Embedded Testing Architectures

DISSERTATION

submitted to the Inquiry Commission,
designated by the General Assembly of Special Composition of
the Department of Computer Science & Engineering
of the School of Sciences of University of Ioannina,

by

Vasileios Tenentes

in partial fulfillment of the requirements

FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

June 2013

Advisory Committee

Xrysovalantis Kavousianos
University of Ioannina

Yiorgos Tsiatouhas
University of Ioannina

Krishnendu Chakrabarty
Duke University

Inquiry Commission

Xrysovalantis Kavousianos
University of Ioannina

Yiorgos Tsiatouhas
University of Ioannina

Krishnendu Chakrabarty
Duke University

Aristides Efthymiou
University of Ioannina

Dimitris Nikolos
University of Patras

Dimitris Gizopoulos
University of Athens

Emmanouil Kalligeros
University of the Aegean

This thesis is deticated to my parents.

For their endless love, support and understanding

Acknowledgements

First and foremost I have to thank my *parents* for their love and support throughout my life.

I would like to sincerely thank my supervisor, *Prof. Kavousianos*, for his guidance and support throughout this study, and especially for his confidence in me. I wish that the priviledge of working closely with him blessed me with just some of his virtues as a scientist and a person. I would also like to thank *Prof. Chakrabarty* for serving as a member of my thesis committee and for the oportunity he gave me to spend some months at Duke University. Not just his insight and his ingenious mind, but also his multicultural personality make the work with him a delightful experience. Also, I thank *Prof. Tsiatouhas* for the discussions we had and his important tutoring and *Prof. Kalligeros* for the interpretation of some results presented in this thesis.

Also, I want to thank my colleagues Vartziotis and Sfikas for the important discussions we had throughout many lunches.

To all my *friends*, thank you for your understanding and encouragement. Although, I cannot list your names, you are always part of my life.

I am also grateful to all those people that supported with their hard work the "*Heraclitus II*" scholarships.

Table of Contents

List of Figures

List of Tables

Glossary

AMC	Airborne Molecular Contamination
ASA	Average Switching Activity
ATE	Automatic Test Equipment
ATPG	Automated Test Pattern Generation
BCE^{+}	Bridging Coverage Estimation
BF	Bridging Fault
BIST	Built-In Self-Test
CMOS	Complementary Metal-Oxide-Silicon
CUT	Circuit Under Test
DFT	Design For Testability
DRAM	Dynamic Random-Access Memory
EDT	Embedded Deterministic Test
FA	Fill Adjacent
IC	Integrated Circuit
IDDQ	Technique for monitoring steady-state power supply (I_{ddq})
I/O	Input/Output
IP	Intellectual Property
ISCAS	International Symposium on Circuits and Systems (refers to benchmark' suite)
ITA	Interface Test Adapter
ITRS	International Technology Roadmap for Semiconductors
IWLS	International Workshop on Logic and Synthesis (refers to benchmark' suite)
KGD	Known Good Die
LFSR	Linear Feedback Shift Register
LOC	Launch-On-Capture
LOS	Launch-On-Shift
LSI	Large-Scale Integration
LSSD	Level-Sensitive Scan Design
MCSoC	Multi-Core System on Chip
MFA	Modified Fill Adjacent
MID	Mobile Internet Device
ORA	Output Response Analyzer
OSH	Optimal Selective Huffman
PCI	Peripheral Component Interconnect

PDA	Personal Digital Assistant
PI	Primary Input
PO	Primary Output
PPI	Pseudorandom Primary Input
PPM	Parts Per Million
PPO	Pseudorandom Primary Output
RAM	Random-Access Memory
PF	Preferred-Fill
RF	Random-Fill
RISC	Reduced Instruction Set Computing
SE	Scan Enable
SI	Scan Input
SO	Scan Output
SoC	System on Chip
SSI	Small-Scale Integration
SSLFSR	State-Skip Linear Feedback Shift Register
SSS_LFSR	Single-State-Skip Linear Feedback Shift Register
TAT	Test Application Time
TDC	Test Data Compression
TDF	Transition Delay Fault
TDP	Thermal Design Power
TDV	Test Data Volume
TPG	Test Pattern Generator
TRP	Test Resource Partitioning
TSE	Test Set Embedding
TSL	Test Sequence Length
UDL	User Define Logic
VDSM	Very-Deep Sub-Micro
VLSI	Very-Large-Scale Integration
VSS_LFSR	Variable-State-Skip Linear Feedback Shift Register

Abstract

Tenentes, Vasileios, PhD
Department of Computer Science & Engineering, University of Ioannina, Greece.
June, 2013
Title of Dissertation: *Embedded Testing Architectures*
Thesis Supervisor: Xrysovalantis Kavousianos

The shrinking of transistor's size in the Very Deep Sub-Micron (VDSM) technologies enabled the manufacturing of Multi-core Systems-on-Chips (MCSoCs) that contain billions of transistors. The exponential decrease in the transistor's manufacturing cost is the main contributor to the widespread use of electronic systems. However, due to manufacturing process imperfections, which cause manufacturing defects, electronic devices need to be tested for compliance with their specifications before they are shipped to customers. Testing of such complex systems becomes increasingly difficult and costly while the overall cost must remain below certain bounds.

Manufacturing testing of Integrated Circuits (ICs) is conducted by expensive specialized Automatic Test Equipment (ATE) with limited resources, such as communication channels, memory and channels' bandwidth. Test cost depends on the time a chip spends on an ATE as well as on the utilization of these resources. An efficient testing method should be fast, accurate and must utilize the minimum number of ATE resources. At the same time, outdated ATEs are commonly in use due to the high cost associated with upgrading this equipment. To enable the testing of contemporary dense devices on outdated ATE's, Test Resource Partitioning (TRP) techniques emerged. According to TRP, testing architectures are embedded on chip and operate in synergy with ATEs in order to decrease test cost.

Test cost is also affected by both the shrinking of transistor and the high integration of transistors in MCSoCs. As transistor shrinks the amount of tests that are required in order to achieve high defect coverage and assure quality goals increases, due to the ICs becoming more and more sensitive to physical phenomena. Moreover, the high integration of transistors in MCSoCs sets power consumption constraints during testing. Violations of these constraints during testing may cause circuits failures, which do not occur in the field. Power consumption constraints increase further test complexity, since the circuit consumes more power in test mode than in normal mode of operation. As a result, new

low-power testing techniques are required in order to handle in a unified manner all test cost related objectives.

In this dissertation methods are presented that target multiple test cost objectives:

1. To decrease both the test application time and the ATE memory requirements, a new technique is proposed that shortens the test sequence length of Test Set Embedding (TSE) techniques. TSE techniques have very small ATE memory requirements, but they suffer from long test sequences that forbid their practical use. A new device is proposed that skips efficiently the useless parts of the test sequences, making TSE techniques attractive for manufacturing testing.

2. To reduce the power demands of linear-based decompressors without any additional ATE memory requirements, a linear-based method which offers both high compression and low shift power consumption is proposed. A low-cost, flexible scheme is also described, which can be combined with any linear-based method for reducing the shift power during testing.

3. To reduce the average power requirements of tests under peak power constraints, and also to increase the arbitrary defects that can be detected by these tests, a unified algorithmic test generation technique is proposed. The proposed method generates tests that exhibit average power reduction similar to that of the best power-driver algorithmic method in the literature, while at the same time, a) it complies with peak power specifications, and b) it generates tests with enhanced defect coverage on arbitrary defects.

4. To increase the defect coverage of the generated tests, new defect-oriented compression algorithms are proposed that can be applied on most linear-based reseeding schemes. The proposed methods are based on a probabilistic fault model and a novel probabilistic fault coverage measurement metric for grading the tests during the compression of linear-based decompressors. The case studies include classical linear decompressors with both classical static reseeding and window-based static reseeding, as well as the state-of-the-art ring generators with dynamic reseeding. It is shown that, compared to standard compression-driven approaches, higher defect coverage is obtained without any loss on compression.

5. To reduce power consumption during testing and also to increase the defect coverage, a novel deterministic TRP architecture is presented. The proposed method can be applied on both a) linear-based and b) symbol-based TRP techniques. The application of the scheme on code-based decompressors revealed an interesting property: the new decompressor exploits both the low fill rate and the correlations in the test cubes, offering better compression than any existing compression technique. Moreover, this property, together with the decompressor's low pin-count interface, enables the usage of the same decompressor for testing Intellectual Property (IP) and non-IP cores that usually coexist in MCSoCs.

Chapter 1

Introduction

1.1 Prologue

Nowadays, Very-Deep-Sub-Micron (VDSM) integration technology is in our everyday life with portable Multi-Core Systems-on-Chips (MCSoCs) that contain billions of transistors. However, even from their first construction, in the 1960s, the integrated circuits (ICs), commonly referred to as microchips or simply chips, were accompanied by the need of testing. Tens of transistors integrated into Small-scale integration (SSI) devices in the early 1960s and hundreds of transistors integrated into medium-scale integration (MSI) devices in the late 1960s, were relatively simple to test. However, thousands and tens of thousands of transistors integrated into large-scale integration (LSI) devices in the 1970s and hundreds of thousands of transistors integrated into very-large-scale integration (VLSI) devices in the early 1980s introduced serious test challenges.

This trend of higher integration scaling over the years, generally known as Moore's law [104], is the result of the exponential decrease in a transistor's manufacturing cost. However, the test cost of a transistor does not share the same trend. Figure 1.1 illustrates the

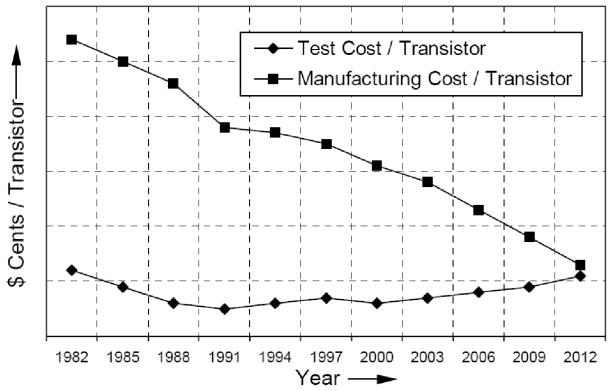


Figure 1.1: ITRS'07 test cost predictions

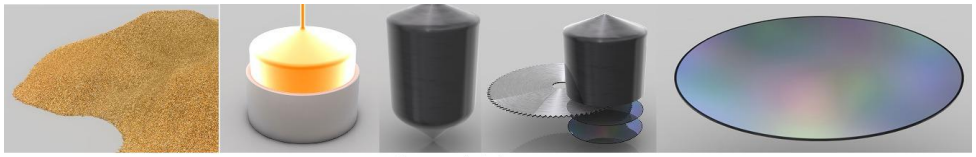


Figure 1.2: Silicon to ingot and then sliced to create wafers.

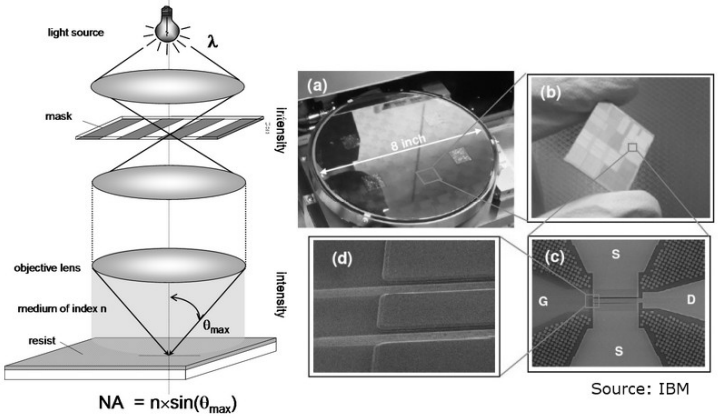


Figure 1.3: Dies printed on wafer

test versus manufacturing cost of a transistor over the years as reported in 2007's International Technology Roadmap of Semiconductors (ITRS) [2]. These news were alarming and IC testing emerged as a potential bottleneck for future exponential integration scaling according to Moore's law.

This section presents the motivation for IC testing and the major drivers that affect test cost. It introduces scan design, which is the most widely used manufacturing testing architecture, and presents the new architectural and designing trends that affect test cost.

Source: NXP

Figure 1.4: Packaging

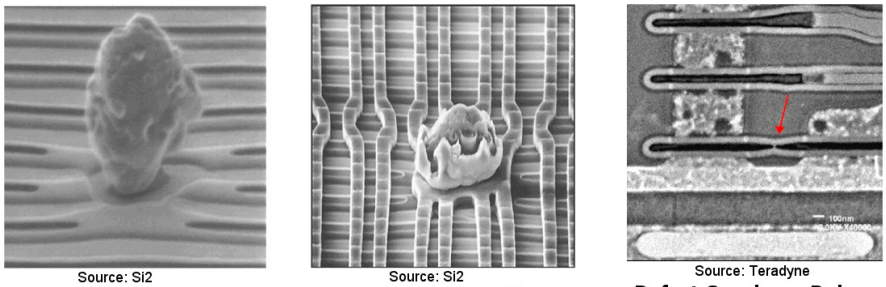


Figure 1.5: Silicon defects

1.2 Manufacturing of Integrated Circuits

The manufacturing of ICs consists of wafer fabrication and packaging. Wafer fabrication involves first creating the wafer from sand (Figure 1.2) and then printing geometric shapes corresponding to the layout onto wafer layers [43] (Figure 1.3). As each wafer contains a large number of chips, the wafer is first cut and each die is packaged (Figure 1.4).

The manufacturing process of ICs is not a perfect process. Imperfections of this process are the sources of defects, which are responsible for chip malfunctions. Defects should be identified and the defective chips must be discarded before shipment of the products. For example, particles (material which is not removed in the areas exposed by the masking process) may cause conductive or non-conductive bridges between two or more lines; incorrect spacing between connections (design rule violation or masking problem) may cause circuit shorts; holes (exposed areas that are unexpectedly etched) may lead to open interconnects or delays. Images taken with electronic microscope of manufacturing defects are presented in Figure 1.5. Figures 1.5a and 1.5b present two sites where particles trapped in the silicon during the manufacturing process caused non-conductive and conductive defects respectively. Figure 1.5c presents a thin interconnection, probably caused by process variations, which may cause the poor performance of the interconnection because of its higher resistivity.

The quality of a manufacturing process is important because it is linked to the profit margin. To grade the quality of an ICs manufacturing process, a metric is used, named *yield*. As *yield* of a manufacturing process is defined the percentage of acceptable parts

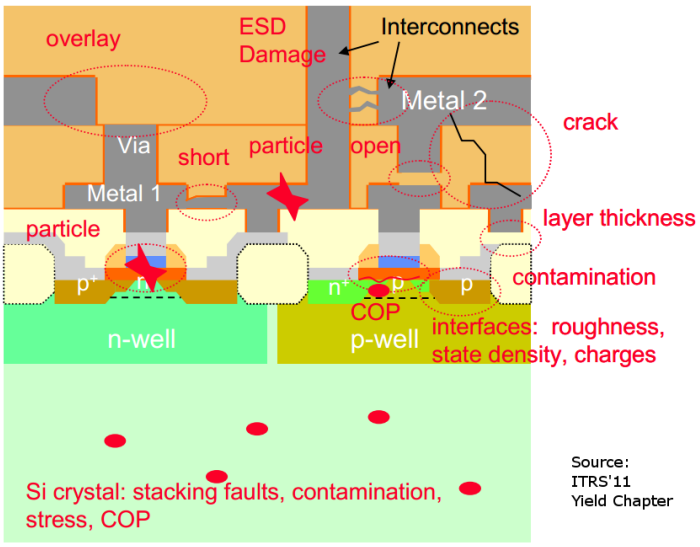


Figure 1.6: Possible causes of failures

among all parts that are fabricated [172].

$$\text{Yield} = \frac{\text{Number of acceptable parts}}{\text{Total number of parts fabricated}}$$

1.3 Defects' Sources

Several examples of contaminations and mechanisms responsible for defects are identified by the ITRS in 2011 [3] and are also shown in Figure 1.6:

- Airborne Molecular Contamination (AMC) or particles of organic or inorganic matter caused by the environment or by the tools.
- Process induced defects as scratches, cracks, and particles, overlay faults, and stress.
- Process variations may result in differing doping profiles or layer thicknesses.
- Deviation from design, due to pattern transfer from the mask to the wafer, results in deviations and variations of layout and critical dimensions.
- Diffusion of atoms through layers and in the semiconductor bulk material.

To overpass completely the reasons that cause the defects is not an easy task and in practice it is also limited by cost constraints. For example, the complete elimination of AMC is impossible and therefore research is conducted on contamination's tolerable levels under cost constraints.

1.4 Manufacturing Testing

Manufacturing testing is the process applied to detect the defective chips during manufacturing in order to avoid shipping them to customers. Because of the high complexity of the chips, manufacturing testing is done using Automated Test Equipment (ATE).

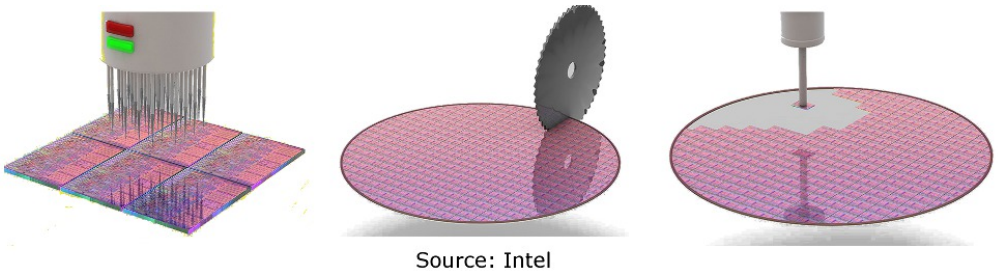


Figure 1.7: Structural testing, wafer slicing and faulty die disard

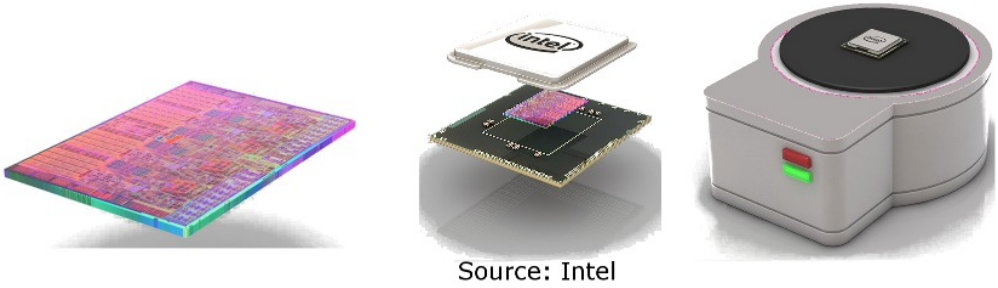


Figure 1.8: KGD packaging and functional testing

The ATE is a precision measurement tool which provides the environment required to test the circuit under test (CUT). ATE's architecture consists of a Workstation (usually a computer) that synchronizes the communication between the ATE, the CUT, and the rest of devices participating in testing. Historically, custom-designed controllers or relays were used by ATE systems. The ATE can be used on packaged dies or directly on the silicon wafer. For packaged dies, the ATE guides a robotic placement tool, called *handler*, in order to place the devices on an Interface Test Adapter (ITA). For silicon wafers, the ATE guides high precision robotic *probes* above the location of the dies to be tested and the testing is conducted through contact. When silicon wafers are tested, an ATE may sometimes test multiple dies at once. At the left-most edge of Figure 1.7 a prober is illustrated, ready to contact a silicon wafer. At the right-most edge of Figure 1.7 a handler is shown removing dies from a silicon wafer. At the right-most edge of Figure 1.8 a packaged die appears mounted on an ITA.

The Workstation is a normal desktop computer with sufficient Peripheral Component Interconnect (PCI) interfaces for accommodating different types of signal-sensing cards. Workstation takes up the role of an administrator in the ATE system: it is used for the development of test applications and storage of responses; it performs measurements on the CUT; it synchronizes measurements, such as I/O waveforms, at the proper timing.

The manufacturing testing flow is different on each technology but some common phases can be isolated and are pin-pointed on Intel's generic flow [34] in Figures 1.7 and 1.8. The wafer is first subjected to wafer sort, where most of the faulty dies are identified. Then the wafer is cut and the faulty dies are discarder. Next, the remaining known good dies (KGD) are packaged and are further tested in order to eliminate the defects which escaped wafer sort (due to electrical limitations). In order to identify weak devices with

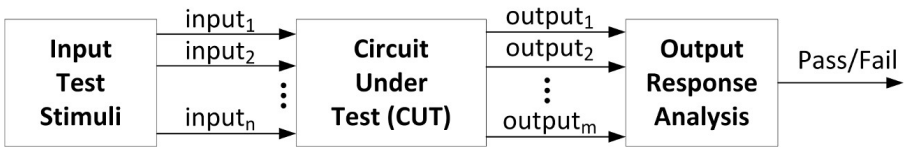


Figure 1.9: Basic Testing Approach.

high risk of failure in the short term, a process called **burn-in testing** is applied. During burn-in testing, the chip is let to operate at high-voltage and high-temperature conditions, in order to stress it and accelerate its infant mortality [6, 16]. Finally, the functionallity of the device is tested for conformance with the specifications. As final step, sometimes the manufactured chips are binned to different categories based on their performance characteristics that were measured through the testing process and then shipped to the customers.

However, manufacturing testing suffers also from imperfections. This is why, grading metrics for the quality of manufacturing testing also exist. When ICs are tested, the following two undesirable situations may occur:

1. A faulty device appears to be a good part passing the test.
2. A good device fails the test and appears as faulty.

As a result of the first case, even if all products pass acceptance test, some faulty devices can still be found in the field. When these faulty devices are returned to the IC manufacturer, they undergo Failure Mode Analysis (FMA) for possible improvements to the ICs designing and manufacturing processes. The ratio of field-rejected parts to all parts passing quality assurance testing is referred to as the reject rate, also called the defect level:

$$\text{Reject rate } = \frac{\text{Number of faulty parts passing final test}}{\text{Total number of parts passing final test}}$$

The reject rate provides an indication of the overall quality of the manufacturing testing process [16]. Generally speaking, a reject rate of 500 parts per million (PPM) chips may be considered to be acceptable, while 100PPM or lower represents high quality. The goal of six sigma manufacturing, also referred to as zero defects, is 3.4PPM or less.

1.5 Structural Manufacturing Testing

There are many techniques developed over the years for IC's manufacturing testing, but the most widely adopted one, that offers the lowest reject rate versus test cost, is *structural testing*. In this section, the basic concepts of structural testing are introduced.

A *fault* is a representation of a defect reflecting a physical condition that causes a circuit to fail to perform as designed. A *failure* is a deviation in the performance of a circuit or system from its specified behavior and represents an irreversible state of a component such that it must be repaired in order for it to provide its intended design

function. A *circuit error* is a wrong output signal produced by a defective circuit. A circuit defect may lead to a fault, a fault can cause a circuit error, and a circuit error can result in a system failure [172].

During testing a set of test stimuli (referred also as test vectors or test patterns) is applied to the n inputs of the CUT, and its m output responses are analyzed, as illustrated in Figure 1.9. Circuits that produce the correct output responses for all input stimuli pass the test and are considered to be defect-free. Those circuits that fail to produce a correct response at any point during the test sequence are assumed to be defective.

The ultimate target of any ICs test mechanism is to test the chips for all possible defects, or in other words, to achieve complete defect coverage. However, such a goal is not realistic, and thus fault models are adopted. Fault models save time and improve test efficiency, as a limited number of test patterns that target specific faults, related to the structure of the CUT, are applied at the circuit's inputs. This process is called structural testing. Any input pattern (test stimuli), that produces a different output response in a faulty circuit from that of the fault-free circuit is a *test vector* that will detect the faults. Any set of *test vectors* is called a *test set*. The goal of Automatic Test Patterns Generation (ATPG) tools is to find an efficient test set that detect as many defects as possible for a given CUT and a given fault model. These tools provide a quantitative measure of the fault-detection capabilities of a given test set for a targeted fault model. This measure is called fault coverage and is defined as:

$$\text{Fault coverage} = \frac{\text{Number of detected faults}}{\text{Total number of faults}}$$

Fault coverage is linked to the *quality* of a manufacturing process, which is expressed by the *yield*, and the *quality* of a the testing process, which is expressed by the *reject rate*, by the following relation [187]:

$$\text{Reject rate} = 1 - yield^{(1-\text{fault coverage})}$$

From this equation, we can show that an SoC with 40 cores, each having 90% fault coverage and 90% yield, could result in a reject rate of 41.9%, or 419,000 PPM. As a result, improving fault coverage can be easier and less expensive than improving manufacturing yield because making yield enhancements can be costly. Therefore, generating test stimuli with high fault coverage is very important.

Unfortunately, structural testing has its own limitations too. Fault models are used as an abstraction description of possible defects on a given design structure.

- A single fault model cannot cover all possible defects. To overcome this limitation, industry uses multiple fault models.
- Even when defects can be modeled by a fault model, sometimes it is impossible to get 100% fault coverage due to testability limitations caused by either the structure of the CUT or by the way the test is conducted (undetected faults[1]).

[1]An undetectable fault occurs where there is no test to distinguish the fault-free circuit from a faulty circuit containing that fault.

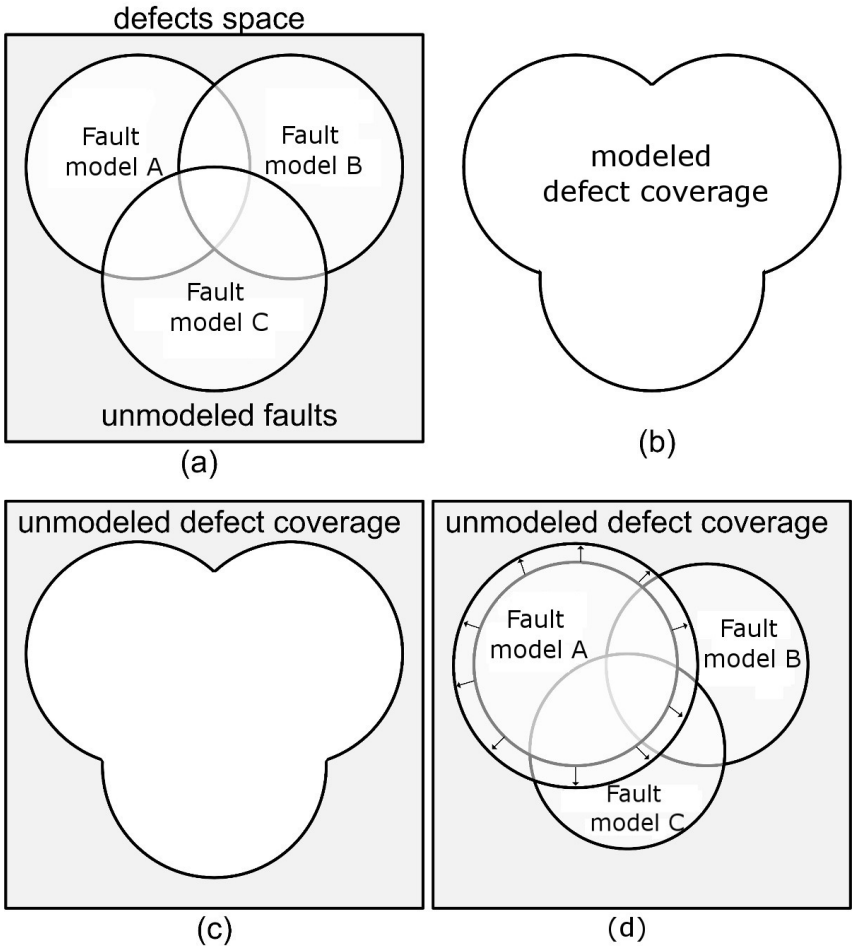


Figure 1.10: Unmodeled defect coverage

1.6 Modeling of ... Unmodeled Defects

Although, improving fault coverage for structural tests can be an easy way to improve the quality of a testing process and *yield*, we should not ignore practical limitations that forbid a complete defect screening process. For example, it is not feasible to test against all known fault models, and even if it was, there still be defects.

Figure 1.10 depicts a set representation of defect coverage. In Figure 1.10a the space of all possible defects is shown as a large square. Three subsets are highlighted which contain defects that can be covered by the three theoretical fault models: A, B, C. In Figure 1.10b the set of modeled defects covered is presented. Finally, Figure 1.10c illustrates the set of defects that are not covered in light grey color. Defects that do not belong to the set of modeled defects are called *unmodeled defects*. It follows from this Figure that a test to cover a fault (some defects) may cover a defect that is also covered from another fault model or a defect that it was not even modeled. This property of a test is called **unmodeled defect coverage**.

Later, contributions on the concept of enhancing the quality of structural testing techniques by increasing the unmodeled defect coverage of their generated tests are presented.

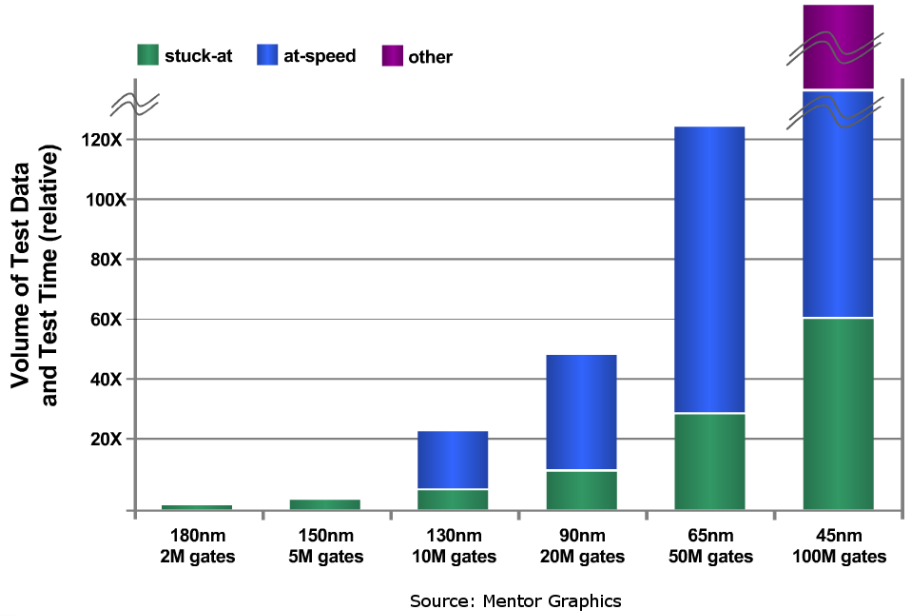


Figure 1.11: Explosion of test data volume.

1.7 Basic Test Cost Factors

Beside the quality enhancement of structural testing methods that indirectly reduce test cost, new testing techniques should also consider the classical test cost factors. Since, the market demands require faster and denser ICs over the years, these basic cost factors have been stressed by the new dense and complex integration technologies. These factors are the cost and the limitations of ATEs, the time required to perform testing and unpredictable human factors.

Equipment Cost: The major contributor to the cost of testing is the cost of the ATEs. As devices continue to grow more complex, the test capabilities need to be constantly improved. Also, the speed of the ATE is required to increase because constant device scaling since the mid-1980s has pushed the device speeds significantly higher. Manufacturers are constantly looking for low-cost ATEs that can reliably test complex and high-speed device during high-volume production testing.

ATE Limitations: The ever-increasing number of gates results in an ever-increasing number of test patterns. The 2007 ITRS test report [2] predicted that the test-data volume for integrated circuits will be as much as 38 times higher and the test-application time will be about 17 times larger in 2015 than it was in 2007. Figure 1.11 captures this trend. While test data increase, the previous ATEs generations cannot cope with the demanding memory and CUT/ATE communication requirements. All these result to the following ATE limitations:

- Bandwidth limitations between Workstation/ATE: the test patterns need to be uploaded from the workstation to the ATE memory. Limited data bandwidth between workstation/ATE may stall this process from several tens of minutes to hours [55].

While ATE remains idle, the cost of test increases [165].

- ATE memory limitations: New generation ATEs with the required memory may not be available (or may be very expensive). Test data are truncated to fit the memory, resulting to quality degradation [55].
- Bandwidth limitations between ATE/CUT: To apply the test patterns at the CUT, they need to be trasnfered from the ATE (where they are stored) to the CUT. Additionally, the responses must pass from the CUT to the ATE in order to be analyzed. The bandwidth of the channels between ATE/CUT are limited. The above process may increase dramatically test time and consequencly test cost [165].

Production Test Time: Apart from the cost of ATEs, large test application time is a major factor for increased test costs. Typically, test time for wireless devices ranges from a few seconds to a few minutes. During production, when millions of devices are tested, even such apparently small test times can create a bottleneck. Suppose, for example, that a device test time required during production is 60 seconds. Therefore, the number of devices that can be tested is 1440 per day ($= 24 \times 3600/60$). Considering that 10 ATEs are used, then to release a million devices to the market requires 70 days. This clearly shows that a small reduction in test time can increase the throughput significantly. Therefore, there is a constant need in the test community to reduce production test time. Production test time is affected by many factors, such as the time needed to design the tests (by the test engineer), the time required for the equipment (handlers and probers) to prepare the environment for the test, the Test Application Time (TAT), which is the time needed to excite the CUT with the test stimuli and get the responses.

Human factor: Additional costs come from engineering errors or other human factors. For example, an improperly designed IC or a bug in the test program can significantly increase the time required to release a product. This can cause the manufacturer to loose significant market share for that product. Such factors can be fatal for small businesses, and the success of the manufacturer relies heavily on the test process.

In general, all the above limitations (except human factor) stem from the same reason: the **increasing amount of test data** (stimulus and response data) [163, 165]. The test cost solution to this problem is to often upgrade ATEs, but this solution is very impractical and extremely costly to be adopted by companies. The necessity to overpass this dead end, decrease test cost and handle the increased complexity of new integration technologies, motivated the consideration of testing during the early life of manufacturing ICs: the design. It was the dawn of Design for Testability (DFT).

1.8 Design for Testability

Test engineers usually have to construct test vectors after the design is completed. This invariably requires a substantial amount of time and effort that could be avoided if testing was considered early in the design flow to make the design more testable. As a result,

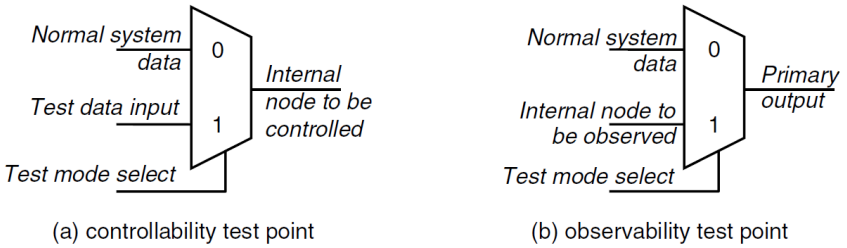


Figure 1.12: DFT test point

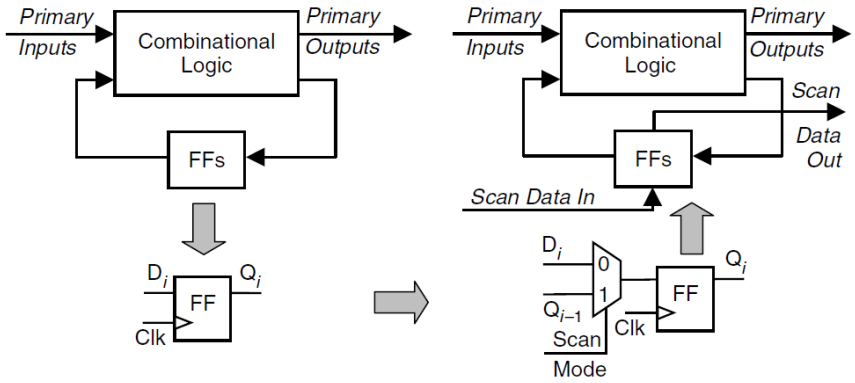


Figure 1.13: Adding test points at a sequential circuit.

integration of design and test, referred to as design for testability (DFT), was proposed in the 1970s.

To test the structure of ICs, we need to control and observe logic values of internal nodes. Unfortunately, some nodes in sequential circuits can be very difficult to control and observe; for example, activity on the most significant bit of an n-bit counter can only be observed after 2^{n-1} clock cycles. Testability measures of controllability and/or observability were first defined in the 1970s [48] to help find those parts of a digital circuit that will be most difficult to test and to assist in test pattern generation for fault detection. Many DFT techniques have been proposed since that time [101]. DFT techniques generally fall into one of the following three categories: (1) ad-hoc DFT techniques, (2) scan design, or (3) built-in self-test (BIST).

1.8.1 Ad-hoc DFT

Ad-hoc methods were the first DFT techniques introduced in the 1970s. The goal was to target only those portions of the circuit that would be difficult to test and to add circuitry to improve the controllability or observability. Ad-hoc techniques typically use test point insertion to access internal nodes directly. An example of a test point is a multiplexer inserted to control or observe an internal node, as illustrated in Figure 1.12.

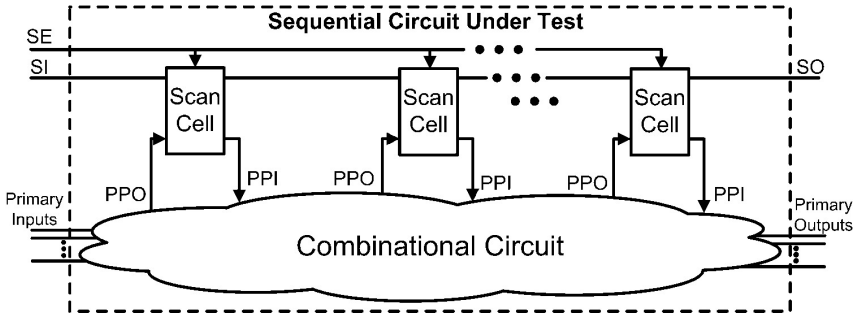


Figure 1.14: A typical scan design.

1.8.2 Scan Design

In scan design [39] external access is provided at the storage elements of ICs in order to increase their controllability and observability. The modified storage elements are commonly referred to as *scan cells*. Once the capability of controlling and observing the internal states of a design is added, the problem of testing a sequential circuit is transformed into a problem of testing combinational logic, which is an easier task. Figure 1.13 presents the re-designing of D flip-flops for a sequential circuit to scan cells. Widely used scan cell designs are: muxed-D scan cell, clocked-scan cell [101], and level-sensitive scan design (LSSD) cell [35, 39].

In order to save I/O pins, the scan cells are connected into multiple shift registers, called *scan chains*. A typical scan design, with a single scan chain, is presented in Figure 1.14. Scan design accomplishes this task by replacing all selected storage elements with scan cells, each having one additional scan input (SI) port and one shared/additional scan output (SO) port. By connecting the SO port of one scan cell to the SI port of the next scan cell a scan chain is created. This way a sequential CUT is transformed into a combinational circuit. The control points of the combinational circuit are called pseudorandom primary inputs (PPIs) and the observable points are called pseudorandom primary outputs (PPOs). The selection between operations of a typical scan design (scan or normal operation modes) is controlled by a scan enable (SE) signal. Testing based on scan design is called *scan testing* and is conducted as follows:

- During the scan mode (when SE=‘1’), the scan chain is used to shift in (or scan in) a test vector to be applied to the combinational logic.
- During one clock cycle in the system mode (when SE=‘0’ and it is also called capture mode) of operation, the test vector is applied to the combinational logic and the output responses are clocked into the flip-flops.
- Also in scan mode, the scan chain is used to shift out (or scan out) the combinational’s logic output response to the test vector while shifting in the next test vector to be applied.

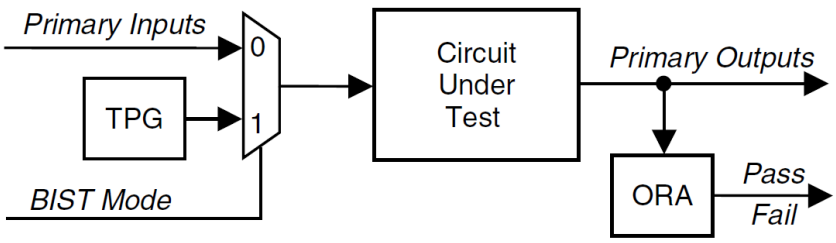


Figure 1.15: BIST scheme.

1.8.3 Built-In Self-Test

Built-in self-test (BIST) was proposed around the 80s [116, 144, 145]. The basic idea is to integrate a test-pattern generator (TPG) and an output response analyzer (ORA) together with the CUT in order to perform testing internal, as illustrated in Figure 1.15, without any need of external tester. Since an external tester is not required, BIST reduces considerably test cost. However, there are many challenges in making a design BIST-ready: efficient logic BIST structures must be integrated that should achieve high test quality. However, there are different efficient BIST architectures [46, 47, 161] based on the nature of the logic inside the CUT. A constant problem remains the automation of the BIST-architecture design with the ICs design without impacting the overall product schedule. In [55] it was shown that with automation of the designing process and with constant upgrade of this automation, BIST can become viable for large industrial designs.

1.8.4 Test Resource Partitioning

Test Resource Partitioning (TRP) is a DFT approach for highly dense ICs that decreases test cost by easing the burden of outdated ATE systems. TRP focuses on transferring test functionalities from the ATE towards the CUT. The basic idea is to compress large volumes of test data to small test sets that fit in the memory of an ATE and they are based on a hybrid scan design/built-in-self-test (BIST) approach. The test data are stored on the ATE in a compressed form downloaded at the CUT where they are decompressed and applied. After their application, the responses are compressed on the CUT before they are sent back to the ATE in compressed form.

Figure 1.16 presents the general TRP architecture. The compressed form of test vectors stored into the ATE are called test data. The size of test data, which is the amount of memory required to store the test data on ATE, is called Test Data Volume (TDV). During testing, the test data are transferred through the low-bandwidth ATE/CUT channels to the CUT where they are decompressed on-chip by embedded decompression architectures. The test vectors are shifted into the scan chains setting the CUT into a predetermined internal state. Afterwards, the CUT is let to operate normally and the response is captured into the scan chains. Then, the procedure starts over, but now with the decompression of the next vector. During the shift-in of the next vector, the responses of the previous vec-

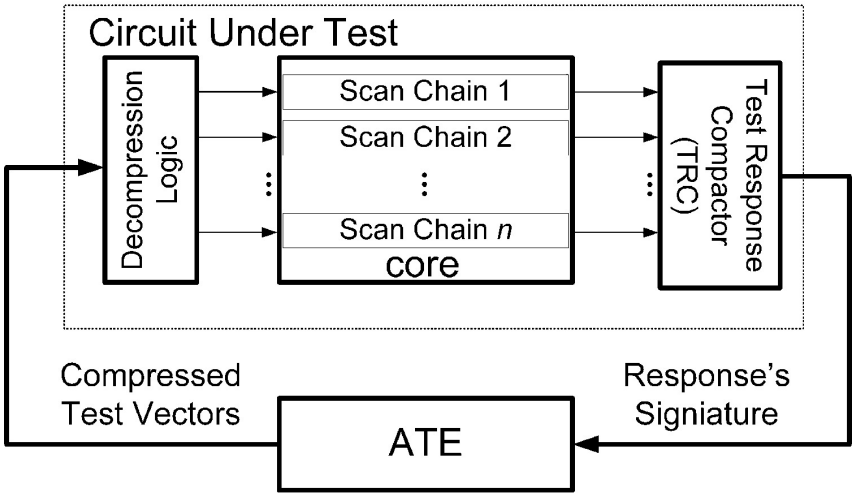


Figure 1.16: Test Resource Partitioning Architecture

tor already contained into the scan chains are shift-out towards the TRC where a unique signature for them is created. Signatures are shifted out towards the ATE, where they are compared against fault-free signatures.

The amount of ATE's participation in the testing procedure is the key of categorizing a compression TRP technique. Thus, there are two categories of TRP techniques:

- Test Set Embedding (TSE): long pseudorandom sequences are generated on-chip with minimum interaction with the ATE. TSE techniques have small ATE's memory requirements but they impose large hardware overhead on the embedded decompression architectures and long TAT.
- Test Data Compression (TDC): compression codes, such as statistical, the Run-length, the Golomb, the Frequency-Directed Run-length (FDR) coding, the Huffman code, are utilized to compress the test data. These methods occupy relatively small ATE's memory space, which however is higher than that of TSE techniques, and they also require more frequent usage of the ATE/CUT channels. On the other hand the hardware overhead of the decompressors is very low and the TAT is very short.

TRP techniques are further categorized based on the nature of both the compression code and decompression logic used. There are TRP techniques based on:

- Compression codes (code-based): the Golomb [56], the Huffman [57], the Run-length etc [11, 18, 20, 21, 49, 60–62, 71–74, 74, 87, 90, 96, 111, 126, 139, 149, 150, 177, 189, 190, 192].
- Linear decompressors (linear-based): Linear Feedback Shift Registers (LFSRs), Ring generators etc [12, 54, 67, 77, 79, 81, 82, 102, 138, 155, 166].
- Broadcast schemes: pseudorandom values broadcasted simultaneously into the scan chains [51, 86, 103, 112, 133, 140, 142, 148, 170, 171].

Commercial tools for test compression are also available [10, 78, 123]. The most widely TRP techniques are based on linear decompressors.

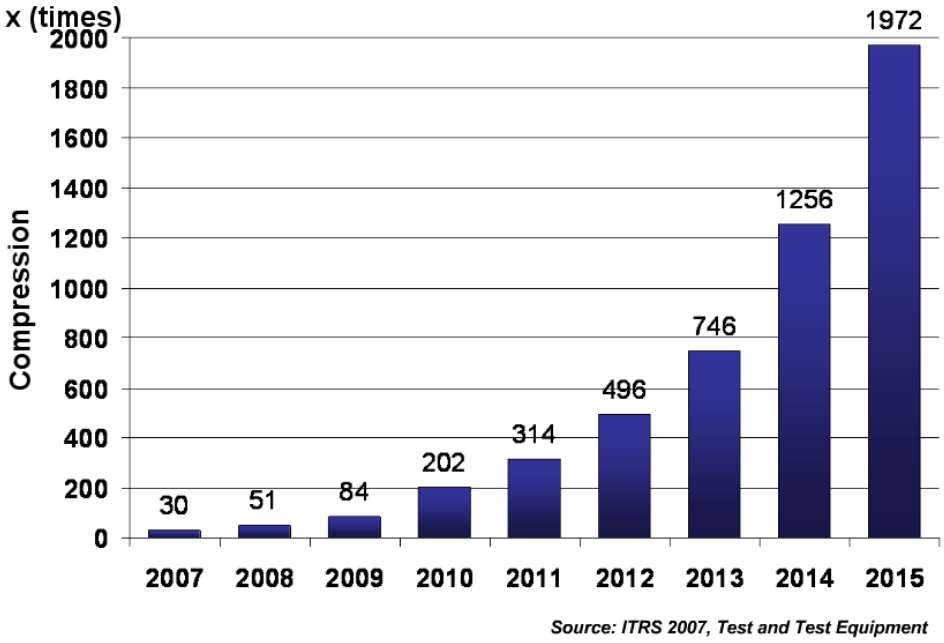


Figure 1.17: ITRS'07 compression prediction requirements

Figure 1.17 depicts the trajectory in compression requirements of contemporary TRP methods in order to cope with the upcoming explosion of test data. The y-axis, which is the compression requirements between 2007 and 2015, shows the ratio of uncompressed test data to compressed test data. Since, the existing TRP techniques cannot achieve those compression requirements, new TRP techniques are needed.

The efficiency of TRP's compression is achieved so far, by exploiting only one out of two properties of the test vectors. Specifically, the test vectors consist of logic values '0', '1' as well as *undefined* values 'x'es. The undefined values can be any logic value ('0' or '1') without affecting the stimulation of the fault that the test vector was generated for (undefined values 'x'es are also refered as *don't cares* or *unspecified* values, and logic values '0' and '1' are refered as *defined* or *specified* values in the literature). When a vector contains undefined values it is called *test cube*. The ratio between specified and unspecified values of a test set is called *fill rate*. The two properties that TRP techniques exploit to offer compression are the low fill rate of the test sets (the large amount of unspecified values 'x'es) and the correlation of the specified values that stem from CUT's structural correlation [158]. Linear-based methods exploit the unspecified values, while code-based techniques exploit the correlations. There is not such technique, so far, that exploits both these properties for compression [158].
Later, a code-based compression method is presented that exploits both the low fill rates of test sets and the correlation in the test cubes.

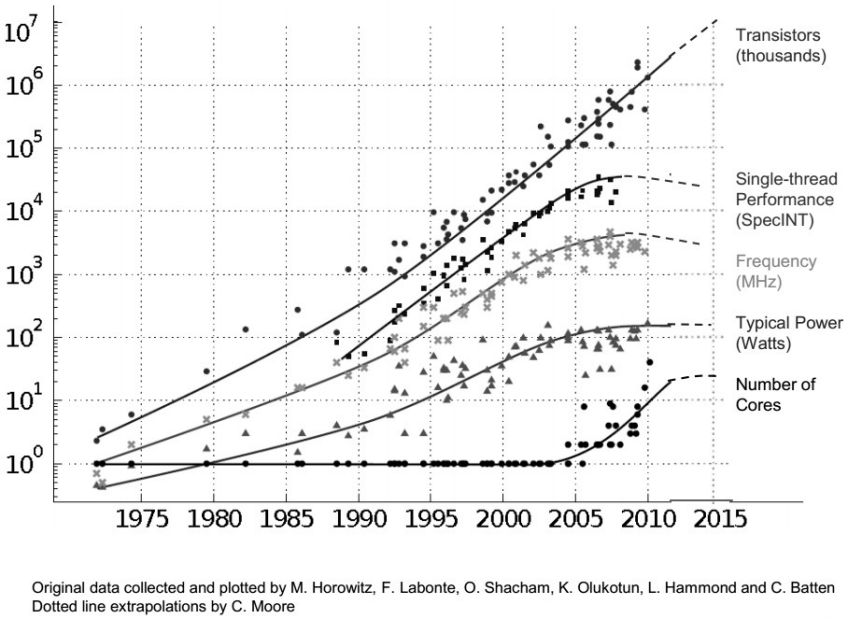


Figure 1.18: Moore's Law in Respect to Transistors Number, Single Thread Performance, Frequency, Power and Number of Cores

1.9 Additional Test Challenges

1.9.1 The post-Dennard Era: Low-power Testing

During the last years we have witness a tremendous change in the industry's target group, since individuals, and not corporations and government agencies, are nowadays the main consumers of semiconductors. During this era the demands for testable low-power mobile devices have been increased dramatically. Nowadays we can find mobile Internet devices (MIDs), personal digital assistants (PDAs) and smartphones which are mobile multimedia-capable devices with wireless Internet access: "supercomputers of older eras in consumers' "pockets". The manufacturing of these portable computing devices became reality because of the huge density and speed of contemporary ICs. During these years we also witnessed the transformation of testing ICs to *low-power testing* ICs.

Density and speed of ICs have increased exponentially for several decades, following a trend described by Moore's Law. The original version of Moore's law states that transistor density doubles every 18-24 months. Although Moore's law still holds true, Dennard scaling [36] does not. Dennard scaling is the observation that as transistors get smaller, the power used by each transistor shrinks. Unfortunately, the shrinking factor (although it still holds true) is not fast enough to cope with the increase in the number of integrated transistors and as a result, the overall circuits' power demands have been increased. Consequently, in the post-Dennard era contemporary ICs with billions of transistors are **underclocked** because of two reasons: a) to dissipate less power in order to extend the battery life of mobile devices and b) to dissipate the power without violating the power dissipation limits that result to overheating.

In the past the higher integration level from era to era was followed by an increase in the operational frequency and so the TAT per transistor was decreasing. As shown in Figure 1.18 the operational frequency of contemporary ICs tends to saturate the last years breaking the frequency prosperity. This seems an inevitable effect as long as the material limits have been reached and there are not any other material level technologies to fill this gap. Manufacturers no longer provide a processors' power consumption characteristic but they provide the Thermal Design Power (TDP) which is the maximum amount of power that can be dissipated. Contremporary processors (like Ivy Bridge which is Intel's 22nm series) exhibit TDP at the range of [35 - 130] Watts. The TDP limitation on the amount of power that a chip can dissipate introduced additional two testing obstacles:

- The underclocked circuits require more testing time compared to overclocked circuits. Although, more tests are needed to test higher density technologies compared to previous technologies, the tests cannot be conducted faster anymore.
- Traditional testing techniques decrease test cost by concurrently targeting as many defects as possible, leading thus to elevated test power consumption, which can be several times higher than that in functional mode [8]. The TDP limits forbid that because the tested devices might be harmed or tests may fail their purpose.

Power unaware testing techniques cause the circuit to consume much more power in test mode than in normal mode [16, 44, 59, 110, 118, 137, 198]. It was shown in [198] that test power can be more than twice the power consumed in normal functional mode. Specifically, some reasons for this gap between normal's and test's mode power consumption include:

- ATPG tools tend to generate test patterns with a high toggle rate in order to reduce pattern count and thus test application time. Therefore, the node switching activity of the device in test mode is often several times higher than that in normal mode.
- Parallel testing is often used to reduce test application time, particularly for testing MCSoCs devices. This parallelism inevitably increases power dissipation during test.
- Circuitry inserted in the circuit to alleviate test issues is often idle during normal operation but may be intensively used in test mode. This surplus of active elements during test also induces an increase of power dissipation.
- Elevated test power can come from the lack of correlation between consecutive test patterns, while the correlation between successive functional input vectors applied to a given circuit during normal operation is generally very high [173].

As a result old test practices are deprecated and low power testing techniques are required. The new techniques ought to be faster than power-unaware testing techniques and on the same time to handle the power dissipation limits. *Later, contributions on algorithmic ATPG techniques as well as on TRP techniques that achieve to reduce the power demands for structural testing are proposed.*

1.9.2 Multi-Core Systems-on-Chips and IP Cores

The sustaining of Moore's law growth is essential not only because it offers prosperity on almost every aspect of human life but also because it provides payback of the huge capital investment of semiconductor's industry (an industry with starting capital that overpasses 3 billion dollars). To fill the processing gap of the no-longer-increased operating frequency, the industry has counter-proposed Multi-Core Systems-on-Chips (MCSoCs). They are based on exploiting the concurrent processing in order to offer faster systems.

However, MCSoCs require specialized assemble processes that increase test cost. There is a general agreement with the *rule of ten*, which says that the cost of detecting a faulty IC increases by an order of magnitude as we move through each stage of manufacturing, from device level to board level to system level and finally to system operation in-the-field. Nevertheless, MCSoCs brought not only challenges but also oportunities. Techniques such as parallel and multi-site testing [52] have been introduced. These techniques exploit capabilities of new generation DFT-aware test equipments [14, 68] for *test resources sharing*. In order for this technology to be efficient, DFT methodologies with *reduced pin-count interface* between ATE/CUT are required.

Intellectual Property (IP) cores that usually reside within MCSoCs complicate further testing. There are two main types of components within an MCSoC: the cores and the user defined logic (UDL). A core is a pre-designed, pre-verified silicon circuit block that can be used in building a larger or more complex application on a semiconductor chip. Cores can perform a wide range of functions (e.g., digital signal processors, RISC processors, or DRAMs) and can be found in a number of technologies (e.g., complementary metal-oxide-silicon (CMOS) logic, DRAM and analog circuits). Furthermore, the more complex cores come in hierarchical compositions (i.e., complex cores comprise a number of simple cores). Often these cores are products of technology, software, and know-how that are subject to patents and copyrights. Hence, a core block represents IP that the core builder licenses to the core user. Therefore, the core user is not always entitled to make changes to the core and is forced to reuse it as is (as a black box), being knowledgeable only about the cores functionality, however, not about the implementation details. In addition, while ICs are delivered to the customer in a manufactured and tested form, cores are delivered in a range of hardware description levels (soft, firm, and hard). These two fundamental differences influence not only the design of the MCSoCs, but also their testing.

Usually, IP cores are accompanied by pre-computed and pre-compacted test sets i.e. test sets with high fill rate. The compression efficiency of linear-based TRP methods drops drammatically when they are applied on test sets with high fill rate, because there are not many undefined values. On the other hand, although code-based compression methods are more efficient at compressing test sets with high fill rate, there are not any industry tools that supports them, because their compression efficiency on the test sets with low fill is moderate.

Later, contributions on TRP techniques for testing IP cores are proposed. A novel TRP method is presented that is based on code-based TRPs and requires low pin-count

Table 1.1: Contributions and Dissertation Structure

	Algorithmic	***Test Resouce Partitioning***			***Goals***		
		Linear-based		**Code-based**	**Power**		**quality**
	X-Filling	**Static**	**Dynamic**	**Huffman**	**shift**	**capture**	
Chapter 3		✓					
Chapter 4			✓		✓		
Chapter 5	✓				✓	✓	✓
Chapter 6		✓	✓				✓
Chapter 7			✓	✓	✓		✓

ATE/CUT interface. In addition, a novel linear-based TSE architecture is proposed that almost eliminates the useless parts from the long test sequences of linear-based TSE techniques.

1.10 Contributions & Dissertation Structure

Providing a low-cost test solution for MCSoCs does not only require the understanding of the test cost factors, but it also requires to understand the implications and limitations of previous approaches which addressed these factors. Therefore, a comprehensive analysis of previous work is given in Chapter 2, which further motivates the usage of test resource partitioning, the low-power testing, and the probabilistic fault models to enhance test quality. The next Chapters present novel contributions in these areas. The contributions of this dissertation (also briefly illustrated in Table 1.1) are discussed below:

Chapter 3: The method presented in Chapter 3 contributes to test set embedding architectures. Test set embedding techniques are the TRP architectures with the best compression in the literature and they could be used for BIST applications. However, their long test sequences renders them impractical. The method presented in this Chapter targets the test sequence length reduction of test set embedding techniques. A trade-off between test sequence length of linear-based test set embedding techniques and hardware overhead is presented. This contribution is important at the post-Dennard era, where the hardware overhead is not the basic bottleneck, and can be traded for test cost reduction.

Chapter 4: The method presented in Chapter 4 contributes to the area of low power decompression of linear-based decompressors. A new low-cost, test-set-independent linear equations generation scheme is presented for deterministic test data compression, which can be combined with any linear decompressor for reducing the shift power during testing. Contrary to existing low power linear-based techniques that required additional test data, the low power property of the proposed method is pseudorandomly handled by an embedded control unit and it does not require data to be stored on the tester. Extensive experiments with the state-of-the art linear decompressors show that the proposed method offers reduced test power, test sequence length and test data volume at the same time, with very small area requirements.

Chapter 5: The techniques that target the enhancement of the unmodeled defect coverage of test vectors exploit the unspecified values ('x'). However, this target contradicts the main objective of the X-filling techniques that decrease power consumption of structural testing. In addition, the low power X-filling techniques introduce high correlation in the test vectors which adversely affects the unmodeled defect coverage. As a result, to optimize different characteristics on a single method requires sophisticated approaches. Another major limitation of the quality enhancement metrics used in the literature is that they evaluate each test vector for either timing-independent or timing-dependent defects. Chapter 5 proposes a unified X-filling method that targets both power reduction (shift and capture) and high-quality test generation. To enhance the unmodeled defect coverage a new output deviations evaluation metric is proposed that maximizes the effectiveness of the selected patterns with respect to the detection of both timing-related and timing-independent defects at the same time. The shift power of the new method is as low as the shift power of the Fill-Adjacent X-filling technique and its unmodeled defect coverage is similar to that of Random X-filling. This simple method, also (besides being a contribution), serves as a simple application example to familiarize the reader with the idea of targeting unmodeled defects using output deviations.
Chapter 6: Exploiting the unspecified values ('x') to increase the unmodeled defect coverage of the test vectors, contradicts with the main objective of linear-based TRP methods, which is to exploit the unspecified values for improving the test data volume. The contributions of Chapter 6 are on high quality enhancements of the test vectors generated by linear-based test data compression. It provides efficient candidates generation algorithms for the unmodeled defect coverage enhancement of the most well-known linear decompressors and reseeding techniques, including the state-of-the-art ring generators.
Chapter 7: This Chapter presents a technique that targets all the aforementioned goals simultaneously for linear-based and code-based TRP architectures: high-quality generated vectors with increase unmodeled defect coverage, low-power dissipation, and compression maximization. Specifically, in Chapter 7 a novel compression method and a low-cost decompression architecture is presented that can be applied on a) linear-based and b) symbol-based decompressors in order to reduce shift power and also increase the unmodeled defect coverage of the generated patterns. Moreover, the application of the scheme on code-based decompressors revealed an interesting property: the new decompressor exploits both the low fill rate and the correlations in the test cubes offering better compression than any existing compression technique. Therefore, it can be applied efficiently on test sets of both IP and non-IP cores, that usually co-exist in MCSoCs. This property together with its low pin-count ATE/CUT interface enable the sharing of the decompressor among different types of cores. As a result it can be used for efficient multi-site testing approaches.

Finally, Chapter 8 summarizes the presented work and concludes this dissertation. The contributions outlined in Chapters 3, 4, 5, 6 and 7 resulted in original work published in [8, 70, 75, 151–157] and itemized in the end of the dissertation.

Chapter 2

Background

2.1 Fault Models

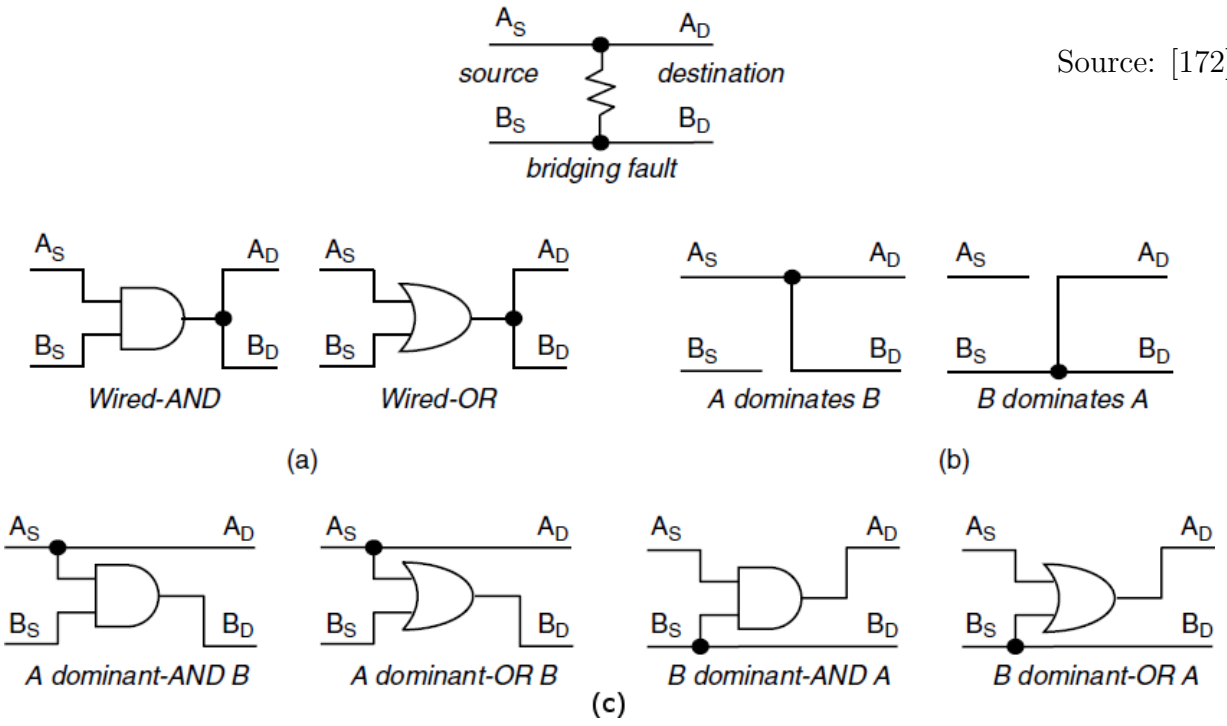
Because of the diversity of VLSI defects, it is difficult to generate tests for real defects. Fault models are necessary for generating and evaluating a set of test vectors. Generally, a good fault model should satisfy two criteria: (1) it should accurately reflect the behavior of defects, and (2) it should be computationally efficient. Many fault models have been proposed [4], but, unfortunately, no single fault model accurately reflects the behavior of all possible defects that can occur. As a result, a combination of different fault models is often used for manufacturing testing.

