational circuit is represented as a directed acyclic graph (DAG), G (V; E), where gates correspond to nodes and the wires correspond to edges in the graph.

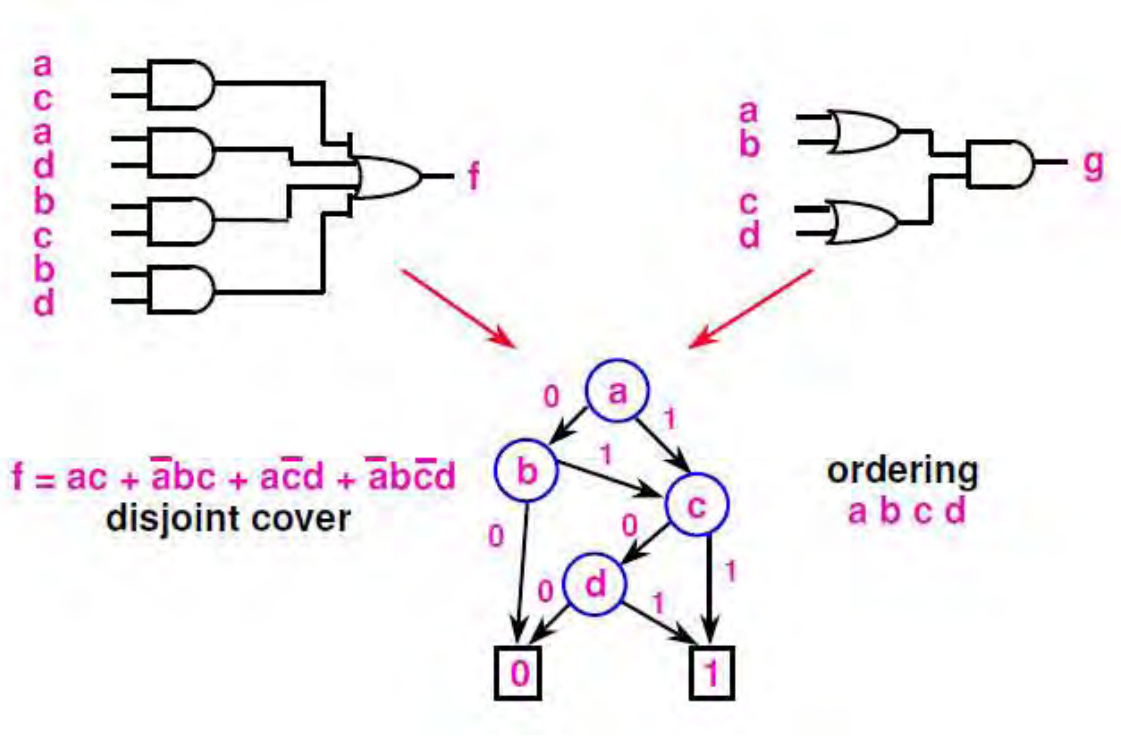


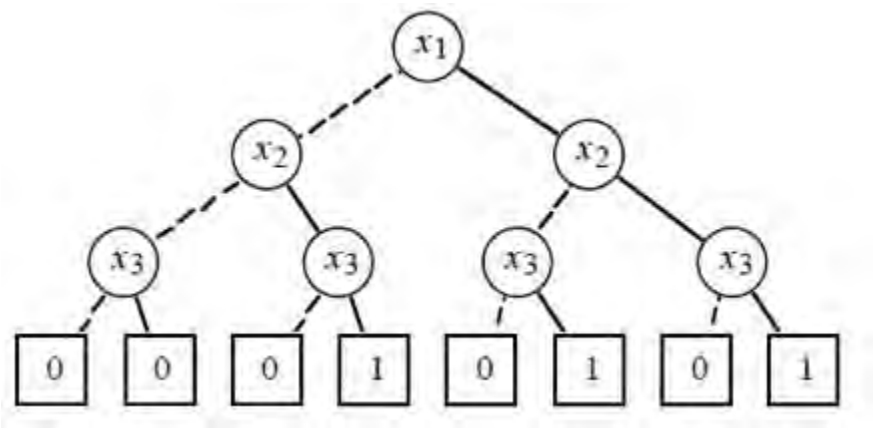
Figure 4.11: Boolean circuit with its respective binary decision diagram

For circuit with large number of paths, an explicit fault representation is not practical. Efficient path manipulation algorithms are necessary for execution speed as well as to maintain manageable memory requirement. An efficient path representation scheme has been presented in [10]. This scheme allows for sequential numbering of the paths in a circuit. This is achieved through a combination of a linear algorithm for path counting and structural ordering of paths through each node. For the work presented here, the two path faults, rising and falling, associated with path I are numbered 2i-1 and 2i, respectively.

4.6.3.Ordering and Reducing in BDD

For an Ordered BDD (OBDD), we impose a total ordering over the set of variables and require that for any vertex u, and either non-terminal child v, their respective variables must be ordered var(u) < var(v). In the decision tree of Figure 4.12, for example, the variables are ordered $x1 < x2 < x3$. In principle, the variable ordering can be selected arbitrarily—the algorithms will operate correctly for any ordering. In practice, selecting a satisfactory ordering is critical for the

efficient symbolic manipulation. We define three transformation rules over these graphs that do not alter the function represented [4][17]:



x1	$x2$	$x3$	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Figure 4.12 Truth Table and Decision Tree Representations of a Boolean Function

A dashed (solid) tree branch denotes the case where the decision variable is 0 (1).

Remove Duplicate Terminals: Eliminate all but one terminal vertex with a given label and redirect all arcs into the eliminated vertices to the remaining one.

Remove Duplicate Non-terminal: If non-terminal vertices u and v have $var(u)$=var(v), $lo(u)$= $lo(v)$, and $hi(u)$=$hi(v)$, then eliminate one of the two vertices and redirect all incoming arcsto the other vertex.

Remove Redundant Tests: If non-terminal vertex v has $lo(v)$=$hi(v)$, then eliminate v and redirect all incoming arcs to $lo(v)$.

Starting with any BDD satisfying the ordering property, one can reduce its size by repeatedly applying the transformation rules. We use the term "OBDD" to refer to a maximally reduced graph that obeys some ordering. For example, Figure 4.13 illustrates the reduction of the decision tree shown in Figure 4.12 to an OBDD. Applying the first transformation rule (A) reduces the eight terminal vertices to two. Applying the second transformation rule (B) eliminates two of the vertices having variable x_3 and arcs to terminal vertices with labels 0 (low) and 1 (high). Applying the third transformation rule (C) eliminates two vertices: one with variable x_3 and one with variable x_2. In general, the transformation rules must be applied repeatedly, since each transformation can expose new possibilities for further ones [4].

The OBDD representation of a function is canonical for a given ordering; two OBDDs for a function are isomorphic. This property has several important consequences. Functional equivalence can easily be tested. A function is satisfiable if and only if its OBDD representation does not correspond to the single terminal vertex labeled 0. Any tautological function must have the terminal vertex labeled 1 as its OBDD representation. If a function is independent of variable x, then its OBDD representation cannot contain any vertices labeled by x. Thus, once OBDD representations of functions have been generated, many functional properties become easily testable [19].

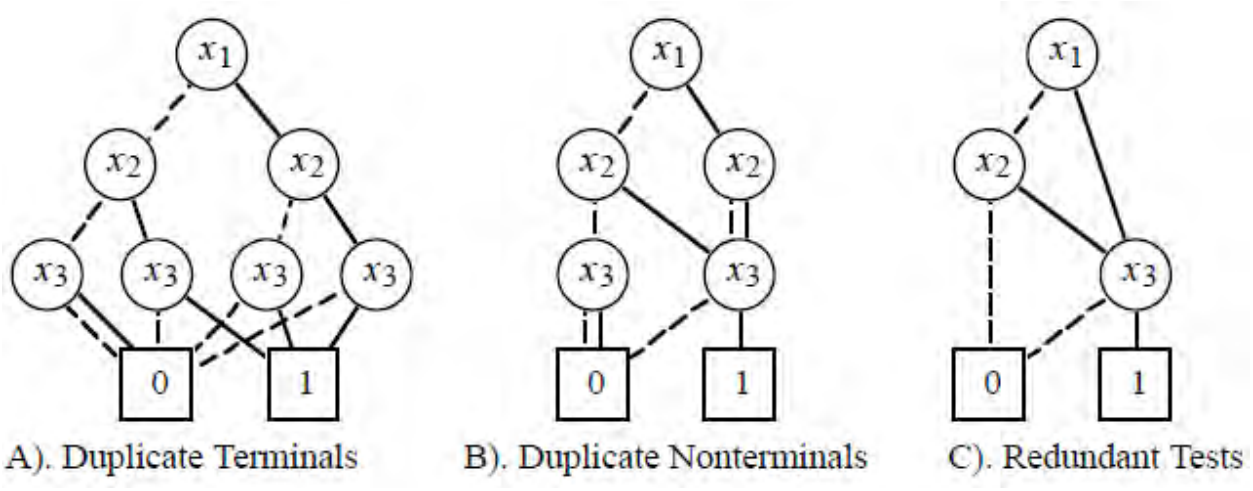


Figure 4.13 **Reduction of Decision Tree to OBDD.**

Applying the three reduction rules to the trees of Figure 4.12 yields the canonical representation of the function as an OBDD. As Figures 4.12 and 4.13 illustrate, we can construct the OBDD representation of a function given its truth table by constructing and reducing a decision tree.

