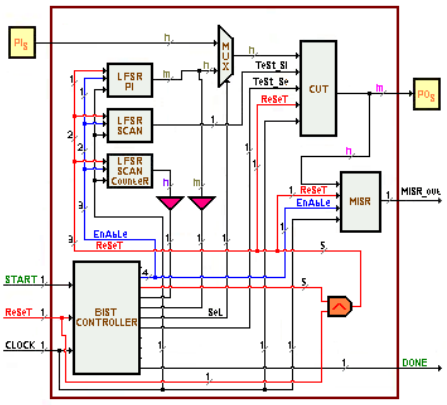
UAlg
UNIVERSIDADE DO ALGARVE

UNIVERSIDADE DO ALGARVE

INSTITUTO SUPERIOR DE ENGENHARIA



Aging Monitoring Methodology for Built-In Self-Test Applications

Metodologia de Monitorização do Envelhecimento para Aplicações de Auto-teste Embutido

João Ricardo dos Santos Coelho

Dissertação para obtenção do Grau de Mestre em

Engenharia Eléctrica e Electrónica

Área de Especialização em Tecnologias de Informação e Telecomunicações

Orientador: Professor Doutor Jorge Filipe Leal Costa Semião

Setembro, 2013

Aging Monitoring Methodology for Built-In Self-Test Applications

Declaração de autoria de trabalho

Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

Assinatura: _____

To my family

ACKNOWLEDGMENTS

This study is not only the result of an individual effort, but rather a set of efforts that made it possible and without them it would have been much more difficult to reach the end of this step, which represents an important milestone in my personal and professional life. Therefore, I express my gratitude to all those who were present at complex times.

To Professor Jorge Semião, in particular, I want to express my thanks for the guidance printed to the whole process, combining the stamp of high scientific standards, an abiding and fruitful interest, which helped to catalyze the present investigation. I also want to highlight the critical, objective and motivated vision, dedicated to the pursuit and constant improvement of this thesis.

To my daughter and to my wife, who during these years have been a constant support and encouragement, I want to express a word of thanks for the consideration, generosity and affection, contributing to tread this path until the end and to all my family, who encouraged me in the decision to start, continue and complete this project, and made me taste the true solidarity, when it showed the complex challenge of ensuring the link between family roles, and professional research.

I thank also to my colleague and friend Engº Vasco Fernandes, for sharing again your motivator character in times of special relevance. And to my colleague and friend Engº Hugo Cavalaria, who helped me in important areas like VHDL, making faster and effective my learning process in a relevant area to the development of this work.

Finally, to all who have provided documentation and miscellaneous information, I also want to leave a word of thanks.

João Ricardo dos Santos Coelho, Faro, September 30th, 2013

ABSTRACT

The high integration level achieved as well as complexity and performance enhancements in new nanometer technologies make IC (Integrated Circuits) products very difficult to test. Moreover, long term operation brings aging cumulative degradations, due to new processes and materials that lead to emerging defect phenomena and the consequence are products with increased variability in their behaviour, more susceptible to delay-faults and with a reduced expected lifecycle.

The main objectives of this thesis are twofold, as explained in the following. First, a new software tool is presented to generate HDL (Hardware Description Language) for BIST (Built-In Self-Test) structures, aiming delay-faults, and inserted the new auto-test functionality in generic sequential CMOS circuits. The BIST methodology used implements a scan based BIST approach, using a new BIST controller to implement the Launch-On-Shift (LOS) and Launch-On-Capture (LOC) delay-fault techniques.

Second, it will be shown that multi-V_{DD} tests in circuits with BIST infra-structures can be used to detect gross delay-faults during on-field operations, and consequently can be used as an aging sensor methodology during circuits' lifecycle. The discrete set of multi-V_{DD} BIST sessions generates a Voltage Signature Collection (VSC) and the presence of a delay-fault (or a physical defect) modifies the VSC collection, allowing the aging sensor capability.

The proposed Design for Testability (DFT) method and tool are demonstrated with extensive SPICE simulation using three ITC'99 benchmark circuits.

Keywords: Built-In Self-Test, Aging Sensor Methodology, Multi-V_{DD} Tests, HDL automatic generation, Launch-On-Shift, Launch-On-Capture.

RESUMO

O elevado nível de integração atingida, complexidade, assim como performances melhoradas em novas tecnologias nanométricas tornam os produtos em circuitos integrados tecnológicos muito difíceis de testar. Para além disso, a operação a longo prazo produz degradações cumulativas pelo envelhecimento dos circuitos, devido a novos processos e materiais que conduzem a novos defeitos e a consequência são produtos com maior variabilidade no seu funcionamento, mais susceptíveis às faltas de atraso e com um tempo de vida menor.

Os principais objectivos desta tese são dois, como explicado em seguida. Primeiro, é apresentada uma nova ferramenta de *software* para gerar estruturas de auto-teste integrado (*BIST, Built-In Self-Test*) descritas em linguagens de descrição de *hardware* (*HDL, Hardware Description Language*), com o objectivo de detectar faltas de atraso, e inserir a nova funcionalidade de auto-teste em circuitos genéricos sequenciais *CMOS*. A metodologia de *BIST* utilizada implementa um procedimento baseado em caminhos de deslocamento, utilizando um novo controlador de *BIST* para implementar técnicas de faltas de atraso, como *Launch-On-Shift* (*LOS*) e *Launch-On-Capture* (*LOC*).

Segundo, irá ser mostrado que testes multi-V_{DD} em circuitos com infra-estruturas de *BIST* podem ser usados para detectar faltas de atraso grosseiras durante a operação no terreno e, consequentemente, pode ser usado como uma metodologia de sensor de envelhecimento durante o tempo de vida dos circuitos. Um número discreto de sessões *BIST* multi-V_{DD} geram uma Colecção de Assinaturas de Tensão (*Voltage Signature Collection*, *VSC*) e a presença de uma falta de atraso (ou um defeito físico) faz modificar a colecção *VSC*, comportando-se como sensor de envelhecimento.

O trabalho foi iniciado com o estudo do estado da arte nesta área. Assim, foram estudadas e apresentadas no capítulo 2 as principais técnicas de *DfT* (*Design for Testability*) disponíveis e utilizadas pela indústria, nomeadamente, as técnicas de *SP* (*Scan Path)*, de *BIST* e as técnicas de *scan* para *delay-faults*, *LOS* e *LOC*. No capítulo 3, ainda referente ao estudo sobre o estado da arte, é apresentado o estudo sobre os

fenómenos que provocam o envelhecimento dos circuitos digitais, nomeadamente o *NBTI* (*Negative Bias Temperature Instability*), que é considerado o factor mais relevante no envelhecimento de circuitos integrados (especialmente em nanotecnologias).

Em seguida, iniciou-se o desenvolvimento do primeiro objectivo. Relativamente a este assunto, começou-se por definir qual o comportamento das estruturas de *BIST* e como se iriam interligar. O comportamento foi descrito, bloco a bloco, em *VHDL* comportamental, ao nível *RTL* (*Register Transfer Level*). Esta descrição foi então validada por simulação, utilizando a ferramenta *ModelSim*. Posteriormente, esta descrição comportamental foi sintetizada através da ferramenta *Synopsys*, com a colaboração do INESC-ID em Lisboa (instituição parceira nestes trabalhos de investigação), e foi obtida uma *netlist* ao nível de porta lógica, que foi guardada utilizando a linguagem de descrição de *hardware Verilog*. Assim, obtiveram-se dois tipos de descrição dos circuitos *BIST*: uma comportamental, em *VHDL*, e outra estrutural, em *Verilog* (esta descrição estrutural em *Verilog* irá permitir, posteriormente, fazer a simulação e análise de envelhecimento).

A nova estrutura de *BIST* obtida é baseada no modelo clássico de *BIST*, mas apresenta algumas alterações, nomeadamente ao nível da geração de vectores de teste e no controlo e aplicação desses vectores ao circuito. Estas modificações têm como objectivo aumentar a detecção de faltas e permitir o teste de faltas de atraso. É composto por três blocos denominados *LFSRs* (*Linear Feedback Shift Registers*), um utilizado para gerar os vectores pseudo-aleatórios para as entradas primárias do circuito, outro para gerar os vectores para a entrada do *scan path*, e o último utilizado como contador para controlar o número de *bits* introduzidos no *scan path*. Relativamente ao controlador, este foi especificamente desenhado para controlar um teste com estratégia de *test-per-scan* (ou seja, um teste baseado no caminho de varrimento existente no circuito) e tem uma codificação de estados que permite implementar as estratégias de teste de faltas de atraso, *Launch-On-Shift* (*LOS*) e *Launch-On-Capture* (*LOC*). Na secção de saída do novo modelo de *BIST*, o processo de compactação usa o mesmo princípio do modelo tradicional, utilizando neste caso um *MISR* (*Multiple Input Signature Register*).

Ainda relativamente ao primeiro objectivo, seguiu-se o desenvolvimento da ferramenta *BISTGen*, para automatizar a geração das estruturas de *BIST* atrás

mencionadas, nos dois tipos de descrição, e automaticamente inserir estas estruturas num circuito de teste (*CUT, Circuit Under Test*). A aplicação de *software* deve permitir o manuseamento de dois tipos de informação relativa ao circuito: descrição do circuito pelo seu comportamento, em *VHDL*, e descrição do circuito pela sua estrutura, em *Verilog*. Deve ter como saída a descrição de *hardware* supra citada, inserindo todos os blocos integrantes da estrutura num só ficheiro, contendo apenas um dos tipos de linguagem (*Verilog* ou *VHDL*), escolhida previamente pelo utilizador. No caso dos *LFSRs* e do *MISR*, o programa deve permitir ao utilizador a escolha de *LFSRs* do tipo linear ou do tipo modular (também conhecidos por *fibonacci* ou *galois*), e deve também possuir suporte para automaticamente seleccionar de uma base de dados quais as realimentações necessárias que conduzem à definição do polinómio primitivo para o *LFSR*. Será necessário ainda criar uma estrutura em base de dados para gerir os nomes e o número de entradas e saídas do circuito submetido a teste, a que chamamos *CUT*, de forma a simplificar o processo de renomeação que o utilizador poderá ter de efectuar. Dar a conhecer ao programa os nomes das entradas e saídas do *CUT* é de relevante importância, uma vez que a atribuição de nomes para as entradas e saídas pode vir em qualquer língua ou dialecto, não coincidindo com os nomes padrão normalmente atribuídos.

Relativamente às duas linguagens que o programa recebe através do *CUT* na sua entrada, no caso *VHDL* após inserir *BIST* o ficheiro final terá sempre uma estrutura semelhante, qualquer que seja o ficheiro a ser tratado, variando apenas com o *hardware* apresentado pelo *CUT*. No entanto, para o caso *Verilog* a situação será diferente, uma vez que o programa tem de permitir que o ficheiro final gerado possa surgir de duas formas dependendo da escolha desejada. A primeira forma que o *software* deve permitir para o caso *Verilog* é gerar um ficheiro contendo módulos, de uma forma semelhante ao que acontece no caso *VHDL*. No entanto, deve permitir também a obtenção, caso o utilizador solicite, de um ficheiro unificado, sem sub-módulos nos blocos, para que o ficheiro final contenha apenas uma única estrutura, facilitando a sua simulação e análise de envelhecimento nas etapas seguintes.

Relativamente ao segundo objectivo, com base no trabalho anterior já efectuado em metodologias para detectar faltas de *delay* em circuitos com *BIST*, foi definida uma metodologia de teste para, durante a vida útil dos circuitos, permitir

avaliar como vão envelhecendo, tratando-se assim de uma metodologia de monitorização de envelhecimento para circuitos com *BIST*.

Um aspecto fundamental para a realização deste segundo objectivo é podermos prever como o circuito vai envelhecer. Para realizar esta tarefa, sempre subjectiva, utilizou-se uma ferramenta desenvolvida no ISE-UAlg em outra tese de mestrado anterior a esta, a ferramenta *AgingCalc*. Esta ferramenta inicia-se com a definição, por parte do utilizador, das probabilidades de operação das entradas primárias do circuito (probabilidades de cada entrada estar a '0' ou a '1'). De notar que este é o processo subjectivo existente na análise de envelhecimento, já que é impossível prever como um circuito irá ser utilizado. Com base nestas probabilidades de operação, o programa utiliza a estrutura do circuito para calcular, numa primeira instância, as probabilidades dos nós do circuito estarem a '0' ou a '1', e numa segunda instância as probabilidades de cada transístor *PMOS* estar ligado e com o seu canal em stress (com uma tensão negativa aplicada à tensão V_{GS} e um campo eléctrico aplicado ao dieléctrico da porta). Utilizando fórmulas definidas na literatura para modelação do parâmetro *Vth* (tensão limiar de condução) do transístor de acordo com um envelhecimento produzido pelo efeito *NBTI* (*Negative Bias Temperature Instability*), o programa calcula, para cada ano ou tempo de envelhecimento a considerar, as variações ocorridas no *Vth* de cada transístor *PMOS*, com base nas probabilidades e condições de operação previamente definidas, obtendo um novo *Vth* para cada transístor (os valores prováveis para os transístores envelhecidos). Em seguida, o programa instancia o simulador *HSPICE* para simular as portas lógicas do circuito, utilizando uma descrição que contém os *Vth* calculados. Esta simulação permite calcular os atrasos em cada porta para cada ano de envelhecimento considerado, podendo em seguida calcular e obter a previsão para o envelhecimento de cada caminho combinatório do circuito. É de notar que, embora a previsão de envelhecimento seja subjectiva, pois depende de uma previsão de operação, é possível definir diferentes probabilidades de operação de forma a estabelecer limites prováveis para o envelhecimento de cada caminho.

Tendo uma ferramenta que permite prever como o circuito irá envelhecer, é possível utilizá-la para modificar a estrutura do circuito e introduzir faltas de *delay* produzidas pelo envelhecimento por *NBTI* ao longo dos anos de operação (modelados pelo *Vth* dos transístores *PMOS*). Assim, no capítulo 5 irá ser mostrado que testes multi-V_{DD} em circuitos com infra-estruturas de *BIST* podem ser usados para detectar

faltas de atraso grosseiras durante a operação no terreno, podendo em alguns casos identificar variações provocadas pelo envelhecimento em caminhos curtos, e consequentemente, estes testes podem ser usados como uma metodologia de sensor de envelhecimento durante o tempo de vida dos circuitos. Um número discreto de sessões *BIST* multi-V_{DD} geram uma Colecção de Assinaturas de Tensão (*Voltage Signature Collection*, *VSC*) e a presença de uma falta de atraso (ou um defeito físico) faz modificar a colecção *VSC*, comportando-se como sensor de envelhecimento. O objectivo será, especificando, fazer variar a tensão de alimentação, baixando o seu valor dentro de um determinado intervalo e submetendo o circuito a sucessivas sessões de *BIST* para cada valor de tensão, até que o circuito retorne uma assinatura diferente da esperada. Este procedimento de simulação será feito para uma maturidade de até 20 anos, podendo o incremento não ser unitário. Na realidade os circuitos nos primeiros anos de vida em termos estatísticos não sofrem envelhecimento a ponto de causar falhas por esse efeito. As falhas que podem acelerar o processo de envelhecimento estão relacionadas com defeitos significativos no processo de fabrico mas que ainda assim não são suficientes para no início do seu ciclo de vida fazer o circuito falhar, tornando-se efectivas após algum tempo de utilização.

Os métodos e ferramentas propostos de *DfT* são demonstrados com extensas simulações *VHDL* e *SPICE*, utilizando circuitos de referência.

Palavras-chave: Auto-Teste Incorporado, Metodologia para Sensor de Envelhecimento, Testes Multi-V_{DD}, geração automática de *HDL*, *Launch-On-Shift*, *Launch-On-Capture*.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ASIC	Application Specific Integrated Circuit
ATE	Automatic Test Equipment
BIST	Built-In Self-Test
CAD	Computer Aided Design
CHC	Channel Hot Carrier
CRC	Cyclic Redundancy Check
CUT	Circuit Under Test
DFT	Design for Testability
DVS	Dynamic Voltage Scaling
EDA	Electronic Design Automation
FSM	Finite State Machine
HDL	Hardware Description Language
HTOL	High Temperature Operating Life
ICs	Integrated Circuits
JTAG	Joint Test Action Group
LFSR	Linear Feedback Shift Register
MISR	Multiple Input Signature Register
NBTI	Negative Bias Temperature Instability
ORA	Output Response Analysis
RM	Response Monitor
RTL	Register Transfer Level
SIV	Stress Induced Voiding
SoC	System on Chip
SRAM	Static Random Access Memory
STF	Slow to Fall
STR	Slow to Rise
TDDB	Time Dependent Dielectric Breakdown
TF	Transition Fault
TG	Test Generator

VHDL Very high speed integrated circuits (VHSIC) Hardware Description Language

VHSIC Very High Speed Integrated Circuits

VLSI Very Large Scale Integration

VSC Voltage Signature Collection

LIST OF DEFINITIONS

[*Aliasing*] – During circuit response compaction, because of the information loss, it is possible that a signature of a bad circuit may match the good circuit signature, which is called aliasing. In such cases, a failing circuit will pass the testing process.

[*Compaction*] – A method of drastically reducing the number of bits in the original circuit response during testing in which some information is lost.

[*Compression*] – A method of reducing the number of bits in the original circuit response during testing in which no information is lost, so the original output sequence can be fully regenerated from the compressed sequence.

[*Delay Fault*] – A delay-fault is a fault that causes the combinational delay of a circuit to exceed the clock period.

[*Negative Bias Temperature Instability*] – Translate an increase in the absolute threshold voltage causing a degradation of the mobility, drain current and transconductance of P-channel MOSFETs. It is almost universally attributed to the creation of interface traps and oxide charge by a negative gate bias at elevated temperature.

[*Path Delay Fault*] – A delay defect in a circuit is assumed to cause the cumulative delay of a combinational path to exceed some specified duration. The combinational path begins at a primary input or a clocked flip-flop, contains a connected chain of gates, and ends at a primary output or a clocked flip-flop. The specified time duration can be the duration of the clock period (or phase), or the vector period. The propagation delay is the time that a signal event (transition) takes to traverse the path. Both switching delays of devices and transport delays of interconnects on the path, contribute to the propagation delay.

[*Signature*] – A statistical property of a circuit, usually a number computed for a circuit from its responses during testing, with the property that faults in the circuit usually cause the signature to deviate from the signature of the non-faulty circuit.

[*Signature Analysis*] – A method of circuit response compaction during testing, whereby the entire good circuit response is compacted into a good circuit signature. The actual circuit signature is generated during the testing process on the CUT, and then compared with the good machine signature to determine whether the CUT is faulty.

[*Transition Delay Fault Model*] – "It is assumed that in the fault-free circuit all gates have some nominal delay and the delay of a single gate has changed. The gate-delay, usually an increase over the nominal value, is assumed to be large enough to prevent a passing transition from reaching any output within the clock period, even when the transition propagates through the shortest path. Possible transition faults of a gate are slow to-rise and slow-to-fall types and hence the total number of transition faults is twice the number of gates. Transition faults model spot defects and are also called gross-delay-faults" (excerpted from [27]).

1. INTRODUCTION

Electronic systems have increased its complexity in the last years in nano technologies, which leads to a growth of system functionalities integrated in a single chip. High performance applications with Integrated Circuits (IC) are commonly found in the networking, banking, aerospace/defence, automotive, computer, telecommunications and healthcare industries, and have greatly increased in usability and complexity. Such, evolution requires additional fault control in the test environment, as testing of IC has a crucial importance to ensure a high level of quality in product functionality. Due to the increased complexity in modern ICs, the impact of testing affects both IC design and manufacturing. Moreover, given this range of design involvement, a major concern is, definitely, how to achieve a high level of confidence in IC operation and this desire to attain high quality levels, conflicts with the demand for reduced costs and shorten time involved in the development process. These two design considerations are at constant odds.

The traditional solution to achieve a high level of confidence is ruled by advanced testers denominated Automated Test Equipment (ATE). Traditionally ATE's cost is only measured using a simple digital cost pin approach which leads to a lack of considerations making the cost per-test in many ways disproportionate. In the last years other calculations have been made and proposed [1] to improve the traditional test cost measurement, considering also base system costs associated with equipment infrastructure, central instruments and the beneficial scaling that occurs with increasing pin count. As an example, Figure 1 shows the test cost evolution vs. manufacturing cost in the last 30 years.

Therefore, it became essential to find/implement alternative test methods to reduce financial costs. Among these methods is Built-In Self-Test (BIST), and has become a major design consideration in Design for Testability (DFT) methods. BIST has many advantages. This technique can drastically reduce the external test equipment dependency. If external test equipment is a part of the enterprise legacy, BIST will reduce the global cost and test time even more, making possible to re-direct the test equipment towards other devices in the current design, if necessary.

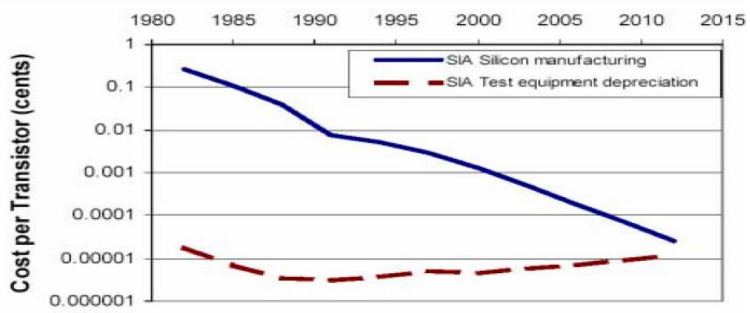


Figure 1: Test Cost vs Manufacturing Cost (From Semiconductor Industry Association [2])

Moreover, new technology products need high speed testers, not always available, as ATE is usually a few years behind the latest technology products. Considering that testing represents a key cost factor in the production process (up to 70% of total product cost is reported in [3] [4] [5]), an optimal test strategy can be a substantial competitive advantage in a market comprising billions of electronic components and systems. It is therefore not a surprise that the International Technology Roadmap for Semiconductors (ITRS), in its last report (2012) has placed the design for self-test on the future opportunities in the "Test and Test Equipment" group report [6].

Another important advantage is that BIST allows not only circuit tests during production, but also to test the circuits during their entire lifetime, which is an important feature when long-term degradation effects start to limit circuits expected life-cycle for nanotechnology ICs. This opens a new concept and a new era in system quality and testing. In addition, BIST can overcome pin limitations due to packaging, make efficient use of available extra chip area, and provide more detailed information about the faults present.

The main disadvantages for BIST usability are, commonly, the increased die size and design complexity. However, the addition of BIST features to IC design nowadays doesn't significantly increase a product's size, cost, and production time, as was the case in the past. All the benefits are plentiful motivations for BIST technique to become an important DFT technique in the future.

The present work deals with the automatic generation of BIST structures and studies its behaviour during circuit's expected lifetime, using statistical predictions for aging degradations. The accelerated aging effects observed in new technologies ICs are also a motivation to develop new techniques to enhance circuit's reliability. In fact, aging effects caused by phenomena like Negative Bias Temperature Instability

(NBTI) (the dominant long-term effect in nanometer CMOS technologies [72]), Hot Carrier Injection (HCI), or Time Dependent Dielectric Breakdown (TDDB), among others, are gaining increase relevance in new nanometer technologies and degrade circuit performance over time [73]. These aging effects are cumulative and cause circuit's safety margins (time slack) to be shrinked, reducing the expected circuit's life time. Therefore, new technology products have a smaller expected lifetime than previous technology's products, imposing the need for auto-test during on-field operation (and not only in the production stage), along circuit's lifecycle.

1.1 OBJECTIVES

With the previous motivations in mind, this work tries to put a milestone in the development of ICs with BIST capability. The goal is to develop automatic BIST structures for generic sequential ICs, aiming the detection of delay-faults, and re-use on-chip variable power supply voltage source to implement an aging aware test strategy to detect long-term degradations during circuit's lifetime.

The objectives for this work are, mainly, twofold:

1. Implement a software tool to generate BIST structures automatically in a circuit under test (CUT), aiming the detection of delay-faults;
2. Show that a set of auto-tests using a variable V_{DD} power-supply voltage source (a set of BIST runs, each run using a different power-supply voltage value) can be used as an aging and performance sensor for long-term degradations (during circuits' lifespan).

The first main objective is a pre-requisite to the second one. It is important to have a tool to insert in a general sequential CMOS circuit BIST structures to allow the auto-test of the circuit. Starting from a HDL (Hardware Description Language) netlist (or behavioural description), the tool must generate automatically a new HDL netlists (or behavioural description) of the new circuit with BIST structures and functionality. To accomplish this first objective, the BIST structures have to be defined, using VHDL (Very high speed integrated circuits Hardware Description Language) and

Verilog languages, and defining the structures in a behavioural and netlist representation, using a CMOS generic standard cell library designed in a previous M.Sc. thesis in ISE-UAlg). The BIST controller defined should also implement LOS and LOC based BIST approaches, aiming the detection of delay-faults.

The second main objective will use as a test vehicle the BIST structures defined with the proposed software tool (from the first objective), already inserted in a Circuit Under Test (CUT), and the purpose is to show by simulation (SPICE simulations) that using by reusing a variable power-supply already present in the IC, it is possible to identified a set of BIST signatures (known as Voltage Signatures Collection, VSC), from a set of BIST sessions performed each one at a different power-supply voltage. This VSC is unique for each sample circuit, and as aging degradations start to occur during circuit's lifetime, this unique VSC will differ, allowing to detect not only gross delay-faults but also to define an aging sensor methodology for BIST circuits.

1.2 CONTEXT

This research work was conducted at the Instituto Superior de Engenharia (*ISE*), University of Algarve (*UAlg*), in close collaboration with *INESC-ID* Lisbon and with the industrial partner Silicongate in Lisbon. The work team formed in the Portuguese institutions are working in collaboration with other foreigner R&D institutes and universities, namely University of Vigo in Spain, the *INAOE* institute in Mexico and *PUCRS* University in Brazil. The team has been developing in the last 5 years some research work on aging sensors, both for ASIC (Application Specific Integrated Circuit) and for emulated circuits in FPGAs (Field-Programmable Gate Array). Moreover, in this context, 2 M.Sc. thesis were already finished, and another one is currently being developed, in *ISE-UAlg*, and furthermore M.Sc. and Ph.D. thesis were finished and are currently being developed in partner institutions.

1.3 OUTLINE

This thesis is organized as follows:

- Chapter 2 reviews basic concepts on Fault Modelling, conventional BIST methodology and its architecture. Emphasis is placed on scan design for delay-fault detection, namely Launch on Capture and Launch on Shift techniques.
- Chapter 3 outlines the main phenomena and effects that contribute to the aging of digital CMOS integrated circuits like NBTI phenomenon.
- The fourth chapter describes the new proposed dynamic BIST methodology. It gives the details about the new methodology, the proposed BIST architecture and the characteristics of all their structural components.
- Chapter 5 explains the BISTGen Application Software, its composition and hierarchy levels.
- Chapter 6 presents the test results.
- Chapter 7 concludes the work with a summary of the proposed methodology, its achievements and limitations. It also outlines directions for future work.

2. DESIGN FOR TESTABILITY

The design of a feasible system solution for a given problem is only half of the task. Considering that the production stage in the IC design process involves very complex procedures, it is very important to be able to test the system to a degree which ensures a high confidence level that it is fully functional and this is generally not a straight forward task. In very small digital systems scale, it is possible to test it exhaustively, and the system can exercise over its full range of operating conditions. However, in a larger scale system, it is no longer possible to do this procedure and therefore other strategies has to be found to ensure that the system will properly be tested.

When testing a digital logic device, stimulus are applied to its inputs and check its response at the outputs to identify if it is performing correctly. The set of input stimulus is referred as a test pattern. In general, the response of the device is observed at its normal output pins. However, it is possible that the device is specially configured during the test, to allow observing some internal nodes, which generally would not be accessible to the user. The response of the device is evaluated by comparing it to an expected response, which may be obtained by saving the response of a known good device, or using simulation on a computer. If the CUT passes the test, isn't possible to say categorically that it is a good device. The only possible conclusion is that the device does not contain any of the faults for which it was tested. It is important to grasp this point; a device may contain a huge number of potential faults, some of which may even mask each other under specified operating conditions. The designer can only be sure that the device is 100% good if it has been 100% tested, this is rarely possible in real life systems.

2.1 DELAY FAULTS

Physical failures and fabrication defects cannot be easily modeled mathematically. As a result, these failures and defects are modeled as logical faults. **Structural faults** relate to the structural model of a system and affect

interconnections among components of a design. **Functional faults** relate to a functional model, for example an RTL/HDL (Register Transfer Level / Hardware Description Language) model, and these affect the nature of components operation in a design. Testing for functional faults validates the correct operation of a system, while testing of structural faults targets manufacturing defects.

The faults can be **static**, if represent a defect that is always present and is independent of circuit operation and performance, and **dynamic**, if the fault only manifests itself in pre-determined circuit operating conditions and, therefore, it is not always present. **Delay faults** are dynamic faults related with the delay of paths. In other words, if a given timing response is not met, due to a dynamic defect or even due to an excessive clock frequency operation, an error is captured by a memory cell (usually a flip-flop or latch), and is conclusive that a delay-fault occurred.

Two popular structural fault models are prevalent in the industries today which are the **stuck-at fault model** and the **transition fault model**. Stuck-at faults affect the logical behaviour of the system and are a representation of static faults. However, transition faults affect the timing/temporal behaviour of the system and are a representation of dynamic faults. An additional fault model being used is the **path delay-fault model**, which is also based on the timing behaviour of the system, but cumulative delays along paths are considered, instead of delays at each net as in the transition fault model. This previous fault model is also a representation of dynamic faults. Therefore, transition and path delay-fault models are commonly mention as two delay-fault models.

2.1.1 TRANSITION FAULTS

The transition fault model is similar to the stuck-at fault model in many ways. The effect of a transition fault at any P point in a circuit is that any transition at P will not reach a scan flip-flop or a primary output within the stipulated clock period of the circuit. According to the transition fault model [28], there are two types of possible faults on all lines (nodes) in the circuit: a slow-to-rise fault (STR) and a slow-to-fall fault (STF). A slow-to rise fault at a node means that any transition from ‘0’ to ‘1’ on the node does not produce the correct result when the device is operating at its

maximum operating frequency. Similarly, a slow-to fall fault means that a transition from '1' to '0' on a node does not produce the correct result at full operating frequency. In any circuit, the time slack can be defined as the difference between the clock period and the propagation delay of the path under consideration (i.e. the remaining and unused time of the clock period, in signal propagation). For a gate level delay-fault to cause an incorrect value to be latched at a circuit output, the size of the delay-fault must be such that it exceeds the slack of at least one path from the site of the fault to the site of an output pin or flip-flop. If the propagation delays of all paths passing through the fault site exceed the clock period, such a fault is referred to as a gross delay-fault [29].

Any test pattern that successfully detects a transition fault comprises of a pair of vectors {V1, V2}, where V1 is the initial vector that sets a target node to the initial value, and V2 is the next vector that not only launches the transition at the corresponding node, but also propagates the effect of the transition to a primary output or a scan flip-flop [30]. In other words, a set of test vectors that test for a delay-fault at the output or input of a gate are such that:

- A desired transition is launched at the site of the fault
- If the fault is a slow-to rise fault, the final pattern is a test for a corresponding stuck-at-0 fault, and if the fault is a slow-to fall fault, the final pattern is a test for a corresponding stuck-at-1 fault.

When compared with tests for stuck-at faults, it can be seen that the only additional requirement to test for transition faults is the presence of a pattern that initializes a node to the required value, just before the application of a stuck-at fault pattern. One might expect that the fault coverage attained by testing transition fault patterns will be close to that attained by testing stuck-at fault patterns. However, should be remembered that the fault coverage obtained for transition fault patterns represent only gross delay-faults. More detailed analysis will be necessary to evaluate for smaller delay-faults [31].

