Lehrstuhl für Rechnerarchitektur
Institut für Technische Informatik

Efficient Programmable Deterministic Self-Test

Abdul-Wahid Hakmi
aus Gujrat/Pakistan

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Hauptberichter
Prof. Dr. habil. Hans-Joachim Wunderlich
Mitberichter
Prof. Dr.-Ing. Heinrich Theodor Vierhaus

Tag der mündlichen Prüfung: 13 August 2010

Institut für Technische Informatik 2010

Universität Stuttgart

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Hilfsmittel und Literatur angefertigt habe.

Stuttgart, den 16.08.2010 Abdul-Wahid Hakmi

With the Name of Allah, the Most Gracious, the Most Merciful

. . . . . .

Read in the Name of thy Lord who created,

created Man from a blood-clot.

Read, and thy Lord is the Most Generous,

who taught by the pen,

taught man that he knew not.

(Al-Quran, 96:1-5)

Dedicated to my beloved parents.
It is only under the shadow of their prayers
that I have achieved this milestone.

Acknowledgments

It is a pleasure to thank those who made this work possible. First and foremost I offer my sincerest gratitude to my supervisor, Professor Hans-Joachim Wunderlich, who has supported me throughout my work with his patience and knowledge whilst allowing me the room to work in my own way.

I am very grateful to Professor Heinrich Theodor Vierhaus[1] for his survey of this work. I would also like to mention the precious technical support of Friedich Hapke[2], Juergen Schloeffel[2], Michael Garbers[2], Andreas Glowatz[2] and Laurent Souef[2].

I am indebted to many of my current and former colleagues for our discussions and exchange of knowledge. My heartiest thanks go to Stefan Holst who has made available his support in a number of ways. Indeed his invaluable suggestions shaped this thesis in the right form. One simply could not wish for a better or friendlier colleague. I am also very grateful to Christian Zoellin and Valentin Gherman for their worthful help in this work. Mirjam Breitling, Michael Imhof, Michael Kochte, Melanie Elm, Rafal Baranowski, Claus Braun, Tobias Bergmann, Günter Bartsch, Karin Angela, Alexandra Wiedmann, Nicoleta Pricopi, Talal Arnaout, I am thankful for your continuous support.

I would like to show my gratitude to dear friends Abdullah Mumtaz, Haroon Ahmed Khan and Owais Sami for their help and assistance in writing down the manuscript.

This work would not have been possible without the great support of my family. I would like to thank my family for the endless love, understandings and encouragement that accompany me through difficulties all along.

[1]Brandenburg University of Technology at Cottbus
[2]NXP Semiconductors GmbH, in Hamburg

Abstract

In modern times, integrated circuits (ICs) are used in almost all electronic equipment ranging from household appliances to space shuttles and have revolutionized the world of electronics. Continuous reductions in the manufacturing costs as well as the size of this technology have allowed the development of very sophisticated ICs for common use. Post fabrication testing is necessary for each IC in order to ensure the quality and the safety of human life. The improvement in technology as well as economies of scale are continuously reducing fabrication costs. On the other hand, the increasing complexity of circuits is leading to higher test costs. These increasing test costs affect the market price of a chip.

A test set is a set of binary patterns that are applied on the circuit inputs to detect the potential faults. Only a small number of bits in a test set are specified to *0* or *1* called *care bits* while other bits called *don't care bits* may assume random values. Test sets volume is characterized by the number of patterns as well as the size of each pattern in a test set. The increasing number of gates in nanometer ICs has resulted in an explosive increase in test sets volume. This increase in test sets volume is the major cause for rapidly growing test costs. An IC is tested either by using an automatic test equipment (ATE) or with the help of special hardware added on-chip that performs a self-test. These two approaches as well as their hybrid derivatives offer various trade-offs in test costs, quality, reliability and test time. In ATE testing high test sets volume leads to the requirement of expensive testers with large storage capacity while in self-test it results in significant hardware overhead.