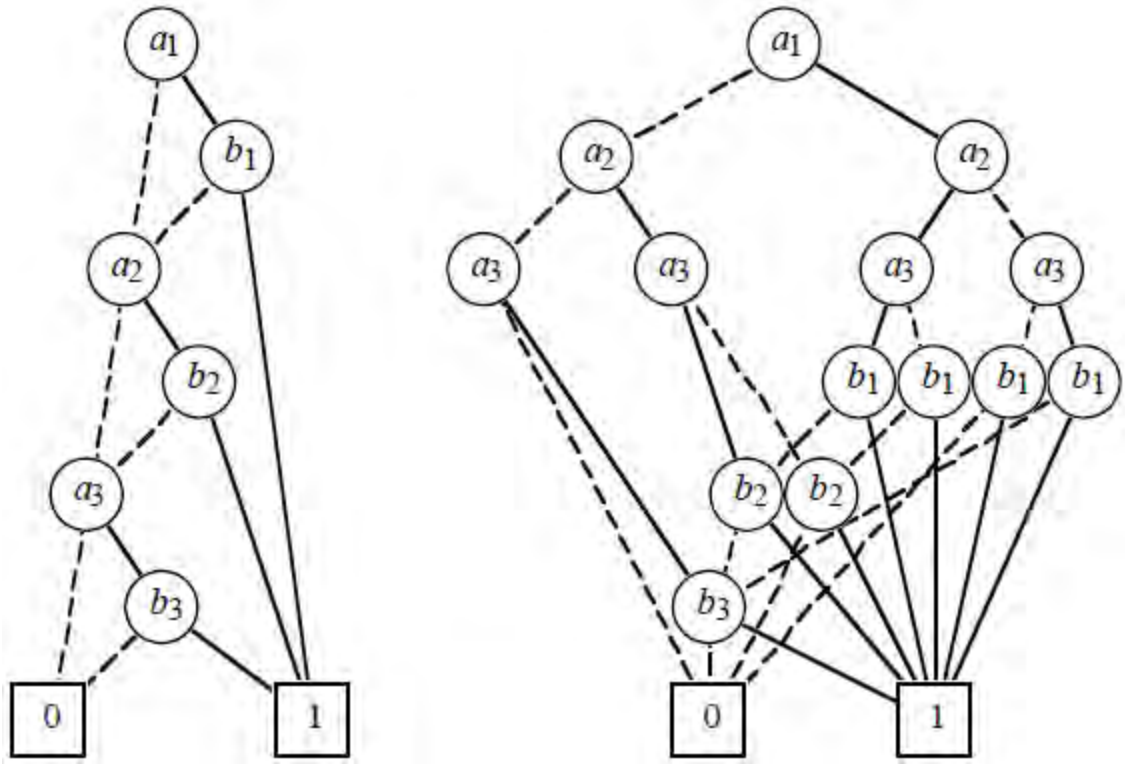


Figure 4.14: OBDD Representations of a Single Function for Two Different Variable Orderings.

This approach is practical, however, only for functions of a small number of variables, since both the truth table and the decision tree have size exponential in the number of variables. Instead, OBDDs are generally constructed by "symbolically evaluating" a logic expression or logic gate network using the some reduction operation.

The form and size of the OBDD representing a function depends on the variable ordering. For example, Figure 4.13 shows two OBDD representations of the function denoted by the Boolean expression $a1.b1 + a2.b2 + a3.b3$, where. Denotes the AND operation and + denotes the OR operation. For the case on the left, the variables are ordered $a1 < b1 < a2 < b2 < a3 < b3$ while for the case on the right they are ordered $a1 < a2 < a3 < b1 < b2 < b3$. We can generalize this function to one over variables a_1, a_2…….a_n and b1, b_2, b_3…….bn given by the expression:

$a1.b1 + a2.b2 + \cdots an.bn$----------4.3.

Generalizing the first variable ordering to $a_1 < b_1 < \ldots\ldots < a_n < b_n$ yields an OBDD with $2n$ nonterminal vertices—one for each variable. Generalizing the second variable ordering to $a_1 <$

.....< a_n < b_1 < ....<b_n, on the other hand, yields an OBDD with $2(2^n - 1)$ non-terminal vertices. For large values of n, the difference between the linear growths of the first ordering versus the exponential growth of the second has a dramatic effect on the storage requirements and the efficiency of the manipulation algorithms [4].

Examining the structure of the two graphs of Figure 4.13, we can see that in the first case the variables are paired according to their occurrences in the Boolean expression $a_1 .b_1 + a_2 .b_2 + a_3.b_3$. Thus, from every second level in the graph, only two branch destinations are required: one to the terminal vertex labeled 1 for the case where the corresponding product yields 1, and one to the next level for the case where every product up to this point yields 0.

Table 4.2: OBDD complexity for common function classes

Function class	Complexity	
	Best	Worst
Symmetric	Linear	Quadratic
Integer Addition (any bit)	Linear	Exponential
Integer Multiplication (middle bits)	Exponential	Exponential

On the other hand, the first 3 levels in the second case form a complete binary tree encoding all possible assignments to the variables. In general, for each assignment to the variables, the function value depends in a unique way on the assignment to the b variables. As we generalize this function and ordering to one over 2n variables, the first n levels in the OBDD form a complete binary tree [8].

Most applications using OBDDs choose some ordering of the variables at the outset and construct all graphs according to this ordering. This ordering is chosen either manually, or by a heuristic analysis of the particular system to be represented. For example, several heuristic methods have been devised that, given a logic gate network, generally derive a good ordering for variables representing the primary inputs. Others have been developed for sequential system analysis. Note that these heuristics do not need to find the best possible ordering—the ordering chosen has no effect on the correctness of the results. As long as an ordering can be found that avoids exponential growth, operations on OBDDs remain reasonably efficient [4][47].

4.7. Coverage of Multiple Path Delay Faults Using BDD

The main objective of delay fault testing is to check for the timing performance of a designed circuit. The problem of calculating the number of faults identified by a given test set under a specific fault model is termed as non-enumerative fault grading or fault coverage. Under the path delay fault (PDF) model, a circuit is considered faulty if the propagation delay of a path exceeds the pre-determined clock period of the circuit, the delay fault can be observed by propagating a rising or falling transition through the path. This requires that each of the tests in the test set consists of 2 vectors as described in section 4.5.

Existing enumerative and non-enumerative [6], [8],[10], [18] fault grading techniques only examine the single path delay faults (SPDF). It has been shown that multiple path delay faults can affect the timing performance of a circuit. Even though there exists methods for generating test patterns for this category of delay faults, there is no existing technique to calculate the exact coverage for the test sets aimed at multiple paths.

The problem of fault grading for multiple path delay faults is studied and a method of obtaining the exact coverage is presented. The faults covered are represented and manipulated as sets by OBDD, which are shown to be able to store a very large number of path delay faults. For the extreme case of memory problem, a method to estimate the coverage of the test set is also presented. The problem of fault grading is solved with a polynomial number of BDD operations. Experimental results on the ISCAS'85 benchmark include test sets from ATPG tools and specifically designed tests in order to investigate the limitations and properties of the proposed method [46].

Delay testing of VLSI circuits involves a test of the ability of the combinational or combinational part of sequential circuits to propagate signals within a specified amount of time. Usually, if a test fails and there are no DC faults such as stuck-at, then we look for transition, gate, segment, and path delay faults in a circuit [6]. The PDF model represents a delay along a path from a primary input or the output of a flip-flop to a primary output or the input of a flip-flop, and is considered to be the most accurate of these fault models. The number of possible PDFs, twice the number of paths in a circuit, is far higher than the possible number of transition delay or stuck-at faults in a circuit. Therefore, test vector generation and fault coverage calculation for PDFs are very time-consuming tasks in comparison to those of stuck-at or other delay fault models.

4.8. Equivalence Checking for Identification of Differences Between Two Descriptions of Circuits with Path Delay Fault

The register transfer level (RTL) behavior of a digital chip is usually described with a hardware description language, such as Verilog or VHDL. This description is the golden reference model that describes in detail which operations will be executed during which clock cycle and by which pieces of hardware. Once the logic designers, by simulations and other verification methods, have verified register transfer description, the design is usually converted into a net-list by a logic synthesis tool. Equivalence is not to be confused with functional correctness, which must be determined by functional verification [17].

