EMBEDDED DETERMINISTIC TEST FOR SYSTEMS-ON-A-CHIP

EMBEDDED DETERMINISTIC TEST FOR SYSTEMS-ON-A-CHIP

BY
ADAM KINSMAN, B. ENG. (ENGINEERING PHYSICS)
JUNE 2005

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF MCMASTER UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

MASTER OF APPLIED SCIENCE (2005) McMaster University
(Electrical and Computer Engineering) Hamilton, Ontario

TITLE:	EMBEDDED DETERMINISTIC TEST FOR SYSTEMS-ON-A-CHIP
AUTHOR:	Adam Kinsman, B. Eng. (Engineering Physics)
SUPERVISOR:	Dr. Nicola Nicolici
NUMBER OF PAGES:	xiv, 122

Abstract

Embedded deterministic test (EDT) is a manufacturing test paradigm that combines the compression advantage of built-in self-test with the high fault coverage of deterministic stimuli inherent to methods based on automatic test pattern generation and external testers. Despite enabling the use of low cost testers for rapidly achieving high fault coverage, EDT must consciously use the available tester channels to ensure non-disruptive scaling to future devices of increased complexity. The focus of this thesis is to introduce a new EDT approach for systems-on-a-chip (SOCs) that are designed using embedded cores that are intellectual property (IP)-protected.

Following an introduction to integrated circuit testing and an overview of the related work, we define the criteria that must be satisfied by the EDT approaches for the future SOCs of ever growing complexity. Then we observe that the necessary amount of compressed volume of test data transferred from the tester to the embedded cores in an SOC varies significantly during the testing process. This motivates a novel approach to compressed SOC testing based on time-multiplexing the tester channels. It is shown how the introduction of test control channels will reduce the number of required test data channels which will then have increased usage, as the embedded cores will receive compressed test data only when necessary. Through the use of modular and scalable hardware for on-chip test control and test data decompression, we define a new algorithmic framework for test data compression that is applicable to SOCs comprising IP-protected blocks. Experimental results indicate that our approach compares to the existing approaches for EDT that have similar design criteria and methodology constraints, while providing a seamless integration to low cost test equipment.

Acknowledgments

Many more people have contributed in some way to this work than can be listed here, but a few certainly deserve special mention. I am thankful for the administrative and technical staff of the ECE department at McMaster for much practical assistance. I am also appreciative to Dr. Terry Todd and Dr. Alex Jeremic for their time and valuable feedback which helped to bring this thesis to its final form.

To many friends both personal and professional I owe thanks, especially those in the Computer Aided Design and Test Research group of McMaster, Qiang Xu, Ho Fai Ko, David Lemstra, David Leung and Ehab Anis. To them I am grateful for many hours of stimulating discussion and technical advice which have helped me during the course of my study. To my supervisor Dr. Nicola Nicolici I am deeply indebted for much guidance in both research and in life and for many hours spent in motivating discussion and in helpful revision of this work. This thesis would not have been possible without him.

I am appreciative of my sister in law Amanda and my brothers Joshua, Matthew and Philip for their emotional support and for providing an environment of relaxation in which I can recharge. My father and mother in law Bruce and Jane and my dad and mom Bruce and Jan deserve a great deal of thanks for instilling in me the values which have been so important to do good research, and for supporting me through difficult decisions. For her in-exhaustible patience with me, my wife Pamela deserves more credit for this thesis than any of us, myself included. Her perpetual support has made this work possible.

Above all I am grateful to God for the ability and desire to do research, and for placing these people in my life who have supported me in my endeavors. All things are possible through Him who gives me strength.

Glossary

Abstraction removal of detail to facilitate design description

ATE automated test equipment, used to apply test patterns to ICs

ATE buffer the memory used in ATEs where the test data to be applied is stored

ATPG automatic test pattern generation, deriving of fault based test patterns

At-speed operating with the same clock period as used in normal operation

Automation handing over of control to computers for management of design details

BIST built-in self-test, test hardware placed on-chip to apply test patterns

CAD computer aided design, tools for design automation (also EDA)

Core a pre-designed and pre-verified module which can be reused

Core (firm) a core provided in the form of a netlist

Core (hard) a core provided in the form of a physical layout

Core (soft) a core provided in the form of an RTL description

Core provider one who develops a core, in contrast to system integrator

CUT circuit (or core) under test

Defect a manufacturing error or process variation

Defect coverage the percentage of defects detected by a test set

Design reuse use of already designed cores in a system to ease the design process

Design verification comparison of design against specification

Deterministic BIST fault targeted BIST, in contrast to pseudo-random BIST

DFT design for test, steps/hardware added in design to facilitate test

Diagnosis determining what caused a failed device to fail

EDA electronic design automation, tools for design automation (also CAD)

EDT embedded deterministic test, on-chip hardware facilitating test application

Fault a model which captures the effect of one or many types of defects

Fault coverage the percentage of faults detected by a test set

Fault simulation evaluation of a test pattern to determine what faults it detects

FDR frequency directed run-length encoding

FSM finite state machine, algorithmic hardware control

Functional inputs the inputs of the device used during normal operation

Functional test tests focused on circuit operation, in contract to structural test

Golomb a type of statistical coding

HDL hardware description language, syntax for RTL descriptions e.g. Verilog/VHDL

Huffman a type of statistical coding

IC integrated circuit, many transistors integrated in a single package, a device

IEEE 1500 a standard for interfacing to IP-protected cores

IP intellectual property, details of a core which are not disclosed

IP-consistency respecting IP-protection by not requiring IP details

ISCAS'89 benchmarks benchmark circuits widely used in literature on test

ITRS International Technology Roadmap for Semiconductors

JTAG (IEEE 1149) a widely used device programming interface

Layout low level description of a circuit in terms of transistor placements

LFSR linear feedback shift register, a type of PRPG

Low cost tester an ATE without special expensive features

Manufacturing test comparison of manufactured devices against the design

MISR multiple input signature register

Modularity ease of plugability into a system architecture

Moore's law an empirical observation of exponential transistor density growth

Netlist a low-mid level description of a circuit in terms of logic gate connections

Phase shifter a network placed on a PRPG output to reduce shift correlations

Phase shifter reconfiguration use of muxes to modify a phase shifter

Physical layout see layout

Programmability (WRT TDC) ability of a decompressor to pass any test

PRPG pseudo-random pattern generator, produces a pseudo-random sequence

Pseudo-random BIST BIST by random patterns, contrast to deterministic BIST

RTL	register transfer level, a mid-high level description using register operations
Run-length encoding	compression by encoding long runs of 1s or 0s as a symbol
Scalability	the ability to cope with increasing circuit size, VTD, etc.
Scan	access to the internal state registers of an IC
Scan chain reconfiguration	linking of scan chains end to end
Scan chains	shift register structure facilitating scan
Scan frequency	the clock frequency of the scan chains
SCU	scan chain utilization - useful portion of decompressor output
SOC	system-on-a-chip, an integrated circuit comprised of cores
Specification	the formal description of the design requirements for a system
Structural test	tests focused on circuit logic pathways, contrast to functional test
Stuck-at-0	a fault model assuming a node shorted to ground
Stuck-at-1	a fault model assuming a node shorted to power
STUMPS	Self-Test Using a MISR and Parallel Shift register sequence generator
Synthesis	transformation of a description to a lower level of abstraction
Synthesis (architectural)	behavioral $\rightarrow$ RTL
Synthesis (logic)	logic equations $\rightarrow$ netlist
Synthesis (RTL)	RTL $\rightarrow$ logic equations
System integrator	one who links cores to form a system, contrast to core provider
TCU	tester channel utilization - useful portion of decompressor input
TDC	test data compression, reduction in the amount of test data applied to a CUT
Test cube	a partly specified test pattern (containing x-bits)
Test data compression	see TDC
Test pattern	a vector of circuit input values applied with the intent of testing
Test set	a collection of test patterns
Test set compaction	merging of test cubes which do not conflict
Test set dependent	decompression methods tuned to a given test set
Test set independent	decompression methods equally applicable to many test sets
Testbench	stimuli and expected responses used by simulation during verification
VLSI	very large scale integrated circuits, containing millions of transistors
VTD	volume of test data, the amount of data applied to a CUT

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Integrated circuits (ICs) are thin chips made out of semiconductor material (e.g., silicon or germanium), which consist of a connection of semiconductor devices like transistors, and passive components such as resistors or capacitors. The devices on an IC can be used to build logic gates that process electronic signals using the formalism of the Boolean algebra. The ICs based predominantly on logic gates are called digital with examples being microprocessors, microcontrollers, digital signal processors and application specific integrated circuits (ASICs). From simple electronic gadgets such as digital clocks to more sophisticated appliances like cellular phones and personal computers, our lives are influenced by digital ICs. They even affect our lives in ways which are not directly evident. For instance, banks rely heavily on complex computer systems for financial management and security. Although the circuits themselves in all these applications may be very diverse, what they all have in common is that they require design and manufacture. In almost all cases the complexity of the device is well beyond human management at the transistor level. This has spawned the electronic design automation (EDA) industry which provides tools for dealing with designs of ever increasing levels of complexity.

In the remainder of this chapter discussion will be provided on the design flow for very large scale integrated (VLSI) circuits (Section 1.1), providing justification for manufacturing test (Section 1.2) and in turn the need for test data compression (Section 1.3), the focus of this work. Thesis organization is outlined in Section 1.4.

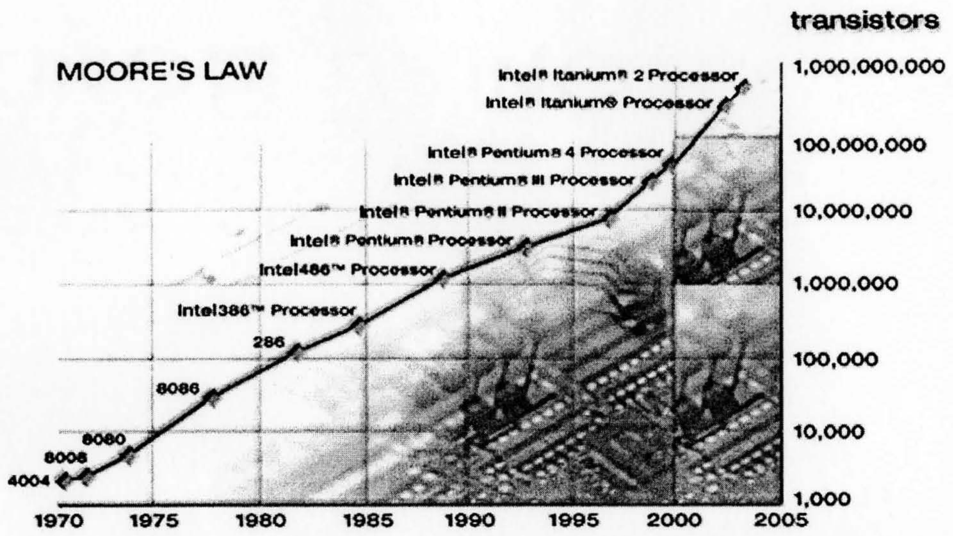


Figure 1.1: Scaling of transistor count (©Intel [29]).

1.1 VLSI Design

The International Technology Roadmap for Semiconductors (ITRS) [36] predicts that by 2010 the number of transistors per chip will exceed four billion. Simultaneously, time to market pressure is stretching the limits of designers to keep pace with the demand for faster, smaller devices. At the same time, device pin counts are increasing at a much slower rate, meaning that we have relatively less access to each internal circuit node as time goes on. Moore's law[1], illustrated in Figure 1.1, further emphasizes the challenge illustrating the exponential growth in number of transistors per chip over the last few decades. The increased transistor density translates into a relentlessly increasing complexity for each new generation of devices. Delivery of these new generations of increasingly complex devices is enabled by a few key factors as listed below and detailed in the forthcoming sub-sections:

- **Design Reuse**: Construction of large systems is facilitated by reusing modules and components which have been pre-designed and pre-verified, and linking them together to obtain the desired functionality (Section 1.1.1).

[1]According to Moore's law the number of transistors on a chip doubles about every two years. It should be noted that although Figure 1.1 shows the scaling trend for Intel Microprocessors, Moore's law is applicable to any type of integrated circuit in general.

- **Abstraction**: Different layers of abstraction have been created between the physical transistor level at which manufacturing occurs, and the design specification which comes to the design engineer (Section 1.1.2).
- **Automation**: By placing the extremely computationally intensive tasks of design in software, the designer may focus on adding value to his specific product while tools handle the implementation tasks common to all the integrated circuits (Section 1.1.3).

1.1.1 Design Reuse

Due to the complexity of current day devices with their enormous roster of embedded features, the time required to design each new system from the ground up would seriously impede progress from a technological stand point. In fact it may in some cases even exceed the anticipated duration of the application for which the device is being designed. To enable designs of such complexity to be completed in a timely fashion design reuse is employed where the functionality of each application is divided into sub-functions for which the hardware may be designed and verified separately and then used again in other designs. As a simple example, a 32 bit adder once designed, can be used in a number of applications without the need to explicitly redesign that functionality into each kind of device which uses it. In a similar fashion to reuse in software development, libraries of functional modules called *cores* are assembled and when a new design is started, the designer can select the required cores from these libraries. As a result the designer may focus on 1) development of the special application cores which may not be available and 2) system level assembly of the cores to invoke the desired functionality. The resulting overall system is called a system-on-a-chip (SOC).

Although reuse is a significant enabler of complex designs with many features, standards are required to ensure smooth integration of cores designed in one place to systems designed in another. For instance there are a few choices in existence for on-chip busses which connect multiple cores in an SOC. There is also the issue of intellectual property (IP). Some businesses are built around the development of cores for particular applications and if the IP is not protected others may reap benefits instead of the company that put the work into developing the core. The issue of IP protection raises challenges, especially in test, which will be addressed in Section 1.2. Simply note for now that SOCs may consist of IP-protected and non-IP-protected cores, that is, some cores to which the system integrator does not have access to the internal circuit details, and some for which access is granted.

1.1.2 Abstraction

In addition to relieving some of the design effort there is a sense in which reuse enables levels of abstraction by providing convenient borders on which to divide the design. Coming back to the adder, there is no need to consider the inner workings of an adder once they are in place and the design may be handled at a level of abstraction where the adder is considered a black box. This level is often referred to as *register transfer level* (RTL). Although reuse facilitates abstraction it is not necessary for abstraction. Levels of abstraction simply provide a way of linking the very high level design specification to the very low level transistor implementation in silicon. Instead of jumping directly from specification to implementation, tasks which the system must perform can be divided and each task handled individually, then each task may be divided into separate blocks and so on with more detail added at each level. For designs of current day complexity the specification essentially gives the behavioral description of the circuit. The job of the designer is to translate the behavioral description into an RTL description using standard hardware description languages (HDLs) such as Verilog HDL or VHDL [98, 99]. For small designs the designer would then be able to progress deeper to the gate level and the transistor level, but complexities of today's chips are beyond the scope of what people can reasonably manage. This has resulted in the development of design automation tools.

1.1.3 Automation

In order to cope with such large complexities, the designer will generally not proceed past the RTL when designing a circuit. Once a complete RTL description in some

HDL has been amassed the HDL is given to a tool called a *logic synthesis* tool. This process transforms the RTL description to a gate level description of the SOC called a gate level netlist. Note that because of the intractability of the problem the gate level netlist will not be optimal for the design at hand, however the results obtained in the time required far outweigh the extra area and performance overhead which may arise from giving control of this level to the tool. Once a gate level netlist has been obtained, it is fed into a *physical design/layout* tool to determine the locations of transistors and how the wires (vias, metal layers) must be routed to obtain the circuit described by the gate level netlist. This process too is sub-optimal however. Were time to be spent searching for an optimal solution, likely no devices would ever be constructed.

1.1.4 VLSI Design Process

Figure 1.2 summarizes the process of designing VLSI integrated circuits. The process begins with a specification which describes in a precise way what the device is expected do. This comes from the application (for instance if the device is to be designed to solve a specific problem), or may be market driven as in the case of devices built to suit consumer demand. The role of the designer is to extract from the specification a high level behavioral description of the circuit. This level of abstraction organizes the system into several large tasks providing convenient divisions of the design effort. The designer also decides what parts of the design may be reused from an assembled library of cores. Once the behavioral description of the circuit is clear, it must be translated to the RTL using a hardware description language such as Verilog HDL or VHDL. For some small classes of designs this process is automated under the term *architectural synthesis* , however in the general case a skilled designer can obtain much better results than an architectural synthesis tool can, at least for designs of current complexity. As a note, scaling trends indicate the likelihood of a level of design complexity being reached in the near future that will require the use of architectural synthesis to obtain acceptable design times. Cores for reuse are also generally instantiated at this level, explicitly in the case of soft cores (HDL descriptions) and as black boxes for firm

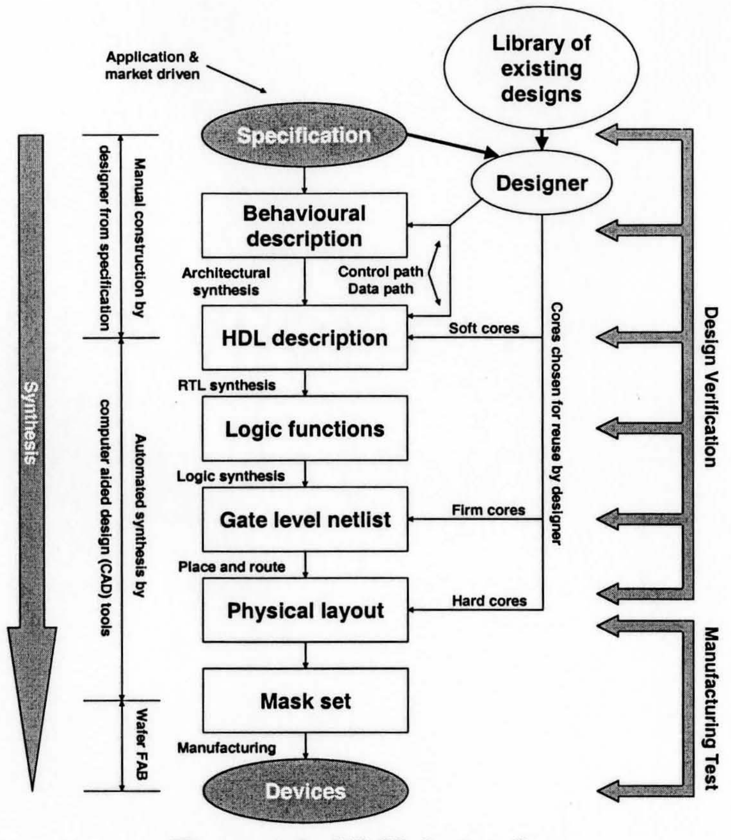


Figure 1.2: VLSI design flow.

(gate level netlists) and hard (layouts) cores. The RTL provides more detail about the operation of the device rather than simply an organization of tasks. It provides information about how data and control interact and evolve in the system to provide the desired functionality given in the behavioral model.

An RTL description of the circuit generally marks the end of heavy manual involvement in the design process as automated tools take over at this point adding more detail at each lower level of abstraction. First an RTL synthesis tool generates the logic functions which describe operations laid out in the RTL. Logic synthesis maps these equations to interconnections of gates called the gate level netlist. Place and route tools take this netlist and provide a layout of transistors and metal interconnects which will be formed in manufacturing. This layout is used for the masks which the wafer fab uses to manufacture the devices, the back-end of the design flow[2].

[2]Note that Figure 1.2 shows an idealized flow with independence between abstraction levels. In reality, cross level interaction is required, especially with movement to smaller technology nodes.

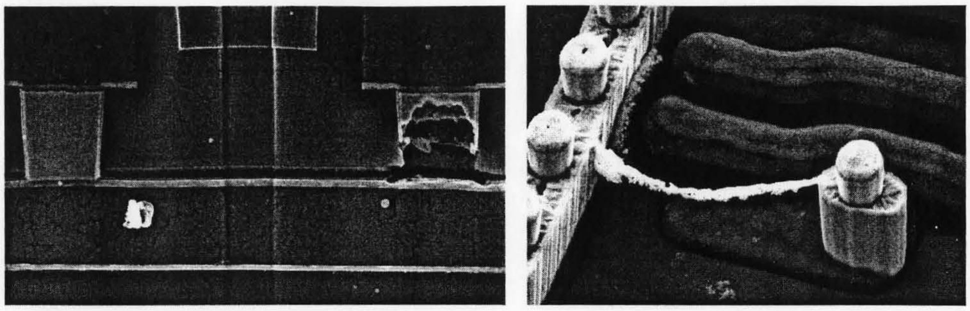
It is to be expected that some design errors will occur in light of the fact that subtle mistakes can be made when handling such a degree of complexity, and that the stages from RTL synthesis onward are handled by software tools which have no explicit "understanding" about the function of the device. At each level of abstraction a tool merely adds detail according to rules applied to input from the level above. To cope with such design errors, *design verification* methods have been developed. Formal verification methods exist, as do simulation based methods which are used much more frequently. Generally speaking testbenches are developed which, through software simulation, apply stimulus to a simulation model of the circuit to check whether operation of the model will align with the specifications. A simulation model exists at each level of abstraction with lower level models containing more detail to reflect the greater detail in the circuit description itself. Because of this greater detail, simulation is more computationally expensive at lower levels of abstraction and so verification is performed at all levels during synthesis to try and catch and correct as many errors as possible as early as possible thereby avoiding having to spend more computational effort on simulation later on. The other reason for verification at each level is that each time a circuit change is made synthesis must be re-done. Accordingly simulation on the gate level model for instance does not require place and route to be run which saves time on each design correction. At the end, design verification should ensure (to the limits of: 1) accuracy of the simulation models and 2) extensiveness of the testbenches) the correlation between the specification and the fabrication instructions (layout).

Other than its role in the design process, design verification is outside the scope of this thesis. It is mentioned more so to bring contrast to the process of manufacturing test since distinction is not often made between test and verification, especially in software development. We have seen in the previous paragraph that the role of verification is to ensure that what we designed is what we intended to design. *The role of manufacturing test is to ensure that what we have manufactured is what we have designed.* The right extreme of Figure 1.2 shows the levels between which comparison occurs for both verification (upper section) and test (lower section). It is immediately clear that while verification is done through software simulation, as we have discussed, manufacturing test makes comparisons at the device level and is therefore a physical process. At the simplest level this process consists of deciding a set of test vectors which are applied to each circuit, and the response returned by each circuit is compared against the expected response which is obtained by simulation. The details of this process, including the generation and application of vectors, are provided in the next section.

1.2 VLSI Test

Due to the stochastic nature of the manufacturing process we cannot have confidence in every device which is manufactured, especially for emerging technology nodes. This stochastic nature leads to what are called process variations, that is for example, not all wafers nor all parts of the same wafer get precisely equal exposure to precisely equal concentrations of developing or etching chemicals. Process variations may affect transistor threshold voltages or wire thicknesses and can result in some metal not being entirely etched away, or metal which should remain being etched away. Other sources of manufacturing errors (defects) include silicon impurities in the wafer or dust particles on the chip during manufacturing. Figure 1.3 shows two examples of the many possible types of defects which may occur. In Figure 1.3(a), the area may have been over-etched or the photo-resist may have been too thin or under-developed in the region of the defect, resulting in less metal being present than should be and thus a resistive open (resistive since not all metal has been removed, some conductivity remains). Figure 1.3(b) shows the opposite case where metal that should have been removed was not, leading to a resistive short along the same line of reasoning just above. In both these cases and for the many other types of defects which may occur, the circuit will not behave as intended. This problem is only becoming more serious as increasingly intricate technologies are used. This section lays the foundations for manufacturing test which addresses those issues just mentioned. More in-depth details about manufacturing test may be found in [1, 7, 13].

Having resolved that manufacturing test must be employed, one may ask “how can it be accomplished?”. A straightforward approach would be to simply apply all



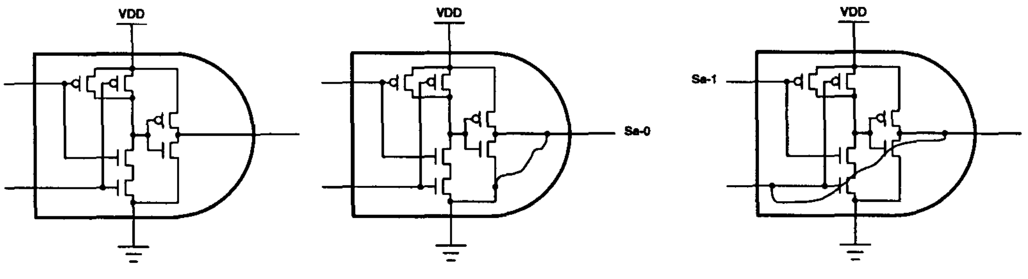
(a) Resistive open defect (b) Resistive short defect

Figure 1.3: Manufacturing defects responsible for device failure [86].

possible circuit inputs and check that the circuit output is correct for each given input combination. This method is called exhaustive functional testing and although it could give extremely high confidence about the operability of the device, it becomes entirely impractical for anything more than the most trivial of devices. For example, consider a 64 bit ripple carry adder. This circuit would have 129 inputs (64+64+1), and thus there are 2^{129} possible input combinations. Assuming automated test equipment (ATE) is used to apply these patterns at 1 GHz (extremely fast by today's standards), the test would take 2.16×10^{22} years to complete [13].

1.2.1 Structural Testing and ATPG

To deal with this issue, the concept of a structural test has been defined. Due to the large number and variety of manufacturing defects which exist, it is impossible to test explicitly for each kind. To overcome this obstacle, fault models have been developed which group defects according to their impact on the logical functionality of the circuit, i.e. we link defects to their manifestation as faults in the logical domain and apply logical tests to look for the presence of faults. The two most common fault models used are the *stuck-at* and *delay* fault models [13]. Delay fault models cover those defects which affect timing, causing a transition to occur later than it ought to. They are outside the realm of material dealt with in this thesis. This work is aimed rather at tests relating to the stuck-at fault model, which covers those defects which manifest themselves as a particular circuit node being fixed to a logical 1 (stuck-at 1) or to a logical 0 (stuck-at 0).



(a) Defect free AND gate (b) Stuck-at-0 fault (c) Stuck-at-1 fault

Figure 1.4: Stuck-at fault models for different defects.

Figure 1.4 shows the fault modeling of defects. In Figure 1.4(a), the transistor level detail of a logical AND gate has been shown. The most straightforward example of a stuck-at fault is the case where the node in question has actually become shorted to a power or ground rail. Figure 1.4(b) shows the case where the output has become shorted to ground, resulting in a stuck-at 0 fault on the output. Not only explicit shorts to power or ground manifest as stuck-at faults however. Take for example Figure 1.4(c), in this figure the output has become shorted to one of the inputs. In this case, when testing the gate (applying patterns to the inputs and observing the output) it will behave as though the other input (the one to which the output is not shorted) is stuck-at 1 since the shorted input will always be passed directly to the output. Hence, a very important observation is that most short or open defects that do not make any particular node to be connected to power/ground rails will still be detected by stuck-at test patterns. In this way a small number of fault models are able to cover many classes of defects. It should be noted that after manufacturing test the failing devices are analyzed and the failing pattern information will be used for diagnosis, which is a separate problem not covered in this thesis.

With fault models in place, the insurmountable task of targeting every kind of defect reduces to the task of targeting each fault at each node, a still difficult but more manageable problem. To this end, a tool called *automatic test pattern generation* (ATPG) is employed which systematically works through the nodes in the circuit description and generates tests according to selected fault models, these being the aforementioned structural tests. Figure 1.5 shows the process of generating a test for a particular stuck-at fault in a circuit. At the basic level, a test for a given stuck-at

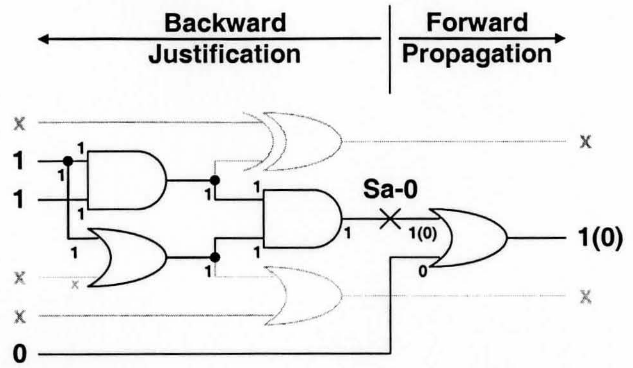


Figure 1.5: Structural test vector generation.

fault on a given node is simply to try to drive that circuit node to the opposite value of the fault. In the figure, the test is for stuck-at 0 so the test involves driving that node to a 1. After establishing this node, values to control the primary inputs are determined through *backward justification* - values to excite the fault and *forward propagation* - values to carry the effect of the fault to the primary outputs where it can be observed. Fittingly, the ease of setting a node to a given value through the primary inputs is called *controllability* while the ease of propagating the effect of the fault to the primary outputs is called *observability.* It is also important to note that ATPG is a known NP-complete problem [25].

As can be seen in Figure 1.5, values which are not directly on the excitation or propagation path of the fault are assigned x's - "don't care" values indicating that they may take on a 0 or 1 value without affecting the effectiveness of the test, hereafter referred to as x-bits. This fact is important for test data reduction methods, as will be addressed in detail later. As a side note, stimulus patterns which contain x-bits are referred to as *test cubes.* Also note that after the fault site each point on the propagation path should be of the form *a(b)*, where *a* is the expected response of a correctly operating circuit and *b* is the response of a defective circuit. Obviously these two must be different to be able to discern between good and defective circuits. As a note, fault simulation tools, which examine an input pattern to a circuit and indicate what faults are detected by that pattern, are often used in concert with ATPG. These tools can be used to reduce the number of patterns which must be

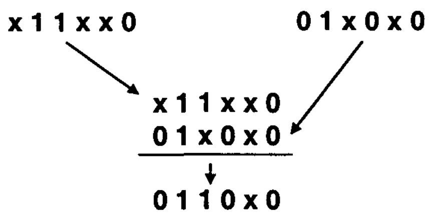
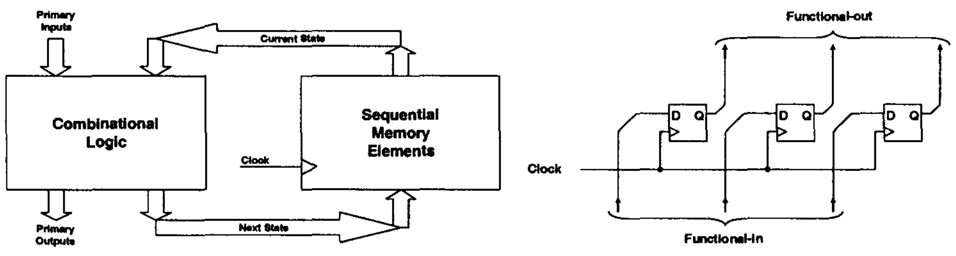


Figure 1.6: Merging of two test vectors into one by compaction.