2.1.2 PATH DELAY FAULTS

The path delay-fault model [34] takes the sum of all delays along a path into effect, while the transition fault model accounts for localized faults (delays) at the inputs and outputs of each gate. There may be cases where the gate delays of individual faults are within specified limits, but the cumulative effect of all faults on a path may cause an incorrect value to be latched at the primary outputs, if the total delay exceeds the functional clock period. The transition fault model cannot account for such defects, but the path delay-fault model can. However, in a design containing n lines, there can be a maximum on $2 \times n$ transition faults (a slow-to rise and slow-to fall fault on each line), but there can potentially be 2^n path delay-faults (considering all possible paths) [29]. Since all the paths cannot be tested, the path delay model requires identification and analysis of critical paths in the design. This makes it more complicated to use on large designs and hence, the transition fault model has been accepted as a good method to test for delay-faults in the industry [35] [36].

2.2 DFT TECHNIQUES FOR STATIC FAULTS

DfT techniques have been used in digital ICs to achieve, fault detection, test circuit insertion, fault coverage analysis and test pattern generation, among other things related to test. Digital circuits are usually tested using the stuck-at fault model, which considers all faults in a digital IC as either tied up to logic '1' or down to logic '0'. All digital faults can be categorized into either stuck-at-0 or stuck-at-1 faults and can assume that every node can have either one of these two possible faults. For any given combinational circuit, a truth-table can be generated by simulation of all possible inputs. For a certain single-fault existing in the circuit-under-test (CUT), it is called a detectable fault if a different truth table is generated by the simulation of all possible inputs. For a test sequence, the ratio of detectable faults to all possible faults of a digital circuit is called fault coverage. The input values that can detect at least one fault are considered test patterns. Thus, test patterns are generated to detect faults in a digital device and the testability of the given device can be measured by fault

coverage. A path sensitization technique [7] is used to find proper test patterns for any given detectable fault. Finally, fault collapsing techniques [8] are used to remove many stuck-at faults and to reduce the total number of test patterns. Over the years, two major methods have been widely adopted by integrated circuit (IC) industry to address the digital testing issues: **Scan Path** and **BIST**.

2.2.1 SCAN PATH

Since the inception of IC design in the mid-1960s, IC test has been an integral part of the manufacturing process. Initially, tests were either randomly generated or created from verification suites. But as chips got larger, this process required a more targeted approach, one that needed to be easily replicated from one design to another. This led to the invention of scan, which made designs combinational and simplified the test generation process.

Scan path is a method to set and observe every flip-flop inside a digital IC chip by replacing all regular flip-flops (FF) with scan FFs and two additional input pins, test enable (TE) and test input (TI). All SFFs are in a chain which is connected through TI pin and SCANOUT pin, as shown in Figure 2.

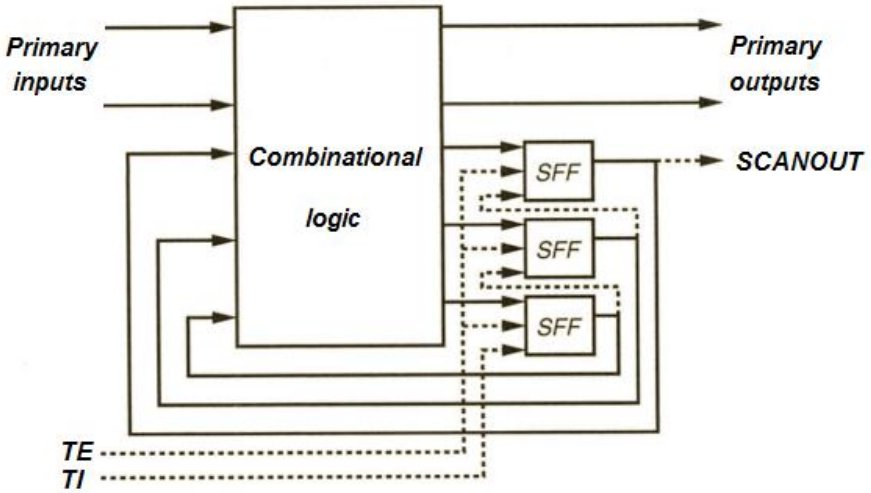


Figure 2: A Scan design schematic.

When TE pin is enabled which means shift mode, the scan chain can be accessed by standard JTAG I/O [9] pins to read and set all SFFs. After all SFFs are

settled into a desired state, TE pin is disabled (capture mode) and output of combinational logic can be captured in SFFs. Then TE pin is enabled again to shift out the *Q* pin of SFFs, bit by bit through the scan chain to SCANOUT, and at the same time, a new pattern is shifted in to set all SFFs to the next desired state (through TI). Scan chain makes it possible to assign an arbitrary internal state to a digital IC and thus may achieve higher test coverage with fewer test patterns.

In the modern System-on-Chip (SoC) design, many cores are integrated into a single chip. Some of them are embedded, and cannot be accessed directly from the outside of the chip. Such SoC designs make the test of these embedded cores become a great challenge.

2.2.2 BIST

BIST is one of most popular test solutions to test embedded cores [10]. As the digital circuit technology is moving to high densities of integration, BIST has become a primary issue in the realm of VLSI (Very Large Scale Integration) circuit design. Techniques for design for testability and BIST consider the testing problem during the design stage of digital devices and have been found to be extremely effective. The central idea behind BIST is to have the chip to test itself. This technique generates test patterns and evaluates output responses inside the chip [11] [12] [13]. Built-in Self-test is gaining popularity as a means to address test issues at the different packaging levels of digital systems. One of the benefits of BIST is the fact that no patterns need to be stored in the test equipment, which is simply required to provide a clock and a few control signals. This is especially important when high performance systems are being tested. BIST also makes the chip/board/system more independent of the specific test resources available at each manufacturing stage. BIST is also a convenient way of applying more test patterns, to compensate for the weaknesses of the stuck-at fault model [14]. BIST can significantly improve the testability of VLSI chips and save testing time as well [15]. BIST is a DFT technique that places the testing functions physically with the CUT, as illustrated in the Figure 3.

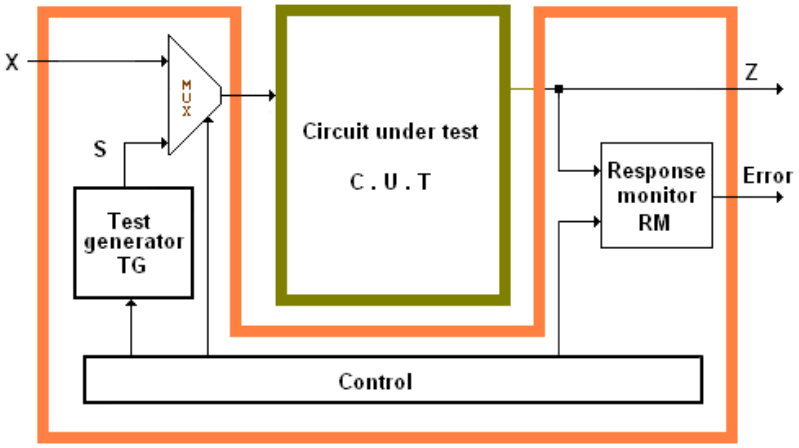


Figure 3: Basic BIST Architecture

In normal operating mode, the CUT receives its inputs X from other modules and performs the function for which it was designed. In test mode a test generator (TG) through a Linear Feedback Shift Register (LFSR) applies a sequence of test patterns to the CUT, and the response monitor (RM or Output Register Analyser (ORA)) using a multiple input signature register (MISR) for the effect compact test responses received from primary output. The response signatures are compared with reference signatures generated or stored on-chip, and the error signal indicates any discrepancies detected. The basic blocks that forms the BIST are: TG (LFSR), CUT, RM (SISR/MISR, Single/Multiple Input Signature Register), BIST controller and signature analyzer. BIST techniques make testing of a digital IC chip easier, faster, more efficient and less costly. At the cost of approximate by 20% – 30% overhead in the chip area and a small penalty in performance due to additional BIST hardware [16], the IC chip can now perform testing through internal scan chains without an external automatic testing equipment (ATE).

2.2.2.1 TEST PATTERN GENERATION

BIST is a DFT technique which allows the circuit to test itself without any external equipment [23]. BIST implementation requires primarily two components: a pseudo-random test pattern generator (for test vector generation) and a data compactor (for output response analysis) [24]. 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. These components are mostly implemented using LFSRs and Cellular Automata (CA).

LFSR is constructed using flip-flops connected as a shift register with feedback paths that are linearly related using XOR gates. An LFSR can be used for generation of pseudo-random patterns, polynomial division, and response compaction. The CA is very similar to the LFSRs except that the registers in CA have a logical relationship with their neighbours only. This leads more randomness in the pattern generated. LFSR is more popular for implementation of both TPG and ORA due to its compact and simple structure. However, CA is gaining popularity in many cases because of their characteristics and ease of modification.

Linear Feedback Shift Register or LFSR is a shift register whose output is the result of XOR of some of its inputs [22]. There are two ways to implement LFSRs: internal feedback and external feedback. These techniques differ in the way feedback is applied. All the flip-flops that feed a XOR gate are known as taps. These taps decide the pattern generated by the LFSR and hence define the characteristic polynomial of an LFSR, where *n* is the degree of the polynomial which is defined by the number of bits/nodes of the LFSR. Notice that the terms 'x_0' and 'x_{n-1}' are always present and the remaining terms indicate the location of the taps in the circuit. The degree of the polynomial *n* is equal to the number of bits in an n-bit LFSR pattern. An all zeroes state is invalid for an LFSR with XOR gates (the same for all '1' bits for an LFSR with XNOR gates), as the state would never change if all the bits are '*0*' or '1'. Therefore, the maximum number of unique patterns an n-bit LFSR can generate is $2^n - 1$, where *n* is the number of bits. Special LFSRs can be constructed to generate the all zeroes (ones) state also, but they have a larger area overhead associated with them, as described in [25]. In case of an external feedback LFSR, the XOR gates are in the feedback path and the input to the shift register is the XOR of all the taps.

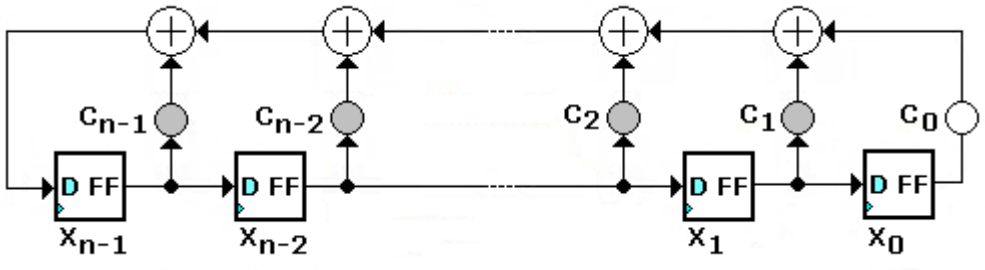


Figure 4: Linear LFSR External

But let's take a close look with a mathematical model support and start with the standard type (Linear LFSR or external). In the Figure 4 each tap of the coefficient C_i indicates the presence or absence of feedback from that particular flip-flop position into flip-flop position X_{n-1}. This is indicated by setting $Ci\,(0 \le i \le n-1)$ to '1' if the feedback exists, and to '0' if there is no feedback in that particular position. In the actual hardware, if C_i is '0', then there is no XOR gate in the feedback network for that bit position; otherwise, the XOR gate is included. Multiplication by x is equivalent to a right shift in the LFSR register by one bit, and the addition operation is the XOR ($\oplus$) operator. Therefore, addition is equivalent to XOR subtraction, so $0-0=0, 0-1=1, 1-0=1, 1-1=0$. This is because there are no carries or borrows in XORing arithmetic. The following matrix describes the system of equations:

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ \vdots \\ X_{n-3}(t+1) \\ X_{n-2}(t+1) \\ X_{n-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & -C_1 & -C_2 & \cdots & -C_{n-2} & -C_{n-1} \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ \vdots \\ X_{n-3}(t) \\ X_{n-2}(t) \\ X_{n-1}(t) \end{bmatrix}$$

Table 1: Linear LFSR System of Equations

This system is written as:

$$X(t+1) = Ts\ \mathrm{X(t)}$$

The first column of *Ts* is '0', except for the last row, to indicate that the flip-flops shift right. The 2nd through n^{th} columns and 1st through $n-1^{st}$ rows are the identity matrix,

to indicate that X_0 receives input from X_1, and so one. Finally, the n^{th} element in the first column is '1' to indicate that X_0 always feeds back into X_{n-1} through the XOR feedback network. The remaining elements in the n^{th} row are the feedback coefficients C_i, which indicate whether the remaining flip-flops feed back into X_{n-1} or not. We also see why this LFSR cannot be initialized to all zeros. If that were done, the feedback network and the right shifts of the flip-flops would always produce all zeros, and the LFSR would hang in the all-zero state. Note that the + operator implied in this matrix system is actually the XOR ($\oplus$) operator. If X is the LFSR initial state, the LFSR will progress through the states: $X, Ts\ X, Ts^2 X, Ts^3 X, ...$The matrix period is the smallest integer *k* such that:

$$Ts^k = I$$

Where *I* is the identity matrix, *k* is the LFSR cycle length (*k* = 0 for X = 0), and *Ts* is known as the companion matrix. Recall that multiplication by *x* is equivalent to shifting a bit through the D flip-flop register of this LFSR. Therefore, we view X_0 as the constant 1 and $X_1 = x.X_0 = x,\ X_2 = x.X_1 = x^2,\ \ldots,\ X_n = x.X_{n-1} = x^n$ .This hardware system can be described by the characteristic polynomial:

$$P(x) = \left|T_s - I.X\right| = 1 + c_1 x + c_2 x^2 + \ldots + c_{n-2} x^{n-2} + c_{n-1} x^{n-1} + x^n = \sum_{i=0}^{n} c_i x^i$$

The modular, internal exclusive-OR, or Type 2 LFSR is described by a companion matrix $T_M = T_S{}^T$, which is the transpose of T_S. It is called an internal XOR LFSR because the feedback XOR gates are located between adjacent flip-flops. The modular LFSR can run somewhat faster than the standard LFSR because it has at most one XOR gate delay between adjacent flip-flops. However, this is not a serious consideration in testing because actual circuits always have more logic gates between flip-flops than there are XOR gates in the feedback network of the external XOR LFSR. Moreover, for practical tests the test patterns generated by LFSRs are not more than 22-25 bits wide, so bigger circuits are partitioned into small sub-circuits of

less than 25 primary inputs [26]. The Figure 5 shows the modular LFSR circuit implementation.

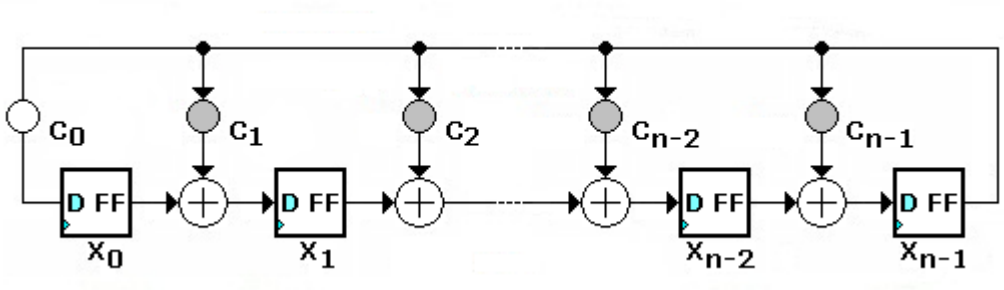


Figure 5: Modular LFSR Internal

The mathematical respective system of equations is presented in the next matrix.

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \\ \vdots \\ X_{n-3}(t+1) \\ X_{n-2}(t+1) \\ X_{n-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 & -C_1 \\ 0 & 1 & 0 & \cdots & 0 & 0 & -C_2 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & -C_{n-3} \\ 0 & 0 & 0 & \cdots & 1 & 0 & -C_{n-2} \\ 0 & 0 & 0 & \cdots & 0 & 1 & -C_{n-1} \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \\ \vdots \\ X_{n-3}(t) \\ X_{n-2}(t) \\ X_{n-1}(t) \end{bmatrix}$$

Table 2: Modular LFSR System of Equations

This system is written as:

$$X(t+1) = T_M \ \mathrm{X(t)}$$

This hardware system can be described by the characteristic polynomial:

$$P(x) = \left|T_M - I.X\right| = 1 + c_1 x + c_2 x^2 + \ldots + c_{n-2} x^{n-2} + c_{n-1} x^{n-1} + x^n = \sum_{i=0}^{n} c_i x^i$$

In the LFSR of the Figure 5, a right shift is equivalent to multiplying the register contents by x, and then dividing its value by the characteristic polynomial and storing the remainder.

Every LFSR can be realized either in standard or modular form. Both use *m* XOR (or XNOR) gates, where *m* is the number of non-zero C_i feedback coefficients in the LFSR.

2.2.2.2 OUTPUT RESPONSE ANALYSIS

During BIST, it is necessary to reduce the enormous number of circuit responses to a manageable size that can be stored on the chip. For example, consider a circuit with a hardware pattern generator that computes 5 million test patterns during testing, and where there are 300 PO_S. The total number of resulting responses will be:

$$5\,000\,000 \times 300 = 1500\,000\,000 \text{ bits!}$$

This huge amount of information cannot be economically stored, so the circuit responses must be compacted.

In this matter, we must distinguish between compression and compaction. Circuit response compression is lossless, because the original output sequence (1.5×10^9 bits in the previous example) can be completely regenerated from the compressed sequence. Compaction, however, results in information loss, so regenerating the original circuit response information is not possible. Compression schemes, at present, are impractical for BIST response analysis, because they inadequately reduce the huge volume of data, so only compaction schemes are used. In mathematical words, compression functions are invertible, but compaction functions are not.

Signature analysis is the process of compact the circuit responses into a very small bit length number, representing a statistical circuit property, for economical on-chip comparison of the behaviour of a possibly defective chip with a good one. Frohwerk [81] invented signature analysis in 1977 at Hewlett-Packard. Also, the signature must preserve as much as possible of the fault information contained in the circuit output response before compaction, and the circuitry used to implement the compacter should be small [31]. All compaction techniques require that the fault-free circuit signature be known.

Some schemes for response compaction are; (i) Parity checking, where parity is formed across all circuit responses; (ii) Ones counting, where the number of ones is counted in the output responses from the circuit. Savir [82] pioneered syndrome testing, in which pattern generation must be exhaustive, and ones counting is used for response compaction.

Aliasing occurs when the compacted response of the bad circuit matches the compacted response of the good circuit, and there is always a problem with compaction because information is lost. In parity checking, aliasing frequently happens. Also, with ones counting, it is possible to permute the placement of ones in the circuit's Karnaugh map, and still obtain a correct ones count, so it is also very prone to aliasing and also requires significant arithmetic hardware.

Hayes [83] described transition count testing. The transition count, C(R), is the number of times signals in the circuit response R change during BIST. Transition count test aliases less than ones counting, because it not only checks for the correct number of ones and zeros in the circuit output response, but also partially test for the correct ordering of the ones and zeros in the response.

2.2.2.2.1 LFSR FOR RESPONSE COMPACTION

Frohwerk [81] introduced the LFSR for response compaction by signature analysis. The signature is any statistical property of the circuit that is used for checking its correct operation. He used the data compaction method of the Cyclic Redundancy Check (CRC) code generator, which requires an LFSR hardware device. In this method, the circuit output data stream is treated as a descending order coefficient polynomial. The output response compacter LFSR performs polynomial division of this data stream polynomial by the characteristic polynomial of the LFSR. The Figure 6 shows a specific modular LFSR as a response compacter. The Table 3 presents the response of the circuit as the bits (01010001) are shifted into the LFSR through the XOR gate and the respective mathematical support for remainder generation.

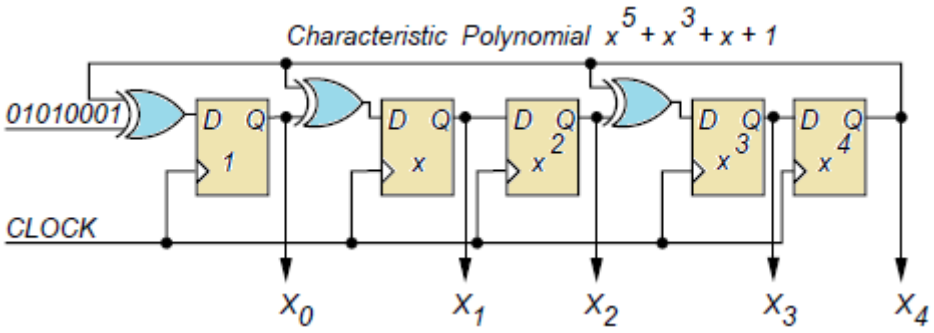


Figure 6: Modular LFSR as Response Compacter

Inputs	X_0	X_1	X_2	X_3	X_4
Initial State	0	0	0	0	0
[1°] 1	1	0	0	0	0
[2°] 0	0	1	0	0	0
[3°] 0	0	0	1	0	0
[4°] 0	0	0	0	1	0
[5°] 1	1	0	0	0	1
[6°] 0	1	0	0	1	0
[7°] 1	1	1	0	0	1
[8°] 0	**1**	**0**	**1**	**1**	**0**

Table 3: Five bits Modular LFSR Circuit Response

Data stream polynomial = **(0 1 0 1 0 0 0 1)** $\Leftrightarrow$

$\Leftrightarrow$ Data stream polynomial = $0.x^0 + 1.x^1 + 0.x^2 + 1.x^3 + 0.x^4 + 0.x^5 + 0.x^6 + 1.x^7$

$\Leftrightarrow$ Data stream polynomial = $x + x^3 + x^7$

Remainder = **(1 0 1 1 0)** $\Leftrightarrow$

$\Leftrightarrow$ Remainder = $1.x^0 + 0.x^1 + 1.x^2 + 1.x^3 + 0.x^4$

$\Leftrightarrow$ Remainder = $1 + x^2 + x^3$

x^7		$+x^3$		$+x$		x^5+x^3+x+1	
x^7	$+x^5$	$+x^3$	$+x^2$			x^2+1	
	x^5		$+x^2$	$+x$			
	x^5	$+x^3$		$+x$	$+1$		
		x^3	$+x^2$		$+1$	$\leftarrow$	Remainder

Table 4: LFSR Polynomial Division Result

The final state of the modular LFSR is the polynomial remainder of the division. The final state of the standard LFSR is not always the polynomial remainder of this division, but is related to the true remainder through a different state assignment. The error diction hypothesis is that a faulty data stream changes the output data stream, and hence the remainder of this polynomial division, which is used as signature in the compaction method. The LFSR must be initialized to the seed value, and after data compaction, the signature must be observed and compared with the known good circuit signature [31]. The signature analyzer circuit is also easily testable.

The Figure 6 shows a modular LFSR that has an extra XOR gate at the input to the flip-flop driving the least significant bit X_0. This XOR gate XORs the circuit output response stream, (01010001) in this case, into the least significant bit of the modular LFSR. Here, (01010001) is interpreted as:

$$0.x^0+1.x^1+0.x^2+1.x^3+0.x^4+0.x^5+0.x^6+1.x^7 = x+x^3+x^7$$

Reading the LFSR tap coefficients from left to right in Figure 6, we see that the characteristic polynomial of this modular LFSR is $1+x+x^3+x^5$. The Table 3 shows how eight clock periods are simulated after the LFSR is initialized do (00000). It also shows in Table 4 the long division of the reversed data stream polynomial by the reversed characteristic polynomial of the LFSR. The remainder of the division, $1+x^2+x^3$, also matches the remainder left after eight clock periods in the LFSR, because only X_0, X_2 and X_3 are ones. Thus we have agreement between the

signature predicted by polynomial division and the signature produced by logic simulation.

2.2.2.2.2 MULTIPLE INPUT SIGNATURE REGISTER

In the example of the Figure 6, [84] one primary circuit output requires an LFSR for signature analysis with 5 flip-flops and 3 XOR gates. However, consider the case where the circuit of Figure 6 has 300 outputs. Then, we would need 300 x 5 =1500 flip-flops and more than 300 x 3 = 900 XOR gates. This is a serious hardware overhead. Fortunately, we can exploit the fact that the hardware pattern generation and response compaction system using LFSRs is a linear system, obeying the equation $X(t+1) = T_S\ \mathrm{X(t)}$. Therefore, because of its linearity, this system also obeys the superposition principle. If we superimpose all the responses of the 300 circuit outputs in the same LFSR for response compaction, then the final remainder will be the sum (under XOR logic arithmetic) of the remainders due to all of the circuit outputs. This is highly advantageous, as it reduces the flip-flop count from 1500 to 300 and the XOR gate count from more than 900 to approximately 3+300. The 300 added XOR gates are needed to XOR all of the circuits outputs into different bits of the LFSR, where there must be one bit for each circuit PO, called d_i. This new response compacter is known as Multiple Input Signature Register (MISR), and an example is shown in the Figure 7 with a linear type 1, and Figure 8 with a modular type 2.

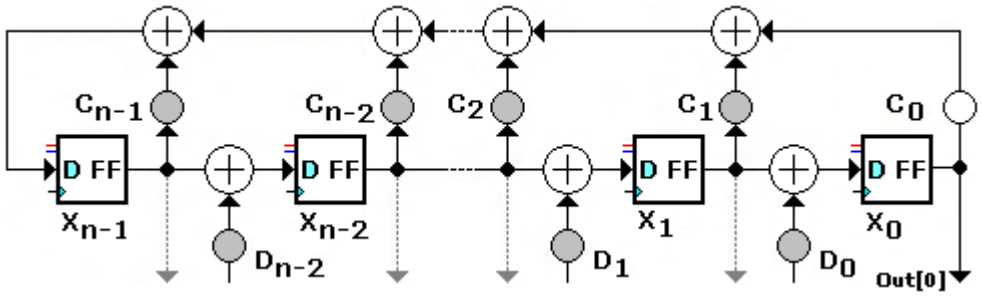


Figure 7: Linear Multiple Input Signature Register

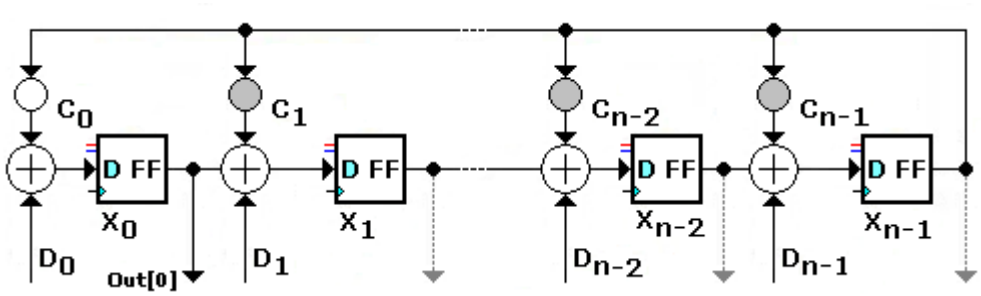


Figure 8: Modular Multiple Input Signature Register

The alternative to use the MISR structure is to provide only one simple LFSR for one circuit output, but multiplex it among the 300 different outputs. This then requires 300 different testing epochs, where for each epoch the LFSR compacts the response from a different circuit output. It is much more attractive to use the MISR, because it eliminates a 300 to 1 MUX, and also because the response compaction time with the MISR is 300 times less than the time with a multiplexed LFSR. The generic linear MISR can be represented by the following system of equations (Table 5):

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ \vdots \\ X_{n-3}(t+1) \\ X_{n-2}(t+1) \\ X_{n-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & -C_1 & -C_2 & \cdots & -C_{n-2} & -C_{n-1} \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ \vdots \\ X_{n-3}(t) \\ X_{n-2}(t) \\ X_{n-1}(t) \end{bmatrix} + \begin{bmatrix} d_0(t) \\ d_1(t) \\ \vdots \\ d_{n-3}(t) \\ d_{n-2}(t) \\ d_{n-1}(t) \end{bmatrix}$$

Table 5: Linear MISR System of Equations

The vector of $d_i(t)$ values represents the circuit outputs at time PO_i.

The modular MISR can be translated by the following system of equations (Table 6):

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \\ \vdots \\ X_{n-3}(t+1) \\ X_{n-2}(t+1) \\ X_{n-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 & -C_1 \\ 0 & 1 & 0 & \cdots & 0 & 0 & -C_2 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & -C_{n-3} \\ 0 & 0 & 0 & \cdots & 1 & 0 & -C_{n-2} \\ 0 & 0 & 0 & \cdots & 0 & 1 & -C_{n-1} \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \\ \vdots \\ X_{n-3}(t) \\ X_{n-2}(t) \\ X_{n-1}(t) \end{bmatrix} + \begin{bmatrix} d_0(t) \\ d_1(t) \\ d_2(t) \\ \vdots \\ d_{n-3}(t) \\ d_{n-2}(t) \\ d_{n-1}(t) \end{bmatrix}$$

Table 6: Modular MISR System of Equations

The next example in the Figure 9 shows a modular LFSR converted into a MISR, by XORing a different circuit output into each flip-flop position.

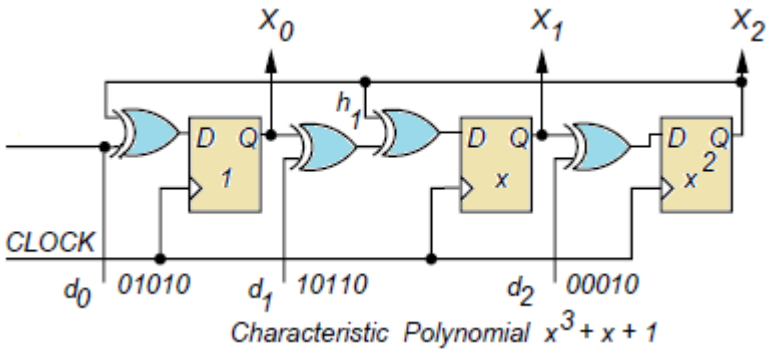


Figure 9: Modular Multiple Input Signature Register with 3 bit Input Pattern

The resulting signature, since this system is linear, is the XORing of the three different signatures due to the polynomial division from each of the three PO_s. It implements the following equation system presented in Table 7:

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ X_2(t+1) \end{bmatrix} = \left| \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ X_2(t) \end{bmatrix} + \begin{bmatrix} d_0(t) \\ d_1(t) \\ d_2(t) \end{bmatrix} \right|$$

Table 7: Modular MISR System of Equations with 3 Input bits

2.3 DELAY FAULT TESTING USING TRANSITION FAULT MODEL

There are three main methods that can be used to generate and apply transition fault tests. The first method, termed **Broad-side delay test**, is also referred to as functional justification or the launch-from-capture technique, or even launch-on-capture (LOC). In this technique, the first vector of the pair is scanned into the chain and the second vector is derived as the combinational circuit's response to the first vector [32].

The second method, termed **Skewed load transition testing**, is also referred to as the launch-from-shift technique, or even launch-on-shift (LOS). In this method, both the first and second vectors of the pair are delivered through the scan cells themselves [32]. If the scan-chain is *n* bits long, an *n*-bit vector is loaded by scanning in the first (*n*-1) bits. The last shift clock is used to launch the transition, followed by a quick capture.

In the third method, termed **Enhanced-scan transition testing**, the two vectors (V1, V2) are stored in the tester memory. Vector V1 is first applied and this initializes the circuit. Vector V2 is then scanned in, followed by applying it to the circuit under test and capturing its response. The important point is that it is assumed the initialization provided by V1 is not lost while loading V2. Therefore, this type of test assumes a hold-scan design [33]. For inclusion of hold-scan cells, an area overhead is evident and there is an additional routing requirement for the control signal. As a result, such hold-scan cells are not used in the ASIC industry and thus, enhanced scan-design is not always useful in a practical environment.

2.3.1 LAUNCH ON CAPTURE

This technique is also known as the broad-side or functional justification technique. As we know, transition fault tests require a pair of vectors, one, to set a target node to an initial value, and the next, to launch the transition and propagate the effect to a primary output or scan cell [6] [19]. In this technique, the first vector of the

pair is scanned into the chain and the second vector is derived as the combinational circuit's response to the first vector [15].