The initial net-list will usually undergo a number of transformations such as optimization, addition of Design for Test (DFT) structures, etc., before it is used as the basis for the placement of the logic elements into a physical layout. Contemporary physical design software will occasionally also make significant modifications (such as replacing logic elements with equivalent elements that have a higher or lower drive strength) to the net-list. Throughout every step of a very complex, multi-step procedure, the original functionality and the behavior described by the original code must be maintained. When the final tape-out is made of a digital chip, many different EDA programs and possibly some manual edits will have altered the net-list [50].

In theory, a logic synthesis tool guarantees that the first net-list is logically equivalent to the RTL source code. All the programs later in the process that make changes to the net-list also, in theory, ensure that these changes are logically equivalent to a previous version.

In practice, programs have bugs and it would be a major risk to assume that all steps from RTL through the final tape-out net-list have been performed without error. Also, in real life, it is common for designers to make manual changes to a net-list, commonly known as Engineering Change Orders, or ECOs, thereby introducing a major additional error factor. Therefore, instead of blindly assuming that no mistakes were made, a verification step is needed to check the logical equivalence of the final version of the net-list to the original description of the design (golden reference model) [5].

Historically, one way to check the equivalence was to re-simulate, using the final netlist, the test cases that were developed for verifying the correctness of the RTL. This process is called gate level logic simulation. However, the problem with this is that the quality of the check is only as good as the quality of the test cases. Also, gate-level simulations are notoriously slow to execute, which is a major problem as the size of digital designs continues to grow exponentially [29].

An alternative way to solve this is to formally prove that the RTL code and the netlist synthesized from it have exactly the same behavior in all (relevant) cases. This process is called formal equivalence checking and is a problem that is studied under the broader area of formal verification. A formal equivalence check can be performed between any two representations of a design: RTL <> netlist, netlist <> netlist or RTL <> RTL, though the latter is rare compared to the first two. Typically, a formal equivalence checking tool will also indicate with great precision at which point there exists a difference between two representations. The flow of equivalence checking is shown in Figure 4.15.

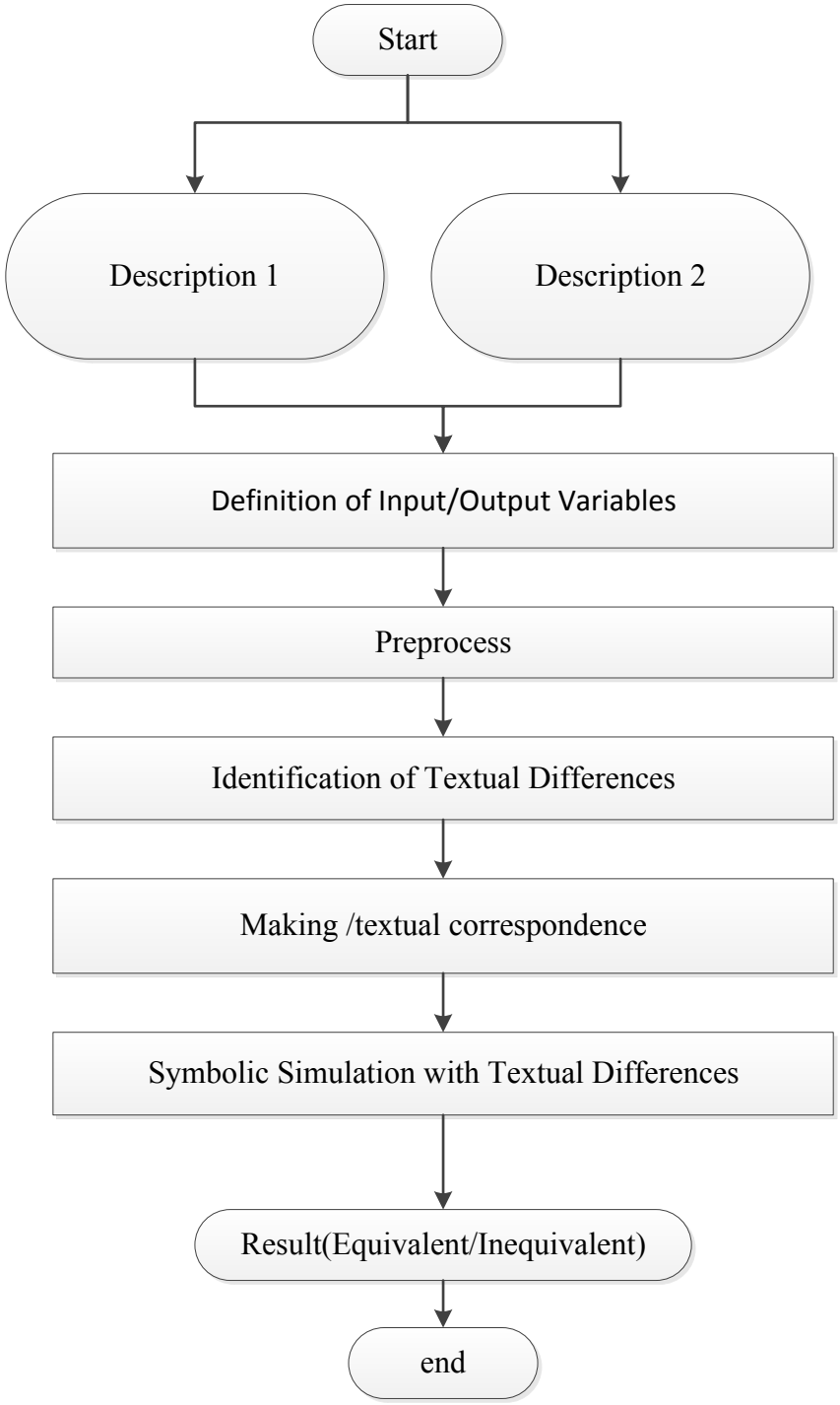


Figure 4.15: Equivalence checking flow

4.8.1. Identification of Differences Between Two Hardware File Descriptions

Textual differences between the two given design descriptions are identified. By using information of textual differences, we can establish a one-to-one correspondence between expressions in the two descriptions. This is based on the assumption that the two design descriptions are not much different. If they are, the one-to-one mapping generation may simply fail, which is not dealt with here.

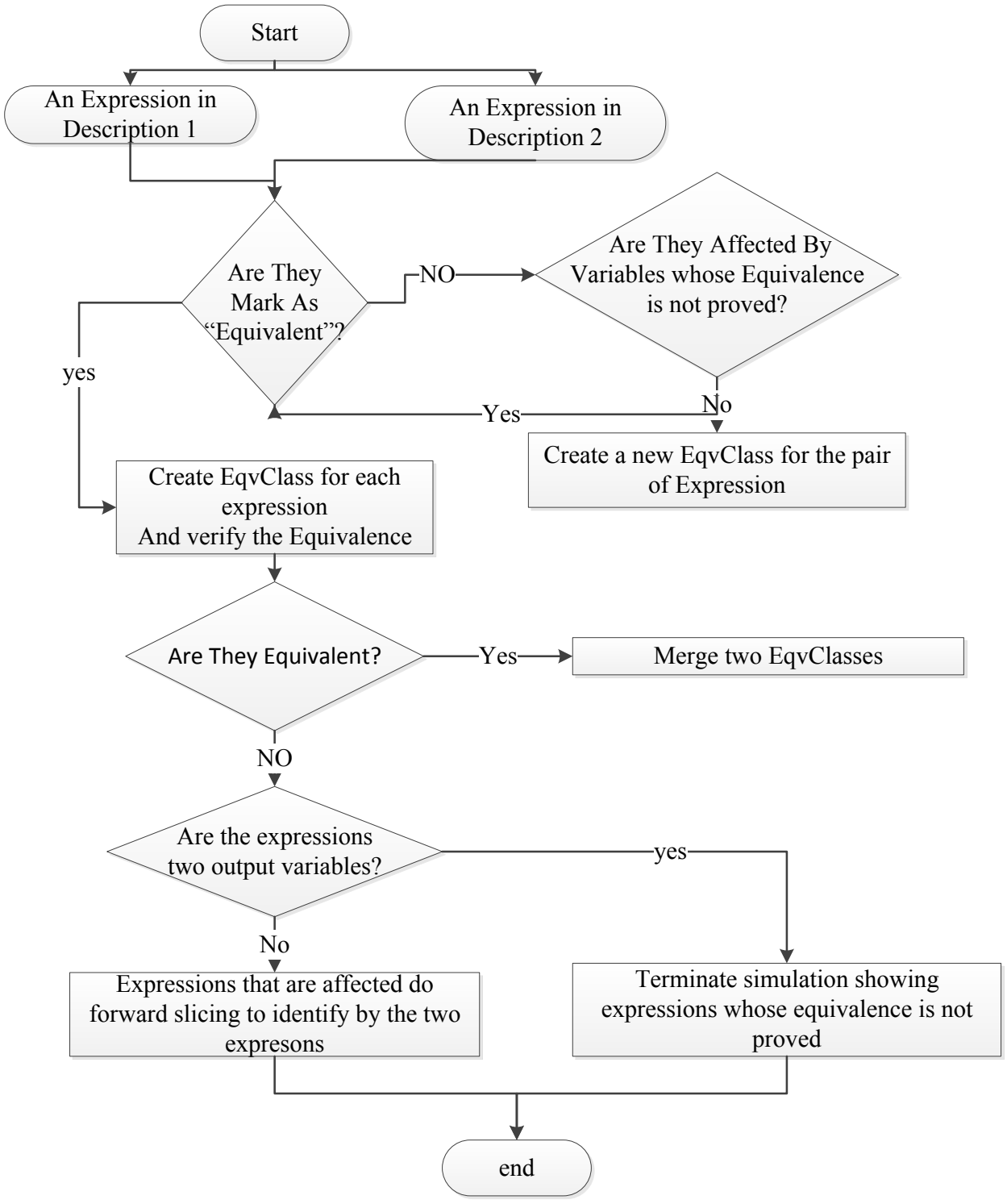


Figure 4.16 Equivalence checking for a pair of expressions.