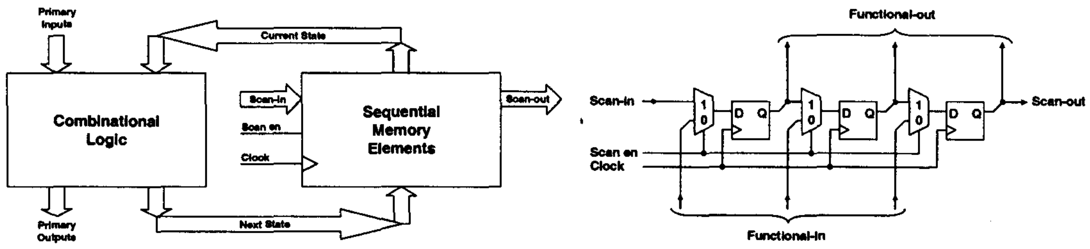
applied to the circuit since some patterns may cover multiple faults. Another method of reducing the number of patterns is called compaction, which merges vectors whose stimulated primary inputs form compatible sets, as shown in Figure 1.6. Also, since some faults are equivalent to one another (e.g., a stuck-at-0 fault on one node may result in a stuck-at-1 fault on another node), the number of required test patterns which must be applied can be further reduced resulting in significantly fewer than are required in the exhaustive functional case. In the same example as given above with a 64 bit ripple carry adder, ATPG for this circuit results in 1728 tests which would take 0.000 001 728s on a 1 GHZ ATE [13].

This example has centered around combinational circuits although in reality most circuits contain a large number of sequential elements. We would naturally ask if the same process may be applied to sequential circuits. Sequential ATPG does in fact exist, however it is extremely impractical because of its intractability caused by huge state spaces. The increased complexity arises from having to not only justify a node through primary inputs, but also through the current state which must be reached through application of successive patterns to the primary inputs during successive clock cycles. In Section 1.1, mention was made that device pin counts are not keeping pace with transistor densities which results in an overall reduction in controllability and observability of nodes in the circuit. This is a significant challenge even for combinational circuits, which is what invalidates sequential ATPG when added to the need to set-up a required state through functional means. This overwhelming complexity is sidestepped through the introduction of design-for-test (DFT) hardware and in the following section the scan-based DFT method is discussed.



(a) Block diagram. (b) Sequential memory element detail.

Figure 1.7: General sequential circuit.



(a) Block diagram. (b) Sequential memory element detail.

Figure 1.8: General sequential circuit with scan DFT.

1.2.2 Scan Based Testing

Figure 1.7(a) shows the block diagram of a general sequential circuit. Based on the present state and the primary inputs, the primary outputs and the next state are computed by the combinational circuit block. When a clock edge arrives at the sequential block, the next state is stored in the memory elements (flip-flops - shown in Figure 1.7(b)) and therefore becomes the present state, starting the cycle again.

The modified circuit can be seen in Figure 1.8 where scan has been inserted. In Figure 1.8(b), it can be seen that the only change to the circuit is the addition of a multiplexer in front of each flip-flop (FF) which allows it to take data either from its respective functional pin of the combinational circuit (when *Scan en = 0*), or from the adjacent FF (when *Scan en = 1*, the test mode). In this way, the FFs in the design can be linked into a shift register structure, and thus any desired state can be shifted in through *Scan-in* and the state at any given time can be shifted out through *Scan-out*. One may ask how we can be sure that the operation of the scan chain (or all the DFT hardware for that matter) is not corrupted by circuit defects. We can in fact assume with reasonable certainty that if the scan chains are not working correctly then at some point a test should fail and thus the chip will be rejected anyway. As a precaution, before any other test patterns are applied the sequence ...0110... is shifted through the scan chains. Since this sequence contains all transitions (0→0, 0→1, 1→0, 1→1), an uncorrupted sequence emerging from the scan chain will also give a high degree of confidence about the reliability of the scan hardware.

By increasing the number of Scan-in and Scan-out pins, the FFs may be linked into multiple scan chains to reduce scan chain length and thereby the time required to shift in a new current state. For example, if a circuit contains 1 million FFs, this may be organized as a single scan chain of length 1 million (using two scan pins) or 2 scan chains with length 500,000 (using 4 scan pins) requiring half the time to scan in a new state. The number of chains can therefore be extended as much as required providing there are sufficient package pins available. The introduction of the muxes obviously incurs area and performance penalties. The penalty in area penalty is obvious, and the performance penalty arises because the current state instead of passing only through the combinational logic to arrive at the FF inputs, must now pass through the scan muxes as well, adding extra propagation delay. This increases the time which must be left between consecutive clock cycles and therefore decreases the maximum operating frequency of the circuit. These penalties are relatively small and well worth accepting given the increased observability and controllability gained. By introducing scan the combinational block may be accessed directly, and thus combinational ATPG as discussed in the previous section may be used to generate test vectors, and the need for sequential ATPG in its intractability is removed.

In addition to the small area and performance penalty brought about by the introduction of scan, the problem of volume of test data (VTD) arises when scan based testing is used. Without scan the size of each pattern corresponds to the number of primary inputs to the circuit, roughly a few hundred at most for even current day large designs. Once scan is integrated however, each pattern must now drive not only the primary inputs but also all the flip-flops for the entire design, a number exceeding 1 million for current day large designs. Delivery of this enormous VTD to the circuit under test (CUT) has posed significant challenges.

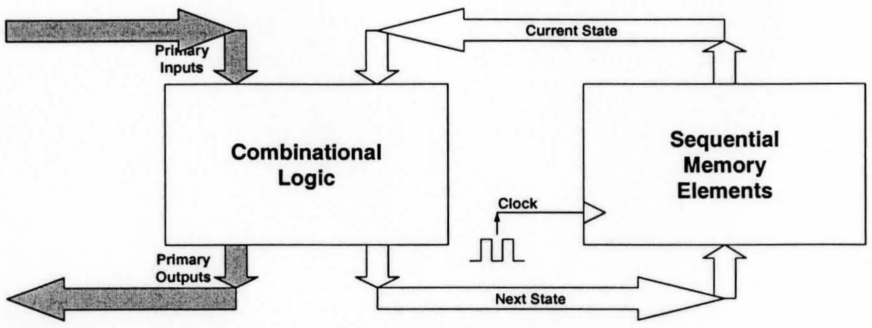
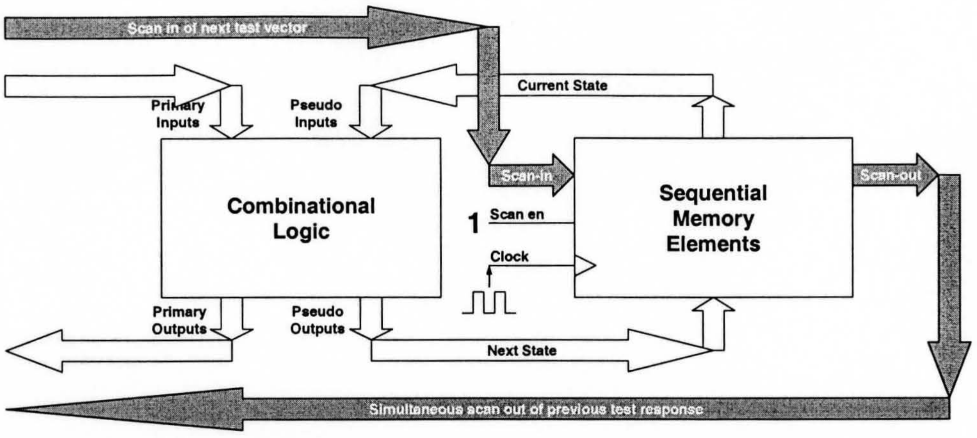


Figure 1.9: Functional test application.

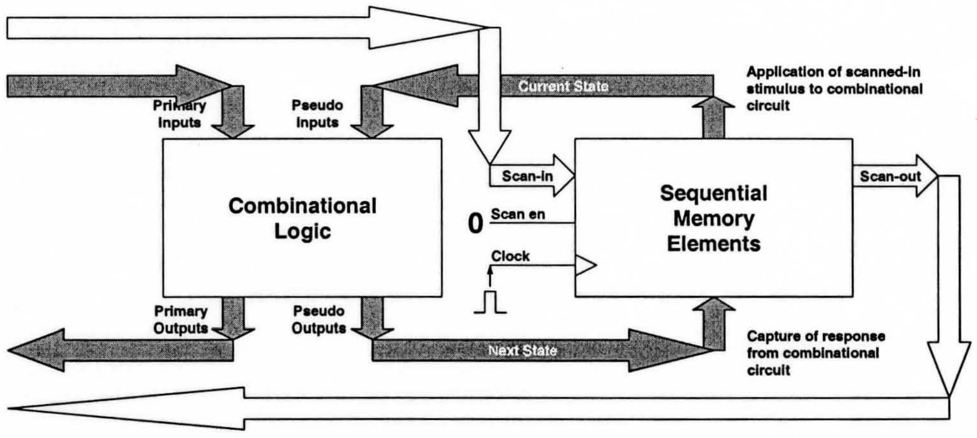
1.2.3 Test Application

With all the discussion surrounding generation of tests for particular faults, mention should be made about the actual application of the patterns to the CUT. Before discussing application of scan vectors generated as detailed in the previous section, we will backtrack slightly and discuss application of functional patterns. Even though functional testing cannot be relied on to give required fault coverage (because of the intractability of sequential ATPG as discussed earlier), it is often used for at-speed testing. Instead of testing exhaustively, structural scan tests are used in conjunction with functional tests which excite the critical path of the circuit (the longest combinational path). It should be noted that the use of scan for at-speed testing has been researched in the recent years and production methods have emerged. Nevertheless, for most designs a very small number of functional patterns (usually obtained from simulation testbenches) are still employed. For this reason, application of functional patterns will be covered first, and then application of scan patterns will be addressed.

Functional testing consists of applying test patterns to the primary inputs of the circuit while applying a clock as in normal functional mode (although the maximum clock speed need not be used) and observing the response of the circuit during each clock cycle on the primary outputs. This scheme is also often called *test per clock* since a test occurs in each clock cycle. Figure 1.9 emphasizes the application of data to the primary inputs and observation at the primary outputs by the shaded arrows. Note that although data is not explicitly applied to the sequential elements, in each clock cycle the values on the inputs influence what is stored in the memory and each current



(a) Scanning in/out new stimulus/previous response.


Primary Inputs
Pseudo Inputs
Current State
Application of scanned-in stimulus to combinational circuit
Combinational Logic
Scan-in
0 Scan en
Clock
Sequential Memory Elements
Scan-out
Primary Outputs
Pseudo Outputs
Next State
Capture of response from combinational circuit

(b) Single clock pulse test application/capture.

Figure 1.10: Scan test application.

state in the memory influences the values appearing on the outputs. As mentioned above, this approach is very efficient in terms of VTD since each pattern consists of only the values for primary inputs, plus primary outputs for the response. It is also much faster since each test requires only a single clock cycle rather than many as is the case for scan. Where this method fails is its inability to ensure adequate fault coverage with reasonable computational effort and thus we require scan also as discussed above.

Figure 1.10 shows the process of applying a scan test to a circuit. The main difference between this and functional testing is that a given state can be reached by scanning it in as opposed to reaching it by manipulation through the primary inputs. In fact, scan enables application of patterns which may not be possible in functional testing since normally unreachable states may be scanned in. This is one of the mechanisms by which scan testing enables higher fault coverage. To apply a scan pattern two steps must occur. The first step is setup of the new state through scan, shown in Figure 1.10(a). Here the scan input arrow has been shaded to emphasize the path along which the new state data is provided. The scan output path has also been highlighted to show that simultaneous to the setup of a new state, the result from application of the previous pattern to the circuit (which is stored in the FFs) is scanned out and compared with the expected response. Note also that the *Scan en* pin has been asserted which links the scan cells into chains as covered in Section 1.2.2. The second step of scan testing is to apply the newly updated state to the circuit and capture the response as in Figure 1.10(b). During this step the *Scan en* pin is de-asserted to put the circuit into normal functional mode so that each FF draws its data from its respective output of the combinational logic. A single clock cycle is applied to the circuit so that the combinational output is captured in the FFs, thus it may be scanned out as the next pattern is scanned in. At the same time that the values from the sequential elements are applied and the results captured, the primary inputs are stimulated and the primary outputs observed, as indicated by the shaded arrows in the figure. Note that scan FFs may be added to drive/capture the primary inputs/outputs in the test mode to circumvent application and observation of them directly. Henceforth in this thesis we refer only to data in the scan chains without loss of generality because of this fact.

In contrast to test per clock, this method is often called test per scan because each test pattern requires multiple clock cycles amounting to a single scan to apply. The increased number of clock cycles required to scan in a new state results in much longer test times than the test per clock scheme. In addition to longer test times, the data for each pattern now consists of values for each FF (likewise for the expected responses) which drastically increases the VTD. Addressing this issue of increased VTD is the topic of the next section.

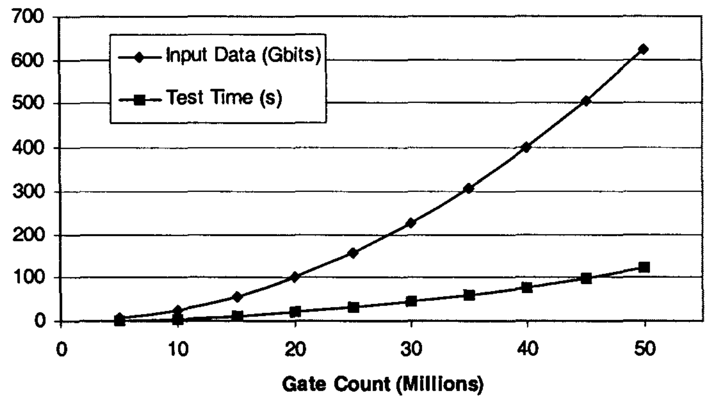


Figure 1.11: Scaling of volume of test data - assumes 20 gates/FF, 1 pattern/10 FFs, 256 scan channels and 20 MHz scan clock.

1.3 Test Data Compression

In the last section we have seen how introduction of scan to enable sufficient fault coverage results in large VTD. We have also provided discussion regarding application of test patterns to a circuit without giving details about what provides those test patterns. In this section we will discuss trends in VTD, followed by elaboration on possible methods of application of test patterns to the CUT and the impact that trends in VTD have on each.

Section 1.1 addressed the challenges arising in design due to trends in device complexity (increasing transistor count). In Section 1.2.1 we saw that pin counts, although increasing, are not even close to keeping up with the rate of increase in transistor count. Both these trends are driving the increasing difficulty in effectively testing VLSI circuits. Rising transistor counts are a direct translation of rising gate counts and FF counts. Rising FF counts directly increase the size of each test pattern which must be applied. At the same time, the reduced observability and controlability coupled with the increased number of faults per circuit (as a result of the increased gate count) is driving up the pattern count. To be direct, increased circuit sizes are yielding increased pattern counts *and* increased data per pattern causing VTD to increase at an accelerated rate as discussed in the next paragraph.

Figure 1.11 shows how VTD is affected by circuit size. The values in the graph are calculated from the gate count according to the assumptions given in the caption. It is assumed that there are 20 gates per flip-flop, a widely accepted empirical result used by many (e.g., [77]). We have assumed here 1 pattern for 10 FFs, another empirical observation which we believe to be optimistic. In reality we believe it to be more patterns per FF - giving an even larger VTD than we obtain here. The assumption of 256 scan channels is justifiable since 256 must be used for each input and output scan channels giving 512 in total, a significant number of pins for the range of device sizes given. Finally, a 20 MHz scan clock is assumed mainly for the sake of containing power during test. For volume of test data only the stimulus has been considered, so the total ATE memory required will be twice the value in the graph (stimulus + response). It has been calculated as the product of $patterns \times FFs$ or $Gates^2/200$ (based on the assumptions). The test time is $patterns \times time\_per\_pattern$ where $time\_per\_pattern = FFs/scan\_channels$ giving $Gates^2/51200$ or $VTD/256$. As mentioned above, the situation is only aggravated by Moore's law which is fueling the increasing gate count. With gates scaling as the square of time, and VTD scaling as the square of gates, VTD has become a serious concern. This problem is further aggravated by the fact that the gate count (and hence scan flip-flops and test patterns) will continue to increase at a faster rate than the tester channel count (determined by the packaging constraints on the chip pin count) and the scan clock frequency (limited by the power constraints).

Having established that VTD is large and is only getting larger, we must evaluate its impact on current test application methods and ensure the problem is addressed in new methods which are proposed. We have seen in Section 1.2.3 that functional testing leads to comparatively small volumes of test data, however, as addressed it cannot provide the fault coverage that is necessary for required product quality. Constrained to use scan based testing, let us consider different pattern application methods and the impact of VTD scaling on them.

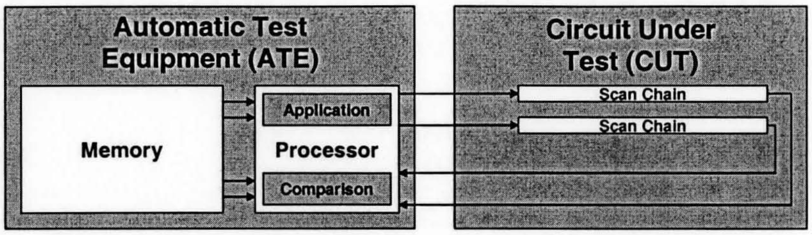


Figure 1.12: Test pattern application with an ATE.

1.3.1 Automated Test Equipment

The most obvious source of test patterns is an external piece of equipment which can apply the required sequences to the CUT and observe and compare the output response. Such systems are called automatic test equipment (ATE), and the test application process is shown in Figure 1.12. The pattern memory stores both the test stimuli and the expected responses. The processor accesses the memory in a stream mode passing stimuli for the next pattern to the scan chains (application block), and comparing the response from the previous pattern to the expected response from the memory (comparison block). The data lines that connect the ATE to the CUT are called tester channels. Note that this is the setup used for functional testing as well where the tester channels are connected to the primary inputs and outputs instead of the scan chains. This method certainly accomplishes the task and allows any possible sequence to be applied. Increasing volume of test data however is raising challenges with this method. Firstly, the memory should be large enough to contain the entire test set since loading it takes significantly longer than applying it to the chip and doing so in the middle of the test would cause unacceptably expensive test times (the test time values shown in Figure 1.11 assume that no ATE buffer reload occurs). ATEs are also very expensive because they must have a significant amount of fast memory. This makes testing expensive since the cost of the tester must be justified by the sales of the devices it is testing. The longer each device takes, the less can be tested in a given time frame, and thus the more each device costs to test. Furthermore, with ATE technology always lagging behind the devices being tested, a way to more effectively leverage the technology at hand to test the devices of the future is essential to avoid prohibitively large testing costs. In answer to the cost

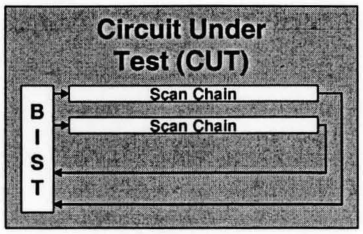


Figure 1.13: BIST test pattern application.

associated challenges arising in test using ATEs, a method known as built-in self-test (BIST) has been explored, as discussed in the next section.

1.3.2 Built-in Self-test

Figure 1.13 shows the philosophy of BIST, placing hardware on the chip to generate test patterns on the fly as well as make the comparison between the expected responses and the actual circuit response. This method entirely solves the problem of large memory and bandwidth requirements between the ATE and the chip. In addition, with BIST in place a chip could use idle time to run checks on itself to ensure continued proper operation. The main problem with BIST is the area overhead incurred. To generate fully deterministic patterns generally requires fully functional finite state machines whose size rival that of the circuits they are testing. An exception to this is memory BIST where the deterministic tests are very regular, and the expected responses are very easily derived from the stimuli. For this reason, memory BIST has enjoyed a great deal more success than logic BIST where the patterns are irregular and responses need dedicated generation.

In addition to the cost of the actual silicon estate, the problem of yield arises if BIST area is unconstrained. With increased BIST area there is a greater probability of a defect in the BIST logic and chips which are functionally acceptable may be discarded on the basis of malfunctioning BIST hardware, decreasing yield. If the size of the BIST hardware were to become comparable to the circuit size, redundant logic is likely to give better results than BIST. Alternatively we can opt for smaller BIST hardware that does not provide deterministic test patterns. Many approaches

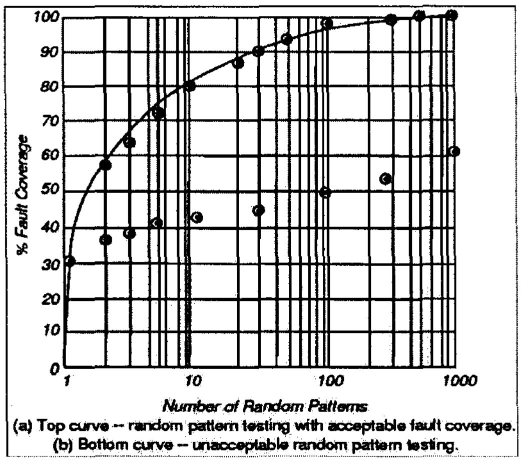


Figure 1.14: BIST fault coverage (©Kluwer [13]).

use some form of pseudo-random pattern generator (PRPG) which continually cycles providing pseudo-random sequences to the circuit. The benefit of PRPGs is a very compact implementation however, it is a accompanied by a reduction in fault coverage.

Figure 1.14 shows the fault coverage obtained by BIST in the ideal case and in practice. In the ideal case random pattern testing gives very good fault coverage, as indicated in the upper curve. However, because of the existence of *random pattern resistant* faults, fault coverage as shown in the lower curve is what can be expected for realistic cases. This level of fault coverage is clearly unacceptable and given the trend in the curve a much larger number of patterns would have to be applied to obtain a reasonable fault coverage level. The use of control points in the circuit may alleviate this problem, nevertheless it will introduce both hardware and performance overhead, in addition to complicating the design process. Thus the fundamental tradeoff for BIST is established, either unacceptably large area overhead or unacceptably long test times are necessary to reach the required fault coverage. As expressed, this tradeoff results from the aforementioned random pattern resistant faults, aside from which random testing is able to substantially reduce the volume of test data. For this reason, hybrid testing has been proposed where pseudo-random patterns are first applied to cover easily detectable faults, after which deterministic pattern application is used to catch the random pattern resistant faults. This marriage of random and deterministic testing is mainly what has sustained BIST research and due to the

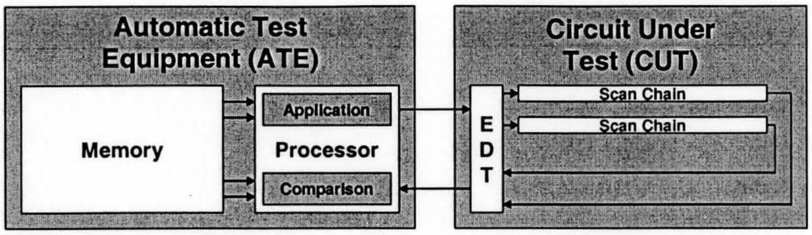


Figure 1.15: EDT - a combination of BIST and ATE.

extensive use of hybrid testing, some previous methods on BIST will be given in Chapter 2, in Section 2.2.

Even though the use of hybrid testing has enabled smaller VTD, we have begun to reach transistor densities that even the deterministic data which must be applied is stressing the capabilities of current day ATEs. In order to be able to keep up with the increasing VTD caused by more complex devices, alternative means of data application will be required. These scaling concerns have been addressed in the form of embedded deterministic test which is the topic of the next section.

1.3.3 Embedded Deterministic Test

Hybrid testing as discussed in the last section can be viewed as a mixture of BIST and ATE testing since we apply patterns through BIST followed by patterns through ATE. There is another way in which ATE and BIST testing can be mixed and that is through simultaneous use. Embedded deterministic test (EDT) accomplishes just this by applying data from the tester to guide on-chip hardware in providing patterns to the circuit as shown in Figure 1.15. EDT captures the benefits of BIST - shorter pattern application time by increased number of scan chains, and reduced external influence; as well as the benefits of ATE - determinism in patterns and reduced on-chip area. The terms EDT and test data compression are often used interchangeably since the data coming from the tester can be seen as "compressed", and the EDT hardware is seen as a decompressor which feeds uncompressed data into the scan chains. Since the compressed data is smaller than the actual test set, the memory requirements on the ATE are less strict and a less expensive tester may be used, thereby reducing the cost of test.

Test data compression is made possible by the presence of the x-bits in the test set produced by ATPG, as seen in Section 1.2.1. Since these bits may take on any value, they are irrelevant information. The compression is accomplished by building a decompressor in such a way that the tester is able to communicate the location and value of the specified bits to it, and having it fill the x-bit values in an alternate way. The irrelevant information is thereby stripped from the test set greatly reducing the volume of test data which must be applied to the chip. What makes EDT challenging is the development of new CAD (or EDA) algorithms for compressing test sets and generating hardware architectures that effectively trade-off compression factor, embedded hardware complexity, ATE features and capacity, performance impact, test time, design time and methodology. It should be noted that a full EDT solution also provides compression for the output responses since often the number of internal scan chains is more than the available pins, and also to reduce the volume of stored output responses to a level on par with the stimuli. This task of compressing output responses comes with its own unique challenges [67, 81, 106] and as a result is considered a separate problem, not in the scope of this work.

1.4 Thesis Organization

In light of what has been presented regarding test data compression, it is clear that many potential methods exist for compression algorithms and on-chip decompression hardware. The relevant work that has been done so far in this field will be established in Chapter 2. The new method we propose for improved compression [51], taking into account not only abstraction and automation, but also design reuse (as discussed in Section 1.1) will be shown in Chapter 3. Following this, Chapter 4 will summarize providing a conclusion and work which remains to be done in the future.

Chapter 2

Related Work

Chapter 1 of this thesis introduced the VLSI design flow, explained the relevant steps in integrated circuit testing, and motivated the need for test data compression. Given the importance of the test data compression problem, this chapter attempts to answer the question: why do we need yet another approach for embedded deterministic test? This is done by reviewing the recent related approaches in the field and summarizing their main strengths and weaknesses.

In Section 2.1 we start by reviewing the ATPG and ATE solutions for test data compression. Some recent BIST approaches relevant to EDT technology are outlined in Section 2.2. Then Section 2.3 gives the basic equation of test data compression which is used for introducing the different research directions in EDT that have been undertaken recently. Section 2.4 reviews several test set dependent approaches that use the specific test cube data to hardwire the on-chip decompression engine and stresses their main limitation. The opposite research direction is labeled as test set independent and can focus either on temporal and spatial decompression. On the one hand the key feature of temporal decompression, detailed in Section 2.5, is that it exploits the ratio between the scan frequency and ATE frequency. On the other hand the spatial decompression exploits the ratio between internal scan chains and tester channels and due to its practical relevance, has been widely adopted as outlined in Section 2.6. Section 2.7 overviews multi-level test data compression and Section 2.8 summarizes this chapter.

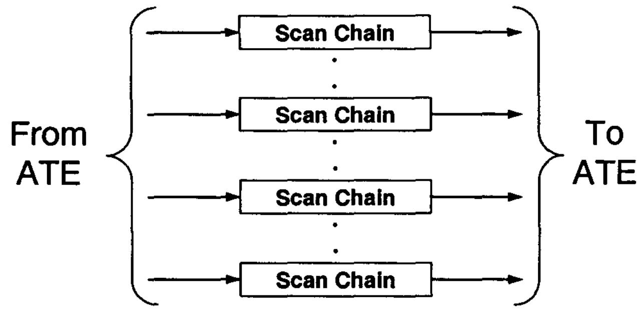


Figure 2.1: ATE test application.

2.1 VTD Reduction by ATPG and ATE

The methods provided in this section rely on external processing methods to reduce the volume of test data required, and as a result, test data application resembles the standard test application where no on-chip decompression hardware is used as shown in Figure 2.1. Two main directions for external reduction of VTD have been taken. The first is the use of ATPG to reduce VTD by dynamic compaction. In this method, compaction (as described in Section 1.2.1) is performed during ATPG to increase the number of faults covered by each pattern and thereby decrease the number of patterns which must be applied. Examples of this approach can be found in [23, 30, 76, 103].

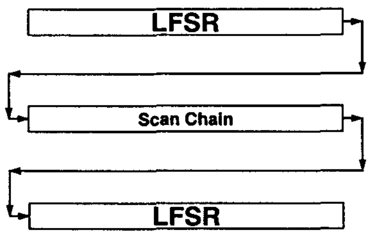
Another way in which the volume of test data to be applied may be reduced is to exploit specific ATE features. The method proposed in [8] pairs filling of x-bits by ATPG with filling features provided by the ATE, i.e. random-fill, 0-fill, 1-fill, and repeat-fill. A similar method is used by [47] which also discusses how the fill features of the ATE may be put on-chip. In [22] the hybrid approach discussed in Section 1.3.2 is used but the patterns consist of some scan chains being filled pseudo-randomly and some being filled from the ATE which requires separate control of channels for random fill and stream from memory. The method in [31] stores the vectors in the ATE memory column-wise instead of row-wise and processes them before delivery to the device while [101] reduces the stored volume of test data by exploiting the repeat per group feature of ATEs to facilitate run-length encoding.

Both ATPG and ATE based methods can provide reasonable VTD reduction and directly target one of the motivating factors for test data compression - the reduction of data stored on the tester. As an added benefit they accomplish VTD reduction in a non-intrusive way since no decompression hardware must be placed on the chip. A drawback however is that although they reduce the VTD stored on the tester, the bandwidth between the ATE and the CUT is not reduced at all as it is clearly shown in Figure 2.1. This means that a tester channel is required for each scan chain and since the number of scan chains is limited by the package pin constraints (Section 1.3), long pattern application times result which cannot be compensated by the reduction in number of patterns mainly because conflicting test patterns limit pattern reduction. In addition, ATE based methods can be expensive since the special features on which they rely cost extra. Besides which, if the tester at hand does in fact have these features they can be used to augment an EDT approach. Furthermore, application of more patterns improves the probability that a defect (not modeled by the targeted fault model) will be covered, and thus improves the quality of the test. It is therefore better to concentrate on reducing the VTD per pattern as accomplished by EDT, rather than targeting only a reduction in the number of patterns.

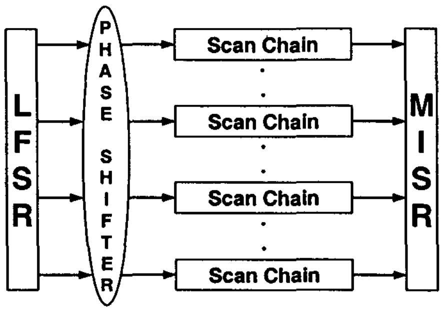
2.2 BIST

The concept of BIST, first introduced in Section 1.3.2, is the holy grail of EDT. By placing test hardware on the chip it seeks to remove entirely the need for external input, which reduces test cost to essentially nothing in comparison to external test application using expensive ATEs. Despite the fact that BIST cannot be used to entirely remove the need for external data application it does provide the basis for EDT and as a result some of the previous work done in BIST is summarized here.

