Upgradation of Design for Testability (DFT) Analysis Flow

Thesis submitted in partial fulfillment of the requirement for the award of degree of

Master of Engineering
in
Electronic Instrumentation and Control

By:
Gurkiran Kaur
(Roll No. 80751011)

Under the supervision of:
Mr. Bhanu Prakash
Section Manager, ST Microelectronics India Pvt Ltd

and

Ms. Gagandeep Kaur
Lecturer (SG), EIED

Department of Electrical and Instrumentation Engineering
THAPAR UNIVERSITY
PATIALA – 147004

July 2009

Declaration

I hereby declare that the report entitled "Upgradation of Design for Testability (DFT) Analysis flow" is an authentic record of my own work carried out as requirements for the award of degree of M.E Electronic Instrumentation & Control at Thapar University, Patiala under the supervision of Mr. Bhanu Prakash, Section Manager, ST Microelectronics Pvt Ltd & Ms. Gagandeep Kaur, Senior Lecturer, EIED during January to July 2009.

Name : Gurkiran Kaur

Roll No. : 80751011

Date: 15/07/09

It is certified that the above statement made by the student is correct to the best of our knowledge & belief

Mr. Bhanu Prakash
Section Manager, DSMG
ST Microelectronics India Pvt Ltd, Noida

Ms. Gagandeep Kaur
Senior Lecturer, EIED
Thapar University, Patiala

Dr. Smarajit Ghosh
Professor & Head, EIED
Thapar University, Patiala

Dr. R. K. Sharma
Dean of Academic Affairs
Thapar University, Patiala

Acknowledgment

The real spirit of achieving a goal is through the way of excellence and austere discipline. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by various personalities

With deep sense of gratitude I express my sincere thanks to my esteemed and worthy supervisors, Ms. Gagandeep Kaur, Senior Lecturer, Department of Electrical and Instrumentation Engineering, Thapar University, Patiala and Mr. Bhanu Prakash, Section Manager, ST Microelectronics India Pvt. Ltd., for their valuable guidance in carrying out this work under their effective supervision, encouragement, enlightenment and cooperation.

I shall be failing in my duties if I do not express my deep sense of gratitude towards Mr. Smarajit Ghosh, Professor & Head of the Department of Electrical & Instrumentation Engineering, Thapar University, Patiala who has been a constant source of inspiration for me throughout this work.

Acknowledgement is due to Home Video Division at ST Microelectronics Pvt Ltd, Greater Noida, UP, India for providing me the necessary software and other resources to deliver my research work.

My work would not have been possible without the helpful and insightful guidance of my team members, Mr. Arvind Kumar, Mr. Azad Singh, Mr. Anshuman Bansal, Mr. Udit Kumar and Mr. Johann Meleard. Working with them was a great learning experience. Most of the novel ideas and solutions found in this thesis are the result of our numerous stimulating discussions. I am thankful to them for helping me in every possible way and making my stay at ST Microelectronics memorable.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the ALMIGHTY who bestowed self-confidence, ability and strength in me to complete this work

Place: Thapar University, Patiala *Gurkiran Kaur*

Date:

Abstract

The work proposed in this thesis was done at ST Microelectronics Pvt Ltd, Greater Noida, U.P, India with Design Support and Methodology (DSM) team at Home Video Division (HVD). This team comprises of ex-designers who are now entrusted to develop and maintain the design flow, as used by HED designers worldwide. They are looking into full spectrum of Electronic Design Automation (EDA) flow. One of such EDA tools is TetraMax which is a high speed, high capacity test validation and pattern generation tool

The proposed work deals with the study of design flow using DFT Compiler and TetraMax ATPG tool and consequent upgradation of Design for Testability (DFT) analysis flow at HVD. This is required to maintain synchronous methodology during TetraMax run across various designs as well as various sites of ST Microelectronics all over the world. In the present work, we propose to develop internal STB tool which would uniquify the DFT procedures. The tool proposed is expected to check the design for testability after first synthesis itself so that DFT issues are discovered well in advance in the development of a design. In addition, customized test reports generated by the tool will help the designer to identify where he can simplify logic cones and do a specific design optimization. Therefore this will prove to be of great help to backend designers who can use it to discover DFT issues well in advance in development process of a design.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ATE	Automatic Test Equipment
ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BIST	Built in Self Test
CAD	Computer Aided Design
CLK	Clock
CLKGEN	Clock Generator
CMOS	Complementary Metal Oxide Semiconductor
DFT	Design for Testability
DRC	Design Rule Checks
DL	Defect Level
EDA	Electronic Design Automation
FF	Flip Flop
GSV	Graphical Schematic Viewer
GUI	Graphical User Interface
HDL	High Level Descriptive Language
IEEE	Institute for Electrical and Electronic Engineers
IO	Input Output
JTAG	Joint Test Action Group
LSSD	Level Sensitive Scan Design
LB	Lower Bound
MCM	Mixed Signal Multi Chip Modules
MUX	Multiplexer
PI	Primary Input
PO	Primary Output
PLL	Phase Locked Loop
RTL	Register Transfer Logic
RAM	Random Access Memory
SA	Stuck At

SSF	Single Stuck-At Fault
UB	Upper Bound
VHDL	Very High Speed Integrated Circuit Hardware Descriptive Language
VLSI	Very Large Scale Integration

Literature Survey

Vivek Chickermane [1] et.al introduces comparative study of a gate-level test generator and a high-level test generator in this paper. Based on an evaluation of the results, techniques to automatically extract information from the high level circuit description have been proposed. These techniques could improve the performance of both ATPG tools. Description of an automatic DFT tool that utilizes VHDL descriptions of the circuit to make an intelligent selection of flip flops for partial scan have been given. It is concluded that DFT tool can make a more efficient and effective selection of partial scan flip-flops by using the high-level circuit information. It can accurately predict hard to test areas in the circuit early in the design phase. Significant improvements in fault coverage and ATPG efficiency, and speedups in ATPG time can be obtained by a gate-level and a high-level test generator after high-level scan selection

T.W Williams [2] et.al discusses the basics of design for testability in this paper. A short review of testing is given along with some reasons as to why one should test. The different techniques of design for testability are discussed in detail. These include techniques which can be applied to today's technologies and techniques which have been recently introduced and will soon appear in new designs. These techniques include the three main areas of Design for Testability, 1) Ad Hoc approaches 2) Structured approaches and, 3) Self Test/Built-in Test approaches

Irith Pomeranz [3] et.al describes a design for testability (DFT) approach for synchronous sequential circuits that combines scan with nonscan DFT in a transparent way. DFT control inputs and scan chain inputs are used as primary inputs of the circuit, and scan chain outputs are used as primary outputs of the circuit during test generation to eliminate the distinction between functional clock cycles and the various types of non-functional clock cycles. The result is 1) short test application times due DFT modes and the ability to use limited scan operations and 2) the ability to detect all the combinationally irredundant faults due to the scan mode

A. Chatterjee [4] et.al discusses the problem of area and performance overhead, fault model coverage, the DFT/BIST automation process, and interface to digital tools in analog and mixed signal DFT and BIST. The study reveals that already very complex mixed-signal multi-chip modules (MCMs) are being designed. Such MCMs of the future will contain buried passives, optoelectronic interconnections, RF components and very closely coupled digital and analog parts.

Sanghyeon Baeg [5] et.al describes a novel DFT scheme of controlling clock lines in this paper. CLC divides the test generation problem into small segments by selectively clocking sequential modules. A new conflict solving technique, time frame split is implemented in a test generator. Time frames are expanded in the unit of CCB during test generation. It is shown that it can be easily extended to high-level test generation. Sequential test generation using clock control have also been shown to improve in CPU time for test generation and test efficiency

Indradeep Ghosh [7] et.al presents in this paper an effective and practical technique for testing systems composed of predesigned embedded cores. The method first makes each individual core highly testable using some core-level DFT hardware and precomputes a test set for each core. Then it does some further analysis on each core and adds more DFT hardware, if necessary, to make it transparent so that test data can be propagated through it during system test without information loss. Finally, the system is modified with some global test hardware, and a test schedule is found for testing each embedded core with its precomputed test set. This system-level testing strategy is independent of the way in which the individual cores are tested. On applying the test strategy to some example core-based systems, it has been found that the average area overhead for the system-level DFT (5.9%) is much lower than existing system-level DFT techniques for core-based systems. The average delay overhead is negligible. The test efficiency obtained on the part of the circuit which could be accurately fault simulated is very high (>99.5%). The test application time is much lower than what would be required by current industrial techniques of employing cores with full scan and either test bus or boundary scan for system-level DFT

Samah Hassan [8] et.al present in this paper a new tool that performs ATPG directly from VHDL behavioural descriptions. Performance analysis shows that the patterns generated by this behavioral ATPG tool achieve high fault coverage when tested on benchmark circuits. The experimental results obtained show that behavioral ATPG is able to generate high quality test sets, which provide a fault coverage comparable with that of the traditional ATPG. Although the system was mainly implemented for ATPG purposes it can also be used for verification, and optimization of tasks.

Vishal J. Mehta [9] et.al proposes in this paper a novel approach for performing delay-fault diagnosis forrobust and nonrobust tests. Diagnostic resolution is enhanced by utilizing passing patterns, processing failure logs at various slower frequencies, and applying *n*-detection and

timing-aware automatic test pattern generation sets. Experimental results show that this approach can diagnose delay faults with good resolution. The algorithm is stable with respect to delay variations that manufactured chips might experience

Maria K. Michael [10] et.al presents in this paper, a function-based nonenumerative automatic test pattern generation (ATPG) methodology for detecting path delay faults. The proposed technique consists of a number of topological circuit traversals during each a linear number of Boolean functions is generated per circuit line. From each such function we derive a test that detects many PDFs. The two major strengths of the approach, that stem from the function based formulations used, are very compact test sets, and scalability in test efficiency. The performance of an implementation based on binary decision diagrams is evaluated and compared with existing compact methods to demonstrate the superiority of the proposed method

Amitav Majumdar [12] et.al presents several novel results in this paper which can be applied to the problems of fault coverage and test length estimation in combinational circuits under random or pseudorandom test. The uniqueness of these results lie in the fact that expressions for the distribution functions of the relevant quantities are derived. While earlier techniques allowed estimation of only expected values of fault coverage or bounds for test length in restricted cases, these theoretical results allow estimation of all important statistics, including variance and other confidence intervals. A technique is developed for estimating its expected value and variance for any desired coverage specification. Experiments prove that these theoretical results can be used to provide excellent estimates of test length. In these respects the results and techniques presented in this paper significantly enhance the state of the art in test length as well as fault coverage estimation. Furthermore, a way of improving accuracy is also proposed by considering a sampling model for fault equivalence classes

Von-Kyoung Kim [14] et.al proposes a new fault coverage estimation model in this paper which can be used in the early stage of VLSI design. The fault coverage model is an exponentially decaying function with three parameters, which include the fault coverage upper bound, UB, the fault coverage lower bound, LB, and the rate of fault coverage change. The fault coverages using three diferrent testing scenarios, which are no DFT, scan, iddq testing, are predicted using circuit design information, such as gatecount, IO count, and FF count Experimental result showed a 1.9% model estimation error with a given circuit information in the early design.

Ananta K. Majhi [15] et.al in this paper have reviewed delay fault models, discussed their classifications and have examined fault coverage metrics that have been proposed in the recent literature. A comparison between delay fault models, namely, gate delay, transition and path delay, line delay and segment delay faults, is done and their benefits and limitations are shown. Various classifications of the path delay fault model that have received the most attention in recent years, are reviewed. It is concluded that an understanding of delay fault models is essential in today's VLSI design and Test environment.

T. Riesgo [17] et.al presents in this paper, a fault model for VHDL descriptions at the Register Transfer Level and its evaluation with respect to a logic level fault model (single-stuck-at). The proposed fault model may be used for early estimations of the fault coverage before the synthesis is made in the design process of an integrated circuit. The obtained results show a high correlation between the fault coverages achieved with the proposed fault model and logic fault models on a set of examples. The main contribution of this work is the proposal of a new fault model for VHDWRT descriptions and the demonstration of its usefulness for estimating the achieved fault coverage with a set of test vectors in design phases previous to synthesis

Emilio Gaudette [18] et.al presents a method to evaluate a fault model by comparing its fault coverage to error coverage based on a subset of design errors. This type of evaluation is essential if new fault models are to be accepted and used in an industrial setting. It is proposed that the analysis which is presented will be expanded in the future to reveal detailed strengths and weaknesses of existing fault models and to provide direction for the development of new fault models in the future.

Irith Pomeranz [19] et.al presents in this paper a transition fault model which is easy to simulate under test sequences that are applied at-speed, and provides a target for the generation of at-speed test sequences. At-speed test application allows a circuit to be tested under its normal-operation conditions. However, fault simulation and test generation for standard transition faults become significantly more complex due to the need to handle faulty signal transitions that span multiple clock cycles. As a result, each transition fault needs to be considered multiple times, with multiple sizes of the extra delay on the faulty line. The proposed fault model alleviates this shortcoming by introducing unspecified values into the faulty circuit when fault effects may occur, thus allowing faults of all possible sizes to be encompassed in a single fault. Fault detection potentially occurs

when an unspecified value reaches a primary output. “Pessimistic,” “optimistic,” and “random” versions of the fault model and corresponding fault coverages are defined. If a single fault coverage is to be computed, the pessimistic one provides the lowest fault coverage. By using the optimistic or random version, it is possible to obtain a range of possible fault coverages that is analogous to the range of sizes of transition faults. For certain applications, it is also possible to include more than one version of every fault in a single set of target faults and to compute a single fault coverage. Experimental results of fault simulation and test generation are presented to demonstrate the behavior of the model and to compare it with other fault models

Irith Pomeranz [20] et.al defined a transition fault model for use with at speed test sequences. The model is referred to as the unspecified transition fault model since it introduces unspecified values into the faulty circuit when fault effects may occur. Fault detection potentially occurs when an unspecified value reaches a primary output. Pessimistic and optimistic unspecified transition faults are defined that can be used to obtain ranges of fault coverages and numbers of detections for unspecified transition faults. Such ranges fit with the ranges of possible sizes of the standard transition faults. Random unspecified transition faults are also defined that combine the pessimistic and optimistic fault activation conditions randomly. Experimental results demonstrated that the model behaves as expected in terms of fault coverage and numbers of detections of target faults. Thus, the model provides a target for the generation of at-speed test sequences, which is more effective than *n*-detection test generation for stuck-at faults. A variation of an *n*-detection test-generation procedure for stuck-at faults was used for the generation of test sequences under this model. It is concluded that while the pessimistic unspecified transition faults are sufficient for obtaining a fault coverage metric for a test sequence applied at-speed, a consideration of optimistic or random unspecified transition faults adds more information about the potential detection of delay defects

Chapter 1

Design for Testability

1.1 Introduction

In today's complex systems, testability is an increasing concern in almost every application and in every area of application development. Manufacturers that thoroughly address the issue of testability at the device, board, and system levels deliver more consistently reliable and cost effective products to the marketplace. This means building in test capabilities in every phase of development and deployment, including design verification, hardware and software integration, manufacturing, and in the field.

The increasing use of hardware description languages (HDL's) in VLSI design and the emergence of high-level test generation programs has led to an interesting problem [1]. There is a need for design for testability (DFT) techniques that can be applied early in the design phase to improve the effectiveness of Automatic Test Pattern Generation Programs (ATPG) on hard-to-test circuits. By an early identification of hard-to-test areas of a circuit, testability can be inserted prior to logic synthesis.

The sooner process and electrical defects are caught, the lower the total cost of ownership will be. Defect finding and analysis at early manufacturing stages are critical to lower cycle time and cost for high-volume production. Therefore, a key challenge facing electronics manufacturers of high complexity system on chips is the issue of test strategy. An effective test strategy can be critical in determining a company's ability to succeed in an environment of extreme cost and time to market pressures, and requirement to faster volume ramp with higher yield and quality. Design for testability is a key to lower life cycle cost of the product from design, manufacturing, and field support [1]. Poor design for testability makes product testing difficult and with low test coverage. More and more failures escape to the field and would create a larger bonepile of circuits that need to be debugged and repaired. In order to improve quality, a special attention should be given to design for testability at early design stages while defining good test process strategy for the high volume production. Tests are applied at several steps in the hardware manufacturing flow and products may be used for hardware maintenance in the customer's environment [2]. The tests generally are driven by test programs that execute in Automatic Test Equipment (ATE) or, in the

case of system maintenance, inside the assembled system itself. In addition to finding and indicating the presence of defects, tests may be able to log diagnostic information about the nature of the encountered test fails. This diagnostic information can be used to locate the source of the failure.

1.2 Functional Testing Versus Structural Testing

IC test is comprised of two primary approaches: functional testing and structural testing.

Functional testing

Functional testing verifies that the circuit performs as it is designed to perform. For example, if we assume that the design is an adder circuit. Functional testing verifies that this circuit performs the addition function and computes the correct results over the range of values tested [2]. However, exhaustive testing of all possible input combinations grows exponentially as the number of inputs increases. To maintain a reasonable test time, one needs to focus functional test patterns on the general function and corner cases.

Structural testing

Structural testing makes no direct attempt to determine if the overall functionality of the circuit is correct. Instead, it tries to make sure that the circuit has been assembled correctly from some low level building blocks as specified in a structural netlist. For example, it checks if all specified logic gates are present, operating correctly, and connected correctly [3]. The stipulation is that if the netlist is correct, and structural testing has confirmed the correct assembly of the circuit elements, then the circuit should be functioning correctly.

Note that this is very different from Functional testing, which attempts to validate that the circuit under test functions according to its functional specification. This is closely related to functional verification problem of determining if the circuit specified by the netlist meets the functional specifications, assuming it is built correctly.

One benefit of the Structural testing is that test generation can focus on testing a limited number of relatively simple circuit elements rather than having to deal with an exponentially exploding multiplicity of functional states and state transitions. While the task of testing a single logic gate at a time sounds simple, there is an obstacle to overcome. For today's highly complex designs, most gates are deeply embedded whereas the test equipment is only connected to the primary input

outputs (I/Os) and some physical test points. The embedded gates, hence, must be manipulated through intervening layers of logic [1]. If the intervening logic contains state elements, then the issue of an exponentially exploding state space and state transition sequencing creates an unsolvable problem for test generation. To address such issues, designers make use of DFT techniques

1.3 What is Design for Test?

Design for Test is a name for design techniques that add certain testability features to a microelectronic hardware product design. The premise of the added features is that they make it easier to develop and apply manufacturing tests for the designed hardware. The purpose of manufacturing tests is to validate that the product hardware contains no defects that could adversely affect the product's correct functioning.

DFT techniques [1] have been used at least since the early days of electric/electronic data processing equipment. Early examples from the 1940-50s are the switches and instruments that allowed an engineer to scan i.e. selectively probe the voltage/current at some internal nodes in an analog computer. DFT often is associated with design modifications that provide improved access to internal circuit elements such that the local internal state can be controlled (controllability) and/or observed (observability) more easily. The design modifications can be strictly physical in nature e.g. adding a physical probe point to a net and/or add active circuit elements to facilitate controllability/observability e.g. inserting a multiplexer into a net. While controllability and observability improvements for internal circuit elements definitely are important for test, they are not the only type of DFT. Other guidelines, for example, deal with the electromechanical characteristics of the interface between the product under test and the test equipment. Examples are guidelines for the size, shape, and spacing of probe points, or the suggestion to add a high-impedance state to drivers attached to probed nets such that the risk of damage from back-driving is mitigated.

