ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THESIS PRESENTED TO
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
A MASTER'S DEGREE IN ELECTRICAL ENGINEERING
M.Eng.

BY
Roger EL-KAFROUNI

ENHANCEMENT AND VALIDATION OF A TEST TECHNIQUE FOR INTEGRATED CIRCUITS

MONTRÉAL, MAY 10, 2010

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS:

M. Claude Thibeault, Thesis Supervisor
Département de génie électrique à l'École de technologie supérieure

M. Ghyslain Gagnon, President of the Board of Examiners
Département de génie électrique à l'École de technologie supérieure

M. Christopher Fuhrman, Member of the Board of Examiners
Département de génie logiciel et des TI à l'École de technologie supérieure

THIS THESIS HAS BEEN PRESENTED AND DEFENDED

BEFORE A BOARD OF EXAMINERS AND PUBLIC

April 9, 2010

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens tout d'abord à exprimer ma reconnaissance envers mon directeur de maîtrise et professeur titulaire, Dr. Claude Thibeault, pour son appui et sa confiance envers moi durant mon projet de recherche. Sa méthode pédagogique était enrichissante et a rendu mon projet de plus en plus intéressant.

Je remercie aussi les étudiants du LACIME dont l'aide, le dynamisme et le travail remarquable ont contribué à créer une ambiance de travail coopérative et chaleureuse.

Je remercie mes parents et mes deux frères Michel et Paul, qui m'ont donné un support infini, beaucoup de passion et d'espoir pour arriver à compléter ma maîtrise.

ACKNOWLEDGEMENTS

First and foremost, I wish to express my sincere appreciation and gratitude to my supervisor, Dr. Claude Thibeault, for his guidance and encouragement during my master's work.

I would like to thank the graduate students of LACIME; they have created a great cooperative and pleasant working environment.

I would like to dedicate this work to my parents and my brothers Michel and Paul. Their support, encouragement and understanding have been monumental during the course of my education. Without them, I doubt I would have been successful in my academic work.

AMÉLIORATION ET VALIDATION D'UNE TECHNIQUE DE TEST POUR CIRCUITS INTEGRÉS

Roger EL-KAFROUNI

RÉSUMÉ

Ce mémoire s'intéresse à une approche de test récemment développée à l'ÉTS. Cette approche, appelée méthode de test de délai sans capture (*Capture-less Delay Testing*, CDT), a été proposée comme technique complémentaire aux approches plus traditionnelles de test visant à s'assurer que les circuits intégrés fonctionnent à la fréquence prévue, afin d'améliorer la couverture de test de ce type de test. CDT utilise entre autres des capteurs permettant de détecter la présence de transitions à des endroits stratégiques.

L'objectif de ce projet est d'améliorer certains aspects de cette nouvelle approche. Dans un premier temps, nous allons analyser la distribution de délai des nœuds non couverts par les méthodes traditionnelles de test, afin de développer la meilleure manière de déployer les capteurs CDT. Nous présentons l'ensemble d'outils, utilisant le langage Perl, développé à cette fin. Les résultats obtenus confirment que les chemins passant par les nœuds non couverts sont plus longs que ceux qui passent par les nœuds couverts. La différence entre les deux types de chemins représente plus de 20% de la période d'horloge si l'on considère les délais des chemins les plus courts.

Dans un deuxième temps, nous proposons un algorithme entièrement automatisé qui permet, pendant les premières étapes du processus de génération automatisé des vecteurs de test: 1) d'identifier les nœuds non couverts, 2) d'identifier les emplacements des senseurs CDT sur les entrées des bascules afin d'améliorer la couverture de test, et 3) de minimiser le nombre de senseurs selon le besoin. Nos résultats indiquent que lorsque nous appliquons CDT en complément aux méthodes transitionnelles basées sur le modèle de pannes de type transition nous pouvons augmenter la couverture de test de près de 5%. De plus, l'algorithme de minimisation du nombre de senseurs de CDT permet de réduire de plus de 85% le nombre de ces senseurs avec une perte de couverture minimale, en moyenne de 1.6%.

Mots clés: circuits intégrés analogiques, générateur algorithmique de séquence de test, méthode de test de délai sans capture, méthode de test pour circuits intégrés.

ENHANCEMENT AND VALIDATION OF A TEST TECHNIQUE FOR INTEGRATED CIRCUITS

Roger EL-KAFROUNI

ABSTRACT

This thesis focuses on a scan-based delay testing technique that was recently developed at ÉTS. This new approach, called Captureless Delay Testing (CDT), has been proposed as a technique that complements traditional methods of test, ensuring the integrated circuits will function at their proposed clock speed, further improving the test coverage of the particular type of test. Furthermore, CDT incorporates the use of sensors enabling the detection of the presence of transitions at strategic locations.

The purpose of this project is to improve on certain aspects of this novel technique. At first, we analyze the delay distribution of the non-covered nodes by traditional methods of test, in order to develop the best way possible of placement of the CDT sensors. We present, using Perl Language, the ensemble of tools developed for this purpose. The end results obtained confirm that the paths that pass through the non-covered nodes are longer than those that traverse the covered ones. The difference between the two types of paths exceeds 20% of the clock period when considering the shorter path delay values.

Secondly, we propose a fully automated algorithm that enables, at the earliest stages of the test vectors generation process: 1) the identification of the non-covered nodes, 2) the identification of the placements of the CDT sensors at the inputs of the flip-flops for further improvement of the test coverage, and 3) the minimization of the number of sensors with regards to requirements. Our results indicate that when we apply CDT on top of transition-based fault model we can improve the test coverage by 5%. Moreover, the algorithm of CDT sensors minimization allows a reduction of more than 85% the number of those sensors with a minimal test coverage loss, on average of 1.6%.

Keywords: analogue circuits, automatic test pattern generation, captureless delay testing, integrated circuit testing, low cost testing, scan-based test technique.

TABLE OF CONTENTS

LIST OF TABLES

Page

LISTE DES FIGURES

LIST OF ABREVIATIONS

ASIC	Application-specific integrated circuit
ATPG	Automatic test pattern generation
ATE	Automatic test equipment
CDT	Capture-less delay testing
CDTP	Capture-less delay testing pattern
CMP	Chemical mechanical polishing
CTVC	Current to voltage conversion block
CUT	Circuit under test
CVP	Current voltage pulse
DC	Dynamic compensation block
DFT	Design for testability
DM	Delay measurement block
DVP	Data voltage pulse
LBIST	Logic built-in self test
LFSR	Linear feedback shift register
LOC	Launch from capture
LOS	Launch from last shift
SOC	System on chip
SSS	Sensors switching at the same time
STA	Static timing analyzer
VLSI	Very Large Scale Integrated Circuit

CHAPTER 1

INTRODUCTION

1.1 Motivations

"The success of the semiconductor industry has been due in large part to its ability to continuously increase the complexity, and therefore the processing power, of integrated circuits" [Nanowerk Spotlight]. Moore's law predicts that the number of transistors in a computer chip doubles every two years, due to miniaturization of the components. However, as device and interconnect dimensions continue to scale down from sub-micron to nanometer towards thousand-pico dimensions, IC designers and test engineers have to deal with an increase in process variation and the manifestation of new defect mechanisms.

Integrated circuits fabricated using older technologies, based on larger feature size, were relatively insensitive to process variation. As the feature size has approached the 32 nm dimensions and the wafer size has grown to 450 mm (Samsung-TSMC, Intel Fabs), process variation impact on the operation of a chip has become non-deterministic. This is mainly attributed to a decrease of feature dimensions without a corresponding increase in manufacturing machine precision. As technology has been scaling down to nanometer and feature sizes shrink accordingly, photolithography became a concern. The wavelength of light used for geometry imaging is longer than the one desired for printing [Mak 2004]. For example, a 248 nm light source is used for a 130 nm to 180 nm gate length. This issue required using the light diffraction method causing the printed image to be different than the intended shape. To solve this issue, lithography engineers generate shaping rules in order to add or subtract geometries to the mask. This method is successful to a large degree, but can still create variations on the width and uniformity of the metal lines, and the shape of vias. Furthermore it might affect the poly-silicon layer that defines the gate length of a transistor. The polishing process in Chemical Mechanical Polishing (CMP) technology that is used to help planarize the metal layers or the interlayer dielectrics for successive layer deposition depends on the geometries underneath it. A dishing phenomenon occurs when there are less

dense materials underneath, thus increasing the interlayer capacitance. Due to CMP process, copper wires that are widely used nowadays to decrease wire resistance, tends to wear down much faster than the neighboring dielectrics, hence creating erosion and dishing effect that might affect the copper interconnects resistance [Mak 2004]. All these phenomena may lead to faults, including the so-called timing related failures that need to be detected, as affected ICs do not meet the frequency specifications. In other words a chip might work at a particular speed but fails at the desired clock frequency.

IC manufacturing defects can also cause faults, including the timing related ones. Defects might occur randomly during fabrication process and are related to photolithography, CMP mentioned above, and some other fabrication processes that are beyond the scope of this work. In the so-called nanometer designs, new types of manufacturing defects have been introduced with the ever increasing number of interconnects, namely timing induced delay defects [Lin 2003]. As a consequence, more attention nowadays is being given to the test of these delay defects, this kind of test being known as delay testing.

Most of the techniques for delay testing used in the industry inject transitions through patterns to the device under test on some dedicated input ports and check its response on the output. Those kinds of techniques can be categorized as slack based delay testing. Scan-based delay testing is the dominant delay testing technique applied today as it generally provides fair coverage results and that it is fully automated. However, the quality of this kind of test is often limited by the tester memory which is not large enough to store all the required test patterns [Saxena 2002]. CPU time required by the automated test pattern generation (ATPG) tools is also a limiting factor. Consequently, transition test coverage of 80% is typical in the industry [Mentor Graphics website]. Moreover, conventional ATPG tools do not use timing information, and tend to select the shortest paths to propagate transitions, leaving undetected most of the faults that lie on the longest most critical paths [Lin 2006].

A new technique, called Capture less Delay Testing (CDT), has been recently developed to increase the delay test coverage [Thibeault 2006]. With this technique, coverage is improved by special sensors. An outstanding advantage of CDT is that it does not require any additional test patterns. In this thesis, we present a robust set of tools to automate the selection of test points where CDT sensors are required. The newly introduced procedure uses CDT on top of conventional delay testing and works in harmony with current industry used ATPG tools. With this new procedure, test engineers can: 1) pin-point the left non-covered nodes by the tools during ATPG flow and automatically select the appropriate CDT sensor locations, 2) identify the potential percentage increase of test coverage with each selection of CDT sensors, and 3) optimize the number of needed sensors to achieve a reasonable test coverage increase with reduced area overhead, in a timely manner.

1.2 Thesis Outline

In Chapter 2, we review the types of delay defects that are rendering manufactured ICs with sub nanometer technologies more prone to defect and harder to spot. We further analyze the delay fault model and how it is used in conventional ATPG tools. The discussion encompasses the concept of transition delay fault model as well as shed light on the IC speed failure due in large to manufacturing defects.

In Chapter 3, we investigate the current delay testing techniques as well as unravel the shortcomings of each method and show the aspects and challenges that limit current timing insensitive ATPG tools from achieving higher test coverage.

In Chapter 4, we propose a methodology that allows the DFT engineer to better understand the timing delay distribution of transition model left undetected faults. A set of tools was implemented to allow the user to pin point those remaining non-covered nodes in any particular design, identify all those combinatorial paths and capture all the appropriate transition delay estimations in order to better analyze the switching activity of a circuit as well as the maximum achievable frequency it can run at.

In Chapter 5 we present the CDT technique and explain in details all the aspects of its implementation stage by stage as well as analyze its functionality and potential in the real world of DFT design.

In Chapter 6, we present our proposed procedure to automate CDT application. This procedure is implemented through a set of tools that enables the test engineer to achieve during the ATPG process, a proper robust placement of CDT sensors along specific non-covered paths, as well as optimize the number of needed sensors to achieve an optimal coverage in terms of area overhead and the highest possible test coverage.

In conclusion, Chapter 7 reviews the objectives of this thesis and summarizes the contributions made in the field of scan-based delay testing. Possible future work is also discussed in this chapter.

1.3 Contribution

Significant contributions of this thesis include:

- The development of an algorithm that enables the test engineers to pinpoint the remaining non-covered nodes by the conventional ATPG tools as well as placing the sensors at the appropriate end flip flops to ensure optimal test coverage.
- An optimized algorithm that minimizes the number of needed CDT sensors to achieve a rather similar final test coverage with less area overhead and higher achievable at speed tester frequency.
- An investigation of the shortcomings of current ATPG tools from both Mentor Graphics Fastscan and Synopsys Tetramax timing insensitive tools that might leave thousands of non-covered combinatorial paths along the way and lead to potential IC test escapes.

CHAPTER 2

MANUFACTURING DEFECTS AND DETECTION MECHANISM

2.1 Introduction

Manufacturing defects have a direct impact on VLSI circuit behavior and can drastically alter its functionality. Those undesired phenomena in the silicon structure of an IC range from mild to catastrophic defects. They can take different forms from missing pieces of manufacturing materials to having extra added materials at the wrong spot inside a die. The latest ICs designed with over 2 billion transistors on a die represent a serious challenge in terms of manufacturing process precision, i.e., photolithography, as well as the detection process of potential manufacturing defects [Groeneveld 2002]. Heat and voltage drop are also critical factors to be considered, but they are beyond the scope of this work.

According to [Sachdev 2007], defects range from global defects such as mask misalignments, non-uniformity of critical dimensions, shifting of dopants under etching, to more localized spot defects of the silicon layer structure caused by dust, process variations, etc. Any process error during manufacturing process might have a tremendous impact on the chip by introducing a defect. Such a defect that alters circuit behavior is rendered as a fault. Faults in turn can be classified as catastrophic, or parametric. A fault is catastrophic when the functional behavior of the IC is incorrect. "On the other hand, according to [Sachdev 2007], parametric faults are those faults for which the IC is functional but it fails to meet its specifications, e.g. timing, power budget, leakage, etc". In today's sub-micron very large scale integration (VLSI) manufacturing demands, the soft parametric faults can drastically limit the maximum frequency the IC can run at, and might develop with time into critical catastrophic faults due to fault site being more susceptible and vulnerable to excess of heat, resistance and electromigration.

2.2 IC catastrophic defects

Catastrophic defects occur during IC manufacturing process and have direct impact on the functionality of the chip. For example, these IC deformations are due in part to wafer contamination as dust particle that can break a metal line, or flakes due to fabrication machinery errors. Figure 2.1 shows some types of global and local spot defects occurring during IC manufacturing process.

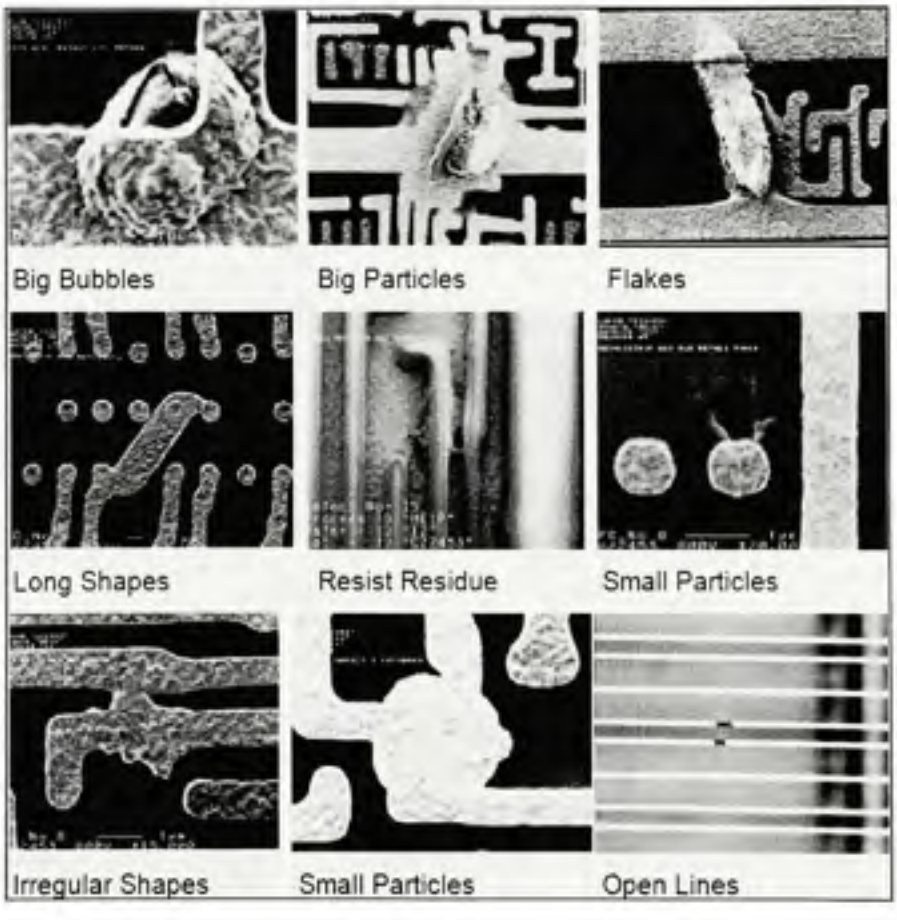

Figure 2.1 Global and local manufacturing defects.
Extracted from Sachdev (2007, p. 25)

2.2.1 IC catastrophic defects detection

These types of catastrophic defects on a chip can be detected using the traditional "stuck-at" fault model. Over the past decade, the "single-stuck-at" fault model was the most widely used in the industry on digital circuits to detect manufacturing defects. The "single-stuck-at" is a static approximation of a physical defect, in other words, it models all the failures as if all gate level pins or nets connected to the gate as they were stuck or connected or shorted to power or ground. Figures 2.2 and table 2.1 respectively illustrate an AND gate and its single stuck-at truth table. In Fig.2.3, the shaded cells represent the faults that are detected bay applying the corresponding AB combination. As an example, when AB = 11, 3 single stuck-at faults are detected: A sa0, B sa0 and Z sa0.

This type of fault model is insensitive to clock frequency the device operates on and it assumes that one fault exists at a time during test mode. This makes it applicable under any circumstance, regardless of frequency and time domain. Its simplicity allows a fast computation during Automated Test Pattern Generation and time spent on tester during diagnosis.

Figure 2.2 Logic AND gate.

Higher operational frequency, higher complexity, smaller area, and lower power consumption usually are the design objectives. Unfortunately, all of these criteria have caused ICs to become susceptible to various yield loss mechanisms which are parametric in their nature and that are not necessarily covered by single stuck-at based test patterns [Sachdev 2007].

The following section 2.3 is a brief summary of a study done by [Hawkings 2003] that sheds light on certain types of IC parametric defects.

Table 2.1 Stuck-at truth table of a 2 input AND gate

AND		A		B		Z	
AB	Z	Sa0	Sa1	Sa0	Sa1	Sa0	Sa1
00	0	0	0	0	0	0	**1**
01	0	0	**1**	0	0	0	**1**
10	0	0	0	0	**1**	0	**1**
11	1	**0**	1	**0**	1	**0**	1

2.3 IC parametric defects

Parametric failures have been there since the beginning of CMOS technology, but their significance is now more serious and growing. According to [Hawkins 2003], inaccuracies of lithography with CMOS IC nanometer technologies and increasing lack of manufacturing control of circuit parameter variance have shown that allow transistor and interconnect variations. Temperature variation across the circuit as well as power supply levels within the die, and during switching activities may result in inaccuracies that impact circuit quality, and can provoke erroneous functional behaviors and might lead to chip failure.

Interconnect properties include crosstalk errors arise from poor design rule implementation or fluctuations in metal line spacing and width. In the following a description of three different parametric failures is provided: resistive vias, metal mousebites, and metal slivers in ultrathin technologies.

2.3.1 Resistive vias

Nowadays ICs might contain billions of transistors and approximately ten times that number of metal vias. Contacts and vias at the lowest metal level are close to minimum technology feature size. With nanometer technology it is not surprising to see defective vias with elevated resistance.

Figure 2.3 Resistive vias.
Extracted from Cook (2003)

Crack in metal lines show the same characteristic of a resistive via, even though it is less common to induce failure mechanism on a chip.

2.3.2 Metal mousebites

As mentioned above, missing parts of interconnect metal are called mousebites. They can happen during IC manufacturing process due to particles defects, or electro-migration. Figures below show a defect-free and a defective (mousebite) section of interconnect.

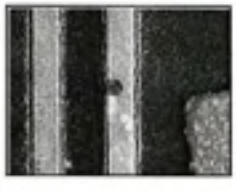

Figure 2.4 Defect-free and a defective interconnect.
Extracted from Cook (2003)

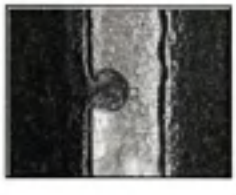

Figure 2.5 Zoom-in defective interconnect.
Extracted from Cook (2003)

Mousebites might have a minor effect on the overall delay on the metal line, but if we divide a healthy metal line to squares of 0.5 micron each, then if 90% of the middle square as shown in figure 2.8, is missing, then the new ratio becomes 0.5 µm /0.05µm.