2.1.1 Stuck-at Faults

The stuck-at fault is a logical (logic-level) fault model that has been used successfully for decades. A stuck-at fault affects the state of logic signals on lines in a logic circuit, including primary inputs, primary outputs, internal gate inputs and outputs, fanout stems (sources), and fanout branches. A stuck-at fault transforms the correct value on the faulty signal line to appear to be stuck at a constant logic value, either a logic 0 or a logic 1, referred to as stuck-at-0 or stuck-at-1, respectively.

2.1.2 Transistor Faults: Stuck-open and Stuck-short

At the switch level, a transistor can be stuck-open or stuck-short, also referred to as stuck-off or stuck-on, respectively. An example of stuck-open fault is a transistor isolated from



Source: [172]

Figure 2.1: Bridging fault models

ground (V_{SS}) which causes the transistor to keep its previous state. In order to detect this fault in a CMOS combinational circuit a sequence of two vectors for detection rather than a single test vector for a stuck-at fault is required.

Stuck-short faults, on the other hand, will produce a conducting path between V_{DD} and V_{SS}. For example, if a transistor is stuck-short, there will be a conducting path between V_{DD} and V_{SS}. This may create a voltage divider at a node that may or may not be interpreted as an incorrect logic level by the gate inputs driven by the gate with the transistor fault.

Stuck-short transistor faults may be detected by monitoring the power supply current during steady state, referred to as I_{ddq}. This technique of monitoring the steady-state power supply current I_{ddq} to detect transistor stuck-short faults is referred to as IDDQ testing.

2.1.3 Wire Open and Short Faults

Defects in VLSI devices can include opens and shorts in the wires that interconnect the transistors of the circuit. Opens in wires that interconnect transistors' gates tend to behave like transistor stuck-open faults. On the other hand, opens in wires interconnecting logic gates tend to behave like stuck-at faults. Therefore, a set of test vectors that provide high stuck-at fault coverage will also detect open faults; however, a resistive open does not behave the same as a transistor or stuck-at fault but instead affects the propagation delay of the signal path [172].

Wired Bridging Faults

A short between two elements is commonly referred to as a bridging fault. These elements can be transistor terminals or connections between transistors and gates. The case of an element being shorted to power (V_{DD}) or ground (V_{SS}) is equivalent to the stuck-at fault; however, when two signal wires are shorted together, a logical fault model is required. A logical fault model is more flexible because it can be easily fault-simulated using stuck-at fault simulators with the proper changes on the simulated netlist (circuits' logical level structure). According to the first bridging fault model, the logic value of the shorted nets modeled as a logical AND or OR of the logic values on the shorted wires. This model is referred to as the wired-AND/wired-OR bridging fault model. The wired-AND bridging fault means the signal net formed by the two shorted lines will take on a logic 0 if either shorted line is sourcing a logic 0, while the wired-OR bridging fault means the signal net will take on a logic 1 if either of the two lines is sourcing a logic 1. Therefore, this type of bridging fault can be modeled with an additional AND or OR gate, as illustrated in Figure 2.1a, where A_S and B_S denote the sources for the two shorted signal nets and A_D and B_D denote the destinations for the two nets. This new structure can then be combined with a stuck-at fault simulator to provide fault coverage.

Dominant Bridging Faults

The wired-AND/wired-OR bridging fault model was originally developed for bipolar VLSI and does not accurately reflect the behavior of bridging faults typically found in CMOS devices. The dominant bridging fault model was proposed for CMOS VLSI where one driver is assumed to dominate the logic value on the two shorted nets. Two fault types are normally evaluated per fault site, where each driver is allowed to dominate the logic value on the shorted signal net (see Figure 2.1b). The dominant bridging fault model is more difficult to detect because the faulty behavior can only be observed on the dominated net, as opposed to both nets in the case of the wired-AND/wired-OR bridging fault model. However, it has been shown that a set of test vectors that detects all dominant bridging faults is also guaranteed to detect all wired-AND and wired-OR bridging faults [172].

The dominant bridging fault model does not accurately reflect the behavior of a resistive short in some cases. A recent bridging fault model has been proposed based on the behavior of resistive shorts observed in some CMOS VLSI devices [41]. In this fault model, referred to as the dominant-AND/dominant-OR bridging fault, one driver dominates the logic value of the shorted nets but only for a given logic value (see Figure 2.1c). While there are four fault types to evaluate for this fault model, as opposed to only two for the dominant and wired-AND/wired- OR models, a set of test vectors that detect all four dominant-AND/dominant-OR bridging faults will also detect all dominant and wired-AND/wired-OR bridging faults at that fault site.

Bridging faults commonly occur in practice and can be detected by IDDQ testing. It has also been shown that many bridging faults are detected by a set of test vectors that obtains high stuck-at fault coverage, particularly with N-detect single stuck-at fault

test vectors. In the presence of a bridging fault, a combinational logic circuit can have a feedback path and behave like a sequential logic circuit, making the testing problem more complicated. Another complication in test generation for bridging faults is the number of possible fault sites versus the number of realistic fault sites. While there are many signal nets in a VLSI circuit, it is impractical to evaluate detection of bridging faults between any possible pair of nets; for example, a circuit with N signal nets would have $N - choose - 2 = N \times (N-1)/2$ possible fault sites, but a bridging fault between two nets on opposite sides of the device may not be possible. One solution to this problem is to extract likely bridging fault sites from the physical design after physical layout.

Bridging Fault Coverage Estimation Metrics

The bridging fault coverage cannot be accurately measured since the set of bridging faults is huge and not all of them are equally possible. Bridging fault coverage can only be accurately (and reasonable) estimated when layout information is available (after the routing of the interconnections during the final steps of a designing process). Using layout information, it is feasible to isolate the most possible pairs and drop the complexity of estimating accurately the bridging fault coverage. However, this approach is infeasible during test generation, because that is taking place during the early stage of designing process, when layout information is not yet available. As a result various metrics and methods have been proposed to measure the bridging fault coverage during the early stages of a designing process. Although these metrics do not offer an accurate estimation of the bridging fault coverage, they are very useful for comparison purposes. In this dissertation we used the following two approaches:

- **The Bridging Coverage Estimation (BCE^+) metric:** In [178] a metric has been proposed for evaluation of tests in terms of their achieved bridging-fault coverage. That BCE^+ metric is:

$$BCE^+ = \sum_{i=1}^{n} \frac{f_i^{sa-v}}{|F|} \cdot \left[\sum_{j=1}^{|S|} \frac{1}{|S|} \left(1 - (1 - p_{j,v})^i\right) \right]$$

  where $v = 0, 1$. The parameter f_i^{sa-v} refers to the number of stuck-at-0 faults (for $v = 0$) and stuck-at-1 faults (for $v = 1$) that are detected i times by the test vectors (n is the maximum number of detections for any stuck-at fault). $|S|$ is the number of circuit lines, $|F|$ is the total number of stuck-at faults and $p_{j,v}$ is the probability of signal j to receive the logic value 0 (for $v = 0$) and 1 (for $v = 1$). As noted in [178], BCE^+ is not very accurate for estimating the real bridging fault coverage of a method, but it is very useful for comparing two different methods (the method with the highest value of BCE^+ is deemed to be more effective for defect screening).
- **Random bridging fault coverage:** Another approach to compare the bridging fault coverage between two test sets is to use fault simulation against a set of random bridging faults e.g. n pairs of lines can be selected randomly from the CUT. For

each pair, four bridging faults are simulated by considering the four dominant-AND/dominant-OR bridging faults $4 \cdot n$ faults are finally simulated. The larger the n the better the estimation on bridging fault coverage.

2.1.4 Delay testing and Delay Fault Models

Motivation for delay testing

Fault-free operation of a logic circuit requires performing the logic function correctly and within a specified time limit. A delay fault causes excessive delay along a path such that the total propagation delay falls outside a specified time limit. Delay faults have become more prevalent with decreasing feature sizes. In the early days of VLSI technologies, most defects affecting the performance could be detected using tests for gross delay defects [168]. Later, the aggressive timing requirements of high-speed designs have introduced the need to test smaller timing defects and distributed faults caused by statistical process variations [15, 97, 113, 117, 119, 120, 191, 193, 194]. Moreover, the increase of the circuit size has resulted in fault models that can detect distributed defects localized to a certain area of the chip [95, 143]. Finally, with the introduction of VDSM technologies, noise effects are becoming significant contributors to timing failures [16] and further adaptations of the fault models and testing strategies are required.

The objective of delay testing is to detect timing defects and ensure that the design meets the desired performance specifications. The need for delay testing has evolved from a common problem faced by the semiconductor industry: designs that function properly at low clock frequencies might fail at the desired operational speed. To detect such faults, functional tests created for design verification are applied at system operational speed to screen out parts with delay defects. However, applying functional tests is becoming very expensive, given the need for a high-speed tester to apply such tests. This approach is still used extensively for high performance parts, such as microprocessors and digital signal processors (DSPs) for which the functional tests can be loaded into on-chip caches and then applied with a low-cost tester. Another problem with using functional tests is the lack of assurance for high test quality. Several industrial experiments (e.g., [100]) have shown that tests not specifically targeting delay faults have limited success in detecting timing defects. The above-mentioned problems can be alleviated by using structurally based generated tests tests that target specific delay fault models and which can be applied through design for testability (DFT) structures using lower-cost testers. For the rest of the section, our discussion is focused on such structurally based delay testing approaches.

To observe delay defects, it is necessary to create and propagate transitions in the circuit running at-speed (at its specified operating frequency). Creating transitions requires application of a vector pair, $V =< v_1, v_2 >$, at the inputs of the combinational part of the circuit. The first vector initializes the relevant internal signals to desired initial logic values, while the second vector causes the desired transitions and sensitizes the transition from the target fault site to an output.

The most popular delay fault models are the transition-delay fault model, the gate-delay fault model, and the path-delay fault model. It is assumed that in the nominal design each gate has a given fall (rise) delay from each input to the output pin. Also, the interconnects are assumed to have given rise (fall) delays. Because the gate pin-to-pin delays and the interconnect delays can be combined together, the term "gate delay" will be used to denote this sum. Transition and gate-delay models are used for representing delay defects lumped at gates, while the path-delay model addresses defects that are distributed over several gates. The transition-delay fault model is briefly presented below.

Transition-delay Fault Model

The transition-delay fault model [25, 88, 168] assumes that the delay fault affects only one gate in the circuit. There are two transition faults associated with each gate: a slow-to-rise fault and a slow-to-fall fault. It is assumed that in the fault-free circuit, each gate has some nominal delay. Delay faults result in an increase of this delay. Under the transition fault model, the extra delay caused by the fault is assumed to be large enough to prevent the transition from reaching any primary output at the time of observation. In other words, the delay fault can be observed independent of whether the transition propagates through a long or a short path to any primary output; therefore, this model is also referred to as the gross-delay fault model. To detect a transition fault in a combinational circuit it is necessary to apply two input vectors, $V =< v_1, v_2 >$. The first vector, v_1, initializes the circuit, while the second vector, v_2, activates the fault and propagates its effect to some primary output. Vector v_2 can be found using stuck-at fault test generation tools. For example, for testing a slow-to-rise transition, the first vector initializes the fault site to 0, and the second vector is a test for a stuck-at-0 fault at the fault site. A transition fault is considered detected if a transition occurs at the fault site and a sensitized path extends from the fault site to some primary output.

The main advantage of the transition fault model is that the number of faults in the circuit is relatively small (linear in terms of the number of gates). Also, the stuck-at fault test generation and fault simulation tools can be easily modified for handling transition faults. On the other hand, the expectation that the delay fault is large enough for the effect to propagate through any path passing through the fault site might not be realistic because short paths may have a large slack (slack is defined as the difference between the clock period and the nominal delay of the path for the fault-free circuit). The assumption that the delay fault only affects one gate in the circuit might not be realistic either. A delay defect can affect more than one gate, and even though none of the individual delay faults is large enough to affect the performance of the circuit, several faults can together result in performance degradation. For practical simplicity, the transition fault model is frequently used as a qualitative delay model, and circuit delays are not considered in deriving tests.

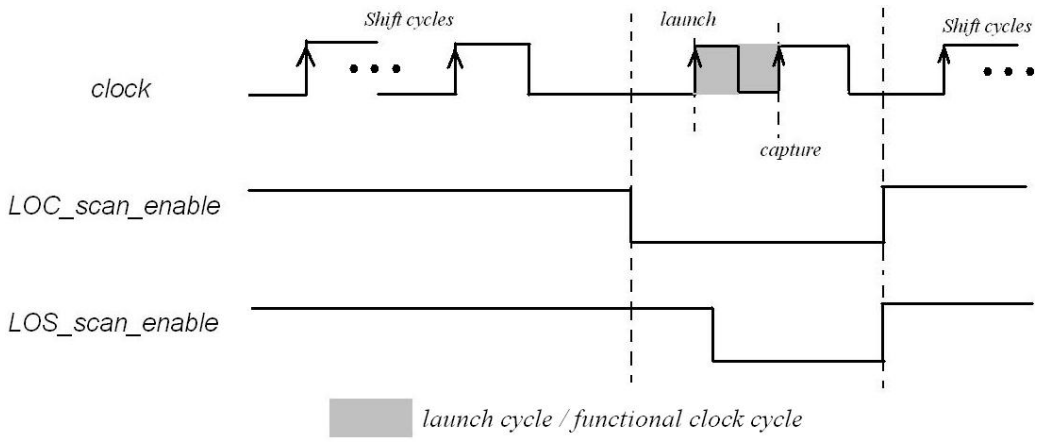


Figure 2.2: LOC and LOS operation for delay testing

Test Application Schemes for Testing Delay Defects

Testing schemes for scan design have been proposed in the literature [25, 115, 135, 136]. These techniques are Launch-On-Capture (LOC), also called broad-side test [136], Launch-On-Shift (LOS), also called skewed-load test [115, 135], and enchanced scan testing [37].

Enchanced scan testing is based on a modified scan cell with an additional latch in order to store and aply two possible bits for delay testing. Its main advantage is that it can achieve high delay-fault coverage, by applying any arbitrary pair of test vectors, that otherwise would have been impossible. A disadvantage, though, is that many false paths, instead of functional data paths, may be activated during test, causing an over-test problem. Also, in enhanced-scan testing, both vectors in the vector pair have to be stored in the tester memory. The first vector is loaded into the scan chain, followed by its immediate application to initialize the circuit under test. Next, the second vector is scanned in, followed by an immediate application and capture of the response. Note that the node values in the circuit is preserved during the shifting-in of the second vector. In order to reduce over-test, and the requirement to store both vectors on the ATE, the conventional LOS and LOC delay test techniques using normal scan chains can be used.

In LOC based testing, all the n bits of a vector are scanned into the circuit at slow speed, followed by another clock which creates the transition. Finally, an at-speed functional clock is applied that captures the response. Thus, only one vector has to be stored per test, and the second vector is directly derived from the initial vector by pulsing the clock. In LOS testing, the first $n-1$ bits of an n-bit vector are shifted in at slow speed. The final n^{th} shift is performed, and it is also used to launch the transition. This is followed by an at-speed quick capture. Similar to LOC, only one vector has to be stored per test, as the second vector is simply the shifted version of the first vector.

Figure 2.2 illustrates the timing diagram of these two schemes. In LOC, the activation pattern is launched by capture; in LOS, the activation pattern is launched by last shift. Once the activation pattern is launched, the test response is captured after the functional

clock period to see whether the transitions reach the primary output and flip-flops in time. This functional cycle is called launch cycle.

2.1.5 Automatic Test Pattern Generation

Generating effective test patterns for a digital circuit is the goal of any Automatic Test Pattern Generation (ATPG) tool. ATPG tools can determine a list of faults, based on the targeting fault model, and the test vectors which detect the faults with the corresponding fault free responses. In addition, using fault detection simulation (or simply fault-simulation) ATPG tools can estimate the fault coverage (how many from the total modelled faults can be detected), and they can also be used for fault diagnosis i.e. identifying the possible root that caused the defect). The test vectors generated by ATPG can be completely specified (e.g., 1100) or incompletely specified (e.g., 1x0x) in which case they are referred to as test cubes. Hereafter "test cube" refers to a test pattern consisting of specified ('0' or '1') and unspecified values ('X'), while "test vector" refers to a completely specified test pattern.

2.1.6 N-detection

In order to enhance the quality of a test set, one may wish to derive different test sets targeting different fault models as an attempt to capture potential defects that could arise. However, this requires multiple ATPG engines, each targeting a different fault model. While this may be theoretically possible, it may not be possible in practice. Instead, to increase the coverage of all possible defects, one may generate a test set that achieves multiple detections of every fault under a given fault model. A fault is detected multiple times, if it is detected with different vectors. By exciting the fault and propagating the fault effect different ways, it is hoped that any defect locally close to the target fault will have an increased chance of being detected [23, 38, 98]. The size of an N-detect test set grows approximately linearly with respect to N [127]. Note that N-detection can be considered as an ad-hoc approach to increase the unmodeled defect coverage of the generated tests.

2.1.7 Unmodeled Faults

The need for more and more complex fault models is rapidly increasing the test data volume for integrated circuits testing. Moreover, in VDSM technologies many defects cannot even be accurately modeled using known fault models [162]. It is, therefore, important to grade the tests patterns of well-practiced fault models (such as stuck-at) based on their ability to detect unmodeled defects. For this reason probabilistic fault models have been developed. This section describes a probabilistic gate oriented fault model and a technique which is called output deviations that grades a test vector on its ability to detect faults from the probabilistic fault model (similar to fault simulation, but

in a probabilistic approach). Most importantly, this methodology is not biased towards any particular fault model. As shown in [169], unbiased testing provides higher test quality than a test method that is biased by a particular fault model.

Output Deviations

In [176, 178–180] a probabilistic gate-level fault model is presented together with a technique to compare tests patterns for their ability to detect arbitrary defects. This model is based on probability measures, named *output deviations*, at primary outputs and pseudo-outputs (all referred to as outputs) that reflect the likelihood of error detection at these outputs. It was shown in [178], test patterns with high deviations tend to be more effective for fault detection.

According to this model a probability map (referred to as the confidence-level vector) is assigned to every gate in the circuit. The confidence level R_i of a gate G_i with m inputs and a single output is a vector with 2^m components, $R_i =< r_i^{0\ldots00}, r_i^{0\ldots01}, \ldots, r_i^{1\ldots11} >$, where each component denotes the probability the gate's output to be correct for the corresponding input combination. This gate-level confidence level vectors can be generated in a number of ways, e.g., using layout information, inductive fault analysis [42], and failure data analysis. It can also be estimated using simple transistor-level failure probabilities. In practice, multiple sets of confidence-level vectors estimates can be used. Two rules apply on this probabilistic fault model:

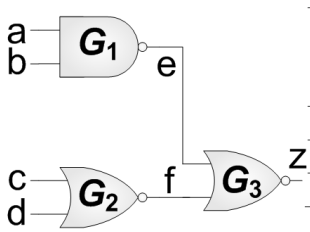
1. Each gate can fail independently of other gates.
2. The fault behavior of the circuit is described by the confidence-level vectors of all its gates.

For grading a test based on its ability to detect faults of this fault model, signal probabilities are required. So, signal probabilities $p_{i,0}$ and $p_{i,1}$ are associated with each line i in the circuit (the term "line" is used to denote a "net"), where $p_{i,0}$ and $p_{i,1}$ are the probabilities for line i to be at logic 0 and 1, respectively. Obviously we have $p_{i,0} + p_{i,1} = 1$. The calculation of the signal probabilities is along the same lines, as introduced in [114], and used later in [132]. To reduce the amount of computation, as in [114, 132], signal correlations due to reconvergent fanout are not considered. The following example illustrates the simple *propagation function* of the signal probabilities based on the confidence-level r of a NAND gate G_i with output wire y and input wires a, b:

$$p_{y,0} = p_{a,1}p_{b,1}r_i^{11} + p_{a,0}p_{b,0}(1 - r_i^{00}) + p_{a,0}p_{b,1}(1 - r_i^{01}) + p_{a,1}p_{b,0}(1 - r_i^{10}),$$
$$p_{y,1} = p_{a,0}p_{b,0}r_i^{00} + p_{a,0}p_{b,1}r_i^{01} + p_{a,1}p_{b,0}r_i^{10} + p_{a,1}p_{b,1}(1 - r_i^{11})$$

Likewise, the signal probabilities can be computed for other gates. Under this fault model, the expected output values of the circuit in response to an input pattern is no longer deterministic. Rather, it is given by the signal probabilities at primary outputs. Note that the circuit behavior is assumed to be deterministic after manufacturing; the probabilistic fault model is only used during test development.

Based on this propagation method for signal probabilities under the confidence-level vectors, we can compute a measurement of the *expected* signal probability for an observed



Input pattern (a,b,c,d)	z	$p_{e,0}$	$p_{e,1}$	$p_{f,0}$	$p_{f,1}$	$p_{z,0}$	$p_{z,1}$
0000	0	0.1	0.9	0.2	0.8	0.886	**0.114**
0101	0	0.1	0.9	0.9	0.1	0.837	**0.163**
1111	1	0.8	0.2	0.9	0.1	**0.396**	0.604

Figure 2.3: Output deviations example

output (based on an input pattern and the fault-free value for this input pattern on the observed output), that is proportional to the likelihood for that pattern to produce an observable error at that output. Specifically, for any gate G_i in a circuit, let its fault-free output value for any given input pattern t_j be d, with $d \in \{0, 1\}$ ($d = 0$ or $d = 1$). The *output deviation* $\Delta G_{i,j}$ of G_i for t_j is defined as $p_{G,\overline{d}}$, where $\overline{d}$ is the complement of d. The probability that a pattern will produce an observable error at an output for this probabilistic fault model is directly proportional to that output's deviation.

Example 2.1. Figure 2.3 depicts a small combinational circuit and a Table that contains patterns to be evaluated using output deviations for their unmodeled defect coverage. The first column of this Table contains different test patterns for the primary inputs a, b, c, d of this circuit. Under column z the fault-free values for the output z for the corresponding input pattern are listed. The next 6 columns contain the expected output signal probabilities for all the lines and possible observable logic values '1' or '0' in the circuit (e, f, z). On the primary output z of the circuit, we observe $P_{z,0}$ and $P_{z,1}$ as the expected signal probabilities for values 0 and 1 respectively. The output deviation of the output z for all the input patterns are shown in bold inside the table (only the complements of fault-free values are evaluated). These output deviations are proportional to the probability to observe a fault at the output z for the given input combination. At this example the last pattern 1111 is the one with the highest output deviation value ($p_{z,0} = 0.396$ and its fault-free value is '1') and the most likely to offer the highest defect coverage on this probabilistic fault model for this circuit (among the patterns evaluated here). ■

Although, the basic idea behind grading tests based on the output deviations is simple, the generation and grading of candidates is not. As a result there are different metrics for this purpose. Their basic idea is to first, calculate the maximum output deviations that can be observed on a single output and, then to aggregate only the highest output deviation values. Additionally, there is not a general rule for the test generation of candidates, except that the candidates should contain diversity in order to increase the probability some of them to maximize the output deviation values. For example under TRP compression environments the candidates generation cannot be done by the filling of the undefined 'x' values because that would compromise compression and it would increase test cost. As result, the efficiency of a candidates' generation algorithm is important to quality enhancements based on the propabilistic fault model of output deviations.

Validation of Output Deviations

To overcome the problem of accurately measuring the unmodeled defect coverage surrogate fault models were used, i.e. fault models that are not targeted by the generated test sets. For example, for a test set generated to cover stuck-at faults, surrogate fault models could be the transitions fault model and the bridging fault model. This idea is based on the assumption presented in Figure 1.10d: if a test covers the same modeled-defects but more unmodeled-defects than another, then the set of the overall covered defects of that test is deemed to be larger. A larger set of covered defects is more likely to contain defects of another fault model (i.e. a fault modeled that was not considered during the generation of the test). As a result, observing the fault coverage on surrogate fault models is a way to evaluate the unmodeled defect coverage of tests.

Literature on Output Deviations

In [176, 179] test set enhancement techniques for selection of test patterns that maximize output deviations were presented. The reordering technique of test patterns presented in [178] maximizes the defect coverage ramp-up for an abort-on-first-fail test environment, while [69] describes a static compaction technique for stuck-at test vectors that maximizes output deviations. The compression method of [180] presents a quality enchancement for classical LFSR reseeding [78] technique that maximizes the output deviations of the generated patterns by exploiting wasted variables.

2.2 Test Response Partitioning Techniques

In order to offer high compression, TRP techniques usually exploit the following inherent properties of test cubes (test cubes are vectors consisting of '0', '1' and 'x' values):

1) The correlation between the specified '0', '1' values that stems from the structural correlation of faults [158],
2) The large amounts of unspecified ('x') values.

Code-based techniques exploit the correlations between the specified values, while linear-based techniques exploit the large amount of unspecified values.

The most widely adopted linear-based method is that of reseeding LFSRs [77, 79, 80]. LFSR reseeding exploits the low fill rate of test cubes. In [106] ring generators were proposed as an alternative to classical LFSRs and in [123] embedded deterministic test (EDT) was presented. Other well known techniques have been presented in [7, 13, 30, 31, 125, 156, 196, 197]. However linear-based methods do not exploit the high correlation between test cubes' specified bits. In addition, they are ineffective for testing IP-cores which are usually accompanied by pre-computed and pre-compacted test sets. The main idea behind LFSR reseeding is to exploit the low density of specified bits in the test cubes (i.e., test patterns with 'x' logic values) in order to compress test cubes into LFSR seeds. A seed is computed by solving a system of linear equations, where the initial state of each

LFSR cell is considered to be a binary variable. Although there are many LFSR reseeding techniques, each technique falls in one of the following categories: a) static reseeding or b) dynamic reseeding. In static LFSR reseeding the contents of the linear decompressor are flushed during reseeding, while in dynamic approaches they are not (flushed).

Many TRP techniques have been proposed that are suitable for cores of known structure [12, 53, 54, 66, 79, 102, 124, 159, 164, 188]. The high efficiency of these techniques is mainly attributed to the exploitation of the capabilities offered by the ATPG and fault simulation tools during the compression process. However, in the case of IP cores, where the structure of embedded cores is hidden from the system integrator, the utilization of such tools is not an option. The only option provided in these cases is to directly compress a pre-computed and usually pre-compacted test set which is provided by the core vendor. As a result, various methods have been proposed so far for compressing pre-computed test sets of IP cores. Among them, many methods utilize linear decompressors [7, 81, 82, 82, 85, 138, 166, 181] whereas others utilize various compression codes [20–22, 49, 60, 71–74, 111, 130, 149, 150, 171]. Although, these techniques are efficient for compressing pre-compacted test sets of IP cores, they are less efficient for cores of known structure. Also, there are also methods that do not belong in any of the above categories, e.g., [92] and [125]. Commercial tools have also been developed [10, 78, 123].

The next sections present, briefly, some of the most popular TRP techniques: the classical static LFSR reseeding, the window-based LFSR reseeding, the dynamic/partial LFSR reseeding, and the Optimal Selective Huffman (OSH) code-based technique.

2.2.1 Static LFSR Reseeding Techniques

In static reseeding, test cubes are encoded into seeds, and every seed is loaded into a Linear Feedback Shift Register (LFSR) before decompression begins. Static reseeding, in its classical form, uses one new initial LFSR state (seed) for encoding a single test cube of the test set [77]. The major drawback of this approach is that it offers limited compression. Many other static LFSR reseeding methods have been proposed in the past [60, 67, 81, 82, 102, 155, 166] which offer better compression than [77]. A particularly efficient approach is window-based reseeding [67], where each seed is used to generate more than one test vector i.e., each seed is expanded into a window of test vectors.

Classical LFSR Reseeding

The classical LFSR reseeding scheme [77] uses an LFSR-based decompression logic as presented in Figure 2.4. Every n-bit seed (n is the LFSR size) is transferred from the ATE to the LFSR, where it is expanded into a test vector of $m \times r$ bits (m is the scan-chain volume and r the scan-chain length) and is loaded into the scan chains. When a test vector is loaded into the scan chains of the CUT, the response of the previous vector is shifted out to the Test Response Compactor. The phase shifter [9, 58, 99] is used to reduce the linear dependencies [107] of the bit sequences generated by the LFSR cells.

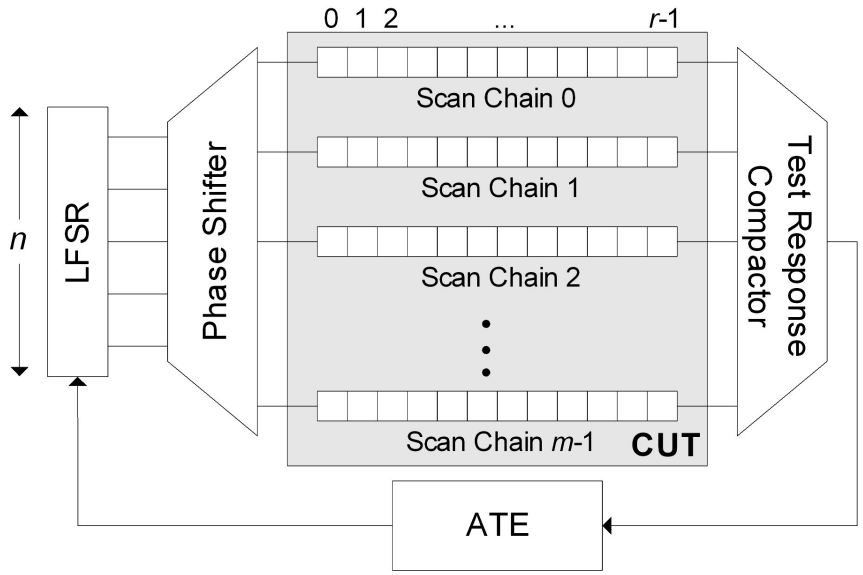


Figure 2.4: Classical LFSR-based decompression architecture

Algorithms for designing synthesizable phase shifters are presented in [106, 121, 122]. The test set of the core consists of test cubes of $m \times r$-bits each, and every one of them is compressed into an n-bit seed ($n \ll m \times r$) which is calculated by solving a system of linear equations. This system is formed according to the specified bits of the test cube [77] (the 'X' bits are filled with pseudorandom data during decompression). Specifically, the initial state of the LFSR is considered as a set of binary variables $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$. At every clock cycle, m linear expressions of these variables are generated at the m outputs of the phase shifter. During r successive clock cycles, $m \times r$ linear expressions are generated at the outputs of the phase shifter, and each one of them corresponds to one of the $m \times r$ scan cells. Thus, each bit of a test cube corresponds to exactly one linear expression. Every linear expression corresponding to a specified bit of a test cube is set equal to that bit, and in this way the system of linear equations is formed (the unspecified bits of the test cubes are not considered during this step). The solution of this system is the seed of the LFSR. The system with the maximum number of linear equations corresponds to the test cube with the maximum number of specified bits, s_{max}, which in turn determines the minimum required LFSR size. As it was shown in [77], if the LFSR size n is equal to $s_{max} + 20$, then the probability of not being able to solve the linear system for encoding a test cube is less than 10^{-6}. However, LFSR polynomials with size less than $s_{max} + 20$ exist, which can compress all test cubes [12].

Example 2.2. Figure 2.5 presents a reseeding example. On the upper left corner of the figure there is the test cube to be encoded, while under it there is the utilized LFSR. At the left of the LFSR there are the clocks numbered and the symbolic states of the LFSR, presented line-by-line for each cycle though symbolic simulation (the terms $\alpha_i \alpha_j \ldots \alpha_k$ at the symbolic simulation are used to denote the values $\alpha_1 \oplus \alpha_j \oplus \ldots \oplus \alpha_k$, where $\oplus$ is the XOR logic function). Suppose that a scan chain is directly loaded with the contents of

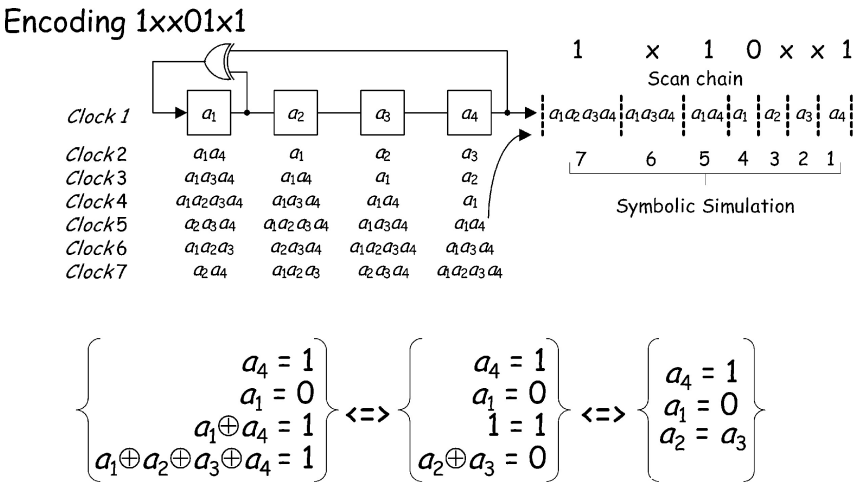


Figure 2.5: Classical LFSR reseeding example

the last cell of the LFSR. The contents of the scan chain for each cycle can be seen at the right of the LFSR. By applying the test cube on this symbolic representation of the scan chain's contents and equalizing its defined bits with the symbolic representations, we form the linear system. The solution of this system is the LFSR's seed that generates the encoded test cube. Notice that the number of equations that form the linear systems strongly depends to the number of specified bits of the test cube. ■

Window-Based LFSR Reseeding

According to the classical LFSR reseeding, every seed is used for encoding a single test cube. The achieved compression in this case is moderate, since usually in a test set there are many test cubes with fewer specified bits than the bits of the maximum specified test cube. As a result, a lot of variables remain unspecified when the corresponding systems are solved, and therefore much of the potential of LFSR's encoding is wasted.

Various methods have been proposed for better utilization of LFSR's variables, [79, 123, 166, 195] to name a few. A very attractive one is to utilize the same seed for encoding more than one test cube in a sequence of L pseudorandom vectors. In other words, each seed is expanded into a window of L vectors, instead of one. The number of test cubes encoded in the window is usually much smaller than L, which means that useless vectors are also applied to the CUT. This approach is very effective since for every test cube, L (and not just one) systems of equations are constructed, and among the solvable systems, the one resulting in the highest compression is selected. In other words, each test cube is encoded in such a way so as to maximize the overall encoding efficiency. There are many ways to encode multiple test cubes in an L-vector window. One very effective algorithm for minimizing the number of seeds is the following [33, 65]: initially, the test cube with the highest number of specified bits is selected and the system corresponding to the first vector of the window is solved (the selection of the LFSR polynomial and the phase shifter guarantees that this system is always solvable). The remaining test cubes are selected

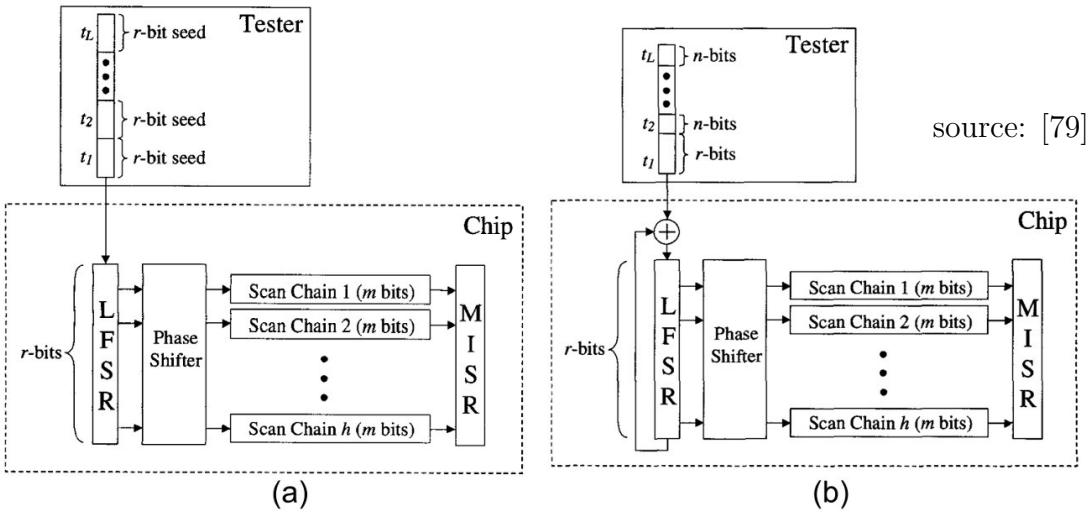


Figure 2.6: (a) Static reseeding versus (b) dynamic reseeding

iteratively according to the following criteria:

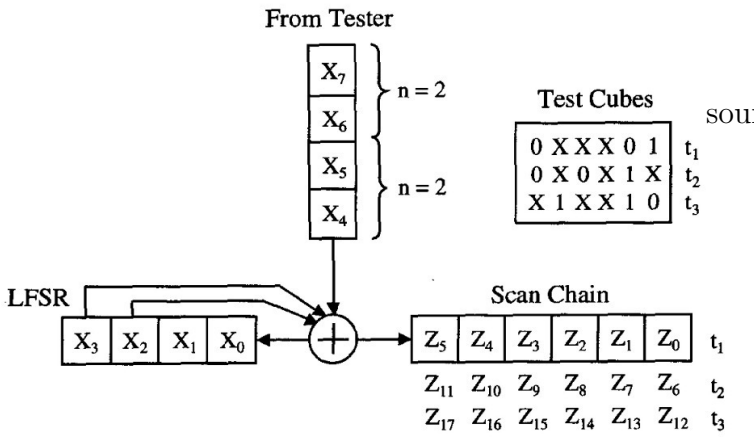
- Among the solvable systems that correspond to the test cubes containing the maximum number of specified bits, we identify those that their solution leads to the replacement of the fewest variables in the L-vector window.
- Among them, we find those corresponding to the cube that can be encoded the fewest times in the window.
- Finally, among them we select the system nearest to the first vector of the window.

After solving the selected system, some of the variables are replaced by logic values, whereas the rest remain unspecified and they are utilized for encoding additional test cubes. The construction of a seed is completed when no system for any of the unencoded test cubes can be solved in the L-vector window. Although, test set embedding techniques, such as window-based LFSR reseeding, can achieve high compression efficiency they suffer from long test sequences. *Later, a new technique is presented for shortening the test sequence length of window-based LFSR technique making this encoding method feasible and attractive for testing IP cores.*

2.2.2 Dynamic LFSR Reseeding

Dynamic reseeding methods [78, 79, 123] constitute another class of methods that offer high compression. In these approaches the content of the linear decompressor are not flushed during the reseeding and as a result any remaining unsolved variables inside the decompressor can be still exploited for compression.

As we mentioned for classical LFSR reseeding in Section 2.2.1, and we highlight again in Figure 2.6a, an r-bit LFSR is loaded with an r-bit seed and then generates the desired test vectors. Afterwards, it flushes its contents and it is being loaded with a new r-bit seed etc. This kind of reseeding results into wasted variables because the size of the LFSR r



source: [79]

Test Cube t_1	Test Cube t_2	Test Cube t_3
$Z_0 = X_2 + X_3$	$Z_6 = X_1 + X_2 + X_3 + X_4$	$Z_{12} = X_2 + X_4 + X_6$
$Z_1 = X_1 + X_2$	$Z_7 = X_0 + X_1 + X_2 + X_5$	$Z_{13} = X_1 + X_5 + X_7$
$Z_2 = X_0 + X_1$	$Z_8 = X_0 + X_1 + X_2 + X_3$	$Z_{14} = X_0 + X_4$
$Z_3 = X_0 + X_2 + X_3$	$Z_9 = X_0 + X_1 + X_3 + X_4$	$Z_{15} = X_2 + X_3 + X_4 + X_5 + X_6$
$Z_4 = X_1 + X_3$	$Z_{10} = X_0 + X_3 + X_4 + X_5$	$Z_{16} = X_1 + X_2 + X_4 + X_5 + X_6 + X_7$
$Z_5 = X_0 + X_2$	$Z_{11} = X_3 + X_5$	$Z_{17} = X_0 + X_1 + X_4 + X_5 + X_7$

Figure 2.7: Partial/Dynamic LFSR symbolic simulation

depends on the number of bits s_{max} of the most specified test cube in a test set. Dynamic LFSR reseeding is shown in Figure 2.6(b). Note that an extra XOR gate is included in the feedback of the LFSR. The LFSR of the figure has only one input and loads serially the seeds on it. The initial r-bit seed it used to initialize the LFSR and it is let to operate and generate the generated test vector. Afterwards, instead of flushing its contents, it dynamically (the term *dynamic* used by [79] to denote "without flushing") loads the next n-bit seed (where $n < r$; from this property stems the term *partial* LFSR reseeding and it was first introduced by [195] with the term *variable-length seeds*) without flushing. As a result any unresolved variables from the previous r-bit seed remain active in the LFSR and may be utilized in a later phase. The next generated vector can now exploit any unresolved variables from previous seeds together with the newly inserted n-bit seed.

The contents of an LFSR that is dynamically and partially reseeded may be symbolically simulated in a similar way with the symbolic simulation of static LFSR reseeding. An example of the new symbolic simulation procedure and linear systems forming is illustrated in Figure 2.7. However, there is a difference between the symbolic simulation of static and dynamic reseeding. The symbolic simulation for dynamic reseeding is a very time consuming process because the linear equations for the test cubes need to be solved altogether [79]. As a result, dynamic reseeding may not be scalable to large designs unless proper actions are taken. To this end [79] proposed a test set partitioning method which provides sub-optimal results but reduces the CPU time of dynamic reseeding symbolic

simulation and linear equations solving. The reported CPU times at [79] are in the order of hours (on CPUs of that time) when the partitions consist of hundreds of test cubes. In the experiments presented in this dissertation and concern dynamic LFSR reseeding we have not implemented this partitioning technique as we intended to provide the most favorable results in terms of compression for dynamic reseeding. Nevertheless, even for our largest benchmark circuit, "Ethernet" with 10 thousands of test cubes from the IWLS [1] benchmarks suite, the CPU run-times for forming and solving the equations of dynamic LFSR reseeding without the partitioning technique of [79] are in the order of hours (on contemporary CPUs).

2.2.3 Code-based Techniques

Code-based schemes use data compression codes to encode the test cubes. This involves partitioning the original data into symbols, and then replacing each symbol with a code word to form the compressed data. To perform decompression, a decoder simply converts each code word in the compressed data back into the corresponding symbol. Code-based compression techniques are classified depending on whether the symbols have a fixed or variable size (symbols have the same of different numbers of bits respectively) and whether the codewords have a fixed or variable size. Therefore, four categories follow: fixed-to-fixed [126, 190], fixed-to-variable [11, 60, 61, 72, 90, 96, 139, 192], variable-to-fixed [62, 177, 189] and variable-to-variable[18, 20–22, 71, 73, 74, 87, 111, 149, 150].

The first data compression codes that researchers investigated for compressing scan vectors encoded runs of repeated values. In [61, 62] a scheme based on run-length codes that encoded runs of repeated '0' values using fxed-length code words is proposed. In [20] a technique based on Golomb codes that encodes repeated values with variable-length codewords is presented. The use of variable-length code words allows efficient encoding of longer runs, although it requires a synchronization mechanism between the tester and the chip. Further optimization is achievable by using frequency-directed run-length (FDR) codes [21, 22, 40] and variable-input Huffman codes [49, 60, 71, 72, 74], which customize the code based on the distribution of different run lengths in the data. Other techniques that utilize other compression codes or multiple codes simultaneously are [11, 111, 149, 150, 189].

Code-based schemes are very effective in exploiting correlations in test cubes and they do not depend on the Automatic Test Pattern Generation (ATPG) process used. Consequently, they are very effective on pre-computed (and usually pre-compacted and densely specified) test sets for Intellectual Property (IP) cores. However, they suffer from several serious drawbacks that prohibit their use in industrial designs: they do not exploit the low fill rate of test cubes; they impose long testing times as they cannot exploit the large number of scan chains; they require extensive interaction with the tester.

Table 2.1: Test Set Partitioned to Data Blocks and Distinct Blocks' Frequencies

Test Set T	Distinct Blocks	Occur. Freq.
1010 0000 1010 1111	1010	9/20
1111 0000 1010 0001	0000	5/20
1010 0000 0010 1010	1111	3/20
0000 1010 1010 0000	0001	2/20
1010 1111 1010 0001	0010	1/20

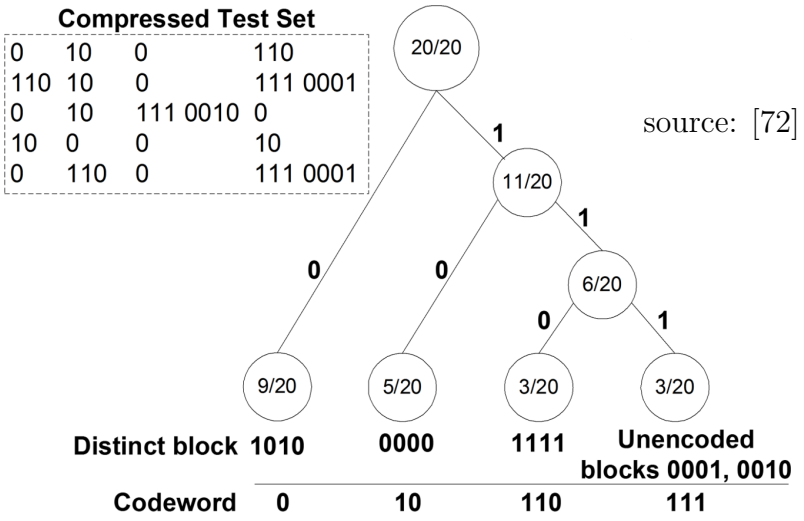


Figure 2.8: Optimal Selective Huffman Example

Optimal Selective Huffman

The Huffman code [57] is a fixed-to-variable code that uses short codewords to encode frequently occurring blocks and long codewords for the less-frequent ones. The Optimal Selective Huffman (OSH) code encodes only the m most frequent blocks [60] while the rest of the blocks remain un-encoded and they are distinguished by using an extra Huffman codeword [72].

Let us assume a core with n scan chains of length r (we assume a balanced scan structure where the shorter scan chains are padded with 'x' logic values). Each scan slice constitutes a single block. Let T be a set of test cubes and $|T|$ be its size in bits. T is partitioned into $|T|/l$ data blocks of size l. Among these blocks the m most frequent distinct blocks, $b_1, b_2, ..., b_m$, with frequencies (probabilities) of occurrence $f_1 \geq f_2 \geq \cdots \geq f_m$ respectively are stored in a dictionary and they are encoded using m Huffman codewords. The rest of the blocks with an aggregate frequency $f_{un} = f_{m+1} + f_{m+2} + \ldots$ remain not-encoded and a single codeword is used to precede them (they are stored in a raw form in the compressed test data [72]). A binary tree is constructed beginning from the leaves and moving towards the root. For every dictionary entry b_i a leaf node is generated, and a weight equal to f_i is assigned to it. The pair of nodes with the smallest weights is selected first and a parent node is generated with a weight equal to the sum

of the weights of both nodes. This is repeated iteratively, until the root is left unselected (each node can be selected only once). After the tree is constructed each leaf node is assigned a codeword as follows: starting from the root, all nodes are visited once and the logic '0' ('1') value is assigned to each left (right)-child edge. The codeword of block b_i is the sequence of the logic values of the edges on the path from the root to the leaf node corresponding to b_i.

Example 2.3. Consider the test set of Table 2.1 and that $m = 3$, that is 0001 and 0010 are the unencoded blocks. The sum of the occurrence frequencies of 0001 and 0010 is equal to $2/20 + 1/20 = 3/20$. The OSH encoding as well as the compressed test set are given in Figure 2.8. The encoding distinct blocks 1010, 0000 and 1111 are encoded by codewords 0, 10 and 110 respectively. The unencoded data blocks are distinguished by the 3-bit codeword 111. Finally, the number of bits for the compressed test data is 42 bits (from the nested Table of Figure 2.8), while the uncompressed test data were 80 bits (from Table 2.1). ■

Despite the fact that there are many blocks in a test set consisting mostly (or even entirely) of 'x' values, each and every one of them has to be encoded using a separate codeword. As a result even if a test was fully specified still many bits would be required for its encoding. Assume that the test set of Table 2.1 was fully unspecified. Then 20 bits (1 bit per block) would be the size of the compressed test set by the OSH method. It becomes obvious that although, the selective Huffman code [60], [72] offers low cost decompressors and high compression at the same time, it can not be used for industrial applications because it cannot exploit the unspecified values in the test sets. Another important drawback is that it requires a synchronization mechanism between the ATE and the CUT.

Later, an OSH based decompression architecture is described that does not require synchronization between ATE/CUT. Moreover the new compression method that exploits both the correlations and the undefined values in the test sets offering higher compression than any other known test data compression method.

2.2.4 Industry Practice: Embedded Deterministic Test (EDT)

Embedded Deterministic Test (EDT) was proposed in [123] and it is constantly being enriched with new properties since then. So, it is a collection of tools and methods to create a successful embedded testing architecture based on a modified LFSR called *ring generator*.

Similar to linear-based approaches, ring generators are based on prime LFSR polynomials. Usually (if not always), prime polynomials XOR taps synthesis result to high fan-outs of the decompressors and as a result slow decompression feedback operation. In [106] a transformation method was presented of an LFSR to a more synthesizable-friendly form with XOR's fan-out maximum value of 2. In Figure 2.9 a ring generator is presented. Figure 2.10 illustrates the basic EDT architecture:

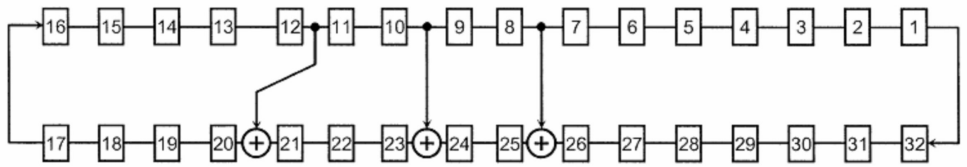


Figure 2.9: A Ring Generator

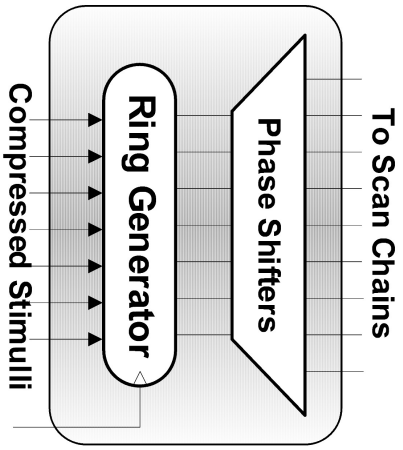


Figure 2.10: EDT basic Architecture

- Compressed data are provided to the ring generator [106] from the ATE.
- The pseudorandom test sequences generated by the ring generator are shifted by the Phase Shifters [9, 58, 99, 106, 121, 122] and then fill the scan chains.

Every generated vector of length L is loaded into the scan chains as K slices of S size each where S is also the number of the scan chains and $L = K \times S$.

EDT utilizes dynamic/partial reseeding on ring generators. Symbolic simulation for dynamic reseeding requires all the variables to be handled together (see Section 2.2.2). This is a bottleneck for the execution time and it was handled by [79] with a partitioning of the test set. But, the original algorithm of EDT proposed in [123] is not applied on a pre-computed test set, so this partitioning is not feasible. So, EDT has adopted a variable's elimination strategy to handle this bottleneck. Moreover, variable elimination exploits contemporary ATE's REPEAT command [167]. Suppose that variables are injected from a channel between the ATE and the ring generator. Any unresolved variable is decided if it will be eliminated (the elimination is to be set to '1' or '0' for symbolic simulation scaling reasons) or not based on some criteria (Variables Elimination criteria). Some, criteria are based on profiles on the number of specified bits of the test cubes (Non-Adaptive Variables Elimination) or ad-hoc criteria during the compression based on the remaining free variables (Adaptive Variables Elimination). In order not to compromise the compression by this approach, ATE's REPEAT command is exploited and the eliminated variables are set to the previous value that was injected from the same channel. This way the compression tool of EDT overpasses the bottleneck of handling all the variables together and also becomes applicable in synergy with ATPG and fault simulation.

EDT uses fault simulation after the generation of a test pattern in order to drop any easy-to-detect faults (faults with test cubes that have few specified values and are randomly tested). The ATPG generates test cubes during the compression (as a result the compression algorithms gets the next-to-compress test cube directly from the ATPG tool). This interaction between ATPG/compression-tool can maximize compression, especially for N-detection test sets, because faults are directly dropped during the fault simulation step and they are not considered from the ATPG tool for the generation of the next test cube.

2.3 Low-Power Testing Techniques

A drawback of both linear-based and symbol-based test data compression techniques is that they elevate switching activity beyond acceptable levels and thus degrade production yield [141]. Power consumption during scan testing consists of two switching activity components, namely *shift* and *capture* power. In particular, shift switching activity (referred also as shift power) is caused during the shift (scan in-out) mode of scan designs when successive complementary logic values are shifted into the scan chains. When complementary values are shifted into a scan chain they generate transitions on the scan cells while they travel to their final destination. The transitions on the scan cells, inevitably, generate more transitions at the combinational part of the circuit that is attached at the scan chain. The result is increased switching activity during the shift in-out mode. This increased switching activity is responsible for the average power consumption that increases the generated heat during testing beyond the acceptable TDP limits. On the other hand, capture switching activity (referred also as capture power) is caused during the capturing of the responses on the scan cells. The transitions generated during capture mode may increase the instantaneous power demand of the CUT leading to ground/voltage bounces that introduce noise at CUTs signal values. Capture power values above certain limits can undermine the reliability of the testing procedure causing yield loss of operational CUTs that appear mistakenly as faulty. To alleviate switching activity during scan testing various techniques have been developed.

Numerous methods have been proposed in the literature for limiting power consumption during testing, targeting shift power [17, 18, 28, 32, 45, 64, 85, 105] or capture power [24, 93, 129, 182–186]. In addition, some methods simultaneously target the reduction of both shift and capture switching activity [19, 76, 89, 128, 134]. These methods can be further categorized as being either structural [18, 24, 28, 32, 45, 76, 85, 105] or algorithmic [134, 182, 183] based on their nature. Structural methods interfere with the scan design architecture by modifying it for low power purposes. On the other hand at the algorithmic methods there are low power ATPG techniques and test cubes manipulation techniques [17, 19, 64, 89, 93, 128, 129, 182, 185, 186] also known as X-filling.

Below the most known structural and algorithmic low power testing techniques are briefly presented.

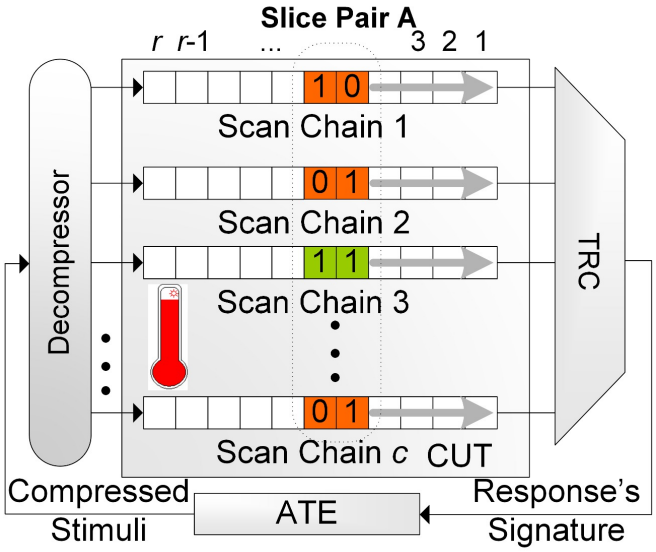


Figure 2.11: Switching Activity caused by Successive Slices

2.3.1 Structural Low-Power Testing Approaches

Even though traditional TDC techniques (like for example [7, 13, 60, 72, 77, 80, 123]) are very efficient in compressing test data, they become deprecated under power dissipation limitations. Especially large power demands exhibit the linear decompressors, because they fill the 'X' values pseudorandomly and they increase thus both the shift and capture power during scan testing. Specifically, linear decompressors are very effective in compressing the test data, they elevate the power dissipation during testing above the functional power budget of the circuit. A few symbol-based TDC techniques such as [20–22, 61, 111], inherently offer low shift power but they are not suitable for cores with multiple scan chains.

To comply with power consumption requirements, linear decompressors which offer low switching activity during testing have emerged [29, 32, 84, 105]. These techniques require additional data to control the switching activity. Specifically, the state-of-the-art low power dynamic reseeding [26, 105] utilizes a shadow register to offer low power shift testing by repeating test data but it requires additional test data compared to EDT [123] for controlling the low power operation of the decompressor. In [27] selective scan enable deactivation is used for low capture power and in [160] presents a TDC technique with narrow ATE-bandwidth requirements. The method proposed in [31] exploits similarities between test cubes to offer higher compression and utilizes both shadow registers and scan enable deactivation to generate low power vectors.

Low-power Linear Decompressors

Figure 2.11 presents the classical scan based architecture. The CUT consists of c scan chains of length r (for simplicity we assume that all scan chains are of equal length). The compressed test data are downloaded from the ATE, they are decompressed using the

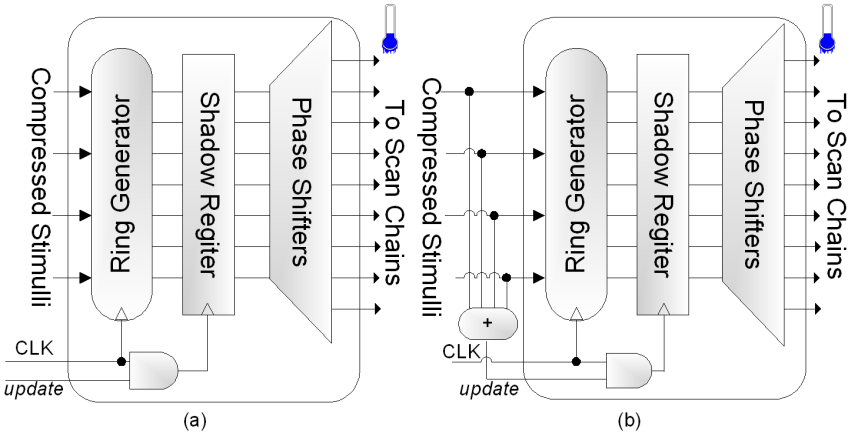


Figure 2.12: (a) Low power EDT controlled by an additional "update" channel, (b) Low power EDT controlled by compressed stimuli

embedded decompressor and they are shifted into the scan chains. For applying a test vector to the CUT the decompressor first generates r successive test slices of size c which are shifted into the scan chains to reach their respective scan slices (hereafter, the term test slice t_j refers to the test bits of test cube t which correspond to scan slice j with $j \in [1, r]$. After the last test slice of t (i.e. t_r) is shifted into the scan chains, t is applied to the CUT and the response is shifted out concurrently with the loading of the next test vector. Linear decompressors fill 'X' values pseudorandomly, and thus they fail to control the number of incompatibilities between successive test slices.

In Figure 2.11, every pair of successive test slices exhibits potential bitwise incompatibilities, i.e. pairs of successive complementary test bits loaded into the same scan chains. For example test slices denoted as "Slice Pair A" in Figure 2.11 are incompatible in the bit positions corresponding to scan chains 1, 2, c. As the test slices travel through the scan chains during the scan-in process, every pair of complementary successive test bits causes transitions on the scan chains which propagate through the combinational logic and cause switching activity to the CUT. The number of incompatibilities between successive test slices can be reduced by exploiting the unspecified values which exist in large volumes in test sets. However, linear decompressors fill 'X' values pseudorandomly, and thus they fail to control the number of incompatibilities between successive test slices.

Low-power EDT

The authors of [105] proposed a linear based encoding method which exploits the 'X' values, wherever they exist, to reduce incompatibilities between successive test slices, and thus to reduce shift power. According to this method, whenever a group of k ($k > 1$) successive test slices of a test cube are compatible (i.e., every slice in this group exhibits no bitwise incompatibilities with any other slice in this group) one test slice S_k is computed which is compatible with all k test slices. This slice is encoded using the ring generator and it is loaded into the scan chains for k successive clock cycles. This is achieved by

Table 2.2: Fill-Adjacent X-Filling*

	Test Cube Block	FA
i	0x...x0, 0x...x, x...x0	00...00
ii	1x...x1, 1x...x, x...x1	11...11
iii	0xx...x1	011...11
iv	1xx...x0	100...00

*the rightmost bit is loaded first into the scan chain

the use of a shadow register shown in Figure 2.12 which can hold its contents if it is properly controlled. Specifically, instead of generating the first slice of this group, the ring generator generates slice S_k and it transfers this slice to the shadow register. This is called UPDATE operation. During the next k successive clock cycles, the shadow register holds its contents and loads the scan chains with slice S_k. This is called HOLD operation. The selection between these two operations of the shadow register requires additional control data which are either provided directly from the ATE (Figure 2.12a) or they are encoded as compressed stimuli (Figure 2.12b). In both cases the additional cost is considerable especially when the number of ATE channels is small and the number of slices per vector is large.

2.3.2 Algorithmic Approaches: Low-Power X-Filling Techniques

X-filling techniques aim at a power-aware logic assignment of the unspecified X-bits. X-filling has negligible impact on ATPG process, and affects neither the scan chain structure nor the circuit under test (CUT). Moreover, they can be combined with other techniques for further reducing test power.

A popular X-filling method for reducing shift power is Fill-Adjacent (FA) technique [17]. This technique targets only the scan-in portion of the shift power, but it also reduces the scan-out power, because, as shown in [19], the scan-in power is highly correlated to the scan-out power. In addition, it can be easily combined with capture-power reduction techniques such as the Preferred-Fill (PF) technique [128, 129], to provide an efficient unified power-reduction solution. The FA and the PF X-filling techniques, for shift and capture power reduction respectively, are briefly presented below.