Over the years the industry has developed and used a large variety of more or less detailed and more or less formal guidelines for desired and/or mandatory DFT circuit modifications [4]. The common understanding of DFT in the context of Electronic Design Automation (EDA) for modern microelectronics is shaped to a large extent by the capabilities of commercial DFT software tools as well as by the expertise and experience of a professional community of DFT engineers researching, developing, and using such testability is a design attribute that measures

how easy it is to create a program to comprehensively test a manufactured design's quality. Traditionally, design and test processes were kept separate, with test considered only at the end of the design cycle. But in contemporary design flows, test merges with design much earlier in the process, creating what is called a design-for-test process flow. In a testable design setting specific values on the primary inputs results in values on the primary outputs that indicate whether or not the internal circuitry works properly. To ensure maximum design testability, designers must employ special DFT techniques at specific stages in the development process. Therefore, Testability is the ability to create a Test Program that maximizes the quality of a manufactured design that will minimize Defect Level (DL)

Testability is the combination of two properties of a design: Controllability and Observability

1.3.1 Controllability

Controllability is the ability to force a known value on a net from the primary input. An uncontrollable logic example is shown in Figure 1.1. In this circuit, internal node F is difficult to control to logic 0

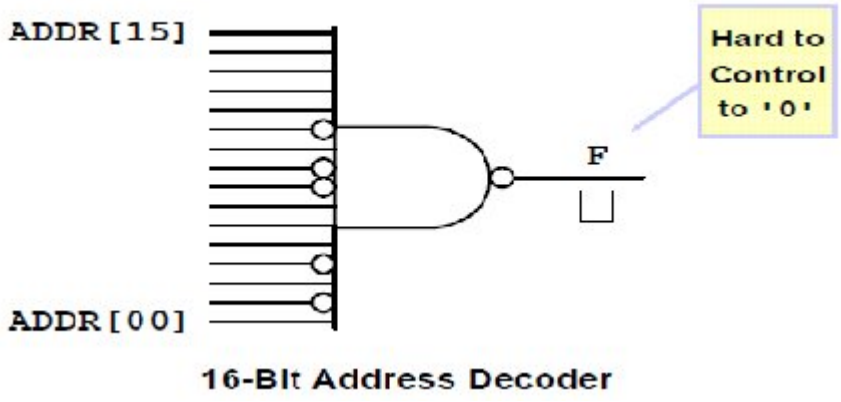

Figure 1.1 An uncontrollable logic example

1.3.2 Observability

Observability is the ability to observe the value of a net from the primary outputs. It can also be defined as the ease with which ATE equipment can propagate a fault effect to primary output by applying values to primary inputs The more observable a logic network, the more testable it is [2]. It is easier for the test pattern generation tools to propagate the effects of faults and detect them. An unobservable logic example is shown in Fig 1.2

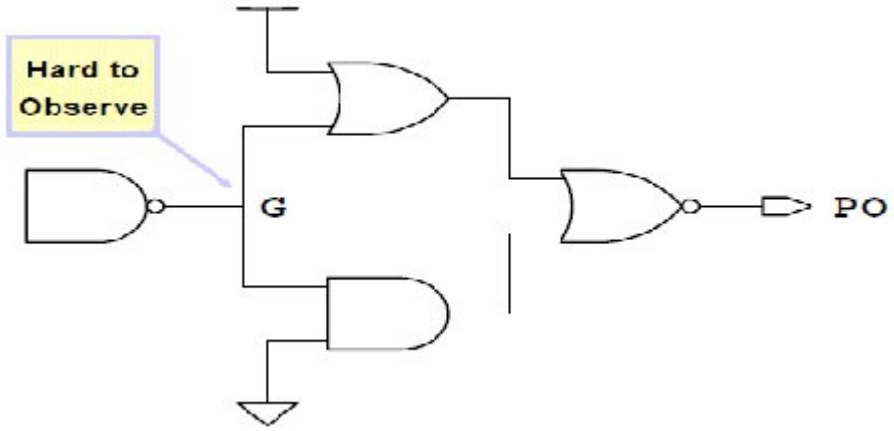

Figure 1.2 An unobservable logic example

Therefore extra gates are inserted in the design to make internal nodes more observable and controllable

1.4 Advantages of DFT

- DFT simplifies the task of the Automatic Test Pattern Generation tool.
- DFT reduces cost, despite additional gates, with respect to functional test by decreasing pattern count, thereby allowing lower cost testers as well as by decreasing test time due to reduced pattern count
- DFT reduces time to market of a product
- DFT makes debugging of complex system on chips easier and faster
- DFT reduces field returns, increasing customer's confidence

Chapter 2

Fault Modeling and Simulation

2.1 What are Faults?

Fabrication of an ASIC is a complicated process requiring hundreds of processing steps. Problems may introduce a defect that in turn may introduce a fault. Any problem during fabrication may prevent a transistor from working and may break or join interconnections. Common examples include defects occuring in metallization due to either underetching of the metal which results in a bridge or short circuit between adjacent lines or overetching the metal and causing breaks or open circuits.

Defects may also arise after chip fabrication is complete while testing the wafer, cutting the die from the wafer, or mounting the die in a package. Wafer probing, wafer saw, die attach, wire bonding, and the intermediate handling steps each have their own defect and failure mechanisms [5]. Many different materials are involved in the packaging process that have different mechanical, electrical, and thermal properties, and these differences can cause defects due to corrosion, stress, adhesion failure, cracking, and peeling.

Yield loss also occurs from human error using the wrong mask, incorrectly setting the implant dose as well as from physical sources like contaminated chemicals, dirty etch sinks, or a troublesome process step. It is possible to repeat or rework some of the reversible steps if there are problems. However, reliance on rework indicates a poorly controlled process.

Figure 2.1 shows various types of defects that occur during fabrication. Defect density for a modern CMOS process is of the order of 1 cm^{-2} or less across a whole wafer. The logic cell shown in Figure 2.1 is approximately 6432 l^2 or 250 mm^2 for a l = 0.25 mm process. We would thus have to examine approximately 1 cm^{-2} /250 mm^2 or 400,000 such logic cells to find a single defect.

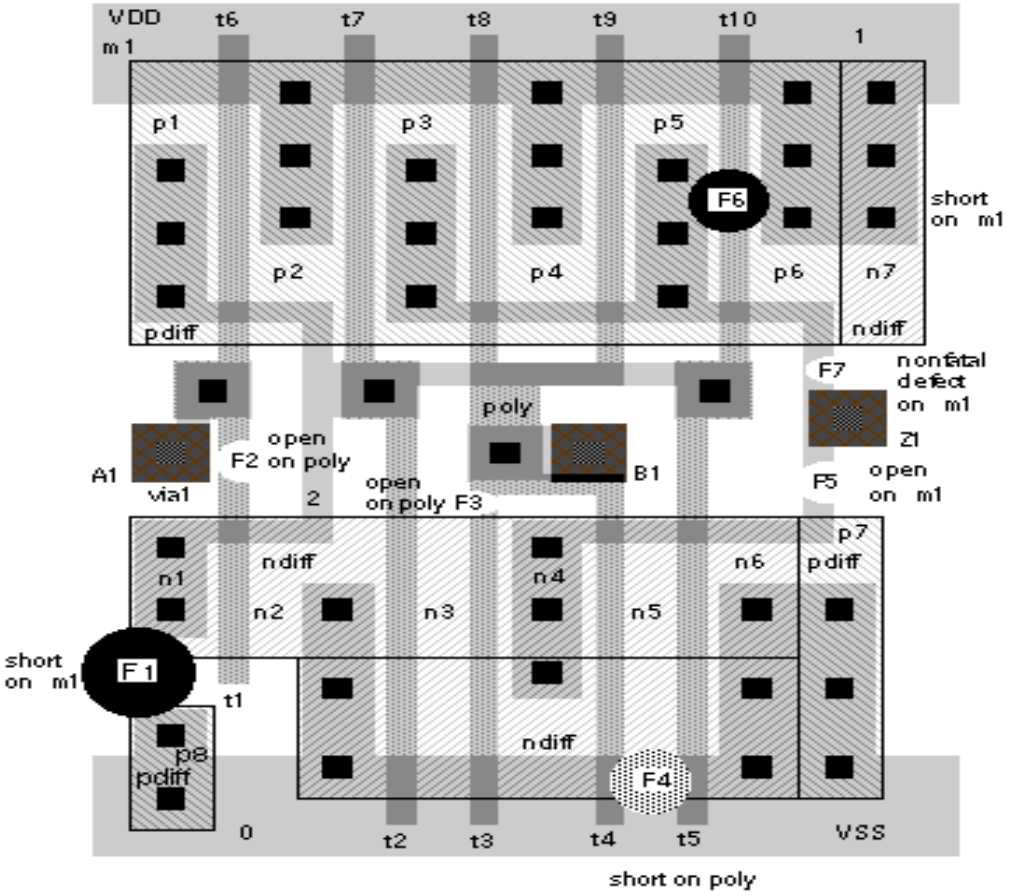

Figure 2.1 Defects and Physical faults

- F1 is a short between m1 lines and connects node n1 to VSS.
- F2 is an open on the poly layer and disconnects the gate of transistor t1 from the rest of the circuit.
- F3 is an open on the poly layer and disconnects the gate of transistor t3 from the rest of the circuit.
- F4 is a short on the poly layer and connects the gate of transistor t4 to the gate of transistor t5.
- F5 is an open on m1 and disconnects node n4 from the output Z1.
- F6 is a short on m1 and connects nodes p5 and p6.

2.2 What Are Fault Models?

Physical failures and fabrication defects cannot be easily modelled mathematically. As a result, these failures and defects are modelled as logical faults. Structural faults relate to the structural model of a system and these affect interconnections among components of a design [6]. Functional faults relate to a functional model, for example an RTL/HDL model and these affect the nature of operation of the components in a design. Testing for functional faults validates the correct operation of a system, while testing of structural faults targets manufacturing defects. The physical defects are modelled into their equivalent Fault Models in order for an ATPG to generate effective pattern(s) that can identify a fault within a circuit netlist

A Defect is defined as on-chip flaw introduced during fabrication or packaging of an individual ASIC that leads to a logical malfunction A manufacturing defect has a logical effect on the circuit behaviour. An open connection can appear to float either high or low, depending on the technology. A signal shorted to power appears to be permanently high. A signal shorted to ground appears to be permanently low

When a fault changes the circuit behavior, the change is called the fault effect. Fault effects travel through the circuit to other logic cells causing other fault effects. This phenomenon is fault propagation. If the fault level is at the structural level, the phenomenon is structural fault propagation. If there are one or more large functional blocks in a design, one wants to apply faults to the functional blocks only at the inputs and outputs of the blocks. It is not desirable to place (or do not place) faults inside the blocks, but faults should propagate through the blocks. This is behavioral fault propagation

Designers adjust the fault level to the appropriate level at which they think there may be faults. Suppose we are performing a fault simulation on a board and we have already tested the chips. Then we might set the fault level to the chip level, placing faults only at the chip pins. For ASICs logic-cell level are used. It should be noted though that if behavioral level and structural level models are mixed in a mixed-level fault simulation, then one should be sure that the behavioral models propagates faults correctly. In particular, if the behavioral model responds to faults on its inputs by propagating too many unknown 'X' values to its outputs, this will decrease the fault coverage, because the model is hiding the logic beyond it. So it can be said that Fault Modelling is representing defects at abstract logic level

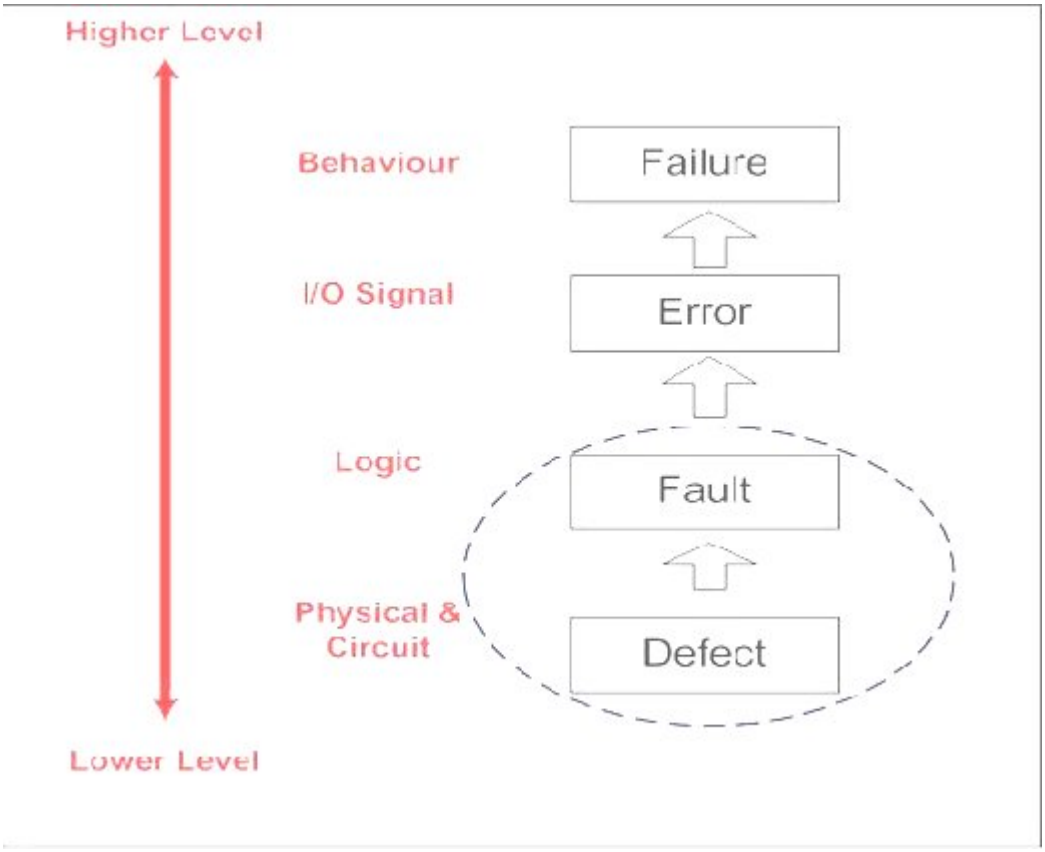

Figure 2.2 Fault Modeling

A defect leads to system/functional failure. To explain this, let us consider an AND gate with the input b is shortened to ground during manufacture as shown in Fig 2.3

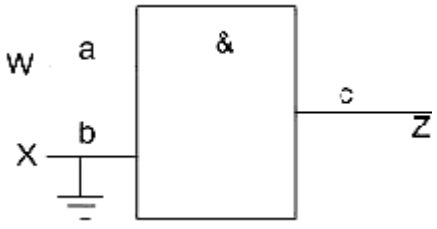

Figure 2.3 A faulty AND gate

The Defect in this case is the unwanted wires which are shorted to ground whereas the Fault in this case is b stuck at logic 0. This will result in an Error output of 0 when a=b=1. This ultimately results in system failure.

2.3 Stuck-At Fault Model

The stuck-at-0 model represents a signal that is permanently low regardless of the other signals that normally control the node. The stuck-at-1 model represents a signal that is permanently high regardless of the other signals that normally control the node. For example, Figure 2.4 below shows a two-input AND gate that has a stuck-at-0 fault on the output pin. Regardless of the logic level of the two inputs, the output is always 0.

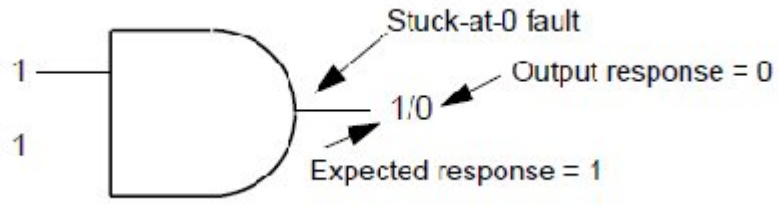

Figure 2.4 Two-input AND gate that has a stuck-at-0 fault on the output pin

2.3.1 Detecting Stuck-At Faults

The node of a stuck-at fault must be controllable and observable for the fault to be detected. A node is controllable if one can drive it to a specified logic value by setting the primary inputs to specific values [7]. A primary input is an input that can be directly controlled in the test environment. A node is observable if one can predict the response on it and propagate the fault effect to the primary outputs where one can measure the response. A primary output is an output that can be directly observed in the test environment.

To detect a stuck-at fault on a target node, one must do the following:

- Control the target node to the opposite of the stuck-at value by applying data at the primary nputs.
- Make the node's fault effect observable by controlling the value at all other nodes affecting the output response, so the targeted node is the active (controlling) node

The set of resulting values at the primary outputs, assuming a fault-free design, is called the expected response. The set of actual values measured at the primary outputs is called the output response. If the output response does not match the expected response for a given input stimulus, the input stimulus has detected the fault. To detect a stuck-at-0 fault, one needs to apply an input stimulus that forces that node to 1. For example, to detect a stuck-at-0 fault at the output the two-input AND gate shown in Figure 2.5, one needs to apply a logic 1 at both inputs. The expected response for this input stimulus is logic 1, but the output response is logic 0. This input stimulus detects the stuck-at-0 fault. This method of determining the input stimulus to detect a fault uses the single stuck-at fault model. The single stuck-at fault (SSF) model assumes that only one node is faulty and that all other nodes in the circuit are good. This type of model greatly reduces the complexity of fault modelling and is technology independent [7]. In a more complex situation, one may need to control all other nodes to ensure observability of a particular target node.

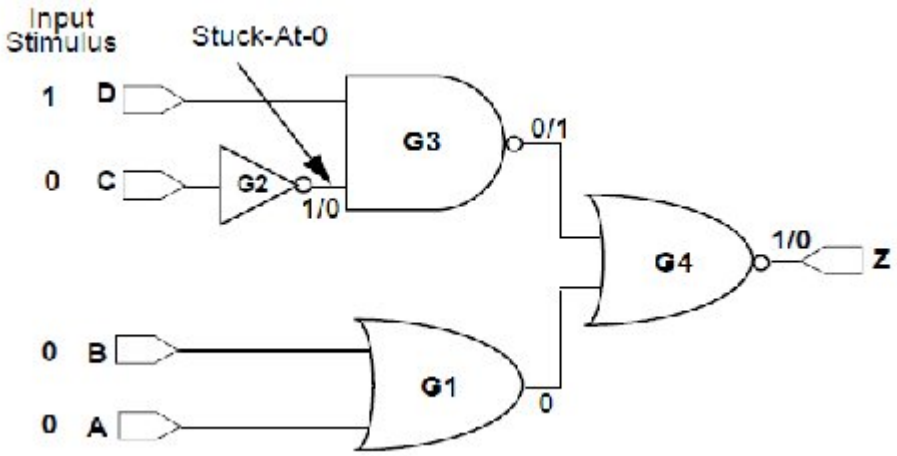

Figure 2.5 Simple circuit with detectable Stuck At Fault

There are other fault models. For example, it can be assumed that faults are located in the transistors using a stuck-on fault and stuck-open fault (or stuck-off fault). Fault models such as these are more realistic in that they more closely model the actual physical faults. However, in practice the simple SSF model has been found to work and work well. Therefore more emphasis will be on the SSF model.