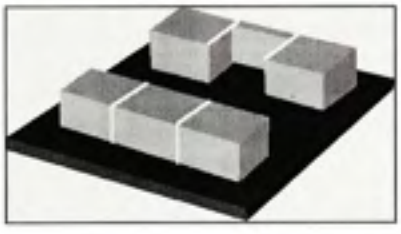

Figure 2.6 Normal Metal line and one with mousebite.
Extracted from Segura (2003)

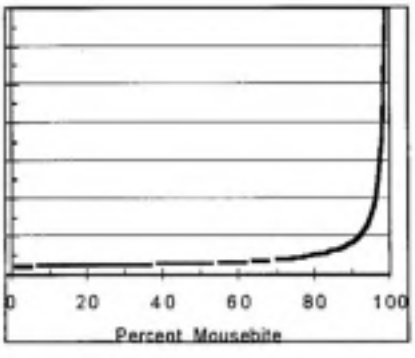

Figure 2.7 Voided metal resistance (mΩ) versus percent metal voiding using Rs = 70mΩ/sq.
Extracted from Segura (2003)

Assume that sheet resistance is R_s = 70mΩ/sq, the resistance of the square with the Mousebite defect will be equal to R□ = (70mΩ/sq)(0.5 µm /0.05µm) = 700mΩ. Therefore, the original segment of one square was 140mΩ and now it becomes 840mΩ, yielding an increase in resistance by a factor of 6.

2.3.3 Metal Slivers

Metal sliver defect is due to a metal particle that falls between two metal conductors and slightly contacts the signal line. It can be formed from any of the metal layers used in the fab. With temperature change, this metal can expand and touches or connect the two interconnect lines. This bridge resistance might be permanent and might cause noise on the two signal lines or even cause a fatal functional failure.

2.4 Parametric failures due to defects

IC parametric defects can lead to parametric failures. Here, we focus on timing failures. Any device with logic network is considered faulty if it does operate correctly at a slower clock speed but fails at the targeted or desired clock frequency.

The cause of failure in a synchronous sequential logic might be extra induced propagation delay on combinational data path reaching a storage element such as a flip-flop or a latch. Each flip-flop has a setup time and a hold time. If the signal propagating through the data path doesn't arrive or be stable before the flip-flop setup time, it is called to be violating the long path timing constraints (setup time violation). In other hands, if the signal is not stable long enough to be captured by the flip-flop it is called to be violating the short path timing constraints (hold timing violation).

According to [Kim 2003], delay defect testing is critical to insure fault free integrated circuits in the overall test strategy. A demonstration of these types of IC defects has been established by Stanford University's Murphy and ELF35 experiments (0.7-and 0.35-micron technology, respectively) on logic circuits designed using standard cells showed that 3 out of 116 defective parts were not detected when tested at lower speed than the expected functional operating speed.

2.5 Parametric timing failure due to process variation

Timing failures can also be caused by regular process variations. During IC manufacturing process, a small natural variation in physical parameters can alter the operating frequency (f_{max}) and varies in severity from one unit to another. The die location on a wafer, differences in materials and equipments can cause such a variation according to [Hawkins 2003]. It can affect the entire die or can be localized in a part or a block within the die. They might introduce a delay changing without killing the entire die by decreasing the desired frequency the device should operate on.

According to [Chandrakasan 2000], parametric variations might be due from optical effects during lithography processes, resulting in wafer images different from the original layout. It might degrade transistor parameters and might lead to catastrophic manufacturing defects occurring in the poly-silicon layer. Metal interconnect lines can also suffer from variation due to chemical-mechanical polishing (CMP). A chip might function at certain power supply voltages, but not on all its specified V_{DD} range. The die might pass at high speed with high temperature and might fail with colder temperature. It might have windows of pass and fail.

To overcome this issue, engineers should design the chip at a higher frequency targeting all longest path delays called critical path with the worst case conditions. This approach might be very expensive in terms of die size and packaging and difficult to implement and might require extraordinary engineering efforts and time.

2.6 IC delay defects detection

"From the SOC testing point of view, test solutions must address new fault models and failure mechanisms caused by manufacturing defects at the 65-*nanometer* (nm) process node and below" [Kaufman 2008]. In practice, delay faults can be a combination of direct manufacturing defects caused by lithography as resistive shorts and opens and capacitive crosstalk that can impact a local island or functional circuit path within the die, whereas

power supply noise, intra-die temperature distribution, and process variation might cause a global defects that might affect a large part or even the entire chip.

As above underlined, in the real silicon design not all faults can be simply described by the single-stuck-at model that does not include any timing effects. As discussed in the next chapter, this led to the development of delay models that are very similar to the single stuck-at one but that also take into consideration the timing relationship. The application of these models implies the use of transitions. In the test terminology, AC scan refers to using a scan chain to launch transitions through a combinatorial circuit and capture the response to those transitions within the period of the system clock. However, testing delay faults in a sequential circuit using standard scan (scan based delay testing), has its own limitations, and structural transitional fault model tests might not cover all delays defects leading to a yield escape. Chapter 3 discusses in details the pros and cons of each delay testing approach, be it functional testing, logic built-in self test, as well as scan-based delay testing and the limitations of nowadays ATPG tools.

2.7 Summary

As shown in this chapter, with nanometer technology, new types of manufacturing defects have risen to the surface, mainly due to the decrease in feature size and probable lack of manufacturing precision of lithography mask. With that in mind, IC manufacturing faults can be classified as catastrophic or parametric. The latter type of faults might occur during manufacturing process due to IC machinery fabrication flaws. These types of faults, that can also be caused by process variations, might alter the desired functional speed on a given ASIC, triggering the necessity of what is called delay testing. These delay defects are not covered by the traditional stuck-at fault model, which is timing insensitive. Delay testing techniques are a must in today ASIC manufacturing process to insure fair test coverage. In the next chapter we discuss in details these delay test techniques and their limitations, and the possibility of enhancing such a test scheme.

CHAPTER 3

EXISTING DELAY TESTING TECHNIQUES

3.1 Introduction

In this chapter we present and discuss the three main approaches used to detect delay failures, namely: Functional testing, Logical Built In Self Test (LBIST), and scan based delay testing. It shows the pros and the cons, and opens up the discussion about a new testing technique called CDT that complement the actual ones and might boost up the testing coverage to an acceptable level without the need of much engineering effort and time.

3.2 Functional testing

Functional testing consists on applying test patterns derived based upon the functionality of a chip. According to [Bareisa 2008], when used for delay testing, functional test patterns are specifically derived to detect delay failures (caused by defects or by process variations), which most likely affect the longest combinational paths on a chip, or what it is called "the critical paths". According to [Ahmed 2006], the identification of these critical paths, usually performed by using a tool called Static Timing Analyzer (STA), is part of the design flow to guarantee that the chip will work at the desired speed. To derive these functional test patterns for delay testing, test engineers must manually generate these patterns such that they exercise those critical paths (namely, sending a transition along those paths) when applied later on the tester. In order to detect a delay failure, those patterns should be applied at-speed, hence exercising all relevant combinations on the targeted functional blocks using the operation speed or the desired clock frequency. The disadvantages of such approach are the following:

Functional Pattern Development effort: According to [Thibeault 2006], developing suitable efficient functional test patterns requires long and difficult engineering work. Functional at-speed test task is very expensive and requires lots of manual engineering works targeting on some large chips thousands of critical paths rendering it extremely difficult and

somehow impossible to implement. Furthermore, importing such patterns to the tester requires also an extra effort of manual debugging, changing the timing sequences from simulation to tester environment.

High tester costs: According to [Bareisa 2008], applying functional test patterns to a tester at the desired product speed, using device primary inputs and checking the response at the device primary outputs require a high-end tester that can operate at a very high frequency along with a very high pin counts.

3.3 LBIST

LBIST is a test approach where most (if not all) test patterns are pseudo-random ones generated on chip, using a linear-feedback shift registers (LFSR), and where the response of the injected patterns are verified on-chip by a signature analyzer (or more information about LBIST structure and design flow, please refer to Annex I). Therefore BIST data exchange with the tester is minimal and drastically reduced. Test costs are generally reduced due to reduced test time, tester memory requirements, or tester investment costs, as most of the tester functions reside on-chip itself. Another positive aspect of BIST is that the test can be performed at-speed.

According to [Thibeault 2006], the main disadvantages of LBIST are the area/performance penalty and the extra design effort to deal with: 1) the propagation of the necessary at-speed scan-enable signal (discussed later, section 3.3.1, as this issue is shared by other approaches), 2) the elimination of don't care conditions and multi-cycle paths (when the circuit under test (CUT) requires two or more clock cycles to settle), and 3) the issues related to multiple clock domains and test clock skews peculiar to this test mode.

3.4 Scan-based testing

This technique is well known and a common design for testability (DFT) application through the entire semiconductor industry and has been used for decades. In this approach, automated

scan insertion tools, such as Synopsys TetraMax and Mentor DFTAdvisor, arrange part or all of the internal flip-flops of a particular device in scan chains. With this architecture in mind, test patterns generated by ATPG tools are applied to the device under test using the sequence of events depicted in figures 3.1 and 3.2.

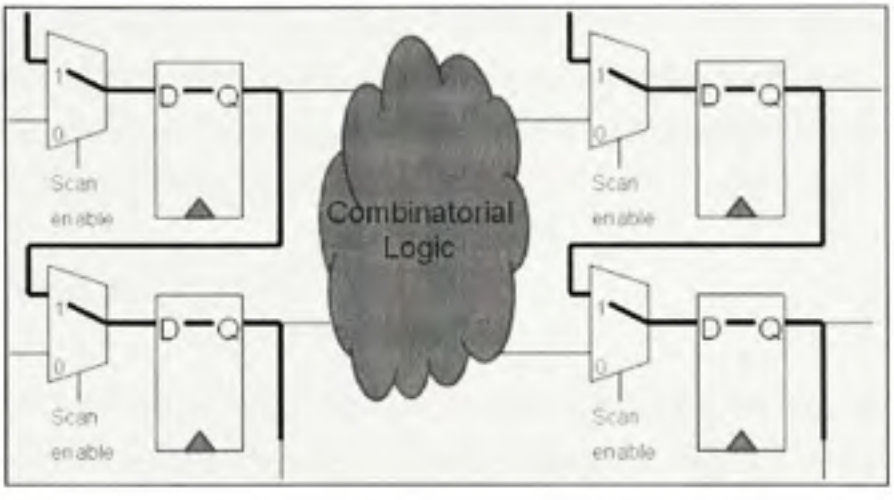

Figure 3.1 Shifting patterns in scan chains.

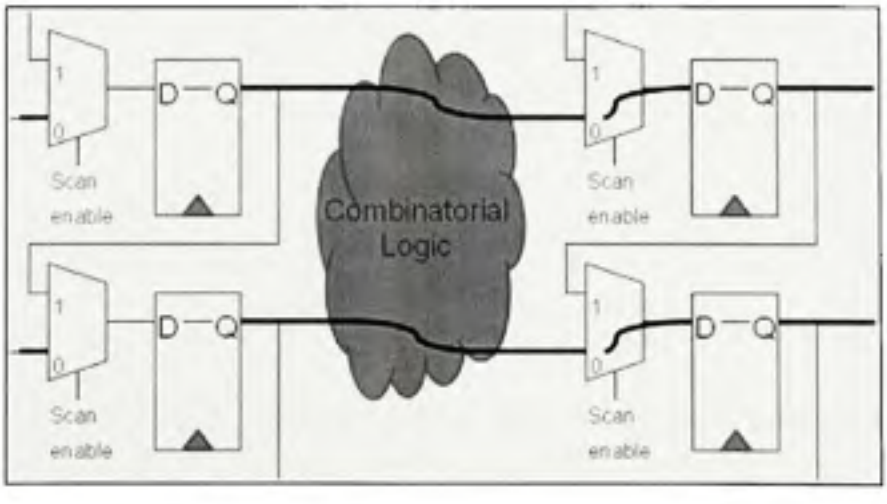

Figure 3.2 Capturing the response of the combinatorial logic.

The tester puts the chip in test mode by setting the scan enable signal to 1 on each scan converted flip-flop. It then shifts each scan pattern serially on the scan primary input. In the

second phase, tester de-assert scan enable for one or two clock cycles (depending on the selected transition launch strategy), bringing back the chip to its normal functionality, allowing the capture of data and the circuit response is stored in the device storage elements i.e. flip-flops. The third phase starts by asserting again the scan enable allowing the stored values in scan chain flip-flops to be shifted out, while shifting in a new pattern. As for the other delay testing techniques, the detection of delay defects requires that transitions are launched and propagated along combinational paths, i.e. there is no dependence between test vectors. With scan-based test techniques, there are two main transition test strategies: the launch on shift (LOS) and the launch on capture (LOC).

3.4.1 Launch on shift (LOS)

With LOS, the logic value launching the transition is initiated during the last scan shift cycle, when the scan enable signal (scan-en) is still active (figure 3.3). The fault is exercised at this period and the new logic value is captured by the first active clock edge in the capture phase, when the scan-en has been de-asserted (low).

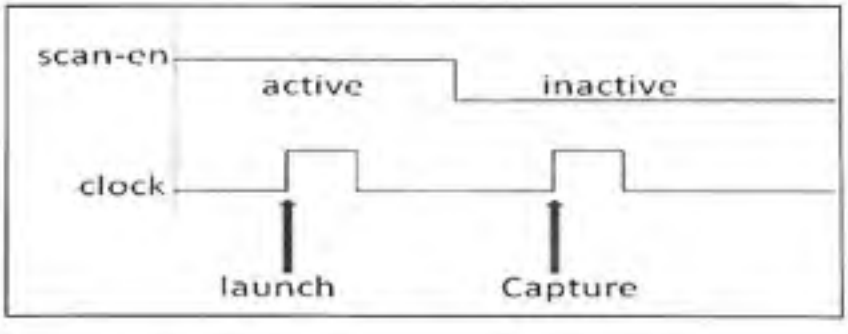

Figure 3.3 Launch on shift transition delay fault pattern generation.

LOS takes advantage of a single capture clock pulse to catch the result of the launched logic value; moreover, as the fault is sensitized during the chains load/unload, any of the available clocks can be used. For this reason, a basic scan combinational engine can be used for the test pattern generation, which brings to a much compact set of vectors in a reasonable amount of time. This is the mostly comparable technique with the single stuck- at ATPG. According

to [Benayahu 2007], the significant drawback is the scan enable signal management. In delay test, the scan enable signal must switch between the launch and capture clock; in design, it fans out to every flip-flop. Therefore, the skew effect plays a relevant role and, if the developed design is not very robust in timing, the balancing of this signal may be required with evident criticalities in routing. Additionally, if the scan enable is slower on the automatic test equipment (ATE) than predicted (different load, more delay induced on the ATE board, induced skew between Scan Enable and CLK, etc.) the device can easily fail also if fault-free. To avoid ATE induced failures, when applying LOS it is recommended to implement the pipelined scan enable technique [Synopsys DFT Compiler Manual].

Otherwise, the scan-enable signal must be routed like a clock signal. Moreover, according to [Benayahu 2007], a very accurate pin-to-pin timing between the scan-enable, scan-in, scan-out, and clock pins must be provided by the tester.

3.4.2 Launch on capture (LOC)

With LOC, both launch and capture operations occur when the scan enable signal is inactive, meaning that the chip is in its normal operation mode (figure 3.4) Therefore, the logic value launching the transition comes from the combinational paths, sampled at the regular flip-flop inputs. It exercises the target delay fault at the first active edge of the clock after the scan-enable signal is disabled, and then it captures the corresponding effect at the next clock edge.

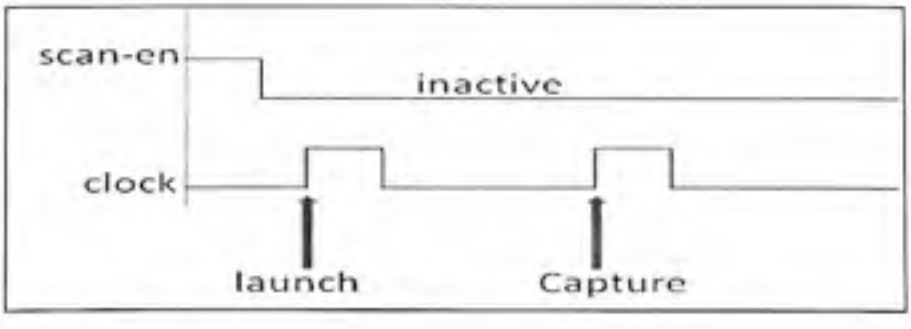

Figure 3.4 Launch on capture transition delay fault pattern generation.

The big advantage of LOC is that it relaxes the timing constraints on the scan enable signal, which becomes a regular combinational one. The disadvantage is that a multiple clock capture (sequential) procedure is requested, making the ATPG more compute-intensive and time-consuming. This brings to the generation of more vectors, which may arise potential test data volume issues. The usage of scan compression techniques is strongly recommended to reduce the impact of the pattern count. However, such techniques are often considered too costly in terms of area penalty. In spite of its disadvantages, LOC is often the preferred launch strategy. Once the transition launching strategy is selected, one must also choose the delay fault model on which the test patterns generation will be based.

3.5 Delay fault models

There are 2 main fault models that can be used in order to generate the scan-based delay test patterns: the path delay fault and the transient delay fault models.

3.5.1 Path delay model

The path-delay model is used for testing delay failures on selected paths. According to [Qiu 2004], a circuit is considered faulty if the delay of any path exceeds the specification. As it assumes that the delay fault may be distributed all over a path, this model is most suitable (and used) to detect delay failure caused by process variations. The path-delay fault model requires that the transition traverse a specific path previously defined. Since the number of paths in a real device grows exponentially with the number of nodes in the circuit, it will be impossible to target all the possible combinations of paths because this number may become really huge. Therefore, only the critical paths are addressed when the path-delay model is applied.

3.5.2 Transient fault model

The transient delay fault assumes that any delay defect is significant enough to cause a delay failure and that it is associated with (located at) the output of the gate driving the selected node. With this fault model, the single "gate" delay fault represents itself as a pin value of a gate component that works as if it has a "Slow-To-Rise" or "Slow-To-Fall" logic transition, and test patterns are created that passes the transition throughout a single gate only, no matter which path it follows. Test patterns generation in a transition fault model can use the same techniques as stuck-at-faults and can cover theoretically 100% of fault coverage. It requires minor modification for existing stuck-at-fault test patterns generation and simulation tools, and doesn't need any timing analysis. Definitely, transition fault patterns should be injected in the design under test using the highest desired frequency. It can then detect a delay fault on a particular data path, if the data arrival time to the end flip-flop is violating the setup or hold time.

Unfortunately, it must be underlined that the extra-step of applying both 0 and 1 to the identified fault renders the delay test more difficult to compute and more time-consuming. For this reason, the transition delay test coverage of large complex SOCs is typically lower covering, typically 80% of all faults. It is rather more difficult to propagate a transition along the longest paths, as it becomes more difficult to control or observe a particular fault on a given combinatorial site. Nevertheless, the transient delay fault model is the preferred model to generate scan-based delay testing test patterns.

3.6 Current DFT techniques limitations

There are some limitations to the currently used DFT techniques. Some of the limitations are discussed in the following sections.

3.6.1 Small delay defect

Unfortunately it has been shown that timing related defects often introduce a delay which size is less than the at-speed cycle time. According to [Kim 2003], this makes traditional transition fault testing less accurate for this class of faults, as transition ATPG tools attempt sensitizing a fault on the shortest (minimum slack) paths. Transition fault test vectors are therefore unable to individuate a defect which manifests itself on a long circuit path.

3.6.2 Testers limitations

It is a well known observation in the semiconductor industry, that even by using state of the art ATPG tools, several gigabits of test data may be required to exercise transition, stuck-at, and path delay faults for a multi-million gate SOC. According to [Pateras 2003], in many cases the testers used in the industry don't have enough memory to store all patterns, forcing test engineers to load and reload test patterns, or use a subsets of the test patterns at a time, hence increasing the test time and cost. Typically every second of test time might cost between 25 to 50 cents, moreover reports from high-end testers used in large Microprocessors, that amortization time for such testers is around $6000 per hour, [Hetherington 1999]. To summarize, large volume of test patterns needed to detect manufacturing defects on a particular ASIC creates a bottleneck for testers in terms of capacity and diagnosis time. As mentioned before, scan compression and LBIST are often considered too complex or costly.