A test set is highly compressible due to the presence of a large number of *don't care bits*. The Test data compression techniques are used to limit test sets volume and hence the involved test cost. These compressed test sets are applicable to both ATE and Self-test methodologies. Compression of a test set depends on its statistical attributes such as the percentage and the distribution of *care bits*. The available test compression schemes assume that all the test sets have similar statistical attributes which is not always true. These attributes vary considerably among various test sets depending on the circuit structure and the targeted trade-offs. To get optimized reduction in test sets volume, test sets with different statistical attributes have to be addressed separately.

In this work we analyze various test sets of industrial circuits and categorize them into three classes based on their statistical attributes. By examining each class differently, three novel compression methods and decompression architectures are proposed. The proposed test compression methods are equally adaptable in ATE testing and self-test. Three low cost programmable self-test schemes offering various trade-offs in testing are developed by applying these methods.

The experimental results obtained with the test sets of large industrial circuits show that the proposed compression methods reduce storage requirements by more than half compared to the most efficient available methods. First time in literature the total number of bits in a compressed test set are lesser than the number of care bits in the original test set. The additional advantages of proposed methods include guaranteed encoding, significant reduction in decompression time overhead and programmability of decompression hardware.

Zusammenfassung

Heute haben Integrierte Schaltkreise (ICs) in fast allen elektronischen Geräten vom Haushalt bis hin zur Raumfahrt Einzug gehalten und die Welt der Elektronik revolutioniert. Ständige Senkungen der Herstellungskosten bei gleichzeitiger Miniaturisierung erlauben die Entwicklung von komplexen ICs für den jeden Bedarf. Jeder produzierte Chip muss getestet werden, um die Qualität sicherzustellen und menschliches Leben nicht zu gefährden. Die Verbesserung der Fabrikationstechnologie sowie die Gesetze der Skalierung reduzieren kontinuierlich die Herstellungskosten. Auf der anderen Seite führt die zunehmende Komplexität der Schaltungen zu höheren Testkosten. Diese zunehmenden Kosten schlagen sich im Marktpreis eines Chips nieder.

Ein Test ist eine Menge von binären Mustern, die an die Schaltungseingänge angelegt werden, um potentielle Fehler zu erkennen. Nur eine kleine Anzahl von Bits in einem Testmuster sind auf 0 oder 1 spezifiziert (sog. Care-Bits), während die anderen Bits (sog. Don't-Care Bits) beliebige Werte annehmen können. Das Testdatenvolumen ist durch die Anzahl von Mustern und die Größe eines einzelnen Musters gegeben. Die zunehmende Anzahl von Gattern in Nanometer-ICs hat zu einem explosiven Anstieg dieses Testdatenvolumens geführt. Dieser Anstieg des Datenvolumens ist die Hauptursache für die rasch wachsenden Testkosten. Ein IC wird entweder von einem externen Testgerät (Automatic Test Equipment, ATE) getestet oder der Chip führt einen Selbst-Test mit Hilfe von speziellen Hardwarestrukturen durch. Diese beiden Ansätze sowie deren Kombinationen bieten zahlreiche Möglichkeiten, Testkosten, Qualität, Zuverlässigkeit und Testzeit gegeneinander abzuwägen. Beim externen Test werden bei hohem Testdatenvolumen teure Testgeräte mit viel Speicher benötigt, während ein hohes Datenvolumen beim Selbst-Test einen erheblichen Hardware-Overhead nach sich zieht.

Eine Testmenge kann stark komprimiert werden, da sie viele nicht spezifizierte Bits enthält. Die Techniken zur Testdatenkompression werden verwendet, um das Testdatenvolumen zu verringern und damit die anfallenden Testkosten zu senken. Sie können sowohl auf den externen Test als auch beim Selbst-Test angewendet werden. Die Kompression einer Testmenge hängt von ihren statistischen Eigenschaften wie dem prozentualen Anteil und der Verteilung der spezifizierten Bits ab. Die verfügbaren Test-Kompressionsverfahren gehen davon aus, dass alle Testmengen ähnliche statistische Eigenschaften haben, was jedoch nicht immer der Fall ist. Die Attribute unterscheiden sich erheblich zwischen verschiedenen Testmengen je nach Schaltungsstruktur und den angesprochenen Abwägungen. Für eine optimale Reduzierung des Testdatenvolumens müssen Testmengen mit unterschiedlichen statistischen Eigenschaften getrennt behandelt werden.