The method that utilizes textual differences to reduce the number of equivalence checks during symbolic simulation (as described above) checks the equivalence of only the corresponding pairs of assignments. The method that does not utilize textual differences simulates whole descriptions separately with equivalence checking. All variables and expressions in both descriptions are checked for their equivalence.

There are two basic technologies used for Boolean reasoning in equivalence checking programs:

- Binary decision diagrams or BDDs: A specialized data structure designed to support reasoning about Boolean functions. BDDs have become highly popular because of their efficiency and versatility.
- Conjunctive Normal Form Satisfiability: SAT solvers returns an assignment to the variables of a propositional formula that satisfies it if such an assignment exists. Almost any Boolean reasoning problem can be expressed as a SAT problem.

In general, there is a wide range of possible definitions of functional equivalence covering comparisons between different levels of abstraction and varying granularity of timing details [4].

- The most common approach is to consider the problem of machine equivalence which defines two synchronous design specifications functionally equivalent if, clock by clock, they produce exactly the same sequence of output signals for any valid sequence of input signals.
- Microprocessor designers use equivalence checking to compare the functions specified for the instruction set architecture (ISA) with a register transfer level (RTL) implementation, ensuring that any program executed on both models will cause an identical update of the main memory content. This is a more general problem.
- A system design flow requires comparison between a transaction level model (TLM), e.g., written in SystemC and its corresponding RTL specification. Such a check is becoming of increasing interest in a system-on-a-chip (SoC) design environment.

We use an equivalence checking method based on BDDs. The basic idea is the following: given an abstraction function, ROBDDs reduce the size of BDDs by merging nodes that have the same abstract value as described in sections 4.12, 4.13 and 4.14 above. An OBDD has bounded size

and can be constructed without constructing the original BDD. It is complete for an important class of arithmetic circuits that includes integer multiplication [4][17][47].

4.8.2.Combinational Equivalence Checking

This work also leverages the success of RTL vs. gate-level equivalence checking of combinational circuits expressed in textual form (BLIF) file format [48] for path delay fault simulation. In this section, we are going to introduce some basic background for combinational equivalence checking.

The goal of combinational equivalence checking is to check whether two combinational circuits are functionally equivalent (i.e., for all possible inputs, both combinational circuits have the same outputs). The basic mathematical tool for reasoning about digital circuits is Boolean algebra (using BDD or SAT). There-fore, the equivalence of two circuits corresponds to the problem of determining the equivalence of Boolean formulas [47].

Combinational equivalence checking is a co-NP (non-polynomial complexity) complete problem. Hence, finding a method with polynomial worst-case time complexity to solve this problem is extremely unlikely. However, practical instances of the problem are often more tractable. Usually, two combinational circuits being verified have a certain degree of similarity due to the steps of synthesis or optimization. There is some successful research to develop techniques that yield acceptable performance for verifying practical combinational circuits of realistic size [17].

BDD-based techniques are commonly used for combinational equivalence checking. As we mentioned in previous sections, BDDs are canonical representations of Boolean formulas. By building BDDs for the two circuits, equivalence can be proven if the BDDs for the outputs of both circuits are identical. In this case for path delay simulation, the textual form of the good circuit model and faulty circuit model are converted in to OBDD and the outputs are compared. For the delay case, we have compared the depths of the fault and good circuit models [48].

To be specific, the BDD approach is normally as follows: For circuit C_1 and C_2, we build a circuit C which merges the primary inputs of C_1 and C_2 together and has separate primary outputs for C_1 and C_2. Then, we construct BDDs for the primary outputs of C, and compare

BDDs for the corresponding primary out-puts. The equivalence (or nonequivalence) of circuits is proven by the results of the comparison.

One of these techniques is cut point theory [1][2][3][4][16][40][50]. In Figure 4.18, we have two circuits F and G. The point's z_1 and z_2 are internal cut points for circuit F and circuit G respectively. The primary outputs of circuit F and circuit G are another pair of cut points called external cut points. If we can prove that $\forall$ x, f_1 (x) = g_1 (x), i.e., $z_1 = z_2$, we introduce another primary input variable z to substitute z_1 and z_2 .Then we have $\forall$ z, y, f_2 (z, y) = g_2 (z, y) $\Rightarrow$ F = G. i.e., if we can prove all the cut points are equivalent, so are the functions. In general, the method is conservative (i.e., success proves equivalence, but failure doesn't prove in equivalence) because constraints on the cut point "inputs" are lost. The situation that two circuits are equivalent but verification reports in equivalence is called false negative or false in equivalence. The general solution is to re-introduce constraints on the cut points, either in advance [1] or as needed [16][40].

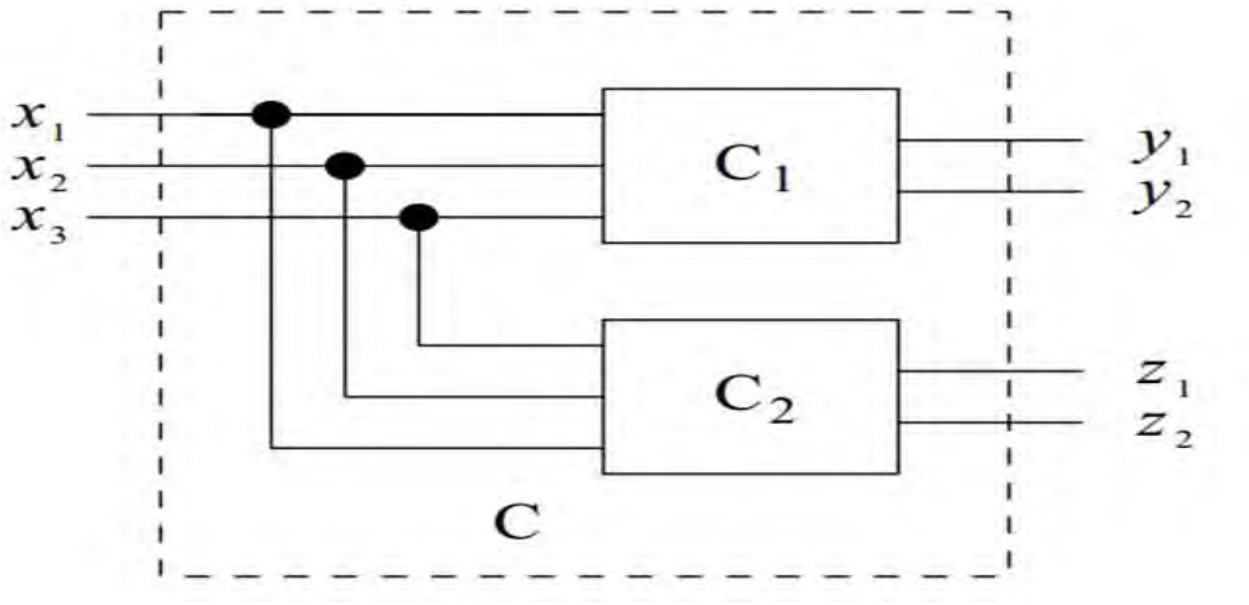


Figure 4.17: Combinational equivalence checking example [49]

For example, as shown in Figure 4.17, circuit C merges the primary inputs of C_1 and C_2 together as x_1, x_2, and x_3. The output variables are y_1, y_2, z_1, and z_2. After that, we construct BDDs for circuit C. If the BDDs for y_i and z_i are identical (i =1, 2…), we can declare that circuits C1 and C2 are equivalent.