3.7 Conclusion

The scan-based LOC approach is the dominant delay testing technique, at it eases the scan insertion and design and as it minimizes the area/performance overhead penalty, at the expense of extra test patterns and ATPG CPU time. In this project, our goal is to improve the test coverage of the LOC patterns along the longest critical paths, using DFT techniques, mainly a Capture-less Delay Testing technique that incorporates analog circuitry into the

early stages of the design, widening the area of coverage of traditional delay test patterns. Next in chapter 4, we discuss CDT structure and shed light on the major components that make up this novel complementary DFT technique.

CHAPTER 4

CDT (CAPTURE-LESS DELAY TESTING)

4.1 Introduction

As mentioned in Chapter 3, one of the most difficult test challenges is the ever-growing number of test vectors that need to be applied on the tester to increase the required fault coverage. This is a big challenge in terms of cost and time required for device under test. Multiple types of test patterns are required in order to get high coverage as possible for all sorts of manufacturing defects. The conventional approach is to apply all test patterns until the tester memory is full, LBIST and test compression being often seen as too complex and costly. And as it was also discussed in Chapter 3, LOC transient delay testing is the dominant way to detect faults in nanometer technologies that take into consideration timing related defects.

In this chapter, we present CDT, a recent type of scan based delay testing that requires no additional test patterns, and increase the potential of detecting such delay faults. We discuss the major components that make up this novel complementary delay test technique. It is worth mentioning that CDT was proposed before this master project started. Therefore, CDT is not a contribution of this thesis, which rather covers how to integrate CDT in a traditional design flow.

4.2 CDT Overview

CDT (Capture-less Delay Testing) [Thibeault 2006], is a scan-based delay testing technique that increases the potential of detecting imperfections that might lead to less bad chips that are tested "Good". This novel approach uses analog test circuitry in the digital world. A redundant circuitry is added on chip, mainly sensors, with a main purpose to test and measure delay affected by defects occurring during manufacturing process. The term captureless means that no logical value is captured during CDT application. As explained later, the most

outstanding CDT aspect is that it does not require any additional test patterns to be loaded in the tester. The overall area and speed penalty of redundant analog testing blocks is minimal. Finally, the CDT potential for automation is partly demonstrated later in this thesis, which should keep low the required additional design effort.

4.3 CDT functionality

CDT requires adding sensors at selected scan flip-flop data inputs and scan flip-flop clock inputs. While the device is under test during transition fault pattern shift mode, all data transitions are captured by the sensors located at the input of the modified-to-be CDT scan flip flops.

According to [Thibeault 2006], as seen in figure 4.1, the transition capture is performed by the sensors (in this example: S_{D0} and S_{D1} for the data, S_{C0} and S_{C1} for the clock) that transform the voltage transitions into current pulses collected by the parallel power rails (SVD and SGD for the data, SVC and SGC for the clock). These current pulses are then converted back to voltage pulses by the CTVC (Current-To-Voltage Converter) blocks. Finally, the DM (Delay Measurement) block estimates the delay by comparing the clock and the data voltage pulses.

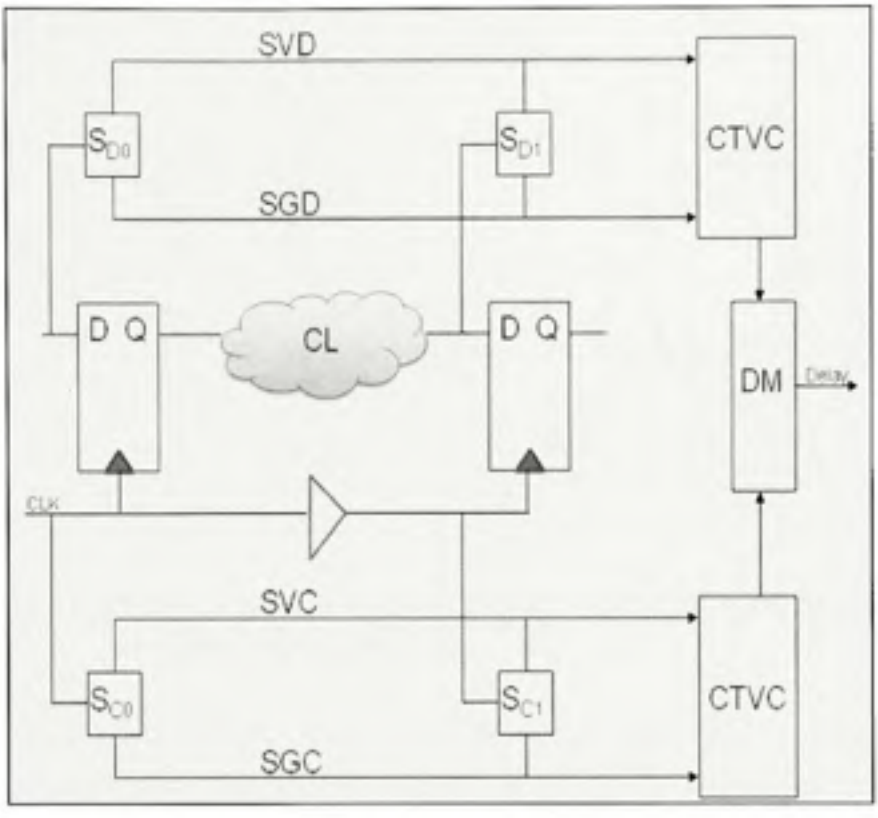

Figure 4.1 Scan based CDT architecture.

4.3.1 Implementation of CDT sensor

Each sensor consists of two small inverters twice the minimum size for the target technology and one small capacitor in series according to [Thibeault 2006]. The dual inverters sensor structure in figure 4.2 ensures a much balanced and stable voltage to current conversion. The transient behavior of the cascaded inverter pair is influenced by the intrinsic and extrinsic related parasitic capacitances. Cgd1 and Cgd2 are the gate drain capacitances due to overlap in M1 and M2. This sensor parasitic model assumes M1 and M2 are either cut-off or in saturation, which means the transistors are functioning in steady state.

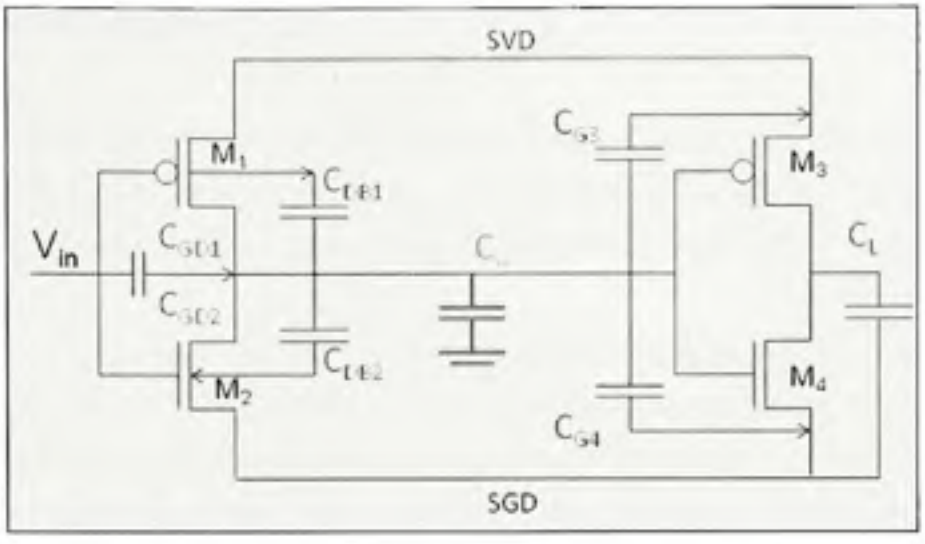

Figure 4.2 CDT sensor implementation: intrinsic (blue), extrinsic (burgundy), and load (red) capacitances.

Cdb1 and Cdb2 are the diffusion capacitances due to the reverse-biased pn-junction. Cw is the wiring capacitance that depends on the length and width of the connecting wire as it is a function of the fan-out of the gate and the distance to reach those gates. Cg3 and Cg4 are the gate capacitances of the fan-out gate that depends primarily on the width of M3 and M4 which includes both linear overlap and nonlinear gate capacitances.

The CDT sensor uses small equally sized transistors, with a small capacitance in series that matches the input capacitance of the inverter driving it. The intrinsic and extrinsic parasitic capacitances might limit the number of sensors that can be connected on one CTVC block. The cumulative parasitic capacitances may be a limiting factor when multiple sensors in chain are switching at the same time, and might require a dynamic compensation to take care for voltage attenuations on the power rail.

4.3.2 CTVC operation

When the CTVC analog block receives the collected current pulses generated by the sensors during switching activity, as transitions happen at the input of the related flip flops chain, it converts the current pulses into voltage pulses (SVD→DVP, SVC→CVP).

The CTVC block as seen figure 4.3, consists of multiple stages, starting from the dynamic compensation on the SVD and SVC current-collecting power rails (DC1, DC2), the pre-amplification stage using a low gain amplifier (LA), calibration circuitry (DA1), a current to voltage conversion stage formed with a current mirror load differential amplifier (DA2), and a final buffering stage consisting of two inverters (I) in series. The low level transistor schematics for each sub-block are further analyzed and explained in the upcoming sections.

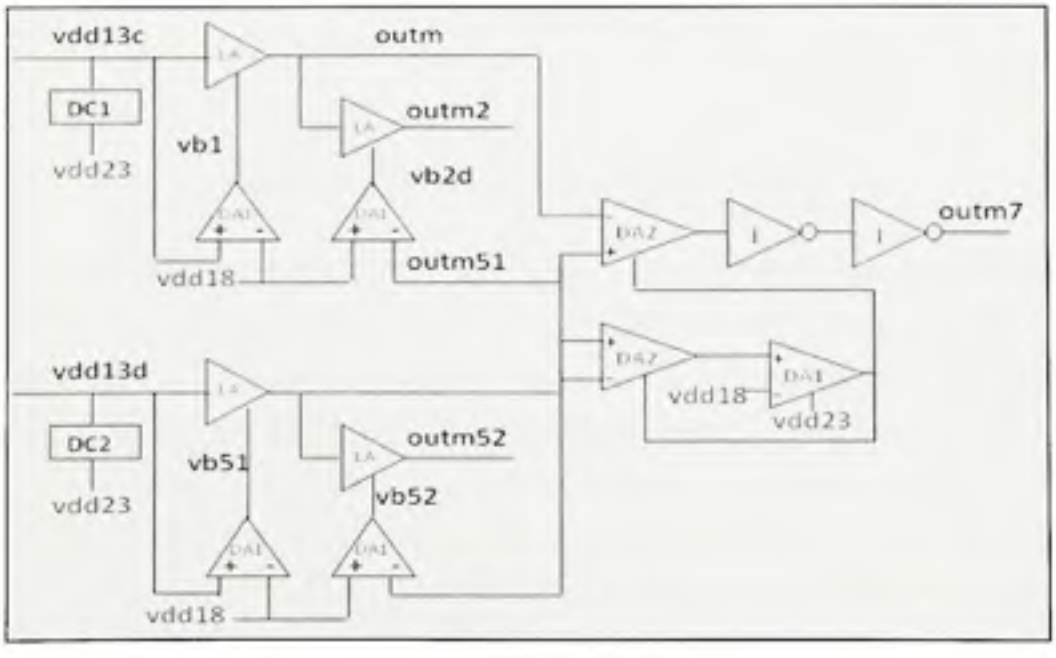

Figure 4.3 Current to voltage conversion CTVC block.

4.3.3 Dynamic compensation

The dynamic compensation block works like a bleeder that compensates for any potential attenuation on the power rails voltages. It forms an internal on the fly compensation during switching activity of the sensors. The implementation of the DC block is seen in figure 4.4.

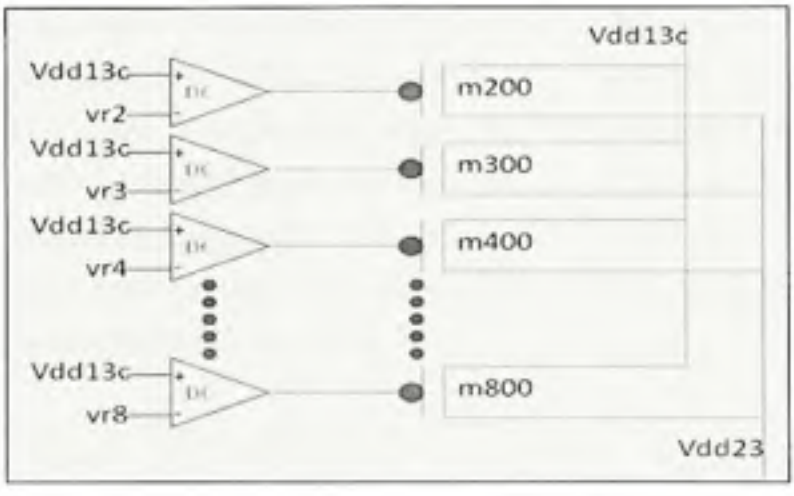

Figure 4.4 Dynamic compensation on Vdd13c.

As the voltage decrease on the vdd13c node, when transitions occur at the input of multiple sensors, the dynamic compensation structure rectifies vdd13c pulling it up to vdd23. When both differential amplifier inputs are at the same potential, the comparison between vdd13c with one of the different internally generated voltage sources (vr2-vr8) leads to logic 0 that turns the related PMOS transistor on as it enters the resistive region of operation. Hence, the current flows through the active transistor and pulls up the depleted vdd13c. This helps to reduce the impact of the number of simultaneously switching sensors (SSS) on the measured delay at the final DM stage.

4.3.4 CTVC 1st stage: Low Gain Amplification

The low gain amplifier L, as seen in figure 4.5(a), is biased by the differential amplifier DA1 which is twice the transistor sizing of DC1. One way to control the quiescent current is to sense and feedback a copy of the input current. The opted way limits the variation in the quiescent current by designing the amplifiers to have low gain. The quiescent current is controlled by the gate-source voltages on the M3-M4 NMOS transistors of LA, which in turn is controlled by the output of the differential amplifier DA1. Therefore, reducing the preamp gain reduces the variation of gate-source voltages and the quiescent current for a given variation in the offset voltages. Also a second LA amplifier is added as a load on the outm51 node, for offset calibration purposes. As M1 is connected in diode mode, it mirrors the current from M3 and flows into transistor M2 through M4. M5 is always branched always in saturation as an output PMOS resistor bleeder.

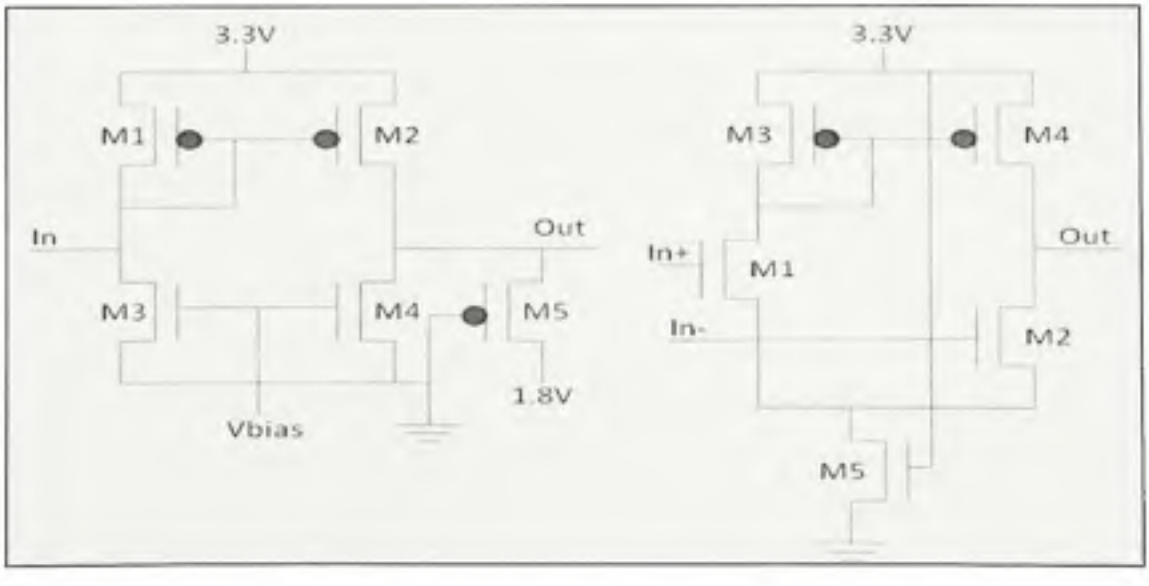

Figure 4.5 (a)Low gain amplifier, (b) and a differential amplifier.

4.3.5 CTVC 2nd stage: Differential Amplification

A current mirror load differential amplifier, as seen in figure 4.5(b), is used in the CTVC block to produce an output voltage proportional to the input current. The amplifier has a current mirror load so any imbalance in the drain currents of M1 and M2 causes the output of the differential amplifier to swing either towards Vdd23 or Vss.

When V_{in-} is larger than V_{in+}, the current in M2 is larger than the current in M1 as $V_{GS2} > V_{GS1}$. The current in M1 flows through M3 and is mirrored by M4. This causes DA2 output to go towards V_{SS} until the current in M2 equals the current in M4. The internally generated reference voltage outm51 is deliberately connected on the inverting node of DA2 in order to achieve maximum gain. Since M3 is connected in a diode configuration it has a lower resistance than M4 hence the gain from V_{in-} to the output out is larger.

The amplifier operates as a current to voltage converter due to its near zero input and output impedance. The self-biased differential amplifier DA2 receives the voltage *outm*, as previously seen in figure 4.3, on the inverting input while its non-inverting input receives the reference voltage outm51 which is always grounded.

4.3.6 CTVC Buffering Stage

As the CMOS inverter can be modeled as a dynamic equivalent output resistance r_0, there are more aspects that need to be taken care of, such as the intrinsic and extrinsic parasitic capacitances that play a major role during the switching activity of the cascaded inverters as seen in figure 4.3.

The resulting current to voltage conversion at the output of DA2 gets buffered through a cascade of two inverters in series generating the final DVP (data voltage pulse) and CVP (clock voltage pulse) output voltages to be later processed by the delay measurement DM block. DA2 and the cascaded inverters at the final stage form a high speed input buffer which

transforms the input signal that might have uneven slow-to-rise and slow-to-fall transitions into a clean digital signal with correct pulse width and level.

4.3.7 Delay measurement stage

The delay between the incoming voltage pulses from the data path is then measured against the voltage pulse received from the clock network. As seen in figure 4.6, the delay measurement is taking place between the falling edges of the CVP and DVP voltage pulses by the delay measurement (DM) block.

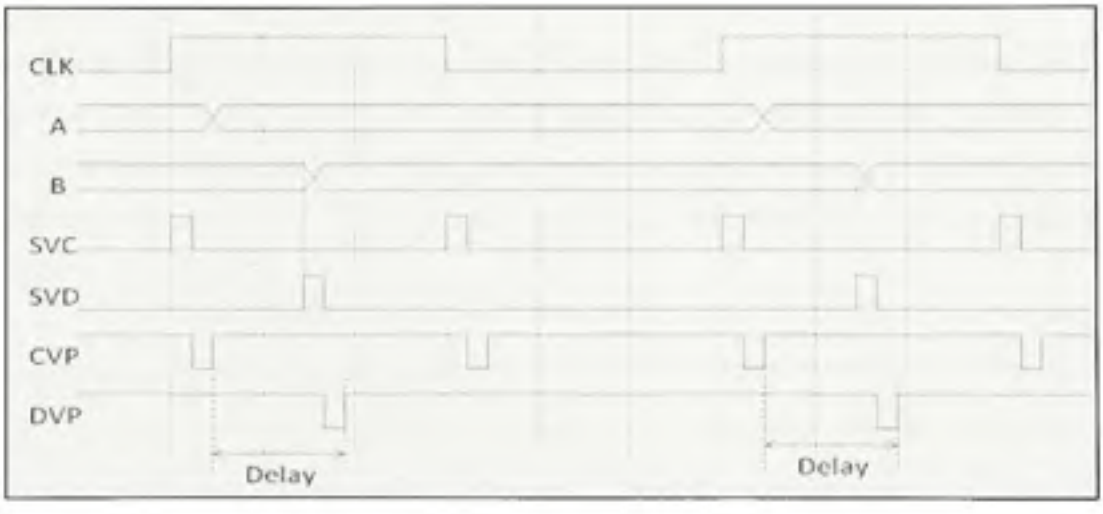

Figure 4.6 CDT Timing Diagram.

The delay measurement can be done on chip or off-chip during wafer probing. With this approach we can estimate the propagation delay of the combinatorial clouds and therefore the IC's maximum achievable clock frequency. When the CUT's frequency is exceedingly lower than specified frequency, the circuit under test is declared faulty. It is worth mentioning that the delay measurement differential approach should mostly compensate for any offset error along the path. CDT limitations are discussed in [Thibeault 2006].