Overview of Fill-Adjacent

A well-known technique to reduce shift power dissipation is the Fill-Adjacent (FA) technique [17]. This technique targets only the scan-in portion of the shift power. The simplicity of FA and the reason that it can reduce the overall shift power (both scan-in and scan-out, as shown in [19]) is the key of its success.

Every two complementary consecutive test bits loaded into a scan chain generate switching activity as they travel along the scan chain. The FA technique minimizes the shift power by exploiting the X-bits of the test cubes in order to minimize the volume

of the consecutive complementary test bits loaded into the scan chains as well as the distance they travel along the scan chains. For instance, consider a CUT with c scan chains, and assume that the test cube segment $S_j = XXX1XXX01XX0XXX1$ has to be loaded into scan chain j ($1 \leq j \leq c$) from right to left. By applying FA to fill the Xs, we get the test vector segment $T_j = 1111000010001111$. Table 2.2 shows all possible X-fillings produced by the FA technique. The first column shows all possible blocks of test bits comprising any test cube segment that consists of n ($n \geq 1$) unspecified logic values bounded at the left and/or right by specified logic values. The second column shows the X-filling produced for all these blocks.

Overview of Preferred-Fill Techniques

The Preferred Fill (PF) technique (denoted hereafter as PF) is an X-filling technique for reducing the switching activity during capture [16, 17]. Consider a two-pattern Launch-On-Capture (LOC) test $< V_1, V_2 >$ where $V_1 = (v_{11}, v_{12}, v_{13}, \ldots, v_{1n})$ is the first n-bit vector applied on the CUT and $V_2 = (v_{21}, v_{22}, v_{23}, \ldots, v_{2n})$ is the response of V_1 which is applied as the second test vector to the CUT. If the logic value of V_1 corresponding to cell i, (i.e., v_{1i}) is unspecified then it should be filled with value 1(0) provided that the probability of v_{2i} (i.e., the logic value of V_2 corresponding to the scan cell i) taking the value 1(0) is higher than taking the value 0(1). In other words, the v_{1i} bit is filled with a value that is more likely to be held after the capture in the i^{th} scan cell.

X-Filling Limitations

A major drawback of power-aware X-filling techniques is that they are often accompanied by a reduction in defect coverage, since the impact on unmodeled fault coverage is not considered during X-filling. ATPG engines, on the other hand, increase the fortuitous detection of modeled as well as of unmodeled faults by filling randomly the Xs. However, this step elevates the test power.

Later, a unified X-filling method that simultaneously targets power reduction and high defect coverage is proposed.

Chapter 3

State-Skip LFSRs: Bridging the Gap between Test Data Compression and Test Set Embedding

3.1 Overview

Test set embedding techniques offer a very effective means for compressing test sets of IP cores. Test set embedding techniques require considerably less test data storage than test data compression methods, as they use long pseudorandom sequences generated on-chip in order to embed the pre-computed test vectors of IP cores. Various test set embedding methods have been proposed so far in the literature. In [63] and [146] the pseudorandom sequences are generated by counters. In [91] an area demanding reconfigurable interconnection network is presented that achieves a vast reduction of the test data stored on ATE. The main drawback of these techniques is their prohibitively long test application time. The multiphase method proposed in [67] has small hardware overhead and generates shorter test sequences than [63, 91, 146]. An even higher reduction of the test sequence

length is achieved in [33] at the expense of a slight increase in test data volume. However, [67] and [33] still require long test sequences which are unacceptable for testing modern SoCs. Therefore, despite the fact that test set embedding techniques are very attractive in terms of compression ratio, their excessively long test application times render them inapplicable in nanoscale technologies.

To alleviate this problem we present two new types of Linear Feedback Shift Registers, the Single-State-Skip and the Variable-State-Skip LFSRs (SSS_LFSRs and VSS_LFSRs respectively). SSS_LFSRs are normal LFSRs with the addition of a small linear circuit, the State-Skip circuit, which can be used, instead of the characteristic-polynomial feedback structure, for advancing the state of the LFSRs. In such a case, the LFSRs perform successive jumps of constant length in their state sequence, since the State-Skip circuit omits a predetermined number of states by calculating directly the state after them. However, the test-sequence length reduction potential of SSS_LFSRs cannot be fully exploited when multiple non-identical IP cores in a SoC should be tested. In this case it is rather unlikely that the single constant length jump performed by an SSS_LFSR will be sufficient for minimizing the test sequence length of every different core embedded on the SoC. Thus, the low cost solution of using a common SSS_LFSR for all cores is not an optimal one. To overcome this problem, the second LFSR architecture, i.e., the Variable-State-Skip LFSR (VSS_LFSR) is proposed. VSS_LFSRs are very flexible and can fully exploit the State-Skip property in the case of testing multiple IP cores in a SoC, since they embed multiple State-Skip circuits and thus they can perform jumps of variable lengths in the normal LFSR state sequences. VSS_LFSRs achieve greater test-sequence-length reduction compared to SSS_LFSRs at the expense of a moderate increase in the hardware overhead. We have to note however that this hardware overhead is comparable to that of most state of the art compression schemes and it is also compensated by the sharing of VSS_LFSR among the multiple cores.

The combined effect of test set embedding techniques (our case study is the window-based LFSR encoding process) with the powerful test sequence length reduction property of both SSS_LFSRs and VSS_LFSRs offer very short test sequences, close to the test sequences of test data compression methods, with significantly smaller test data volumes. By using SSS_LFSRs for testing single or multiple identical cores and VSS_LFSRs for testing multiple non-identical cores State-Skip LFSRs bridge the gap between test data compression and test set embedding techniques, rendering the latter a very attractive testing approach for IP cores.

3.2 Motivation

The classical LFSR reseeding scheme is shown in Figure 3.1. Every n-bit seed (n is the LFSR size) is transferred from the ATE to the LFSR, where it is expanded into a test vector of $m \cdot r$ bits (m is the scan-chain volume and r the scan-chain length) and is loaded into the scan chains. The test set of the core consists of test cubes of $m \cdot r$ bits, and every

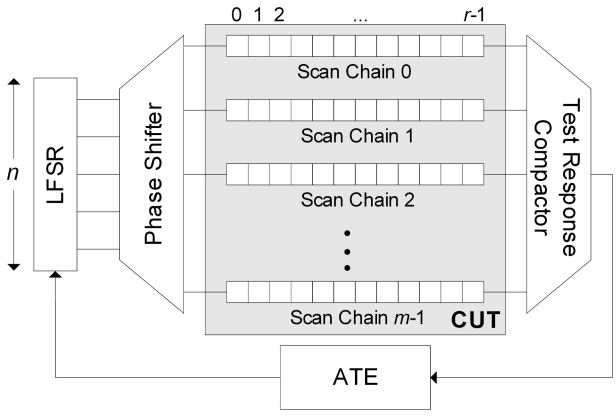


Figure 3.1: Classical LFSR-based decompression architecture

Table 3.1: TDV and TSL of classical ($L = 1$) and window-based ($L > 1$) LFSR reseeding

Circuit	LFSR Size	Classical Reseeding ($L = 1$)		Window-Based Reseeding ($L > 1$)					
				L=50		L=200		L=500	
		TDV	*TSL*	*TDV*	*TSL*	*TDV*	*TSL*	*TDV*	*TSL*
s9234	44	10692	243	8008	9100	7128	32400	6688	76000
s13207	24	8856	369	5328	11100	3816	31800	2688	56000
s15850	39	11622	298	7410	9500	6669	34200	6201	79500
s38417	85	58225	685	50660	29800	48110	113200	47005	276500
s38584	56	22680	405	10584	9450	7056	25200	5152	46000

one of them is compressed into an n-bit seed ($n \ll m \cdot r$) which is calculated by solving a system of linear equations (for details refer to Section 2.2.1).

According to the classical LFSR reseeding proposed in [77], every seed is used for encoding a single test cube. The achieved compression in this case is moderate, since usually in a test set there are many test cubes with fewer specified bits than the number of specified bits s_{max} of the most defined test cube [77]. Window-based LFSR reseeding, which is also covered in Section 2.2.1, is an approach to eliminate these wasted variables.

To show the compression superiority of the window-based method over the classical LFSR reseeding we conducted the following experiment: uncompacted test sets generated by Atalanta [83] for complete coverage of stuck-at faults for the largest ISCAS'89 benchmark circuits were compressed using the LFSR encoding proposed in [33] for $L = 1$, as well as for L =50, 200, 300, 500 and 1000 (window-based encoding). 32 scan chains were assumed for each circuit. For providing a fair comparison, the same encoding approach was applied for all examined window sizes. Hence, even for $L = 1$, each seed was let encode as many test cubes as possible (i.e., all compatible test cubes that can be merged into a single cube, as long as the linear system for calculating the corresponding seed is still solvable). Note that this approach provides much better compression than the classical reseeding approach of [77] where compacted test cubes are encoded into LFSR seeds (in [77] one seed encodes just one cube). For every circuit we calculated the required test

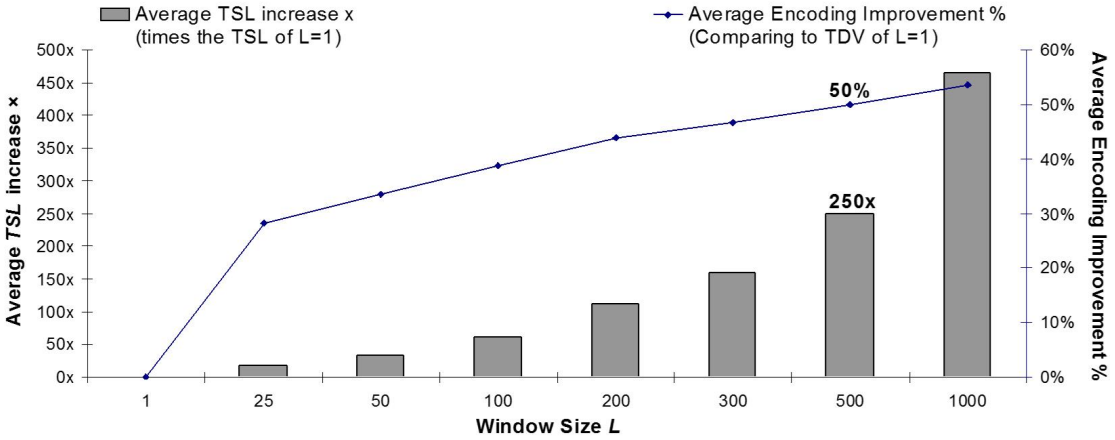


Figure 3.2: Average TDV improvement and average TSL increase of window-based LFSR encoding compared to the encoding of $L = 1$

data volume (TDV) and test sequence length (TSL) i.e. the amount of patterns applied at the CUT for all the aforementioned values of L. The results for L =1, 50, 200 and 500 are shown in Table 3.1. Then, for each circuit and each window size, we calculated both the test data volume improvement and the test sequence length increase of window-based reseeding according to the formulas:

$$TDV_{impr} = 1 - TDV_{L>1}/TDV_{L=1}, \;\; TSL_{impr} = 1 - TSL_{L>1}/TSL_{L=1}$$

Figure 3.2 illustrates the average test data volume improvement and the average test sequence length increase of all examined ISCAS circuits. The x-axis represents the window size (i.e. the values of L). The right y-axis (curves) presents the average improvement in test data volume and the left y-axis (bars) presents the average test sequence length increase of window-based reseeding compared to the reseeding case of $L = 1$.

It is obvious that as the window size L increases, the compression improves considerably. However, at the same time, the test sequences grow rapidly and become prohibitively long, especially for large windows. For example, in order to achieve 50% better compression comparing to the LFSR encoding of $L = 1$ we have to use window size of $L = 500$ and thus apply 250× (times) more test vectors. Note that the TSL does not increase proportionally to the window size, due to the reduction of the seed-volume achieved as the window size increases.

3.3 State-Skip Circuit And Proposed LFSR Encoding

In this section we first present the State-Skip circuit and then we propose a window-based LFSR encoding method which exploits the test sequence length reduction offered by the State-Skip circuit and provides short test application time.

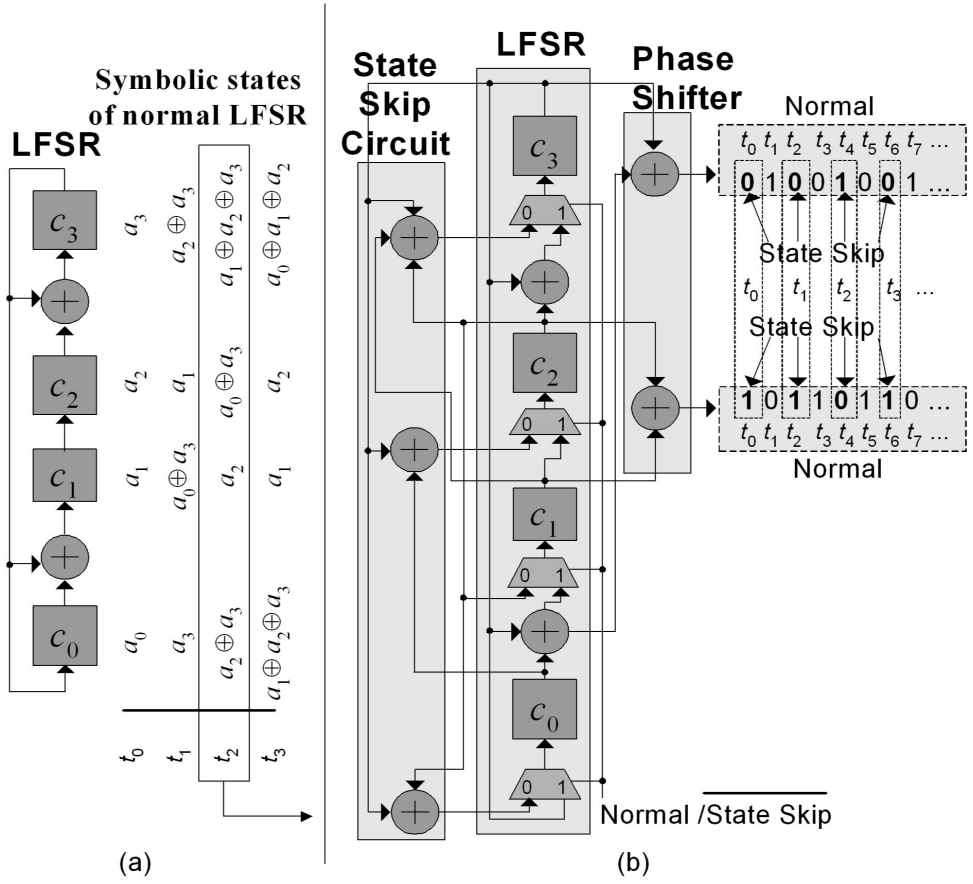


Figure 3.3: Example of ordinary LFSR (a) and State-Skip LFSR (b) for $k = 2$

3.3.1 State-Skip Circuit

Consider the LFSR shown in Figure 3.3a, which consists of four cells c_0, c_1, c_2, c_3, and two exclusive-or gates between cells c_0, c_1 and c_2, c_3 implementing the characteristic polynomial $x^4 + x^3 + 1$. Let the initial state of the LFSR be $(c_0, c_1, c_2, c_3) = (a_0, a_1, a_2, a_3)$. Then the symbolic contents of each LFSR cell during cycles $t_0 \dots t_3$ are shown in Figure 3.3a, next to the respective cell. Let us focus on the contents of the LFSR cells during clock cycles t_0 and t_2. We observe that the value of cell c_0 at cycle t_2 is equal to the Exclusive-OR of the values of cells c_2, c_3 at cycle t_0, i.e., $c_0(t_2) = c_2(t_0) \oplus c_3(t_0)$, where $c_i(t_j)$ is the value of cell c_i during cycle t_j. For the rest cells we derive similar relations: $c_1(t_2) = c_2(t_0)$, $c_2(t_2) = c_0(t_0) \oplus c_3(t_0)$, $c_3(t_2) = c_1(t_0) \oplus c_2(t_0) \oplus c_3(t_0)$. These relations depend solely on the characteristic polynomial and the distance between the clock cycles of interest (2 cycles in the above example). Hence, they are satisfied for every pair of cycles t_{i+2}, t_i, i.e.: $c_0(t_{i+2}) = c_2(t_i) \oplus c_3(t_i)$, $c_1(t_{i+2}) = c_2(t_i)$, $c_2(t_{i+2}) = c_0(t_i) \oplus c_3(t_i)$ and $c_3(t_{i+2}) = c_1(t_i) \oplus c_2(t_i) \oplus c_3(t_i)$. For example, if we set $i = 1$ we can easily verify from Figure 3.3a that the relations are satisfied in this case too. Generally, for an LFSR of size n and for every $k \geq 1$, n linear expressions $F_0^k, \dots F_{n-1}^k$ exist that satisfy the following relations:

$$c_0(t_{i+k}) = F_0^k(c_0(t_i), c_1(t_i), \ldots, c_{n-1}(t_i))$$
$$\vdots$$
$$c_{n-1}(t_{i+k}) = F_{n-1}^k(c_0(t_i), c_1(t_i), \ldots, c_{n-1}(t_i))$$

When $k = 1$, the above expressions represent the operation of the LFSR according to its characteristic polynomial. The linear expressions $F_0^k, \ldots F_{n-1}^k$ are easily calculated by setting $i = 0$ and simulating the LFSR symbolically [equations 3.3.1 are satisfied for every value of i and thus they are satisfied for $i = 0$ too]. Specifically, the LFSR is initialized at symbolic state $(c_0(t_0), c_1(t_0), \ldots, c_{n-1}(t_0))$=$(a_0, a_1, \ldots, a_{n-1})$, and is clocked k times. After the k-th clock cycle, the contents of the LFSR cells $c_0(t_k), c_1(t_k), \ldots, c_{n-1}(t_k)$ are linear expressions of the variables $a_0, a_1, \ldots, a_{n-1}$, which correspond to the initial contents $c_0(t_0), c_1(t_0), \ldots, c_{n-1}(t_0)$ of the LFSR cells, and they constitute the required linear expressions $F_0^k, \ldots F_{n-1}^k$.

The basic idea proposed in this chapter is to integrate $F_0^k, \ldots F_{n-1}^k$ in the LFSR structure. The modified LFSR, which is called hereafter State-Skip LFSR, operates in two different modes, Normal and State-Skip. In Normal mode, the sequence of the LFSR states is generated according to the characteristic polynomial, whereas in State-Skip mode, the state sequence is generated by the integrated linear circuit embedding the $F_0^k, \ldots F_{n-1}^k$ functions. When the LFSR operates in State-Skip mode, it performs a jump of k states ahead at every cycle, skipping in this way the $k-1$ intermediate states which would have been generated if the LFSR had operated in the Normal mode. Therefore, in State-Skip mode, the generated vector sequence is shortened by a factor k, which is called hereafter speedup factor.

Example 3.1. Figure 3.3b presents the State-Skip version of the LFSR of Fig 3a, for $k = 2$. At the input of every LFSR cell, a 2:1 multiplexer selects either the logic value generated by the characteristic polynomial (Normal mode) or the value generated by the State-Skip circuit (State Skip mode). Assuming that the initial state of the LFSR is $(c_0, c_1, c_2, c_3) = 1011$, the logic values generated at the outputs of the phase shifter are shown in the upper right part of Figure 3.3b, for operation either in Normal mode (all logic values inside the grey horizontal bars) or in State-Skip mode (boldfaced and highlighted by the vertical bars). In State-Skip mode only half of the logic values are generated at the outputs of the phase shifter and thus the test sequence is shortened by a factor 2 $(= k)$■

We have to note that the concept of state skipping has been reported in the past [159] in a circular BIST environment, but it exhibits fundamental differences comparing to the proposed State-Skip circuit. Specifically, [159] presents a method to design "state skipping" logic which causes the circular chains to break out of the limit cycles and correlations, and hopefully reach a state with greater potential to detect random-resistance faults. In this way, fault coverage is increased. On the contrary, the proposed State-Skip circuits are systematically designed to perform successive jumps of constant length in the LFSR state sequence, in order to reduce the test application time in a test set embedding environment.

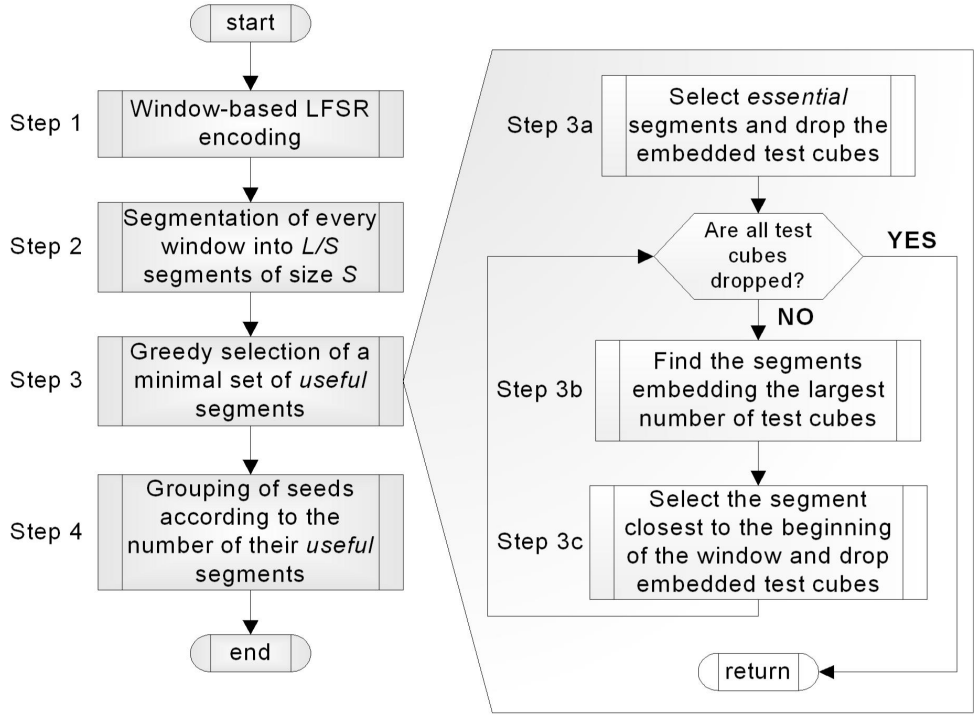


Figure 3.4: State-Skip LFSRs encoding process.

3.3.2 LFSR Encoding using State-Skip Circuits

The flowchart of the proposed encoding method is shown in Figure 3.4. At the first step, the test cubes are encoded using the window-based LFSR reseeding. Every seed is computed so as to encode as many test cubes as possible in the sequence of L successive test vectors (L is the size of the window). There are many ways to encode multiple test cubes in an L-vector window. One very effective algorithm for minimizing the number of seeds has been presented in [33]: initially, the test cube with the highest number of specified bits is selected and the system corresponding to the first vector of the window is solved (the selection of the LFSR polynomial and the phase shifter guarantees that this system is always solvable). The remaining test cubes are selected iteratively according to the following criteria:

1) Among the solvable systems that correspond to the test cubes containing the maximum number of specified bits, we identify those that their solution leads to the replacement of the fewest variables in the L-vector window.
2) Among them, we find those corresponding to the cube that can be encoded the fewest times in the window.
3) Finally, among them, we select the system nearest to the first vector of the window.

These criteria are applied for selecting every new test cube, in the order they appear above. After solving the selected system, some of the variables are replaced by logic values, whereas the rest remain unspecified and they are utilized for encoding additional test cubes. The generation of a seed is completed when no system for any of the unencoded test cubes can be solved in the L-vector window.

Based on the above process, it is obvious that when a test cube t is encoded at the window position i of seed s, then the i^{th} test vector generated by seed s is compatible with t (i.e., they match in all the specified bit positions of t). Usually though, most of the test vectors of a seed do not encode any test cubes. These pseudorandom test vectors can be omitted as they do not contribute to the resulting fault coverage. By using the State-Skip mode to skip the majority of the window positions corresponding to useless pseudorandom vectors, the test sequence length can be considerably shortened.

The highest test-sequence-length reduction can be achieved by skipping every window position that does not encode a test cube. However, this requires complex decoding logic, especially for large window sizes. To this end, we adopt the following cost-effective solution: we first partition the window of test vectors into L/S segments of size S each (step 2 at Figure 3.4), where S is a designer-defined parameter in the range $[1, L]$. Then we examine every segment and we determine if it is useful or useless (step 3 at Figure 3.4). A segment is useful if it embeds at least one test cube not embedded in any other useful segment; otherwise it is a useless one. Note that we do not rely solely on the encoding process to determine if a segment is useful or not. If for example, during the encoding process a test cube is encoded at window position i of seed s, the corresponding segment is not necessarily determined as useful segment. This is based on the following very important property of the window-based reseeding: although, during the encoding process, each test cube is encoded only once, a sparsely specified test cube may fortuitously appear in more than one segment, due to the generation of pseudorandom vectors by every seed (it is very possible for a pseudorandom vector to be compatible with a test cube with very few defined bits). Since there are various sparsely specified cubes in the test sets, this happens quite frequently. We exploit this property in order to minimize the number of segments which are determined as useful, and consequently, the test sequence length.

At the right part of Figure 3.4 we present the selection process of useful segments (step 3) in detail. At first we identify all essential segments. Essential segments are those segments that uniquely embed test cubes, i.e., they include test cubes that are not embedded in any other segment. At this step, essential segments are labeled as useful and all the test cubes embedded in these segments are not further considered. We have to note that, by default, the first segment of every seed is considered as an essential one. This stems from the window-based encoding according to which, the first test cube encoded by every seed is the most specified one and it is encoded at the first position (and thus at the first segment) of the window. Since it is rather unlikely that a densely specified test cube is fortuitously embedded in other segments too, we can safely consider the first segment of every seed as essential. This step simplifies considerably both the selection of the useful segments and the implementation of the decompressor.

After the identification of the essential segments, and provided that there are test cubes not embedded in the selected segments so far, the selection process continues by choosing the minimal number of segments which embed the remaining test cubes. To this end, we adopt the following greedy heuristic: we find those segments that embed the

largest number of the remaining test cubes (step 3b at Figure 3.4) and if more than one such segments exist, we select the one which is closest to the beginning of the window (step 3c at Figure 3.4). Step 3c favors the selection of segments closest to the beginning of the window, since, as it will become apparent soon, this offers additional benefits for test sequence length reduction. Then, we drop the test cubes that are embedded in the selected segment and we repeat steps 3b and 3c until all the test cubes are dropped (i.e., all test cubes are embedded in segments labeled as useful).

We have to note that the test sequence of every seed can be shortened even further, if the generation of the test vectors of every seed terminates immediately after the generation of the last useful segment. In this way the generation of the last useless segments for every seed is completely eliminated instead of just being shortened by using the State-Skip mode of operation. However, this complicates the decompressor, since dedicated logic should be implemented to determine the number of useful segments for each seed. In order to overcome this problem, we follow a different approach (step 4 at Figure 3.4): the seeds are grouped according to their useful segment volume, and the groups are sorted in ascending order of this volume (i.e., group 1 contains all the test cubes with 1 useful segment, group 2 contains all the test cubes with 2 useful segments etc). By applying the seeds in this order the decompressor uses only a counter to indicate the current group and embeds only the functionality required to increase this counter every time the last seed of each group is loaded into the LFSR. At the same time, every seed belonging to group k consists of exactly k useful segments (and $L/S - k$ useless ones). The decompressor keeps track of the number of useful segments applied at the CUT for every seed, and when the last (i.e., the k^{th} useful segment) is applied, it immediately terminates the generation of the test vectors of the current seed and initiates the loading of the next seed from the ATE. Thus the decompression logic is considerably simplified. The following example illustrates the above process.

Example 3.2. Figure 3.5a presents the encoding locations of 12 test cubes ($T_1, \ldots, T_{12}$) into 6 different seeds (each seed corresponds to one row) using window of size $L = 30$ vectors. Every window position which is compatible with a test cube is labeled after the respective test cube (note that some cubes are embedded to multiple locations). Figure 3.5b presents the partitioning of the window of every seed into 6 segments of $S = 5$ vectors. Additionally, the essential segments, which uniquely embed test cubes $T_1, T_3 - T_7, T_9$, and T_{10}, are marked using dashed lines. These selected essential segments are marked as useful and the test cubes embedded in these segments are dropped. The selection process continues since test cubes T_2, T_8, and T_{11} are not yet embedded into any of the selected segments. Figure 3.5c presents the segments embedding the remaining test cubes, as well as the segments selected for covering these test cubes (the useful segments selected at the previous step are denoted with the dark color). Note that segment 3 of seed 3 is marked as useful as it embeds the highest number of remaining test cubes (i.e. T_2 and T_{11}). The segment 5 of seed 5 is the last segment selected as it embeds the last test cube (T_8) and additionally it is closer to the beginning of the window than the other segments

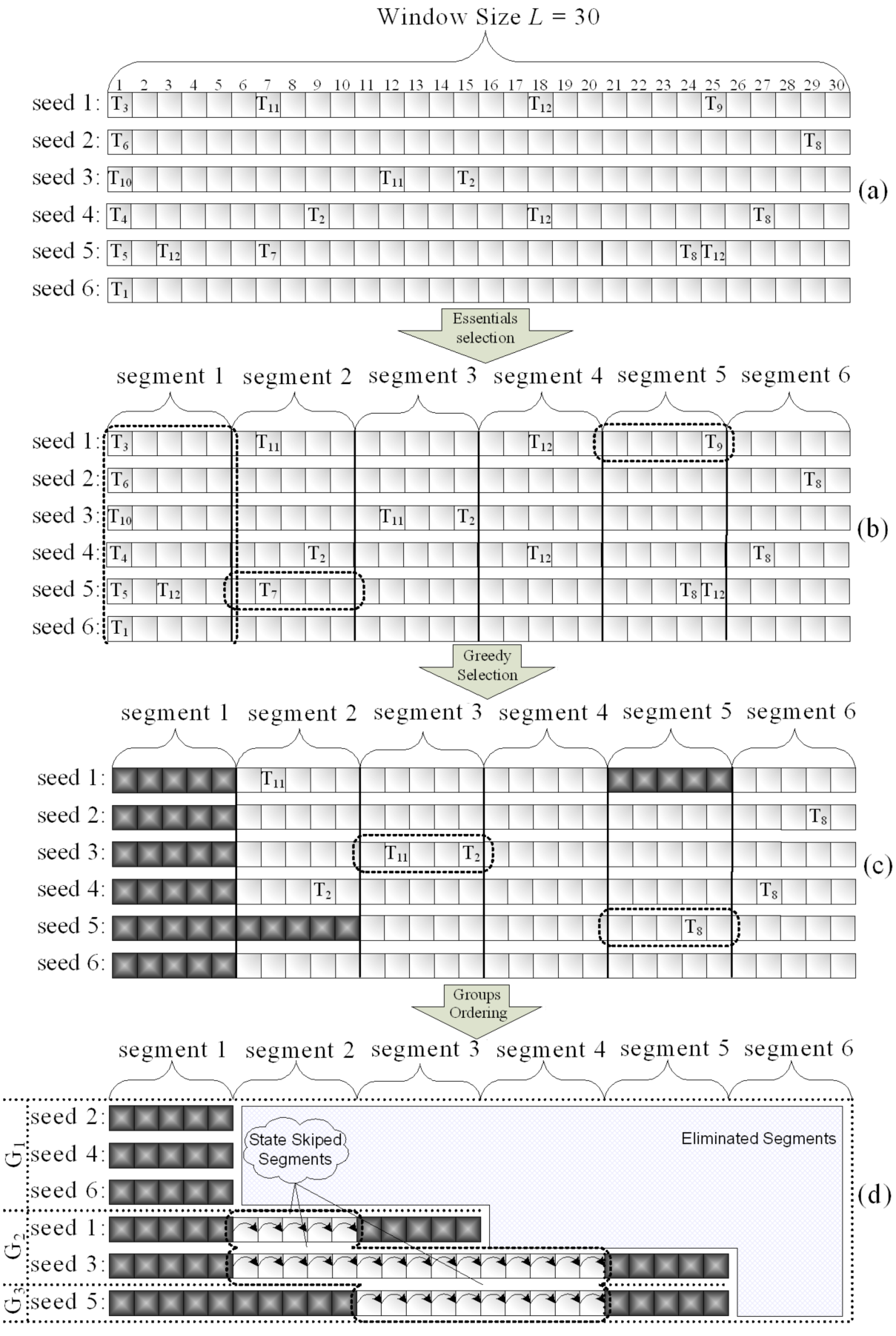


Figure 3.5: Example of State-Skip LFSR encoding.

embedding T_8. Finally, the seeds are grouped according to their useful segments' volume and the groups are sorted in ascending order of this volume. Group G_1 consists of seeds 2, 4 and 6 with one useful segment, group G_2 consists of seeds 1 and 3 with two useful segments and finally group G_3 consists of seed 5 with three useful segments. The last useless segments of each group which are totally eliminated are denoted as "Eliminated Segments", whereas the useless segments which are skipped using the State-Skip mode, are denoted with the arrows inside the vector boxes. Note that only 10 of the 36 segments are normally generated, 6 segments are skipped and the rest 20 are completely eliminated, reducing thus considerably the test sequence length. ■

The efficiency of the described test-sequence-reduction process strongly depends on the value of S (segment size). As it will be shown later, small segments lead to higher test-sequence-length reductions compared to large segments, but impose a little higher hardware overhead than the large ones. In an area constraint design the best value of S should be the smallest one for which the area of the decompressor does not violate the area constraint. Another design problem is that if $S \cdot r$ (r is the scan chain length) is not divided exactly by the speedup factor k, then the last vector of each useless segment will be shorter than the others and it will not completely fill the scan chains. This case can be trivially handled by the control unit of the decompressors with negligible cost. However, if the shift-capture sequence should remain untouched, then $S \cdot r$ and k must be properly selected so as the product $S \cdot r$ to be divided exactly by k.

3.4 Single-State-Skip LFSRs

In this section we present the architecture of the Single-State-Skip LFSRs (SSS_LFSRs). SSS_LFSRs operate as follows:

1) They generate the test vectors of useful segments according to the characteristic polynomial structure of the LFSRs.
2) They skip the useless segments using the State-Skip mode according to a speedup factor k.
3) They totally eliminate the last useless segments of every seed.

The main advantages of SSS_LFSRs are the vast reduction of the test sequence length of window-based reseeding as well as the simplicity and the low hardware overhead of the decompressors. Before describing the architecture in detail, let us present an example which illustrates the potential of this architecture for reducing the test sequence length of window-based reseeding.

Example 3.3. We refer again to the example of Figure 3.5, and we consider a SSS_LFSR with k=5. Suppose that each test vector is generated in c clock cycles. Then the generation of a useful segment using the characteristic polynomial requires 5c clock cycles while the generation of a useless one using the State-Skip mode requires 5c/5=1c cycles. Consequently, the initial test sequence length is equal to $36 \cdot 5c = 180c$ cycles which is

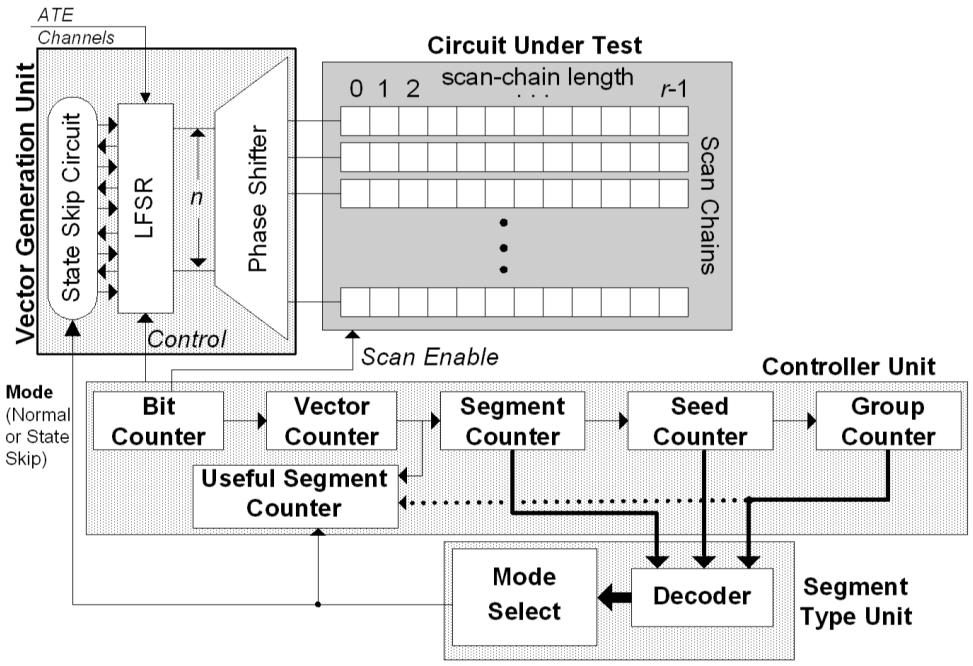


Figure 3.6: SSS_LFSR Decompression Architecture.

reduced to $10 \cdot 5c + 6 \cdot 1c + 20 \cdot 0c = 56c$ cycles using the SSS_LFSRs. This corresponds to 69% reduction of the test sequence length. ■

In the rest of this section we present the SSS_LFSR architecture, the experimental results and finally we will discuss the limitations of this architecture.

3.4.1 Decompression Architecture

The SSS_LFSR decompression architecture is shown in Figure 3.6. It consists of the Vector Generation unit, the Controller unit and the Segment Type unit. The Vector Generation Unit consists of the LFSR, the phase shifter and the State-Skip circuit and it is already described in Section 3.3.1. In the sequel we will describe in details the rest two units.

Controller Unit: This unit controls the operation of the decompressor and specifically the generation of all segments. It consists of the following six counters:

- Group Counter: it counts the number of seed-groups.
- Seed Counter: it counts the number of seeds loaded into the LFSR from the ATE for every seed-group.
- Segment and Useful Segment Counters: they count the total number of segments and the number of useful segments respectively generated for every seed.
- Bit and Vector Counters: they control the loading of the test vectors into the scan chains.

The Group Counter is initialized to the value '1' and retains this value until all seeds of the first group are loaded into the LFSR from the ATE. Every time a new group of seeds is initiated, the Group Counter increases by one. Specifically, when the first seed of each group is loaded into the LFSR the seed counter is initialized to 0 and it increases every

time a new seed of the group is loaded into the LFSR. When the seed counter reaches the last value for each group (i.e., the number of seeds of this group), then the Group Counter is triggered to increase by one in order to begin the processing of the next group of seeds.

As we explained in the previous section, every seed of group i consists of exactly i useful segments. Therefore, every time a new seed is loaded into the LFSR, Useful Segment Counter is loaded with Group Counter's value and thus it is set equal to the number of useful segments of the seed. At the same time the Segment Counter is initialized to 0. For every new segment generated, the Segment Counter increases by one and the Segment Type unit determines if this segment is useful or not. For every useful segment generated, Useful Segment Counter decreases by one. When the Useful Segment Counter reaches 0, Seed Counter increases by one and the next seed is loaded into the LFSR from the ATE. In that way all the last useless segments of each seed are completely eliminated.

Segment Type Unit: This unit consists of the Mode Select block and the Decoder block. The Mode Select block is a combinational circuit that determines if the next segment is a useful or a useless one. It receives the decoded outputs of the Segment, Seed and Group Counters which are provided by the Decoder block and generates the Mode signal that is driven to the Vector Generation unit. Mode signal is equal to 1 (Normal Mode) if the segment is a useful one, else it is equal to 0 (State-Skip Mode). The overhead of this combinational circuit depends mainly on the total number of useful segments which are only a very small portion of the total segments. Moreover, as we noted in Section 3.3.2, the first segment of every seed is always considered as a useful one. Consequently, the first segment of each seed requires minimum decoding logic and therefore the implementation of Mode Select unit is significantly simplified. Additionally, in a multi-core environment, only the Mode Select unit has to be re-implemented for every core, whereas the rest of the units are common for all cores.

Finally, we have to note that in order to avoid the ATE-SoC synchronization problem, a small FIFO has to be inserted between the LFSR and the ATE channels as proposed in [50]. The size of the FIFO, the number of ATE channels used and the frequency of transferring test data can be adjusted in such a way as to avoid FIFO overflow, and thus eliminate the need for sending a synchronization signal back to the ATE.

3.4.2 Experimental Results

The proposed method was implemented in the C programming language and experiments were conducted for the larger ISCAS '89 benchmark circuits, assuming 32 scan chains for each one of them. Uncompacted test sets generated by Atalanta [83] that offer complete stuck-at fault coverage (100%) were used in all cases.

Initially, we study the influence of speedup factor k, segment size S and window size L on the test sequence length (TSL) improvement achieved by the SSS_LFSR architecture.

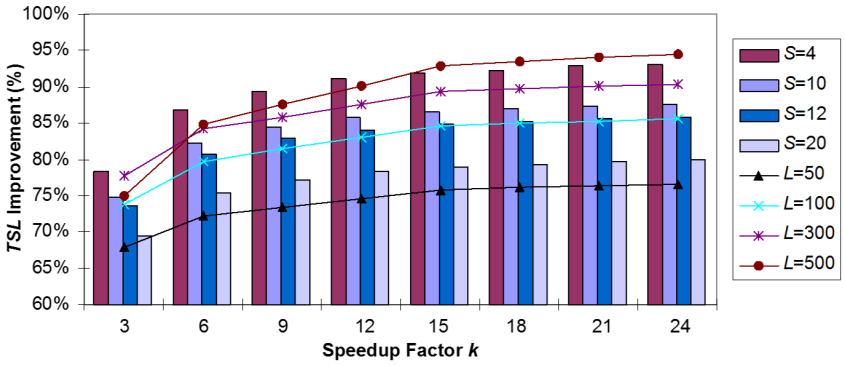


Figure 3.7: TSL Improvement for Various Values of k, S and L.

In every experiment the TSL improvement is calculated by the following formula:

$$TSL \text{ Improvemet}(\%) = \left(1 - \frac{TSL \text{ of prop method}}{TSL \text{ of original window-based method}}\right)$$

To present the trade-offs we focus on s13207, since the rest circuits exhibit similar behavior. In the sequel the TSL is reported as the number of test vectors applied to the CUT and the test data volume (TDV) as the number of bits stored in the tester. Please note that the TDV results of the proposed method are the same with those of the window-based LFSR reseeding approach contained in Figure 3.1, since the proposed method targets only the shortening of the TSL.

Figure 3.7 demonstrates the effect of speedup factor k on the TSL improvement. At first we study this effect for various segment sizes S (we refer to the bars of Figure 3.7). We present results for $3 \leq k \leq 24$, and $S = 4$, 10, 12 and 20, assuming windows of $L = 300$ vectors. From the bars shown in Figure 3.7, it is obvious that the TSL improvement is significant (from 69%-78% for $k = 3$, to 80%-93% for $k = 24$) for all segment sizes. The improvement increases when speedup factor k increases and/or segment size S decreases. When k increases, the number of cycles required for the generation of useless segments decreases, and thus TSL decreases too. When S decreases, the segmentation becomes finer, i.e., the total size of useful segments (in terms of vector count) decreases while the total size of useless segments increases (their sum though remains constant). This is explained by the fact that most useful segments contain also some useless pseudorandom vectors, the number of which depends on size S. By decreasing S, fewer useless vectors remain in the useful segments, and since a useless segment is generated faster than a useful one (its major portion is skipped or it is completely eliminated), the overall TSL decreases.

We next study the effect of speedup factor k on the TSL improvement for various window sizes (L). The curves in Figure 3.7 present the TSL improvement for $3 \leq k \leq 24$ and L =50, 100, 300 and 500 (S was set equal to 5 in these experiments). We observe that as L increases, the TSL improvement increases too. This is explained by the fact that large windows contain more useless segments than the small ones, and the length of useless segments is drastically shortened by the proposed technique.

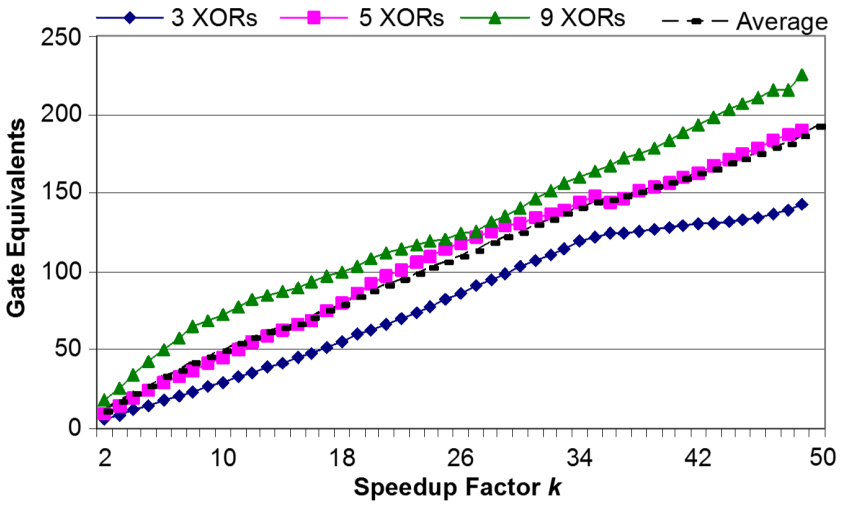


Figure 3.8: Hardware overhead of State Skip Circuit.

As we have already seen, the speedup factor k is critical for the reduction of the TSL. However, k also affects the hardware overhead of the State-Skip circuit. For assessing this overhead, we used a commercial tool for synthesizing the State-Skip circuits of various LFSRs, with primitive characteristic polynomials. Results concerning LFSRs of size 65 with 3, 5 and 9 taps (internal XORs implementing the characteristic polynomial), considering every value of k in the interval $[2, 50]$ are shown in Figure 3.8, where we present the hardware overhead of the State-Skip circuits in gate equivalents (a gate equivalent corresponds to a two input NAND gate). We observe that this overhead: a) increases almost linearly with k and, b) increases as the number of the LFSR internal XOR gates increases. Both these trends are explained by the fact that during every cycle in Normal mode, a number of XOR operations equal to the volume of internal XOR gates are executed by the LFSR. In State-Skip mode, the LFSR executes in one cycle the linear operations of k successive cycles in Normal mode. Therefore, when the number of internal XOR gates and/or the value of k increase, the number of linear operations increases and consequently the hardware overhead of the State-Skip circuit increases too. However, it is obvious from Figure 3.7 that a value of k in the range $[12, 24]$ maximizes the TSL improvement, and, as can be seen in Figure 3.8, the hardware overhead of the corresponding State-Skip circuits is small for all polynomials (between 60 and 100 gate equivalents in the average case). Thus, we conclude that a value of k in the range $[12, 24]$ is a very good choice for SSS_LFSRs.

In Table 3.2 the test sequence length reduction achieved by the SSS_LFSRs for $L =$ 50, 200, 500, S=2, 5, 10, and $5 \leq k \leq 24$ is reported (the best result for the various values of S and k is shown). Columns labeled "Orig." present the test sequence length (# vectors) of the window-based approach with ordinary LFSRs, whereas the columns with label "SSS" present the test sequence length of the window-based approach with SSS_LFSRs. Note that the TSLs of SSS_LFSRs are reported in vector volumes in order to be compared with the window-based approach (i.e., the total number of cycles in each

Table 3.2: TSL improvements

Circ.	L=50			L=200			L=500		
	Orig.	SSS	Impr.	Orig.	SSS	Impr.	Orig.	SSS	Impr.
s9234	9100	1082	88%	32400	1784	94%	76000	3055	96%
s13207	11100	1309	88%	31800	1756	94%	56000	2701	95%
s15850	9500	1129	88%	34200	1740	95%	79500	2791	96%
s38417	29800	7626	74%	113200	13113	88%	276500	21865	92%
s38584	9450	3805	60%	25200	6639	74%	46000	9054	80%

case was divided by the number of cycles required for applying a single test vector to the CUT). Columns labeled "Impr." present the reduction percentage for each case. Note that both approaches (the original and the proposed one) have the same test data volumes (the TDVs of these methods are reported in Table 3.1). It is obvious that the TSL reduction achieved by the proposed method is very high (60%-96%).

Finally, we present the total hardware overhead of the proposed decompressors. We again focus on s13207 (the results for the rest circuits are similar, since apart from the LFSR and the Mode Select unit, the hardware overhead of the rest decompressor does not depend on the test set). The overhead of the State-Skip circuit is very low for the speedup factors of interest ($k \leq 24$). For example, in the case of s13207, as k increases from 12 to 32, the overhead of the State-Skip circuit increases from 52 to 119 gate equivalents. For the same circuit and for various values of L and S, the average total overhead of the rest of the decompressor, excluding the Mode Select unit (i.e., LFSR, Phase Shifter, Controller unit, and Decoder of the Segment Type unit), was 320 gate equivalents. This overhead is very small and similar to that of most test data compression and test set embedding techniques in the literature. Moreover, the aforementioned decompressor units, as well as the State-Skip circuit have to be implemented only once in a SoC and reused for all cores. On the other hand, the hardware overhead of the Mode Select unit, which has to be implemented for every core separately, was between 44 and 262 gate equivalents, for $50 \leq L \leq 500$ and $2 \leq S \leq 50$.

3.4.3 Limitations

The test-sequence length reduction potential of State-Skip circuits cannot be fully exploited by the SSS_LFSR decompression architecture, when multiple non-identical IP cores exist in a SoC. This is mainly attributed to the strong dependence of the TSL reduction on the segment size S, the window size L, and the speedup factor k as it is shown in the previous subsection. The design of a SSS_LFSR has to be based on a single set of values for S, L, k. However, it is unlikely that the TSL of every core will be drastically shortened using the same values of S, L and k. Therefore, the system integrator has to resort to the very expensive solution of using a separate decompressor for each core, for minimizing the overall TSL. On the other hand, if an area efficient solution is required,

a single decompressor must be shared among all cores, which however cannot achieve maximum TSL reduction.

Another limitation of the SSS_LFSR architecture is that, for simplifying the decoder, k should divide exactly the product $S \cdot r$, which is the number of clock cycles required for the normal generation (i.e., not in State-Skip mode) of each segment (each vector requires r clock cycles for loading the scan chains). If this condition is satisfied then a whole segment is traversed by using the State-Skip circuit for exactly $S \cdot r/k$ successive clock cycles and thus the design of the decompressor is simpler. However, since small segments are preferable (usually in the range $[2, 10]$), the maximum value of k that divides exactly the product $S \cdot r$ is bounded by the value of r and thus a large speedup factor may not be possible. This limitation has an even more serious effect when multiple non-identical IP cores should be tested in a SoC. In this case, it is almost certain that the value of r will be different for every core, and thus the probability of finding a suitable value of k for the State-Skip circuit is very low. Thus, the only way to develop SSS_LFSR for testing all cores is to consider the same segment size S for every core and to select a value of k which divides exactly S, so as to satisfy the above condition for every core. In this way though, it is impossible to select a small value for S and, at the same time, a large value for k (this combination offers the highest efficiency). In fact, the values of k and S will be either both large or both small. In the first case, every useful segment will contain many useless pseudorandom vectors and thus the total number of cycles required for its generation will increase. In the second case, the small speedup factor k will not be able to drastically shorten the time required for the generation of the useless segments. Both scenarios negatively affect the performance of the proposed method. In order to overcome these limitations we present in the next section the Variable-State-Skip LFSRs.

3.5 Variable-State-Skip LFSRs

In this section we present a very efficient decompression architecture, the Variable-State-Skip LFSRs (VSS_LFSRs). VSS_LFSRs consist of multiple State-Skip circuits. Each State-Skip circuit implements a different speedup factor, and thus VSS_LFSRs are able to perform jumps of variable lengths. We confine our study in the case of VSS_LFSRs integrating two State-Skip circuits, one with small speedup factor (k) and one with large speedup factor (K), since we observed that two speedup factors are sufficient to achieve very high TSL reduction. Apart from implementing and utilizing multiple speedup factors, the VSS_LFSR architecture is also more flexible compared to the SSS_LFSR architecture, in the sense that both values of k and K can be selected independently of the values S, r, as will be explained shortly. This enables the designer to fully exploit the "test time – hardware overhead" trade-off for single cores, as well as to achieve maximum test-sequence-length reduction for every core that is tested by a common decompressor in a multi-core SoC.

3.5.1 VSS_LFSRs Scheme

A VSS_LFSR with two embedded State-Skip circuits operates in two State- Skip modes: a) K-mode which enables the VSS_LFSR to perform a long jump of K cycles ahead and b) k-mode which enables the VSS_LFSR to perform a short jump of k cycles ahead. Let A be the number of useless segments between two useful segments S_i, S_j ($j = i + A + 1$). The total length (in clock cycles) of these A useless segments is $C = A \cdot S \cdot r$. Then an ordinary LFSR requires C cycles for traversing these useless segments in Normal mode. By using the Variable-State-Skip LFSR, these A segments can be traversed much faster. Specifically, at first K- mode is used (the LFSR performs long jumps of length K) for $C_1 = \lfloor C/K \rfloor$ (successive cycles. Then the remaining part has length $L_1 = C - C_1 \cdot K$ which is smaller than K and it cannot be traversed using K-mode. Thus, VSS_LFSR switches to k-mode (the LFSR performs short jumps of length k) for $C_2 = \lfloor L_1/k \rfloor$ cycles. Finally, the remaining part has length $L_2 = L_1 - C_2 \cdot k$ which is smaller than k and it cannot be traversed using either K or k mode. Consequently Normal mode is used for $C_3 = L_2$ cycles. Note that $C = C_1 \cdot K + C_2 \cdot k + C_3$ as each one of the C_1, C_2 and C_3 cycles corresponds to a jump of length K, k and 1 respectively. Therefore, instead of C cycles, only $C_1 + C_2 + C_3$ cycles, are required for traversing the useless segments. Note that the use of Normal Mode for traversing the last part of the useless segments (the one that its size cannot be divided by either K or k) eliminates the requirement that the values of K and k should divide exactly the product $S \cdot r$.

Example 3.4. Consider a VSS_LFSR architecture with $K = 100$ and $k = 15$ which is used to shorten 60 successive useless segments ($A = 60$) with size $S = 6$ vectors each. Let us assume also that $r = 7$ cycles are required for the application of every test vector at the core under test. Then, we have $C = 2520$, $C_1 = 25$, $C_2 = 1$, and $C_3 = 5$ and thus 31 cycles are required instead of 2520 cycles. For the same values of S and r an SSS_LFSR can be alternatively used with a value of k in the set {2, 3, 6, 7, 14, 21, 42} (note that k must divide exactly the product $S \cdot r$). Even for the highest possible value of $k = 42$, the reduced TSL is 60 cycles which is almost twice as long as that provided by the VSS_LFSR. ■

3.5.2 Decompression Architecture

The VSS_LFSR architecture is shown in Figure 3.9. It consists of four main units:

1) The Vector Generation unit, which comprises the LFSR, the Phase Shifter and the Variable-State-Skip circuit. Signal SelectMode is used to select between the various modes of operation of the LFSR.
2) The Controller, which comprises various counters with the same functionality as in the SSS_LFSR case. This unit controls the operation of the whole decompressor.
3) The Segment Type unit, which consists of a decoder and a combinational logic block, the Segment Type Select block, and determines whether a segment is useful or not. This unit is identical to the respective unit of SSS_LFSR.

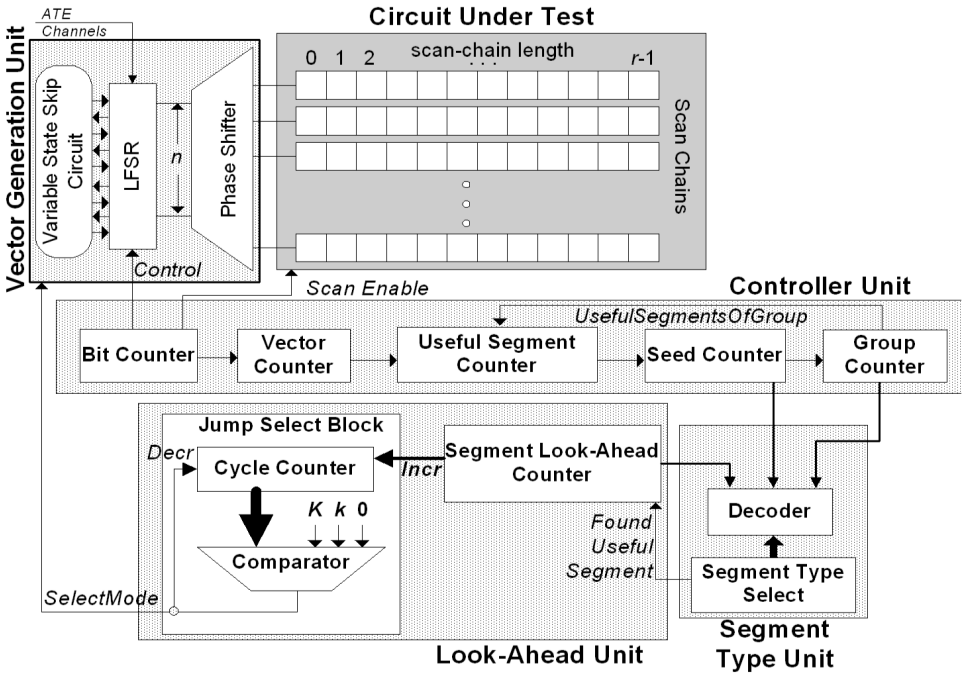


Figure 3.9: VSS_LFSR Decompression Architecture.

4) The Look-Ahead unit, which consists of the Segment Look-Ahead counter and the Jump Select block. It first locates the next useful segment and then it controls the Vector Generation unit so as to traverse the intermediate useless segments in K-mode, k-mode, or Normal mode.

Most of the VSS_LFSR units resemble the units of SSS_LFSR and they have the same functionality (the reader can refer to Section 3.4 for further details). The major difference between the SSS_LFSR and the VSS_LFSR is the Look-Ahead unit. The Look-Ahead unit has two different modes of operation: a) the C-calculation mode and b) the C-skipping mode. The Look-Ahead unit enters the first mode at the beginning of the generation of every useful segment, let say S_i. Then, during the generation of useful segment S_i, it calculates the value of C, i.e., the number of cycles than must be skipped after the generation of useful segment S_i, in order to reach the next useful segment, let say S_j. This calculation is done on the fly, concurrently with the generation of the test vectors of S_i. Specifically, while the test vectors of segment S_i are loaded and applied to the core, the next segments $(S_{i+1}, S_{i+2}, \ldots)$ are examined one by one until the next useful segment S_j is found. For every useless segment found, the value $S \cdot r$ (i.e., its size in cycles) is added to a counter. Consequently, when the next useful segment is found, this counter contains the number C of intermediate clock cycles between S_i and S_j. After the calculation of C, and upon completion of the generation of the test vectors of useful segment S_i, the Look-Ahead unit enters the second mode (the C-skipping mode) and controls the operation of the Vector Generation unit in order to skip the C cycles. Let us now present the operation of this unit in detail.

The Look-Ahead unit is activated at the beginning of seed-group 2 (seeds of group 1 consist of only one useful segment, which is the first one, and thus they do not contain any useless segments). The Segment Look-Ahead counter is reset at the beginning of every seed. The values of this counter are in the range $[0, L/S)$ and correspond to the L/S segments of every window. At the beginning of the generation of any useful segment, let say S_i ($i \in [0, L/S)$), this counter contains the value i (i.e., it points to the currently generated segment). At every successive clock cycle and concurrently with the loading of the test vectors of segment i into the scan chains, the Segment Look-Ahead counter increases by one and examines the segments that follow the useful segment that is currently generated (i.e., $S_{i+1}, S_{i+2}, S_{i+3}, \ldots$) until the next useful one S_j is found. This is indicated by the Segment Type unit, which monitors the value of the Look- Ahead counter. Depending on the values of Seed and Group counters, it responds by setting the signal *FoundUsefulSegment*=0 every time the value of the Look-Ahead counter corresponds to a useless segment, or by setting *FoundUsefulSegment*=1 when the value of this counter corresponds to useful segment. For each increase of the Segment Look-Ahead counter by a step of one, the Cycle counter inside the Jump Select unit increases by a step of $S \cdot r$. When the Segment Look-Ahead counter reaches the value j, then signal *FoundUsefulSegment* is set to '1' to indicate that S_j is the next useful segment. At this point Cycle counter contains the value C and the Look- Ahead unit waits until the generation of the segment S_i (which is currently generated by the Vector Generation unit) finishes.

When the generation of the test vectors of the current useful segment S_i completes, and provided that Cycle counter has non-zero value (i.e., one or more of the next segments are useless), the Look-Ahead unit enters the second mode of operation and, at the same time, the LFSR operation is switched to the Variable-State-Skip mode. Then, the value of Cycle counter is compared against K, and while it is greater than or equal to K, the K-mode is used and the counter is decremented by K (i.e., at every clock cycle, K states of the LFSR sequence are skipped). When the value of Cycle counter drops below K, the above process continues with comparisons against k. While the Cycle counter value is greater than or equal to k, the k-mode is used and the counter is decremented by k (i.e., at every clock cycle, k states of the LFSR sequence are skipped). When Cycle counter drops below k, then the Normal mode is used and the counter is decremented by 1 until it reaches 0 (note that in this case, the LFSR simply passes through the states, i.e., no vector is loaded in the scan chains). At this point the LFSR is already at the first state of segment S_j, and thus the generation of the useful segment S_j begins.

3.5.3 Experimental Results of VSS_LFSRs

In the first set of experiments we study the effect of using two speedup factors K, k on the test sequence length. We focus on the large factor K as it has more profound effect on the TSL. To this end we set $k = 15$ and we vary K in the range $[50, 240]$. Figure 3.10 presents the TSL improvement obtained for the s13207 and s15850 benchmark circuits (the remaining circuits exhibit similar behavior). The segment size used in all cases was

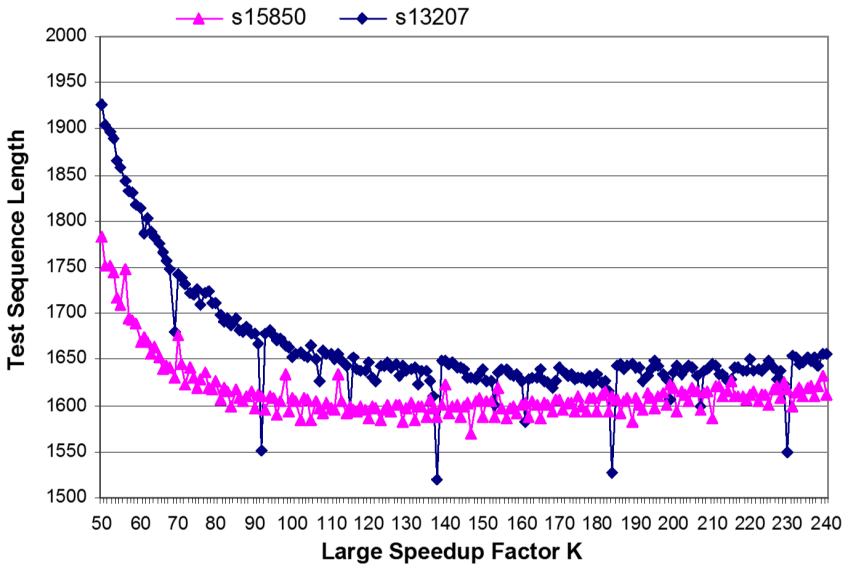


Figure 3.10: TSL using one additional speed up factor K.

