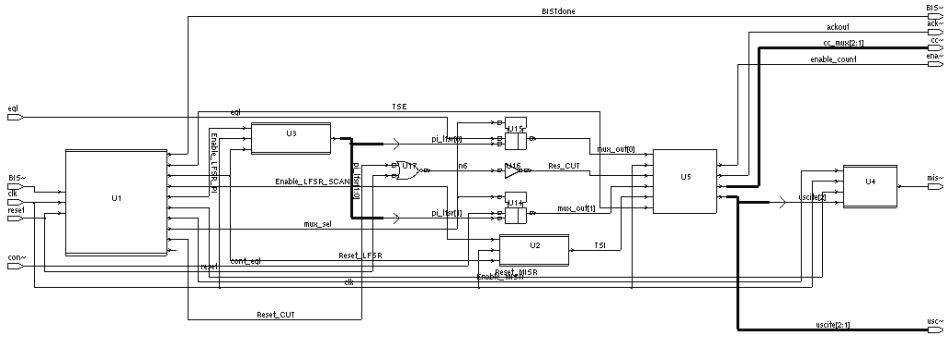
INSTITUTO SUPERIOR TÉCNICO

Universidade Técnica de Lisboa



BIST Architectures for Dynamic Test

Arquitecturas BIST para Teste Dinâmico

Sílvia Andrea Teixeira Gomes

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Júri

Presidente: José António Beltran Gerald

Orientador: João Paulo Cacho Teixeira

Vogais: Marcelino Bicho dos Santos

Isabel Maria Silva Nobre Parreira Cacho Teixeira (co-orientadora)

Outubro 2007

Acknowledgments

I would like to thank to Professors João Paulo Teixeira and Isabel Teixeira for all the support, encouragement and guidance throughout this work.

I would also like to thank to Professor Marcelino Santos and Abílio Parreira for all the available time, help and support. A special thank to Professor António Serralheiro for the attention and motivation during this work and for the last four years.

I am extremely thankful to Jorge Semião for all the support, motivation, advices, available time and help during the last mouths.

Finally, I would like to thank to my family for their unconditional love and support on me.

Abstract

The increasing complexity and performance of integrated Systems on a Chip (SoC) implemented in Deep Sub-Micron (DSM) technologies do impose severe requirements in test setups. A significant number of physical defects in new node technologies require dynamic fault testing, eventually at-speed-testing. Delay Fault Testing, using transition and path delay fault models, can provide a good coverage for delay defects. The transition fault model is widely used in industry, since it leads to manageable test time (proportional to the number of test vectors composing the test pattern). For sequential digital modules, basically two transition fault test pattern generation methods can be applied with scan-based test: Launch-on-Shift (LOS) and Launch-on-Capture (LOC). In LOS, an at-speed transition of the Scan Enable signal is required; however, this feature is not supported in many designs. Most of them only implement a low speed solution and, thus, can only implement the LOC method. Even so, LOS achieves a higher fault coverage with fewer test vectors than the one obtained with LOC.

Digital SoC testing has become a significant problem. External test solutions can become prohibitive, and can only be applied in production test, not during lifetime testing, which may be required for many new products. Built-In Self Test (BIST) is considered an attractive solution, as it can verify a failure-free status autonomously, without any external stimuli. The Scan BIST approach merges the benefits of conventional BIST and Scan-based design. Although leading to higher area overhead than scan design, scan BIST can lead to good fault coverage, with low cost test generation, and it can reduce maintenance costs. Moreover, it can provide a low-cost solution to implement at-speed test.

In this work, a new methodology for SoC dynamic test is proposed. The underlying principle is to combine conventional BIST with dynamic scan design (performing LOS or LOC). The proposed methodology introduces, using a single architecture, three solutions to implement LOC, LOS or both (LOS or LOC can be activated using only one control signal). The proposed methodology and the corresponding design flow are demonstrated using ITC'99 circuit benchmarks. Simulation results show that LOS-based scan BIST leads to higher fault coverage than the LOC-based scan BIST.

Resumo

O aumento da complexidade e desempenho dos Sistemas Integrados inseridos no mesmo chip (SoC, Systems on a Chip) e implementados em tecnologias sub-micrométricas torna o processo de teste de circuitos integrados digitais muito desafiador. Um número significativo de defeitos físicos presentes nas novas tecnologias requer teste dinâmico das faltas, à velocidade do relógio (teste at-speed). O teste de atrasos, através da utilização dos modelos de faltas de transição (TF) e de atraso em caminhos (PDF), permite conseguir uma boa cobertura desses defeitos. O modelo TF é extensamente usado na indústria, uma vez que conduz a tempos de teste aceitáveis (proporcional ao número de vectores de teste do padrão de teste). Para circuitos digitais sequenciais com scan (varrimento) utilizam-se dois métodos de geração de padrões de teste para faltas de transição: Activação no Deslocamento (LOS) e Activação na Captura (LOC). No LOS o sinal Scan Enable exige uma transição at-speed; porém esta característica não é suportada em muitos projectos. A maioria apenas permite a implementação de uma solução mais lenta e, consequentemente, apenas permitem a implementação do LOC. Ainda assim, o LOS conduz a uma cobertura de faltas mais elevada com menos vectores de teste do que o LOC.

As soluções de teste externo são cada vez mais onerosas, se só podem ser aplicadas na produção, e não durante o tempo de vida útil do produto. O Auto-teste Incorporado (BIST) é uma solução muito atractiva, uma vez que permite verificar autonomamente o estado do circuito, sem recurso a estímulos externos. O BIST com scan junta as vantagens do BIST convencional e da técnica de varrimento. Embora conduza a uma área mais elevada do que a técnica de scan, o BIST com scan permite conseguir uma boa cobertura de faltas, eventualmente teste at-speed, com uma geração de teste de baixo custo.

Neste trabalho, propõe-se uma nova metodologia para o teste dinâmico. O princípio subjacente é combinar o BIST com scan, executando o LOS e o LOC. A metodologia introduz, com uma única arquitectura, três soluções que executam o LOS, o LOC ou ambos, podendo seleccionar-se a aplicação de um ou outro com um único sinal de controlo. A metodologia proposta e o correspondente fluxo de projecto são demonstrados recorrendo a alguns circuitos de referência ITC'99. Os resultados de simulação mostram que a arquitectura BIST com scan, activada com o sinal de relógio operacional, baseada no LOS permite conseguir uma cobertura de faltas mais elevada do que aquela que é conseguida pelo BIST com scan LOC.

Keywords

Built-In Self-Test, Fault Coverage, *Delay Test, Transition Faults*, *Launch on Shift*, *Launch on Capture*

Palavras-Chave

Auto-teste incorporado (BIST), *Cobertura de Faltas*, *Faltas de Transição, Teste de Atrasos, Activação no Deslocamento, Activação na Captura*

Contents

Acknowledgments ..... **i**

Abstract ..... **ii**

Resumo ..... **iii**

Keywords ..... **iv**

Palavras-Chave ..... **iv**

Contents ..... **v**

List of Figures ..... **viii**

List of Tables ..... **ix**

List of Acronyms ..... **x**

CHAPTER 1 – INTRODUCTION **1**

1.1. MOTIVATION 1

1.2. WORK OBJECTIVES 6

1.3. DISSERTATION OUTLINE 7

CHAPTER 2 – DESIGN FOR TESTABILITY **8**

2.1. FAULT MODELLING 8

2.1.1. Static Faults (Single Line Stuck-At) *8*

2.1.2. Dynamic Faults *9*

2.1.2.1. Transition Faults 9

2.1.2.2. Path Delay Faults 10

2.2. SCAN DESIGN 11

2.3. BIST (BUILT IN SELF TEST) 12

2.3.1. BIST Methodology *13*

2.3.2. Scan BIST Architecture and Operation *15*

2.3.3. Test Pattern Generation: LFSR (Linear Feedback Shift Register) *17*

2.3.4. Signature Analysis: MISR (Multiple Input Shift Register) *18*

2.3.5. Multiplexer *18*

2.3.6. BIST Controller *18*

2.3.7. Advantages and Disadvantages of Scan BIST *19*

2.4. DELAY FAULT TESTING 20

2.4.1. Enhanced Scan *20*

2.4.2. Launch on Capture *21*

2.4.3. Launch on Shift *22*

2.4.4. Methods Comparison *23*

2.4.5. ATPG for Delay Faults ... *24*

CHAPTER 3 – DYNAMIC BIST METHODOLOGY ... **27**

3.1. DESIGN FLOW ... 27
3.2. METHODOLOGY: SCAN-BASED DYNAMIC BIST ... 29
3.3. SCAN BIST ARCHITECTURE BASED ON LOS ... 31
3.3.1. BIST Controller for LOS ... *31*
3.3.1.1. Implementation of the Finite State Machine (FSM) ... 32
3.3.2. LFSR ... *35*
3.3.3. MISR ... *36*
3.3.4. CUT (Circuit under Test) ... *36*
3.3.5. Multiplexer ... *37*
3.4. SCAN BIST ARCHITECTURE BASED ON LOC ... 38
3.4.1. BIST Controller for LOC ... *38*
3.5. SCAN BIST ARCHITECTURE BASED ON LOS AND LOC ... 40
3.5.1. BIST Controller for LOS and LOC ... *40*

CHAPTER 4 – TEST FLOW AND SIMULATION RESULTS ... **42**

4.1. TRANSITION DELAY FAULT TEST FLOW ... 42
4.2. CHARACTERISTICS OF THE BENCHMARK CIRCUITS UNDER TEST ... 45
4.2.1. B06 ... *45*
4.2.2. B10 ... *45*
4.2.3. B13 ... *46*
4.3. TRANSITION FAULTS SIMULATION ... 46
4.3.1. Fault Coverage for B06 ... *47*
4.3.2. Fault Coverage for B10 ... *48*
4.3.3. Fault Coverage for B13 ... *50*
4.4. ANALYSIS OF RESULTS ... 51

CHAPTER 5 – CONCLUSIONS AND FUTURE WORK ... **55**

5.1. CONCLUSIONS ... 55
5.1. FUTURE WORK ... 58

References ... **60**

ANNEXES ... **62**

VHDL CODE OF SCAN BIST ARCHITECTURE BASED ON LOS AND LOC ... 62
BIST Controller ... *62*
LFSR ... *70*
MISR ... *74*
Top Level and MUX ... *76*
BIST TESTBENCH FOR SCAN CHAIN BEHAVIOUR ... 82

SYNOPSYS TETRAMAX™ TCL SCRIPT – EXTERNAL PATTERNS 85

List of Figures

FIGURE 1 – EXAMPLE CIRCUIT FOR TRANSITION FAULTS 10
FIGURE 2 – IMPLEMENTATION OF SCAN DESIGN-BASED DFT. 12
FIGURE 3 – BASIC BIST ARCHITECTURE. 13
FIGURE 4 – BASIC SCAN BIST ARCHITECTURE 16
FIGURE 5 – LINEAR OR EXTERNAL LFSR. 17
FIGURE 6 – MODULAR OR INTERNAL LFSR. 17
FIGURE 7 – CLASSIFICATION OF DELAY TEST. 20
FIGURE 8 – WAVEFORM FOR CLOCK AND SCAN ENABLE FOR LOC. 22
FIGURE 9 – WAVEFORM FOR CLOCK AND SCAN ENABLE FOR LOS. 22
FIGURE 10 – *SYNOPSYS* FLOW: (A) *SYNOPSYS DFT COMPILER*™ FLOW, (B) *SYNOPSYS TETRAMAX*™ ATPG FLOW... 25
FIGURE 11 – DESIGN FLOW USED IN THE INITIAL STEPS OF THE DESIGN PROCESS (REFERRING THE USED TOOLS)... 27
FIGURE 12 – DYNAMIC BIST ARCHITECTURE PROPOSED IN THIS WORK. 29
FIGURE 13 – INPUT AND OUTPUT BIST CONTROLLER SIGNALS. 31
FIGURE 14 – LOS TESTING STATE MACHINE. 32
FIGURE 15 – CODE FRAGMENT INCLUDING ALL SIGNAL STATES IN LOS IMPLEMENTATION. 33
FIGURE 16 – FUNCTIONAL EXPLANATION OF LOS BEHAVIOUR (*MODELSIM*™). 34
FIGURE 17 – INPUT AND OUTPUT SIGNALS FOR THE TWO USED LFSR: LFSR_PI AND LFSR_SCAN. 35
FIGURE 18 – LINEAR LFSR FEEDBACK IMPLEMENTING THE PRIMITIVE POLYNOMIAL $P(x) = x^4 + x + 1$. 35
FIGURE 19 – INPUT AND OUTPUT MISR SIGNALS. 36
FIGURE 20 – OPTIMIZED B06 CUT WITH SCAN CHAIN. 37
FIGURE 21 – INPUT AND OUTPUT CUT SIGNALS. 37
FIGURE 22 – INPUT AND OUTPUT MUX SIGNALS. 37
FIGURE 23 – LOC TESTING STATE MACHINE. 38
FIGURE 24 – FUNCTIONAL EXPLANATION OF LOC BEHAVIOUR (*MODELSIM*™). 39
FIGURE 25 – CODE FRAGMENT WHERE THE SWITCH BETWEEN LOS AND LOC BASED TEST IS PERFORMED. 40
FIGURE 26 – FUNCTIONAL EXPLANATION OF LOS AND LOC IMPLEMENTATION BEHAVIOUR (*MODELSIM*™). 41
FIGURE 27 – DESIGN FLOW OF THE METHODOLOGY AND EDA TOOLS. 43
FIGURE 28 – TRANSITION FAULT COVERAGE EVOLUTION FOR B06. 47
FIGURE 29 – TRANSITION FAULT COVERAGE EVOLUTION FOR B10. 49
FIGURE 30 – TRANSITION FAULT COVERAGE EVOLUTION FOR B13. 50
FIGURE 31 – COMPARISON OF TRANSITION FAULT COVERAGE ACHIEVED BY DETERMINISTIC TEST PATTERNS FOR THE THREE TESTING METHODS. 52
FIGURE 32 – COMPARISON OF TRANSITION FAULT COVERAGE ACHIEVED BY PSEUDO-RANDOM TEST PATTERNS FOR THE THREE TESTING METHODS. 53
FIGURE 33 – COMPARISON OF TRANSITION FAULT COVERAGE ACHIEVED BY BOTH TEST PATTERNS FOR B06. 54
FIGURE 34 – COMPARISON OF TRANSITION FAULT COVERAGE ACHIEVED BY BOTH TEST PATTERNS FOR B10. 54
FIGURE 35 – COMPARISON OF TRANSITION FAULT COVERAGE ACHIEVED BY BOTH TEST PATTERNS FOR B13. 54

List of Tables

TABLE 1 – CHARACTERISTICS OF THE THREE METHODS TO TEST DELAY FAULTS. 23
TABLE 2 – BIST CONTROLLER SIGNALS. 32
TABLE 3 – TRANSITION FAULT COVERAGE ACHIEVED BY DETERMINIST TEST PATTERNS FOR B06. 48
TABLE 4 – TRANSITION FAULT COVERAGE ACHIEVED BY DETERMINIST TEST PATTERNS FOR B10. 49
TABLE 5 – TRANSITION FAULT COVERAGE ACHIEVED BY DETERMINIST TEST PATTERNS FOR B13. 51

List of Acronyms

ASIC – Application Specific Integrated Circuit

ATE – Automatic Test Equipment

ATPG – Automatic Test Pattern Generation

BIST – Built-In Self-Test

CA – Cellular Automata

CAD – Computer-Aided Design

CC – Capture Cycle

CUT – Circuit under Test

DFT – Design for Testability

DRC – Design Rule Checking

DSM – Deep Sub-Micron (IC technology)

EDA – Electronic Design Automation

FPGA - Field Programmable Gate Array

FSM – Finite State Machine

HDL – Hardware Description Language

IC – Initialization Cycle

I/O – Inputs/Outputs

ITC – International Test Conference

LC – Launch Cycle

LFSR – Linear Feedback Shift Register

LOC – Launch on Capture

LOS – Launch on Shift

LSA – Line Stuck-At

MISR – Multiple Input Signature Register

MUX – Multiplexer

OPC – Optical Proximity Correction

ORA – Output Response Analyzer

PDF – Path Delay Fault (model)

PI – Primary Inputs

PO – Primary Outputs

PRPG – Pseudo-Random Pattern Generator

RTL – Register Transfer Level

SoC – System on a Chip

SPF – STIL Protocol File

STIL – Standard Test Interface Language

TE – Test Effectiveness

TF – Transition Fault (model)

TL – Test Length

TO – Test Overhead

TP – Test Power

TPG – Test Pattern Generator

VHDL – VHSIC Hardware Description Language

VHSIC – Very High Speed Integrated Circuit

CHAPTER 1 – INTRODUCTION

1.1. Motivation

Microelectronics industry profitability is based on Moore's Law, which predicts an exponential decrease in feature size [1]. Hence, after four decades, integrated circuits became SoC (Systems on a Chip) with ever increasing complexity, density and performance. Such trend also enhanced power consumption, leading to the need of low-power design, and sophisticated power and thermal management solutions. In order to constraint internal electric fields (and constraint power consumption), for each node technology a reduction of the power supply voltage level, V_{DD}, is required. Today's technologies (down to 65 nm, V_{DD}=1 V, and hundred million gates operating in the GHz range) use new processing materials and manufacturing processes, like low-K and high-K dielectric materials and OPC (Optical Proximity Correction) [2]. New materials and processing technology lead to new, emerging physical defects. Complexity, performance, power consumption and low pin count bring a difficult challenge: how to specify and run a cost-effective test process [1]. In fact, for digital systems, not only it is difficult to derive a cost-effective test to cover static faults (like those described e.g. by the classic single Line-Stuck-At (LSA) fault model), but also dynamic faults, as a significant set of emerging defects manifest themselves only as time-related defects. Hence, cost-effective solutions to uncover dynamic faults (namely, delay faults) became mandatory [3]. Delay test quests for *two-vector sequences*, the first vector to initialize the circuit, and the second vector to trigger a transition and/or the activation of a Boolean difference through a propagation path to an observable output.

The goal during new product development is, thus, to provide high-quality test in a cost effective way. As a matter of fact, testing is crucial in product life-cycle. First, in the design environment, test is used in design validation i.e., to identify design errors that can cause an incorrect functionality. Second, in the manufacturing environment, test must uncover manufacturing defects, discriminating good from defective parts. In this respect, it is usual to define the product *Defect Level*, as the percentage of defective parts that pass successfully the production test. Typical Defect Level values, for quality products, are today in the range of 10-1 ppm (parts per million). Some defects lead to incorrect Boolean functionality; others lead only to incorrect performance (timing) of the system. Third, during product lifetime, for

many products it is necessary to detect any defects that can produce a failure operation of the system.

Test quality can be measured using various metrics, describing different attributes. Usually, the following attributes (and corresponding metrics) are relevant:

- *Test effectiveness (TE)* – the ability of the test pattern to uncover the likely physical defects which may occur in production, or during product lifetime. TE is usually measured by the *fault coverage*, the percentage of listed faults that are uncovered by the test pattern. This metrics depends on the test pattern and on the fault model used to describe the impact of the physical defect on circuit behaviour.
- *Test Length (TL)* – the number of individual test vectors in the test pattern. This value is relevant, as the test application time is directly proportional to TL and the clock period. It also has impact on test development costs, as the test pattern generation process costs depends on TL.
- *Test Overhead (TO)* – the additional cost associated with the implementation of test functionality built in the SoC. It is usually measured by the percentage Si area overhead, and clock frequency decrease due to test. It is also measured by the increase in pin count.
- *Test Power (TP)* – the power consumption (average and peak) associated with the test sessions. Structural test normally requires high node toggling, leading to high power dissipation in test mode. It is usually measured by the weighted switching activity of the circuit nodes in CMOS [24].

Of these four attributes, the first one – test effectiveness – is the crucial one. Its measure is a necessary condition to proceed with the test process. If TE leads to unacceptably high values of the *Defect Level*, the test pattern must be improved. Therefore, in this work, the attention is focused on TE and on fault coverage evaluation through fault simulation. Comments will be made, when appropriate, regarding the other attributes and their metrics, especially in the final chapters (Results, Conclusions and Further Work).

Device scaling down is not followed by a reduction of Automatic Test Equipment (ATE) specifications and cost. On the contrary, to perform external test with today's SoC, ATE attributes become so demanding (data volume, signal bandwidth and so forth) that its cost is skyrocketing. Consequently, some test cannot be externally performed in a cost-

effective way. It has to be made on-chip. Therefore, Built-In Self-Test (BIST) [4] is an attractive alternative and is becoming more popular as a complementary technique of performing a device or system test. BIST application is expected to increase, and new BIST techniques need to be developed, as new defects need to be uncovered in the self-test process. The BIST methodology proposed in this work is intended as a valuable contribution to self-test delay faults in a cost effective way.

Delay Fault Test is mandatory due to the increasing performance requirements and to the likely physical defects occurring in DSM technologies. High performance also makes the products more sensitive to signal integrity loss, which may affect also the quality of the test process [25]. Several delay fault models have been used to describe the impact of defects on circuit performance [1] [3] [6]. The most common are the Transition Fault (TF) and Path Delay Fault (PDF) Model. In both models, sequences of test patterns are required: a vector pair (T_1, T_2), where T_1 is the initialization vector and T_2 is the vector that launches the appropriated transition to a primary output or to a flip-flop. A transition fault models excessive delay on a single logic node in the circuit. In this work, for the proposed methodology and case study, the Transition Fault model is the used delay fault model. Generally, the TF model is used to capture gross defects whereas the path delay fault model is used for detecting small defects. Thus, in order to achieve high test quality, high transition and path-delay fault coverage are required. Costs of delay test based on the PDF model may become prohibitive as circuit complexity increases, due to the fact that the number of possible paths grows exponentially. Many of these paths are false paths, in the sense that there is no input combination that allows their activation to a primary output. Computational costs to identify and delete these false paths, as well as to generate test sequences for the huge amount of faults can also be prohibitive. Hence, in industry, this delay model is less used.