4.4 Advantages of CDTP (Capture-less Delay Testing Patterns)

The majority of DFT engineers are using a combination of stuck-at and LOC transition fault model patterns to achieve an acceptable level of fault coverage. Traditional ATPG tools fail to cover those hard to control/observe faults that lie on the longest critical paths. It's much more convenient to integrate CDT in the architecture along those critical paths then spending time writing functional path delay patterns to exercise the faults. CDT might detect potential faults and allows for seamless integration with current techniques, as DFT engineer can aim for higher if not near perfect test coverage with a very reasonable amount of area/speed overhead.

As memory tester is often limiting the number of test patterns that can be applied. CDT then becomes a very efficient way to boost the delay fault coverage without requiring any additional tester memory. CDT takes advantage of the fact that no data is captured to transform the intermediate values contained in the scan chains during the shifting (in and out) into CDT test patterns.

The CDT patterns can be characterized as:

$$CDT_{Patterns} = (Sc - 1) \times Patterns_{Traditional} \quad (4.1)$$

Where Sc in the total number of scan flip flops in one scan chain.

As an example, if there are 5000 scan test patterns, 1 scan chain of 1000 scan cells; it creates around 5 million additional (CDT) test patterns. This means that the number of delay test patterns can be increased by order of magnitude.

It is important to disable all sensors during normal functional mode in order to eliminate any additional dynamic or static power consumption. The CDT architecture allows such important low power feature, as it consists of multiple separate power domains for the CTVC

and DM blocks as well as the data current pulses (SVD, SGD) and the clock related current pulses (SVC, SGC).

4.5 Conclusion

CDT is a Top-Off technique that uses the already generated transition faults model patterns. This complementary technique doesn't require any additional patterns to be generated nor stored, hence no additional tester memory load and no restrictions on the CPU computation time. CDT is an at-speed testing technique. It captures transitions at the input pin of a CDT Scan Flip Flop during LOC shift mode, and allows the test engineer to measure the delay differences between data paths and clock network, hence giving an accurate estimation of the highest frequency the circuit under test can operate on. It doesn't require any post layout information and it can be inserted during early stages of DFT / ATPG flow. Next in chapter 5, we discuss the timing based delay distribution of the left undetected faults by traditional mainstream ATPG tools, the ones we will target with CDT.

CHAPTER 5

Timing Based Delay Distributions of Transition Undetected Faults Model

5.1 Introduction

Any testing that is not aware of the delay that might occur due to process variation or any other defects in nanometer technology is not complete. A small delay on a critical path might cause a timing failure that can render the chip unusable or operate at a lower frequency. Commercial ATPG tools usually exercise transition fault test patterns on the shortest path, and are timing unaware, as transition ATPG technique attempts sensitizing a fault on the shortest (minimum slack) paths without incorporating the SDF annotation timing. Furthermore, since the ATPG tool doesn't consider the timing constraints and the actual delays of the devices and interconnects in a given design, a transition test pattern that detects a fault on a long path might fail to detect the same fault if that path was critically timed.

In the industry, according to [Davidson 2007], test coverage of 75-85% for transition fault patterns is considered acceptable when factoring in time, cost and reliability of the test. Hence, CDT implementation might bridge the gap by complementing the LOC transition fault model and further improving the test coverage with minimum time and engineering effort, at no extra tester cost. This chapter discusses this issue in details by analyzing the delay distribution of the left non-covered faults by conventional ATPG tools such as Mentor Fastscan. The ATPG process is targeting low cost testers and meant to respect reasonable industry standard test abort limit.

5.2 Scan based structural test techniques

Scan-based structural tests are widely used in the industry for their cost/coverage test effectiveness versus at-speed functional test. Transition fault models targets for slow-to-rise and slow-to-fall the output on each gate in the design. Path Delay targets the full path from the start point (output of a flip-flop) to the end point (input of a flip-flop) of the total delay

(gates and interconnects) on a specific circuit path. Detecting a delay induced defect on a chip, transition faults and path delay faults models are so far effective in producing good fault coverage. Unfortunately they have some limitations that are discussed in chapter 3.

Transition fault test are applicable on one clock domain, assuming a fixed cycle time. When a transition occurs and reaches the end point and being captured and observed, a defect might be detected if it doesn't meet the timing slack of the exercised and observed circuit path. Relatively to the clock domain, if the slack of this particular circuit path is big, the delay defect might not be detected. This defect escape might introduce a failure later in the life cycle of the chip. The quality of the chip is then reduced, and might trigger a costly recall.

A small delay defect that might occur on a short path might have subsequent aging failure on a chip, while a defect occurring on the longest path might have a catastrophic immediate effect on the correct operation of an integrated circuit. By using current commercial ATPG tools, such a defect on the longest path might be left undetected with transition fault models. In this chapter we will focus on the longest path fault detection and its impact on the overall test coverage.

5.3 ATPG methodology

In this study we determine the ATPG undetected faults and observe where they lie in the timing domain of each related path. For simplicity we used a single clock domain design, especially that transition fault model requires one clock domain in each scan chain to be exercised and observed one at a time. A delay distribution of the undetected faults paths using Launch-On-Capture (LOC) transition fault model is evaluated. In the following figure, the implementation flow is described. A set of tools using Perl scripting is implemented at each stage of the flow to extract and process data.

In this study, we target some of the Politecnico di Torino circuits belonging to the International Test Conference (ITC99) benchmark suite that were meant to be used for

experimentation on DFT and ATPG. Those benchmarks correspond to synthesizable RT-level descriptions of different size, complexity, and type [ITC 99 Benchmarks].

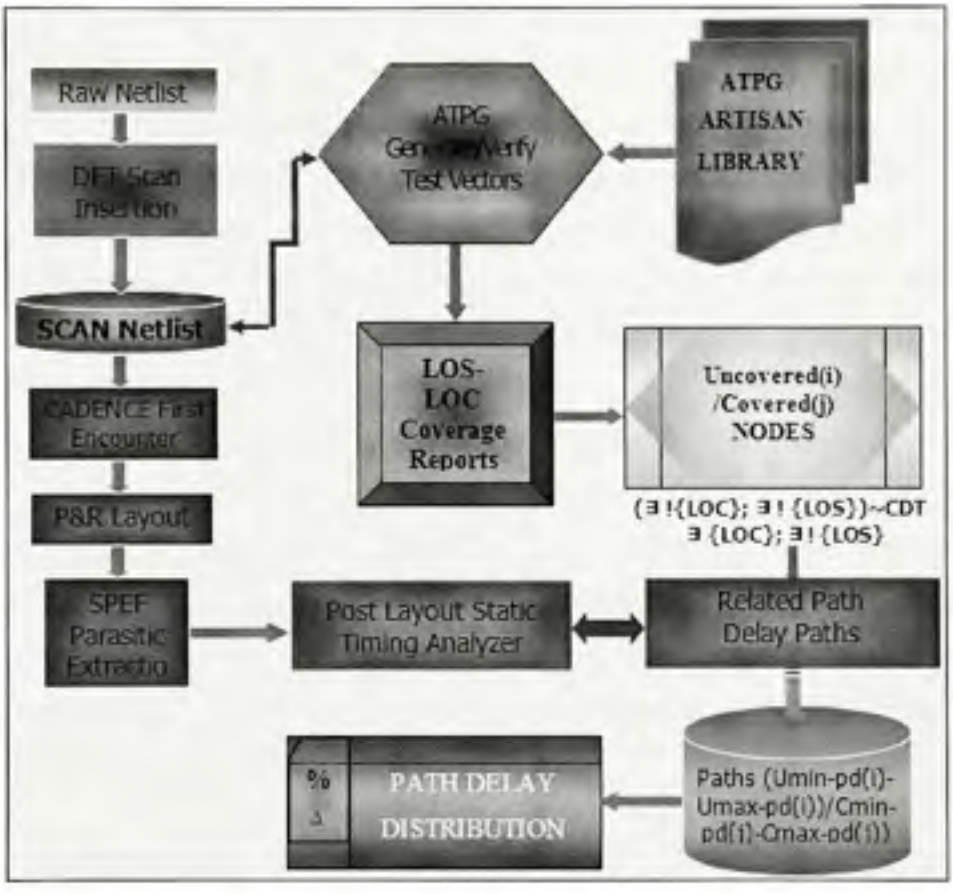

Figure 5.1 Path delay distribution extraction flow.

Perl scripts were created to run DFT /ATPG processes that are applied on the ITC99 benchmark b05 and to analyze results. It determines the min and max path delays of those nodes that are non-covered by LOC. It can further analyze and determine the final Delay distribution of any selection of covered nodes in relation to non-covered ones. This methodology allows for proper investigation of the placement of the undetected faults along the respective paths as well as it shows the occurrences of the suitable path transition delays which is crucial to understanding the switching activity of the CDT sensors that are going to be inserted at a later stage.

5.4 Simulated implementation steps

Scan chains insertions are implemented by DFTAdvisor in the design and Mentor Graphics Fastscan ATPG tools using transition fault with Launch-On-Capture and Launch-On-Shift. Fastscan is then used to report all Fan-in and Fan-out of all non-covered nodes on both LOC and LOS. A set of Perl scripting tools compare the LOC undetected faults and the LOS undetected ones to create the following lists: 1) the list of nodes non-covered by LOC but covered by LOS (those are the nodes we want to cover with CDT and from which the destination FFs, where sensors are added, are determined), 2) the list of nodes non-covered by LOS but covered by LOC, and 3) the list of nodes that are non-covered by both.

After generating the destination FF file, we use the paths links and run it in Prime-Time Static Timing Analyzer, with type 'Max" meaning we are taking into account the maximum delay. The worst case process corner is used, for slack timing calculation. Parsing through the STA report, the arrival time for each path is calculated. The minimum path delay umin-pd(i), and the maximum path delay umax-pd(i), of all the paths passing through the same node i are measured. The number of occurrences of all minimum and maximum path delays, are then computed on all undetected nodes.

As can be seen in figure 5.1, the previous procedure is repeated for all the covered nodes to calculate their path delay occurrences. For each covered node j, we respectively define cmin-pd(j) and cmax-pd(j) as the minimum and maximum path delay of all the paths passing through this node.

5.5 Simulation results

5.5.1 Minimum path delay distribution of non-covered faults

The following chart shows the delay distribution of all the shortest paths, namely all Umin-pd(i) values, along all non-covered faults. The fastest path of all shortest paths namely

min(umin-pd(i),) is taking 18.7% of the cycle period to arrive at the (to be inserted) CDT sensor. Whereas the slowest path max(umin-pd(i), ∀ i) is taking 77.1% of the clock cycle period to arrive at the to be inserted CDT sensor that lies at the input of the destination FF.

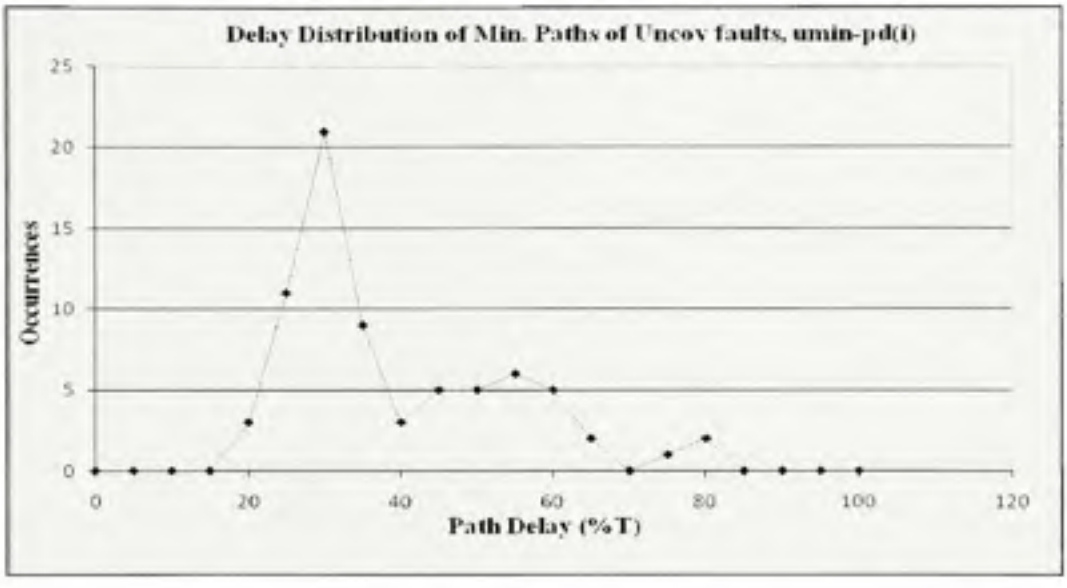

Figure 5.2 Minimum path delay distribution of non-covered faults.

The mean delay of shortest paths passing by all undetected faults, mean(umin-pd(i), ∀ i), is approximately equal to 45% of the cycle time.

5.5.2 Maximum path delay distribution of non-covered faults

The following chart shows the delay distribution of all the longest paths, namely all the umax-pd(j) values, along all non-covered faults. Among these paths, the one with the most slack (the lowest max-pd(j) value), is taking 30.3% of the cycle period to arrive at the sensor. Whereas, the one with the least slack (the highest umax-pd(j) value), is taking 94.1% of the clock cycle period to arrive to the sensor at the input of each of their destination FFs.

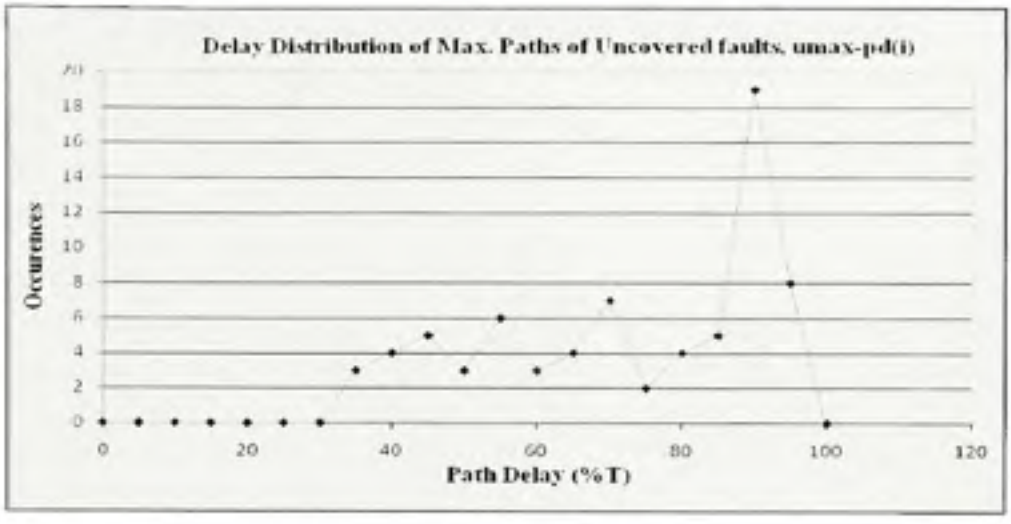

Figure 5.3 Maximum path delay distribution of non-covered faults.

The mean delay of longest paths passing by all undetected faults, mean(umax-pd(i), ∀ i), is approximately equal to 72% of the cycle time.

In order to evaluate ATPG tools fault detection scheme, and in order to verify and compare delay distribution along undetected faults; in the following we evaluate delay distribution along all ATPG detected faults.

5.5.3 Minimum path delay distribution of covered faults

The following chart shows the delay distribution of all the shortest paths along all covered faults, namely all the cmin-pd(j) values. The fastest path of all minimum delay paths along covered faults, min(cmin-pd(j)), is taking 6.1% of the cycle period to arrive at their destination flip-flops. The slowest path of all minimum paths, max(cmin-pd(j)), takes 77.1% of the clock period to arrive to its path destination FF.

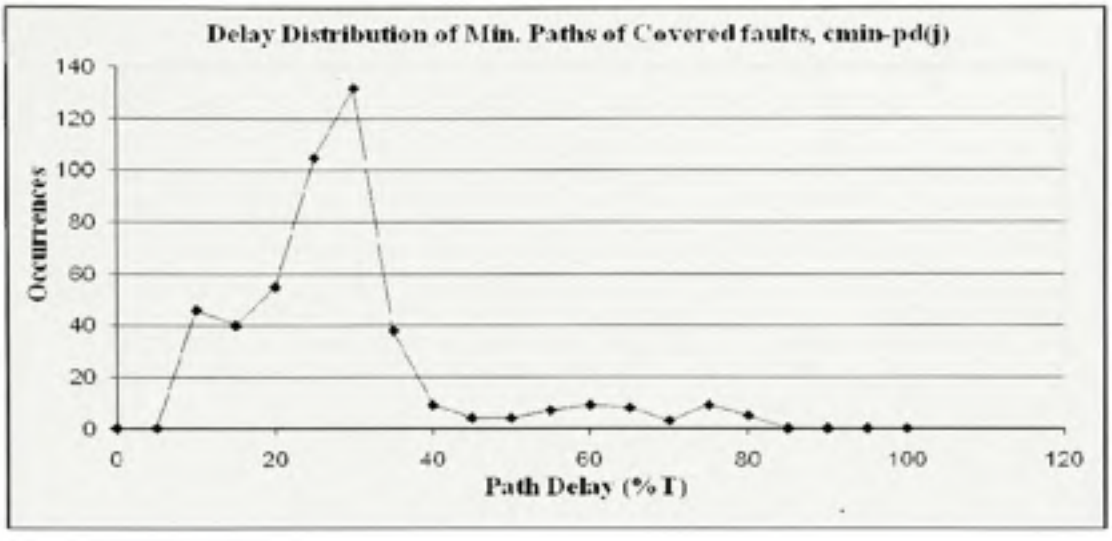

Figure 5.4 Minimum path delay distribution of covered faults.

The mean delay of shortest paths passing by all detected faults, mean(cmin-pd(j), ∀ j), is equal to the total number of detected faults is approximately equal to 24% of the cycle time.

5.5.4 Maximum path delay distribution of covered faults

The delay distribution of all longest paths along all covered faults, namely all the cmax-pd(j) values, is shown in chart below. The fastest path of all longest paths passing through covered faults, min(cmax-pd(j), ∀ j) is taking 6.1% of the cycle period to arrive at their destination flip flops. The slowest path, max(cmax-pd(j), ∀ j) takes 95.8% of the clock period to arrive to its path destination flip flops.

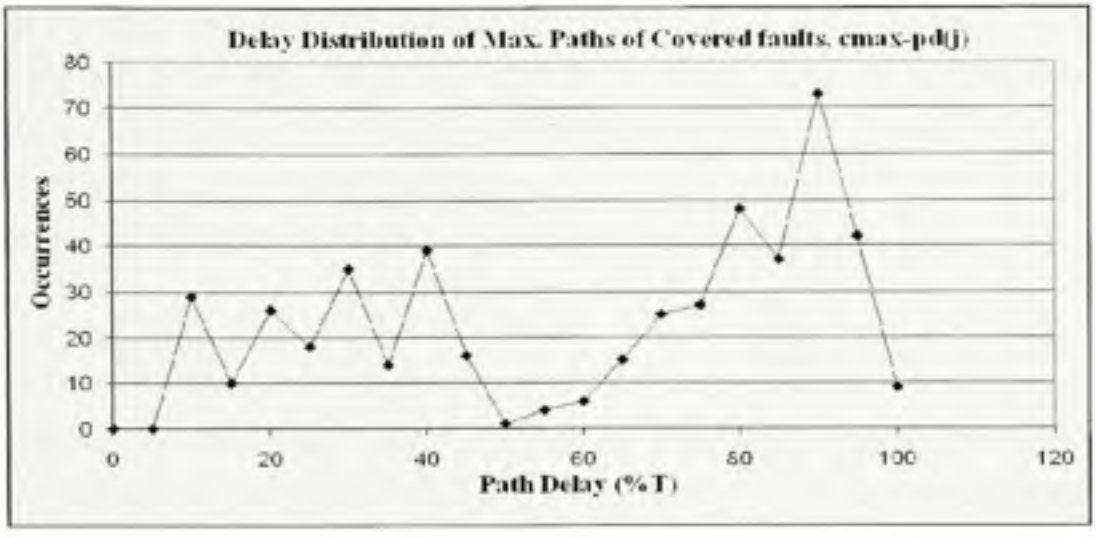

Figure 5.5 Maximum path delay distribution of covered faults.

The mean delay of longest paths passing by all detected faults, mean(cmax-pd(j), ∀ j) is equal to the total number of detected faults is approximately equal to 64% of the cycle time.

5.5.5 Comparing delay distribution of non-covered & covered faults

Table 5.1 sums up the previous results, expressed as a percentage of the clock period.