$S = 2$ and the window size was $L = 200$. It is obvious that when K increases, TSL reduces. The achieved reduction saturates when K increases above a value, which depends on the core under test. We observe that for some values of K, the test sequence length exhibits large instant drops. The reason is that the corresponding values of K happen to divide exactly the number of cycles required for generating normally a number of, let say m, segments. Consequently, every time the number of useless segments between two useful ones is a multiple of m, these segments are traversed by using only the large speedup factor K (i.e., without using the small speedup factor and/or the Normal Mode of the LFSR).

In the next set of experiments, we study the area overhead of the State-Skip circuits for various speedup factors. Figure 3.11 presents this overhead (reported in gate equivalents) for the examined benchmark circuits (the LFSRs used are the same with those reported in Table 3.1). Note that in Figure 3.11 every examined State-Skip circuit implements one speedup factor. The results in Figure 3.11 are partitioned into two regions separated by the dashed line: the left region corresponds to the State-Skip circuits implementing the small speedup factors k (this case is already studied in Figure 3.8), whereas the right region corresponds to the State-Skip circuits implementing the large speedup factors K. By looking at the left region (small speedup factors k) we can see that the overhead of the State-Skip circuit is low (below 120 gate equivalents in all cases) and increases almost linearly with k. By looking at the right region (large speedup factors K) we see that the overhead is higher compared to the results of the left region, but interestingly it exhibits significant fluctuations (i.e., ups and downs). According to the experiments, this behavior is caused by exactly the same fluctuations in the mean number of binary variables per LFSR cell, during symbolic LFSR simulation. The designer can take advantage of this property and choose a high speedup factor that is near to a local minimum so as to

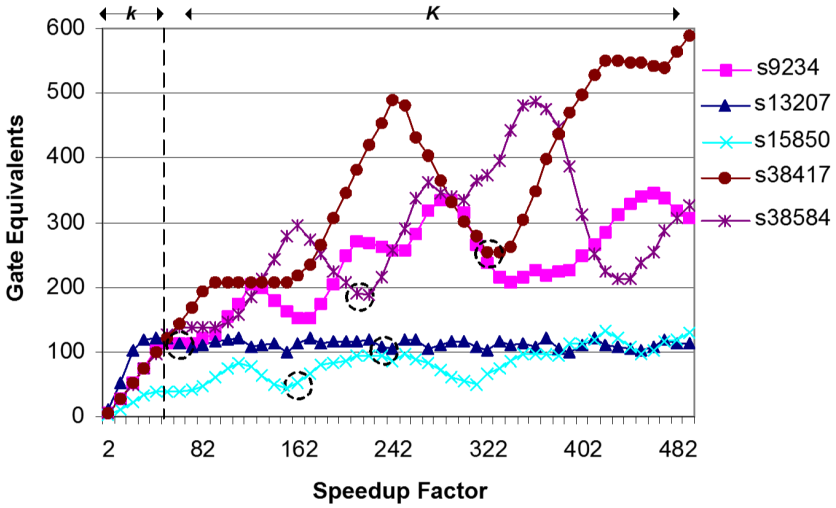


Figure 3.11: Hardware overhead of State-Skip circuits in the range $[2, 500]$.

Table 3.3: TSL improvements of VSS_LFSR architecture

Circuit	L=200				L=500			
	K/k	TSL of VSS	Imp. (%) over		K/k	TSL of VSS	Imp. (%) over	
			Norm.	SSS			Norm.	SSS
s9234	54/18	1465	95.50%	17.90%	165/15	2455	96.80%	19.60%
s13207	230/46	1180	96.30%	32.80%	230/46	1440	97.40%	46.70%
s15850	168/42	1091	96.80%	37.30%	168/42	1470	98.20%	47.30%
s38417	318/53	3026	97.30%	76.90%	318/53	3230	98.80%	85.20%
s38584	235/47	1935	92.30%	70.90%	235/10	1958	95.70%	78.40%

achieve high performance and low area overhead at the same time. The large speedup factors that were chosen for each benchmark circuit are circled in Figure 3.11. Even though the selected values of K are very high (between 54 and 318) the hardware overhead of the corresponding State-Skip circuit is between 50 and 250 gate equivalents, which is rather small. Note that the overall area overhead of the Variable- State-Skip circuits will be higher, due to the addition of the State-Skip circuit implementing the small speedup factor k.

Table 3.3 presents the test sequence length (TSL) results of the VSS_LFSRs and comparisons against the classical window-based LFSR encoding (labeled "Imp. % over Norm") and the SSS_LFSRs (labeled "Imp. % over SSS") for L=200 and L=500. The test data volumes of the cases "Norm.", "SSS" and "VSS" are the same and can be found in Table 3.1 for both values of L. Note that excluding the VSS case, the TSLs of the rest methods can be found in Table 3.2. The first column presents the circuit names. Columns 3 and 7 present the TSL values achieved by the VSS_LFSRs (labeled as "TSL of VSS") for L=200 and L=500 respectively. Note that the TSL of VSS_LFSRs is reported in number of vectors in order to be compared with the rest methods. Columns labeled "K/k" present the values of the large/small speedup factors of the VSS_LFSRs

Table 3.4: Variable VS Single State-Skip for multiple cores

S	SSS-LFSR			VSS-LFSR				TSL
	k	TSL	Hardw. Overh.	K	k	TSL	Hardw. Overh.	Impr. (%)
2	2	53471	9%	318	21	8511	10,5%	84.10%
5	5	31358	7.70%	159	5	15682	8.80%	50.00%
10	10	33736	6.60%	18	10	26731	7.80%	20.80%

that were used in each case [the large speedup factors are near to local area-minimums (see Figure 3.11) in order to keep the hardware overhead low]. It is obvious VSS_LFSRs offer very short test sequences in all cases. Moreover, the test-sequence-reduction ability of VSS_LFSRs is only slightly affected by the utilized window size, giving the designer the opportunity to increase the compression as much as possible with very small test-sequence overhead.

We next present the hardware overhead results of the proposed method for the case of s13207 (the results for the rest circuits are similar, since excluding the LFSR and the Segment Type Select unit, the hardware overhead of the remaining decompressor units does not depend on the core under test). The overhead of the Variable-State-Skip circuit for $k = 46$ and $K = 230$ is equal to 203 gate equivalents. For the same circuit, the total overhead of the LFSR, Phase Shifter, Controller unit, Look-Ahead unit, and the Decoder of the Segment Type unit, for $L = 200$ and $S = 5$, is 627 gate equivalents. All the above units need to be implemented only once in a SoC, where a common decompressor is used for testing different cores. This makes their overall cost much smaller. The only unit that has to be implemented separately for every core is the Segment Type Select unit, whose hardware overhead for s13207 is between 44 and 262 gate equivalents, for $50 \leq L \leq 500$ and $2 \leq S \leq 50$.

In our last experiment we used the SSS_LFSRs as well as the VSS_LFSRs on a hypothetical multi-core SoC consisting of the 5 larger ISCAS '89 benchmarks. In both cases a common decompressor was used and only the Segment Type Select unit was implemented separately for each core. Table 3.4 presents the TSL and area overhead results for three segment sizes, 2, 5 and 10, and for LFSR size=85. The hardware overhead is reported as the percentage of the hardware overhead of the decompresssor to the total hardware overhead of the 5 cores. It is obvious that the TSL gain offered by VSS_LFSRs is very high compared to SSS_LFSR and reaches 84.1%. However, this comes at the expense of a small increase on the hardware overhead (between 1-1.5% of the total area of the 5 cores).

3.6 Comparisons

We will now compare the proposed methods against the most efficient test set embedding and test data compression methods, which are suitable for IP cores of unknown structure. Note that no comparisons are provided against approaches that need structural informa-

Table 3.5: Comparisons of State Skip with other TSE methods

Circuit	Test Data Volume			Test Sequence Length			
	[33]	[91]	SSS, VSS	[33]	[91]	SSS	VSS
s9234	6688	648	6688	24592	135765	3055	2455
s13207	2688	162	2688	24724	152596	2701	1440
s15850	6201	396	6201	27630	222336	2791	1470
s38417	47005	5440	47005	85885	625273	21865	3230
s38584	5152	228	5152	29358	383009	9054	1958

tion of the CUT or require ATPG synergy. Such methods target cores of known structure and thus employ fault simulation, and, most of the times, specially constrained ATPG processes, which reduce significantly and tailor to the encoding method the data that need to be compressed. We mention that for cores of unknown structure neither ATPG nor fault simulation can be performed.

In Table 3.5, the TSL comparisons of the proposed SSS_LFSRs and VSS_LFSRs, for $L = 500$, against the test set embedding approaches of [33] and [91] are presented (comparisons against [67] are omitted, since [33] reports much shorter test sequences than [67] with comparable TDVs). Note that the TDV of both of the proposed methods is exactly the same with that of [33], since the proposed method is a post-processing step on that technique in order to reduce its TSL. The first three columns present the TDV comparisons between the proposed method and the methods of [33] and [91]. The next four columns present the respective TSL comparisons. As can be seen from Table 3.5, both SSS_LFSRs and VSS_LFSRs exhibit very short test sequences compared to both [33] and [91]. The approach of [91] has very small ATE-memory requirements, but its test sequences are extremely long. Moreover, in [33] it is analytically shown that the hardware overhead required for implementing this method is prohibitively large, especially for high scan-chain volumes (estimated between 1300-9800 gate equivalents for 32 scan chains, and 4500-12500 gate equivalents for 64 scan chains, for the larger ISCAS 89 circuits).

Table 3.6 compares the proposed Single-State-Skip and Variable-State-Skip LFSRs for $L = 200$ against the most efficient test data compression methods which are suitable for IP cores of unknown structure and provide results for the ISCAS benchmarks. Note that we omit the comparisons against [20–22, 49, 60, 72, 72, 73, 111, 130, 149, 150] as we compare against [73], which is more efficient than all these methods. Additionally, in order to provide comparisons with dynamic reseeding, we implemented this technique omitting the fault simulation step (in this case, ring generators [106] were utilized). Note that [81, 85, 181] as well as [92, 125] require the same TSLs and for that reason they have been reported under one common column in both cases. In all but one case (s38417) the proposed method performs better than the compared test data compression methods, in terms of test data volume. We have to note though that in the case of s38417 the volume of specified bits is very high (93123 specified bits) and this negatively affects the compression achieved.

Table 3.6: Comparisons with TDC methods*

Circuit	[7]		[81] [85] [181]				[92] [125]			[82]		[138]		Dynamic Reseed.		[74]		SSS_LFSRs (L=200)		VSS_LFSRs (L=200)	
	TSL	*TDV*	*TSL*	*TDV*			*TSL*	*TDV*		*TSL*	*TDV*	*TSL*	*TDV*	*TSL*	*TDV*	*TSL*	*TDV*	*TSL*	*TDV*	*TSL*	*TDV*
s9234	170	15.1	205	12.4	10.3	-	159	30	-	-	-	161	17.2	477	10.6	159	12.8	1784	**7.1**	1465	**7.1**
s13207	229	12.8	266	11.9	10.5	10.8	236	21	74	266	14.3	242	26	536	8.2	236	14,6	1756	**3.8**	1180	**3.8**
s15850	244	15.5	269	12.7	11.4	12.4	126	25	26	226	15.1	306	32.2	524	10.8	126	16.6	1740	**6.7**	1091	**6.7**
s38417	376	37	376	36.4	**32.2**	32.2	99	85	45	376	49	854	89.1	920	55.2	99	58.7	13113	48.1	3026	48.1
s38584	296	31.6	296	30.4	31.2	31	136	57.1	74	296	29	599	63.2	639	21.2	136	55.4	6639	**7.1**	1935	**7.1**

* TDV reported in Kbits (1Kbit = 10^3 bits)

The test sequences of the proposed methods are longer but close to those of the test data compression methods. The test data compression methods offer very short test sequences mainly due to the utilization of compacted test sets, and also due to the fact that they do not apply any pseudorandom vectors. On the contrary, the utilization of uncompacted test cubes and the application of pseudorandom vectors are the main reasons for the prohibitively long test sequences of window-based reseeding. However, the proposed methods shorten the test sequences of window-based reseeding drastically, thus offering test sequence lengths comparable to that of test data compression methods. In fact, in the case of VSS_LFSRs the TSLs achieved are relatively short, especially for the largest ISCAS s38417 and s38584.

Tables 3.5 and 3.6 demonstrate the two options for testing IP cores of unknown structure: test data compression (many data, small test sequences) and test set embedding (few data, greater test sequences). Until now, the test sequences of the latter category of techniques were prohibitively long. State-Skip LFSRs bridge this gap by offering the well-known high compression efficiency of test set embedding with very small test sequences. Taking also into account the high volume of scan chains (a few, fast, internal-clock cycles are required for loading each vector) and the fact that, compared to test data compression, significantly fewer data need to be transferred through the slow ATE-SoC connections in test set embedding, we conclude that the test application time is really not an issue in State-Skip LFSR-based test set embedding, rendering therefore test set embedding a very attractive testing approach.

3.7 Conclusions

Two new types of LFSRs, the Single-State-Skip and Variable-State-Skip LFSRs were introduced, which drastically shorten the test sequences of LFSR- reseeding-based test-set-embedding methods. Single-State-Skip LFSRs offer relatively short test sequences with small area requirements, especially when they are used for testing single cores or multiple identical cores. On the other hand, Variable-State-Skip LFSRs offer higher test sequence length reduction and more flexibility for testing multiple cores, with a relatively small increase in their hardware overhead, which can however be compensated in a multi-core environment. Both types of State-Skip LFSRs bridge the gap between test data compression and test set embedding by offering the high compression efficiency of test set embedding with test sequences reduced to such an amount (up to 98.8%) that approach the length of the sequences of test data compression methods. In this way, test set embedding becomes an attractive approach for testing IP cores.

Chapter 4

Self-Freeze Linear Decompressors for Low Power Testing

4.1 Overview

Even though linear decompressors constitute a very effective solution for compressing test data, they cause increased shift power dissipation during scan testing. Recently, new linear decompression architectures were proposed which offer reduced shift power at the expense however of increased test data volume and test sequence length. This chapter presents a linear encoding method which offers both high compression and low shift power dissipation at the same time. A low-cost, test-set-independent scheme is also described which can be combined with any linear decompressor for reducing the shift power during testing. Extensive experiments show that the new method offers reduced test power dissipation, test sequence length and test data volume at the same time, with very small area requirements.

The method proposed in this chapter offers low shift power dissipation and high compression efficiency at the same time. This technique exploits inherent properties of the test data to provide a fairly simple and low-cost weighted pseudorandom scheme which controls the decompression process and enables the power efficient encoding of test data, without the need of any additional control data. The major advantages of this scheme

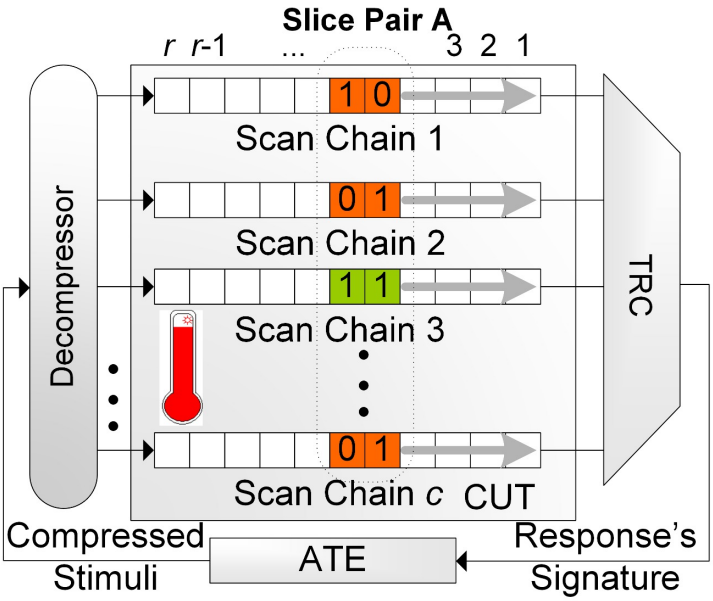


Figure 4.1: Switching activity caused by successive slices

are: a) it constitutes a generic test-set-independent architecture, and b) it can be combined with any linear decompressor scheme for reducing shift power. Moreover, it offers a tradeoff between area overhead and shift power reduction. The combined use of this scheme with the test pattern generator proposed in [105] reduces the test data volume of [105] and achieves great power reductions with very small hardware cost.

4.2 Background

Figure 4.1 presents the classical scan based architecture. The CUT consists of c scan chains of length r (for simplicity we assume that all scan chains are of equal length). The compressed test data are downloaded from the ATE, they are decompressed using the embedded decompressor and they are shifted into the scan chains. For applying a test vector to the CUT the decompressor first generates r successive test slices of size c which are shifted into the scan chains to reach their respective scan slices (hereafter, the term test slice t_j refers to the test bits of test cube t which correspond to scan slice j with $j \in [1, r]$. After the last test slice of t (i.e. t_r) is shifted into the scan chains, t is applied to the CUT and the response is shifted out concurrently with the loading of the next test vector. Linear decompressors fill 'X' values pseudorandomly, and thus they fail to control the number of incompatibilities between successive test slices.

In Figure 4.1, every pair of successive test slices exhibits potential bitwise incompatibilities. These incompatibilities appear between the successive bits loaded into the same scan chains when their value is complementary. For example test slices denoted as "Slice Pair A" in Figure 4.1 are incompatible in the bit positions corresponding to scan chains 1, 2, c. As the test slices travel through the scan chains during the scan-in

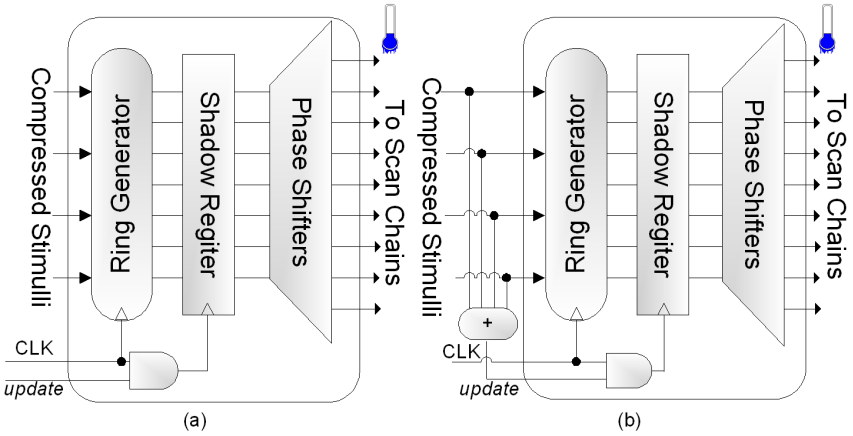


Figure 4.2: (a) Low power PDT controlled by an additional "update" Channel, (b) Low power EDT controlled by compressed stimuli

process, every pair of complementary successive test bits causes transitions in the scan chains which propagate through the combinational logic and cause switching activity to the CUT. The number of incompatibilities between successive test slices can be reduced by exploiting the unspecified values which exist in large volumes in test sets. However, linear decompressors fill 'X' values pseudorandomly, and thus they fail to control the number of incompatibilities between successive test slices.

Recently, the authors of [105] proposed a linear based encoding method which exploits the 'X' values, wherever they exist, to reduce incompatibilities between successive test slices, and thus to reduce shift power. According to this method, whenever a group of k ($k > 1$) successive test slices of a test cube are compatible (i.e., every slice in this group exhibits no bitwise incompatibilities with any other slice in this group) one test slice S_k is computed which is compatible with all k test slices. This slice is encoded using the ring generator and it is loaded into the scan chains for k successive clock cycles. This is achieved by the use of a shadow register shown in Figure 4.2 which can hold its contents if it is properly controlled. Specifically, instead of generating the first slice of this group, the ring generator generates slice S_k and it transfers this slice to the shadow register. This is called UPDATE operation. During the next k successive clock cycles, the shadow register holds its contents and loads the scan chains with slice S_k. This is called HOLD operation. The selection between these two operations of the shadow register requires additional control data which are either provided directly from the ATE (Figure 4.2a) or they are encoded as compressed stimuli (Figure 4.2b). In both cases the additional cost is considerable especially when the number of ATE channels is small and the number of slices per vector is large. The additional control data can be completely eliminated by exploiting inherent properties of the test data. Specifically, during the generation of test slice t_j of any test cube t, the Update operation occurs with a unique probability. This probability depends solely on the test cubes and in particular on the probability a test slice t_j to be incompatible with the test slices corresponding to its predecessor test slices (i.e. t_{j-1}, t_{j-2}, ...) for any test cube t. By controlling the Update operation using predetermined

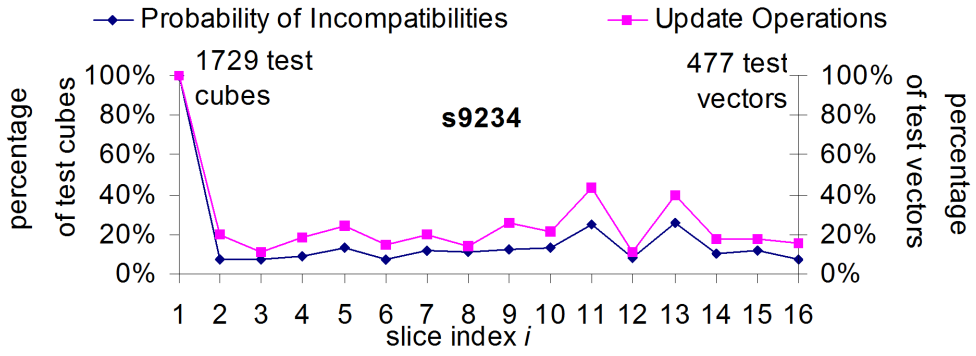


Figure 4.3: Incompatibilities of the test set and UPDATES per slices of FDR

weighted pseudorandom sequences generated by these probabilities, the additional control data are eliminated. Pseudorandomly controlled Update and Hold operations provide high power reduction and they can be easily implemented using embedded low-cost hardware modules. Let as see an example.

Example 4.1. An uncompacted test set for s9234 was encoded using the method proposed in [105], for $r = 16$, $c = 16$. X-axis in Figure 4.3 shows the index of each scan slice. For each scan slice left y-axis shows the percentage of test cubes where the respective test slice was incompatible with its predecessor groups of $k \geq 1$ successive compatible test slices (line labeled "Probability of incompatibilities"). The right y-axis presents the percentage of test vectors which triggered an Update operation at scan slice i (line labeled "Update Operation"). The number of test vectors is smaller than the number of test cubes, as the ring generator encodes multiple test cubes on the same test vector (this elevates a slice's probability of incompatibility with its predecessors). The correlation between these two cases is clear. ■

To estimate the scan power dissipation we will use the metric proposed in [105], which counts the number of invoked transitions in successive scan cells, while taking into account their relative positions. Let t_j^i,t_{j+1}^i be the two successive test bits (between succesive slices that belong on the same scan chain) of test vector t loaded into scan chain i. t_j^i is of scan slice j and t_{j+1}^i is of scan slice $j+1$. The average shift power dissipated is given by the formula:

$$S_{av}(t) = 2[cr(r-1)]^{-1} \sum_{i=1}^{c} \left[\sum_{j=1}^{r-1} (r-j)(t_i^j \oplus t_i^{j+1}) \right] \tag{4.1}$$

4.3 Power Aware Encoding

In this section we will first present the statistical analysis of test data and then we will present the encoding method.

4.3.1 Test Data Analysis

Let TS be a test set of N number of test cubes for testing a CUT with c scan chains of length r (i.e., each test cube consists of r test slices of size c bits). Hereafter, we will refer to every scan cell using its location in the scan chain structure (for example, scan cell (j, i) is the cell located at the scan slice j, scan chain i). Let $N_0(j, i)$, $N_1(j, i)$ be the number of test cubes of TS with logic value '0', '1' respectively at the scan cell (j, i).

Definition 1: The *Zero (One) Fill Rate* of scan cell (j, i) is the probability scan cell (j, i) to be assigned to logic value '0' ('1') for any test cube of TS. The Zero, One Fill Rates of scan cell (j, i) are denoted as $f_0(j, i)$, $f_1(j, i)$ and they are computed as follows: $f_0(j, i) = N_0(j, i)/N$, $f_1(j, i) = N_1(j, i)/N$, with $j \in [1, r]$, $i \in [1, c]$.

Definition 2: The *Zero (One) Fill Rate* of scan slice j ($j \in [1, r]$) is the probability any scan cell of slice j to be assigned to logic '0' ('1') for any test cube of TS. The Zero, One Fill Rates for slice j are denoted as $f_0(j)$, $f_1(j)$ respectively and they are computed using formulas:

$$f_0(j) = \frac{1}{c} \cdot \sum_{i=1}^{c} f_0(j, i), \quad f_1(j) = \frac{1}{c} \cdot \sum_{i=1}^{c} f_1(j, i), \quad j \in [1, r]$$

Theorem 1: The probability two test slices x, y of any test cube in TS to be compatible is given by the formula:

$$P_{SC}(x, y) = (1 - f_0(x)f_1(y) - f_1(x)f_0(y))^c \tag{4.2}$$

Proof: Let x_i, y_i be two bits of test slices x, y corresponding to scan chain i. If x_i, y_i are both specified and complementary then test slices x, y are incompatible. The probability for x_i, y_i to be incompatible is equal to $P_{inc}(x_i, y_i) = f_0(x)f_1(y) + f_1(x)f_0(y)$ and thus the probability for x_i, y_i to be compatible is equal to $P_c(x_i, y_i) = 1 - P_{inc}(x_i, y_i)$. Slices x, y are compatible when all bit pairs (x_1, y_1), (x_2, y_2), ..., (x_c, y_c) are compatible. Thus, $P_{SC}(x, y) = P_C(x_1, y_1) \cdot P_C(x_2, y_2) \cdot \ldots \cdot P_C(x_c, y_c)$, which gives equation 4.2. ■

Lemma 1: The probability a group of k successive test slices j, $j+1$, $j+2$, ..., $j+k-1$ of any test cube in TS to be compatible is:

$$P_{gc}(j, j+1, \ldots, j+k-1) = \prod_{a=j}^{j+k-2} \prod_{b=j+1}^{j+k-1} P_{SC}(a, b) \tag{4.3}$$

Proof: A group of successive test slices is compatible if every two slices in this group are compatible. Thus the probability $P_{gc}(j, j+1, \ldots, j+k-1)$ is equal to the product of the probabilities $PSC(a, b)$ of every possible slice pair a, b ($a, b \in [j, j+k-1]$), thus $P_{gc}(j, j+1, \ldots, j+k-1) = \prod_{a=j}^{j+k-2} \prod_{b=j+1}^{j+k-1} P_{SC}(a, b)$. ■

Let $u_j = 1$ ($u_j = 0$) denote the occurrence of an Update (Hold) operation during the generation of the test data loaded into scan slice j ($j \in [1, r]$. Then the *Update* vector $U = (u_1, u_2, \ldots, u_r)$ represents the Update-Hold operations occurring at the shadow

register during the generation of a vector. Since the first scan slice of each vector has no predecessors we set $u_1 = 1$, that is an Update operation always occurs during the generation of it for every vector. Let t be a test cube consisting of r test slices i.e. $t = (t_1, t_2, \ldots, t_r)$.

Lemma 2: Test cube t is encodable for Update vector $U = (u_1, u_2, \ldots, u_r)$, if for every $j \in [1, r]$, $k \leq r$ with $u_j = 1$ and $u_{j+1} = u_{j+2} = \ldots = u_{j+k} = 0$ $(j + k \leq r)$ test slices $t_j, t_{j+1}, \ldots, t_{j+k}$ are compatible.

Proof: Since $u_j = 1$, during the generation of the test slice t_j the shadow register will be updated from the linear generator with a test slice s_j, and since $u_{j+1} = \ldots = u_{j+k} = 0$ then the same slice s_j will be loaded into scan slices $j, j + 1, \ldots, j + k$. If test slices $t_j, t_{j+1}, \ldots, t_{j+k}$ are compatible then for every $i \in [1, c]$ the test bits of all test slices corresponding to scan chain i are either unspecified or exhibit the same logic value ('0' or '1'). Then, the test slice s_j can be computed as follows: for every $i \in [1, c]$ if any of the test slices $t_j, t_{j+1}, t_{j+2}, \ldots, t_{j+k}$ exhibit a logic value $v =$'0' or $v =$'1' the respective bit of s_j is set equal to v, else it is left unspecified. In that way s_j is compatible with all test cubes $t_j, t_{j+1}, t_{j+2}, \ldots, t_{j+k}$ and thus test cube t is encodable. ■

The most power-efficient Update vector is $U = [1, 0, \ldots, 0]$ which can be used for encoding only those test cubes which have all their slices compatible. On the other hand, the most power consuming but at the same time highly efficient in respect to its encoding ability Update vector is $U = [1, 1, \ldots, 1]$. This vector can encode any test cube which is encodable by the decompressor. In order to maximize the power efficiency of linear decompressors without compromising their encoding efficiency, we need to maximize the volume of zeros in the Update vector of the decompressor and minimize at the same time the probability any test cube to become un-encodable. However, it is rather unlikely that a single Update vector will suffice to encode all test cubes. We will show that multiple Update vectors achieving these goals can be generated in a weighted-pseudorandom fashion.

Let R_j be the probability of an Update operation during the generation of scan slice j ($1 - R_j$ is the probability of a Hold operation during the generation of scan slice j). We denote hereafter as *Pseudorandom-Configuration Vector* or simply as *Configuration*, the probability vector $R = [R_1, R_2, \ldots, R_r]$. Since $u_1 = 1$, we also set $R_1 = 1$.

Theorem 2: The probability any test slice in TS corresponding to scan slice j to be encodable using configuration vector $R = [R_1, R_2, \ldots, R_r]$ is given by the formula:

$$P_E(j) = \sum_{m=1}^{j} R_m \cdot P_{GC}(m, \ldots, j) \cdot \prod_{k=m+1}^{j} (1 - R_k) \tag{4.4}$$

Proof: Any arbitrary test slice t_j corresponding to scan slice j is encodable if either the update operation occurs during the generation of this slice or if the update operation occurs during the generation of a predecessor slice t_k (of the same test cube) and all test slices $t_k, t_{k+1}, t_{k+2}, \ldots, t_j$ are bitwise compatible. Therefore, for slice j we have the

following (also j in number) cases:
1: $P_1 = R_j$ is the probability of an update operation at slice j.
2: $P_2 = (1 - R_j)R_{j-1}P_{gc}(j-1, j)$ is the probability the update operation to occur at slice $j-1$ (and not at slice j) and at the same time test slices $j-1, j$ to be compatible.
...
j: $P_j = (1 - R_j)(1 - R_{j-1}) \ldots (1 - R_2)R_1P_{gc}(1, 2, \ldots, j)$ is the probability the update operation to occur at slice 1 (and not at slices $2 \ldots j$) and test slices $1, 2, \ldots, j$ to be compatible. Thus $P_E(j) = P_1 + P_2 + \ldots + P_j$ which gives equation 4.4. ■

Finally, since every test cube is encodable when all its test slices are encodable, we have that the overall probability P_{ET} for any test cube in TS to be encodable using configuration vector $R = [R_1, R_2, \ldots, R_r]$ is given by formula:

$$P_{ET}(R) = P_E(1) \cdot P_E(2) \cdot \ldots \cdot P_E(r) \tag{4.5}$$

Besides the encoding ability of the decompressor, the Configuration vector R affects also the switching activity during the scan-in process, which is calculated as follows.

Theorem 3: The average scan-in switching activity SC_{av} for any test cube t in TS under Configuration $R = [R_1, R_2, \ldots, R_r]$ is:

$$SC_{av}(R) = \frac{1}{r(r-1)} \sum_{j=1}^{r-1} (r - j)R_{j+1} \tag{4.6}$$

Proof: Let t_j, t_{j+1} be two successive test slices, and let t_j^i, t_{j+1}^i be the test bits of these slices which correspond to scan chain i. Relation 4.1 gives the average switching activity for any test cube t in TS. The term $t_j^i \oplus t_{j+1}^i$ in 4.1 is equal to '1' if t_j^i, t_{j+1}^i are different else it is equal to '0'. Given a Configuration vector R, these bits can be different only if an update operation occurs during the generation of slice t_{j+1}. Since R_{j+1} is the probability of an update operation at slice t_{j+1} and 1/2 is the probability t_{j+1}^i to be generated complementary to t_j^i (assuming linear independent generation) the probability these test bits to be different is $P_{diff}(t_j^i \oplus t_{j+1}^i) = R_{j+1}/2$. Then, relation 4.1 becomes:

$$S_{av}(t) = 2[cr(r-1)]^{-1} \sum_{i=1}^{c} \left[\sum_{j=1}^{r-1} (r - j)(P_{diff}(t_i^j, t_i^{j+1})) \right]$$

and provided that t is generated using configuration R we have:

$$SC_R = 2[cr(r-1)]^{-1} \sum_{i=1}^{c} \left[\sum_{j=1}^{r-1} (r - j)(\frac{R_{j+1}}{2}) \right]$$

which gives 4.6. ■

In the next Section we will give an algorithm to compute the configuration vector $R = [R_1, R_2, \ldots, R_r]$ for any given set of test cubes, which maximizes the switching activity reduction and does not violate a minimum encoding probability $P_{ET}(R)$.

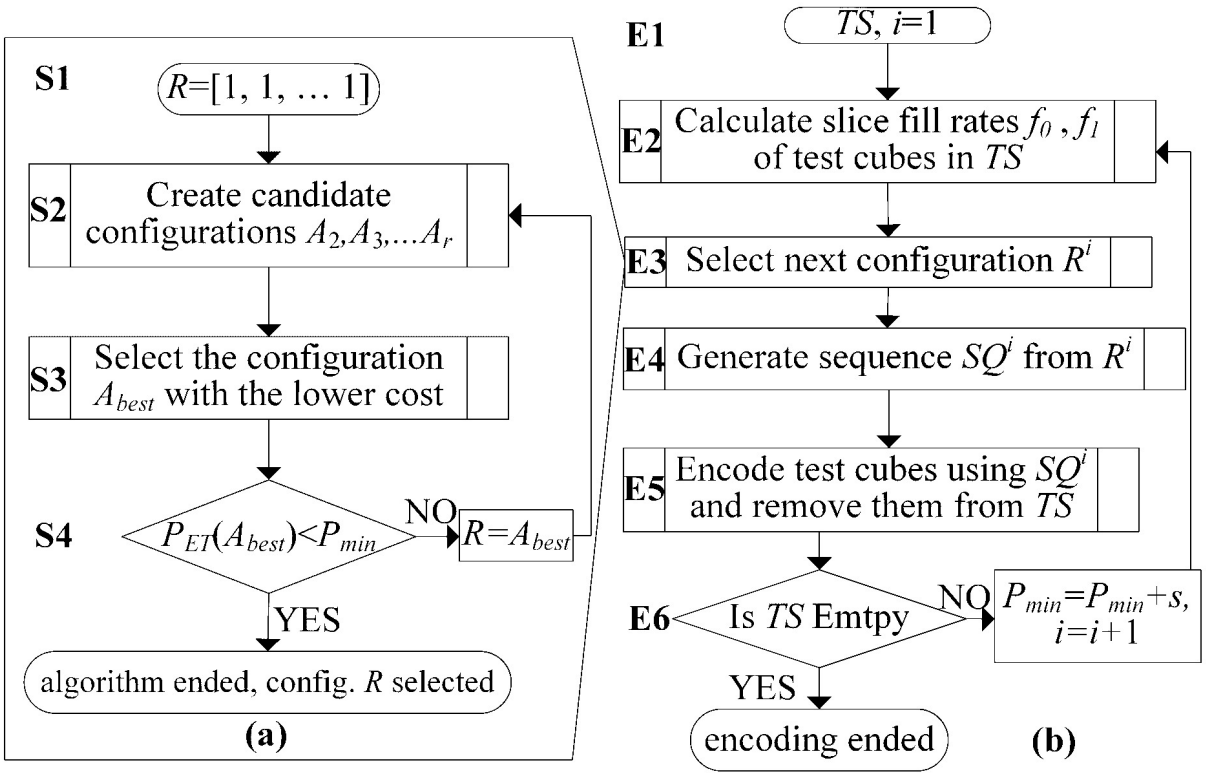


Figure 4.4: a) Configuration selection algorithm, b) Test set encoding

4.3.2 Encoding Algorithm

The flowchart of the proposed encoding method is shown in Figure 4.4 (Figure 4.4a presents step E3 in details). The main target of the encoding method is to calculate the configuration R which offers the minimum average switching activity without compromising the encoding efficiency of the decompressor. This is shown in Figure 4.4a. Specifically, $R = [R_1, R_2, \ldots, R_r]$ is initially set equal to $[1, 1, \ldots, 1]$ which is the configuration offering the maximum encoding probability $P_{ET}(R) = 1$. Then, the values of R_j ($j \in [2, r]$) are iteratively decreased until $P_{ET}(R)$ drops below a pre-determined threshold Pmin or when all $R_2, R_3, \ldots, R_r$ reach their minimum values and they cannot be further reduced. We remind that as the values of R_j decrease, both the average switching activity during scan-in and the encoding probability $P_{ET}(R)$ decrease too.

During every iteration, $r - 1$ candidate configurations $A_2, A_3, \ldots, A_{r-1}$ are generated based on R. Specifically, the candidate configuration A_j ($j \in [2, r]$) is derived from R by decreasing the probability R_j by a predetermined value p (all the other probabilities remain intact). Thus $A_j = [R_1, R_2, \ldots, R_{j-p}, \ldots, R_r]$, with $j \in [2, r]$ (note that R_1 is set always equal to 1). Next, candidate configurations are evaluated using the following formula:

$$CostA_j = \frac{\Delta P_{ET}(A_j)}{\Delta SC_{av}(A_j)} \quad \text{with} \quad \begin{array}{c} \Delta P_{ET}(A_j) = P_{ET}(A_j) - P_{ET}(R) \\ \Delta SC_{av}(A_j) = SC_{av}(A_j) - SC_{av}(R) \end{array} \tag{4.7}$$

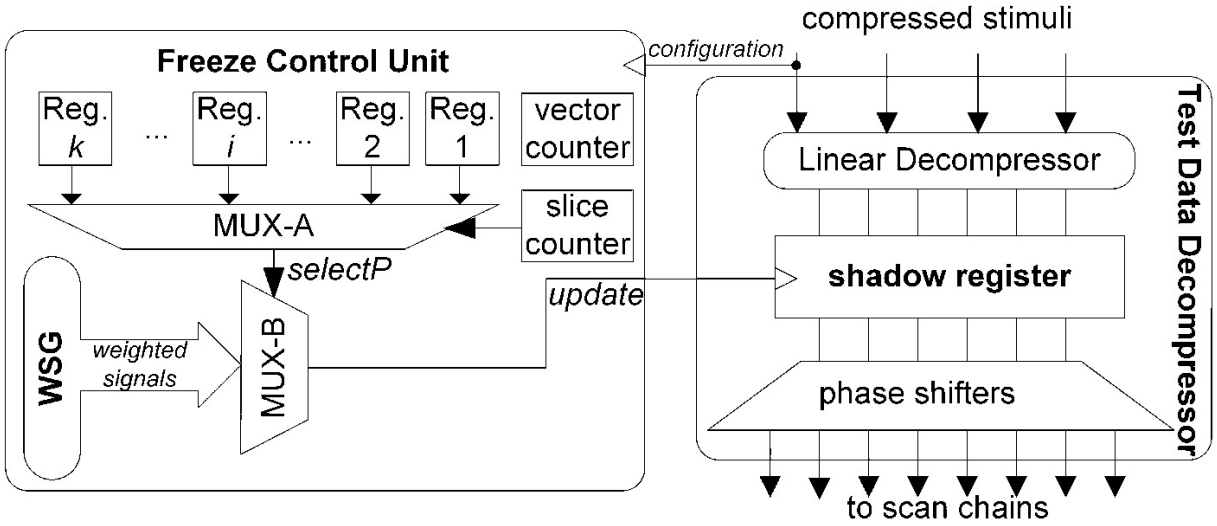


Figure 4.5: Self-freeze architecture

$\Delta P_{ET}(A_j)$ is the reduction of the encoding probability and $\Delta SC_{av}(A_j)$ is the average switching activity reduction of A_j compared to R. The candidate A_{best} with the lower value of $Cost(A_{best})$ is selected and R is set equal to A_{best}.

Usually, one configuration does not suffice to encode all test cubes. Thus, multiple configurations must be generated using the algorithm shown in Figure 4.4b. The algorithm begins with set TS of test cubes and it selects the first configuration, let say R^1 using the algorithm shown in Figure 4.4a. Based on R^1, it generates a weighted pseudorandom bit sequence SQ^1, which controls the Update operation during the decompression process (the generation of this sequence is based on pseudorandom properties of simple hardware modules as we will show in the next section). Using SQ^1 the encoding process attempts to encode as many test cubes as possible and it drops the encoded test cubes from TS. This process is repeated and configurations $R^2, R^3, \ldots$ (and thus sequences $SQ^2, SQ^3, \ldots$) are selected, until TS becomes empty. At each iteration, the value of P_{min} increases by a step s in order to favor the encoding ability of the next configurations and decrease thus their volume, at the expense however of an increase in the switching activity. Relations 4.2-4.7 are recomputed in each iteration using the remaining set of test cubes.

4.4 Architecture

The low power decompression architecture is shown in Figure 4.5. It consists of the Test Data Decompression Unit (TDU) and the proposed Freeze Control Unit (FCU). TDU is a classical decompression architecture and it consists of the linear decompressor, the shadow register, and the phase shifter. Even though any linear decompressor can be used, ring generators [106] were used, as in the case of [105]. FCU generates the update signal which controls the shadow register based on the configuration R (when

update=1 the Update operation is applied). It consists of a set of r registers which store the configuration vector $R_1, R_2, \ldots, R_r$, the slice counter which selects the register for the next generated test slice, and the Weighted Signal Generation Unit (WSG) which generates a set of weighted pseudorandom signals with pre-determined weights. WSG unit generates a set of n pseudorandom signals $WS_0, WS_1, \ldots, WS_{n-1}$ with probabilities $W_0 \leq W_1 \leq \ldots \leq W_{n-1}$ respectively. Specifically, signal WS_i is assigned to logic value '1' with probability W_i and to logic value '0' with probability $1 - W_i$. Depending on the configuration R, register j is loaded from the ATE before the decompression begins with a value d in the range $[0, n-1]$. d selects the input of MUX-B which corresponds to signal WS_d with probability W_d equal to R_j. Slice counter counts from 1 to j and whenever it is equal to j, register j selects signal WS_d which is driven to the update input of the shadow register. Thus the Update operation is applied with probability R_j during the generation of the test data loaded into scan slice j.

Many techniques have been presented in the past for designing WSG units ([5, 174, 175]). A small LFSR which is loaded initially with a randomly selected seed is utilized, and a few AND gates of 2, 3 and 4 inputs driven by the LFSR cells (note that this small LFSR operates only as a pseudorandom generator and it does not participate in the decompression process). Since each LFSR cell is set to the logic value '1' with probability $P_1 = 1/2$, every q-input AND gate produces a weighted pseudorandom signal at its output with probability P_1 equal to $(1/2)^q$. By using three AND gates of 2, 3 and 4 inputs driven by different LFSR cells, and by using both the normal and the inverted outputs of the AND gates, we generate signals with the following P_1 probabilities: 0.0625, 0.125, 0.25, 0.5, 0.75, 0.875, 0.9375. During the encoding process (Figure 4.4a) the values R_j are selected among the P_1 probabilities only for the remaining test cubes. The encoding of the test cubes is done using the pseudorandom sequences generated at the outputs of the WSG unit. After the calculation of a configuration R^i, the WSG unit is simulated and it generates a pseudorandom sequence SQ^i using signals $WS_0, WS_1, \ldots, WS_{n-1}$. The predetermined sequence SQ^i is used for encoding remaining test cubes.

The area overhead of this architecture (Figure 4.5) increases as the number of slices (and thus the number of registers) increases. To overcome this problem an area-efficient alternative architecture will be described which reduces the number of register at the expense of a slight performance degradation. Specifically, k ($k < r$) registers are used and every register corresponds to more than one slices. The registers are assigned to scan slices in a modulo-k fashion. For example, register j is used for controlling the update signal during the generation of scan slices $j, j+k, j+2k, \ldots$ etc (note that scan cell 0 is excluded from this process because an update operation occurs always during the generation of this slice. In this case, the encoding method shown on 4.4a is modified accordingly in order to consider the reduced set $R_1, R_2, \ldots, R_k$. Thus, the process begins with set R where $R_j = R_{j \bmod k}$ and at each iteration k candidate configurations are generated.

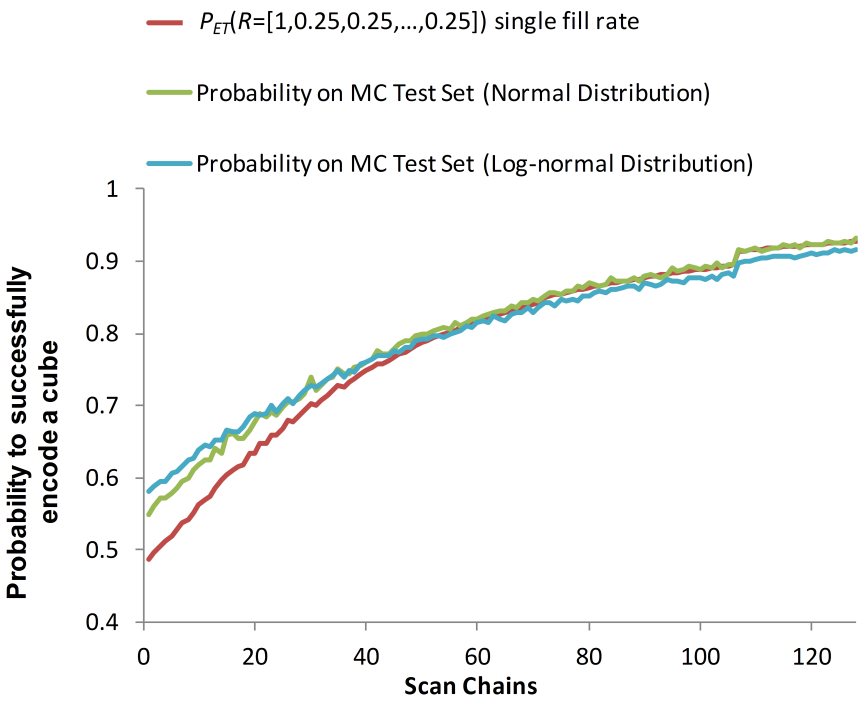


Figure 4.6: $P_{ET}(R)$ metric validation using Monte Carlo generated test sets

4.5 Experiments

The proposed method was developed using the C programming language. We conducted experiments on test sets for complete stuck-at coverage generated using a commercial ATPG tool for the largest ISCAS'89. All the shift power estimations were done using formula 4.1. The proposed method does not negatively impact the complexity of the encoding method since it only requires a constant additional time per test cube for computing the generated pseudorandom sequences by simulating the operation of the freeze control unit.

In order to examine the efficiency of the $P_{ET}(R)$ metric of formula 4.5 we conducted the following experiments. First we examined the efficiency of the metric on Monte Carlo generated test sets based on the fill rate value 4.5% (this is the fill rate of s5378 benchmark circuit). Monte Carlo generated test sets were generated using normal distribution with mean value 0.045 and variance value 0.045. These Monte Carlo generated test sets contain test cubes with normally distributed number of defined values. However, actual test sets contain some test cubes with very high number of defined values, while most of the test cubes have just few defined values. In order to capture this property of real test sets on the Monte Carlo generated test sets, we used a log-normal distribution. The log-normal distribution with mean value 0.045 and variance 0.075 was used. The length of test cubes generated was selected as 218 (the same with the length of the test cubes of s5378) and the configuration used was the $R = [1_1, 0.25_2, 0.25_3, \ldots, 0.25_r]$, where r the number of scan slices. The parameter that changes on this set of experiments is the number of scan chains

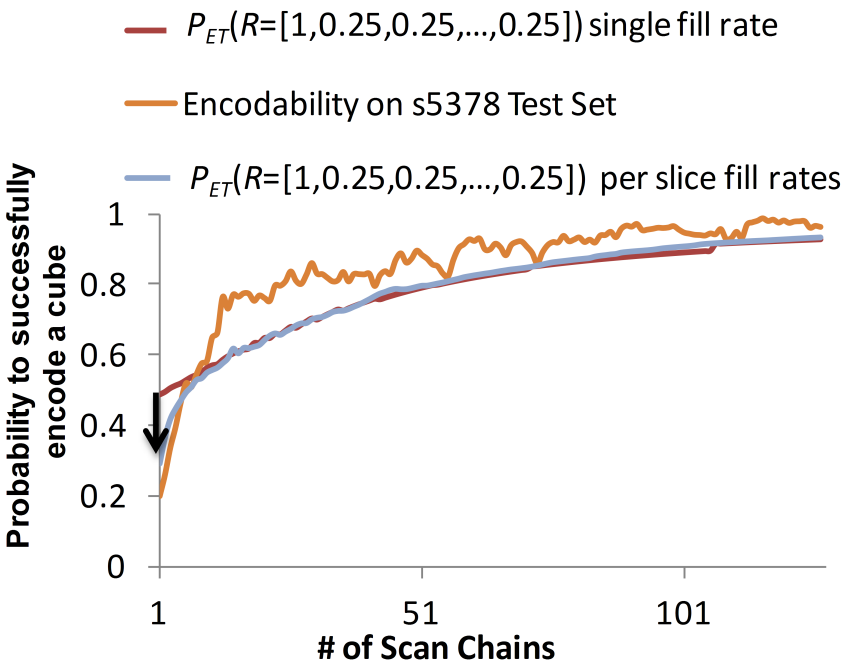


Figure 4.7: $P_{ET}(R)$ metric validation on actual test set (s5378)

c, which also afects the number of scan slices r. The scan chains are selected in the interval $c \in [2, 120]$ and for each instance of scan chains selected c the probability to encode a cube $P_{ET}(R = [1_1, 0.25_2, 0.25_3, \ldots, 0.25_r])$ is computed. Also for each instance, two test sets with 100K random generated test cubes each are created based on the distributions discussed above. The probability to encode a test cube with configuration R on a Monte Carlo generated test set is computed using formula $P = \frac{\#\text{ of encoded test cubes}}{100000}$. Figure 4.6 depicts the results of this experiment. The results show that the $P_{ET}(R)$ metric can succesfully capture the trend of the probability to encode test cubes for a given configuration R. The error of the metric is high for very small number of scan chains because then the number of scan slices increases. Moreover, for small scan chains values c the error becomes high since the metric uses just a single fill rate as statistical information input for the generated test sets and does not exploit scan slices's fill rates.

For the next experiment we applied the metric on the test set of s5378 benchark circuit. The results of this experiment are shown in Figure 4.7. The encodability on specific test sets is computed by the formula $Encodability = \frac{\#\text{ of encoded test cubes}}{\text{test set size}}$. Again the P_{ET} metric labeled as "$P_{ET}(R = [1, 0.25, 0.25, \ldots, 0.25])$ single fill rate" succesfully predicts the trend for large number of scan chains (small number of slices) but exhibits large error for small number of scan chains i.e. large number of slices. In this Figure another metric is shown labeled as "$P_{ET}(R = [1, 0.25, 0.25, \ldots, 0.25])$ per slice fill rates", which is the metric of formula 4.5 using the per slice fill rates of s5378. From the Figure becomes obvious that the usage of slice fill rates decreases considerably the error for the computation of encodability for small number of scan chains c (large number of scan slices r).

Figure 4.8 presents the test data volume (TDV) increase (right y-axis) and the switch-

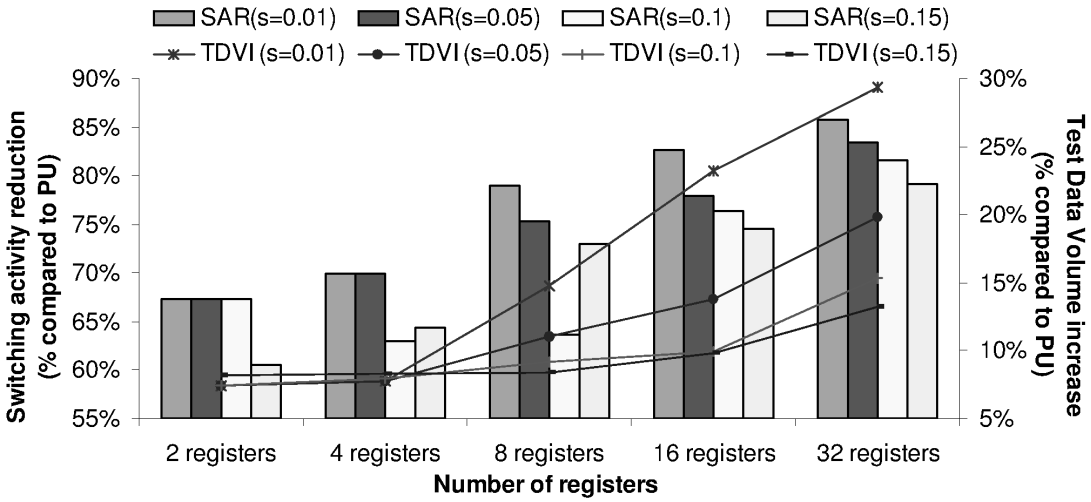


Figure 4.8: Switching activity reduction, test data volume increase trade-off

Table 4.1: Proposed method results. TDV reported in Kbits.

circuit	SA reduction %		TSL			TDV without repeat			TDV with repeat		
	DF	Prop.	PU	DF	Prop.	PU	DF	Prop.	PU	DF	Prop.
s5378	90	78	250	463	392	7	25	10	3	6.3	5.2
s9234	80	67	309	611	419	19	38	26	7	14	10
s13207	96	78	276	432	380	24	57	33	18	28	22
s15850	94	80	293	511	347	22	60	27	17	30	20
s38417	98	85	626	2374	924	65	493	96	33	124	48
s38584	98	82	267	1088	373	49	300	68	37	150	51

ing activity reduction (SAR at the left y-axis) of the proposed technique against the power unaware dynamic encoding (PU) method. In both cases the s13207 benchmark circuit was used assuming $c = 16$ and $r = 44$. The proposed method was applied for 2, 4, 8, 16 and 32 registers and various values of parameter s ($s = 0.01$, $s = 0.05$, $s = 0.1$ and $s = 0.15$). It is obvious that as the number of registers increase, the pseudorandom sequences reflect more accurately the specific requirements of the scan slices and thus the switching activity reduction improves. It is worth noting however, that even a relatively small number of registers suffices to achieve very high reduction of the switching activity. On the other hand, the TDV increases as the number of registers increase, because more data are required for loading the registers for every configuration. In respect with parameter s, it is obvious that small values of s improve the power reduction compared to PU but also increase the test data volume. The reason is that small values of s favor the switching activity reduction at the expense however of generation of more configurations.

Table 4.1 presents the results of a) the proposed technique using 8 registers, b) the power unaware dynamic encoding (PU) and c) the deterministic freeze method (DF) presented in [105] and re-implemented here. We note that in the implemented DF method we assume that the control data are sent from the ATE to the CUT using an extra chan-

nel (Figure 4.2a). In all cases 8 or 16 scan chains and 1 or 2 ATE channels were used (excluding the control channel for DF). Note that both DF and PU methods were implemented by omitting the fault simulation step in order to provide fair comparisons with the proposed method (the fault simulation step can be trivially included in all cases). The first column presents the circuit's name, while the next two columns present the average switching activity reduction of both DF and the proposed method against the power unaware method (PU). The next three columns in Table 4.1 present the test sequence length of the PU, the DF and the proposed technique. The results indicate that the proposed technique achieves a vast reduction of the average switching activity (67%-85%). Note that, although the switching activity of the proposed method is larger compared to DF, this is attributed to the high TSL of the DF method. The large TSL of DF method is a consequence of its trend to minimize the volume of Update operations and to limit thus the ability of the decompressor to encode multiple test cubes on the same generated vector. As a result, the number of generated vectors (i.e. the TSL) increases considerably especially for large test sets. Nevertheless, the switching activity of the proposed technique remains significantly lower than PU and thus the probability to comply with the functional power budget of the CUT (which is the most important target of any low power testing technique) is still very high.

The next six columns present the test data volume (TDV) comparisons between the DF and the proposed method. The first three of these columns report the TDV results assuming that the repeat command is not supported by the ATE, while the next three report the TDV results assuming that the repeat command is supported by the ATE. As it has already been mentioned in [105] the use of the repeat command considerably reduces the TDV. The proposed method achieves very high TDV reduction against DF in both cases (in the range of [30%-81%] whenever the repeat command is not supported and in the range of [17%-66%] whenever the repeat command is supported).

Finally, we synthesized the proposed scheme for 8 registers. The hardware overhead of the proposed FCU unit is less than 100 gate equivalents (one gate equivalent corresponds to a 2-input nand gate). This overhead is less than the 25% of the overhead of the TDU unit. Additionally, we note that the same decompressor can be used for testing any number of cores, which makes its application very attractive to modern SoCs.

4.6 Conclusions

A new linear encoding method which exploits inherent properties of test data to reduce the scan-in switching activity during testing was presented. A low-cost embedded scheme was also presented which can be combined with any linear-decompressor architecture and achieves very high reduction of the switching activity at the expense of only a small increase on the test data volume. Compared to the state-of-the-art power aware linear encoding method, the method described in this chapter provides comparable shift power reduction with considerably lower test data volume.

Chapter 5

Defect Aware X-Filling for Low-Power Testing

5.1 Overview

Various X-filling methods have been proposed for reducing the shift and/or capture power in scan testing. The main drawback of these methods is that X-filling for low power leads to lower defect coverage than random-fill. We propose a unified low-power and defect-aware X-filling method for scan testing. The proposed method reduces shift power under constraints on the peak power during response capture, and the power reduction is comparable to that for the Fill-Adjacent X-filling method. At the same time, this approach provides high defect coverage, which approaches and in many cases is higher than that for random-fill, without increasing the pattern count. The advantages of the proposed method are demonstrated with simulation results for the largest ISCAS and the IWLS benchmark circuits.

In this chapter, a new X-filling technique is presented that achieves the following goals:

1) It provides substantial reduction in shift power during scan testing, close to that obtained using Fill-Adjacent X-filling.
2) It ensures that the capture switching activity is less than a pre-determined limit.
3) It provides increased defect coverage, which approaches and even outperforms in many cases, the random filling of Xs, without increasing the test pattern count.

Table 5.1: FA and Proposed X-Filling*

	Test Cube Block	FA	MFA
i	0x...x0, 0x...x, x...x0	00...00	00...00
ii	1x...x1, 1x...x, x...x1	11...11	11...11
iii	0xx...x1	011...11	011...11, 001...11, ..., 000...01
iv	1xx...x0	100...00	100...00, 110...00, ..., 111...10

*the rightmost bit is loaded first into the scan chain

4) It offers a tradeoff between power efficiency and defect coverage and thus it can be adjusted to the specific requirements of a design.

The new method exploits different ways of filling the Xs, and selects the most effective one with respect to defect coverage and shift power, under constraints on the capture power. High defect coverage is ensured by the use of a surrogate metric based on output deviations [178], for evaluating the quality of test vectors. Output deviations provide an efficient probabilistic means to evaluate test vectors based on their potential for detecting arbitrary defects and, most importantly, without being biased towards any particular fault model. As shown in [169], unbiased testing provides higher test quality than a test method that is biased by a particular fault model. The efficiency of the new method is demonstrated through experiments with the ISCAS and IWLS [1] benchmark circuits. To the best of our knowledge this is the first X-filling method that achieves power reduction and high defect coverage in a unified manner.

5.2 Modified Fill Adjacent - the new X-Filling Method

The new method generates multiple power-efficient candidate test vectors by filling the Xs of each test cube in multiple power-efficient ways. The candidate test vectors for each test cube are evaluated with respect to their potential for detecting defects, using an output-deviation based quality metric, and the most efficient one is selected. In this section, we describe first the process of generating the candidate test vectors and then the selection of the most efficient ones.

5.2.1 Generation of Power Efficient Candidate Test Vectors

In order to reduce the average shift power for the candidate test vectors, a modified version of the FA technique, hereafter called MFA, is presented. MFA fills the Xs of a test cube in multiple power efficient ways by compromising only a very small portion of the shift power efficiency offered by FA technique. Specifically, as it is shown in column 2 of Table 5.1, only the blocks of types *iii*, *iv* cause scan-in switching activity when they are filled according to FA technique because they contain one pair of consecutive complementary test bits. FA fills the Xs in such a way as to locate every such pair at the leftmost position of each block in order to minimize the distance that this pair has to travel during scan-

Table 5.2: X-Filling for test cube T=xxx1xxx0xxx0xxxxx1

	MFA	**MFA+20**
Random Fill	Moderate SA	Moderate SA
010110100110101001	111111000000001111	101110000000001111
PSI(T): 75.16%	PSI(T): 13.1%	PSI(T): 15%
FA	Worst SA	Worst SA
111100000000111111	111111100000000001	010111100000000001
PSI(T): 10.5%	PSI(T): 15.7%	PSI(T): 19.6%

in (note that the leftmost position is loaded last). To retain a low power profile of the candidate test vectors, MFA fills blocks of types *i* and *ii* in the same way as FA, while for blocks of types *iii* and *iv*, MFA allows the pairs of consecutive complementary test bits to be located at any point relative to the scan output; see Column 3 of Table 5.1. Consequently, FA can be considered as a special case of MFA. For any block of either type *iii* or *iv* consisting of n unspecified bits, $n+1$ different fillings exist according to MFA, and for any test cube consisting of m such blocks with $n_1, n_2, \ldots, n_m$ unspecified bits each, $(n_1+1)\cdot(n_2+1)\cdot\ldots\cdot(n_m+1)$ different candidate vectors can be generated. Note that as we move from the first to the last filling of MFA shown in Table 5.1 at both *iii*, *iv* types of blocks, scan-in switching activity increases because the pair of complemented test bits travels a longer distance in the scan chain.