In a scan-based design, if the scan chain contains *n* cells, a vector pair is obtained by applying the following steps:

- Shift the data into the scan-chain *n* times.
- Toggle the scan-enable signal and allow the circuit to settle (new PI values may be applied if required).
- Pulse the clock twice. The first pulse will launch the transition and the second pulse will capture the response from the combinational portion of the circuit.
- If required, primary input (PI) or primary output (PO) changes could be made with the application of the first clock pulse.
- If the tester hardware does not support at-speed PI changes, the PI values across launch and capture cycles will have to be held constant. If at-speed output strobing is not supported, the effects of all faults have to be observed only at flip-flops on the scan chain.

The timing diagram for this method is shown in next picture.

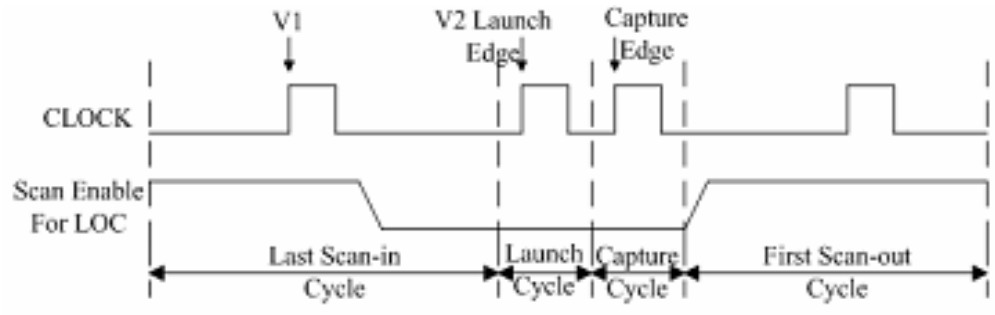


Figure 10: Launch on Capture

The important point to note here is that the launch and capture are performed with the scan-enable signal set to functional mode. The scan-shift frequency is much slower than the functional operation frequency in most industrial designs. The scan-shift speed may also be limited by the maximum frequency supported by the tester hardware being used. As a result, two different waveforms (or timesets), one to enable

scan-shift and the other to perform the at-speed capture, may need to be applied to the same clock pin, while the device is being tested.

2.3.2 LAUNCH ON SHIFT

This technique is also known as the Skewed-Load or Transition Shifting technique. Here, both the first and second vectors of the pair are delivered through the scan cells themselves [15]. In a scan chain containing *n* cells, this approach consists of the following steps:

- Shift the scan-chain (*n*-1) times to obtain the first vector in the pair.
- Simultaneously, apply the first of the two sets of PI values to the non-scan pins.
- Most designs consist of a muxed data scan cell, where a mux is used to choose between the value from the combinational logic and the value from the scan-chain. The scan-enable signal is used to control this mux. In such designs, setting the scan enable signal to scan mode and shifting the scan chain once more generates the second of the two vectors.
- Toggle the scan-enable pin
- Change the PI values as required.
- Pulse the clock to capture the response data into the scan flip-flops.
- If the tester hardware supports at speed output strobing, the PO pins are strobed during this cycle to detect transition faults propagating to the POs.

The timing diagram for this method is shown in the Figure 11.

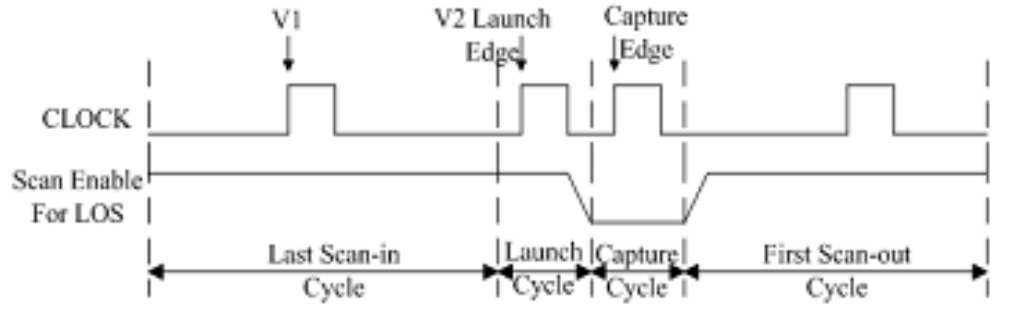


Figure 11: Launch on Shift

The most important difference between the two techniques described above with respect to muxed data scan designs is the need for at-speed scan-enable operation in the launch-on-shift technique. Further, the launch-on-capture technique requires a sequential ATPG algorithm, while launch from-shift patterns can be generated with a purely combinational ATPG algorithm.

3. AGING EFFECTS IN CMOS NANO TECHNOLOGIES

With relentless scaling of CMOS technology, circuit timing uncertainty due to temporal degradation and static process variations poses a dramatic challenge to IC design [74][85]. The deterioration of circuit performance over time, i.e., aging, is usually caused by several physical mechanisms such as channel-hot-carrier (CHC), negative-bias-temperature-instability (NBTI), and time-dependent-dielectric-breakdown (TDDB) [86][87][88][44][89]. Among these effects, NBTI is the leading mechanism that is responsible for the majority part of circuit aging [90][88] in [88], the authors show that for 65nm technology, CHC degradation is much smaller than NBTI degradation, almost one order lower in the degradation magnitude). NBTI primarily increases the threshold voltage (Vth) of PMOS devices and it significantly affects circuit lifetime and performance (e.g., power, speed and failure rate). In the worst case condition, it may even result in a complete parametric failure of a system [92][88][44][93][17][42].

To cope with this threat and guarantee circuit lifetime, it is critical to include NBTI into circuit analysis and adaptively develop design techniques to effectively mitigate its negative impact on performance. For a VLSI design, an accurate prediction of circuit performance degradation under NBTI remains as a tremendous challenge. As shown in [88], NBTI has a strong dependence on dynamic operation conditions, such as supply voltage (V_{DD}), temperature (T) and input signal probability (α_s). Usually these parameters are not spatially or temporally uniform, but vary significantly from gate to gate and from time to time. Similar to the burning process, we may use high voltage and high temperature to guardband the worst case condition. However, the search for the worst case α_s is computationally inhibitive due to the extremely large space of signal probabilities for each input node.

The expected lifetime of a circuit is, then, limited by these long-therm and cumulative degradations, that we call aging. Although NBTI is the dominant phenomena, as mentioned, it is the effect of simultaneous causes that could easily make a circuit to fail. And, considering other effects that could also cause a delay-

fault, in literature one can identify static and dynamic effects like Process, power-supply Voltage and Temperature variations (PVT), just to mention the most important ones. These parametric variations, operation dependent or not, along with cumulative degradations (PVT and Aging, PVTA), can seriously impose a high variation in a critical path delay, causing the circuit to fail.

In this section we are focusing on aging effects, and a description of the most important phenomena will be presented in the following.

3.1 NEGATIVE BIAS TEMPERATURE INSTABILITY

Negative bias temperature instability has been known since 1966 [37]. However, only in the last few years it has become a reliability issue in silicon integrated circuits, because the gate electric fields have increased as a result of scaling, increased chip operating temperature, surface p-channel MOSFETs have replaced buried channel devices, and nitrogen is routinely added to thermally grown SiO2. In 2003 for example, it was poorly understood that the time between NBTI stress and measuring the effect after terminating the stress was important, because the NBTI recovery was just beginning to be understood. Now it is understood that the sooner a degraded device is measured after stress, i.e., within mili-seconds (*ms)* or sooner, the more relevant are the data.

In the recent years, NBTI has been identified as a major and critical reliability issue for PMOS devices in nano-scale designs, and with the continuous decrease of the transistor dimensions, it will continue to be one of the biggest effects (if not the higher effect). It manifests as a negative threshold voltage shift, thereby degrading the performance of the PMOS devices over the lifetime of a circuit, and the degradation worsens at high temperatures, causing a larger shift in the threshold voltage. As a result, considering degradations in a long period of time, the threshold voltage shift can potentially cause a significant increase in delay of the p-MOSFET devices ([17], [20]) and, ultimately, a delay-fault may occur.

A vast number of studies have already been conducted to investigate the effect of NBTI on digital circuits [43] [38] [42] [40] [41]. Moreover, many studies have also developed several design-time and run-time techniques to cope with the NBTI

degradation, like [39][45][40][44]. These studies include the use of CAD tools for managing transistor degradation mechanism [39], the use of dynamic voltage scaling (DVS) [45], the use of data flipping to recover the static noise margin of the static random access memory (SRAM) [45], and the use of device parameter tuning (V_{DD}, Vth and gate-size) in digital circuits [44].

For more information on NBTI, on the degradation process caused by the generation of traps, and the partial recovery associated with the reduction in traps, please refer to [17][18][19].

3.2 TIME DEPENDENT DIELECTRIC BREAKDOWN

Time Dependent Dielectric Breakdown (TDDB) is a phenomenon where the oxide underneath the gate degrades. As the name implies, it is the breakdown of a dielectric over time. There are other ways a dielectric can breakdown but in a digital system, the only variables are: operating frequency, voltage supply, MOSFET characteristics (such as gate area or dielectric material), temperature, and time. As the gate-oxide is scaled down, breakdown of the oxide and oxide reliability becomes more of a concern. Higher fields in the oxide increase the tunneling of carriers from the channel into the oxide and these carriers slowly degrade the quality of the oxide and, over time, leads to failure of the oxide [46].

Once a dielectric breaks down, current is able to flow more easily through the gate into the drain/source of a P/NMOSFET, completely destroying functionality. Evidence of TDDB are changes in the threshold voltages and the drain currents, as well as a great increase in current through the dielectric [47] and, ultimately, the gate breakdown, as shown in the Figure 12.

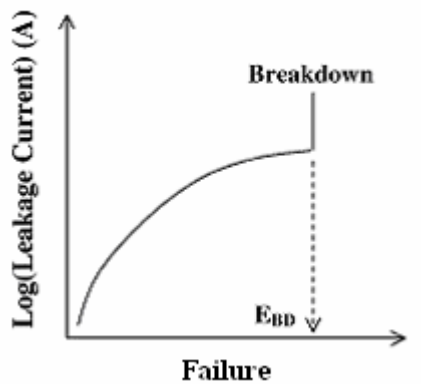


Figure 12: Relationship between TDDB and Leakage Current [49].

There are many hypotheses for why TDDB occurs. Many models describe what occurs in the dielectric material over time, and each model, consequently, has a mathematical model that can predict the expected failure of a device. There has been much speculation for the last 50 years as to which model correctly predicts the failure time. However, there is general consensus that the electric field through the dielectric material is the direct cause of TDDB. This relationship is shown in Figure 13.

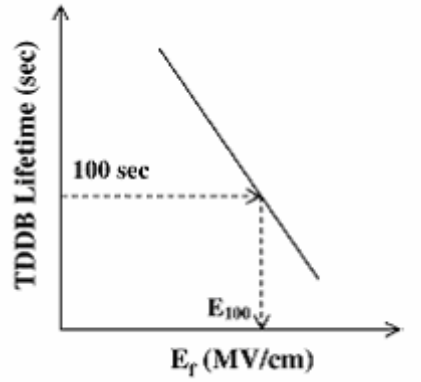


Figure 13: Relationship between TDDB and the Electric Field [49].

The simple explanation is that the electric field breaks down the oxide, but electric fields could be the cause of more specific phenomena, such as band-to-band impact ionization, hole trapping near the injecting interface, and electron trapping [47]. Nevertheless, it is accepted that it is caused by charge that remains in the oxide [48]. Ideally, the charge should not pass through the oxide, but thinner oxides and stronger electric fields make it possible.

3.3 HOT CARRIER INJECTION

Hot carrier damage has been one of the important degradation mechanisms in MOSFETs [50]. The major source of the hot carriers is the electric field inside the channel of a transistor. The energetic carriers themselves, or the carriers generated through impact ionization, can cause the parametric degradation, i.e., shifts in device characteristics or catastrophic failure such as oxide breakdown. Significant effort has been focused on understanding the hot carrier phenomena and its implications for circuits.

One of the early pioneering works was done in 1980s, and involved the calculation of HCI lifetime, based on experimental device characteristics during hot carrier stressing [51]. In that work, it was assumed that the carriers heated by the channel electric field can lead to impact ionization. For an NMOSFET, the holes generated by ionization flow out of the substrate contact, giving rise to substrate current (I_{sub}), whereas the electrons contribute to the drain current (and if they are injected into the oxide, constitute the gate current, I_G).

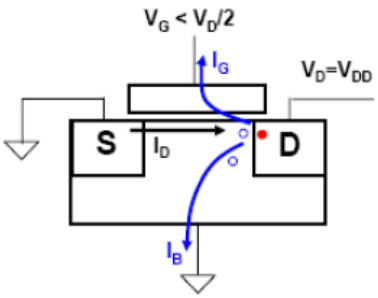


Figure 14: Substract and Gate Currents in a NMOSFET at Low VG

In the Figure 14 it is possible to see the holes (open circles) generated by impact ionization, flow out of the substrate. Some fraction can be injected into the gate oxide, since the vertical electric field favours holes at low VG. The Figure 15 shows that at high VG, the vertical electric field attracts electrons (filled circles) into the gate oxide and the electrons form the gate current. The substrate current is still due to holes.

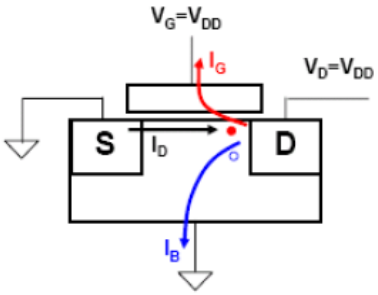


Figure 15: Substract and Gate Currents in a NMOSFET at High VG

The hot carrier damage is also attributed to the energetic electrons. The I_{sub} was conventionally taken as a monitor for the hot carrier damage, because it reflects the energy of the hot electrons.

3.4 ELECTROMIGRATION

When a sufficiently strong electric current is passed through a metal interconnect, a diffusive motion of impurities and/or vacancies takes place in a direction along or opposite to the current flow. This phenomenon is called electromigration (EM). The technological interest in EM arises from its manifestation as a cause of failure in integrated circuits.

The phenomenon of electromigration has been known for over 100 years. The earliest observation can be traced back to 1861, when Gerardin observed EM in lead [52]. Following, was the work of Sakupy in 1907 [53], who studied mass transport of impurities in molten metals. Sakupy was also the first to use the term ‘‘electron wind’’.

More recently, the technological interest for EM started in 1966, when IBM, Fairchild, Motorola, and Texas Instruments independently observed failures in integrated circuits, which could not be explained. At the time, the metal interconnects in ICs were still about 10 micrometers wide and EM surprised, and briefly threatened the existence of the integrated circuit industry [54]. Currently, interconnects are only hundreds to tens of nanometers in width, making research in electromigration increasingly important.

In general, EM decreases the reliability of chips. It can cause the eventual loss of connections or failure of a circuit. Since reliability is critically important for space travel, military purposes, anti-lock braking system, medical equipment (like Automated External Defibrillators) and is also important for personal computers or home entertainment systems, the reliability of chips (ICs) is a major focus of research efforts. Due to difficulty of testing under real conditions, Black's equation [55] is used to predict the life span of integrated circuits. To use Black's equation, the component is put through High Temperature Operating Life (HTOL) testing. The component's expected life span under real conditions is extrapolated from data gathered during the testing [55].

Athough EM damage ultimately results in failure of the affected IC, the first symptoms are intermittent glitches, and are quite challenging to diagnose. As some interconnects fail before others, the circuit exhibits seemingly random errors, which may be indistinguishable from other failure mechanisms.

With increasing miniaturization the probability of failure due to electromigration increases in circuits, because both power density and current density increase. In advanced semiconductor manufacturing processes, copper has replaced aluminium as the interconnect material of choice. Despite its greater fragility in the fabrication process, copper is preferred for its superior conductivity. It is also intrinsically less susceptible to electromigration. However, EM continues to be an ever present challenge to device fabrication and, therefore, the EM research for copper interconnects is ongoing (though a relatively new field).

In modern consumer electronic devices, ICs rarely fail due to electromigration effects. This is because proper semiconductor design practices incorporate the effects of electromigration into the IC's layout. Nearly all IC design houses use Electronic Design Automation (EDA) tools to check and correct electromigration problems at the transistor layout-level. When operated within the manufacturer's specified temperature and voltage range, a properly designed IC device is more likely to fail from other (environmental) causes, such as cumulative damage from gamma-ray bombardment.

Nevertheless, there are documented cases of product failures due to electromigration. In the late 1980s, one line of Western Digital (WD) desktop drives suffered widespread, predictable failure 12–18 months after field usage. Using forensic analysis of the returned bad units, engineers identified improper design-rules

in a third-party supplier's IC controller. By replacing the bad component with another one from a different supplier, WD was able to correct the flaw, but not before significant damage to the company's reputation.

EM can also be a cause of degradation in some power semiconductor devices, such as low voltage power MOSFETs, in which the lateral current through the source contact metalization (often aluminium) can reach the critical current densities during overload conditions. The degradation of the aluminium layer causes an increase in on-state resistance, and can eventually lead to complete failure.

3.5 STRESS INDUCED VOIDS

The phenomenon of stress induced voiding is generally understood as a result of stress mismatch in materials [56] and structures [57] in copper interconnect. As mentioned in the previous section, in the last years copper has replaced aluminium as the interconnect metal of choice in microchip fabrication. The main advantage of copper is its low electrical resistivity and high resistance to electro-migration and stress-migration, (in comparison with aluminium). Lower resistance means that smaller and more tightly packed metal lines can carry the same amount of current. This leads to fewer levels of metal, faster speed, and lower production costs. The main drawback to copper is its high diffusivity. To prevent copper from diffusing into transistors, it must be encapsulated in a barrier film, usually a derivative of tantalum or titanium. In addition, to reduce the extra parasitic capacitance in denser circuits, dielectrics with lower dielectric constants must be used. The spin-on-coat process of low-k dielectric material requires furnace annealing to cure the film. During this thermal processing, however, the copper is mechanically confined in the bulk layer by the barrier metal and in the vias/trenches by sidewalls. As the copper and dielectric materials are heated and cooled, their different thermal coefficients of expansion lead to a mismatch in the residual stress of the copper in the bulk layer and trenches. The mismatch leads to stress migration and to stress induced voiding (SIV) in the copper during chip operation. Voids increase the resistance and lead to chip failure. Obviously, this causes a severe problem in chip reliability.

Microstructural analysis of copper thin film is increasingly important for understanding stress-induced voiding kinetics. Microstructure dependence of stress induced voiding in copper thin films mainly comes from its effects on vacancy diffusion and void nucleation [58][59][60][61]. The grain boundaries themselves are full of vacancies, and the free volume released by grain growth as the result of grain boundary elimination creates sizeable voids. Also, the grain boundary is one of the fast diffusion paths in copper interconnect, and the diffusivities are influenced by the misorientation angle of grain boundaries [59][60]. Moreover, twin boundaries have been found to be nucleation sites for stress induced voiding due to thermal stress concentration at their interfaces [61]. So, copper films with larger grains (fewer grain boundaries) that also maintain strong crystallographic orientation and minimum twin formation are preferred for stress induced voiding resistance in copper interconnects.

Many methods have been suggested to suppress stress voiding in copper interconnects. Most of these, involve either altering the geometry of the line/via structure, changing the dielectric materials to improve passivation, or optimizing the thermal cycling process in an attempt to make it more robust [58]. In addition, it has been theorized that the inclusion of a small amount of a second metal in copper thin films during electroplating, and its subsequent segregation at grain boundaries by thermal treatment suppresses the copper grain boundary diffusivity. Also, in addition to possibly creating interstitial defects in the copper crystallite lattice, the alloyed co-element may fill the vacancies inherent at grain boundaries. The co-element thereby affects both the grain size distribution and thermalmechanical properties (i.e. flow stress) of the copper thin films by particle pinning of grain boundaries.

3.6 TOTAL IONIZING DOSE

The need to follow, as much as possible, Moore's law, pushes the commercial manufacturer to increase the device density of modern Integrated Circuits (ICs) down to the feasibility limit [62][63]. At the time, Intel's 22nm microprocessors are available on commercial market, though, looking at the next step, down to 14 nm. This scaling trend impacts the ionizing radiation response, as well as introducing new challenges, while removing some historical issues. The main degradation mechanism

that occurs in a MOS device subjected to ionizing radiation is the oxide charge trapping [64][65] and [66]. A schematic band diagram for a NMOS device is reported in Figure 16.

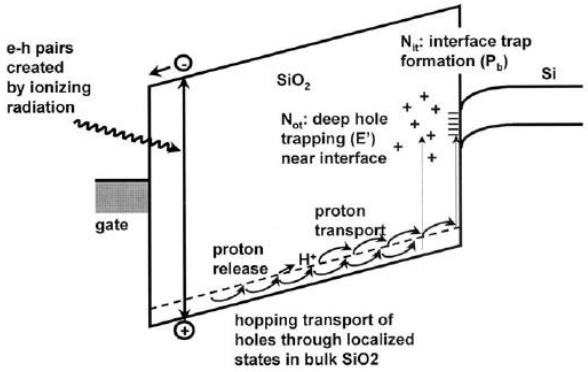


Figure 16: Schematic Representation of the Damage Induced by Radiation in a MOS Structure [64].

Immediately after electron-hole pair generation, induced by radiation, the electrons and holes that survive the initial recombination are split by the electric field and drift toward the Si/SiO2 interface (holes) and gate (electrons). As the holes arrive at the interface, some fractions are trapped in pre-existing localized defects, leading to a net positive charge N_{ot}. The positively charged hydrogen can be released as well from the gate/oxide interface and drift to the Si/SiO2 interface, where it can react, forming interface traps, N_{it} [67]. Both interface traps, which can be negatively or positively charged and trapped charges, influence the electrostatics of CMOS transistors, affecting the main parameters, such as threshold voltage, drain current, transconductance, and carrier mobility [66]. The thinning of the gate oxide below 5 nm has significantly mitigated the Total Ionizing Dose (TID) effects, reducing the charge trapping phenomena that plagued the older technologies built with thicker oxides, when employed in radiation environments [68].

In contrast, the very thick lateral oxide has become the Achilles heel of modern CMOS transistors exposed to ionizing radiation. In fact, the large amount of charge that can be trapped at the edges of the device influence the electrostatics of the transistor, leading to large shifts of the characteristics parameters [69]. As a consequence, the lateral isolation engineering will be one of the key points to have commercial electronics with a good resilience to total ionizing dose effects [70].

However, despite the increased STI sensitivity, the total dose hardness of commercial CMOS devices increased during the last ten years, featuring for the 130 nm and 90 nm technology nodes a TID tolerance of about 200 krad(SiO2) [71], doses of interest for space applications.

4. BIST FOR DELAY-FAULTS

The purpose of this chapter is to present and generate the BIST architecture and structures necessary to implement a self-test that aims the detection of delay-faults. The goal is to reuse BIST functionality and base structures, and combine it with standard DfT techniques for delay-faults, namely: LOC and LOS. The idea of combining these two DfT techniques, BIST and LOC/LOS, was previously published in [75] [76]. However, only the concept and a limited set o test circuits were implemented. In fact, the controller functionality was defined for scan based BIST for sequential circuits, but it was never simulated with all BIST infrastructure and CUT.

In the present work, the BIST strategy used to detect delay-faults is the same as described in [75] [76]. However, the new contributions in this matter are:

1. Redesign the BIST controller, to allow full auto-test with simulation of the complete BIST infrastructure and CUT;
2. Implement and simulate the behavioural description of the BIST infrastructure's RTL level in VHDL;
3. Implement the structural description of the BIST infrastructure's gate level in Verilog;
4. Implement and simulate the SPICE netlist for BIST infrastructure and CUT, from the gate level description, using the generic CMOS library described in [77];
5. Study the aging degradation (expected) of BIST infrastructure and CUT;
6. Implement a software tool to automatically insert BIST structures in a CUT, both in behavioural RTL level VHDL format, and in structural gate level Verilog format.

In the first section the BIST structures and BIST functionality are presented, and the second section is dedicated to the BISTGen software tool, developed to automatically generate the BIST structures of first section.

It is also important to mention that all the BIST structures and circuitry was developed and described in VHDL format, using a behavioural RTL level description of the blocks. After the VHDL description of each block and circuitry was validated through logic simulation in ModelSim environment, it was synthesized with Synopsys™ software environment, at *INESC-ID* in Lisbon, to generate the Verilog structured gate level description. Therefore, each circuitry has two identical behaviour implementations, although different in the format. Moreover, the library used to synthesize the structure of each gate level netlist was the AMS (Austria Micro Systems) 350nm CMOS technology library, that was also previously been translated to a generic SPICE netlist in a previous M.Sc. thesis at *UAlg* (please refer to [77]).

4.1 SCAN BASED BIST FOR DELAY-FAULTS

The main idea of the scan based BIST for delay-faults is to implement a traditional scan based BIST approach that implements the delay-fault techniques used traditionally with scan: LOS and LOC. In fact, it implements 3 possible test methods, the mentioned LOS, LOC and a combined test with LOS and LOC used together in the same test set.

The test methodology is defined by the architecture shown in Figure 17. It's an enhanced approach from the traditional scan based BIST architecture. Taking a closer look, the block diagram is a bit more complex than the traditional one, special in the number and type of modules used, and in their interconnections. As shown, the architecture have in its composition a MUX, three LFSR chains, two triangular blocks representing each one a comparator circuit of *n* and *m* inputs for one output, a BIST Controller, which is the main core of the entire circuit, the CUT (pre-reconfigured with scan flip-flops), a MISR which is a modified LFSR to operate as an *n* input data register and a six input '*and*' gate to one output.

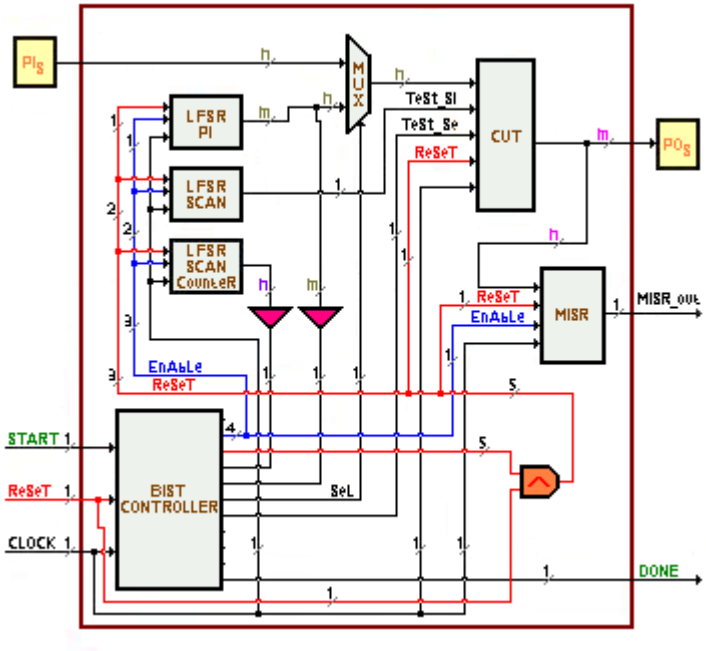


Figure 17 : Parent BIST Block Structure

All the referred blocks will be detailed and explained in the next sections.

4.1.1 MUX BLOCK

In order to switch between the primary inputs in normal circuit operation and the test inputs, which in test mode are the outputs of the pseudo-random Linear Feedback Shift Register (LFSR), a switch was designed for the effect. The switch have in its structure an array of *n* 2x1 Multiplexers, where *n* is the number of inputs to select from.

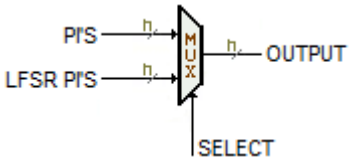


Figure 18: Switch Multi MUX

The Table 8 shows an example in Verilog format where the global input is composed of two entries.

```
wire SeLMuXCuT;
wire [1:0] InAMuXCuT;
wire [1:0] InBMuXCuT;
wire [1:0] DataOutMuXCuT;

MUX21 U1MuXCuT ( .A(InAMuXCuT[0]), .B(InBMuXCuT[0]), .S(SeLMuXCuT), .Q(DataOutMuXCuT[0]) );
MUX21 U2MuXCuT ( .A(InAMuXCuT[1]), .B(InBMuXCuT[1]), .S(SeLMuXCuT), .Q(DataOutMuXCuT[1]) );
```

Table 8: Mux code slice in Verilog

One of the best correlations between two languages to describe circuits is when both describe the same behaviour. From now one, it will be presented for the generality of the examples also its equivalent in VHDL. Table 9 presents the same two inputs for the Mux entries but now in VHDL description code.

```
entity MuXCuT is
 port( SeL : in std_logic;
 InA, InB : in std_logic_vector(1 downto 0);
 DataOut : out std_logic_vector(1 downto 0));
end MuXCuT;

architecture comportamento of MuXCuT is

begin

 process(Sel, InA, InB)
 begin
  if Sel='0' then
   DataOut <= InA;
  else
   DataOut <= InB;
  end if;
 end process;

end comportamento;
```

Table 9: Mux code slice in VHDL

4.1.2 LFSR PI BLOCK

This LFSR PI stands for Linear Feedback Shift Register for Primary Inputs, and it's basically the LFSR block that will generate the inputs in test mode for CUT's primary inputs. When the controller receives the information for switching the circuit from normal operation mode to test mode, it places the reset line that connects this block to logic value '0', setting the initial seed in the LFSR. This initial seed will

define the flip-flop composition of the LFSR, as when a specific bit should be '0', the flip-flop should have a RESET input connected to the reset signal of the LFSR, whereas when a bit should be '1' in the initial seed, the flip-flop should have its SET input connected to the reset signal.

The LFSR will also have an enable signal to pause the operation of the LFSR, if necessary, and a clock signal. The output consists of a bus where the number of lines can be equal or bigger than the number of primary inputs in the CUT. The question that may arise is: why not the same number of bits than CUT's primary inputs? The answer lies in the randomness and test length that we want to achieve with the flip-flop chain that constitute the LFSR. It is generally known that a bigger LFSR will have a more arbitrary sequence than a smaller one, even if both are used with an equal test length. Moreover, the internal feedback connection in the LFSR can also define two possibilities: a linear feedback and a modular feedback structure. The linear type is usually a smaller structure; however, the modular type usually leads to better test results, due to a higher randomness in test vectors.

The maximum number of different test vectors generated by an LFSR is established by the formula $2^n - 1$. As the initial seed is always the same, defined by LFSR structure, if the test length is constant, we guarantee a test with always the same test vectors applied to the CUT. Thus, this LFSR's outputs will also be used to define the test length, by identifying a final LFSR output and indicating the controller to stop the BIST section.

The Figure 19 presents the LFSR PI block diagram of the test pattern generator.

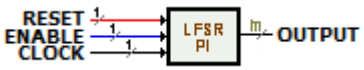


Figure 19: LFSR PI

The block when requested is capable to generate two different types of LFSRs. The Table 10 shows two examples, the first, a linear one and the second a modular type. The default value to start, known as the seed is '10110' in binary format representing an *m* of five outputs.

Also an analogue example but this time in VHDL is provided in the Table 11.