Table 5.1 Minimum, mean & maximum values of umin-pd(i), cmin-pd(j), umax-pd(i), cmax-pd(j) expressed as a percentage of the clock period (T)

Node Path Delay	Min	Mean	Max
umin-pd(i)	18.7	45	77.1
cmin-pd(j)	6.1	24	77.1
umax-pd(i)	30.3	72	94.1
cmax-pd(j)	6.1	64	95.8

The delay distribution of the non-covered faults along the shortest paths has a mean value of 45% of the cycle period, while the same distribution of the covered faults has a mean value

of 24%. Thus it means a shift of 21% along the delay axis. Furthermore, the delay distribution of the non-covered faults on the longest paths has a mean value of 72%, whereas the corresponding covered faults delay distribution have a mean value of 64% of the clock cycle, leading to a shift of 8% along the delay axis. These results validate the assumption according to which non-covered nodes are located along longer paths.

5.6 Conclusion

On average, the paths passing through the non-covered faults are longer than the ones passing through the covered faults. The difference between the two types of paths exceeds 20% of T when considering the shorter path delay values (umin-pd(i), cmin-pd(j)). As longer paths usually means more gates or longer interconnects along these paths, these non-covered paths should have a higher probability of being affected by (delay) defects. Therefore, it is important to cover as much as possible these paths.

The CDT tackles this issue, and might significantly improve the overall test coverage. In the next chapter, we will implement a complementary fully automated ATPG method that would identify the left non-covered faults, determine the related combinational paths, and allow the test engineer to place the CDT sensors along selected paths achieving higher test coverage, with minimal effort and area overhead.

CHAPTER 6

ANALYSIS OF DELAY TEST EFFECTIVENESS WITH CDT ON TOP OF LOC

6.1 Introduction

As mentioned earlier, delay testing is widely used by the ASIC industry. Two types of fault models, transition and path delay, are the most considered in this testing category. Transition fault model is typically the same as stuck-at-fault model. It doesn't require a timing aware ATPG tools and is used for large size of delay manufacturing defects. It is widely used in the industry as at-speed testing, to verify the timing structure of a circuit and to detect delay manufacturing defects. In transition fault, the single "gate" delay fault represents itself as a pin value of a gate and toggles as if it has a "Slow-To-Rise" or "Slow-To-Fall" logic transition. As an example, ATPG tools need to exercise at least 4 vectors to mimic a transition from "Slow-To-Rise" and "Slow-To-Fall" on a two input identified fault logic gate, rendering the delay test more time-consuming and difficult to compute, with a very large number of test patterns. In spite of this large number of patterns, the transition delay test coverage is typically lower (around 8 0%) than the single-stuck-at coverage (> 99%). Furthermore, as underlined before, the analysis of the defect is usually performed along a very short path, leaving some faults that lie on a critical path left undetected.

The reasoning behind accepting test coverage of 80% using transition fault model from industry nowadays is based on some statistical data that ASIC vendors have evaluated, and the lack of a better approach that is feasible in terms of time and cost. Running transition faults patterns at speed theoretically can detect higher than 90% of the delay induced defects, with stuck-at-fault on top, leading to a decent test and fault coverage. DFT engineers rely also on path delay testing that cover around 5% of the overall long paths that are left non-covered by transition faults. Even though this approach is recommended by the industry, it is not flawless and the probability of not detecting a fault is not zero. Moreover, it requires extensive engineering work, and a thorough investigation and analysis about the overall circuit timing using STA tools.

In chapter 4 we evaluated the Capture-less Delay Testing "CDT" which is a Top-off technique that complements traditional delay testing scan techniques. It doesn't require an increase on the pattern count to be stored in the tester memory, and it is suitable for low cost testers. In chapter 5 we showed that most of the ATPG undetected faults using transition fault model lie on the longest circuit paths. In this chapter we discuss the potential on increasing the overall delay test coverage by introducing CDT DFT circuitry. Multiple benchmarks are processed and the results are thoroughly discussed.

6.2 Capture-less Delay Testing CDT

As described in chapter 4, the approach consists of adding sensors at selected scan flip-flop data inputs and scan flip-flop clocks. The scan flip-flops do not capture data when CDT is applied. The capture is rather performed by the sensors that transform the voltage transitions into current pulses collected by the parallel power rails. This approach helps to accurately estimate the propagation delay of the IC combinatorial parts and therefore the maximum achievable clock frequency. When this frequency is too low, the circuit under test is declared faulty [Thibeault 2006]. This test technique doesn't impose any restrictions on scan signals, nor does it involve into generating more test patterns as it doesn't require any additional tester memory. It works in harmony with other conventional techniques such as LOC transition fault model, where it automatically generates its patterns during the shifting of test patterns into scan chains.

6.3 Experiments CDT on Top-off LOC technique

This section will deal with the basics of how Capture-less Delay testing can improve test coverage, and discuss in part the potential of applying CDT partially on a given SOC.

6.3.1 Evaluation Framework

After applying scan insertion successfully on a given benchmark, ATPG tools are used to generate transition fault patterns based on Launch-On-Capture technique. Deterministic test patterns generated by ATPG tools are used for transition fault LOC model as well as random patterns in the experiments. As shown in figure 6.1, an evaluation of the test coverage of each benchmark is performed.

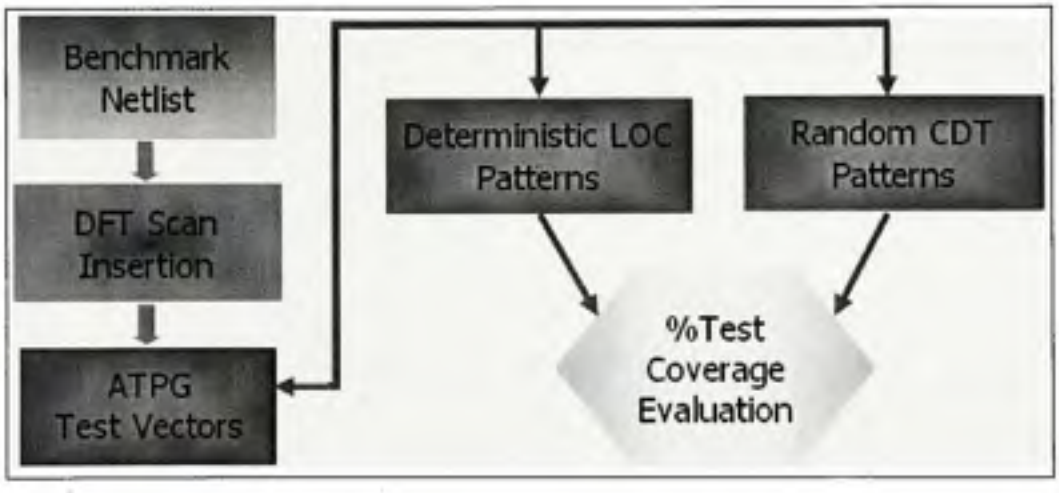

Figure 6.1 Benchmark test coverage evaluation.

Test coverage, which is a measure of test quality, consists of the percentage of all testable faults that the test pattern set tests. Typically, this is the number of most concern when the testability of design is considered.

$$TestCoverage = \frac{FaultsDetected}{Detectable Faults} \times 100$$

Here we assume that low cost testers are used, meaning that primary input values are frozen and that the primary outputs are not observed during the ATPG process. Based on this assumption, the test coverage becomes:

$$TestCoverage = \frac{\#DT + (\#PD \times PosDetCredit)}{\#Testable} \times 100 \quad (6.1)$$

Where DT are the detected faults. Please refer to Annex II for the complete description of the fault classes.

6.4 Contribution

Our work is to observe ATPG identified faults that lie in the timing domain of each related path as well as determine the related destination flip flops where the CDT sensors are going to be placed in order to achieve the extended test coverage. Furthermore, the goal is to find the optimal list of destination flip flops that ensures the highest possible test coverage with minimal area overhead and engineering effort. Figure 6.2 shows the implemented algorithm flow.

6.4.1 Algorithm general steps

1) Create the list of non-covered nodes.
2) For each non-covered node, determine its destination FFs, and create a structure that allows to pinpoint from each node all the relating destination FFs and from each FF all the non-covered nodes that can reach to it.
3) Create the list of identified FF as node destinations. This is the unoptimized list of FFs representing the entries at which we are going to insert CDT sensors in order to maximize the fault coverage.
4) Count the total number of non-covered nodes for each related destination FF prior to starting the sensor minimization process.
5) Create a list of the identified destination FFs classified in a descending order with respect to the total number of related non-covered nodes.
6) Choose the FF with the highest number of related non-covered nodes and include it in the optimized list of destination FFs where the CDT sensors are going to be inserted to maximize the test coverage with minimal area overhead.

7) Remove the chosen FF out of the optimal list of FF destinations and update the list of non-covered nodes by eliminating all the nodes that are related to the chosen FF.
8) Repeat steps 4 to 7 until there are no more nodes left in the pool to cover.

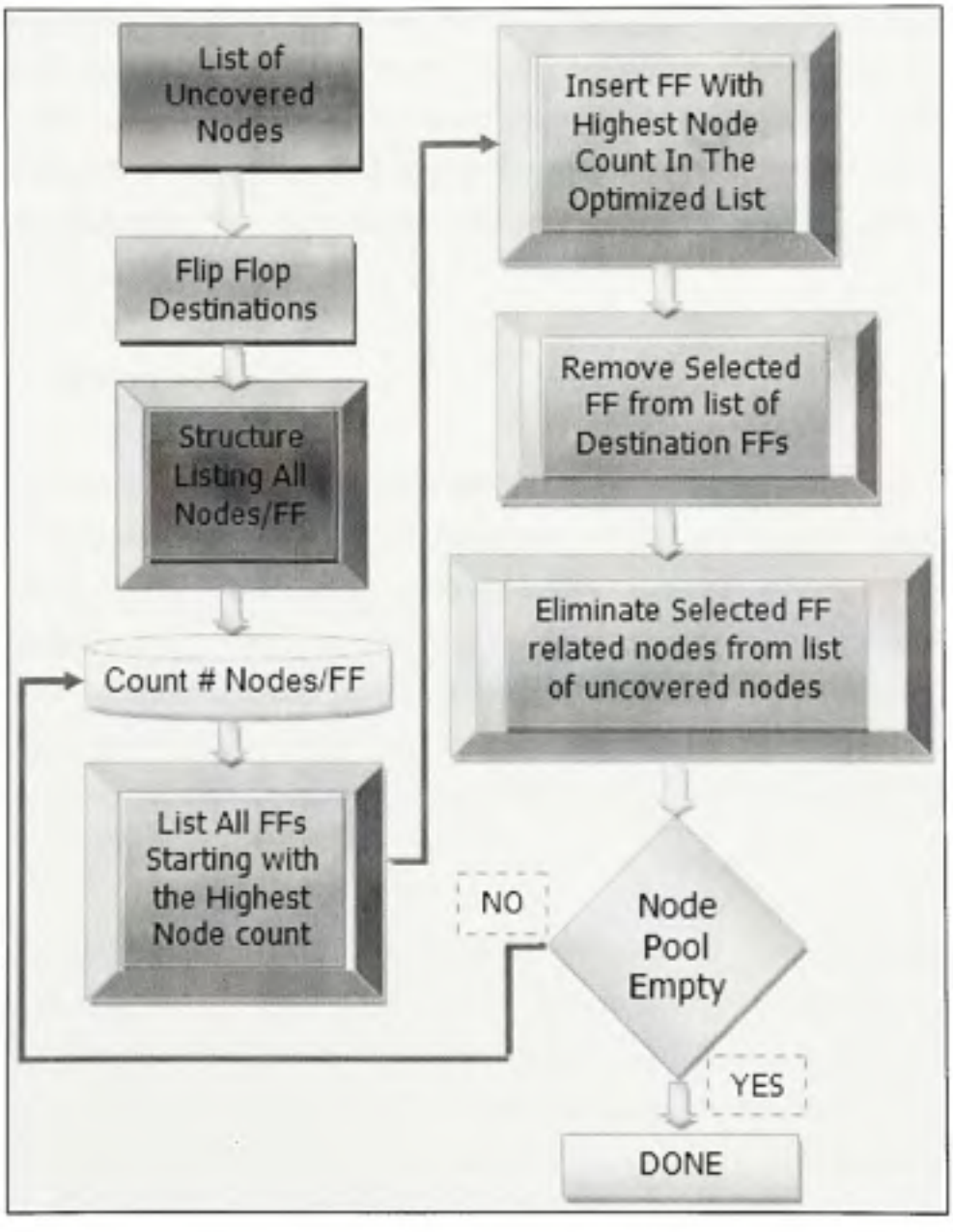

Figure 6.2 CDT sensor allocation, placement and optimization flow.

6.5 Algorithm implementation

Perl scripts were created to run DFT /ATPG processes that are applied on the ITC99 benchmarks b14, b15, and b18 (2 copies of b14 and b17), and several other Perl scripts are used to implement the CDT sensor placement and optimization algorithm. Those developed tools can be used during the ATPG process to analyze the list of non-covered nodes and determine the final optimized list of destination FFs where the CDT sensors are going to be placed in order to generate the optimal increase in test coverage with minimal area overhead and engineering effort. The major implementation steps are presented in Figure 6.3, and discussed next.

6.5.1 Implementation steps

Scan chains insertions are implemented by DFTAdvisor in the design and Mentor Graphics Fastscan ATPG tools using transition fault with Launch-On-Capture (step 1). Fastscan is then used to report all Fan-in and Fan-out of all non-covered nodes using transition fault LOC patterns (step 2). A Perl script uses the gate report file to generate a structure that shows all the non-covered nodes and their related destination FFs. For each non-covered node, the script determines its destination FFs, and creates a structure that allows pinpointing from each node all related destination FFs and from each FF all related non-covered nodes (step 3).

Another script is used to filter out repetitive lines in the generated structure that shows each non-covered node and its related destination FFs, prior to creating the new structure where we show each destination FF with its related non-covered nodes (step 4). A Perl script is then used to create the structure that shows for each destination ff all related non-covered nodes (step 5). Next script represents the first step in the FF list optimization iteration; it generates the report of node count for each destination FF (step 6). Another script processes the selection list of the FF with the highest node count and purges only the nodes in a separate file for later use to eliminate any same instances of non-covered nodes from the big list,

further reducing the number of needed destination FFs (step7). Another script is created to scan through the Nodes/FF structure and removes all FF instances with nodes similar to that of the FF destination with the highest node count (step 8). Next step requires repeating steps 6-8 until no non-covered nodes are left in the pool (step 9). This algorithm gives us two lists, one that determines the destination FFs representing where we are to put the CDT sensors that is needed for optimal fault coverage, as well as the optimized list of destination FFs that gives the highest possible test coverage with minimal area overhead.

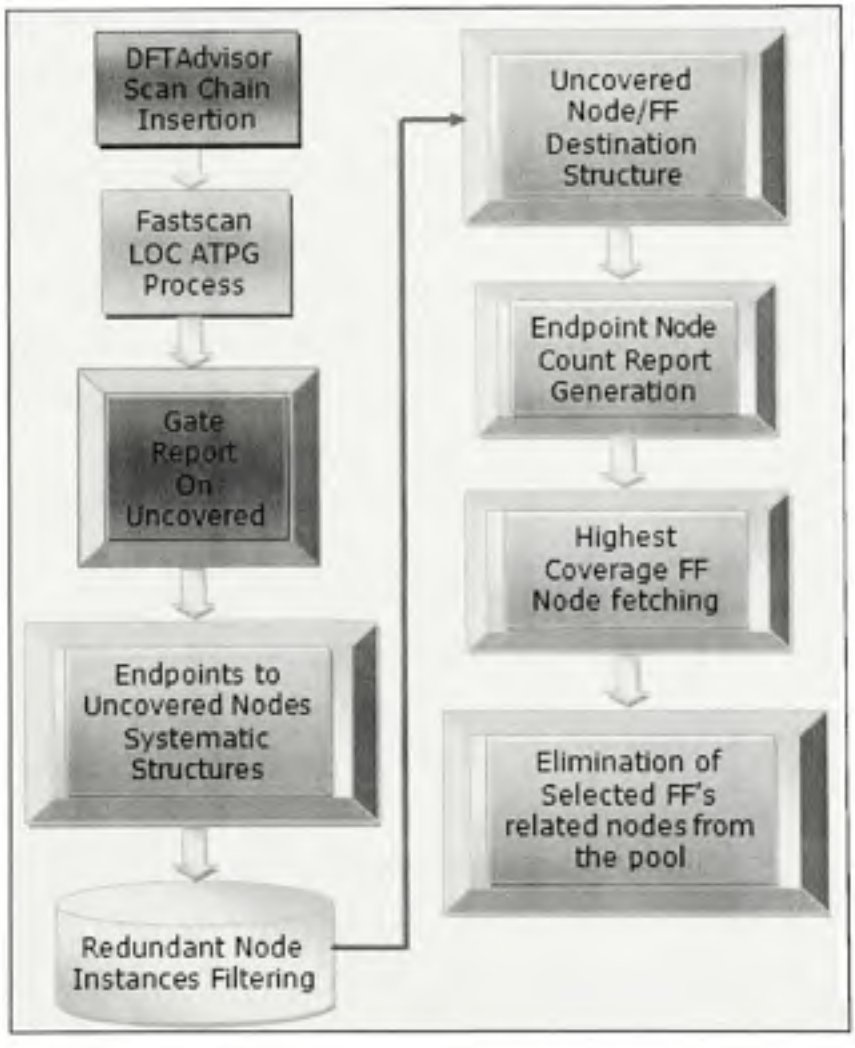

Figure 6.3 CDT sensor allocation, placement and optimization implementation steps.

6.6 Running CDT on top of LOC patterns on multiple ITC99 benchmarks

The ATPG process consists of three steps. First, we run LOC patterns on the selected benchmark and determine the test coverage. Second step consists of running the random patterns that emulates the CDTP "CDT patterns" on the full list of destination FFs and determine the test coverage increase of CDT over that of LOC. Third step we only target the optimized selection of flip flop destinations using them as observe endpoints and masking all the rest, then we run CDT on top LOC patterns and determine the final test coverage. The test engineer can easily repeat the process on a new optimized selection of endpoints using the set of implemented scripts as needed until achieving more pleasing results.

6.6.1 Proposed complementary ATPG process

The first step we launch the transition fault LOC patterns on each benchmark. One scan chain is applied and LOC is used to generate deterministic test patterns. Table 6.1 summarizes the LOC fault coverage statistics.

Table 6.1 LOC test coverage results of ITC benchmarks

ITC Benches	Total # of Faults	LOC
B14	26236	91.63 %
B15	31704	78.97 %
B18	276838	77.89 %

6.6.2 Applying CDT random patterns on Un-detected faults

In an effort to emulate the CDT patterns that are automatically triggered during shifting of the LOC patterns into the scan chain, we chose to use Random patterns and run them on the left non-covered UD undetected faults. Because we are targeting low cost testers, CDT will be running in real testing environment using the same pattern of LOC during LOC shift time.

Therefore, if the number of simulated LOC patterns is equal x, the CDT sensors will be triggered y-1 times (where y is number of flops in the scan chain) on each pattern input data exercised at the scan input of the chip, leading us to the formula (2) shown below. To emulate these patterns, we will be running the Random ATPG, on top-off LOC, and we will be using a set of random patterns of size equal to x(y-1). Thus, the number of simulated random patterns can be characterized by the following Formula:

$$SimulatedPatterns_{Random} = (\#FFs - 1) \times SimulatedPatterns_{LOC} \quad (6.2)$$

6.6.3 Un-optimized CDT sensors count coverage

In here we assume that sensors are applied on all endpoint destination flip flops inputs. As predicted, the application of random patterns minimized the number of un-detected faults and increased the overall test coverage. Table 6.2 shows the selected benchmarks test coverage results that might mirror the optimal test coverage of CDT on top of LOC ATPG process.

Table 6.2 Un-optimized SS- random patterns test coverage results

ITC99 Test Benches	CDT on Top of LOC Coverage
B14	94.71 %
B15	83.22 %
B18	85.37 %

6.6.4 Optimized CDT sensors count coverage

We want to minimize the number of needed sensors by which we should be able to cover most of the undetected faults and propagate the transition along their paths. Thus, we should be able to minimize area overhead by reducing sensor count and obtain reasonably optimal fault coverage. The algorithm was applied using Perl scripts. It forms a set of tools that

determines from the left non-covered faults of a particular design, all the destination FFs. We assume that the reduced optimized list of sensors will almost give us an equal coverage, further minimizing undetected node counts with minimal effect on the test coverage.

From here arise the two following scenarios: the first ones, described in the previous section "Un-optimized Sensors count", with its results derived from the un-optimized list of destination FFs. The second scenario consists of analyzing the list of optimized sensor count. This list results from running all the scripts of the algorithm and hence determining the minimal number of destination FFs that have the highest undetected node counts. The following table 6.3 shows the fault coverage results that might reflect the optimized sensor test coverage of CDT.