For sequential circuits, typically with low controllability and observability of the feedback registers, a test mode is included, with *scan design* [1] [4]. In fact, in test mode all feedback registers are reconfigured as one or several shift registers (scan chains), which allow controllability (scan-in) and observability (scan-out) of the register's states. Using scan design, the three most common TF pattern generation methods are Enhanced Scan, Launch-on-Capture (LOC) and Launch-on-Shift (LOS) [3] [6]. Basically, they differ on the way of applying the second vector in each sequence. In LOC, the second vector is the Circuit under Test (CUT) response to the first vector applied. In LOS, the second vector is a shifted version of the first one. Yet, the most important difference between the two methods is associated

with the *Scan Enable* signal: in LOS it must switch *at-speed*. Consequently, this method is difficult to implement. However, it leads to higher fault coverage than LOC with much less vectors applied and a reduced test application cost.

As a consequence, recently new architectures have been proposed that allow a LOC fault coverage improvement and an easier LOS implementation. The first one, named *Enhanced Launch-off-Capture*, uses a new cell inserted in the scan chain to generate local *Scan Enable* signals [12] [13], which control the transition launch path. This approach provides better controllability than the traditional LOC and affords higher transition fault coverage. The second proposed architecture is a solution for implementing LOS tests by adding a small amount of logic in each flip-flop to align the slow scan enable signal to the clock edge [14] [15]. These last papers also present results for LOS and LOC test implementation, which are better than those afforded by LOC and LOS. However, in these proposed architectures, the area overhead is very large, because a significant amount of hardware is inserted for each flip-flop, to guarantee a better fault coverage.

To face the testing problem, BIST is hence an attractive solution. Test and diagnosis must be fast and have high fault coverage. In a BIST approach, test pattern generation and output response analysis are integrated on-chip, without the need of externally applied test. Low hardware Test Pattern Generators (TPG) can be built with pseudo-random generators, like Linear Feedback Shift Registers (LFSR) or Cellular Automata (CA) [1]. Output response analysers (also referred as signature analysers) can be implemented in a cost-effective way using Multiple Input Signature Registers (MISR). For sequential circuits, in self-test, *scan* can be associated, in what is referred as Scan BIST [4]. In this work, the scan BIST architecture implemented is a *test-per-scan*, *embedded* BIST architecture where the scan chain is built using existing flip-flops. Scan BIST combines the advantages of a basic BIST approach and Scan-based design to obtain higher fault coverage, with low cost test generation, maintenance and system test. Scan BIST may be operated *at-speed*. The limitations of the architecture are related to the area, performance and pin overhead.

In the past, BIST techniques have been developed for the detection of static faults, such as single LSA, or bridging faults. Delay test has been developed for scan-based external test, not for BIST. Test effectiveness with pseudo-random test vectors is limited for uncovering static faults. It is expected that they will perform even worse when targeting the coverage of dynamic faults, as the cumulative probability of generating an adequate *sequence* of two test

vectors to activate and propagate these faults is low. This can be one reason not to perform delay test based on Scan BIST.

In the literature, many solutions have been presented to enhance test effectiveness of BIST solutions for static faults: weighted pseudo-random TPG [26], LFSR reseeding [20], and deterministic BIST [23], combining PR and deterministic test vectors. The injection of deterministic test vectors may be performed by bit-flipping **Error! Reference source not found.** or bit-fixing [22] techniques.

More recently, a new technique was proposed and combines a BIST structure with the standard scan design to target delay faults [28]. The idea is to use a new TPG in a BIST architecture that combines a pseudo-random pattern generator with a scan shift register, to generate test vectors to detect path delay faults in a sequential circuit. Although this scan-BIST structure requires a short test application time (comparable with the double Flip-flop solution), it requires a significant area overhead. Another technique presented in [29] tries to minimize the over-testing problem of logic BIST for delay and crosstalk-induced failures, but the use of a monitor to filter out in real-time the non-functional patterns increases the complexity of the test structure implementation. In [30] is presented a deterministic logic BIST for transition fault testing. This approach is based on the Deterministic logic BIST applied to stuck-at faults and uses a bit-flipping scheme adapted to transition fault testing based on functional justification. The DLBIST can be applied to a standard BIST structure to improve fault coverage (for stuck-at faults and for transition faults) to reduce test times at the expense of a more complex BIST controller architecture and additional area overhead.

Nevertheless, scan BIST solutions need to be devised for covering dynamic faults, namely delay faults. As scan-based test using enhanced scan, LOS or LOC have been applied widely in industry, it may be rewarding to derive novel scan BIST solutions adapting dynamic scan to BIST. In this work, I try to develop a new methodology and several architectural solutions for this problem, merging dynamic scan with scan BIST and the results, as I will show, are promising. TE of pseudo-random TPG to uncover transition faults, and path delay faults, is also investigated.

1.2. Work Objectives

The aim of this work is to propose a novel at-speed Scan BIST methodology targeting the coverage of delay faults. In the proposed methodology, two scan design techniques for delay fault coverage (LOC and LOS) are used in a self-test environment. During this work three architectures have been derived: a Scan BIST based on LOC, a scan BIST based on LOS and a hybrid solution that merges the two techniques and allow the implementation of the two techniques, LOC or LOS, by changing only the value of one input variable.

The Scan BIST structure is the same for the three developed approaches. The CUT, with scan insertion is surrounded by an LFSR, MISR, multiplexers and a BIST Controller. In this work, the CUT used to demonstrate the usefulness of the methodology are ITC'99 circuit benchmarks [18], with an inserted single scan chain. The proposed methodology is modular, as the same architecture can be used for the three possible implementations. The difference among the three architectures is clustered in the BIST Controller, which is the only one that exhibits different functionality in each case.

The fault coverage, based on transition fault coverage, obtained with the dynamic scan BIST architecture based on LOS is higher than the one that the LOC approach can achieve. That is to be expected. Moreover, the new architectures exhibit area overhead, which is similar to the one in the Scan based on LOC and Scan based on LOS, suffering an improvement of about 1.2% related to those in the Scan based on LOS and LOC, as well as the pin overhead. This is a consequence of BIST implementation. Fault coverage results must be generated for deterministic and pseudo-random test patterns, in order to distinguish the limitations of the proposed methodology due to architectural reasons, and due to TPG quality. The fault coverage results with pseudo-random test patterns, based on transition fault coverage, are not very high, as expected; however, as I will show, they behave reasonably well for LOC. Different TPG may be used. For instance, if the pseudo-random patterns used have polynomial temporal correlation, the fault coverage would be lower.

1.3. Dissertation Outline

This dissertation is organized as follows:

- Chapter 2 outlines the basic definitions and terminology used. It reviews basic concepts on Fault Modelling, conventional BIST methodology and its architecture, and on the Scan design approach. Emphasis is put on scan design for delay fault detection, namely on Enhanced Scan, Launch on Capture and Launch on Shift techniques.
- Chapter 3 describes the proposed Dynamic BIST methodology. It gives the details about the new methodology, the proposed BIST architectures and the characteristics of all their structural components.
- Chapter 4 presents the design flow, taking into account the proposed methodology. It also describes the benchmark circuits used to demonstrate the usefulness of the methodology and presents the most relevant simulation results, based on the transition fault model. At the end of the chapter, a discussion of the results and their significance is presented.
- Chapter 5 concludes the work with a summary of the proposed methodology, its achievements and limitations. It also outlines directions for future work.

CHAPTER 2 – DESIGN FOR TESTABILITY

This Chapter briefly reviews the state of art for this work. First, the details about the commonly used Fault Models (for static and dynamic faults) are given. Next, two of the most important techniques for static and structured test, namely the Scan Design technique and the Built-In Self-Test (BIST), are presented. Finally, three different approaches for Scan-based Delay Testing are reviewed.

2.1. Fault Modelling

Physical failures and fabrication defects are difficult to be modelled mathematically. What is really important is to model their impact on circuit behaviour. Moreover, the abstraction level in which they are modelled is crucial, as it severely impacts the costs of Automatic Test Pattern Generation (ATPG) and fault simulation, in the design environment. Hence, for digital systems, they are modelled as logical faults, or at higher abstraction levels, like RTL (Register Transfer Level). The presence of a Delay Faults results in a circuit whose operation is logically correct but does not perform at the required operating frequency [4]. The goal of the Delay Fault Model is to guide the ATPG process (if deterministic test generation is performed) and to uncover the defects that may exist in a manufactured device. The most common Delay Faults models are the Transition Fault (TF) and Path Delay Fault (PDF).

2.1.1. Static Faults (Single Line Stuck-At)

It has been shown that stuck-at fault tests are effective in capturing a wide range of defects on manufactured chips. This model represents defects inducing opens, shorts with power and ground lines, and other internal defects in the components. This fault is modelled by assigning a fixed (0 or 1) value to a signal line in the circuit. A signal line is a line in the logic-level description of the circuit, i.e., an input or an output of a logic gate or a flip-flop [1].The most commons are the single stuck-at faults: *the stuck-at-1* and the *stuck-at-0*. This fault model assumes that only one fault can exist in the circuit at a time [4].

To generate stuck-at test vectors, two steps are needed: the first step is to generate a partially specified test vector that activates the fault and the second one is to complete the test stimuli, in such a way that the fault effect (a Boolean Difference between the fault-free and the faulty CUT) is propagated to a scan flip-flop or a primary output. Test vectors are typically generated by Automatic Test Pattern Generation (ATPG) tools. It is relatively easy to generate patterns for stuck-at faults and the corresponding Test Length (TL) is considerably low.

2.1.2. Dynamic Faults

2.1.2.1. Transition Faults

The Transition Fault Model is similar to the Stuck-at Fault Model in some aspects. The gate delay, increased over the nominal value in the presence of a defect, is assumed to be large enough to prevent an input transition from reaching any output [1]. The consequence of a transition fault at a given node in a circuit is that a rising or a falling transition at this node will not reach a scan flip-flop or a primary output within the desired time (a clock period). Because of the nature of these faults, the possible transitions faults of a gate are *slow-to-rise* and *slow-to-fall* types [1]. As a result, the total number of transitions is twice the number of gates[1]. A *slow-to-rise* fault means that any input from 0 to 1 does not produce the correct logic value when the device is operating at the desired frequency. Comparably, a *slow-to-fall* fault means that an input from 1 to 0 does not also produce the correct result.

In a circuit, the *time slack* is defined as the difference between the time allowed for signal propagation (the clock period) and the time required to propagate a transition through the critical path of the Circuit under Test (CUT) [5]. For a transition fault to cause an incorrect value at a circuit output it is necessary that the size of the delay fault be such that it exceeds the *slack* of at least one path from the fault location to a primary output or scan flip-flop. If the propagation delays of all paths passing through the fault place exceed the clock period, such fault is also called *gross-delay fault* [1] [7].

[1]In the LSA fault model, the number of listed faults is also twice the number of logic nodes, N. Fortunately, many equivalent and dominant faults allow fault collapsing, in such a way that the relevant LSA list is much smaller that 2N.

Any test pattern that can detect a transition fault is based on stuck-at procedures and requires two vectors, T_1 and T_2 [6]. T_1 is used to initialise the circuit and set up a sensitive path. T_2 is the subsequent vector used to cause an input change and propagate the effect of the transition to a primary output or scan flip-flop within a specified time period.

Consider the circuit shown in Figure 1. It has two NAND gates, one primary input and two observation points. By the application of the displayed vector pair, it can be observed that there is a rising transition at nodes *x* and *y*. These transitions are propagated to the primary outputs and the result is two falling transitions detecting a total of four transition faults, i.e., two rising transition faults (at *x* and *y*) and two falling transition faults (at *w* and *z*).

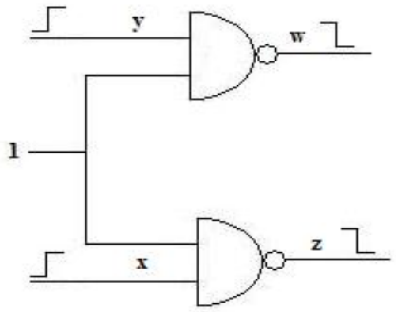


Figure 1 – Example circuit for transition faults

The transition faults are related to single delay faults that have to do to local (random) defects. These types of faults can be caused by threshold voltage shifts, low transistor path conductance, narrow interconnect lines, spot defects, open (or resistive) vias, IR drop on power supply lines, and crosstalk, among others [6].

2.1.2.2. Path Delay Faults

Path Delay Faults causes the cumulative propagation delay of a combinational path to increase beyond some specified time duration [1]. A physical path is an interconnection of gates from an input (primary input or scan flip-flop) to an observation point (primary output or scan flip-flop). The *path length* is defined as the number of gates that a given path contains.

This fault model is used to detect the presence of a sum of excessive small delays along a path. The Delay Faults due to process variations [6] can manifest as small delays which individually do not make the circuit faulty. Since the extra delay in each gate is small, the transition fault model may not detect such faults. As a result, path delay faults are used to

detect an error on the specified paths. For each path there are two path delay faults, which correspond to the *rising* and *falling* transitions, respectively [1].

Usually the number of paths is large; hence, the selection of paths is critical in test pattern generation. However, the major difficulty is to know when a test fails [6] and becomes necessary the use of statistical processes and accurate simulations.

Path Delay Faults may be caused by mask misalignment, line registration errors or even an omitted process stage [6]. They are based on global defects affecting all elements in a path. Usually, the critical path is selected by static timing analysis (more realistic) because a large number of problems are related to false paths.

2.2. Scan Design

Design for Testability (DFT) refers to the design techniques that make test generation and application cost-effective [8]. The underlying principle is to influence design from the start, in order to guarantee that the final implementation is more testable. Scan Design is a structured method of DFT approach for digital circuits. The main idea in this technique is to obtain control and observability for flip-flops [1]. This is obtained by adding a *test mode* (also known as *scan mode*) such that when the circuit is in this mode the flip-flops are reconfigured into one or more shift registers. The inputs and outputs of these shift registers (known as scan registers) are made into primary inputs and primary outputs [1]. The transformation of a sequential logic circuit to a scan design is represented in Figure 2. One input of the multiplexer is connected to the normal system logic and the other is connected to the output of another scan flip-flop. The mode of operation is controlled by the *select* input to the multiplexer: either system mode or scan mode. The *Scan In* input to the multiplexer of the first flip-flop in the shift register chain is connected to the primary input to scan in test data. Similarly, the output of the last flip-flop in the chain is connected to a primary output to *Scan Out* test results [4]. Therefore, all flip-flops in the scan chain are easily controllable and observable, which also offer good controllability and observability of the embedded combinational logic of the CUT.

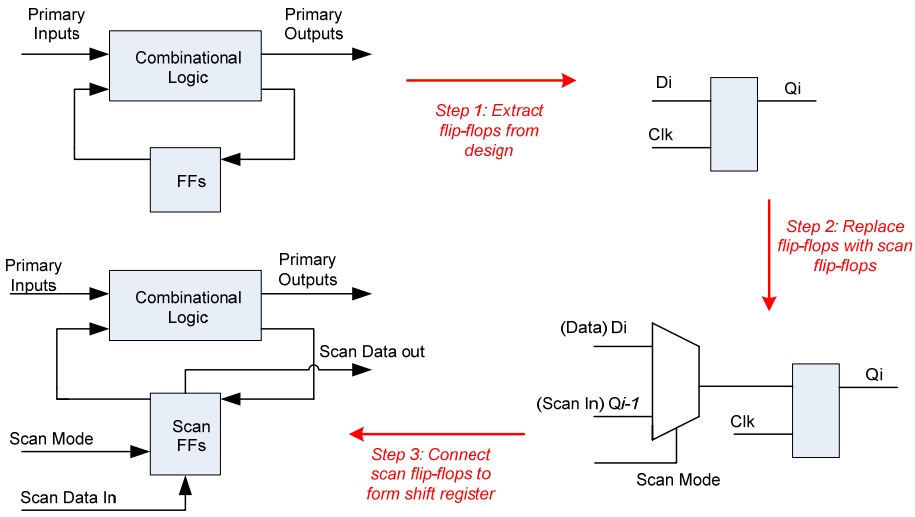


Figure 2 – Implementation of scan design-based DFT.

Nowadays, with all the available Computer-aided Design (CAD) tools, scan design can be entirely automated, including scan flip-flops insertion, interconnection of the scan chain, generation of the test vectors and the analysis of the fault coverage. At the same time, with this test method, high fault coverage can be achieved, using a moderate increase in area and decrease in speed. It also has the advantage of reducing the effort of test generation, as it basically allows keeping ATPG in the domain of combinational logic [1]. As disadvantages, this technique needs a large test data volume and a long test time (the number of test vectors is proportional to the size of the scan chain), which means that there is a low speed test [8]. Moreover, a significant amount of power is required in the scan-in and scan-out operations, especially when long scan chains are needed. However, scan design is the most popular technique and is probably the best overall DFT approach ever developed [4].

2.3. BIST (Built In Self Test)

This section is intended to provide the necessary background to understand the advantages and disadvantages of BIST, as well as the implementation of the various BIST architectures.

2.3.1. BIST Methodology

During its lifetime, a digital system is tested on numerous occasions. Test and diagnosis must be fast (low TL) and lead to high fault coverage. With self-test, test is specified as part of the system functionality. BIST methodology incorporates test pattern generation and output response analysis capabilities on chip [4]. The basic BIST architecture is shown in Figure 3. This architecture is very efficient when the CUT is combinational. If the CUT is a sequential circuit, it is necessary to partition it in a combinational CUT, and one or more scan chains in self-test mode, in order to increase the observability to the internal nodes.

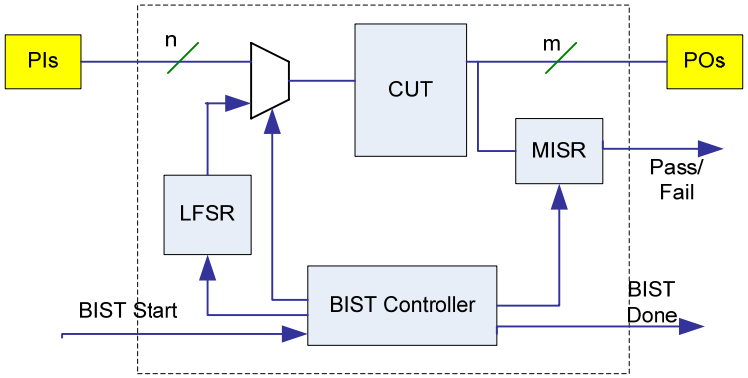


Figure 3 – Basic BIST architecture.

The two most common BIST approaches are the Random Logic BIST (also known as Scan BIST) and the Memory BIST [1]. The first one is based on the addition of a Pseudo-Random Pattern Generator (PRPG) [9] to the primary and to the scan inputs and the addition of a Multiple Input Signature Register (MISR) to the primary and scan outputs. This is shown in Figure 3. Usually the PRPG is implemented using a Linear Feedback Shift Register (LFSR). The BIST Controller generates the necessary clock pulses to load the pseudo-random patterns into the scan chain. All the responses are captured by the MISR, which compacts the circuit responses into a *signature*. Any signature (a specific digital word) different from the one of the good machine indicates a faulty circuit [1]. The integration of BIST also requires additional I/O pins for activating the BIST sequence (the *BIST Start*), for reporting the results (the *Pass/Fail*) and an optional indication that the BIST session is complete and that the results are valid (the *BIST Done*).

The disadvantages of BIST are related to (1) chip area overhead, since a significant amount of hardware is added (a BIST Controller, TPG, ORA (Output Response Analyzer) and multiplexer), to (2) I/O pin overhead, because at least one additional pin (*BIST Start*) is needed to activate the BIST operation, and to (3) performance penalties inherent to the extra

path delays. At the same time, due to increased area, the reliability is reduced. When the BIST hardware is made testable, that also may increase its complexity. On the other hand, the BIST benefits are lower test generation cost, the reduced testing and maintenance cost, the lower cost Automatic Test Equipment (ATE) required to test the product parts which have no BIST, the lower test time[2], and the ability of enabling *at-speed testing* [8].

There are several general classifications of BIST architectures that are used to describe BIST approaches [4]. BIST circuitry is normally referred as *centralized* or *distributed*. The centralized type of architecture can be effective in reducing area overhead. However, it requires multiple executions of the BIST sequence. In the distributed type, each CUT has it own Test Pattern Generator (TPG) and Output Response Analyzer (ORA) such that BIST of each individual CUT can be executed in parallel. Often, power consumption limitations demand careful test scheduling, the test of each module being launched in consecutive time slots, to avoid overheating in test mode.

When the TPG and ORA are implemented using existing flip-flops and registers of the CUT, the BIST architecture is referred to as *embedded* [4]. If the TPG and ORA functions are independent of the CUT, the architecture is called *separate*.

Another classification of BIST has to do with the state of the system during BIST operation [4]: *on-line* and *off-line*. On-line BIST implies that the system is performing in its normal mode of operation, while test is performed (a kind of watchdog process). In off-line BIST, the system is out-of-service during the self-test process, leading eventually to the detection of a fault via concurrent fault detection circuit or for periodic testing to determine if the system is fault-free. Many products exhibit latency of operation that can easily accommodate self-test for limited periods of time.

In the on-line BIST, the normal mode of operation includes the test mode. As a consequence, this type is difficult to implement and is expensive. By other side, in off-line BIST, the CUT is tested while it is not in the normal mode of operation. This is easy to implement and more cost-effective. For most applications, fault latency may be acceptable, and hence off-line BIST is the preferred solution. I refer *fault latency* as the time between fault occurrence and fault detection. In this work, off-line BIST is considered.