LINEAR TYPE

```
wire resetLfsrPICuT;
wire enableLfsrPICuT;
wire [4:0] DataOutLfsrPICuT ;

wire [4:0] QoutLfsrPICuT ;
wire y1LfsrPICuT;

DFEC1 U0LfsrPICuT ( .D(QoutLfsrPICuT[1]), .E(enableLfsrPICuT), .C(clock), .RN(resetLfsrPICuT), .Q(QoutLfsrPICuT[0]) );
DFEP1 U1LfsrPICuT ( .D(QoutLfsrPICuT[2]), .E(enableLfsrPICuT), .C(clock), .SN(resetLfsrPICuT), .Q(QoutLfsrPICuT[1]) );
DFEP1 U2LfsrPICuT ( .D(QoutLfsrPICuT[3]), .E(enableLfsrPICuT), .C(clock), .SN(resetLfsrPICuT), .Q(QoutLfsrPICuT[2]) );
DFEC1 U3LfsrPICuT ( .D(QoutLfsrPICuT[4]), .E(enableLfsrPICuT), .C(clock), .RN(resetLfsrPICuT), .Q(QoutLfsrPICuT[3]) );
DFEP1 U4LfsrPICuT ( .D(y1LfsrPICuT), .E(enableLfsrPICuT), .C(clock), .SN(resetLfsrPICuT), .Q(QoutLfsrPICuT[4]) );
XOR20  U5LfsrPICuT ( .A(QoutLfsrPICuT[0]), .B(QoutLfsrPICuT[2]), .Q(y1LfsrPICuT);
assign DataOutLfsrPICuT = QoutLfsrPICuT;
```

MODULAR TYPE

```
wire resetLfsrPICuT;
wire enableLfsrPICuT;
wire [4:0] DataOutLfsrPICuT ;

wire [4:0] QoutLfsrPICuT ;
wire y1LfsrPICuT;

DFEC1 U0LfsrPICuT ( .D(QoutLfsrPICuT[4]), .E(enableLfsrPICuT), .C(clock), .RN(resetLfsrPICuT), .Q(QoutLfsrPICuT[0]) );
DFEP1 U1LfsrPICuT ( .D(QoutLfsrPICuT[0]), .E(enableLfsrPICuT), .C(clock), .SN(resetLfsrPICuT), .Q(QoutLfsrPICuT[1]) );
DFEP1 U2LfsrPICuT ( .D(y1LfsrPICuT), .E(enableLfsrPICuT), .C(clock), .SN(resetLfsrPICuT), .Q(QoutLfsrPICuT[2]) );
XOR20  U5LfsrPICuT ( .A(QoutLfsrPICuT[1]), .B(QoutLfsrPICuT[4]), .Q(y1LfsrPICuT));
DFEC1 U3LfsrPICuT ( .D(QoutLfsrPICuT[2]), .E(enableLfsrPICuT), .C(clock), .RN(resetLfsrPICuT), .Q(QoutLfsrPICuT[3]) );
DFEP1 U4LfsrPICuT ( .D(QoutLfsrPICuT[3]), .E(enableLfsrPICuT), .C(clock), .SN(resetLfsrPICuT), .Q(QoutLfsrPICuT[4]) );
assign DataOutLfsrPICuT = QoutLfsrPICuT;
```

Table 10: LFSR PI Linear and Modular code slice in Verilog

LINEAR TYPE

```
entity LfsrPICuT is
port( clock, reset, enable: in std_logic;
     DataOut: out std_logic_vector(4 downto 0));
end LfsrPICuT;

architecture comportamento of LfsrPICuT is
  signal Qin, Qout: std_logic_vector (4 downto 0);
begin

  comb_LfsrVhdlLinear: process(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(0) xor Qout(2);
   end if;
end process;

sinc_LfsrVhdlLinear: process(clock,reset)
begin
  if reset = '0' then
    Qout <= "10110";
  elsif clock'event and clock = '1' then
    Qout <= Qin;
  end if;
end process;

   DataOut <= Qout;

end comportamento;
```

MODULAR TYPE

```
entity LfsrPICuT is
port( clock, reset, enable: in std_logic;
     DataOut: out std_logic_vector(4 downto 0));
end LfsrPICuT;

architecture comportamento of LfsrPICuT is
  signal Qin, Qout: std_logic_vector (4 downto 0);
begin

  comb_VhdlLFSRModular: process(Qout,enable)
  begin
   if enable = '0' then
    Qin <= Qout;
   else
     Qin(0)<=Qout(4);
     Qin(1)<=Qout(0);
     Qin(2)<=Qout(1) xor Qout(4);
     Qin(3)<=Qout(2);
     Qin(4)<=Qout(3);
   end if;
end process;

sinc_VhdlLFSRModular: process(clock,reset)
begin
  if reset = '0' then
    Qout <= "10110";
  elsif clock'event and clock = '1' then
    Qout <= Qin;
  end if;
end process;

   DataOut <= Qout;

end comportamento;
```

Table 11: LFSR PI Linear and Modular code slice in VHDL

4.1.3 LFSR SCAN

The LFSR Scan block is similar to LFSR PI. The differences rely on the fact that this LFSR will generate the test vectors for the CUT's scan chain. This fact implies that only one output will be used, and the vectors are serialized to CUT's scan chain. It is possible to use a unique LFSR module to generate simultaneously the CUT's primary input test vectors and the scan chain test vectors. However, as it is shown in [75], two separate LFSR blocks will lead to better test results, achieving higher test coverage results.

When this block is sending data, the Test_SE line (Controller to CUT) has to be enabled in order to switch all the internal flip-flops to scan mode. The objective is to load it with known values contained in the LFSR structure. The number of clock pulses when in scan mode is the same as the number of the flip-flops contained in the sequential part of the CUT in order to shift all.

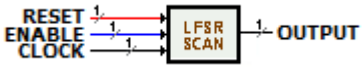


Figure 20: LFSR Scan

The block when requested is also capable to generate two different types of LFSRs. The Table 12 shows two examples, the first, a linear one and the second a modular type. The seed is also '10110' in binary format. This is five bits LFSR like the previous one and its task is to load in serial the flip-flop chain of the CUT.

Also an analogue example but this time in VHDL format is provided in the Table 13.

LINEAR TYPE

```
wire resetLfsrSCANCuT;
wire enableLfsrSCANCuT;
wire DataOutLfsrSCANCuT ;

wire [4:0] QoutLfsrSCANCuT ;
wire y1LfsrSCANCuT;

DFEC1 U0LfsrSCANCuT ( .D(QoutLfsrSCANCuT[1]), .E(enableLfsrSCANCuT), .C(clock), .RN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[0]) );
DFEP1 U1LfsrSCANCuT ( .D(QoutLfsrSCANCuT[2]), .E(enableLfsrSCANCuT), .C(clock), .SN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[1]) );
DFEP1 U2LfsrSCANCuT ( .D(QoutLfsrSCANCuT[3]), .E(enableLfsrSCANCuT), .C(clock), .SN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[2]) );
DFEC1 U3LfsrSCANCuT ( .D(QoutLfsrSCANCuT[4]), .E(enableLfsrSCANCuT), .C(clock), .RN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[3]) );
DFEP1 U4LfsrSCANCuT ( .D(y1LfsrSCANCuT), .E(enableLfsrSCANCuT), .C(clock), .SN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[4]) );

XOR20  U5LfsrSCANCuT ( .A(QoutLfsrSCANCuT[0]), .B(QoutLfsrSCANCuT[2]), .Q(y1LfsrSCANCuT);

assign DataOutLfsrSCANCuT = QoutLfsrSCANCuT[0];
```

MODULAR TYPE

```
wire resetLfsrSCANCuT;
wire enableLfsrSCANCuT;
wire DataOutLfsrSCANCuT ;

wire [4:0] QoutLfsrSCANCuT ;
wire y1LfsrSCANCuT;

DFEC1 U0LfsrSCANCuT ( .D(QoutLfsrSCANCuT[4]), .E(enableLfsrSCANCuT), .C(clock), .RN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[0]) );
DFEP1 U1LfsrSCANCuT ( .D(QoutLfsrSCANCuT[0]), .E(enableLfsrSCANCuT), .C(clock), .SN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[1]) );
DFEP1 U2LfsrSCANCuT ( .D(y1LfsrSCANCuT), .E(enableLfsrSCANCuT), .C(clock), .SN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[2]) );
XOR20  U5LfsrSCANCuT ( .A(QoutLfsrSCANCuT[1]), .B(QoutLfsrSCANCuT[4]), .Q(y1LfsrSCANCuT));
DFEC1 U3LfsrSCANCuT ( .D(QoutLfsrSCANCuT[2]), .E(enableLfsrSCANCuT), .C(clock), .RN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[3]) );
DFEP1 U4LfsrSCANCuT ( .D(QoutLfsrSCANCuT[3]), .E(enableLfsrSCANCuT), .C(clock), .SN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[4]) );

assign DataOutLfsrSCANCuT = QoutLfsrSCANCuT[4];
```

Table 12: LFSR Scan Linear and Modular code slice in Verilog

LINEAR TYPE

```
entity LfsrSCANCuT is
port( clock: in std_logic;
      reset: in std_logic;
      enable: in std_logic;
      DataOut: out std_logic);
end LfsrSCANCuT;

architecture comportamento of LfsrSCANCuT is
  signal Qin: std_logic_vector (4 downto 0);
  signal Qout: std_logic_vector (4 downto 0);
begin

  comb_LfsrVhdlLinear: process(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(0) xor Qout(2);
   end if;
end process;

sinc_LfsrVhdlLinear: process(clock,reset)
begin
  if reset = '0' then
    Qout <= "10110";
  elsif clock'event and clock = '1' then
    Qout <= Qin;
  end if;
end process;

   DataOut <= Qout(0);

end comportamento;
```

MODULAR TYPE

```
entity LfsrSCANCuT is
port( clock: in std_logic;
      reset: in std_logic;
      enable: in std_logic;
      DataOut: out std_logic);
end LfsrSCANCuT;

architecture comportamento of LfsrSCANCuT is
  signal Qin: std_logic_vector (4 downto 0);
  signal Qout: std_logic_vector (4 downto 0);
begin

  comb_VhdlLFSRModular: process(Qout,enable)
  begin
   if enable = '0' then
    Qin <= Qout;
   else
     Qin(0)<=Qout(4);
     Qin(1)<=Qout(0);
     Qin(2)<=Qout(1) xor Qout(4);
     Qin(3)<=Qout(2);
     Qin(4)<=Qout(3);
   end if;
end process;

sinc_VhdlLFSRModular: process(clock,reset)
begin
  if reset = '0' then
    Qout <= "10110";
  elsif clock'event and clock = '1' then
    Qout <= Qin;
  end if;
end process;

   DataOut <= Qout(4);

end comportamento;
```

Table 13: LFSR Scan Linear and Modular code slice in VHDL

4.1.4 LFSR SCAN COUNTER

The LFSR Scan Counter structure is related with the number of flip-flops comprising the CUT's chain, since its function is to count the number of clocks to scan in to CUT's flip-flops the test vectors generated in LFSR Scan block. Therefore, the flip-flops number in LFSR Scan Counter block should be the round up next integer from $\log_2(k)$ where k is the flip-flops number in the CUT's scan chain. The block starts its count and, when it reaches the end, receives information to suspend the process for some time. When the block receive a new instruction to continue, the LFSR returns to the starting position and begin all the process again. This can be repeated several times depending on the number of the LFSR PI test patterns.

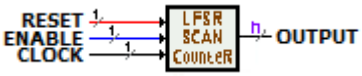


Figure 21: LFSR Scan Counter

Two different types of LFSRs can be generated with the LFSR Scan Counter. The Table 14 shows the differences with the first, a linear one and the second a modular type. The seed for the Verilog example is '01' and is two bits LFSR because the number of the CUT flip-flops is two.

The VHDL analogue example is provided in the Table 15. The seed value this time is '11' because a random process is present to select the seed for the block.

LINEAR TYPE

```
wire resetLfsrSCountCuT;
wire enableLfsrSCountCuT;
wire [1:0] DataOutLfsrSCountCuT ;

wire [1:0] QoutLfsrSCountCuT ;
wire y1LfsrSCountCuT;

DFEP1 U0LfsrSCountCuT ( .D(QoutLfsrSCountCuT[1]), .E(enableLfsrSCountCuT), .C(clock), .SN(resetLfsrSCountCuT), .Q(QoutLfsrSCountCuT[0]));
DFEC1 U1LfsrSCounterCuT ( .D(y1LfsrSCounterCuT), .E(enableLfsrSCounterCuT), .C(clock), .RN(resetLfsrSCounterCuT), .Q(QoutLfsrSCountCuT[1]));

XOR20  U2LfsrSCountCuT ( .A(QoutLfsrSCountCuT[0]), .B(QoutLfsrSCountCuT[1]), .Q(y1LfsrSCountCuT);

assign DataOutLfsrSCountCuT = QoutLfsrSCountCuT;
```

MODULAR TYPE

```
wire resetLfsrSCountCuT;
wire enableLfsrSCountCuT;
wire [1:0] DataOutLfsrSCountCuT ;

wire [1:0] QoutLfsrSCountCuT ;
wire y1LfsrSCountCuT;

DFEP1 U0LfsrSCountCuT ( .D(QoutLfsrSCountCuT[1]), .E(enableLfsrSCountCuT), .C(clock), .SN(resetLfsrSCountCuT), .Q(QoutLfsrSCountCuT[0]) );
DFEC1 U1LfsrSCountCuT ( .D(y1LfsrSCountCuT), .C(clock), .RN(resetLfsrSCountCuT), .Q(QoutLfsrSCountCuT[1]) );
XOR20  U2LfsrSCountCuT ( .A(QoutLfsrSCountCuT[0]), .B(QoutLfsrSCountCuT[1]), .Q(y1LfsrSCountCuT));

assign DataOutLfsrSCountCuT = QoutLfsrSCountCuT;
```

Table 14: LFSR Scan Counter Linear and Modular code slice in Verilog

LINEAR TYPE

```
entity LfsrScanCounterCuT is
port( clock: in std_logic;
      reset: in std_logic;
      enable: in std_logic;
      DataOut: out std_logic_vector(1 downto 0));
end LfsrScanCounterCuT;

architecture comportamento of LfsrScanCounterCuT is
  signal Qin: std_logic_vector (1 downto 0);
  signal Qout: std_logic_vector (1 downto 0);
begin

  comb_LfsrVhdlLinear: process(Qout,enable)
  begin
   if enable = '0' then
    Qin <= Qout;
   else
     Qin(0)<=Qout(1);
     Qin(1)<=Qout(0) xor Qout(1);
   end if;
end process;

sinc_LfsrVhdlLinear: process(clock,reset)
begin
  if reset = '0' then
    Qout <= "11";
  elsif clock'event and clock = '1' then
    Qout <= Qin;
  end if;
end process;

   DataOut <= Qout;

end comportamento;
```

MODULAR TYPE

```
entity LfsrScanCounterCuT is
port( clock: in std_logic;
      reset: in std_logic;
      enable: in std_logic;
      DataOut: out std_logic_vector(1 downto 0));
end LfsrScanCounterCuT;

architecture comportamento of LfsrScanCounterCuT is
  signal Qin: std_logic_vector (1 downto 0);
  signal Qout: std_logic_vector (1 downto 0);
begin

  comb_VhdlLFSRModular: process(Qout,enable)
  begin
   if enable = '0' then
    Qin <= Qout;
   else
     Qin(0)<=Qout(1);
     Qin(1)<=Qout(0) xor Qout(1);
   end if;
end process;

sinc_VhdlLFSRModular: process(clock,reset)
begin
  if reset = '0' then
    Qout <= "11";
  elsif clock'event and clock = '1' then
    Qout <= Qin;
  end if;
end process;

   DataOut <= Qout;

end comportamento;
```

Table 15: LFSR Scan Counter Linear and Modular code slice in VHDL

4.1.5 MISR BLOCK

The MISR (Multiple Input Signature Register) block is based on the LFSR's model but with multiple input bits connected to the flip-flops of the MISR by XOR gates. The number of inputs should be fewer than the number of flip-flops in the MISR and these inputs are actually the primary outputs of the CUT and the output of its scan chain. To avoid aliasing in a test sequence, the MISR should be as high as possible, considering that as the higher as the length is, the higher area overhead we will have in the circuit, but the slowest possibility of having aliasing. The block is presented in Figure 22 and it's composed by the following signals; reset, enable, clock, input bus lines presented as *n* variable, and the MISR_out.

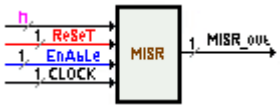


Figure 22: MISR Block Diagram

The inputs of the MISR will provide connection to CUT's outputs through a bus which also connects to the outputs of the overall block. In the following is presented an example in Verilog and VHDL format of a MISR specific case with five flip-flops.

Seed example - 01100

```
module LfsrMisrCuT ( InputSLfsrMisrCuT, clock, resetLfsrMisrCuT, enableLfsrMisrCuT, DataOutLfsrMisrCuT );
  input clock, resetLfsrMisrCuT, enableLfsrMisrCuT;
  input [1:0] InPutSLfsrMisrCuT ;
  output DataOutLfsrMisrCuT;

  wire [4:0] QoutLfsrMisrCuT ;
  wire y1LfsrMisrCuT, x1LfsrMisrCuT, x2LfsrMisrCuT;

  DFEC1 U0LfsrMisrCuT ( .D(x1LfsrMisrCuT), .E(enableLfsrMisrCuT), .C(clock), .RN(resetLfsrMisrCuT), .Q(QoutLfsrMisrCuT[0]) );
  XOR20  U6LfsrMisrCuT ( .A(QoutLfsrMisrCuT[1]), .B(InputSLfsrMisrCuT[0]), .Q(x1LfsrMisrCuT));
  DFEC1 U1LfsrMisrCuT ( .D(x2LfsrMisrCuT), .E(enableLfsrMisrCuT), .C(clock), .RN(resetLfsrMisrCuT), .Q(QoutLfsrMisrCuT[1]) );
  XOR20  U7LfsrMisrCuT ( .A(QoutLfsrMisrCuT[2]), .B(InputSLfsrMisrCuT[1]), .Q(x2LfsrMisrCuT));
  DFEP1 U2LfsrMisrCuT ( .D(QoutLfsrMisrCuT[3]), .E(enableLfsrMisrCuT), .C(clock), .SN(resetLfsrMisrCuT), .Q(QoutLfsrMisrCuT[2]) );
  DFEP1 U3LfsrMisrCuT ( .D(QoutLfsrMisrCuT[4]), .E(enableLfsrMisrCuT), .C(clock), .SN(resetLfsrMisrCuT), .Q(QoutLfsrMisrCuT[3]) );
  DFEC1 U4LfsrMisrCuT ( .D(y1LfsrMisrCuT), .E(enableLfsrMisrCuT), .C(clock), .RN(resetLfsrMisrCuT), .Q(QoutLfsrMisrCuT[4]) );
  XOR20  U5LfsrMisrCuT ( .A(QoutLfsrMisrCuT[0]), .B(QoutLfsrMisrCuT[2]), .Q(y1LfsrMisrCuT) );
  assign DataOutLfsrMisrCuT = QoutLfsrMisrCuT[0];

endmodule
```

Table 16: MISR Linear code slice in Verilog

Seed example - 01100

```
entity LfsrMisrCuT is
port( DataIn: in std_logic_vector(1 downto 0);
      clock: in std_logic;
      reset: in std_logic;
      enable: in std_logic;
      DataOut: out std_logic);
end LfsrMisrCuT;

architecture comportamento of LfsrMisrCuT is
  signal Qin: std_logic_vector (4 downto 0);
  signal Qout: std_logic_vector (4 downto 0);

begin

  comb_VhdlMISRLinear: process(DataIn,Qout,enable)
  begin
   if enable = '0' then
    Qin <= Qout;
   else
     Qin(0)<=Qout(1) xor DataIn(0);
     Qin(1)<=Qout(2) xor DataIn(1);
     Qin(2)<=Qout(3);
     Qin(3)<=Qout(4);
     Qin(4)<=Qout(0) xor Qout(2);
   end if;
end process;

sinc_VhdlMISRLinear: process(clock,reset)
begin
  if reset = '0' then
    Qout <= "01100";
  elsif clock'event and clock = '1' then
    Qout <= Qin;
  end if;
 end process;

  DataOut <= Qout(0);

end Comportamento;
```

Table 17: MISR Linear code slice in VHDL

4.1.6 COMPARATORS

The comparators have the function to integrate comparison logic for a known vector that in a certain moment may arise at the input of this block. Once this vector arrives, the logic that is purely combinatorial, will present at the output a logical ‘1’. For any other combination that can be presented at the input, the logic value is always opposite. It is thus possible in this way to establish a specific point to stop the iterative process of the LFSRs (for the scan counter block and for test length count).

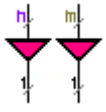


Figure 23: Comparator Blocks (LFSR PI at right / LFSR Scan Counter at left)

The Table 18 and Table 19 show code slices in Verilog and VHDL respectively. Both tables describe two blocks (the LFSR PI and the LFSR Scan Counter). The comparators process will be detailed hereafter in the LFSR's configuration.

LFSR PI Comparator Code	Seed Vector - 10110	Trigger Vector - 00101

```
wire WireConnectLpO;
wire [4:0] LpOInvOutNandIn0;

assign LpOInvOutNandIn0[0] = LfsrPiOut[0];
INV0 LpO00 (.A(LfsrPiOut[1]), .Q(LpOInvOutNandIn0[1]));
assign LpOInvOutNandIn0[2] = LfsrPiOut[2];
INV0 LpO01 (.A(LfsrPiOut[3]), .Q(LpOInvOutNandIn0[3]));
INV0 LpO02 (.A(LfsrPiOut[4]), .Q(LpOInvOutNandIn0[4]));
wire [1:0] LpONandOutInvIn0;
wire [1:0] LpOInvOutNandIn1;
NAND20 NLpO10 (.A(LpOInvOutNandIn0[0]), .B(LpOInvOutNandIn0[1], .Q(LpONandOutInvIn0[0]));
INV0 LpO10 (.A(LpONandOutInvIn0[0]), .Q(LpOInvOutNandIn1[0]));
NAND20 NLpO11 (.A(LpOInvOutNandIn0[2]), .B(LpOInvOutNandIn0[3], .Q(LpONandOutInvIn0[1]));
INV0 LpO11 (.A(LpONandOutInvIn0[1]), .Q(LpOInvOutNandIn1[1]));
wire LpONandOutInvIn1;
wire LpOInvOutNandIn2;
NAND20 NLpO20 (.A(LpOInvOutNandIn1[0]), .B(LpOInvOutNandIn1[1]), .Q(LpONandOutInvIn1));
INV0 LpO20 (.A(LpONandOutInvIn1), .Q(LpOInvOutNandIn2));
wire LpONandOutInvIn2;
wire LpOInvOutNandIn3;
NAND20 NLpO30 (.A(LpOInvOutNandIn2), .B(LpOInvOutNandIn0[4]), .Q(LpONandOutInvIn2));
INV0 LpO30 (.A(LpONandOutInvIn2), .Q(LpOInvOutNandIn3));

assign WireConnectLpO = LpOInvOutNandIn3;
```

LFSR Scan Counter Comparator Code	Seed Vector - 01	Trigger Vector - 10

```
wire WireConnectLscO;
wire [1:0] LscOInvOutNandIn0;

INV0 LscO00 (.A(LfsrScanCounterOut[0]), .Q(LscOInvOutNandIn0[0]));
assign LscOInvOutNandIn0[1] = LfsrScanCounterOut[1];

wire LscONandOutInvIn0;
wire LscOInvOutNandIn1;

NAND20 NLscO10 (.A(LscOInvOutNandIn0[0]), .B(LscOInvOutNandIn0[1], .Q(LscONandOutInvIn0));
INV0 LscO10 (.A(LscONandOutInvIn0), .Q(LscOInvOutNandIn1));

assign WireConnectLscO = LscOInvOutNandIn1;
```

Table 18: LFSR Comparators code slice in Verilog

LFSR PI comparator Code	LFSR Scan Counter comparator Code

```
BistCountEnd: process(LfsrPiOut)
begin
 if LfsrPiOut = "00101" then
  BistCountFinishedOut <= '1';
 else
  BistCountFinishedOut <= '0';
 end if;
end process;
```

```
ScanCountEnd: process(LfsrScanCounterOut)
begin
 if LfsrScanCounterOut = "10" then
  ScanCountFinishedOut <= '1';
 else
  ScanCountFinishedOut <= '0';
 end if;
end process;
```

Table 19: LFSR Comparators code slice in VHDL

4.1.7 CUT

Most integrated circuits incorporate combinational and sequential logic. When a particular company designs a circuit for a specific application and want to add a scan based test, the circuit has to be changed for such purpose. It is necessary to reconfigure circuit's description, by introducing a scan path, signals and functionality.

Basically, for a full-scan methodology, all flip-flops in the CUT are replaced by scan flip-flops, that are able to choose between two inputs: the normal input and a scan input. After all flip-flops have been replaced and when Test_SE signal is in scan mode, the output is connected to the scan input of another flip-flop forming a chain connecting all flip-flops, as shown in Figure 24.

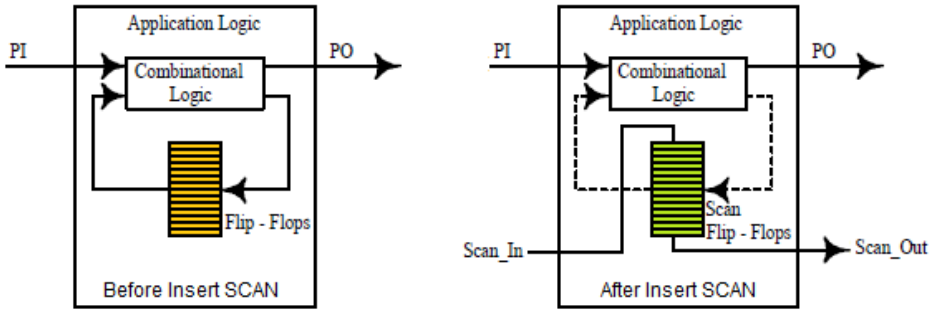


Figure 24: Insertion of a Scan Chain into a CUT

The Table 20 describes a simple circuit with only two inputs and one output in order to clarify the scan insertion method in VHDL environment. Only two flip-flops has two be replaced.

CUT VHDL without scan	CUT VHDL with scan

entity CuT is
port(a, b, clock, reset : in std_logic;
z: out std_logic);
end CuT;
 	
architecture Comportamento of CuT is
 	
signal Qout, Qin: std_logic_vector(1 downto 0);
begin
 	
sinc: process(clock,reset)
begin
if reset = '0' then
Qout <= "00";
elsif clock'event and clock = '1' then
Qout <= Qin;
end if;
end process;
	entity CuT is
	port(a, b, teste_se, teste_si, clock, reset : in std_logic;
	z, scan_out : out std_logic);
	end CuT;
	
	architecture Comportamento of CuT is
	
	signal Qout, Qin, Qin_data, Qin_test: std_logic_vector(1 downto 0);
	begin
	sinc: process(clock,reset)
	begin
	if reset = '0' then
	Qout <= "00";
	elsif clock'event and clock = '1' then
	Qout <= Qin;
	end if;
	end process;
	Qin_data(0) <= b;

```
Qin(0) <= b;
Qin(1) <= Qout(0);

  comb: process(clock,Qout,Qin,a,b)
    begin
      Qin(1) <= not( Qout(0) and a );
      z <= not( not( Qin(1) ) and Qout(1) );
  end process;

end comportamento;
```

```
Qin_test(0) <= teste_si;
Qin_test(1) <= Qout(0);
scan_out <= Qout(1);
  comb_mux: process(clock,teste_se,Qin_data,Qin_test,Qout,Qin,a,b)
    begin
      Qin_data(1) <= not( Qout(0) and a );
      z <= not( not( Qin_data(1) ) and Qout(1) );
        if teste_se = '0' then
          Qin <= Qin_data;
        else
          Qin <= Qin_test;
        end if;
  end process;

end comportamento;
```

Table 20: VHDL CUT before and after Scan insertion.

4.1.8 BIST CONTROLLER

The BIST controller is certainly the most important block of the whole BIST structure. It's the core unit responsible for controlling the instructions that are given to the various blocks, in order to rule the entire self-test functionality. It is also responsible for switching between the normal and test mode. The signal responsible for initialize the self-test is the START pin. Once this line receives a logic '1', the circuit enters in test mode and the finite state machine will change its state. It will leave the *idle* state and it will go to the *reset* state and initiate the test.

The purpose of the *reset* state is to prepare the five blocks, LFSR PI, LFSR Scan, LFSR Scan Counter, CUT and MISR, so that in the next clock pulse these blocks are ready to begin the test sequence. The same analogy has to be applied to the ENABLE line of each block that integrates it, except that this signal will also be used during test to enable or disable specific blocks (e.g., the LFSR PI have to remain disabled when controller is at *scan* state). It is also important in the *reset* state to de-activate the BIST_done pin and enable (logic '1') the Test_SE signal of the CUT, in order to switch all the internal flip-flops to *scan* mode. The MUX_Sel signal should also disable primary inputs and connect the LFSR's signals to CUT's inputs. This state lasts only one clock cycle.

In *scan* mode, starts the loading process of the CUT's scan chain. The data is received serially through the LFSR scan output line and when the load is complete, the LFSR Scan Counter informs the controller that the scan chain is reloaded with a new test pattern.

As soon as the controller receives the command, it will jump to *launch* state and will suspend (logic '0') the enable line of the SCAN Counter block. At the same time the enable LFSR PI line will be activated, supplying a valid and known vector in the respective bus where the CUT primary inputs connect. There is a particular signal in the interconnection between the two blocks of the entire circuit that is very important, and the way it's treated defines the fault coverage as well as the cost to implement the application, which is, the Test_SE line. Two methods can and were used to define the test strategy: LOS and/or LOC.

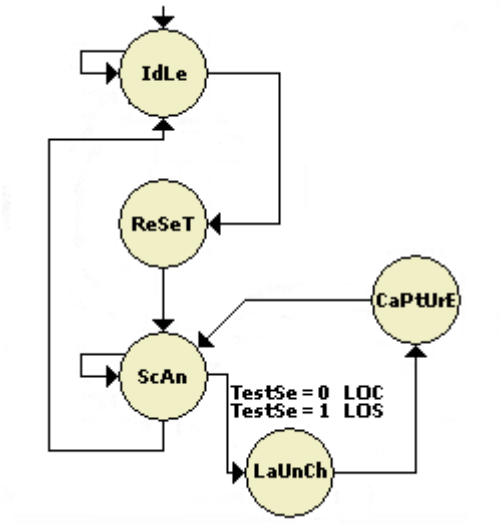


Figure 25: BIST Specific State Machine

In LOC the Test_SE line become logic '0' in the beginning of the *launch* state, making each individual scan flip-flops inside the CUT to switch to normal operation mode and waiting for *capture*. But this in not true for the LOS method that will drag the off state of this line until the beginning of *capture* mode arrives, and is also more complex to implement it in a standard scan test environment (due to the fast clock between Launch and Capture).

Capture mode has finally arrived and is now possible to obtain the first output vector, generated by the first one applied to CUT. Because the MISR block has its inputs connected to the CUT outputs, the output vector is present in the MISR inputs. It is also here that the LFSR Scan and LFSR Scan Counter enable signals will be prepared to be activated again in the next state, and the LFSR PI enable signal has to be disabled at this moment also.

The next state is again the *scan*, and here all the previous process is repeated until the last vector that was defined in the LFSR PI block. As the CUT flip-flops chain is loaded again with the new values, the old ones will be putted clock by clock, one by one, in the MISR Scan_out input, allowing to test not only the combinational logical but also all the flip-flops in the CUT and their interconnection when in *scan* mode. The BIST Controller block is presented in the Figure 26.

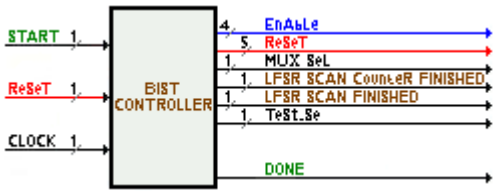


Figure 26: BIST Controller Block Diagram

The Table 21 and the Table 22 show the code that instructs the controller for desired operation. The first table contains the code in Verilog language and the second table in VHDL description.