Diese Arbeit untersucht verschiedene Testmustermengen von industriellen Schaltungen und kategorisiert sie anhand ihrer statistischen Eigenschaften in drei Klassen. Durch die getrennte Untersuchung jeder Klasse werden drei neue Kompressionsmethoden und Dekompressions-Architekturen entwickelt. Die vorgeschlagenen Methoden sind auf den externen Test als auch auf den Selbst-Test gleichermaßen anwendbar. Durch die Anwendung dieser Methoden werden drei günstige, programmierbare Selbst-Test Verfahren vorgestellt, die unterschiedliche Ziele und Schwerpunkte bedienen.

Die experimentellen Ergebnisse mit Testmustern für große industrielle Schaltungen zeigen, dass die vorgeschlagenen Kompressionsverfahren den Speicherbedarf verglichen mit den besten bekannten Methoden nochmals auf weniger als die Hälfte reduzieren. Zum ersten Mal in der Literatur konnte die Zahl der Bits in der komprimierten Testmenge unter die Zahl der spezifizierten Bits der ursprünglichen Testmenge gedrückt werden. Die zusätzlichen Vorteile der vorgeschlagenen Methoden sind die garantierte Kodierung, die signifikante Reduzierung der Zeit-Overheads und Programmierbarkeit der Dekompressions-Hardware.

Contents

List of Figures

List of Tables

List of Abbreviations

$\oplus$	XOR operator
CP_{max}	Maximum compression percentage
RV	Set of restrict vectors stored in memory
V_P	Vector set containing all the unique vectors in test set P
AND	Boolean conjunction operator
ATE	Automatic test equipment
ATPG	Automatic test pattern generation
BIST	Built-in self-test
BOST	Built-out self-test
CP	Compression percentage
CPU	Central processing unit
CR	Compression ratio
CUT	Circuit under test
DSP	Digital signal processor
e	Encoding efficiency of reseeding
EE	Encoding efficiency
FC	Fault coverage
FE	Fault efficiency
FPGA	Floating point gate array
FSM	Finite state machine

H	Entropy
IC	Integrated circuit
IR	Instruction register
k	Number of scan chains in a circuit
LFSR	Linear feedback shift register
MISR	Multiple input signature register
NAND	Boolean negation and conjunction operator
NCR	Nearly complete reseeding
OR	Boolean disjunction operator
P	Test set
PLL	Phase lock loop
PRPG	Pseudo random pattern generator
q	Number of test patterns in test set P
RE	Restrict encoding
REWPD	Run-length encoding with parallel decompression
SOC	System-on-chip
STUMPS	Self-test using MISR and parallel shift register sequence generator
t	Maximum scan chain length
TAP	Test access port
TCK	Test clock
TDI	Test data input
TDO	Test data output
TMS	Test mode select
TRP	Test resource partitioning
V	Set of test vectors

v	Test vector
X	Don’t care bit in a test pattern
XOR	Boolean exclusive-OR operator

CHAPTER 1

Introduction

1.1 Motivation and Goals of this work

Defects may be introduced during the fabrication of *integrated circuits* (IC) due to unavoidable flaws in materials and manufacturing process. These manufacturing defects can cause a malfunction in the circuit and result in system failure. Every IC needs to be tested after fabrication in order to ensure that only a good circuit is delivered to the customer. This test has to be comprehensive for the ICs used in safety critical and mission critical systems. The advancement of semiconductor industry into the age of nanometer technology has resulted in an explosive increase in gate count and complexity of integrated circuits. These nanometer ICs have brought new test challenges from the chip level to the board and system level which has made *testing* one of the important issues in the design and development of nanometer *system-on-chip* (SOC).