---

[2] Scan test requires long test sequences (due to the shift-in and out operations) and is performed at low speed. BIST, with pseudo-random test vectors, may require a large number of vectors to reach acceptable fault coverage, but may be applied at speed. Ultimately, test power (i.e., the power consumption in the test process) will determine test length and speed in the two techniques**Error! Reference source not found.**

One final classification of BIST architectures is based on the application of test patterns to the CUT [1]. In a *test-per-clock* solution, a new test vector is applied with each clock cycle. The advantage of this BIST approach is that it has a shortest fault simulation time. However, it is hard to implement. On the other hand, the BIST architecture that uses one or more scan chains as the basic mechanism to deliver test patterns to the CUT as well as to recover the output responses from the circuit is referred as *test-per-scan*. This scan BIST solution takes more time (i.e., more clock cycles) than a *test-per-clock* to detect the same number of faults in a given circuit, which significantly increases fault simulation costs[3]. The advantage of *test-per-scan* systems is related to the hardware savings in the MISR. In the present work, *test-per-scan* is selected as the most appropriate solution for dynamic BIST. Consequently, it is important to emphasize one problem in *test-per-scan* systems: usually, the input patterns are generated using a pseudo-random technique. However, as the patterns are time shifted and repeated to the circuit through the scan chain, those become correlated. As a result, the pattern effectiveness for fault detection is reduced. Usually to solve this problem it is necessary to use an input network of XOR gates to de-correlate the input patterns.

2.3.2. Scan BIST Architecture and Operation

The goal of creating a BIST architecture based on the scan design is to incorporate a TPG in the form of an LFSR in the *Scan In* input of the scan chain and an ORA in the form of a MISR in the *Scan Out* output of the scan chain [4]. This approach is presented in Figure 4. For this scan-based architecture system input isolation is required, as well as the capacity of applying test patterns to the primary inputs and data compaction to the primary outputs. That function is guaranteed by the BIST Controller, which provides the *Scan Mode* control to switch the scan flip-flops between system mode (to apply test patterns and recover the output responses) and shift mode (to shift in the test patterns from the LFSR, and shift out the response to the MISR). At the same time, this must disable the output response compaction until valid output responses are available in the primary outputs and in the *Scan Out* output.

As a traditional BIST approach, the scan BIST sessions begins with the activation of the *BIST Start*. At this instant, the BIST Controller initializes the LFSR and the MISR. After that,

[3] If fault simulation costs become prohibitive, hardware fault emulation, using reconfigurable devices, like FPGA (Field Programmable Gate Arrays) may prove to be cost-effective [27].

the LFSR begins to generate pseudo-random test patterns that are shifted into the scan chain. When this chain is totally filled, the BIST Controller changes the control of the scan chain to system mode during a clock cycle to apply the test patterns to the CUT. At the same time, the test patterns are applied to the primary inputs by the LFSR. The CUT responses are clocked into the scan chain during this clock cycle. After that, the BIST Controller enables the MISR to start output response compaction and switches back the scan chain to shift mode. As the output responses is loaded back into the scan chain and shifted out for compaction, the next pattern is shifted into the scan chain [10]. That process is repeated until the end of the BIST sequence. When this is completed, the MISR is disabled until the resultant signature can be read for *pass/fail* determination of BIST.

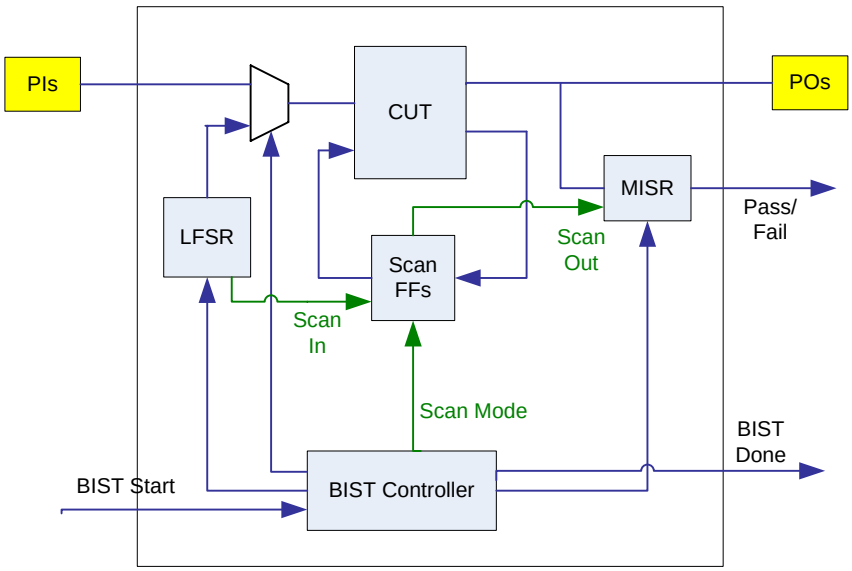


Figure 4 – Basic Scan BIST architecture

The length of the BIST session (Test Length, TL) is given by the number of scan patterns needed to obtain the desired fault coverage. Usually, for the scan BIST a higher number of patterns are necessary (approximately ten times more) to achieve comparable fault coverage than the one required with deterministic patterns generated for the CUT using CAD tools (ATPG).

This scan BIST is a *test-per-scan*, *embedded* BIST architecture where the scan chain is created from existing flip-flops. That is also the architecture implemented in the present work.

2.3.3. Test Pattern Generation: LFSR (Linear Feedback Shift Register)

One of most important components in a BIST structure is the TPG. The fault coverage obtained is a direct function of the test patterns generated by the TPG. There are several types of test patterns that can be used in BIST: deterministic, algorithmic, exhaustive, pseudo-exhaustive or even random. However, due to hardware costs, the most commonly used are the pseudo-random test patterns. The primary hardware for generating on-chip these test patterns is the LFSR [4]. The patterns generated by an LFSR have all the desirable properties of random numbers, but are generated by a TPG and are repeatable, which is essential for BIST [1].

One of the reasons LFSR to be the most ordinarily used TPG is that requires less combinational logic per flip-flop [4].

There are two basic types of LFSR implementations: an *external* or *linear* LFSR and an *internal* or *modular* LFSR [1]. The first one is depicted in Figure 5 and the second in Figure 6. In terms of hardware, both implementations require the same amount of logic, as far as flip-flops and exclusive-OR gates are concerned [4]. However, the modular LFSR provides the implementation with the highest maximum operating frequency, due to the fact that it has, at most, one exclusive-OR gate in any path between flip-flops. On the other hand, linear LFSR, that has two exclusive-OR gates, in the worst case, between the output of the last flip-flop and the input of the first one, has the benefit of the uniformity of the shift register. In view of this attribute, sometimes this is the preferred.

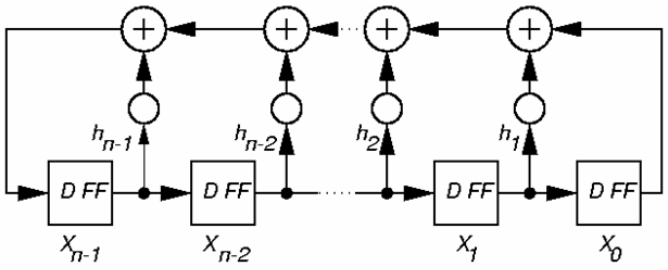


Figure 5 – Linear or external LFSR.

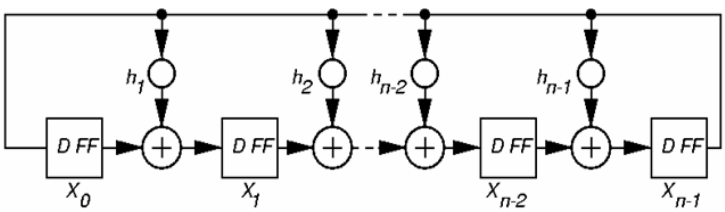


Figure 6 – Modular or internal LFSR.

Different placement of the exclusive-OR gates in the feedback chain result different sequences of test patterns produced at the outputs of the flip-flops in an LFSR. That placement is defined as the *characteristic polynomial* of the LFSR. One common way of generating test patterns is to apply at *primitive polynomials* (there are polynomials that result in a maximum length sequence). In an LFSR with N registers, if it implements a primitive polynomial, it is able to generate a sequence of (2^N-1) test vectors. For example, for N=20, an extremely long sequence is generated. Typically, only a window of $M << (2^N-1)$ is used in a BIST session. In this work that approach is used to generate the pseudo-random test patterns.

2.3.4. Signature Analysis: MISR (Multiple Input Shift Register)

During BIST, it is necessary to reduce the enormous number of circuit responses [1]. In most ORA techniques those responses are compacted into a *signature* that is compared with the expected signature for the fault-free circuit. The Signature Analysis is the most commonly used technique for ORA in BIST implementations [4]. This method uses an LFSR as the basic component. This LFSR is different from the one used in TPG, since it needs an input data, which in this case is the output response of the CUT. That response can be represented by a polynomial. The basic idea is to divide that polynomial by the characteristic polynomial of the LFSR. The remainder of this division is the signature used to determine the status (faulty or not) of the CUT and the end of the BIST sequence.

The solution used is referred as MISR because it can compact multiple outputs into a single LFSR [8].

2.3.5. Multiplexer

This component selects the inputs to be applied to the CUT during BIST: in the normal system operation the primary inputs are directly applied to the CUT; during the shift operation the test patterns generated by the LFSR are applied to circuit.

2.3.6. BIST Controller

As the name suggests, this FSM (Finite State Machine) controls all the BIST operation. When the *BIST Start* is activated, the BIST Controller (1) initializes the LFSR and the MISR,

(2) isolates the primary inputs by selecting the alternative inputs to the input isolation multiplexers and (3) drives the scan chain into the shift mode [4].This can be observed in Figure 4.

Every time the scan chain changes the operation mode, this is triggered by the BIST Controller. At the same time, all the MISR activity is also controlled by this FSM. To begin the output response compaction, the BIST Controller has to enable the MISR. When the BIST session is completed, the MISR is disabled and the *BIST Done* is activated.

2.3.7. Advantages and Disadvantages of Scan BIST

As the basic BIST approach, the Scan BIST has advantages and disadvantages of being implemented. The main benefits of this architecture are associated to the advantages of implement a scan design-based DFT [4]. Those benefits include low area overhead, performance penalties and the high level of automatic synthesis of scan chain. The area overhead is obtained by the addition of flip-flops, exclusive-OR gates, multiplexers and other logic gates needed to implement each component. The performance penalty is basically related to the multiplexer delay at the input of each flip-flop. Although that knew increase, the scan BIST architecture own is as an advantage due to the fact that scan design can be used to lower area overhead and to avoid performance penalties in critical timing paths. Hence, that increase in area and performance is not so high in comparison to the one associated with the basic BIST approach. The high level of automation synthesis of scan chain and the easiness to implement scan design using CAD tools are key benefits of that architecture. At the same time, this architecture has also the advantage of enabling *at-speed* test, as well as reducing the cost of maintenance and test generation. Moreover, it requires less system test and diagnosis [8].

Reaching higher fault coverage is a goal is this architecture, but in this case it becomes a disadvantage. By adding an LFSR in the beginning of the scan chain and a MISR at its end, it is not enough to improve the fault coverage. As a result, the increase in hardware does not necessarily represent an improvement in fault coverage as expected. Despite these limitations, Scan BIST is one of the most popular BIST approaches.

2.4. Delay Fault Testing

As mentioned before, delay tests require a vector pair to detect a fault. Since the patterns must be applied at the rate speed, at-speed testing is needed. For full-scan circuits[4], both vectors in the scan flip-flops must be ready for consecutive time frames to ensure at-speed test. The three most common approaches to apply deterministic at-speed test are the Enhanced Scan, Launch on Capture (LOC) and Launch on Shift (LOS) (Figure 7). Basically, the three approaches differ in the way of storing and applying the second vector of each vector pair.

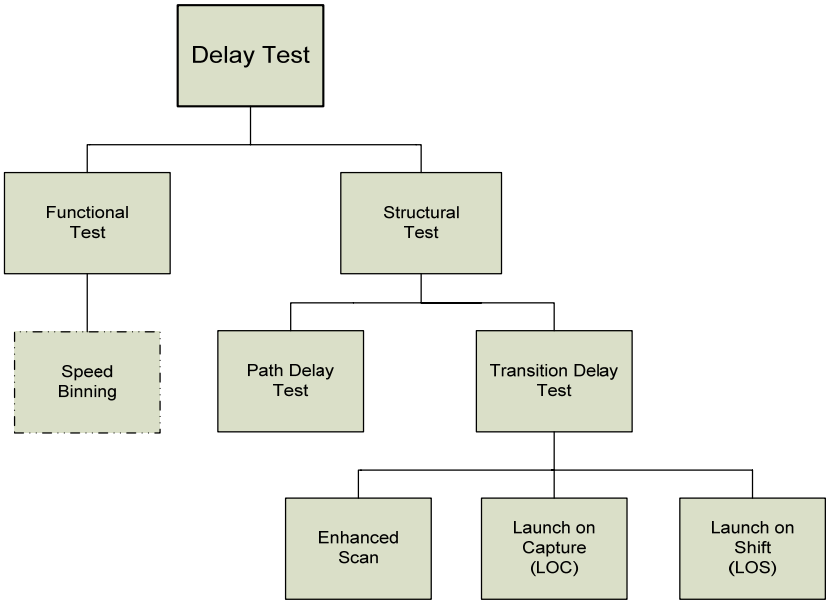


Figure 7 – Classification of Delay Test.

2.4.1. Enhanced Scan

In this method both vectors (T_1 and T_2) are shifted in during the shift process to the scan flip-flops. During the shift of T_2 it is assumed that the initialization of T_1 is not destroyed. Therefore, special scan flip-flops are needed; these are referred as *hold-scan flip-flops* [11], to

[4] In *full scan* circuits, all registers are reconfigured in test mode as shift registers. Other approaches advocate *partial scan*, in which key registers are included in the chain(s), namely those with low controllability and/or observability in the normal mode of operation [1].

store the two values at the same time. After both vectors are shifted in, the scan-enables go low and the clock is pulsed two times. Then the response is shifted out.

This method has two main benefits: it can achieve higher fault coverage (since both vectors are controllable) and the test data volume is limited. However, this technique has also an important disadvantage: the higher area overhead related to the need to implement the hold-scan flip-flops.

In this case, the two vectors (T_1, T_2) are independent. In the next two techniques (LOC and LOS) the second vector to be applied is derived from the first one.

2.4.2. Launch on Capture

As previously mentioned, to perform a transition fault test it is necessary to apply to the CUT a vector pair (T_1, T_2). Vector T_1 is the initialization vector and T_2 the launch vector. The response of the CUT to T_2 must be captured at functional speed (rated clock period). The whole operation can be divided into 3 cycles: the first one is the Initialization Cycle (IC), where T_1 is applied and the CUT is initialized; the second one is the Launch Cycle (LC), where T_2 is applied and a transition is launched at a gate terminal; the third one is the Capture Cycle (CC), where the transition is propagated and captured at an observation point [12] [13].

Launch on Capture is the most common form of delay fault application method [7] [14] [15]. That requires only one vector to be shifted in during the shift cycle. The initialization vector of the vector pair (T_1, T_2) is first loaded into scan chain by *n* consecutive *scan shift operations*, where *n* is the number of scan flip-flops in the scan chain. Vector T_2 is the circuit response obtained by the application of the first vector.

Hence, the scan flip-flops are changed into the normal mode of operation by lowering the *Scan Enable* signal before the LC. As a result, in that method the LC is separated from the shift operation (it is shown in Figure 8). At the end of the shift mode, T_1 is applied and the CUT is set to an initialized state. After that, a pair of at-speed clock pulses is applied to launch and capture the transition response [12]. Consequently, this approach does not require at-speed transition of the *Scan Enable* signal. Hence, it can be implemented with low hardware overhead. This method is a cost-effective solution. However, it has the inconvenient of achieve lower fault coverage (using the same number of patterns as the Launch-on-Shift). That is a consequence of having a second vector that is the response to the first one applied.

In this manner the number of second vectors that can be applied is limited, what would reduce the fault coverage.

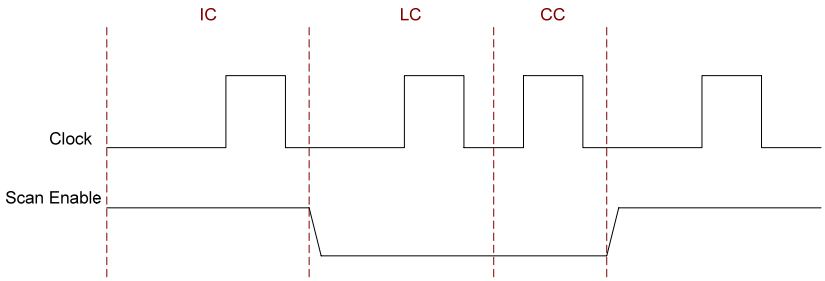


Figure 8 – Waveform for Clock and Scan Enable for LOC.

2.4.3. Launch on Shift

As the name specifies, this method uses for each vector pair a shifted version of the first vector as its second vector. So, T_2 is no more the response of the circuit. The second vector is obtained by shifting in T_1, which is loaded into the scan chain, by one more scan flip-flop and scanning in a new value into the scan chain input. In that approach the *Scan Enable* signal is high during the last shift (or LC) and must go low to enable response capture at the CC [12]. That is presented in the diagram of Figure 9. As it can be observed, the *Scan Enable* signal is high during the last shift and must go low to enable capture response during CC clock edge. That is the main disadvantage of this method, because it is very difficult to make a switch precisely between the two clocks. Hence, the design and implementation of that approach is too expensive and result in a longer design time. Nevertheless, LOS is the preferable solution because it can achieve higher fault coverage within significantly fewer test vectors [14].

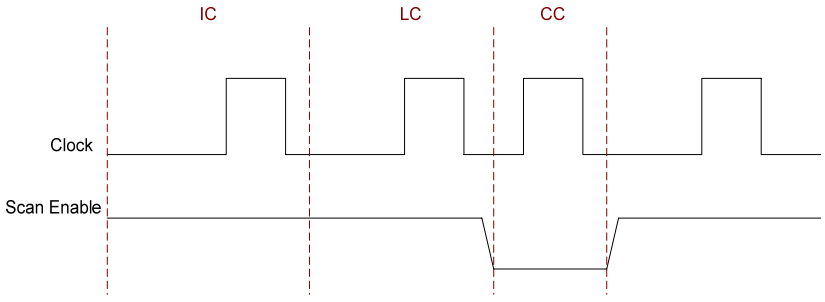


Figure 9 – Waveform for Clock and Scan Enable for LOS.

2.4.4. Methods Comparison

The three techniques can be compared through the advantages and disadvantages of each of them [6].

The fault coverage of LOC is the lowest between the three approaches. Since there are no dependencies between the two applied vectors, the best results are achieved by Enhanced Scan, even better than the LOS transition test. However, due to the use of special scan flip-flops (hold-scan flip-flops) that architecture has the highest area overhead, while LOS and LOC have lower overhead, which are comparable between them. Nevertheless, the Enhanced Scan has another advantage: the test data volume is the lowest of the three approaches. In the opposite site is the LOC, which needs more (some times ten times more) test patterns to lead to good fault coverage results. That characteristic increases the ATPG computational costs. Hence, LOC requires sequential ATPG, which is far more complex and requires the use more test vectors, as compared to the number of test vectors required by the other two approaches, which use combinational ATPG. Consequently, ATPG time is the highest in the LOC approach.

The advantages and disadvantages of the three techniques are summarized in Table 1.

	Enhanced Scan	***LOS***	***LOC***
Fault Coverage	Highest	Higher	Lowest
Hardware overhead	Highest	Higher	Lowest
Test size	Smallest	Smaller	Largest
ATPG Complexity	Lowest	Higher	Highest
ATPG Time	Shortest	Shorter	Longest
Scan Cell Type	Hold-Scan	Standard	Standard

Table 1 – Characteristics of the three methods to test Delay Faults.

Enhanced Scan is not an attractive technique to implement due to the higher area overhead and the need of special hardware. The two most common approaches are LOC and LOS. This last one presented is preferable based on the ATPG complexity and the fewer number of test vectors required to reach higher fault coverage. However, it requires a fast *Scan Enable*, which is not supported by most designs [15]. Thus, although all the advantages over LOC, that one is the best choice of scan-based test method in many cases, due to the difficulty in meeting the design requirements of LOS. Yet, this technique incurs in significantly higher test cost and results in lower test quality.

In order to select the most adequate solution, all design trade-offs have to be analysed, namely the trade-off between hardware and fault coverage.

2.4.5. ATPG for Delay Faults

To perform ATPG for any circuit it is necessary use, at least, two different CAD tools. In the *Synopsys* CAD environment, used in this work, these tools are: *Synopsys DFT Compiler*™ and *Synopsys TetraMax*™. The first is used for scan chain insertion in the design. The second software tool is a commercial ATPG tool used for test pattern generation. The respective flow diagram is depicted in Figure 10.

To start the test process sequence it is necessary to have the HDL description of the Circuit under Test (CUT)[5]. The *Synopsys DFT Compiler*™ reads that description and compiles the design. If no errors are identified during the Design Rule Checking (DRC), the tool automatically substitutes normal flip-flops by scan flip-flops. After scan insertion has been completed it is necessary to perform a new DRC operation to check the scan chain and make a design arrangement and optimization. Hence, all the inserted scan cells are reported as well as their order in the scan chain. An evaluation of the area that the scan circuit would fill (based on that it can be know the circuit area overhead with the scan design) is also performed. By then, the compiled and scanned circuit description is saved in a *Verilog* netlist that would be read by *TetraMax*™. To finalize the *DFT Compiler*™ process it is indispensable to write the Standard Test Interface Language (STIL) Process File (SPF), without which the *TetraMax*™ can not provide any ATPG result. That file is a subset of STIL syntax for input to describe scan chains, clocks, constrained ports and pattern/response data [16].

[5] In this work, the HDL descriptions for ITC'99 Benchmarks [18], developed by the CAD Group of Polytechnic of Torino, have been used.