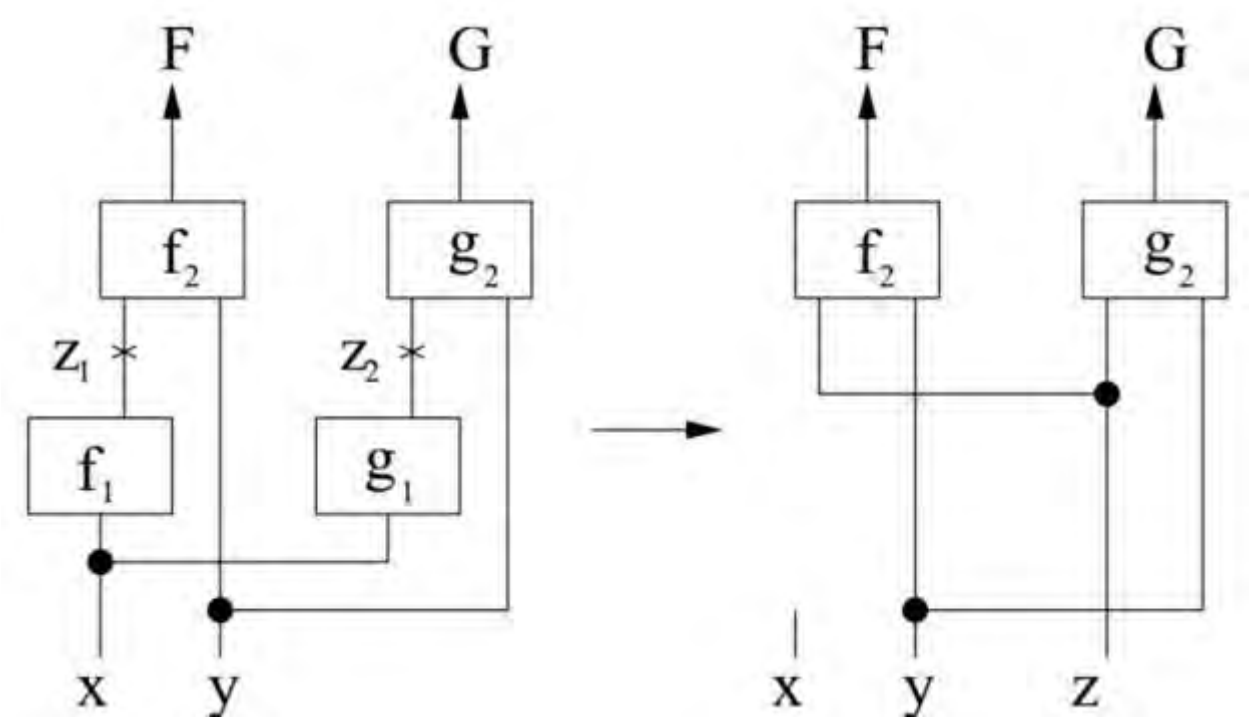


Figure 4.18: Cut point Example [50]

The size of the BDDs is determined by the types of functions and the selected variable ordering. In practice, the BDDs for big circuits are beyond the memory capability of current computers. Many abstraction techniques have been applied to attack this problem. As we mentioned at the beginning of this section, the design procedure often results in some similarities of the two circuits being verified. By identifying and exploiting these structural similarities, a combinational equivalence checking tool can greatly reduce the complexity of verification.

Chapter Five

5. Experimental Output and Result Discussion

5.1. Introduction

An approach for selecting high-risk paths based on the user defined threshold (90%) along which testable path delay faults can exist and a Verilog based RTL level fault modeling technique is presented. The proposed method and model is particularly helpful on path intensive circuits. Critical paths are selected implicitly with the aid of a combination of decision diagrams. The effectiveness of the approach is demonstrated on path intensive ISCAS'85 benchmark circuit.

The number of functionally testable PDFs is extremely large to target. In practice, testing focuses on a subset of PDFs whose typical delay values are very large. A PDF is critical if it can affect the temporal behavior of a circuit and thus only critical PDFs must be targeted. It is common to test the longest (functionally) testable PDFs in a circuit but this definition is accurate only under a fixed delay typical value. It has been observed in [8] that process variation (inter die and intra die) induces a large path delay variation in sub-micron technology. Using the fixed delay model the set of testable longest paths from one die is not correlated to the set from another die. They may be completely disjoint sets. The bounded gate delay can be used to tackle such issues in nanometer technology. It can also take into consideration noise induced delays (like power supply noise, simultaneous switching noise, etc.).

Techniques like [9] also study the problem of identifying the set of longest paths for testing purposes using fixed delay models. These paths do not reflect the actual set of critical paths for circuits fabricated using the sub-micron technology. Under the bounded delay model, critical PDFs are those testable PDFs whose delay may exceed a given delay threshold Dth. The threshold delay Dth is a delay bound that can be defined as a fraction (say 90%) of the clock period (identified using static timing analysis). More precisely, the delay threshold is used to eliminate the set of all testable PDFs whose delay never exceeds the threshold. The remaining testable PDFs are then defined as the set of high-risk PDFs.

This work considers the problem of finding the testable high-risk PDF set in combinational circuits. Even though the problem of identifying testable critical PDF s has been extensively

addressed in the literature, see [8],[9],[10],[19],[46] among many others, none of these methods considers primitive PDFs with reduced fault list explicitly. This work is the first to define a path delay fault model and identify critical primitive PDFs a reduced fault list using the proposed model. A major challenge in such a problem is the large number of paths needed to be considered in order to identify primitive faults, even when the problem is restricted to a small number of critical faults. The reported experimental results show that only a small number of multiple primitive PDF s is testable (when compared to the set of single primitive PDFs), implying that a small number of additional tests suffices to guarantee the circuit's timing correctness under the multiple fault criterion.

5.2. Stuck-At Fault Simulation and Analysis

We have implemented the proposed approach in C on top of the CUDD package of [15] and run on a 2.20 GHz HP-G56 personal computer with 3GB of memory. The performance of the proposed approach was experimented on the ISCAS'85 benchmarks. The results of the experimentation are presented in Table5.1. We compare the obtained results to the method of [9] and [19] due to its effectiveness in identifying all possible testable faults. Table5.4 reports comparisons on the number of faults that are tested and the reduced fault size. Column 3 of table 5.1 shows the total number reduced PDFs in the corresponding circuit. Column 4 shows the number of lines from primary input gates identified as robustly testable and Column 5 shows the number of lines from primary output gates for that method. Column 12 of Table5.1 shows the number of applied tests (test set sizes).

The BDD variable ordering provided along with the tool of [15] was used, which is optimal for most of these circuits. The circuits were processed on a primary output basis, i.e., only the part of the circuit with paths terminating to a single primary output was considered at a time. All paths in a primitive PDF terminate at the same output and, thus, no fault is missed or double counted under this scenario. The total number of critical primitive PDFs is the sum of the critical primitive PDFs in each primary output partition. As we can see from table 5.1, the fault coverage, as defined in chapter 4 is 100%.

Table 5.1: stack at fault simulation output of ISCAS'85 benchmarks

circuit	#lines in the net list	Reduced Fault size	Lines from primary input gates	Lines from primary output gates	Lines from interior gate outputs	Lines from fan-out stems	Avg. fan-in	Avg. fan-out	Max fan-in	Max fan-out	# of test vector	# s@1 faults	# s@0 faults	Coverage (%)
C5	5	22	5	2	4	6	2.00	2.00	2	2	5	5	0	100
C17	17	22	5	2	4	6	2.00	2.00	2	2	22	17	5	100
C17a	17	22	5	2	4	6	2.00	2.00	2	2	22	17	5	100
C432	432	524	36	7	153	236	2.10	2.65	9	9	1152	354	170	100
C499	499	758	41	32	170	256	2.02	4.34	5	12	2132	451	307	100
C880	880	942	60	26	357	437	1.90	3.5	4	8	1020	611	331	100
C880a	880	942	60	26	357	437	1.9	3.5	4	8	1020	611	331	100
C1355	1355	1574	41	32	514	768	1.95	2.97	5	12	3471	1275	299	100
C1355a	1355	1574	41	32	514	768	1.95	2.97	5	12	3471	1275	299	100
C6288	6288	7744	32	32	2384	3840	1.99	2.64	2	16	384	2000	5744	100

5.2.1. Path Delay Fault Simulation with Ninety Percent Threshold

Criticality (high-riskiness) is determined by a user defined delay threshold T, which is a % of the maximum path delay as determined by static timing analysis. Hence, a threshold of 80% will identify paths with delay greater than 80% of the maximum path delay. We have experimented for numerous thresholds between 80% and 90%, but due to space limitations, we only report the results for T=90% and 80%. Potentially testable critical paths were not included in order to consider a strictly critical path set. This is an implementation detail which has no impact on the conclusions drawn by the obtained results, especially since any delay model and critical path selection criterion and method can be used for this preprocessing step.

The number of paths can be an exponential function of gates. Parallel multipliers are notorious for having huge numbers of paths. One path in a DAG, specifically in BDD is one product term (a product of literals from root to the one (1) terminal). BDD are DAG with two sinks i.e., a 0 sink and a 1-sink as described in chapter four above. But in this work, only high-risk paths are considered. The risky nature in a certain path is determined based on the delay thresh hold (Dth) which is user defined. In this work 80% and 90 % of the delay threshold is considered. Any path that has a delay greater than 80% or 90% of the maximum path delay is considered high-risk.