Table 6.3 Optimized SS- random patterns fault coverage results

ITC99 Test Benches	**Optimized CDT Sensor List Coverage**
B14	94.03 %
B15	82.37 %
B18	82.20 %
B18(n-detect)	80.20 %

6.6.5 Summary of obtained test coverage results on selected ITC 99 benchmarks

We exploit the benchmarks b14, b15 and b18 in order to analyze the effectiveness of the proposed ATPG process. The following table summarizes the simulation results for the rest of the benchmarks. The results gathered on the benchmarks show that our proposed ATPG process that incorporates CDT on top of LOC vectors was able to generate higher test coverage with results comparable and exceeding those generated by path delay testing.

Table 6.4 Summary of simulated ITC 99 benchmark test coverage

ITC Benches	Detected Faults	Testable Faults	CDT Sensors	Optimized CDT Sensors	LOC Test Coverage	Optimal CDT Coverage	Optimized Sensor Coverage
B14	23788	25961	187	22	91.63 %	94.71 %	94.03 %
B15	24727	31311	398	62	78.97 %	83.22 %	82.37 %
B18	211962	257861	2785	330	77.89 %	85.37 %	82.20 %

Thus as seen in Table 6.4, our proposed complementary ATPG process shows the large benefits of incorporating CDT in any given design with greater improvements that might broaden the test coverage. The proposed algorithm identifies those endpoints for sensory placement and serves the test engineer early on to incorporate the CDT structure in the design netlist in the most optimal way possible. What sets our proposed approach apart from the completion is that it will make it feasible for designers to evaluate the testability of their circuits that incorporate CDT prior to layout while tremendously reducing the test cost and engineering effort.

From the acquired results, we can assume higher test coverage by using the optimized list of sensors, with a substantial decrease in the total number of needed CDT sensors. In the following table 6.5, we summarize the improvements that were achieved as we can deduce that a test engineer could on average minimize the number of CDT sensors by ~87% and still achieve similar test coverage results with a minimal loss of ~1.57%.

Table 6.5 A compromise of Test coverage with minimal CDT sensors use

ITC99 Benchmarks	CDT Sensor Minimization	Overall Test Coverage Loss
B14	88.24%	0.68%
B15	84.40%	0.85%
B18	88.20%	3.17%

6.6.6 Validating the obtained test coverage with the optimized list of sensors

We choose a different selection of equal number of destination ffs, mainly those flops with highest node count that result from the first iteration of the proposed algorithm. We run the simulation to check the test coverage of CDT on the optimized list of sensor endpoints targeting benchmark B18. This method emulates the N-detect algorithm ensuring that each targeted node is covered N-times. The following table 6.5 shows the resulting fault coverage of the new selection of CDT sensors. If we analyze the ATPG untestable faults, the test generator was able to find more patterns to create a test, and yet cannot prove the rest of the faults redundant. These faults might be caused by the constraints, or limitations, purposely placed on the ATPG tool such as scan cell constraints that masks the excluded scan flops during simulation. Thus, from the remaining detected unobserved faults, we got a number of ATPG untestable faults that might become possibly detected or detected if we remove some constraints or in our case trying a slightly modified selection of sensors could further enhance the overall test coverage.

6.7 Conclusion

The proposed algorithm has tremendously reduced the number of needed CDT sensors. The results showed that using CDT on top of LOC can produce higher test coverage results. One can always enhance the selection of sensory endpoints, doing a couple of minor trial and error steps to further optimize the final selection. Further improvements on the sensor

minimization algorithm might be needed to achieve higher test coverage, close to the ideal unrestrained CDT coverage. Some longer combinatorial paths are harder to propagate a transition along to an observable node. Some circuits might have paths with multiple reconvergent fan-outs, others have multi-cycle paths and can be hard to observe on a given endpoint further reducing the effective coverage or observability of certain selected endpoints. Therefore some endpoints that share coverage of an equally high number of targeted nodes might have a better probability of achieving higher test coverage. CDT as shown, not only improves the overall test coverage by ~ 5% but also eliminates the over testing issue since there is no need to run more LOC transition vectors to improve the coverage, or manually generate time consuming functional patterns on top of transition patterns targeting millions of critical paths in today's multi-billion transistor SOCs.

CONCLUSION

This thesis has attempted to address the need for Capture-less Delay Testing technique (CDT) as a complementary technique on top of LOC transition fault model. In this project, our goal was to improve the test coverage of the LOC patterns along the longest critical paths, using DFT techniques, mainly CDT that incorporates analog circuitry into the early stages of the design, widening the area of coverage of traditional delay test patterns.

It further emphasized on the area of test coverage that current conventional techniques could lack to resolve, and fail to ensure a feasible test coverage especially in sub-nanometer high density modern SOCs that are potentially defect-prone. We propose an algorithm that implements a fully automatable process and allows a test engineer to locate and insert CDT sensors. These sensors are inserted on specific flip flop endpoints that can be triggered by the LOC vector's transitions through the left non-covered nodes and serve to estimate, at the final stage, the total combinatorial propagation delay of an IC determining whether it meets or violates the maximum operating clock frequency.

As discussed in this thesis, with nanometer technology, new types of manufacturing defects has emerged to the surface mainly due to the decrease in feature size and probable critical lack of lithography mask manufacturing precision. The high tech industry is running nowadays into unanticipated problems with clock speeds due to on-chip variations with the evolving huge die size, high current leakage and elevated power consumption. New types of faults are emerging that can be caused by process variations, might alter the desired functional speed on a given ASIC, triggering the necessity of what is called delay testing. These delay defects are not covered by the traditional stuck-at fault model, which is timing insensitive. Delay testing techniques are a must in today ASIC manufacturing process to insure fair test coverage that protects the end user from potentially defective and low performance SOCs.

The scan-based LOC approach is the dominant delay testing technique, as it eases the scan insertion and design and as it minimizes the area/performance overhead penalty, at the expense of extra test patterns and ATPG CPU time. CDT is an at-speed testing technique. It captures transitions at the input pin of a CDT Scan Flip Flop during LOC shift mode, and allows the test engineer to measure the delay differences between data paths and clock network, hence giving an accurate estimation of the highest frequency the circuit under test can operate on. It doesn't require any post layout information and it can be inserted during early stages of DFT / ATPG flow.

In this project, we conducted a study that seeks to understand the delay distribution of ATPG undetected faults and observe where they lie in the timing domain of each related path. We chose and followed a specific ATPG process as previously discussed in chapter 5. For simplicity we used a single clock domain design, especially that transition fault model requires one clock domain in each scan chain to be exercised and observed one at a time. A delay distribution of the undetected faults paths using Launch-On-Capture (LOC) transition fault model is evaluated. A set of tools using Perl scripting was implemented at each stage of the flow to extract and process data as we proceed from RTL synthesized netlist to determining the path delay distribution of all nodes that are covered by LOS but non-covered by LOC. The purpose of our non-covered node selection is to ensure that we are only choosing the functional paths and are capable of propagating a transition along any given path.

We found through simulation results that on average, the paths passing through the non-covered faults are longer than the ones passing through the covered faults. The difference between the two types of paths exceeds 20% of clock period when considering the shorter path delay values.

We have further implemented a fully automatable algorithm that allows the test engineer during the early stages of the ATPG process to seek those non-covered nodes, place CDT sensors on related flip flop endpoints further increasing the area of test coverage, as well as

the ability to minimize on need the sensors count. Our results have showed that when we apply CDT on top LOC transition patterns, we can achieve higher test coverage results with an estimated 5% increase. Our CDT sensor optimization algorithm also produces equally high test coverage with minimal loss on average of 1.57% with an outstanding reduction of sensor count by around 87%.

ANNEX I

LOGIC BUILT-IN SELF TEST BIST

BIST architecture

A well known logic BIST architecture using a single clock based on STUMPS technique can be shown in figure A-I-1. The core logic, which is the combinational and sequential main functional logic of the chip, is the circuit under test. It is surrounded by the BIST components that include test pattern generation block using Linear Feedback Shift Register LFSR, a phase shifter circuit, the output response analysis block composed of multiple input signature register MISR, and a space compactor. Two counters are used; one is the test pattern counter and the shift counter or bit counter that keeps track of how many cycles are needed to fill in the scan chains. A test control block controls all test points in the design, and the BIST control block that depicts the BIST diagnosis steps. In some cases a normal ATPG mode might be used on the same chip requires a multiplexing scheme between the BIST phase shifter and scan inputs to convert some or all BIST controlled scan chains in deeper ATPG mode scan chains.

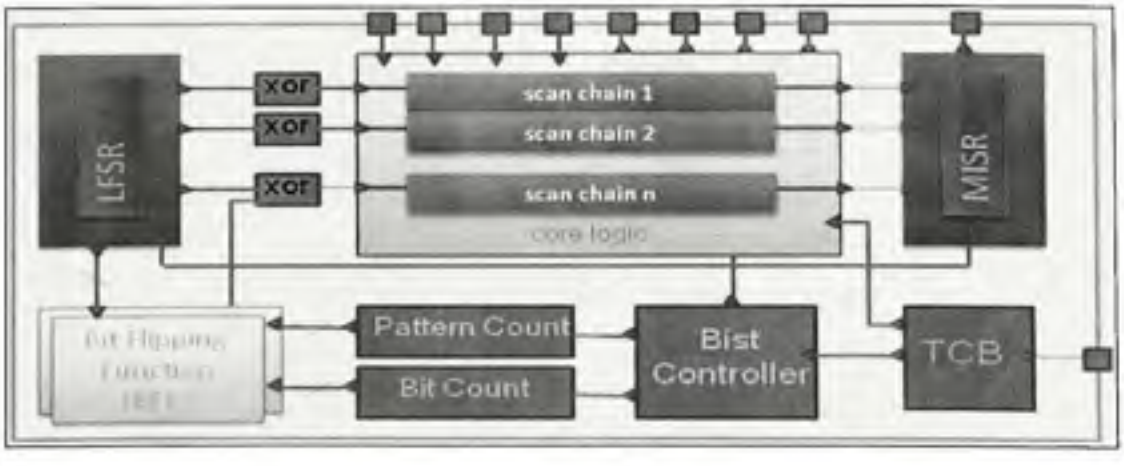

Figure-A I-1 Built-in self test architecture.

Logic BIST architecture

The BIST doesn't require any primary inputs or outputs to the external world, unless it is used as stand-alone mode logic BIST. Through Boundary scan TAP controller with few test control pattern the BIST can be initiated. In order to minimize the time of the device under test, the core logic can be divided into many shallow scan chains. During test shift time, new test patterns generated by LFSR are loaded into the scan chains while simultaneously unloading and checking the previous patterns in the MISR block. Whenever all patterns are fully loaded into the scan chains, all scan flops and test points are put in normal system mode for one at speed clock cycle allowing capturing the circuit's under test response. As shown in figure A-I-2, once all test patterns are applied, the output of MISR signature that depicts a BIST failure or success can be captured and scanned out using TAP controller output.

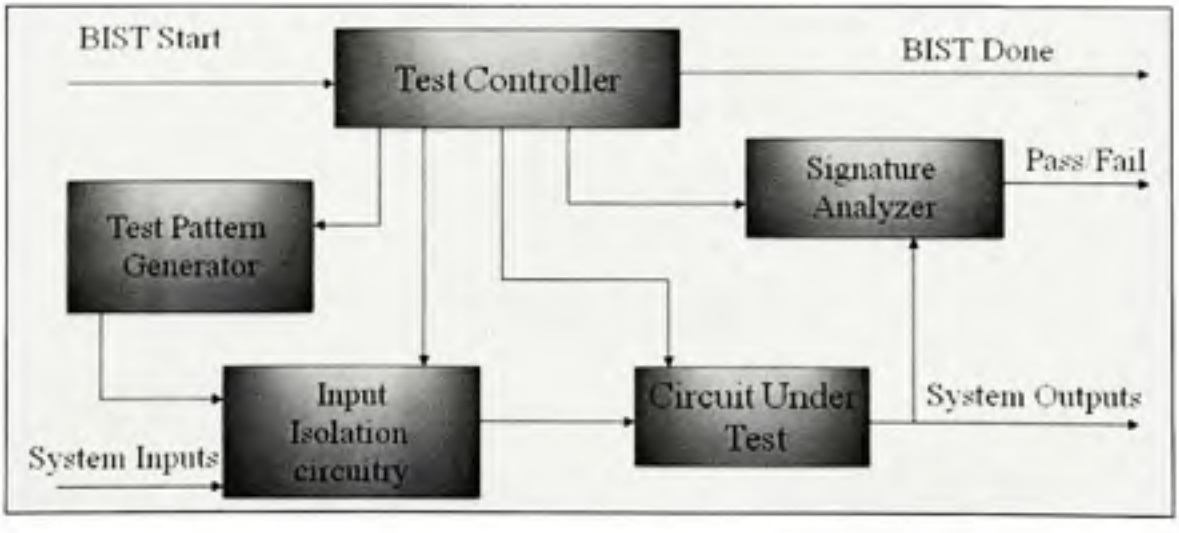

Figure-A I-2 Logic BIST functional architecture.

Logic BIST design flow

BIST timing should be analyzed using Static Timing Analyzers tools, and treating it the same as any functional logic within the ASIC is a must, in the whole ASIC flow from synthesis, formal verification (equivalence checking), timing, layout, place and route and back annotation as seen in figure A-I-3.

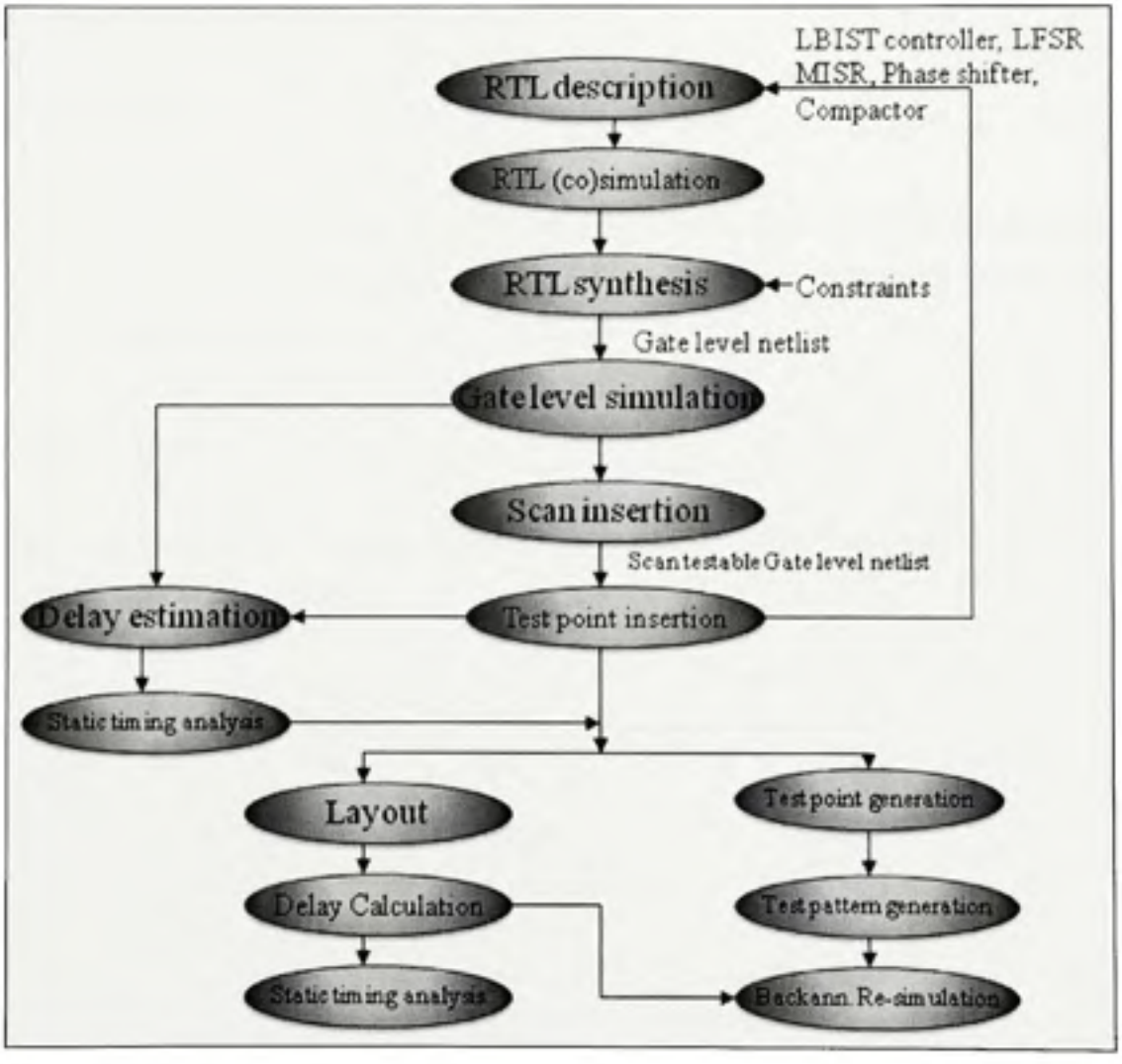

Figure-A I-3 Logic BIST design flow.

ANNEX II

TYPES OF FAULT CLASSES

FastScan categorizes faults into fault classes, based on how the faults were detected or why they could not be detected. Each fault class has a unique name and two character class code. The following, presents some excerpts of the [Mentor Graphics ATPG Guide] showing different types of identified Fault Classes.

UNTESTABLE

Untestable (UT) faults are faults for which no pattern can exist to either detect or possibly detect them. Untestable faults cannot cause functional failures, so the tools exclude them when calculating test coverage.

Unused (UU)

The unused fault class includes all faults on circuitry unconnected to any circuit observation point as shown in figure A-II-1.

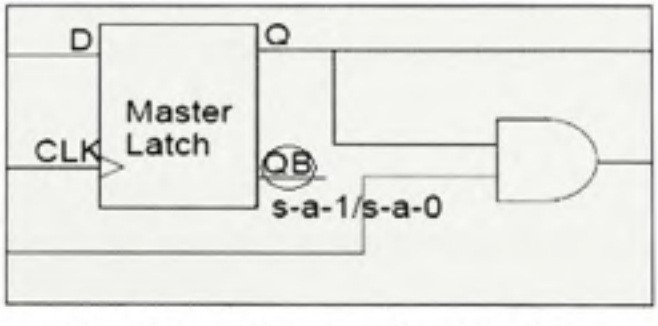

Figure-A II-1 Example of unused fault.

Tied (TI)

The tied fault class includes faults on gates where the point of the fault is tied to a value identical to the fault stuck value.

The tied circuitry could be due to:

- Tied signals
- AND and OR gates with complementary inputs
- Exclusive-OR gates with common inputs
- Line holds due to primary input pins held at a constant logic value during test by CT0 or CT1 pin constraints you applied with the the FastScan or FlexTest.

Because tied values propagate, in figure A-II-2 below, the tied circuitry at A causes the tied faults at A, B, C, and D.

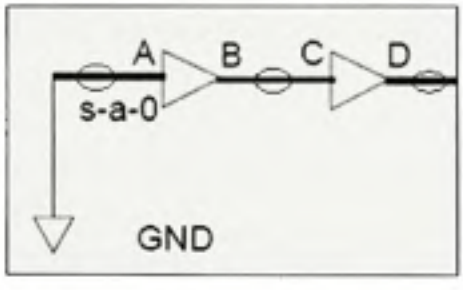

Figure-A II-2 Example of tied fault.

Blocked (BL)

The blocked fault class includes faults on circuitry for which tied logic blocks all paths to an observable point.

The tied circuitry could be due to:

- Tied signals
- AND and OR gates with complementary inputs
- Exclusive-OR gates with common inputs

Tied faults and blocked faults can be equivalent faults. Figure A-II-3 shows the site of a blocked fault.

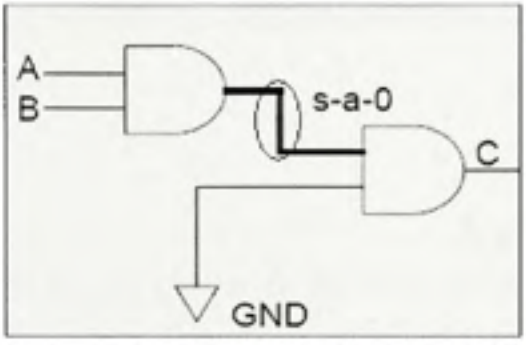

Figure-A II-3 Example of blocked fault.

Redundant (RE)

The redundant fault class includes faults the test generator considers undetectable. After the test pattern generator exhausts all patterns, it performs a special analysis to verify GND.
The fault is undetectable under any conditions. Figure A-II-4 shows the site of a redundant fault.