Figure 2.6 and the following list show how the defects and physical faults translate to logical faults, although not all physical faults can be translated to logical faults

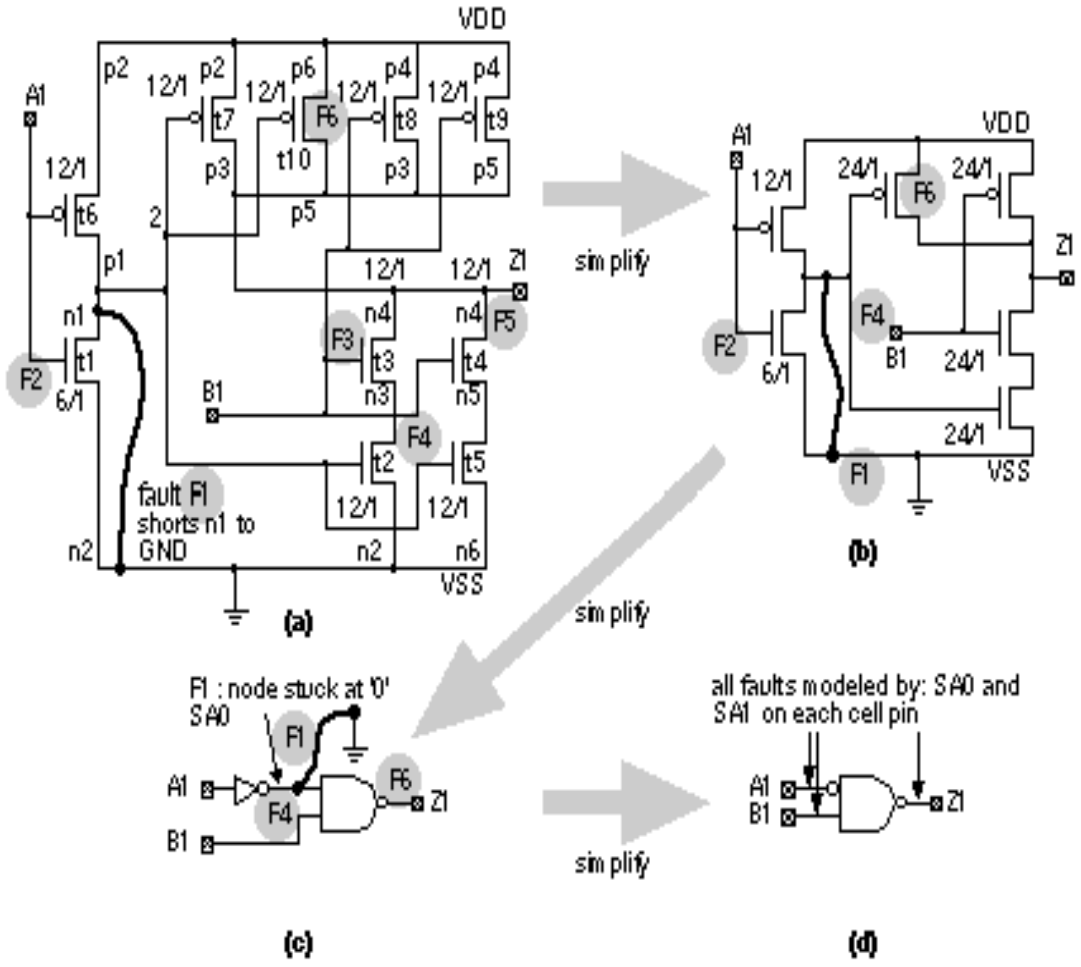

Figure 2.6 Fault models (a) Physical faults at the layout level due to problems during fabrication translate to electrical problems on the detailed circuit schematic. The location and effect of fault F1 is shown. The locations of the other fault examples of figure 2.1 (F2–F6) are shown, but not their effect. (b) Some of these faults can be translated to the simplified transistor schematic. (c) Only a few of the physical faults still remain in a gate-level fault model of the logic cell. (d) Finally at the functional level fault model of a logic cell, the connection between physical and logical faults is abandoned and all faults are modeled by stuck-at faults

2.4 IDDQ Fault Model

For CMOS circuits, an alternative testing method is available, called IDDQ testing. IDDQ testing is based on the principle that a CMOS circuit does not draw a significant amount of current when the device is in the quiescent state [9]. The presence of even a single circuit fault, such as a short from an internal node to ground or to the power supply, can be detected by the resulting excessive current drain at the power supply pin. IDDQ testing can detect faults that are not observable by stuck-at fault testing

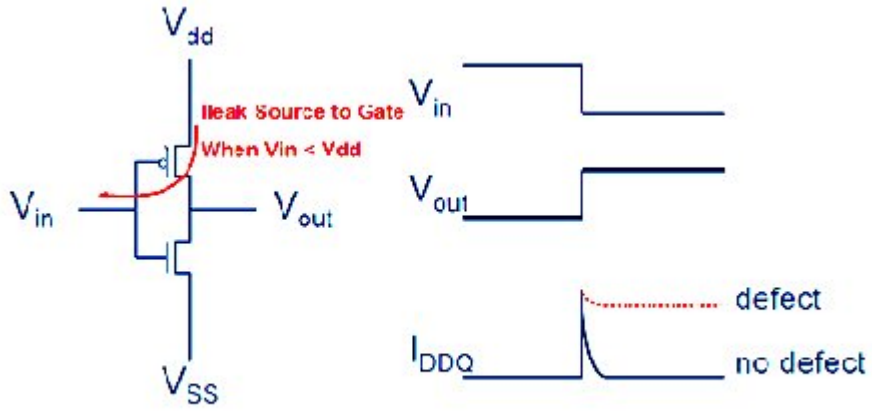

Figure 2.7 IDDQ Detectable Defects

For the IDDQ testing, the ATPG algorithm uses an IDDQ fault model rather than a stuck-at fault model. The generated test patterns only need to control internal nodes to 0 and 1 and comply with quiescence requirements. The patterns do not need to propagate the effects of faults to the device outputs. The ATPG tool attempts to maximize the toggling of internal states and minimize the number of patterns needed to control each node to both 0 and 1 for IDDQ testing

2.5 Delay Fault Model

This fault model is used to model physical defects which logically behave as transition delay. There are two types of delay related defects

2.5.1 Transition Delay Fault Model

The transition delay fault model is similar to the stuck-at fault model, except that it attempts to detect slow-to-rise and slow-to-fall nodes, rather than stuck-at-0 and stuck at-1 nodes [10]. A slow to rise fault at a defect means that a transition from 0 to 1 on the defect does not produce the correct results at the maximum operating speed of the

device. Similarly, a slow-to-fall fault means that a transition from 1 to 0 on a node does not produce the correct results at the maximum operating speed of the device

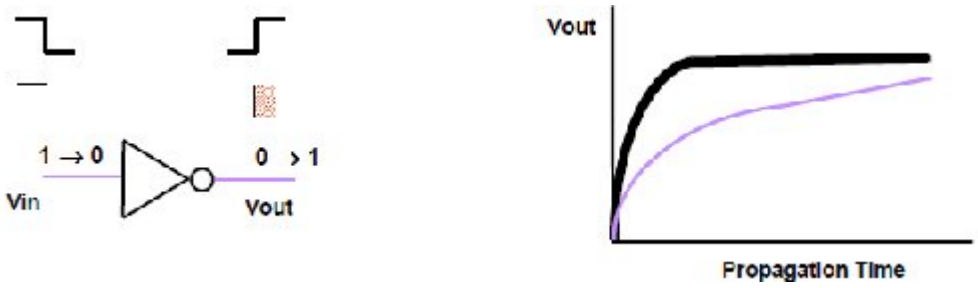

Figure 2.8 Logic example depicting transition delay

2.5.2 Path Delay Fault Model

The path delay fault model is useful for testing and characterizing critical timing paths in the design. Path delay fault tests exercise the critical paths at-speed (the full operating speed of the chip) to detect whether the path is too slow because of manufacturing defects or variations [10].

Path delay fault testing targets physical defects that might affect distributed regions of a chip. For example, incorrect field oxide thicknesses could lead to slower signal propagation times, which could cause transitions along a critical path to arrive too late. By comparison, stuck-at, IDDQ, and transition delay faults are generally targeted at single-point defects. Logic example depicting path delay is shown in figure 2.9

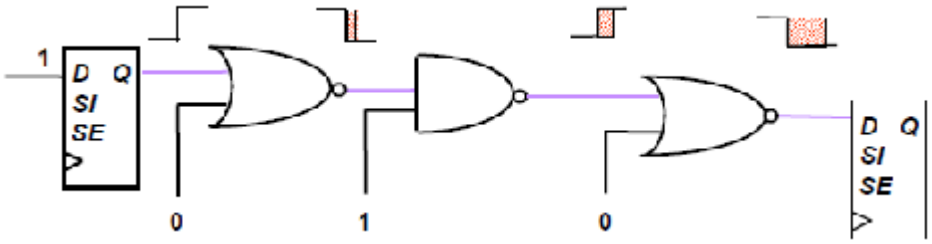

Figure 2.9 Logic example depicting path delay

2.6 Bridging Fault Model

The model is that of logical lines in the circuit shorted together. The possibility of lines internal to a logic element shorted together is typically not considered because of the more complex circuit model required and the fact that these types of defects are most often covered by other models (stuck-at). However, at the same time ATPG algorithms are more complex and testing requires setting the two-bridged nodes to opposite values and observing the effect. Bridging fault model is used to represent shorted nets [11]. Usually assumed to be a low resistance path (hard short). Simplified model assumes 0 ohm resistance (short) between two lines as shown in fig 2.10

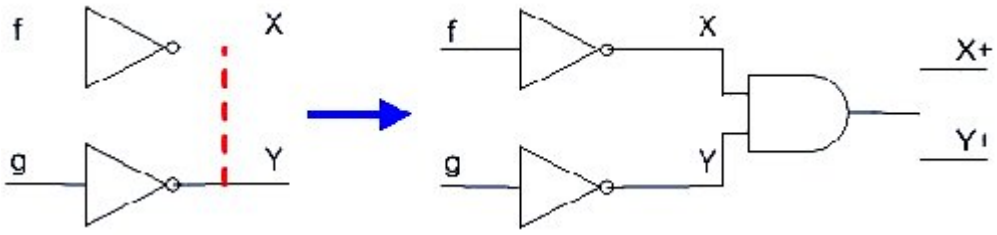

Figure 2.10 Wired and Bridge Fault

Y = 0 implies X is s-a-0. In order to test this kind of fault set Y to 0 and test for X s-a-0 i.e. Set the logic 0 at Y by proper input values. Now, test for saf-0 at X. If this effects the logic value at Y means there is a bridge fault between X and Y or Set X to 0 and test for Y s-a-0

Different occurrences of Bridge faults are as explained below

Output-to-Output

- Between metal lines in routing channels.
- Between outputs of different gates.

Input-to-Input

- Between inputs of the same gate in polysilicon

Input-to-Output

- Between an input and output of the same gate.
- In a simple CMOS gate, if the short causes an error, then input value is forced upon the output. This is also true for complex CMOS gates such as And-Or-Invert (AOI) and Or-And-Invert (OAI) gates.

Source-to-Drain

- Between source and drain of the same transistor in diffusion. It is also called transistor stuck-on fault. It is not considered strictly a logic fault.

2.7 Fault Collapsing

Figure 2.11 (a) shows a test for a stuck-at-1 output of a two-input NAND gate whereas Figure 2.11 (b) shows tests for other stuck-at faults. It is assumed that the NAND gate still works correctly in the bad circuit (also called the faulty circuit or faulty machine) even if there is an input fault. The input fault on a logic cell is presumed to arise either from a fault from a preceding logic cell or a fault on the connection to the input. Stuck-at faults attached to different points in a circuit may produce identical fault effects [7]. Using fault collapsing, equivalent faults (or indistinguishable faults) can be grouped into a fault-equivalence class. To save time only one fault is needed to be considered which is called the prime fault or representative fault, from a fault-equivalence class. For example, Figure 2.11 (a) and (b) show that a stuck-at-0 input and a stuck-at-1 output are equivalent faults for a two-input NAND gate. Only one fault Z1 (output stuck at 1) needs to be checked to catch any of the equivalent faults

Suppose that any of the tests that detect a fault B also detects fault A, but only some of the tests for fault A also detect fault B. Let us say A is a dominant fault, or that fault A dominates fault B. Clearly to reduce the number of tests using dominant fault collapsing test for fault B will be

picked up. For example, Figure 2.11 (c) shows that the output stuck at 0 dominates either input stuck at 1 for a two-input NAND. By testing for fault A1, fault Z1 is automatically detected.

Confusion over dominance arises because of the difference between focusing on faults (Figure 2.11 d) or test vectors (Figure 2.11 e). Figure 2.11 (f) shows the six stuck-at faults for a two-input NAND gate. SA1 or SA0 can be placed on each of the two input pins (four faults in total) and SA1 or SA0 on the output pins. Using fault equivalence (Figure 2.11 g) six faults can be collapsed to four: SA1 on each input, and SA1 or SA0 on the output. Using fault dominance (Figure 2.11 h) six faults can be collapsed to three. There is no way to tell the difference between equivalent faults, but if dominant fault collapsing is used, information about the fault location may get lost.

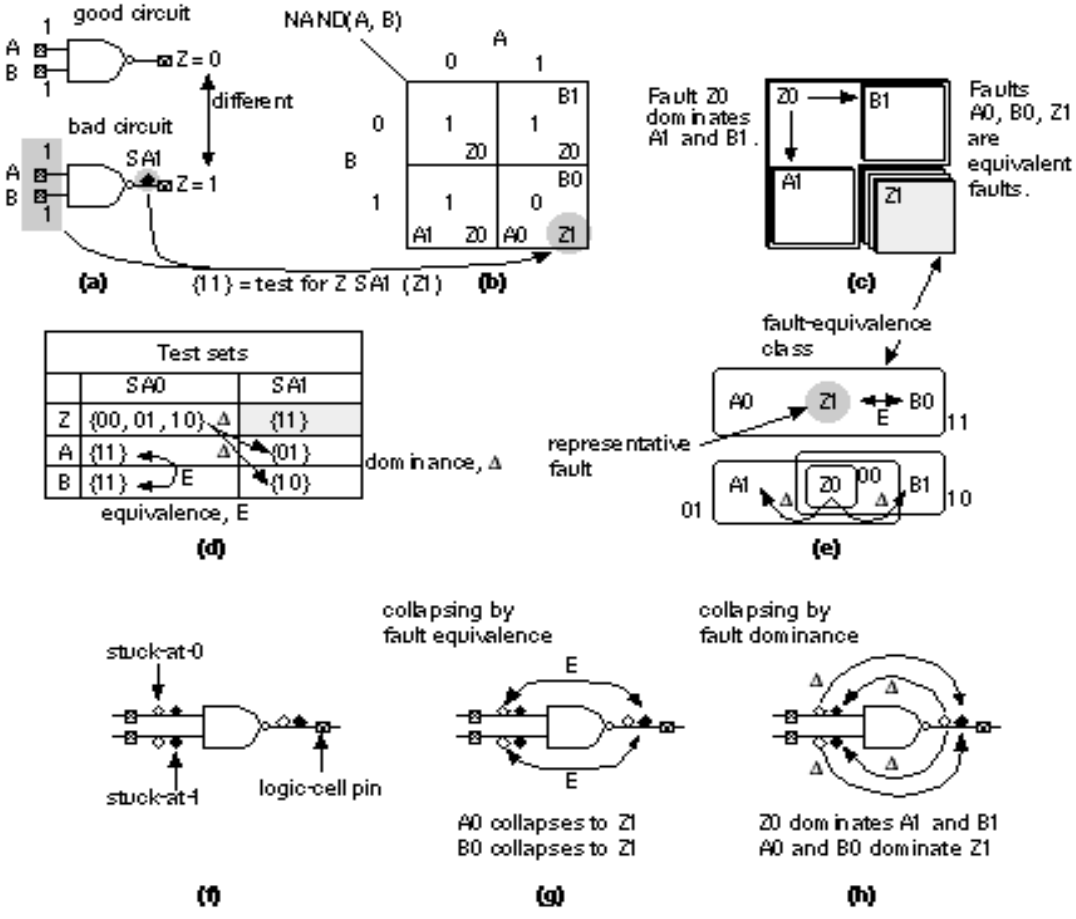

Figure 2.11 Fault dominance and fault equivalence

2.8 Test Patterns

These are the input boolean values for a specific fault. These boolean values are placed on primary input pins during the manufacturing test process to determine if the chip is functioning properly [1]. These are also sometimes called Test Vectors. Test Set is a set of test patterns

2.9 Fault Statistics

A common definition of the testability of a design is the extent to which the design can be tested for the presence of manufacturing defects, as represented by the single stuck-at fault model. The three possible quantitative measures to determine the test pattern quality are defined as follows

2.9.1 Test Coverage

Test coverage gives the most meaningful measure of test pattern quality and is the default coverage reported in the fault summary report. Test coverage is defined as the percentage of detected faults out of detectable faults, as follows:

TEST COVERAGE = 100 x Detected faults (DT) / Total no. of testable faults

2.9.2 Fault Coverage

Fault coverage is defined as percentage of all the faults detected by the pattern set

FAULT COVERAGE = 100 x Detected faults (DT) / Total no. of faults

2.10 Fault Simulation

Fault simulation is used after logic simulation has been completed to see what happens in a design when faults are introduced deliberately. In a production test one can only have access to the package pins—the primary inputs (PIs) and primary outputs (POs) [11]. To test an ASIC we must series of sets of input patterns must be devised that will detect any faults. A stimulus is the application of one such set of inputs (a test vector) to the PIs of an ASIC. A typical ASIC may have several hundred PIs and therefore each test vector is several hundred bits long. A test program consists of a set of test vectors. Typical ASIC test programs require tens of thousands and sometimes hundreds of thousands of test vectors.

The test-cycle time is the period of time the tester requires to apply the stimulus, sense the POs, and check that the actual output is equal to the expected output. Suppose the test cycle time is 100

ns (corresponding to a test frequency of 10 MHz), in which case one might sense (or strobe) the POs at 90 ns after the beginning of each test cycle. Using fault simulation one mimics the behavior of the production test. The fault simulator deliberately introduces all possible faults into ASIC, one at a time, to see if the test program will find them.

2.10.1 Serial Fault Simulation

Serial fault simulation is the simplest fault-simulation algorithm. Two copies of the circuit are simulated. The first copy is a good circuit. A fault is picked and inserted into the faulty circuit [13]. In test terminology, the circuits are called machines, so the two copies are a good machine and a faulty machine. This process is repeated, simulating one faulty circuit at a time. Serial simulation is slow and is impractical for large ASICs.

2.10.2 Parallel Fault Simulation

Parallel fault simulation takes advantage of multiple bits of the words in computer memory. In the simplest case only one bit is needed to represent either a '1' or '0' for each node in the circuit. In a computer that uses a 32-bit word memory a set of 32 copies of the circuit can be simulated at the same timeat the same time. One copy is the good circuit, and different faults into the other copies are inserted. When a logic operation is needed to be performed, to model an AND gate for example, the operation across all bits in the word simultaneously can be performed In this case, using one bit per node on a 32-bit machine, parallel fault simulation can be expected to be about 32 times faster than serial simulation. The number of bits per node that one needs in order to simulate each circuit depends on the number of states in the logic system we are using. Thus, if a four-state system with '1' , '0' , 'X' (unknown), and 'Z' (high-impedance) states is used, two bits per node are required.

Parallel fault simulation is not quite as fast as our simple prediction because one has to simulate all the circuits in parallel until the last fault in the current set is detected. If serial simulation is used, one can stop as soon as a fault is detected and then start another fault simulation. Parallel fault simulation is faster than serial fault simulation but not as fast as concurrent fault simulation. It is also difficult to include behavioral models using parallel fault simulation.

2.10.3 Concurrent Fault Simulation

Concurrent fault simulation is the most widely used fault-simulation algorithm and takes advantage of the fact that a fault does not affect the whole circuit. Thus it is not needed to simulate

the whole circuit for each new fault. In concurrent simulation, first the good circuit is completely simulated. Then a fault is injected and a copy of only that part of the circuit that behaves differently is resimulated. For example, if the fault is in an inverter that is at a primary output, only the inverter needs to be simulated, everything preceding the inverter can be removed
Keeping track of exactly which parts of the circuit need to be diverged for each new fault is complicated, but the savings in memory and processing that result allow hundreds of faults to be simulated concurrently. Concurrent simulation is split into several chunks, one can usually control how many faults (usually around 100) are simulated in each chunk or pass. Each pass thus consists of a series of test cycles. Every circuit has a unique fault-activity signature that governs the divergence that occurs with different test vectors. Thus every circuit has a different optimum setting for faults per pass. Too few faults per pass will not use resources efficiently. Too many faults per pass will overflow the memory.

2.10.4 Nondeterministic Fault Simulation

Serial, parallel, and concurrent fault-simulation algorithms are forms of deterministic fault simulation. In each of these algorithms set of test vectors are used to simulate a circuit and discover which faults can be detected. If the fault coverage is inadequate, the test vectors are modified and the fault simulation is repeated. This is a very time-consuming process

As an alternative, trying to simulate every possible fault can be given up and instead, using probabilistic fault simulation, a subset or sample of the faults are simulated and fault coverage is explorated from the sample. In statistical fault simulation a fault-free simulation is performed and the results are used to predict fault coverage. This is done by computing measures of observability and controllability at every node. A node is not stuck if one can make the node toggle, that is, change from a '0' to '1' or vice versa. A toggle test checks which nodes toggle as a result of applying test vectors and gives a statistical estimate of vector quality, a measure of faults detected per test vector. There is a strong correlation between high-quality test vectors, the vectors that will detect most faults, and the test vectors that have the highest toggle coverage. Testing for nodes toggling simply requires a single logic simulation that is much faster than complete fault simulation [16].

A considerable improvement in fault simulation speed can be obtained by putting the high-quality test vectors at the beginning of the simulation. The sooner the faults can be detected and eliminated from having to be considered in each simulation, the faster the simulation will

progress. Same approach is taken when running a production test and initially the test vectors are ordered by their contribution to fault coverage. This assumes that all faults are equally likely. Test engineers can then modify the test program if they discover vectors late in the test program that are efficient in detecting faulty chips.

Chapter 3

Automatic Test Pattern Generation

3.1 Introduction

The process of generating the test patterns on a design via a CAD (Computer Aided Design) tool is called ATPG or Automatic Test Pattern Generation. Test patterns, sometimes called test vectors, are sets of 1s and 0s placed on primary input pins during the manufacturing test process to determine if the chip is functioning properly. When the test pattern is applied, the Automatic Test Equipment (ATE) determines if the circuit is free from manufacturing defects by comparing the fault-free output which is also contained in the test pattern with the actual output measured by the ATE

The goal of ATPG is to create a set of patterns that achieves a given test coverage, where test coverage is the total percentage of testable faults the pattern set actually detects. The ATPG run itself consists of two main steps i.e generating patterns and performing fault simulation to determine which faults the patterns detect [17]. The two most typical methods for pattern generation are random and deterministic. Additionally, the ATPG tools can fault simulate patterns from an external set and place those patterns detecting faults in a test set.

3.1.1 Random Test Pattern Generation

An ATPG tool uses random pattern test generation when it produces a number ofrandom patterns and identifies only those patterns necessary to detect faults. It then stores only those patterns in the test pattern set. The type of fault simulation used in random pattern test generation cannot replace deterministic test generation because it can never identify redundant faults [16]. Nor can it create test patterns for faults that have a very low probability of detection. However, it can be useful on testable faults aborted by deterministic test generation. Using a small number of random patterns as the initial ATPG step can improve ATPG performance.