A basic single scan chain architecture for BIST is shown in Figure 2.2(a) where pseudo-random patterns are fed into the scan chains and applied to the circuit. The response is shifted out and captured in a linear feedback shift register (LFSR) that has been configured as a signature register. Figure 2.2(b) shows BIST applied to multiple scan chains, which substantially improves the test time by cutting down the time required to shift each pattern into the scan chains. This configuration is known as the STUMPS configuration for **S***elf*-**T***est* **U***sing a* **M***ISR and* **P***arallel* **S***hift register sequence generator* [7, 13]. To drive multiple chains, all the FFs in the LFSR



(a) Single scan chain BIST.



(b) Multiple scan chain BIST.

Figure 2.2: BIST pattern application.

are used instead of just one as in Figure 2.2(a). In order to break up the strong shift correlations between adjacent FFs in the LFSR, a phase shifter is introduced as illustrated. By taking linear combinations of the LFSR outputs, a separation (called the channel separation) in the pseudo-random sequences can be obtained, hence the name phase shifter. More details regarding the requirement and design of phase shifters can be found in [79]. At the output, the response across the multiple scan chains is collected into the multiple input signature register (MISR). As mentioned earlier, most BIST methods rely on some sort of pseudo-random pattern generator, and LFSRs are very commonly chosen as the PRPG. Figure 2.3 details an example of an LFSR and a MISR. In the LFSR, the state is modified by the feedback polynomial only, whereas in the MISR the next state depends on both the feedback and the inputs, which allows it to be used for response collection.

Although many existing BIST methods use LFSRs, other types of PRPGs operate on the same principle and are equally applicable to BIST. For example, [70] uses linear hybrid cellular automata to drive the scan chains through a phase shifter. In [14], a twisted ring counter is used as the PRPG, but the work attempts to encode seeds for the ring counter to enable increased determinism. Self reseeding hardware is used in [2] for the same reason as in [14] but is applied to LFSRs. An LFSR is also used by [4] where a seed selection algorithm is proposed, and also in [42] which explores the relationship between seeds of a given LFSR. A method similar to reseeding the PRPG is the insertion of state skipping logic which enables dropping of entire sections of the state sequence if desired, as in [96].

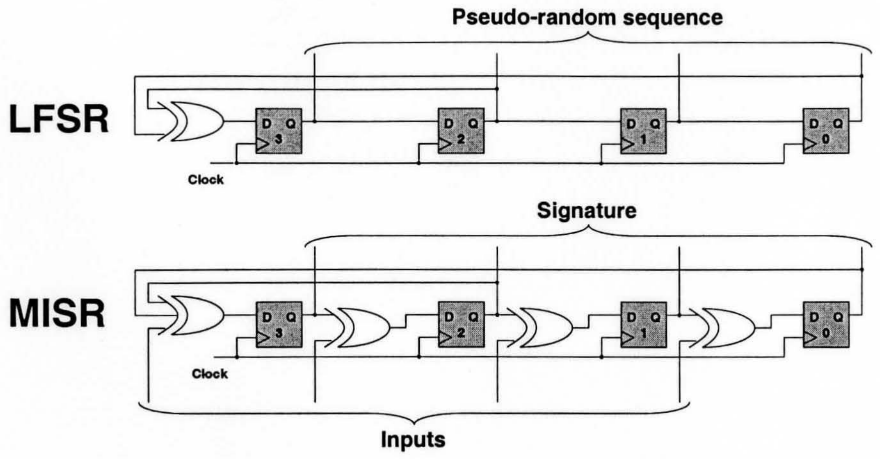


Figure 2.3: Example LFSR and MISR.

Alternatively to encoding seeds in an attempt to increase determinism, there is a research direction in which logic is placed around the PRPG to modify its output. One particular approach skews the probability distribution of bits generated at the output of the PRPG according to a set of weights and is known as weighted pseudo-random pattern testing. Another example, [97] fixes a given output of an LFSR to a 1 or a 0 for some pattern depending on the fixing logic. Similarly to this, [48] flips bits according to some logic instead of fixing them. Both bit flipping and fixing are incorporated in [26]. A reconfigurable interconnect network between the PRPG and the scan chains which is controlled by a configuration counter and a decoder is used by [64]. In [43, 44], different sections of the CUT are tested in different phases, controlled by logic between the PRPG and the scan chains.

Another class of methods including [62, 111] modify the circuit to facilitate BIST by increasing observability/controlability of selected circuit nodes. Although this is sometimes done even in ATPG for scan based deterministic test (using an ATE), it is only required in very extreme cases and generally is not done at all. On the other hand, its usefulness in BIST is to increase effectiveness of random patterns but often the circuit must be significantly modified. It is preferable not to modify the circuit for a couple of reasons. The first is that it interferes with the automation aspect of design described in Section 1.1. Modifications to the circuit, even those performed by an automated tool, can affect the circuit in an undesirable way (e.g. timing closure can be disrupted) requiring feedback in the design flow, which complicates matters. Moreover, circuit modifications do not fully respect the reuse aspect of design since for IP-protected cores no access to internal details of the core is provided, thereby invalidating methods which rely on circuit modifications. The final type of method that will be discussed here is the kind that places certain constraints on the circuit to be tested. As an example take [90] which requires that the SOC have an embedded processor and an FPGA core on which the BIST machinery may be formed. Such methods can be effective in their domain of applicability but often that domain is not nearly general enough.

In addition to BIST limitations outlined in the previous chapter, another consideration which should be made is that BIST hardware can be ineffective during diagnosis of a circuit given its randomness and resistance to determinism. For diagnosis, a high degree of control is desired to enable tracking down of a defect. Where BIST cannot effectively provide such a level of control EDT can work well because of its built-in determinism. In light of these facts and the discussion in Section 1.3.2, we can conclude that EDT is the most viable choice as a solution to the problem of VTD. The next section introduces the governing relationship of EDT, which has motivated its main research directions.

2.3 Test Data Compression

As we have seen in Section 1.3.3, embedded deterministic test places a small decompressor on-chip and uses compressed streams arriving from the ATE, expanding them and applying them to the internal scan chains of the circuit as shown in Figure 2.4. Note that an important part of EDT is compression of the expected output responses since they must be stored on the tester in addition to input stimuli. As discussed in Section 1.3.3 this is considered a separate problem. Consequently, henceforth in this thesis the output response compression hardware will be omitted from figures and discussion without loss of generality.

In light of Figure 2.4, the on-chip decompression hardware can be viewed as a system which takes a small bandwidth stream of data in (from the ATE) and delivers

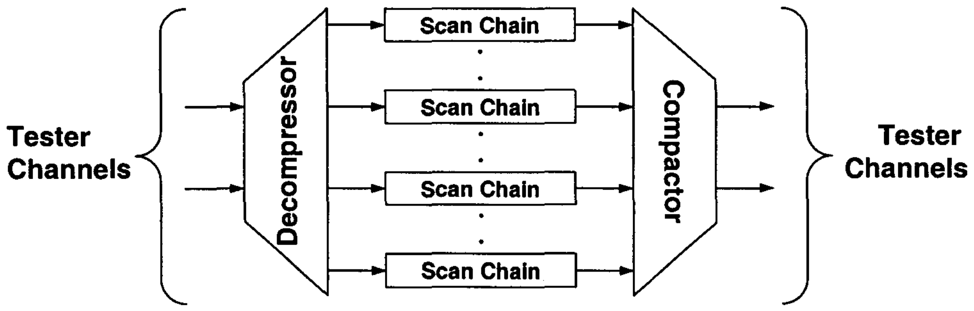


Figure 2.4: General EDT hardware.

a high bandwidth stream of data on the output (to the scan chains), and the compression ratio is seen as the ratio of the output bandwidth to the input bandwidth.

The following discussion sets up the framework for the different directions which have been taken in test data compression research through elaboration of the basic equation of test data decompression:

$$CR = \frac{BW_{out}}{BW_{in}}. \tag{2.1}$$

Equation 2.1 is the very basic relationship between the compression ratio (CR) and the bandwidth (BW) in and out of a decompressor. Since we may write the bandwidth as the product of the bus-width carrying the data and the frequency at which that data is sent, we can obtain:

$$\frac{BW_{out}}{BW_{in}} = \frac{BITS_{out} \times f_{out}}{BITS_{in} \times f_{in}}. \tag{2.2}$$

Now consider that the test set must contain some amount of "useful" data, which cannot be modified by the decompressor and thus remains the same on both the input and output sides. This gives:

$$\frac{BITS_{out} \times f_{out}}{BITS_{in} \times f_{in}} = \frac{f_{out} \times \frac{BITS_{out}}{BITS_{useful}}}{f_{in} \times \frac{BITS_{in}}{BITS_{useful}}}. \tag{2.3}$$

Finally if we write f_r as the frequency ratio between the (CUT) and the ATE ($f_r = f_{out}/f_{in}$), and if we use TCU and SCU to denote tester channel utilization and scan chain utilization respectively, that is, the percentage of data in the tester channels or in the scan chains respectively which is "useful", we obtain:

$$CR = \frac{f_{out} \div SCU}{f_{in} \div TCU} = f_r \frac{TCU}{SCU}. \tag{2.4}$$

We have established that automatic test pattern generation (ATPG) provides test vectors which are not completely specified. This unspecified portion of the information is obviously "useless", and so it can be seen that the measure of *SCU* relates to the concept of care bit density, the percentage of bits in the test set which are useful or take on a specified value (as opposed to being left free as x-bits). In this way the *SCU* value is fixed by the ATPG process and the only free parameters in determining compression ratio are f_r and *TCU*. These two parameters embody the two main directions of research in test data compression. We call those methods which focus on f_r - *temporal compression* methods (Section 2.5) and those which focus on *TCU* - *spatial compression* methods (Section 2.6). In addition to these divisions a distinction can be made regarding what are called *test set dependent* and *test set independent* compression methods. Test set dependent methods exploit the structure of the test set itself (e.g., specific locations and values of care bits) to produce compression, while test set independent methods exploit only very general features (e.g., average care bit density). While the sections just mentioned above will focus on the test set independent methods of their respective approaches, Section 2.4 gives some work done on test set dependent compression methods and a justification as to why test set independent approaches are favorable.

2.4 Test Set Dependent Methods

Somewhat like the a few of the BIST approaches mentioned earlier, some compression methods modify a circuit to ease testing. Unlike the BIST approaches however, this is not so much a case of increasing controllability or observability but more so of increasing correlation between specified bits in an attempt to enable better compression. The improved compression is often enabled by a decompressor set up to exploit such correlations. As an example [75] crafts a decompressor based on the test patterns at hand as do [10, 82], and [94] uses the test set to determine switch connections for a reconfigurable switch based decompressor. A finite state machine (FSM) is used in [63] to drive a combinational expander which is derived from the test set. A linear decompression network is derived from the linear dependencies of the test set by [83], and [5] does the same but includes inversion of the scan outputs when determining the network. In [88] consideration is given to power by reordering the test set and partitioning scan chains based on examination of the test set and [11, 68] construct a scan tree where the output of a scan cell branches to multiple chains and the structure of the tree (i.e. where to branch) is determined by dependencies in the test set. In [74] scan chains are separated into groups by evaluating the test set and determining which groups may be tested using random, periodic or external tests respectively, somewhat like hybrid testing, but across different sections of the circuit instead of across different parts of the test set.

Although test set dependent methods can often be used to obtain better compression results than test set independent methods, they suffer some serious drawbacks. Aside from the IP issues discussed previously with respect to BIST for those methods which change the circuit (e.g. modify scan cell order) there are methods which do not interfere with the circuit itself, however they still do not allow alternate pattern sets to be used. It is very desirable to reuse the existing manufacturing test infrastructure on-chip for embedded debug and diagnosis when tracing a problem in the circuit. Related to this is the application of failure analysis patterns when more detail is required than the simple pass/fail provided by manufacturing test. Further, new tests may need to be added even for manufacturing test to detect an un-anticipated fault (perhaps due to an un-modeled defect) which is resulting in test escapes (devices which pass manufacturing test but which are in reality defective and are caught later in-field). In all these cases, the additional patterns would almost certainly be outside the expansion space of the decompressor since it would have been crafted to contain just the original test set (to increase compression). To avoid all these problems, the practical approach is to design a decompressor which obtains relatively extensive coverage of the entire potential space of test patterns and in addition provide a mechanism whereby any pattern not in the output space of the decompressor may

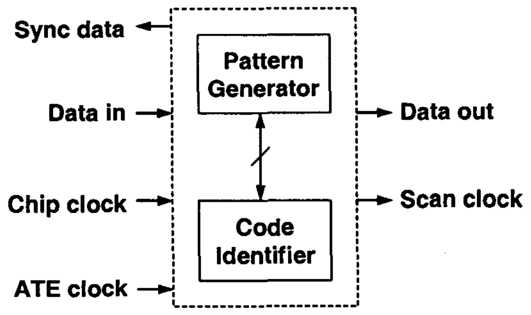


Figure 2.5: General temporal decompressor.

be applied, at reduced compression of course. Such methods are called test set independent and for the reasons just listed will constitute the remainder of the discussion on previous work. The remaining discussion will be in light of temporal and spatial compression, the two main research directions in EDT as outlined in Section 2.3.

2.5 Temporal Decompression

Temporal decompression methods achieve reduction of data by employing a few-to-many mapping in the temporal domain. This means that data sent in a short number of clock cycles from the ATE is expanded for a long number of clock cycles on the CUT. To avoid excessive accumulation of test data in the decompressor temporal methods therefore require that the ATE and the scan chains of the CUT be clocked at different frequencies, hence they focus on increasing f_r as discussed in Section 2.3.

Figure 2.5 shows a very general architecture used for temporal decompression of test data. Because the ATE and CUT scan chains operate at different frequencies, the *chip clock* and *ATE clock* are provided to the decompressor which are used to drive the *scan clock* as shown in the figure. Compressed test data is provided to the decompressor (on *data in* - synchronized to the ATE clock) while decompressed data arrives on *data out*, synchronized to the *scan clock* provided by the decompressor. The *sync data* is required to indicate to the ATE that the decompressor is ready for more data, to balance the arrival of data at the input with its consumption at the output. This is in fact one of the drawbacks of this approach and will be discussed in more detail a little further down.

Temporal decompression in general relies on some manner of statistical coding which assigns short codewords to sequences appearing frequently in the test set. X-bits are mapped in such a way as to increase the occurrence of sequences which map to small codewords and thereby provide better compression. The following details the work which has been done on temporal decompression methods.

In [16, 18], Golomb codes are used for test data compression and comparison is made against conventional run-length encoding, showing the advantages to be higher compression with a scalable, low cost decompressor on-chip. In [15] Golomb coding is also used, taking scan power into consideration. In [17, 21] another class of codes called frequency-directed run-length (FDR) codes are discussed, which improve compression by addressing the low probability of long runs of zeros. [19, 37, 38] also use FDR codes, but focus on partitioning of test resources at the system level. Similar to FDRs are alternating run-length codes presented in [20] which enable switching between coding of runs of 1s and runs of 0s. Alternate run-length encoding is also used by [87] which modifies scan cell order to improve compression. In [109] run-length encoding is used, but longer run-length codes (which are shown to occur infrequently) are stored in an on-chip dictionary to increase efficiency of coding. The Burrows-Wheeler transform is applied by [112] followed by run-length encoding, and in [95] the test set is divided into blocks and compressed using 9 code words one of which contains an explicit uncompressed word (for the case where a block does not fit any of the other 8 codewords).

Huffman coding has also been used extensively in test data compression such as in [39]. Huffman coding is also used by [108], but in the spatial domain (i.e., the ATE and CUT frequencies are the same), causing it to suffer from large area overhead of the decoder. Variable length input Huffman compression (VIHC) is proposed in [27] of which Golomb coding is shown to be a special case and system level considerations for VIHC are given in [28]. Selective Huffman vertical coding (SHVC) is implemented by [113] but they require an embedded FPGA in the SOC, otherwise suffering from large area. In [34] the need for synchronization with the ATE is eliminated at the cost of a large on-chip buffer. By using Tunstall codes, [32] improves error resilience of the test set. Finally, the well known LZW algorithm used in software compression has been applied to the test data compression problem in [52, 107]. In [107] the boundary scan cells are modified to enable low overhead, while [52] reuses on-chip memory for decompression.

One of the main strengths of statistical coding methods for test data compression is the ability to compress highly specified test sets. This makes them particularly suitable for application to IP-protected circuits. Because IP-protection implies that there is no disclosure of internal circuit information, ATPG and fault simulation may not be performed by the system integrator and thus test generation is the responsibility of the core provider while the system integrator must translate the core level tests of each core into an entire system level test. To reduce the amount of data that must be communicated between the core provider and system integrator the core tests may arrive highly compacted. As will be shown later, other approaches suffer loss of compression when the care bit density is high which makes statistical coding favorable in such cases.

On the other hand, the synchronization referred to towards the beginning of this section is a substantial drawback of this approach. Essentially most ATEs are unequipped with any type of reactive functionality, that is they operate in the stream mode and so the aforementioned *sync data* pin shown in Figure 2.5 cannot control the tester. Since codewords vary in length as do the sequences which they represent, a short codeword may arrive representing a long sequence and so the decompressor may not finish expanding a given codeword before the next one arrives. Addressing this synchronization problem leads to additional overhead either in terms of area or compression ratio and more importantly, it complicates the implementation process due to the number of distinct clock domains that need to be considered in the decompressor.

Another difficulty encountered by statistical coding lies in the fact that the scan chains are clocked at a higher frequency. Because the scanning shifts values through all the flip-flops in the scan chains simultaneously, the number of transitions during any clock cycle is greatly increased over the normal transitions occurring in the native mode. As a result, the power rails may be unable to deliver sufficient current to switch a flip-flop, or worse, the chip may not be able to dissipate quickly enough the power

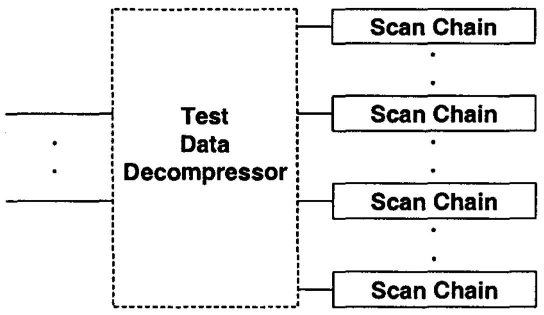


Figure 2.6: General spatial decompressor.

generated from a large number of transitions resulting in overheating and permanent damage. To combat this problem, a substantial amount of work has been done on scan chain partitioning to reduce the power during test. Nevertheless integrating scan chain power management with temporal decompression imposes further implementation constraints, such as additional gated clock domains, not easily satisfiable in practice.

Finally, in the case where the care bit density is not sufficiently high as to adversely affect the operation of other compression methods, such other compression methods (which will be presented next) can provide much better compression. This will be elaborated on later, for now it will suffice to say that the drawback of statistical encoding is that it must assign x-bits then compress the entire test set, whereas the methods addressed next compress only the information carried by the specified bits.

2.6 Spatial Decompression

In light of the above discussion, temporal decompression does not provide a compellingly practical solution to the problem of test data compression, an observation which is reinforced by the fact that (to the authors knowledge) no commercially available tools utilizing statistical based methods for test data compression/decompression are available. This next section details the concept of spatial decompressors which form the basis for the most prominent commercially available test data compression tools. Spatial decompression is enabled by a few-to-many mapping in the spatial domain, that is a small number of channels from the ATE drive a large number of channels in parallel on the CUT during test as shown in Figure 2.6. By doing this, the clock frequency of the ATE and the CUT scan chains may be kept the same and the ATE memory needed may be reduced either by using a smaller number of channels from the ATE, or by driving a larger number of scan chains from the same number of tester channels resulting in reduced test time. Such decompressors come in combinational and sequential varieties, each of which will be described in the following sections.

2.6.1 Combinational Decompression

Combinational decompression occurs when the *Test Data Decompressor* network between the ATE tester channels and the CUT scan chains as shown in Figure 2.6 does not contain any flip-flops, that is it is entirely combinational. In this case each slice of compressed test data from the tester (the data sent in one clock cycle) corresponds directly to the expanded slice of test data in the scan chains for the same clock cycle. There are many options for what may constitute the combinational network. The simplest option is mere fan-outs where multiple scan chains are connected to a single scan input as in [33], which is known as the Illinois Scan Architecture and is depicted in Figure 2.7. It is immediately clear that there is a high correlation in each clock cycle between the scan cells of all scan chains connected to the same scan pin. This high degree of correlation results in a high probability of lockout (conflicting assignments to values in the compressed stream) meaning some patterns may not be testable. For patterns which cannot be satisfied in the *broadcast* mode (as depicted in the figure), a *serial scan* mode is employed where the scan chains are concatenated resulting in a number of scan chains equal to the number of scan inputs. In this case the patterns are filled in directly with no compression. The overall compression therefore relates to the product of the percentage of patterns which may be tested in the broadcast mode and the compression factor for patterns in the broadcast mode. Some work has been done to try and increase the percentage of patterns testable in the broadcast mode, for example [85] allows multiple mappings of the CUT scan pins to the scan chain inputs and [89] extends this work. Another modification of Illinois Scan is [72] which allows a prelude vector to be loaded that modifies the vector being

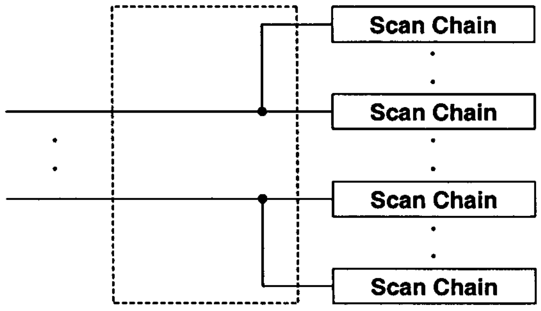


Figure 2.7: Illinois Scan Architecture.

shifted in. Multiple configuration modes between the broadcast and serial modes are added by [53] to reduce the cost in compression of a broadcast-untestable pattern. In any case, what makes Illinois Scan attractive is the very low area overhead and the simple compression algorithm. Unfortunately, despite efforts to decrease lockout, many patterns are often pushed to the serial scan mode and as a result compression suffers.

An alternative to Illinois Scan is known as *dictionary based* compression shown in Figure 2.8. In this approach, the x-bits of the test set are mapped in such a way as to minimize the number of distinct scan slices in the test set. This set of distinct words is stored in an on-chip ROM and indexed by the address lines. In [110] the tester channels are divided into two groups; one to select a dictionary entry (address lines of ROM) and one to flip a single bit of the dictionary output. In this approach the entire set of distinct words need not be stored, only the distinct words which differ by at most 1 position from any given scan slice. The advantage of this method over Illinois Scan is that it reduces the correlation between scan chains, thus the probability of lockout and as a result the compression is improved. The implementation is also relatively straightforward, although more work is involved in choosing a set of words for the dictionary. With respect to area, this method does not perform exceedingly well especially if the number of scan chains is large, since the number of scan chains dictates the word size. The approach in [110] addresses this by placing a combinational network at the output of the ROM, however the drawbacks of this are the same as those of general combinational networks discussed shortly. The dictionary based method is also inherently test set dependent since the set of words for

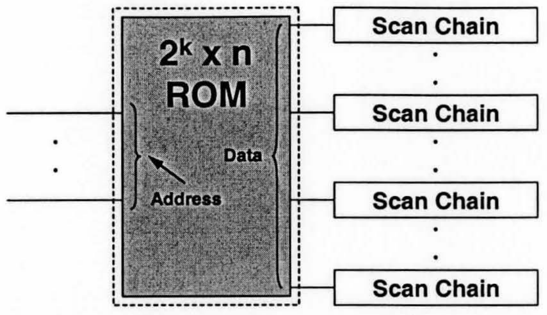
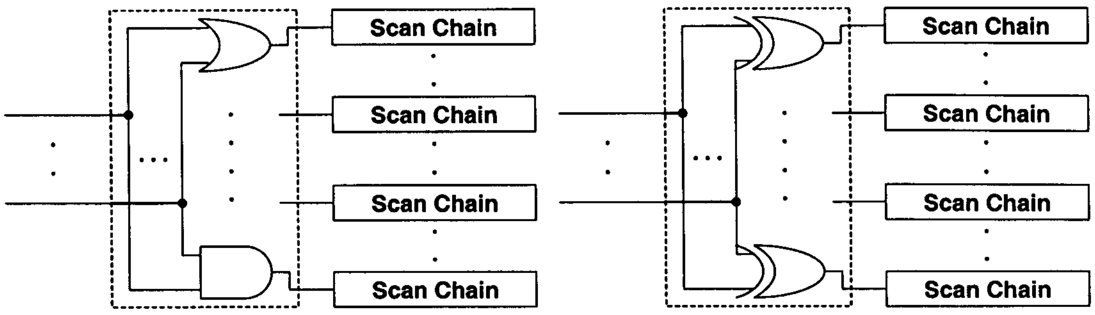


Figure 2.8: Dictionary based decompression.

one test set may not work efficiently with another test set. The reason these methods were not addressed at the same time as the test set dependent methods is that a case can be made for test set *in*dependent methods when the dictionary is reconfigurable, for instance if a RAM is used instead of a ROM. Allowing reconfigurability also addresses the area problem since a smaller dictionary may be used and updated during test, dividing the entire test into multiple sessions with a different dictionary configuration for each session. Although this leads to smaller area, reconfiguration becomes a test-time bottleneck increasing the cost of test.

Extending the concept of the dictionary decompressor, a general combinational circuit may be used as the decompressor as shown in Figure 2.9(a). This approach is in essence equivalent to the ROM based method but with smaller area since the logic synthesis will optimize the network to remove redundant logic which is not the case in a ROM. This method also suffers from test set dependence (since the network is generally crafted according to the test set), and in addition from computational complexity. In exchange for the area that is saved in comparison to the dictionary, the satisfiability problem must now be solved to determine the compressed stream. Satisfiability solves the problem of what input values applied to the decompressor will justify the required output values. For even a modest number of scan chains this problem becomes intractable, however this step makes the case for the next type of decompressor described.

Figure 2.9(b) shows a special case of the combinational decompressor discussed in the previous paragraph where the only type of gate used is the XOR gate. This method is indicative of those such as [66]. One drawback of this approach is that use



(a) General combinational decompressor. (b) Linear combinational decompressor.

Figure 2.9: Combinational decompressors.

of XORs results in linear dependencies between channels, increasing the probability of lockout. Lockout is addressed in [9] through the use of integrated ATPG/fault simulation and extremely short scan chains. The approach in [60] also uses linear decompressors but splits the scan chains into groups and clocks each group separately so that the linear decompressor may decompress to all scan chains simultaneously or just one group, thereby reducing lockout. Although the use of XORs results in linear dependencies, the solution to this particular case of the satisfiability problem can be obtained in polynomial time, since the network may be modeled by equations in GF_2 (each XOR is a modulo-2 addition) and Gaussian elimination can be used to obtain input sequences which will justify the required output scan slices. In addition, use of generic logic makes the issue of test set independence difficult to address whereas the concept of test set independence is more clear for linear decompressors. For these reasons, XOR based decompressors have become the most popular type of combinational test data decompressor.

2.6.2 Sequential Decompression

In the previous section we have discussed combinational decompressors, and area and computational complexity considerations have led us to conclude that decompressors based on linear networks (XOR gates) provide the best tradeoff between area, compression and computational complexity. Given that compressed data in each clock cycle is obtained by observing which scan channels contain care bits for that clock cycle, and equating the linear expressions for those channels each to the value of their

Average care bits per block		**Maximum**
Single scan clock	1 5 3 7 5 3 4 6 2 3 4 7 1 1 2 8 6 2 1 3	8
Block of 2	3 5 4 5 2.5 5.5 1 5 4 2	5.5
Block of 4	4 4.5 4 3 3	4.5

Figure 2.10: Averaging of care bits in a test pattern.

respective care bit, it can be seen that in each clock cycle there is an equation only for each care bit. Thus in each clock cycle a system of equations is formed in terms of the variables representing the input channels from the tester, having a number of equations equal to the number of care bits for that clock cycle. Since we desire compression, the number of tester channels (number of variables) should be less than the number of scan chains (potential number of care bits per clock cycle). Thus it is immediately clear that a situation may arise where the number of equations is greater than the number of variables, often resulting in a linear inconsistency and therefore lockout. In order to reduce the probability of lockout more variables may be provided, however this requires more tester channels and as a result adversely affects compression. In [54], a proof is given that a system of n equations in m variables has a probability of lockout $< 10^{-6}$ when $m > n + 20$. Because in general we use large m and n we assume that so long as $m = n$, a lockout will not occur. Nonetheless, provision is made to deal with lockout even though it is rare.

Given the reasoning just provided we can consider that the number of tester channels required for a linear combinational decompressor to ensure sufficient variables arriving from the tester is given by the maximum number of care bits in any clock cycle (frame) for the entire test set. Consider now the fact that care bits are not uniformly distributed within the test set. Figure 2.10 shows how averaging over a few consecutive frames can reduce the maximum demand for free variables from the tester. In the case where the frames are considered independent of one another (first row) the maximum care bits appearing at a time is 8, and thus 8 variables are required per frame (clock cycle). It is important to note that the channels are physically allocated to the chip during test and as a result if they must be allocated even

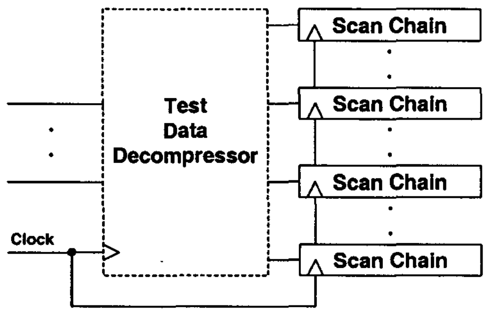


Figure 2.11: General sequential decompressor.

for one frame, they must be allocated for the entire test set. As a result, during the rest of the test set the channels are under-utilized more free variables are provided than necessary and so there is room for improvement in compression. Observe the second row of the figure which indicates the average number of care bits per frame which exist when a block length of two is used, that is, the test set is split into blocks of two consecutive frames and considered block by block. Now, (taking the ceiling function of 5.5) 6 variables are needed every frame indicating that no block contained more than 12 care bits. Likewise when four frames are considered at a time (third row), now a maximum of only 5 ($\lceil 4.5 \rceil$) variables are required in each clock cycle. Averaging across a number of clock cycles reduces the demand for variables from the tester and so frames which have a high number of care bits may borrow some variables from an adjacent frame which contains fewer care bits.