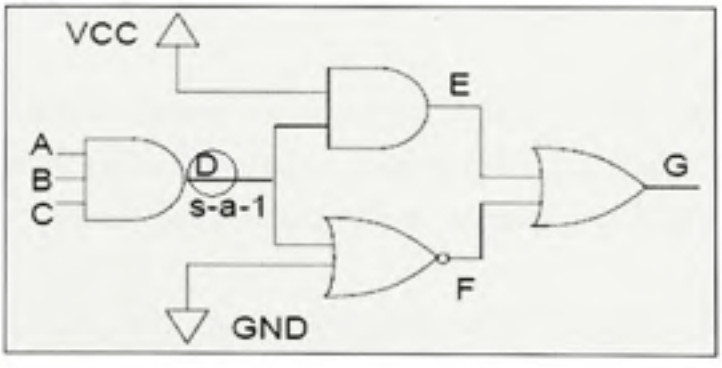

Figure-A II-4 Example of redundant fault.

In this circuit, signal G always has the value of 1, no matter what the values of A, B, and C. If D is stuck at 1, this fault is undetectable because the value of G can never change, regardless of the value at D.

TESTABLE

Testable (TE) faults are all those faults that cannot be proven untestable. The testable fault classes include:

Detected (DT)

The detected fault class includes all faults that the ATPG process identifies as detected. The detected fault class contains two subclasses:

- Det_simulation (DS) - faults detected when the tool performs fault simulation.
- Det_implication (DI) - faults detected when the tool performs learning analysis.

The det_implication subclass normally includes faults in the scan path circuitry, as well as faults that propagate ungated to the shift clock input of scan cells. The scan chain functional test, which detects a binary difference at an observation point guarantees detection of these faults.

Posdet (PD)

The posdet, or possible-detected, fault class includes all faults that fault simulation identifies as possible-detected but not hard detected. A possible-detected fault results from a 0-X or 1-X difference at an observation point. The posdet class contains two subclasses:

posdet_testable (PT) - potentially detectable posdet faults. PT faults result when the tool cannot prove the 0-X or 1-X difference is the only possible outcome. A higher abort limit may reduce the number of these faults.
posdet_untestable (PU) - proven ATPG_untestable and hard undetectable posdet faults.

Uninitialized (UI)

The uninitialized fault class includes faults for which the test generator is unable to:

- find an initialization pattern that creates the opposite value of the faulty value at the fault pin.
- prove the fault is tied.

In sequential circuits, these faults indicate that the tool cannot initialize portions of the circuit.

ATPG_untestable (AU)

The ATPG_untestable fault class includes all faults for which the test generator is unable to find a pattern to create a test, and yet cannot prove the fault redundant. Testable faults become ATPG_untestable faults because of constraints, or limitations, placed on the ATPG tool (such as a pin constraint or an insufficient sequential depth). These faults may be possible-detectable, or detectable, if you remove some constraint, or change some limitation, on the test generator (such as removing a pin constraint or changing the sequential depth).

Undetected (UD)

The undetected fault class includes undetected faults that cannot be proven untestable or ATPG_untestable. The undetected class contains two subclasses:

- uncontrolled (UC) - undetected faults, which during pattern simulation, never achieve the value at the point of the fault required for fault detection—that is, they are uncontrollable.
- unobserved (UO) - faults whose effects do not propagate to an observable point.

All testable faults prior to ATPG are put in the UC category. Faults that remain UC or UO after ATPG are aborted, which means that a higher abort limit may reduce the number of UC or UO faults. Uncontrolled and unobserved faults can be equivalent faults. If a fault is both uncontrolled and unobserved, it is categorized as UC.

ANNEX III

PATH DELAY DISTRIBUTION PERL SCRIPTS

```
############Dofiles  used to automate the process#####################
##   Author      : Roger EL-KAFROUNI                                  ##
##  beginning    : 17 June 2008.                                      ##
##  modified     : 30 october 2008.                                   ##
##   revised     : 18 February 2009.                                  ##
########################LOS pattern Generation#######################

//fastscan -verilog Top_ELABMEM_syn_fs.v -lib atpglib_atsisan18.3.0.atpg -dofile
LOS_batch.dofile
//fastscan -verilog top_ELABMEM.v -lib atpglib_atsisan18.3.0.atpg -dofile
LOS_batch.dofile
set system mode setup
add clocks 0 /t_CLOCK /t_RESET
add scan groups grp1
/users/kafrouni/TDF/SCANPERL/FINALWORK/Tweaks/Top_ELABMEM_syn_fs.testproc
add scan chains chain1 grp1 /scan_in1 /scan_out1
set output masks on
set transition holdpi on

//----------Create Launch off last shift transition fault patterns--------

set system mode atpg
set fault type transition
set pattern type -sequential 0
add faults -all
set Abort Limit 100
create patterns
write faults LOS100FaultsRPT -replace -class FULL
save patterns LOS100PaternsRPT.ascii -ascii -parallel -replace
report aborted faults all >!

/users/kafrouni/TDF/SCANPERL/FINALWORK/Tweaks/LOS100AbortFaultsRPT

############################LOC Pattern Generation#####################
//fastscan -verilog Top_ELABMEM_syn_fs.v -lib atpglib_atsisan18.3.0.atpg -dofile
LOC_batch.dofile
//fastscan -verilog top_ELABMEM.v -lib atpglib_atsisan18.3.0.atpg -dofile
LOC_batch.dofile
set system mode setup
add clocks 0 /t_CLOCK /t_RESET
```

```
add scan groups grp1
/users/kafrouni/TDF/SCANPERL/FINALWORK/Tweaks/Top_ELABMEM_syn_fs.testproc
add scan chains chain1 grp1 /scan_in1 /scan_out1
add pin constraint scan_en1 c0
set output masks on
set transition holdpi on

//----------Create broadside transition fault patterns--------

set system mode atpg
set fault type transition
set pattern type -sequential 2
add faults -all
set fault type transition -no_shift_launch
set Abort Limit 100
create patterns
write faults LOC100FaultsRPT -replace -class FULL
save patterns LOC100PaternsRPT.ascii -ascii -parallel -replace
report aborted faults all >!
/users/kafrouni/TDF/SCANPERL/FINALWORK/Tweaks/LOC100AbortFaultsRPT


#PrimeTime SCRIPTS  Used to determine the post layout static timing analysis Process.
#This script is an example of the main script, where I am not including all the paths used
#in the process.
###Roger EL-KAFROUNI, Masters degree candidate, ETS 2008#####

#PrimeTime script
#Author: Roger EL-KAFROUNI, ETS, Lacime
#June 17 2008

#Start of scripts:
#================================
echo "running Report Timing script"
set scenario slow
#set scenario typical

#----- set these for setup -----------
set type setup
set fname elabmem.$scenario.$type.timing
set dtype max
#-------------------------------------

#----- set these for hold -------------
#set type hold
#set fname elabmem.$scenario.$type.timing
```

```
#set dtype min
#-------------------------------------

#=============================================================================
source .synopsys_pt.setup
echo "read in 3 essential files: netlist, constraints and spef file + link the design"
read_verilog top_ELABMEM.v
link_design top_ELABMEM
echo "reading spef files..."
read_parasitics -quiet -increment top_ELABMEM.spef
#=============================================================================
#=============================================================================
echo " create a virtual clock on the output of the clock pad"
create_clock -name CLK -period 10 -waveform [list 0 5] [get_pins clk_pad/C]
#create_clock -name RST -period 10 -waveform [list 0 5] [get_pins res_pad/C]
#create_clock -name CLK -period 10 -waveform [list 0 5] [get_pins s_CLOCK/C]
#create_clock -name CLK -period 10 -waveform [list 0 5] [get_pins clk_pad/PAD]

set_dont_touch_network [get_clocks CLK]
set_propagated_clock [get_clocks CLK]

#set_dont_touch_network [get_clocks RST]
#set_propagated_clock [get_clocks RST]
#=============================================================================


echo "Propagating the transition along the paths"
#echo "set case analysis to enable caputre mode"
set_case_analysis 0 [get_ports scan_en1]

#echo "set case analysis to enable shift mode"
#set_case_analysis 1 [get_ports scan_en1]

#==========A sample from a nodes Gate report=================================

echo "report timing on all non-covered nodes from output of Inflops to input of outflops"
echo "Non-covered Node U328/Y " >> $fname
report_timing -from link_ELABMEM_MAX_reg_8/Q   -through U328/Y  -to
link_ELABMEM_MAX_reg_8/D   -nets -nosplit -delay_type max >> $fname
echo "Non-covered Node U328/Y " >> $fname
report_timing -from link_ELABMEM_MAX_reg_8/Q   -through U328/Y  -to
link_ELABMEM_MAX_reg_7/D   -nets -nosplit -delay_type max >> $fname
echo "Non-covered Node U328/Y " >> $fname
report_timing -from link_ELABMEM_MAX_reg_8/Q   -through U328/Y  -to
link_ELABMEM_MAX_reg_5/D   -nets -nosplit -delay_type max >> $fname
```

```
echo "Non-covered Node U328/Y " >> $fname
report_timing -from link_ELABMEM_MAX_reg_8/Q  -through U328/Y  -to
link_ELABMEM_MAX_reg_6/D  -nets -nosplit -delay_type max >> $fname
echo "Non-covered Node U328/Y " >> $fname
report_timing -from link_ELABMEM_MAX_reg_8/Q  -through U328/Y  -to
link_ELABMEM_MAX_reg_3/D  -nets -nosplit -delay_type max >> $fname
echo "Non-covered Node U328/Y " >> $fname
report_timing -from link_ELABMEM_MAX_reg_8/Q  -through U328/Y  -to
link_ELABMEM_MAX_reg_1/D  -nets -nosplit -delay_type max >> $fname
echo "Non-covered Node U328/Y " >> $fname
report_timing -from link_ELABMEM_MAX_reg_8/Q  -through U328/Y  -to
link_ELABMEM_MAX_reg_2/D  -nets -nosplit -delay_type max >> $fname
echo "Non-covered Node U328/Y " >> $fname
report_timing -from link_ELABMEM_MAX_reg_8/Q  -through U328/Y  -to
link_ELABMEM_MAX_reg_4/D  -nets -nosplit -delay_type max >> $fname
echo "Non-covered Node U328/Y " >> $fname
report_timing -from link_ELABMEM_MAX_reg_8/Q  -through U328/Y  -to
link_ELABMEM_MAX_reg_0/D  -nets -nosplit -delay_type max >> $fname
echo "Non-covered Node link_ELABMEM_sub_80_U2_7/B " >> $fname
report_timing -from link_ELABMEM_MAX_reg_7/Q  -through
link_ELABMEM_sub_80_U2_7/B -to link_ELABMEM_MAX_reg_8/D  -nets -nosplit -
delay_type max >> $fname
echo "Non-covered Node link_ELABMEM_sub_80_U2_7/B " >> $fname
report_timing -from link_ELABMEM_MAX_reg_7/Q  -through
link_ELABMEM_sub_80_U2_7/B -to link_ELABMEM_MAX_reg_7/D  -nets -nosplit -
delay_type max >> $fname
echo "Non-covered Node link_ELABMEM_sub_80_U2_7/B " >> $fname
report_timing -from link_ELABMEM_MAX_reg_7/Q  -through
link_ELABMEM_sub_80_U2_7/B -to link_ELABMEM_MAX_reg_5/D  -nets -nosplit -
delay_type max >> $fname
echo "Non-covered Node link_ELABMEM_sub_80_U2_7/B " >> $fname
report_timing -from link_ELABMEM_MAX_reg_7/Q  -through
link_ELABMEM_sub_80_U2_7/B -to link_ELABMEM_MAX_reg_6/D  -nets -nosplit -
delay_type max >> $fname
echo "Non-covered Node link_ELABMEM_sub_80_U2_7/B " >> $fname
report_timing -from link_ELABMEM_MAX_reg_7/Q  -through
link_ELABMEM_sub_80_U2_7/B -to link_ELABMEM_MAX_reg_3/D  -nets -nosplit -
delay_type max >> $fname
echo "Non-covered Node link_ELABMEM_sub_80_U2_7/B " >> $fname
report_timing -from link_ELABMEM_MAX_reg_7/Q  -through
link_ELABMEM_sub_80_U2_7/B -to link_ELABMEM_MAX_reg_1/D  -nets -nosplit -
delay_type max >> $fname
echo "Non-covered Node link_ELABMEM_sub_80_U2_7/B " >> $fname
```

```
report_timing -from link_ELABMEM_MAX_reg_7/Q   -through
link_ELABMEM_sub_80_U2_7/B  -to link_ELABMEM_MAX_reg_2/D   -nets -nosplit -
delay_type max >> $fname
echo "Non-covered Node link_ELABMEM_sub_80_U2_7/B " >> $fname
report_timing -from link_ELABMEM_MAX_reg_7/Q   -through
link_ELABMEM_sub_80_U2_7/B  -to link_ELABMEM_MAX_reg_4/D   -nets -nosplit -
delay_type max >> $fname
echo "Non-covered Node link_ELABMEM_sub_80_U2_7/B " >> $fname
report_timing -from link_ELABMEM_MAX_reg_7/Q   -through
link_ELABMEM_sub_80_U2_7/B  -to link_ELABMEM_MAX_reg_0/D   -nets -nosplit -
delay_type max >> $fname


#ETS 2008-2009, Roger EL-KAFROUNI
# The results are LosFaults, LocFaults, and commonLosLoc faults
# The non-covered faults by LOC but covered by LOS are found in
# the file "LocFaults". From it we'll find the FF destinations
# leading to finding the appropriate number of sensors needed to
# cover all faults.

#!/usr/bin/perl -w

open (IN_FILE_LOS, "<./LOS-UO-FAULTS.rpt") or die;
open (IN_FILE_LOC, "<./LOC-UO-FAULTS.rpt") or die;
#open (IN_FILE_LOS, "<$ARG[0]") or die("can't find los fault coverage file\n");
#open (IN_FILE_LOC, "<$ARG[1]") or die("can't find loc fault coverage file\n");
open (OUT_FILE_COM, ">commonLosLoc.log");
open (OUT_FILE_LOS, ">LosFault.log");
open (OUT_FILE_LOC, ">LocFault.log");
open (OUT_FILE_ReportGates_FS, ">ReportGates_FS.log");
#open (OUT_FILE_TEST, ">TestLine.log");

@los_data = <IN_FILE_LOS>;
@loc_data = <IN_FILE_LOC>;

$FAULT_LOS = 0;
$FAULT_LOC = 0;

$FOUND_LOS_LOC_COM = 0;
$FOUND_LOS = 0;
$FOUND_LOC = 0;

#=======================================
# Find LOS only faults and common faults
#=======================================
$LineNum_los = 0;
```

```
$LineNum_loc = 0;

 foreach $line (@los_data)
 {

   #remove leading and trailing spaces
   #$line =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;
   #remove trailing white space
   #$line =~ s/\s+$//;
   #remove leading white sapce
   #$line =~ s/^\s+//;

   if ($line =~ /^(.*)(.UO)(.*)$/)
       {
        $TargetNodeLOS = $2."\n";
        $TargetNodeLOS =~ s/^\s+//;
        chomp($TargetNodeLOS);
#         print OUT_FILE_TEST "$TargetNodeLOS \n";
       }


   $FAULT_LOS[$LineNum_los]=$TargetNodeLOS;

  foreach $line (@loc_data)
   {

   if ($line =~ /^(.*)(.UO)(.*)$/)
       {
        $TargetNodeLOC = $2."\n";
        $TargetNodeLOC =~ s/^\s+//;
        chomp($TargetNodeLOC);
#         print OUT_FILE_TEST "$TargetNodeLOC \n";
       }


   $FAULT_LOC[$LineNum_loc]=$TargetNodeLOC;
   if ($FAULT_LOS[$LineNum_los] eq $FAULT_LOC[$LineNum_loc])
    {
     $FOUND_LOS_LOC_COM[$LineNum_los] = $FAULT_LOS[$LineNum_los];
    }
   elsif ($FAULT_LOS[$LineNum_los] eq $FAULT_LOC[$LineNum_loc])
    {
     $FOUND_LOS[$LineNum_los] = $FAULT_LOS[$LineNum_los];
    } #elsif
   $LineNum_loc = $LineNum_los + 1;
```

```
}# foreach (loc)

   print OUT_FILE_COM "$FOUND_LOS_LOC_COM[$LineNum_los]\n";
   print OUT_FILE_LOS "$FOUND_LOS[$LineNum_los]\n";
       $LineNum_loc = 0;

}

#================
# LOC faults only
#================
$LineNum_los = 0;
$LineNum_loc = 0;

foreach $line (@loc_data)
{

  if ($line =~ /^(.*)(.UO)(.*)$/)
      {
       $TargetNodeLOC = $3."\n";
       chomp($TargetNodeLOC);
      }

  $FAULT_LOC[$LineNum_loc]=$TargetNodeLOC;

  foreach $line (@los_data)
  {

    $TargetNodeLOS = $3."\n";
    $TargetNodeLOS =~ s/^\s+//;
    chomp($TargetNodeLOS);

  $FAULT_LOS[$LineNum_los]=$TargetNodeLOS;
  if ($FAULT_LOC[$LineNum_loc] ne $FAULT_LOS[$LineNum_los])
   {
    $FOUND_LOC[$LineNum_loc] = $FAULT_LOC[$LineNum_loc];
   }

  } #foreach (los)

    if ($LineNum_loc > 0) {
      if ($FOUND_LOC[$LineNum_loc] ne $FOUND_LOC[@LineNum_loc - 1]){

        print OUT_FILE_LOC "$FOUND_LOC[$LineNum_loc]\n";
```

```
        print OUT_FILE_ReportGates_FS qq{echo "CMD> report gates -endpoints -
backward $FOUND_LOC[$LineNum_loc]" >> gates_report.rpt\n};
        print OUT_FILE_ReportGates_FS "report gates -endpoints -backward
$FOUND_LOC[$LineNum_loc] >> gates_report.rpt\n";

        print OUT_FILE_ReportGates_FS qq{echo "CMD> report gates -endpoints -
forward $FOUND_LOC[$LineNum_loc]" >> gates_report.rpt\n};
        print OUT_FILE_ReportGates_FS "report gates -endpoints -forward
$FOUND_LOC[$LineNum_loc] >> gates_report.rpt\n";

        $LineNum_loc = $LineNum_loc + 1;

      } else {$LineNum_loc = $LineNum_loc + 1; }
    } else {  # first line in file

        print OUT_FILE_LOC "$FOUND_LOC[$LineNum_loc]\n";

        print OUT_FILE_ReportGates_FS qq{echo "CMD> report gates -endpoints -
backward $FOUND_LOC[$LineNum_loc]" >! gates_report.rpt\n};
        print OUT_FILE_ReportGates_FS "report gates -endpoints -backward
$FOUND_LOC[$LineNum_loc] >> gates_report.rpt\n";

        print OUT_FILE_ReportGates_FS qq{echo "CMD> report gates -endpoints -
forward $FOUND_LOC[$LineNum_loc]" >> gates_report.rpt\n};
        print OUT_FILE_ReportGates_FS "report gates -endpoints -forward
$FOUND_LOC[$LineNum_loc] >> gates_report.rpt\n";

        $LineNum_loc = $LineNum_loc + 1;
        $LineNum_los = 0;
        }
  }

close IN_FILE_LOS;
close IN_FILE_LOC;
close OUT_FILE_COM;
close OUT_FILE_LOS;
close OUT_FILE_LOC;
close OUT_FILE_ReportGates_FS;


#ETS 2009, Roger EL-KAFROUNI
#The results are, the arrival times of the paths. I am rendering the timing file that I generated
#using sysnopsys PrimeTime, in order to process the data, scan through and extrapolate the
#needed information.
```

```
#!/usr/bin/perl -w

open (IN_FILE_PathDlysData, "<./elabmem.slow.setup.timing") or die;
open (OUT_FILE_TEST, ">ArrivalTime.rpt");

@node_data = <IN_FILE_PathDlysData>;

$flag = 0;

 foreach $line (@node_data)
 {
 if (($line =~ /Non-covered Node/) && ($flag eq 1))
  {
   chomp($line);
   print OUT_FILE_TEST "$line: ";

   #$uncovNode = $2."\n";
   $flag = 1;
   #print OUT_FILE_TEST "$uncovNode:  ";
  }

 # get slack data
 #if (($line =~ /^( *)(.slack)(.*)$/) && ($flag eq 0))
 if (($line =~ /data arrival time/) && ($flag eq 1))
  {
   s/data arrival time//g foreach ($line);
   $line =~ s/^\s+//;
   chomp($line);
   print OUT_FILE_TEST "$line ns\n";
   $flag = 0;
  }
 }#foreach

close IN_FILE_PathDlysData;
close OUT_FILE_TEST;


#ETS 2009, Roger EL-KAFROUNI
#The results are, MinPathDelay.rpt, MinPathDelay_Node.rpt, and
#MinPathDelay_NodeValue.rpt.

#!/usr/bin/perl -w

open (IN_FILE_testdelay, "<./ArrivalTime.rpt") or die;
open (OUT_FILE_MIN, ">MinPathDelay.rpt");
```

```
open (OUT_FILE_MIN1, ">MinPathDelay_Node.rpt");
open (OUT_FILE_MIN2, ">MinPathDelay_NodeV.rpt");

@node_data = <IN_FILE_testdelay>;

$flag = 0;
$line_count = 0;

 foreach $line (@node_data)
 {
   #find Node
  if (($line =~ /Non-covered Node/) && ($flag eq 1))
   {
   $Node = $line;
   s/Non-covered Node //g foreach ($Node);
   s/\ :\ [0-9]+/ /g foreach($Node);
   $Node =~ s/\s+$//;
   chomp($Node);
   $Found_Node = $Node;
   $flag = 0;
    if ($line_count eq 0) { $Found_Node_prev = $Found_Node; }

   #print OUT_FILE_MIN "$Found_Node \n";
   }

    # find Node Value
  if (($line =~ /Non-covered Node/) && ($flag eq 2))
   {
   $NodeValue = $line;
   s/Non-covered Node //g foreach ($NodeValue);
   s/$Found_Node \://g foreach ($NodeValue);
   #remove leading white sapce
   $NodeValue =~ s/^\s+//;
   $Found_NodeValue = $NodeValue;
   $flag = 2;
    if ($line_count eq 0) { $Found_NodeValue_prev = $Found_NodeValue; }
   $line_count = $line_count + 1;

   #print OUT_FILE_MIN "$Found_NodeValue \n";
   }

  if (($Found_Node eq $Found_Node_prev) && ($flag eq 1))
   {
   if (($Found_NodeValue eq $Found_NodeValue_prev) or ($Found_NodeValue >
$Found_NodeValue_prev))
```

```
    {
     $flag = 0;
    }
  elsif ($Found_NodeValue < $Found_NodeValue_prev)
    {
     $Found_Node_prev      = $Found_Node;
     $Found_NodeValue_prev = $Found_NodeValue;
    }
 }
 elsif (($Found_Node ne $Found_Node_prev) && ($flag eq 2))
 {
    chomp($Found_Node_prev);
    chomp($Found_NodeValue_prev);
    print OUT_FILE_MIN  "$Found_Node_prev: $Found_NodeValue_prev ns\n";
    print OUT_FILE_MIN1  "$Found_Node_prev\n";
    print OUT_FILE_MIN2 "$Found_NodeValue_prev\n";
    $flag = 0;
    $Found_NodeValue_prev = $Found_NodeValue;

 }

}#foreach

close IN_FILE_testdelay;
close OUT_FILE_MIN;
close OUT_FILE_MIN1;
close OUT_FILE_MIN2;


#ETS 2008, Roger EL-KAFROUNI
#The results are, MinDlyDistUncov.rpt, MinPathDlyData.rpt, and #MinPathDlyFreq.rpt.
#This script determines the Paths delays and the equivalent frequency of occurrences.

#!/usr/bin/perl -w
#output file: MinNodeOcc.rpt, MinOccurence.rpt, MinNodeValue.rpt
#perl find_min_Occ.pl

open (IN_FILE_testdelay, "<./MinPathDelay_NodeV.rpt") or die;
open (OUT_FILE_MIN, ">MinDlyDistUncov.rpt");
open (OUT_FILE_MIN1, ">MinPathDlyData.rpt");
open (OUT_FILE_MIN2, ">MinPathDlyFreq.rpt");

@node_data = <IN_FILE_testdelay>;

$counter = 0;
```

```
$index = 0;
$LoadedValues=0;
$Value=0;
$Data=0;

print OUT_FILE_MIN "Value \t Occurence\n";
print OUT_FILE_MIN "\******   *********\n";

foreach $line (@node_data)
  {
  $Data=$line;
  chomp($Data);
  $LoadedValues[$index] = $Data;
 #print OUT_FILE_MIN "$LoadedValues[$index]\n";
  $index = ++$index;
 }#foreach

foreach $line (@node_data)
 {
    $Value=$line;
    chomp($Value);
 for ($i=1; $i<=$#LoadedValues; $i++)
  {
   if($Value ne $LoadedValues[$i])
     {
     $counter = $counter + 1;
     }#if
  }#for
 chomp($line);
 chomp($counter);
#print OUT_FILE_MIN "Value \     Occurence\n";
 if ($counter ne 1)
  {
   print OUT_FILE_MIN "$line \     $counter\n";
   print OUT_FILE_MIN1 "$line\n";
   print OUT_FILE_MIN2 "$counter\n";
  }
 }#foreach
close IN_FILE_testdelay;
close OUT_FILE_MIN;
```