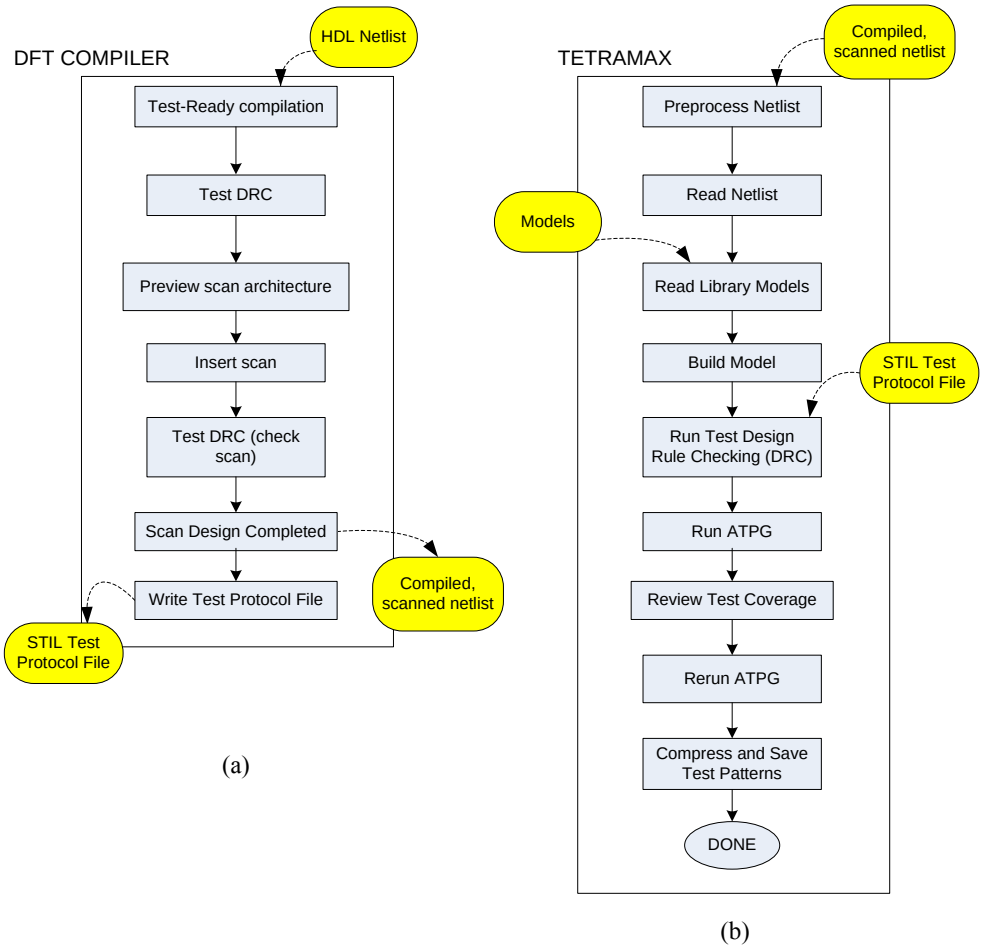


Figure 10 – *Synopsys* flow: (a) *Synopsys DFT Compiler*™ flow, (b) *Synopsys TetraMax*™ ATPG flow.

Before starting *TetraMax*™ tool it is necessary to specify the libraries that will be used. Those libraries contain all the technology information, manufacturer and design kit. In this work, the CMOS 0.35 µm technology from *Austria Micro Systems* is used with a nominal power supply voltage of 3.3V. Note that the flow presented in Figure 10(b) is a general approach for ATPG flow and can be used for generating stuck-at and transition delay patterns [17].

After reading the *Verilog* netlist and building the circuit model this CAD tool performs the ATPG. For at-speed tests using delay faults the ATPG engine generates patterns with launch and capture cycles. The launch cycle can be some cycles after shift is completed.

The tool can support the next three ATPG modes: basic-scan, fast-sequential and full-sequential. Normally, for transition fault testing, the only two modes used are the basic-scan and the fast-sequential. The basic-scan is a combinational only mode with only one capture clock between pattern scan-in and scan-out of response. The fast-sequential uses a sequential

time-frame ATPG algorithm [13]. As a consequence, LOS uses the basic-scan mode, while LOC uses the fast-sequential one.

Generally, test patterns generated by ATPG (deterministic test vectors) provide high fault coverage. Yet it can be review and the ATPG could be rerun to obtain better fault coverage results. At the end of the process, the patterns can be compressed and saved. In some cases they can be used later on as external patterns.

The most efficient at-speed test is scan-based ATPG with the transition fault model [17].

CHAPTER 3 – DYNAMIC BIST METHODOLOGY

In this chapter a new dynamic scan-BIST methodology is proposed and three architectures implementing it are introduced. First, the design flow, using commercial EDA (Electronic Design Automation) tools is described. Second, the Scan BIST based on LOS approach is introduced, as well as all its building blocks. Third, the Scan BIST based on LOC approach is presented. Finally, an architecture implementing the Scan BIST based on LOS and LOC is described. These last two architectures are explained based on the main differences to the first architecture.

3.1. Design Flow

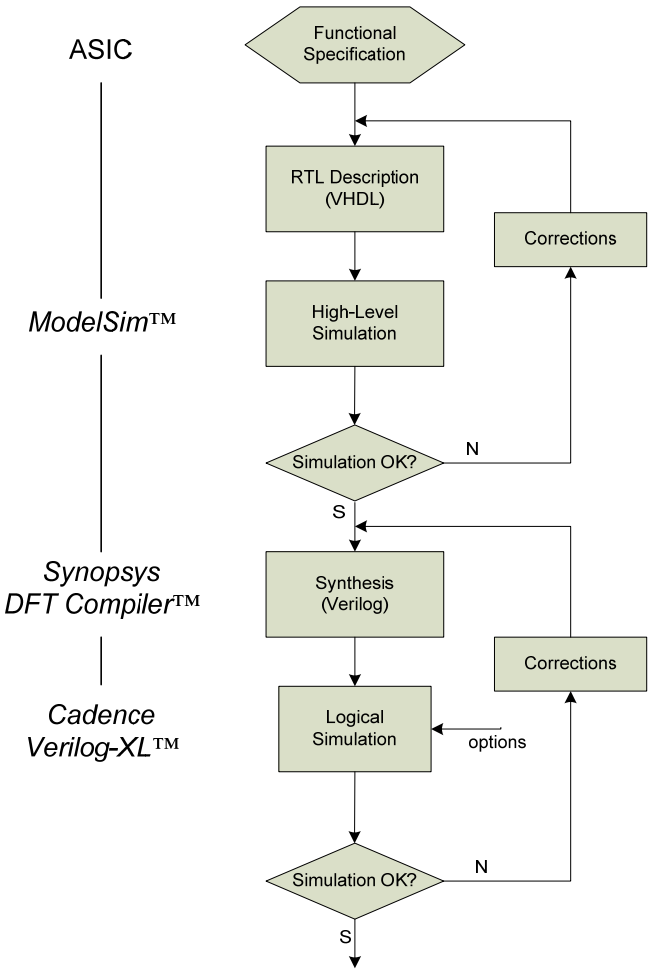


Figure 11 – Design flow used in the initial steps of the design process (referring the used tools).

The initial steps of the design flow are depicted in Figure 11. User's requirements allow design engineers to identify the functional specification to be mapped in hardware, using the integrated circuit. The functionality is described using a hardware description language (HDL), like the VHDL language. Usually, this is done at a high-level of abstraction, namely the Register-transfer Level (RTL). In order to verify that the VHDL description correctly performs the specified functionality, I used the *ModelSim*™ tool in this work. Typically, functional test patterns are used in RTL simulation[6]. If there is any problem during this process, the RTL description should be revised and again simulated. On the other hand, if the correct functionality is verified, the next step is to perform logic synthesis. In this algorithmic process, the behavioural RTL description is translated into a structural description, at a lower level of abstraction, mapped on a target cell library, associated with a manufacturing process. This process is carried out using the commercial *Synopsys DFT Compiler*™ tool that performs automatic logic synthesis. The logic level description can be written in VHDL or in Verilog. The user can control the synthesis parameters by selecting some tool options. For example, the maximum area, the clock period or the delay time can be externally specified. After that, the synthesized circuit is simulated (using the same functional test pattern) to prove that its functionality is the same of the RTL description. The logic simulation is performed by *Cadence Verilog-XL*™ tool, using a Verilog gate-level circuit description.

In the design flow, the next step (not shown in Figure 11) is to modify the Circuit under Test (CUT) to introduce scan, and to add the Scan BIST structures and components described in the following sub-sections. All the designed modules are manually generated (at present) at RTL, in VHDL. Before the synthesis step, all BIST modules and the reconfigured CUT are interconnected, so that when logic synthesis is performed, the result is a compact circuit implementing the desired functionality of the CUT (in normal mode) and the BIST functionality (in self-test mode). After that, the proposed solution is evaluated in the design environment, to prove how good it is: test effectiveness (TE) (delay fault coverage), test overhead (TO) (area overhead, speed degradation, additional pin count), test length, and test power should be evaluated. In this work, emphasis is put on TE and TO evaluation.

[6] Design validation, using simulation, is not a trivial task, as simulation uses only a limited sub-space of the possible test vectors (and their combination). Formal verification is a much more complete method than simulation-based methods. However, design validation is out of the scope of this work.

3.2. Methodology: Scan-based Dynamic BIST

As previously mentioned, the implemented Scan based BIST is a *test-per-scan*, *embedded* architecture, where the scan chain is created from the existing flip-flops. In the following, it is assumed that a *single* scan chain is generated. However, the methodology can be applied to more complex designs, for which multiple scan generation is a more adequate solution.

The proposed methodology for scan-based dynamic BIST merge the LOS and/or LOC techniques for delay testing, used in scan design with external test, with Built-In Self Test. The methodology is implemented with the architecture shown in Figure 12. This architecture exhibits some changes in comparison with the traditional Scan BIST approach. The visible and first difference is in the modules used and their interconnection. As shown in Figure 12, the architecture is composed of the CUT (reconfigured with scan flip-flops), a BIST Controller, a MISR, the input MUX and two LFSR. Due to fact that this new architecture will be used for at-speed test, some changes have to be made, especially in the BIST Controller (with different functionality) and in the TPG (using two LFSR).

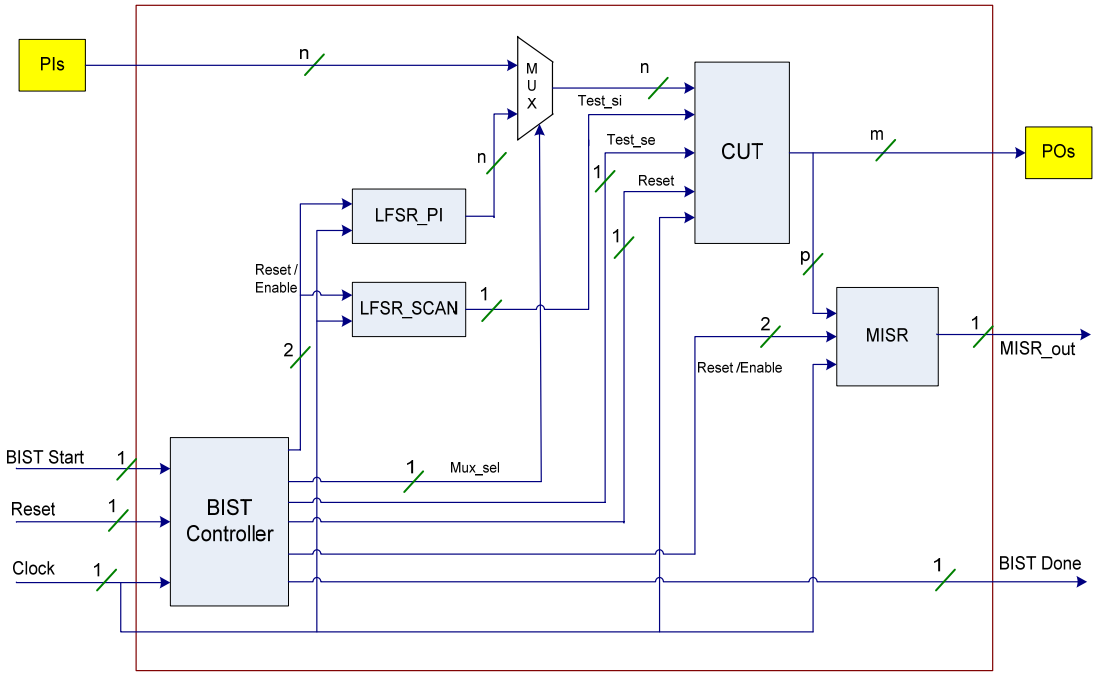


Figure 12 – Dynamic BIST Architecture proposed in this work.

As stated, I use two linear LFSR, namely LFSR_PI and LFSR_SCAN. The first one is used to generate pseudo-random (PR) test patterns to be applied to the CUT's primary inputs,

in self-test mode. The second LFSR generates PR test patterns to be applied at the first flip-flop of the scan register (as the test_si signal). Although the proposed two-LFSR solution introduces an area penalty, I decided to implement this configuration because it can reduce the polynomial correlation and, as a result, it can increase delay fault coverage. In general, it is important to analyse all design trade-offs and weight the area overhead, speed degradation and transition fault coverage, in order to select the best implementation of dynamic Scan BIST

The proposed methodology and its architecture can accommodate LOS, LOC or both. The difference among the three presented approaches is clustered in the BIST Controller functionality. This is the key module in the proposed architecture and one of the attractive features of the proposed methodology. Due to the fact that I have to implement different techniques to test delay faults and that the state of *Scan Enable* signal (defined as test_se signal in this work) is different in the LC of LOS and LOC, I have to add a new state at the module definition (referred as LAUNCH). That decision implies an increase area in the BIST Controller, as compared to the traditional module. This area overhead can be significant (of about 47%, in the examples presented in chapter 4). Consequently, the Dynamic BIST architecture introduces an additional overhead (about 17.5% area overhead, in the mentioned examples, for the overall structure). This cost occurs in the three approaches, and is similar in all, as they all exhibit similar total area.

Power consumption is another critical issue. During the at-speed self-test session, it can be much higher than in the normal operation. This issue deserves further attention, as in dynamic scan design, much of the test process operates at low speed, and the test vectors sequences, generated to uncover delay faults, are applied at-speed. In the proposed solution, I improve test quality and am expecting the power consumption to increase. Test power can be limited by reducing the test units within test sessions or reducing the clock frequency [19]. However, in this case that is not an option, as I am performing at-speed testing. As this is a *test-per-scan* architecture, the energy and power consumption may be reduced by toggle suppression, as proposed in the literature.

3.3. Scan BIST Architecture based on LOS

As referred, the proposed architecture has five modules: CUT, BIST Controller, LFSR, MISR, and MUX. The BIST Controller is the only one that changes in the other two approaches. In the following section the five modules will be explained

3.3.1. BIST Controller for LOS

As is name indicates, this module is the most important in the dynamic scan BIST architecture because it controls all the other modules and operations. Figure 13 identifies all input and output signals of the module. In the outputs it is referred the module where each output signal will be the input to. In Table 2 the same signals are identified, as well as their specified function performed in the circuit.

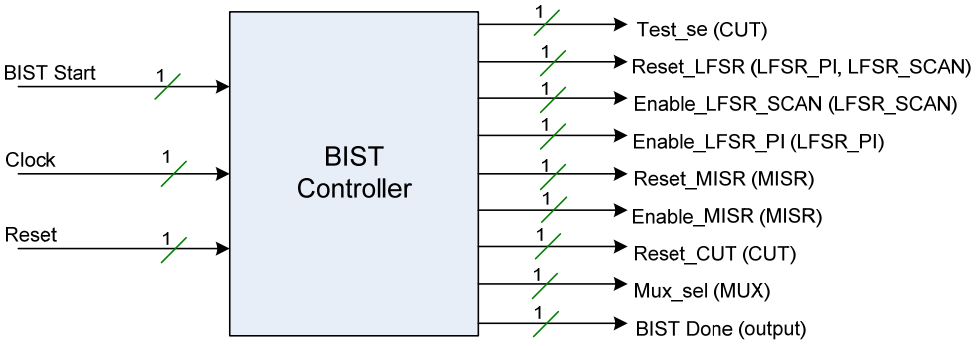


Figure 13 – Input and output BIST Controller signals.

In the proposed architecture, two counters are used for the BIST Controller: one counter (referred as the *scan counter,* C_SCAN) to count the number of flip-flops in the scan chain for shifting in the scan vectors and another counter (referred as the *vector counter*, C_VECT) to count the number of scan vectors applied to the CUT.

Signal	Type	Bits	Description
BISTstart	input	1	When high the self-test starts
Clock	input	1	Clock signal
Reset	input	1	Reset to the register
Test_se	output	1	When high activates the shifting in of scan register (CUT)
Reset_LFSR	output	1	Reset of LFSR
Enable_LFSR_SCAN	output	1	When high enables LFSR_SCAN
Enable_LFSR_PI	output	1	When high enables LFSR_PI
Reset_MISR	output	1	Reset of MISR
Enable_MISR	output	1	When high enables MISR
Reset_CUT	output	1	Reset of CUT
Mux_sel	output	1	When high the test patterns are injected in PI of CUT
BISTdone	output	1	When high indicates the end of the self-test

Table 2 – BIST Controller signals.

3.3.1.1. Implementation of the Finite State Machine (FSM)

The BIST Controller for this dynamic scan BIST with LOS is implemented based on the finite state machine whose state diagram is presented in Figure 14. This was designed to allow a credible implementation with an area optimization.

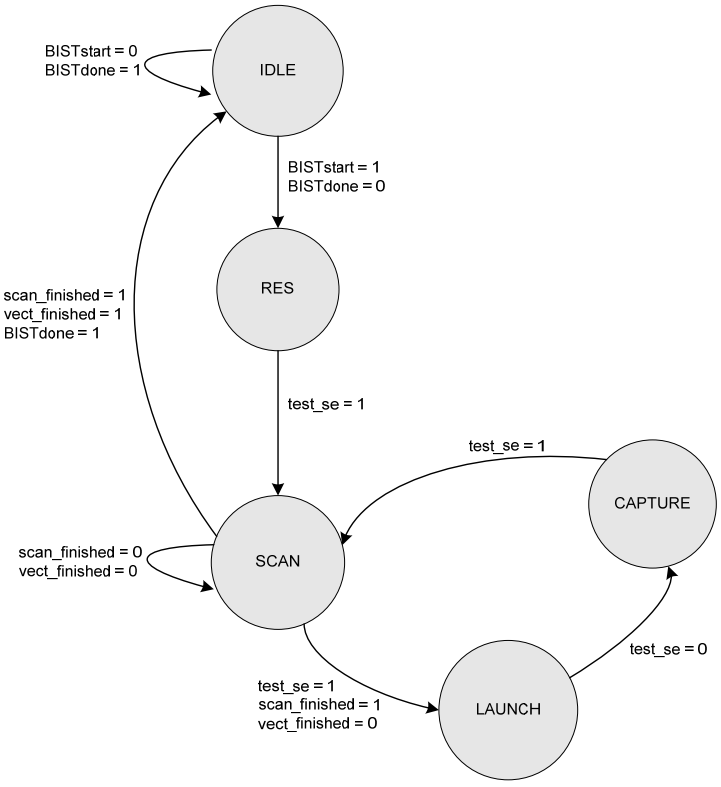


Figure 14 – LOS testing state machine.

The state machine only leaves IDLE state if the BISTstart signal goes high. While this condition is false, the scan counter and the vector counter are disabled and the test in not performed. This can be seen in Figure 15, in the three first lines of the IDLE state.

When BISTstart goes high and the state machine moves to the RES state, counters are immediately activated and the reset to LFSR, MISR and CUT are done, to guarantee that they start their operation always at the same time and at the same way. Actually, as it can be seen by the name, in the RES state the modules are activated, preparing for the self-test. After all, the changing of test_se from 0 to 1 is the condition for the state machine to move to next state: the SCAN state.

```
when IDLE =>
  reset_cscan <= '1';
  reset_cvect <= '1';
  cvect_enable <= '0';
  test_se <= '0';
  Reset_LFSR <= '1';
  Enable_LFSR_SCAN <= '0';
  Enable_LFSR_PI <= '0';
  Reset_MISR <= '1';
  Enable_MISR <= '0';
  BISTdone <= '1';
  mux_sel <= '0';
  Reset_CUT <= '0';
  if estado_seg = RES then
    reset_cscan <= '0';
    reset_cvect <= '0';
    Reset_LFSR <= '0';
    Reset_MISR <= '0';
    BISTdone <= '0';
    Reset_CUT <= '1';
  end if;

when RES =>
  reset_cscan <= '1';
  reset_cvect <= '0';
  cvect_enable <= '0';
  test_se <= '1';
  Reset_LFSR <= '1';
  Enable_LFSR_SCAN <= '1';
  Enable_LFSR_PI <= '0';
  Reset_MISR <= '1';
  Enable_MISR <= '1';
  BISTdone <= '0';
  mux_sel <= '1';
  Reset_CUT <= '0';
```

```
when SCAN =>
  reset_cscan <= '1';
  reset_cvect <= '1';
  cvect_enable <= '0';
  test_se <= '1';
  Reset_LFSR <= '1';
  Enable_LFSR_SCAN <= '1';
  Enable_LFSR_PI <= '0';
  Reset_MISR <= '1';
  Enable_MISR <= '1';
  BISTdone <= '0';
  mux_sel <= '1';
  Reset_CUT <= '0';
  if estado_seg = LAUNCH then
    Enable_LFSR_SCAN <= '0';
    Enable_LFSR_PI <= '1';
    Enable_MISR <= '0';
  elsif estado_seg = IDLE then
    test_se <= '0';
    Enable_LFSR_SCAN <= '0';
    Enable_MISR <= '0';
    BISTdone <= '1';
    mux_sel <= '0';
    Reset_CUT <= '1';
  end if;
```

```
when LAUNCH =>
  reset_cscan <= '0';
  reset_cvect <= '1';
  cvect_enable <= '1';
  test_se <= '0';
  Reset_LFSR <= '1';
  Enable_LFSR_SCAN <= '0';
  Enable_LFSR_PI <= '1';
  Reset_MISR <= '1';
  Enable_MISR <= '0';
  BISTdone <= '0';
  mux_sel <= '1';
  Reset_CUT <= '0';

when CAPTURE =>
  reset_cscan <= '1';
  reset_cvect <= '1';
  cvect_enable <= '0';
  test_se <= '1';
  Reset_LFSR <= '1';
  Enable_LFSR_SCAN <= '1';
  Enable_LFSR_PI <= '0';
  Reset_MISR <= '1';
  Enable_MISR <= '1';
  BISTdone <= '0';
  mux_sel <= '1';
  Reset_CUT <= '0';
```

Figure 15 – Code fragment including all signal states in LOS implementation.