3.1.2 Deterministic Test Pattern Generation

An ATPG tool uses deterministic test pattern generation when it creates a test pattern intended to detect a given fault. The procedure is to pick a fault from the fault list, create a pattern to detect the fault, fault simulate the pattern, and check to make sure the pattern detects the fault

More specifically, the tool assigns a set of values to control points that force the fault site to the state opposite the fault-free state, so there is a detectable difference between the fault value and the fault-free value. The tool must then find a way to propagate this difference to a point where it can observe the fault effect. To satisfy the conditions necessary to create a test pattern, the test generation process makes intelligent decisions on how best to place a desired value on a gate. If a conflict prevents the placing of those values on the gate, the tool refines those decisions as it attempts to find a successful test pattern

If the tool exhausts all possible choices without finding a successful test pattern, it must perform further analysis before classifying the fault. Faults requiring this analysis include redundant, ATPG-untestable, and possible-detected-untestable categories. Identifying these fault types is an important by-product of deterministic test generation and is critical to achieving high test coverage. For example, if a fault is proven redundant, the tool may safely mark it as untestable. Otherwise, it is classified as a potentially detectable fault and counts as an untested fault when calculating test coverage.

3.1.3 External Test Pattern Generation

An ATPG tool uses external pattern test generation when the preliminary source of ATPG is a pre-existing set of external patterns that already exists. The tool analyzes this external pattern set to determine which patterns detect faults from the active fault list. It then places these effective patterns into an internal test pattern set [17]. The “generated patterns”, in this case, include the patterns (selected from the external set) that can efficiently obtain the highest test coverage for the design.

3.1.4 Translation for the Manufacturing Test Environment

To test for manufacturing defects in your chips, one needs to translate the generated test patterns into a format acceptable to the automated test equipment (ATE). On the ATE, the logic 0s and 1s in the input stimuli are translated into low and high voltages to be applied to the primary inputs of the device under test. The logic 0s and 1s in the output response are compared with the voltages measured at the primary outputs. For combinational ATPG, one test vector corresponds to one ATE cycle. One might use more than one set of test vectors for manufacturing testing. The term test program means the collection of all test vector sets used to test a design.

3.2 Scan Design

Scan design or internal scan is the internal modification of design's circuitry to increase its testability. Scan design uses either full or partial scan techniques, depending on design criteria. Internal scan design is the most popular DFT technique and has the greatest potential for high test coverage. The principle of this technique is to modify the existing sequential elements in the design to support serial shift capability, in addition to their normal functions and to connect these elements into serial chains to make, in effect, long shift registers [19]

The goal of scan design is to make a difficult-to-test sequential circuit behave (during the testing process) like an easier-to-test combinational circuit. Achieving this goal involves replacing sequential elements with scan able sequential elements (scan cells) and then stitching the scan cells together into scan registers, or scan chains. One can then use these serially-connected scan cells to shift data in and out when the design is in scan mode.

Each scan chain element can operate like a primary input or primary output during ATPG testing, greatly enhancing the controllability and observability of the internal nodes of the device. This technique simplifies the pattern generation problem by effectively dividing complex sequential designs into fully isolated combinational blocks (full-scan design) or semi-isolated combinational blocks (partial-scan design).

Figure 3.1 shows an example of the multiplexed flip-flop scan style, where a D flip-flop has been modified to support internal scan by the addition of a multiplexer. Inputs to the multiplexer are the data input of the flip-flop (d) and the scan-input signal (scan_in). The active input of the multiplexer is controlled by the scan-enable signal (scan_enable). Input pins are added to the cell

for the scan_in and scan_enable signals. One of the data outputs of the flip-flop (q or qn) is used as the scan-output signal (scan_out). The scan_out signal is connected to the scan_in signal of another scan cell to form a serial scan (shift) capability.

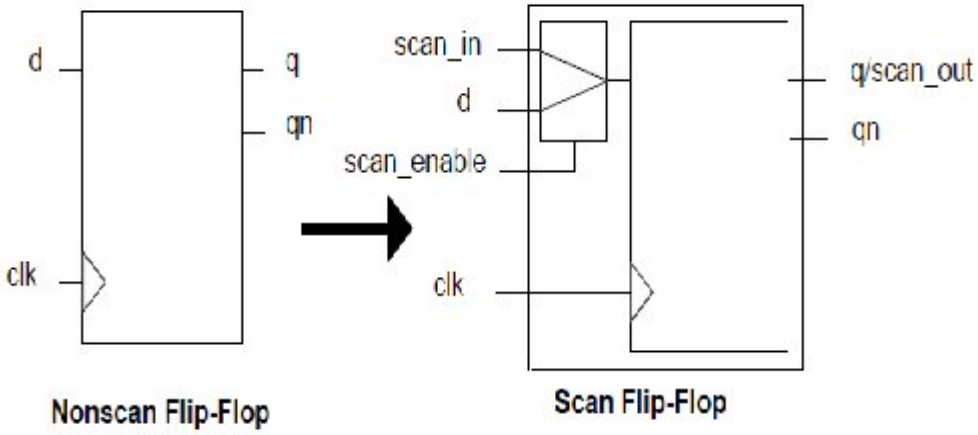

Figure 3.1 D Flip flop with scan capability

The modified sequential cells are chained together to form one or more large shift registers. These shift registers are called scan chains or scan paths. The sequential cells connected in a scan chain are scan controllable and scan observable. A sequential cell is scan controllable when you can set it to a known state by serially shiftingin specific logic values. ATPG tools treat scan controllable cells as pseudo-primary inputs to the design. A sequential cell is scan observable when you can observe its state by serially shifting out data. ATPG tools treat scan-observable cells as pseudo-primary outputs of the design.

Most semiconductor vendor libraries include pairs of equivalent nonscan and scan cells that support a given scan style. One special test cell is a scan flip-flop that combines a D flip-flop and a multiplexer. One can also implement the scan function of this special test cell with discrete cells, such as the separate flip-flop and multiplexer shown in Figure 3.1

Adding scan circuitry to a design usually has the following effects:

- Design size and power increases slightly because scan cells are usually larger than the nonscan cells they replace, and the nets used for the scan signals occupy additional area

- Design performance (speed) decreases marginally because of changes in the electrical characteristics of the scan cells that replace the nonscan cells

- Global test signals that drive many sequential elements might require buffering to prevent electrical design rule violations

The effects of adding scan circuitry vary depending on the scan style and the semiconductor vendor library you use. For some scan styles, such as level-sensitive scan design (LSSD), introducing scan circuitry produces a negligible local change in performance [19]. The Synopsys scan DFT synthesis capabilities fully optimize for the user's design rules and constraints (timing, area, and power) in the context of scan [23]. These scan synthesis capabilities are available in DFT Compiler, the Synopsys test-enabled synthesis configuration

For scan designs, an ATPG tool generates input stimuli for the primary inputs and pseudo-primary inputs and expected responses for the primary outputs and pseudo-primary outputs. The set of input stimuli and output responses is called a test pattern or scan pattern. This set includes the data at the primary inputs, primary outputs, pseudo-primary inputs, and pseudo-primary outputs. A test pattern represents many test vectors because the pseudo-primary-input data must be serialized to be applied at the input of the scan chain, and the pseudo-primary-output datamust be serialized to be measured at the output of the scan chain

3.3 Applying Test Patterns

Test patterns are applied to a scan-based design through the scan chains. The process is the same for a full-scan or partial-scan design. Scan cells operate in one of two modes: parallel mode or shift mode. In the multiplexed flip-flop scan style shown in Figure 3.1, the mode is controlled by the scan_enable pin. When the scan_enable signal is inactive, the scan cells operate in parallel mode, the input to each scan element comes from the combinational logic block. When the scan_enable signal is asserted, the scan cells operate in shift mode; the input comes from the output of the previous cell in the scan chain (or from the scan input port, if it is the first chain element). Other scan styles work similarly

The target tester applies a scan pattern in the following sequence:

1. Select shift mode by setting the scan-enable port. This test signal is connected to all scan cells
2. Shift in the input stimuli for the scan cells (pseudo-primary inputs) at the scan input ports

3. Select parallel mode by disabling the scan-enable port.
4. Apply the input stimuli to the primary inputs.
5. Check the output response at the primary outputs after the circui has settled and compare it with the expected fault-free response. This process is called parallel measure.
6. Pulse one or more clocks to capture the steady-state output response of the nonscan logic blocks into the scan cells. This process is called parallel capture
7. Select shift mode by asserting the scan-enable port.
8. Shift out the output response of the scan cells (pseudo-primary outputs) at the scan output ports and compare the scan cell contents with the expected fault-free response

3.4 Scan Design Requirements

Best test coverage results are achieved when all nodes in the design are controllable and observable. Adding scan logic to the design enhances its controllability and observability [19]. The rules governing the controllability and observability of scan cells are called test design rules

3.4.1 Controllability of Sequential Cells

For sequential cells, design rules require that all state elements can be controlled, by scan or other means, to desired state values from the boundary of the design. These requirements are primarily involved with the shift operations in scan test. In an ideal full-scan design, the scan chain contains all state elements, the circuit is fully controllable, and any circuit state can be achieved. Using a partial-scan methodology, not all state elements need to be in the scan chain. As long as the nonscan state elements can be brought to any required state predictably through sequential operation, the circuit remains sufficiently controllable

3.4.2 Observability of Sequential Cells

For sequential cells, test design rules require predictable capture of the next state of the circuit and visibility at the boundary of the design. In the context of scan design, one can ensure that sequential cells are observable if one successfully clock the scan cells in the circuit, and then shift their state to the scan outputs

The following operations define circuit observability:

1. The primary outputs of the circuit are observed after scan-in. Normally, this does not involve DFT and does not present problems

2. The next state of the circuit is captured reliably. If the functional operation is impaired, unpredictable, or unknown, the next state is unknown. This unknown state makes at least part of the circuit unobservable

3. Next state is extracted through a scan-out operation. This process is similar to scan-in. The additional requirement is that the shift registers pass data reliably to the output ports

3.5 Full-Scan Design

With a full-scan design technique, all sequential cells in the design are modified to perform a serial shift function [20]. Sequential elements that are not scanned are treated as black box cells (cells with unknown function).

Full scan divides a sequential design into combinational blocks as shown in Figure 3.2

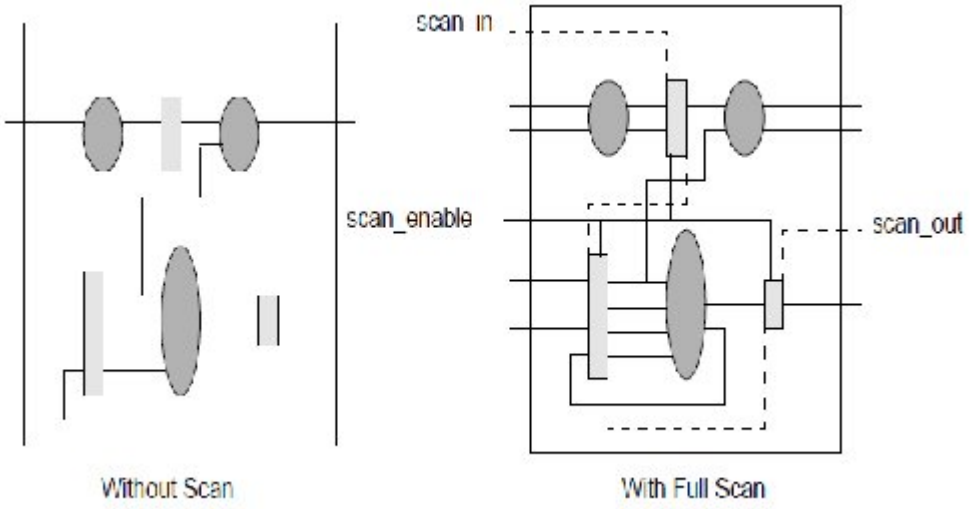

Figure 3.2 Scan Path through a full scan design. Ovals represent combinational logic and rectangles represent sequential logic

The full-scan diagram shows the scan path through the design. Through pseudo-primary inputs, the scan path enables direct control of inputs to all combinational blocks. The scan path enables direct observability of outputs from all combinational blocks through pseudo-primary outputs

3.6 Partial Scan ATPG Design

With a partial-scan design technique, the scan chains contain some, but not all, of the sequential cells in the design. A partial-scan technique offers a tradeoff between the maximum achievable test coverage and the effect on design size and performance The default ATPG mode of TetraMAX, called Basic-Scan ATPG, performs combinational ATPG. To get good test coverage in partial-scan designs, we need to use Fast-Sequential or Full-Sequential ATPG [21]. Partial scan divides a complex sequential design into simpler sequential blocks as shown in Figure 3.3.

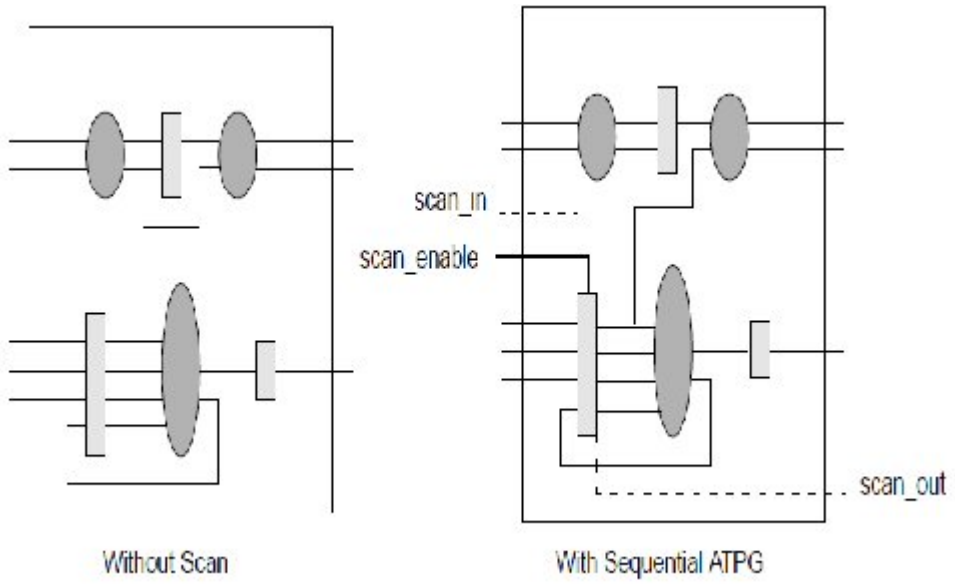

Figure 3.3 Scan Path Through a Partial-Scan Design. Ovals represent combinational logic and rectangles represent sequential logic

Typically, a partial-scan design does not allow test coverage to be as high as for a similar full-scan design. The level of test coverage for a partial-scan design depends on the location and number of scan registers in that design, and the ATPG effort level selected for the Fast-Sequential or Full-Sequential ATPG algorithm.

3.7 Boundary Scan

Boundary scan is a DFT technique that simplifies printed circuit board testing using a standard chip-board test interface. The industry standard for this test interface is the *IEEE Standard Test Access Port and Boundary Scan Architecture* (IEEE Std 1149.1). The boundary-scan technique is

often referred to as JTAG. JTAG is the acronym for Joint Test Action Group, the group that initiated the standardization of this test interface [20]. Boundary scan enables board-level testing by providing direct access to the input and output pads of the integrated circuits on a printed circuit board. Boundary scan modifies the I/O circuitry of individual ICs and adds control logic so the input and output pads of every boundary scan IC can be joined to form a board-level serial scan chain. The boundary-scan technique uses the serial scan chain to access the I/O ports of chips on a board. Because the scan chain comprises the input and output pads of a chip's design, the chip's primary inputs and outputs are accessible on the board for applying and sampling data. Boundary scan supports the following board-level test functions

- Testing of the interconnect wiring on a printed circuit board for shorts, opens, and bridging faults
- Testing of clusters of non-boundary-scan logic
- Identification of missing, misoriented, or wrongly selected components
- Identification of fixture problems
- Limited testing of individual chips on a board

Although boundary scan addresses several board-test issues, it does not address chip-level testability. To provide testability at both the chip and board level, combine chip-test techniques (such as internal scan) with boundary scan Figure 3.4 shows a simple printed circuit board with several boundary scan ICs and illustrates some of the failures that boundary scan can detect [20]

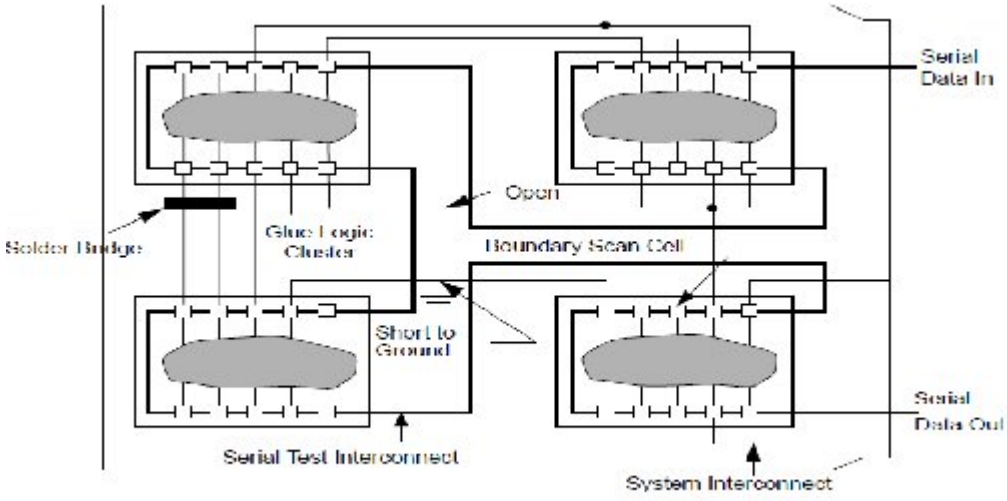

Figure 3.4 Board Testing with IEEE Std 1149.1 Boundary Scan

3.8 Scan Insertion Rules

These are the guidelines which should be followed for ATPG testing and as well as for identifying ports to be used for test I/O. These provided guidelines if implemented properly help prevent many problems that commonly occur during Automatic Test Pattern generation [17]

3.8.1 Internally generated pulsed signals

While TetraMax is in ATPG test mode, it should always be ensured that clocks and asynchronous set or reset signals come from primary inputs. The benchmark design should not include internally generated clocks or asynchronous set or reset signals

- Clock dividers should not be used while ATPG testing. If the design contains clock dividers, These should be bypassed in ATPG test mode A scan chain must shift one bit for one scan clock. TEST signal of ATPG tool can be used to control the source of the internal clocks, so that in ATPG test mode one can bypass the clock divider and source the internal clocks from the primary CLK output

- Gated clocks should not be present in the design .If the design at all contains clock gating, the control side of the gating element should be constrained, while in ATPG test mode as shown in figure 3.5

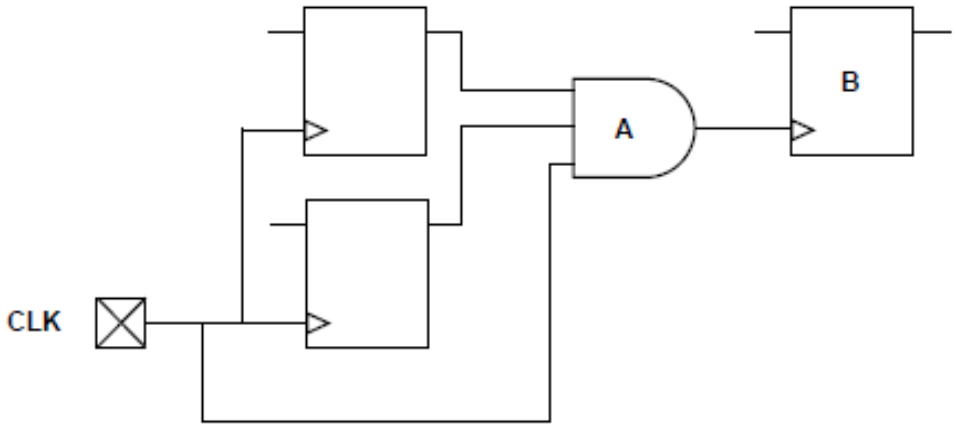

Figure 3.5 Problem of Gated Clock

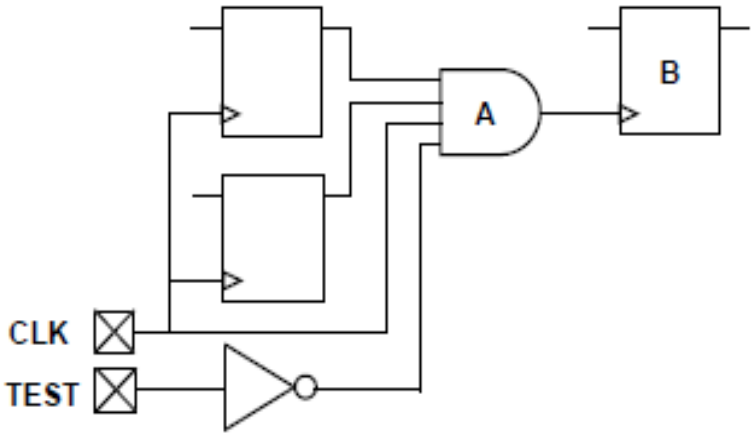

Figure 3.6 Solution to gated clock problem

In Figure 3.6 shown above, the TEST input controls a MUX that changes the clock source for register B. Another way to achieve the same results can be to add gates C1 and C2 as shown in figure 3.7. This will provide observability for the output of gate A, otherwise gate A is unobservable and all faults into A are ATPG non testable.