To test a circuit, a binary test pattern is applied on its inputs and the response on the outputs is compared with the expected response of a fault free circuit. Timing related faults need a two pattern test where the first pattern sets initial values and the second pattern propagates the circuit response to the outputs. In order to ensure high test quality particular test patterns based on circuit structural information have to be applied. *Fault models*, reflecting the behavior of defects in a circuit, are used to generate these patterns. No single fault model accurately represents the behavior of all possible defects that can occur, so a combination of different fault models is often employed in the generation and evaluation of test patterns. The set of all potential faults of a specific fault model and the set of patterns generated to detect them are termed as *fault set* and *deterministic test set* respectively. Only a few inputs of a circuit need to be assigned a *0* or *1* to detect a fault while other inputs may assume random values. The bits specified to *0* and *1* in a test pattern are called *care bits* and the remaining bits represented with an *X* are called *don't care* bits.

Sizes of fault sets and in turn deterministic test sets are increasing rapidly with circuit complexity. Beside this, small feature sizes of modern devices lead to many new defect mechanisms. The representation of these defects needs new complex fault models and their detection requires large deterministic test sets with many more care bits. These facts have resulted in an explosive increase in test sets volume. According to international technology road map for semiconductors 2007 [ITRS 07], the test sets volume will increase up to 120 times during next decade.

The *automatic test equipment* (ATE) testing and the self-test are two major approaches to perform test out of which many hybrid schemes are derived to achieve different trade-offs in test quality, cost and time-to-market. In ATE testing, deterministic test sets are stored in the memory of an ATE and during the test the patterns are transmitted to the *circuit under test* (CUT) and the responses are read back for comparison with expected responses. Bandwidth between ATE and CUT is restrained by slower speed of testers and their limited number of I/O pins. In self-test, the test functionality is implemented in hardware. This hardware performs an autonomous test and delivers a pass/fail judgment after completion of a test. Here a deterministic test set is stored either in a memory or as a logic function.

The self-test may provide advantages over ATE testing in terms of test time, quality and reliability. Since self-test can be performed at circuit speed, test time is reduced and timing related fault detection is improved. Unlike ATE testing, the test hardware is available in the field. This test hardware can be used for periodic self-test in order to enhance the reliability of circuits and the entire system. If a circuit fails the in-field test, same test hardware can be used to perform an economical diagnosis and repair [Wang 08a].

The rapid growth in test sets volume is causing a rapid increase in the capital and operational costs to test nanometer ICs. In ATE testing, expensive ATE with additional memory depth per I/O pin is required to store high test sets volume and long test time per IC is needed to apply this data. In self-test, high test sets volume results in significant hardware overhead. Today test cost per transistor is almost as high as its fabrication cost [Wang 08a]. The high test cost negatively impacts the market price of a chip. For the successful operation of semiconductor industry in the future, it is inevitable to search for new test solutions that can drastically reduce test costs without compromising the quality of fabricated chips.

Since rapid increase in test cost is the result of growing test sets sizes, test data compression is widely used to reduce test sets volume. Test sets are inherently highly compressible because of the presence of large number of don't care bits. In test data compression test sets are stored in compressed form which are later decompressed during test with the help of additional hardware on-chip. Test compression schemes are broadly classified into code based and linear decompressor reseeding based schemes. Code based schemes exploit the regularity in test sets where regularity in a test set is defined as the presence of repeating sequences of bits. The efficiency of reseeding depends on number of care bits in a test set. If encoding efficiency (EE) is defined as the number of encoded care bits divided by the required amount of storage bits, the maximum achievable encoding efficiency with reseeding is 1 [Wang 08a]. The code based schemes do not have this limitation.