As can be confirmed by Figure 15, test_se signal goes high, which means that the shifting in of the scan chain of the CUT is initialized. The input MUX is also enabling the local TPG to drive the CUT, and thus the LFSR_SCAN starts to generate the pseudo-random test patterns to be applied to the CUT. In this mode of operation, the SCAN mode, the test is performed and the state machine stays in this mode until the scan and vector counting are not completed. Therefore, the FSM only moves to the next mode (LAUNCH) when the scan

chain is totally filled (scan_finished = 1) and the test vectors are totally applied to the CUT (vect_finished = 0). If the counting is completed, BISTdone goes high, which means that the self-test session is concluded. As a result, the state machine returns to the IDLE state.

When the FSM is in LAUNCH state, and because this is the LOS approach, test_se does not change and stays at the high level. The LFSR_PI generates the test vectors to be applied as primary inputs to the CUT (MUX is enable) and LFSR_SCAN stops generating test patterns to be applied to the scan chain. As a result the second vector is a shifted version of the first one applied.

Test_se must go low to allow the state machine to change state. This is the condition to move to the CAPTURE state. In this state, the self-test is stopped and the circuit's response is captured, which is performed by enabling the MISR. Before continuing the test operation and returning to the SCAN state, the scan counter and the CUT are reset. The Test_se signal must go high again, which is the condition to return to the SCAN mode of operation.

In Figure 16 typical signals waveforms of the LOS operation are represented. As it can be seen, the test_se signal is only at low level during the CAPTURE cycle, after the scan counting has been completed. Then reset to scan counter is performed and a new SCAN cycle is carried out. When both counters finish counting (scan_finished and vect_finished waveforms are at a high level), BISTdone goes high and the state machine returns to the IDLE state. Since BISTstart is still at a high level, the all process starts again.

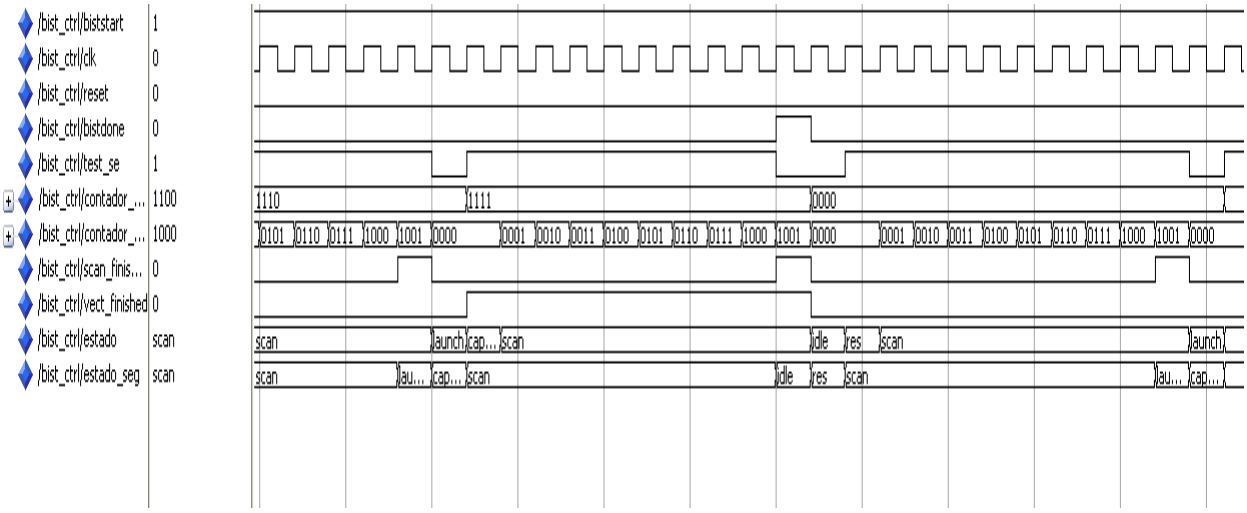


Figure 16 – Functional explanation of LOS behaviour (*ModelSim*™).

3.3.2. LFSR

In this work are used two LFSR: LFSR_PI and LFSR_SCAN. They are both depicted in Figure 17, with the identification of the input and output signals of each one. Both are controlled by the BIST Controller that produces the Enable and Reset control signals. The LFSR generate the pseudo-random test patterns to be applied to the CUT as primary inputs (LFSR_PI) and to be applied and shifted in to the scan chain (LFSR_SCAN).

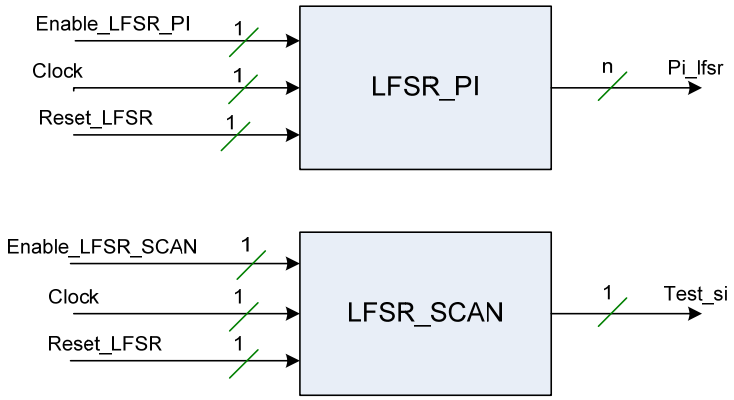


Figure 17 – Input and output signals for the two used LFSR: LFSR_PI and LFSR_SCAN.

As previously mentioned, the two-LFSR solution was implemented to decrease the polynomial correlation of the test vectors and thus to improve the transition fault coverage.

In the proposed dynamic scan BIST methodology is used two linear LFSR because they are easily implemented due to the uniformity of the scan register and for their suitability for scan design. The degree of the polynomial (i.e., the number of registers) depends on the CUT. In this work, I implemented two 11 bit LFSR generators (with a *degree of polynomial* = 11) mapping a primitive polynomial $P(x) = x^{11} + x^2 + 1$. Hence, I implement an LFSR with 11 flip-flops and 1 exclusive OR gate (x^2). Since the 11 registers LFSR is big to be drawn, in Figure 18 there is show a linear LFSR with 4 flip-flops and 1 exclusive OR gate, implementing also a primitive polynomial, $P(x) = x^4 + x + 1$.

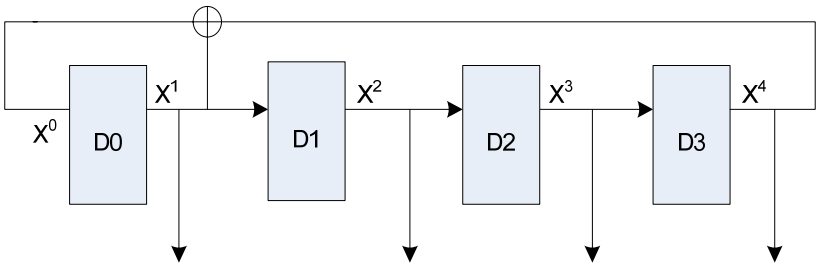


Figure 18 – Linear LFSR feedback implementing the primitive polynomial $P(x) = x^4 + x + 1$.

3.3.3. MISR

Again, the number of flip-flops of the Multiple Input Shift Register (MISR) depends on the number of PO of the CUT to be observed. In this work, to the MISR implementation it is used an LFSR with a *degree of polynomial* of 7. The input and output signals of the MISR module are shown in Figure 19. Note that the LFSR used as an Output Response Analyser (ORA) needs input data, namely the output response of the CUT. That input is referred in Figure 19 as PO and has a size signal *p*. That signal may have a different size from the primary outputs of CUT (represented in Figure 12 with an *m*) because it can changes from 1 to *m* (in the compaction process, one or more PO can be used as MISR input data).

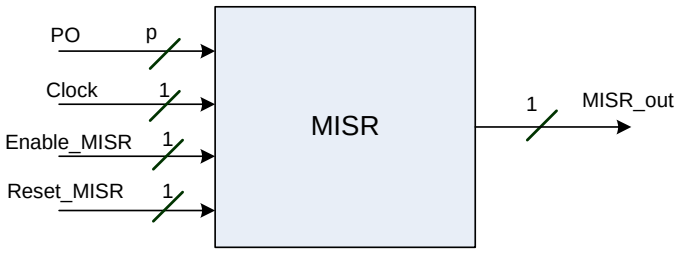


Figure 19 – Input and output MISR signals.

As it can be seen in the next sub-section, one of the CUT used to illustrate the usefulness of the proposed methodology (b06) has 6 PO. As a result, a MISR with 6 flip-flops is required. The aliasing probability in this case is about $1/2^6 \approx 1.56\%$. In order to reduce the aliasing probability, I decided to implement the MISR with 7 flip-flops, leading to $1/2^7 \approx 0.78\%$, which is much more acceptable. That probability decrease can be achieved with a modest area overhead. Note that this is done to the worst possible case, when PO signal reaches its maximum value.

3.3.4. CUT (Circuit under Test)

The CUT used to test the proposed dynamic scan BIST methodology was the ITC'99 b06 benchmark, which is a sequential circuit [18]. The benchmark circuit is previously synthesized and reconfigured for scan chain insertion. Before made this interconnection to other modules it was compiled to perform an area optimization. The structural implementation (at gate level) is represented in Figure 20.

In Figure 21 the input and output CUT signals are shown. The *n* and *m* represents, respectively, the maximum number of PI and PO of the circuit.

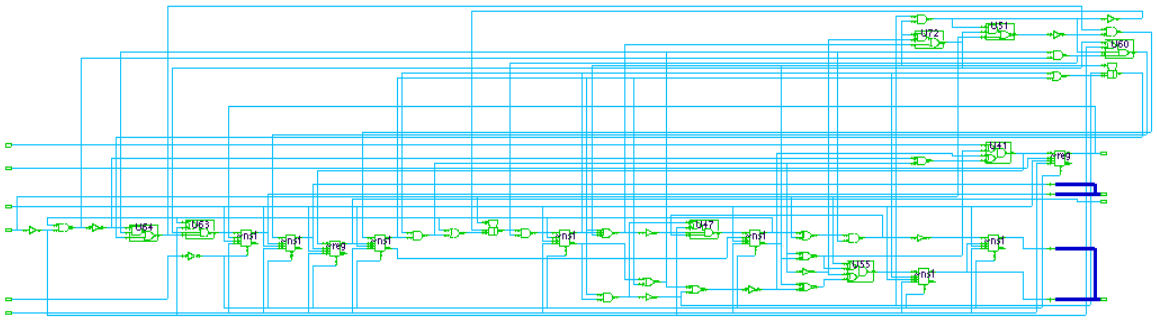


Figure 20 – Optimized b06 CUT with scan chain.

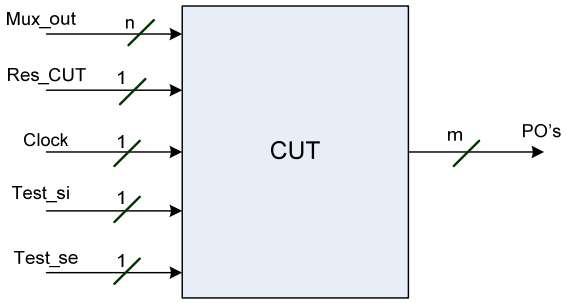


Figure 21 – Input and output CUT signals.

3.3.5. Multiplexer

The number of implemented multiplexers depends on the number of PI. It is necessary one MUX for each PI. The input and output signals of this module are identified in Figure 22. The function of that component is to select if the primary input signals applied to CUT are the external ones, or the ones generated by LFSR_PI. When the mux_sel signal is enabled, the PR test vectors are applied to the CUT, which happens in the self-test mode.

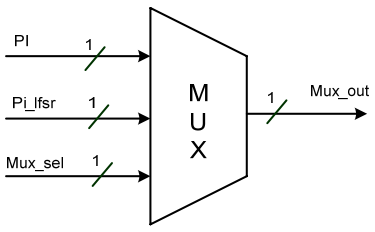


Figure 22 – Input and output MUX signals.

3.4. Scan BIST Architecture based on LOC

This Scan BIST architecture is similar to the first one presented. It has also five modules, and the only changes are made in the BIST Controller. For this reason, the only module described is this sub-section is the BIST Controller.

3.4.1. BIST Controller for LOC

The BIST Controller to implement delay testing with LOC is made based in the finite state machine shown in Figure 23. This was designed to allow a credible implementation with area optimization.

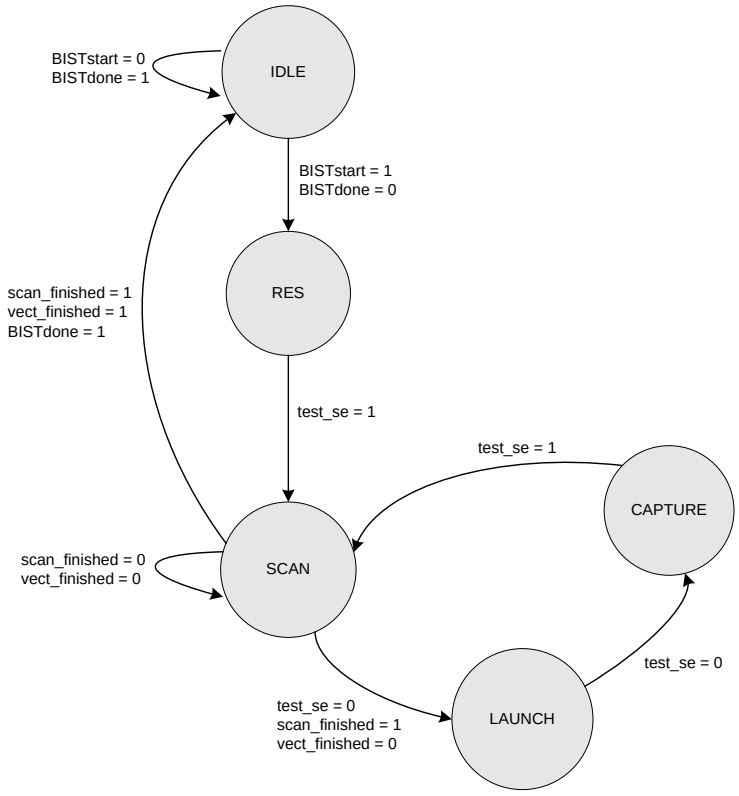


Figure 23 – LOC testing state machine.

Comparing Figures 23 and 14, it can be seen that the only difference between the state diagrams of the two FSM is in the test_se signal logic value between the SCAN and LAUNCH states.

The condition for the state machine to move from the RES to the SCAN state is that the test_se signal must go high, which means that the shifting in of the scan chain of the CUT is

initialized. In that state, the input MUX is enabled and LFSR_SCAN starts the generation of pseudo-random test patterns to apply to CUT. The state machine stays in that mode, performing self-test, until the scan and vector counting are completed. That means that the FSM only moves to the next state (LAUNCH) when the scan chain is totally filled (scan_finished = 1) and the test vectors are totally applied (vect_finished = 0). If the counting is completed, BISTdone goes high, which means that the test session ends. Hence, the state machine returns to IDLE mode.

As this architecture it is based on LOC, to move from SCAN to the LAUNCH state, test_se signal must go low (that is the main difference between the approaches). LFSR_SCAN stops generating test patterns to be applied to the scan chain and the MUX is enabled, so the LFSR_PI generated test vectors can be applied as PI at the CUT.

As can be observed by Figure 23, test_se does not change his value when the FSM moves from LAUNCH to the CAPTURE state. In that case the circuit response is captured, which is performed by enabling the MISR. Before continuing the test operation and returning to the SCAN state, a reset is done to the scan counter and to the CUT. The Test_se control signal must go high again, which is the condition to return to SCAN mode of operation.

In Figure 24, typical signal waveforms of the LOC behaviour are depicted. As it can be seen, the test_se signal goes low when the scan counting is completed and stays at low level during LAUNCH and CAPTURE states. When scan and vector counting are finished, the BISTdone signal goes high and the state machine returns to IDLE mode.

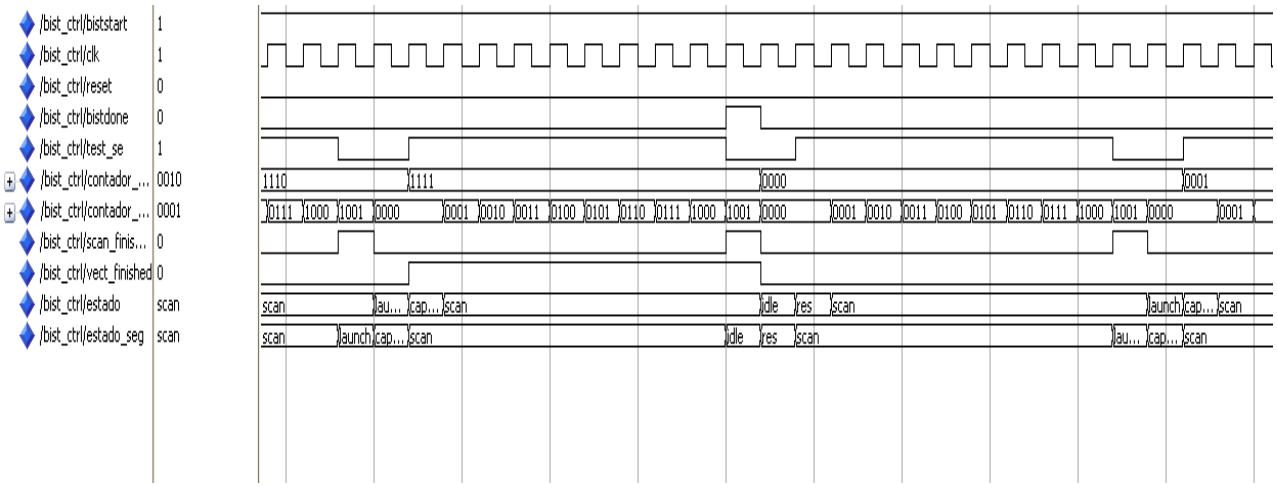


Figure 24 – Functional explanation of LOC behaviour (*ModelSim*™).

3.5. Scan BIST Architecture based on LOS and LOC

In this third alternative (with the same architecture), LOS and LOC-based delay self-testing may be activated. The proposed solution is based on the two previous ones. The changes to implement this architecture are made in the BIST Controller and are explained in the following section.

3.5.1. BIST Controller for LOS and LOC

In this case, the modules and signals are the same that were presented in sections 3.3.1 and 3.4.1. The state machine is basically the same; however, during the SCAN state the test operation described in Figure 25 is executed: if the BISTstart is at high level, in the next state (LAUNCH) the test_se signal is at high level too and a LOS-based test is performed. On the other hand, if the BISTstart is at low level, test_se goes low in LAUNCH mode a LOC-based test is executed.

This is a very interesting implementation because in the same structure I can perform two different delay tests, *by changing the logical state of only one input signal, namely the BISTstart.* Usually, and as it will be confirmed in the next chapter, this new solution improves transition fault coverage with the same area overhead of the other two presented solutions.

```
if estado_seg = LAUNCH and BISTstart = '1' then  -- BISTstart = '1': LOS
  test_se <= '1'; -- difference between LOC do LOS
  Enable_LFSR_SCAN <= '0';
  Enable_LFSR_PI <= '1';
  Enable_MISR <= '0';
elsif estado_seg = LAUNCH and BISTstart = '0' then  -- BISTstart = '0': LOC
test_se <= '0';
  Enable_LFSR_SCAN <= '0';
  Enable_LFSR_PI <= '1';
  Enable_MISR <= '0';
```

Figure 25 – Code fragment where the switch between LOS and LOC based test is performed.

In Figure 26 the signal waveforms of LOS and LOC behaviour are presented. As can be observed, when the BISTstart goes low, the type of test performed changes. Thus, before the signal changes logical state, test_se is at low level only during CAPTURE mode and the LOS-based delay test is performed. After the changing, test_se is at low level at LAUNCH and CAPTURE states and, as a result, the LOC-based test is performed.

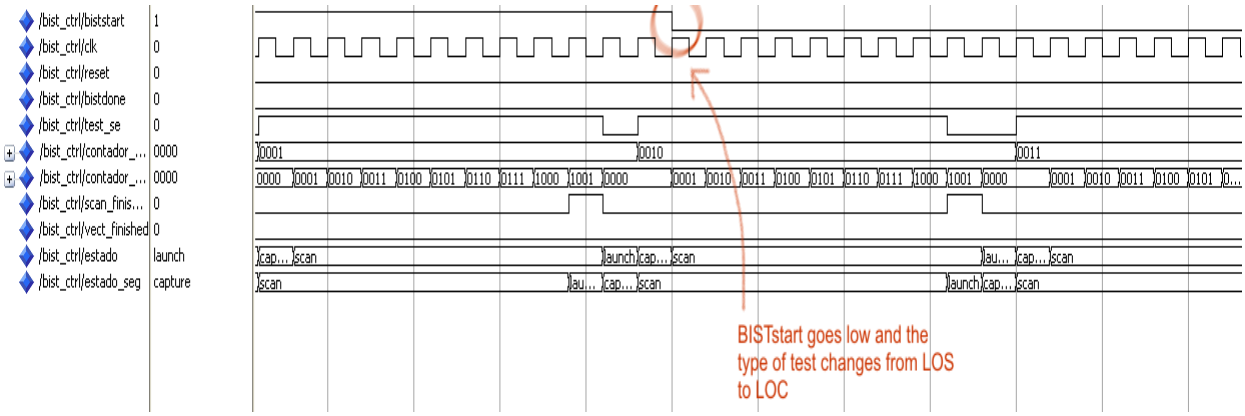


Figure 26 – Functional explanation of LOS and LOC implementation behaviour (*ModelSim*™).