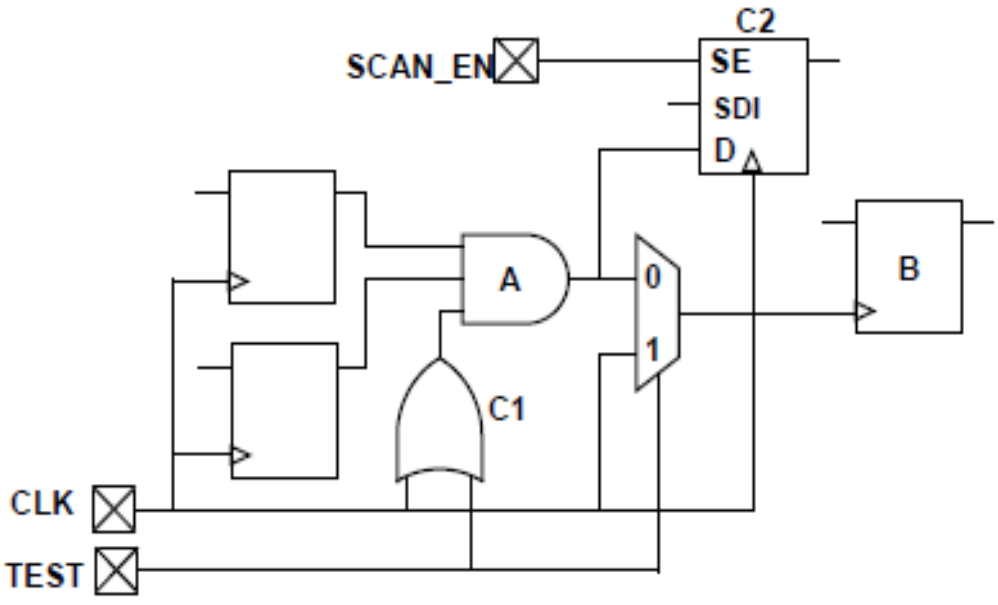

Figure 3.7 Alternate solution to gated clock problem

- It should also be taken care of that Phase-locked loops (PLLs) are never used as clock sources. If the design contains PLLs, the clocks should be bypassed while in ATPG test mode.
- Similarly pulse generators should not be used in circuits for testability. Presence of pulse generators result in skew problems in ATPG test mode, such as the one shown in figure 3.8 below

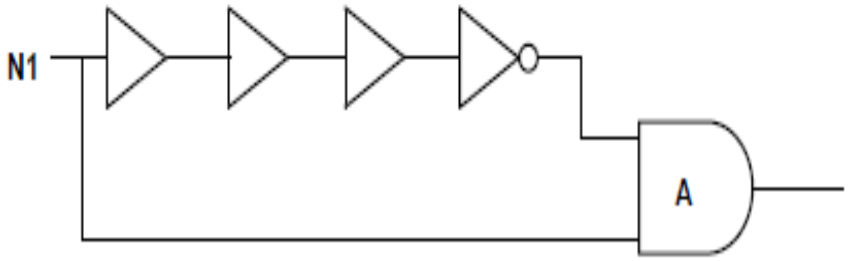

Figure 3.8 Problem of Pulse Generator

Therefore if at all the design contains pulse generators, they are bypassed using a MUX with the select line constrained to a constant value or shunted with AND or OR logic so that the pulse generators do not pulse while in ATPG test mode, as shown in Solution 1 and Solution 2 in figures 3.9 and 3.10 below

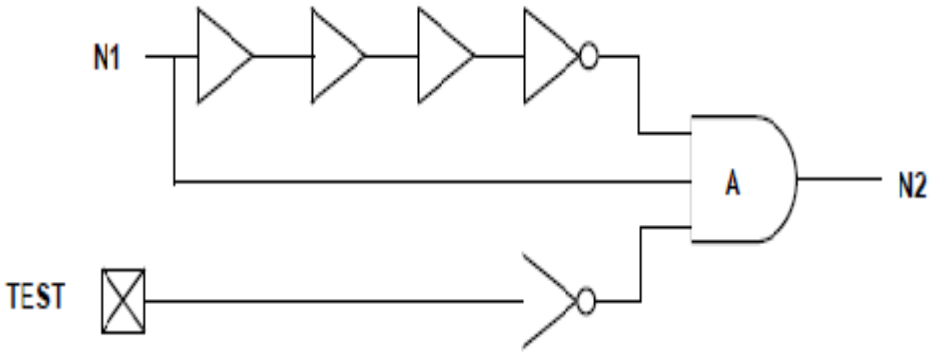

Figure 3.9 Solution 1 to problem of pulse generator

In Solution 1, the TEST input disables the pulse generator. However, if this solution is implemented then, any sequential elements that use N2 as a clock source would no longer have a clock source.

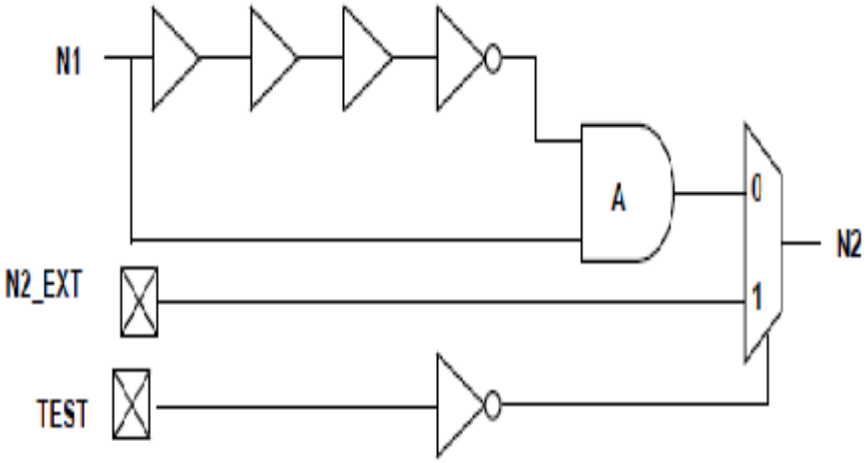

Figure 3.10 Solution 2 to problem of pulse generator

This problem can be solved by solution 2as shown in figure 3.10. In this case, the TEST input is used to multiplex out the original pulse and replace it with access from a top-level input port. Hence other circuits will not get affected by implementing this solution.

- A power-on reset circuit should never be used in ATPG test mode. A power-on reset circuit is essentially an uncontrolled internal clock source that operates when the power is initially applied to the circuit as shown in figure 3.11 below. This will make the circuit uncontrollable and testing will not be reliable.

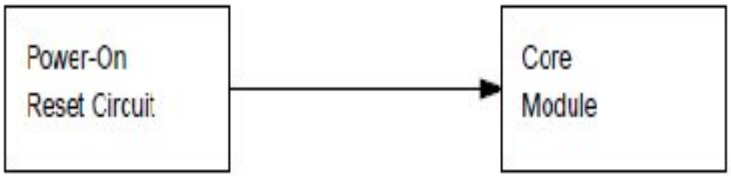

Figure 3.11 Power on reset circuit configuration to avoid

Measures are always taken to prevent a power-on reset circuit from operating during automatic test pattern generation. One of the ways to overcome this problem is by using the test mode control signal to multiplex the power-on reset signal so that it comes from an existing reset input or some other primary input during test [21]. This is shown in figure 3.12 below

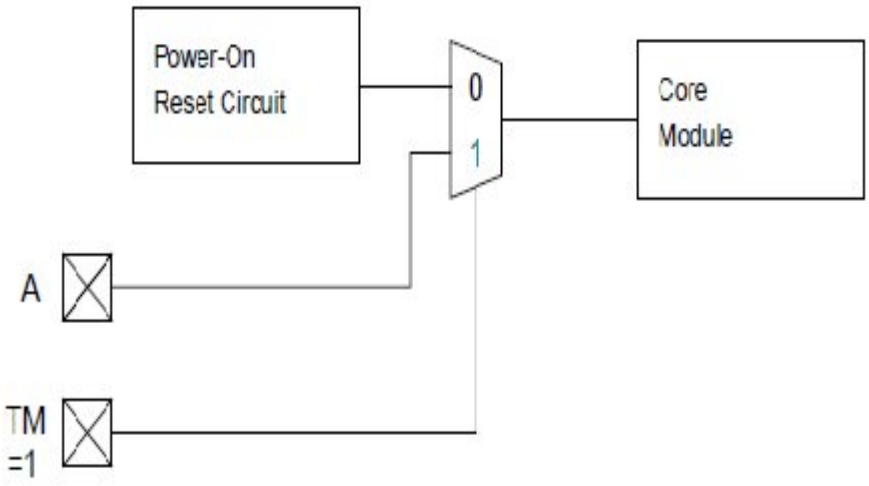

Figure 3.12 Power-On Reset Circuit Test Method 1

Another alternative is to use the test mode control signal to block the power-on reset source so that it does not have any effect during test pattern generation. This is shown in figure 3.13 below

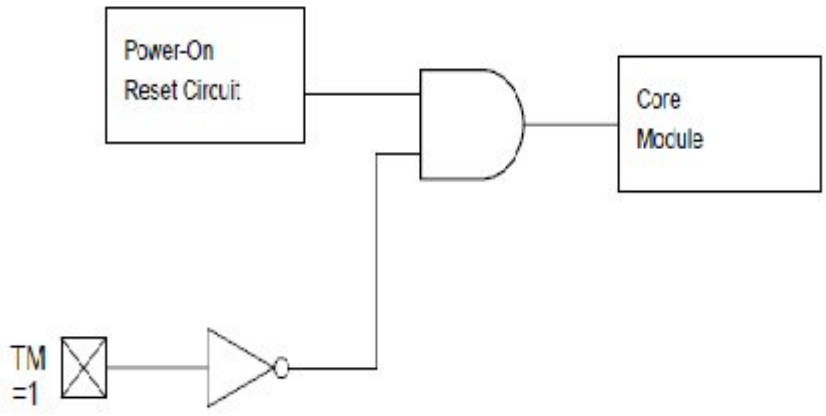

Figure 3.13 Power-On Reset Circuit Test Method 2

The first of the two methods discussed previously is usually better because it is less likely to cause a reduction in test coverage.

3.8.2 Clock Control

While performing ATPG test mode, control of clock paths is completely provided to scan chain flip-flops. The clock, set, reset paths to scan chain elements must be fully controlled in test mode Skew can arise from clock tree buffers, but a more serious source of skew is gating and multiplexing logic that is inserted on clock lines to obtain supposedly congruent clocks for the scan chain. Gating logic is not necessarily discouraged, but its introduction can cause holdtime problems that must be addressed. Therefore scan chains are constructed to reduce the risk of skew outside of tolerable limits. Skew can result in hold-time violations if it delays downstream flip-flop clocks longer than it delays upstream flip-flop clocks. If in a given circuit clock passes through a multiplexer, then the select line of the multiplexer is constrained to a constant value. If a clock passes through a combinational gate, the other inputs of the gate are constrained to a constant value. This is shown in figure 3.14 and figure 3.15

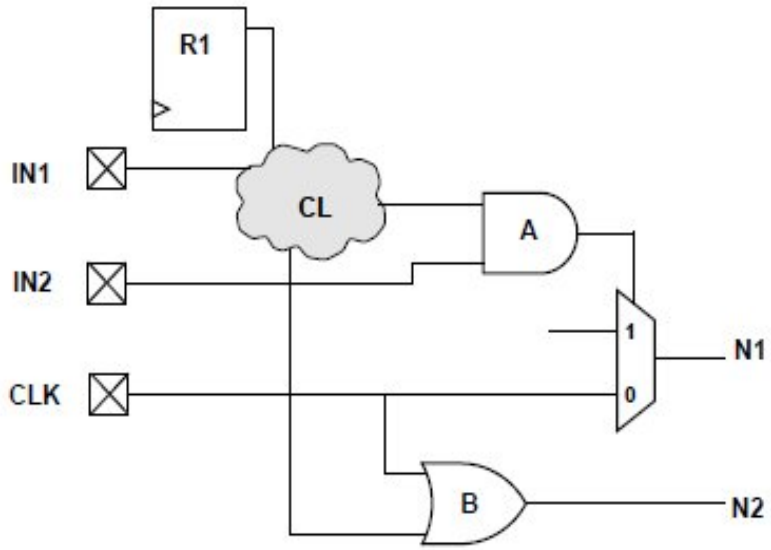

Figure 3.14 Problem of Clock paths through combinational gates

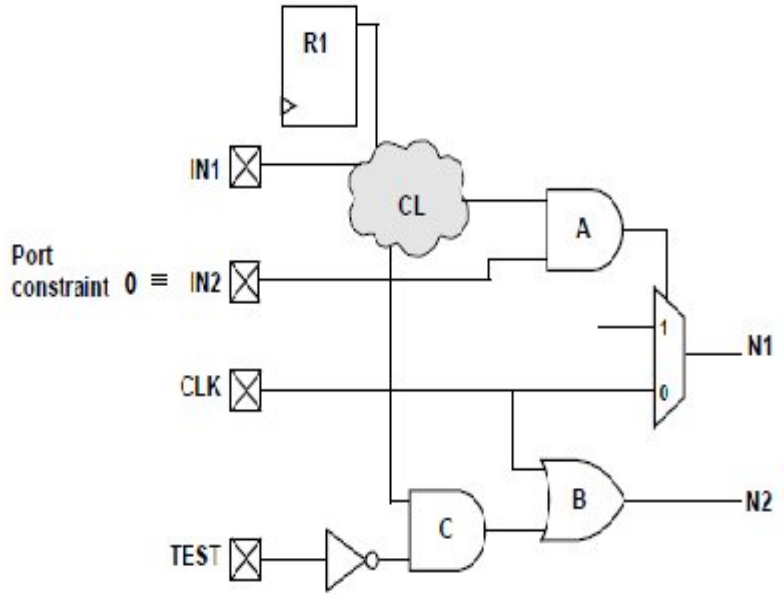

Figure 3.15 Solution to problem of clock paths through combinational gates

It is also observed that clock signals should pass directly through JTAG I/O cells without passing through a multiplexer, unless the multiplexer control can be constrained as shown in figure 3.16 below

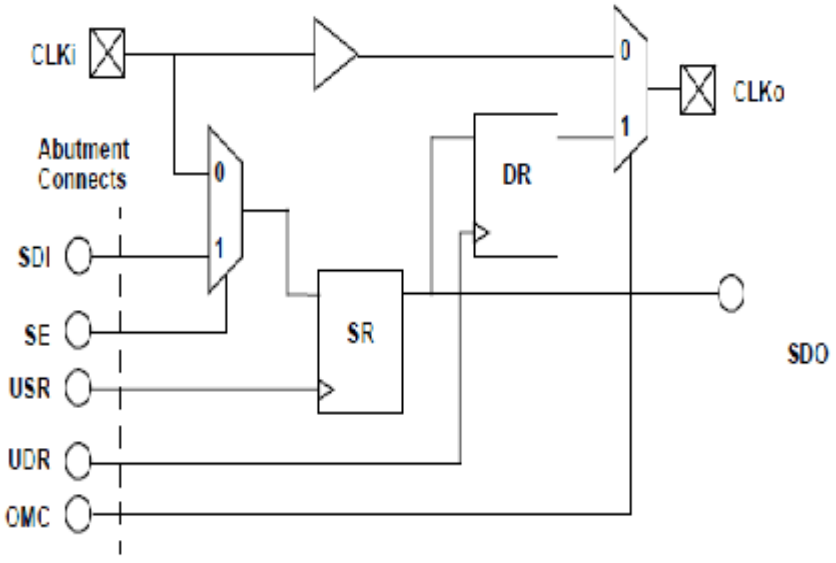

Figure 3.16 Problem of Clock/Set/Reset Inputs and JTAG I/O Cells

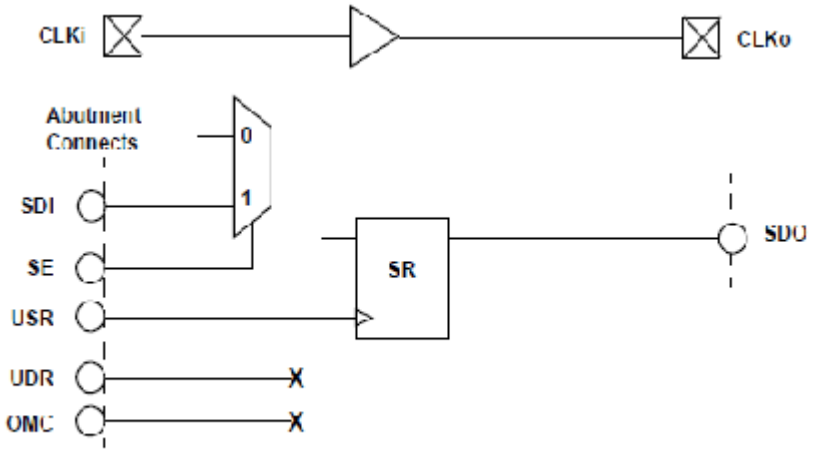

Figure 3.17 Solution to problem of Clock/Set/Reset Inputs and JTAG I/O Cells

Usage of bidirectional clocks and asynchronous set/reset ports is also avoided for the same reasons shown in figure 3.18. If the design supports bidirectional clocks or asynchronous set or reset ports, then these are forced to operate as unidirectional ports while in ATPG test mode

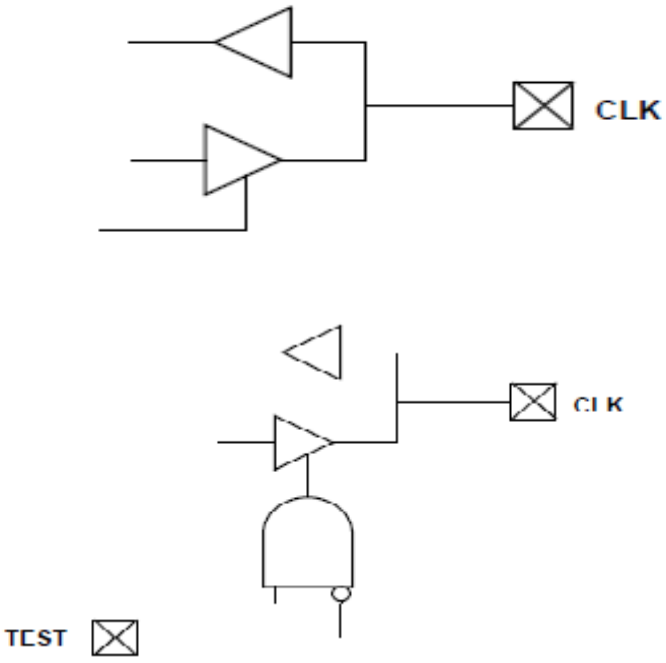

Figure 3.18 Solution to problem of Bidirectional Clock/Set/Reset

3.8.3 Pulsed signals to sequential devices

An open path from a pulsed input signal either from clock or asynchronous set or reset to a data-capture input of a sequential device is not allowed during ATPG test. Similarly a path from a pulsed input to both the data input and the clock of the same flip-flop is also avoided. As shown in figure 3.19 below, the value of the data capture cannot be determined in the absence of timing analysis.

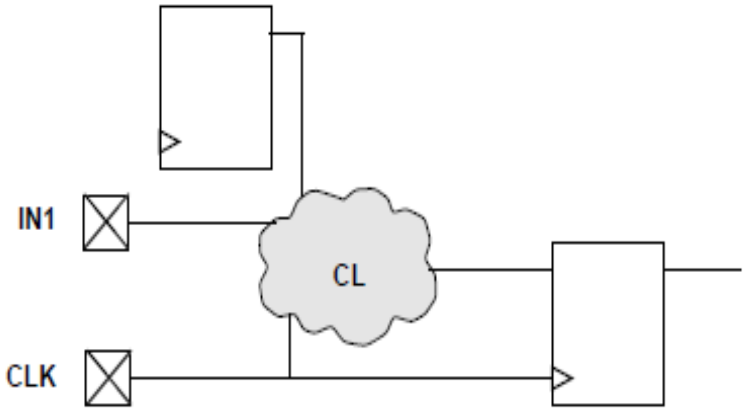

Figure 3.19 Problem of Sequential Device Pulsed Data Inputs

If the design contains such a path, then while in ATPG test mode, the path to either the data or clock pin is shunted with AND or OR logic, or with a multiplexer, as shown by Solution 1 and Solution 2 in figure 3.20. Therefore this problem can be avoided by using either a controllable top level input to replace the path of the clock/set/reset into the combinational cloud. This is shown as solution 1. The other way can be to use the TEST input to block the path of the clock, set, reset into the combinational cloud so that it does not pass the clock pulse while in ATPG test mode.While using ATPG test, a path from a pulsed input to both the data input and the asynchronous set or reset input of the same flip-flop is also avoided otherwise this will make the circuit uncontrollable and hence untestable.