In light of this discussion on averaging it is clear that we desire to lend/borrow variables arriving from the tester across multiple clock cycles, which requires the use of some sequential elements. Figure 2.11 shows the generic sequential decompressor. The presence of the clock signal indicates the use of memory elements (flip-flops) to store accumulated free variables. Note that although any type of generic sequential circuitry may be placed in the decompressor, machines based on XOR networks modeled by linear equations in GF_2 are favored for the same complexity reasons considered for combinational decompressors, especially in light of the fact that sequential satisfiability is of even greater complexity than combinational satisfiability. The remainder of this section will discuss the existing solutions in this category.

As mentioned above, providing sequential elements in the decompressor enables the storing of variables to be shared among a block of frames, that is, the care bits in a block need not be considered as existing in separate frames, only as being being a part of a particular block. The most straightforward approach to compressing the data is to consider the data on a block by block basis and encode the data needed for the care bits of each block. Figure 2.12 shows a general architecture which uses this scheme based on the STUMPS configuration [7]. The compressed data comes in the form of an initial state (seed) for the LFSR which is loaded at the beginning of a block. During the clock cycles corresponding to that block the LFSR progresses from one state to the next producing a pseudo-random sequence which matches the original test vector in all the care bit positions. The purpose of the XOR network between the LFSR and the scan chains (called a phase shifter) is to remove shift correlations. Without a phase shifter the pseudo-random sequences generated from adjacent cells in the LFSR are shifted by only a single clock cycle, and are thus very much correlated. The introduction of a phase shifter produces a greater shift between these channels. The seed is computed in a similar way as for the linear combinational decompressor discussed in the previous section, where each scan chain for each time frame in the block (instead of simply each scan chain) has an associated equation. By assigning the equation for each care bit position to its respective value, a system of equations can be obtained and solved to determine the required seed. The *Buffer* shown in the figure captures data from the tester so as not to disturb the operation of the LFSR while a seed is provided from the tester for the duration of the block. The seed is then loaded into the LFSR at the beginning of the next block. It is this buffer which enables solution of the blocks independently. This architecture exactly is used by [105] to map each pattern onto a seed, and ATPG and fault simulation are integrated to cover as many faults as possible for each pattern and merge multiple patterns on a single seed (to reduce the number of patterns). This work is further elaborated in [104] to give more consideration to output response compression. An interesting take on this approach is to encode the number of clock cycles required to reach the next LFSR seed instead of the seed itself, as proposed in [3].

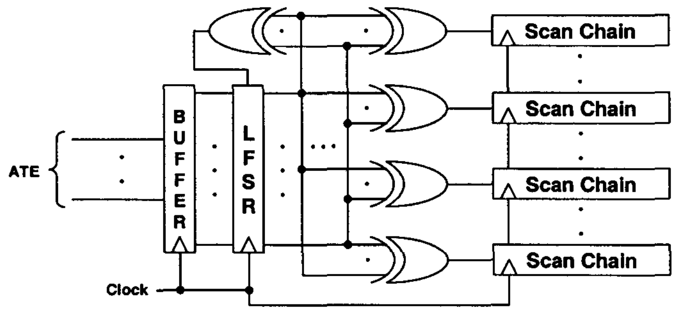


Figure 2.12: Seed buffering architecture.

We have seen that the care bit density dictates the number of equations and therefore the number of variables needed for acceptable probability of lockout, while the number of variables used indicates the achievable level of compression. When care bit density is increased more equations appear and linear dependencies cause either pattern lockout resulting in loss of fault coverage, or decreased compression. A couple of methods addressing linear dependencies have been proposed such as [50] which adds multiplexers on either side of the phase shifter to provide reconfigurability (at the expense of increased area and computational complexity), and [49] which uses a multiple polynomial LFSR (different feedback polynomials may be selected) of variable size by dictating the location of least significant bit of feedback polynomial (through configuration). Both of these approaches incur some extra area overhead and involve increased complexity in the calculation of compressed streams, but enable compression of test sets with higher care bit densities.

The common feature of the methods just addressed is the accumulating of variables separate from the operation of the LFSR and transfer of those variables to the LFSR once, at the beginning of a block. This means that variables are not correlated in any way from one block to another and for sufficiently large buffers the probability of lockout is very small [54]. Unfortunately, such large buffers come at the expense of impractical on-chip area and so smaller buffers must be used. The use of smaller buffers leads either to increased probability of lockout or more severe over allocation of resources since the averaging period is reduced. An alternative to buffering all the variables for each block is to provide the variables to the LFSR in a continuous manner, a few in every clock cycle instead of many during one clock cycle. A general

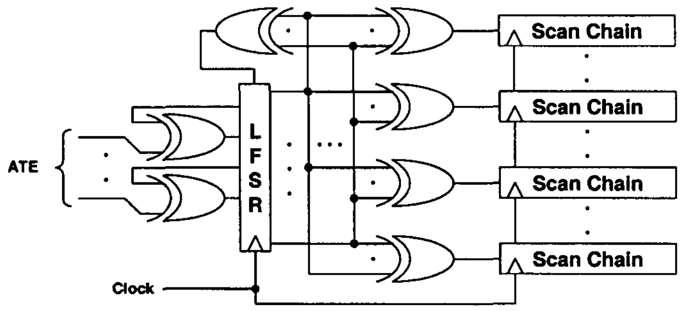


Figure 2.13: Variable injection architecture.

depiction of such a method can be seen in Figure 2.13. This approach is used in [58, 61] both of which inject the variables coming from the ATE during each clock cycle in the feedback of the LFSR. The new variables arriving provide more degrees of freedom in each clock cycle for solving the systems of equations and thereby satisfying the care bits of the test set. Both [71, 78] also work on this principle of injecting free variables, however they use ring generators instead of LFSRs. Ring generators are claimed to operate at higher frequencies because of smaller fan-out of gates and shorter signal propagation distances, however the principle of operation and seed solution methods are the same for all intents and purposes. Ring generators are explored further in [69, 80]. The injection approach is also used in [41] which connects a few scan chains to an LFSR and the injection input to the LFSR is seen as a virtual scan chain. In a similar way, [102] splits scan chains into smaller scan chains and provides a shift register and XOR broadcast network but allows the chains to be concatenated into a single long chain (reconfiguration) to deal with lockout. Injection is also used by [45, 46] where some encoding of the injection stream is proposed. In [91], injection to an LFSR is used and a dictionary is added for frames which cannot be expanded from the LFSR.

The advantage to variable injection as opposed to variable buffering discussed earlier is that the on-chip area is much smaller since large buffers are no longer needed. In addition, the variables injected remain in the decompressor (in linear combination with each other) for a much longer time due to the fact that in the injection method the decompressor continues cycling and accumulating more variables while in the buffering method, the variables are flushed each time a new block is started. In

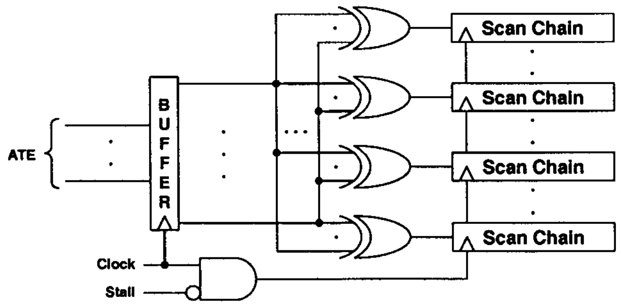


Figure 2.14: Implementation of stalling.

practice this corresponds to a longer averaging period since variables are alive for longer. At the same time, this fact that the variables are alive for longer results in larger systems of equations needing solution when calculating seeds, and thus run-time may increase significantly. Also, when bursts of care bits arrive there may not be sufficient free variables accumulated to deal with the added equations and lockout may still occur.

To address the problem arising from bursts of care bits, a couple of similar methods have been proposed which are represented in Figure 2.14. By halting the shift process in the scan chains, free variables can continue to accumulate in the decompressor until enough free variables can be provided to the circuit to overcome the lockout. This approach is used in [57] to achieve good compression at relatively low area overhead. This method is also used by [100] which enables the use of a smaller LFSR, however the area is increased since they use it in concert with the buffering approach described earlier. In this case fewer channels are allocated than are required to ensure sufficient free variables may be provided during the worst case block (the block with the most care bits). Relying on the fact that these blocks occur infrequently they stall shifting of the scan chains to fill in the buffer when lockout occurs in the middle of a block. Although this method is attractive due to low area overhead gating of the scan clock is a methodology issue because routing of the scan clock is sensitive, and gating may cause clock skew. Furthermore, test time may be adversely influenced if bursts of care bits are frequent or too little tester bandwidth is allocated. This will also have a negative impact on the compression factor.

2.7 Multi-level Compression

In addition to the research directions overviewed above there is a small class of multi-level compression methods which have been proposed. Multi-level compression compresses the test patterns onto compressed streams but then applies compression (usually a different type) to the compressed streams. An example of this is [59] which first maps care bits onto seeds as in general LFSR based compression and then applies statistical coding to compress the LFSR seeds. This however suffers from the same challenges discussed earlier with temporal compression methods. The LFSR seeds are also compressed in [65] but using folding counters. The approach in [40] delivers compressed weight sets for weighted pseudo-random BIST. In [55, 56], weighted pseudo-random pattern testing is also used with compressed weights sent from the tester. In [56], the weight set decompressor is a RAM/ROM connected to a multiple input signature register (MISR) which allows it to act like a PRPG. All of these methods exploit is some sort of structure in the compressed form of the test set to achieve compression. Both [55, 56] attribute the compressibility largely to clustering of care bits within a test set. This observation of clustered care bits also enables the work presented in the next chapter of this thesis. Instead of complicating the decompressor by adding a second level of compression to the incoming data, we have aimed to address care bit clustering at the first level of compression to improve the VTD. The methods mentioned above also suffer from increased test time because of reliance on pseudo-random testing, whereas we try also to improve the test time. In addition to these facts, multi-level compression can always benefit from improving the compression in one of its layers and so our method can also be used in conjunction with multi-level compression to further enable delivery of test data to large scale circuits with low cost testers.

2.8 Summary

This chapter provided a survey of the recent advancements in embedded deterministic test and it was established that spatial decompression based on linear decompressors is the most suitable approach for large scale integrated circuits. Nevertheless, given the continuing increase in circuit complexity and the new emerging design methodologies based on the core reuse philosophy, adapting spatial decompression to core based SOCs using IP-protected blocks faces its own challenges, as shown in the following chapter where a novel approach is introduced.

Chapter 3

Compression in Core Based SOCs

In this chapter, the main contribution of this work is presented, a compression methodology for core based SOCs with IP-protected blocks. In Section 3.1, the desired features of the methodology are set up as objectives. Exploitation of key observations given in Section 3.2 has led to the development of the central scheme described in Section 3.3. This is followed by incremental advancements presented in Sections 3.4 and 3.5. Finally, Section 3.6 will give some experimental results and discuss the relation to other approaches and Section 3.7 will summarize.

3.1 Desired Features

In light of discussion in Chapters 1 and 2, we may summarize here the five key features we desire in a compression scheme for core based SOCs with IP-protected blocks. They are as follows:

- **IP-consistency**: We desire a method that does not require access to the internal details of the core to provide compression, thus fault simulation and ATPG should not be relied upon (Section 3.1.1);
- **Modularity**: In order to facilitate reuse, the EDT method should be easily pluggable to any system and furthermore, the hardware generation should be easily automated (Section 3.1.2);

- **Scalability**: To remain relevant in the long term, the hardware architectures employed and the algorithms used should scale reasonably with circuit size (Section 3.1.3);

- **Programmability**: Any test pattern should pass through the decompression engine and changes to the test set should not influence its compressibility so long as the general features of the test set (i.e., care bit density) remain intact (Section 3.1.4);

- **Low Cost Test Platform**: No special requirements should be made about the ATE and what features it has, we should assume a very low cost basic tester which operates only in the stream mode without any sequencing-per-pin options (Section 3.1.5).

3.1.1 IP-consistency

We have seen that the SOC design paradigm facilitates design reuse by enabling construction of complex systems through the assembly of pre-designed modules called cores. This brings with it some unique test challenges as some cores may be IP-protected in which case no internal details of the circuit are available. IEEE 1500 standard has been developed to ensure core test reuse and isolation during test [35], however no provisions were made yet for test data compression. Many compression methods rely heavily on integrated fault simulation/ATPG to obtain small volumes of test data and these methods are un-applicable to IP-protected cores. As shown in Figure 3.1, methods which require fault simulation restrict compression to the domain of the core provider. This poses a problem if some core providers choose not to provide compression hardware, or if the method of compression they use does not suit the system level test methodology. By creating the decompression engine without the need for IP disclosure, freedom to perform compression on either the core provider or system integrators side is enabled. This gives the system integrator the ability to perform compression in such a way as to facilitate packing of the compressed streams at the system level. Moreover, compression can still be done by the core provider as long as the compression method used fits the system level methodology.

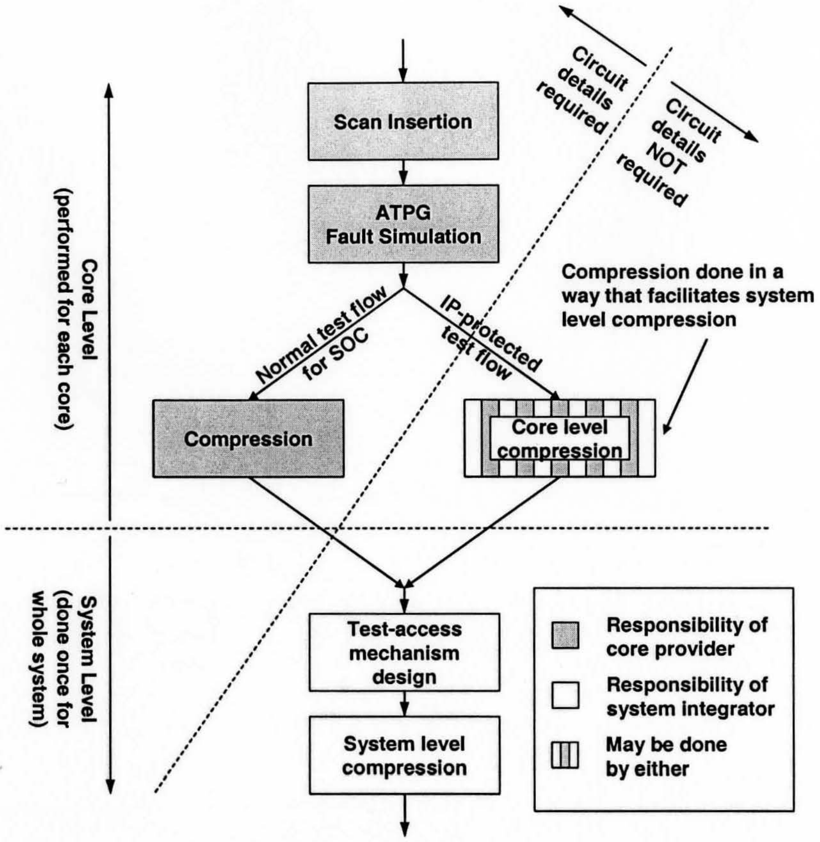
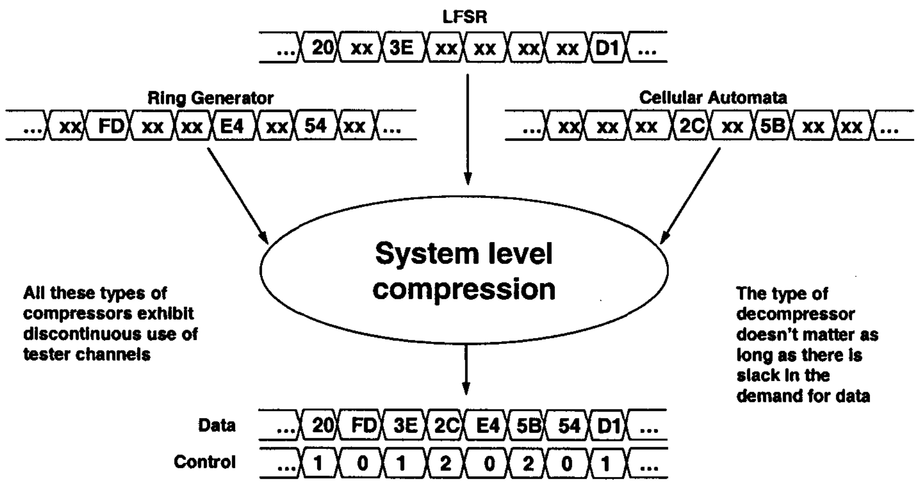

Scan Insertion
ATPG Fault Simulation
Normal test flow for SOC
IP-protected test flow
Compression
Core level compression
Compression done in a way that facilitates system level compression
Circuit details required
Circuit details NOT required
Core Level (performed for each core)
System Level (done once for whole system)
Test-access mechanism design
System level compression
Responsibility of core provider
Responsibility of system integrator
May be done by either

Figure 3.1: IP-consistency.

3.1.2 Modularity

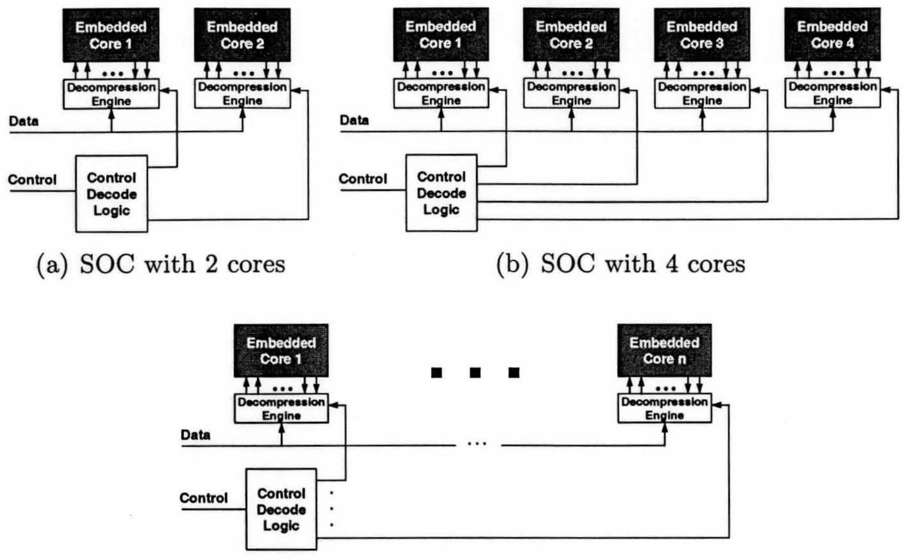
In order to remain as flexible as possible we want to be able to use cores with a number of different core level compression hardware types, as long as they share the same interfaces required to be plugged into in the same system level architecture. Of course this is closely linked with the IP-consistency issue, since the system integrator will most likely opt to use the same decompressor type system wide if they have access to all the information of all cores. If however, some IP protected cores come with decompression hardware already in place, some cores may use LFSRs, some ring generators, some cellular automata, etc. To achieve a modular approach a common feature of all the decompression hardware can be exploited in order to ensure an easy plug-n-play into the system level decompression architecture. For example, as it is the case in this thesis, the discontinuous demand of data from the tester can be used


LFSR
...20 xx 3E xx xx xx xx D1 ...
Ring Generator
...xx FD xx xx E4 xx 54 xx ...
Cellular Automata
...xx xx xx 2C xx 5B xx xx ...
System level compression
All these types of compressors exhibit discontinuous use of tester channels
The type of decompressor doesn't matter as long as there is slack in the demand for data
Data ...20 FD 3E 2C E4 5B 54 D1 ...
Control ...1 0 1 2 0 2 0 1 ...
Figure 3.2: Modularity.

as the common feature of the core level decompression hardware which can be used by a system level test engine to improve utilization of the tester channels, and thereby improve compression (Figure 3.2).

3.1.3 Scalability

To remain applicable in the long term, scalability needs to be taken into account. Even if the hardware and computational overhead are acceptable for current day devices, if they do not scale effectively they cannot be used in the future. Figure 3.3 shows how the hardware should be kept relatively low, even when many cores are used. Since the core level decompressor size is linked to the size of the core it accompanies, the area overhead for this hardware will be kept from scaling to an unreasonable level. At the system level, some small control logic is required. The size of the control logic should scale roughly logarithmically with the number of cores, and thus the area for this hardware will remain reasonable as well. In terms of computation, the execution time for core level compression should scale linearly with the number of cores since it must be performed for each core, while system level compression must also remain practical even as the system level design size increases.



(a) SOC with 2 cores (b) SOC with 4 cores

(c) SOC with n cores

Figure 3.3: Scalability.

3.1.4 Programmability

Programmability is of great importance when forming a test compression methodology since changes to the test set (modified patterns, extra patterns) may be required. By keeping the hardware as generic as possible and providing a multi-mode feature, *any* test set desired may be passed through the decompressor. Although the compression will suffer as care bit density is increased, through multi-mode configuration the decompressor retains the ability to pass any pattern through (Figure 3.4).

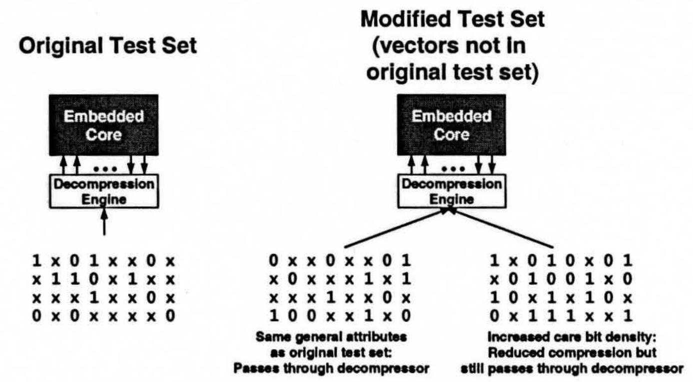


Figure 3.4: Programmability.

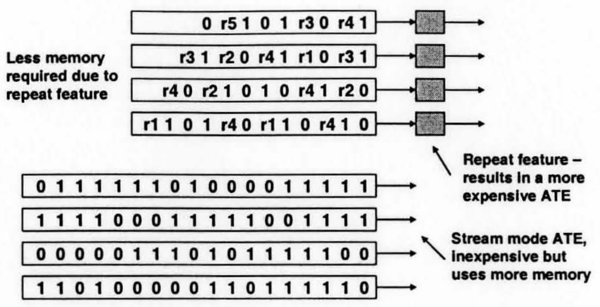

0 r5 1 0 1 r3 0 r4 1
r3 1 r2 0 r4 1 r1 0 r3 1
r4 0 r2 1 0 1 0 r4 1 r2 0
r1 1 0 1 r4 0 r1 1 0 r4 1 0
Less memory required due to repeat feature
Repeat feature – results in a more expensive ATE
0 1 1 1 1 1 1 0 1 0 0 0 0 1 1 1 1 1
1 1 1 1 0 0 0 1 1 1 1 1 0 0 1 1 1 1
0 0 0 0 0 1 1 1 0 1 0 1 1 1 1 1 0 0
1 1 0 1 0 0 0 0 0 1 1 0 1 1 1 1 1 0
Stream mode ATE, inexpensive but uses more memory

Figure 3.5: Low cost test platform.

3.1.5 Low Cost Test Platform

Some issues relating to ATE features used for compression were discussed in Section 2.1. Because of the higher cost of testers with special features we desire a compression methodology which does not rely on ATE features to improve compression, by allowing the ATE to operate in a simple stream mode as illustrated in Figure 3.5. The upper half of the figure shows how a repeat per pin feature can reduce the amount of data which needs to be stored, but the additional processing capability of each channel increases the cost of the tester. On the other hand, if the compression method used is suited to a simple stream mode tester as in the lower half of the figure, more memory may be required (which can be compensated by increased compression) but the tester will be much less expensive. Furthermore, if a tester with repeat features is already available it may be used in conjunction with the on-chip compression method, that is, it can be applied to the system level compressed streams to further reduce the volume of test data stored on the tester.

3.2 Key Observations

Figure 3.6 shows the most basic observation which enables this work. This figure illustrates the fundamental ways in which multiple cores in an SOC can be tested. In part (a) all the cores are given independent tester channels so that they may be accessed and tested in parallel. Although this leads to favorable test times, the number of tester channels required is large which results in a large volume of test data and may conflict with the number of device pins available. In part (b), the number

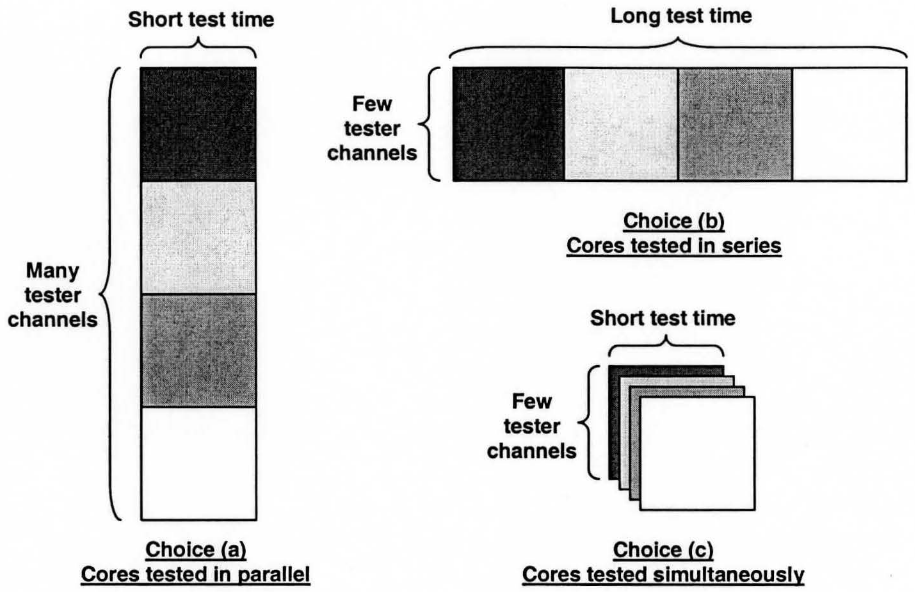
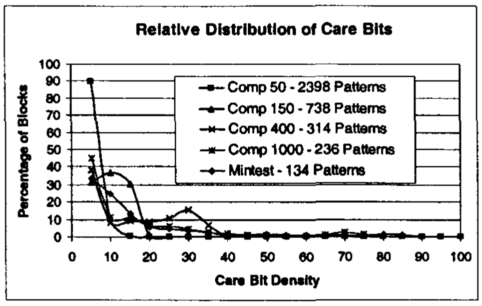


Figure 3.6: Choices for SOC test application (each square = 1 core).

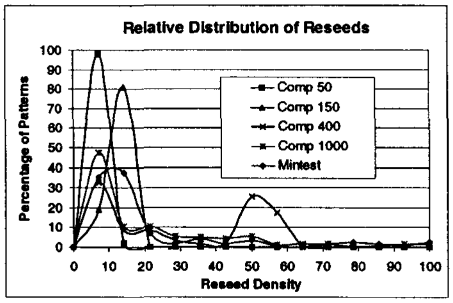
of tester channels required is much fewer as one set of channels is allocated and used by the cores in sequence, i.e., the cores are tested in series. However, the test time now becomes a problem and the volume of test data is still large. Both problems are solved through part (c) of the figure, where a small number of tester channels are allocated and the cores are tested *simultaneously.* This is only possible if the cores being tested do not demand 100% exclusivity of the tester channels. In other words, cores may only be tested simultaneously provided that for each core there are times during which the core does not need the channels, an issue addressed in the next paragraph.

In order to exploit the observation that simultaneously tested cores use fewer tester channels *and* require less time to test (reducing VTD), the test sets of the cores must satisfy the condition that the cores do not require 100% bandwidth from the tester during 100% of the testing time. Figure 3.7 shows empirically how this is enabled (notes: the test set used here is from the ISCAS'89 Benchmark set [12], and *Comp-(number)* gives a relative average care bit density of the test set, both will be discussed further in Section 3.6). In Figure 3.7(a), a test set has been broken into blocks[1] which

[1]A block is a set of consecutive clock cycles and will be further detailed in Section 3.3.



(a) Care bit density distribution



(b) Reseed density distribution

Figure 3.7: Non-uniform distributions enabling improved compression.

are binned according to care bit density, and the number of blocks in each bin is plotted on the vertical axis against the care bit density of that bin. It is immediately clear that a large percentage of the blocks have a very low care bit density while only a few have a very high care bit density. Recalling from Section 2.6 that there is a direct correlation between care bits and seed bits required from the tester, we would expect a correlation between the care bit density and the required density of reseeds. This correlation can be seen in Figure 3.7(b). This figure was obtained in a similar way to Figure 3.7(a), each pattern from the test set was broken into blocks and as many blocks as possible were satisfied from a single seed, so some patterns were reseeded frequently (those with high care bit densities) and some reseeded infrequently (those with low care bit densities). The patterns were then binned according to number of reseeds per pattern and graphed as number of patterns against number of reseeds. As expected, nearly all of the patterns have very few reseeds while only a few require reseeding in every block. Because the tester operates in the stream mode the test data in the time during which reseeding is not required is wasted, i.e., the tester channels are under-utilized. *It is this fact that we exploit: during the time in which one core does not require the seed channels for reseeding, we use the same tester channels to reseed another core and thereby improve the tester channel utilization, which leads to reduced VTD.*

Since the intuition behind what has been done may seem obvious, some of the obstacles which needed to be overcome in order to enable this approach are discussed here along with an overview of how we have addressed these challenges. The first and most obvious challenge is the design of a hardware architecture which can support this time-multiplexing of tester channels. Because the cost in area should be kept to a minimum, the hardware should be relatively simple yet it must be robust and modular so that it can be generated in an automated way. This desire for automation brings to light the other main challenge of enabling this work, the CAD support. In addition to being able to generate efficient hardware structures automatically, we want an integrated flow into which we can dump the test specifications and then collect our compressed data which we can transfer directly to the tester. In order to cope with ever increasing volumes of test data, human involvement in managing design and test flows should be kept to a minimum. On the hardware front, ideas have been borrowed from the previous work in compression with modifications to suit the time-multiplexing proposed. With this hardware in mind an algorithmic framework for compression at the core level has been developed which we call *StreamBIST*, and a pattern reordering algorithm which we call *StreamPack* for merging of the core level compressed streams has been tackled at the system level. In the next section, these hardware, core level CAD and system level CAD solutions are discussed in more detail.

3.3 Seed-only Packing

This section introduces the central contribution of this thesis. It presents a novel hardware architecture for EDT and the CAD support at the core and system levels that will fully leverage the proposed hardware for improved compression. It is important to note seed-only refers to sharing only the seeds transferred from the tester in contrast to sharing the mux lines used by a more elaborate decompressor introduced in the next section. Before getting into the details of the decompressor, we define the term *frame* to mean the data in a slice across all the scan chains during one clock cycle. Said another way, during one clock cycle, a single frame of the test pattern is loaded into the scan chains. We next define a block as being a collection of consecutive frames. Figure 3.8 illustrates these definitions.