Example 5.1. Table 5.2 presents a hypothetical test cube that is filled a) randomly, b) using FA, and c) using MFA. In order to evaluate the scan-in switching activity, $P_{SI}(T)$, of every test cube T generated using each of these fillings, we use the normalized weighted switching activity [105]. This metric counts the number of transitions in successive scan cells, taking also into account their relative positions, and normalizes this value by dividing it by the upper bound of the volume of switching flip- flops. The values of this metric are in the range 0% (no switching activity) to 100% (all flip flops are switching at every cycle). In the first column of Table 5.2, we show a potential random filling of the cube, the filling provided by FA technique, and their respective $P_{SI}(T)$ values. It is obvious that FA causes less switching activity than random filling. In the second column, we present two different X-fillings using MFA: one filling with moderate scan-in switching activity and one filling with the highest scan-in switching activity that can be possibly generated by MFA (i.e., for every block of either type *iii* or *iv*, the last combination shown in the third column of Table 5.1 is used). We can see that even in the worst case, the scan-in switching activity of the MFA method is only slightly higher than that of FA, while it is still much lower than that of random filling. ■

Even though the shift power of MFA is only slightly higher compared to FA, the test vectors generated using MFA exhibit significant differences with respect to their potential to detect un-modeled defects. The magnitude of these differences depends mainly on the diversity of these vectors, which is greatly affected by the way the Xs are filled. In order

to increase the diversity of the candidate test vectors, a step that slightly increases the switching activity is required. This step is used sparingly in our X- filling technique. We randomly fill a small and carefully selected portion of the Xs of each test cube. Specifically, for every test cube segment loaded into any scan chain, we fill randomly the Xs corresponding to the leftmost scan cells (i.e. the scan cells that are closer to the input of the scan chain) that make a small contribution to the scan-in switching activity (they travel the shortest distance in the scan chains during scan-in). Thus, depending on a user-defined parameter P, all Xs corresponding to the $P\%$ leftmost scan cells of every scan chain are filled randomly. As P increases, the defect coverage of the test vectors increases but they consume more shift power. Thus, P offers a tradeoff between scan-in switching activity and defect coverage. This enhanced version of MFA is called MFA+P. Note that MFA is a special case of MFA+P with $P = 0\%$.

Example 5.2. In Column 3 of Table 5.2, we present two fillings, one with moderate and one with the highest possible scan-in switching activity, using MFA+20 ($P = 20\%$). It is obvious that the scan-in switching activity is increased compared to MFA but it is still much lower than that for random fill. ■

It has been observed that the FA technique adversely affects the peak capture power, which may even be higher than the peak capture power for random filling [19]. To eliminate this problem in the MFA method, we invoke the Preferred Fill (PF) technique [128, 129] for specifying as many Xs as necessary in order to limit the peak capture power under the power budget. This is done in a stepwise fashion and concurrently with the application of MFA/MFA+P technique in order to minimize the number of Xs specified according to PF. The capture power is measured as the Hamming distance between the test vector and the first response (this pair always exhibits the peak power as noted in [129]. Other, more sophisticated metrics can be also used. The functional limit on peak capture is considered as a maximum number L of scan cells switching during capture.

The complete flow is shown in Figure 5.1. The goal of this process is to generate a set $CS(t)$ of at most N candidate test vectors (N is a constant value pre-determined by the designer) for every test cube t. At first one test cube t of test set TS is selected, and it is filled using solely MFA or MFA+P (i.e., PF is not applied yet) in order to generate $N \cdot C$ candidate test vectors (C is also a constant pre-determined by the designer). All these $N \cdot C$ candidate test vectors are checked for the violation of the peak capture power limit, and the test vectors that violate this limit are discarded. The remaining test vectors are inserted into the set of candidate test vectors $CS(t)$. If these vectors are more than N, then N of them are randomly selected else the PF technique is invoked to provide additional test vectors as follows: at first the 10% (arbitary selected step value, it can be selected according to designer needs) of the Xs of the test cube t which are the most highly potential to reduce the peak capture power according to PF are specified. Then again $N \cdot C$ candidate test vectors are generated using the MFA or MFA+P for the modified test cube t and these test vectors are checked for violating the peak capture power limit. Again the test vectors that do not violate this limit are appended into set $CS(t)$. If

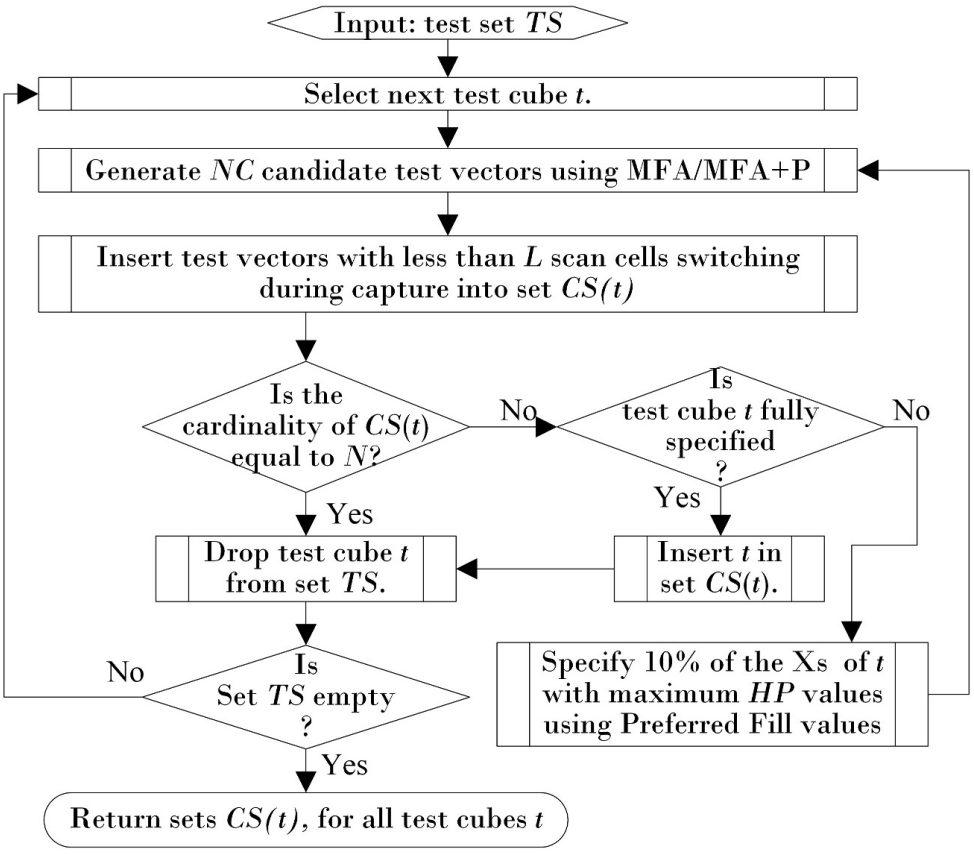


Figure 5.1: X-Filling Flow for MFA/MFA+P

$CS(t)$ at this step still consists of less than N candidate test vectors, then the same flow is repeated by specifying another 10% of the Xs of test cube t with the highest possibility to reduce the peak capture power. When either $CS(t)$ contains N test vectors or test cube t is fully specified by PF, the generation of $CS(t)$ stops and the process continues with the next test cube. C and N values affect the execution time of the proposed method as well as the quality of the result. The larger they are the better the results are and the longer is the execution time. In the experiments section more details follow about these parameters value selection.

5.2.2 Evaluation and Selection of Test Vectors

The candidate test vectors $CS(t)$ for every test cube $t \in TS$ are evaluated using an output-deviation-based quality metric and the best test vector is selected for every test cube. This metric is an advanced version of the quality metric proposed in [69] as it evaluates the defect coverage potential of a test vector using concurrently both its first and its second test response (we consider LOC scheme). Thus the proposed metric targets both timing-dependent and timing-independent defects at the same time. The quality metric exploits the following properties:

1) For every candidate test vector v, the output deviation values (see Chapter 7.2,

Section 2.1.7) for both responses are calculated. Then, the outputs where the deviations reach their highest values among all candidate vectors are the most promising for detecting defects (the rest are not further considered). These outputs are partitioned into four sets for each vector v as follows. For the first response of vector v, the outputs with maximum deviation values and fault free logic value 0 (1) form set $MS_0(v,0)$ ($MS_0(v,1)$). The respective outputs for the second response form sets $MS_1(v,0)$ and $MS_1(v,1)$.

2) Every circuit output is weighted according to its potential to detect defects. This weight depends: a) on the amount of logic at the fan-in logic cone of the respective output (more defects can be potentially observed at the outputs of the large cones than at the outputs of the small ones), b) on the fault-free response vaue at each output (it considers the possbility to detect different defects at every output according to different fault free logic values) and c) on the volume of potential defects at each cone which are not yet detected by previously selected test vectors at the respective output. This volume is tracked by considering the number of previously selected test vectors which maximize the deviation at this output. The higher this number, the higher is the expected volume of defects already detected at this output, and thus the lower is the volume of defects remaining to be detected (at this output).

Based on the above properties, the evaluation process is conducted as follows. Initially, a set CS of all candidate vectors is generated as the union of sets $CS(t)$ for all $t \in TS$. Then, one pair of weights is assigned at each circuit output i corresponding to the first response assuming both fault free logic values 0, 1, that is $wo_0(i,0)$ and $wo_0(i,1)$ respectively, and another pair of weights is assigned at each circuit output i corresponding to the second response and fault free logic values 0, 1, that is $wo_1(i,0)$ and $wo_1(i,1)$ respectively. All these weights are initially set equal to the number of lines (nets) in the fan-in logic cone of the respective output. Next, for every test vector $v \in CS(t)$ the output deviation values are computed and the sets $MS_0(v,0), MS_0(v,1), MS_1(v,0), MS_1(v,1)$ are generated. Then the following process is repeated and at each repetition one test vector is selected.

At first the next formula is used to compute the quality metric of each vector:

$$WT(v) = \sum_{i=0,1} \sum_{j=0,1} \sum_{k \in MS_i[v,j]} wo_i(k,j)$$

Intuitively, $WT(v)$ is the sum of the weights of all outputs which have maximum deviation value at the first and/or second response when test vector v is applied. Among the evaluated test vectors, the one with the highest value of this metric is selected since it is the most promising one for defect detection. After the selection of vector v, the remaining vectors of set $CS(t)$ are discarded from set CS and the weights $wo_0(k,0)$ for all $k \in MS_0(v,0)$, $wo_0(k,1)$ for all $k \in MS_0(v,1)$, $wo_1(k,0)$ for all $k \in MS_1(v,0)$ and $wo_1(k,1)$ for all $k \in MS_1(v,1)$, are divided by a constant factor F_2 (these outputs are expected to detect many defects, after the application of test vector v, and thus they are considered as less effective for the selection of the next vectors). As proposed in [69],

Table 5.3: Total average power reduction (% compared to RF)

circuit	# cubes	FA	FA*	MFA	MFA+10	MFA+20
s5378	134	51.39	37.68	45.93	30.3	25.74
s9234	166	36.96	33.91	34.64	31.36	27.11
s13207	269	43.63	41.11	42.2	39.99	37.41
s15850	162	49.97	49.86	49.93	45.55	41.9
s38417	143	55.31	55.47	54.94	51.21	48.02
s38584	185	49.5	49.08	49.29	45.21	40.94
ac97_ctrl	66	46.32	46.32	45.56	42.94	39.04
mem_ctrl	603	59.65	59.5	58.37	53.31	47.59
pci_bridge32	298	55.93	56.14	55.61	51.35	46.73
tv80	757	59.84	59.87	58.88	53.94	50.45
usb_funct	136	38.84	36.28	36.74	34.73	32.47
ethernet	1113	73.4	73.47	73.17	66.01	58.68

the value of F_2 was set equal to 8. Then, the new weights $WT(v)$ are calculated for all remaining vectors v, and the next vector is selected.

5.3 Experiments

The simulation platform was developed using the C language and the power simulations were done using commercial tools. We report total power values that include the power consumed in the circuit (scan-in, capture, scan- out). We conducted experiments on the largest ISCAS'89 circuits and a subset of the IWLS'05 circuits [1] for multiple scan chains. All methods were applied on dynamically compacted test sets generated using a commercial ATPG tool for complete stuck-at fault coverage. Please note, that the execution time of the proposed method is very fast because its complexity is linear. For a single threaded implementation the overall complexity is $O(N \cdot C \cdot |T|)$ where $|T|$ is the size of the test set. For very small values of N and C the expression $N \cdot C$ can be considered as constant and the complexity is linear to the size of test set $O(|T|)$. Although, higher N and C values theoritically result to better quality results, our experiments indicate that the quality gain saturates as these values increase. In all our case studies the values of $N = 30$ candidate test vectors generated for every test cube and $C = 3$ value achieve a near the maximum quality gain result with very fast execution time (it requires some minutes for all the benchmark circuits and almost 10 minutes for the largest ethernet benchmark circuit). Moreover, the proposed method can be easily parallelized by letting each thread handle a test cube. This way the $N \cdot C$ factor that impacts complexity can be shortened. A parallel implementation with 32 threads on a 4-cores CPU increases by only 4X the execution time for $N = 30$ and $C = 3$ (that would otherwise theoritically increase the execution time by 90X) compared to the PF X-filling approach. Nevertheless,

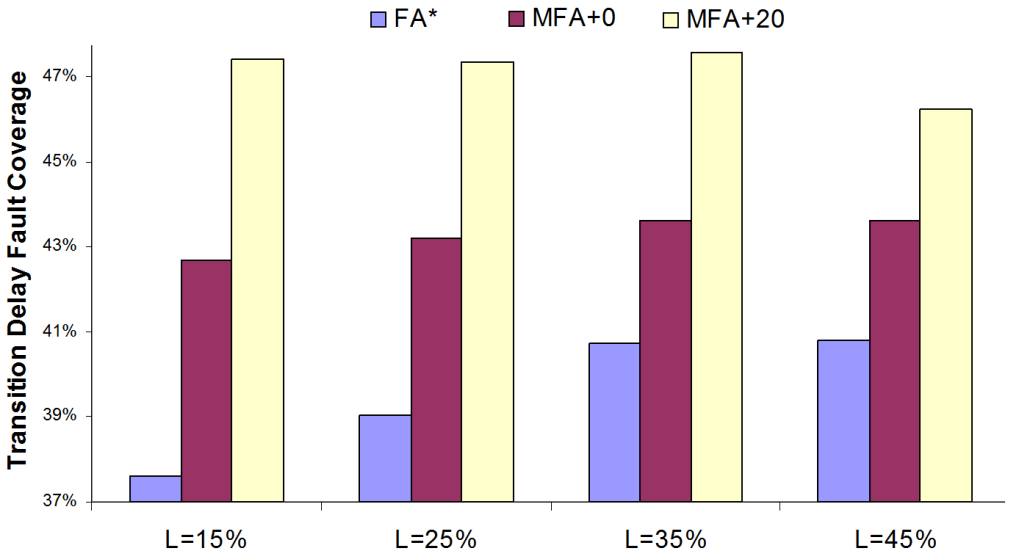


Figure 5.2: Transition delay fault coverage for various values of L (for s9234).

even the theoritical 90X execution time increase of a single threaded implementation, still requires very short execution time, even for very large designs, because the PF is fast.

Due to the dynamic compaction performed by the ATPG, the first few test cubes generated are usually densely specified and thus decrease the potential of the PF technique to reduce capture power below the pre-determined limit. This pre-determined limit can be selected according to the power profile of the circuits during their normal operation. In order to avoid time consuming bit-relaxation techniques, we replace these few but very densely specified test cubes with a small number of less specified test cubes generated in a second ATPG pass. We note that in the absence of the power profile of the benchmark circuits during normal operation, we set (unless otherwise noted) the capture power limit L (max number of cells switching during capture) equal to 30% of the scan cells switching during capture. The reason for this selection is twofold: a) the compacted nature (large volume of specified test bits) of the test sets prevents the reduction of the peak-capture power below certain values of L, b) as it will be apparent soon, the value of L does not affect the effectiveness of the MFA method to provide X-filling with high defect coverage. For even further reduction of the capture power, bit relaxation techniques [108, 109] can be utilized, and/or less compacted test sets can be used. Nevertheless, this study is beyond the scope of this work. We note that, as was expected, the FA method in most cases violated the capture power limit. Thus for providing a fair comparison with the MFA methods that do not violate this limit, we also implemented a slightly modified version of FA, denoted as FA*: for every cube violating the capture power limit when filled according to FA, 10% of the Xs (the most efficient ones) of the test cube are filled according to PF and the rest according to FA. If the test vector still violates the capture power, then the percentage of the test cube's Xs specified using PF is increased by 10%. This is repeated until a test vector is generated which does not violate the capture power limit.

Table 5.4: Defect Coverage (%)

Circuit	Transition-Fault Coverage						Bridging-Fault Coverage											
							BCE+						400K Random Faults Coverage					
	RF	**FA**	**FA***	**MFA**	**MFA+ 10**	**MFA+ 20**	**RF**	**FA**	**FA***	**MFA**	**MFA+ 10**	**MFA+ 20**	**RF**	**FA**	**FA***	**MFA**	**MFA+ 10**	**MFA+ 20**
s5378	61.47	55.48	55.18	56.95	61.34	61.81	95.2	93.81	93.55	94	94.14	94.26	94.27	92.19	92.08	92.54	92.79	92.91
s9234	41.47	40.82	41.01	43.26	44.04	48.32	87.51	87.12	87.28	87.28	87.44	87.42	86.38	85.49	85.77	86	86.24	86.34
s13207	62.29	60.31	61	64.05	65.43	65.9	92.77	92.71	92.16	92.93	93.02	93.11	91.97	91.3	91.14	91.73	91.8	92.06
s15850	51.53	51.05	50.33	52.51	52.56	54.03	94.24	94.09	93.84	93.82	93.91	93.98	93.52	93.11	93	93.03	93.15	93.29
s38417	79.53	76.2	76.22	78.48	79.09	80.38	98.2	97.56	97.62	97.75	97.83	97.85	97.16	96.28	96.32	96.6	96.68	96.73
s38584	61.8	61.07	60.83	61.67	62.17	62.08	90.31	90.1	89.53	89.8	89.85	89.91	89.86	89.58	89.29	89.5	89.55	89.56
ac97_ctrl	42.62	42.53	42.48	44.31	44.39	44.96	94.54	94.11	94.09	94.24	94.29	94.38	96.94	96.66	96.65	96.72	96.76	96.83
mem_ctrl	40.96	36.97	36.76	38.18	38.92	40.26	62.32	59.72	59.59	60.09	60.34	60.7	74.56	72.54	72.43	72.84	72.99	73.24
pci_bridge	64.4	61.74	61.82	65.38	66.68	67.5	95.75	95.41	95.46	95.62	95.67	95.71	96.61	96.33	96.36	96.48	96.52	96.54
tv80	53.47	51.1	51.1	53.26	57.48	58.14	91.49	91	90.99	91.11	91.15	91.12	89.3	88.38	88.38	88.87	88.87	88.9
usb_funct	63.94	63.14	62.46	64.41	63.98	64.27	93.74	93.21	93.34	93.39	93.44	93.49	95.14	94.77	94.86	94.96	95.03	95.04
ethernet	47.58	46.74	46.79	48.24	48.58	49.07	88.81	88.57	88.54	88.68	88.74	88.79	90.7	90.35	90.36	90.53	90.52	90.56

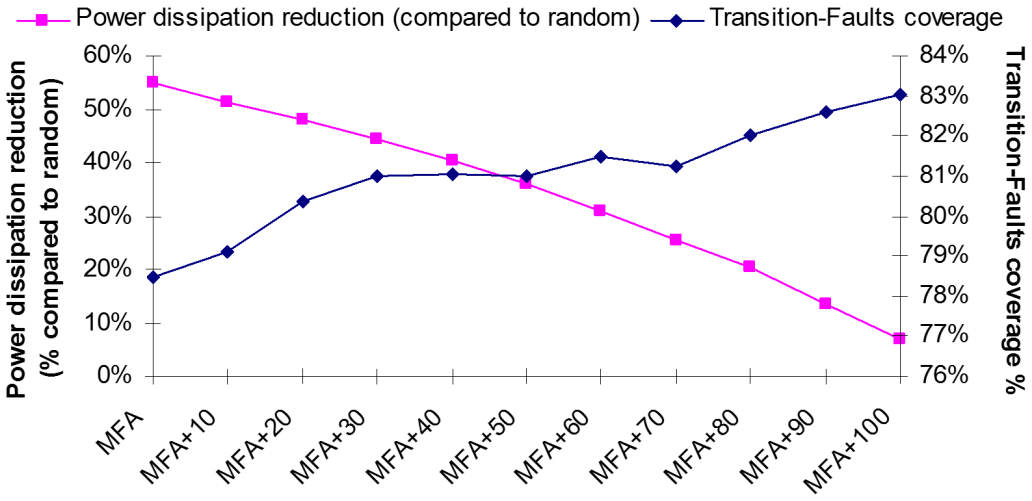


Figure 5.3: Power-reduction/Defect-coverage tradeoff for s38417.

Table 5.3 presents the power consumption during testing (shift and capture) comparisons between FA, FA, MFA, MFA+10 and MFA+20 methods. Note that all the methods, except the FA, that are compared in this Table honor the same capture power limit and their shift power is the bigest contributor at the overall power consumption (capture power is negligible compared to shift power because "number of shift clock cycles" $\gg$ "number of capture cycles" is very high). The first two columns present the circuit name and the number of test cubes for each circuit, which is the same for all methods. The remaining columns present the percentage reduction in average power consumption achieved by each method compared to random fill (RF). It is obvious that the highest average power reduction is offered by FA; however, FA is a capture power-unaware method. In the case of FA*, the filling of a portion of the Xs for reducing the capture power increases, in most cases, the average power consumed compared to FA. The reductions in average power offered by FA and MFA compared to RF are almost the same. Methods MFA+10 and MFA+20 provide smaller reduction, which still remains significant. Note that there is an unexpected result in a few cases where the FA* technique is inferior to MFA. This is caused by the PF technique, which tends to specify more Xs in the FA case than in the MFA case. This is explained by the fact that MFA generates many candidate test vectors for each test cube and thus the possibility of some of them to comply with the capture power limit at early stages of the generation process increases (at early stages PF has only limited effect on the filling of Xs). This does not happen in the case of FA*, where any violation of the capture power limit causes an immediate increase in the number of bits that have to be specified according to PF technique.

For evaluating the effectiveness of the MFA methods for defect screening, we consider the coverage of un-modeled faults, namely transition and bridging faults, obtained by applying to the circuit under test the stuck-at test vectors generated by the MFA methods. As it is common in industry, we use the launch-on-capture (LOC) scheme, also referred to as broadside scan, to apply test-vector pairs. Note that none of these two fault models

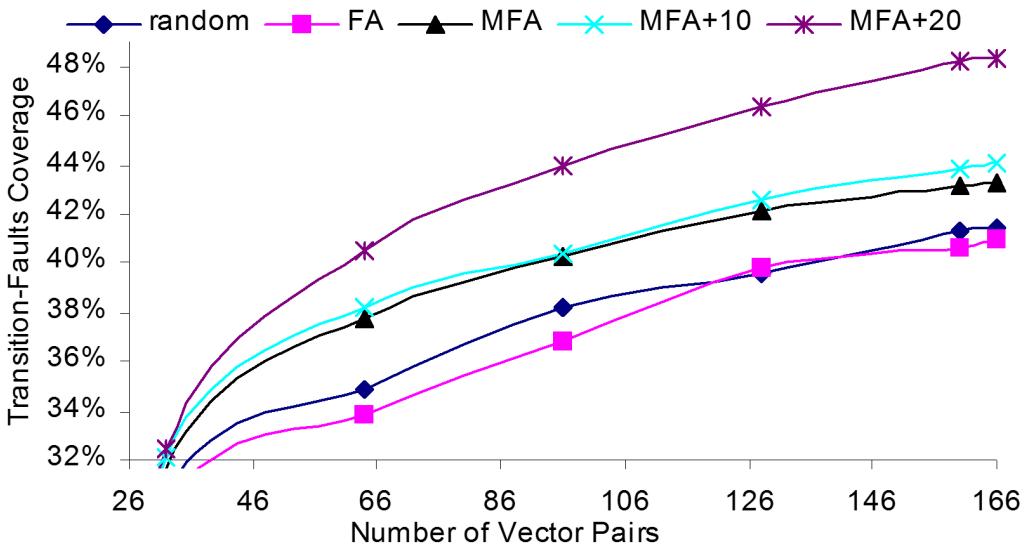


Figure 5.4: Transition delay fault coverage ramp-up for s9234.

were targeted by the stuck-at test sets (transition and bridging faults are used as surrogate fault models). For evaluating the defect-screening potential of the MFA methods in respect to bridging faults, we first used the BCE+ metric [147], which is useful for comparing different methods (the method with the highest value of BCE+ is deemed to be more effective for defect screening). Since BCE+ is not accurate for estimating the real bridging fault coverage, we also simulated 400K bridging faults as follows: 100K pairs of lines were selected randomly for each circuit, and four bridging faults were simulated for each pair by considering both lines as aggressors and victims, and by considering both logic values 0 and 1 at the aggressors.

At first we present the transition delay fault coverage of the FA, MFA and MFA+20 methods for various values of the power limit L in the range $[15\%, 45\%]$. Note that the transition delay faults were not targeted by the stuck-at test sets used for these experiments. We present trade-off results for s9234 in Figure 5.2 (the remaining circuits exhibit similar behaviour). We can see that, for every value of L shown in this Figure (even the less one) the transition fault coverage improvement is significant. However, we have to note that due to the compacted nature of the test sets, in the case of L=15%, the PF technique fails to provide test vectors with capture power below the limit L for many test cubes. This problem can be overcome with the use of bit-relaxation techniques and/or less specified test sets. Figure 5.3 presents the trade-off between power reduction and defect coverage based on the value of P. Specifically the power reduction of MFA+P techniques for P=0% (i.e., MFA), 10%, 20%,..., 100% as well as the respective transition fault coverage achieved for circuit s38417 are reported (the other circuits exhibit similar behaviour). It is obvious that the defect coverage increases as P increases. On the other hand, the power reduction achieved compared to RF decreases linearly and tends to zero as P approaches 100%, where all Xs are filled randomly.

Table 5.4 presents the defect coverage results. The first column presents the circuits'

name and the next six columns present the transition fault coverage for RF, FA, FA*, MFA, MFA+10, and MFA+20, respectively. It is obvious that the FA and FA* techniques provide the lowest coverage, while all the MFA methods provide higher coverage, which is even higher than that for the RF method in the majority of cases. This indicates that the proposed method exloits the Xes better than RF in terms of maximizing the unmodeled defect coverage. We note that the MFA techniques exhibit higher coverage ramp-up than RF and FA*. This is a significant advantage as it decreases the test time in an abort-at-first- fail environment. Due to lack of space, only the graph for s9234 is presented in Figure 5.4.

The next twelve columns in Table 5.4 present the bridging fault coverage comparisons of the above mentioned methods (the first six present the BCE+ comparisons and the next six present the random bridging fault coverage comparisons). All results indicate that the MFA methods achieve higher coverage than FA, FA*, approaching that for the RF method.

5.4 Conclusions

We presented two novel X-filling methods, MFA MFA+P, for reducing the power consumption during testing and for enhancing defect coverage. MFA considerably increases the defect coverage of the resulting (filled) test vectors compared to the power-efficient FA technique, with comparable average power consumption. MFA also ensures that peak power limits during response capture are not violated. Further improvements in defect coverage are achieved by the MFA+P technique, at the cost of a small increase in the average power consumption.

Chapter 6

LFSR Reseeding Techniques for High-Quality Testing

6.1 Overview

Defect screening is a major challenge for nanoscale CMOS circuits, especially since many defects cannot be accurately modeled using known fault models. The effectiveness of test methods for such circuits can therefore be measured in terms of the coverage obtained for unmodeled faults. In this Chapter, we present new defect-oriented LFSR reseeding techniques for test-data compression. The proposed techniques are based on a new "output deviations" metric for grading stuck-at patterns derived from LFSR seeds, and include a window-based static reseeding method as well as a dynamic reseeding method based on a ring generator. We show that, compared to standard compression-driven LFSR reseeding and a previously proposed deviation-based method, higher defect coverage is obtained using stuck-at test cubes without any loss of compression. The defect coverage for the proposed reseeding methods based on stuck-at test cubes is evaluated using two surrogate fault models, namely the transition fault model and the bridging fault model.

In this Chapter, a new encoding method that offers high compression and increased unmodeled defect coverage, for static and dynamic LFSR reseeding is presented. The main contributions are:

1) The new encoding method is suitable for both static window-based and dynamic LFSR reseeding, which are among the most efficient reseeding techniques.
2) High unmodeled defect coverage is achieved by using a new output- deviation-based metric that is more effective than [180] for detecting defects.
3) The encoding method enhances the defect-detection potential of the generated seeds without compromising compression.
4) Instead of exploiting free variables, defect detection is facilitated simply by carefully encoding the test cubes into seeds using compression as well as defect-oriented criteria.

Simulation results are presented for stuck-at test sets generated for the ISCAS'89 and IWLS'05 benchmark circuits [1]. These results show that the defect-aware window-based and dynamic reseeding methods offer higher defect coverage than the original (defect-unaware) window-based and dynamic reseeding methods, without any adverse impact on compression. In addition, due to the efficient output deviation metric introduced in this Chapter, the new method clearly outperforms [180] in terms of defect coverage. Finally, by grading the seeds in the case of static window-based reseeding and applying the most efficient seeds first, faster coverage ramp- up is achieved, thus reducing the test-application time in an abort-at- first-fail environment. Even for dynamic reseeding, where seed ordering is not an option, the carefully-tuned defect-oriented reseeding provides steeper coverage ramp-up.

The rest of the Chapter is organized as follows. Section 6.2 presents motivation for this work and Section 6.3 presents the new output deviation-based metric. Section 6.4 describes the procedure used for generating high-quality seeds using this metric. Simulation results are presented in Section 6.5 and Section 6.6 concludes the Chapter.

6.2 Motivation

Figure 6.1 presents the decompression architecture used in static as well as in dynamic reseeding. It consists of an L-bit sequential linear decompressor, which can be either an LFSR or a ring generator [106], and a phase shifter that receives the outputs of the decompressor and drives m scan chains ($m > L$). A test response compactor is also included in the scheme. The decompressor is reseeded by the Automatic Test Equipment (ATE), and it generates a test vector through the phase shifter. In static reseeding, the seed of the decompresor is its initial state and it is considered as a set of binary variables $a_0, \ldots, a_{L-1}$ that are loaded directly from the ATE. In dynamic reseeding, the decompressor is reseeded at every cycle from the ATE by injecting test bits into it through the ATE channels. Every injected test bit is considered as a binary variable and all variables injected during the generation of one test vector constitute the dynamic seed for this test vector. In both cases, a seed is determined by solving a system of linear equations, which is formed according to the specified bits of the test cubes and the feedback polynomial of the LFSR [77].

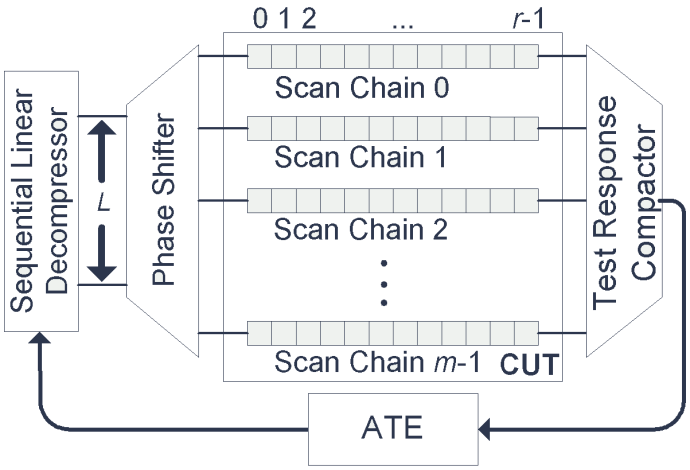


Figure 6.1: Generic LFSR reseeding architecture

The main disadvantage of the static LFSR reseeding is that every new seed flushes the decompressor contents and thus any variables left unspecified (free) during the seed-computation process are wasted (see Section 2.2.1). The method proposed in [180] exploits these free variables in order to increase unmodeled defect coverage. In order to achieve this goal, it utilizes the notion of output deviations [178]. Output deviations are probability measures at primary outputs and pseudo-outputs that indicate the likelihood of error detection at these outputs. As it was shown in [178], test patterns with high deviations tend to be more effective for fault detection. The method proposed in [180] attempts to improve the output deviations of the seeds as follows: it first applies multiple random fillings on the variables that remain free after each system of linear equations is solved, in order to generate multiple candidate seeds for each test cube. Then it selects the seeds that generate the vectors with the highest output deviation values.

Even though [180] constitutes an effective way to utilize the otherwise wasted variables, it still suffers from limited compression. Window-based reseeding [67] and dynamic reseeding [78, 79, 123] offer considerably better compression than [77], [180] as they manage to exploit very efficiently the seed variables. In the case of window-based reseeding every seed is expanded into $w > 1$ test vectors (w is referred to as the window size). Every position of the window can be used for encoding a different test cube, and thus multiple incompatible test cubes (i.e., test cubes that differ in at least one of their specified bit positions) as well as multiple compatible test cubes can be encoded at the same window (i.e. seed). Moreover, by carefully encoding each test cube at the window position that requires the replacement of the fewest variables, the probability of encoding additional test cubes at the same seed using the remaining variables increases. In the case of dynamic reseeding high compression is achieved by continuously injecting test data from the ATE channels into the decompressor, through linear (exclusive-or) operations with the current data of the decompressor. Thus the decompressor is not flushed and the unspecified variables remain inside the decompressor and they are exploited at a later step for encoding other test cubes.

Test Cubes

	c_1	c_2	c_3	c_4
t_1:	1	0	0	x
t_2:	0	x	x	x

Classical Reseeding			Window Based Reseeding		
Seed $\alpha_1\alpha_2\alpha_3\alpha_4$	Vect. $c_1c_2c_3c_4$	Deviations	Seed $\alpha_1\alpha_2\alpha_3\alpha_4$	Vect. $c_1c_2c_3c_4$	Deviations
0 1 x 0	1 0 0 0	0.107	0 1 1 0	1 0 0 1	0.107
C_sd1 (t_1)	1 0 0 1	0.107	***W_sd1***	1 0 1 1	0.156
	0 0 0 0	0.107	(t_1, t_2)	0 1 0 1	0.107
	0 0 0 1	0.107			
x 0 x x	0 0 1 0	0.107	0 1 0 0	1 0 0 0	0.107
C_sd2 (t_2)	0 1 0 0	0.107	***W_sd2***	0 1 1 0	0.107
	0 1 1 0	0.107	(t_1, t_2)	1 1 1 1	0.452
	0 1 1 1	0.156			

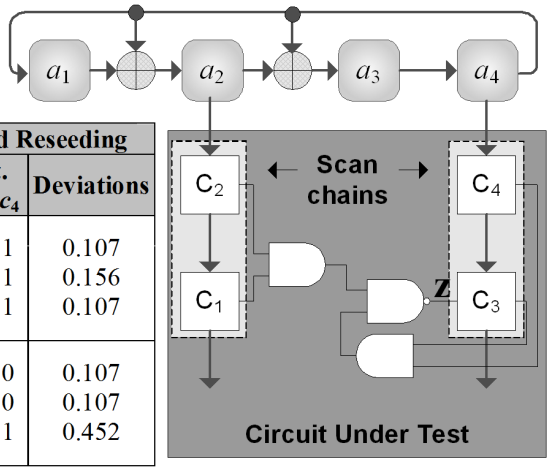


Figure 6.2: Example of classical and window-based LFSR reseeding

Both window-based and dynamic reseeding techniques exploit almost all variables in order to reduce the seed volume, thus they offer high compression. Moreover, [180] requires all the candidate seeds for all the test cubes to be computed before the selection process begins, which is not feasible in window-based and dynamic reseeding. Therefore, it is clear that the approach of [180] cannot be used in these cases. To overcome this problem, a different approach is followed in this Chapter, which efficiently compresses the test cubes, and also provides high defect coverage of the resulting vectors. The main idea of the new method is to generate multiple candidate seeds that implement different unique encodings of the test cubes. The encoding of each candidate seed is different from the encodings of the other candidate seeds, therefore, the probability of generating a vector with high output deviation values increases. At the same time, high compression is ensured by intelligently generating the candidate seeds in such a way that best exploits the variables for decreasing the seed volume. Let us see an illustrative example.

Example 6.1. Consider the circuit shown in Figure 6.2, which consists of two scan chains loaded using a 4-bit LFSR. The initial state of the LFSR is $\alpha_1\alpha_2\alpha_3\alpha_4$. Let us first encode test cubes t_1, t_2 shown in Figure 6.2 into seeds using [180]. Since test cubes t_1, t_2 are incompatible (they differ at the specified test bit corresponding to c_1), they can not be encoded by the same seed, and consequently two separate seeds are required. By solving the system of equations for t_1, we compute the seed $\alpha_1\alpha_2\alpha_3\alpha_4 = 01x0$ (denoted as C_sd1 in the Table of Figure 6.2). Note that α_3 is a free variable and it can be replaced by either logic value '0' or '1' providing thus two candidate seeds $\alpha_1\alpha_2\alpha_3\alpha_4 = 0100$ or 0110. In the same way we compute seed C_sd2 for encoding t_2 and we calculate the eight candidate seeds by replacing the three free variables with all eight possible binary combinations. The test vector applied to the circuit for each of these seeds and the respective output deviations are shown in Columns 2, 3 of the Table (the output deviations are computed as in [180]). By selecting the candidate seed corresponding to the first vector in the case of seed C_sd1 and the last vector in the case of C_sd2 the maximum output deviation value

achieved is equal to 0.156. Now let us apply the window-based encoding for window size equal to 3. In this case, higher compression can be achieved because both test cubes t_1, t_2 can be encoded into a single seed. For example t_1 can be encoded at the first position of the window, and t_2 can be encoded at the third position. However, the computed seed (labeled as W_sd1) has no free variables and thus it cannot provide any candidate seeds. We can easily see though, that t_2 can be also encoded at the second position of the window (W_sd2 in Table 1), which provides a second candidate seed offering the same compression as the first one (i.e. a single seed suffices to encode both test cubes in this case too). However, it is obvious from the output deviation values of the respective test vectors of both seeds (sixth column of table), that by selecting W_sd2 instead of W_sd1, the maximum value of output deviation increases from 0.156 to 0.452. ■

It is therefore obvious that different window-based encodings yield similar results in terms of seed volume, but they exhibit significant variations in terms of output deviation values and potentially of defect coverage. In fact these variations are more significant than in [180]. This is because the encoding of different combinations of test cubes into a candidate seed affects the generated vectors much more than the random replacement of the free variables.

Finally, we emphasize that the metric proposed in [180] is not efficient since it ignores important parameters such as the structure of the circuit under test and the output deviations of previously selected seeds. A more efficient output deviation-based metric was proposed in [69] for generating test sets with high defect coverage. This metric takes into account structural information of the circuit under test in order to further increase the defect coverage. A major limitation of this metric is that it evaluates each test vector for either timing-independent or timing-dependent defects. In this Chapter, we use the metric proposed in Chapter 5 in order to evaluate the unmodeled defect coverage of a set of test vectors.

6.3 A Deviation-based Metric for Time-related Defects

In this section we present the proposed output deviation-based metric for evaluating a candidate seed s (we present the metric assuming that the s is calculated using window-based reseeding, as the extension to the dynamic reseeding case is straightforward). We assume that each seed s is expanded into w test vectors (w is the size of the window) and each one of them is applied using two capture cycles r_1, r_2. In other words we assume the Launch-On-Capture (LOC) technique as it is common in industry. Although, the metric described here is based on the one proposed in Chapter 5, it still requires to support the window-based LFSR encoding, because of a special property of window-based method's seeds. During window-based LFSR encoding a seed generates w vectors (contrary to the other LFSR based techniques). So, it is not clear how a seed is evaluated using the metric of Chapter 5, which is applied on a single vector. The answer is that the metric is applied to all the vectors inside w simultaneously, as if all w vectors were a single huge

test vector. Moreover, the calculation of maximum expected deviations values is slightly different compared to Chapter 5, since it was done with random assignement. In this Chapter, the metric is applied on linear decompressors and the maximum values should capture any limitations imposed by the linear correlations of the decompressors used. Below, the formulation of the new metric is given:

The Maximum Expected Deviation value for output i at capture cycle r_k (k = 1, 2) and fault-free response v (v = 0, 1), denoted as $MED(i, r_k, v)$ is an estimate of the maximum deviation value expected throughout the seed-computation process on output i when its fault-free response is v at capture cycle r_k. It is calculated as follows: initially, for every test cube, a predetermined number of single-vector seeds (i.e., seeds encoding only one test cube) are generated by randomly replacing the free variables. For each output i, the generated test vectors are partitioned into four groups: those producing fault-free responses 0 and 1 at capture cycles r_1 and r_2. The output-deviation values of all generated test vectors are calculated and the greatest value for every output i and for each fault- free response v = 0, 1 at capture cycle r_k constitutes $MED(i, r_k, v)$. After calculating the $MED(i, r_k, v)$ values, the generated single-vector seeds are discarded.

With the use of the MED values, the evaluation of the candidate seeds at each step of the seed computation process is done as follows. Let $D(s, j, i, r_k, v)$ be the deviation value at output i for the j^{th} test vector in the window of candidate seed s ($j \in [1, w]$), where w is the window size), which produces fault-free response v at that output at capture cycle r_k. The value $D(s, j, i, r_k, v)$ is considered to be maximum if it is very close to $MED(i, r_k, v)$, or equivalently, if the following inequality is true:

$$D(s, j, i, r_k, v) \geq F_1 \cdot MED(i, r_k, v), \quad v = 0, 1 \tag{6.1}$$

F_1 is a real-valued parameter that must be close to 1 for selecting seeds with output deviation values that are very close to the maximum expected deviation. However, a value of $F_1 = 1$ must be avoided since sometimes it results in a failure to select seeds (the predicted $MED(i, r_k, v)$ value becomes hard to reach during the selection of the seeds). As in the case of [69] we also verified that a value of F_1 in the interval $[0.99, 0.995]$ provides high-quality seeds in all cases, and for our experiments we set $F_1 = 0.995$.

The second task of the evaluation process is to rank all outputs according to their potential of observing errors due to defects. Every output i is assigned two pairs of weights $wo(i, r_k, 0)$, $wo(i, r_k, 1)$ for $k = 1, 2$, which are initially all set equal to the number of lines in the logic cone of the corresponding output. These weights are indicative of the volume of undetected defects that can be possibly detected for both fault-free responses 0 and 1 at output i during both capture cycles r_1, r_2. The set of weights $\{wo(i, r_1, 0)$, $wo(i, r_1, 1)$, $wo(i, r_2, 0)$, $wo(i, r2, 1)\}$ and the output deviation values are used during the evaluation of the candidate seeds for determining a weight $WS(s)$ for every candidate seed s as follows: assume that seed s is expanded into a window of w test vectors, and let j be one of the w window positions, i.e., $j \in [1, w]$. Let the number of observable outputs in the circuit be k. For test vector j, the sets $MS[s, j, r_1, 0]$, $MS[s, j, r_1, 1]$, $MS[s, j, r_2, 0]$,

$MS[s, j, r_2, 1]$ consist of all outputs i, with $1 \leq i \leq k$, for which any of the deviation values $D(s, j, i, r_1, 0)$, $D(s, j, i, r_1, 1)$, $D(s, j, i, r_2, 0)$, $D(s, j, i, r_2, 1)$ satisfy inequality 6.1.

Finally, for evaluating each candidate seed, the sum of its weights is calculated using the formula:

$$WS(s) = \sum_{k=1,2} \sum_{j \in [1,w]} \left[\sum_{i \in MS[s,j,r_k,0]} wo(i, r_k, 0) + \sum_{i \in MS[s,j,r_k,1]} wo(i, r_k, 1) \right] \quad (6.2)$$

The above formula means simply that, for either fault-free response 0 or 1, only the weights of the outputs that get near-maximum deviation values for capture cycles r_1, r_2 (i.e., those belonging to $MS[s, j, r_1, 0]$,$MS[s, j, r_1, 1]$, $MS[s, j, r_2, 0]$, and $MS[s, j, r_2, 1]$) participate into the final weights sum $WS(s)$. Note that the first response targets the timing- independent defects, while the second response targets timing-dependent defects. The seed with the highest WS value is selected as the one with the best potential to detect timing-independent as well as timing-dependent unmodeled defects.

The weight $WS(s)$ enables the selection of seeds that generate vectors with the maximum deviation values at the outputs of large cones of the CUT. The larger the cones are the greater is the probability of detecting unmodeled defects. However, maximizing the deviations only at a subset of outputs may result in low defect coverage, even when this subset consists of the outputs of the largest logic cones. To this end, for every selected seed, every output i which satisfies equation 6.1 is identified, and the respective weight $wo(i, r_k, v)$ is divided by a constant factor F_2. In that way, the outputs with reduced weight have much smaller impact on the selection of the next seeds. This is motivated by the fact that if seed s provides a high deviation at output i for fault-free response v at capture cycle r_k then it is likely that many defects at the fan-in cone of i will be detectable at output i when s is applied. Thus, test vectors that maximize the deviation at output i for the same fault-free response and the same capture cycle will be less effective for increasing the defect coverage during the application of the next seeds. We have chosen the value of F_2 to be equal to 8, as we verified experimentally that a value of F_2 in the interval $[2, 10]$ is sufficient to maximize the deviations at all outputs.

6.4 Generation of Defect-Aware Seeds

In this section, we first describe the encoding algorithm for defect-aware window-based LFSR reseeding. Next we discuss the special case of window-based reseeding with window size $w = 1$, and the defect-aware dynamic reseeding method.

6.4.1 Window-Based Reseeding

The window-based reseeding approach proposed in [67] attempts to maximize compression by using the following very effective encoding criterion:
Compression Maximization Criterion: "The first test cube encoded by every seed is the

one with the highest number of specified bits, and it is encoded at the first window position. The next most-specified test cube is then encoded at the window position that results to the replacement of the minimum number of variables. If more than one such test cubes exist, the test cube that requires the replacement of the fewest variables is encoded. This continues until no system for any of the unencoded test cubes can be solved in the same window."

This criterion efficiently exploits all the properties offered by window-based reseeding mentioned in Section 2.2.1. In addition, it attempts to increase the number of densely specified test cubes encoded by every seed, which, as shown in [67], tends to decrease the overall seed volume even more. These are all major differences with other strategies used in the literature for encoding test cubes, e.g., the incremental solver proposed in [123].

In the method presented in this Chapter, two objectives are simultaneously addressed: the efficient compression of test cubes, and the high defect coverage of the resulting patterns. High defect coverage is targeted by generating candidate seeds implementing different unique encodings of test cubes. For the selection of every seed, T candidate seeds (T is a user-defined parameter) are first generated and then evaluated using the output-deviation-based metric presented in section 6.3. The best candidate seed according to this metric is selected each time. High compression is ensured by carefully generating the candidate seeds using a new encoding criterion which ensures that all candidate seeds provide nearly the same level of compression that is obtained if the seed is generated using the compression- maximization criterion (i.e., according to [67]).

The generation of the T candidate seeds is done as follows: we start by encoding the most-specified test cube (say t_1) in the first position of the corresponding window. Next, for initiating the generation of the T different candidate seeds, we independently apply the compression-maximization criterion T times in that window, excluding each time all the previous decisions. In other words, we identify the best T different test-cube encodings that can be independently performed in the window that embeds t_1 in its first position. As a result, T different windows with t_1 in their first position, and other test cubes in the rest of the positions are determined.

The above procedure implies that we initially target windows that embed two different test cubes. Note that this does not necessarily mean that the T chosen windows embed T different pairs of cubes (i.e., t_1 along with another cube). Test cube t_1 can be combined with the same test cube, t_i, more than once, if t_i can be encoded in different positions of the window and the corresponding solutions are among the T best solutions according to the compression-maximization criterion. Hence, among the T chosen windows, there may be more than one embedding t_1 and t_i, with t_i encoded in a different window-position every time. However, if all possible windows that embed t_1 with a second test cube are fewer in number than T, then we increase the volume of the already chosen windows by encoding in them a different third test cube. Two new different windows embedding three (n) test cubes can be derived from one window which embeds two ($n-1$) test cubes, by separately encoding in the latter either two different test cubes (one for each

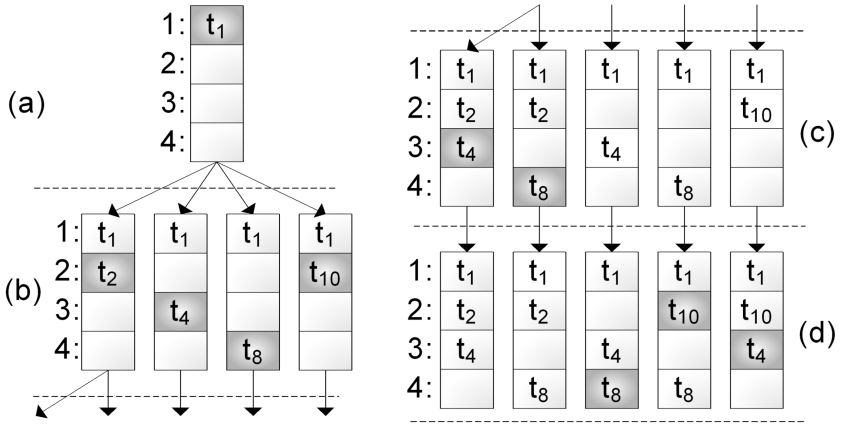


Figure 6.3: An example to illustrate the generation of T candidate seeds

new window), or the same cube in two different positions. The same procedure is repeated until we get T different windows, corresponding to the T candidate seeds. At this point, the set of candidate seeds has T members; therefore, we continue by encoding as many test cubes as possible in the window of each candidate seed by using only the compression-maximization criterion. Finally, the T generated candidate seeds are evaluated using formula 6.2, and the most promising one for increasing the defect coverage is selected. Then, the test cubes encoded in its window are dropped and the seed-computation process continues in the same way for selecting the next seed. We provide insights into the above process with the following example.

Example 6.2. Let $t_1, t_2, \ldots, t_{10}$ be 10 test cubes sorted in descending order according to their number of specified bits, $w = 4$ be the window size, and $T = 5$ be the number of candidate seeds. In Figure 6.3, we present each window as a column with 4 cells, one for each window position. Each encoded test cube is reported inside the corresponding cell and the newly encoded test cubes are highlighted at each step. Initially, we encode test cube t_1 (the most specified one) in window position 1 (Figure 6.3a). Let us assume that the systems of equations for test cubes t_2, t_4, t_8, and t_{10} are independently solvable in the same window with t_1 (t_2 is the first cube selected by the compression-maximization criterion, t_4 is the next selection, i.e., if we exclude t_2, and so on). As a result, we initiate the generation of four new candidate seeds (Figure 6.3b) by encoding each one of these test cubes separately into the window that we previously encoded t_1 (i.e., each one of the four seeds encodes one of the following pairs of test cubes: t_1 and t_2, t_1 and t_4, t_1 and t_8, t_1 and t_{10}). Since though, the upper limit of $T = 5$ candidate seeds has not been reached yet, we continue and attempt to encode a third test cube in the windows generated so far. This is shown in Figure 6.3c. After encoding test cube t_4 first, and then t_8, in the window embedding t_1 and t_2 (the compression-maximization criterion is again used for these selections), we reach the limit of 5 candidate seeds. Therefore, the expansion of the tree (i.e., the generation of new windows) now terminates and we continue by encoding in each of the T windows only the test cubes that maximize compression (Fig 3d). Finally,

we have generated five candidate seeds $s_1, \ldots, s_5$ which are subsequently evaluated using formula 6.2. Note that the leftmost seed (s_1 in this case) provides the best compression. Assuming though that seed s_3 has the highest weight among all seeds according to 6.2, s_3 is selected and test cubes t_1, t_4 and t_8 are dropped from the set of test cubes to be encoded. ■

In contrast to [67], we examine various encoding options, apart from the one that maximizes compression (i.e., t_1 along with t_2 and t_4 used in the previous example). Thus, several choices are available for maximizing defect coverage. By trying different encodings early on in the encoding process (i.e., after the selection of just the first cube for every window) we guarantee that the T candidate seeds will be sufficiently different (and hence they will potentially provide sufficiently different defect coverage). By selecting these different encodings using the compression-maximization criterion, we ensure that compression is not compromised.

Note that by generating multiple candidate seeds with diverse encoding, we cannot always guarantee an increase in defect coverage. However, we experimentally found that among the candidate seeds, there exist seeds that increase defect coverage, and these seeds are effectively identified and selected by the metric presented in Section 6.3. In addition, note that by trying different encodings, the complexity of the encoding process increases. However, we found in our work that even a small value of T can provide significant increase in the defect coverage offered by the resulting seeds, and thus the encoding process is feasible for large circuits. This can be easily concluded with the following analysis. Suppose that the proposed deviation based enhancement method is applied on an encoding technique with complexity $O(N)$, where N is the size of the test set that needs to be encoded. The proposed method adds a multiplier factor T at the complexity of the encoding method that is applied on, and the new complexity becomes $O(T \cdot N)$. The gain in unmodeled defect coverage saturates very fast as T increases. As a result, an efficient gain can be achieved for very small values of T (experiments show that small values of T even with $T \leq 30$ can almost maximize that gain and, so, we selected $T = 30$ for all our experiments). Given that, the proposed method can be treated as adding a constant complexity factor to the complexity of the encoding method and consequently the complexity remains unaffected.

After all the seeds are generated, they are sorted according to their potential to detect defects. Seeds with higher potential are loaded first in the LFSR in order to detect defects as quickly as possible and thus to decrease the test application time in an abort-at-first-fail environment. The ranking of the seeds is based on an evaluation process that is similar to the T candidate seeds evaluation procedure. The difference lies in the fact that this procedure is now applied to all the selected seeds, and not to candidate seeds. Moreover, since all seeds are known at this step, the actual maximum deviation value $MD(i, r_k, v)$ for each output i and fault-free response $v = 0, 1$ at capture cycle r_k can be easily computed (it is the largest among the output-deviation values of all test vectors generated by all calculated seeds). Equation 6.2 is applied in this case too, but this

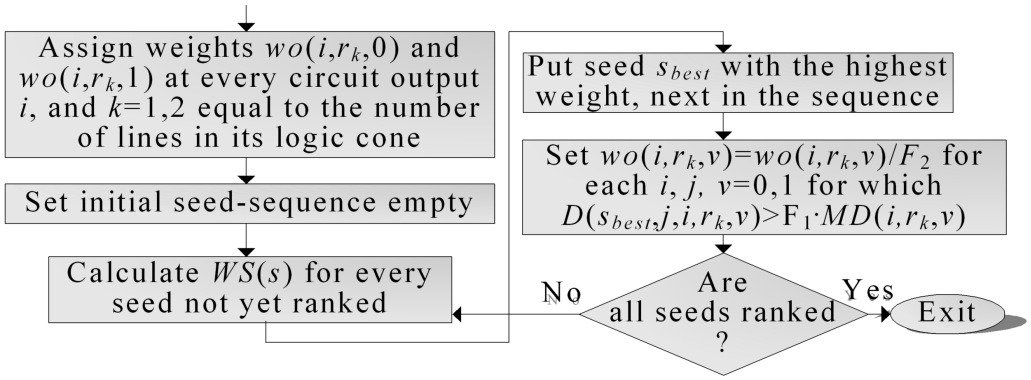


Figure 6.4: Final ranking of the selected seeds

time the set $MS[s, j, r_k, v]$ is calculated by replacing values $MED(i, r_k, v)$ with values $MD(i, r_k, v)$ in inequality 6.1. A flowchart for this final ranking of the seeds is shown in Figure 6.4.

6.4.2 Classical Static LFSR Reseeding and Dynamic Reseeding

One of the advantages of window-based reseeding is that the size of the window (w) offers a tradeoff between compression and test sequence length. Specifically, large values of w offer very high compression at the expense of relatively increased test sequence length, whereas small values of w offer short test sequence length at the expense of relatively reduced compression. In the degenerate case of $w = 1$, every seed generates only one test vector. The test-application time is minimized, but only compatible test cubes can be encoded by each seed. This restriction limits the encoding ability of the T candidate seeds' generation process described in the previous section, and consequently it adversely affects both the encoding ability and the defect-screening potential of the resulting seeds. However, the use of uncompacted test cubes combined with the defect aware compression-maximization criterion presented in the previous section almost eliminates these adverse effects and also offers the potential for a wide range of encoding options. This is the significant difference between the classical and the window-based reseeding approach as for $w = 1$; in classical reseeding, as proposed in [77] (and adopted in [180]), only one test cube is encoded by each seed, whereas in window-based reseeding for $w = 1$, the utilization of the defect-aware compression-maximization criterion offers an efficient way to combine more than one compatible test cubes in the same encoded pattern. Thus, as will be shown in the experimental section, the volume of the defect-aware seeds is low and their quality is high for $w = 1$ as well.

Dynamic reseeding resembles window-based reseeding for $w = 1$, in the sense that they both generate one test vector per seed. Thus, even though the way in which the systems of equations are formed is different from static reseeding, the criterion for generating the T candidate dynamic seeds can be applied in this case too. However, using this criterion, the seeds are generated in no particular order of effectiveness in terms of defect coverage.

Most of the time, a seed selected near the end of the selection process may be more efficient than a seed near the beginning. In the case of static window-based reseeding, the final sorting of the generated seeds, according to their output deviations (Figure 6.4) solves this problem and provides steep defect coverage ramp-up. Note that in the case of static reseeding, there is no dependency between the static seeds, because each seed flushes the contents of the LFSR; therefore, sorting of the seeds is possible. However, in the case of dynamic reseeding, reordering of the dynamic seeds is not possible since the LFSR is never flushed. To provide both high defect coverage and steep coverage ramp-up in dynamic reseeding, we allow a small reduction in compression offered by the T-candidate seeds generation process, in order to facilitate the generation of high-quality seeds. Specifically, instead of encoding the most-specified test cube as the first test cube of every candidate seed, we select the T most-specified test cubes which have not yet been encoded. Each one of these test cubes is encoded as the first test cube of each of the corresponding T candidate seeds. Consequently, every candidate seed encodes a different test cube as its first test cube. Then, for each candidate seed, we continue by encoding the test cubes providing the highest compression (i.e., the most specified ones that also require the replacement of the fewest variables), excluding all the T test cubes selected at the first step.

This modification allows us to increase the likelihood of generating high- quality candidate seeds as early as possible but it can also potentially reduce the amount of test compression. However, in the modified criterion, the candidate seeds are still among the most efficient ones in respect to the achieved compression. Experimental results show that the reduction in compression is infrequent and very small.

Very frequently during the candidate-seed generation process, a single test cube should be selected from a subset of equivalent, according to the Compression Maximization Criterion, test cubes (i.e., test cubes that include the same maximum number of specified bits and, at the same time, their encoding requires the replacement of the same minimum number of variables). In most of these cases, only one of them can be encoded, because the selection of any such test cube prevents the encoding of the others in the same seed (i.e., after any one of them is encoded the rest become un-encodable). We exploit this property to increase the quality of the candidate seeds without sacrificing compression. Specifically, during the generation of every candidate seed, the first time that a set of test cubes, say ST, is found with the above property, we select m of them (m is a predetermined parameter) and we separately encode them in the candidate seed. Thus the candidate seed is replaced by m new ones, and each one of them embeds all the test cubes of the initial candidate seed (i.e., the one that we replace with the m new ones) as well as one of the test cubes of set ST. Note that this is done only for the first (and consequently most- specified) m test cubes found for each one of the initially generated T candidate seeds, in order to keep the candidate-seeds' volume low. To bound the number of candidate-seeds' volume, we set the maximum value of m equal to 2. Thus, the volume of generated candidate seeds cannot exceed $2 \cdot T$, which is relatively small.

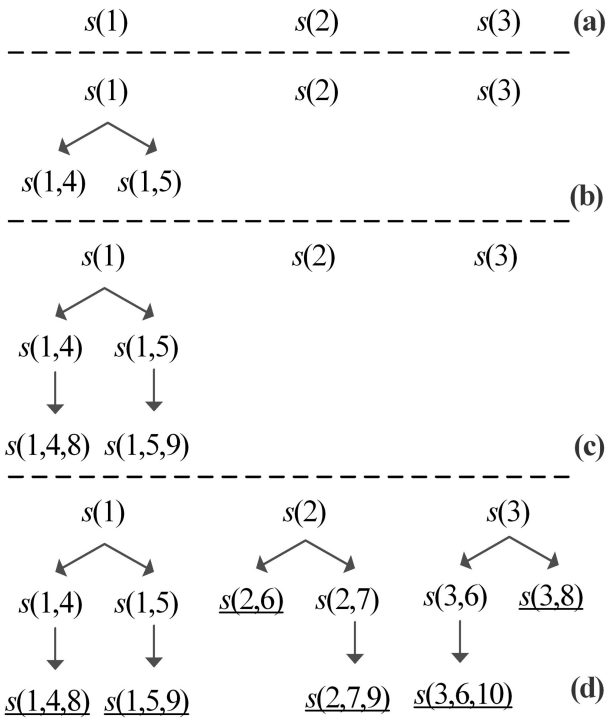


Figure 6.5: Illustration of the generation of candidate seeds for dynamic reseeding

*Example 6.3.*Let $t_1, t_2, \ldots, t_{10}$ be 10 test cubes sorted in descending order according to their number of specified bits, $T = 3$ be the number of candidate seeds, and $m = 2$. Figure 6.5 presents the various steps of the encoding process. Each dynamic seed encoding test cubes t_a and t_b is denoted as $s(a, b)$. At the first step (Figure 6.5a) test cubes t_1, t_2, t_3 are selected (they are the most specified ones) and the candidate seeds $s(1)$, $s(2)$, $s(3)$ are determined. Next we proceed with the seed $s(1)$ and we select the most specified test cubes (excluding tC_2, t_3) that can also be encoded by this seed. Let cubes t_4, t_5 and t_6 have the same number of specified bits. Suppose that they also require the replacement of the same number of variables, and let all three of them be separately encode-able (only one at a time) at seed $s(1)$. Since $m = 2$ we select the first two among them, namely t_4, t_5. Thus candidate seed $s(1)$ is replaced by candidate seeds $s(1, 4)$ and $s(1, 5)$ – see Figure 6.5b. We proceed with each one of them separately by encoding cubes t_8 and t_9 at seed $s(1, 4)$ and $s(1, 5)$, respectively – Figure 6.5c. At this point, the generation of the first two candidate seeds terminates and we proceed with the next candidate seed $s(2)$ in the same way. Finally, six (equal to the upper limit of $2 \cdot T$) candidate seeds are generated. These seeds are underlined in Figure 6.5d. ■

Note finally that in dynamic reseeding, each dynamic seed may not be fully specified. Some variables may remain unspecified and are utilized by the next seeds. However, in order to apply relation 6.2 for computing the output-deviation metric of each candidate dynamic seed, every unspecified variable has to be assigned either the logic '0' or '1' value. To overcome this problem, the encoding estimates the output deviation values by

temporarily replacing these variables randomly, and based on these estimates, it selects the best dynamic candidate seed. Then it removes the random assignment from the variables (i.e., they become unspecified again) and proceeds to the computation of the next dynamic seed. In this way, compression is not compromised as the variables are utilized only for encoding test cubes, whereas, at the same time, a good estimate of the defect-screening potential of each candidate seed is obtained.

6.5 Fault simulation Results

In this section, we evaluate the effectiveness of the defect- aware reseeding methods. The simulation platform was developed using the C programming language, and all ATPG and fault simulations were carried out using commercial tools. We conducted experiments using the largest ISCAS'89 circuits and a subset of the IWLS'05 circuits [1]. The number of scan chains was set equal to 30 for the ISCAS circuits, 50 for the medium sized IWLS circuits, and 100 for the large ethernet IWLS circuit. For evaluating the window-based reseeding method, we considered two window sizes, $w = 1$ and $w = 5$. For each benchmark circuit, a dedicated LFSR with a characteristic primitive polynomial of near minimum size was selected following the $s_{max} + 20$ rule (see Section 2.2.1 and [77]).