CHAPTER 4 – TEST FLOW AND SIMULATION RESULTS

In this chapter I present the most relevant test results and simulations performed during this work. First, the implemented transition delay fault design flow is described, summarizing the work flow and the tools used. Second, the benchmark circuits that have been used to validate the methodology are characterised. Finally, fault simulations results are presented and discussed.

4.1. Transition Delay Fault Test Flow

In order to perform the proposed dynamic scan BIST methodology for Transition Delay Fault testing, several software tools have been used. The corresponding design flow is presented in Figure 27. This design flow summarises all the performed steps starting with insertion, passing through the development of the Scan BIST architectures and logic synthesis, down to the transition fault simulation using pseudo-random test patterns.

Before starting the generation of the RTL description of the desired BIST structure, scan chain insertion is performed in the circuit (ITC'99 Benchmark, previously defined) that I want to make more delay-testable. Scan chain insertion is automatically performed by the *Synopsys DFT Compiler*™. After that, the structural circuit is synthesized to guarantee area optimization[7].

The Dynamic BIST structure can now be generated and described in a high-level circuit description language. At present, this task in carried out manually. To confirm that the developed architecture executes the correct functionality, I use the *ModelSim*™ tool to perform RTL simulation. If there is any problem during this process, the RTL description should be revised and again simulated. On the other hand, if a correct functional behaviour is obtained, the next step is do its logical synthesis, passing to the structural level and generating the gate-level circuit description. The *Synopsys DFT Compiler*™ tool executes automatically

[7] Other synthesis options, like speed optimization, may be selected, depending on the application.

the required task, doing an area optimization and generating the necessary files to be read by other CAD tools, as *Verilog* netlist.

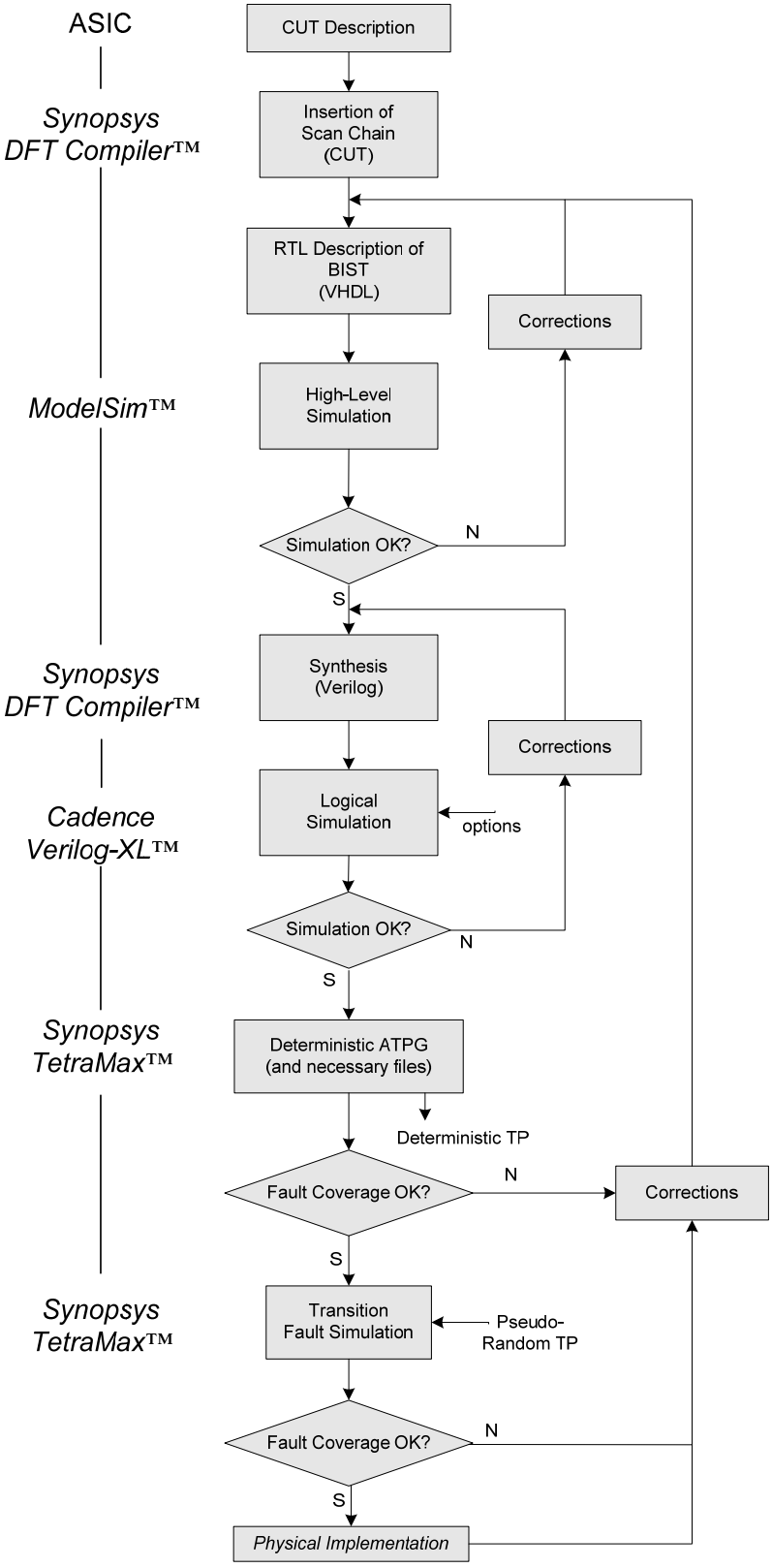


Figure 27 – Design flow of the methodology and EDA tools.

After that, using *Cadence Verilog-XL*™ CAD tool, logic simulation can be performed to confirm that the structural description performs the desired and required functionally, equal to the one performed before the synthesis step.

Afterwards, Automatic Test Pattern Generation (ATPG) is performed by *Synopsys TetraMax*™ CAD tool. The first step that has to be performed is the *deterministic* test pattern generation, running this tool. Usually, the deterministic test pattern is composed of a limited number of test vectors, M_d, in a sequence such that test vector pairs uncover (if possible) all the listed transition faults[8]. At the same time, *TetraMax*™ will compute the transition fault coverage achieved by the tool with the deterministic test set. *TetraMax*™ also generates the necessary files to be used in a future step and that can be externally read. If the achieved fault transition fault coverage has not an acceptable value, it has to be done some corrections and improvements to the hardware circuit description and rerun all the process flow. On the other hand, if it is achieved an acceptable transition fault coverage, the next step is the *pseudo-random* test generation. This is not performed by a software tool, but by hardware, through the implemented two-LFSR TPG. I have two degrees of freedom, which may lead to various PR test patterns: (1) test length, TL (i.e., the number of PR test vectors ($M < 2^N-1$), and (2) the LFSR seeds (i.e., the initial test vector in each LFSR). Although different PR test patterns may lead to different TF coverage values (as well as different evolution of TF coverage with the number of applied test vectors), the experience shows that limited gains are obtained using these degree of freedom. In the following, simulation results will be presented for only one TL, and one LFSR seed set. Typically, it is to be expected that $M>>M_d$. Even so, the transition fault coverage obtained by the PR test pattern may not reach the fault coverage value obtained with the deterministic test pattern.

In order to perform PR test pattern generation, I have to build a *testbench* file for being used in *Cadence Verilog-XL*™, which allows us to save all circuit characteristics to perform *Synopsys TetraMax*™ simulations: (1) the pseudo-random test patterns generated by LFSR, (2) the output patterns (the circuit response), (3) the scan chain behaviour and (4) the test sequence that has to be performed. Finally, the last step of the design flow can be carried out: transition fault simulation using the pseudo-random test patterns and the determinist patterns.

[8] The number of test vectors, M_d, may be even or odd, as sometimes a second vector of a vector pair can be used as the first vector for another vector pair. The ATPG tool tries to reduce, as much as possible, M_d.

If there is any problem all the process has to be repeated. If all the results are acceptable the work is concluded.

4.2. Characteristics of the Benchmark Circuits under Test

The circuits that have been tested in this work are ITC´99 Benchmarks [18], developed in the CAD Group at Polytechnic of Torino. The target IC technology is the CMOS 0.35 μm technology from *Austria Micro Systems* and the c35_CORELIB, c35_IOLIB_4M and c35_IOLIBV5_4M cell libraries. All the used benchmarks are sequential circuits and have been synthesised by *Synopsys DFT Compiler*™ CAD tool, performing scan chain insertion and area optimization.

4.2.1. B06

This benchmark circuit is an interrupt handler [18] with 56 gates, 2 primary inputs (PI), 6 primary outputs (PO) and 9 feedback flip-flops. Consequently, the reconfigured CUT has 9 scan flip-flops. The logic-level circuit can be observed in Figure 20.

During the transition fault simulation process, 308 single transition faults were automatically injected in the circuit by the *Synopsys TetraMax*™ tool. To perform the test, TL_{06} = 130000 pseudo-random test vectors were generated and applied by the on-chip TPG s.

The CUT estimated area is about 6912 μm^2. The implemented BIST Controller has an area overhead of 47% and the Scan BIST architecture has an area overhead of 17.5% as compared to the traditional BIST approach.

4.2.2. B10

This second benchmark circuit is a voting system [18] with 206 gates, 11 PI, 6 PO and 17 scan flip-flops. In order to perform the transition fault simulation, 696 transition faults were automatically injected in the circuit by the *TetraMax*™ tool. In this case, TL_{10} = 300000 pseudo-random test patterns were applied. LFSR_SCAN with a *degree of polynomial* of 31 and LFSR_PI with a *degree of polynomial* of 29 were the two used LFSR s. Although, with

11 PI I would expect a *degree of polynomial* = 11, those LFSR s were used to improve the transition fault coverage and diminish patterns correlation.

The CUT estimated area is about 14484.6 μm^2. The implemented BIST Controller has an area overhead of 48% and, as a consequence, the Scan BIST approach has an implementation overhead of 19% as compared to the traditional BIST approach.

4.2.3. B13

This b13 benchmark circuit is an interface to meteo sensors [18] with 362 gates, 10 PI, 10 PO and 53 flip-flops (reconfigured as scan flip-flops by the *Synopsys DFT Compiler*™ tool). During transition fault simulation, *TetraMax*™ automatically injects 1392 single transition faults in the CUT. For this more complex benchmark, TL_{13} = 650000 pseudo-random test patterns generated by two LFSR were used (LFSR_SCAN with a *degree of polynomial* of 47 and LFSR_PI with a *degree of polynomial* of 28). Please note that the high *degree of polynomial* has been used to decrease patterns correlation and thus obtain a better transition fault coverage (as mentioned before).

The CUT estimated area is about 36461.4 μm^2. The implemented BIST Controller has an area overhead of about 48% and the Scan BIST approach has, in this case, an implementation overhead of 18.5%.

4.3. Transition Faults Simulation

In this section a significant set of the transition fault simulation results for the three above mentioned benchmark circuits is presented. In order to demonstrate the usefulness of the proposed methodology, with a single architecture and three self-test functionalities, the results are referred to the three proposed test solutions (Scan BIST based on LOS, based on LOC and based on both). Moreover, results are given for deterministic and for PR test patterns.

4.3.1. Fault Coverage for B06

The transition fault coverage achieved by pseudo-random patterns, generated by the two used LFSR, is shown in Figure 28. In the same figure, the results of the three proposed methods are presented. Those simulations are obtained with 130000 pseudo-random patterns and a TF coverage value of about 65.91% is obtained in the LOC approach, 66.88% in LOS and 77.27% in LOS and LOC architecture, respectively. As expected, the LOS and LOC approach is the one that leads to the best TF coverage results. Even with low fault coverage, the other two BIST solutions lead also to acceptable results. These need to be compared to those achieved by determinist test patterns (see Table 3).

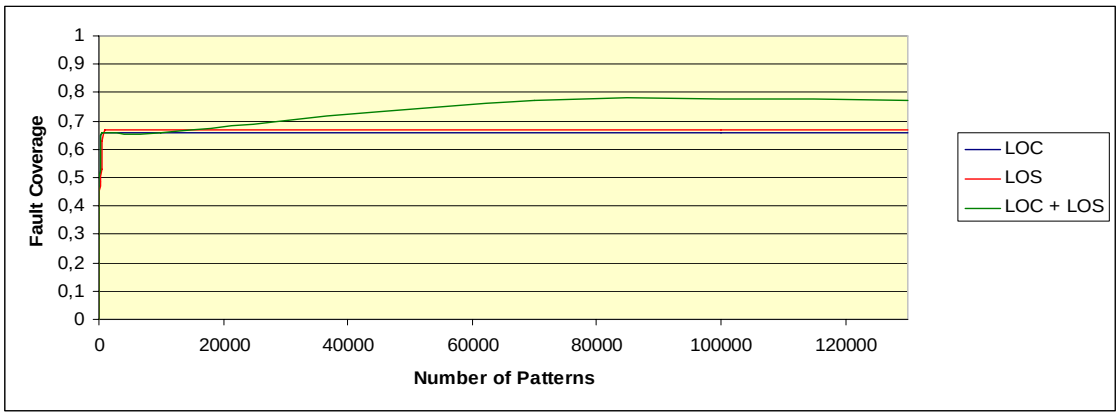


Figure 28 – Transition fault coverage evolution for b06.

In Table 3, data on transition fault coverage and the number of deterministic test vectors in the different delay test approaches is given for the b06 benchmark circuit[9]. Comparing Figure 28 and Table 3, it can be concluded that, according to the expectations, with deterministic test patterns, I can achieve higher transition fault coverage than with PR test patterns, with fewer test patterns applied. The same trend as in the pseudo-random test can be observed for deterministic test: the transition fault coverage is higher in the LOS and LOC architecture and has the lowest value in the Scan BIST based on LOC approach. This confirms the known results published for LOS and LOC techniques in scan design with external test, and extend the results for scan BIST.

[9] The number provided in Table 3 is the number of test vectors generated by the *TetraMax™* tool, not the actual number of clock cycles needed to perform the scan test.

Circuit	**b06**	
Type of Test	Fault Coverage	Number of Patterns
LOC	72.08%	11
LOS	85.06%	12
LOC + LOS	98.05%	22

Table 3 – Transition fault coverage achieved by determinist test patterns for b06.

Comparing the results achieved by pseudo-random and deterministic patterns, the better results are reached by the latter ones (deterministic), as expected. Nevertheless, the TF coverage of the PR test pattern is also acceptable and shows that it is possible to guarantee good fault coverage with (at least some of) the developed solutions. However, to be honest, PR test patterns seem to do a good job (although at the expense of large TL) for the LOC-based delay test approach (65.91% with PR, against 72.08% with deterministic test). For the solutions leading to better test performance (LOS, and LOS/LOC), the PR test patterns lag behind (e.g., 66.88% with PR, against 85.06% with deterministic test, for the LOS approach). This may indicate that, for dynamic BIST, and as it happens with BIST for static faults, high fault coverage may require a hybrid solution, combining PR test vectors and some deterministic test vectors, to uncover the hard-to-detect faults.

In order to perform pseudo-random transition fault simulation, a CPU time of about 5.99 seconds is required in a Pentium 5 at 3 GHz, with 2 GHz of RAM memory, while in the deterministic test only 0.01 seconds are required. The same can be observed with the memory usage: for pseudo-random test, memory requirements are three times the value of the ones for the on determinist case. Yet, this is to be expected, due to the number of applied patterns in both cases (e.g., I am comparing 130000 with 10 or 20 patterns).

4.3.2. Fault Coverage for B10

The transition fault coverage values achieved by pseudo-random test patterns, generated by the two used LFSR, are shown in Figure 29. As for the b06 circuit, the Scan BIST based on LOC, based on LOS and based on LOS and LOC evolution results are represented in the same graphic. For performing those simulations 300000 pseudo-random patterns where used and the reached transition fault coverage is about 51.15% in LOC approach, 57.47% in LOS and 59.37% in the LOS and LOC case. As it can be observed, the third BIST implementation

(LOS and LOC) is the one that reaches the best results, although in this case the three of them exhibit low transition fault coverage. This can be justified by the circuit itself, being more complex and having a higher number of nodes that are not observable.

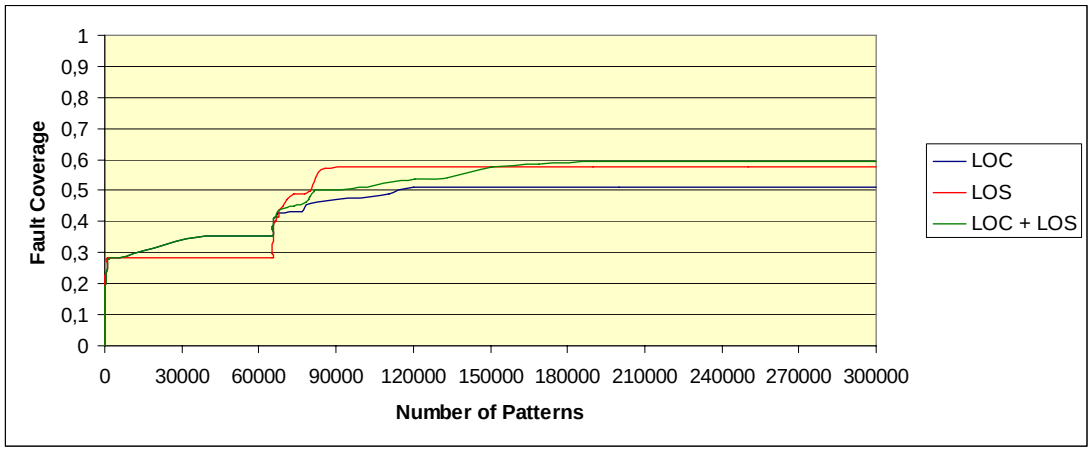


Figure 29 – Transition fault coverage evolution for b10.

In Table 4 the TF coverage results for the determinist test patterns are presented. As expected, the fault coverage achieved is much higher than with the pseudo-random test patterns (Figure 29) and with fewer test patterns applied. One more time, the LOC and LOS approach results are the best of the three methods. In the opposite site is the LOC approach with the worst fault coverage.

Circuit	**b10**	
Type of Test	Fault Coverage	Number of Patterns
LOC	58.76%	29
LOS	81.75%	32
LOC + LOS	97.70%	41

Table 4 – Transition fault coverage achieved by determinist test patterns for b10.

In this case the required CPU time to perform pseudo-random transition fault simulation is about 43.5 seconds, 4350 times higher than the one required by deterministic transition fault simulation, which is the same that in the last case, namely 0.01 seconds. The CPU memory usage was increased more than 6 times (in pseudo-random test 71.74 MB were required, against the 12.05 MB used to perform determinist test). One more time, that is to be expected, due to the number of applied number test patterns with PR self-test.

4.3.3. Fault Coverage for B13

The transition fault coverage achieved by applying pseudo-random test patterns in this last case is presented in Figure 30. A test length of 650000 patterns was used, that lead to a transition fault coverage of about 63.90% in the Scan BIST based on LOC approach, 65.21% in Scan BIST based on LOS and 69.39% in Scan BIST based on LOS and LOC implementation. In the three cases the reached TF coverage value is interesting and can be considered as good fault coverage for using only PR stimuli. One more time, the LOS and LOC architecture allows the best results. In the opposite site is the Scan BIST based on LOC approach.

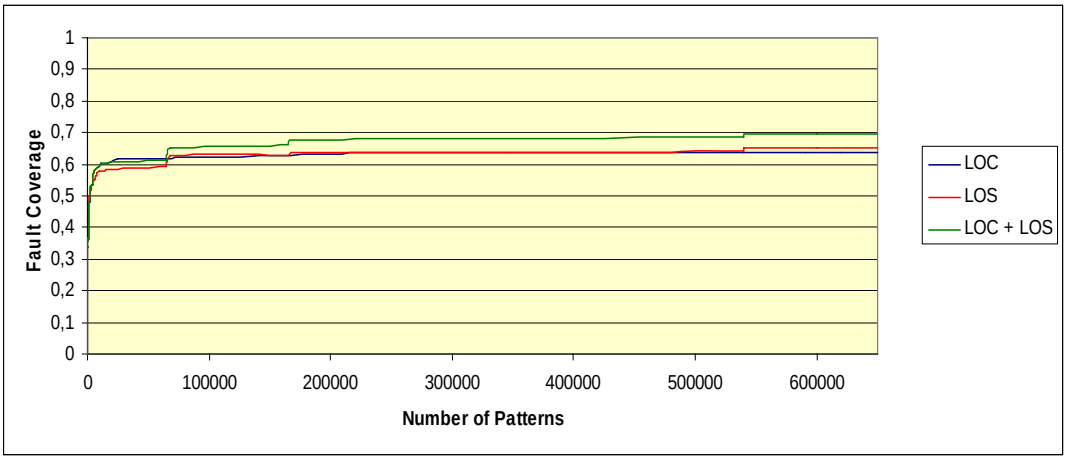


Figure 30 – Transition fault coverage evolution for b13.