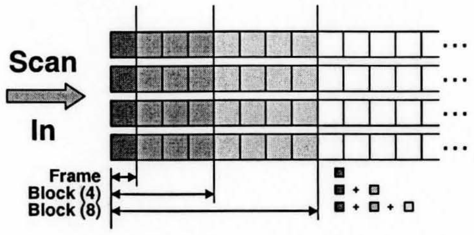
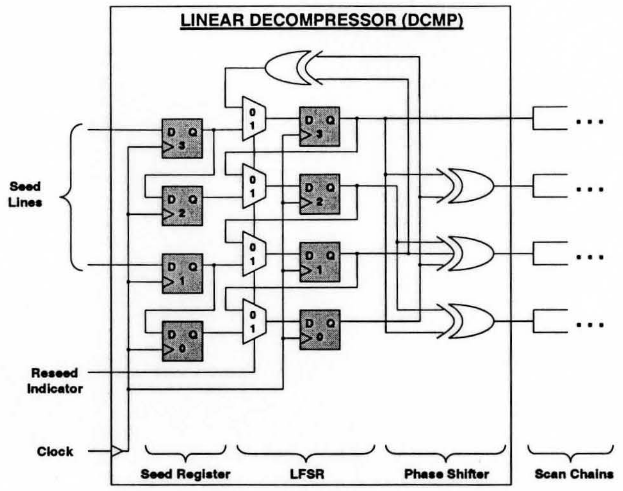


Figure 3.8: Illustration of frames and blocks.

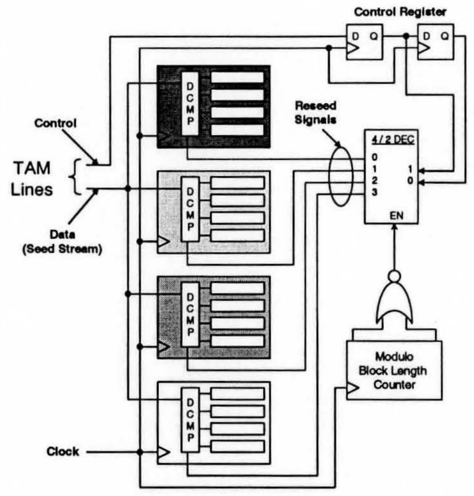
3.3.1 Seed-only Hardware

Figure 3.9 shows the hardware proposed for time-multiplexing of tester channels for seed-only sharing. Figure 3.9(a) shows the decompressor at the core level, based on the same architecture used in [105]. In the duration of a block, a seed is shifted into the *Seed Register* on the *Seed Lines.* In the clock cycle where the *Reseed Indicator* is asserted, the seed in the seed register will be loaded into the *LFSR.* For any clock cycle where the reseed indicator is not asserted, the LFSR will continue to cycle and produce a pseudo-random sequence through the phase shifter. Thus, for any given block, there is an option to pick up a new seed or to continue expanding the next block from the previous seed, thus allowing some slack to be introduced in the demand for tester channels by a single core. As a note, work done in [6] has sought to establish theoretical limits in compression through the use of entropy. Their work shows that LFSR based approaches such as [105] which compress a fixed size block of data onto a fixed size seed approach the compression limit determined by the test set entropy. Where our approach is subtly different from [105] is the compression of *variable* sized blocks of test data onto fixed size seeds (enabled by the reseed indicator line) for which determination of entropy is left as an open problem by [6]. It is from this variable to fixed encoding that the slack arises, and as the next paragraph describes it is this slack that enables improved compression.

In Figure 3.9(b) we see the system level architecture which exploits the slack created at the core level, by trading off demand for the tester channels among the cores. Note that the small blocks with "DCMP" inside refer to the core level decompressors of Figure 3.9(a). The architecture is similar in concept to the Illinois Scan Architecture [33] discussed in Section 2.6.1, however some control must be added to regulate



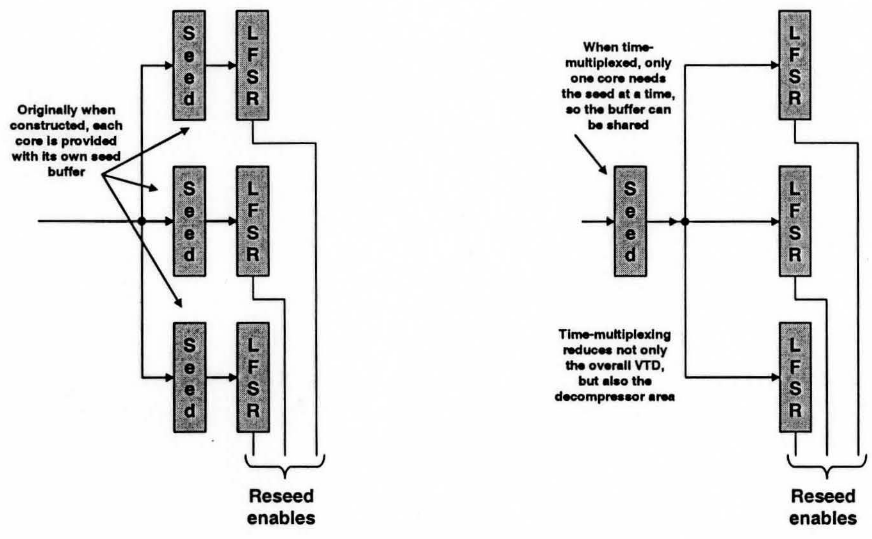
(a) Core level decompressor.



(b) System level architecture.

Figure 3.9: Decompression scheme for seed-only packing.

which core is using the tester channels during any given block. As shown in the figure this can be accomplished relatively simply, a control register is used to buffer the control stream and a periodic reseed signal is generated in every block. Depending on what value is in the control register when the reseed signal arrives, the decoder will allow only one core to use the buffered seed during that block. On this point a note should be made about the buffering of seeds. Figure 3.10(a) shows how the decompressors constructed at the core level would be linked at the system level (only the seed portion has been shown for clarity). Because time-multiplexing is used only one core will reseed during a given block and the other cores will ignore the seed, and thus the buffer may be shared at the system level as shown in Figure 3.10(b). This would not be possible if the cores were tested in parallel so not only does time-multiplexing reduce the volume of test data required, it also reduces the area of the decompressor. Note that in the following two sections, the architectures are given with the seed buffer near the core level decompressor for illustrative purposes, however, the same principle applies to those methods as well. It can be seen that the hardware used in this compression methodology is not exceedingly elaborate, especially at the system level. This simplicity of the hardware involved however is misleading, such simple hardware is only made possible by sophisticated CAD support as discussed next.


Originally when constructed, each core is provided with its own seed buffer
Seed
LFSR
Reseed enables
When time-multiplexed, only one core needs the seed at a time, so the buffer can be shared
Time-multiplexing reduces not only the overall VTD, but also the decompressor area

(a) One seed buffer per decompressor.

(b) Seed buffer shared at the system level.

Figure 3.10: Transformation from buffering seeds at the core level to the system level.

3.3.2 Seed-only CAD Support - Core Level

The CAD flow for compressing test sets for an SOC onto this decompression architecture begins with calculation of seeds at the core level. Before discussing seed calculation, we will first cover hardware modeling. Figure 3.11 shows explicitly how the model equations for a given hardware configuration are obtained. Since the XOR gate is equivalent to addition in GF_2 the operations are linear and may be represented as matrix manipulations. The LFSR feedback polynomial determines which LFSR FFs are used to calculate the value of the most significant bit of the LFSR in the next clock cycle, and as such becomes the top row of the LFSR transition matrix (shown in the figure). Since the remainder of the LFSR is a simple shift register, each remaining row in the matrix is simply the basis vector corresponding to the immediately adjacent LFSR cell. This matrix therefore can be used to compute the next LFSR state from the current state, and of course multiplication of the current state with the n^{th} power of the transition matrix will give the state after n clock cycles. It follows that the LFSR matrix is a square matrix with number of rows and columns equal to the LFSR size.

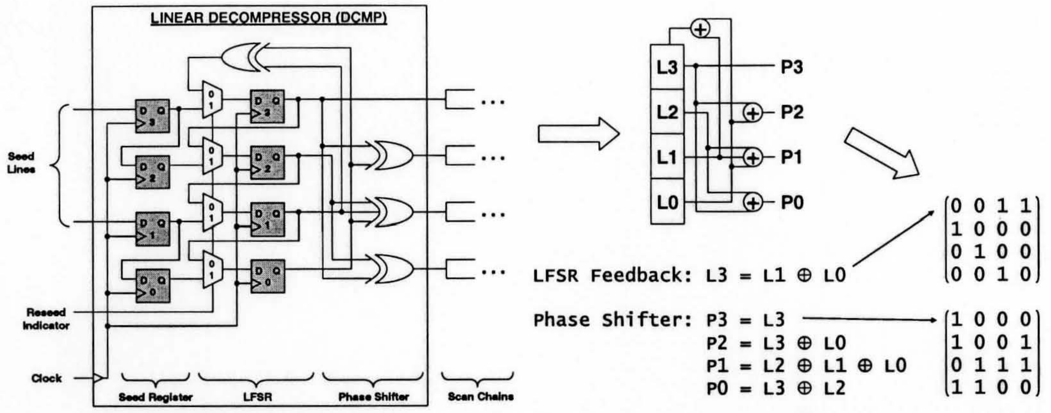


Figure 3.11: Modeling of LFSR and phase shifter in GF_2.

Moving past the LFSR we find the phase shifter. Since each phase shifter output is a GF_2 linear combination of the LFSR outputs, we may model this with a matrix as well. Each column of the matrix corresponds to an LFSR output and each row to a phase shifter output. For the example in Figure 3.11, the bottom output of the phase shifter is $L3 \oplus L2$ and thus the bottom row of the phase shifter matrix is *1 1 0 0*. Although in this example the phase shifter matrix is square, this is not necessary. In general, the number of columns = LFSR size, and the number of rows = number of scan chains (typically larger than LFSR size).

Having discussed how the hardware is modeled it is natural to wonder how the test cubes are mapped onto equations corresponding to the hardware. Figure 3.12 shows the process of extracting equations from a test cube and solving them to obtain a seed (initial state of the LFSR - compressed test data). The upper left portion of the figure shows the GF_2 model of the hardware as derived in the previous paragraph. Immediately to the right of this the matrix multiplication is given explicitly which allows calculation of the outputs of the phase shifter from a given seed $(a_3a_2a_1a_0)$ provided at time $t = 0$. Since we know how to calculate the decompressor outputs at a given time, we simply match the specified bits in the test cube to the equations (in terms of the seed variables) for the same outputs/times. Down the left part of the figure the matrix products have been calculated for the given times. This then gives a matrix which when multiplied with the seed vector, will give the decompressor output for that time frame. The lower middle section of the figure shows matching specified

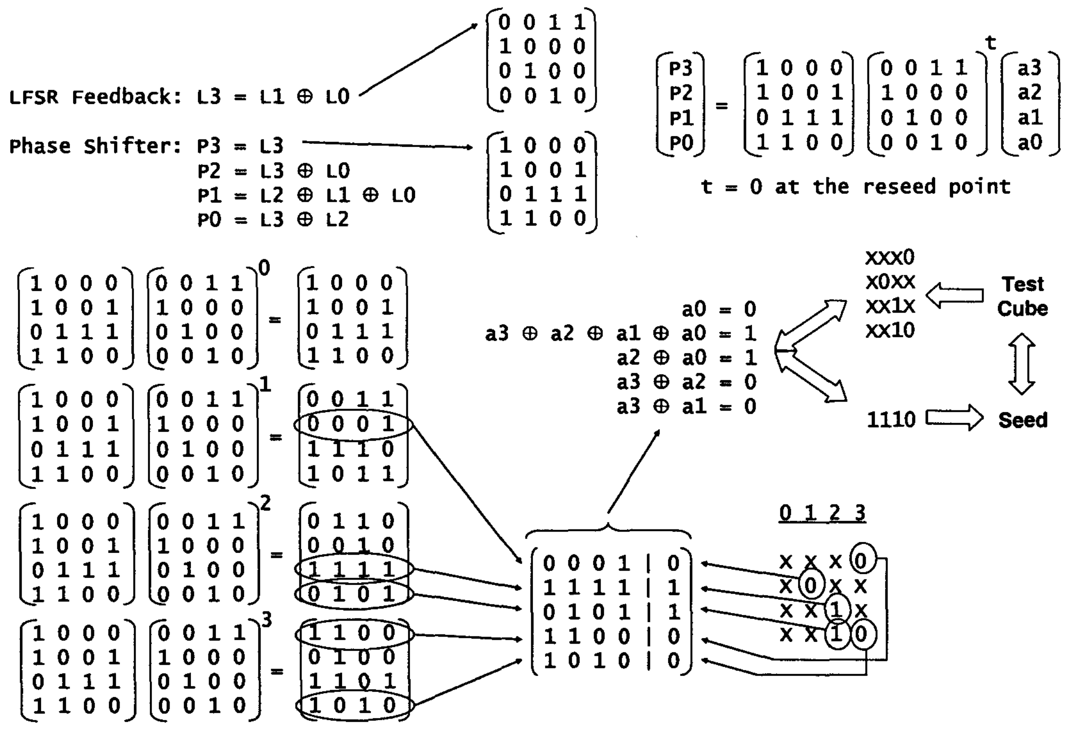


Figure 3.12: Generating equations for a test cube.

bits to the corresponding equations to obtain the system of equations in the center of the figure. Applying Gaussian elimination to this system yields values 1,1,1,0 for a_3, a_2, a_1, a_0, i.e., the seed. Thus we can conclude that if we seed the decompressor shown in Figure 3.11 with the seed 1110, the expanded pseudo-random sequence will match the test cube shown in Figure 3.12 in all care bit positions.

Coming back to the clustering feature of test sets as discussed in Section 3.2, this paragraph will outline how the basic seed calculation procedure just described can be modified to produce "packable" compressed sequences for each core. The basic procedure (named *StreamBIST*) is described by Algorithm 1 where Table 3.1 may be consulted for symbol descriptions. As its input, the algorithm takes a set of test cubes and a set of parameters describing the test hardware setup, that is, what size LFSR should be used, what characteristic polynomial, how many scan chains, the length of the scan chains, the block size, etc. Using this information, the function **GenerateCoreHardware** constructs the hardware and generates an HDL (VHDL

Variable Name	Representation
P_i	The i^{th} test pattern
$test\_cubes$	The set of all P_i
$Compressed\_P_i$	The set of seeds and ressed locations for P_i
$parameters$	The set $\{Block_{size}, Num_{SC}, SC_{len}, ...\}$
$NULL$	The empty set
B_k	The k_{th} block of the test pattern
$Block_{size}$	The length of a block
$Block_{count}$	Indicates the starting block of the current block chain
$Chain_{count}$	Indicates the number of blocks in the current block chain
$prev\_equations$	The accumulated equations of a block chain
$new\_equations$	Equations of the most recent block added to a block chain
$Chain_i$	Group of cores tested simultaneously
IN_i	Patterns still waiting to be placed for core chain i
OUT_i	Patterns that have been placed for core chain i
$NOOP$	An empty block requiring no reseeding

Table 3.1: Terminology for StreamBIST and StreamPack algorithms.

Algorithm 1: StreamBIST

```
Input  : $test\_cubes$, $parameters$ (see table 3.1)
Output: $compressed\_stream_{core}$, $incompressible_{core}$, $decomp\_hardware_{core}$
1 $decomp\_hardware_{core}$ = GenerateCoreHardware($parameters$);
2 $compressed\_stream_{core}$ = NULL;
3 $incompressible_{core}$ = NULL;
4 foreach test pattern $P_i$ in $test\_cubes$ do
5     $compressed\_P_i$ = CompressPattern($P_i$, $parameters$) // Algorithm 2;
6     if ($compressed\_P_i \neq NULL$) then
7         Add $compressed\_P_i$ to $compressed\_stream_{core}$;
      else
8         Add $P_i$ to $incompressible_{core}$;
      end
  end
9 Return $compressed\_stream_{core}$, $incompressible_{core}$, $decomp\_hardware_{core}$;
```

and/or Verilog HDL) description ($decomp\_hardware_{core}$) to which the core can be connected at the system level. Actual compression begins in the next step, as the algorithm progresses through the test set one pattern at a time attempting to compress each one. For each pattern that is compressible with the given hardware configuration, the compressed data or seed ($compressed\_P_i$) is stored in the overall compressed stream for the core ($compressed\_stream_{core}$). At the end, a list of all incompressible patterns ($incompressible_{core}$) is provided in addition to the compressed stream and the decompression hardware description. Note that the presence of incompressible patterns will negatively affect fault coverage since the faults in the incompressible patterns which cannot be applied may not be covered by other patterns. In this case the designer has the option to increase the size of the LFSR and thus decrease the probability of an incompressible pattern however, this results in poorer compression. This has motivated the modifications to the architecture which will be discussed in Sections 3.4 and 3.5. In the next paragraph we will focus on the details involved in compressing a single pattern, the **CompressPattern** function which is actually the core of the compression engine, where the "packability" of patterns is augmented.

Algorithm 2 gives the core of the compression procedure for a single pattern which augments slack in the demand for seed bits, thus facilitating the packing algorithm which will be used later to merge the compressed streams of all the cores at the system level. The pattern to be compressed (P_i) is first divided into blocks according to the block size determined by the designer ($Block_{size}$). The basic idea behind making patterns easier to pack is reducing the number of reseed points required, that is, expanding as many blocks as possible from the same seed. To do this blocks are checked for compatibility and strung together in a chain (if compatible). The counter $Block_{count}$ keeps track of the starting point of the current chain, and the counter $Chain_{count}$ keeps track of how many blocks are in the current chain. Let us examine the algorithm in detail with an example.

Example 1 *Figure 3.13 illustrates the process of compressing a pattern using Algorithm 2. The test pattern shown in the figure has been divided (line 1) into blocks with $Block_{size} = 4$ (upper right portion of the figure). Line 2 assigns $Block_{count} = 1$ and the outer WHILE loop (line 3) is entered. Next, loop conditions for the inner*

Algorithm 2: CompressPattern

```
Input  : $P_i$, $parameters$
Output: $compressed\_P_i$
1  Divide $P_i$ into $Num_{blocks}$ blocks ($B_k$) of length $Block_{size}$;
2  $Block_{count} = 1$;
3  while ($Block_{count} \leq Num_{blocks}$) do
4      $prev\_equations$ = NULL;
5      $compressible$ = TRUE;
6      $Chain_{count} = 0$;
7      $k = Block_{count} + Chain_{count}$;
8      while (($k \leq Num_{blocks}$) AND ($compressible$)) do
9          $new\_equations$ = GenerateEquations($B_k$, $parameters$);
10         $compressible$ = CheckEquations($prev\_equations$, $new\_equations$);
11         if ($compressible$) then
12             $prev\_equations = prev\_equations \bigcup new\_equations$;
13             if ($k = Num_{blocks}$) then
14                 $Seed_k$ = CalculateSeed ($prev\_equations$);
15                 Add $Seed_k$ to $compressed\_P_i$;
16                 Assign ressed point at $Block_{count}$ in $compressed\_P_i$;
               end
17             $Chain_{count} = Chain_{count} + 1$;
18             $k = Block_{count} + Chain_{count}$;
           else
19             if ($Chain_{count} \neq 0$) then
20                 $Seed_k$ = CalculateSeed ($prev\_equations$);
21                 Add $Seed_k$ to $compressed\_P_i$;
22                 Assign ressed point at $Block_{count}$ in $compressed\_P_i$;
23                 $Block_{count} = k$;
               else
24                 $compressed\_P_i$ = NULL;
25                 $Block_{count} = Num_{blocks} + 1$;
               end
           end
       end
   end
26 Return $compressed\_P_i$;
```

WHILE loop (line 8) are set up. Inside the inner WHILE loop we begin with generating the equations for block number k - B_k (line 9). Since at this point $Block_{count} = 1$, $Chain_{count} = 0$ and $k = Block_{count} + Chain_{count}$, $k = 1$, we generate equations for the first block. Equations are generated in the same way as the example given above in Figure 3.12 in fact the B_1 in this figure is the same block from Figure 3.12, so the call **GenerateEquations**$(B_1, parameters)$ *would follow precisely the steps outlined in Figure 3.12 (with the exception of seed calculation since we intend to string multiple blocks together if possible). In line 10 the consistency of the equations is checked by* **CheckEquations** *which determines whether there are any linear incompatibilities among the equations in new_equations and prev_equations. Since we are on the first block of a chain at the moment, prev_equations is still empty so we are checking for linear inconsistencies in just the equations of B_1. Since the equations for this block (as shown in Figure 3.13) are consistent,* **CheckEquations** *returns TRUE to compressible and so the first branch of the IF statement on line 11 is taken. On line 12, the equations of the block are merged to the rest of the equations in the chain (again, this is currently the first block in the chain so after line 12 prev_equations will contain only the equations for B_1). Line 13 checks if we are currently at the last block, and if so assigns a seed since there are no more blocks to chain. Lines 17 and 18 update $Chain_{count}$ to 1 and k to 2 respectively. We now begin the second iteration of the inner WHILE loop. The equations for B_2 (since $k = 2$) are obtained as shown in the figure beside the equations for B_1. Only the first equation for B_2 has been given in the figure since this equation causes an linear inconsistency. As a result of the inconsistency, the second branch of the IF on line 11 is taken. The check at line 19 is to ensure that this is not the first block of the chain. If this were the case, the equations for a single block would be inconsistent and thus the pattern would be incompressible (and thus a NULL would be returned by the algorithm). Since $Chain_{count} = 1$ at this point, the first branch is taken. At line 20* **CalculateSeed** *uses Gaussian elimination to obtain the reduced set of equations (not including those of the most recent block) and thereby the seed. As the figure shows, (uppermost two streams) the resulting seed is assigned to the compressed stream and a reseed point is asserted to accompany the seed. Also, $Block_{count}$ which indicates the starting position* *of the current chain, is updated to block B_2, the block which could not be added because of a linear inconsistency to prev_equations.*

Because at this point compressible is false, the inner WHILE loop exits and a second iteration of the outer WHILE loop is performed. This is illustrated as the second row of systems of equations (middle of the three) in the figure, beginning with $a_3 \oplus a_2 = 1$. It should be noted that this equation is different from the equation for the same care bit when that block (B_2) was in the previous chain - $a_2 \oplus a_1 \oplus a_0 = 1$. This is because in the last chain B_2 was the second block, so the LFSR power (t from figure 3.12) for the associated care bit would have been 4. Now, B_2 is the first block in the chain, so t = 0 for the care bit in question (bottom left 1 of B_2). Moving on to the next iteration of the inner WHILE loop we have the next set of equations in figure 3.13 corresponding to block B_3, where $Block_{count}$ = 2 and $Chain_{count}$ = 1. Since the equations remain consistent, we move to the next block B_4 with $Block_{count}$ = 2 and $Chain_{count}$ = 2. Finally for $Block_{count}$ = 2 and $Chain_{count}$ = 3 (B_5), an inconsistency arises. The seed is calculated (line 20) and passed to the compressed stream at position 2 ($Block_{count}$). Notice that the seed is calculated based on prev_equations which at this point contains equations for B_2, B_3 and B_4. Next the outer WHILE loop iterates again, with $Block_{count}$ = 5. This process continues until either an inconsistency arises in a single block ($Chain_{count}$ = 0) in which case the pattern is incompressible, or until all the blocks have been covered in which case the compressed stream is returned (compressed_P_i).

By applying this method to every pattern in the test set, a compressed stream containing gaps during which the seed lines are not needed can be formed for each core. The next section details the CAD support at the system level which reorders the patterns from the multiple cores to enable sharing of a single set of tester channels among all the cores, by interlacing periods of activity on the seed lines for one core with periods of inactivity for the other cores.

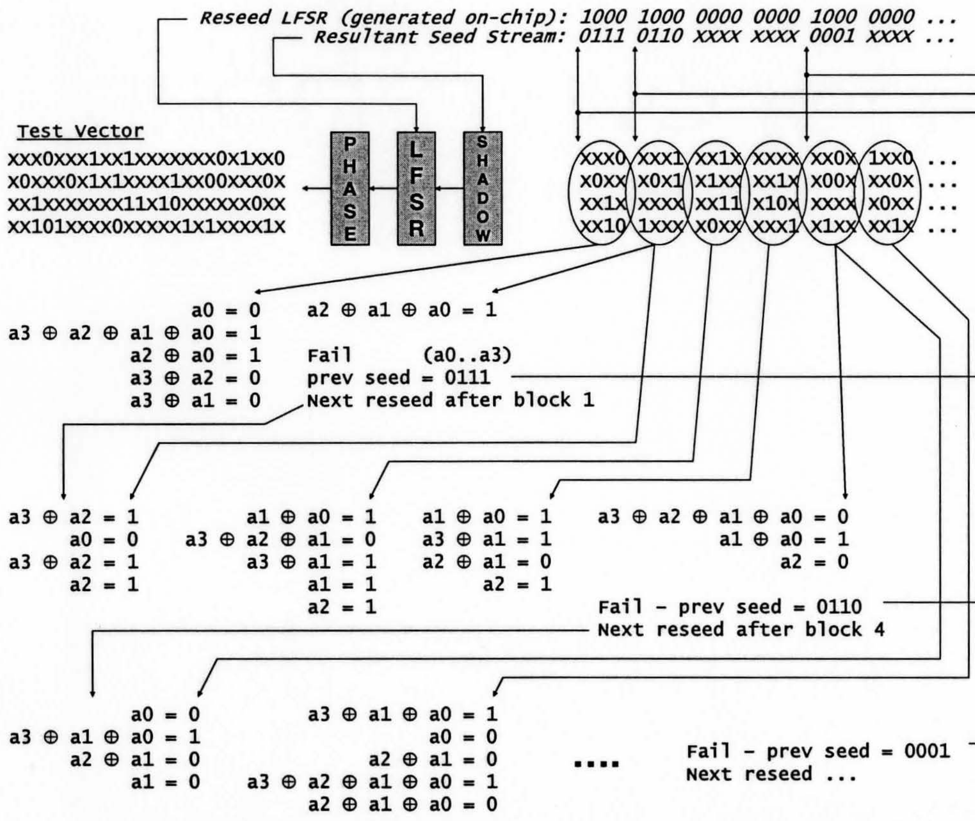


Figure 3.13: Calculation of seeds for StreamBIST.

3.3.3 Seed-only CAD Support - System Level

After performing StreamBIST on all the cores in the system, a collection of core level decompressors (*decomp_hardware*$_{1,2,...}$) and accompanying compressed streams (*compressed_stream*$_{1,2,...}$) are obtained. The process of merging all this information into a system level test architecture and data stream (named *StreamPack*) is illustrated in Figure 3.14. As with StreamBIST, the hardware is kept relatively simple and the complexity of the problem is handled by concentrating effort on stream calculation which is done in software (CAD tools). The StreamPack algorithm for reordering patterns to enable seed line sharing (time-multiplexing) is given as Algorithm 3. In addition to each cores' StreamBIST results, some system level detail (*core_connect*) is provided which indicates how the cores are connected to the test infrastructure. This simply permits cores to be grouped into chains, so that instead of all cores in

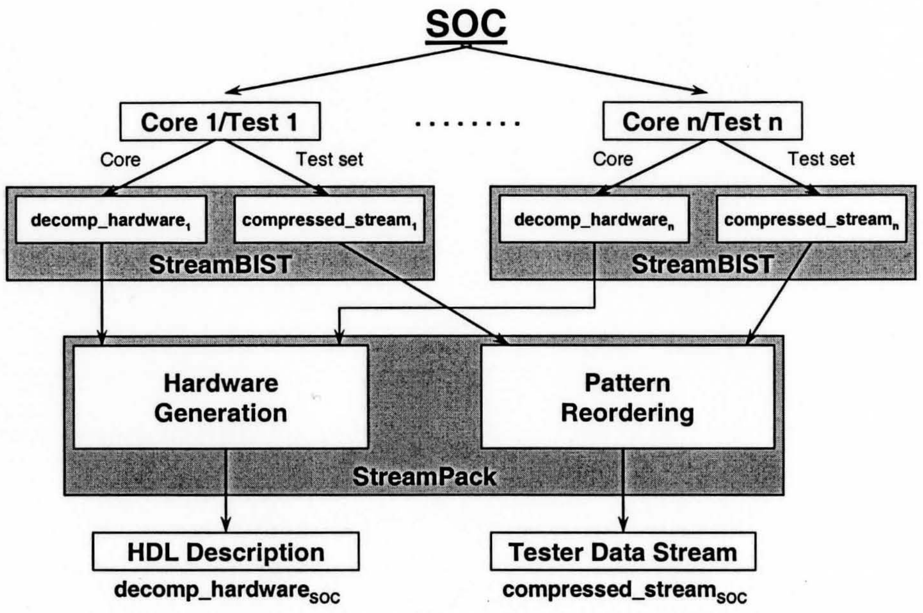


Figure 3.14: Compression flow for StreamPack.

the SOC being tested simultaneously (as in Figure 3.6(c)), each chain is tested simultaneously with all other chains while cores within a single chain are tested serially (as in Figure 3.6(b)). This can be useful if for example there are certain constraints on what cores may be active at the same time, but the simplest and most frequently used configuration is one core per chain. This feature also enables smooth integration of multi-mode StreamPacking as will be discussed in Section 3.5.