For evaluating the dynamic reseeding method, we conducted experiments with various ATE-channel volumes (the best results are reported). In addition, the LFSRs used in window-based reseeding were replaced by ring generators of the same size in dynamic reseeding. In the rest of the section, two cases are reported for both reseeding approaches: a) the case, denoted as "Cmp" which refers to the encoding that targets only compression (the original approach without applying the proposed enhancement method), and b) the case noted as "Cmp & Def", which refers to the encoding that targets compression and defect coverage at the same time. For the "Cmp & Def" case, T was set to 30, m was set to 2 (m is used only in dynamic reseeding) and the constants F_1 and F_2 were set equal to 0.995 and 8 respectively.

To demonstrate the advantage of the proposed method compared to the classical reseeding-based method of [180], which uses a different output deviation-oriented metric, we have implemented the method of [180] as well as the defect-unaware classical reseeding method [77]. We conducted experiments for these two methods using compacted test sets generated by the same commercial ATPG engine used for the rest of the experiments. Note that in contrast to the other methods (window-based and dynamic reseeding), for the classical LFSR reseeding approaches of [77] and [180] we used compacted stuck-at test sets in order to minimize both the number of required seeds as well as the test-sequence length. These methods are not accompanied by a dynamic compaction technique. So, their TSL is the same with the size of the test set. As a result, these methods exhibit the best TSL and TDV when they are applied on compacted test sets.

In Table 6.1, we present the TDV in Kbits (1Kbit $=10^3$ bits) for the window-based and dynamic reseeding methods, as well as for the classical (static) reseeding approaches.

Table 6.1: Test data volume results (in Kbits)

Circuit	Classical Reseeding		Window-Based Reseeding				Dynamic Reseeding	
			$w = 1$		$w = 5$			
	Test Set Size	[77, 180]	Cmp	Cmp & Def	Cmp	Cmp & Def	Cmp	Cmp & Def
s5378	28.7	16.1	8	8	6.2	6.3	7.6	7.9
s9234	41	23.2	16.9	18.4	14.2	14.3	17.3	17.6
s13207	188.3	78	12	12.8	8.2	8	15.1	15.5
s15850	99	47	18.6	19	13.6	14	16.2	16.7
s38417	238	117.3	64.6	65.4	58.2	59.7	65.6	68.4
s38584	270.8	148	34	34	27.2	26.9	42.3	41.6
ac97_ctrl	148.7	68.6	11	10.9	7.2	7.3	13.9	13.7
mem_ctrl	720	373.9	113.9	117.8	79.7	86.6	126.8	128.8
pci_bridge	1160.6	343.2	111.4	110.3	100.2	99.7	123	122.9
tv80	281.6	151.4	99.8	102.5	54.3	55.7	81.4	82.7
usb_funct	252.7	129.2	57.5	57.4	48.9	49.4	55.1	56.2
ethernet	11.8×10^3	1.7×10^3	203.8	225.5	162.5	165.1	231.3	233.1

The first column lists the names of the benchmark circuits. The next column presents the size of the compacted test set used for the evaluation of both the classical defect unaware LFSR reseeding method ([77]) and the method proposed in [180]. The third column presents the TDV for these two methods, which is the same for both of them (note that [180] differs from the classical LFSR reseeding approach only in the way that the free variables are filled and does not impact TDV). The next three pairs of columns present the TDV of the $w = 1$, $w = 5$ and dynamic reseeding cases, in their defect-unaware versions ("Cmp") and at the proposed defect-aware versions ("Cmp & Def").

As it is obvious from Table 6.1, window-based and dynamic reseeding clearly outperform the classical static reseeding approaches ([77] and [180]), while the highest compression is always achieved by window-based reseeding for $w = 5$. Dynamic and window-based reseeding for $w = 1$ provide comparable results. The most important observation though is that for both the window-based and the dynamic reseeding method, the proposed defect-aware encoding (columns labeled "Cmp & Def") provides nearly the same compression as the original defect-unaware encoding (columns labeled "Cmp"). In a few cases, the proposed defect-aware encoding provides even better compression than the original defect-unaware encoding. We attribute this result to the window-based encoding criteria, which consist a heuristic encoding approach and they do not offer an optimal solution. As a result, arbitary changes on the encoding order caused by the proposed method may result to better compression results. Nevertheless, it is obvious that the utilization of the output-deviation metric has no significant adverse impact on compression for both static and dynamic reseeding methods.

Table 6.2: Test sequence length results (# vectors applied)

Circuit	Classical Reseeding [77], [180]	Window-Based Reseeding				Dynamic Reseeding	
		w=1		w=5			
		Cmp	Cmp & Def	Cmp	Cmp & Def	Cmp	Cmp & Def
s5378	134	199	199	770	785	232	243
s9234	166	282	307	1185	1195	317	324
s13207	269	300	320	1030	1005	313	322
s15850	162	310	316	1130	1170	385	396
s38417	143	808	818	3635	3730	585	611
s38584	185	485	485	1945	1920	288	283
ac97_ctrl	66	274	273	895	910	151	149
mem_ctrl	603	876	906	3065	3330	879	894
pci_bridge	330	1238	1226	5565	5540	867	866
tv80	757	1663	1708	4525	4645	1693	1719
usb_funct	136	959	956	4075	4115	724	739
ethernet	1111	2912	3222	11610	11790	2155	2182

Table 6.2 presents the test-sequence lengths (TSLs) of the examined reseeding methods in terms of the test vectors applied to each circuit. Column 2 presents the TSLs of the classical reseeding approaches (which are the same for both [77] and [180]). The next three pairs of columns present the TSLs of the $w = 1$, $w = 5$ and dynamic reseeding cases, in their defect unaware versions ("Cmp") as well as in their proposed defect aware versions ("Cmp & Def"). As expected, the classical, dynamic, and window-based reseeding for $w = 1$ offer short and comparable, in many cases, test sequence lengths. Note that the TSLs of the classical reseeding approaches are shorter than those of window-based reseeding for $w = 1$, due to the use of compacted test sets in the former case. As expected, the TSLs of window-based reseeding for $w = 5$ are greater than those of the other methods, due to the larger value of w. However, this can be also attributed to the small LFSR sizes used here. Larger LFSRs offer considerably shorter TSLs (due to the smaller number of calculated seeds) with minimal impact on compression. For example, if we increase the size of the LFSR used for the pci_bridge circuit from 90 bits to 200 bits, the test-sequence length decreases form 5565 vectors to 2695 vectors, whereas the compressed test data volume has only a limited increase from 100.2 Kbits, to 107.8 Kbits. It is obvious that as the size of the LFSR increases, the test application time drops considerably, while at the same time the compression is not significantly affected. In our experiments we selected the size of the LFSRs based on the $s_{max} + 20$ rule (see Section 2.2.1 and [77]), where s_{max} the number of defined bits of the most specified test cube.

For evaluating the effectiveness of the proposed defect-aware reseeding methods for defect screening, we consider the coverage of unmodeled faults, namely transition and bridging faults, obtained by applying to the circuit under test the test vectors generated

Table 6.3: Transition-fault coverage (%)

Circuit	Classical Reseeding		Window-Based Reseeding				Dynamic Reseeding	
			w=1		w=5			
	[77]	[180]	Cmp	Cmp & Def	Cmp	Cmp & Def	Cmp	Cmp & Def
s5378	61.1	63.49	62.9	66.38	65.66	70.32	63.64	66.2
s9234	40.7	49.63	43.04	53.08	53.94	58.41	45.84	53.52
s13207	62	69.48	62.94	68.28	64.31	70.32	60.18	68.52
s15850	52.8	55.25	53.58	56.95	57.58	58.31	53.87	58.04
s38417	79.2	80.24	85.42	87.93	88.85	90.6	84.99	87.32
s38584	61.5	62.21	65.03	66.32	68.1	69.07	63.25	64.02
ac97_ctrl	42.7	45.6	47.18	56.42	52.4	63.95	45	50.81
mem_ctrl	41.1	44.24	42.69	46.01	44.03	47.36	43.32	45.72
pci_bridge	65.2	69.5	77.39	85.8	82.96	87.5	73.78	82.97
tv80	53.8	59.31	60.16	64.76	61.97	64.9	59.44	62.45
usb_funct	63.2	64.49	71.4	75.53	74.53	79.39	70.01	74.2
ethernet	47.6	49.56	53.94	63.79	71.37	83.14	50.75	54.92

by the computed seeds. As is common in industry, we use the launch-on-capture (LOC) scheme, also referred to as broadside scan, to apply test-vector pairs. Note that none of these two fault models were targeted by the test sets (they are only used as surrogate fault models). The concept of n-detection has been also used in the literature as a surrogate defect coverage model. However, as shown in [69] n-detection is not always indicative of defect coverage, therefore we do not use this metric in this Chapter. Finally, as mentioned in Section 6.3, for the proposed reseeding methods ($w = 1$, $w = 5$ and dynamic reseeding) the output-deviation metric considers both the responses of each test vector pair. On the other hand, [180] considers only one response (either the first or the second). Therefore, for generating results using [180], we chose to evaluate the generated seeds using the second response of each test-vector pair. This decision favors the timing-dependent defects of [180] i.e. the transition-fault coverage of its generated patterns.

First we evaluate the proposed encoding with respect to the achieved transition-fault coverage. The corresponding results are shown in Table 6.3. Columns 2, 4, 6 and 8 present the transition-fault coverage achieved by the classical defect-unaware, window-based (for $w = 1$ and $w = 5$) and dynamic reseeding approaches respectively, while columns 3, 5, 7 and 9 present the transition fault coverage achieved by the classical defect-aware, reseeding of [180] and the proposed defect-aware approaches. We see that in both the window-based and dynamic reseeding, the use of the proposed output- deviation metric increases the transition fault coverage significantly. Compared to [77], the method described in [180] achieves higher transition- fault coverage. Moreover, in nearly all cases, the proposed window-based and dynamic reseeding approaches offer higher transition-fault coverage than [180].

It is obvious from Table 6.3 that the defect coverage achieved by the proposed method for $w = 5$ is higher than the defect coverage achieved by the proposed method for $w = 1$ and dynamic reseeding. This is mainly a result of the increased diversity of the candidate seeds in case of $w = 5$. This diversity can be attributed in part to the fact that many seeds encode incompatible test cubes when $w > 1$. Note that the increased test sequence length in the case of $w = 5$ contributes also to the increased defect coverage compared to the other cases. However, according to the results shown in Table 6.3, this contribution is less significant than the contribution of the proposed encoding method. Specifically, in most cases, the defect- unaware window-based reseeding for $w = 5$ offers lower transition fault coverage than the defect-aware window-based reseeding for $w = 1$, even though the test sequences in the former case are much longer.

Figure 6.6 illustrates the transition fault coverge ramp-up achieved by the window-based reseeding method for $w = 5$ for selected circuits. In each chart, the x-axis presents the number of the applied vector pairs and the y-axis the transition-fault coverage. The seeds for the defect-unaware window-based reseeding method have been sorted: a) randomly (curves "Cmp(Rnd)"), and b) in descending order of their stuck-at-fault coverage (curves "Cmp(Stuck)"). The curves "Cmp & Def" correspond to the proposed defect-aware window-based reseeding method. It can be seen that the defect coverage of the "Cmp & Def" method is considerably higher than that for the other methods. Moreover, the proposed method exhibits higher coverage ramp-up than both the other methods, with the "Cmp(Stuck)" being better than the "Cmp(Rnd)". Finally, for the largest benchmark ethernet, which consists of 136.2K gates and 10.5K scan flip flops and is more representative of real-life industry circuits, the improvement in transition-fault coverage is striking. We have also verified that the "Cmp & Def" method in the case of window-based reseeding for $w = 1$ exhibits also higher ramp-up than the "Cmp" method.

Figure 6.7 shows the coverage ramp-up achieved by the dynamic reseeding method for the same circuits reported in Figure 6.6. As we can see, the proposed defect- aware method offers steeper coverage ramp-up than the baseline defect- unaware approach for the dynamic reseeding case as well.

The transition-fault coverage (or the coverage of any other fault model) can be further improved by using ATPG to generate top-off test cubes, and by subsequently compressing these test cubes using either static or dynamic reseeding. The advantage offered by this strategy, when combined with the proposed encoding method is twofold: first, the encoding of the baseline stuck-at test cubes using the defect-aware encoding will cover a large number of the targeted faults (i.e., the transition faults in our case) and thus the number of generated top-off test cubes will be relatively small. Second, if the encoding of the newly generated top-off test cubes is properly tuned using the proposed output-deviation metric, the generated seeds for them will offer high coverage of other unmodeled defects.

In our final experiment, we evaluate the proposed method and [180] in terms of the achieved bridging-fault coverage. For evaluating the examined reseeding methods in terms

Table 6.4: BCE^+ and random bridging-fault coverage results (%)

Circuit	BCE^+								Random Bridging Faults							
	Classical Reseeding		Window Based Reseeding				Dynamic Reseeding		Classical Reseeding		Window Based Reseeding				Dynamic Reseeding	
			w=1		w=5						w=1		w=5			
	[77]	[180]	Cmp	Cmp & Def	Cmp	Cmp & Def	Cmp	Cmp & Def	[77]	[180]	Cmp	Cmp & Def	Cmp	Cmp & Def	Cmp	Cmp & Def
s5378	95.06	95.22	95.49	95.91	95.77	96.4	95.48	96.01	94.14	94.35	94.85	95.19	95.72	96.26	94.99	95.32
s9234	87.66	87.39	88.76	89.11	88.61	89.08	89.1	89.22	86.56	86.58	87.95	88.29	88.7	89	88.05	88.37
s13207	92.85	92.93	92.96	93.78	93.43	94.15	92.81	93.2	91.99	92.14	92.08	92.95	92.92	93.57	91.82	92.14
s15850	94.2	94.17	94.45	94.58	94.58	94.76	94.5	94.62	93.47	93.59	94.38	94.51	94.71	94.89	94.45	94.54
s38417	98.2	98.1	98.29	98.56	98.27	98.48	98.43	98.6	97.13	97.15	97.88	98.15	98.26	98.44	97.85	98.03
s38584	90.36	90.35	91.36	91.56	92.43	92.81	90.81	91.04	89.85	89.91	90.89	91.09	91.67	91.98	90.41	90.52
ac97_ctrl	94.66	94.57	97.96	98.27	98.71	98.92	97.33	97.45	97.02	97.02	98.75	98.87	99.1	99.23	98.42	98.49
mem_ctrl	62.34	62.33	63.3	63.62	64.33	64.74	63.46	63.59	74.6	74.61	75.08	75.44	75.78	76.1	75.2	75.36
pci_bridge	96.04	96.05	98.27	98.46	98.68	98.86	98.13	98.18	96.78	96.82	98.14	98.28	98.45	98.55	98.06	98.06
tv80	91.4	91.37	93.49	93.79	93.7	93.94	93.62	93.78	89.26	89.33	90.86	91.23	91.57	91.74	90.91	91.1
usb_funct	93.71	93.71	95.47	96.1	96.03	96.48	95.38	95.65	95.15	95.19	96.73	97.16	97.17	97.45	96.63	96.83
ethernet	88.78	88.83	92.79	93.66	96.06	96.32	91.63	92.24	90.63	90.77	93.59	94.18	95.57	95.71	92.81	93.17

of their bridging fault coverage, both the BCE^+ metric and the random bridging fault coverage overviewed in Section 2.1.3 were used. As noted before, BCE^+ is not very accurate for estimating the real bridging fault coverage of a method, but it is very useful for comparing two different methods (the method with the highest value of BCE^+ is deemed to be more effective for defect screening). Table 6.4 presents the results. Regarding the proposed window-based and dynamic reseeding approaches, we find that in all cases, both BCE^+ values and random bridging-fault coverage indicate that the proposed defect-aware encoding "Cmp & Def" achieves higher coverage of bridging faults than the original "Cmp" method. In contrast, in the method described in [180], the improvement is small compared to the classical defect-unaware reseeding [77], and in some cases, there is even a decrease in the BCE^+ values. Moreover, all the proposed encoding methods offer higher BCE^+ values as well as bridging fault coverage than [180]. The main reason for this observation is that [180] considers only one of the two responses of each LOC vector-pair (either the first or the second) for calculating the output deviations. In our experiments, we considered only the second response, as stated earlier, to enhance the detection of timing related defects. However, bridging faults are detected by the first response (i.e., the response of each stuck-at test). This is another weakness of [180], compared to the proposed method, which is able to consider both responses of each pair. Consequently, we conclude that the proposed method improves the bridging fault coverage, which is also a significant advantage over [180].

6.6 Conclusions

We have presented a defect-oriented LFSR reseeding technique that allows us to detect unmodeled defects using stuck-at test sets in a test- compression environment. This technique is based on the output-deviations metric for grading the test patterns produced by the LFSR seeds. We have considered both static and dynamic reseeding, and evaluated unmodeled defect coverage using transition faults and bridging faults as surrogate fault models. Our results show that compared to compression-driven LFSR reseeding, which is largely in use today, higher defect coverage and faster coverage ramp-up are obtained using stuck-at tests and output deviations, without any loss of compression.

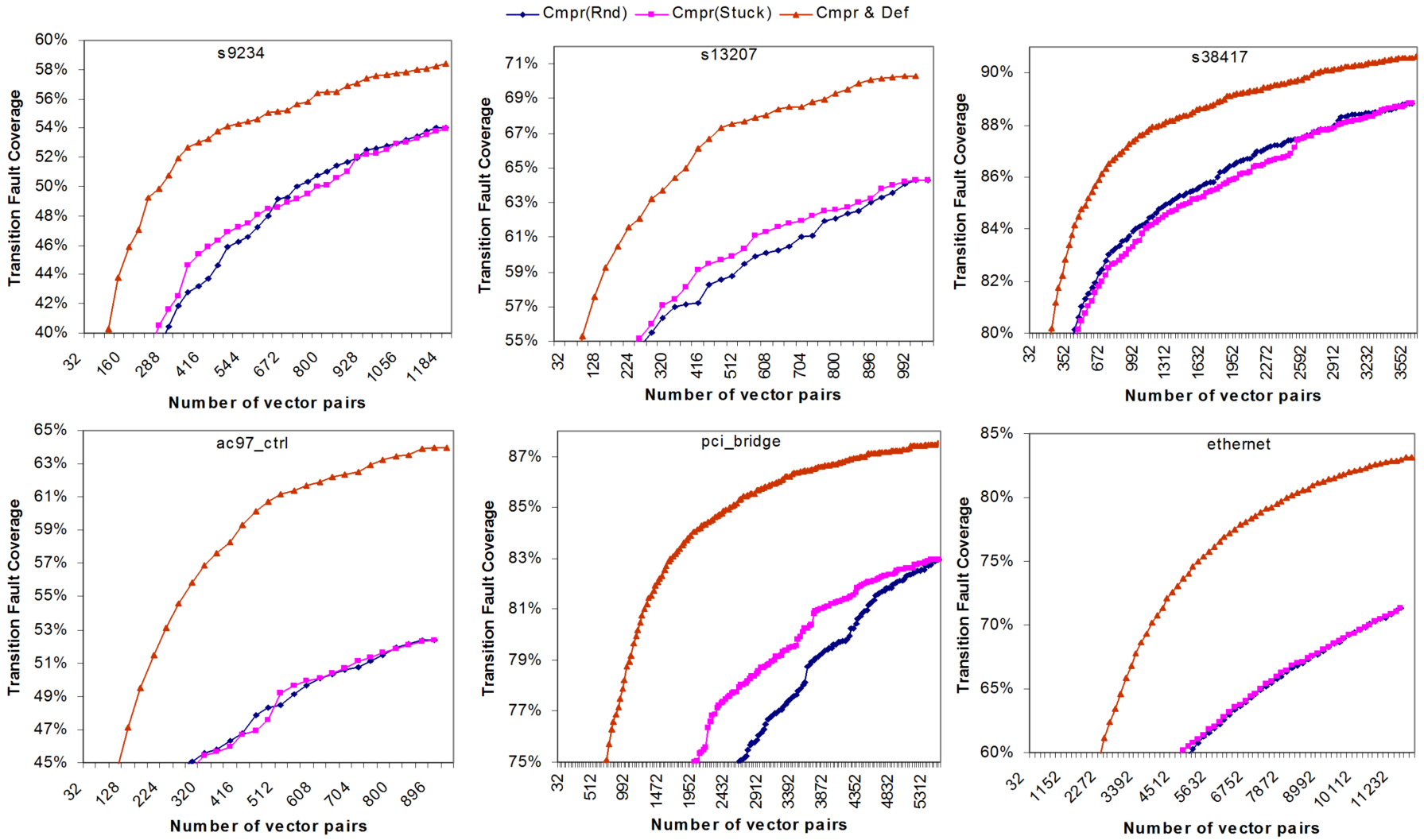


Figure 6.6: Transition fault coverage ramp-up for window-based reseeding ($w = 5$)

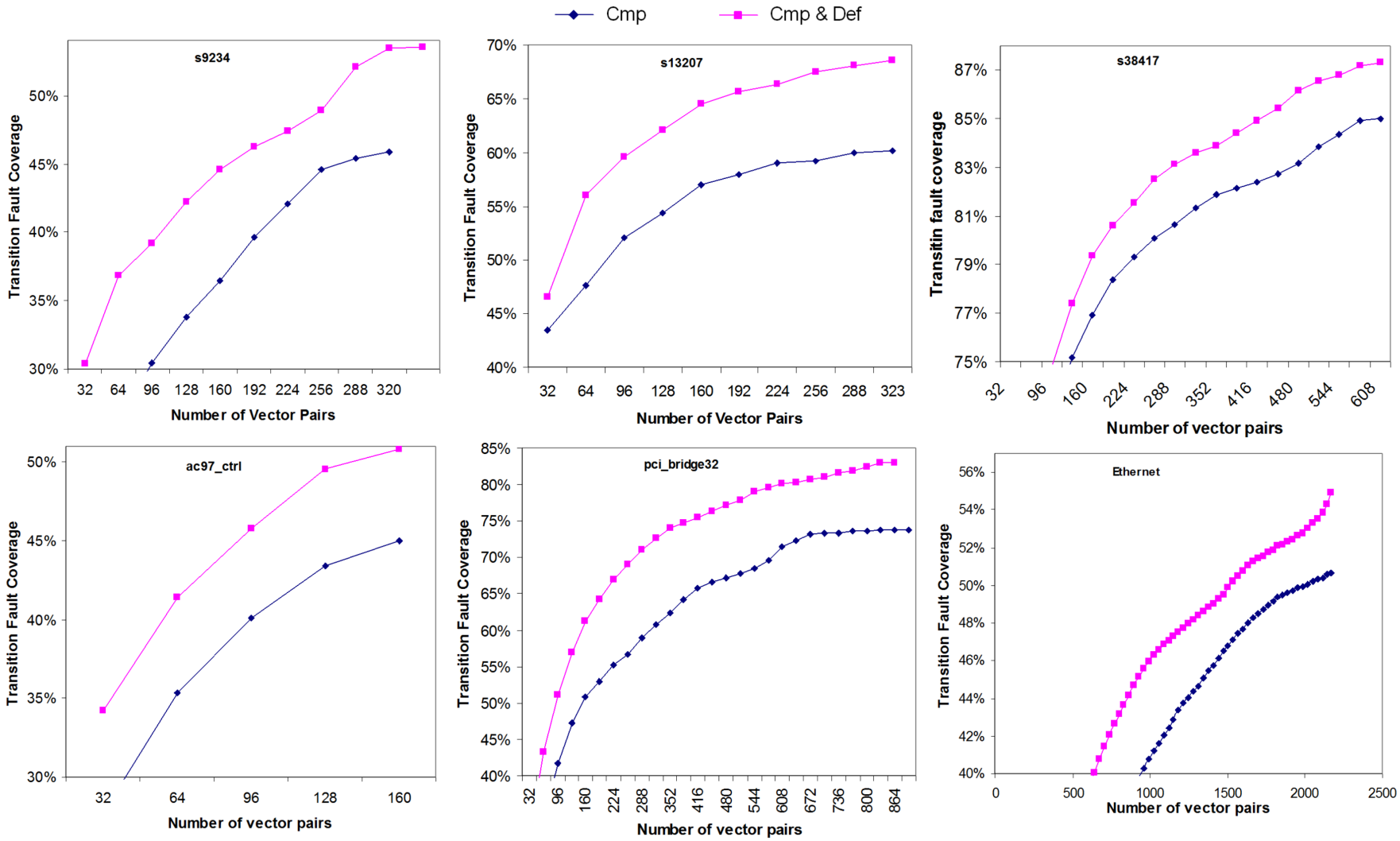


Figure 6.7: Transition fault coverage ramp-up for dynamic reseeding

Chapter 7

Low-Power and High-Quality Test Data Compression

Test data decompressors targeting low power scan testing introduce significant amount of correlation in the test data and thus they tend to adversely affect the coverage of unmodeled defects. In addition, low power decompression needs additional control data which increase the overall volume of test data to be encoded and inevitably increase the volume of compressed test data.

In this Chapter we show that both these deficiencies can be efficiently tackled by a novel pseudorandom scheme and a novel encoding method. The proposed scheme can be combined with existing low power decompressors to increase unmodeled defect coverage and almost totally eliminate control data.

The first Section of this Chapter applies the scheme on dynamic LFSR reseeding and the second Section on an optimal selective Huffman decompressor. Extensive experiments using ISCAS and IWLS benchmark circuits [1] show the effectiveness of the proposed method when it is combined with state-of-the-art decompressors.

7.1 Linear-based Decompressor

In this Section a new low cost scheme which can be combined with classical linear decompresssors to improve the unmodeled defect coverage of the generated test vectors and at the same time to reduce shift power is presented. The proposed method exploits inherent properties of test sets to generate multiple diverse power-efficient encodings of test cubes, and it selects those offering the highest unmodeled defect coverage using the outputs de-

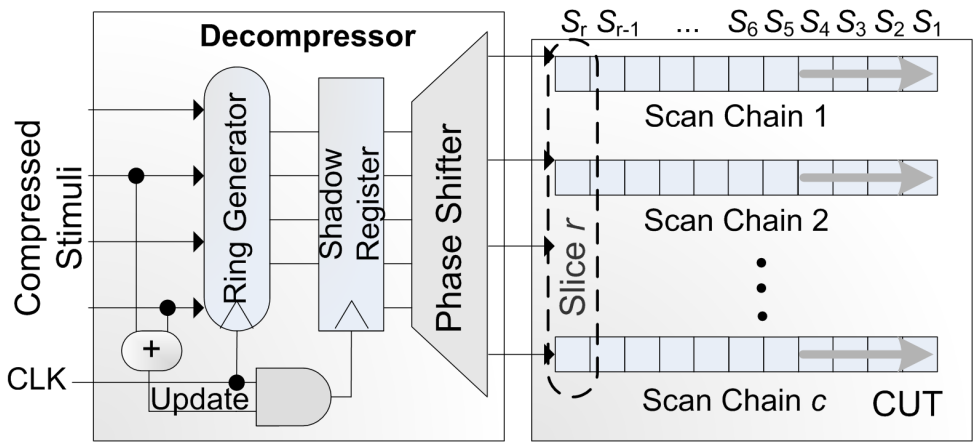


Figure 7.1: Low power decompressors

viation evaluation metric proposed in Chapter 5. Contrary to the state-of-the-art low power decompressors, the proposed scheme does not increase the volume of test data to be encoded and thus it achieves higher compression.

The proposed architecture is simple, test set independent and can be combined with linear and code-based decompressors. In particular, it can be combined with the state-of-the-art linear decompressors presented in [29], [26], [105] as well as with the symbol-based decompressors presented in [60], [72] to improve both their unmodeled defect coverage and their compression efficiency. Extensive experiments show the effectiveness of the proposed method in terms of shift power, test data volume (TDV), test application time (TAT) and unmodeled defect coverage measured as coverage of surrogate fault models. We note that, to the best of our knowledge, this is the first test data compression technique for low power testing which targets unmodeled defect coverage.

7.1.1 Motivational Example

Excessive shift power during scan testing has been traditionally tackled by exploiting unspecified bits ('x') of test cubes (i.e. test vectors consisting of '0', '1' and 'x' logic values) in order to reduce the pairs of successive complementary test bits shifted into scan chains. For example, the Fill Adjacent technique [17] fills 'x' values in such a way as to load successive scan cells with the same logic value in order to minimize transitions during scan-in. Even though this (and other similar techniques) is very effective in reducing the shift power, the generated test vectors tend to suffer from low unmodeled defect coverage compared to the test vectors generated by randomly filling the 'x' values [8].

Using a similar concept, the linear decompressors proposed in [84] decrease shift power by partitioning test data of each scan chain into blocks. Each unspecified value ('x') in blocks is filled with the last encountered specified value. When the block size is small the shift power is considerably reduced because many blocks are generated as repeated versions of the same specified bits. However, one additional control bit per block is needed which increases the test data. A similar approach was adopted in [29], [26], [105] with the

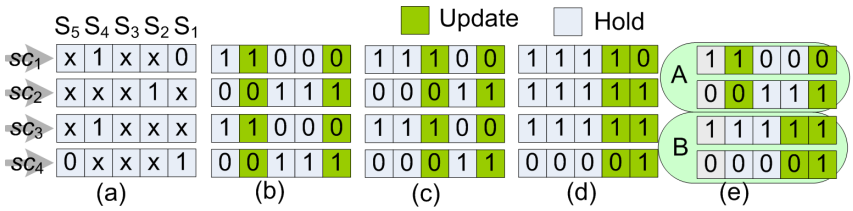


Figure 7.2: Example of encoding using shadow registers

addition of a shadow register between the linear decompressor and the phase shifters, and it has already been discussed in Section 2.3.1. It was shown that this type of encoding requires additional data to control the signal Update of the shadow register.

Specifically, the decompressor generates the test data in slices i.e., groups of c bits concurrently loaded into the c scan chains. Whenever a group G_k of k ($k > 1$) successive test slices of a test cube are compatible (i.e., every slice in this group exhibits no bitwise incompatibilities with any other slice in this group) one test slice S_k which is compatible with all test slices of group G_k is encoded and it is loaded into the scan chains for k successive clock cycles. This is achieved by the use of a shadow register located between the ring generator and the phase shifter (see Figure 7.1). When S_k has to be generated for the first time, the ring generator generates and transfers the test data corresponding to slice S_k to the shadow register by setting signal *Update* to logic value '1' (Update operation). During the next k successive clock cycles, the shadow register holds its contents by setting the *Update* signal to logic value '0' and thus it continues loading the scan chains with the same slice S_k (Hold operation).

Even though these methods reduce the shift power they suffer from limited unmodeled defect coverage due to the correlation induced in the way the 'x' values of test cubes are filled during the decompression. We will show that the adverse effects of this correlation on the unmodeled defect coverage can be significantly reduced by following a different encoding method. An example is presented below:

Example 7.1. Figure 7.2a presents a test cube for a circuit with 4 scan chains. Figure 7.2b presents the test vector generated when this cube is encoded using a shadow register. The decompressor encodes slice 01x1 (this is the result of merging the compatible slices S_1,S_2,S_3), as well as the slice 1x10 (this is the result of merging slices S_4, S_5) and loads them into the shadow register using two Update operations at the 1st and 4th scan cycle highlighted in Figure 7.2b (the 'x' values are randomly filled). The rest of the slices are generated using Hold operations. This encoding provides low switching activity but it is not unique. There are other groups of compatible successive slices that can be encoded, as shown in Figure 7.2c and Figure 7.2d (the 2nd Update operation is applied sooner). ■

Different power efficient encodings of test cubes generate different test vectors which detect different unmodeled defects. If the proper encoding for each test cube is selected then the unmodeled defect coverage of the generated vectors will improve. Higher volume of power efficient and diverse test vectors can be generated by partitioning scan chains into groups loaded by separate and independently controlled shadow registers. For example,

suppose that scan chains SC_1, SC_2 in the example of Figure 7.2 are loaded from shadow register A and scan chains SC_3, SC_4 are loaded from shadow register B. Then there are three possible power efficient encodings for test data of scan chains SC_1, SC_2 and another four encodings for scan chains SC_3, SC_4, providing thus 3×4 different encodings (we do not count the first Update operation as it is always applied before the loading of the first slice). One example is shown at Figure 7.2e.

A similar (but for a different purpose) approach was proposed in [29]. Specifically, in [29] it was noted that multiple independently controlled shadow registers can be potentially used for further reducing the shift power during scan testing. However this approach causes test data volume expansion. Consider for example a core consisting of 100 scan chains and 100 scan slices (i.e., each scan chain consists of 100 scan cells) and let us assume a typical fill rate of 1% (i.e., in average each test cube consists of 100 specified and 9900 unspecified bits). Then the number of control bits per cube is equal to 100 (one control bit per slice) which have to be encoded in conjunction with the 100 specified bits of the test cube. This results to duplicating the test bits to be encoded and inevitably results to reduced overall compression. If we use 2, 3, 4, ... shadow registers in the same example the test data to be encoded increase by 3x, 4x, 5x, ... which renders this approach impractical.

In this Section we show that the large amount of unspecified bits in test cubes can be exploited to almost eliminate these control data. This enables the application of an advanced encoding method which offers a wide variety of unique power-efficient encodings. These encodings are screened by an output-deviation based metric which selects the encodings offering the highest unmodeled defect coverage. The proposed method is based on a pseudorandom scheme which controls the shadow register(s) independently of the decompressor at no additional overhead on control data even when a large number of shadow registers are used. When this pseudorandom scheme is combined with linear and symbol-based decompressors it achieves a significant reduction of the volume of compressed test data as it eliminates the need for controlling the shadow register. We note that the proposed method can be combined with techniques like scan chain disabling [29] to reduce capture and scan out power as well.

7.1.2 Proposed Method

Basic Concept

Consider a decompressor and a shadow register partitioned into g modules $SR_1, \ldots, SR_g$ as shown in Figure 7.3. Each module SR_i drives a different group of scan chains and $Update_i$ is the Update signal driving module SR_i. Let TS be a set of test cubes generated using ATPG for a certain type of faults. The basic characteristic of the proposed method is that the Update operations of each module SR_i are determined prior to the encoding

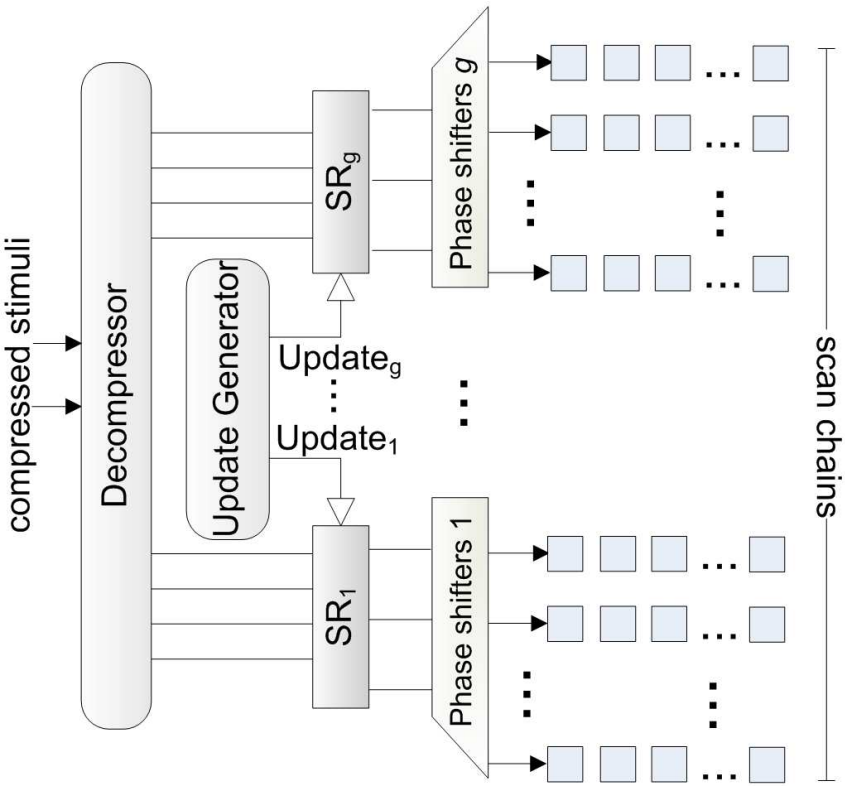


Figure 7.3: Proposed Architecture

process by using a pseudorandom binary sequence[1] PS_i generated using a probability $P_{update-i}$. $P_{update-i}$ is the probability signal $Update_i$ to be set to logic value '1'. The encoding of test cubes of TS is adjusted to sequences PS_i. Specifically, if $PS_i(S_j) = 0$ (i.e. $Update_i = 0$, during generation of slice S_j) then SR_i holds its contents and the decompressor does not provide data to the SR_i (i.e. no test bits are encoded). If $PS_i(S_j) = 1$ ($Update_i = 1$, during generationg of slice S_j), then the contents of SR_i are updated with test data from the decompressor which are calculated in order to match all subsequent slices $S_{j+1}, S_{j+2}, \ldots$ which will be generated without updating the shadow register i.e. $PS_i(S_{j+1}) = PS_i(S_{j+2}) = \cdots = 0$.

The generation of the control sequences PS_i is very important for the effectiveness of the proposed method. Test cubes which exhibit bitwise incompatibilities in slices corresponding to successive Hold operations in sequence PS_i are not encode-able for PS_i (the potential of PS_i to encode the test cubes of a test set is hereafter referred to as the encode-ability of PS_i). A large number of Hold operations (i.e., a small value of $P_{update-i}$) degrades the encode-ability of PS_i while a large value of $P_{update-i}$ improves it (note that a value $P_{update-i} = 1$ can encode every test cube). However, a large value of $P_{update-i}$ increases the number of Update operations and thus the number of complementary bits shifted into the scan chains. So, when $P_{update-i}$ increases, shift power increases too. On

[1]The term *pseudorandom sequence* is used in a different meaning than in the rest of the literature. It refers to the way the shadow registers are controlled. The encoding of test cubes **still remains deterministic**.

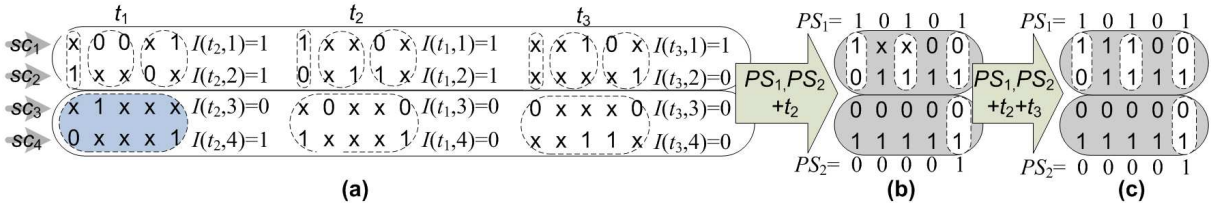


Figure 7.4: Encoding example

the other hand, a small value of $P_{update-i}$ introduces high correlation in test data and the unmodeled defect coverage tends to drop (many adjacent scan cells are assigned the same logic value). Note that PS_i, determines only the groups of compatible slices for any encoded test cube (their specified bits are not affected). The generation of each PS_i sequence is part of the encoding method described next.

Encoding Method

At first we introduce a metric which is representative of the incompatibilities of test cubes and shows the likelihood a test cube to be encode-able using a pseudorandom sequence PS_i. Let t be a test cube. The volume of incompatibilities, $I(t, m)$, of scan chain $m \in [1, SC]$ for t, is defined as the number of successive complementary bits of t corresponding to scan chain m. Note that test cubes consist also of 'x' logic values which affect measure $I(t, m)$ based on the way they are filled. Since this is not known before the encoding, we adopt the following approximation: every 'x' logic value shifted into the scan chain is considered to be equal to the last specified logic value '0' or '1' which was encountered during the loading of this scan chain for t. This is a reasonable approximation as the proposed encoding tends to fill test cubes in a similar manner. Note that the 'x' values of test cubes are not actually filled which remain unaffected by this process. For example, for the test cube t of Figure 7.2a we have $I(t, 1) = I(t, 4) = 1$, $I(t, 2) = I(t, 3) = 0$. The volume of incompatibilities $I(t)$ for test cube t is defined as the maximum value $I(t, m)$ for any of its scan chains $m \in [1, SC]$.

A test cube with a high (low) value of $I(t)$ is considered as a hard-to-encode (easy-to-encode) test cube due to its high (low) volume of incompatibilities between successive scan cells[2]. The same classification is done among scan chains. Specifically, for every scan chain m, $IS(m) = \sum_{t \in TS} I(t, m)$ is the measure of its incompatibilities among all test cubes of a test set TS. A scan chain m with high (low) value of $IS(m)$ is considered a hard-to-encode (easy-to-encode) scan chain. Both these measures can be used to improve the encoding process. The $IS(m)$ values are used to partition the scan chains into groups where each group is driven by its own shadow register. The $I(t)$ values are used to bias the encoding process towards the early encoding of the most hard-to-encode test cubes which can decrease the overall volume of test data.

[2]Among two test cubes t_1, t_2 with $I(t_1) = I(t_2)$ the most hard to encode is the cube with the highest value among $\sum_m I(t_1, m)$, $\sum_m I(t_2, m)$.

At first the scan chains are partitioned into a pre-determined (selected by the designer) number of groups, g, according to their IS values (scan chains with similar IS values are grouped together). Since every group is independently controlled by a separate shadow register module, groups consisting of scan chains with small IS values are assigned a low initial value $P_{update-i}$, as the encode-ability of the corresponding pseudorandom sequences is not affected and the gains in switching activity reduction are high. For groups consisting of scan chains with large IS values large initial $P_{update-i}$ values are assigned to enhance the encode-ability of the respective sequences. Let G_i be the number of incompatibilities of group i defined as the sum of the IS values of the scan chains comprising group i. Let SR_w ($w \in [1, g]$) be the module driving the group of scan chains with the highest volume (G_{worst}) of incompatibilities. Then the $P_{update-w}$ value for SR_w is set equal to a parameter P_{init} selected by the designer among a number of discrete values $P_1, P_2, \ldots, P_k$. The value of P_{init} is set according to the design objectives for shift power and test sequence length since if offers a trade-off between these two objectives. A low value of P_{init} provides low shift power but increases the test sequence length. A high value of P_{init} increases the shift power but offers shorter test sequences. The initial probabilities $P_{update-i}$ of the rest modules are set to lower values which are calulated proportionally to P_{init}. **Specifically, $P_{update-i}$ is set equal to the rounding of the value $P_{init} \times G_i/G_{worst}$ to a discrete value in the set $P_1, P_2, \ldots, P_k$ (note that $G_i/G_{worst} < 1$).**

After the initial value of every $P_{update-i}$ is determined, the sequence PS_i of each group corresponding to the first generated vector (i.e., for the first r cycles, where r is the length of the longest scan chain) is generated. This is achieved by the means of a trivial LFSR-based pseudorandom unit which will be presented in Section 7.1.2. The test cubes of TS are then examined for encode-ability for the given sequences. The encode-able cubes are those cubes which consist of slices without bitwise incompatibilities when they are successively loaded using Hold operations. These cubes are encoded as follows: every test slice S_j corresponding to an update operation ($PS_i(S_j) = 1$) and its following slices $S_{j+1}, S_{j+2}, \ldots$ corresponding to hold operations ($PS_i(S_{j+1}) = PS_i(S_{j+2}) = \cdots = 0$) are merged into one test slice which is encoded by the decompressor and it is loaded into SR_i when the update operation is applied. The encoding begins from the most hard-to-encode test cubes in order to minimize both the test data volume and the test sequence length. Additional test cubes can be encoded by the same sequence PS_i provided that a) they are encode-able for the sequence PS_i at hand, b) they are bitwise compatible with the previously encoded (by the same sequence PS_i) cubes and c) the decompressor has variables left to encode them. When no more test cubes can be encoded this process continues to the next vector (i.e., the sequence PS_i for each SR_i is generated for the next r cycles and the encoding continues with the remaining cubes). The following example illustrates the encoding process.

Example 7.2. Figure 7.4 shows three test cubes t_1, t_2, t_3 and their I values. Based on I values the IS values are: $IS(sc_1) = 1 + 1 + 1 = 3$, $IS(sc_2) = 2$, $IS(sc_3) = 0$, $IS(sc_4) = 1$. Scan chains sc_1, sc_2 form the first group with $G_1 = 3 + 2 = 5$ and

scan chains sc_3, sc_4 form the second group with $G_2 = 0 + 1 = 1$. Let $P_{init} = 1/2$, $k = 8$ and $[P_1, P_2, \ldots, P_k]$= $[1/16, 1/8, 1/4, 1/2, 3/4, 7/8, 15/16, 1]$. Since $G_1 > G_2$ we have $P_{update-1} = P_{init} = 1/2$ and $P_{update-2} = P_{init} \times G_2/G_1 = P_{init} \times 1/5 = 1/10$ which is rounded to the closest discrete P_i value, that is $P_2 = 1/8$. Let us assume that based on these probabilities the sequences $PS_1 = 10101$, $PS_2 = 00001$ are generated i.e. for the shadow register driving the first group three Update operations occur at the 1st, 3rd and 5th scan slice; for the shadow register driving the second group one Update operation occurs at the 1st slice. The test slices that must be compatible in order the test cubes to be encoded using PS_1, PS_2 are shown for t_1, t_2, t_3 inside dotted lines. Test cube t_1 is not encodeable for PS_2 as the test slices of the second group are not compatible (they are shown highlighted in Figure 7.4). On the contrary, t_2, t_3 are both encode-able for PS_1, PS_2. Test cube t_2 is more hard-to-encode than t_3 because $I(t_2) = I(t_3)$ but $\sum_m I(t_2, m) > \sum_m I(t_3, m)$ and thus it is encoded first. Only the contents of the shadow registers at the Update operations (shown inside dotted lines in Figure 7.4b) are encoded by the decompressor. The remaining of the slices are generated using Hold operations. After encoding t_2, few unspecified bits still exist which offer the potential for encoding also cube t_3. The final test vector is shown in Figure 7.4c. ■

Certain incompatibilities in scan chains prohibit the encoding of some test cubes. When no test cubes can be further encoded for a number of successive test vectors, we increase $P_{update-i}$ of every group to the next higher discrete value and we initiate a new pseudorandom session. Each pseudorandom session is retained for as long as test cubes are encoded. In every successive session a different sequence is used for every signal $Update_i$ with increased rate of Update operations and thus more test cubes become encode-able.

As the values of $P_{update-i}$ increase in successive pseudorandom sessions, the switching activity increases (see Section 7.1.2) and its peak value may reach a predetermined limit. This happens because the remaining test cubes have many incompatibilities and thus they need a large number of Update operations which cannot be easily matched by pseudorandom sequences unless probabilities $P_{update-i}$ increase a lot. This means that the pseudorandom mode fails to further adhere with the power specifications of the circuit and it terminates. Then the deterministic mode is initiated with a global signal controlling all shadow registers like being one (the control data are encoded in this case as proposed in [29]).

The above encoding process owns its efficiency to the low fill rates of test sets. Specifically, as it is common in test sets, the vast majority of test cubes are sparsely specified while only a very small fraction of them are densely specified. The proposed method efficiently encodes the first ones during the pseudorandom sessions and the second ones during the deterministic session. In order to show the effectiveness of pseudorandomly generated sequences PS_i to encode large test sets we performed an experiment using the *Ethernet* circuit of IWLS suite [1]. This circuit consists of more than 10,000 scan cells and a dynamically compacted test set for complete coverage of stuck-at faults for this circuits is almost 12 Mbits in size; therefore it is more representative of realistic industrial

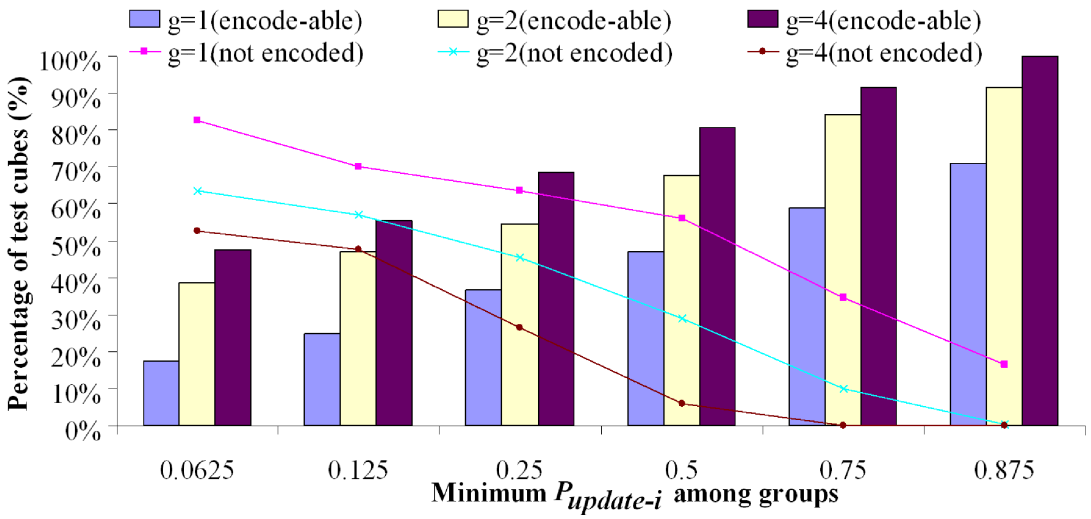


Figure 7.5: Percentage of encode-able test cubes for the *Ethernet* benchmark

designs than the rest of the benchmark circuits. Figure 7.10 presents the percentage of test cubes which are encode-able for various sequences PS_i generated pseudorandomly. We run three different experiments by using $g = 1$, 2 and 4 shadow register modules. The x-axis presents the minimum value $P_{update-i}$ used among the groups (this value is increased from left to right of the x-axis as successive pseudorandom sessions are applied). At each pseudorandom session 100 different pseudorandom sequences PS_i were generated and the percentage of test cubes which are encode-able for at least one of them is reported by the means of bars.

It is obvious that at each successive session more test cubes become encode-able for the generated sequences. In addition, as the number of shadow register modules increases more test cubes become encode-able as the pseudorandom sequences match in a better way the specific characteristics of each group of scan chains. The curves show the test cubes which remain not encoded at the end of each pseudorandom session (test cubes which are encode-able for any generated sequence are immediately dropped in this case). It is obvious that the vast majority of test cubes are easily encoded at the first sessions which offer very low switching activity. Especially in the case of $g = 4$ shadow register modules all test cubes are encoded very fast and the deterministic mode can be eliminated (no test cubes remain unencoded after the 5th session). Thus it is evident that the effectiveness of the proposed pseudorandom encoding depends on the specified bits density of test cubes, which is fairly low in large circuits, and not on the size or amount of test cubes. Therefore, we conclude that the proposed method is scalable to very large test sets.

Unmodeled Defect Coverage Improvement

The encoding of test cubes is done in two steps: a) at first n different encodings are generated which all offer the same high compression and low shift power (the method to generate candidates on dynamic LFSR reseeding can be found in Chapter 6) and b) the n test vectors corresponding to the n encodings are screened for detecting unmodeled defects

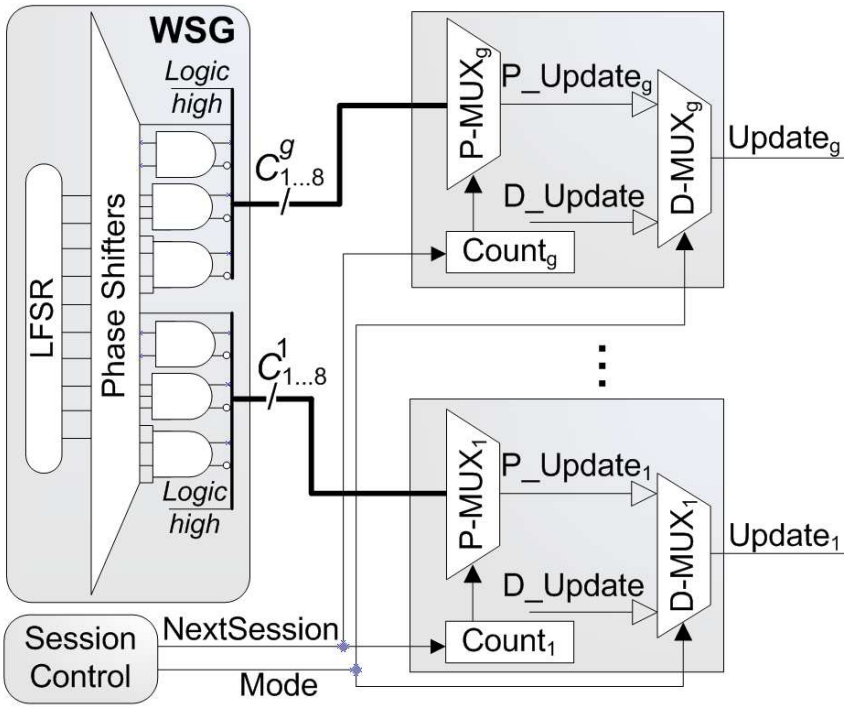


Figure 7.6: Update Generation Module

and the most promising one is selected (the best candidate is selected using the output deviations metric proposed in Chapter 5). Specifically, the n most hard-to-encode test cubes $t_1, t_2, \ldots, t_n$ which are encode-able by the current pseudorandom sequence PS are selected and n candidate encodings $e_1, e_2, \ldots, e_n$ are generated. Each candidate encodes one of the n selected test cubes and as many additional test cubes as possible. All n-candidates offer high compression as they encode hard-to-encode cubes and the same low switching activity as they use the same sequence of Update/Hold operations. However, they generate different test vectors which detect different defects.

Proposed Architecture

The proposed architecture consists of the decompressor and the shadow register modules SR_i shown in Figure 7.3 as well as of the Update Generation unit shown in Figure 7.6. This module consists of the weighted signals generation unit (WSG), multiplexers (P-MUX$_i$) selecting between pseudorandom signals with different probabilities, a control unit which triggers the initiation of each next session and multiplexers (D-MUX$_i$) used for switching from pseudorandom to deterministic mode. At each clock cycle any number of 0 up to g modules may concurrently update their contents (note that as it is shown in Figure 7.3 different outputs of the decompressor feed each shadow register module, thus the decompressor can even load all modules at the same cycle if necessary).

The weighted signals generator (WSG) consists of a small LFSR which is initially loaded with a known random seed and a very small combinational logic which generates pseudorandom signals with various probabilities in the range [0, 1]. This is achieved by feeding the outputs of different LFSR cells to combinational gates. For example, the

output of a two-input AND gate driven by two LFSR cells has probability $P_{out} = (1/2)^2 = 25\%$. We verified that k =8 signals $C_1, C_2, \ldots, C_8$, with probabilities $0 < P_1 < P_2 < \ldots < P_8 \leq 1$ respectively, are sufficient to implement our scheme with negligible cost. Note that a phase shifter is also included in the WSG unit in order to provide multiple groups of linearly independent pseudorandom signals $C^1_{1\ldots8}, \ldots, C^g_{1\ldots8}$. Each of these groups of signals is used to drive a different shadow register module.

One among the signals $C^i_1, \ldots, C^i_8$ is selected by each P-MUX$_i$ (which is an $8 \rightarrow 1$ multiplexer in our case) for generating the PS_i sequence to drive signal $Update_i$. Signals $C^i_1, \ldots, C^i_8$ are connected to the inputs of P-MUX$_i$ in ascending order of signal probability, i.e, C^i_1 is connected to the first multiplexer input, C^i_2 is connected to the second input etc. Thus, in order to increase the probability that controls each group, a higher order input of P-MUX$_i$ is selected using a counter Count$_i$ which is very small (equal to 3 bits each for the case at hand). Count$_i$ stores the selection address of P-MUX$_i$ (let say value $1 \leq sel \leq 8$) for the entire session (it selects C_{sel}, thus $P_{update-i} = P_{sel}$). The value of Count$_i$ remains unchanged throughout every session and increases by one every time a new session is initiated. Note that each session has its own control counter Count$_i$. Although these counters are triggered simultaneously, their initial value is different and depends on the incombatibilies observed on the group of scan chains the counter controls, as described in Section 7.1.2.

In order to simplify the decompression process, at every successive session all counters simultaneously increase by one and thus every value $P_{update-i}$ increases to the next higher probability of WSG. This is triggered by internal registers of Session Control unit which are loaded from the ATE before the decompression process begins with the number of test vectors applied at each session. Since at most 8 pseudorandom sessions are applied ($k = 8$ probabilities are used) the area required for these session-registers is negligible. After the last pseudorandom session, Session Control switches the D-MUX$_i$ to the input D_Update which is a global signal common for all groups and the deterministic mode begins. The control data for signal D_Update are encoded by the decompressor.

The proposed scheme shown in Figure 7.6 operates as a "low power converter" of the test cubes. It converts the test data from the decompressor into low power vectors compatible with the test cubes of TS. It is independent of the decompressor used to encode the converted data and thus it can be combined with linear as well as symbol-based decompressors.

7.1.3 Experimental Results

We implemented the standard dynamic reseeding (SDR), the state-of-the-art low power dynamic reseeding proposed in [105] (LPDR) and the proposed method using the C++ programming language. For all the methods we used the same ring generators as decompressors and their size was selected by the $s_{max} + 20$ rule, where s_{max} is the number of specified bits of the most specified test cube. We run experiments on a 4-cores CPU Linux workstation. The complexity of the proposed method follows the complexity anal-

Table 7.1: Comparisons TSL, TDV, TDF & BF (%)

Circuit	TDV (in Kbits)			TSL (# of vectors applied)			ASA			TDF Cov. (%)			BF Cov. (%)		
	SDR	LPDR	Prop.	SDR	LPDR	Prop.	SDR	LPDR	Prop.	SDR	LPDR	Prop.	SDR	LPDR	Prop.
s5378	5.6	10	6.9	273	305	331	50.1	5.8	12.4	63.54	62	66.72	94.9	94.39	95.11
s9234	11.3	20.9	13.9	477	504	597	49.6	11.6	19.3	47.41	49.81	52.59	88.47	88.17	88.81
s13207	10.9	20.8	13.4	342	419	415	50.1	5.4	13.3	62.27	61.25	69.43	92.55	92.09	93.05
s15850	14.7	26.8	18.4	498	552	611	50.1	7	11.7	55.4	54.68	58.39	95.96	94.42	94.67
s38417	64	97.5	69.3	1685	1548	1875	50	6.2	17.9	87.38	87.81	88.32	98.1	98.21	98.19
s38584	51.1	89.7	59.4	1115	1179	1281	50	7	13	67.68	67	68.52	91.65	91.57	91.74
ac97_ctrl	41.2	67.2	44.4	1547	1543	1665	50	3.8	3.9	57.74	57.44	66.88	99.54	99.49	99.53
pci_bridge	148.7	233	154.9	3614	3435	3731	53.3	2.6	5.7	83.83	81.88	84.6	98.6	98.56	98.6
tv80	40.3	72.5	47.6	2257	2330	2684	49.9	10.8	12.8	61.06	59.88	64.27	91.24	91.06	91.37
usb_funct	73.9	123.7	84.9	1709	1748	1895	50	5.2	11.7	74.67	74.32	77.02	97.35	97.32	97.44
ethernet	299.3	494.9	322	2385	2501	2574	50	3.3	12	53.19	53.21	57.01	93.47	93.35	93.61

ysis discussed on Chapter 6. The number of candidates T used for the proposed method was set to $T = 30$ and our parallel implementation indicates execution times almost 2.5 times the execution time of the LPDR method.

We conducted experiments on the largest ISCAS'89 and a subset of the IWLS'05 [1] benchmark circuits. We examined various scan chain configurations and we selected the one that yielded the best result for the baseline SDR method and then all other methods used that scan chain configuration. For each circuit a test set TS was generated using a commercial ATPG engine targeting complete coverage of stuck-at faults.

For LPDR a single shadow register was used to keep its TDV low. We implemented the shadow register control using both techniques proposed in [29], [105] (internal XOR tap or one additional ATE channel) and the best result is reported. For the proposed method we used four shadow register modules, $n = 30$ candidate encodings and the threshold on peak switching activity was set close to that of LPDR. Various initial values of $P_{update-i}$ were used and the best results are reported. The WSG unit implements k=8 probabilities: 1/16, 1/8, 1/4, 1/2, 3/4, 7/8, 15/16, 1. The ATE-repeat command was utilized to reduce the TDV for all methods. We further improve the TDV of both LPDR and SDR methods by filling free variables in a repeat-friendly way similar to [123]. In the proposed method all free variables are filled in a non-repeat-friendly way to improve output-deviations. For all the results we present the test data volume (TDV), the test sequence length (TSL) and the average scan-in switching activity (ASA) measured using the metric of [29], [105].

For evaluating the unmodeled defect coverage we used two surrogate fault models, namely the transition delay (TDF) and the bridging fault model (BF). **None of these models were targeted by the stuck-at test sets encoded**. For detecting transition faults each stuck-at test vector generated by the decompressors is applied on the circuit using two capture cycles according to Launch-On-Capture (LOC) technique. For the bridging fault model 100K pairs of lines were selected randomly for each circuit. For each pair, four bridging faults were simulated by considering both lines as aggressors and victims, and both logic values '0' and '1' at the aggressors. Fault simulations were carried out using a commercial tool. Note that similar approaches were adopted in many techniques (e.g. [8], [178]) for evaluating the unmodeled defect coverage.

Table 7.1 presents the TDV in Kbits, the TSL in number of vectors applied and the ASA values for each method (note that the same number of clock cycles is needed in all cases to generate, load and apply each test vector). Columns 2-10 present the TDV, TSL and ASA values of SDR, LPDR and the proposed method. For the proposed method various initial values of P_{update_i} were used (the best results are reported). The SDR approach offers the best compression but its ASA is unacceptable. LPDR offers very low ASA, but increases the TDV compared to SDR considerably due to the additional data required for controlling the shadow register. The proposed method offers short TSL and small TDV, which approach the respective values of SDR method, and very low ASA which approaches that of LPDR. The superiority of the proposed method compared to LPDR in respect to TDV stems from the fact that almost no control data are required

by the proposed method (the proposed method requires control data only during the deterministic mode which constitutes a very small portion of the test mode). The ASA of the proposed method is a little higher than that of LPDR, but it is still very low and in most cases lower than 12.5%. This value is important because it is the 1/4 of the switching activity of the power unaware SDR method, which exhibits normalized switching activity of 50%. It is widely accepted that the test mode consumes 4 times the power consumed during normal operation, so it is expected the normal operation to require around 12.5% normalized weighted switching activity. Note, that the results of 7.1 were taken by minimizing the TSL of the proposed method in order to have comparable TSL with that of LPDR. The proposed method can achieve further ASA reduction but that would increase the TSL. As a result, the proposed method requires power consumption that it would most probably comply with the power specifications of the circuit, which is the most critical goal for low power scan testing.