ANNEX IV

CDT SENSOR PLACEMENT AND OPTIMIZATION PERL SCRIPTS

```
###   Author       : Roger EL-KAFROUNI                    ###
###   beginning    : 22 october 2008.                     ###
###   modified     : 25 February 2009.                    ###
###   revised      : 18 January 2010.                     ###
```

rmduplicate.pl

##We filter out the lines from repetitive lines in the Destination file prior to creating the new ##structure where we show each destination ff with its related non-covered nodes. This list ##corresponds at the maximum coverage list of ff destinations at which we are going to input ##sensors.

```
open (IN_FILE_NodeData, "<./DESTINATION.rpt") or die;
open (OUT_FILE_RMD, ">RMDUP.rpt");
@node_data = <IN_FILE_NodeData>;
foreach $line (@node_data)
{
 if ($line =~ /^(Destination:)(.*)$/)
 {
  $DestPoint = $2."\n";
  $DestPoint  =~ s/\/\D//;
  $DestPoint  =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;
  @a[$myline] = $DestPoint;
  $myline = $myline + 1;
 }
}
close IN_FILE_NodeData;
sub remove_duplicates(\@)
{
```

```
    my $ar = shift;
    my %seen;
    for ( my $i = 0; $i <= $#{$ar} ; )
    {
        splice @$ar, --$i, 1
            if $seen{$ar->[$i++]}++;
    }
}
remove_duplicates( @a );
for ($i=0; $i<= $#a; $i++)
 {
  print OUT_FILE_RMD "$a[$i]\n";
 }
close OUT_FILE_RMD;
```

DestFFallNodes.pl

```
#Create the structure that shows for each destination ff its relating non-covered nodes.
#!/usr/bin/perl -w
open (IN_FILE_FFDest, "<./RMDUP.rpt") or die;
open (IN_FILE_FFDestNode, "<./DESTINATION.rpt") or die;
open (OUT_FILE_DESTINATION, ">ListFFDest.rpt");

@node_FFDest = <IN_FILE_FFDest>;
@node_FFDestNode = <IN_FILE_FFDestNode>;

$DestPoint = 0;
$NodePoint = 0;
$flag=0;
$DestFF=0;
```

```
$counter = 0;

foreach $line (@node_FFDest)
{
$DestFF=$line;
$DestFF =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;
$counter = 0;

 foreach $line (@node_FFDestNode)
 {
  if ($line =~ /^(Destination:)(.*)$/)
  {
   $DestPoint = $2."\n";
   $DestPoint  =~ s/\/\D//;
   $DestPoint  =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;

    if ($DestFF eq $DestPoint)
    {
    $flag=1;
    }
  }#if
  if (($line =~ /^(Node:)(.*)$/) && ($flag eq 1))
  {
   $NodePoint = $2."\n";
   $NodePoint =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;
   print OUT_FILE_DESTINATION "Destination: $DestPoint Node: $NodePoint\n";
   $counter = $counter + 1;
   $flag=0;
  }
 }
```

```
}
close IN_FILE_FFDest;
close IN_FILE_FFDestNode;
close OUT_FILE_DESTINATION;
```

RMDUPLICATE2.pl

```
#We filter out the lines from repetitive lines in the Destination file prior to creating the new
#structure where we show each destination ff with its related non-covered nodes. This list
#corresponds at the maximum coverage list of ff destinations at which we are going to input
#sensors.

open (IN_FILE_NodeData, "<./ListFFDest.rpt") or die;
open (OUT_FILE_RMD, ">RMDUP2.rpt");

@node_data = <IN_FILE_NodeData>;

foreach $line (@node_data)
{
 if ($line =~ /^(Destination:)(.*)$/)
 {
  $DestPoint = $2."\n";
  $DestPoint  =~ s/\/\D//;
  $DestPoint  =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;
  @a[$myline] = $DestPoint;
  $myline = $myline + 1;
 }
}
close IN_FILE_NodeData;

sub remove_duplicates(\@)
```

```
{
   my $ar = shift;
   my %seen;
   for ( my $i = 0; $i <= $#{$ar} ; )
   {
      splice @$ar, --$i, 1
         if $seen{$ar->[$i++]}++;
   }
}

#my @a = qw( a a b c c c d e f e f e a f g h h h );
remove_duplicates( @a );
for ($i=0; $i<= $#a; $i++)
 {
  print OUT_FILE_RMD "$a[$i]\n";
 }
close OUT_FILE_RMD;
```

FF2NodeCount.pl

```
#Here we start STEP 1 of the iteration.
#we are generating the report of node count for each destination FF.

#!/usr/bin/perl -w
#open (IN_FILE_FFDest, "<./RMDUPEX2.rpt") or die;
#open (OUT_FILE_DESTINATION, ">FF2NodeCnt.rpt");
open (IN_FILE_FFDest, "<./ExcludedList1.rpt") or die;
open (OUT_FILE_DESTINATION, ">FF2NodeCnt2.rpt");

@node_FFDest = <IN_FILE_FFDest>;
```

```
$DestPoint = 0;
$flag=0;
$counter = 1;
$SavedDestPoint = 0;

foreach $line (@node_FFDest)
 {
  if (($line =~ /^(Destination:)(\ )(.*)(\ )(Node:)(\ )(.*)$/) && ($flag eq 0))
  {
   $DestPoint = $3."\n";
   $DestPoint  =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;
   $SavedDestPoint = $DestPoint;
   $flag = 1;
  }

  if (($line =~ /^(Destination:)(\ )(.*)(\ )(Node:)(\ )(.*)$/) && ($flag eq 1))
  {
   $DestPoint = $3."\n";
   $DestPoint  =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;
   if ($DestPoint eq $SavedDestPoint)
    {
     $counter = $counter + 1;
    }
    else {
        print OUT_FILE_DESTINATION "Destination:  $SavedDestPoint \t $counter\n";
        $counter = 1;
        $SavedDestPoint = $DestPoint;
        }
  } #if
 }# foreach
```

```
close IN_FILE_FFDest;
close OUT_FILE_DESTINATION;
```

Sample1Nodes.pl

```
#Here we start STEP 2 of the iteration.
#This script takes the sample data with ff destination and nodes and output the nodes only for
#further use to eliminate any same instances of non-covered nodes from the big list further
#reducing the number of needed Destination FFs.

#!/usr/bin/perl -w
open (IN_FILE_FFDestSmpl1, "<./sample1.rpt") or die;
open (OUT_FILE_RMD, ">NodeSample1.rpt");

@node_FFDestSample = <IN_FILE_FFDestSmpl1>;

$SampleNodePoint = 0;
$pointer = 0;

foreach $line (@node_FFDestSample)
 {
  if ($line =~ /^(Destination:)(\ )(.*)(\ )(Node:)(\ )(.*)$/)
   {
   $SampleNodePoint = $7."\n";
   $SampleNodePoint =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;
   $SampleNodes[$pointer] = $SampleNodePoint;
   $pointer = $pointer + 1;
  }
}
```

```
for ($i=0; $i<=$#SampleNodes; $i++)
{
 print OUT_FILE_RMD "$SampleNodes[$i]\n";
}

close IN_FILE_FFDestSmpl1;
close OUT_FILE_RMD;
```

PreIteration.pl

```
#Here we start STEP 3 of the iteration.
#This script scans through the file and removes all the FFs whose nodes are similar to that of
#the highest FF destination nodes.

#!/usr/bin/perl -w
#use strict;
#use warnings;

open (IN_FILE_FFDestSmpl, "<./NodeSample1.rpt") or die;
open (IN_FILE_FFDest, "<./RMDUP2.rpt") or die;
open (OUT_FILE_DESTINATION, ">ExcludedList1.rpt");

@node_FFDestSmpl = <IN_FILE_FFDestSmpl>;
@node_FFDest = <IN_FILE_FFDest>;
$SampleNodePoint=0;
$SampledLine=0;
$NodePoint=0;
#$count=0;
#$SavedLines[$count]=0;

$flag=0;
```

```
$line_cntr = 0;
$StoreLine[$line_cntr] = 0;
$SampleLine = 0;

foreach $line (@node_FFDestSmpl)
{
  if ($line =~ /^(.*)$/)
  {
   $SampleNodePoint = $1."\n";
   $SampleNodePoint =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;
  }

#We are reading the file and storing the unmatched first iteration
#lines in an array.
if($flag eq 0)
{
  foreach $line (@node_FFDest)
   {
    #Store the line first than see if you want to keep it.
    if ($line =~ /^(Destination:)(\ )(.*)(\ )(Node:)(\ )(.*)$/)
    {
     $SampledLine = $line;
     $NodePoint = $7."\n";
     $NodePoint =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;
     $SampledLine =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;
     $StoreLine[$line_cntr] = $SampledLine;
     #print OUT_FILE_DESTINATION "$SampledLine\n";
    }#if
    if ($SampleNodePoint ne $NodePoint)
    {
```

```
    #Update the line_cntr to store the current line. Otherwise,
    #next line will overwrite the current one in the array
    $line_cntr = $line_cntr + 1;
    }

  }# foreach OF node_FFDest

 $flag = 1;
}

 if ($flag ne 0)
 {
    for ($index = 0; $index <= $#StoreLine; $index++)
    {
     $SampleLine = $StoreLine[$index];
   if ($SampleLine =~ /^(Destination:)(\ )(.*)(\ )(Node:)(\ )(.*)$/)
      {
       $NodePoint = $7."\n";
       $NodePoint =~ s/^\s*(\S*(?:\s+\S+)*)\s*$/$1/;

#Update counter to delete the current line. Otherwise, leave it in the array.
if ($SampleNodePoint eq $NodePoint)
         {

          $StoreLine[$index] = "";
         }
      }
    }# for loop
  }# flag not equal to 0
}#foreach $line (@node_FFDestSmpl)
```

```
for ($i=0; $i<= $#StoreLine; $i++)
{
 if ($StoreLine[$i] ne "")
 {
  print OUT_FILE_DESTINATION "$StoreLine[$i]\n";
 }
}
close IN_FILE_FFDest;
close IN_FILE_FFDestSmpl;
close OUT_FILE_DESTINATION;
```

BIBLIOGRAPHY

Ahmed N., Tehranipoor M., Jayaram V. 2006. "Timing-based delay test for screening small delay defects". Design automation conference, 43rd ACM/IEEE, pp. 320-325.

Bareisa E., Jusas V., Motiejunas K., Seinauskas R. 2008. "Functional delay clock fault models". 124X, Information technology and control. Volume 37, No. 1.

Benayahu N., A. Chechik, Ron Press. 2007. "Launch-off-shift At-Speed Test". Test and Measurement World. pp.37-42.

Chadrakasan A., Robert W. B. 2000. Design of High-Performance Micro-processor Circuits. Wiley-IEEE press, pp. 98-116.

Crouch A., G. Eide. 2006. Advances In Electronic Testing: Challenges and Methodologies: DFT-Oriented, Low-Cost Testers, edited by Dimitris Gizopoulos, Springer 2006, page 179.

Davidson S., Helen D. 2007. "The phsychology of electronic test"'. IEEE design and test of computers, vol. 24, n°5, p.495-501.

Datta R., Sebastine A., Gupta R., J. Townsend W., A. Abraham J. 2004. "Test and debug in deep-submicron technologies". ACAS 2004, Computer Eng. Research Center, University of Texas at Austin.

Groeneveld P. R. 2002, "Physical Design Challenges for Billion Transistor Chips". IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 78-83, 2002.

Hawkings C., Keshavarzi A., Segura J. 2003. "A view from the bottom: Nanometer technology AC parametric failures – Why, where, and how to detect. IEEE international symposium on defect and fault tolerence in VLSI systems, pp. 267-276.

Hetherington G., Fryars T., Tamarapalli N., Kassab M., Hassan A., Rajski J. 1999. "Logic BIST for large industrial designs: real issues and case studies". Test conference proceedings international, vol. issue, pp. 358-367, 1999.

ITC Benchmark. 2010. The International Test Conference, ITC99 Benchmark Home Page. On line. <www.cerc.utexas.edu/itc99-benchmarks/bench.html>. Consulted October 2008.

Kaufman M. 2008. "System-on-Chip Test Architectures: Nanometer Design For Testability", edited by Laung-Terng Wang, Charles E. Stroud, Nur A. Touba, Elsevier Inc. 2008.

Kim et al.K. S. 2003. "Delay Defect Characteristics and Testing Strategies," IEEE Design & Test, vol. 20, no.5, Sept. 2003, pp. 8-16.

Krstic A., K. Cheng, J. Liou, M. Abadir. 2003. "Delay Defect Diagnosis Based Upon Statistical Timing Models - The First Step". Computers and Digital Techniques, IEEE Proceedings, Stevenage, United Kingdom, pp. 346-54.

Liou J.-J., Wang, L.-C., Cheng K.-T., Dworak J., Mercer M.R., Kapur R.,Williams T.W. 2002. "Analysis of delay test effectiveness with a multiple-clock scheme". ITC international test conference. (Dec. 10 2002), pp. 407-416.

Lin X., Kun-Han T., Chen W., Mark K., Janusz R., Takeo K., Randy K., Yasuo S., Shuji H., Takashi A. 2006. "Timing-Aware ATPG for High Quality At-speed Testing of Small Delay Defects". Asian Test Symposium, ats 2006, pp. 139-146.

Lin X., Press R., Rajski J., Reuter P., Rinderknecht T., Swanson B., Tamarapalli N. (2003). "High-frequency, at-speed scan testing". Volume 20, Issue 5, pp. 17 – 25.

Madge R., G. Oregon, B. Benware, R. Turakhia, J. Ruffler. 2004. "In search of the optimum test set – adaptive test methods for maximum defect coverage and lowest test cost". Test conference, 2004, proceedings, ITC 2004, international, pp.203-212.

Mak T.M., Krstic A., Cheng K.-T., Wang Li.-C. 2004. "New challenges in delay testing of nanometer, multigigahertz designs". Vol. 21, Issue 3, May-June 2004, pp. 241 – 248.

Mentor Graphics ATPG Guide. 2006. Scan and ATPG Process Guide (DFTAdvisor™, FastScan™ and FlexTest™), Software Version 8.2006_3. August 2006.

Nanowerk Spotlight. Fullerene resist materials for the 32nm node and beyond. 2008. Nanowerk Nanotechnology Spotlight posts. On ligne.
< http://www.nanowerk.com/spotlight/spotid=6625.php>. Consulted August 5 2008.

Pateras S. 2003. "Achieving at-speed structural test". Design & test of computers: IEEE, pp. 26-33.

Qiu W., Wang J., Lu X., Li Z., D.M.H. W., Shi W. 2004. "At-speed test for path delay faults using practical techniques". Current and defect based testing: DBT 2004. Proceedings. IEEE international workshop. Volume, issue, pp. 61-66.

Sachdev M., J. P. D. Gyves. Eide. 2007. "Defect Oriented Testing for Nano-Metric CMOS VLSI Circuitsnd Methodologies", 2nd ed. Springer 2007, page 25.

Saxena, J. Butler, K.M. Gatt, J. Raghuraman, R. Kumar, S.P. Basu, S. Campbell, D.J. Berech, J. 2002. "Scan-based transition fault testing - implementation and low cost test challenges". Test Conference, 2002. Proceedings, International, pp. 1120-1129.

Sengupta S., S. Kundu, S. Chakravarty, P. Parvathala, R. Galivanche, G. Kosonocky, M. Rodgers, and T. M. Mak, 1999. Defect-based test: A key enabler for successful migration to structural test, *Intel Technology J.*, pp. 1–14, Q1 1999.

Synopsys DFT Compiler, "User Manual for SYNOPSYS Toolset Version 2007.12," Synopsys, Inc., 2007.

Thibeault C. 2006. "Improving Digital IC testing with Analog Circuits". Circuits and Systems, IEEE North-East Workshop, Gatineau, Quebec, pp. 285-288, 30 Nov. 2006.

Vorisek V., Koch T., Fischer H. "At-speed testing of SOC ICs". 2004. Design, Automation and Test in Europe Conference and Exhibition. Volume 3, pp.120- 125.