The first step in StreamPack is to generate the system level hardware which connects all the individual core level compressors. This is the purpose of the **GenSOCHW** function cited in the algorithm which acts in essentially the same way as the **GenerateCoreHardware** function from StreamBIST (Algorithm 1), the structure of the hardware is generated based on the information from the cores and *core_connect* and an HDL description is automatically generated. After this step the actual packing begins. First, each chain (as discussed above) is analyzed to determine which cores it contains after which all the patterns from all the cores in $Chain_i$ are loaded into a single list called IN_i, i.e., there is one IN list for each chain. Once all the patterns have been loaded, the patterns in each IN list are sorted in descending order of number of reseeds. The intuition behind this step is that patterns with more reseeds will be

Algorithm 3: StreamPack

```
Input  : $compressed\_stream_{1,2,...}$, $decomp\_hardware_{1,2...}$, $core\_connect$
Output: $compressed\_stream_{SOC}$, $decomp\_hardware_{SOC}$
1  $decomp\_hardware_{SOC}$ = GenSOCHW($decomp\_hardware_{1,2...}$, $core\_connect$);
2  foreach $Chain_i$ in $core\_connect$ do
3      foreach $Core_j$ in $Chain_i$ do
4          foreach $Pattern_k$ in $compressed\_stream_j$ $(Core_j)$ do
5              Read reseed profile for $Pattern_k$ into list $IN_i$;
           end
       end
6      Sort patterns in $IN_i$ by decreasing number of reseed points;
7      List $OUT_i$ = NULL;
   end
8  while Some $IN_i$ still contains patterns do
9      $Current\_OUT$ = Shortest $OUT_i$ | $IN_i$ still contains patterns;
10     $Current\_IN$ = $IN_i$ which corresponds to $Current\_OUT$;
11     $Fit$ = FALSE;
12     while $(Fit \neq TRUE)$ do
13         $Current\_pattern$ = First pattern in $Current\_IN$;
14         $Fit$ = FALSE;
15         while $((Fit \neq TRUE)$ AND (not end of $Current\_IN$)) do
16             $Fit$ = CheckFit($Current\_pattern$, $OUT_{1,2,...}$);
17             if $(Fit = TRUE)$ then
18                 Move $Current\_pattern$ from $Current\_IN$ to $Current\_OUT$;
               else
19                 $Current\_pattern$ = Next pattern in $Current\_IN$;
               end
           end
20         if $(Fit \neq TRUE)$ then
21             Insert $NOOP$ block in $Current\_IN$;
           end
       end
   end
22 {$Data$,$Control$} = ParseLists($OUT_{1,2,...}$,$decomp\_hardware_{SOC}$);
23 $compressed\_stream_{SOC} = Data \bigcup Control$;
24 Return $compressed\_stream_{SOC}$, $decomp\_hardware_{SOC}$;
```

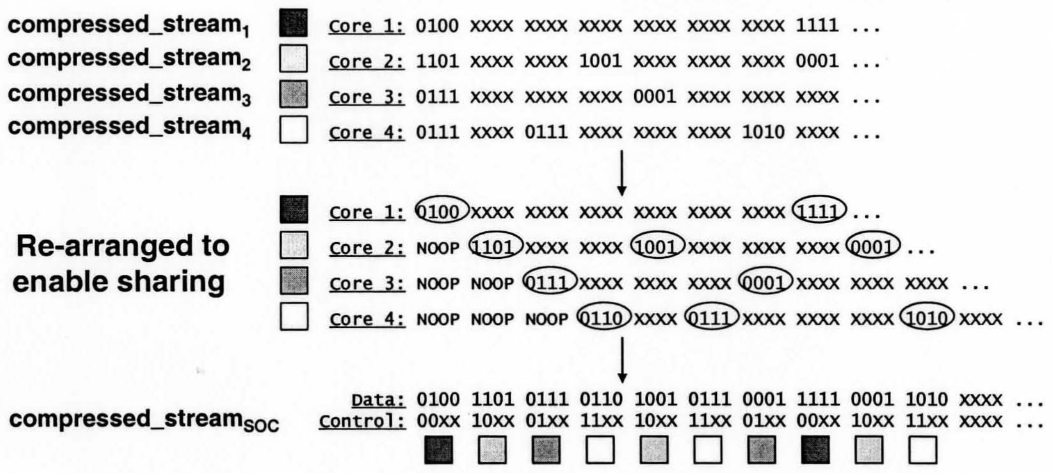

compressed_stream$_1$
Core 1: 0100 xxxx xxxx xxxx xxxx xxxx xxxx 1111 ...
compressed_stream$_2$
Core 2: 1101 xxxx xxxx 1001 xxxx xxxx xxxx 0001 ...
compressed_stream$_3$
Core 3: 0111 xxxx xxxx xxxx 0001 xxxx xxxx xxxx ...
compressed_stream$_4$
Core 4: 0111 xxxx 0111 xxxx xxxx xxxx 1010 xxxx ...
Re-arranged to enable sharing
Core 1: 0100 xxxx xxxx xxxx xxxx xxxx xxxx 1111 ...
Core 2: NOOP 1101 xxxx xxxx 1001 xxxx xxxx xxxx 0001 ...
Core 3: NOOP NOOP 0111 xxxx xxxx xxxx 0001 xxxx xxxx xxxx ...
Core 4: NOOP NOOP NOOP 0110 xxxx 0111 xxxx xxxx xxxx 1010 xxxx ...
compressed_stream$_{SOC}$
Data: 0100 1101 0111 0110 1001 0111 0001 1111 0001 1010 xxxx ...
Control: 00xx 10xx 01xx 11xx 10xx 11xx 01xx 00xx 10xx 11xx xxxx ...

Figure 3.15: Packing of seed streams and generation of control.

more difficult to pack, and since the lists are traversed in order when trying to pack we desire to pack the more difficult patterns early, to leave more packing flexibility later in the packing process. The core of the algorithm which consists of moving patterns from the IN lists to the OUT lists is discussed in the next paragraph.

The basic principle of merging seed streams is shown in Figure 3.15. Each of the compressed streams in the upper part of the figure has been generated by StreamBIST which has produced discontinuous data requirement on the seed lines. By shifting the patterns with respect to one another an arrangement can be found where there are no seed conflicts, only one core requires the seed lines at a time. In Algorithm 3, the WHILE loop on line 8 ensures that the algorithm continues as long as there are unplaced patterns and line 9 chooses the chain in which the next pattern will be placed to the be the shortest chain which still has unplaced patterns (the shortest OUT_i such that IN_i is not empty). The purpose of this is to ensure even treatment among the chains. If the patterns for one chain were fully assigned before the others, there would be too much rigidity to effectively assign patterns to the remaining chains. Starting with the WHILE loop on line 12 we come to the core of the pattern reordering algorithm. Because in reality patterns are much more difficult to pack than in the simple example in Figure 3.15, we must provide a means to deal with seed sharing conflicts. Figure 3.16 shows how this core part of the algorithm deals with these conflicts.

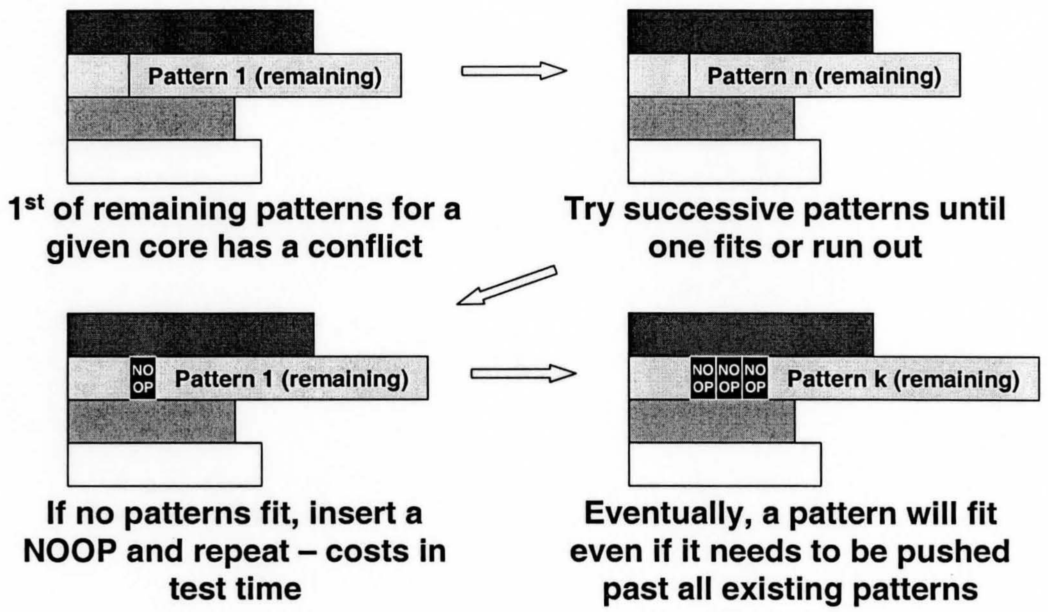
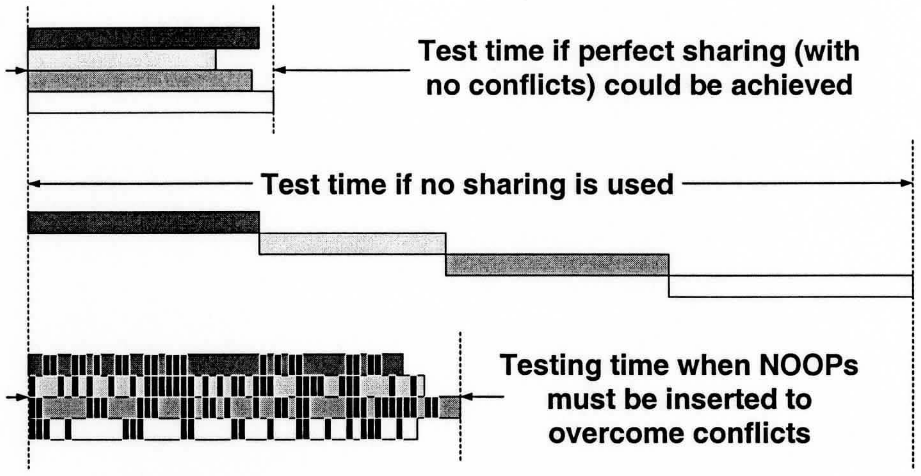

Pattern 1 (remaining)
1st of remaining patterns for a given core has a conflict
Pattern n (remaining)
Try successive patterns until one fits or run out
NO OP
Pattern 1 (remaining)
If no patterns fit, insert a NOOP and repeat – costs in test time
NO OP NO OP NO OP
Pattern k (remaining)
Eventually, a pattern will fit even if it needs to be pushed past all existing patterns

Figure 3.16: NOOP insertion to address reseed conflicts.

Starting at the upper right hand corner of Figure 3.16 and moving to the right, we can see the operation of the WHILE loop on line 15 where all the un-placed patterns in the placement chain (*Current_IN*) are tried in the current placement slot (*Current_OUT*). Each pattern is evaluated against the already placed patterns by **CheckFit** to see if there are any seed conflicts. If a seed conflict arises, the next pattern is tried and so on. If all the patterns have been tried and none fit, the WHILE loop on line 15 exits and a NOOP is inserted (line 21) as shown in the lower left portion of Figure 3.16. A NOOP is a single block on one chain where no seed is provided, which guarantees that it will not conflict with any other chain. The WHILE loop on line 15 then repeats and all the patterns are tried after the inserted NOOP, meaning they have been shifted by one block and therefore might no longer be in conflict with the already placed patterns. The process repeats with NOOPs continually being added until finally there are no seed conflicts. This is guaranteed to happen since eventually the NOOPs will push the placement location past the end of all the previously assigned patterns. Once a pattern fits it is transferred from its *IN* list to the corresponding *OUT* list and, as mentioned above, this continues until all the *IN* lists are empty.


Test time if perfect sharing (with no conflicts) could be achieved
Test time if no sharing is used
Testing time when NOOPs must be inserted to overcome conflicts
Figure 3.17: Test time comparison for StreamPack.

Although insertion of NOOPs does a good job of solving the seed conflict problem, they adversely affect test time by delaying the placement of vectors whenever conflicts occur. This effect can accumulate significantly as shown in Figure 3.17. In the case where perfect packing can be achieved (i.e. there are no seed conflicts at all), the test time as shown in the upper section of the figure is obtained. In contrast to this, the middle section of the figure shows the test time required if the cores were tested serially. The test time resulting from the insertion of NOOPs is longer than the case where no seed conflicts arise, but it is still significantly better than serial testing. Note that all three cases in Figure 3.17 use a single set of seed lines, the test time for parallel testing is the same as for no seed conflicts, but independent tester channels would be required for each chain. Thus even with NOOP insertion, StreamPack is able to provide improved VTD.

Once pattern reordering has been completed it is a very straightforward job to merge the seed streams and calculate the control streams. This step is given on lines 23 and 24 of Algorithm 3, and is illustrated in the bottom part of Figure 3.15. The hardware generated by StreamPack and the compressed test data/control stream thus provide the end product of the StreamBIST/StreamPack flow. In the next two sections, some modifications to StreamBIST and StreamPack are discussed which enable reduction of lockout probability, thereby reducing the number of incompressible patterns while avoiding substantial cost in compression.

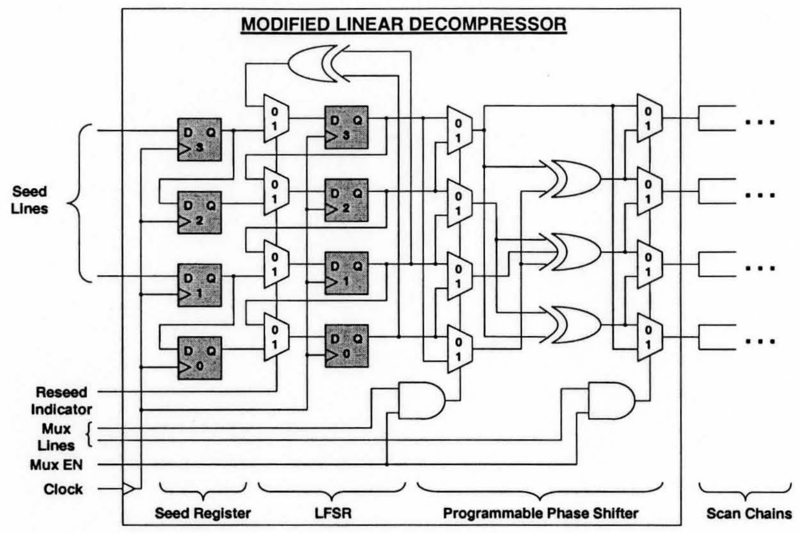


Figure 3.18: Core level decompressor for seed and mux packing.

3.4 Seed and Mux Packing

As extensively covered thus far in the thesis, the use of linear machines as test data decompressors is largely a result of the reduced complexity of seed calculation enabled by the use of Gaussian elimination. Unfortunately, this benefit comes at the cost of increased probability of lockout caused by correlations (linear dependencies) between different points in the test set. If one of these linear dependencies is violated by a given test pattern, that pattern cannot pass through the decompressor. In order to cope with these linear dependencies, the most immediately obvious solution is to introduce some non-linearity into the output sequence of the decompressor to enable compression of an extended set of patterns. That is precisely the approach that has been used in the *seed and mux packing* architecture, described in this section.

3.4.1 Seed-mux Hardware

Figure 3.18 shows the architecture proposed for reduced probability of lockout. Notice the similarity to the seed-only architecture of the previous section, the only difference being the addition of multiplexers on either side of the phase shifter network. The multiplexers on the input side of the phase shifter (input muxes) enable connection of each phase shifter output either to its respective LFSR output or the adjacent LFSR output. Likewise on the output (output muxes), each scan chain may be driven either by its respective phase shifter output or the adjacent one. The *Mux Lines* allow selection of what configuration in which to place the phase shifter. The end result of this additional hardware is increased flexibility in what equations may be used when calculating a seed. Each configuration value on the mux lines corresponds to a different phase shifter, accompanied by a distinct phase shifter matrix. Thus if equations in one frame conflict with equations from another, the value on the mux lines may be altered during one of the clock cycles to break the linear dependency. This brings a great deal of benefit as much denser test cubes can now be compressed however, this comes at the cost slightly increased area, extra data required and computational overhead. The additional computational complexity is addressed in Section 3.4.2 while area is covered in the next paragraph.

Observing Figure 3.18, the additional hardware required amounts to two muxes per scan chain and two AND gates (discussed in the next paragraph). Note that although 2-to-1 muxes are shown in the figure, any size mux may be used. The choice of 2-to-1 has been made for a few reasons. First, by keeping only 2-to-1 muxes the area overhead is kept relatively low. Also, it has been observed that as the size of the mux is increased more mux lines are obviously needed, and the benefit of solving denser blocks is lost against the increased data which must be supplied. In addition, the size of the mux used has a severe impact on computational complexity as will be seen in Section 3.4.2. As a result of these facts, 2-to-1 muxes are preferred. Now we will discuss hardware at the system level.

In the same way that sometimes seeds are frequently demanded, while at other times seed requirements are scarce, it has been observed that some blocks are satisfiable without the need for a reconfigurable phase shifter while some require it. This stems from exactly the same reasons as in the case of seeds, the clustering of care bits. As a result, the mux lines can be shared in the same way as the seed lines. Analogous to the reseed enable signal for the LFSR which instructs the it either to accept or ignore the seed buffered in its seed register, the *Mux EN* signal tells the core to either use the values coming on the mux lines or ignore them. This is accomplished through a simple AND gate, when the *Mux EN* signal is 0 the mux lines are masked and the phase shifter configuration is the same as the case for seed-only. Also similar to the reseed enable, the mux enable must be driven at the system level by some control. The control is almost identical to that for seeds, consisting of a control register to buffer the incoming control stream and a decoder implemented for both input and output muxes, which allows the input and output mux lines to be passed independently between cores on a block by block basis (during one block, one core may receive the seed lines, another the input mux lines and another the output mux lines). Although the area of the control logic is now in triplicate of the control area for the seed-only case, it is still well within the acceptable overhead range. Next we detail some of the implications on CAD when using this approach.

3.4.2 Seed-mux CAD Support - Core Level

With changes to the core level architecture it is expected that changes will be required to the algorithmic framework. This is in fact the case. Although at the conceptual level the changes are minor there are some issues at the practical level that have needed to be addressed. Most importantly, in keeping with our established paradigm we have sought to keep the hardware simple by pushing complexity into the algorithms that do the compression.

The major change to the algorithm involves calculation of equations for multiple systems of equations, since now the phase shifter may be reconfigured. The major challenge here, as mentioned above, is the issue of computational complexity. To see why computational overhead increases with the use of a reconfigurable phase shifter consider the case where that mux lines are used, one for the input muxes and one for the output muxes (as in Figure 3.18). This means that in each clock cycle, a choice of 4 distinct phase shifter matrices exists. Over two clock cycles, 16 systems of equations may be formed (any of 4 matrices in the first clock cycle with any of 4 in the second). Extrapolating, it is easy to see that the number of potential systems of equations which must be checked grows exponentially with the block length. At the algorithmic level this is manifested through backtracking as shown in the next paragraph.

To solve for a seed to a block when a reconfigurable phase shifter is used, the equations are added frame by frame as is the case for the seed-only architecture. The difference is that for the seed-only case, once the equations for a frame have been added there is no room to maneuver. If a linear inconsistency exists the block must be abandoned. In the case of the reconfigurable phase shifter, if adding a frame causes a linear inconsistency the other configurations are tried for that frame. If the linear inconsistency occurs for all the configurations in that frame the algorithm will back-track, change the configuration for the previous frame and try all the configurations again. Having recorded what configuration the phase shifter used in each clock cycle, the mux line stream can be determined immediately in the case that the block is compressible. It is clear that if a block is incompressible all the configurations will have to be tried, resulting in explosive runtimes. In order to combat this, a backtrack limit has been imposed which allows only a set number of reconfiguration trials before abandoning the block. Although this moderately affects the compression since some patterns which may be compressible after a large number of reconfigurations are abandoned, the runtime is significantly improved. The loss of compression arising from the backtrack limit is more than balanced by the increase in block length which is permitted by it. Since a seed (of fixed width = LFSR size) is loaded during one block, if the block length is increased the number of seed lines required is reduced. Thus we desire a longer block length for improved compression, however this would seriously impact computational time during compression if the backtrack limit were not in place.

It is therefore for both the reasons of area overhead and computational complexity that 2-to-1 muxes are preferred. As for hardware, we have discussed the hardware at the system level which moderates whether the mux lines driven from the ATE (outside the chip) are observed by a given core or ignored. In the next section we discuss the impact of this modification on the algorithm at the system level.

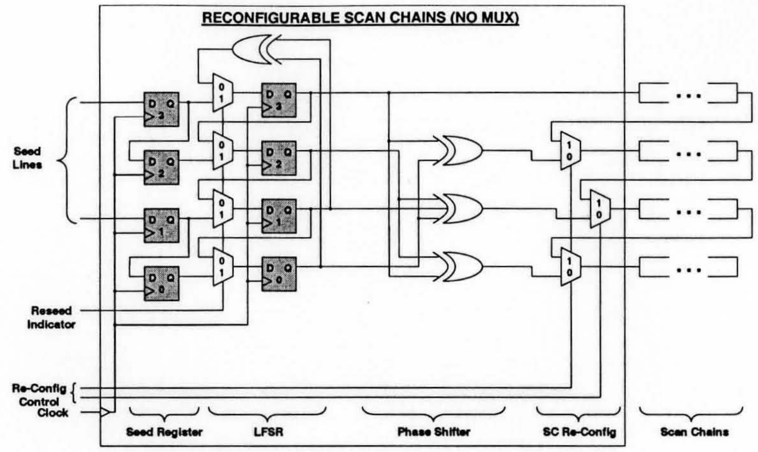
3.4.3 Seed-mux CAD Support - System Level

Unlike the modifications to the algorithm required at the core level which are conceptually simple but practically complex, the modifications to the system level algorithm are simple in both aspects. Recall that the StreamPack algorithm (Algorithm 3) traverses the lists of patterns attempting to slot them in at progressively increasing times. This process hinges on the **CheckFit** function (line 16) which determines whether a pattern will fit at the current location, by checking if during any block of the pattern, more than 1 core requires reseeding. With the modified streams from StreamBIST with muxes, we now have an in-mux profile and an out-mux profile in addition to the reseed profile. **CheckFit** now simply checks to ensure that not more than 1 core requires reseeding, not more than 1 core requires the in-mux lines, and not more than 1 core requires the out-mux lines during any block of the test pattern. This generally results in more NOOP insertion, since there is an increased probability of conflict between any of three sets of lines than when there is just one set of lines as for seed-only. This increased level of NOOPs and the extra data and control for the muxes should be offset however by the reduced data per seed which is enabled by the longer block length. In addition to the changes to the **CheckFit** algorithm just described, the only other major difference is the generation of control streams for passing the mux lines between cores in addition to the seed lines.

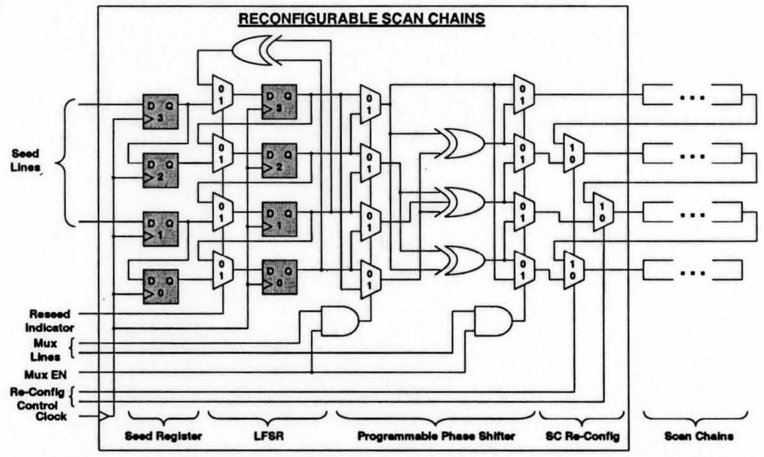
Although this method is able to improve compression somewhat by enabling increased block lengths, the difference is not substantial as will be discussed in the experimental results (Section 3.6). For this reason, another method of addressing linear dependencies has been proposed, multi-mode scan chain reconfiguration as outlined in the next section.

3.5 Multi-mode Packing

As we have seen in the previous section, introduction of non-linearity (muxes and mux data streams) in the decompressor has enabled better compression by improving block length. As discussed however, the improvement in compression is not substantial. This is a result of the fact that an increased number of data streams to be packed



(a) Without programmable phase shifter.



(b) With programmable phase shifter.

Figure 3.19: Multi-mode core level decompressor.

together translates to an increased probability of conflict, and therefore a larger number of NOOPs inserted. Thus the extra NOOPs dilute the effect of the increased block length. This section proposes another way to address the linear dependencies responsible for lockouts which reduce compression.

Figure 3.19 shows the modifications made to the decompressor of the last section to outfit it for multiple mode scan chain reconfiguration. Recall that the number of seed bits provided translates roughly to how many care bits may be expanded by that seed. In the last section, we have attempted to boost the number of care bits which may be expanded by increasing the set of equations from which we may choose in any clock cycle. The architecture given here uses reconfiguration to reduce the number of care bits which should appear in any block by reducing the size of the block by shrinking the number of scan chains. Note that both varieties of decompressor may employ scan chain reconfiguration, both the seed-only packing (Figure 3.19(a) and the seed-mux packing (Figure 3.19(b)). In the forthcoming discussion explanations will be centered around the seed-only architecture for clarity, but anything discussed applies equally to the multi-mode seed-mux architecture.

To see scan chain reconfiguration in effect, consider a design using 32 scan chains and a 32 bit LFSR using a block length of 4. This results in 8 seed lines being needed. During a given block, $4 \times 32 = 128$ bits are expanded from the LFSR while only $4 \times 8 = 32$ are shifted in on the seed lines. This indicates that the block may contain 32 care bits and still remain compressible - this translates to $32/128 = 25\%$ care bit density. After the first reconfiguration (we are now in mode 2), the muxes between the phase shifter outputs and the scan chain inputs in Figure 3.19(a) are activated through the *Re-Config Control.* This is done in such a way as to link the input of every second scan chain to the output of the previous chain thereby doubling the length of each chain, but cutting the number of chains in half. If the block length is kept at 4, we now have $4 \times 16 = 64$ care bits expanded in each block but the number of blocks which we must expand per pattern has doubled. Since the seed end has not changed, we may still satisfy 32 care bits and so the maximum tolerable care bit density is $32/64 = 50\%$. Thus by reducing the number of scan chains, the allowable care bit density is increased at the expense of scan time which lowers compression for the patterns tested in later modes.

In a sense, this architecture also exploits clustering of care bits within a test set but across patterns instead of within a pattern. It has been observed that some patterns have very high average care bit densities while others are very sparse. Thus we can use the high compression mode (which will tolerate only lower care bit densities) to compress the sparse patterns, and only the very dense patterns need to suffer the reduced compression of the later scan chain reconfiguration modes. In this way the average compression of the entire test set is improved. Of high importance in this approach is that reconfiguration can be performed a sufficient number of times to enable *any* pattern to be passed through it, so long as one is willing to pay the penalty in reduced compression. This means that application of a pattern with 100% care bit density would be straightforward with this architecture whereas other approaches (including the seed-only and seed-mux without reconfiguration as discussed earlier) might not be able to apply the pattern. This fits well with the *programmability* feature which we desire as discussed in Section 3.1.4.

3.5.1 Multi-mode Hardware

In terms of hardware overhead it is clear that the penalty is small at the core level. Another mux per scan chain is required, bringing the total to three 2-to-1 muxes per scan chain if the multi-mode seed-mux (the greatest area overhead) approach is used. Before discussing the system level implications, it should be noted that all these approaches may be used in the case of a single core. Although they are presented here in light of decompression for the system level, they will still provide data reduction if applied to a single circuit. If this approach in particular is used for a single core, no data overhead is required for the scan chain reconfiguration signals (re-config control). This is due to the fact that all the patterns for one mode may be tested, then the scan chains can be reconfigured and all the patterns of the next mode tested and so on. Because the re-config control signals do not have to be updated every clock cycle they are deemed static, and may be accessed through some means other than the tester channels. A method of accessing such signals is through a JTAG interface [73], first developed for board testing and is now used for many applications similar to this (eg. programming FPGAs). In this way, reconfiguration can come at no additional data cost when used on a stand-alone core.

On the other hand, when reconfiguration is used for multiple cores in a system under the StreamPack methodology, we desire the ability to switch between modes during any given block for reasons that will be explained in Section 3.5.3. To enable this, the mode information must be provided in the configuration (control) stream which comes on the tester channels. How this is handled is shown in Section 3.5.3.

Algorithm 4: MMStreamBIST

```
Input  : $test\_cubes$, $parameters$
Output: $compressed\_stream^{1,2,\cdots}_{core}$, $decomp\_hardware_{core}$
1 $decomp\_hardware_{core}$ = GenerateCoreHardware($parameters$);
2 $Patterns_{remain} = test\_cubes$;
3 $Mode = 1$;
4 while ($Patterns_{remaining} \neq NULL$) do
5     {$compressed\_stream^{Mode}_{core}$,$incompressible^{Mode}_{core}$} =
      StreamBIST($Patterns_{remain}$, $parameters$) // Algorithm 1;
6     $Patterns_{remain}$ = Reformat($incompressible^{Mode}_{core}$);
7     $Mode = Mode+1$;
  end
8 Return $compressed\_stream^{1,2,\cdots}_{core}$, $decomp\_hardware_{core}$;
```

3.5.2 Multi-mode CAD Support - Core Level

As far as CAD support for the multi-mode approach is concerned, at the core level not much different needs to be done. Algorithm 4 shows how an algorithm wrapped around the StreamBIST algorithm (Algorithm 1) can enable simple calculation of the multi-mode streams. In this case, the hardware description is generated first as usual. Then, starting with the test patterns formatted for the first mode, StreamBIST is run to compress what patterns can be compressed and return a list of incompressible patterns. Next, the incompressible patterns are reformatted by the **Reformat** function (line 6) which simply parses the incompressible test patterns and writes them with half the scan chains of the original test set, that is, the way they would appear to the decompressor looking through the scan chain reconfiguration hardware. These reformatted patterns are passed again through StreamBIST and the cycle continues until all patterns have been compressed. This gives a set of streams, one for each mode, and a reconfigurable hardware architecture which can expand the streams. In the next section, use of these streams in StreamPack at the system level is discussed.

3.5.3 Multi-mode CAD Support - System Level

When packing streams from multiple mode decompressors, the feature of the core connection scheme discussed with the introduction of StreamPack for seed-only becomes of significant importance. The very convenient feature of this ability to connect cores into chains of exclusive test activity enables different modes of the same core to be placed in the same chain as virtual cores. In this way, no mode of any core will be tested at the same time as another mode of itself. Thus we can directly reuse the same StreamPack algorithm to obtain multi-mode packed streams. Some care must be taken during parsing to re-create the control streams, to ensure that the virtual core signals which are generated are hooked up in the appropriate way to the re-config control signals of a single multi-mode core. Aside from this detail the algorithm is completely reusable. This point also emphasizes the importance of having block-by-block access to the reconfiguration signals. By allowing this freedom to test the core in one mode during one pattern and another in the next at will reduces the probability of NOOP insertion and thus the overall test time. These effects are reflected in the experimental results as discussed in the following section.

3.6 Experimental Results

In this section, experimental results pertaining to the methods proposed in the previous sections are provided. A brief setup (Section 3.6.1) details some specifics relating to our experiments after which tester channel utilization results are given (Section 3.6.2). This is followed by compression, hardware area overhead and central processing unit (CPU) runtime results presented in light of the seed-only, seed-mux and multi-mode compression schemes. Most importantly, relation to prior work and relevance are addressed in Section 3.6.8.

3.6.1 Experimental Setup

With respect to experimental setup, a couple of issues must be addressed. First note that the experiments were carried out on the test sets of the ISCAS'89 benchmark circuits [12]. To determine the effects of care bit density the experiments were performed on different versions of the same test set, which had undergone different levels of static compaction. These test sets are denoted with the label *Comp-(num)*, where *(num)* indicates the maximum allowed specified bits per pattern. For example, in the *Comp-50* test set merging of two patterns was only permitted if this would result in a vector with less than 50 specified bits. In this way, compaction could be moderated to produce test sets with varying degrees of average care bit density, *Comp-50* being the most sparse and *Comp-1000* being the most dense. Another compacted version of the test set was obtained through the use of MINTEST [30], for which compaction was unconstrained resulting in much denser patterns.