In Table 5 the transition fault coverage achieved for the three different proposed methods are shown, together with the number of deterministic test patterns applied. According to the expectations, the results achieved by applying deterministic patterns are better than those obtained with pseudo-random and with fewer applied test patterns. Again, the LOS and LOC approach is the architecture with the higher transition fault coverage. One more time is confirmed that the proposed architectures can lead to relatively good transition fault coverage.

Circuit	**b13**	
Type of Test	Fault Coverage	Number of Patterns
LOC	64.78%	35
LOS	82.90%	22
LOC + LOS	96.15%	37

Table 5 – Transition fault coverage achieved by determinist test patterns for b13.

In this case, due to the fact that 650000 pseudo-random patterns and only about 30 deterministic patterns were applied, the CPU time increases more than 14413 times (114.13 seconds to first class of test against 0.01 seconds on the second one). The memory usage was increased to more than 12 times, reaching 151.57 MB in the pseudo-random test. This is an expected result.

4.4. Analysis of Results

In order to perform a results analysis I can start by comparing the three proposed architectures (Scan BIST based on LOC, Scan BIST based on LOS and Scan BIST based on LOS and LOC) for each study circuit and for the type of test patterns (deterministic or PR) applied.

For each circuit analysed, the transition fault coverage results achieved by applying to the CUT deterministic test patterns are presented in Figure 31. As it can be seen, LOC approach leads to the worst fault coverage, followed by LOS. The highest transition fault coverage is obtained by merging LOS and LOC solutions. These are the better results that I can achieve with deterministic test sets. The aim is, using LFSR test generators, to reach values as close as possible to those presented.

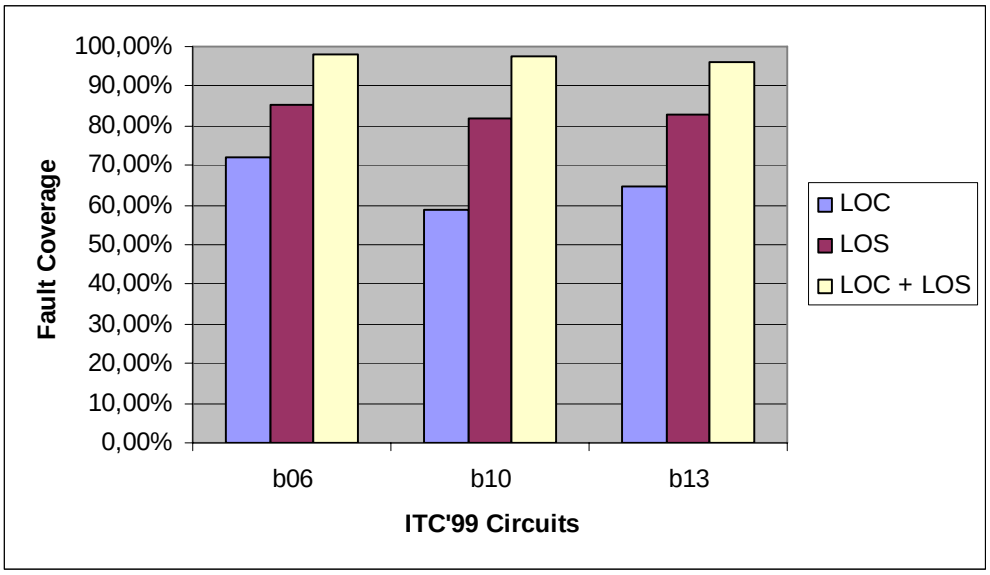


Figure 31 – Comparison of transition fault coverage achieved by deterministic test patterns for the three testing methods.

In Figure 32 the transition fault coverage results for the three proposed architectures and for each circuit are shown, applying the pseudo-random test patterns to the CUT. As in the last case, the LOC approach shows the worst results, followed by LOS. The LOS and LOC proposed architecture achieves the best transition fault coverage. The three circuits have worst results in comparison with those obtained by determinist patterns. This is an expected situation since I used pseudo-random patterns. However, I achieved reasonably good transition fault coverage values and it is possible to say that those results may be acceptable for the presented proposed solutions. Anyway, it is possible to improve the TF coverage, using another LFSR test generator with low polynomial temporal correlation, or adding some deterministic test vectors. One way to solve this problem is to increase the degree of polynomial, despite the area increase caused by that.

The fault coverage has low values in the b10 case, for both types of applied test patterns. It can be justified by the circuit itself, that is, the circuit has a considerable number of not observable nodes, which decreases the fault coverage.

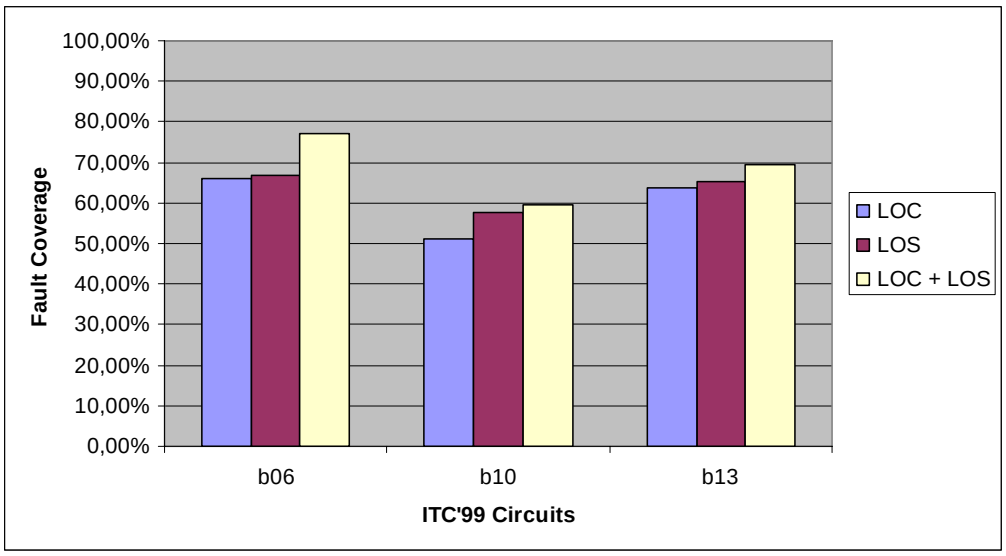


Figure 32 – Comparison of transition fault coverage achieved by pseudo-random test patterns for the three testing methods.

Figures 33, 34 and 35 show the transition fault coverage achieved by doing a comparison between the types of test patterns applied, respectively, for b06, b10 and b13 benchmark circuits. As it can be seen, for both cases, determinist and pseudo-random patterns, the fault coverage increases from LOC delay test approach to LOS and LOC. At the same time, the deterministic transition fault coverage is always higher than pseudo-random transition fault coverage.

As a summary, the transition fault coverage depends on the CUT complexity and sequential depth, of the type of delay test that is carried out (LOC, LOS or LOS and LOC) and on the used pseudo-random test generator, due to the polynomial correlation. In this manner, it is mandatory to have a good pseudo-random sequence generator.

In terms of CPU time and memory usage, the observable values are as expected, as previously referred. This has to do to the higher number of applied pseudo-random patterns.

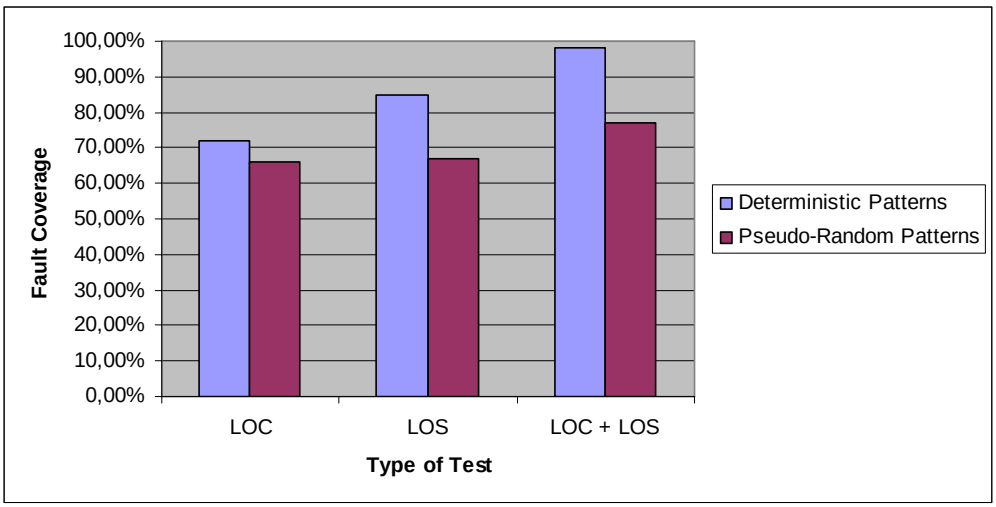


Figure 33 – Comparison of transition fault coverage achieved by both test patterns for b06.

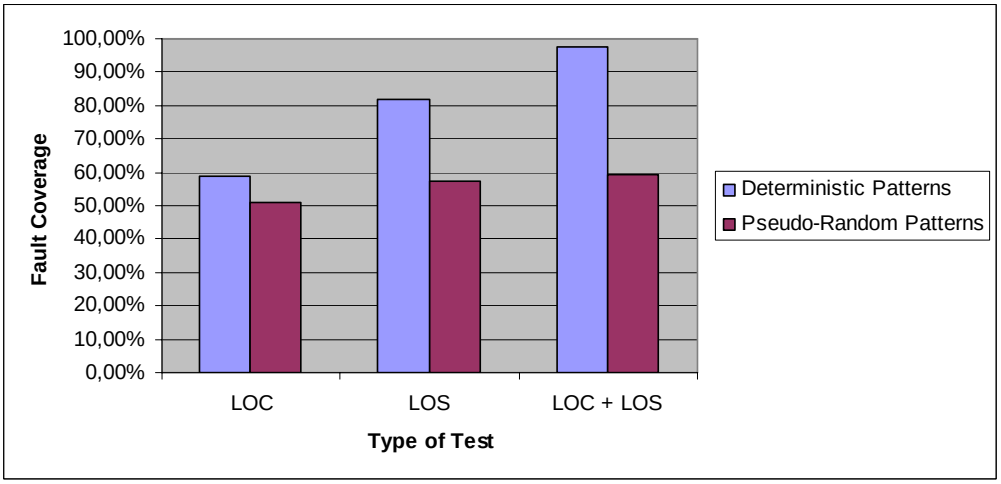


Figure 34 – Comparison of transition fault coverage achieved by both test patterns for b10.

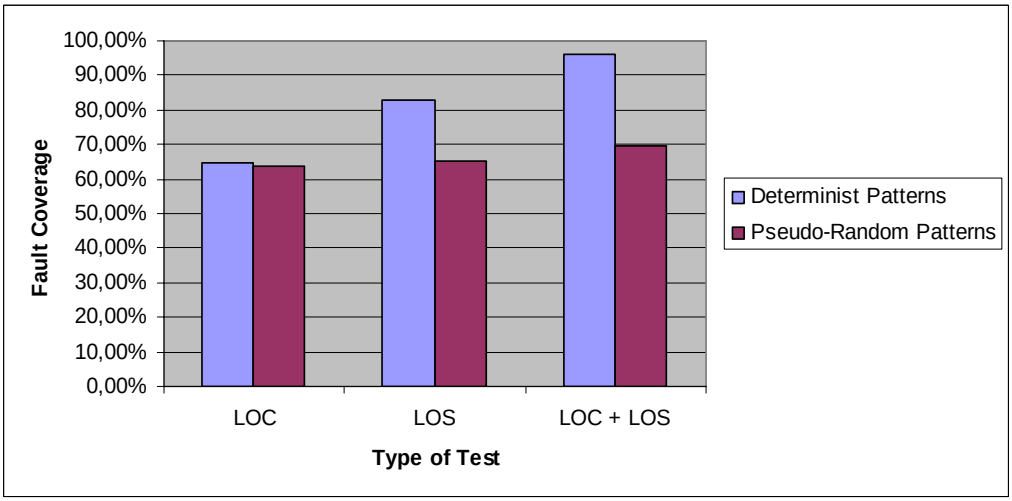


Figure 35 – Comparison of transition fault coverage achieved by both test patterns for b13.

CHAPTER 5 – CONCLUSIONS AND FUTURE WORK

5.1. Conclusions

Although a large amount of work has been done in the past to derive efficient test processes for transition faults and path delay faults, delay test still remains one of the greatest challenges in the field of testing. Due to the new 90nm and 65nm technologies, hundred of million gates operating in the GHz range and new processing materials and manufacturing processes, delay testing is becoming more and more important. Consequently, new methods are required to test small delay faults along with the usual transition faults. The great difficulty is how to derive a cost-effective test process with the increasing complexity, performance, power consumption and low pin count of today's SoC.

BIST is an attractive and common technique of digital system test. In the future it has to be an explored solution to help to manage cost-effective testing. Scan BIST merges BIST and Scan Design techniques, with their associated costs and benefits. Silicon area, pin and performance overheads are some of the most important disadvantages. Low test generation cost and the ability of perform at-speed test are some of the most important advantages.

For external test, scan design is widely used, and has been extended to cope with delay testing. Enhanced Scan, Launch-on-Capture (LOC) and Launch-on-Shift (LOS) are the three most common transition fault pattern generation methods, differing on the way of applying the second vector. Launch-on-Capture is easier to implement but leads to low transition fault coverage. On the other hand, Launch-on-Shift leads to higher transition fault coverage; however, due to the *at-speed* change of the *Scan Enable* signal, it is much difficult to implement.

In this work I addressed the problem of adapting scan BIST to uncover delay faults, which is mandatory for digital SoC implemented in nano-CMOS. The work presented is a preliminary step, and still requires further research. Anyway, in this Dissertation a new methodology for dynamic scan BIST has been proposed. The underlying principle is to apply LOS and LOC techniques to scan BIST. The proposed methodology uses a single architecture – hence, it is modular. However, using the same architecture, if I modify the functionality of a single module – the BIST controller – I am able to implement three new Scan BIST solutions

that implement the transition fault pattern generation methods. They are referred as Scan BIST based on LOC, Scan BIST based on LOS and Scan BIST based on LOS and LOC (which merges the other two techniques). The architecture is composed by five modules: the CUT (reconfigured to accommodate scan insertion), an input MUX, the on-chip TPG (which includes two LFSR: LFSR_PI and LFSR_SCAN), the ORA (using a MISR) and the BIST Controller. The BIST controller, as FSM, has one additional state – the LAUCH state – as compared to the traditional scan BIST controller. In Scan BIST based on LOC approach, the BIST Controller act in a way that the test_se signal goes low during LAUNCH and CAPTURE states. In Scan BIST based on LOS, it only goes low in CAPTURE state. The third proposed solution allows the two TF detection techniques, by switching only one input signal: if the BISTstart control signal is at high level, LOS is performed; if it is at low level, the LOC approach is performed. All three solutions have approximately the same total silicon area, performance degradation and pin count although the hardware changes. Performance degradation is similar to the one of classic scan BIST – basically, the additional propagation delays associated with the input MUX, and with the replacement of the CUT's flip-flops by scan flip-flops.

In order to analyse the three solutions associated with the single architecture, two issues should be evaluated: first, the costs and benefits of the new Dynamic BIST proposed architectures (associated to Test Overhead, TO) and secondly the Test Effectiveness (TE), measured by the ability of each solution to lead to high transition fault coverage.

The associated costs and benefits of the new approaches are similar to the traditional Scan BIST. The proposed solutions introduce pin overhead, because three pins must be added: (1) the BISTstart pin, to control whether BIST operates or not, (2) the BISTdone terminal and (3) the MISR_out pin. This is the same pin overhead that a traditional Scan BIST approach introduces. At the same time, the MUX (and scan insertion in the CUT) introduce performance penalty, similar to the one of classic scan BIST. The area overhead is other critical problem: due to the fact that I implemented a structure that requires one more state in the BIST Controller (LAUNCH), the part of the module that requires now 3 flip-flops (up to 8 states), instead of 2 flip-flops (the 4 states of the classic scan BIST) suffers an area enhancement of about 47% which provokes an increase of about 18% of area in the Dynamic Scan BIST architectures (as compared to the traditional BIST). Area overhead in the BIST Controller, however, may not be very significant, as a relevant part of the silicon area is occupied by the two counters. Other costs are related to the increase in power consumption

during test and the increased design effort and time overhead to perform self-test. As far as I know, test power should be the key attribute that needs to be carefully monitored and improved, as will be discuss it in the following sub-section.

Associated benefits are the reduction of maintenance and test costs, the lower test pattern generation cost and the reduced need of external test equipment. At the same time, it has the advantage of perform *at-speed* test and lead to higher fault coverage.

In terms of transition fault coverage, the Scan BIST based on LOS and LOC architecture leads to the best coverage results, while the lower TF coverage is obtained by Scan BIST based on LOC. This confirms that similar results, obtained with scan design and external test, also apply to built-in self test. Despite of the area overhead related to the implemented BIST Controller and the two LFSR used, the proposed solutions, especially the Scan BIST based on LOS and LOC, are promising solutions, leading to good transition fault coverage results.

One relevant conclusion of the simulation results is that pseudo-random test patterns can do nicely to uncover a significant subset of the transition faults. In fact, with an area overhead of 18% it is obtained, by applying pseudo-random test patterns, a transition fault coverage around 70% (depending on the used CUT and on the *degree of polynomial* of the used LFSR test generator). At the same time, with a small area increase of 1.2% I can obtain a better transition fault coverage with Scan BIST based on LOS and LOC implementation.

Of course, transition fault coverage critically depends on circuit functionality (and topology), on the type of test (deterministic, or PR) and on the LFSR test generator used. Probably the results obtained with presented implementation would be better if the pseudo-random test patterns would be generated by an LFSR with a higher *degree of polynomial*. This, however, increases area overhead.

An additional feature of the proposed solutions is Test Length, TL. In order to enhance the transition fault coverage using pseudo-random test patterns requires a large amount of test vectors, thus a time overhead to perform test. This also boosts fault simulation costs. In terms of time application of the BIST session, a large amount of test vectors may not be a serious drawback, as I apply them at-speed.

5.1. Future Work

As stated, I addressed in this work the problem of adapting scan BIST to uncover delay faults, which is mandatory for digital SoC implemented in nano-CMOS. I am aware that the reported work is a preliminary step, and still requires further research. Anyway, I believe it can be a relevant contribution for establishing a cost-effective dynamic scan BIST methodology. In the following, several directions of future work are outlined.

First, work must be carried out to evaluate power consumption during the BIST sessions. During the at-speed self-test session, it can be much higher than in the normal operation. This issue deserves further attention, as in dynamic scan design, much of the test process operates at low speed, and the test vectors sequences, generated to uncover delay faults, are applied at-speed. It is planed to carry out this task, using the *VeriDOS* tool and computing the weighted switching activity. I realize that test power may exceed the power consumption associated with normal operation. Hence, new improvements (probably associated with the use of two clock signals, a low speed and an at-speed clock), may be required.

Second, as stated, test effectiveness is a mandatory requirement. The limited TF coverage results obtained with only PR test sets may require, as it happens in scan BIST solutions to uncover static faults, the injection of a limited subset of deterministic test vectors (in this case, deterministic test vector *pairs*).

Third, the coverage of dynamic faults must be obtained without jeopardizing the coverage of static faults. Hence, fault simulation results to compute the fault coverage results for static faults must be carried out. I believe that this may not be a problem. Nevertheless, data must be collected to assert this. An additional line of work could be to use the deterministic scan BIST solutions available (in terms of the definition of the hybrid test sets), and evaluate the coverage of dynamic faults. If needed, a small subset of additional test vector pairs would then be generated and added. More accurate results can be obtained if the fault simulation process uses post-layout information, namely the capacitive effects that induce time delays.

Fourth, the proposed methodology (and its correspondent design flow) needs to be fully automated, if it is to be used with complex designs. Hence, design flow automation must be

completed to automate the process step in which, after automatic scan insertion, the scan-based BIST modules (described at RTL in VHDL) are interconnected with the CUT.

References

[1] M. L. Bushnell, V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, 2000.

[2] Optical Proximity Correction (OPC), http://www.synopsys.com/products/ntimrg/opc_ds.html 2005.

[3] Angela Krstic, Tim Cheng. *Delay Fault Testing for VLSI Circuits*. Kluwer Academic Publishers, Boston, 1998.

[4] Charles E. Stroud. *A Designer's Guide to Built-In Self-Test*. Kluwer Academic Publishers, Boston, United States of America, 2002.

[5] M. Rodriguez-Irago, J. J. Rodriguez Andina, F. Vargas, M. B. Santos, I. C. Teixeira, J. P. Teixeira. "Multiple-Clock and Multi-V_{DD} Schemes for Dynamic Fault Detection in Nanometer Digital Circuits". Proc. DCIS, 2005.

[6] B. Bennetts, B. Kruseman. "Tutorial 2: Delay-Fault Testing: From Basics to ASICs". Eleventh IEEE European Test Symposium (ETS), 2006.

[7] J. Savir, S. Patil. "Scan-Based Transition Test". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 12, Issue 8, pp. 1232-1241, August 1993.

[8] Testing of ULSI Circuits, http://www.caip.rutgers.edu/~bushnell/testwebsite 2007

[9] Nur A. Touba. "Synthesis Techniques for Pseudo-Random Built-In Self-Test". Stanford University, June 1996.

[10] Nur A. Touba, E. J. McCluskey. "Altering a Pseudo-Random Bit Sequence for Mixed-Mode Scan BIST". 3rd. International Test Synthesis Workshop, Stanford University, 1996.

[11] B. Dervisoglu, G. Stong. "Design for Testability: Using Scan path Techniques for Path-Delay Test and Measurement", Proc. IEEE International Test Conference (ITC), pp. 365-374, 1991.

[12] Nisar Ahmed, C. P. Ravikumar, M. Tehranipoor, J. Plusquellic. "At-Speed Transition Fault Testing with Low Speed Scan Enable". Proc. IEEE VLSI Test Symposium (VTS), pp. 42-47, 2005.