The last 6 columns of Table 7.1 present defect coverage comparisons. As it was expected, in the majority of the cases the LPDR method offers reduced defect coverage compared to the SDR approach. In almost all cases the proposed method achieves much higher TDF and higher BF coverage than both LPDR and SDR methods. We also note that the improvement of the proposed method against the other methods in terms of BF coverage is less than the improvement in terms of TDF coverage. However, this is due to the fact that the bridging fault coverage is very high in all cases and thus there is no much potential for further improvement. In particular, the average (over all circuits) number of bridging faults that remain undetected after the application of the proposed method is less than 2.6% of the total number of faults simulated. This clearly show that the proposed technique has already achieved very high bridging fault coverage.

Figure 7.7 presents the TDF coverage ramp-up achieved for the representative ac97_ctrl benchmark circuit. It is obvious that the use of the proposed encoding combined with the output deviation-based metric offers higher coverage and coverage ramp-up than the rest methods reducing thus the TAT in an abort-at-first-fail environment.

For evaluating the hardware overhead we synthesized the proposed scheme for a) one and b) four shadow register modules. The proposed decompressors including all units (i.e., ring generator, shadow register, phase shifter, WSG, P-MUX, etc) is 15% larger in case (a) and 55% larger in case (b) than the decompressors of LPDR for a single shadow register module which are admittedly very small. These results indicate that the hardware overhead of the proposed method depends on the number of shadow registers, but they were computed by duplicating all the NAND gates partitipating for the generation of each $C^i_{1...8}$ signal for each group of scan chains that corresponds to a shadow register. Note that in all the experiments conducted maximum four shadow register groups were used and the method was efficiently applied with relatively small hardware overhead. However, if more power reduction is required then one approach would be to use more shadow registers. At the unlike case (since hardware cost is no longer an issue in the post-Dennard Era) where the hardware overhead of the proposed method would be formidable, then it could

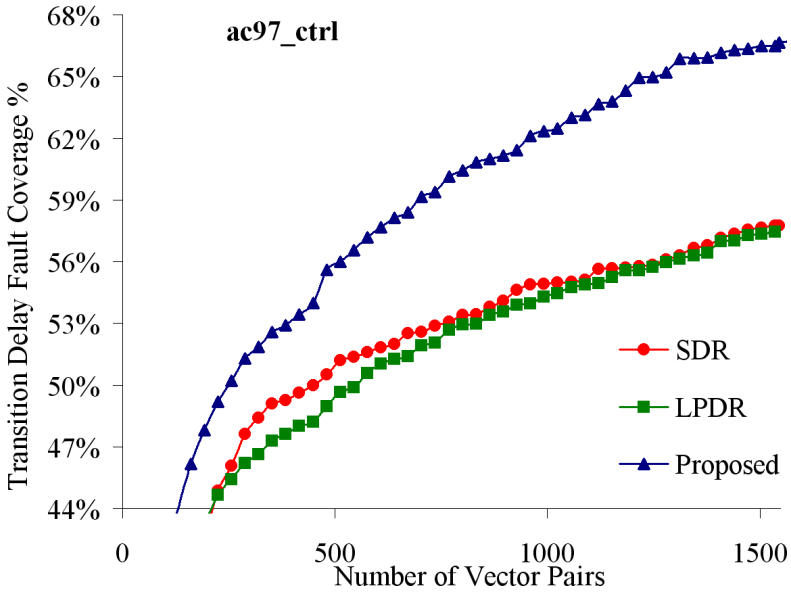


Figure 7.7: Transition delay fault coverage ramp-up

be reduced with the following approaches. Large initial $P_{update-i}$ values for some groups might render some NAND gates from the WSG units useless and they can be removed from the synthesis. Moreover, even itentionally some NAND gates can be excluded from the WSG unit and also left out of the model (decreasing the choices of probabilities, but reducing hadware overhead). Another approach would be to reuse the NAND gates that generate $C^i_{1...8}$ signals between different groups provided that their outputs are not utilized simultaneously. This last hardware-overhead minimization basic idea is that even if 100 groups are used, then the number of 2-input NAND gates that are required to be embedded at the WSG unit depends only on the simultaneously used values 0.25 and 0.75 signal probabilities (these two values correspond to the two outputs of a 2-input NAND gate: normal output Q and complementary Q'). If only two groups from the 100 use simultaneously these probability values, then only two 2-input NAND gates need to be embedded and the hardware overhead can be reduced considerably.

Finally, in order to show the advantages of the proposed scheme when combined with other decompressors, we implemented the statistical encoding proposed in [72] as follows: the test data corresponding to each scan slice are partitioned into multiple constant-length blocks and each block is encoded using the selective Huffman code. Data sent by the ATE are decoded using an Optimal Selective Huffman decompressor (see Section 2.2.3), and the decoded blocks fill a register with length equal to the number of scan chains. When all blocks of a test slice are loaded into the register the slice is loaded into the scan chains. The proposed scheme is applied to this decompressor as follows: the aforementioned register plays the role of the shadow register which is partitioned into modules. Each codeword is used to encode the test data required to load a shadow register module whenever it is updated according to the pseudorandom sequences. The parts of the register corresponding to shadow register modules which are not updated at a scan cycle require no codewords (no encoding is done for these modules similar to the encoding

method described in Section 7.1.2). Thus, the proposed method in this case reduces both TDV and TAT (less codewords and clock cycles are required for loading each test slice into the scan chains). We provide results only for the ac97_ctrl benchmark circuit for 32 scan chains, block size equal to 8 bits and number of encoded blocks equal to 16. When the proposed scheme is applied to this decompression architecture, the TDV drops from 55.3Kbits to 24.7 Kbits, the ASA drops from 36.4% to 4.6% and the test application time is reduced by 25.6% compared to the OSH approach discussed in Section 2.2.3. At the same time TDF coverage increases from 41.8% to 50.24% and BF coverage increases from 95.8% to 98.6%. Thus the proposed method offers considerable gains in this case too. The next Section presents extensively this idea.

7.2 Code-based Decompressor

This Section presents a novel low-cost decompression architecture that combines the advantages of both symbol-based and linear-based techniques and offers a very attractive unified solution that removes the barriers of existing test data compression techniques. Besides the traditional goals of high compression and short test application time, the proposed method also offers low shift switching activity and high unmodeled defect coverage at the same time.

The novel unified test data compression approach is based on code-based decompressors and accomplishes both low power and high quality testing. Moreover, it favors multi-site testing as it requires a very low pin-count interface to the Automatic Test Equipment. Finally, contrary to existing techniques, it provides an integrated solution for testing multi-core SoCs as it is suitable for cores of both known and unknown structure that usually co-exist in SoCs.

The contributions of this work are:

1) It exploits both the low fill rate and the correlations in the specified bits of test cubes and thus outperforms both symbol-based and linear-based encoding methods.
2) It offers low shift power during testing.
3) It supports very low pin-count interface as a single ATE channel suffices for fast downloading test data on-chip.
4) It offers short test application times as it exploits the large number of scan chains of modern cores.
5) It does not require any kind of synchronization between the ATE and the Circuit Under Test (CUT).
6) It is decoupled from ATPG process and offers high compression even on highly compacted test sets of IP-cores.
7) It is suitable for both IP and non-IP cores of modern SoCs.
8) It exploits ATE's repeat command (wherever available) as an embedded feature of the encoding process to further decrease test data volume.

In addition we show that statistical codes introduce significant correlation in the gen-

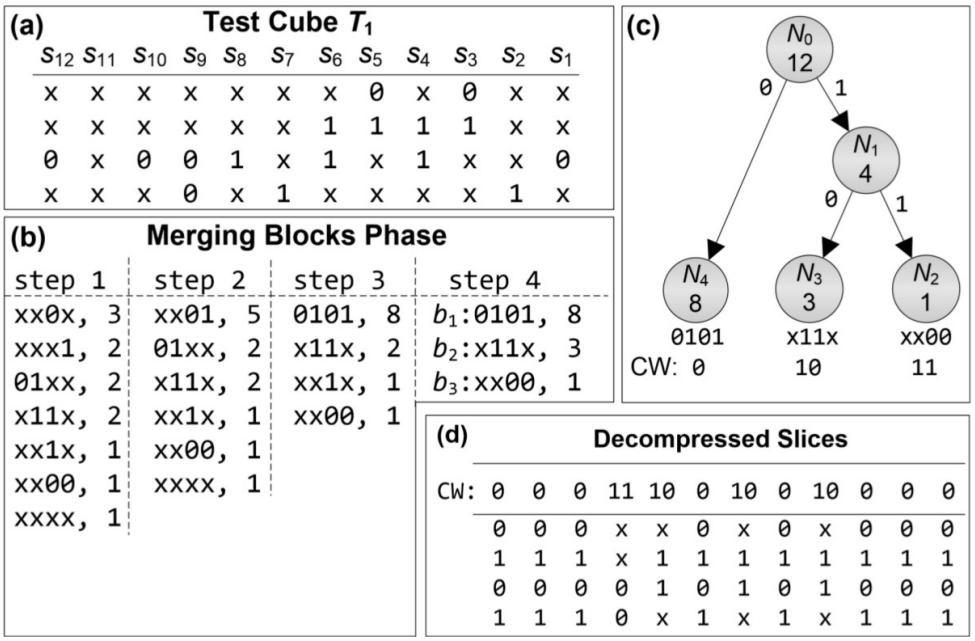


Figure 7.8: Classical selective Huffman coding

erated test vectors and thus offer low unmodeled defect coverage. To this end, a new technique is proposed that offers a trade-off between test data volume and unmodeled defect coverage. This objective has not been targeted yet in the literature by any code-based TDC technique.

Finally, we present a low-cost decompression architecture that can be shared among multiple cores without compromising compression. In particular, it offers the potential to share hardware resources and test data, in order to cost effectively test multiple cores with different characteristics. The proposed decompressors can be shared even between IP and non-IP cores and thus they offer an additional advantage to achieve higher quality test solutions for SoCs at lower cost. Extensive experiments with largest ISCAS and a subset of large IWLS benchmark circuits [1] show the benefits of the proposed method as compared to already existing TDC methods.

7.2.1 Motivation

Optimal Selective Huffman was described in Section 2.2.3. At this point we present an OSH example that intends to help the reader understand the proposed method.

Example 7.3. In Figure 7.8a test cube T_1 is presented which is partitioned into $r = 12$ scan slices $s_1, s_2, ..., s_{12}$. Scan slice s_i is the part of T_1 that simultaneously loads the n scan chains $SC_1, SC_2, ..., SC_n$ at clock cycle c_i (the scan slices are loaded from right to the left, i.e. scan slice s_1 is loaded first, s_2 is loaded second etc). The number of scan chains is $n = 4$ and the block size is $l = 4$ (each scan slice is considered as a test data block). The test data blocks and their numbers of occurrences are shown in the first column of Figure 7.8b. As it was proposed in [60] the two most frequent compatible

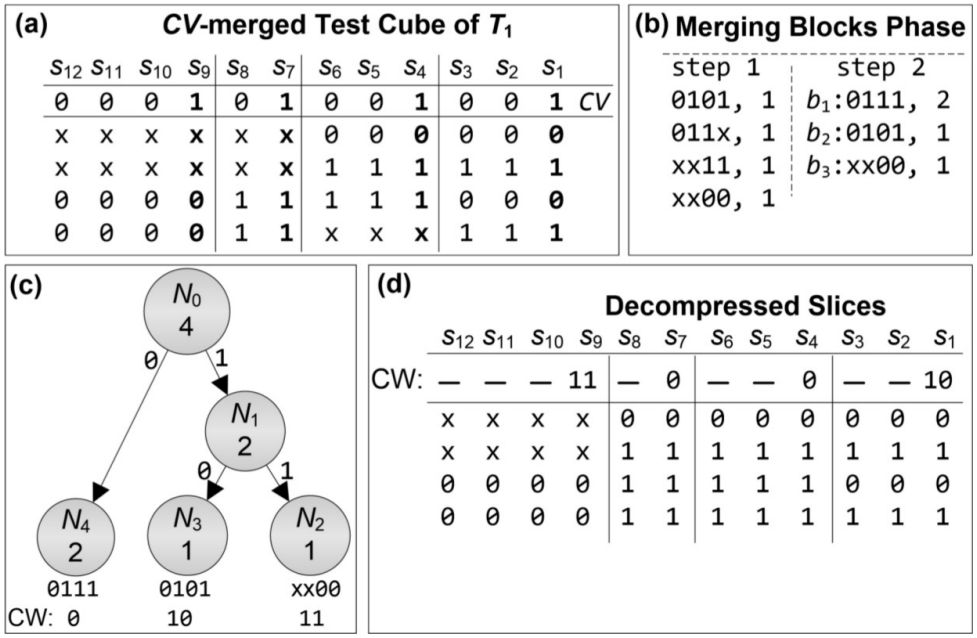


Figure 7.9: Selective Huffman coding with pre-merged blocks

blocks are merged (two blocks are compatible when they do not differ at any bit position where they are both specified). The merging provides a new block which has the same specified bits with the two blocks at the same bit positions with them. This is iteratively applied until no further merging of blocks is possible. Columns 2-4 of Figure 7.8b present the merged blocks generated after the most frequent blocks are merged each time (next to each block its frequency of occurrence is reported). The final encoded distinct blocks are presented in the fourth column of Figure 7.8b and they constitute the dictionary of the OSH. The Huffman tree that corresponds to the dictionary example of Figure 7.8a is shown in Figure 7.8c. The codewords assigned to entries '0101', 'x11x' and 'xx00' are '0', '10' and '11' respectively. The compressed test set has a size of 16 bits and it is shown in the first row of the "Decompressed Slices" in Figure 7.8d (one codeword per slice). The decompressed test data are shown below that row. ■

7.2.2 Basic Idea

Despite the fact that there are many blocks in a test set consisting mostly (or even entirely) of 'x' values, each and every one of them has to be encoded using a separate codeword. However, the blocks corresponding to scan slices s_1, s_2, s_3 of the test cube are compatible and thus they can be merged into a single block. Another block can be generated by merging scan slices s_4, s_5, s_6, s_7, s_8 and another one by merging $s_9, s_{10}, s_{11}, s_{12}$. If we encode these three blocks instead of the blocks shown in the fourth column of Figure 7.8b then we can use the first block for loading the scan chains for the first 3 cycles, the second block for the next 5 cycles and the third block for the last 4 cycles. This way, we use only three codewords to encode the test cube. In order to know during each scan-in cycle if

the last block is repeated or not we need one control bit per block. If the next block is the repetition of the previous one this bit is set to logic value '0', otherwise it is set to logic value '1'. The control bits of all slices of a test cube comprise the *Control Vector* (CV). The CV that corresponds to the merging of scan slices $s_1 \dots s_3$, $s_4 \dots s_8$ and $s_9 \dots s_{12}$ is $CV = 000100001001$ (the rightmost bit corresponds to s_1 and the leftmost to s_{12}). This CV indicates that: a) blocks $s_1 \dots s_3$, $s_4 \dots s_8$ and $s_9 \dots s_{12}$ are merged, b) the resulting blocks are loaded at the 1st, 4th and 9th clock cycles and they are repeated for another 2, 4 and 3 clock cycles respectively. The test cube formed after merging its blocks according to CV is called *CV-merged*.

The storage of CV along with the two codewords results to worse compression than the original encoding shown in the first example of this Chapter. However, for sparsely specified test cubes (that is test cubes with many 'x') there are many different CVs that can be used. Even for the test cube at hand, which consists of 33.3% specified bits and is rather densely specified, there are many CV that can be used, like for example 000100001001, 000100000011, 000100100101, etc (as shown in many studies [123, 125] the vast majority of test cubes of industrial designs consists of less than 5% of specified test bits). Each different CV implies a different merging process. We exploit this property **to generate pseudorandomly the CV vectors based on probabilistic properties of test cubes**. **CVs are generated prior to the encoding and they do not need to be stored somewhere** and thus apply a *blocks pre-encoding merging phase* which minimizes the number of blocks which will be encoded using OSH. This process can be better illustrated by the means of an example.

Example 7.4. Let us assume that for test cube T_1 of the first example of this Chapter the control vector $CV = 000101001001$ has been pseudorandomly generated. Figure 7.9a presents the CV-merged T_1 for this CV. There are four groups of merged blocks while only the first block of each group is encoded using OSH (these blocks are shown in bold and correspond to the logic value '1' in CV). The rest of the blocks are simply repeated versions of the first block in each group. The CV constitutes the guide for the encoding process as it is shown in Figure 7.9b. The number of occurrencies of the blocks are calculated based on the number of times each block has to be encoded (note that only the blocks corresponding to logic values '1' in the CV, i.e. the first one of each group, are encoded and thus the number of occurrencies of each block is equal to 1). The generated code and its tree are presented in Figure 7.9c. Finally, Figure 7.9d shows the codeword used for each block corresponding to the logic value '1' in the vector CV, as well as the decompressed blocks loaded in the scan chains. It is obvious, since CV is not stored and it is generated pseudorandomly, that only 6 bits are needed to compress the given test cube. ■

The compressed bits in the above example are much less than the specified bits of the test cube. Linear methods cannot reduce the size of the compressed test set below the volume of its specified bits unless methods like [31] are employed. At the same time the unspecified values of test cubes are compressed adequately. In addition, the encoding of

the most frequently occurring blocks decreases the size of the encoded test data and thus exploits the correlation between specified test bits. Moreover by loading the same values in successive scan slices reduces also the shift power during scan-in (and consequently the scan-out shift power as it was shown in [19]). Further optimizations can be achieved by employing techniques like [31] on top of the proposed technique.

The idea of grouping scan slices for the purpose of low power shift-in has been proposed earlier in [29, 32]. This grouping requires the encoding of additional control data and specifically one bit per test slice. Every control bit is encoded together with the test data bits. Depending on the number of scan chains, the volume of control bits can be high compared to the average number of specified bits for a test set. Assume that the 10K scan cells of the largest circuit in our suite, the Ethernet, are organized into 100 scan chains (100 test slices per vector), 100 control bits must be encoded for any test cube. For an average fill rate of 1% (i.e. 100 specified bits per test cube), along with the 100 specified bits for each test cube we need to encode an additional 100 control bits. In this Section we show that control data can be completely avoided and still achieve very high shift power reduction.

7.2.3 Encoding Method

Generation of Control Vector

The encoding method and the generation of the pseudorandom vector CV are strongly interdependent processes. Each CV implies a specific merging of the blocks of test cubes. Some of the test cubes will have their respective blocks compatible and thus they can be generated using this particular CV (we call hereafter those test cubes CV-mergeable, or mergeable for the CV). The probability that a given CV can be used to pre-merge a given test set depends on two factors:

1) The volume of '0' logic values of CV: a large volume of '0' logic values imposes extensive compatibility restrictions between successive test slices of test cubes and decreases the possibility of an arbitrary test cube t to be mergeable for this CV. The opposite happens when the CV consists of many '1' logic values. For example, every test cube is mergeable for the all-ones CV as every slice is encoded independently of the rest of the slices.
2) The volume of 'x' values of test cubes: the higher is this volume, the higher is expected to be the number of compatible slices of the test cubes and thus a larger population of CVs can be used for such test cubes.

CVs with many '0' values can be used for merging sparsely specified test cubes while CVs with many '1' values are needed for the densely specified test cubes. However, from the compression and power perspective, CVs with large volumes of '0' values are more effective than CVs with large volumes of '1' values (in the first case less blocks are encoded and less transitions occur during the scan-in as more blocks are repeated versions of their preceding blocks). A good practice is to begin from CVs with a small volume of '1' values

to merge sparsely specified test cubes, and then gradually increase this volume to merge the remaining densely specified cubes.

A vector CV is generated pseudorandomly as a signal that is set to logic value '1' with a probability P_{CV}. P_{CV} is set equal to various discrete probability values $p_1, p_2, \ldots, p_b$ ($0 \leq p_1 < p_2 < \cdots < p_b = 1$). P_{CV} is initially set to p_1 and then it is gradually increased to p_2, p_3, etc. For each value of P_{CV} many CVs are generated and each one is used to load one test vector in the scan chains. Every CV is generated by a pseudorandom unit which will be described in the next Section. This unit is synthesized prior to the encoding process and thus the exact CV sequences are known during the encoding. Note that if we set $P_{CV} = 1$ throughout the whole encoding process, then all blocks are encoded using traditional OSH [72]. Therefore, this approach is a generalization of OSH.

In order to show the effectiveness of pseudorandomly generated CVs to encode large test sets we performed an experiment using the *Ethernet* circuit of the IWLS suite [1]. This circuit consists of more than 10,000 scan cells, so it is more representative of realistic industrial designs than the rest of the IWLS benchmark circuits. Figure 7.10 depicts the percentage of test cubes which are mergeable for various CVs generated pseudorandomly (note that the generated test cubes achieve 100% coverage of stuck-at faults). The x-axis presents P_{CV} values used in this experiment for generating CVs. For each P_{CV} value, 100 different CVs were generated and the percentage of test cubes which are mergeable for at least one of the CVs is reported by the means of bars. As the value P_{CV} increases, more test cubes become mergeable for the generated CVs. The curve shows the test cubes which remain not merged at the end of each step (test cubes which are mergeable for any CV generated are immediately dropped in this case). Note, that more than 80% of the test cubes are mergeable for CVs consisting of less than 25% logic values '1'. Also, less than 2% of the test cubes require CVs with 50%-75% of the logic values being equal to '1'. As a result, the vast majority of blocks do not have to be encoded at all (for the above experiment more than 70% of blocks are generated as repeated versions of other encoded blocks and thus they require no test data). In conclusion, the effectiveness of CV depends on the fill rate of test sets, which is fairly low in large circuits, and not on the size or amount of test cubes. Therefore, the proposed method is scalable to very large test sets.

Control vectors also affect static and/or dynamic compaction of test cubes. Note that the term compaction refers to a different process than encoding or compression. In particular, static compaction is the process of merging all compatible test cubes to be encoded later (static compaction preceeds encoding), while dynamic compaction is the process of merging test cubes during the encoding (after encoding the first test cube, additional compatible test cubes are encoded in the same test vector generated by the decompressor). These processes, applied during (or prior to) the encoding process, decrease the volume of test vectors generated and offer additional compression and test time benefits. Both types of compaction can be applied in the proposed method. In fact, they can be applied even more aggressively than in linear-based methods, reducing the volume of applied vectors.

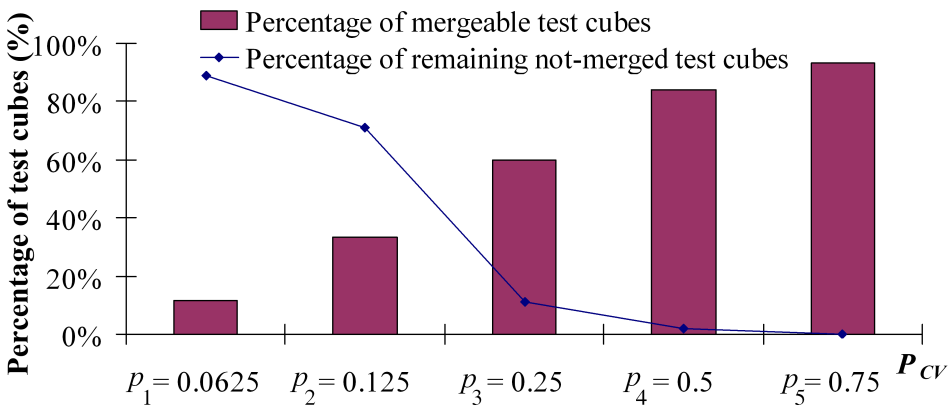


Figure 7.10: Percentage of mergeable test cubes for ethernet

Pre-Encoding Merge Process

The pre-encoding merge process is a step-by-step process which is applied before the OSH encoding. The objective of this process is to reduce as much as possible the volume of test cubes to be encoded, and decrease thus both the test data volume (less test slices to be encoded) and the test sequence length (less test vectors to be generated). At the beginning, all test cubes are appended in a set TS and the probability value of CV is set to the minimum discrete value p_1. Then an iterative process begins and at each iteration a new CV is generated based on the value of P_{CV} (note that one CV is a sequence of r logic values which are needed for loading one test vector into the scan chains). Then the test cubes of set TS that are mergeable for the generated CV are identified and they are moved to an empty set MS. If no test cube is mergeable for the current CV then the probability P_{CV} is increased to the next higher discrete value, and the iteration starts over by generating a new CV. One test cube of set MS is selected and it is merged using the CV. Then, all test cubes of set MS which cannot be statically compacted with t_1 are removed from set MS and they are appended again back to set TS. Out of the remaining test cubes in MS one test cube is selected, let say t_2 and it is statically compacted with t_1. This is iteratively applied until set MS becomes empty. While TS is not empty the process continues, until TS becomes empty, by generating the next CV for the current P_{CV} value.

At each iteration, among the test cubes of MS, we select first the hardest-to-merge test cubes, i.e., those test cubes that are less likely to be mergeable by CVs generated using low values of P_{CV}. This is done in order to increase the number of test cubes that are merged at the early stages for low P_{CV} values; the ones that offer better compression and shift power than higher P_{CV} values. To this end we rank the test cubes using a measure which is representative of this likelihood. Let t be a test cube and $i \in [1, n]$ be a scan chain of the core. The volume of incompatibilities, $INC(t, i)$, of scan chain i for test cube t, is the number of successive test slices of t with complementary logic values at their positions corresponding to scan chain i. Note that test cubes consist also of 'x'

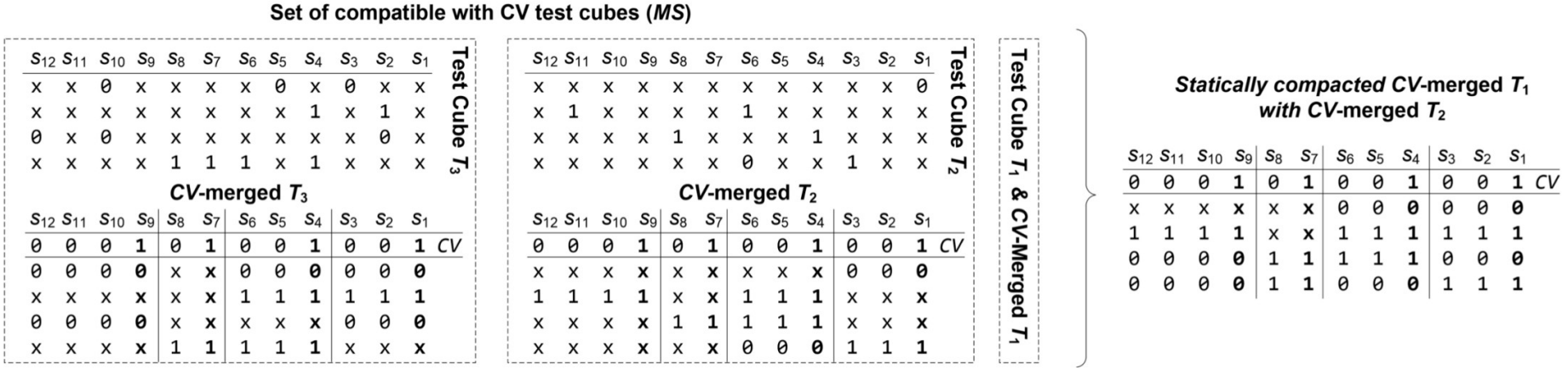


Figure 7.11: Pre-Merging Example

logic values which affect this measure based on the way they are filled. To alleviate this problem we adopt an approximation according to which every 'x' logic value shifted into the scan chain is considered to be equal to the last specified logic value '0' or '1' which was encountered during the loading of this scan chain for cube t. Note that the 'x' values of test cubes are not actually filled and thus test cubes remain unaffected by this process. For example, for the test cube t of Figure 7.8a we have $INC(t,1) = 0$, $INC(t,2) = 0$, $INC(t,3) = 2$ and $INC(t,4) = 1$. This approximation is reasonable as the proposed method fills the 'x' values in a similar manner, therefore there is a high probability that the 'x' values will be eventually filled in this manner (for example this is the case for the cube when it is encoded as shown in Figure 7.9). Finally we calculate the volume of incompatibilities $INC(t)$ of test cube t as the sum of the incompatibility values $INC(t,i)$ of all scan chains $i \in [1,n]$. The test cube with the highest value of $INC(t)$ is considered as the hardest-to-be-merged test cube and it is always the first one selected from set MS.

Example 7.5. Consider the test cube T_1 of the first example and the CV shown in Figure 7.9, and assume that T_1 along with test cubes T_2 and T_3 form test set TS shown in Figure 7.11 (T_1 is ommited in Figure 7.11). T_1, T_2, T_3 are all merge-able for the CV shown in Figure 7.9 and thus they are moved to set MS. The incompatibility values are $INC(T_1) = 3, INC(T_2) = 1, INC(T_3) = 0$ and thus T_1 is first selected to be encoded (the CV-merged version of T_1 was shown in Figure 7.9a). We can see that the the CV-merged versions of T_2, T_3 (shown below T_2, T_3 in Figure 7.11) are both compatible with the CV-merged version of T_1. Since $INC(T_2) > INC(T_3)$ cube T_2 is selected next and the CV-merged versions of T_1, T_2 are statically compacted as shown at the right of Figure 7.11. The CV-merged cube T_3 is no longer compatible with the resulting CV-merged test cube and thus it is moved back to set TS to be encoded using a new CV. ■

Slice Partitioning

So far, we have assumed that a whole slice is encoded as a single Huffman block. This is realistic when the size of a slice (and thus the volume of scan chains) is small (we recall that OSH achieves good compression for relatively small sized blocks [60], [72]). For cores with many scan chains we partition the set of scan chains into groups of equal size and each group is encoded separately. Let l be the required block size and n be the number of scan chains. Then the set of scan chains is partitioned into $k = \lceil n/l \rceil$ groups $G_1, G_2, \ldots, G_k$. Each group is assigned its own $CV_1, CV_2, \ldots, CV_k$ which is generated by its own properly selected probability P_{CV_j}. This partitioning process is applied before the merging phase and the same groups are retained for the whole test period.

Scan chains with similar volumes of incompatibilities (as they were calculated in the previous Section) are appended to the same group. Then, groups with low volume of incompatibilities are assigned lower initial probability P_{CV} than those groups with higher volume of incompatibilities. This increases the probability of test cubes to be mergeable with the generated CVs. In order to keep the encoding process (and the decompression

1: TS: set of test cubes, n: number of scan chains, l: size of block
2: Partition n scan chains into $k = \lceil n/l \rceil$ groups
3: Calculate P_{CV_j} value for each group G_j.
4: **while** TS is not empty **do**
5: Generate $CV_1, \ldots CV_k$ of r bits each using $P_{CV_1}, \ldots P_{CV_k}$.
6: Move test cubes which are mergeable for the CVs into MS.
7: **if** set MS is empty **then**
8: increase all $P_{CV_1}, P_{CV_2}, \ldots P_{CV_k}$ values to the next higher discrete value and go to the next iteration
9: **else**
10: Statically compact and drop test cubes of MS.
11: Move test cubes remaining in MS back to set TS.
12: Encode the resulting blocks using repeat-friendly OSH.

Figure 7.12: Encoding process

process) simple, each time that a set $CV_1, CV_2, \ldots CV_k$ fails to encode a test cube (i.e. no test cube is mergeable for this set of control vectors), then each one of the probabilities $P_{CV_1}, P_{CV_2}, \ldots P_{CV_k}$ is increased to its next higher discrete value until all of them reach the highest discrete values (i.e., those that are equal to p_1 are increased to p_2, those that are equal to p_2 are increased to p_3 etc).

First we present how the scan chains are partitioned into disjoint groups of size l each. Let i be a scan chain. We define the *incompatibility load* $L(i)$ of scan chain i as the sum of the $INC(t, i)$ values of all test cubes t of test set T, that is $L(i) = \sum_{t \in T} INC(t, i)$. The incompatibility load of each scan chain is then normalized by the worst load, i.e. the highest value $L_{max} = max\{L(i)\}$ found for any scan chain i, using formula $NL(i) = L(i)/L_{max}$. Then, the n scan chain are appended in a list of ascending order of their $NL(i)$ values. The first l scan chains of this list comprise the first group G_1, the next l scan chains comprise the next group G_2, etc. Finally we define the normalized load $NL^G(j)$ of group G_j as the maximum $NL(i)$ value of its members i, with $0 \leq NL^G(1) \leq NL^G(2) \leq \cdots \leq NL^G(k) = 1$.

After partitioning the scan chains into groups, the initial probability P_{CV_j} used for generating CV_j for each group G_j is determined. This probability depends on two factors: a) the relation between the normalized load values of different groups and b) the trade-off between compression, power and test application time as set by the test engineer. The test engineer selects the initial probability, let say 'a', for generating CV_k that is the CV of the group with the largest load. The rest of the groups with lower load than G_k are automatically assigned an initial probability which is lower than 'a' proportionaly to their NL^G value. Specifically, for each group G_j a probability value pg_j is calculated using the formula $pg_j = a \times NL^G(j)$ (we remind that $NL^G(j) \leq 1$). In that way group G_k is assigned the probability 'a' (note that $NL^G(k) = 1$) and the rest of the groups are assigned lower probabilities. The higher is the value of 'a', the lower is the TSL as many test cubes are mergeable right from the beginning and higher levels of static compaction can be reached.

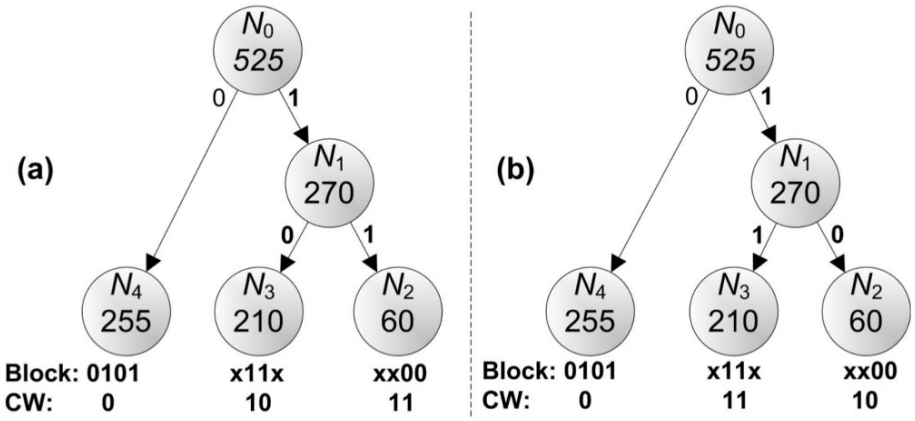


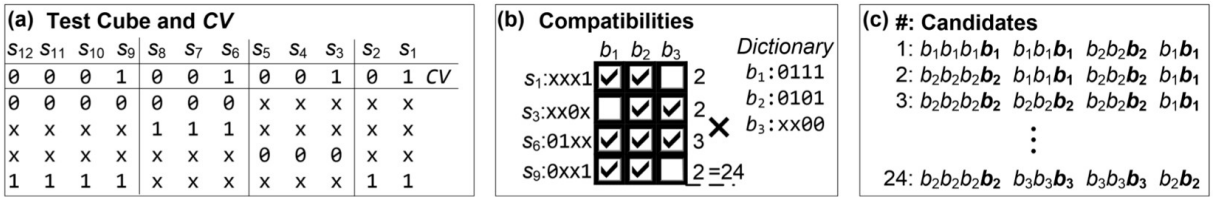
Figure 7.13: Swap procedure on node N_1 for repeat-friendly code

However, a large value of 'a' usually results to less compression and higher average shift power. The opposite trend is observed when a small value of 'a' is used. The encoding process is shown in Figure 7.12.

Repeat-Friendly Huffman Code

Even though Huffman is a very effective code, further improvements can be achieved by exploiting certain ATE utilities, like the repeat command [167]. Using the repeat command, multiple successive identical logic values can be stored only once in the ATE-channel memory and they can be repeatedly transmitted over the ATE channel in successive cycles. Huffman code optimizes the length of the codewords, but the codewords are not repeat-friendly. For example, the codewords "10" and "11" of Figure 7.9c require 2 and 1 bits respectively when the repeat command is used (note that the first codeword has no identical successive logic values and thus the repeat command has no effect, whereas the second codeword has only one logic bit repeated two times and thus the repeat command eliminates the need to store the last bit of this codeword each time it is used). It is obvious that higher gain can be achieved by assigning repeat-friendly codewords to frequently occuring blocks.

As noted in Section 7.2.1, after the tree is constructed each leaf node is assigned a codeword by assigning the logic value '0' ('1') to each left (right)-child edge. In order to provide repeat-friendly codewords, we propose a very simple modification of the edge-assignment process. At the beginning, the edge to the left (right) child of the root is arbitrarily assigned the logic value '0' ('1'). Then we visit the rest of the nodes starting from these two nodes and moving towards the leafs. Every node is processed only when the edge connecting the node to its parent has been assigned a logic value. Let node A be one of these nodes. We find the child of A with the highest weight and we assign at the edge connecting node A with this child the same logic value that is assigned to the edge connecting node A with its own parent. The opposite logic value is assigned to the edge connecting node A to its other child with the smallest weight. This way, the more frequently occuring blocks are assigned more repeat-friendly codewords.



s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	
0	0	0	1	0	0	1	0	0	1	0	1	*CV*
0	0	0	0	0	0	0	x	x	x	x	x	
x	x	x	x	1	1	1	x	x	x	x	x	
x	x	x	x	x	x	x	0	0	0	x	x	
1	1	1	1	x	x	x	x	x	x	1	1	

Figure 7.14: Blocks replacement and candidates generated

Example 7.6. Figure 7.13a presents the Huffman tree of Figure 7.8c enhanced with more realistic block frequencies. The memory space required for storing the compressed test data using the repeat command (ignoring logic-bit repetitions between different codewords) is equal to $255 \times 1 + 210 \times 2 + 60 \times 1 = 735$ bits. The modified Huffman tree is shown in Figure 7.13b. Even though codewords' length are the same with Figure 7.13a, the leaf nodes are assigned different codewords. In Figure 7.13b the edge connecting nodes N_1 and N_3 is assigned the logic value '1' and the codeword for b_2 is '11' instead of '10' that was in Figure 7.13a Huffman tree. Memory space required in this case is $255 \times 1 + 210 \times 1 + 60 \times 2 = 585$ bits, which is considerably lower than that required by the encoding of Figure 7.13a. ■

7.2.4 Unmodeled Defect Coverage Improvement

The repetitive loading of identical test data into successive slices and the biased encoding of the blocks towards the most frequent blocks induces correlation, which adversely impacts the unmodeled defect coverage of the generated test vectors as shown in Chapter 5. Unmodeled defect coverage can be improved by decreasing the correlation between test vectors and also by adopting effective quality metrics to assess the test quality of each vector. The correlation of test vectors can be reduced by relaxing the tight objective of the encoding process to use the most frequent dictionary entries for encoding test data blocks. The use of a small minority of test data blocks repeatedly is responsible for biasing the generated test vectors towards similar patterns of specified bits. Test quality can be assessed using the output deviations metric of Chapter 5. Based on this evaluation metric, the encoding process selects the best test vectors in terms of unmodeled defect coverage.

The block substitution technique is a post-processing technique on the proposed encoding process, so it is applied after the step 12 of the encoding process shown in Figure 7.12. Candidate test vectors are generated by exploiting the potential of sparsely specified test data blocks to be encoded using multiple dictionary entries. Specifically, according to OSH, every test data block is encoded using the most frequent dictionary entry in order to be favored by the shortest possible codeword length. If we remove the requirement of encoding at the minimum cost, then sparsely specified blocks can be encoded using compatible less frequent dictionary entries.

Example 7.7. To understand how the blocks replacement technique works let us again consider the Huffman code of Figure 7.9. We also consider the CV-merged test cube presented in Figure 7.14a that needs to be encoded using this code. Based on the CV shown on the top of this test cube the blocks to be encoded are the three blocks corresponding to slices $s_1, s_3,\ s_6$ and s_9 shown in Figure 7.14a. The block 'xxx1' is compatible with entries b_1 and b_2 of the dictionary (see Figure 7.9) and thus both codewords '0' and '10' can be used to encode this block. In each case a different test vector will be generated which is compatible to the encoded test cube (note that these two encodings are actually two different ways of filling some of the unspecified values of the test cube). The two encodings are not equally effective in terms of test data volume because codeword '10' is more expensive than codeword '0'. The same can be done for the rest of the slices. In Figure 7.14b we present all possible compatibilities between the slices $s_1,\ s_3,\ s_6$ and s_9 of Figure 7.14a and dictionary entries b_1, b_2, b_3. Based on this table, we can generate all possible candidates which are equal to $2 \times 2 \times 3 \times 2 = 24$ and are shown in Figure 7.14c in ascending order of test data volume (bold entries correspond to the encoded blocks, non-bold to repeated blocks). ■

In most cases the encoding cost is expected to increase as we can only use less frequent dictionary entries for each block than those used by the original encoding. The highest overhead is imposed by the extreme case: when an encoded by a dictionary entry block is left un-encoded and it is preceded by the codeword corresponding to the un-encoded blocks (OSH provides this option [72]). Even though this is always an available option for any block (any block can be simply left un-encoded), it is very expensive and should be wisely and rather rarely used.

Block substitution changes the dictionary block frequencies and thus codeword lengths generated for the initial frequencies will not be any further optimal for the new frequencies. The additional overhead can be moderated if the codewords are re-generated to properly reflect the new frequencies of the dictionary entries. To this end, the tree is generated again for the new frequencies resulting after block substitution and a new codeword is assigned to each dictionary entry. In order to further reduce the additional overhead of this process, the proposed method bounds the volume of the blocks that are involved in this substitution process. This is achieved through the use of a pre-determined probability P, called hereafter as "probability of blocks change". For example, when $P = 10\%$ only a 10% of randomly selected blocks will be encoded by sub-optimal entries. Higher values of P increase the TDV cost but also the gain in unmodeled defect coverage.

7.2.5 Decompression Architecture

The proposed decompression architecture is shown in Figure 7.15. It consists of four main units: the Selective Huffman Decoder (SHD), the Signal Probabilities Generation (SPG) unit, the Control Vector Generation (CVG) unit and scan-registers $SR_1, SR_2, \ldots, SR_k$ which load the scan chains. It operates as follows: the SHD unit receives the compressed data from ATE and decodes the codewords. It loads one scan-register at a time with

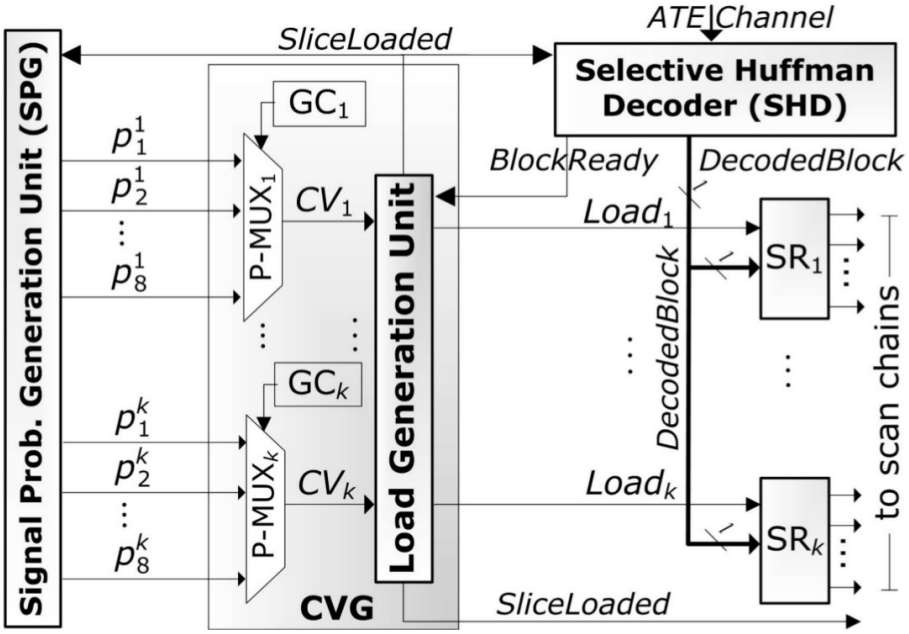


Figure 7.15: Proposed decompression architecture

the decoded block. This register is selected by the unit CVG which generates the control vectors $CV_1, \ldots, CV_k$. When CV_i is asserted, scan-register SR_i is loaded with the block decoded at the output of the SHD unit. Signals $CV_1, \ldots, CV_k$ are generated from pseudorandom signals produced at the SPG unit. When all registers SR_i (which have their CV_i signals asserted) are loaded with new test data, the slice is shifted into the scan chain and the decompression proceeds to the next scan slice (the rest of the registers hold their contents). Let us describe each unit in details.

Selective Huffman Decoder Unit (SHD). This unit loads serially the compressed test data from a single ATE channel and provides the decoded blocks of size l each. It consist of a finite state machine (FSM) which decodes the codewords and a small dictionary which stores the distinct block encoded by each codeword. It generates two signals namely *DecodedBlock* and *BlockReady*. It asserts signal *BlockReady* when a new block is available at the *DecodedBlock* output.

Scan Registers (SR). This unit consists of k, $l-$bit registers, $SR_1, \ldots, SR_k$ which correspond to groups $G_1, \ldots, G_k$ respectively. Each scan register SR_i is controlled by signal $Load_i$ which is asserted whenever CV_i is asserted and the respective test data are available at the *DecodedBlock* output. When $Load_i$ is not asserted then the register holds its contents.

Signal Probabilities Generation Unit (SPG). The signal probabilities generation unit is shown in Figure 7.16. It consists of a small LFSR which is initially loaded with a random seed, and a very small combinational logic which generates pseudorandom signals of various probabilities in the range of [0, 1]. The operation of this unit is very simple: each LFSR output has probability of 50% to receive logic value '1'. A 2-input AND gate driven by two signals with probability 50% provides at its output a signal with probability $P_{out} =$

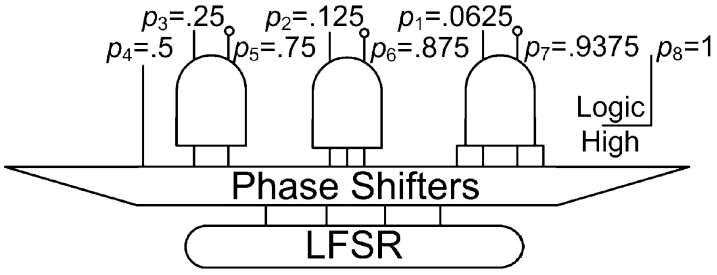


Figure 7.16: Signal probabilities generation unit

25%. By using various combinational gates with various numbers of inputs, signals with many discrete probabilities can be generated. In the proposed scheme the 8 probabilities shown in Figure 7.16 are implemented by just few 2-4 input gates. Since an independent control vector has to be generated for each group, a different group of signals $p_1, \ldots, p_8$ is needed for each control vector ($p_1^1, p_2^1, \ldots p_8^1$ are used for CV_1, $p_1^2, p_2^2, \ldots, p_8^2$ for CV_2, etc). In order to eliminate correlations between signals $CV_1, CV_2, \ldots, CV_k$ phase shifters are inserted between the LFSR and the combinational logic. The SPG unit is controlled by the CVG unit with signal *SliceLoaded*. This signal is asserted whenever the scan chains are loaded with the current slice and it enables the LFSR to move to its next state and to generate signals $p_1^j, p_2^j, \ldots, p_8^j$ ($j = 1 \ldots k$) for the next slice.

Control Vector Generation Unit (CVG). This unit is responsible for both vectors $CV_1, CV_2, ..., CV_k$ and signals $Load_1, Load_2, \ldots, Load_k$. For each CV_j signal one multiplexer 8-to-1, namely P-MUX$_j$, is used to select one of the $p_1^j, p_2^j, \ldots, p_8^j$ signals generated by SPG unit. p_1^j is connected to the first input of P-MUX$_j$, p_2^j is connected to the second input etc. The selection inputs of P-MUX$_j$ are connected to counter GC$_j$. This counter is initialized before decompression process begins with the value corresponding to the initial P_{CV_j} calculated at the beginning of the encoding process. Each time P_{CV_j} has to be increased to the next higher discrete value counter GC$_j$ is triggered once to count up. This way it selects the next input of P-MUX$_j$ which is already connected to the pseudorandom signal with the next higher probability. When *BlockReady* is asserted CVG unit loads the decoded block to the first scan register that has its respective CV signal asserted and waits until the next test data block is decoded. When all scan registers with logic value '1' at their control vectors for the current slice are loaded then current slice is shifted into the scan chains and decoding continues to the next slice.

All GC counters are simultaneously triggered during the decoding process whenever P_{CV} values are increased. This is done at most 7 times as the minimum possible initial value for P_{CV} is equal to p_1 and it can be potentially increased up to p_8. Each triggering has to be done at a specific vector in the test sequence which is determined during the encoding (see Figure 7.12). The whole vector sequence is controlled by the means of a vector-counter (not shown in Figure 7.15 for simplicity). We use 7 registers (also not shown in Figure 7.15) which are loaded before the decompression process begins with the specific vector-counter values that should trigger each time the GC counters. Thus, each

time the vector counter reaches the value stored in any of these registers all the the GC counters are triggered once.

The rate at which the blocks are decoded at the output of the SHD unit depends on the length of the codewords (long codewords need more cycles to be decoded). As a result, there might be cases that the loading of the scan registers has to wait for the next block to be decoded and vice versa. The first case is handled by the CVG unit which controls the loading of the shift registers and stalls both this loading and the scan-in operation when it is necessary. For the second case there are two solutions: firstly a FIFO can be used at the output of the SHD unit to hold all blocks which are decoded early (usually a very small FIFO suffices to store all such blocks). The second solution is to let the SHD hold the last decoded block and ignore any additional test data sent by the ATE. If the repeat command is available then it can be used to eliminate these data by repeating the last useful bit for as many cycles as needed without incurring additional overhead. When the ATE-repeat command is not available both techniques can be used at the same time to offer a trade-off between hardware overhead and test data storage. As a result, no handshaking is required between the ATE and the decompressor.

The SHD unit of the proposed architecture is test set dependent. SHD unit can be designed in a test-set-independent way if a) the dictionary is implemented using a small RAM that is loaded from the ATE before the decompression process begins and b) the FSM is designed to be generic (similar to [94, 131]) - provided of course that it decodes a pre-determined number of codewords (the exact codewords can be determined at a later step of the design process). As it has been shown in [60], [72], a very low number of blocks, 8-16, suffices to provide high compression efficiency. Even in the case that the SHD is designed to be test set dependent, last minute design changes are neither expected to affect the Huffman decoder nor the dictionary entries (the same decompressor can be still used even at the cost of a marginal reduction of the compression achieved). Only in the case of extensive design modifications (that cause also extensive changes on the test sets of the SoC's cores) the SHD must be re-designed to reflect these changes.

Multiple cores residing in the same SoC require in many cases decompressors tuned to different parameter values. This requires developing a dedicated decompressor for each core which is an expensive approach. In order to tackle this issue we propose the development of a low-cost reconfigurable decompressor which can adjust its characteristics to the requirements of multiple cores at the expense of a slight increase in test data volume. This decompressor can be shared among multiple cores for decreasing hardware cost, without sacrificing compression. Specifically, we assume a single decompression unit for muliple cores which uses a common FSM for the cores but a separate dictionary for each one (note that the hardware implementation of the dictionary can be also common if it is implemented as a RAM that is loaded with the particular contents of each core before it is tested). Using this technique, the same codewords are used for the cores that share the decompressor, but each codeword corresponds to a different entry that is found at a separate dictionary for each core. In that way, the FSM, which would be the major

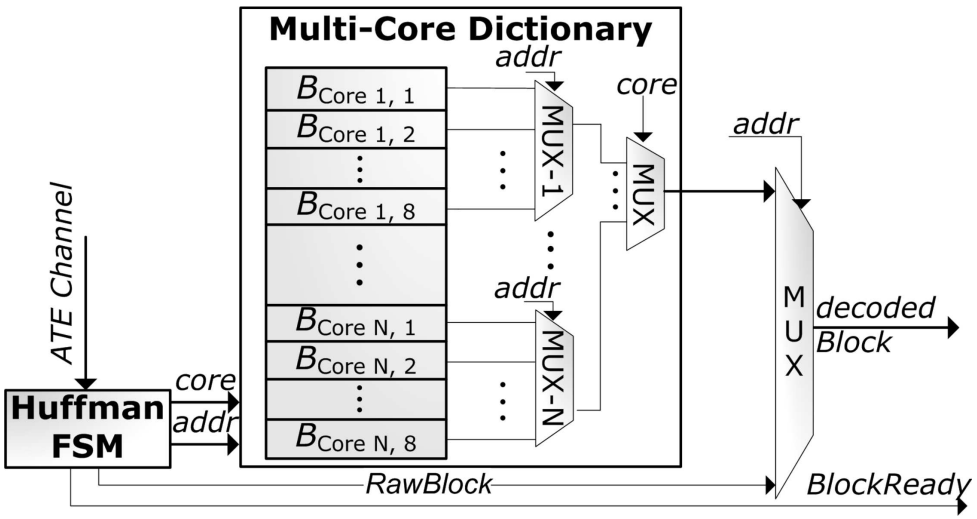


Figure 7.17: Selective Huffman Decoder

contributor to the overhead of the decompressor (if not shared), is shared among multiple cores while at the same time the dictionaries which occupy very limited space and offer great compression benefits are dedicated and optimized to the characteristics of each core.

The multicore decompressor is presented in Figure 7.17. To account for different scan chain configurations among different cores we use the same block size l for those cores that share the decompressor, and we equip the decompressor with the maximum number of SRs used by any of the cores (we remind that each core requires a number of SRs that is equal to the number of scan chains divided by the block size). While testing each core only the necessary SRs are activated.

The use of common codewords for a group of cores requires proper adjustments of the encoding process. In particular, the repeat-friendly encoding cannot be applied for any core until the blocks corresponding to the dictionary entries of every core are generated. The aggregate number of occurrences of these blocks are used to generate common Huffman codewords. Since in each core the most frequently met block will be encoded using the shortest codeword, the aggregation of the frequencies is done in such a way as to bias the frequencies of the common codewords. In particular, the number of occurencies of the most frequent blocks are summed to provide the frequency of the most frequent codeword of the group, and these blocks are stored in the first entry of each dictionary. Then, the same process is applied to the second most frequent block of each core, etc.

Example 7.8. Let us assume an SoC consisting of two cores with 4 dictionary entries and number of occurrencies (in descending order) $f_1^1 = 30$, $f_2^1 = 22$, $f_3^1 = 20$, $f_4^1 = 18$ for the first core and $f_1^2 = 20$, $f_2^2 = 15$, $f_3^2 = 10$, $f_4^2 = 9$, for the second core. The aggregate frequencies for both cores become $f_1^{1+2} = 50$, $f_2^{1+2} = 37$, $f_3^{1+2} = 30$, $f_4^{1+2} = 27$. Based on the aggregate frequencies a Huffman tree is constructed and it is optimized with the repeat friendly optimization method presented in Section 7.2.3. The resulting codewords are used to build a common FSM for the cores. ■

By properly grouping cores and allocating one decompressor at each group, the area cost can be retained low. In addition, cores that share a common decompressor should be located close in the floorplan to minimize the routing overhead. Finally, the ability of the proposed decompressors to work well with both IP and non-IP cores enables the sharing of decompressors among both of these types of cores and offers a further degree of freedom during the grouping process and the allocation of decompressors to each group. Note that scheduling techniques can be applied to further decrease the test application time but they are out of the scope of this work.

7.2.6 Experimental Results

We implemented the proposed encoding scheme using the C++ programming language and we synthesized the decompression logic using commercial tools. The proposed method was applied on the largest ISCAS'89 and a subset of the large IWLS benchmark circuits [1]. We used a commercial tool to generate test sets for 100% stuck-at fault coverage. Unless otherwise stated, the parameters used for the proposed method were set equal to $m = 8$ codewords and $l = 8$ bits block size. The running time for the largest circuit, the *Ethernet*, was less than 3 minutes. Note, that this execution time is much shorter compared to the execution time of dynamic LFSR encoding, which can reach the order of hours without partitioning techniques to speed up the process (see Section 2.2.2). Table 7.2 contains information on the benchmarks and in particular the size in combinational gates, the pseudorandom primary input/output pins count (PPI/PPO), the scan cells count and scan structure represented as number of scan chains (n) multiplied by the length of scan chains (r).

We implemented the state of the art low power dynamic reseeding proposed in [105] (LPDR) using ring generators with sizes in the range $[40, 150]$. In this case we used a single shadow register to favor the Test Data Volume (TDV) measurements of this method. The shadow register was implemented using both techniques proposed in [29], [105] (internal XOR tap or one additional ATE channel) and the best result is reported in every case. In the case of LPDR, the repeat command was utilized to further reduce the compressed test data. Moreover, as suggested in [105, 123] the unsolved variables were filled in a repeat-friendly way to improve further the test data compression of LPDR. Even though both the ATPG and fault simulation steps can be straightforwardly embedded in both LPDR method and the proposed encoding, we omitted these steps as they cannot be applied in the case of IP cores (their internal structure is unknown). We also compare our method to various code-based methods [72], [74], [92] and [125]. For all methods we use the minimum number of ATE channels, that is one channel for the proposed method and code-based methods, and two channels for LPDR (a very small number of channels is highly desirable in a multisite test environment).

For evaluating the unmodeled defect coverage we used a surrogate fault model, i.e. a fault model that is not targeted by the generated test sets. That fault model is the transition delay fault model (TDF). For detecting transition faults each test vector gen-

Table 7.2: Benchmarks Information

circuit	gates	PPI/PPO	scan cells	$n \times r$
s5378	4285	86	214	16×14
s9234	4190	77	247	16×16
s13207	10 103	154	700	32×22
s15850	11 919	103	611	32×20
s38417	30 460	136	1664	64×26
s38584	26 864	292	1464	48×31
ac97_ctrl	28 554	132	2253	32×71
mem_ctrl	11 440	267	1194	48×25
pci_bridge	45 055	369	3517	128×28
tv80	14 223	46	372	32×12
usb_funct	27 081	249	1858	32×59
ethernet	157 520	211	10 647	128×84

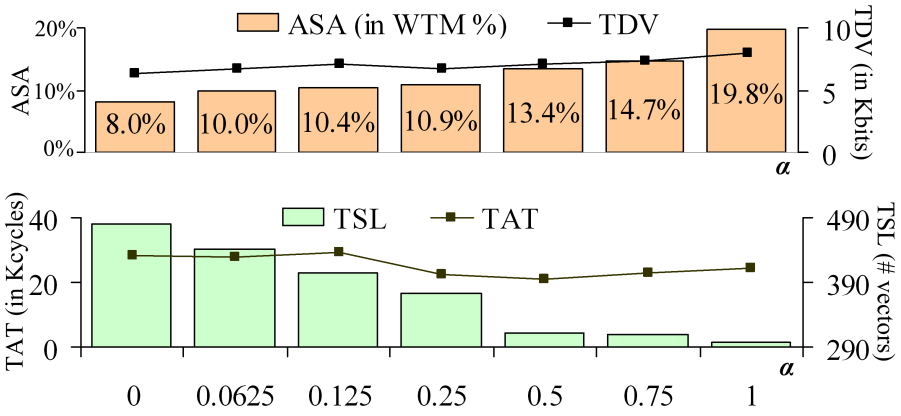


Figure 7.18: Tradeoffs for a value for s13207

erated by the decompressors is applied on the circuit using two capture cycles according to Launch-On-Capture (LOC) technique. Note that similar approaches were adopted in many techniques (e.g., [8], [178], [180]). The measurements of average switching activity (ASA) were done using the normalized weighted transitions metric (WTM) [29].

Impact of parameters a and l.

In Figure 7.18 we present the Test Data Volume (TDV), Average Switching Activity (ASA), Test Sequence Length (TSL) and Test Application Time (TAT) results of the proposed method for the s13207 benchmark circuit for various values of the parameter 'a' ('a' is used to generate the starting signal probabilities of the CVs as shown in Section 7.2.3). It consists of two parts which are aligned on a common X-axis (shown at the bottom) that presents the selected values for parameter 'a'. The top part presents the TDV curve (1Kbit = 1000bits) and the ASA measures (bars) of the proposed method,

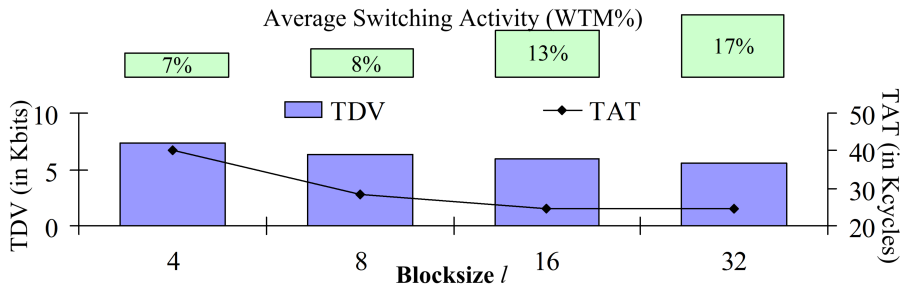


Figure 7.19: TDV, TAT and ASA for various blocksize l values on s13207

while the bottom bar presents the TAT (line) and the TSL measures (bars). The smaller is the value of parameter 'a', the more sparse are the generated CVs in terms of their '1' logic values and thus the more power efficient are the generated test vectors ('0' logic values in the CVs load the scan chains with the same test data and thus reduce the shift power dissipation). Higher values of parameter 'a' cause the test vectors to become less power efficient as they increase the number of update operations on the shadow register. At the same time, as 'a' increases the encoding process is less constrained by the shift power objectives and thus more test cubes are encoded into each test vector during the pre-merging process. As a result TSL drops considerably but the ASA increases.

An interesting property of the proposed method is that TAT is not affected by the value of 'a' as much as TSL is affected. TSL depends on the number of vectors applied to the core, while TAT depends mostly on the codeword decoding process and thus on the TDV which does not depend much on the value of a. Note that the loading of the scan chains is done in parallel with the decoding of incoming data from the ATE. When the value of a is high many update operations occur in a short time and the Load Generation Unit has to stall in these cases for new test data to be decoded through the Selective Huffman Decoder. Even though the TSL is low (and consequently the overall number of test slices loaded into the scan chains is low), when the value of 'a' is high, there is a high number of update operations that render the SHD unit the bottleneck of the test generation process. As the value of 'a' drops, the overall number of test slices loaded into the scan chains increases (due to the increase of TSL) but most of the additional test slices are repeated versions of their previous ones and are generated without incurring any additional decoding cost (no test data need to be decoded for those slices). As a result, the Load Generation Unit stalls less frequently and the decoding process is very well parallelized with loading the scan chains using mostly repeated test data. Therefore, we conclude that a low value of 'a' is more preferable from both TDV and power perspectives, while the additional test vectors applied due to the increased TSL in that case, can be exploited to increase the unmodeled defect coverage with a very small impact on TAT.