For setup of the experiments themselves, we have used a hypothetical SOC comprised of the 5 largest ISCAS'89 benchmark circuits: s13207, s15850, s35932, s38417 and s38584. The experiments have been carried out for 8, 16 and 32 scan chains for seed-only and seed-mux packing, and in each case the LFSR size was kept equal to the number of scan chains. The multi-mode (both seed-only and seed-mux) results used a 32 bit LFSR with 32 scan chains in the first mode, 16 in the second and 8 in the final mode of reconfiguration. Finally, different block lengths were used in different experiments, and are reported in parenthesis alongside each VTD result.

3.6.2 Tester Channel Utilization

In this section, we discuss the effect of time-multiplexed tester channels on the tester channel utilization first discussed in Section 3.2. Figure 3.20 shows how employing StreamPack can substantially improve the utilization of the tester channels which was calculated in the following way. Before packing, StreamBIST was first applied to each core and utilization was taken as the ratio of the total number of reseeded blocks across the 5 cores from our hypothetical SOC to the number of blocks in total for the 5 cores, giving the *No Packing* result in the figure. Post packing utilization was

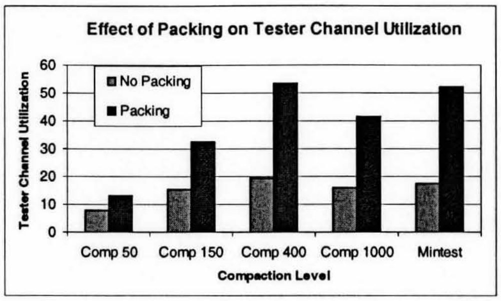


Figure 3.20: Tester channel utilization improvement.

taken (after running StreamPack) as the number of active (reseeded) blocks against the total blocks in the StreamPack output stream. The reason for the improvement is that during StreamPack the algorithm attempts to use the inactive periods on each core (the unseeded blocks) to reseed another core. This reduces the overall number of unused blocks and thus improves the utilization. Another way to consider this point is that the number of reseeds contained in the pre- and post-packed streams is the same, while reducing the total amount of test time from the serial case (as in Figure 3.17). The total number of blocks (the denominator of the utilization expression) is reduced while the numerator remains constant, thus utilization increases.

Presentation of results for utilization are meaningless if not accompanied by a tangible effect, reduction of VTD or test time for example. Improvement of tester channel utilization is not the goal in and of itself, we have focused on its improvement to facilitate improved compression. How tester channel utilization can produce better compression is presented next.

3.6.3 Uncompressed Test Sets

All VTD results reported were on an hypothetical SOC of the 5 largest ISCAS'89 benchmarks, as mentioned above. Table 3.2 shows the uncompressed volumes of test data for this SOC, obtained by taking the simple sum of the VTDs of the individual cores, which is slightly optimistic for an uncompressed solution[2]. Test time is then

[2]The VTD reported as the sum in this way could only be realistically achieved by serial testing where each core has only a single scan chain, resulting in exceedingly long test times. Nevertheless, we report our results for multiple scan chains against this value and therefore our results appear slightly more pessimistic that what may actually be achieved in practice.

Compaction	No. SC	Uncompressed VTD	
		Scan Time	VTD
Comp-50	8	1980667	15845330
	16	990334	
	32	495167	
Comp-150	8	488078	3904623
	16	244039	
	32	122020	
Comp-400	8	199724	1597792
	16	99862	
	32	49931	
Comp-1000	8	168267	1346136
	16	84134	
	32	42067	
Mintest	8	77904	623230
	16	38952	
	32	19476	

Table 3.2: Uncompressed VTD for 5 ISCAS'89 benchmarks.

calculated as VTD / number of scan chains. Also, in this scheme the number of data lines is the same for the uncompressed and compressed cases which shows more clearly the tradeoff between control information overhead and test application time. Said another way, additional control channels from the ATE increase VTD but simultaneously enable packing which reduces scan time (as in Figure 3.17) and thereby VTD. By keeping the number of seed channels the same in compressed and uncompressed cases we can more clearly see whether the benefit in scan time outweighs the cost in extra control channels, thus we can evaluate the effect of packing directly. In the next section, VTD and scan time are discussed for the seed-only and seed-mux decompressors not using scan chain reconfiguration.

3.6.4 Single Mode Testing VTD

In this section the compression results for the seed-only and seed-mux cases as shown in Table 3.3 are discussed. For the Comp-50 and Comp-150 test sets, both seed-only and seed-mux provide improved test time and VTD results. Starting with Comp-400, seed-only continues to provide compression while seed-mux falls behind even the uncompressed VTD, but still outperforms seed-only in test time. The reason for this is as follows. In seed-only StreamBIST the demand for data must be satisfied on only

		Seed-only (Section 3.3)		Seed-mux (Section 3.4)	
Compaction	No. SC	Scan Time	VTD	Scan Time	VTD
Comp-50	8	1314950	15779400 (1)	1441552	5766208 (8)
	16	660630	13212600 (1)	700776	3503880 (8)
	32	337016	6066288 (2)	346624	2426368 (8)
Comp-150	8	260975	3131700 (1)	268482	2684820 (2)
	16	131536	2630720 (1)	152972	1223776 (4)
	32	69980	1259640 (2)	74580	894960 (4)
Comp-400	8	109251	1311012 (1)	108966	2397252 (1)
	16	66167	1323340 (1)	55263	1657890 (1)
	32	36991	1331676 (1)	28502	1311092 (1)
Comp-1000	8	86157	1033884 (1)	87192	1918224 (1)
	16	48137	962740 (1)	44039	1321170 (1)
	32	26250	945000 (1)	22200	1021200 (1)
Mintest	8	38564	462768 (1)	37294	820468 (1)
	16	20695	413900 (1)	18655	559650 (1)
	32	11658	419688 (1)	9612	442152 (1)

Table 3.3: Seed-only and seed-mux packing VTD and scan time.

the seed lines, while seed-mux StreamBIST can distribute the demand over seed, input mux and output mux lines. As a result, the seed lines are less utilized and therefore easier to pack. The mux lines must be simultaneously packed however, which is why the improvement in scan time is not substantial. The test time is therefore slightly lowered at the expense of significant additional mux lines (to modify the phase shifter) and control overhead (to distribute the mux lines to the appropriate core).

The main advantage that seed-mux packing has over seed-only packing is on the programmability front (Section 3.1.4) since patterns with higher care bit density may be passed through the seed-mux architecture, as made clear by the increased block length (reported in parenthesis in the table) for the Comp-50 and Comp-150 test sets. This fact also explains why VTD is improved for the Comp-50 and Comp-150 test sets. While in these two cases the addition of the mux lines enabled increased block length (and therefore reduced number of tester data channels), from Comp-400 on the mux lines were not able to allow a larger block to be used than in the seed-only case, and thus the compression suffered at the hands of increased number of tester channels. So in light of programmability, even though seed-mux cannot provide the same level of compression the better compression obtained through seed-only StreamBIST may not matter if the test set contains patterns which cannot be passed by the seed-only

	Seed-only		Seed-mux	
Compaction	Scan Time	VTD	Scan Time	VTD
Comp-50	438752	1316256 (16)	422128	2110640 (16)
Comp-150	120752	603760 (8)	122240	855680 (8)
Comp-400	56464	564640 (4)	81720	653760 (8)
Comp-1000	49300	493000 (4)	42968	558584 (4)
Mintest	23480	234800 (4)	22172	288236 (4)

Table 3.4: Seed-only and seed-mux multi-mode VTD results (Section 3.5).

architecture. As we have seen earlier, multi-mode addresses incompressible patterns in a different way than seed-mux and the next section shows the benefits of multi-mode testing in this respect, that any pattern may be applied and at the same time the VTD and scan time can be further reduced.

3.6.5 Multi-mode Testing VTD

In Table 3.4, VTD and scan time for both the seed-only and seed-mux multi-mode approaches are shown. Note that the results are not given for varying numbers of scan chains, since the multi-mode approach uses different numbers of scan chains in different modes to enable application of any pattern. One may wonder why the seed-mux architecture is kept, now that any pattern may be passed using reconfiguration. The answer is that although seed-mux does not bring any added programmability benefit, it may still enable increased packing by sharing the load of test data delivery among the seed, in-mux and out-mux lines instead of just the seed lines as discussed in the previous section. This is reflected here where we see a similar trend between the seed-only and seed-mux VTD results as for the case without reconfiguration. Notice that for the most part the scan times for seed-mux in Table 3.4 are on par with, and usually slightly better than, those for seed-only. Despite longer scan times seed-only still performs better in VTD, since the overhead of mux lines and additional control are not present. As before, the slightly increased ease of packing does not sufficiently compensate for the additional data overhead in the streams which drive the muxes, and control overhead to determine which core observes the mux lines. An important distinction to be made here is that where the last section pointed out the case for seed-mux in light of the ability to satisfy more specified patterns, that need

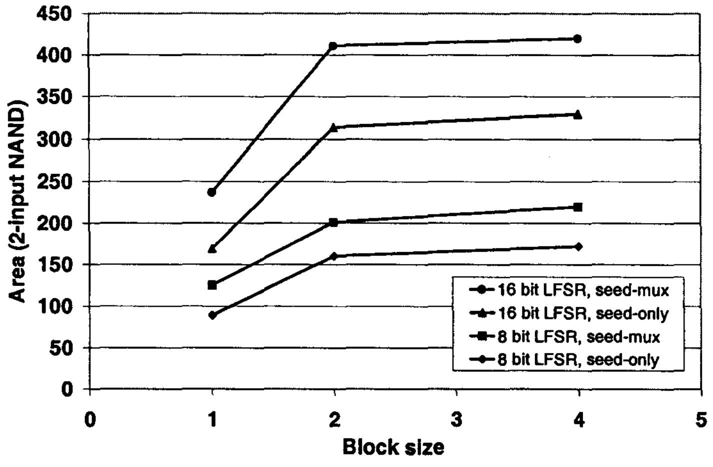


Figure 3.21: Core level area (8,16 bit LFSRs).

is no longer here since the multi-mode feature takes care of programmability. Since any pattern may be passed through the reconfigurable seed-only architecture and the VTD is improved, it is obviously the best choice for the decompression methodology.

In short, the preceding discussion on VTD results for the differing implementations has allowed us to conclude that 1) the benefits brought about by the addition of mux lines barely justify the incurred overhead (if at all) and 2) the multi-mode seed-only architecture performs the best overall since it gives the best compression and the near best scan time all while satisfying the programmability objective. In the next section we discuss the impact to on-chip area for each decompressor type.

3.6.6 Area Overhead Estimation

To obtain results for area, the HDL descriptions generated by StreamBIST and StreamPack for each different sized decompressor were compiled for TSMC [93] 0.18μm CMOS technology using Synopsys Design Compiler [92] for synthesis to obtain the results which are given as equivalent number of 2-input NAND gates.

The relationship between block length and decompressor area is shown for 8 and 16 bit LFSR decompressors (with 8 and 16 scan chains respectively) in Figure 3.21 and for 32 bit LFSR and multi-mode decompressors (with 32 scan chains) in Figure 3.22. Note first of all the large jump between the decompressor area for a block

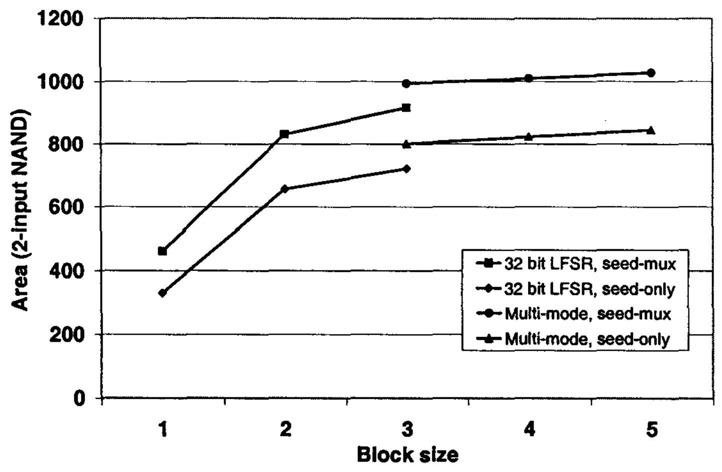


Figure 3.22: Core level area (32 bit LFSR, multi-mode).

length of 1 and that for a block length of 2. This results from the fact that use of a block length of 1 removes the need for a seed buffer since no variables need to be buffered, they are all passed from the seed lines to the LFSR in a single clock cycle. When the block length is increased beyond 1 the area for the seed buffer must be added (resulting in the noticeable jump) however, once the buffer has been added the area remains relatively constant with the block size. The small variations in area that appear across different block sizes in the 8, 16 and 32 bit LFSR cases result from a different phase shifter being generated for differing block sizes and although always responsible for an increase in the figures shown, may just as well cause a small decrease in area. In the case of multi-mode decompressors the phase shifter area may still decrease however, as the block size increases the number of reconfiguration modes which must be provided increases, and thus the number of reconfiguration muxes added increases which adds to the overall area. In the worst case however, the number of reconfiguration muxes (2-to-1) required is one less than the number of scan chains and so remains acceptable. Note that the area reported here is pessimistic since no optimization of the phase shifter has been done, as in [79]. Because of this, our phase shifter area exceeds the lower bound of one gate per scan chain and as a result if the phase shifter were optimized, our area results would be improved.

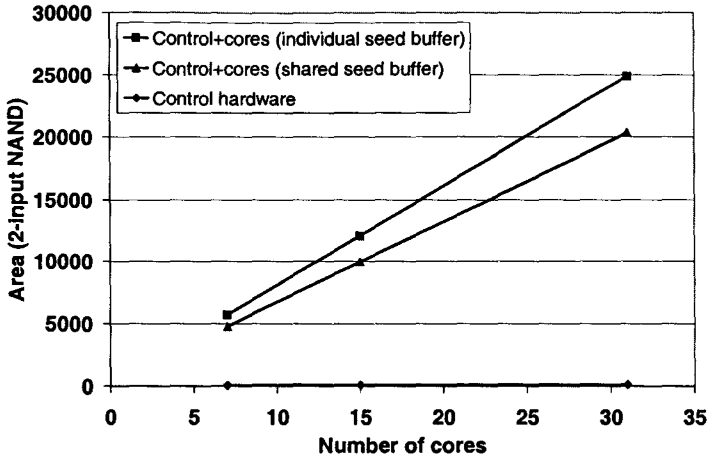


Figure 3.23: System level area.

Notice also in Figures 3.21 and 3.22 the trend between a given sized seed-only decompressor and its respective seed-mux counterpart. In each case the gap between the two is essentially constant and it is greater for larger decompressors. The difference in area directly reflects the presence of the multiplexers on either side of the phase shifter as shown in Figures 3.18 and 3.19(b). The number of multiplexers added does not depend at all on the block length, it is equal to twice the number of scan chains which explains the constancy of the gaps for a given number of scan chains and the larger gap when there are more scan chains. Since the 32 bit LFSR and multi-mode decompressors share the same sized LFSR, number of scan chains and phase shifter, additional muxes are also responsible for the increased area between these two shown in Figure 3.22. The gap here is smaller than between seed-only and seed-mux since the worst case area added for scan chain reconfiguration is one 2-to-1 multiplexer for each scan chain instead of two.

In Figure 3.23, some system level considerations are illustrated. The most notable feature of the graph is the minuscule area of the control hardware. As mentioned before, this area consists only of a small register and some decoding logic and it grows logarithmically with the number of cores in the system. Note that because the hardware is discrete, the control hardware size is the same for 4 cores or 7 cores and for 8 cores or 15 cores etc. Despite this, as the upper two curves in the graph show the control area can be neglected when compared to the total system level area.

The uppermost curve (control+cores, individual seed buffers) of Figure 3.23 shows the total area required at the system level, as the sum of the system level decompressors and the control hardware. Since the area is dominated by the core level decompressors, it is no surprise that it grows linearly with the number of cores. This case includes a seed buffer for each core (see Figure 3.10(a)) however, earlier discussion surrounding Figure 3.10(b) showed how the seed buffer can be shared because the channels are time-multiplexed. As a result the area may be reduced to the values shown in the middle curve (control+cores, shared seed buffer) of Figure 3.23. It should be noted that the system level area for our approach is not dependent on the circuit size, only on the number of scan chains and the number of cores. In the case of 30 cores and 32 scan chains per core, we may make a pessimistic assumption of 100 FFs per scan chain and 20 gates per FF giving 1 920 000 gates for the total circuit area. Figure 3.23 shows an area overhead of $\approx$ 20 000 gates for this case which amounts to an overhead of $\approx$ 1% and it is expected that even with variations in FFs/chain and gates/FF it is very unlikely that the overhead will exceed 2%. Furthermore, as the number of cores is increased the area saved will increase (i.e. the number of seed buffers replaced by a single seed buffer will increase), and if the phase shifter area is optimized as mentioned before the seed buffer will constitute a greater portion of each core level decompressor and as a result the savings will further increase. In this way, time-multiplexing has enabled *increased compression* not at the expense of extra area, but *with reduced area* compared to using [105] for each core.

3.6.7 Algorithm Execution Time

Most of the results seen thus far pertain more directly to the implementation and application of the decompressor than to its CAD flow. Execution time on the other hand is related mainly to the CAD flow. The time spent on computation for compression of test data in this approach can be divided into four main steps. Input test data from each core is parsed into a form acceptable to StreamBIST, StreamBIST is applied to each core individually to produce packing amenable streams, streams from all the cores are collected together and reordered (packed) by StreamPack, and finally the resulting stream is parsed into the form required by the user (e.g. functional simulation streams or ATE pin vector sequences). The core processing in terms of sophistication occurs in the middle two steps and in terms of execution time in StreamBIST in particular. Input parsing and output parsing are straightforward algorithms which essentially run in linear time (they make a single pass through the test set to process it). For the experiments provided, the duration was less than a second (per core) for input parsing and less than 1 minute for output parsing at the system level. Experiments were performed on a 3.06 GHz Pentium 4 Xeon processing node with 1GB of memory running Red Hat Linux 7.3.

As just mentioned, StreamBIST dominated the overall execution time. This was a result of a number of factors. First, it is the complexity of StreamBIST which produces the streams of increased packability and thereby facilitates easier and faster execution on StreamPack's part. In addition, StreamPack operates once at the system level while StreamBIST must be performed on each core in the system. At the heart of StreamBIST (Algorithm 1) is the **CompressPattern** function (Algorithm 2). The number of calls to **CompressPattern** is clearly linear in the number of test patterns in the test set. Inspecting **CompressPattern** it can be seen that the number of calls to **CheckEquations** is essentially linear in the number of blocks per pattern. At the core of **CheckEquations**, Gaussian elimination is performed to reduce the system of equations and check for linear inconsistencies which is of worst-case quadratic complexity in the number of equations. Since the number of equations depends on the number of care bits it is difficult to quantify, however it is strongly linked to the block size. In the seed-only case then, we can say the entire algorithm is dependent linearly on the total number of blocks in the test set and quadratically on the size of each block, not entirely unmanageable. In the seed-mux case however, recall that backtracking is used to search the possible systems of equations leading to explosive runtimes if left uncapped (Section 3.4.2). To avoid this a backtrack limit has been imposed but runtimes remain significantly longer than the seed-only case. For the experiments performed, the runtimes per core for StreamBIST ranged anywhere from under a minute to in excess of 40 hours, which can become serious since StreamBIST must be performed on each core. At the same time this large result of 40 hours is for the seed-mux case just mentioned, and as we have seen in the VTD discussion earlier, seed-only packing actually provides better compression removing the need for the computationally expensive seed-mux case especially since reconfiguration takes care of the programmability issue which seed-mux was devised to address. If seed-mux packing were required, this CPU runtime bottleneck could be considered a limitation of this methodology as discussed in Section 4.2 however, the next paragraph discusses why it is more important to have fast system level algorithms over core level ones. Just before moving on, note that in multi-mode StreamBIST, StreamBIST is called iteratively but with a reducing number of patterns each time. The execution times for the multi-mode StreamBIST approaches essentially matched their respective single mode partners, largely due to the fact that incompressible patterns are often spotted before to much wasted execution time is spent trying to compress them.

The good news in light of the above is that once time has been spent generating the streams for StreamPack, its operation is very fast. Noting Algorithm 3, it can be seen that lists of patterns awaiting placement are cycled through, calling **CheckFit** each time. Because **CheckFit** is a very simple function which checks only the reseed points of the pattern to be placed against other placed patterns, it runs quickly. With each placed pattern reducing the size of the remaining lists, operation continuously accelerates. For this reason StreamPack execution times have been well below a minute for all the experiments run. This can be used advantageously to address StreamBIST runtime, by storing with each core its StreamBIST compressed test set and simply plugging it in to StreamPack when the core is used in a system. StreamBIST would only need to be run when there are changes to the test set, likely an infrequent occurrence especially for IP-protected cores (at which this method is targeted). So in this case it is advantageous to have the runtime intensive part at the core level where changes are less frequent, as per the design reuse philosophy. With it likely that the core will be used more times in its lifetime than it will undergo changes, increased cost to the test set at the core level is acceptable provided it comes to facilitate less cost at the system level, which is exactly the case here.

Having summarized the main aspects of the proposed test methodology and discussed tradeoffs and their influencing factors within the methodology, we will now turn attention to how this method fits in the framework of test data compression methods at large by comparison with other methods.

3.6.8 Comparison to Existing Solutions

Although a fair amount has been said about VTD and compression in this thesis, we have also tried to emphasize the other aspects which contribute to the practical viability of a compression method. By narrowing the scope of comparison, important issues can be missed and good work with desirable features may be overlooked. Even within this work, as mentioned earlier when discussing compression results for single mode seed-only and seed-mux methods (Section 3.6.4), it was noted that although the seed-only approach delivers smaller VTDs it is less programmable when compared to the seed-mux architecture (for block lengths greater than 1), and thus if programmability is of utmost importance the seed-mux method may be chosen over seed-only despite poorer compression. It is mainly for reasons such as these that we have enumerated the desirable criteria for test data compression toward the beginning of this chapter. By understanding what is required in a test compression solution, the problem can be solved more effectively and solutions provided which meet the needs of those who use them, keeping in mind that different features are required in different situations.

In light of the above, the following discussion details some representative approaches to test data compression not only with respect to the compression which they achieve but also what features they provide. Having established the importance of IP-consistency, modularity, scalability, programmability and a low cost test platform, the methods presented here are evaluated in terms of these 5 criteria. Table 3.5 summarizes the VTD comparison for the approaches we discuss, and Table 3.6 summarizes their compliance to the established criteria. Noting that all the methods respect the low cost test platform criteria, it will be omitted from the discussion on each circuit.

Before discussing other approaches with distinct features, note first of all that since this method builds off the architecture proposed in [105] (see Section 3.3.1) a comparison against their work has been given inherently in the tester channel utilization discussion in Section 3.6.2. The improved tester channel utilization showed how VTD could be reduced from the case in [105] while simultaneously reducing the area (and thereby providing scalability) through system level exploitations as discussed in Section 3.6.6. Furthermore, the modifications we have provided enable programmability and IP-consistency. Nevertheless, [105] can be deemed modular, having established the modularity of our approach and given that [105] provided a good starting point for the modifications we have made. The remainder of this section gives comparison against other works with a variety of combinations of the desired features.

The first method shown in Table 3.5 was proposed by [95] and uses a very simple compression code consisting of nine codewords. Because the circuit is not modified in any way, the method is IP-consistent. A fairly significant drawback of this approach is that it requires an acknowledgement signal to drive the tester for the sake of synchronization. Because this means the design cannot be integrated easily into a test infrastructure the method is not modular. In spite of this, the method is scalable and programmable due to well scaling hardware sizes and a codeword set up to allow any word of test data to be passed. Since results for the circuit s35932 were not provided by [95], comparison against the proposed approach is done for a different hypothetical SOC discussed in the next paragraph which yields a total VTD of 212160 for our approach, $\approx 12\%$ more than [95] but without the need for an acknowledgement signal from the tester.

The hypothetical SOC mentioned in the preceding paragraph is composed of only s13207, s15850, s38417 and s38584 from the ISCAS'89 benchmark circuits[3]. The same comparison applies for [5, 41, 59, 80] in the forthcoming paragraphs. The s35932 benchmark circuit is commonly neglected in experimental results for compression methods (especially linear decompressor based methods) since it is *random-pattern resistant*, which means the patterns contain long runs of 1s and 0s and are therefore not easily encodable in pseudo-random sequences like those produced by linear

[3] VTDs marked by $^+$ in the table indicate s35932 is not included.

	[95]	[41]	[59]	[5]	[80]	Proposed
s13207	29224	17160	11285	9708	10585	-
s15850	25883	14560	12438	10726	9805	-
s35932	-	-	-	-	-	-
s38417	64857	37206	34767	36864	31458	-
s38584	68631	40406	29397	27555	18568	-
Total	188595$^+$	109332$^+$	87887$^+$	84853$^+$	70416$^+$	212160$^+$

	[28]	[84]	[68]	[63]	[82]	Proposed
s13207	-	74423	33605	8517	14145	-
s15850	-	26021	21534	13873	13919	-
s35932	-	7222	804	1400	4492	-
s38417	-	45003	51156	62939	52793	-
s38584	-	73464	43320	53287	26644	-
Total	279837*	226133	150419	140016	111993	234800

Table 3.5: VTD for compared approaches ($^+$ = footnote 3, * = footnote 4).

	[95]	[41]	[59]	[5]	[80]	[28]	[84]	[68]	[63]	[82]	Proposed
IP-consistent	√	X	√	X	X	√	√	X	√	X	√
Modular	X	X	X	√	√	√	X	X	X	X	√
Scalable	√	√	√	√	√	√	X	X	√	X	√
Programmable	√	X	X	X	X	√	√	√	√	X	√
Low cost ATE	√	√	√	√	√	√	√	√	√	√	√

Table 3.6: Criteria satisfied for compared approaches.

decompressors. In light of the programmability criteria any complete compression method should be able to address random pattern resistant circuits as well as random pattern amenable ones, and lack of results for s35932 is often an indicator of lack of programmability. Although [95] is programmable, it will be shown in the following paragraphs that [5, 41, 59, 80] are not programmable. A strength of our approach is the ability to apply test sets for random pattern resistant circuits, even if some patterns in the test set are 100% specified (containing no x-bits), hence the programmability of our method. Furthermore, application of these fully specified patterns can be done with a reduced VTD, both by providing compression to the other partially specified patterns in the test set and by time-multiplexing at the system level.

The next approach listed in the table was proposed in [41], which forms a virtual scan chain appearing shorter than the number of actual scan cells contained therein, accomplished through the use of an LFSR. A drawback of the approach is the use of integrated fault simulation/ATPG to obtain compressible test cubes, removing IP-consistency. A very nice feature of this approach on the other hand is that after compression, the virtual scan chain appears exactly like a regular scan chain but shorter (and with different values of course). Such an approach would certainly be modular aside from the fact that there is no apparent system level exploitable feature of the test sets. The method is scalable since the overhead in hardware and computation are reasonable but since no mechanism for application of any pattern has been provided, the method is not programmable. This method is able to achieve lower VTD than our approach by compromising on the features of IP-consistency, modularity and programmability.

Next in the table is [59] which proposes a multi-level scheme using Huffman decoders to decompress LFSR seeds, which LFSRs then expand into test patterns. This method provides IP-consistency since no details internal to the circuit are required. Modularity is lacked however because the Huffman compression takes advantage of seed stream features for compression leaving little to exploit at the system level. With regard to scalability, no challenges should be faced since the hardware and algorithms seem manageable for increased design sizes however this is difficult to assess since no details regarding implementation for the statistical coding part have been provided. Statistical coding methods by in large are known to suffer from either synchronization difficulty as a result of different ATE and CUT clock frequencies, or from requirement of a synchronization signal from the CUT to the ATE and so this method is likely to meet with methodology issues when implemented. Finally, the method cannot be classified as programmable since the patterns must pass through the LFSR which fails when the care bit density is high. Again, the VTD obtained is better than the proposed method but it comes at the cost of modularity and programmability.

Another method against which we compare is [5]. In this approach the output space of the linear decompressor is analyzed with the test set to determine a set of scan cells to invert, which will increase the alignment between the output space and the test set. An advantage of this method is that is can be applied to any linear type decompressor (LFSRs, ring generators, etc.) and can guarantee improved encoding.

However, this comes at the cost of IP-consistency as a result of scan hardware modifications requiring access to internal circuit details, and programmability since there is no mechanism for application of patterns outside the output space of the decompressor. On the other hand, this method may still leave some un-specified bits in seeds or allow multiple patterns to be expanded from a single seed, so it leaves room for exploitation at the system level and is therefore modular. In addition, the method is scalable since 1) the heart of the method is scan cell inversion used to improve the compression of the linear decompressor which does not add any appreciable area and 2) the algorithmic framework is centered around Gaussian elimination which is tractable. By not remaining IP-consistent or programmable, this method is also able to improve VTD.

The next method we consider is [80] which uses ring generator structures to provide compression through variable injection as discussed much earlier along with Figure 2.13. During compression, ATPG and fault simulation are employed and so the method is not IP-consistent, and no mechanism for application of high care bit density patterns is provided so the method is not programmable. Explicit techniques for VTD reduction by the exploitation of ATE features are provided in the work but they are not required to achieve the reported compression, so the method can be used on a low cost test platform. At the same time, although the framework for results reported on the ISCAS circuits is not scalable the method is targeted at large circuits indicating good scalability of the method on the whole. This method also exhibits good modularity since there is a slack present in the variable streams which drive the ring generator. Such streams could be packed using StreamPack, and thus this method could be plugged immediately into the core level in our approach. As in the previous paragraph, this method achieves very good compression in the case where IP-consistency and programmability are of no concern.

The previously discussed approaches neglected reporting of results for s35932. In [28], the opposite is true - more cores have been included than in our experimental SOC[4]. This method uses statistical coding not reliant on the internal circuit details in

[4]VTDs marked by * in the table indicate s5378 and s9234 have also been included.

any way and thus is IP-consistent. The issue of ATE stall synchronization (requirement of the Sync data signal of Figure 2.5 - see Section 2.5) has been addressed in this method by passing data from the tester to other cores on the chip while a given core is still busy decompressing. This system level exploitation of a feature is indicative of modularity, and in essence makes this method the temporal analogue of our spatial approach. The method also scales reasonably, one decompressor per core and some system level decompression logic, and as is usually the case with statistical coding any test set may be applied giving programmability. On the whole this approach meets all the criteria. The only drawback here is the issue of synchronizing the ATE clock and CUT clock which run at different frequencies. In addition power issues can arise due to fast clocking of the scan chains as discussed before. With respect to VTD, this method reports 279837 bits for the *7 largest ISCAS89 cores*: s5378, s9234, s13207, s15850, s35932, s38417. Our result for VTD in this case is 262000, an improvement of $\approx 6\%$.