Controller based on Launch-on-Shift (Verilog)

```
wire BistStart;
wire Clock;
wire ResetController;
wire LfsrPiCountFinished;
wire LfsrScanCountFinished;
wire ResetLfsrPi;
wire ResetLfsrScan;
wire ResetLfsrScanCounter;
wire ResetCut;
wire ResetMisr;
wire EnableLfsrPi;
wire EnableLfsrScan;
wire EnableLfsrScanCounter;
wire EnableMisr;
wire TestSE;
wire MuxSelect;
wire BistDone;

// -- L . O . S

wire LonSn1;
wire LonSn3;
wire LonSn4;
wire LonSn5;
wire LonSn6;
wire LonSn7;
wire LonSn8;
wire LonSn9;
wire LonSn11;
wire LonSn12;
wire LonSn13;
wire LonSn14;
```

```
wire LonSn15;
wire LonSn16;
wire LonSn17;
wire LonSn18;
wire LonSn19;
wire LonSn20;

wire [2:0] estado;
wire [2:0] estado_seguinte;

assign ResetLfsrPi = ResetMisr;
assign ResetLfsrScan = ResetMisr;
assign EnableLfsrScan = TestSE;
assign EnableLfsrPi = estado_seguinte[2];

DFC3 \estado_reg[0]  ( .D(estado_seguinte[0]), .C(Clock), .RN(ResetController), .Q(estado[0]) );
DFC1 \estado_reg[1]  ( .D(estado_seguinte[1]), .C(Clock), .RN(ResetController), .Q(estado[1]), .QN(LonSn1) );
DFC3 \estado_reg[2]  ( .D(estado_seguinte[2]), .C(Clock), .RN(ResetController), .Q(estado[2]) );
INV3 U3ControlleR ( .A(LonSn13), .Q(EnableMisr) );


CLKIN0 U4ControlleR ( .A(LonSn3), .Q(ResetMisr) );
NOR20 U5ControlleR ( .A(estado_seguinte[2]), .B(ResetLfsrScanCounter), .Q(LonSn3) );
AOI2110 U6ControlleR ( .A(LonSn1), .B(LonSn4), .C(LonSn5), .D(LonSn6), .Q(ResetLfsrScanCounter) );
CLKIN0 U7ControlleR ( .A(LonSn7), .Q(LonSn5) );
OAI210 U8ControlleR ( .A(estado[0]), .B(estado[1]), .C(estado[2]), .Q(LonSn7) );
CLKIN0 U9ControlleR ( .A(LonSn8), .Q(ResetCut) );
NOR40 U10ControlleR ( .A(LonSn9), .B(LonSn6), .C(TestSE), .D(estado[2]), .Q(LonSn8) );
CLKIN0 U11ControlleR ( .A(LonSn11), .Q(LonSn6) );
NOR20 U12ControlleR ( .A(estado_seguinte[0]), .B(estado[1]), .Q(LonSn9) );
CLKIN0 U13ControlleR ( .A(LonSn12), .Q(MuxSelect) );
NOR20 U14ControlleR ( .A(EnableMisr), .B(estado_seguinte[0]), .Q(LonSn12) );
NOR20 U15ControlleR ( .A(TestSE), .B(estado_seguinte[2]), .Q(LonSn13) );

OAI210 U16ControlleR ( .A(BistDone), .B(LonSn14), .C(LonSn15), .Q(TestSE) );
OAI310 U17ControlleR ( .A(LonSn14), .B(estado_seguinte[0]), .C(BistDone), .D(LonSn15), .Q(EnableLfsrScanCounter) );
NOR30 U18ControlleR ( .A(estado_seguinte[1]), .B(estado_seguinte[2]), .C(estado_seguinte[0]), .Q(BistDone) );
OAI310 U19ControlleR ( .A(LonSn14), .B(LfsrPiCountFinished), .C(LonSn16), .D(LonSn17), .Q(estado_seguinte[0]) );
NAND30 U20ControlleR ( .A(LonSn4), .B(LonSn1), .C(BistStart), .Q(LonSn17) );
NOR20 U21ControlleR ( .A(LonSn11), .B(estado[2]), .Q(estado_seguinte[2]) );
NAND20 U22ControlleR ( .A(estado[1]), .B(estado[0]), .Q(LonSn11) );
OAI210 U23ControlleR ( .A(LonSn18), .B(LonSn14), .C(LonSn15), .Q(estado_seguinte[1]) );
CLKIN0 U24ControlleR ( .A(LonSn19), .Q(LonSn15) );
AOI2110 U25ControlleR ( .A(estado[0]), .B(estado[2]), .C(LonSn4), .D(estado[1]), .Q(LonSn19));
NAND20 U26ControlleR ( .A(LonSn4), .B(estado[1]), .Q(LonSn14) );
NOR20 U27ControlleR ( .A(estado[0]), .B(estado[2]), .Q(LonSn4) );
NOR20 U28ControlleR ( .A(LonSn20), .B(LonSn16), .Q(LonSn18) );
CLKIN0 U29ControlleR ( .A(LfsrScanCountFinished), .Q(LonSn16) );
CLKIN0 U30ControlleR ( .A(LfsrPiCountFinished), .Q(LonSn20) );
```

Table 21: Verilog Controller code in Launch-on-Shift

Controller based on Launch-on-Shift (VHDL)

```
entity BistControllerCuT is
port( BistStart : in std_logic;
      Clock : in std_logic;
      ResetController : in std_logic;
      ResetLfsrPi : out std_logic;
      ResetLfsrScan : out std_logic;
      ResetLfsrScanCounter : out std_logic;
      ResetCut : out std_logic;
      ResetMisr : out std_logic;
      LfsrPiCountFinished : in std_logic;
      LfsrScanCountFinished : in std_logic;
      EnableLfsrPi : out std_logic;
      EnableLfsrScan : out std_logic;
      EnableLfsrScanCounter : out std_logic;
      EnableMisr : out std_logic;
      TestSE : out std_logic;
      MuxSelect : out std_logic;
      BistDone : out std_logic);
end BistControllerCuT;
```

```
architecture comportamento of BistControllerCuT is

type estados is (IDLE,RESET,SCAN,LAUNCH,CAPTURE);
signal estado,estado_seguinte:estados;

begin

saidas_comb:process(estado,estado_seguinte)

 begin
  case estado is
    when IDLE =>

    ResetLfsrPi <= '0';
    ResetLfsrScan <= '0';
    ResetLfsrScanCounter <= '0';
    ResetCut <= '1';
    ResetMisr <= '0';
    EnableLfsrScan <= '0';
    EnableLfsrPi <= '0';
    EnableLfsrScanCounter <= '0';

   EnableMisr <= '0';
    TestSE <= '0';
    MuxSelect <= '0';
    BistDone <= '1';

      if estado_seguinte=RESET then
        ResetCut <= '0';
        MuxSelect <= '1';
        BistDone <= '0';
      end if;

    when RESET =>

    ResetLfsrPi <= '1';
    ResetLfsrScan <= '1';


    ResetLfsrScanCounter <= '1';
    ResetCut <= '1';
    ResetMisr <= '1';
    EnableLfsrScan <= '1';
    EnableLfsrPi <= '0';
    EnableLfsrScanCounter <= '1';
    EnableMisr <= '1';
    TestSE <= '1';
    MuxSelect <= '1';
    BistDone <= '0';

    when SCAN =>

    ResetLfsrPi <= '1';
    ResetLfsrScan <= '1';
    ResetLfsrScanCounter <= '1';
    ResetCut <= '1';
    ResetMisr <= '1';
    EnableLfsrScan <= '1';
    EnableLfsrPi <= '0';
    EnableLfsrScanCounter <= '1';
    EnableMisr <= '1';
    TestSE <= '1';
    MuxSelect <= '1';
    BistDone <= '0';

      if estado_seguinte=LAUNCH then

        EnableLfsrScan <= '1';
        EnableLfsrPi <= '0';
        EnableLfsrScanCounter <= '0';
        EnableMisr <= '1';
        TestSE <= '1';

      elsif estado_seguinte=IDLE then

        ResetCut <= '0';
        EnableLfsrScan <= '0';
        EnableLfsrPi <= '0';
        EnableLfsrScanCounter <= '0';
        EnableMisr <= '0';
        TestSE <= '0';
        MuxSelect<='0';
        BistDone<='1';
```

```
    end if;

  when LAUNCH =>

  ResetLfsrPi <= '1';
  ResetLfsrScan <= '1';
  ResetLfsrScanCounter <= '0';
  ResetCut <= '1';
  ResetMisr <= '1';
  EnableLfsrScan <= '0';
  EnableLfsrPi <= '1';
  EnableLfsrScanCounter <= '0';
  EnableMisr <= '1';
  TestSE <= '0';
  MuxSelect <= '1';
  BistDone <= '0';

  when CAPTURE =>

  ResetLfsrPi <= '1';


  ResetLfsrScan <= '1';
  ResetLfsrScanCounter <= '1';
  ResetCut <= '1';
  ResetMisr <= '1';
  EnableLfsrScan <= '1';
  EnableLfsrPi <= '0';
  EnableLfsrScanCounter <= '1';
  EnableMisr <= '1';
  TestSE <= '1';
  MuxSelect <= '1';
  BistDone <= '0';

  when others =>

  ResetLfsrPi <= '0';
  ResetLfsrScan <= '0';
  ResetLfsrScanCounter <= '0';
  ResetCut <= '1';
  ResetMisr <= '0';
  EnableLfsrScan <= '0';
  EnableLfsrPi <= '0';
  EnableLfsrScanCounter <= '0';
  EnableMisr <= '0';
  TestSE <= '0';
  MuxSelect <= '0';
  BistDone <= '1';

end case;
end process;

CTRL_comb:process(estado,BistStart,LfsrPiCountFinished,LfsrScanCountFinished)

begin
  case estado is
    when IDLE=>
      if BistStart='1' then
        estado_seguinte<=RESET;
      else
        estado_seguinte<=IDLE;
      end if;
    when RESET=>
      estado_seguinte<=SCAN;
    when SCAN=>
      if LfsrScanCountFinished='1' and LfsrPiCountFinished='0' then
        estado_seguinte<=LAUNCH;
      elsif LfsrScanCountFinished='1' and LfsrPiCountFinished='1' then
        estado_seguinte<=IDLE;
      else
        estado_seguinte<=SCAN;
      end if;
    when LAUNCH=>
      estado_seguinte<=CAPTURE;
    when CAPTURE=>
      estado_seguinte<=SCAN;
    when others=>
      estado_seguinte<=IDLE;
  end case;
 end process;
```

```
  CTRL_seq:process(Clock,ResetController)
    begin
      if ResetController='0' then
        estado<=IDLE;
      elsif Clock'event and Clock='1' then
        estado<=estado_seguinte;
      end if;
  end process;

end comportamento;
```

Table 22: VHDL Controller code in Lunch-on-Shift

4.2 BISTGEN SOFTWARE

In order to automate the whole methodology of the testing process for digital CMOS integrated circuits, a software tool called BISTGen was developed, which integrates and automates all the procedures described in section 4.1. This present section describes it, explaining in detail the most important functions and procedures.

The BISTGen software application was developed with the use of Object Pascal (Pascal version of object-oriented programming), using the compiler Embarcadero ® Delphi ® 2010. It is a tool to be used on Windows XP ® operating system, or all their latest versions (for example, Windows 7 ®).

The main purpose of the tool is to automate a file generation process with BIST functionality inside, preparing circuits for test. Starting from a specific input file containing a circuit's description with scan method already implemented, it will be possible to generate a new circuit description that integrates the BIST mechanism for automatic test that will allow simulating the entire circuit during its period of operation whenever desired.

4.2.1 DATA ENTRY

Data entry is made in the program through a Verilog structural file (.v) or a VHDL behavioural file (.vhdl). Whatever the file that is present, it must include the scan path method and the respective control pins must be present.

4.2.2 APPLICATION FLOWCHART

Once the application is invoked, it must be chosen from two files that may be either Verilog or VHDL. As mentioned, the file should have integrated the scan method, because when loaded, it will be prompted to register the names of the control lines of the CUT including the new associated lines resulting from the method addition.

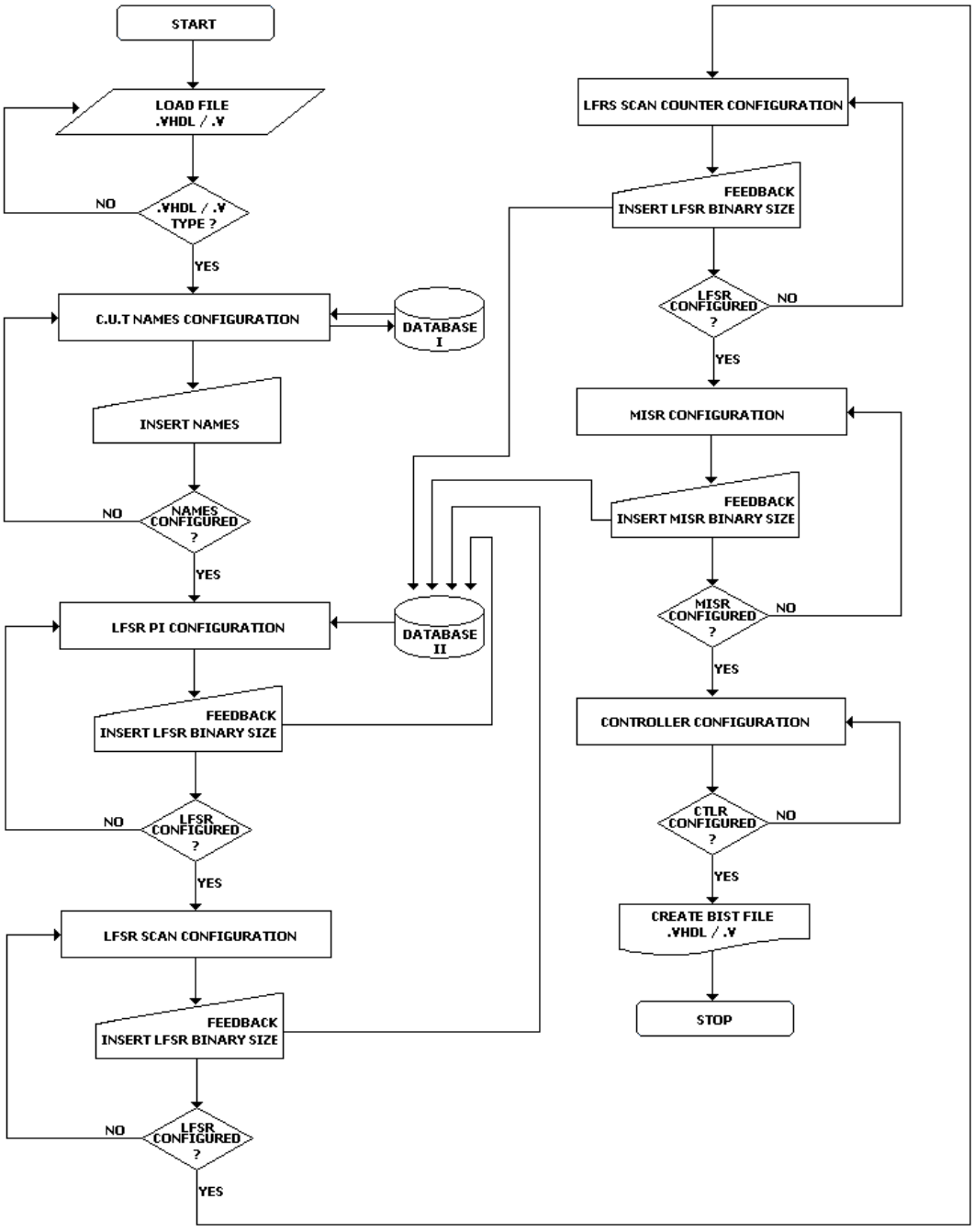


Figure 27: Application Flowchart

During program execution, the user will be guided through each block specification and generation until the all new circuitry (including CUT) is generated. In this work we call it the Aggregate circuit.

4.2.3 DATABASE ARCHITECTURE AND COMPOSITION

The system database chosen was the Paradox. Paradox is a relational database management system currently published by Corel Corporation. It was originally released for DOS by Ansa Software, and then by Borland after it bought the company [21]. A Windows version was released by Borland in 1992. At first glance, the Paradox tables do not show many differences from InterBase tables and the following similarities are evident.

- Access can be done through an alias;
- The types of possible fields are similar, although they have different names;
- Tables can be created with the DataBase Desktop;
- Are used the same components TTable and TQuery to access;

In reality, the BDE (Borland DataBase Engine) creates an illusion that InterBase and Paradox tables behave the same way. For some developers, however this illusion ends soon. The first disappointment comes in database using the Desktop for manipulating tables InterBase. While the Database Desktop is the ideal tool for creating and restructuring tables, Paradox is deficient with respect to InterBase, where the restructuring and the use of more advanced features can only be achieved by mounting scripts that will run on InterBase Windows. Searches and indexes in InterBase are case sensitive, while in Paradox differentiation is configurable. Still in InterBase defining primary and foreign keys is performed easily, but changing these keys is not so trivial. Some operations using InterBase are slower than in Paradox. It quickly becomes clear that the InterBase is not automatically better than Paradox. The two products have significant differences and the choice of which to use is fully dependent on the conditions and objectives of the final application.

The Figure 28 shows the database components architecture and a brief description of its internal behavior will also be given.

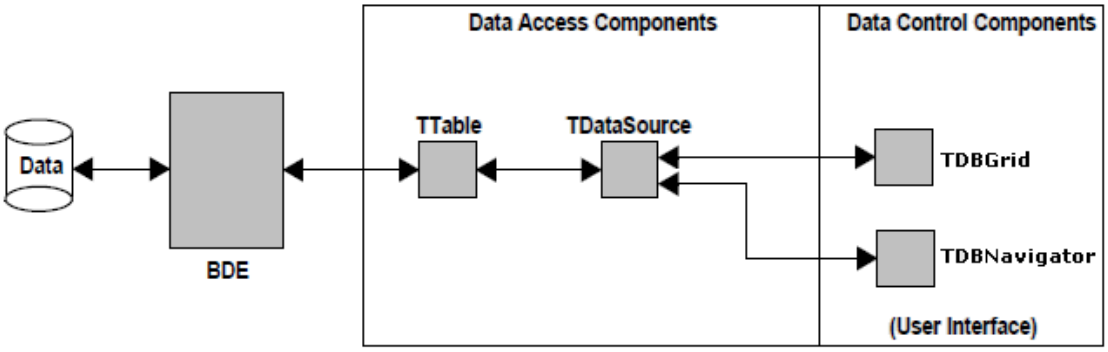


Figure 28: Database Components Architecture

Every dataset that supplies a data control component must have at least one TDataSource Component. TDataSource acts as a bridge between one TTable and one or more data control components that provide a visible user interface to data. TTable can establish connections to a database through the BDE, but cannot display database information on a Form. Data Control components as TDBGrid and TDBNavigator provide the visible user interface and manipulation to data, but are unaware of the structure of the table from which they receive (and to which they send) data.

The application uses two database tables to store data information. In one stores the names of the inputs and outputs of the CUT for further manipulation, and in the other, stores the values of the feedback loops that are associated with the size of the LFSRs.

4.2.4 LFSR'S CONFIGURATION

The user chooses the number of counts in binary format. For example if the software receives the (10110) binary value (the seed), it will count 31 times ($2^5 - 1$). Due to the features of the LFSR, the (00000) value can't be used ($2^n - 1$), otherwise the LFSR would stay in this value indefinitely, because of the XOR properties in the feedback loops.

In the previous section, it was mentioned that the controller receives information when some LFSRs can reach the score limit or the score limit imposed, depending on the case. Whenever the user defines the binary LFSR seed, it also sets the counting number limit, in the case of the LFSR PI. The software, after receiving the first vector, will in background generate all the patterns (process explained hereafter with the LFSR Scan Counter) until repeat the first value. What is important to retain is, at the end, the first and the last values are known. The Table 23 shows the correlation between the binary seed and the type of flip-flops chosen. When a '0' is present the DFEC1 flip-flop is used, which mean a 'D' flip-flop with 'enable' and 'clear', but when a '1' is present the used flip-flop is a 'D' type with 'enable' and 'preset' (DFEP1), which defines the initial state based on the LFSR's seed.

Verilog LFSR PI file with **10011** seed value (First value)

10011 - first generated value

00110 - last generated value

```
module LfsrPICuT ( clock, resetLfsrPICuT,enableLfsrPICuT, DataOutLfsrPICuT );
  input clock, resetLfsrPICuT,enableLfsrPICuT;
  output [4:0] DataOutLfsrPICuT ;

  wire [4:0] QoutLfsrPICuT ;
  wire y1LfsrPICuT;

  DFEP1 U0LfsrPICuT ( .D(QoutLfsrPICuT[1]), .E(enableLfsrPICuT), .C(clock), .SN(resetLfsrPICuT), .Q(QoutLfsrPICuT[0]) );
  DFEP1 U1LfsrPICuT ( .D(QoutLfsrPICuT[2]), .E(enableLfsrPICuT), .C(clock), .SN(resetLfsrPICuT), .Q(QoutLfsrPICuT[1]) );
  DFEC1 U2LfsrPICuT ( .D(QoutLfsrPICuT[3]), .E(enableLfsrPICuT), .C(clock), .RN(resetLfsrPICuT), .Q(QoutLfsrPICuT[2]) );
  DFEC1 U3LfsrPICuT ( .D(QoutLfsrPICuT[4]), .E(enableLfsrPICuT), .C(clock), .RN(resetLfsrPICuT), .Q(QoutLfsrPICuT[3]) );
  DFEP1 U4LfsrPICuT ( .D(y1LfsrPICuT), .E(enableLfsrPICuT), .C(clock), .SN(resetLfsrPICuT), .Q(QoutLfsrPICuT[4]) );

  XOR20  U5LfsrPICuT ( .A(QoutLfsrPICuT[0]), .B(QoutLfsrPICuT[2]), .Q(y1LfsrPICuT);

  assign DataOutLfsrPICuT = QoutLfsrPICuT;

endmodule
```

Table 23: Linear type Verilog LFSR PI File

The stop value (00110) is the output trigger in the comparator block. The hardware description to create it in Verilog format is more complex than in VHDL. In VHDL the code describes a behavioural and then a synthesizer process it; however in Verilog is completely different because the code description is structural and it must be defined at gate level. The Figure 29 shows the dynamic gate design for this case.

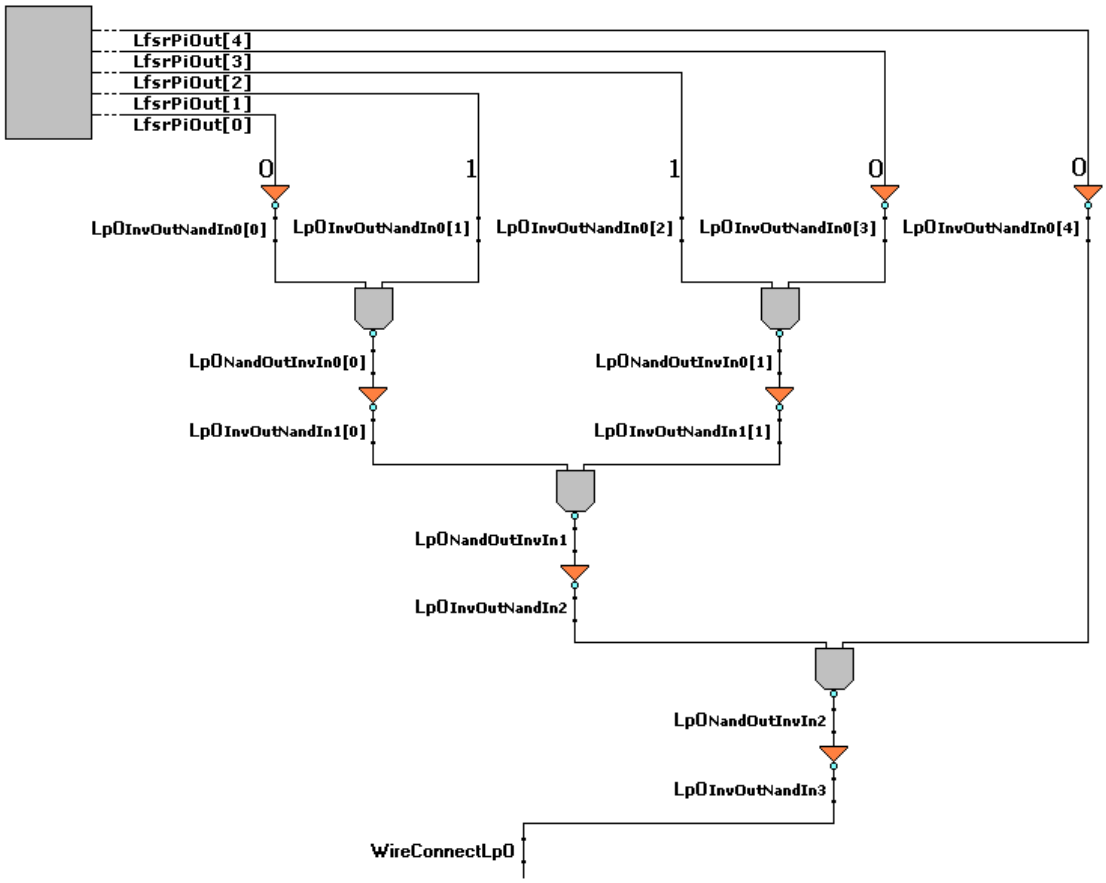


Figure 29: Comparator block for LFSR PI Patterns

This is the internal circuit of the comparator block with 5 inputs coming from the LFSR PI, for the specific pattern (00110). Other pattern or different inputs number lead to another circuit. The goal is when a specific pattern arises, the internal logic give a binary '1' in its output, exclusively. The software after receive the pattern will decide the internal logic to achieve the result. This is a dynamic process where the components are chosen, as the wires to connect it in the right way. The names in the Figure 29 give a more clear idea to understand the code in the Table 24 of the comparator block.

01100 - last generated value (the stop condition)

```
.
.
.
wire WireConnectLpO;
wire [4:0] LpOInvOutNandIn0;
INV0 LpO00 (.A(LfsrPiOut[0]), .Q(LpOInvOutNandIn0[0]));
INV0 LpO01 (.A(LfsrPiOut[1]), .Q(LpOInvOutNandIn0[1]));
assign LpOInvOutNandIn0[2] = LfsrPiOut[2];
assign LpOInvOutNandIn0[3] = LfsrPiOut[3];
INV0 LpO02 (.A(LfsrPiOut[4]), .Q(LpOInvOutNandIn0[4]));
wire [1:0] LpONandOutInvIn0;
wire [1:0] LpOInvOutNandIn1;
NAND20 NLpO10 (.A(LpOInvOutNandIn0[0]), .B(LpOInvOutNandIn0[1], .Q(LpONandOutInvIn0[0]));
INV0 LpO10 (.A(LpONandOutInvIn0[0]), .Q(LpOInvOutNandIn1[0]));
NAND20 NLpO11 (.A(LpOInvOutNandIn0[2]), .B(LpOInvOutNandIn0[3], .Q(LpONandOutInvIn0[1]));
INV0 LpO11 (.A(LpONandOutInvIn0[1]), .Q(LpOInvOutNandIn1[1]));
wire LpONandOutInvIn1;
wire LpOInvOutNandIn2;
NAND20 NLpO20 (.A(LpOInvOutNandIn1[0]), .B(LpOInvOutNandIn1[1]), .Q(LpONandOutInvIn1));
INV0 LpO20 (.A(LpONandOutInvIn1), .Q(LpOInvOutNandIn2));
wire LpONandOutInvIn2;
wire LpOInvOutNandIn3;
NAND20 NLpO30 (.A(LpOInvOutNandIn2), .B(LpOInvOutNandIn0[4]), .Q(LpONandOutInvIn2));
INV0 LpO30 (.A(LpONandOutInvIn2), .Q(LpOInvOutNandIn3));
assign WireConnectLpO = LpOInvOutNandIn3;
.
.
.
```

Table 24: Comparator Block Code for LFSR PI Patterns

With the LFSR Scan Counter is different. When the user chooses the integer number of counts, the software translates this number in binary format using a $\log_2$ mathematical conversion to achieve the purpose. For example with five counts the software converts the integer number in binary format (010 seed), the converted number may be any in the range of possible values that are 7. The question that arises is why it can be any of the 7 values? The answer is because the software uses a random function. If the integer count number is five it must be represented in 3 bits at least, but with 3 bits it is possible to make 7 counts ($2^3 - 1$). Not all the range is used in this particular case but the software 'knows' when to stop as well as the binary number that must be collected.

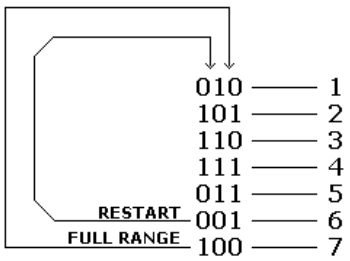


Figure 30 : LFSR Stop Limit and Rotation

Taking the example of the Figure 30, for the first value (010), the next generated six values until repeat the process, will always be the same every cycle. The last one

is (100), index 7, but the software will stop at (001) to establish 5 counts. The stop index is 6 and not 5 because of the design and the software implementation requirements (number of counts plus one (controller issue)). Since we have the exit value, it is possible to establish it as a stop vector in hardware description file. Let's take a look in the Table 25 based this time in VHDL files.

LFSR Scan Counter	BIST File (LFSR stop condition code slice)	Vectors

library IEEE;
use IEEE.std_logic_1164.all;
 	
entity LfsrScanCounterCuT is
port(clock: in std_logic;
reset: in std_logic;
enable: in std_logic;
DataOut: out std_logic_vector(2 downto 0));
end LfsrScanCounterCuT;
 	
architecture comportamento of LfsrScanCounterCuT is
signal Qin: std_logic_vector (2 downto 0);
signal Qout: std_logic_vector (2 downto 0);
begin
 	
comb_LfsrVhdlLinear: process(Qout,enable)
begin
if enable = '0' then
Qin <= Qout;
else
Qin(0)<=Qout(1);
Qin(1)<=Qout(2);
Qin(2)<=Qout(0) xor Qout(1);
end if;
end process;
 	
sinc_LfsrVhdlLinear:process(clock,reset)
begin
if reset = '0' then
Qout <= "011";
elsif clock'event and clock = '1' then
Qout <= Qin;
end if;
end process;
	.
	.
	.
	
	ScanCountEnd:
	process(LfsrScanCounterOut)
	begin
	if LfsrScanCounterOut = "110" then
	ScanCountFinishedOut <= '1';
	else
	ScanCountFinishedOut <= '0';
	end if;
	end process;
	.
	.
	.
		011
		001
		100
		010
		101
		110

Table 25: LFSR Scan Counter Stop Counting Process

The first column shows the VHDL hardware description of the LFSR. The '***Qout <="011"***' row shows the first value obtained through the input random process and is the seed. The stop condition is present not in the generated LFSR Scan Counter file but in the global BIST file obtained in the end, in a particular slice of code (the comparator block) where the '***if LfsrScanCounterOut = "110" then***' row is the stop condition.

The description uses VHDL and Verilog as examples. The concept is the same, but in a global BIST file production, only one type of language can be used at a time.

4.2.5 APPLICATION FORMS FUNCTION AND HIERARCHY

For the final file, first we will need to set up all the necessary parameters required in each window. The hierarchy and the sequence of the integral parts of the application can be seen generally in Figure 31. The setting begins in the main window and prompts to choose the file that contains the test circuit.