Table 5.2 shows the number of stuck at faults at 90% threshold of different benchmark circuits. Columns 13 and 14 report the number of fall and rise delay at 90% threshold.

Table 5.2: rise and fall fault delay simulation output of ISCAS'85 benchmarks at 90% threshold

Circuit	#lines in the net list	Reduced Fault size	Lines from primary input gates	Lines from primary output gates	Lines from interior gate outputs	Lines from fan-out stems	Avg. fan-in	Avg. fan-out	Max fan-in	Max fan-out	# of test vector	Fall delay @90%	Rise delay@90%	Coverage (%)
C5	5	22	5	2	4	6	2.00	2.00	2	2	5	1	0	100
C17	17	22	5	2	4	6	2.00	2.00	2	2	22	2	1	100
C17a	17	22	5	2	4	6	2.00	2.00	2	2	22	17	5	100
C432	432	524	36	7	153	236	2.10	2.65	9	9	1152	36	17	100
C499	499	758	41	32	170	256	2.02	4.34	5	12	2132	46	31	100
C880	880	942	60	26	357	437	1.90	3.5	4	8	1020	62	32	100
C880a	880	942	60	26	357	437	1.9	3.5	4	8	1020	62	32	100
C1355	1355	1574	41	32	514	768	1.95	2.97	5	12	3471	128	30	100
C1355a	1355	1574	41	32	514	768	1.95	2.97	5	12	3471	128	30	100
C6288	6288	7744	32	32	2384	3840	1.99	2.64	2	16	384	200	575	100

5.3. Faulty Blif File Analysis Using CUDD

The source code in appendix is added to the Nanotrav folder of the cudd package for faulty blif file analysis. Nanotrav is a simple reachability analysis tool based on the CUDD package. Nanotrav together with the added source code reads a circuit written in a small subset of blif then creates BDDs for the primary outputs and the next state functions (if any) of the circuit. If passed the –trave option, nanotrave builds BDD (OBDDs) for the characteristics function of the transition relation of the graph. It then builds a BDD for the initial states, and performs reachability analysis. Once it has completed reachability analysis, nanotrave prints results and exits [32]. The results have many sets of outputs like, number of min-terms (product terms), depth of the BDD from which we decided the highest length of the path in the given circuit and many more. In this paper, we considered the worst case scenario (with bad reordering case, i.e., when there are maximum number of paths and the maximum possible length can exist. As we can see from table 5.3, the number of paths is very large, so that instead of remuneratively list all the paths, it is technical to consider a set of high-risk paths using a certain threshold (user defined threshold, i.e. 80% and 90%) to find another method to minimize them for optimized synthesis.

Table 5.3: ISCAS'85, The CUDD built in benchmark circuit and ITC'99 circuit analysis using the proposed delay model

Circuit Name	NO. of Inputs	NO. of Outputs	NO. Of Paths (NP) =NO. Min-terms	Deepest Length (DL)	NP-90%(DL)= high risk paths	CPU time (sec) At 90% threshold
c17	3	2	18	11	9	negligible
C880	60	9	1.44115e+17	4625	1.144115e+17	0.4
mult32a	65	1	4.29497e+09	275	429496725	0.2
s27	7	1	6	2	4	negligible
s298	17	6	5800	125	5675	negligible
rcn25	25	25	1.95734e+07	869	19572531	71.1
vg2	25	8	1.63492e+07	1044	16348156	0.00

To take CPU time as a performance measure, CPU time or CPU usage can be reported either for each thread, for each process or for the entire system. Moreover, depending on what exactly the CPU was doing, the reported values can be subdivided in:

- **User time** is the amount of time the CPU was busy executing code in user space.
- **System time** is the amount of time the CPU was busy executing code in kernel space. If this value is reported for a thread or process, then it represents the amount of time the kernel was doing work on behalf of the executing context, for example, after a thread issued a system call.
- **Idle time** (for the whole system only) is the amount of time the CPU was not busy, or, otherwise, the amount of time it executed the System Idle process. Idle time actually measures unused CPU capacity.
- **Steal time** (for the whole system only), on virtualized hardware, is the amount of time the operating system wanted to execute, but was not allowed to by the hypervisor. This can happen if the physical hardware runs multiple guest operating systems and the hypervisor (virtual machine monitor) chose to allocate a CPU time slot to another one.

But the CUDD package uses CPU time for user time and system time only. So, the CPU time reported in this paper is the sum of user time and system time only. Even if the simulation environment used for this paper and in [9] and [19] is different, it is reasonable to compare the CPU exploitation power of the approaches in these three works. Explicit comparison is made with the approach [9] and [19]. For comparison purposes, Table 5.4 lists the CPU execution time required when the method proposed in [9] and [19] were executed.

Table 5.4 Comparison of the proposed technique with [9] and [19]

Benchmark	# of Test vectors in this work	# of test vectors in [9] given	# of test vectors in [19] is # single PDFs	High-Risk paths in this work at 80% threshold	CPU time (in sec)in this paper at 80% threshold	CPU time(sec) in [19] at 80% threshold	CPU time (sec) in [9] at 80% threshold	CPU Time in this work at 80% by [19] test vector	CPU time(sec) in this work at 80% by [9] test vector
C880	1020	1596	-	1.14391e+18	0.67	-	2.54	-	1.47
C5315	3640	4038	2682610	4.29497e+09	0.45	174.35	22.28	145.4	0.67
C2670	1865	2403	1359920	1.63492e+07	0.23	290.17	6.03	187.8	4.54
C1908	1879	7647	1458114	5800	Negligible	1650.27	8.48	1345.9	6.35
C7552	7550	16,121	1452988	3.35544e+07	0.75	417.74	22.70	345	13.73

The results in table 5.4 show clearly the superiority of the presented method over existing approaches that obtain exact PDF coverage. They also clearly demonstrate that the basic scheme suffices when test sets obtained from existing ATPG tools are applied for currently available benchmarks. This is because the simulation environment used in this paper is lower in performance (has lower specification that the one stated in [9] and [19]) than the one used in those indicated references i.e. [9] and [19]. Boolean problems like problems that are solved using decision diagrams (ROBDD) are NP-complete (Non-Deterministic polynomial complete) problems. That means, in computational complexity theory, a decision problem is NP-complete when it is both in NP and NP-hard. The abbreviation NP refers to "nondeterministic polynomial time". Although any given solution to an NP-complete problem can be verified quickly (in polynomial time), there is no known efficient way to locate a solution in the first place; indeed, the most notable characteristic of NP-complete problems is that no fast solution to them is known. So by using different reductions like what we did in this thesis fault reduction, test vector reduction and using efficient data structures like ROBDD, we can get a fast solution to our problem(this is because heuristic is one type of NP-complete problem). That is, the main difference between [9], [19] and this paper is that even though all papers deal with NP-complete problems, but our work uses reduction techniques to get a fast solution. Using this idea, our CPU

time becomes smaller relative to [9] and [19]. For example, if we see c1908, in [9] there are 7647 test vectors which leads to 8.48 sect CPU time in [9] and 6.35 in our work which is lower relative to [9] due to the efficiency of ROBDD, and the same is true for the others.

So we conclude that our approach, i.e. using ROBDD for path delay fault analysis using fault reduction strategy is more power full than the one indicated in literatures like [9] and [19].

Chapter Six

6. Conclusion and Recommendation for Future Works

6.1. Conclusion

We have presented a Verilog based stuck at fault model for combinational circuits. The delay fault model uses buffer insertion technique to create a delay effect in digital circuits. The performance of the proposed model and simulation scheme is analyzed using different ISCAS'85 and CUDD built-in blif format [48] benchmark circuits. Quantitatively the simulation method is validated against the amount of the CPU power it takes and found satisfactory.

The proposed delay model is easy to understand and visualize using small combinational circuits with simple Verilog simulators. Since it is easy to visualize the difference between the faulty model and the good circuit model using wave analyzers as depicted in section 3.5, the model is efficient than the path delay, gate transition delay, and other delay models that are found in literatures [9] [19][37][38]. Since the buffer insertion is user defined, it is easy to use the proposed delay model in large circuits to analyze the presence of slow to rise and fall delay faults on combinational circuits.

With the progress of semiconductor technology, modeling and testing of VLSI circuits becomes more and more difficult and at the same time cost is also increasing. Therefore it is important to achieve high fault test efficiency with low cost. With this approach RTL designer can have an estimation of the achieved fault coverage before doing synthesis and also it is possible for the designer to locate faults at a higher level of abstraction. At present this approach is applied to combinational logic circuit stuck at fault modeling and analyzing the fault effect.