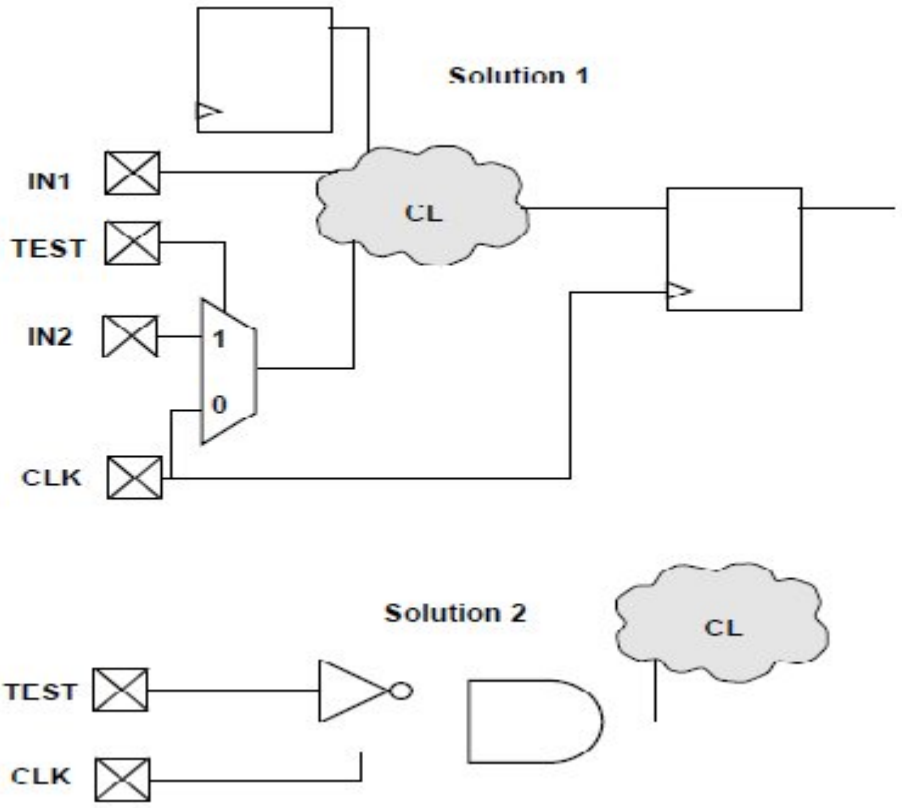

Figure 3.20 Solutions to problem of Sequential Device Pulsed Data Inputs

3.8.4 Multidriver nets

For multidriver nets, it is always ensured that only one driver is enabled during the shifting of scan chains otherwise it may result in erroneous results.

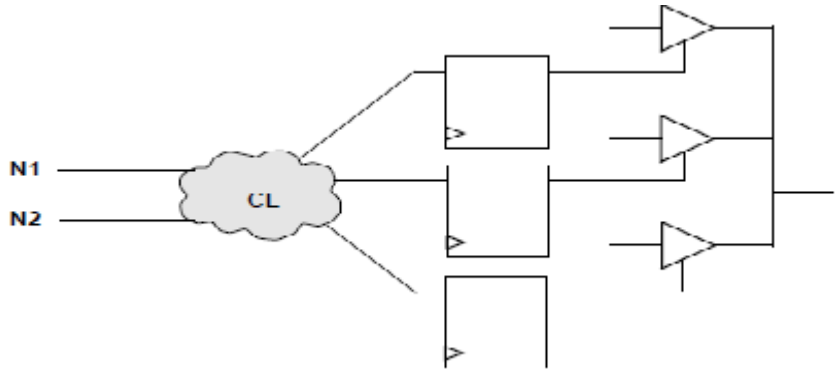

Figure 3.21 Problem of Multidriver Nets

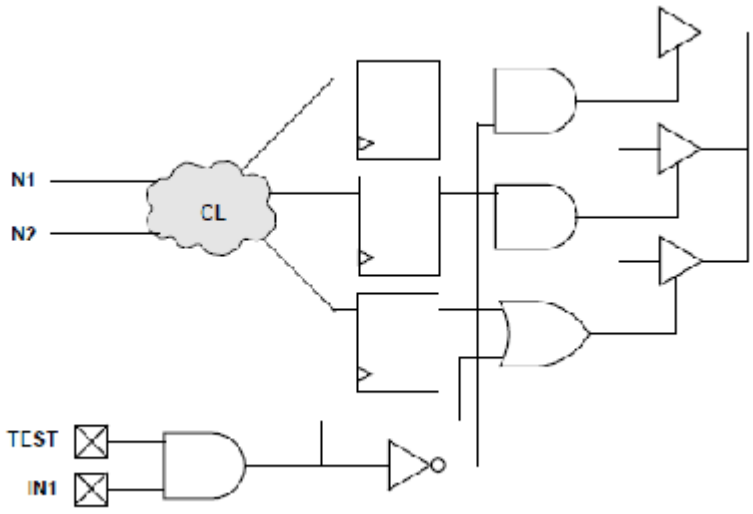

Figure 3.22 Schematic showing Global Override Input

One of the solutions is to have a primary input port that acts as a global override on internal driver enable signals in ATPG test mode, disabling all but one driver of the net and forcing that driver to an on state, as shown in figure 3.22. These primaries input port should be asserted during the scan chain load and unload operation. This design guideline is supported by DFT Compiler and is the default behaviour of DFT Compiler.

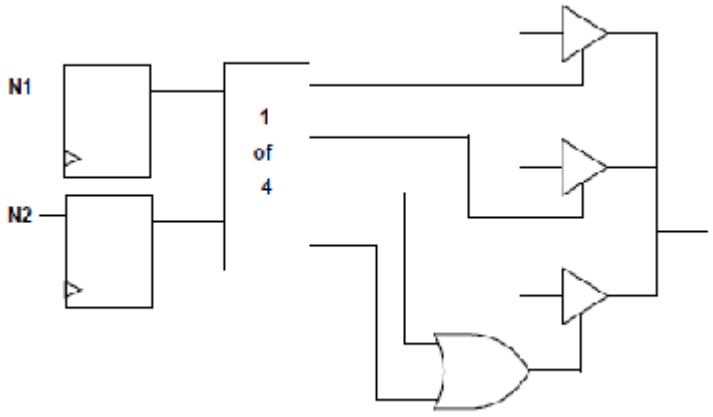

Figure 3.23 Schematic showing Deterministic decoding

The other solution is to use deterministic decoding on the driver enables. Use a 1-of-*n* logic to ensure that only one driver is enabled at all times and that at least one driver is enabled at all times, as shown in figure 3.23. Deterministic decoding might not be appropriate for some designs. For example, for a design with hundreds of potential drivers, a 1-of-*n* decoder would be too large or would add too much delay to the circuit

3.8.5 Bidirectional Port Controls

Presence of bidirectional ports in the design as shown in figure 3.24 renders it untestable.

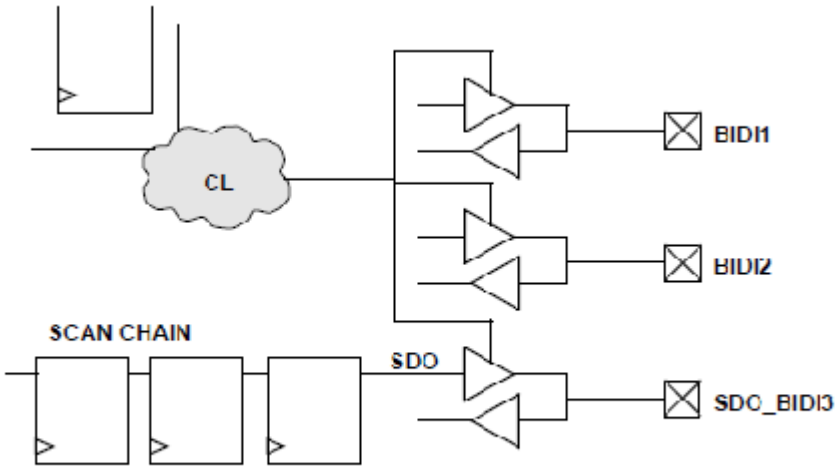

Figure 3.24 Problem of Bidirectional Port Controls

In order to overcome this problem, all bidirectional ports are forced to input mode while shifting scan chains, using a top-level port as control. TEST control is used to control the disabling logic and SCAN_EN ensures that the scan chain outputs are turned on. The top-level port is often tied to a scan enable control port. However, there are advantages of performing this function on a different port, if extra ports are available, because keeping the control of the bidirectional ports separate from the scan enable gives the ATPG algorithm more flexibility in generating patterns. If this guideline is followed then it can be easily ensured that no internal or I/O contention can occur during scan chain load or unload operations. It should also be noted that those scan chain outputs that use bidirectional or three-state ports are forced into output mode while shifting scan chains, using a top-level port usually SCAN_ENABLE

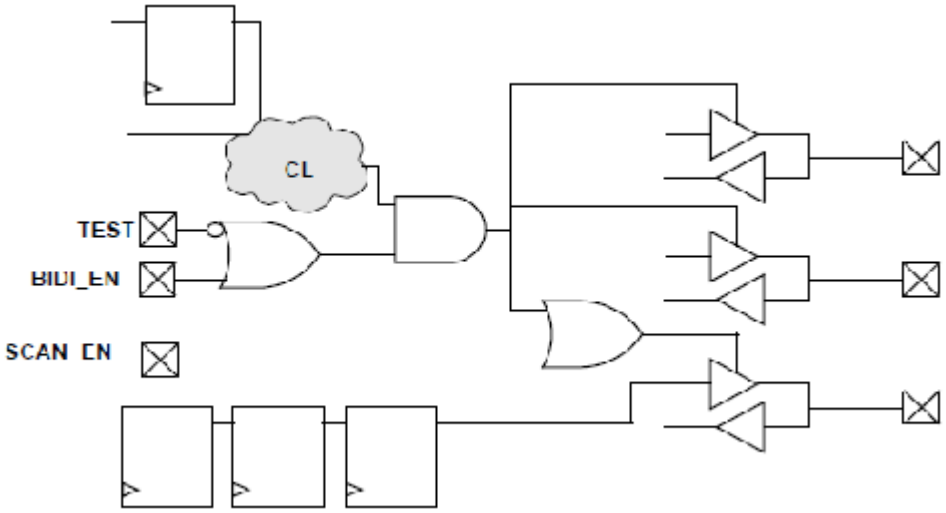

Figure 3.25 Solution to problem of Bidirectional Port Controls

3.8.6 Clocking Scan Chains: Clock Sources, Trees, and Edges

It is advisable to use a single clock tree to clock all flip-flops in the same scan chain in ATPG test mode. As shown in shown in figure 3.26 below, the two clock sources can cause a race condition. For example, if CK1 leads CK2 because of jitter or differences in clock tree delays, then R2 clocks before R3. Because R2's output is changing while R3 is clocking, a race condition results.

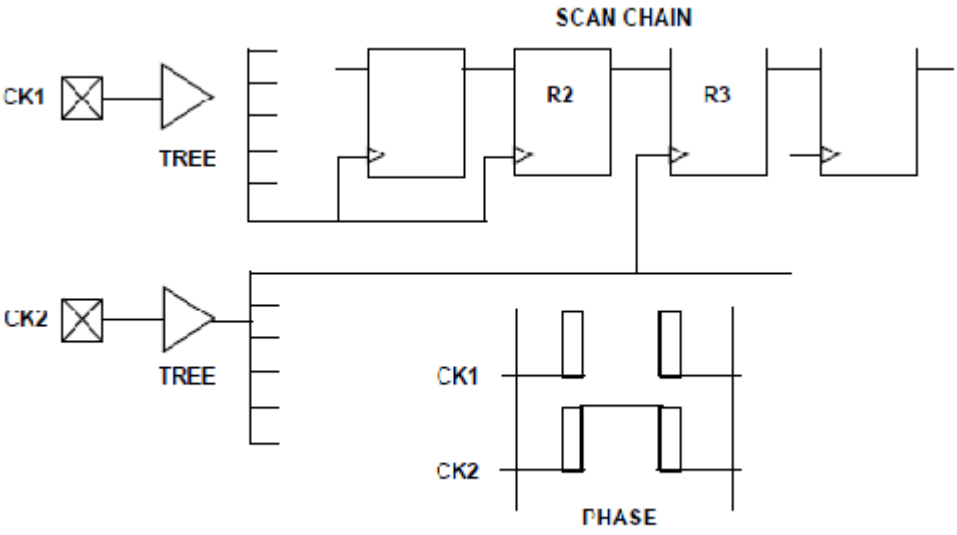

Figure 3.26 Problem of Multiple Clock Trees

The solution to this problem is shown in figure 3.27. A resynchronization register or latch (SYNC) is used between R2 and R3. This resynchronization register is clocked by the opposite phase of the clock used for R2

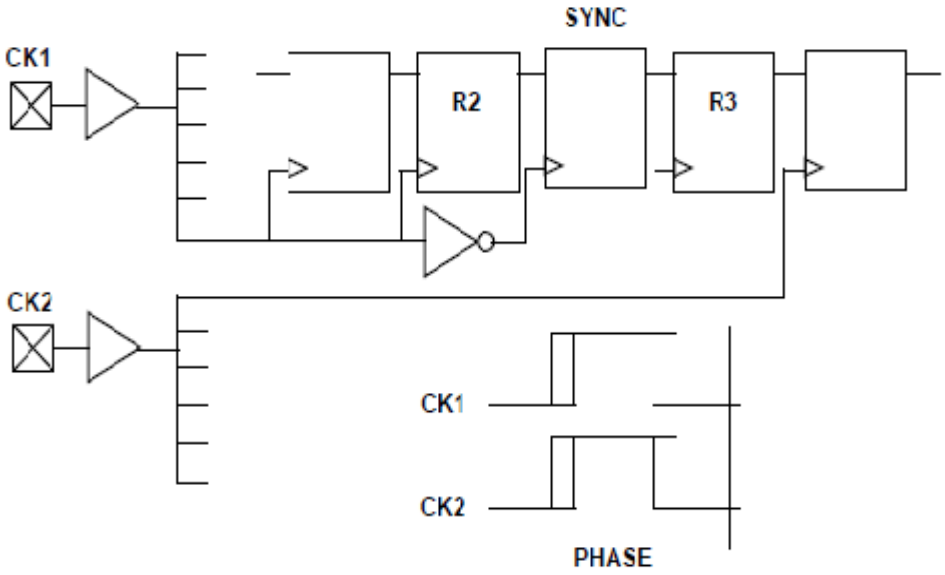

Figure 3.27 Solution to problem of Multiple Clock Trees

In addition to this, each clock tree is treated as a separate clock source in designs that have a single clock input port but multiple clock tree distributions

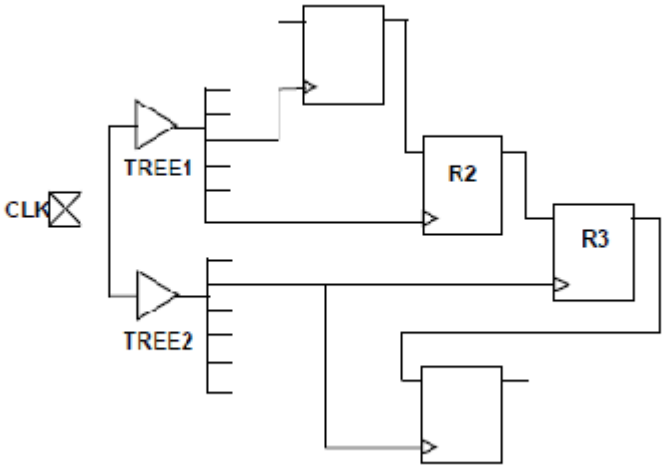

Figure 3.28 Problem of Single Clock with Multiple Clock Trees

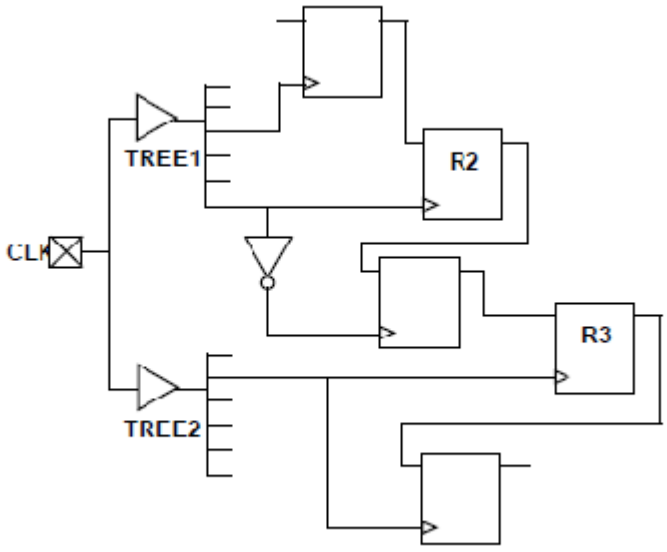

Figure 3.29 Solution: Single Clock with Multiple Clock Trees

If possible, all flip-flops should be clocked on the same scan chain on the same clock edge. If this is not possible, then all flip-flops that are clocked on the trailing clock edge should be grouped together and placed at the front of the scan chain closest to the scan chain input and all flip-flops that are clocked on the leading clock edge should be group together and placed closest to the scan chain output. In Figure 3.30, B1 and B2 are always loaded with the same data as A1 and A4, respectively, during scan chain loading, because they are clocked on the trailing edges. Thus, parts of the circuit that require A1 and B1 (or A4 and B2) to have opposite values are untestable

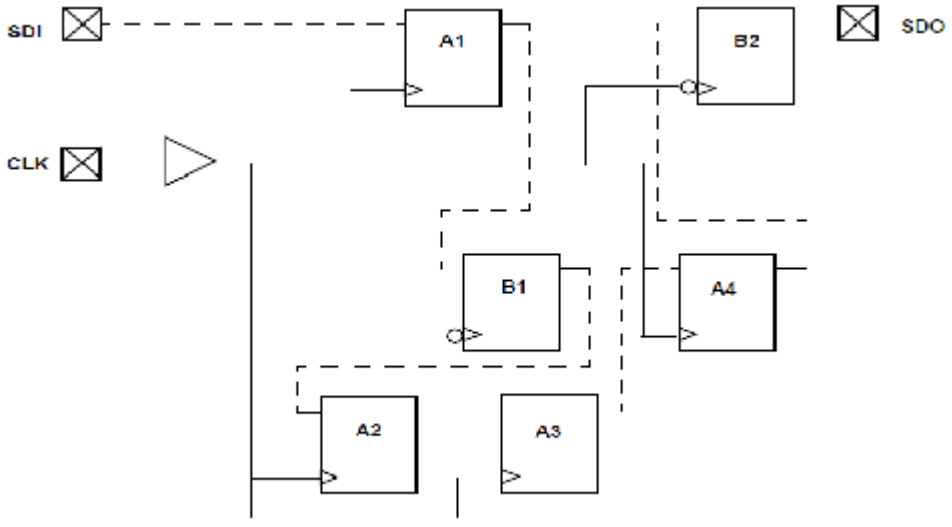

Figure 3.30 Problem of Mixed Clock Edges on a Scan Chain

In Figure 3.31, the scan chain registers are ordered so that all of the trailing-edge cells are grouped together at the front of the scan chain. B1 and B2 can be set independently of A1 and A4