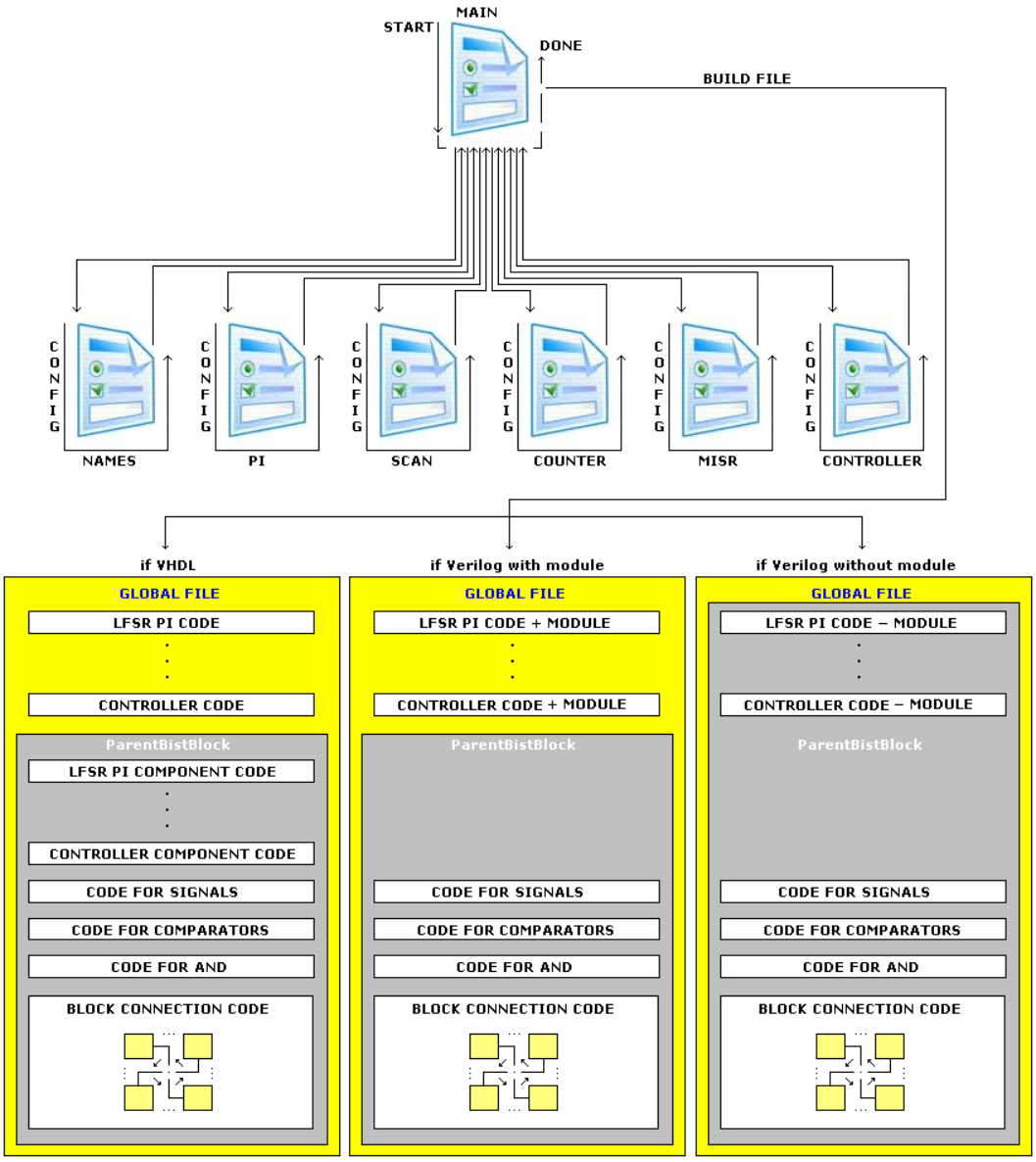


Figure 31: Global File Structure

In the next step, the names contained in the input file will be identified. For this purpose the application provides a window into a second hierarchical level (the leftmost) where the objective is to request the name of the four control signals which by convention are assigned in the program with the names; *clock*, *reset*, *test_se*, *test_si*. All files submitted to the test must already have these names, but may nevertheless have different ones.

Now that the software 'knows' the names, the next step is to configure the LFSRs and the MISR which are the PI, Scan and Counter windows and previously explained in section 4.1. The MISR configuration is similar to the PI configuration since the number of entries is not defined here.

The last window, the rightmost, is intended to configure the controller, to choose which type of method to use: Scan based BIST, LOC, LOS or both LOC and LOS.

After the last window in the second level, the next step is to build the file. Although the program can generate a file in Verilog or VHDL, there are also two possibilities for the file in Verilog. It has to be chosen at the beginning if the final file should integrate modules or not. If the file does not have modules in Verilog, means that the circuit is suited for the AgingCalc software tool, to compute aging and generate SPICE netlists.

The ParentBistBlock was the name chosen for the global block entity or module depending on the case. The Table 26 shows both entity and module for VHDL and Verilog files respectively with two inputs (a, b) and two outputs (z, scan_out) as example.

VHDL Type	Verilog Type
entity ParentBistBlockCuT is port(a : in std_logic; b : in std_logic; clock : in std_logic; reset : in std_logic; z : out std_logic; scan_out : out std_logic; BistStart : in std_logic; BistDone : out std_logic; MisrOut : out std_logic); end ParentBistBlockCuT; . . .	module ParentBistBlockCuT (aPBisTB, bPBisTB, clock, resetPBisTB, zPBisTB, scan_outPBisTB, BistStart, BistDone, MisrOut); . . .

Table 26: VHDL vs Verilog Entity

The main difference between the CUT and the generated ParentBistBlock entities is that there are tree new lines; BistStart, BistDone, MisrOut. These new control lines are essential to the process of BIST based on scan. The BistStart is the line to start the test process, the MisrOut line receives serially the derived signatures from the test patterns and when BistDone became logic '1' the test is completed.

If the file is VHDL type, it integrates components and therefore it is possible to maintain the same names, due to the hierarchy. However, the Verilog type that doesn't use components, must redefine new names for the new circuit primary inputs/outputs. The Table 27, in the left side shows a slice of code where it's presented a CUT component, the respective port map (block connection code) and a signal connection (also part of the connection block), but it can be observed in the right side of the table that there is no CUT component and if the connection code invoke 'z' and 'scan_out' outputs instead of 'zPBistB' and 'scan_outPBistB' to connect with the 'CutOutMisrIn' signal through an assign command, the established connection would be made between the vector signal and the CUT, leading to a undesired connection. This is the reason for different input and output names for VHDL and Verilog files.

VHDL	Verilog

signal CutOutMisrIn : std_logic_vector(1 downto 0);
 	
component CuTRelogio
port(a : in std_logic;
b : in std_logic;
teste_semm : in std_logic;
teste_simm : in std_logic;
relogio : in std_logic;
reiniciar : in std_logic;
z : out std_logic;
scan_out : out std_logic);
end component;
 	
U4 : CuTRelogio
port map (a => MuxOutCutIn(0),
b => MuxOutCutIn(1),
teste_semm => TestSelectEnable,
teste_simm => TestSerialInput,
relogio => clock,
reiniciar => ResetCutInControllerOut,
z => CutOutMisrIn(0),
scan_out => CutOutMisrIn(1));
 	
z <= CutOutMisrIn(0);
scan_out <= CutOutMisrIn(1);
	wire [1:0] CutOutMisrIn;
	
	CuTRelogio U4 (
	.a(MuxOutCutIn[0]),
	.b(MuxOutCutIn[1]),
	.teste_semm(TestSelectEnable),
	.teste_simm(TestSerialInput),
	.relogio(clock),
	.reiniciar(ResetCutInControllerOut),
	.z(CutOutMisrIn[0]),
	.scan_out(CutOutMisrIn[1]));
	
	assign CutOutMisrIn = {zPBistB, scan_outPBistB};

Table 27: Inputs and Outputs different Names

5. AGING SENSOR METHODOLOGY

This chapter will present the aging sensor methodology for circuits with BIST. The methodology is based on reusing on-chip variable power-supply voltages to perform a discrete set of BIST sessions, each using a different power-supply voltage value, to define a set of BIST signatures, which include the correct BIST signature and incorrect ones. However, these set of BIST signatures, called in this work as Voltage Signature Collection (VSC), provide a footprint for circuit's timing behaviour and its analysis can give us information on how the circuit is aging.

5.1 BACKGROUND AND PREVIOUS WORK

The idea of using a variable V_{DD} to allow performing a set of BIST sections, each one with a different V_{DD} value, to detect delay-faults was firstly introduced in [78]. The purpose of the research work was to define a new methodology to detect delay-faults not only in production but also during on-field operation. It was shown, in a limited way and for small circuits, that some delay-faults could be detected with a discrete set of BIST sessions using different power-supply voltage values in the DVS structure. The purpose was to show that not only the gross delay defects could be detected, but also some small delay defects.

However, this work lacked in two aspects: (1) the circuits under test were very small and simple; and (2) Monte Carlo simulations were not performed, to study circuit behaviour and methodology applicability under process variations. In fact, in [76] a more thorough study was performed and it was shown that in bigger circuits with BIST, and considering process variations and using Monte Carlo simulations, some results obtained in the previous work could not be reproduced, i.e., the methodology is suited to detect gross delay defects, but small delay defects can not be identified for each sample. In this work, the VSC was defined and generated for a discrete set of BIST sessions, each one at a different V_{DD} [76]. It was also shown that the presence of a resistive open alters the sequence of BIST signatures in the VSC, for

a single sample. However, each sample has a unique VSC and process variations alters the VSC, namely the BIST signatures when V_{DD} is reduced, i.e., the faulty BIST signatures of the VSC [76].

In Figure 32 it is shown the simulation result, as described in [76], for two samples of the XTRAN circuit (a fleet management system from Tecmic [79]) implemented with BIST structures to allow self-test. In this result we can see that just for these two samples, a different VSC (composed by a BIST signature for each discrete V_{DD}) is obtained in each sample. Only the BIST signatures obtained at higher V_{DD} values (the fault-free signatures) match, for few specific samples / V_{DD} values, and when V_{DD} is reduced the signatures differ [76].

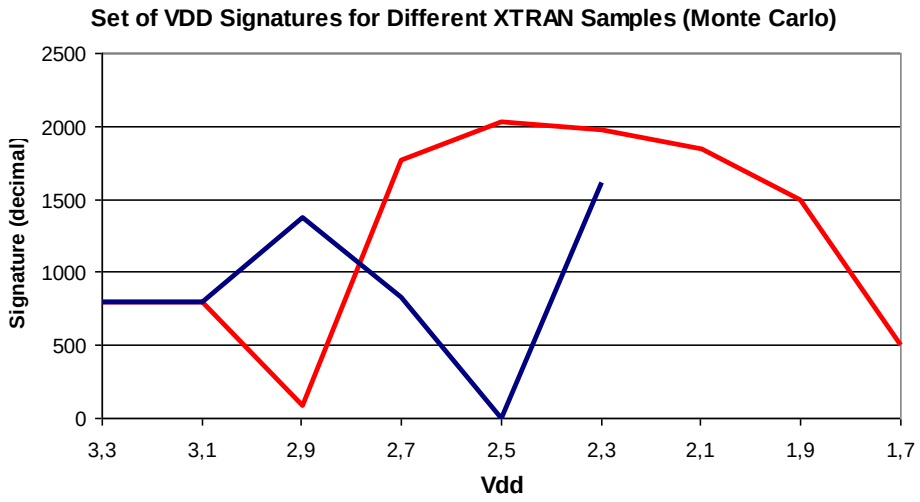


Figure 32: Set of signatures of the XTRAN circuit for two different samples (Monte Carlo analysis), as a function of V_{DD} (1.8 ; 3.3) V [76].

As explained in [76], this indicates that, for this circuit, it is not possible to define a unique set of faulty signatures for all the copies of the design, i.e., a single VSC (Voltage Signature Collection). In fact, it is predictable that only a very low complexity circuit or a very specific circuit topology may allow the use of a unique set of faulty signatures to detect non-critical delay-faults, for all the copies of the design [76]. Nevertheless, the BIST signatures for the fault-free operation are the same in all samples. This means that gross-delay defects are still possible to detect with this method and that small delay defects (delay-faults in non-critical paths) are not possible to detect during production stage, as a unique VSC is not possible to obtain for all samples (as mentioned).

Nevertheless, if lifetime test is crucial (e.g., safety-critical applications), or if aging effects need to be evaluated during circuit's lifetime (the objective of the present M.Sc. thesis), the unique set of faulty signatures of each copy may be used to identify delay defects and characterize the aging process of each unique sample. This assumption opens new perspectives and reveals that a thorough analysis for this aging characterization process may be performed. The purpose of the present work is to prove this assumption and, by collecting the VSC during circuit aging degradation, to identify the impact of such degradation in the circuit operation.

5.2 AGING SENSOR METHODOLOGY FOR SCAN-BASED BIST CIRCUITS

For sequential CUTs, the top-level diagram of the proposed multi-V_{DD} self-test scheme is shown in Figure 33. The underlying idea is to perform a discrete set of BIST sessions for a corresponding discrete set of V_{DD} values, using the BIST methodology described in chapter 4, and using always the nominal clock frequency, $f_{clk}=f_{max}$ (at-speed testing). We assume a DVS operation can be performed, without clock frequency scaling.

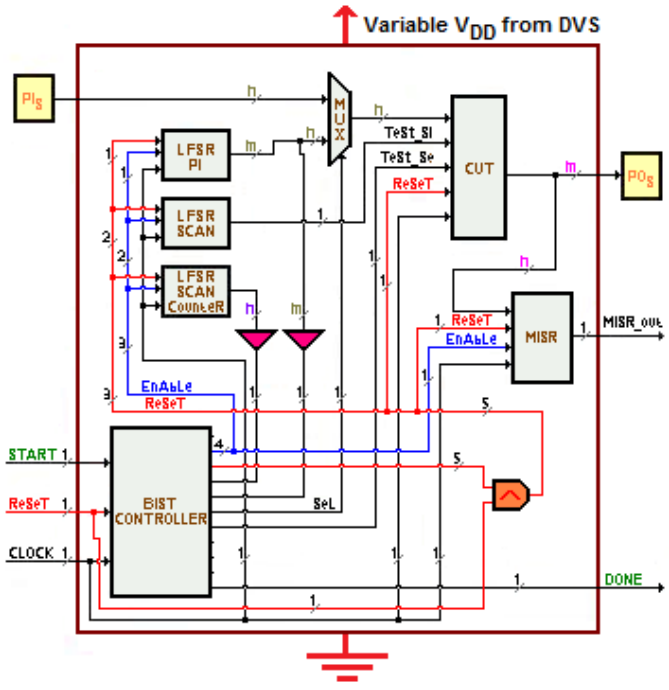


Figure 33: Top diagram of the multi-V_{DD} self-test scheme.

Power supply variations (like temperature variations) modify the time response of the CUT, of the clock distribution network and of the BIST infrastructure [76]. Basically, the following effects can be observed. First, what we refer as the accordion effect, i.e., the time response stretching of the combinational logic. If this stretching exceeds the time slack, a performance error occurs, and a faulty signature is captured in the MISR. Hence, a de-synchronization effect occurs [76]. In fact, the logic values (output response of the CUT) are captured too early, prior to the time instant in which the complete switching of the CUT network occurs. Finally, note that the BIST infrastructure is also powered by V_{DD}. Hence, it may also fail, as far as performing its functionality at lower power supply voltage levels. This last effect will also lead to corrupted signatures, eventually with a fault-free CUT.

For a given technology, design, temperature and set of BIST sessions, each sample of a fault-free device will generate a set of S_i characteristic digital signatures (one for each V_{DD} value), compacted by the MISR as the result of applying n_T test vectors to the CUT, producing the golden VSC (Voltage Signature Collection). In general, VSC is a set of (V_{DDi}, S_i) pairs of values. Temperature variations can shift these digital words along V_{DD} values [76]. Typically, higher temperature shifts the signatures towards lower V_{DD} values [76]. In the presence of aging degradations, some paths will modify their timing response and, as different paths may age differently, the result is a modification in the timing response of the CUT, and the VSC is also modified, allowing the detection of aging degradations in the CUT. This underlying principle of the proposed methodology has been verified by simulation. Results are presented in chapter 6. As stated, we assume that pseudo-random test patterns, generated by the LFSR (Linear Feedback Shift Register), with a sufficiently large number of $n_T \leq (2^n\text{-}1)$ test vectors, are able to uncover the delay-faults caused by aging, which is not necessarily so. But the use of BIST procedures targeting delay-faults, as the one described in chapter 4, increases the delay-fault coverage.

Power consumption is another critical issue. During the at-speed self-test session, it can be much higher than in the normal operation [76]. This is an important issue, as in traditional scan path focusing delay-faults (LOC and LOS), much of the test process operates at low speed, and the test vectors sequences, generated to uncover delay-faults, are applied at-speed [76]. In our proposed solution, as the scan-based BIST for

delay-faults is operating at-speed, we expect the power consumption to increase. Test power can be limited by reducing the test units within test sessions or reducing the clock frequency [75]. However, in this case clock frequency reduction is not an option, because we want to perform all tests at nominal clock frequency, to uncover delay-faults. Nevertheless, in the proposed multi-V_{DD} dynamic BIST methodology, power consumption is reduced when running BIST at depleted V_{DD} values [76]. Moreover, as this is a test-per-scan architecture, the energy and power consumption may be reduced by toggle suppression, as proposed in [80].

5.3 AGING ANALYSIS AND CIRCUIT'S DEGRADATION WITH AGING

In order to validate the Aging Sensor Methodology proposed in previous section 5.2, an aging analysis must be made to predict how circuit will age and to implement in circuit's SPICE netlist the necessary modifications to allow simulation of the aged circuit. This task is performed with the AgingCalc software tool.

AgingCalc was designed to analyze and predict digital circuit's aging induced by NBTI. Agingcalc development started in 2010 at University of Algarve as part of Jackson Pachito's M.Sc thesis [77], with the support of Prof. Jorge Semião, was released in 2011 and is currently under continuum development by the former.

This program evaluate how individual transistors threshold voltages are affected with time, based on the operation probability of each individual PMOS transistor, calculates circuit's path delays, and find which FFs are critical memory elements (i.e., those where combinational critical paths end), and generates SPICE netlists for different aging moments in time. This is a key procedure to obtain a set of VSC, one for each aging year of degradation considered.

As it will be shown in chapter 6, the simulation results will produce a three dimension graph, calculating BIST signatures for different V_{DD} and aging variations. Moreover, the evolution of the VSC with aging allows to determine not only aging variations in the CUT, but also in the BIST circuitry. However, the information that can be gathered from the set of VSC will differ from one circuit to another, depending on circuit architecture and functionality, as will be shown.

6. RESULTS

In this chapter, the simulation results will be presented, to verify: (1) the correctness of the BIST infrastructure developed and the BISTGen software tool operation; (2) the effectiveness of the Aging Sensor Methodology for BIST circuits. To allow these two analysis, simulations and implementations have been carried out in HSpice, CosmosScope and AgingCalc environments, for using SPICE and Verilog circuit netlists, and in ModelSim and WaveEditor environment for VHDL behavioural file descriptions.

The first section will present the test procedures and environments used. The second section will present the results for the BIST infrastructure and BISTGen software tool, whereas the third section will present the results for the Aging Sensor Methodology for BIST circuits.

6.1 SIMULATION ENVIRONMENT AND TEST PROCEDURES

6.1.1 VHDL SIMULATION PROCEDURE

VHDL simulation process is explained in the following. First, the new BIST circuitry is inserted in a given circuit (CUT), which is achieved by the BISTGen software, and this is done in VHDL by opening a VHDL type for VHDL CUT files. Next, the ModelSim software, developed by Altera Corporation, performs the VHDL file's simulation and allows also the graphic view (through the ModelSim Wave editor) of all digital waveforms related with buses and nodes in the circuit. Figure 34 shows the steps of a VHDL simulation file.

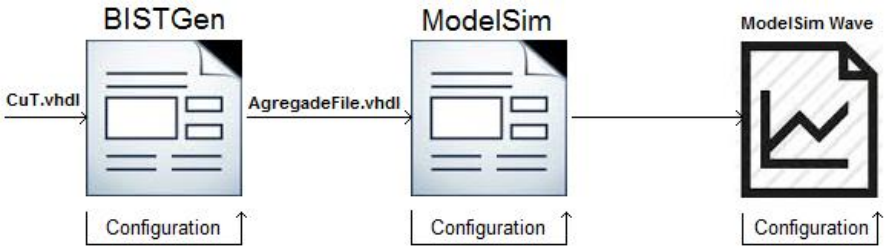


Figure 34: VHDL Simulation Steps

The possibility of Verilog and VHDL simulations also clarifies the reliability process of the BISTGen implementation, supplying two ways of simulation for the same circuit (CUT) that must be described in both languages for the effect.

6.1.2 VERILOG, AGINGCALC, AND SPICE SIMULATION PROCEDURE

The simulations carried out in Verilog files require a set of stages and configurations necessary to obtain graphical results for analyzing aging over the years from a given circuit. Figure 35 shows the necessary steps.

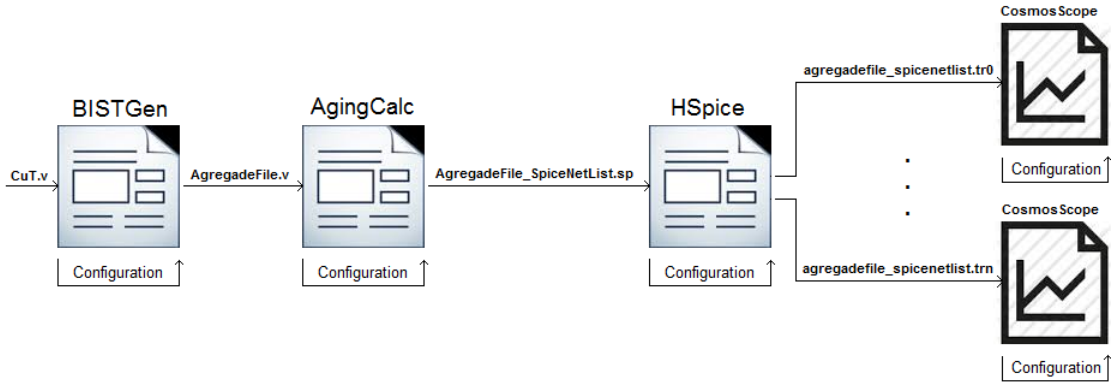


Figure 35: Verilog, AgingCalc and HSpice simulation steps.

First, the new BIST technology is inserted for a given circuit (CUT) which is achieved by the BISTGen software. After that, the AgingCalc tool has the capability of converting a Verilog hardware description file (.v) in its equivalent SPICE netlist for HSpice (.sp type file) simulation, after adding additional aging calculations for a

specific number of years supplied by the user. In fact, AgingCalc instantiates HSPICE to allow automatic transistor level simulations of SPICE netlists, automatically created by the software. These simulations may give a first analysis on circuit's path delays.

In a final stage, AgingCalc exports to a SPICE netlist the circuit at transistor level, mapped to a generic CMOS library. This netlist includes the aging analysis previously performed on AgingCalc, introducing on each PMOS transistor the aging degradation through Vth modulation.

The obtained netlist can then be simulated in HSpice, a software tool developed by Synopsys. The simulation results, which include one set of simulations for each year of aging degradation considered, can be observed with the CosmosScope software, a Synopsys tool also.

Moreover, using delay measurements available in the HSPICE SPICE distribution, it is possible to obtain the final BIST signatures for each BIST session (simulation). With the aid of a graphic suit, like Excel from Microsoft, it is possible to plot the set of VSCs for the period of aging analysis in a 3-D graph, so that the BIST signatures analysis can be straight forward procedure, just by simple inspection.

It is important to mention, that for all SPICE simulations a 65nm CMOS technology is used, with a nominal V_{DD} of 1.1V.

6.2 RESULTS FOR BIST CIRCUITRY AND BISTGEN TOOL

This section will present results for the BISTGen tool, by generating automatically the BIST structures for four test circuits. For all the CUTs the Verilog and VHDL type descriptions will be presented.

The validation of the BIST circuitry will be done in this section by logical simulation of the VHDL type files. However, only for CUT_example circuit the VHDL behavioural description with scan path is available, so only for this CUT will be performed the logic simulation using ModelSim environment. For the remainig CUTs, only the structural gate level Verilog description is available, and their

simulation will be done in HSPICE environment and it will be presented in section 6.3.

6.2.1 CUT_EXAMPLE CIRCUIT

The CUT_example circuit is a simple sequential circuit used to demonstrate and validate the BIST circuitry functionality. It's a 5 gate circuit, with 2 FFs and 3 combinational logic gates (see Figure 36).

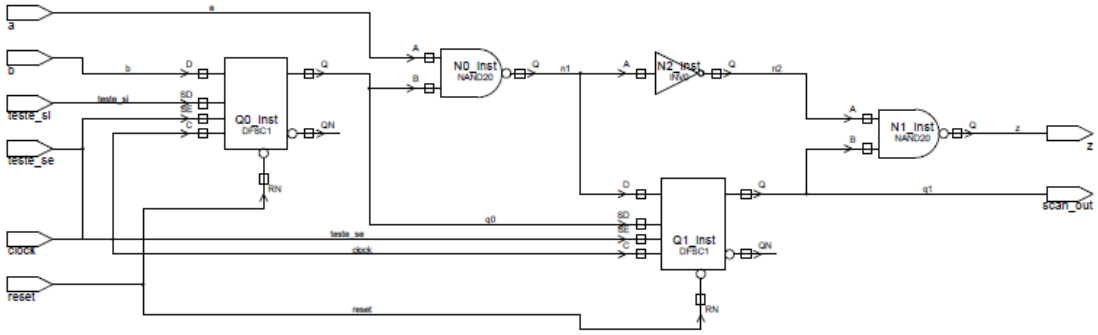


Figure 36: CUT_example circuit schematic.

The referred circuit is presented through its hardware description code for VHDL and Verilog environments, in the Table 28.

CUT VHDL

```
use IEEE.std_logic_1164.all;
entity CuT is
  port( a, b, teste_se, teste_si, clock, reset : in std_logic;
              z, scan_out : out std_logic);
end CuT;
architecture Comportamento of CuT is
signal Qout, Qin, Qin_data, Qin_test: std_logic_vector(1 downto 0);
begin
 sinc: process(clock,reset)
   begin
     if reset = '0' then
       Qout <= "00";
     elsif clock'event and clock = '1' then
       Qout <= Qin;
     end if;
   end process;
Qin_data(0) <= b;
Qin_test(0) <= teste_si;
Qin_test(1) <= Qout(0);
scan_out <= Qout(1);
 comb_mux:
process(clock,teste_se,Qin_data,Qin_test,Qout,Qin,a,b)
   begin
     Qin_data(1) <= not( Qout(0) and a );
     z <= not( not( Qin_data(1) ) and Qout(1) );
```

CUT Verilog

```
module CuT ( a, b, teste_se, teste_si, clock, reset, z, scan_out );
 input a, b, teste_se, teste_si, clock, reset;
 output z, scan_out;
 wire  q0, q1, n1, n2;

 DFSC1 Q0ScanFlipFlop ( .D(b), .SD(teste_si), .SE(teste_se), .C(clock),
.RN(reset), .Q(q0) );
 DFSC1 Q1ScanFlipFlop ( .D(n1), .SD(q0), .SE(teste_se), .C(clock), .RN(reset),
.Q(q1) );
 NAND20 N0NanDScan ( .A(a), .B(q0), .Q(n1) );
 NAND20 N1NanDScan ( .A(n2), .B(q1), .Q(z) );
 INV0 N2InVScan ( .A(n1), .Q(n2) );

 assign scan_out = q1;

endmodule
```

```
      if teste_se = '0' then
        Qin <= Qin_data;
      else
        Qin <= Qin_test;
      end if;
 end process;
end comportamento;
```

Table 28: Generic CUT Hardware Description either VHDL or Verilog

Using BISTGen software tool, the BIST circuitry and functionality was inserted and the LFSR seeds presented in Table 29 were used. In the following, circuit descriptions are presented in Table 29, for VHDL and Verilog LOS based BIST, and they were generated automatically through BISTGen as presented in Table 30 and Table 31, respectively. For simplicity, only the ParentBistBlock is presented, which is the main block that connects the CUT with the BIST blocks.