The testable high-risk PDF set is defined and computed for any potentially testable critical PDF set using the user defined threshold approach in a non-enumerative fashion. It is shown that the non-enumerative path delay fault coverage problem reduces to a simple sequence of basic and efficient OBDD operations. The presented approach is very simple and benefits from the ability of ordered BDDs to store extremely large numbers of PDFs.

6.2. Recommendations to Future Works

Silicon fault modeling is an open area of research with continuous advancements [1-50]. Often, new fault models are devised targeting emerging silicon failure modes or more accurately modeling existing failure mechanisms. Therefore, it is crucial that an analysis framework's existing fault model collection could be easily upgraded with new fault models. There is no universally accepted delay fault model that represented all available delay faults so that new delay models which are feasible to implement for combinational as well as sequential circuits are needed.

Fault Masking is an occurrence, in which one defect prevents the detection of another. It's a used technique (by developers) to prevent error result from appearing. Fault masking is loss of fault detection –the fault effect cannot be propagated out to a detection point (observability problem) [IEEE610]. So how to model those masked faults and analysis their side effect is a question left for future researchers.

REFERENCES

[1]. Laung-Terng Wang et al., *VLSI Test principles and Architectures: Design For testability,* Morgan Kaufmann Publishers,500 Sansome Street, Suite 400, San Francisco, CA 94111, 2006, pp 9-193

[2]. Israel Koren et al., *Fault Tolerant Systems*, Morgan Kaufmann Publishers, 500 Sansome Street, Suite 400, San Francisco, CA 94111, 2007, pp 11-182

[3]. Michael L.Bushnell et al., *Essentials of Electronic Testing for Digital, Memory and Mixed Signal VLSI circuits*, Kluwer Academic Publishers, USA, 2000, PP 24-204 and PP 417-436.

[4]. Christoph Meinel et al., *Algorithms and Data Structures in VLSI Design: OBDD-Foundations and Applications*, Springer-Verlag, Berlin Heidelberg,1998, pp 63-250

[5]. Zainalabedin Navabi, *Digital System Test and Testable Design Using HDL Models and Architectures*, Springer Science and Business Media, 233 Spring Street, New York, NY 10013, USA, 2011, pp 21-343

[6]. Raja K.K.R et al.," Using Hierarchy in Design Automation: The Fault Collapsing Problem", Proc. 11th VLSI Design and Test Symp. (VDAT), Kolkata, August 8-11, 2007, pp174-184,

[7]. Cedell A. Alexander et al., "Near-critical Path Analysis: A Tool For Parallel Program Optimization", The University of Southern Mississippi, Proceedings of the First Southern Symposium on Computing, December 4-5, 1998

[8]. Kyriakos Christou et al., "On the Use of ZBDDs for Implicit and Compact Critical Path Delay Fault Test Generation", J Electron Test conf., Springer Science and Business Media, 2007

[9]. Saravanan Padmanaban and Maria K. Michael, "Exact Path Delay Fault Coverage with Fundamental Zero-Suppressed BDD Operations", Southern Illinois University,International Test Conference, 2001

[10]. Fatih Kocan et al., "Static Variable Ordering in ZBDDs for Path Delay Fault Coverage Calculation", Southern Methodist University, Circuits and Systems, MWSCAS'04, The 2004 47th Midwest Symposium vol. 1,

[11]. Sujit Dey and Li Chen , "Embedded Software-Based Self-Test for Programmable Core-Based Designs" , IEEE Design & Test of Computers, July–August 2002

[12]. Kamran Zarrineh, "System-on-Chip Testability Using LSSD Scan Structures",IEEE Design & Test of Computers, May–June, 2001

[13]. A.Kristic, et al.,"Embedded Software-Based Self-Testing for SOC Design", DAC 2002, June 10-14,New Orleans,Louisiana,USA. Copyright 2002 ACM1-58113-461-4/02/000.2002

[14]. Hoon Chang et al., "An Efficient Critical Path Tracing Algorithm for Designing High Performance VLSI Systems"(on-line), Journal of Electronic Testing: Theory and Applications KL491-03-Chang September 15, 1997

[15]. F.Somenzi, "CUDD: CU Decision Diagram Package", Dept. of ECE, The University of Colorado, 1999, http://vlsi.colorado.edu/~fabio/CUDD/

[16]. Vishwani D.Agrawal, *Advances in Electronic Testing challenges and Methodologies (Frontiers in Electronic Testing)* (First Edition), Published by Springer,P.O. Box 17, 3300 AA Dordrecht, The Netherlands. , September, 2006

[17]. S.Jha, Y.Lu, "Equivalence Checking Using Abstract BDDs", Carnegie Mellon University, Computer Science Department, IEEE,1997

[18]. Hyungwon Kim and John P. Hayes , "Delay Fault Testing of IP-Based Designs Via Symbolic Path Modeling", international test conference, February 1999

[19]. Kyriakos Christou et al., " Identification of Critical Primitive Path Delay Faults Without Path Enumeration ", University of Cyprus, IEEE VLSI test 28th symposium, 2010

[20]. Randal E. Bryant, "Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification", Carnegie Mellon University, International Conference on Computer-Aided Design (ICCAD), November, 1995

[21]. Wei-Cheng Lai et al., "On Testing the Path Delay Faults of a Microprocessor Using its Instruction Set", Department of ECE, University of California, Santa Barbara, IEEE, 2000

[22]. Virendra Singh et al., "Software-Based Delay Fault Testing of Processor Cores", University of Wisconsin, 12th Asian Test Symposium, ATS, 2003.

[23]. Mariusz Wegrzyn et al., "Functional Testing of Processor Cores In FPGA-Based Applications", Computing and Informatics, Vol. 28, 2009, pp 97-113

[24]. Qiang Xu and Nicola Nicolici, "Delay Fault Testing of Core-Based Systems-on-a-Chip", Department of Electrical and Computer Engineering, IEEE ,2003

[25]. Vlado Vorisek et al., "At-speed Testing of SOC ICs", Motorola Munich, IEEE, 2004

[26]. Takkellapati Venu Gopi et al., "High secured and area optimized Online Memory Testing for efficient Fault Diagnostic Systems", International Journal of Engineering Trends and Technology- Volume 4 Issue 2, 2013

[27]. Hussain Al-asaad et al, "Online BIST for Embedded Systems", IEEE Design & Test of Computers, October–December 1998

[28]. Philemon Daniel and Rajeevan Chandel, "Reconfigurable Test Architecture for Online Concurrent Fault Detection, Diagnosis and Repair", International Journal of Design, analysis and tools for integrated circuits and systems, VOL. 3, NO. 1, MARCH,2012

[29]. Kee Sup Kim et al., "Delay Defect Characteristics and Testing Strategies", IEEE Design & Test of Computers,2003

[30]. V.J.Veermak, H.G.Kerkhoff, "Enhanced P1500 Compliant Wrapper Suitable for Delay Fault Testing of Embedded Cores", IEEE Computer Society Washington, 2003

[31]. Ms. Sankari.M.S and Mr.P.SathishKumar, "Oscillation Test Methodology for Built-In Analog Circuits" International Journal Of Computational Engineering Research**,** May-June 2012

[32]. Zarrineh, K et al., "System-on-chip testability using LSSD scan structures", Sun Micro-electron ,IEEE, May, 2001

[33]. S.Prasanna, K.Suganthi1,"Implementation of Delay Measurement Technique Using Signature Register For Small-Delay Defect Detection", IJRET, volume 2,April, 2013

[34]. Irith Pomeranz and Sudhakar M. Reddy, "Functional Test Generation for Delay Faults in Combinational Circuits",University of Iowa, ICCAD '95 Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design

[35]. Parad K. Lala, *An Introduction to Logic Circuit Testing*, Texas A&M University-Texarkana, Morgan & Claypool Publishers, 2009,

[36]. Matthias Beck et al., "Logic Design for On-Chip Test Clock Generation Implementation Details and Impact on Delay Test Quality", IEEE,2005

[37]. Nisar Ahmed and Mohammad Tehranipoor, "Improving Transition Delay Fault Coverage Using Hybrid Scan-Based Technique", 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2005

[38]. Hillary Grimes and Vishwani D. Agrawal, "Analyzing Reconvergent Fanouts in Gate Delay Fault Simulation", Auburn University, Proc. 17th IEEE North Atlantic Test Workshop,2008

[39]. Angela Krastic and Kwang-Ting Cheng, *Delay Fault Testing for VLSI circuits(Frontiers in Electronic Testing)* (First Edition), University of California, Springer, 1998

[40]. Raimund Ubar et al., "Parallel X-Fault Simulation with Critical Path Tracing Technique", Department of Computer Engineering ,Tallinn University of Technology, EDAA, 2010

[41]. Mathias Beck et al., "Logic Design for On-Chip Test Clock Generation-Implementation Details and Impact on Delay Test Quality", Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, 2005

[42]. Kee Sup Kim et al., "Delay Defect Characteristics and Testing Strategies", IEEE Design & Test of Computers,2003