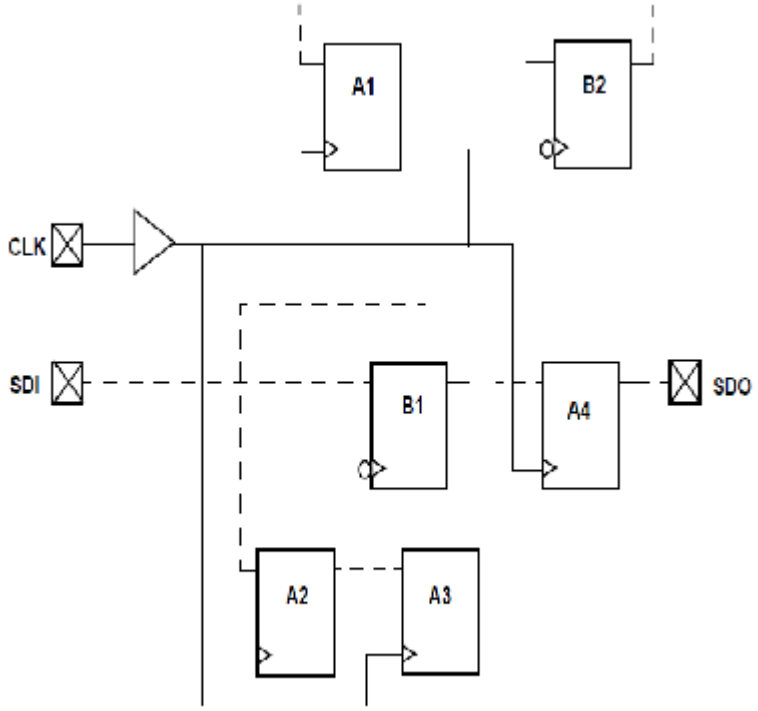

Figure 3.31 Solution: Mixed Clock Edges on a Scan Chain

A common design technique when both edges of the same clock are used for normal operation of scan chain flip-flops is to use an XNOR in place of an INV to form the opposite clock polarity. Then, in test mode, the XNOR can be switched from an inverter into a buffer. This technique is not advisable unless the timing during test mode can be analysed to ensure that no timing violations can occur during the application of any clocks. While in normal operation, there are essentially two clock zones of opposite phase. The phasing of the two clocks is such that reasonable timing is achieved between flip flops that are on opposite phases of the clock. When one of the clocks is no longer inverted, two clock tree distributions are driven by the same-phase signal, resulting in timing-critical configurations in ATPG mode that do not exist in normal functional mode as shown in figure 3.32. To prevent this problem, replace the XNOR gate with an inverter, as shown in Figure 3.33. If you need the XNOR function, use it locally in the vicinity of the affected gates, rather than on the input side of a clock tree

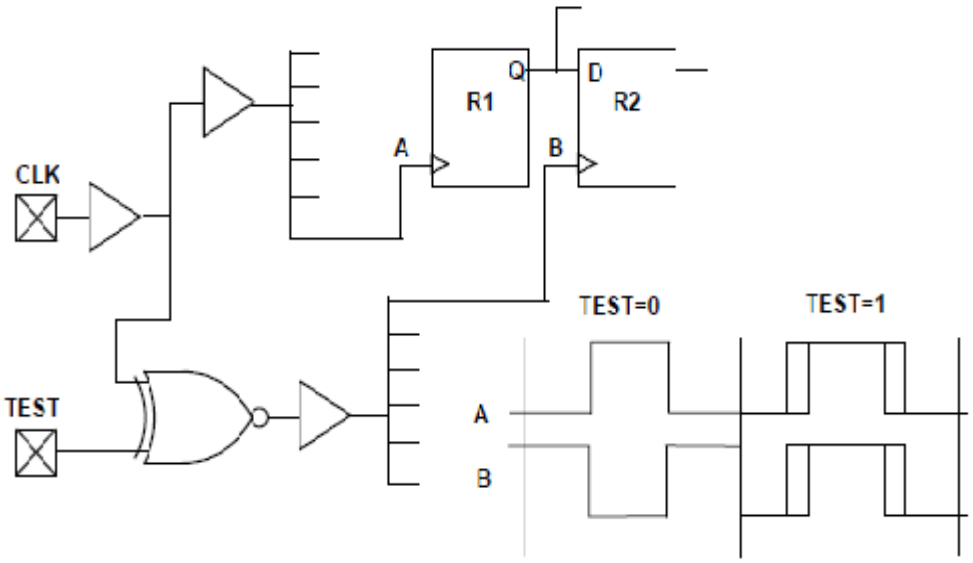

Figure 3.32 Problem of XNOR Clock Inversion and Clock Trees

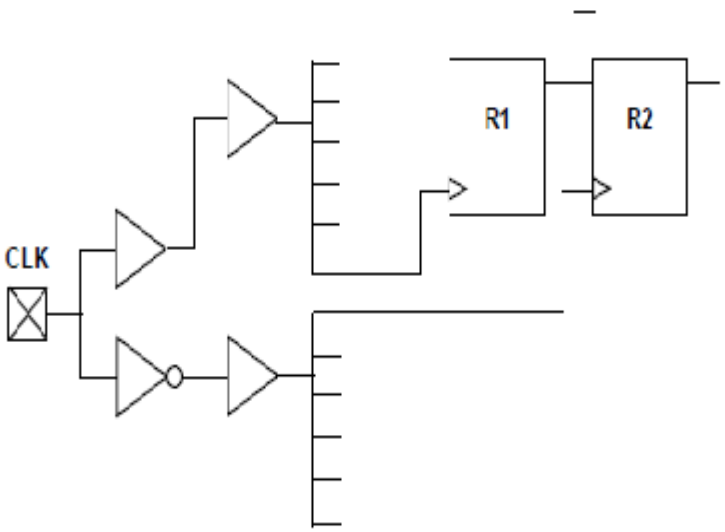

Figure 3.33 Solution to problem of XNOR Clock Inversion and Clock Trees

3.8.7 Protection of RAMs During Scan Shifting

To protect RAMs from random write cycles, RAM write clock or write enable lines are disabled while shifting scan chains**.** In ATPG test mode, RAMs must remain undisturbed by random write

cycles while the scan chains are being shifted. This can be accomplished by disabling the write clock or write enable line to each data write port during ATPG test mode. Often, the SCAN_ENABLE control is used for this function, coupled with an AND or OR gate, as appropriate**.** However, to also achieve controllability over the write port, a separate top-level input other than SCAN_ENABLE should be used. The RAM write control is usually used as a pulsed port, while the SCAN_ENABLE is a constant value. Trying to achieve both at once usually presents problems that can be avoided by using separate ports

If controllability of RAMs and ROMs is required for ATPG generation, their read and write control pins are connected directly to a top-level input during ATPG test mode. This is most conveniently accomplished by using a Multiplexer, which switches control from an internal to a top-level port. Multiple RAMs can share the same control port for the write port. In the figure 3.34 if the registers are in scan chains, random patterns that occur while loading and unloading scan chains are written to the RAMs. Thus, the RAM contents are unknown and treated as X

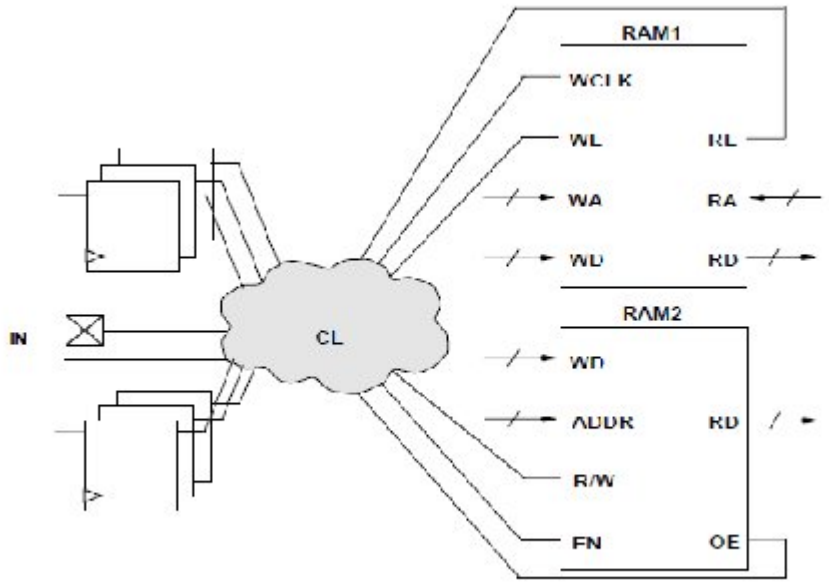

Figure 3.34 Problem of RAM/ROM Control

MUX controls activated by TEST mode bring the write control signals up to the top-level input ports. For achieving controllability and higher test coverage, direct control of write ports is more important than control of read ports.

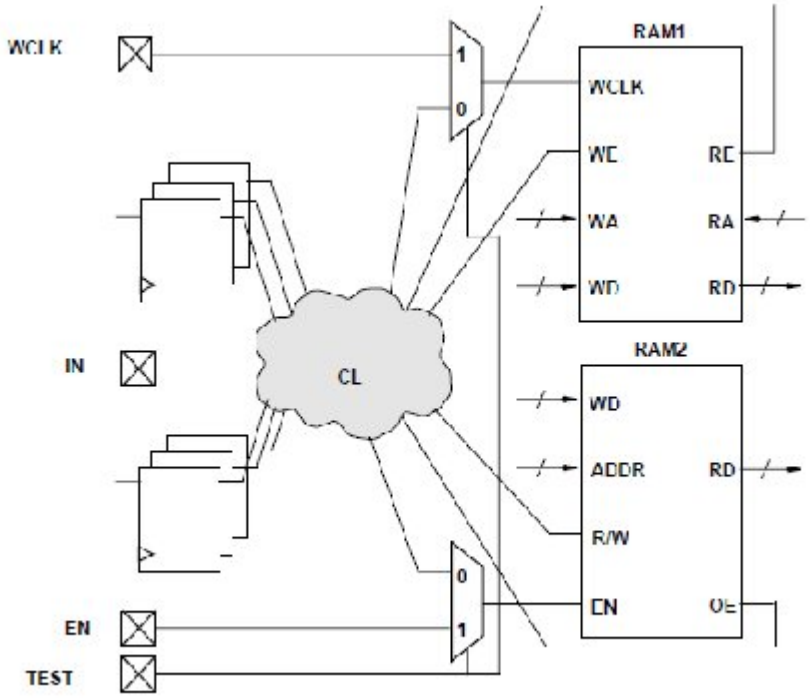

Figure 3.35 Solution to the problem of RAM /ROM control

An open path from a pulsed signal to a RAM's or ROM's data, address, or control inputs (except read/write control) should not be allowed. If a combinational path exists from a defined clock or asynchronous set or reset port to a data, address, or control pin of a RAM or ROM, the ATPG algorithm treats the memory device as filled with X. The exceptions are the read clock and write clock signals, which are operated in a pulsed fashion but should not be mixed with a defined Clock. In Figure 3.36, the address or data inputs are coupled with a clock/ set/reset port, so their values are not constant while capture clocks are occurring elsewhere in the design. The result is that RAM read and write data cannot be determined, instead Xs are used.

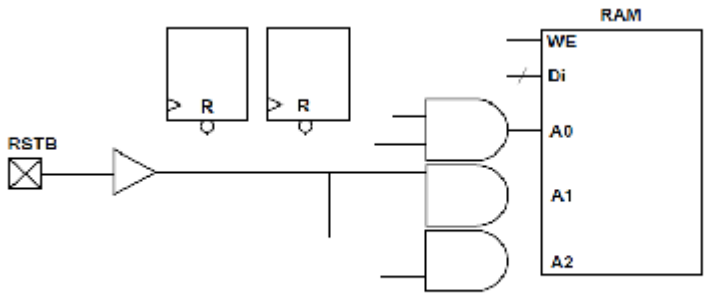

Figure 3.36 Problem of RAMs/ROMs and Pulsed Signals

In Figure 3.37, the TEST input disables pulsed paths during ATPG test mode

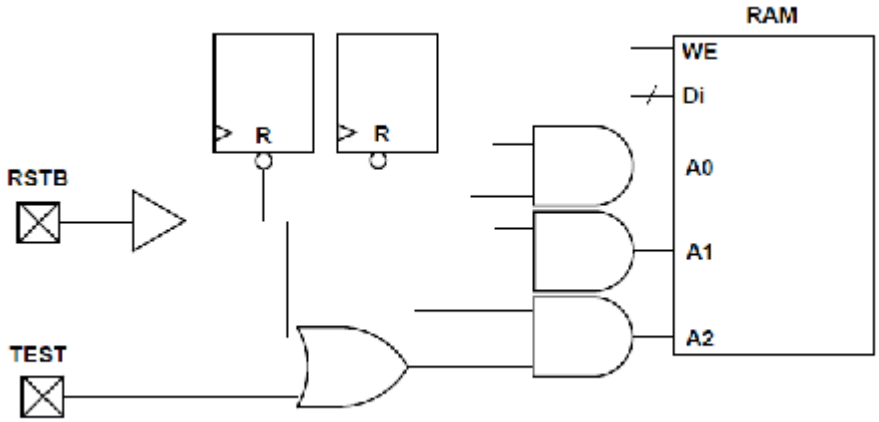

Figure 3.37 Solution to problem of RAMs/ROMs and Pulsed Signals

3.8.8 Bus Keepers

While in ATPG test mode, combinational gate path from any pulsed port should never be allowed to drive the enable controls of three-state drivers that contribute to a multidriver net. In Figure 3.38, the TEST input redirects the control to a top-level port, and the port is constrained to a value that does not affect the driver enables. Then, on the tester and during simulation, the port is driven with the same signal as that on the original CLK port

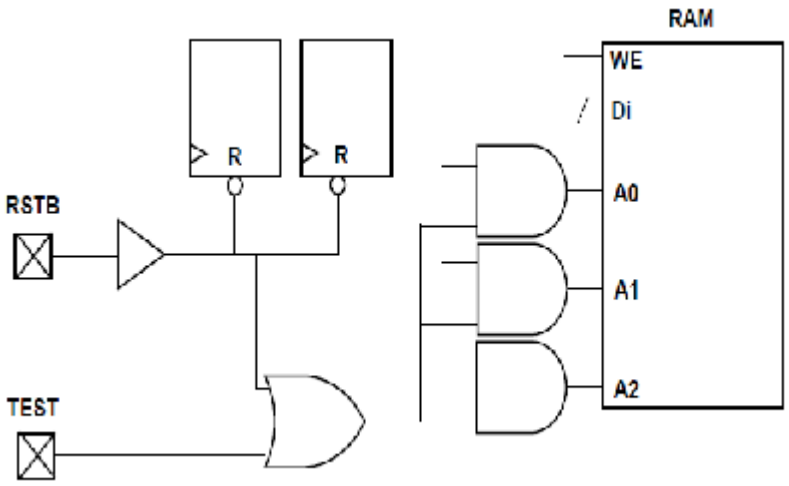

Figure 3.38 Solution to problem of Bus keepers

3.9 What is TetraMAX?

TetraMAX is a high-speed, high-capacity automatic test pattern generation (ATPG) tool. It can generate test patterns that maximize test coverage while using a minimum number of test vectors for a wide variety of design types and design flows. It is well suited for designs of all sizes up to millions of gates [15].

TetraMAX has the following major ATPG capabilities:

- Can read design netlists in Verilog, VHDL, and EDIF formats, and test protocol information in STIL format
- Can write test pattern files in a variety of standard and proprietary formats: WGL, STIL, Verilog, VHDL, Fujitsu TDL, TI TDL91, and Toshiba TSTL2
- Offers a choice of ATPG modes e.g Basic-Scan ATPG, an efficient combinational only mode for full-scan designs, Fast-Sequential ATPG for limited support of partial-scan designs

3.9.1 TetraMAX ATPG Features

- Full-Sequential ATPG for maximum test coverage in partial-scan designs
- Supports the different design-for-test (DFT) styles, namely, multiplexed flip-flop, master, slave, transparent latch
- Provide Internal, nondecoded three-state buses
- Provides bus keepers

- Provides RAM and ROM models
- Produces and verifies ATPG patterns that avoid bus contention and float conditions
- Offers interactive analysis and debugging with the Graphical Schematic Viewer (GSV), for easy analysis of design rule violations and other conditions found in the design
- Provides links to Verilog and VHDL simulators
- Provides an integrated fault simulator that supports fault simulation of functional patterns
- Can perform direct automated test equipment (ATE) diagnostics, allowing to quickly map a test failure to a fault site in the design
- TetraMax offers three different ATPG modes: Basic-Scan, Fast-Sequential, and Full-Sequential
- Supports test pattern generation for five types of fault models: stuck-at faults, IDDQ faults, transition delay faults, path delay faults, and bridging faults

3.9.2 Command Interface

TetraMAX provides an easy-to-use command interface for interactive control of TetraMAX tasks, and a powerful command language that lets you execute command sequences in batch mode [21] Figure 3.5 below shows the main window of the TetraMAX graphical user interface (GUI)

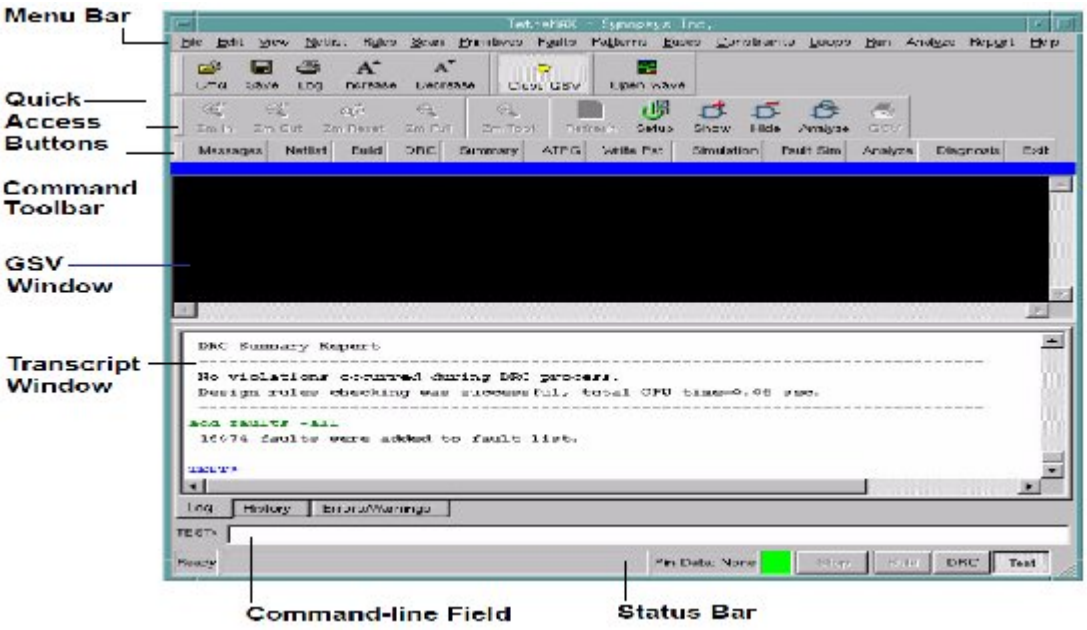

Figure 3.39 TetraMax GUI main window

3.9.3 Design Flow Using DFT Compiler and TetraMAX

TetraMAX is compatible with a wide range of design-for-test tools such as DFT Compiler. The design flow using DFT Compiler and TetraMAX ATPG is recommended for maximum ease of use and quality of results

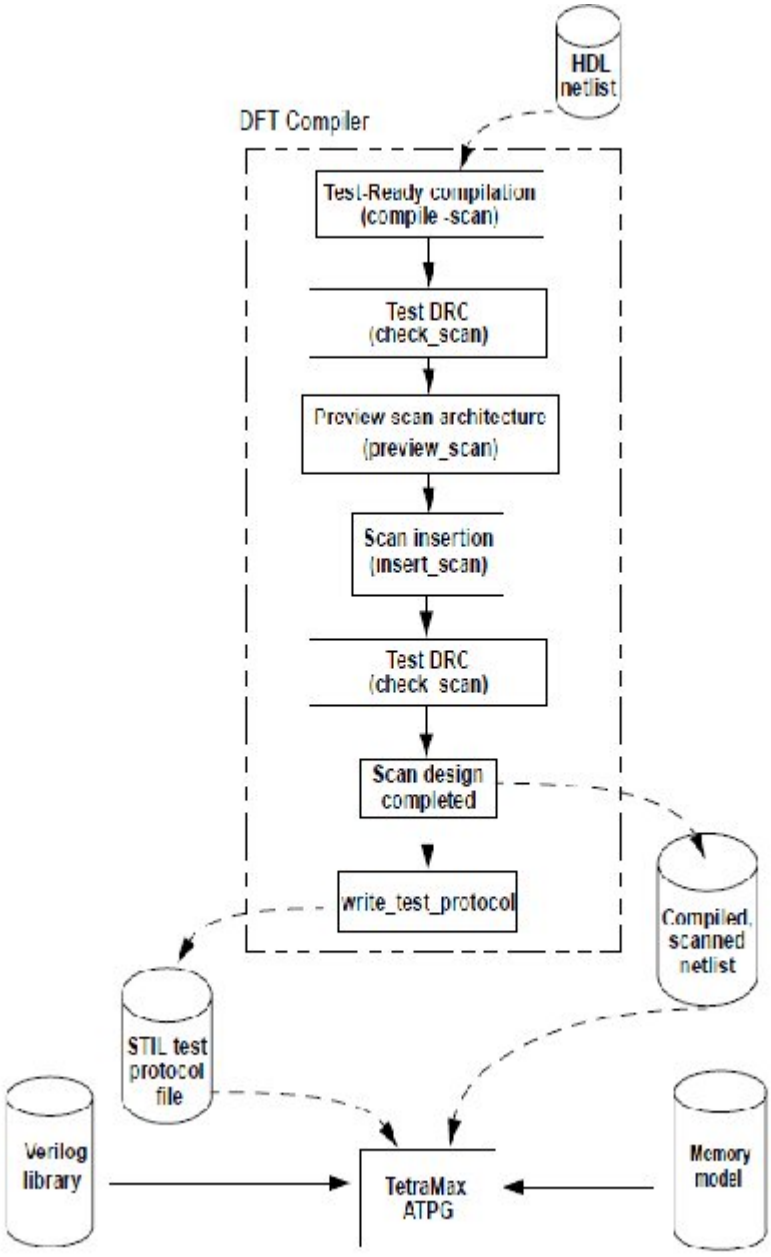

Figure 3.40 Design flow using DFT compiler and TetraMax

To successfully set up TetraMAX for test pattern generation, one must provide the following information on with the design:

• Clock ports

• Asynchronous set and reset ports

• Scan chain input and output ports

• Any ports that place the design in test mode and their active states

• Any ports that enable shifting of scan chains and their active states

• Any ports that globally control bidirectional drive and their active states

3.9.4 Basic ATPG Design Flow using TetraMax

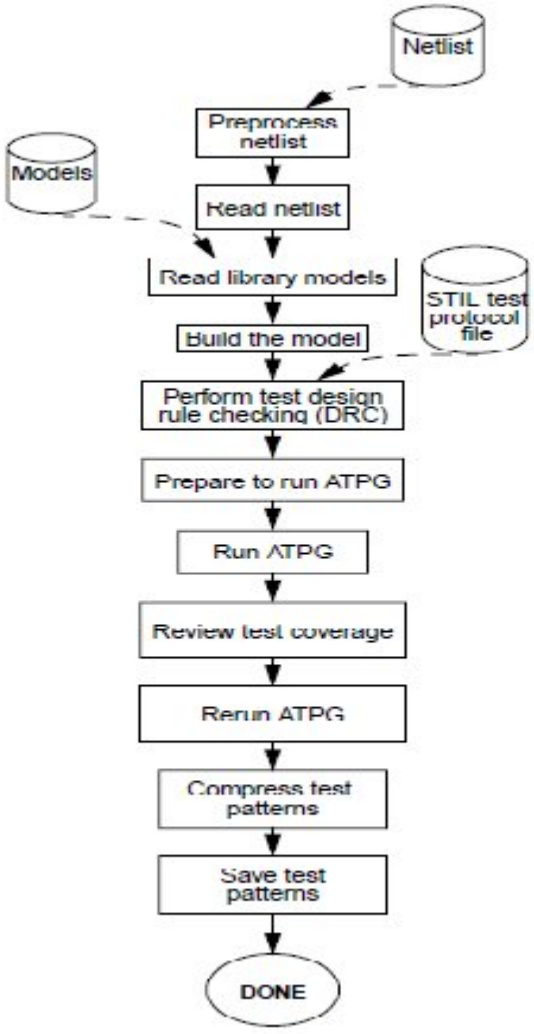

Figure 3.41 Basic ATPG Design Flow using TetraMax

Chapter 4

Problem Definition & Solution

4.1 Problem Definition

At present in ST Microelectronics India Pvt Ltd, TetraMax Version Z-2007.03 by Synopsys is being used as automatic test pattern generation and validation tool. It can generate test patterns that maximize test coverage while using a minimum number of test vectors for a wide variety of design types and design flows.

As already discussed, TetraMax requires various inputs like stil test protocol file, scanned and compiled netlist, all library and memory modules in order to integrate itself into the design flow. If a backend designer wants to use TetraMax, he must supply all the inputs manually. In addition, he must be aware of different variables used in generating TetraMax test reports. But the variable names are not uniform across all designs and sites. This makes it very difficult for the designers to hand off reports across the various sites. Therefore there is a need to maintain synchronous methodology during TetraMax run across various designs as well as across various sites of ST Microelectronics all over the world.

In the present work, we propose to develop internal STB tool which would uniquify the DFT procedures, invoke TetraMax for the user and produce results in terms of ST fault coverage. The tool proposed is expected to be more user friendly, with the designers having to provide minimal inputs, as compared to that with TetraMax ATPG.

This will efficiently maintain the compliance of design flow in the company. In addition the tool proposed is expected to check the design for testability after first synthesis itself so that DFT issues are discovered well in advance in the development of a design. Customized test reports generated by the tool will help the designer to identify where he can simplify logic cones and do a specific design optimization.

4.2 Solution

The solution is based on “tcl” which is a scripting language similar to “shell” with many advanced features. The tool developed is named “mkTmax”.

Reasons for using Tcl as a platform to develop the tool mkTmax are:

- Tcl is the most common platform on which most of EDA tools are based, therefore tool developed will be synchronous with the already existing design flow and hence can be easily integrated
- Tcl language has a straightforward syntax
- Using Tcl, one can extend the TetraMAX command language by writing reusable procedures

Proposed directory structure of “mkTmax” is as shown below:

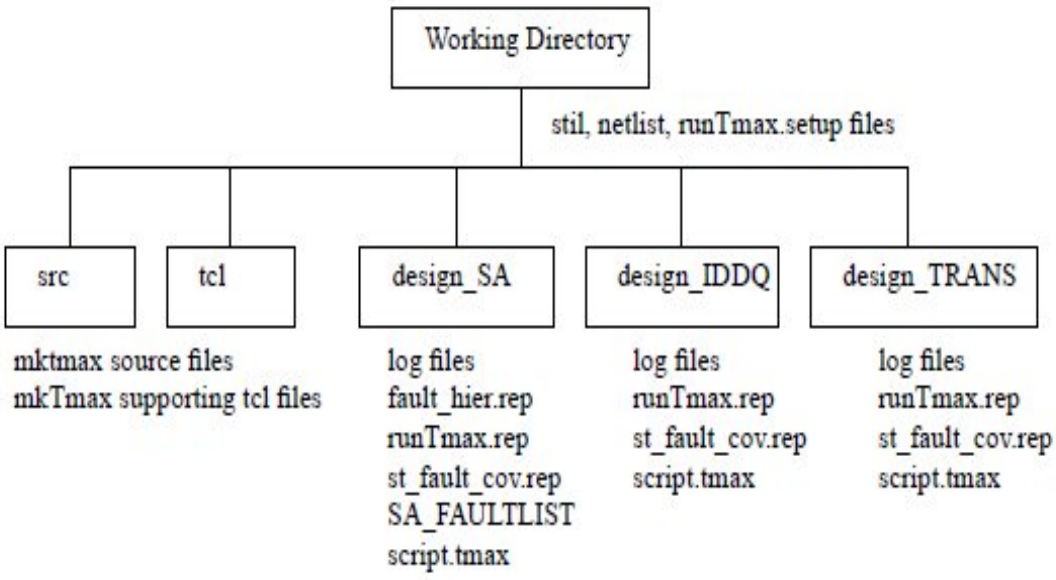

Figure 4.1 Directory Structure of mkTmax tool

4.2.1 Flowchart of Proposed Upgraded Flow

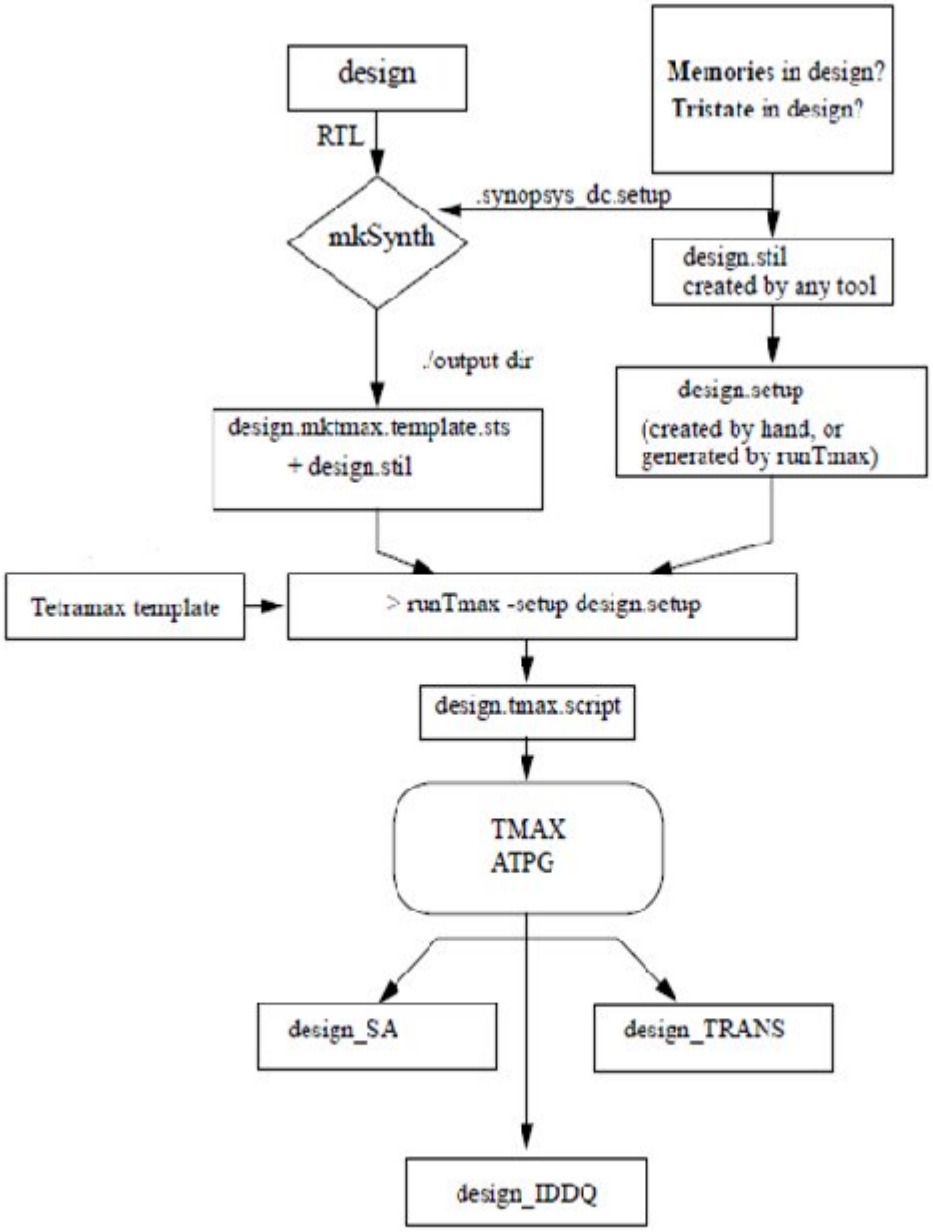

Figure 4.2 Flow Chart

4.2.2 Tetramax template file

“mkTmax” will generate a template file for TetraMax which will be the backbone of testability and it needs some specific informations like test_signals, tri-state nets etc.

The setup file is composed of three kinds of variable:

- **General design informations** : netlist path, design name, stil path,
- **Specific design informations** : test_signals, tri-state, memory
- **Scenario** : Stuck-At, Iddq, Fault transition and Store

The template file is the framework for the Tetramax script. We use “KEY_WORD” to locate where we need some information coming from the design and complete the script by dedicated commands to build the best fitted check. A number of ways have been explored before delivering the best template file. The template file is present in src directory. It defines the name of the log file (./runTmax_session.log) and report file before its post-processing (./runTmax_report.log). All “KEY_WORD” will be replaced when runTmax generates the Tetramax script.

4.2.3 Script to run Tetramax

The generated Tetramax script script (default : <design>.tmax) is the result of the template file completed by informations contained in the setup file. This file can be used with runTmax (runTmax -tmax_file <design>.tmax) or with tmax in standalone (tmax -gui <design>.tmax)

The main steps of the TetraMax script are :

1 reset old TetraMax environment

2 read corner libraries and memory libraries

3 read netlist

4 read stil file, run drc

5 run Stuck-At check (if defined in setup file) :

- run combinational step with store pattern if defined in setup file
- run additional fast sequential to increase coverage around memory

6 run IDDQ faults (if defined in setup file) :

- if there are no tristates in design, run IDDQ step
- else return to the DRC state

- add PI constraints to 0 on tst_pullenable and tst_floatbus
- save iqq_design.stil
- run drc
- run IDDQ step

7 run Transition faults :

- return to drc state
- add PI constraints to 0 on tst_scanenable
- save trans_design.stil
- run drc
- run transition step

4.2.4 Log and Report files

The Tetramax log file is ***runTmax_session.log***. This file has all the warnings and errors messages related to the run. The ***runTmax_report.log*** contains all the commands and the reports of a Tetramax session. In the ***runTmax.rep file*** the summary of the scan chains, primitives, drc errors, plus the test coverage could be found.This file can be presented as the result of the test quality report for a test configuration. This is the file used in the DFT website.

To debug testability problems with Stuck-at-faults, a hierarchical report is generated (***fault_hier.rep***). This file can help the user to identify the blocks with lower fault coverage, and improve the test quality.

4.3 How to initialize and run mkTmax with TetraMax

The following steps are necessary to install mkTmax in the working directory:

Step 1	Add tool mktMax in working directory using command “addtool”
Step 2	Execute mkTmax in the working directory.
Step 3	Check if the procedure above created “tcl” and “src” directories inside the working directory
Step 4	Copy setup fiile $design.mkTmax.template.sts to this work directory.
Step 5	The command to run the testability check is runTmax: runTmax -setup $design.mkTmax.template.sts.

The setup file is a set of tcl variables which will be used to substitute key words in template file by Tetramax command. By default, only Stuck-At scenario will be run. To investigate IDDQ and Transition faults, one can modify mk_tmax(scenario) variable in setup file.

Chapter 5

Results and Discussion

5.1 Simulation and Testing

"mktMax" tool was developed using "tcl" platform. It consists of a set of tcl scripts to determine the testability of a design. "mktMax" was run for two designs, namely, c8vtg_top_generic (a small IP design) and HDTVOut (comparatively larger and more complex design). The test results obtained with "mktMax" were benchmarked against those obtained with running TetraMax without any wrapper. These results have been tabulated as below:

Table 5.1 Comparison of fault statistics for the design c8vtg_top_generic

S.No	Fault Statistic	Values obtained without mkTmax	Values obtained with mkTmax	Error
1	SA ST Fault Coverage	98.335 %	98.334%	- .001%
2	SA ST Test Coverage	99.23 %	99.23 %	0%
3	Transition ST Fault Coverage	88.609%	88.609%	0%
4	IDDQ ST Fault Coverage	79.897%	79.895%	-.002%
5	ATPG Effectiveness	99.27%	99.27%	0%

Table 5.2 Comparison of fault statistics for the design HDTVOut

S.No	Fault Statistic	Value obtained without mkTmax	Value obtained with mkTmax	Error
1	SA ST Fault Coverage	98.332 %	98.331%	- .001%
2	SA ST Test Coverage	99.24 %	99.24 %	0%
3	Transition ST Fault Coverage	88.607%	88.606%	- .001%
4	IDDQ ST Fault Coverage	79.897%	79.895%	- .002%
5	ATPG Effectiveness	99.27%	99.26%	- .01%

5.2 Conclusion

From the results tabulated in table 5.1 and table 5.2, it can be seen that the fault and test coverages obtained by running the tool developed namely mktMax tally fairly with the expected values. For SA ST Fault coverage, it can be concluded that mktMax has been able to produce 100% accurate results. For other fault statistics too, the difference between values obtained when TetraMax ATPG is run with mktMax and those obtained when TetraMax is run without mktMax is negligible.

Further the tool has been found to be producing expected results for both the designs one of them being a fairly large and complex design. Hence it can be concluded that the tool ("mktMax") developed is efficient enough to determine testability of a given design with expected accuracy. It can be of great help to backend designers who can use it to discover DFT issues well in advance in development process of a design

Since the testability of the circuit can be enhanced before the design is committed to a gate-level structure, the approach proves to be quite economical. Significant improvements in fault coverage and ATPG efficiency, and speedups in ATPG time can be obtained by a gate-level and a high-level test generator after high-level scan selection

5.3 Future Scope

At present only one directory is created to store the results of mktMax run for all fault scenarios. The tool can further be upgraded to create different directories for different fault types. In evaluating the efficiency of tool developed, we have not taken any design with memory components. The tool can be developed further to enhance fault coverage of blocks with memory with basic scan run.

It is also proposed to improve the format of summary reports generated so that they can be easily read. This will make the TetraMax ATPG usage more user friendly as well as it's integration into the flow will be more compliant

References

[1] Vivek Chickermane, Janak H. Patel, "Addressing Design for Testability at the Architectural Level", IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems. Vol. 13, No. 7, July 1994

[2] T. W. Williams, "Design For Testability", IEEE 19th Design Automation Conference, 1982

[3] IrithPomeranz, Sudhakar M. Reddy, "Transparent DFT: A Design for Testability and Test Generation Approach for Synchronous Sequential Circuits", IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. 25, No. 6, June 2006

[4] A. Chatterjee, Naveena Nagi, "Design for Testability and Built-In Self-Test of Mixed-Signal Circuits: A Tutorial", IEEE 10th International Conference on VLSI Design, January 1997

[5] Sanghyeon Baeg, William A. Rogers, "A Cost-Effective Design For Testability: Clock Line Control and Test Generation Using Selective Clocking", IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. 18, No. 6, June 1999

[6] Vijaya Raghavan, Mojdeh Shakeri, Krishna R. Pattipati, "Test Sequencing Problems Arising in Test Planning and Design for Testability", IEEE Transactions On Systems, Man, And Cybernetics—Part A: Systems And Humans, Vol. 29, No. 2, March 1999

[7] Indradeep Ghosh, Niraj K. Jha, Sujit Dey, "A Low Overhead Design for Testability and Test Generation Technique for Core-Based Systems-on-a-Chip", IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. 18, No. 11, November 1999

[8] Samah Hassan, Ayman Wahba, Ahmed Badr, "Automatic Test Pattern Generation from High Level Specifications" IEEE 2004

[9] Vishal J. Mehta, Malgorzata Marek-Sadowska Kun-Han Tsai, Janusz Rajski, "Improving the Resolution of Single-Delay-Fault Diagnosis", IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. 27, No. 5, May 2008

[10] Maria K. Michael, Spyros Tragoudas, "Function-Based Compact Test Pattern Generation for Path Delay Faults", IEEE Transactions On Very Large Scale Integration (VLSI) Systems, Vol. 13, No. 8, August 2005

[11] Miron Abramovici, "DOS and DON'TS in Computing Fault Coverage" IEEE International Test Conference 1993

[12] Amitava Majumdar, Sarma B. K. Vrudhula, "Fault Coverage and Test Length Estimation for Random Pattern Testing", IEEE Transactions On Computers, Vol. 44, No. 2**,** February 1995

[13] Weiwei Mao, Ravi K. Gulati, "Improving Gate Level Fault coverage", IEEE International Test Conference, 1996

[14] Von-Kyoung Kim, Tom Chen, Mick Tegetho, "Fault Coverage Estimation for Early Stage of VLSI Design", IEEE 1993

[15] Ananta K. Majhi, Vishwani D.Agrawal, "Tutorial: Delay Fault Models and Coverage", IEEE 1997

[16] T. Riesgo, J. Uceda, "A Fault Model for VHDL Descriptions at the Register Transfer level" IEEE 1996

[17] Emilio Gaudette, Michael Moussa, Ian G. Harris, "A Method for the Evaluation of Behavioral Fault Models", IEEE, 2003.

[18] Irith Pomeranz, Sudhakar M. Reddy, "Transition Path Delay Faults: A New Path Delay Fault Model for Small and Large Delay Defects", IEEE Transactions On Very Large Scale Integration (VLSI) Systems, Vol. 16, No. 1, January 2008

[19] Irith Pomeranz, Sudhakar M. Reddy, “Unspecified Transition Faults: A Transition Fault Model for At-Speed Fault Simulation and Test Generation”, IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. 27, No. 1, January 2008

[20] Ameet Bagwe, Rubin A. Parekhji, “Functional Testing and Fault Analysis Based Fault Coverage Enhancement Techniques for Embedded Core Based Systems”, IEEE 2000

[21] Synopsys ATPG TetraMax User Guide Version Z-2007.03, March 2007

[22] Atrenta SpyGlass DFT_DSM Rules Reference Version 4.1.1, November 2008

[23] SolvNet DFT Overview User Guide Version Z-2007.03

[24] Mentor Graphics Scan and ATPG Process Guide Software Version V8.6_4