Block	LFSR type	Seed
LFSR PI	Linear	0110
LFSR Scan	Modular	0110
LFSR Scan Counter	Linear	01
MISR	Linear	0110

Table 29: Config features for VHDL and Verilog CUT File

Aggregate BIST file generated by BISTGen [VHDL]

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
-- -- -- PARENTBISTBLOCK
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

library IEEE;
use IEEE.std_logic_1164.all;

entity ParentBistBlockCuT is
   port( a : in std_logic;
  b : in std_logic;
  clock : in std_logic;
  reset : in std_logic;
  z : out std_logic;
  scan_out : out std_logic;
  BistStart : in std_logic;
  BistDone : out std_logic;
  MisrOut  : out std_logic);
end ParentBistBlockCuT;

architecture SYN_SYN_BEHAV of ParentBistBlockCuT is

component LfsrPiCuT
port( clock : in std_logic;
      reset : in std_logic;
      enable : in std_logic;
      DataOut : out std_logic_vector(3 downto 0) );
end component;

component LfsrSCANCuT
port( clock : in std_logic;
      reset : in std_logic;
      enable : in std_logic;
      DataOut : out std_logic);
end component;
```

```
component LfsrScanCounterCuT
port( clock : in std_logic;
      reset : in std_logic;
      enable : in std_logic;
      DataOut : out std_logic_vector(1 downto 0) );
end component;

component CuT
port( a : in std_logic;
      b : in std_logic;
      teste_se : in std_logic;
      teste_si : in std_logic;
      clock : in std_logic;
      reset : in std_logic;
      z : out std_logic;
      scan_out : out std_logic);
end component;

component LfsrMisrCuT
port( DataIn : in std_logic_vector(1 downto 0);
      clock : in std_logic;
      reset : in std_logic;
      enable : in std_logic;
      DataOut : out std_logic);
end component;

component MuXCuT
port( SeL : in std_logic;
      InA : in std_logic_vector(1 downto 0);
      InB : in std_logic_vector(1 downto 0);
      DataOut : out std_logic_vector(1 downto 0));
end component;

component BistControllerCuT
port ( BistStart : in std_logic;
      Clock : in std_logic;
      ResetController : in std_logic;
      ResetLfsrPi : out std_logic;
      ResetLfsrScan : out std_logic;
      ResetLfsrScanCounter : out std_logic;
      ResetCut : out std_logic;
      ResetMisr : out std_logic;
      LfsrPiCountFinished : in std_logic;
      LfsrScanCountFinished : in std_logic;
      EnableLfsrPi : out std_logic;
      EnableLfsrScan : out std_logic;
      EnableLfsrScanCounter : out std_logic;
      EnableMisr : out std_logic;
      TestSE : out std_logic;
      MuxSelect : out std_logic;
      BistDone : out std_logic);
end component;

signal MuxOutCutIn : std_logic_vector(1 downto 0);
signal MuxInParentBistBlockIn : std_logic_vector(1 downto 0);
signal MuxSelectInControllerOut : std_logic;
signal CutOutMisrIn : std_logic_vector(1 downto 0);
signal LfsrPiOut : std_logic_vector(3 downto 0);
signal LfsrScanCounterOut : std_logic_vector(1 downto 0);
signal ResetCutInControllerOut : std_logic;
signal ResetLfsrPiInControllerOut : std_logic;
signal ResetLfsrScanInControllerOut : std_logic;
signal ResetLfsrScanCounterInControllerOut : std_logic;
signal ResetMisrInControllerOut : std_logic;
signal AndInControllerOutPiReset : std_logic;
signal AndInControllerOutScanReset : std_logic;
signal AndInControllerOutScanCounterReset : std_logic;
signal AndInControllerOutCutReset : std_logic;
signal AndInControllerOutMisrReset : std_logic;
signal BistCountFinishedOut : std_logic;
signal ScanCountFinishedOut : std_logic;
signal EnableLfsrPiInControllerOut : std_logic;
signal EnableLfsrScanInControllerOut : std_logic;
signal EnableLfsrScanCounterInControllerOut : std_logic;
signal EnableMisrInControllerOut : std_logic;
signal TestSerialInput : std_logic;
signal TestSelectEnable : std_logic;

begin

ResetAll: process(reset, AndInControllerOutPiReset, AndInControllerOutScanReset, AndInControllerOutScanCounterReset, AndInControllerOutCutReset,
AndInControllerOutMisrReset)
begin
 ResetLfsrPiInControllerOut <= reset and AndInControllerOutPiReset;
 ResetLfsrScanInControllerOut <= reset and AndInControllerOutScanReset;
 ResetLfsrScanCounterInControllerOut <= reset and AndInControllerOutScanCounterReset;
 ResetCutInControllerOut <= reset and AndInControllerOutCutReset;
 ResetMisrInControllerOut <= reset and AndInControllerOutMisrReset;
end process;
```

```
BistCountEnd: process(LfsrPiOut)
begin
 if LfsrPiOut = "0110" then
  BistCountFinishedOut <= '1';
 else
  BistCountFinishedOut <= '0';
 end if;
end process;

ScanCountEnd: process(LfsrScanCounterOut)
begin
 if LfsrScanCounterOut = "10" then
  ScanCountFinishedOut <= '1';
 else
  ScanCountFinishedOut <= '0';
 end if;
end process;

MuxInParentBistBlockIn(0) <= a;
MuxInParentBistBlockIn(1) <= b;

z <= CutOutMisrIn(0);
scan_out <= CutOutMisrIn(1);

U1 : LfsrPiCuT
port map ( clock =>
     reset =>
     enable =>
     DataOut =>

U2 : LfsrSCANCuT
port map ( clock =>  clock,
     reset => ResetLfsrScanInControllerOut,
     enable => EnableLfsrScanInControllerOut,
     DataOut => TestSerialInput);

U3 : LfsrScanCounterCuT
port map ( clock =>  clock,
     reset => ResetLfsrScanCounterInControllerOut,
     enable => EnableLfsrScanCounterInControllerOut,
     DataOut => LfsrScanCounterOut);

U4 : CuT
port map ( a =>  MuxOutCutIn(0),
     b => MuxOutCutIn(1),
     teste_se => TestSelectEnable,
     teste_si => TestSerialInput,
     clock => clock,
     reset => ResetCutInControllerOut,
     z => CutOutMisrIn(0),
     scan_out => CutOutMisrIn(1));

U5 : LfsrMisrCuT
port map ( DataIn =>  CutOutMisrIn,
     clock => clock,
     reset => ResetMisrInControllerOut,
     enable => EnableMisrInControllerOut,
     DataOut => MisrOut);

U6 : MuXCuT
port map ( SeL =>  MuxSelectInControllerOut,
     InA => MuxInParentBistBlockIn,
     InB => LfsrPiOut(1 downto 0),
     DataOut => MuxOutCutIn);

U7 : BistControllerCuT
port map ( BistStart =>  BistStart,
     Clock => clock,
     ResetController => reset,
     ResetLfsrPi => AndInControllerOutPiReset,
     ResetLfsrScan => AndInControllerOutScanReset,
     ResetLfsrScanCounter => AndInControllerOutScanCounterReset,
     ResetCut => AndInControllerOutCutReset,
     ResetMisr => AndInControllerOutMisrReset,
     LfsrPiCountFinished => BistCountFinishedOut,
     LfsrScanCountFinished => ScanCountFinishedOut,
     EnableLfsrPi => EnableLfsrPiInControllerOut,
     EnableLfsrScan => EnableLfsrScanInControllerOut,
     EnableLfsrScanCounter => EnableLfsrScanCounterInControllerOut,
     EnableMisr => EnableMisrInControllerOut,
     TestSE => TestSelectEnable,
     MuxSelect => MuxSelectInControllerOut,
     BistDone => BistDone);

end SYN_SYN_BEHAV;
```

Table 30: Main module from VHDL LOS based BIST Aggregate File

Aggregate BIST file generated by BISTGen [Verilog]

```
module ParentBistBlockCuT (
  aPBisTB,
  bPBisTB,
  clockPBisTB,
  resetPBisTB,
  zPBisTB,
  scan_outPBisTB,
  BistStart,
  BistDone,
  MisrOut );

  input aPBisTB;
  input bPBisTB;
  input clockPBisTB;
  input resetPBisTB;
  output zPBisTB;
  output scan_outPBisTB;
  input BistStart;
  output BistDone;
  output MisrOut;

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// -- -- -- PI
// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

   wire resetLfsrPICuT;
   wire enableLfsrPICuT;
   wire [3:0] DataOutLfsrPICuT ;

   wire [3:0] QoutLfsrPICuT ;
   wire y1LfsrPICuT;

   DFEC1 U0LfsrPICuT ( .D(QoutLfsrPICuT[1]), .E(enableLfsrPICuT), .C(clock), .RN(resetLfsrPICuT), .Q(QoutLfsrPICuT[0]) );
   DFEP1 U1LfsrPICuT ( .D(QoutLfsrPICuT[2]), .E(enableLfsrPICuT), .C(clock), .SN(resetLfsrPICuT), .Q(QoutLfsrPICuT[1]) );
   DFEP1 U2LfsrPICuT ( .D(QoutLfsrPICuT[3]), .E(enableLfsrPICuT), .C(clock), .SN(resetLfsrPICuT), .Q(QoutLfsrPICuT[2]) );
   DFEC1 U3LfsrPICuT ( .D(y1LfsrPICuT), .E(enableLfsrPICuT), .C(clock), .RN(resetLfsrPICuT), .Q(QoutLfsrPICuT[3]) );

   XOR20  U4LfsrPICuT ( .A(QoutLfsrPICuT[0]), .B(QoutLfsrPICuT[1]), .Q(y1LfsrPICuT);

   assign DataOutLfsrPICuT = QoutLfsrPICuT;

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// -- -- -- SCAN
// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

   wire resetLfsrSCANCuT;
   wire enableLfsrSCANCuT;
   wire DataOutLfsrSCANCuT ;

   wire [3:0] QoutLfsrSCANCuT ;
   wire y1LfsrSCANCuT;

   DFEC1 U0LfsrSCANCuT ( .D(QoutLfsrSCANCuT[3]), .E(enableLfsrSCANCuT), .C(clock), .RN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[0]) );
   DFEP1 U1LfsrSCANCuT ( .D(y1LfsrSCANCuT), .E(enableLfsrSCANCuT), .C(clock), .SN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[1]) );
   XOR20  U4LfsrSCANCuT ( .A(QoutLfsrSCANCuT[0]), .B(QoutLfsrSCANCuT[3]), .Q(y1LfsrSCANCuT));
   DFEP1 U2LfsrSCANCuT ( .D(QoutLfsrSCANCuT[1]), .E(enableLfsrSCANCuT), .C(clock), .SN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[2]) );
   DFEC1 U3LfsrSCANCuT ( .D(QoutLfsrSCANCuT[2]), .E(enableLfsrSCANCuT), .C(clock), .RN(resetLfsrSCANCuT), .Q(QoutLfsrSCANCuT[3]) );

   assign DataOutLfsrSCANCuT = QoutLfsrSCANCuT[3];

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// -- -- -- SCANCOUNTER
// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

   wire resetLfsrScanCounterCuT;
   wire enableLfsrScanCounterCuT;
   wire [1:0] DataOutLfsrScanCounterCuT ;

   wire [1:0] QoutLfsrScanCounterCuT ;
   wire y1LfsrScanCounterCuT;

   DFEP1  U0LfsrScanCounterCuT  (  .D(QoutLfsrScanCounterCuT[1]),  .E(enableLfsrScanCounterCuT),  .C(clock),  .SN(resetLfsrScanCounterCuT),
.Q(QoutLfsrScanCounterCuT[0]) );
   DFEP1  U1LfsrScanCounterCuT  (  .D(y1LfsrScanCounterCuT),  .E(enableLfsrScanCounterCuT),  .C(clock),  .SN(resetLfsrScanCounterCuT),
.Q(QoutLfsrScanCounterCuT[1]) );

   XOR20  U2LfsrScanCounterCuT ( .A(QoutLfsrScanCounterCuT[0]), .B(QoutLfsrScanCounterCuT[1]), .Q(y1LfsrScanCounterCuT);

   assign DataOutLfsrScanCounterCuT = QoutLfsrScanCounterCuT;

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// -- -- -- CUT
// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

  wire a;
  wire b;
  wire teste_se;
  wire teste_si;
  wire clock;
  wire reset;
  wire z;
  wire scan_out;
```

```
  wire   q0, q1, n1, n2;

  DFSC1 Q0_inst ( .D(b), .SD(teste_si), .SE(teste_se), .C(clock), .RN(reset), .Q(q0) );
  DFSC1 Q1_inst ( .D(n1), .SD(q0), .SE(teste_se), .C(clock), .RN(reset), .Q(q1) );
  NAND20 N0_inst ( .A(a), .B(q0), .Q(n1) );
  NAND20 N1_inst ( .A(n2), .B(q1), .Q(z) );
  INV0 N2_inst ( .A(n1), .Q(n2) );

  assign scan_out = q1;

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// -- -- -- MISR
// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

  wire resetLfsrMisrCuT;
  wire enableLfsrMisrCuT;
  wire [1:0] InputSLfsrMisrCuT ;
  wire DataOutLfsrMisrCuT;

  wire [3:0] QoutLfsrMisrCuT ;
  wire y1LfsrMisrCuT, x1LfsrMisrCuT, x2LfsrMisrCuT;

  DFEC1 U0LfsrMisrCuT ( .D(x1LfsrMisrCuT), .E(enableLfsrMisrCuT), .C(clock), .RN(resetLfsrMisrCuT), .Q(QoutLfsrMisrCuT[0]) );
  XOR20  U5LfsrMisrCuT ( .A(QoutLfsrMisrCuT[1]), .B(InputSLfsrMisrCuT[0]), .Q(x1LfsrMisrCuT));
  DFEP1 U1LfsrMisrCuT ( .D(x2LfsrMisrCuT), .E(enableLfsrMisrCuT), .C(clock), .SN(resetLfsrMisrCuT), .Q(QoutLfsrMisrCuT[1]) );
  XOR20  U6LfsrMisrCuT ( .A(QoutLfsrMisrCuT[2]), .B(InputSLfsrMisrCuT[1]), .Q(x2LfsrMisrCuT));
  DFEP1 U2LfsrMisrCuT ( .D(QoutLfsrMisrCuT[3]), .E(enableLfsrMisrCuT), .C(clock), .SN(resetLfsrMisrCuT), .Q(QoutLfsrMisrCuT[2]) );
  DFEC1 U3LfsrMisrCuT ( .D(y1LfsrMisrCuT), .E(enableLfsrMisrCuT), .C(clock), .RN(resetLfsrMisrCuT), .Q(QoutLfsrMisrCuT[3]) );

  XOR20  U4LfsrMisrCuT ( .A(QoutLfsrMisrCuT[0]), .B(QoutLfsrMisrCuT[1]), .Q(y1LfsrMisrCuT) );

  assign DataOutLfsrMisrCuT = QoutLfsrMisrCuT[0];

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// -- -- -- MUX
// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

  wire SeLMuXCuT;
  wire [1:0] InAMuXCuT;
  wire [1:0] InBMuXCuT;
  wire [1:0] DataOutMuXCuT;

  MUX21 U1MuXCuT ( .A(InAMuXCuT[0]), .B(InBMuXCuT[0]), .S(SeLMuXCuT), .Q(DataOutMuXCuT[0]) );
  MUX21 U2MuXCuT ( .A(InAMuXCuT[1]), .B(InBMuXCuT[1]), .S(SeLMuXCuT), .Q(DataOutMuXCuT[1]) );

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// -- -- -- CONTROLLER
// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

  wire BistStart;
  wire Clock;
  wire ResetController;
  wire LfsrPiCountFinished;
  wire LfsrScanCountFinished;
  wire ResetLfsrPi;
  wire ResetLfsrScan;
  wire ResetLfsrScanCounter;
  wire ResetCut;
  wire ResetMisr;
  wire EnableLfsrPi;
  wire EnableLfsrScan;
  wire EnableLfsrScanCounter;
  wire EnableMisr;
  wire TestSE;
  wire MuxSelect;
  wire BistDone;

// -- L . O . S

  wire LonSn1;
  wire LonSn3;
  wire LonSn4;
  wire LonSn5;
  wire LonSn6;
  wire LonSn7;
  wire LonSn8;
  wire LonSn9;
  wire LonSn11;
  wire LonSn12;
  wire LonSn13;
  wire LonSn14;
  wire LonSn15;
  wire LonSn16;
  wire LonSn17;
  wire LonSn18;
  wire LonSn19;
  wire LonSn20;
  wire [2:0] estado;
  wire [2:0] estado_seguinte;

  assign ResetLfsrPi = ResetMisr;
```

```
    assign ResetLfsrScan = ResetMisr;
    assign EnableLfsrScan = TestSE;
    assign EnableLfsrPi = estado_seguinte[2];

    DFC3 \estado_reg[0]  ( .D(estado_seguinte[0]), .C(Clock), .RN(ResetController), .Q(estado[0]) );
    DFC1 \estado_reg[1]  ( .D(estado_seguinte[1]), .C(Clock), .RN(ResetController), .Q(estado[1]), .QN(LonSn1) );
    DFC3 \estado_reg[2]  ( .D(estado_seguinte[2]), .C(Clock), .RN(ResetController), .Q(estado[2]) );
    INV3 U3ControlleR ( .A(LonSn13), .Q(EnableMisr) );
    CLKIN0 U4ControlleR ( .A(LonSn3), .Q(ResetMisr) );
    NOR20 U5ControlleR ( .A(estado_seguinte[2]), .B(ResetLfsrScanCounter), .Q(LonSn3) );
    AOI2110 U6ControlleR ( .A(LonSn1), .B(LonSn4), .C(LonSn5), .D(LonSn6), .Q(ResetLfsrScanCounter) );
    CLKIN0 U7ControlleR ( .A(LonSn7), .Q(LonSn5) );
    OAI210 U8ControlleR ( .A(estado[0]), .B(estado[1]), .C(estado[2]), .Q(LonSn7) );
    CLKIN0 U9ControlleR ( .A(LonSn8), .Q(ResetCut) );
    NOR40 U10ControlleR ( .A(LonSn9), .B(LonSn6), .C(TestSE), .D(estado[2]), .Q(LonSn8) );
    CLKIN0 U11ControlleR ( .A(LonSn11), .Q(LonSn6) );
    NOR20 U12ControlleR ( .A(estado_seguinte[0]), .B(estado[1]), .Q(LonSn9) );
    CLKIN0 U13ControlleR ( .A(LonSn12), .Q(MuxSelect) );
    NOR20 U14ControlleR ( .A(EnableMisr), .B(estado_seguinte[0]), .Q(LonSn12) );
    NOR20 U15ControlleR ( .A(TestSE), .B(estado_seguinte[2]), .Q(LonSn13) );
    OAI210 U16ControlleR ( .A(BistDone), .B(LonSn14), .C(LonSn15), .Q(TestSE) );
    OAI310 U17ControlleR ( .A(LonSn14), .B(estado_seguinte[0]), .C(BistDone), .D(LonSn15), .Q(EnableLfsrScanCounter) );
    NOR30 U18ControlleR ( .A(estado_seguinte[1]), .B(estado_seguinte[2]), .C(estado_seguinte[0]), .Q(BistDone) );
    OAI310 U19ControlleR ( .A(LonSn14), .B(LfsrPiCountFinished), .C(LonSn16), .D(LonSn17), .Q(estado_seguinte[0]) );
    NAND30 U20ControlleR ( .A(LonSn4), .B(LonSn1), .C(BistStart), .Q(LonSn17) );
    NOR20 U21ControlleR ( .A(LonSn11), .B(estado[2]), .Q(estado_seguinte[2]) );
    NAND20 U22ControlleR ( .A(estado[1]), .B(estado[0]), .Q(LonSn11) );
    OAI210 U23ControlleR ( .A(LonSn18), .B(LonSn14), .C(LonSn15), .Q(estado_seguinte[1]) );
    CLKIN0 U24ControlleR ( .A(LonSn19), .Q(LonSn15) );
    AOI2110 U25ControlleR ( .A(estado[0]), .B(estado[2]), .C(LonSn4), .D(estado[1]), .Q(LonSn19));
    NAND20 U26ControlleR ( .A(LonSn4), .B(estado[1]), .Q(LonSn14) );
    NOR20 U27ControlleR ( .A(estado[0]), .B(estado[2]), .Q(LonSn4) );
    NOR20 U28ControlleR ( .A(LonSn20), .B(LonSn16), .Q(LonSn18) );
    CLKIN0 U29ControlleR ( .A(LfsrScanCountFinished), .Q(LonSn16) );
    CLKIN0 U30ControlleR ( .A(LfsrPiCountFinished), .Q(LonSn20) );

// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
// -- -- -- PARENTBISTBLOCK
// -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

  wire [1:0] MuxOutCutIn;
  wire MuxSelectInControllerOut;
  wire [1:0] CutOutMisrIn;
  wire [3:0] LfsrPiOut;
  wire [1:0] LfsrScanCounterOut;
  wire ResetLfsrPiInControllerOut;
  wire ResetLfsrScanInControllerOut;
  wire ResetLfsrScanCounterInControllerOut;
  wire ResetCutInControllerOut;
  wire ResetMisrInControllerOut;
  wire EnableLfsrPiInControllerOut;
  wire EnableLfsrScanInControllerOut;
  wire EnableLfsrScanCounterInControllerOut;
  wire EnableMisrInControllerOut;
  wire TestSerialInput;
  wire TestSelectEnable;
  wire clock;

  wire [1:0] NoTLpO;
  wire [2:0] NandOutInvInLpO;
  wire [2:0] InvOutNandInLpO;

  INV0 LpO0 (.A(LfsrPiOut[0]), .Q(NoTLpO[0]));
  INV0 LpO1 (.A(LfsrPiOut[1]), .Q(NoTLpO[1]));

  INV0 NanDnaNLpO0 (.A(NandOutInvInLpO[0]), .Q(InvOutNandInLpO[0]));
  INV0 NanDnaNLpO1 (.A(NandOutInvInLpO[1]), .Q(InvOutNandInLpO[1]));
  INV0 NanDnaNLpO2 (.A(NandOutInvInLpO[2]), .Q(InvOutNandInLpO[2]));

  NAND20 NLpO0 (.A(NoTLpO[0]), .B(NoTLpO[1]), .Q(NandOutInvInLpO[0]) );
  NAND20 NLpO1 (.A(InvOutNandInLpO[0]), .B(LfsrPiOut[2]), .Q(NandOutInvInLpO[1]) );
  NAND20 NLpO2 (.A(InvOutNandInLpO[1]), .B(LfsrPiOut[3]), .Q(NandOutInvInLpO[2]) );

  wire NoTLscO;
  wire [0:0] NandOutInvInLscO;
  wire [0:0] InvOutNandInLscO;

  INV0 LscO0 (.A(LfsrScanCounterOut[0]), .Q(NoTLscO));

  INV0 NanDnaNLscO0 (.A(NandOutInvInLscO[0]), .Q(InvOutNandInLscO[0]));

  NAND20 NLscO0 (.A(NoTLscO), .B(LfsrScanCounterOut[1]), .Q(NandOutInvInLscO[0]) );

  wire [4:0] NandOutInvInResetS;
  wire [4:0] NandInControllerOutResetS;

  INV0 INVRST1 (.A(NandOutInvInResetS[0]), .Q(ResetLfsrPiInControllerOut) );
  INV0 INVRST2 (.A(NandOutInvInResetS[1]), .Q(ResetLfsrScanInControllerOut) );
  INV0 INVRST3 (.A(NandOutInvInResetS[2]), .Q(ResetLfsrScanCounterInControllerOut) );
  INV0 INVRST4 (.A(NandOutInvInResetS[3]), .Q(ResetCutInControllerOut) );
  INV0 INVRST5 (.A(NandOutInvInResetS[4]), .Q(ResetMisrInControllerOut) );
```

```
NAND20 NDRST1 (.A(resetPBisTB), .B(NandInControllerOutResetS[0]), .Q(NandOutInvInResetS[0]) );
NAND20 NDRST2 (.A(resetPBisTB), .B(NandInControllerOutResetS[1]), .Q(NandOutInvInResetS[1]) );
NAND20 NDRST3 (.A(resetPBisTB), .B(NandInControllerOutResetS[2]), .Q(NandOutInvInResetS[2]) );
NAND20 NDRST4 (.A(resetPBisTB), .B(NandInControllerOutResetS[3]), .Q(NandOutInvInResetS[3]) );
NAND20 NDRST5 (.A(resetPBisTB), .B(NandInControllerOutResetS[4]), .Q(NandOutInvInResetS[4]) );

assign resetLfsrPICuT = ResetLfsrPiInControllerOut;
assign enableLfsrPICuT = EnableLfsrPiInControllerOut;
assign DataOutLfsrPICuT = LfsrPiOut;

assign resetLfsrSCANCuT = ResetLfsrScanInControllerOut;
assign enableLfsrSCANCuT = EnableLfsrScanInControllerOut;
assign DataOutLfsrSCANCuT = TestSerialInput;

assign resetLfsrScanCounterCuT = ResetLfsrScanCounterInControllerOut;
assign enableLfsrScanCounterCuT = EnableLfsrScanCounterInControllerOut;
assign DataOutLfsrScanCounterCuT = LfsrScanCounterOut;

assign a = MuxOutCutIn[0];
assign b = MuxOutCutIn[1];
assign teste_se = TestSelectEnable;
assign teste_si = TestSerialInput;
assign clock = clockPBisTB;
assign reset = ResetCutInControllerOut;
assign z = CutOutMisrIn[0];
assign scan_out = CutOutMisrIn[1];

assign CutOutMisrIn = {scan_outPBisTB, zPBisTB};

assign InputSLfsrMisrCuT = CutOutMisrIn;
assign resetLfsrMisrCuT = ResetMisrInControllerOut;
assign enableLfsrMisrCuT = EnableMisrInControllerOut;
assign DataOutLfsrMisrCuT = MisrOut;

assign SeLMuXCuT = MuxSelectInControllerOut;
assign InAMuXCuT = {aPBisTB, bPBisTB};
assign InBMuXCuT = LfsrPiOut[1:0];
assign DataOutMuXCuT = MuxOutCutIn;

assign Clock = clockPBisTB;
assign ResetController = resetPBisTB;
assign ResetLfsrPi = NandInControllerOutResetS[0];
assign ResetLfsrScan = NandInControllerOutResetS[1];
assign ResetLfsrScanCounter = NandInControllerOutResetS[2];
assign ResetCut = NandInControllerOutResetS[3];
assign ResetMisr = NandInControllerOutResetS[4];
assign LfsrPiCountFinished = InvOutNandInLpO[2];
assign LfsrScanCountFinished = InvOutNandInLscO[0];
assign EnableLfsrPi = EnableLfsrPiInControllerOut;
assign EnableLfsrScan = EnableLfsrScanInControllerOut;
assign EnableLfsrScanCounter = EnableLfsrScanCounterInControllerOut;
assign EnableMisr = EnableMisrInControllerOut;
assign TestSE = TestSelectEnable;
assign MuxSelect = MuxSelectInControllerOut;