[43]. Andrew B.Kahng et al., *VLSI Physical Design: From Graph Partitioning to Timing Closure* (First Edition), Springer Science and Business Media B.V., 2011,pp 221-246

[44]. Zarrineh, K et al., "System-on-chip testability using LSSD scan structures", Sun Micro-electron ,IEEE, May, 2001

[45]. Hansen et al., "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering", Design & Test of Computers, IEEE (Volume:16 , Issue: 3), 1999

[46]. Kapoor, Bhanu," Computing exact path delay-fault coverage using OBDDs", Circuits and Systems, 1994., Proceedings of the 37th Midwest Symposium on (Volume:1)

[47]. Jyoti Kukreja,“ Application of Binary Decision Diagram in digital circuit analysis“ ,Unpublished, University of Southern California,

[48]. Sandeep Koranne*, Handbook of Open Source Tools,* DOI 10.1007/978-1-4419-7719-9_15, Springer Science and Business Media, 2011, pp 317-347

[49]. Stephan Eggersgl.u,G.orschwin et al., “Monsoon, SAT-Based ATPG for Path Delay Faults Using Multiple-Valued Logics”, Journal of Electronic Testing - Theory and Applications manuscript, January 2010

[50]. P. Molitor, J. Mohnke, *Equivalence Checking of Digital Circuits: Fundamentals, Principles, Methods*, Springer Science and Business Media, USA, 2004

APPENDIX: A Blif file traversal Program partially

The goal of BLIF [48] is to describe a logic-level hierarchical circuit in textual form. A circuit is an arbitrary combinational or sequential network of logic functions. A circuit can be viewed as a directed graph of combinational logic nodes and sequential logic elements. Each node has a two-level, single-output logic function associated with it. Each feedback loop must contain at least one latch. Each net (or signal) has only a single driver, and either the signal or the gate which drives the signal can be named without ambiguity.

The following source code package is used to read and interpret the any blif file in to the CUDD [15] package for circuit analysis.

```
/**************************************************************

* Package for reading blif in to the BDD package of [32] for ROBDD

 *

 **************************************************************/

//#include "cudd-2.5.0/nanotrav/bnet.h"

#include "ntr.h"

//#include "cudd-2.5.0/include/util.h"

//#include "cudd-2.5.0/include/cudd.h"

#include "cuddInt.h"

#include "blif.h"


/**************************************************************************
*/

FILE *

open_file(
```

```
  char * filename,
  char * mode)
{
    FILE *fp;
    if (strcmp(filename, "-") == 0) {
        return mode[0] == 'r' ? stdin : stdout;
    } else if ((fp = fopen(filename, mode)) == NULL) {
        perror(filename);
        exit(1);
    }
    return(fp);
} /* end of open_file */
/**********************************************************************
/
DdManager *
startCudd(
  NtrOptions * option,
  int  nvars)
{
    DdManager *dd;
    int result;
```

```
dd = Cudd_Init(0, 0, option->slots, option->cacheSize, option->maxMemory);
if (dd == NULL) return(NULL);
if (option->maxMemHard != 0) {
        Cudd_SetMaxMemory(dd,option->maxMemHard);
}
Cudd_SetMaxLive(dd,option->maxLive);
Cudd_SetGroupcheck(dd,option->groupcheck);
if (option->autoDyn & 1) {
        Cudd_AutodynEnable(dd,option->autoMethod);
}
dd->nextDyn = option->firstReorder;
dd->countDead = (option->countDead == FALSE) ? ~0 : 0;
dd->maxGrowth = 1.0 + ((float) option->maxGrowth / 100.0);
dd->recomb = option->recomb;
dd->arcviolation = option->arcviolation;
dd->symmviolation = option->symmviolation;
dd->populationSize = option->populationSize;
dd->numberXovers = option->numberXovers;
result = ntrReadTree(dd,option->treefile,nvars);
if (result == 0) {
        Cudd_Quit(dd);
```

```
        return(NULL);
    }
#ifndef DD_STATS
    result = Cudd_EnableReorderingReporting(dd);
    if (result == 0) {
        (void) fprintf(stderr,
                "Error reported by Cudd_EnableReorderingReporting\n");
        Cudd_Quit(dd);
        return(NULL);
    }
#endif
    return(dd);
} /* end of startCudd */
NtrOptions *
mainInit(
  )
{
    NtrOptions  *option;

    /* Initialize option structure. */
    option = ALLOC(NtrOptions,1);
```

```
option->initialTime    = util_cpu_time();
option->verify         = FALSE;
option->second         = FALSE;
option->file1          = NULL;
option->file2          = NULL;
option->traverse       = FALSE;
option->depend         = FALSE;
option->image          = NTR_IMAGE_MONO;
option->imageClip      = 1.0;
option->approx         = NTR_UNDER_APPROX;
option->threshold      = -1;
option->from           = NTR_FROM_NEW;
option->groupnsps      = NTR_GROUP_NONE;
option->closure        = FALSE;
option->closureClip    = 1.0;
option->envelope       = FALSE;
option->scc            = FALSE;
option->maxflow        = FALSE;
option->shortPath      = NTR_SHORT_NONE;
option->selectiveTrace = FALSE;
option->zddtest        = FALSE;
```

```
option->printcover     = FALSE;
option->sinkfile       = NULL;
option->partition      = FALSE;
option->char2vect      = FALSE;
option->density        = FALSE;
option->quality        = 1.0;
option->decomp         = FALSE;
option->cofest         = FALSE;
option->clip           = -1.0;
option->dontcares      = FALSE;
option->closestCube    = FALSE;
option->noBuild        = FALSE;
option->stateOnly      = FALSE;
option->node           = NULL;
option->locGlob        = BNET_GLOBAL_DD;
option->progress       = FALSE;
option->cacheSize      = 32768;
option->maxMemory      = 0;         /* set automatically */
option->maxMemHard     = 0; /* don't set */
option->maxLive        = ~0; /* very large number */
option->slots          = CUDD_UNIQUE_SLOTS;
```

```
option->ordering       = PI_PS_FROM_FILE;
option->orderPiPs      = NULL;
option->reordering     = CUDD_REORDER_NONE;
option->autoMethod     = CUDD_REORDER_SIFT;
option->autoDyn        = 0;
option->treefile       = NULL;
option->firstReorder   = DD_FIRST_REORDER;
option->countDead      = FALSE;
option->maxGrowth      = 20;
option->groupcheck     = CUDD_GROUP_CHECK7;
option->arcviolation   = 10;
option->symmviolation  = 10;
option->recomb         = DD_DEFAULT_RECOMB;
option->nodrop         = TRUE;
option->signatures     = FALSE;
option->verb           = 0;
option->gaOnOff        = 0;
option->populationSize = 0;          /* use default */
option->numberXovers   = 0;          /* use default */
option->bdddump        = FALSE;
option->dumpFmt        = 0;   /* dot */
```

```
    option->dumpfile      = NULL;
    option->store         = -1; /* do not store */
    option->storefile     = NULL;
    option->load          = FALSE;
    option->loadfile      = NULL;
    return(option);
} /* end of mainInit */
/**************************************************************************
/
int
ntrReadTree(
  DdManager * dd,
  char * treefile,
  int  nvars)
{
    FILE *fp;
    MtrNode *root;
    if (treefile == NULL) {
            return(1);
    }
    if ((fp = fopen(treefile,"r")) == NULL) {
```

```
        (void) fprintf(stderr,"Unable to open %s\n",treefile);
        return(0);
    }
    root = Mtr_ReadGroups(fp,ddMax(Cudd_ReadSize(dd),nvars));
    if (root == NULL) {
        return(0);
    }
    Cudd_SetTree(dd,root);
    return(1);
} /* end of ntrReadTree */
/**************************************************************************
 * Print the BDD representation of the function
**************************************************************************
/
int
printBdd (
  BnetNetwork      *net1,       // Circuit network created from blif file
  DdManager      *dd          // BDD of the circuit network
  )
{
   int          noutput;      // number of output node
```

```
char          **outputs;    // array of primary variable name
char          *name;        // a primary output name
BnetNode      *bnode;       //
BnetNode      *xnode;
int           nvars;        // number of variables
int           i;
DdNode        *dnode;
  // print variable order
printf(" Variable Order: \n");
Bnet_PrintOrder(net1, dd);
printf(" \n");
// get variable number
nvars = Cudd_ReadSize(dd);
noutput = net1->noutputs;
outputs = net1->outputs;
for(i=0; i<noutput; i++)
 {
  name = outputs[i];
  printf(" Primary output: %s \n",  name);
     // traversal the linked list
  xnode = NULL;
```

```
    for(bnode=net1->nodes; bnode!=NULL; bnode=bnode->next)
    {
     if(strcmp(bnode->name, name)==0)
     {
      xnode =bnode;
      break;
     }
    }
       if(xnode!=NULL)
     {
       dnode = xnode->dd;
           // print
           Cudd_PrintDebug(dd, dnode, nvars, 4);
     }
       }
    return 0;
}
```