The approach given in [84] places a register and decoder before the scan chains, and compresses data by encoding the bits which need to change from the previous vector. Since no netlist interaction is required, this method can be considered IP-consistent. However, because the compressed streams leave no room for maneuvering there can be no system level exploitation and thus the method cannot be considered modular. At the same time, although hardware remains small increased numbers of scan chains will lead to large runtimes to determine the mapping of care bits and the order in which mutations should be applied, making this method not scalable. A benefit is that the decoder architecture allows as many bits to be mutated as desired and this approach is therefore programmable. With respect to VTD comparison, it should be noted that due to a statement in the paper there is some ambiguity as to what data has been included in the reported results (some control information is used by the decompressor which must be driven by the tester and it is not clear whether this has been reported). In the best case, if all the data has been reported this method is only about 4% better than the proposed approach while not providing as many features as shown in the Table 3.6.

Scan trees are used by [68] to reduce volume of test data, and as a result access to the scan structure is required which renders the method without IP-consistency. Because the main benefits come from dependencies in the test set, the compressed data will likely not have any features which are sufficiently exploitable at the system level to deem this approach modular. In addition, large MISRs for response collection from a larger number of scan tree outputs makes this method not scalable. With respect to programmability however, a serial mode is provided by which any pattern may be scanned in and thus the method is programmable. Although the reported VTD is quite good, the number of features suffers significantly.

A dictionary based coding method is used in [63] which does not require internal circuit details and is thus IP-consistent. The decompression hardware may potentially leave some room for system level exploitation, but due to some other methodology related issues (such as scan clock gating) the method cannot be deemed modular. In this approach, the hardware does not appear to scale too quickly with the number of scan chains and at the same time, provision is made to apply patterns which do not appear in the dictionary so this method is programmable. With most of the desired features in place the method performs well at $\approx 40\%$ better than StreamPack in VTD for the same configuration. Unfortunately as mentioned just above, the method suffers from methodology issues related to scan clock gating but if these can be overcome the method has good value.

The final approach against which we make a comparison is [82]. Table 3.5 shows that this method provides superior results for VTD. However, as Table 3.6 shows, this comes at the high cost of all features except the low cost test platform. VTD is improved through 1) integrated fault simulation (removing IP-consistency), 2) crafting the phase shifter around the test set (removing programmability) and 3) using unreasonably short internal scan chains (removing scalability). After squeezing out this level of compression through exploitation of these multiple facets, there is nothing left to exploit at the system level, and thus the approach lacks modularity. In addition, the scheme is in some respects similar to that from [84] and like that approach it is unclear whether control data has been reported in the totals or not. It is important to note that our results for VTD may likewise be improved through the application of techniques used by this approach and others (e.g., fault simulation) but we have deemed provision of desired features as more important than VTD reduction only. Nevertheless, this method will provide good compression where IP-consistency, modularity, scalability and programmability are un-important.

In light of all these discussed methods, what is clear is that tradeoffs exist between *all* aspects of a compression methodology, features and compression results included. Many approaches exist which allow a system integrator with carefully formulated requirements to judiciously select a method that ensures they are getting the compression they can, while still retaining all the features they need.

3.7 Summary

In this chapter we have devised five basic objectives for test data compression methodologies by analyzing the needs which EDT is set up to address in the first place. These objectives of IP-consistency, modularity, scalability, programmability and the use of low cost test platforms have been set up as criteria for the design of a new test data compression scheme. As a result, StreamBIST and StreamPack have been proposed to address challenges in core based SOC testing and improvements have been made to the architecture with accompanying discussion. The new architecture and sophisticated algorithmic support have been developed and described in detail. We have seen how this new architecture is able to produce improved tester channel utilization, which results in lower VTD. In many cases, the VTD for StreamPack has rivaled other approaches, but it has been achieved with the presence of all five criteria as discussed in the first part of the chapter. In light of this fact, some comparison to other approaches in terms of both VTD and the features which they provide has been given. It has been shown that some levels of compression come at the expense of one or more features, and it is up to the system integrator to make careful decisions when planning for test. Advantages and drawbacks of this method have also been provided, and will be discussed further in the next chapter.

Chapter 4

Conclusion

In this thesis, we have provided a justification for manufacturing test of digital integrated circuits. Volume of test data considerations for those circuits have led us to conclude that test data compression methods must be applied to keep up with scaling trends in both circuit size and volume of test data. In light of this conclusion, a survey of recent test data compression methods has been given which provides a structured framework for evaluation of these methods. This has led to the development of five desirable criteria for compression methods, specifically those in a core based SOC design paradigm. A new method for test data compression in core based SOCs called StreamBIST/StreamPack has been proposed in this thesis which provides competitive compression while simultaneously addressing all five of the key objectives laid out for compression methods. The proposed method achieves this compression at acceptable expense in area and computational complexity, by sharing tester channels among cores in an SOC at the system level, made possible by the discontinuous demand for data which occurs at the core level. In proposing this method, we have opted to ensure that we provide the five discussed criteria as features rather than focus narrowly on minimization of test data. As comparison with other methods in the previous chapter shows, we have been able to accomplish provision of all five features at a comparable level of compression. In this final chapter, the advantages and limitations of the proposed method are discussed leading to considerations for future work.

4.1 Advantages

As has been emphasized throughout this thesis, the main advantage of this work is the enabling of the five criteria set forth for compression methods in core based SOCs. By providing compression in a way that does not require any internal details the method is particularly suitable for SOCs containing IP-protected cores. The structured way in which compression is done, and in which the decompression hardware is generated enables smooth integration into existing tool flows making this method conducive to automation. Furthermore, the scalability considerations which have been provided will enable this method to be applied to SOCs of the future, not just the present and the past. Moreover, movement toward widespread use of the core based SOC paradigm has received recent attention through an initiative to standardize test data compression [24]. Because of the features provided by StreamBIST / StreamPack, the compression methodology proposed in this thesis fits very well in a standardized framework and thus has significant potential for applicability to designs conforming to potentially emerging standards. Furthermore, although our compression method does not rely on some techniques (e.g., fault simulation) which compromise the provision any of the five discussed features, our method does not inherently exclude these techniques. As a result, these techniques could be applied to our method to further reduce the volume of test data in the case that some features are not required. Having discussed the advantages of this work, we discuss its limitations in the next section.

4.2 Limitations

As with all things in design, there is always room for improvement. Test data compression methods can be summarized by the four fundamental aspects of compression achieved, area overhead, CPU runtime and the features they provide. Having exhausted discussion on the features provided, focus will be given here to how the other three aspects may be improved.

The issue of CPU runtime was addressed in Chapter 3 and notice was made that the complexity of StreamBIST can lead to excessive runtimes in some cases and as such, there is obvious room for improvement here. As a brief example, the branch and bound portion of StreamBIST for seed-mux is a clear processing bottleneck. Instead of approaching each block naively frame by frame, equations could be added according to number of care bits since those with more are likely to be the source of lockout. In this way, the solution space of the branch and bound algorithm is pruned more effectively leading to shorter runtimes. Other aspects of the overall runtime may be reduced in a similar way, by development of more sophisticated algorithms which exploit a feature of the tasks to which they are applied.

Turning attention to area overhead, we have seen that at the system level control overhead is miniscule compared to the overhead brought about by the core level decompressors. Thus although there is not much room for improvement on the control end, area may be improved by focusing on the core level decompressor. Again as a single example, the phase shifter within the core level decompressor which consists of many XOR gates can play a significant role in the overhead of the entire decompressor. By exploiting overlap in the linear expressions which describe the channels of the phase shifter portions of the network may be shared and thus the overall area reduced. As before, other opportunities for area improvement exist as well.

With respect to compression, comparison to prior work in the previous chapter has made it clear that improvement can be made in terms of VTD. The challenge is to provide improvement without compromising any features. One way in particular that this could be achieved is by mutual integration of StreamBIST and SteamPack. By providing feedback between the two processes, reseed positions could be recalculated as needed, thus reducing the insertion of NOOPs and improving test time and VTD. This comes with its own challenges, and would require significant effort. Another avenue of opportunity for improved VTD is multi-level compression as discussed in the next section.

4.3 Future Work

With the list of limitations just described, opportunities for future work are abundant. In addition to improvement in the aspects of area, CPU runtime and compression ratio mentioned, practical opportunities exist as well. The concept of multi-level compression as mentioned in the last section provides opportunities for future work by asking what can be done once the limit of care bits = seed bits imposed by linear decompressors is reached. Multi-level approaches, for example LFSR seed compression ([59]), have been proposed before but implemented and verified solutions are not easily obtained. We have seen that the volume of test data continues to scale much faster than the equipment used to deliver it, and so increased levels of compression will be required just to meet VTD demands, to say nothing of reducing cost. Furthermore, by tackling multi-level compression embedded test technology can be enabled through pushing of the complexity of the test generation and application process into powerful on-chip *and* off-chip computing engines. In light of these challenges and opportunities, it is apparent that there exist sufficient problems for test data compression to remain an active area of research well into the foreseeable future.

Bibliography

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design.* IEEE Press, 1990.

[2] A. A. Al-Yamani and E. J. McCluskey. Built-in Reseeding for Serial BIST. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 63 – 68, 2003.

[3] A. A. Al-Yamani, S. Mitra, and E. J. McCluskey. Optimized Reseeding by Seed Ordering and Encoding. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(2):264 – 270, February 2005.

[4] M. Arai, H. Kurokawa, K. Ichino, S. Fukumoto, and K. Iwasaki. Seed Selection Procedure for LFSR-Based BIST with Multiple Scan Chains and Phase Shifters. In *Proc. IEEE Asian Test Symposium (ATS)*, pages 190 – 195, 2004.

[5] K. J. Balakrishnan and N. A. Touba. Improving Encoding Efficiency for Linear Decompressors using Scan Inversion. In *Proc. IEEE International Test Conference (ITC)*, pages 936 – 944, 2004.

[6] K. J. Balakrishnan and N. A. Touba. Relating Entropy Theory to Test Data Compression. In *Proc. IEEE European Test Symposium (ETS)*, pages 94 – 99, 2004.

[7] P. H. Bardell, W. H. McAnney, and J. Savir. *Built-In Self-Test - Pseudorandom Techniques.* John Wiley & Sons, 1987.

[8] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, A. Ferko, B. Keller, D. Scott, B. Koenemann, and T. Onodera. Extending OPMISR Beyond 10x

Scan Test Efficiency. *IEEE Design and Test of Computers*, 19(5):65 – 73, September 2002.

[9] I. Bayraktaroglu and A. Orailoglu. Test Volume and Application Time Reduction through Scan Chain Concealment. In *Proc. IEEE/ACM Design Automation Conference (DAC)*, pages 151 – 155, 2001.

[10] I. Bayraktaroglu and A. Orailoglu. Decompression Hardware Determination for Test Volume and Time Reduction through Unified Test Pattern Compaction and Compression. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 113 – 118, 2003.

[11] Y. Bonhomme, T. Yoneda, H. Fujiwara, and P. Girard. An Efficient Scan Tree Design for Test Time Reduction. In *Proc. IEEE European Test Symposium (ETS)*, pages 174 – 179, 2004.

[12] F. Brglez, D. Bryan, and K. Kozminski. Combinational Profiles of Sequential Benchmark Circuits. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1929 – 1934, 1989.

[13] M. Bushnell and V. D. Agrawal. *Essentials of Electronic Testing.* Kluwer Academic Publishers, 2000.

[14] K. Chakrabarty, B. T. Murray, and V. Iyengar. Deterministic Built-in Test Pattern Generation for High-Performance Circuits using Twisted-Ring Counters. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(5):633 – 636, October 2000.

[15] A. Chandra and K. Chakrabarty. Combining Low-Power Scan Testing and Test Data Compression for System-on-a-Chip. In *Proc. IEEE/ACM Design Automation Conference (DAC)*, pages 166 – 169, 2001.

[16] A. Chandra and K. Chakrabarty. System-on-a-Chip Test-Data Compression and Decompression Architectures Based on Golomb Codes. *IEEE Transactions*

on Computer-Aided Design of Integrated Circuits and Systems, 20(3):355 – 368, March 2001.

[17] A. Chandra and K. Chakrabarty. Test Resource Partitioning for SOCs. *IEEE Design and Test of Computers*, 18(5):80 – 91, September 2001.

[18] A. Chandra and K. Chakrabarty. Test Data Compression and Decompression Based on Internal Scan Chains and Golomb Coding. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(6):715 – 722, June 2002.

[19] A. Chandra and K. Chakrabarty. Test Resource Partitioning and Reduced Pin-Count Testing Based on Test Data Compression. In *Proc. IEEE/ACM Design, Automation and Test in Europe (DATE)*, pages 598 – 603, 2002.

[20] A. Chandra and K. Chakrabarty. A Unified Approach to Reduce SOC Test Data Volume, Scan Power and Testing Time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(3):352 – 363, March 2003.

[21] A. Chandra and K. Chakrabarty. Test Data Compression and Test Resource Partitioning for System-on-a-Chip using Frequency-Directed Run-Length (FDR) Codes. *IEEE Transactions on Computers*, 52(8):1076 – 1088, August 2003.

[22] D. Das and N. A. Touba. Reducing Test Data Volume using External/LBIST Hybrid Test Patterns. In *Proc. IEEE International Test Conference (ITC)*, pages 115 – 122, 2000.

[23] R. Dorsch and H. J. Wunderlich. Tailoring ATPG for Embedded Testing. In *Proc. IEEE International Test Conference (ITC)*, pages 530 – 537, 2001.

[24] EE times - Dylan McGrath. Accelera Exploring Open Compression Standard. http://www.eetimes.com/news/design/showArticle.jhtml;jsessionid= 1XUW5U2KFKN0OQSNDBCSKHSCJUMEKJVN?articleID=162800232, 2005.

[25] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Morgan Freeman, New York, 1979.

[26] V. Gherman, H.-J. Wunderlich, H. Vranken, F. Hapke, M. Wittke, and M. Garbers. Efficient Pattern Mapping for Deterministic Logic BIST. In *Proc. IEEE International Test Conference (ITC)*, pages 48 – 56, 2004.

[27] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolici. Improving Compression Ratio, Area Overhead, and Test Application Time for System-on-a-Chip Test Data Compression/Decompression. In *Proc. IEEE/ACM Design, Automation and Test in Europe (DATE)*, pages 604 – 611, 2002.

[28] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolici. Synchronization Overhead in SOC Compressed Test. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(1):140 – 152, January 2005.

[29] Gordon E. Moore. Moore's Law. http://www.intel.com/technology/silicon/mooreslaw/index.htm, 1965.

[30] I. Hamzaoglu and J. H. Patel. Test Set Compaction Algorithms for Combinational Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(8):957 – 963, August 2000.

[31] H. Hashempour and F. Lombardi. ATE-Amenable Test Data Compression with no Cyclic Scan Registers. In *Proc. IEEE Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 151 – 158, 2003.

[32] H. Hashempour, L. Schiano, and F. Lombardi. Error-Resilient Test Data Compression using Tunstall Codes. In *Proc. IEEE Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 316 – 323, 2004.

[33] F. F. Hsu, K. M. Butler, and J. H. Patel. A Case Study on the Implementation of the Illinois Scan Architecture. In *Proc. IEEE International Test Conference (ITC)*, pages 538 – 547, 2001.

[34] H. Ichihara, M. Ochi, M. Shintani, and T. Inoue. A Test Decompression Scheme for Variable-Length Coding. In *Proc. IEEE Asian Test Symposium (ATS)*, pages 426 – 431, 2004.

[35] IEEE 1500. IEEE 1500 Core Test Standard. http://grouper.ieee.org/groups/1500/, 2005.

[36] ITRS. International Technology Roadmap for Semiconductors. http://www.itrs.net/Common/2004Update/2004Update.htm, 2004.

[37] V. Iyengar and A. Chandra. A Unified SOC Test Approach Based on Test Data Compression and TAM Design. In *Proc. IEEE Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 511 – 518, 2003.

[38] V. Iyengar, A. Chandra, S. Schweizer, and K. Chakrabarty. A Unified Approach for SOC Testing using Test Data Compression and TAM Optimization. In *Proc. IEEE/ACM Design, Automation and Test in Europe (DATE)*, pages 1188 – 1189, 2003.

[39] A. Jas, J. Ghosh-Dastidar, M.-E. Ng, and N. A. Touba. An Efficient Test Vector Compression Scheme using Selective Huffman Coding. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):797 – 806, June 2003.

[40] A. Jas, C. V. Krishna, and N. A. Touba. Weighted Pseudorandom Hybrid BIST. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(12):1277 – 1283, December 2004.

[41] A. Jas, B. Pouya, and N. A. Touba. Test Data Compression Technique for Embedded Cores using Virtual Scan Chains. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(7):775 – 781, July 2004.

[42] D. Kagaris. Multiple-Seed TPG Structures. *IEEE Transactions on Computers*, 52(12):1633 – 1639, December 2003.

[43] E. Kalligeros, X. Kavousianos, and D. Nikolos. Multiphase BIST: A New Reseeding Technique for High Test-Data Compression. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(10):1429 – 1446, October 2004.

[44] X. Kavousianos, D. Bakalis, D. Nikolos, and S. Tragoudas. A New Built-In TPG Method for Circuits with Random Pattern Resistant Faults. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(7):859 – 866, July 2002.

[45] D. Kay, S. Chung, and S. Mourad. Embedded Test Control Schemes for Compression in SOCs. In *Proc. IEEE/ACM Design Automation Conference (DAC)*, pages 679 – 684, 2002.

[46] D. Kay and S. Mourad. Compression Technique for Interactive BIST Application. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 103 – 108, 2001.

[47] A. Khoche, E. Volkerink, J. Rivoir, and S. Mitra. Test Vector Compression using EDA-ATE Synergies. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 97 – 102, 2002.

[48] G. Kiefer, H. Vranken, E. J. Marinissen, and H.-J. Wunderlich. Application of Deterministic Logic BIST on Industrial Circuits. In *Proc. IEEE International Test Conference (ITC)*, pages 105 – 114, 2000.

[49] H.-S. Kim, Y. Kim, and S. Kang. Test-Decompression Mechanism using a Variable-Length Multiple-Polynomial LFSR. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(4):687 – 690, August 2003.

[50] A. B. Kinsman, J. I. Hewitt, and N. Nicolici. Embedded Compact Deterministic Test for IP-Protected Cores. In *Proc. IEEE Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 519 – 526, 2003.

[51] A. B. Kinsman and N. Nicolici. Time-Multiplexed Test Data Decompression Architecture for Core-Based SOCs with Improved Utilization of Tester Channels. In *Proc. IEEE European Test Symposium (ETS)*, pages 196 – 201, 2005.

[52] M. J. Knieser, F. G. Wolff, C. A. Papachristou, D. J. Weyer, and D. R. McIntyre. A Technique for High Ratio LZW Compression. In *Proc. IEEE/ACM Design, Automation and Test in Europe (DATE)*, pages 116 – 121, 2003.

[53] H. F. Ko and N. Nicolici. Functional Illinois Scan Design at RTL. In *Proc. IEEE International Conference on Computer Design (ICCD)*, pages 78 – 81, 2004.

[54] B. Koenemann. LFSR-Coded Test Patterns for Scan Designs. In *Proc. IEEE European Test Conference (ETC)*, pages 237 – 242, 1991.

[55] B. Koenemann. Care Bit Density and Test Cube Clusters: Multi-Level Compression Opportunities. In *Proc. IEEE International Conference on Computer Design (ICCD)*, pages 320 – 325, 2003.

[56] B. Koenemann. STAGE: A Decoding Engine Suitable for Multi-Compressed Test Data. In *Proc. IEEE Asian Test Symposium (ATS)*, pages 142 – 145, 2003.

[57] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater. A SmartBIST Variant with Guaranteed Encoding. In *Proc. IEEE Asian Test Symposium (ATS)*, pages 325 – 330, 2001.

[58] C. V. Krishna, A. Jas, and N. A. Touba. Test Vector Encoding using Partial LFSR Reseeding. In *Proc. IEEE International Test Conference (ITC)*, pages 885 – 893, 2001.

[59] C. V. Krishna and N. A. Touba. Reducing Test Data Volume using LFSR Reseeding with Seed Compression. In *Proc. IEEE International Test Conference (ITC)*, pages 321 – 330, 2002.

[60] C. V. Krishna and N. A. Touba. Adjustable Width Linear Combinational Scan Vector Decompression. In *Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 863 – 866, 2003.

[61] C. V. Krishna and N. A. Touba. Hybrid BIST using an Incrementally Guided LFSR. In *Proc. IEEE Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 217 – 224, 2003.

[62] L. Lai, T. Rinderknecht, W.-T. Cheng, and J. H. Patel. Logic BIST using Constrained Scan Cells. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 199 – 205, 2004.

[63] L. Li and K. Chakrabarty. Test Data Compression using Dictionaries with Fixed-Length Indices. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 219 – 224, 2003.

[64] L. Li and K. Chakrabarty. Test Set Embedding for Deterministic BIST using a Reconfigurable Interconnection Network. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(9):1289 – 1305, September 2004.

[65] H.-G. Liang, S. Hellebrand, and H.-J. Wunderlich. Two-Dimensional Test Data Compression for Scan-Based Deterministic BIST. In *Proc. IEEE International Test Conference (ITC)*, pages 894 – 902, 2001.

[66] S. Mitra and K. S. Kim. XMAX: X-Tolerant Architecture for MAXimal Test Compression. In *Proc. IEEE International Conference on Computer Design (ICCD)*, pages 326 – 330, 2003.

[67] S. Mitra and K. S. Kim. X-Compact: An Efficient Response Compaction Technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(3):421 – 432, March 2004.

[68] K. Miyase, S. Kajihara, and S. M. Reddy. Multiple Scan Tree Design with Test Vector Modification. In *Proc. IEEE Asian Test Symposium (ATS)*, pages 76 – 81, 2004.

[69] G. Mrugalski, N. Mukherjee, J. Rajski, and J. Tyszer. Planar High Performance Ring Generators. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 193 – 198, 2004.

[70] G. Mrugalski, J. Rajski, and J. Tyszer. Cellular Automata-Based Test Pattern Generators with Phase Shifters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(8):878 – 893, August 2000.

[71] G. Mrugalski, J. Rajski, and J. Tyszer. Ring Generators - New Devices for Embedded Test Applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(9):1306 – 1320, September 2004.

[72] N. Oh, R. Kapur, T. W. Williams, and J. Sproch. Test Pattern Compression using Prelude Vectors in Fan-Out Scan Chain with Feedback Architecture. In *Proc. IEEE/ACM Design, Automation and Test in Europe (DATE)*, pages 110 – 115, 2003.

[73] K. Parker. *The Boundary-Scan Handbook.* Kluwer Academic Publishers, Boston, second edition, 1998.

[74] I. Pomeranz. Reducing Test-Data Volume using P-Testable Scan Chains in Circuits with Multiple Scan Chains. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(10):1465 – 1478, October 2004.

[75] I. Pomeranz and S. M. Reddy. Test Data Compression Based on Input-Output Dependence. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(10):1450 – 1455, October 2003.

[76] I. Pomeranz and S. M. Reddy. Static Test Compaction for Full-Scan Circuits Based on Combinational Test Sets and Nonscan Input Sequences and a Lower Bound on the Number of Tests. *IEEE Transactions on Computers*, 53(12):1569 – 1581, December 2004.

[77] J. Rajski. DFT for High-Quality Low Cost Manufacturing Test. In *Proc. IEEE Asian Test Symposium (ATS)*, pages 3 – 8, 2001.

[78] J. Rajski, M. Kassab, N. Mukherjee, N. Tamarapalli, J. Tyszer, and J. Qian. Embedded Deterministic Test for Low-Cost Manufacturing. *IEEE Design and Test of Computers*, 20(5):58 – 66, Sept.-Oct. 2003.

[79] J. Rajski, N. Tamarapalli, and J. Tyszer. Automated Synthesis of Phase Shifters for Built-In Self-Test Applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(10):1175 – 1188, October 2000.

[80] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee. Embedded Deterministic Test. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(5):776 – 792, May 2004.

[81] J. Rajski, J. Tyszer, C. Wang, and S. M. Reddy. Finite Memory Test Response Compactors for Embedded Test Applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(4):622 – 634, April 2005.

[82] W. Rao, I. Bayraktaroglu, and A. Orailoglu. Test Application Time and Volume Compression through Seed Overlapping. In *Proc. IEEE/ACM Design Automation Conference (DAC)*, pages 732 – 737, 2003.

[83] W. Rao, A. Orailoglu, and G. Su. Frugal Linear Network-Based Test Decompression for Drastic Test Cost Reductions. In *Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 721 – 725, 2004.

[84] S. Reda and A. Orailoglu. Reducing Test Application Time through Test Data Mutation Encoding. In *Proc. IEEE/ACM Design, Automation and Test in Europe (DATE)*, pages 387 – 393, 2002.

[85] S. Samaranayake, E. Gizdarski, N. Sitchinava, F. Neuveux, R. Kapur, and T. W. Williams. A Reconfigurable Shared Scan-In Architecture. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 9 – 14, 2003.

[86] C. Schuermyer, J. Ruffler, R. Daasch, and R. Madge. Minimum Testing Requirements to Screen Temperature Dependent Defects. In *Presentation of paper in Proc. IEEE International Test Conference (ITC)*, pages 300 – 308, 2004.

[87] Y. Shi, S. Kimura, N. Togawa, M. Yanagisawa, and T. Ohtsuki. Alternative Run-Length Coding through Scan Chain Reconfiguration for Joint Minimization of Test Data Volume and Power Consumption in Scan Test. In *Proc. IEEE Asian Test Symposium (ATS)*, pages 432 – 437, 2004.

[88] O. Sinanoglu and A. Orailoglu. Test Data Manipulation Techniques for Energy-Frugal, Rapid Scan Test. In *Proc. IEEE Asian Test Symposium (ATS)*, pages 202 – 207, 2003.

[89] N. Sitchinava, E. Gizdarski, S. Samaranayake, F. Neuveux, R. Kapur, and T. W. Williams. Changing the Scan Enable During Shift. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 73 – 78, 2004.

[90] C. Stroud, J. Sunwoo, S. Garimella, and J. Harris. Built-In Self-Test for System-on-Chip: A Case Study. In *Proc. IEEE International Test Conference (ITC)*, pages 837 – 846, 2004.

[91] X. Sun, L. Kinney, and B. Vinnakota. Combining Dictionary Coding and LFSR Reseeding for Test Data Compression. In *Proc. IEEE/ACM Design Automation Conference (DAC)*, pages 944 – 947, 2004.

[92] Synopsys Synthesis Tools. Design Compiler. http://www.synopsys.com/products/logic/design_compiler.html, 2003.

[93] Taiwan Semiconductor Manufacturing Corporation. TSMC Website. http://www.tsmc.com.

[94] H. Tang, S. M. Reddy, and I. Pomeranz. On Reducing Test Data Volume and Test Application Time for Multiple Scan Chain Designs. In *Proc. IEEE International Test Conference (ITC)*, pages 1079 – 1088, 2003.

[95] M. Tehranipour, M. Nourani, and K. Chakrabarty. Nine-Coded Compression Technique with Application to Reduced Pin-Count Testing and Flexible On-Chip Decompression. In *Proc. IEEE/ACM Design, Automation and Test in Europe (DATE)*, volume 2, pages 1284 – 1289, 2004.

[96] N. A. Touba. Circular BIST with State Skipping. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(5):668 – 672, October 2002.

[97] N. A. Touba and E. J. McCluskey. Bit-Fixing in Pseudorandom Sequences for Scan BIST. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(4):545 – 555, April 2001.

[98] Verilog. IEEE 1364-2001 - Verilog 2001. Verilog, 2001.

[99] VHDL. IEEE 1076a - VHDL-00. VHDL, 2000.

[100] E. H. Volkerink and S. Mitra. Efficient Seed Utilization for Reseeding Based Compression. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 232 – 237, 2003.

[101] 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. IEEE International Test Conference (ITC)*, pages 1069 – 1078, 2003.

[102] L.-T. Wang, X. Wen, H. Furukawa, F.-S. Hsu, S.-H. Lin, S.-W. Tsai, K. S. Abdel-Hafez, and S. Wu. VirtualScan: A New Compressed Scan Technology for Test Cost Reduction. In *Proc. IEEE International Test Conference (ITC)*, pages 916 – 925, 2004.

[103] J. Wingfield, J. Dworak, and M. R. Mercer. Function-Based Dynamic Compaction and its Impact on Test Set Sizes. In *Proc. IEEE Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 167 – 174, 2003.

[104] P. Wohl, J. A. Waicukauski, and S. Patel. Scalable Selector Architecture for X-Tolerant Deterministic BIST. In *Proc. IEEE/ACM Design Automation Conference (DAC)*, pages 934 – 939, 2004.

[105] P. Wohl, J. A. Waicukauski, S. Patel, and M. B. Amin. Efficient Compression and Application of Deterministic Patterns in a Logic BIST Architecture. In *Proc. IEEE/ACM Design Automation Conference (DAC)*, pages 566 – 569, 2003.

[106] P. Wohl, J. A. Waicukauski, and T. W. Williams. Design of Compactors for Signature-Analyzers in Built-In Self-Test. In *Proc. IEEE International Test Conference (ITC)*, pages 54 – 63, 2001.

[107] F. G. Wolff and C. Papachristou. Multiscan-Based Test Compression and Hardware Decompression using LZ77. In *Proc. IEEE International Test Conference (ITC)*, pages 331 – 339, 2002.

[108] F. G. Wolff, C. Papachristou, and D. R. McIntyre. Test Compression and Hardware Decompression for Scan-Based SOCs. In *Proc. IEEE/ACM Design, Automation and Test in Europe (DATE)*, volume 1, pages 716 – 717, 2004.

[109] A. Wurtenberger, C. S. Tautermann, and S. Hellebrand. A Hybrid Coding Strategy for Optimized Test Data Compression. In *Proc. IEEE International Test Conference (ITC)*, pages 451 – 459, 2003.

[110] A. Wurtenberger, C. S. Tautermann, and S. Hellebrand. Data Compression for Multiple Scan Chains using Dictionaries with Corrections. In *Proc. IEEE International Test Conference (ITC)*, pages 926 – 935, 2004.

[111] D. Xiang, M.-J. Chen, K.-W. Li, and Y.-L. Wu. Scan-Based BIST using an Improved Scan Forest Architecture. In *Proc. IEEE Asian Test Symposium (ATS)*, pages 88 – 93, 2004.

[112] T. J. Yamaguchi, D. S. Ha, M. Ishida, and T. Ohmi. A Method for Compressing Test Data Based on Burrows-Wheeler Transformation. *IEEE Transactions on Computers*, 51(5):486 – 497, May 2002.

[113] G. Zeng and H. Ito. Non-Intrusive Test Compression for SOC using Embedded FPGA Core. In *Proc. IEEE Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 413 – 421, 2004.

Index