endmodule
```

Table 31: Verilog LOS based BIST File

To validate the generated circuit, the VHDL type representation was simulated at logic level, using ModelSim environment. The logic level simulation is necessary, not only to validate BIST circuitry, but also to obtain the MISR final signature, known as the good signature. Such signature will allow us to identify the failing and fault-free circuits. Figure 37 presents the signals and buses obtained by logic simulation for the circuit in VHDL.

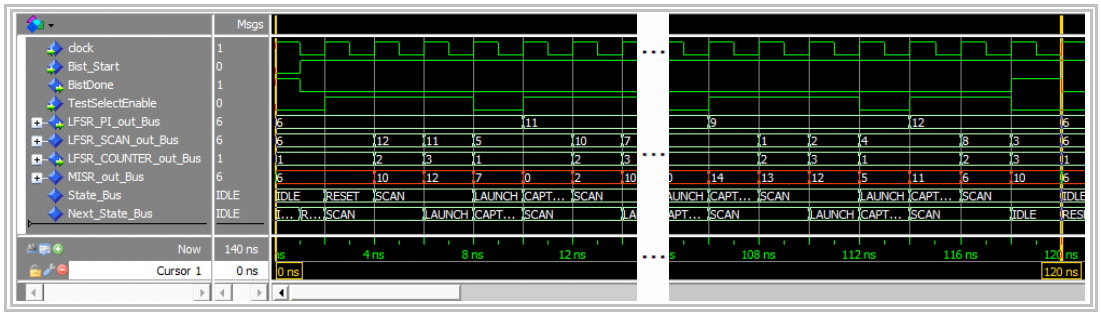


Figure 37: VHDL CUT Signature through ModelSim

The MISR correct signature is the final output of the MISR, obtained at the end of simulation. To allow an easy identification of the BIST signatures, all signatures will be represented in unsigned decimal value, and the correct signature for this circuit is number 10.

As this CUT is available in both representations, structural in Verilog and behavioural in VHDL, the BIST circuitry was also generated for the Verilog file type. After AgingCalc convert it to a SPICE netlist, the circuit was also simulated at transistor level in HSPICE environment. In this case, as the circuit and functionality is the same, if the implementation is correct, the simulation in the SPICE netlist should return the same BIST signature value, at the end of simulation.

As it can be seen in Figure 38, the MISR final signature in the SPICE simulation is also the decimal number 10. Both simulations show the same value near the 120 ns of simulation time.

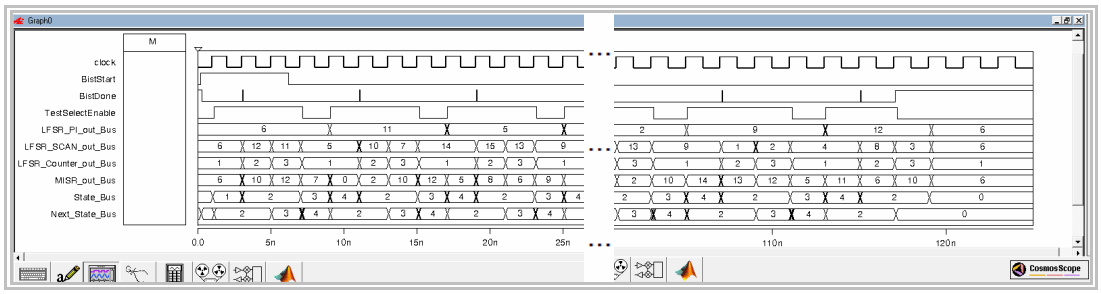


Figure 38: Verilog CUT Signature through HSpice (CosmosScope)

6.2.2 B01, B06 AND PIPELINE MULTIPLIER CIRCUITS

The following example circuits are additional test vehicles for the BISTGen software tool validation. B01 and B06 are two ITC'99 benchmark circuits and Pipeline Multiplier, as the name mentions, a 4 bit multiplier circuit with two pipeline stages. In more detail, B01 is a Finite State Machine (FSM) that compares serial flow, has 49 logic gates, 2 Primary Inputs (PI), 2 Primary Outputs (PO) and 5 FFs. B06 is an interrupt handler with 56 logic gates, 2 PI, 6 PO and 9 FFs. Finally, the Pipeline Multiplier has 4 bits input, with 2 pipeline stages, and multiplies the two 4 bit inputs and places the result at the 8-bit output. It has 52 logic gates, 10 PI, 8 PO and 36 FFs. Table 32, Table 33 and Table 34 present the LFSRs seeds used respectively in B01, B06 and Pipeline Multiplier circuits, when the BISTGen software was used to insert the BIST structures and functionality in these circuits.

Block	LFSR type	Seed
LFSR PI	Linear	0110110
LFSR Scan	Modular	0110101
LFSR Scan Counter	Linear	101
MISR	Linear	011010

Table 32: Config features for Verilog BIST B01 File

Block	LFSR type	Seed
LFSR PI	Linear	0100111
LFSR Scan	Modular	0011100
LFSR Scan Counter	Linear	0111
MISR	Linear	101010

Table 33: Config features for Verilog BIST B06 File

Block	LFSR type	Seed
LFSR PI	Linear	01101010101010
LFSR Scan	Modular	0110101010
LFSR Scan Counter	Linear	001001
MISR	Linear	01101010101

Table 34: Config features for Verilog BIST Pipeline Multiplier 4-2 File

To avoid reproducing here all the VHDL and Verilog codes for the generated circuits, the BISTGen results for these CUTs are available in the Compact Disc (CD) that accompanies this M.Sc. dissertation.

6.3 RESULTS FOR THE AGING SENSOR METHODOLOGY

This section presents the results for the Aging Sensor Methodology. Using the Verilog type netlists obtained with BISTGen tool (previously introduced in section 6.2 and before) with the inserted BIST functionality in the CUTs, the AgingCalc tool was used to performed the aging analysis from 0 to 20 years of lifespan, with an interval of 5 years from one analysis to another. Moreover, the SPICE netlists, with one netlist for each degradation year, were simulated in HSPICE environment. The purpose is to perform a set of 17 simulations of BIST sessions, one for each variable V_{DD} value, and one set for each aging year to evaluate (0, 5, 10, 15 and 20 years considered, with an overall of 85 simulations/BIST runs per circuit). The V_{DD} will be depleted by 40%, from a nominal value of 1.1V and a maximum depleted value of 0.66V (a step of 0.0275V will be used in each new depleted V_{DD} value). The result of all simulations, with V_{DD} and aging variations, will be observed in a graph, to allow easier delay-fault identification (as we will see in the present section). As mentioned previously, the BIST signatures will be represented in unsigned decimal values, for easier depiction.

6.3.1 CUT_EXAMPLE CIRCUIT

For the CUT_example circuit, the HSPICE simulations resulted in the following set of VSCs, which are represented in Figure 39. In the graph we can easily identify 2 aging degradations in the simulations. The left-most is the first aging degradation spotted during circuit lifetime and is a small-delay defect. This degradation does not limit circuit's reliability, as it is degradation in a small path, or a change in path-delay reordering occurred in small-paths, and therefore the safety-margin of the circuit,

known as time-slack, is not changed. However, the right-most degradation spotted is a gross-delay defect and it reduces the circuit's safety margin to accommodate delay variations. To maintain the original circuit's time-slack for all the expected lifetime, one of two actions must be taken for 20 years of operation: (1) reduce clock frequency or (2) increase power-supply voltage, to recover the circuit's initial safety margin.

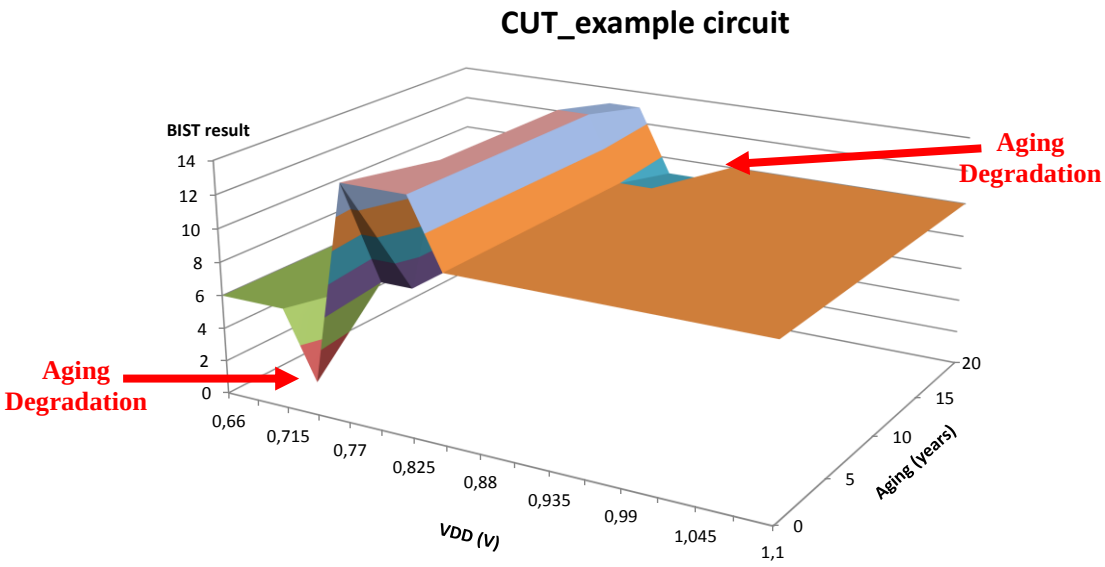


Figure 39: CUT_example's BIST signatures for V_{DD} and aging variations (VSC evolution with aging).

6.3.2 B01 CIRCUIT SIMULATION RESULTS

The aging degradation results for the B01 circuit are represented in Figure 40. This circuit is a more complex circuit, when compared with the previous example, and therefore it is expected that higher number aging degradations should be spotted. In fact, just for 5 years of lifetime it is possible to spot the two left-most aging variations, signalized in the picture. This are variations in small-delay paths and do not reduce circuit's time-slack. For 10 and 15 years of lifetime there are also aging variations detected, but in this graph they are unseen. However, a simple inspection on graph's data allows us to detect them. Finally, for 20 years of life-time, a gross-delay variation alters circuit's time-slack (the right-most variation spotted), making the circuit more vulnerable and reducing its reliability.

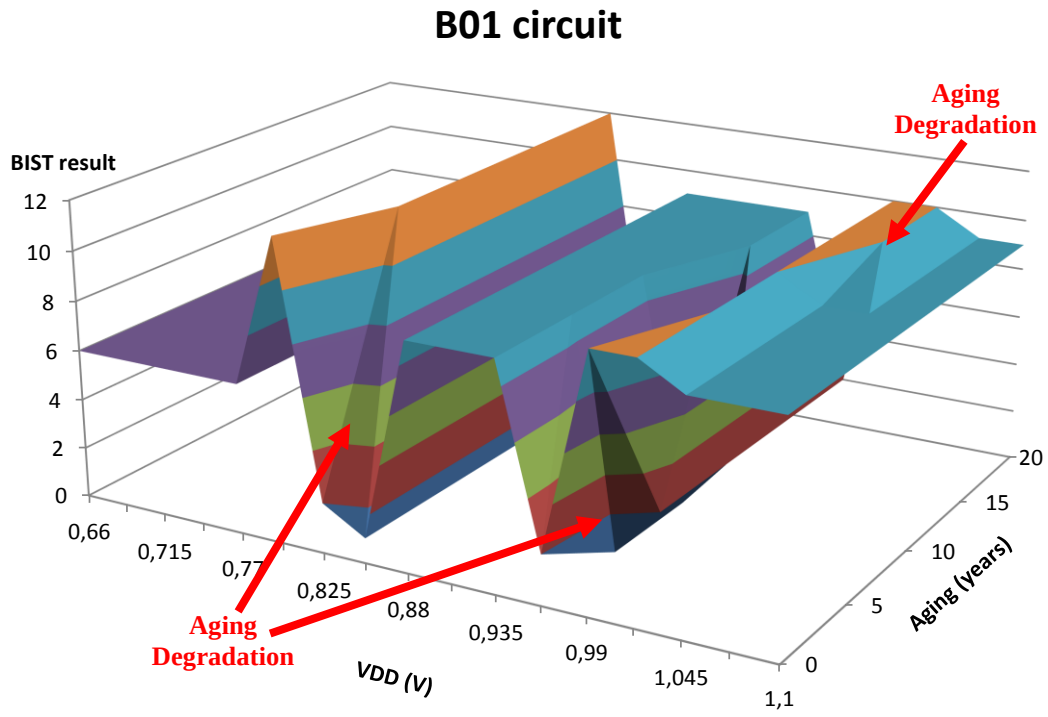


Figure 40: B01's BIST signatures for V_{DD} and aging variations (VSC evolution with aging).

6.3.3 B06 CIRCUIT SIMULATION RESULTS

The last example circuit is B06 and the simulation results are presented in Figure 41. The result is interesting as only 2 BIST signatures were obtained in each VSC (393 and 213). The reason is that this is a particular circuit where several critical paths were obtained in the BIST circuitry and not on the CUT. For that reason, it creates a specific condition that makes CUT's CPs to be masked by the BIST circuitry's CPs, and therefore the BIST signatures are limited in a VSC. Nevertheless, it is possible to identify aging degradations, as can be seen, and in this case circuit's time-slack is reduced just for 5 years of life-time.

The interesting aspect in this circuit example is that, not only CUT's aging degradation can reduce circuit's reliability. The BIST circuitry is also subject to aging variations during circuit lifetime and their CPs may also impose a limit for circuit's performance. However, this aging sensor methodology can identify gross-delay

defects that may limit circuit operation, but also small delay defects that give information on how the circuit is aging (in terms of path-delay variations), regardless of their origin.

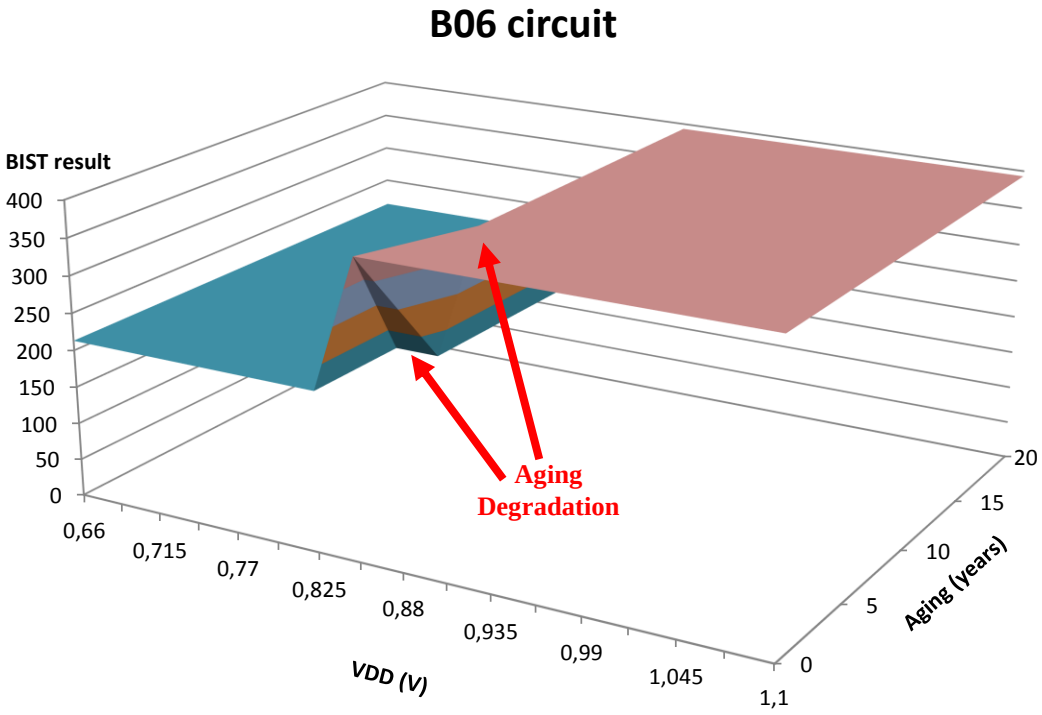


Figure 41: B06's BIST signatures for V_{DD} and aging variations (VSC evolution with aging).

7. CONCLUSIONS AND FUTURE WORK

7.1 CONCLUSIONS

A large amount of investigations has been done in the past to conceive 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 65*nm* technologies and below, delay testing is becoming more and more important. Hundred of million gates are operating now in the GHz range and new processing materials and manufacturing processes were conceived. 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.

7.1.1 CONCLUSIONS ON SCAN-BASED BIST AND BISTGEN TOOL

BIST is an attractive technique for digital system test. Scan BIST merges BIST and Scan Design techniques, with their associated costs and benefits.

For external test, scan design is widely used, and has been extended to cope with delay testing. Launch-on-Capture (LOC) and Launch-on-Shift (LOS) are the two 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 with traditional ATE (Automatic Test Equipment).

Adapting Scan BIST to uncover delay-faults, which is mandatory for digital SoC implemented in nano-CMOS, is the first main objective of this work. In this Dissertation a new methodology for dynamic scan BIST addition has been proposed,

implemented and automated with the new BISTGen tool. The underlying principle is to apply LOS and LOC techniques to scan BIST. The proposed methodology uses linear and modular implementation which is dual architecture permission. However, using the same architecture, the system implements three new Scan BIST solutions to cover fault pattern generation. 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 in the same BIST test). The new architecture is composed also by the addition of new modules, and the BIST controller as Finite State Machine (FSM) has one additional 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 'Teste_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 transistor count (and the same silicon area) and pin count, although the hardware changes. Performance degradation in the CUT by BIST insertion is similar to the one of classic scan BIST and the additional propagation delays associated with the input MUX and with the replacement of the CUT's flip-flops by scan flip-flops don't change substantially.

The generation and insertion in the CUT of this new scan-based BIST approach for delay-faults was automated, and a new software tool, BISTGen, was developed to allow this automated procedure.

In section 6.2, it was demonstrated that BISTGen tool can effectively generate and insert BIST circuitry, aiming delay-fault detection, into a CUT. The BISTGen works with both behavioural descriptions and structural descriptions for the BIST circuitry, and with both VHDL and Verilog type of HDL circuit representations. Its use, allows easier BIST application to a CUT, and reduces design time and effort to include DfT techniques. As a drawback, the fact that BIST sections performed at-speed should increase V_{DD} variations and power consumption, leading to additional delay-faults, not present in normal operation. For these fact, a more thorough analysis on this problematic is mandatory for future work, as it is out of the scope of this work.

7.1.2 CONCLUSIONS ON AGING SENSOR METHODOLOGY

In respect to the second main objective of this M.Sc. thesis, assuming that on-chip power management may be available (to allow applying to a BIST structure a set of static V_{DD} values), an Aging Sensor Methodology was proposed, using dynamic BIST and multi-V_{DD} self test. The output of the multi-V_{DD} self test is a set of digital signatures (one for each V_{DD} value), producing what we refer as the VSC (Voltage Signature Collection). The VSC is a set of (V_{DDi}, Si) pairs of values. For a circuit with no aging degradations, a specific VSC result will be obtained, and referred as the golden VSC. In the presence of cumulative aging degradations and the consequently path-delay variations and, eventually, path reordering, the VSC is modified, allowing the detection of these aging degradations (in CUT's small paths or in CUT's CPs).

Simulations demonstrating this aging degradations detection were presented in section 6.3. From the simulation results it is possible to show that the Aging Sensor Methodology for circuits with BIST can effectively be used to detect aging degradations during circuit's lifetime. In the presence of Temperature variations, it causes a shift on the BIST signatures in respect to V_{DD} values. However, in the presence of aging degradations it causes the BIST signatures to be changed, allowing the identification of an aging degradation. Moreover, this aging degradation can affect circuit's non-critical paths, and no change is made in the circuits' time-slack, which does not affect performance (yet). But, if the aging degradation occurs in CPs, a time-slack reduction is obtained and the circuit is more vulnerable to delay-faults, regardless of their origin. In this last case, to maintain the original slack margin, performance should be reduced by reducing clock frequency, or power dissipation should be increased by increasing power-supply voltage, in order to recover the lost slack margin.

These results may lead us to conclude that, if only gross-delay defects limit circuit's performance, the V_{DD} reduction needed to implement effectively this aging monitoring methodology in real circuits is less demanding than what it was here used in simulations. This is important because reduces complexity in the power management module that allows DVS and in fact allows an easy implementation in real circuits. Moreover, if only gross delay defects are analysed in the performance

degradation monitoring, we may also conclude that circuit complexity does not affect methodology implementation for bigger circuits, and scalability is assured. The correct BIST signature obtained for higher V_{DD} values is the same for all circuit samples, and methodology will monitor the V_{DD} margin for which this correct BIST signature is valid (considering only the analysis of gross-delay aging defects).

In respect to smaller delay defects caused by aging variations, the degradation monitoring of the non-critical paths, accomplished with the BIST sessions performed with lower V_{DD} values, may be important to spot some defects that are not critical but that could become critical in the near future. However, these conclusions can not be drawn from the present work and further research on this topic should be made in the future. The presence of operation induced variations, like power-supply disturbances or temperature hot-spots, may change VSC for the lower V_{DD} signatures, which may limit the diagnosis and identification of these non-critical delay-faults. Moreover, this problem increases with circuit complexity and therefore further investigation is needed to evaluate the impact of operation induced variations on the VSC.

7.2 FUTURE WORK

The work described in this dissertation, as every research & development work, is not a task completed. Some improvements were already identified as future work possibilities and it also opens new perspectives for undone research. This section summarizes these future work possibilities.

Regarding BIST structures and BISTGen software, several topics may be identified as future works:

- BIST circuitry should be optimized, specially the controller block, to optimize the gate level netlist structure in order to reduce the CP of the BIST circuitry. In fact, it is highly recommended that the CPs be located in the CUT and not in the BIST circuitry, so that the modified circuit with BIST functionality maintains its original performance;

- A structural VHDL description and a Verilog behavioural description for the BIST circuitry should also be available and implemented in BISTGen, to improve BISTGen flexibility regarding input and output files and formats;
- Implement on BISTGen the possibility of multiple scan-chains and also the possibility of partial-scan tests.

Regarding the developed Aging Sensor Methodology, the topics identified for future work perspectives are the following:

- A thorough analysis on effective power consumption in test-mode and on the increased variability obtained in test-mode, namely on V_{DD} variations, is necessary, due to the fact that BIST sessions are performed at-speed;
- A real silicon validation of this methodology is required, being necessary to design a test-chip using more complex industry circuits as test vehicles.
- An investigation is needed to evaluate the impact of operation induced variations (like temperature hot-spots and power noise) on the VSC, and to identify procedures to limit these influences (like the use multiple scan-chains and multiple MISR).

REFERENCES

[1] http://www.bitsonchips.com/references/ref36.pdf

[2] Semiconductor Industry Association, The National Technology Roadmap for Semiconductors, 1997, http://www.semichips.org

[3] R. Bennetts, Design of Testable Logic Circuits, Addison-Wesley, Reading, MA, 1984.

[4] R. Williams, IBM perspectives on the electrical design automation industry, in: Keywords to IEEE Design Automation Conference, 1986.

[5] B. Könemann, Creature from the Deep Submicron Lagoon, in: Keywords to the 10th ITG Workshop on Testmethoden und Zuverl□assigkeit von Schaltungen, Herrenberg, Germany, March 1998.

[6] Roger Barth, Test and Test Equipment December 2012 Hsin-Chu, Taiwan, ITRS 2012 Winter Report, www.itrs.net/Links/2012Winter/1205%20Presentation/Test_12052012.pdf.

[7] http://www.cs.colostate.edu/~cs530/digital_testing.pdf

[8] http://www.ece.ucdavis.edu/~halasaad/Data/vlsi02.pdf

[9] "IEEE Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture IEEE Std 1149.7-2009."

[10] Charles E. Stroud, "A Designer's Guide to Built-in Self-Test", Kluwer Academic Publishers, 2002.

[11] Das, S. R. (Oct. 1991). "Built-in self-testing of VLSI circuits", IEEE Potentials, 10, pp. 23-26.

[12] McCluskey, E. J. (April 1985). "Built-in self-test techniques", IEEE Design and Test of Computers,2, pp. 21- 28.

[13] Savir, J. and Bar,dell, P. H. (March 1993). "Built-in self-test: milestones and challenges", VLSI Design, 1, pp. 23-44.

[14] Pancholy A, Rajski J., McNaughton L. J. "Empirical Failure Analysis and Validation of Fault Models in CMOS VLSI", International Test Conference '90, Washington D.C., 10-12. September 1990, pp.938- 947.

[15] SUNIL R. DAS, NITA GOEL, WEN B. JONE and AMIYA R. NAYAK. "Syndrome Signature in Output Compaction for VLSI Built-in Self-Test", VLSI DESIGN, Vol. 7, No. 2, pp. 191-201, 1998.

[16] N. Nicolici, B. M. Al-Hashimi, A. D. Brown, and A. Williams, “BIST Hardware Synthesis for RTL Data Paths Based on Test Compatibility Classes,” IEEE Trans.Computer-Aided Design of Integrated Circuits and Systems, vol. 19, no. 11, pp. 1375–1385, Nov. 2001.

[17] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. An analytical model for negative bias temperature instability. In ICCAD ’06: Proc. of the 2006 IEEE/ACM International Conference on Computer-Aided Design, pages 493–496, New York, 2006.

[18] S. Mahapatra, M. A. Alam, P. B. Kumar, T. R. Dalei, D. Varghese, and D. Saha. Negative bias temperature instability in CMOS devices. Microelectron. Eng., 80(1):114–121, 2005.

[19] L. Tsetseris, X. J. Zhou, D. M. Fleetwood, R. D. Schrimpf, and S. T. Pantelides. Physical mechanisms of negativebias temperature instability. Applied Physics Letters, 86(14):142103, 2005.

[20] B. Kaczer, V. Arkhipov, R. Degraeve, N. Collaert, G. Groeseneken, and M. Goodwin. Temperature dependence of the negative bias temperature instability in the framework of dispersive transport. Applied Physics Letters, 86(14):143506, 2005.

[21] http://en.wikipedia.org/wiki/Paradox_(database) [17-05-2013]

[22] “Linear Feedback Shift Register,” en.Wikipedia.org [22-03-2013]

[23] C. Stroud, A Designer’s Guide to Built-In Self-Test, Kluwer Academic Publishers, Bos-ton MA, 2002

[24] S. Zhang, R. Byrne, J.C. Muzio, D.M. Miller, “Why cellular automata are better than LFSRs as built-in self test generators for sequential-type faults”, IEEE International Symposium on Circuits and Systems, Vol. 1,1994

[25] L. Wang and E. McCluskey, “Complete Feedback Shift Register Design for Built-In Self-Test,” Proc. IEEE International Conference on Computer-Aided Design, pp. 56-59, 1986

[26] C. Stroud, A Designer’s Guide to Built-In Self-Test, Kluwer Academic Publishers, Bos-ton MA, 2002

[27] Michael L. Bushnell and Vishwani D. Agrawal. Essentials of electronic testing for digital, memory, and mixed signal VLSI circuits elektronisk ressurs. Frontiers in electronic testing. Kluwer Academic, New York, 2002.

[28] J.A. Waicukauski, E.Lindbloom, B. Rosen and V.Iyengar, “Transition Fault Simulation by Parallel Single Fault Propagation”, in Proceedings of IEEE International Test Conference, Sept 1986, pp. 542-549.

[29] J.Savir and S. Patil, “Scan-based Transition Test”, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 12, No. 8, Aug 1993.

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

[31] V.S. Iyengar, B.K. Rosen and J.A. Waicukauski, “On Computing the Sizes of Detected Delay Faults”, IEEE Transactions on Computer Aided Design, Vol. CAD-9, pp.299-312, March 1990.

[32] J.Savir and S.Patil, “On Broad-Side Delay Test”, VLSI Test Symposium, Sept 1994, pp 284-290.

[33] B. Dervisoglu and G.Stong, “Design for Testability: Using Scan-path Techniques for Path-delay Test and Measurement”, Proceedings of IEEE International Test Conference, 1991, pp. 365-374.

[34] G.L. Smith, “Model for Delay Faults Based Upon Path”, in Proceedings of IEEE International Test Conference, Nov 1985, pp. 342-349.

[35] N. Tendolkar, R. Raina, R. Woltenberg, X.Lin, B. Swanson and G. Aldrich,“Novel Techniques for Achieving High At-speed Transition Fault Test Coveragefor Motorola’s Microprocessors Based on PowerPCTM Instruction Set Architecture”, in Proceedings of IEEE VLSI Test Symposium., Apr-May 2002, pp. 3-8.

[36] T. L. McLaurin and F. Frederick, “The Testability Features of the MCF5407 Containing the 4th Generation Coldfire Microprocessor Core”, in Proceedings of IEEE International Test Conference, Oct. 2000, pp.151-159.

[37] Miura Y, Matukura Y. Investigation of silicon–silicon dioxide interface using MOS structure. Jpn J Appl Phys 1966;5:180; Goetzberger A, Nigh HE. Surface charge after annealing of Al–SiO2–Si structures under bias. Proc IEEE 1966; 54:1454.

[38] G.Chen, K.Y.Chuah, M.F.Li, D. Chan, C.H.Ang, J.Z.Zheng, Y.Jin, and D.L.Kwong. Dynamic NBTI of PMOS transistors and its impact on device lifetime. In RPSP ’03: Proc. of the 41st annual symposium on Reliability Physics, pages 196–202, Dallas, Texas, 2003.

[39] A. S. Goda and G. Kapila. Design for degradation: Cad tools for managing transistor degradation mechanisms. In ISQED ’05: Proc. of the 6th International Symposium on Quality of Electronic Design, pages 416–420, Washington, DC, 2005.

[40] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. Impact of NBTI on SRAM read stability and design for reliability. In ISQED ’06: Proc. of the 7th International

Symposium on Quality Electronic Design, pages 210–218, Washington, DC, 2006.

[41] N.K.Jha, P.S.Reddy, D.K.Sharma, and V.R.Rao. NBTI degradation and its impact for analog circuit reliability. IEEE Trans. on Electron Devices, 52(12):2609–2615, 2005.

[42] B. C. Paul, K. Kang, H. Kufluoglu, M. A. Alam, and K. Roy. Temporal performance degradation under NBTI: Estimation and design for improved reliability of nanoscale circuits. In DATE ’06: Proc. of the Conference on Design, Automation and Test in Europe, pages 780–785, Leuven, Belgium, 2006.

[43] V. Reddy, J. Carulli, A. Krishnan, W. Bosch, and B. Burgess. Impact of negative bias temperature instability on product parametric drift. In ITC ’04: Proc. of the International Test Conference, pages 148–155, Washington, DC, 2004.

[44] R. Vattikonda, W. Wang, and Y. Cao. Modeling and minimization of PMOS NBTI effect for robust nanometer design. In DAC ’06: Proc. of the 43rd Annual Conference on Design Automation, pages 1047–1052, New York, 2006.

[45] X. Yang, E. F. Weglarz, and K. K. Saluja. On NBTI degradation process in digital logic circuits. In International Conference on VLSI Design, pages 723–730, Bangalore, India, 2007.

[46] Zeghbroeck, B. Van. "Chapter 7: MOS Field-Effect-Transistors." Principles of Semiconductor Devices. 2004. <http://ecewww.colorado.edu/~bart/book/book/Chapter7/ch7_7.htm>.

[47] Subramoniam, R. A Statistical Model of Oxide Breakdown Based on a Physical Description of Wearout. Proceedings of IEEE International Electron Devices Meeting, 13 Dec. 1992, Electron Devices Soc.IEEE.

[48] Lee, J.c., Chen Ih-Chin, and Hu Chenming. "Modeling and Characterization of GateOxide Reliability." IEEE Transactions on Electron Devices 35 (1998).

[49] Noguchi, Junji. "Dominant Factors in TDDB Degradation of Cu Interconnects."IEEE Trransactions on Electron Devices, 52 (2005): 1743-1750.

[50] T. H. Ning et al.,”1 μm MOSFET VLSI Technology: Part IV-Hot electrondesign constraints,” IEEE Trans. Electron Devices, vol.26, pp.346-353, 1979.

[51] C. Hu, S.C. Tam, F. Hsu, P. Ko, T. Chan, K. W. Terrill, ”Hot Electron Induced MOSFET Degradation-Model, Monitor and Improvement,” J. Solid State Circuits, vol.20(1), pp.295-305, 1985.

[52] K.J. Puttlitz, K.A. Statler, Handbook of Lead-Free Solder Technology for Microelectronic

[53] J.R. Lloyd, Appl. Phys. Lett. 79 (7) (2000) 1061–1062.

[54] J.R. Lloyd, Microelectron. Eng. 49 (1999) 51–64.

[55] J.R. Black, IEEE Trans. Elec. Dev. 16 (4) (1969) 338–347.

[56] K. Musaka, B. Zheng, H. Wang, K. Wijekoon, L. Chen, J. Lin, K. Watanabe, K. Ohira, T. Hosoda, K. Miyata, T. Hasegawa, G. Dixit, R. Chueng, M. Yamada and S. Kadomura, Proceedings of the International Interconnect Technology Conference (IEEE, New York, 2001), p.83.

[57] T. Suzuki, S. Ohtsuka, A. Yamanoue, T. Hosoda, T. Khono, Y. Matsuoka, K. Yanai, H. Matsuyama, H. Mori, N. Shimizu, T. Nakamura, S. Sugatani, K. Shono and H. Yagi, Proceedings of the International Interconnect Technology Conference (IEEE, New York, 2002), p. 229.

[58] J. A. Nucci, Y. Shacham-Diamond and J. E. Sanchez, Jr., Appl. Phys. Lett. 66 (26), 3585 (1995).

[59] J. A. Nucci, R. R. Keller, J. E. Sanchez, Jr. and Y. Shacham-Diamond, Appl. Phys. Lett. 69 (26), 4017 (1996).

[60] J. A. Nucci, R. R. Keller, D. P. Field and Y. Shacham-Diamond, Appl. Phys. Lett. 70 (10), 1242 (1997).

[61] A. Sekiguchi, J. Koike, S. Kamiya, M. Saka and K. Murayama, Appl. Phys. Lett. 79 (9), 1264 (2001).

[62] A. Veloso, T. Hoffmann, A. Lauwers, H. Yu, S. Severi, E. Augendre, S. Kubicek, P. Verheyen, N. Collaert, P.Absil, M. Jurczak, S. Biesemans, “Advanced CMOS device technologies for 45nm node and below”, Science and Technology of Advanced Materials, vol.8, pp. 214–218, 2007.

[63] R. H. Dennard, J. Cai, and A. Kumar, “A perspective on today’s scaling challenges and possible future directions”, Solid-State Electronics, 51 (2007), 518–525.

[64] J. Schwank, IEEE NSREC Short Course, 2002.

[65] T. R. Oldham and F. B. McLean, “Total ionizing dose effects in MOS oxides and devices”, IEEE Trans. Nucl. Sci., vol. 50, no. 3, pp. 483-499, June 2003.

[66] H. J. Barnaby “Total-ionizing-dose effects in modern CMOS technologies”, IEEE Trans. Nucl. Sci., vol. 53, no. 6, pp. 3103-3121, December 2006.

[67] S. T. Pantelides, L. Tsetseris, S. N. Rashkeev, X. J. Zhou, D. M. Fleetwood, and R. D. Schrimpf, “Hydrogen in MOSFETs – A primary agent of reliability issues”, Microelectron. Rel., vol. 47, pp. 903-911, 2007.

[68] N. S. Saks, M. G. Ancona, and J. A. Modolo, “Radiation effects inmost capacitors with very thin oxides at 80 K”, IEEE Trans

[69] F. Faccio and G. Cervelli “Radiation-induced edge effects in deep submicron CMOS transistors”, IEEE Trans. Nucl. Sci., vol. 52, no. 6, pp. 2413-2420, December 2005.

[70] M. R. Shaneyfelt, P. E. Dodd, B. L. Draper, and R. S. Flores, “Challenges in hardening technologies using shallow-trench isolation”, IEEE Trans. Nucl. Sci., vol. 45, no. 6, pp. 2584–2592, December 1998.

[71] P. E. Dodd, M. R. Shaneyfelt, J. R. Schwank, and J.A. Felix, “Future challenges in radiation effects”, Short course of the 10th European Conference on Radiation Effects on Components and Systems (RADECS), September 2009.

[72] K. Kang, S. Gangwal, S. Park, And K. Roy, “NBTI Induced Performance Degradation In Logic And Memory Circuits: How Effectively Can We Approach A Reliability Solution?”, In Asia And South Pacific Design Automation Conference, pp. 726–731, March 2008.

[73] C. Martins, “Adaptive Error-Prediction Aging Sensor for Synchronous Digital Circuits”, M.Sc. Thesis, University of Algarve, October, 2012.

[74] International Technology Roadmap for Semiconductors (ITRS) web page: www.itrs.net, accessed in August 2013.

[75] Silvia A. T. Gomes, "BIST Architectures for Dynamic Test", M.Sc. Thesis, DEEC/IST, TUL (Technical University of Lisbon), 2007.

[76] J. Semião, "Power-Supply and Temperature Based Methodologies to Improve Tolerance and Detection of Delay Faults in Synchronous Digital Circuits", Ph.D. Thesis, IST, TUL, July, 2010.

[77] J. Pachito, "Aging Prediction Methodology for Digital Circuits", M.Sc. Thesis, ISE, UAlg, January, 2012.

[78] M. Rodríguez Irago, J. J. Rodríguez Andina, F. Vargas, J. Semião, I. C. Teixeira, J. P. Teixeira, “Dynamic Fault Detection in Digital Systems Using Dynamic Voltage Scaling and Multi-Temperature Schemes”, IOLTS06 – Proceedings of the 12th IEEE International On-Line Testing Symposium, Lake of Como, Italy, July, 2006, DOI: http://dx.doi.org/10.1109/IOLTS.2006.25.

[79] Tecmic, http://www.tecmic.pt/eng/xtran/xtran_intro.html. Last visit on 1th/May/2010.

[80] Stefan Gerstendörfer, Hans-Joachim Wunderlich, “Minimized Power Consumption for Scan-Based BIST”, Journal of Electronic Testing: Theory and Applications, Volume 16, Issue 3, special issue on the European test workshop

1999, June 2000, Pages: 203 – 212, ISSN:0923-8174, Kluwer Academic Publishers.

[81] R. A. Frohwerk, "Signature Analysis: A New Digital Field Service Method", Hewlett-Packard Journal, Vol. 28, Nr. 9, pp.2-8, May 1977.

[82] J. Savir, "Syndrome-Testable Design of Combinational Circuits", IEEE Trans. On Computers, Vol. C-69, nº 6, pp. 442-451, June 1980.

[83] J. P. Hayes, "Transition Count Testing of Combinational Logic Circuits", IEEE Trans. On Computers, Vol. C-25, nº 6, pp. 613-620, June 1976.

[84] R. Davis, "Signature Analysis of Multi-Output Circuits", in Proc. of the International Fault-Tolerant Computing Symp., pp. 366-371, June 1984.

[85] Vijay Reddy, Anand T. Krishnan, Andrew Marshall, John Rodriguez, Sreedhar Natarajan, Tim Rost, and Srikanth Krishnan, "Impact of negative bias temperature instability on digital circuit reliability", Reliability Physics Symposium Proceedings, 40th Annual, Page(s): 248 – 254, ISBN: 0-7803-7352-9, DOI: 10.1109/RELPHY.2002.996644

[86] D. K. Schroder, J. A. Babcock, "Negative Bias Temperature Instability: Road to Cross in Deep Submicron Silicon Semiconductor Manufacturing", Journal of Applied Physics, Vol. 94 Issue: 1, pp. 1-18, DOI: 10.1063/1.1567461, July, 2003.

[87] M. A. Alam, S. Mahapatra, "A Comprehensive Model of PMOS NBTI Degradation", Journal of Microelectronics Reliability, Vol. 45 Issue: 1, pp. 71-81, DOI: 10.1016/j.microrel.2004.03.019, January, 2005.

[88] W. Wang, V. Reddy, A. T. Krishnan, R. Vattikonda, S. Krishnan, Y. Cao, "Compact modeling and simulation of circuit reliability for 65nm cmos technology", IEEE Transactions on Device and Materials Reliability, Vol. 7, No. 4, pp. 509–517, 2007.

[89] E. T. Ogawa, J. Kim, G. S. Haase, H. C. Mogul, J. W. McPherson, "Leakage breakdown and tddb characteristics of porous low-k silica", IEEE International Reliability Physics Symposium, pp. 166–172, 2003.

[90] N. Kimizuka, T. Yamamoto, T. Mogami, K. Yamaguchi, K. Imai, T. Horiuchi, "The impact of bias temperature instability for direct-tunneling ultra-thin gate oxide on mosfet scaling", VLSI Symposium on Technology, pp. 73–74, 1999

[91] S. Borkar, "Electronics beyond nano-scale CMOS", ACM/IEEE Design Automation Conference pp. 807–808, 2006.

[92] M. A. Alam, S. Mahapatra, "A comprehensive model of pmos nbti degradation", Microelectronics Reliability Vol. 45: 71–81, 2005.

[93] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, S. Vrudhula, “Predictive modeling of the nbti effect for reliable design”, IEEE Custom Integrated Circuits Conference, pp. 189– 192, 2006.