[13] Nisar Ahmed, M. Tehranipoor, C. P. Ravikumar. "Enhanced Launch-off-Capture Transition Fault Testing". Proc. IEEE International Test Conference (ITC), pp. 246-255, 2005.

[14] Gefu Xu, A. D. Singh. "Low Cost Launch-on-Shift Delay Test with Slow Scan Enable". Proc. 11th IEEE European Test Symposium (ETS), pp. 9-14, 2006.

[15] Gefu Xu, A. D. Singh. "Delay Test Scan Flip-flop: DFT for High Coverage Delay Testing". Proc. 20th. IEEE International Conference on VLSI Design, pp. 763-768, 2007.

[16] Tetramax ATPG Users Guide, "SYNOPSYS Toolset Version Y-2006.06". Synopsys, Inc., June 2006.

[17] Itai Yarom. "Test Pattern Generation for Sub-Micron Designs". SNUG Israel, 2005.

[18] ITC'99 benchmarks, available at http://www.cad.polito.it/tools

[19] http://fidelio.inesc-id.pt/dft/

[20] S. Hellebrand, S. Tarnick, J. Rajski, B. Courtois. "Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers". Proc. Int. Test Conference (ITC), pp. 120-129, 1992.

[21] H.-J. Wunderlisch, G. Kiefer, "Bit-flipping BIST". Proc. Int. Conf. On Computer-aided Design (ICCAD), pp. 337-343, 1996.

[22] N. A. Touba, E. J. McCluskey. "Altering a Pseudo-random Bit Sequence for Scan-based BIST". Proc. IEEE Int. Test Conf. (ITC), pp. 167-175, 1996.

[23] S. Swaminathan, K. Chakrabarty. "A deterministic scan-BIST architecture with application to field testing of high-availability systems". Proc. IEEE Custom Integrated Circuits Conference, pp. 259-262, May 2001.

[24] P. Girard. "Survey of Low-Power Testing of VLSI Circuits". IEEE Design & Test of Computers, vol. 19, nº 3, pp.82-92, May-June 2002.

[25] J. Semião, J. Freijedo, J. J. Rodríguez-Andina, F. Vargas, M. B. Santos, I. C. Teixeira, J. P. Teixeira. "On-line Dynamic Delay Insertion to Improve Signal Integrity in Synchronous Circuits". Proc. of the IEEE International On-Line Test Symposium (IOLTS), pp. 167-171, July, 2007.

[26] H. D. Schnurmann, E. Lindbloom, R.G. Carpenter. "The Weighted Random Test-Generator". IEEE Trans. Computers, vol. 24, nº 7, pp.695-700, July 1975.

[27] A. Parreira, J. Paulo Teixeira, M. B. Santos. "Built-In Self-Test Quality Assessment Using Hardware Fault Emulation in FPGAs". Computing and Informatics Journal (CAI), vol. 23, pp. 537-556, 2005.

[28] P. Girard, C. Landrault, V. Moreda, S. Pravossoudovitch, A. Virazel. "A BIST structure to Test Delay Faults in a Scan Environment". Proc. of the 7th IEEE Asian Test Symposium (ATS'98), pp. 435-439, December 1998.

[29] Yung-Chieh Lin Feng Lu Kwang-Ting Cheng. "Pseudo-Functional Scan-based BIST for Delay Fault". Proc. of the 23rd IEEE VLSI Test Symposium (VTS'05), May 2005.

[30] V. Gherman, H.-J. Wunderlich, J. Schloeffel and M. Garbers. "Deterministic Logic BIST for Transition Fault Testing". Proc. of the 11th IEEE European Test Symposium (ETS'06), May 2006.

ANNEXES

VHDL Code of Scan BIST Architecture based on LOS and LOC

BIST Controller

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.ALL;

entity BIST_CTRL IS
 port(BISTstart, clk, reset: in std_logic; BISTdone, test_se, Reset_LFSR, Enable_LFSR_SCAN, Enable_LFSR_PI, Reset_MISR, Enable_MISR, mux_sel, Reset_CUT: out std_logic);
end BIST_CTRL;

architecture comportamento_BIST_CTRL of BIST_CTRL is

type estados is (IDLE, RES, SCAN, LAUNCH, CAPTURE);

CONSTANT NUM_BITS_COUNTER_VEC: natural := 4;
CONSTANT FINAL_VECTOR_COUNT: std_logic_vector
(NUM_BITS_COUNTER_VEC-1 downto 0) := "1111";
CONSTANT INITIAL_VECTOR_COUNT: std_logic_vector
(NUM_BITS_COUNTER_VEC-1 downto 0) := "0000";

CONSTANT NUM_BITS_COUNTER_SCAN: natural := 4;
```

```
CONSTANT FINAL_SCAN_COUNT: std_logic_vector (NUM_BITS_COUNTER_SCAN-1 downto 0) := "1001";
CONSTANT INITIAL_SCAN_COUNT : std_logic_vector (NUM_BITS_COUNTER_SCAN-1 downto 0) := "0000";

signal contador_vect: std_logic_vector (NUM_BITS_COUNTER_VEC-1 DOWNTO 0);
signal contador_scan: std_logic_vector (NUM_BITS_COUNTER_SCAN-1 DOWNTO 0);
signal scan_finished, vect_finished, cvect_enable, reset_cscan, reset_cvect: std_logic;
signal estado, estado_seg: estados;

begin
 C_SCAN:process (reset_cscan,clk)
 begin
  if reset_cscan = '0' then
        contador_scan <= INITIAL_SCAN_COUNT;
  elsif (clk'event and clk='1') then
        contador_scan <= contador_scan + 1;
  end if;
 end process C_SCAN;

 C_VECT:process (reset_cvect,clk)
 begin
  if reset_cvect = '0' then
        contador_vect <= INITIAL_VECTOR_COUNT;
  elsif (clk'event and clk='1') then
     if (cvect_enable = '1') then
         contador_vect <= contador_vect + 1;
```

```
        else
          contador_vect <= contador_vect;
        end if;
  end if;
end process C_VECT;

comb1: process (contador_scan, contador_vect)
begin
    if contador_scan = FINAL_SCAN_COUNT then
      scan_finished <= '1';
    else
      scan_finished <= '0';
    end if;
    if contador_vect = FINAL_VECTOR_COUNT then
      vect_finished <= '1';
    else
      vect_finished <= '0';
    end if;
end process comb1;

saidas_comb: process (estado, estado_seg)
begin
       when IDLE =>
      reset_cscan <= '1';
      reset_cvect <= '1';
      cvect_enable <= '0';
```

```
  test_se <= '0';
  Reset_LFSR <= '1';
  Enable_LFSR_SCAN <= '0';
  Enable_LFSR_PI <= '0';
  Enable_MISR <= '0';
  BISTdone <= '1';
  Reset_CUT <= '0';
       if estado_seg = RES then
         reset_cscan <= '0';
         reset_cvect <= '0';
     Reset_LFSR <= '0';
     Reset_MISR <= '0';
     BISTdone <= '0';
     Reset_CUT <= '1';
      end if;

when RES =>
  reset_cscan <= '1';
  reset_cvect <= '0';
  cvect_enable <= '0';
  test_se <= '1';
  Reset_LFSR <= '1';
  Enable_LFSR_SCAN <= '1';
  Enable_LFSR_PI <= '0';
  Reset_MISR <= '1';
  Enable_MISR <= '1';
```

```
  BISTdone <= '0';
  mux_sel <= '1';
  Reset_CUT <= '0';

when SCAN =>
  reset_cscan <= '1';
  reset_cvect <= '1';
  cvect_enable <= '0';
  test_se <= '1';
  Reset_LFSR <= '1';
  Enable_LFSR_SCAN <= '1';
  Enable_LFSR_PI <= '0';
  Reset_MISR <= '1';
  Enable_MISR <= '1';
  BISTdone <= '0';
  mux_sel <= '1';
  Reset_CUT <= '0';

  if estado_seg = LAUNCH and BISTstart = '1' then
    test_se <= '1';
    Enable_LFSR_SCAN <= '0';
    Enable_LFSR_PI <= '1';
    Enable_MISR <= '0';
       elsif estado_seg = LAUNCH and BISTstart = '0' then
       test_se <= '0';
    Enable_LFSR_SCAN <= '0';
```

```
    Enable_LFSR_PI <= '1';
    Enable_MISR <= '0';
       elsif estado_seg = IDLE then
    test_se <= '0';
    Enable_LFSR_SCAN <= '0';
    Enable_MISR <= '0';
    BISTdone <= '1';
    mux_sel <= '0';
    Reset_CUT <= '1';
       end if;

when LAUNCH =>
  reset_cscan <= '0';
  reset_cvect <= '1';
  cvect_enable <= '1';
  test_se <= '0';
  Reset_LFSR <= '1';
  Enable_LFSR_SCAN <= '0';
  Enable_LFSR_PI <= '1';
  Reset_MISR <= '1';
  Enable_MISR <= '0';
  BISTdone <= '0';
  mux_sel <= '1';
  Reset_CUT <= '0';

when CAPTURE =>
```

```
  reset_cscan <= '1';
  reset_cvect <= '1';
  cvect_enable <= '0';
  test_se <= '1';
  Reset_LFSR <= '1';
  Enable_LFSR_SCAN <= '1';
  Enable_LFSR_PI <= '0'; '
  Reset_MISR <= '1';
  Enable_MISR <= '1';
  BISTdone <= '0';
  mux_sel <= '1';
  Reset_CUT <= '0';

when others =>
  reset_cscan <= '1';
  reset_cvect <= '1';
  cvect_enable <= '0';
  test_se <= '0';
  Reset_LFSR <= '1';
  Enable_LFSR_SCAN <= '0';
  Enable_LFSR_PI <= '0';
  Reset_MISR <= '1';
  Enable_MISR <= '0';
  BISTdone <= '1';
  mux_sel <= '0';
  Reset_CUT <= '0';
```

```
  end case;
end process saidas_comb;

CTRL_comb:process (BISTstart, estado, scan_finished, vect_finished)
begin
    case estado is
      when IDLE =>
        if BISTstart = '1' then
              estado_seg <= RES;
            else
              estado_seg <= IDLE;
            end if;
          when RES =>
            estado_seg <= SCAN;
        when SCAN =>
            if scan_finished = '1' and vect_finished = '0' then
              estado_seg <= LAUNCH;
            elsif scan_finished = '1' and vect_finished = '1' then
              estado_seg <= IDLE;
            else
         estado_seg <= SCAN;
            end if;
      when LAUNCH =>
            estado_seg <= CAPTURE;
      when CAPTURE =>
            estado_seg <= SCAN;
```

```
        when others =>
          estado_seg <= IDLE;
      end case;
  end process CTRL_comb;

  CTRL_seq:process (clk,reset)
  begin
    if reset='1' then
      estado <= IDLE;
    elsif clk'event and clk='1' then
      estado <= estado_seg;
    end if;
  end process CTRL_seq;

end comportamento_BIST_CTRL;
```

LFSR

```
--LFSR-SCAN-CHAIN-----------------------
library IEEE;

use IEEE.std_logic_1164.all;

entity LFSR_SCAN_11 is
port(
      Enable: in std_logic;
      Clock: in std_logic;
```

```
    Reset: in std_logic;
    Data_out: out std_logic);
end LFSR_SCAN_11;

architecture Comportamento of LFSR_SCAN_11 is
  signal Qin: std_logic_vector(10 downto 0);
  signal Qout: std_logic_vector(10 downto 0);
begin
  comb_TPG: process(Enable, Qout)
  begin
    if Enable = '0' then
      Qin <= Qout;
    else
      Qin(0) <= Qout(1);
      Qin(1) <= Qout(2);
      Qin(2) <= Qout(3);
      Qin(3) <= Qout(4);
      Qin(4) <= Qout(5);
      Qin(5) <= Qout(6);
      Qin(6) <= Qout(7);
      Qin(7) <= Qout(8);
      Qin(8) <= Qout(9);
      Qin(9) <= Qout(10);
      Qin(10) <= Qout(0) xor Qout(2);
    end if;
  end process;
```

```
 sinc_TPG: process(Clock,Reset)
 begin
  if Reset = '0' then
    Qout <= "00101101001";
  elsif (Clock'event and Clock = '1') then
    Qout <= Qin;
  end if;
 end process;

 Data_out <= Qout(0);

end Comportamento;

--LFSR-PIs------------------------------
library IEEE;

use IEEE.std_logic_1164.all;

entity LFSR_PI_11 is
port(
     Enable: in std_logic;
     Clock: in std_logic;
     Reset: in std_logic;
     Data_out: out std_logic_vector(1 downto 0));
end LFSR_PI_11;
```

```
architecture Comportamento of LFSR_PI_11 is
  signal Qin: std_logic_vector(10 downto 0);
  signal Qout: std_logic_vector(10 downto 0);
begin
  comb_TPG: process(Enable, Qout)
  begin
    if Enable = '0' then
      Qin <= Qout;
    else
      Qin(0) <= Qout(1);
      Qin(1) <= Qout(2);
      Qin(2) <= Qout(3);
      Qin(3) <= Qout(4);
      Qin(4) <= Qout(5);
      Qin(5) <= Qout(6);
      Qin(6) <= Qout(7);
      Qin(7) <= Qout(8);
      Qin(8) <= Qout(9);
      Qin(9) <= Qout(10);
      Qin(10) <= Qout(0) xor Qout(2);
    end if;
  end process;

  sinc_TPG: process(Clock,Reset)
  begin
```

```
    if Reset = '0' then
      Qout <= "11010110010";
    elsif (Clock'event and Clock = '1') then
      Qout <= Qin;
    end if;
  end process;

  Data_out <= Qout(1 downto 0);

end Comportamento;
```

MISR

```
--MISR-----------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity MISR7 is
port(  Data_in: in std_logic;
       Enable: in std_logic;
       Clock: in std_logic;
       Reset: in std_logic;
       Data_out: out std_logic);
end MISR7;

architecture Comportamento of MISR7 is
  signal Qin: std_logic_vector(6 downto 0);
```

```
  signal Qout: std_logic_vector(6 downto 0);
begin
  comb: process(Data_in, Qout, Enable)
  begin
    if Enable = '0' then
      Qin <= Qout;
    else
      Qin(0) <= Qout(1);
      Qin(1) <= Qout(2);
      Qin(2) <= Qout(3);
      Qin(3) <= Qout(4);
      Qin(4) <= Qout(5);
      Qin(5) <= Qout(6);
      Qin(6) <= Qout(0) xor Qout(1) xor Data_in;
    end if;
  end process;

  sinc: process(Clock,Reset)
  begin
    if Reset = '0' then
      Qout <= "1000000";
    elsif Clock'event and Clock = '1' then
      Qout <= Qin;
    end if;
  end process;
```

```
  Data_out <= Qout(0);

end Comportamento;
```

Top Level and MUX

```
library IEEE;

use IEEE.std_logic_1164.all;

entity b06_LOC_lOS is
   port ( eql : in std_logic;
            clk : in std_logic;
            reset, cont_eql : in std_logic;
           cc_mux : out std_logic_vector (2 downto 1);
           uscite : out std_logic_vector (2 downto 1);
           enable_count, ackout : out  std_logic;
           BISTstart: in std_logic;
           BISTdone, misr_serial_out: out std_logic);
end b06_LOC_LOS;

architecture Comportamento of b06_LOC_LOS is

component BIST_CTRL
   port(BISTstart, clk, reset : in std_logic;
       BISTdone, test_se, Reset_LFSR, Enable_LFSR_SCAN, Enable_LFSR_PI, Reset_MISR,
Enable_MISR, mux_sel, Reset_CUT: out std_logic);
```

```
end component;

component LFSR_SCAN_11
port(
    Enable: in std_logic;
    Clock: in std_logic;
    Reset: in std_logic;
    Data_out: out std_logic);
end component;

component LFSR_PI_11 is
port(
    Enable: in std_logic;
    Clock: in std_logic;
    Reset: in std_logic;
    Data_out: out std_logic_vector(1 downto 0));
end component;

component MISR7
port(  Data_in: in std_logic;
     Enable: in std_logic;
     Clock: in std_logic;
     Reset: in std_logic;
     Data_out: out std_logic);
end component;
```

```
component b06
  port( cc_mux : out std_logic_vector (2 downto 1);
        eql : in std_logic;
        uscite : out std_logic_vector (2 downto 1);
        clock : in std_logic;
        test_si, test_se: in std_logic;
        enable_count, ackout : out std_logic;
        reset, cont_eql: in std_logic);
end component;

signal mux_out, pi, pi_lfsr: std_logic_vector(1 downto 0);
signal po: std_logic_vector (5 downto 0);
signal TSI, TSE, Reset_LFSR, Reset_MISR, Enable_LFSR_SCAN, Enable_LFSR_PI,
ResetEnable_LFSR_SCAN_MISR, Enable_MISR, mux_sel: std_logic;
signal SRN_Escan, SRN_Eactual, SR_Escan, SR_Eactual, SR_Eseguinte : std_logic;
signal EScan, Reset_CUT, Res_CUT: std_logic;

begin
U1: BIST_CTRL
  port map ( BISTstart => BISTstart,
    clk => clk,
    reset => reset,
    BISTdone => BISTdone,
    test_se => TSE,
    Reset_LFSR => Reset_LFSR,
    Enable_LFSR_SCAN => Enable_LFSR_SCAN,
```

```
    Enable_LFSR_PI => Enable_LFSR_PI,
    Reset_MISR => Reset_MISR,
    Enable_MISR => Enable_MISR,
    mux_sel => mux_sel,
    Reset_CUT => Reset_CUT);

U2: LFSR_SCAN_11
port map(
    Enable => Enable_LFSR_SCAN,
    Clock => clk,
    Reset => Reset_LFSR,
    Data_out => TSI);

U3: LFSR_PI_11
port map(
    Enable => Enable_LFSR_PI,
    Clock => clk,
    Reset => Reset_LFSR,
    Data_out => pi_lfsr);

U4: MISR7
port map( Data_in => po(5),
     Enable => Enable_MISR,
     Clock => clk,
     Reset => Reset_MISR,
     Data_out => misr_serial_out);
```

```
U5: b06
  port map ( eql => mux_out(0),
        cont_eql => mux_out(1),
        enable_count => po(0),
        ackout => po(1),
        cc_mux => po(3 downto 2),
        uscite => po(5 downto 4),
        clock => clk,
        reset => Res_CUT,
        test_si => TSI,
        test_se => TSE);

mux: process(mux_sel, pi, pi_lfsr)
  begin
        if mux_sel ='0' then
          mux_out <= pi;
        else
          mux_out <= pi_lfsr;
        end if;
  end process;

Res:process (Reset_CUT, reset)
  begin
        Res_CUT <= Reset_CUT or reset;
end process;
```

```
pi(0) <= eql;
pi(1) <= cont_eql;

enable_count <= po(0);
ackout <= po(1);
cc_mux <= po(3 downto 2);
uscite <= po(5 downto 4);

end Comportamento;
```

BIST Testbench for scan chain behaviour

```
`timescale 1ns / 1ps

module TOP_ctl ;

reg [0:3] pattern_mem[0:260022];
reg clock, reset;
wire pin_1_, pin_2_, pin_3_, pin_4_;
wire enable_count, ackout;
wire [2:1] cc_mux;
wire [2:1] uscite;
wire [2:1] state;
wire n65;

integer j, k, index, fileno;

assign {pin_1_, pin_2_, pin_3_, pin_4_} = pattern_mem[index];

b06 TOP_inst ( .test_se(pin_1_), .test_si(pin_2_), .cont_eql(pin_3_), .eql(pin_4_), .reset(reset), .clock(clock), .enable_count(enable_count), .ackout(ackout), .cc_mux(cc_mux), .uscite(uscite), .n65(n65), .state(state));

initial begin

k = 0;
j = 1;
```

```
index = 0;
clock = 0;
#10 clock = 1;
reset = 1;
#10 reset = 0;

$readmemb("bist2_b06_los.vec", pattern_mem);
fileno = $fopen ("/home/sgomes/CADENCE_WORK/BIST_b06_linear/bist2_b06_los.scn");

        forever
           begin
           @(posedge clock)
                 index = index + 1;
                       //k = k +1;
           @(negedge clock)
begin
if (j == 1)
begin
$fdisplay(fileno, "%b%b%b%b%b%b%b%b%b", ackout, cc_mux[1], cc_mux[2], enable_count, n65, state[1], state[2], uscite[1], uscite[2]);
j=2;
end
else
begin
$fdisplay(fileno, "%b%b%b%b%b%b%b%b%b", ackout, cc_mux[1], cc_mux[2], enable_count, n65, state[1], state[2], uscite[1], uscite[2]);
$fdisplay(fileno, "111111111");
```

```
j = 1;


                    end
end
end
$fclose (fileno);
end


always
  #100 clock = ~clock;


initial
   begin
      wait(index > 260022)
        $display("\n");
        $finish;
   end
endmodule
```

Synopsys TetraMax™ TCL Script – External Patterns

```
#read external patterns to perform transition fault coverage
#source /home/sgomes/SYN_WORK/tmax2.tcl

read netlist /opt/ams/ams_v3.70/verilog/udp.v -library
read netlist /opt/ams/ams_v3.70/verilog/c35b4/c35_CORELIB.v -library
read netlist /opt/ams/ams_v3.70/verilog/c35b4/c35_IOLIB_4M.v -library

read netlist /home/sgomes/SYN_WORK/LOS_b06_tmax/b06_scan.v
set learning -verbose
run build_model b06
set drc /home/sgomes/SYN_WORK/LOS_b06_tmax/b06design.spf
run drc
set faults -model transition -atpg_effectiveness -fault_coverage -summary verbose
add faults -all
set patterns external /home/sgomes/SYN_WORK/LOS_b06_tmax/pattern_b06_1 -sensitive
run fault_sim -detected_pattern_storage -detected_pattern_storage -ndetects 1
run atpg -ndetects 1
report summaries memory_usage
```