Another important parameter that affects both the ASA and the TAT of the proposed method is the block size. The larger the block size is, the smaller is the number of scan chain groups and thus the lower is the number of blocks that need to be decoded for every scan slice. However, as the number of scan chain groups decreases, the benefits on

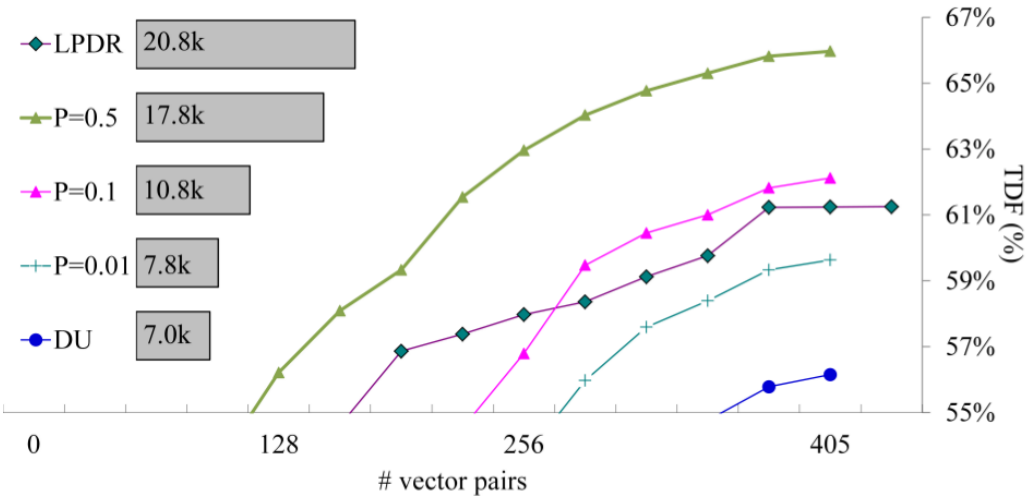


Figure 7.20: Tradeoffs for blocks substitution probability P on s13207

reducing shift power reduces (the probability that a block can be repeatedly loaded into the scan chains drops as the size of the block increases). Figure 7.19 presents results for TDV, TAT and ASA for various blocksize values on s13207 benchmark circuit. As block size l increases, both TDV and TAT drop while ASA increases. Beyond a certain block size (that is equal to 32 in the case at hand) TAT increases again while TDV saturates. Therefore, depending on the power budget of the design, an optimal block size exists for every circuit which can be easily found due to proposed method's short CPU time.

Impact of parameter P.

The impact of parameter P effect is shown in Figure 7.20 where the lines correspond to TDF and the bars to TDV results. We present results for the proposed defect-aware encoding for '$a = 0.125$' and various values of P labeled as $P = 0.01$, $P = 0.1$, $P = 0.5$ and the proposed defect-unaware encoding labeled as 'DU'. The LPDR's method results are labeled as 'LPDR'. The x-axis presents the number of vector pairs applied using the LOC scheme for all the methods and the y-axis presents the TDF measures for each technique. Although the proposed defect unaware technique is superior in terms of TDV as compared to the LPDR method (7.0 Kbits over 20.8 Kbits), it is inferior to LPDR in terms of coverage on the surrogate transitions-delay faults. This is the effect of the increased correlation of the test slices that results from the biasing of the encoding process towards a small number of frequently occurring test data blocks. However, the defect aware proposed scheme improves considerably the TDF coverage. Even for very small values of P (i.e. the case labeled as $P = 0.01$) which correspond to the case that only a very small percentage of blocks are substituted for increasing test quality, TDF becomes 59.8% and almost reaches that of LPDR, with only a slight increase of the test data (they become 7.8Kbits from 7.0Kbits). If we further increase P to the value of $P = 0.1$ and $P = 0.5$ the TDF of the proposed technique reaches higher values than that of LPDR.

Repeat-Friendly Huffman Code: TDV Improvement

We run 10 different experiments for each of the 11 benchmark circuits for the proposed method by varrying the blocks substitution probability P from the set of values $[0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.25, 0.5, 1]$. In each case, the TDV improvement TDV_{impr} offered by using the repeat-friendly version (RF) of the proposed method over the original not optimized encoding (Orig) was calculated by using the formula $TDV_{Impr} = (TDV_{Orig} - TDV_{RF})/TDV_{Orig}$ (we note that in both "RF" and "Orig" cases the compressed test data were further encoded using the ATE repeat command). In the 100 out of the 110 cases, the TDV improved using "RF" and the improvement was as high as 17.5%. The average and median improvement was 5% and 6.18% respectively. In the rest 5 cases the TDV slightly increased and the increment was in the range [0% - 2.3%]. We note that the repeat friendly encoding optimizes the internal characteristics of codewords (intra-codeword) but it does not consider the sequence of codewords (inter-codeword) which also affect the TDV. This may cause a slight reduction of the TDV benefits and in those rare cases that the gains from intra-codeword improvements are not high, it slightly increases the overall TDV.

Comparisons

In Table 7.3 the TDV, TSL and ASA comparisons of the proposed method (labeled as "Prop.") against LPDR and OSH [72] are presented. For this comparison we used uncompacted test sets because LPDR is very efficient with uncompacted test sets. In all cases the proposed method offers the lowest TDV among all methods. The improvement ranges between 1.5x and 4.3x as compared to LPDR method and between 2.6x and 8.8x as compared to the OSH method. The TSL of the proposed method is lower than that of the LPDR method and higher than that of the OSH. We note that the TSL greatly depends on the static and/or dynamic compaction policy followed during the encoding (in our case this is the pre-merge process and, as noted in Section 7.2.3, it belongs to the static compaction processes). Static and/or dynamic compaction in linear encoding methods is generally constrained by the size of the linear decompressor, the number of variables injected into the decompressor and the power constraints. Specifically, the number of free variables available during the encoding of every test cube must be higher than the number of specified bits of the test cube. Since every additional encoded test cube consumes variables for its encoding, it is rather unlike that all compatible test cubes can be encoded together at the same test vector using linear encoding, unless a large number of variables are injected at each clock cycle. On the contrary, code-based decompressors (like for example the Huffman decompressor in the OSH case) do not suffer from this restriction and thus permit the application of very aggressive static compaction processes before the compression which reduce the TSL a lot. Even though this is also the case for the proposed method, the power objectives implemented through the use of CVs introduce additional constraints into the static compaction process (pre-merging) that do not permit the TSL to drop as much as it does in the OSH method. However, due to the

Table 7.3: Comparisons TDV, TSL and ASA

circuit	TDV (in Kbits)			TSL (# of vectors)			ASA (WTM %)		
	LPDR	OSH	Prop.	LPDR	OSH	Prop	LPDR	OSH	Prop.
s5378	10.0	15.7	5.1	305	155	277	5.8	21.3	14.2
s9234	20.9	29.9	11.7	504	259	438	11.6	21.0	18
s13207	20.8	39.5	10.5	419	255	405	5.4	19.9	12.4
s15850	26.8	43.6	11.2	552	243	523	7.0	24.7	12.9
s38417	97.5	150.9	49.5	1548	267	1094	6.2	31.2	13.5
s38584	89.7	112.3	33.9	1179	245	591	7.0	32.4	13.2
ac97_ctrl	57.5	60.8	15.5	2161	77	206	1.3	34.4	11.1
mem_ctrl	115.5	205.7	37.7	2466	884	1581	5.0	14.7	7.4
pci_bridge	233.0	418.3	120.6	3435	654	1062	20.6	21.8	20.4
tv80	72.5	131.8	48.1	2330	1662	2115	10.8	35.8	19.0
usb_funct	98.2	156.6	49.9	2340	231	763	1.1	35.2	15.5
ethernet	612.4	1257.0	143.1	3115	1100	1811	2.7	10.4	15.4

inherent property of the proposed method to encode the vast majority of test slices using repeating test data, the proposed method offers much lower TAT than the OSH approach which counterbalances the increase in TSL.

As far as the ASA measures are concerned, the LPDR gives the lowest WTM values. In most of the cases this is also related to the long TSL of this technique, that is even 10 times higher in one case than that of the proposed method, and that permits higher repetition of the test data loaded into the scan chains. However, the WTM values of the proposed method are very low and much lower than those of OSH technique. Note that, as it was shown in Figure 7.18, ASA can be considerably reduced by using lower values of α parameter, which constitute a favorable selection for the proposed method.

In Table 7.4 we present the unmodeled defect coverage comparisons between the proposed method and LPDR. We consider three different instances of the proposed method: (a) the defect unaware (DU) instance where the test quality improvement technique was not applied, (b) the defect aware encoding with the values of parameter P selected from the range $[0.1, 0.25]$ denoted as medium effort (ME) encoding, and (c) the defect aware encoding with the values of parameter P selected from the range $[0.5, 1]$ denoted as high effort (HE) encoding. We have to note that the unmodeled defect coverage depends a lot on the number of test vectors applied. Thus, for providing a fair comparison, we applied restrictions on the pre-merging phase of the proposed method to increase its TSL (note that the proposed method offers considerably lower TSL than LPDR). In the first two columns the TSL results are presented. The next four columns present the TDV comparisons. Note that the TDV of the proposed method is lower than that of LPDR for all values of P in almost all cases (the best results are bolded). The last four columns present the TDF comparisons (the highest TDF entries are bolded). Note, that "DU" has relatively low TDF which however is considerably improved by using the proposed test

Table 7.4: Comparisons TDF

circuit	TSL (#of vectors)		TDV (in Kbits)				TDF (%)			
			LPDR	Proposed			LPDR	Proposed		
	LPDR	Prop.		DU	ME	He		DU	ME	HE
s5378	305	277	10.0	**4.3**	4.9	5.1	62.0	60.2	62.4	**63.4**
s9234	504	438	20.9	**10.1**	11.2	11.7	49.8	45.6	49.1	**49.9**
s13207	419	405	20.8	**7.0**	13.7	17.8	61.3	56.2	63.5	**66.2**
s15850	552	523	26.8	**10.5**	11.2	13.7	54.7	54.3	56.4	**57.2**
s38417	1548	1629	97.5	**53.0**	64.5	77.2	87.8	83.2	86.3	**87.9**
s38584	1179	1130	89.7	**44.1**	44.1	45.2	67.0	65.4	68.2	**68.4**
ac97_ctrl	2161	2136	57.5	**31.1**	36.7	36.4	53.4	51.3	56.3	**58.8**
mem_ctrl	2469	2357	115.5	**32.3**	74.4	81.3	**43.7**	33.1	36.8	39.3
pci_bridge	3435	2774	233.0	**105.3**	169.7	230.7	81.9	65.5	81.0	**83.2**
tv80	2330	2426	72.5	**45.2**	54.7	74.9	59.9	55.2	60.4	**62.3**
usb_funct	2340	2379	98.2	**52.6**	75.8	94.9	72.7	61.8	71.5	**72.9**
ethernet	3115	1811	612.4	**143.1**	182.9	252.6	55.3	49.1	57.7	**64.4**

Table 7.5: Comparisons with Test Data Compression Techniques (in Kbits)

circuit	[74]	[92]	[125]	[111]	Proposed
s9234	12.8	30	-	20.6	**10.2**
s13207	14,6	21	74	28.9	**6.3**
s15850	16.6	25	26	25.1	**10.5**
s38417	58.7	85	45	59.0	**44.2**
s38584	55.4	57.1	74	74.9	**24**

quality improvement process. In all but one case, when the effort for increasing defect coverage is set to high, the proposed method offers the highest TDF, and still retains lower TDV than LPDR. In the vast majority of the cases, the proposed method offers higher TDF than LPDR even when the effort is set to medium. So, we conclude that the proposed method offers much lower TDV than LPDR and at the same time it offers a trade-off between the TDV improvements and the unmodeled defect coverage, yielding higher unmodeled defect coverage than LPDR for higher values of P.

In Table 7.5 we compare the proposed method against some of the best TDC techinques in the literature in terms of TDV. Methods [74], [92] and [125] focus on test time and TDV optimization and method [111] is a TDC method that targets average power reduction too. In the case of the proposed method we set the value of parameters $l = 8$, $P = 0$ and $a = 0.0625$. For this comparison we used compacted test sets because these methods perform better with them. It is obvious that the proposed method offers the lowest TDV.

In Table 7.6 we present the area overhead of the decompressors for LPDR, OSH and the proposed method for various scan chain volumes n, ring generator sizes d and number of groups k. The hardware overhead is measured in terms of gate equivalents, where

Table 7.6: Hardware Overhead Comparisons

n	Dynamic Reseeding HO					OSH	Proposed HO				
	d=48	d=64	d=96	d=128	d=150*		k=1	k=2	k=4	k=8	k=16
32	626	774	1072	1370	1574	344	560	622	746	994	1490
64	805	954	1251	1549	1754	417	633	695	819	1067	1563
128	1164	1313	1610	1908	2112	563	779	841	965	1213	1709

*150 is the size of ring generator used for *ethernet* in table 7.3

one gate equivalent corresponds to the area of a 2-input NAND gate. The area overhead strongly depends on the values of the parameters used and it is relatively low in all methods. In general the hardware overhead of the proposed method is larger than OSH but lower than dynamic reseeding.

We have to note that the hardware overhead of the proposed decompressors does not depend on the size of the core under test, but on the dictionary size, the number of codewords, the block size and the number of scan chains n and scan chain groups k. As it was shown in previous studies ([60, 72]) a relatively small block size in the range $[8-16]$ and a small number of codewords and dictionary entries in the range $[8-32]$ suffice for high compression. In addition, the value of k is decided by the test engineer and it can be always in the range $[1-16]$ (the value of $k=16$ is already very high). In addition, the overhead increases linearly with k, n. Therefore, the hardware overhead of the decompressors is not expected to increase for even larger circuits than those reported in Table 7.6. In addition, as it is shown in Tables 7.3, 7.4 the largest circuit, the Ethernet, which is almost one order of magnitude larger than most of the rest circuits, gives the best results. Therefore the proposed encoding method is expected to scale very well even for larger circuits.

IP-Cores and Multi-Core Experiments

In order to show the effectiveness of the proposed method for pre-computed test sets of IP cores, we applied it on a pre-compacted test set of the largest circuit, the Ethernet. Note that in the case of IP-cores, pre-computed and most likely pre-compacted test sets are provided to the test engineer of the SoC. The size of the compacted test set was 11.6 Mbits (1 Mbit = 10^6 bits) and after applying the proposed method the TDV dropped by a factor higher than 50x and reached the value of 221.7 Kbits. The number of specified bits of the initially generated (and highly compacted) test set is 263.8 Kbits which clearly show that the proposed method succeeded to reduce the test data volume below this number which is a lower bound for most of the compression techniques. The TSL was 1100 and the WTM value was 9.2% which are both very low. The TDV of the OSH was found to be more than 5 times higher than that of the proposed method, and specifically it is equal to 1246.7K. We note that LPDR technique is not applicable in this case as the large variation of the specified bits in the test set requires the use of unrealistically large (in the range of thousand of cells) ring generators. It is also worth noting that despite

the fact that this is a very compacted test set and thus the proposed pre-merging process has no effect, the compressed TDV of the proposed method is very close to that shown in Table 7.3 which was computed using non-compacted test sets that offer higher degrees of freedom in the encoding process. So, we conclude that the proposed method is very efficient for both IP and non-IP cores.

In the last experiment we study the performance of the proposed method in a multi-core SoC. To this end we synthesized a hypothetical SoC consisting of all the cores presented in Table 7.3. For simplicity we assume that all cores are tested in a non-overlapping manner while concurrency can be straightforwardly applied if multiple decompressors are available. In order to show the effectiveness of the proposed technique even in the extreme case that only one decompression unit is available for the entire SoC, we used a single decompression unit for all the cores and we kept the FSM of the decompression architecture common in all cases i.e. the same codewords were used for all cores (note that any test scenario with multiple decompressors will give an even better solution in terms of compression and test application time). For each of the cores we assumed a different dictionary which was optimized to the particular characteristics of the core. The overall TDV is equal to 529.9Kbits. The overhead in that case was found to be less than 0.5% of the overhead of the SoC. In the case that a different decompressor is used for every core (optimizing thus the FSM and TDV for each particular core separately) the hardware overhead increases to the 3% of the SoC. Therefore, the shared decompressor offers very high TDV benefits at a very small area cost, and thus offers a very compelling solution for testing multicore SoCs.

7.3 Conclusions

In this Chapter a new decompression scheme and a novel encoding method which can be combined with various decompressors to offer low shift power, high unmodeled defect coverage and high compression was proposed. Extensive experiments showed that when the proposed method is combined with state-of-the-art linear and statistical code-based decompressors, both compression and unmodeled defect coverage improve while shift power is retained at very low levels. Especially when it is combined with statistical code-basd decompresors then the results is a TRP method with low pin-count interface of multi-core SoC that is very effective for cores of both known and unknown structure as it offers the combined advantages of symbol-based and linear-based techniques. In addition, the proposed scheme offers an effective low-cost solution (in terms of area overhead) for testing multi-core SoCs. Therefore, we conclude that the proposed method can serve as an attractive alternative to the widely adopted solution of linear-based encoding.

Chapter 8

Conclusions

The wide spreading of Very Deep Sub-Micron (VDSM) Integrated Circuits' (ICs), the architectural advancements that made possible the construction of Multi-core Systems-on-Chips (MCSoCs), and the power dissipation limitations imposed by the post-Dennard era created an explosive mixture for the upcoming manufacturing testing technologies. Failure in sustaining manufacturing testing cost low can make these advancements collapse.

This dissertation has identified and targeted a number of factors that affect test cost: test data volume, test application time, power consumption during testing and defect coverage. Viable solutions that can reduce test cost by targeting these factors have been proposed in the areas of algorithmic post ATPG X-filling algorithms and Test Resource Partitioning architectures as well as compression algorithms. The efficiency of the proposed solutions has been demonstrated with extensive experiments using academic benchmark circuit-suites. The contributions are summarized as follows:

1) A novel architecture was proposed to decrease both the test application time and the ATE memory requirements of Test Set Embedding (TSE) techniques. Two new types of Linear Feedback Shift Registers, the Single-State-Skip and the Variable-State-Skip LFSRs were presented. Single-State-Skip LFSRs perform successive jumps of constant length in their state sequence, while Variable-State-Skip LFSRs embed multiple State-Skip circuits and thus they are able to perform jumps of variable length in the LFSR state sequence. By using Single-State-Skip LFSRs for testing single or multiple identical cores and Variable-State-Skip LFSRs for testing multiple non-identical cores we get the well-known high compression efficiency of test set embedding with substantially reduced test sequences. The length of the shortened test sequences approaches that of test data compression methods, thus bridging the gap between test data compression and test set embedding methods.
2) An architecture and a compression method were proposed to reduce the average switching activity during scan testing for linear-based decompressors. In particular, a new linear encoding method which offers both high compression and low

shift power dissipation at the same time was presented. A new low-cost, test-set-independent scheme was also proposed which can be combined with any linear decompressor for reducing the shift power during testing. Extensive experiments show that the proposed method offers reduced test power dissipation, test sequence length and test data volume at the same time, with very small area requirements.

3) A unified X-filling technique was proposed to reduce the average power requirements of tests under peak power constraints and also to increase the unmodeled defect coverage of the generated tests. The proposed method reduces shift power under constraints on the peak power during response capture, and the power reduction is comparable to that for the Fill-Adjacent X-filling method. At the same time, this approach provides high defect coverage, which approaches and in many cases is higher than that for random-fill, without increasing the pattern count.
4) LFSR reseeding approaches were proposed to increase the defect coverage of the generated vectors. The proposed techniques are based on a new "output deviations" metric for grading stuck-at patterns derived from LFSR seeds. They include a window-based static reseeding method as well as a dynamic reseeding method. It was shown that, compared to standard compression-driven LFSR reseeding and a previously proposed output-deviation-based method, higher defect coverage is obtained using stuck-at test cubes without any loss of compression.
5) A Test Resource Partitioning architecture was proposed that reduces power dissipation during testing and also increases the unmodeled defect coverage of the generated patterns. The scheme can be combined with existing linear-based and code-based decompressors to increase their unmodeled defect coverage and almost totally eliminate control data for low power testing. The application of the scheme on Optimal Selective Huffman provides the advantages of both symbol-based and linear-based techniques and offers a very attractive unified solution that removes the barriers of existing test data compression techniques. In addition, it favors multi-site testing as it requires a very low pin-count interface to the Automatic Test Equipment. Finally, contrary to existing techniques, it provides an integrated solution for testing MCSoCs, as it is suitable for cores of both known and unknown (IP) structure that usually co-exist in MCSoCs.

Bibliography

[1] IWLS'05 circts., online: http://www.iwls.org/iwls2005/benchmarks.html.

[2] "Test and test equipment," in *Internation Technology Roadmap of Semiconductors*, 2007.

[3] "Yield enhancement," in *Internation Technology Roadmap of Semiconductors*, 2011.

[4] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. IEEE Press, Piscataway, NJ, 1994.

[5] N. Ahmed, M. Tehranipour, and M. Nourani, "Low power pattern generation for bist architecture," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, vol. 2, 2004, pp. II–689–92 Vol.2.

[6] R. Aitken, "Nanometer technology effects on fault models for ic testing," *IEEE Computer*, vol. 32, no. 11, pp. 46–51, 1999.

[7] K. J. Balakrishnan and N. A. Touba, "Improving linear test data compression," *IEEE Trans. Comput.-Aided Des.*, vol. 14, no. 11, pp. 1227–1237, 2006.

[8] S. Balatsouka, V. Tenentes, X. Kavousianos, and K. Chakrabarty, "Defect aware x-filling for low-power scan testing," in *Proc. DATE*, 2010, pp. 873–878.

[9] P. Bardell, "Design considerations for parallel pseudorandom pattern generators," *Journal of Electronic Testing*, vol. 1, no. 1, pp. 73–87, 1990. [Online]. Available: http://dx.doi.org/10.1007/BF00134016

[10] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann, "Opmisr: the foundation for compressed atpg vectors," in *Test Conference, 2001. Proceedings. International*, 2001, pp. 748–757.

[11] K. Basu and P. Mishra, "Test data compression using efficient bitmask and dictionary selection methods," *IEEE Trans. Very Large Scale Integr.*, vol. 18, no. 9, pp. 1277–1286, Sep. 2010.

[12] I. Bayraktaroglu and A. Orailoglu, "Concurrent application of compaction and compression for test time and data volume reduction in scan designs," *Computers, IEEE Transactions on*, vol. 52, no. 11, pp. 1480–1489, 2003.

[13] ——, "Test volume and application time reduction through scan chain concealment," in *Proc. DAC*, 2001, pp. 151–155.

[14] J. Bedsole, R. Raina, A. Crouch, and M. S. Abadir, "Very low cost testers: Opportunities and challenges," *IEEE Des. Test*, vol. 18, no. 5, pp. 60–69, Sep. 2001.

[15] S. Bose, H. Grimes, and V. Agrawal, "Delay fault simulation with bounded gate delay mode," in *Test Conference, 2007. ITC 2007. IEEE International*, 2007.

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

[17] K. Butler, J. Saxena, A. Jain, T. Fryars, J. Lewis, and G. Hetherington, "Minimizing power consumption in scan testing: pattern generation and dft techniques," in *Test Conference, 2004. Proceedings. ITC 2004. International*, 2004, pp. 355–364.

[18] A. Chandra and K. Chakrabarty, "Low-power scan testing and test data compression for system-on-a-chip," *IEEE Trans. on CAD*, vol. 21, no. 5, pp. 597–604, may 2002.

[19] A. Chandra and R. Kapur, "Bounded adjacent fill for low capture power scan testing," in *Proc. VTS*, 2008, pp. 131–138.

[20] A. Chandra and K. Chakrabarty, "System-on-a-chip test-data compression and decompression architectures based on golomb codes," *IEEE Trans. Comput.-Aided Des.*, pp. 355–368, 2001.

[21] ——, "Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (fdr) codes," *IEEE Trans. Comput.*, vol. 52, no. 8, pp. 1076–1088, Aug. 2003.

[22] ——, "A unified approach to reduce soc test data volume, scan power and testing time," *IEEE Trans. Comput.-Aided Des.*, vol. 22, no. 3, pp. 352–363, Mar. 2003.

[23] J.-Y. Chang, C.-W. Tseng, C.-M. Li, M. Purtell, and E. McCluskey, "Analysis of pattern-dependent and timing-dependent failures in an experimental test chip," in *Test Conference, 1998. Proceedings., International*, 1998, pp. 184–193.

[24] B.-H. Chen, W.-C. Kao, B.-C. Bai, S.-T. Shen, and J. Li, "Response inversion scan cell (risc): A peak capture power reduction technique," in *Asian Test Symposium, 2007. ATS '07. 16th*, 2007, pp. 425–432.

[25] K.-T. Cheng, "Redundancy removal for sequential circuits without reset states," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 12, no. 1, pp. 13–24, 1993.

[26] D. Czysz, G. Mrugalski, N. Mukherjee, J. Rajski, P. Szczerbicki, and J. Tyszer, "Low power compression of incompatible test cubes," in *Proc. ITC*, 2010, pp. 1–10.

[27] ——, "Deterministic clustering of incompatible test cubes for higher power-aware edt compression," *IEEE Trans. Comput.-Aided Des.*, vol. 30, no. 8, pp. 1225–1238, Aug. 2011.

[28] D. Czysz, G. Mrugalski, J. Rajski, and J. Tyszer, "Low power embedded deterministic test," in *VLSI Test Symposium, 2007. 25th IEEE*, 2007, pp. 75–83.

[29] D. Czysz, M. Kassab, X. Lin, G. Mrugalski, J. Rajski, and J. Tyszer, "Low-power scan operation in test compression environment," *IEEE Trans. Comput.-Aided Des.*, vol. 28, no. 11, pp. 1742–1755, 2009.

[30] D. Czysz, G. Mrugalski, N. Mukherjee, J. Rajski, and J. Tyszer, "On compaction utilizing inter and intra-correlation of unknown states," *IEEE Trans. Comput.-Aided Des.*, vol. 29, no. 1, pp. 117–126, 2010.

[31] D. Czysz, G. Mrugalski, N. Mukherjee, and J. R. J. Tyszer, "Compression based on deterministic test vector clustering of incompatible test cubes," in *Proc. ITC*, 2009, pp. 1–10.

[32] D. Czysz, G. Mrugalski, J. Rajski, and J. Tyszer, "Low-power test data application in edt environment through decompressor freeze," *IEEE Trans. Comput.-Aided Des.*, vol. 27, no. 7, pp. 1278–1290, Jul. 2008.

[33] E. D. Kaseridis, Kalligeros, X. Kavousianos, and D. Nikolos, "Efficient multiphase test set embedding for scan-based testing," in *Inf. Pap. Dig. ETS*, 2005, pp. 147–150.

[34] P. Darling, "From sand to silicon - the making of a chip," *Intel Process, http://newsroom.intel.com/docs/DOC-2476*, feb. 9, 2013.

[35] S. DasGupta, P. Goel, R. G. Walther, and T. W. Williams, "A variation of lssd and its implications on design and test pattern generation in vlsi," in *ITC*, 1982, pp. 63–66.

[36] R. Dennard, "Design of ion-implanted mosfets with very small physical dimensions," *IEEE Journal of Solid State Circuits*, vol. SC-9, no. 5, pp. 256–268, April 1974.

[37] B. Dervisoglu and G. Stong, "Design for testability using scanpath techniques for path-delay test and measurement," in *Test Conference, 1991, Proceedings., International*, 1991, pp. 365–374.

[38] J. Dworak, M. Grimaila, S. Lee, L.-C. Wang, and M. Mercer, "Enhanced do-re-me based defect level prediction using defect site aggregation-mpg-d," in *Test Conference, 2000. Proceedings. International*, 2000, pp. 930–939.

[39] E. B. Eichelberger and T. W. Williams, "A logic design structure for lsi testability," in *Proceedings of the 14th Design Automation Conference*, ser. DAC '77, 1977, pp. 462–468.

[40] A. H. El-Maleh and R. H. Al-Abaji, "Extended frequency-directed run-length code with improved application to system-on-a-chip test data compression," in *Proc. ICECS*, 2002, pp. 449–452.

[41] J. Emmert, C. Stroud, and J. Bailey, "A new bridging fault model for more accurate fault behavior," in *AUTOTESTCON Proceedings, 2000 IEEE*, 2000, pp. 481–485.

[42] F. Ferguson and J. Shen, "A cmos fault extractor for inductive fault analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 7, no. 11, pp. 1181–1194, 1988.

[43] S. Gerez, *Algorithms for VLSI Design Automation*. John Wiley & Sons, 1999.

[44] P. Girard, "Low power testing of vlsi circuits: problems and solutions," in *Quality Electronic Design, 2000. ISQED 2000. Proceedings. IEEE 2000 First International Symposium on*, 2000, pp. 173–179.

[45] P. Girard, L. Guiller, C. Landrault, and S. Pravossoudovitch, "A test vector inhibiting technique for low energy bist design," in *VLSI Test Symposium, 1999. Proceedings. 17th IEEE*, 1999, pp. 407–412.

[46] D. Gizopoulos, A. Pachalis, Y. Zorian, and M. Psarakis, "An effective bist scheme for arithmetic logic units," in *Test Conference, 1997. Proceedings., International*, 1997, pp. 868–877.

[47] D. Gizopoulos, A. Paschalis, and Y. Zorian, "An effective built-in self-test scheme for parallel multipliers," *Computers, IEEE Transactions on*, vol. 48, no. 9, pp. 936–950, 1999.

[48] L. Goldstein, "Controllability/observability analysis of digital circuits," *Circuits and Systems, IEEE Transactions on*, vol. 26, no. 9, pp. 685–693, 1979.

[49] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolici, "Variable-length input huffman coding for system-on-a-chip test," *IEEE Trans. Comput.-Aided Des.*, vol. 22, no. 6, pp. 783–796, Jun. 2003.

[50] P. Gonciari, B. Al-Hashimi, and N. Nicolici, "Synchronization overhead in soc compressed test," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, no. 1, pp. 140–152, 2005.

[51] I. Hamzaoglu and J. Patel, "Reducing test application time for full scan embedded cores," in *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, 1999, pp. 260–267.

[52] H. Hashempour, F. Meyer, F. Lombardi, and F. Karimi, "Hybrid multisite testing at manufacturing," in *Test Conference, 2003. Proceedings. ITC 2003. International*, vol. 1, 2003, pp. 927–936.

[53] S. Hellebrand, H.-G. Liang, and H. Wunderlich, "A mixed mode bist scheme based on reseeding of folding counters," in *Test Conference, 2000. Proceedings. International*, 2000, pp. 778–784.

[54] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers," *Computers, IEEE Transactions on*, vol. 44, no. 2, pp. 223–233, 1995.

[55] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, and J. Rajski, "Logic bist for large industrial designs: real issues and case studies," in *Test Conference, 1999. Proceedings. International*, 1999, pp. 358—367.

[56] D. Huffman, "Run-length encoding," *IEEE Trans. Info Theory*, no. IT-12, pp. 399–401, Jul.

[57] ——, "A method for the construction of minimum-redundancy codes," *Proc. of the IRE*, vol. 40, no. 9, pp. 1098–1101, Sept. 1952.

[58] B. Ireland and J. Marshall, "Matrix method to determine shift-register connections for delayed pseudorandom binary sequences," *Electronics Letters*, vol. 4, no. 21, pp. 467–468, 1968.

[59] J. T. Janusz Rajski, *Arithmetic built-in self-test for embedded systems.* Prentice-Hall, Englewood Cliffs, NJ, 1998.

[60] A. Jas, J. Ghosh-Dastidar, M.-E. Ng, and N. A. Touba, "An efficient test vector compression scheme using selective huffman coding," *IEEE Trans. Comput.-Aided Des.*, vol. 22, no. 6, pp. 797–806, Jun. 2003.

[61] A. Jas, J. Ghosh-Dastidar, and N. A. Touba, "Scan vector compression/decompression using statistical coding," in *Proc. VTS*, 1999, pp. 114–120.

[62] A. Jas and N. Touba, "Test vector decompression via cyclical scan chains and its application to testing core-based designs," in *Test Conference, 1998. Proceedings., International*, 1998, pp. 458–464.

[63] D. Kagaris and S. Tragoudas, "On the design of optimal counter-based schemes for test set embedding," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 18, no. 2, pp. 219–230, 1999.

[64] S. Kajihara, K. Ishida, and K. Miyase, "Test vector modification for power reduction during scan testing," in *VLSI Test Symposium, 2002. (VTS 2002). Proceedings 20th IEEE*, 2002, pp. 160–165.

[65] E. Kalligeros, D. Kaseridis, X. Kavousianos, and D. Nikolos, "Reseeding-based test set embedding with reduced test sequences," in *Quality of Electronic Design, 2005. ISQED 2005. Sixth International Symposium on*, 2005, pp. 226–231.

[66] E. Kalligeros, X. Kavousianos, and D. Nikolos, "Multiphase bist: a new reseeding technique for high test-data compression," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, no. 10, pp. 1429–1446, 2004.

[67] ——, "Efficient multiphase test set embedding for scan-based testing," in *Quality Electronic Design, 2006. ISQED '06. 7th International Symposium on*, 2006, pp. 432–438.

[68] R. Kapur, R. Chandramouli, and T. W. Williams, "Strategies for low-cost test," *IEEE Des. Test*, vol. 18, no. 6, pp. 47–54, Nov. 2001.

[69] X. Kavousianos and K. Chakrabarty, "Generation of compact test sets with high defect coverage," in *Proc. DATE*, 2009, pp. 1130–1135.

[70] X. Kavousianos, K. Chakrabarty, E. Kalligeros, and V. Tenentes, "Defect coverage-driven window-based test compression," in *Test Symposium (ATS), 2010 19th IEEE Asian*, 2010, pp. 141–146.

[71] X. Kavousianos, E. Kalligeros, and D. Nikolos, "Multilevel huffman coding: An efficient test-data compression method for ip cores," *IEEE Trans. Comput.-Aided Des.*, vol. 26, no. 6, pp. 1070–1083, Jun. 2007.

[72] ——, "Optimal selective huffman coding for test-data compression," *IEEE Trans. on Comput.*, vol. 56, no. 8, pp. 1146–1152, Aug. 2007.

[73] ——, "Multilevel-huffman test-data compression for ip cores with multiple scan chains," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 7, pp. 926–931, 2008.

[74] ——, "Test data compression based on variable-to-variable huffman encoding with codeword reusability," *IEEE Trans. Comput.-Aided Des.*, vol. 27, no. 7, pp. 1333–1338, 2008.

[75] X. Kavousianos, V. Tenentes, K. Chakrabarty, and E. Kalligeros, "Defect-oriented lfsr reseeding to target unmodeled defects using stuck-at test sets," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 12, pp. 2330–2335, 2011.

[76] H. Ko and N. Nicolici, "Automated scan chain division for reducing shift and capture power during broadside at-speed test," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 11, pp. 2092–2097, 2008.

[77] B. Koenemann, "Lfsr-coded test patterns for scan designs," in *Proc. ETS/ETC, VDE Verlag*, 1991, pp. 237–242.

[78] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A smartbist variant with guaranteed encoding," in *Test Symposium, 2001. Proceedings. 10th Asian*, 2001, pp. 325–330.

[79] C. V. Krishna, A. Jas, and N. A. Touba, "Test vector encoding using partial lfsr reseeding," in *Proc. ITC*, 2001, pp. 885–893.

[80] ——, "Achieving high encoding efficiency with partial dynamic lfsr reseeding," *ACM Trans. Des. Autom. of Electr. Syst.*, vol. 9, no. 4, pp. 500–516, Oct. 2004.

[81] C. V. Krishna and N. Touba, "Reducing test data volume using lfsr reseeding with seed compression," in *Test Conference, 2002. Proceedings. International*, 2002, pp. 321–330.

[82] ——, "Adjustable width linear combinational scan vector decompression," in *Computer Aided Design, 2003. ICCAD-2003. International Conference on*, 2003, pp. 863–866.

[83] H. K. Lee and D. S. Ha, "Atalanta: An efficient atpg for combinational circuits," in *TR, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University*, 1993, pp. 93–12.

[84] J. Lee and N. A. Touba, "Lfsr-reseeding scheme achieving low-power dissipation during test," *IEEE Trans. Comput.-Aided Des.*, vol. 26, no. 2, pp. 396–401, Feb. 2007.

[85] J. Lee and N. Touba, "Low power test data compression based on lfsr reseeding," in *Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on*, 2004, pp. 180–185.

[86] K.-J. Lee, J.-J. Chen, and C.-H. Huang, "Using a single input to support multiple scan chains," in *Computer-Aided Design, 1998. ICCAD 98. Digest of Technical Papers. 1998 IEEE/ACM International Conference on*, 1998, pp. 74–78.

[87] L.-J. Lee, W.-D. Tseng, R.-B. Lin, and C.-H. Chang, "2^n pattern run-length for test data compression," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 4, pp. 644–648, 2012.

[88] Y. Levendel and P. Menon., "Transition faults in combinational circuits: input transition test generation and fault simulation," in *Proc. Fault Tolerant Computing Symp.*, July 1986, pp. 278–283.

[89] J. Li, Q. Xu, Y. Hu, and X. Li, "ifill: An impact-oriented x-filling method for shift- and capture-power reduction in at-speed scan-based testing," in *Design, Automation and Test in Europe, 2008. DATE '08*, 2008, pp. 1184–1189.

[90] L. Li and K. Chakrabarty, "Test data compression using dictionaries with fixed-length indices [soc testing]," in *VLSI Test Symposium, 2003. Proceedings. 21st*, 2003, pp. 219–224.

[91] ——, "Test set embedding for deterministic bist using a reconfigurable interconnection network," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, no. 9, pp. 1289–1305, 2004.

[92] L. Li, K. Chakrabarty, S. Kajihara, and S. Swaminathan, "Efficient space/time compression to reduce test data volume and testing time for ip cores," in *Proc. ICVD*, 2005, pp. 53–58.

[93] W. Li, S. Reddy, and I. Pomeranz, "On reducing peak current and power during test," in *VLSI, 2005. Proceedings. IEEE Computer Society Annual Symposium on*, 2005, pp. 156–161.

[94] C.-H. Lin and C.-W. Jen, "Low power parallel huffman decoding," *Electronics Letters*, vol. 34, no. 3, pp. 240–241, Feb. 1998.

[95] C. J. Lin and S. Reddy, "On delay fault testing in logic circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 6, no. 5, pp. 694–703, 1987.

[96] S.-P. Lin, C.-L. Lee, J.-E. Chen, J.-J. Chen, K.-L. Luo, and W.-C. Wu, "A multilayer data copy test data compression scheme for reducing shifting-in power for multiple scan design," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 7, pp. 767–776, 2007.

[97] X. Lin, K.-H. Tsai, C. Wang, M. Kassab, J. Rajski, T. Kobayashi, R. Klingenberg, Y. Sato, S. Hamada, and T. Aikyo, "Timing-aware atpg for high quality at-speed testing of small delay defects," in *Test Symposium, 2006. ATS '06. 15th Asian*, 2006, pp. 139–146.

[98] S. Ma, P. Franco, and E. McCluskey, "An experimental chip to evaluate test techniques experiment results," in *Test Conference, 1995. Proceedings., International*, 1995, pp. 663–672.

[99] J. Marshall, B. Ireland, B. Bajoga, and K. Latawiec, "New method of generation of shifted linear pseudorandom binary sequences," *Electrical Engineers, Proceedings of the Institution of*, vol. 122, no. 4, pp. 448–, 1975.

[100] P. Maxwell, R. Aitken, V. Johansen, and I. Chiang, "The effect on different test sets on quality level prediction: when is 80% betterh than 90%?" in *Test Conference, 1991, Proceedings., International*, 1991, pp. 358–364.

[101] E. J. McCluskey, *Logic design principles - with emphasis on testable semicustom circuits*, ser. Prentice Hall series in computer engineering. Prentice Hall, 1986.

[102] S. Mitra and K. S. Kim, "Xpand: an efficient test stimulus compression technique," *Computers, IEEE Transactions on*, vol. 55, no. 2, pp. 163–173, 2006.

[103] K. Miyase, S. Kajihara, and S. Reddy, "Multiple scan tree design with test vector modification," in *Test Symposium, 2004. 13th Asian*, 2004, pp. 76–81.

[104] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, April 1965.

[105] G. Mrugalski, J. Rajski, D. Czysz, and J. Tyszer, "New test data decompressor for low power applications," in *Proc. DAC*, 2007, pp. 539–544.

[106] G. Mrugalski, J. Rajski, and J. Tyszer, "Ring generators - new devices for embedded test applications," *IEEE Trans. Comput.-Aided Des.*, vol. 23, no. 9, pp. 1306–1320, Sept. 2004.

[107] G. Mrugalski, J. Tyszer, and J. Rajski, "Linear independence as evaluation criterion for two-dimensional test pattern generators," in *VLSI Test Symposium, 2000. Proceedings. 18th IEEE*, 2000, pp. 377–386.

[108] S. Neophytou and M. K. Michael, "Test set generation with a large number of unspecified bits using static and dynamic techniques," *IEEE Trans. Computers*, vol. 59, no. 3, pp. 301–316, 2010.

[109] ——, "Test pattern generation of relaxed n-detect test sets," *IEEE Trans. VLSI Syst.*, vol. 20, no. 3, pp. 410–423, 2012.

[110] N. Nicolici and B. Al-Hashimi, *Power-Constrained Testing of VLSI Circuits*. Kluwer Academic, Norwell, MA, 2003.

[111] M. Nourani and M. H. Tehranipour, "Rl-huffman encoding for test compression and power reduction in scan applications," *ACM Trans. Des. Autom. of Electr. Syst.*, vol. 10, pp. 91–115, Jan. 2005.

[112] A. Pandey and J. Patel, "Reconfiguration technique for reducing test time and test data volume in illinois scan architecture based designs," in *VLSI Test Symposium, 2002. (VTS 2002). Proceedings 20th IEEE*, 2002, pp. 9–15.

[113] E. Park, M. Mercer, and T. Williams, "Statistical delay fault coverage and defect level for delay faults," in *Test Conference, 1988. Proceedings. New Frontiers in Testing, International*, 1988, pp. 492–499.

[114] K. Parker and E. McCluskey, "Probabilistic treatment of general combinational networks," *Computers, IEEE Transactions on*, vol. C-24, no. 6, pp. 668–670, 1975.

[115] S. Patil and J. Savir, "Skewed-load transition test: Part ii, coverage," in *Test Conference, 1992. Proceedings., International*, 1992, pp. 714–722.

[116] W. H. M. Paul H. Bardell, "Self-testing of multichip logic modules," in *ITC*, 1982, pp. 200–204.

[117] I. Pomeranz and S. Reddy, "Transition path delay faults: A new path delay fault model for small and large delay defects," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 1, pp. 98–107, 2008.

[118] B. Pouya and A. Crouch, "Optimization trade-offs for vector volume and test power," in *Test Conference, 2000. Proceedings. International*, 2000, pp. 873–881.

[119] R. Putman and R. Gawde, "Enhanced timing-based transition delay testing for small delay defects," in *VLSI Test Symposium, 2006. Proceedings. 24th IEEE*, 2006.

[120] W. Qiu, L.-C. Wang, D. Walker, D. Reddy, X. Lu, Z. Li, W. Shi, and H. Balachandran, "K longest paths per gate (klpg) test generation for scan-based sequential circuits," in *Test Conference, 2004. Proceedings. ITC 2004. International*, 2004, pp. 223–231.

[121] J. Rajski, N. Tamarapalli, and J. Tyszer, "Automated synthesis of phase shifters for built-in self-test applications," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, no. 10, pp. 1175–1188, 2000.

[122] J. Rajski and J. Tyszer, "Design of phase shifters for bist applications," in *VLSI Test Symposium, 1998. Proceedings. 16th IEEE*, 1998, pp. 218–224.

[123] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded deterministic test," *IEEE Trans. Comput.-Aided Des.*, vol. 23, no. 5, pp. 776–792, 2004.

[124] W. Rao, I. Bayraktaroglu, and A. Orailoglu, "Test application time and volume compression through seed overlapping," in *Design Automation Conference, 2003. Proceedings*, 2003, pp. 732–737.

[125] S. Reda and A. Orailoglu, "Reducing test application time through test data mutation encoding," in *Proc. DATE*, 2002, pp. 387–393.

[126] S. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz, "On test data volume reduction for multiple scan chain designs," in *VLSI Test Symposium, 2002. (VTS 2002). Proceedings 20th IEEE*, 2002, pp. 103–108.

[127] S. Reddy, I. Pomeranz, and S. Kajihara, "Compact test sets for high defect coverage," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 16, no. 8, pp. 923–930, 1997.

[128] S. Remersaro, X. Lin, S. Reddy, I. Pomeranz, and J. Rajski, "Scan-based tests with low switching activity," *Design Test of Computers, IEEE*, vol. 24, no. 3, pp. 268–275, 2007.

[129] S. Remersaro, X. Lin, Z. Zhang, S. Reddy, I. Pomeranz, and J. Rajski, "Preferred fill: A scalable method to reduce capture power for scan based designs," in *Test Conference, 2006. ITC '06. IEEE International*, 2006, pp. 1–10.

[130] P. Rosinger, P. Gonciari, B. Al-Hashimi, and N. Nicolici, "Simultaneous reduction in volume of test data and power dissipation for systems-on-a-chip," *Electronics Letters*, vol. 37, no. 24, pp. 1434–1436, 2001.

[131] M. Rudberg and L. Wanhammar, "High speed pipelined parallel huffman decoding," in *Proc. ISCAS*, 1997, pp. 2080–2083.

[132] L. P. S. C. Seth and V. D. Agrawal, "Predict-probabilistic estimation of digital circuit testability," in *Intl. Symp. Fault-Tolerant Computing*, 1985, pp. 220–225.

[133] S. Samaranayake, E. Gizdarski, N. Sitchinava, F. Neuveux, R. Kapur, and T. Williams, "A reconfigurable shared scan-in architecture," in *VLSI Test Symposium, 2003. Proceedings. 21st*, 2003, pp. 9–14.

[134] K. Sankaralingam, R. Oruganti, and N. Touba, "Static compaction techniques to control scan vector power dissipation," in *VLSI Test Symposium, 2000. Proceedings. 18th IEEE*, 2000, pp. 35–40.

[135] J. Savir, "Skewed-load transition test: Part i, calculus," in *Test Conference, 1992. Proceedings., International*, 1992, pp. 705–713.

[136] J. Savir and S. Patil, "On broad-side delay test," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 3, pp. 368–372, 1994.

[137] J. Saxena, K. M. Butler, V. B. Jayaram, S. Kundu, N. V. Arvind, P. Sreeprakash, and M. Hachinger, "A case study of ir-drop in structured at-speed testing," in *Proc. ITC*, vol. 1, Oct. 2003, pp. 1098–1104.

[138] L. Schafer, R. Dorsch, and H. Wunderlich, "Respin++ - deterministic embedded test," in *Test Workshop, 2002. Proceedings. The Seventh IEEE European*, 2002, pp. 37–44.

[139] S.-W. Seong and P. Mishra, "Bitmask-based code compression for embedded systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 4, pp. 673–685, 2008.

[140] M. Shah and J. Patel, "Enhancement of the illinois scan architecture for use with multiple scan inputs," in *VLSI, 2004. Proceedings. IEEE Computer society Annual Symposium on*, 2004, pp. 167–172.

[141] C. Shi and R. Kapur, "How power-aware test improves reliability and yield," *EE Times EDA news online*, 09/15/2004.

[142] N. Sitchinava, E. Gizdarski, S. Samaranayake, F. Neuveux, R. Kapur, and T. Williams, "Changing the scan enable during shift," in *VLSI Test Symposium, 2004. Proceedings. 22nd IEEE*, 2004, pp. 73–78.

[143] G. L. Smith, "Model for delay faults based upon paths," in *ITC*, 1985, pp. 342–351.

[144] C. Stroud, *A Designer's Guide to Built-In Self-Test*. Springer, Boston, MA, 2002.

[145] ——, "An automated bist approach for general sequential logic synthesis," in *Design Automation Conference, 1988. Proceedings., 25th ACM/IEEE*, 1988, pp. 3–8.

[146] S. Swaminathan and K. Chakrabarty, "On using twisted-ring counters for test set embedding in bist," in *JETTA, vol. 17, no. 6*, Dec. 2001, pp. 529–542.

[147] H. Tang, G. Chen, S. Reddy, C. Wang, J. Rajski, and I. Pomeranz, "Defect aware test patterns," in *Design, Automation and Test in Europe, 2005. Proceedings*, 2005, pp. 450–455 Vol. 1.

[148] H. Tang, S. Reddy, and I. Pomeranz, "On reducing test data volume and test application time for multiple scan chain designs," in *Test Conference, 2003. Proceedings. ITC 2003. International*, vol. 1, 2003, pp. 1079–1088.

[149] M. H. Tehranipoor, M. Nourani, and K. Chakrabarty, "Nine-coded compression technique for testing embedded cores in socs," *IEEE Trans. Very Large Scale Integr.*, vol. 13, no. 6, pp. 719–731, Jun. 2005.

[150] M. Tehranipour, M. Nourani, K. Arabi, and A. Afzali-Kusha, "Mixed rl-huffman encoding for power reduction and data compression in scan test," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, vol. 2, 2004, pp. II–681–4 Vol.2.

[151] V. Tenentes and X. Kavousianos, "Self-freeze linear decompressors for low power testing," in *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*, 2010, pp. 63–68.

[152] ——, "Low power test-compression for high test-quality and low test-data volume," in *Test Symposium (ATS), 2011 20th Asian*, 2011, pp. 46–53.

[153] ——, "Test-data volume and scan-power reduction with low ate interface for multi-core socs," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, 2011, pp. 747–754.

[154] ——, "High-quality statistical test-compression with narrow ate interface," *accepted for publication in IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 2013.

[155] V. Tenentes, X. Kavousianos, and E. Kalligeros, "State skip lfsrs: Bridging the gap between test data compression and test set embedding for ip cores," in *Design, Automation and Test in Europe, 2008. DATE '08*, 2008, pp. 474–479.

[156] ——, "Single and variable-state-skip lfsrs: Bridging the gap between test data compression and test set embedding for ip cores," *IEEE Trans. Comput.-Aided Des.*, vol. 29, no. 10, pp. 1640–1644, Oct. 2010.

[157] V. Tenentes and X. Kavousianos, "Self-freeze linear decompressors: Test pattern generators for low power scan testing," in *VLSI 2010 Annual Symposium*, ser. Lecture Notes in Electrical Engineering, N. Voros, A. Mukherjee, N. Sklavos, K. Masselos, and M. Huebner, Eds. Springer Netherlands, 2011, vol. 105, pp. 217–230. [Online]. Available: http://dx.doi.org/10.1007/978-94-007-1488-5_13

[158] N. A. Touba, "Survey of test vector compression techniques," *IEEE Design & Test*, vol. 23, no. 4, pp. 294–303, Apr. 2006.

[159] N. Touba, "Circular bist with state skipping," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 10, no. 5, pp. 668–672, 2002.

[160] J. Tyszer, D. Czysz, G. Mrugalski, N. Mukherjee, and J. Rajski, "On deploying scan chains for data storage in test compression environment," *IEEE Design & Test*, Early access article, 2012.

[161] H. Vergos, D. Nikolos, M. Bellos, and C. Efstathiou, "Deterministic bist for rns adders," *Computers, IEEE Transactions on*, vol. 52, no. 7, pp. 896–906, 2003.

[162] B. Vermeulen, C. Hora, B. Kruseman, E. Marinissen, and R. van Rijsinge, "Trends in testing integrated circuits," in *Test Conference, 2004. Proceedings. ITC 2004. International*, 2004, pp. 688–697.

[163] E. Volkerink, A. Khoche, L. Kamas, J. Rivoir, and H. Kerkhoff, "Tackling test trade-offs from design, manufacturing to market using economic modeling," in *Test Conference, 2001. Proceedings. International*, 2001, pp. 1098–1107.

[164] E. Volkerink, A. Khoche, and S. Mitra, "Packet-based input test data compression techniques," in *Test Conference, 2002. Proceedings. International*, 2002, pp. 154–163.

[165] E. Volkerink, A. Khoche, J. Rivoir, and K.-D. Hilliges, "Test economics for multi-site test with modern cost reduction techniques," in *VLSI Test Symposium, 2002. (VTS 2002). Proceedings 20th IEEE*, 2002, pp. 411–416.

[166] E. Volkerink and S. Mitra, "Efficient seed utilization for reseeding based compression," in *VLSI Test Symposium, 2003. Proceedings. 21st*, 2003, pp. 232–237.

[167] H. Vranken, F. Hapke, S. Rogge, D. Chindamo, and E. Volkerink, "Atpg padding and ate vector repeat per port for reducing test data volume," in *Proc. ITC*, 2003, pp. 1069–1078.

[168] J. Waicukauski, E. Lindbloom, B. K. Rosen, and V. Iyengar, "Transition fault simulation," *Design Test of Computers, IEEE*, vol. 4, no. 2, pp. 32–38, 1987.

[169] L.-C. Wang, M. Mercer, S. Kao, and T. Williams, "On the decline of testing efficiency as fault coverage approaches 100%," in *VLSI Test Symposium, 1995. Proceedings., 13th IEEE*, 1995, pp. 74–83.

[170] L.-T. Wang, X. Wen, H. Furukawa, F.-S. Hsu, S.-H. Lin, S.-W. Tsai, K. Abdel-Hafez, and S. Wu, "Virtualscan: a new compressed scan technology for test cost reduction," in *Test Conference, 2004. Proceedings. ITC 2004. International*, 2004, pp. 916–925.

[171] L.-T. Wang, K. Abdel-Hafez, X. Wen, B. Sheu, S. Wu, S.-H. Lin, and M.-T. Chang, "Ultrascan: using time-division demultiplexing/multiplexing (tddm/tdm) with virtualscan for test cost reduction," in *Test Conference, 2005. Proceedings. ITC 2005. IEEE International*, 2005, pp. 946–953.

[172] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[173] S. Wang and S. K. Gupta, "Ds-lfsr: A new bist tpg for low heat dissipation," in *Proceedings of the 1997 IEEE International Test Conference*, ser. ITC '97, 1997, pp. 848–.

[174] S. Wang and S. Gupta, "Lt-rtpg: a new test-per-scan bist tpg for low heat dissipation," in *Test Conference, 1999. Proceedings. International*, 1999, pp. 85–94.

[175] ——, "Lt-rtpg: a new test-per-scan bist tpg for low switching activity," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 8, pp. 1565–1574, 2006.

[176] Z. Wang and K. Chakrabarty, "An efficient test pattern selection method for improving defect coverage with reduced test data volume and test application time," in *Test Symposium, 2006. ATS '06. 15th Asian*, 2006, pp. 333–338.

[177] ——, "Test data compression using selective encoding of scan slices," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 11, pp. 1429–1440, 2008.

[178] ——, "Test-quality/cost optimization using output-deviation-based reordering of test patterns," *IEEE Trans. Comput.-Aided Des.*, vol. 27, no. 2, pp. 352 –365, Feb. 2008.

[179] Z. Wang, K. Chakrabarty, and M. Goessel, "Test set enrichment using a probabilistic fault model and the theory of output deviations," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, 2006, pp. 1–6.

[180] Z. Wang, H. Fang, K. Chakrabarty, and M. Bienek, "Deviation-based lfsr reseeding for test-data compression," *IEEE Trans. Comput.-Aided Des.*, vol. 28, no. 2, pp. 259 –271, Feb. 2009.

[181] S. Ward, C. Schattauer, and N. Touba, "Using statistical transformations to improve compression for linear decompressors," in *Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on*, 2005, pp. 42–50.

[182] X. Wen, K. Miyase, S. Kajihara, H. Furukawa, Y. Yamato, A. Takashima, K. Noda, H. Ito, K. Hatayama, T. Aikyo, and K. Saluja, "A capture-safe test generation scheme for at-speed scan testing," in *Test Symposium, 2008 13th European*, 2008, pp. 55–60.

[183] X. Wen, S. Kajihara, K. Miyase, T. Suzuki, K. Saluja, L.-T. Wang, K. Abdel-Hafez, and K. Kinoshita, "A new atpg method for efficient capture power reduction during scan testing," in *VLSI Test Symposium, 2006. Proceedings. 24th IEEE*, 2006, pp. 6 pp.–65.

[184] X. Wen, K. Miyase, T. Suzuki, S. Kajihara, L.-T. Wang, K. Saluja, and K. Kinoshita, "Low capture switching activity test generation for reducing ir-drop in at-speed scan testing," *J. Electron. Test.*, vol. 24, no. 4, pp. 379–391, Aug 2008.

[185] X. Wen, K. Miyase, T. Suzuki, Y. Yamato, S. Kajihara, L.-T. Wang, and K. Saluja, "A highly-guided x-filling method for effective low-capture-power scan test generation," in *Computer Design, 2006. ICCD 2006. International Conference on*, 2006, pp. 251–258.

[186] X. Wen, Y. Yamashita, S. Morishima, S. Kajihara, L.-T. Wang, K. Saluja, and K. Kinoshita, "Low-capture-power test generation for scan-based at-speed testing," in *Test Conference, 2005. Proceedings. ITC 2005. IEEE International*, 2005, pp. 10 pp.–1028.

[187] T. W. Williams and N. C. Brown, "Defect level as a function of fault coverage," *IEEE Trans. Comput.*, vol. 30, no. 12, pp. 987–988, Dec. 1981.

[188] P. Wohl, J. Waicukauski, S. Patel, F. DaSilva, T. Williams, and R. Kapur, "Efficient compression of deterministic patterns into multiple prpg seeds," in *Test Conference, 2005. Proceedings. ITC 2005. IEEE International*, 2005, pp. 10 pp.–925.

[189] G. Wolff and C. Papachristou, "Multiscan-based test compression and hardware decompression using lz77," in *Proc. ITC*, 2002, pp. 331–339.

[190] A. Wurtenberger, C. Tautermann, and S. Hellebrand, "Data compression for multiple scan chains using dictionaries with corrections," in *Test Conference, 2004. Proceedings. ITC 2004. International*, 2004, pp. 926–935.

[191] H. Yan and A. Singh, "A new delay test based on delay defect detection within slack intervals (ddsi)," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 11, pp. 1216–1226, 2006.

[192] M. Yi, H. Liang, L. Zhang, and W. Zhan, "A novel x-ploiting strategy for improving performance of test data compression," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 2, pp. 324–329, 2010.

[193] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "Test-pattern grading and pattern selection for small-delay defects," in *VLSI Test Symposium, 2008. VTS 2008. 26th IEEE*, 2008, pp. 233–239.

[194] ——, "Test-pattern selection for screening small-delay defects in very-deep submicrometer integrated circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 5, pp. 760–773, 2010.

[195] N. Zacharia, J. Rajski, and J. Tyszer, "Decompression of test data using variable-length seed lfsrs," in *VLSI Test Symposium, 1995. Proceedings., 13th IEEE*, 1995, pp. 426–433.

[196] G. Zeng and H. Ito, "Concurrent core test for soc using shared test set and scan chain disable," in *Proc. DATE*, 2006, pp. 1–6.

[197] Q. Zhou and K. Balakrishnan, "Test cost reduction for soc using a combined approach to test data compression and test scheduling," in *Proc. DATE*, 2007, pp. 1–6.

[198] Y. Zorian, "Testing the monster chip," *Spectrum, IEEE*, vol. 36, no. 7, pp. 54–60, 1999.

Author's Publications

Book Chapters

1. V. Tenentes and X. Kavousianos, "Self-freeze linear decompressors: Test pattern generators for low power scan testing," in VLSI 2010 Annual Symposium, ser. Lecture Notes in Electrical Engineering, N. Voros, A. Mukherjee, N. Sklavos, K. Masselos, and M. Huebner, Eds. Springer Netherlands, 2011, vol. 105, pp. 217–230.

Journal Papers

2. V. Tenentes, X. Kavousianos and E. Kalligeros, "Single and Variable State Skip LFSRs: Bridging the Gap Between Test Data Compression and Test Set Embedding for IP Cores", Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD), vol. 29, no 10, pp. 1640–1644, Oct. 2010.
3. X. Kavousianos, V. Tenentes, K. Chakrabarty, and M. Kalligeros, "Defect-oriented LFSR reseeding to target unmodeled defects using stuck-at test sets", Transactions on Very Large Scale Integrated Circuits & Systems (TVLSI), vol. 19, no 12, pp. 2330-2335, Dec. 2011.
4. V. Tenentes and X. Kavousianos, "High-Quality Statistical Test-Compression with Narrow ATE Interface", accepted for publication in Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD).

Refereed Conference Papers

5. V. Tenentes, X. Kavousianos and E. Kalligeros "State Skip LFSRs: Bridging the Gap between Test Data Compression and Test Set Embedding for IP Cores", Design, Automation & Test in Europe (DATE) Conference, pp. 474-479, March 2008.
6. S. Balatsouka, V. Tenentes and X. Kavousianos and K. Chakrabarty, "Defect Aware X-Filling for Low-Power Scan Testing", Design, Automation & Test in Europe (DATE) Conference, pp. 873-878, March 2010.
7. V. Tenentes and X. Kavousianos, "Self-Freeze Linear Decompressors for Low Power Testing", Computer Society Annual Symposium on VLSI (ISVLSI), pp. 63-68, July 2010.
8. X. Kavousianos, K. Chakrabarty, E. Kalligeros and V. Tenentes, "Defect coverage-driven window-based test compression", 19th Asian Test Symposium (ATS), pp. 141-146, Dec. 2010.
9. V. Tenentes and X. Kavousianos, "Test-Data Volume and Scan-Power Reduction with Low ATE Interface for Multi-Core SoCs", International Conference on Computer-Aided Design (ICCAD), session 10B, San Jose, Nov. 2011

10. V. Tenentes and X. Kavousianos, "Low Power Test-Compression for High Test-Quality and Low Test-Data Volume", 20th Asian Test Symposium (ATS), session A2, New Delhi, Nov. 2011

Posters

11. V. Tenentes, X. Kavousianos and E. Kalligeros, "Shrinking the Application Time of Test Set Embedding by Using Variable-State Skip LFSRs", European Test Symposium(ETS), Inf. Digest., May 2008.

Workshops

12. V. Tenentes and A. Papanikolaou "Interactive field-directed floorplan prototyping for 2D/3D IC's," D43D: 4th Design for 3D Silicon Integration Workshop, June 25th-27th 2012, Lausanne.

Short Vita

Vasileios Tenentes

Mr. Tenentes received his Bachelor degree in Computer Science from the University of Piraeus (Greece) in 2003, and the M.S. degree in Technologies and Applications from the Department of Computer Science at the University of Ioannina (Greece) in 2007. He has worked as a Senior Developer for Voice over IP applications with Siemens Enterprise Networks. He participated, as a Software Engineer with Helic S.A, during the researching, designing and implementation of an automation-tool for the designing and simulation of Mixed-Signal Architectures. The tool is now a successful commercial product. Then, Mr. Tenentes started pursuing his Ph.D. in "Embedded Testing Architectures" at the Department of Computer Science and Engineering of University of Ioannina, in Greece, under a scholarship granted by ESF and National support. His research interests include electronics' design automation tools in particular for test data compression architectures, power consumption modeling, probabilistic simulation and timing analysis, interactive distributed-optimization, test scheduling techniques as well as computational geometry algorithms and synthetic biology. He is a member of the Test Technology Technical Council (TTTC) and a student member of the Institute of Electrical and Electronics Engineers (IEEE) since 2007.

Grant Acknowledgement

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.