In ATE testing the test data compression enables the use of low cost testers with smaller memories and reduces the test time as well because lower volume of data has to be transferred across the limited bandwidth between the ATE and the chip. However, the fundamental gap between the speeds of ATE and CUT puts limits on test time reduction. In addition, the decompression hardware added on chip cannot be reused later in the field.

The test data compression reduces the cost of self-test by scaling down the area overhead needed to store deterministic test sets. Nevertheless the decompression hardware of employed scheme must be able to decompress test sets fast enough in order to retain the speed advantage of self-test. Compressed test sets are stored either in the memory of a circuit or as a hardwired logic function. The advantage of memory based self-test over hardwired logic self-test is its programmability. If CUT structure and as a result test sets are changed due to late design changes, the new test sets could be compressed and stored in the same memory structure. The disadvantage is that it needs more area compared to hardwired logic self-test [Wund 96].

Available test compression methods significantly reduce test cost but the rapid increase in test sets volume demands the development of novel test data compression method that can drastically reduce storage requirements while overcoming the limitations in available methods. Compression of a test set depends on its statistical attributes like care bits percentage and distribution of zeros and ones. Available compression methods assume that all test sets have similar attributes which is not always true. Depending on the circuit structure and the required trade-offs, the statistical attributes vary considerably among different test sets. Due to varying attributes, no single test compression solution is possible that can guarantee optimized efficiency for all test sets. In order to have optimized compression, test sets with different statistical attributes have to be addressed differently.

The goals of our research are to analyze test sets of industrial circuits and categorize them into different classes based on their statistical attributes. Then by addressing each class differently, invent highly optimized test compression methods and decompression hardware. Self-test schemes are attractive for ensuring high quality and reliability of modern day ICs. Therefore our research shall take special care that the developed compression methods can be employed in self-test without compromising any of its benefits. Considering very large sizes of future circuits, the hardwired logic self-test does not seem practical because a re-synthesis of self-test circuitry would be required after each late design change resulting in significant resources and time overhead. Accordingly, the developed schemes would facilitate a fully programmable self-test. Using each of the proposed method, efficient programmable self-test schemes offering different trade-offs in testing would be implemented in order to demonstrate their application and effectiveness.

1.2 Major contributions

After the analysis of different test sets, it has been discovered that test sets with different statistical attributes could generally be categorized into three classes namely strongly regular, weakly regular and irregular test sets. Consequently three novel compression methods and decompression architectures have been proposed that offer optimized compression and decompression for each class. The proposed test compression methods are equally adaptable in ATE testing and self-test and three low cost programmable deterministic self-test schemes offering different trade-offs have been developed by applying these methods.

Test sets with low care bits percentages show strong regularities. These regularities are efficiently exploited with the proposed method *restrict encoding* (RE) [Hakm 09] by restricting successive repeating sequences to a single precomputed value. The portions of test set showing irregularity are encoded with the seeds. For the first time in literature the restrict encoding always offers an encoding efficiency higher than 1. The self-test scheme developed by employing restrict encoding achieves high test quality with very less storage cost.

Test sets with high percentages of care bits do not contain strong regularity. However if *0* and *1* are in unbalanced proportions, a weak regularity can be found in such test sets. The proposed *run-length encoding with parallel decompression* (REWPD) [Hakm 05] uses this regularity to offer compression ratio close to theoretical limit. Unlike conventional run-length decompressors, the proposed decompressor regenerates a complete run-length symbol in a single clock that significantly reduces the test time. A self-test scheme offering small on-chip area overhead has been developed using REWPD. Comparing to the state of the art similar method [El M 08], employing REWPD offers significant reductions in test time and storage requirements.

The third scheme *nearly complete reseeding* (NCR) [Hakm 07] offers higher encoding efficiency for irregular test sets by discovering regularity in encoding capabilities of a linear decompresser. The NCR is based on the observation that ignoring few inconsistent bits during seed computation enables to encode significantly large number of care bits in the seed of a linear decompressor in a regular way. The ignored bits are stored separately and embedded into generated patterns during test. Unlike other reseeding methods the size of linear decompressor is independent of maximum number of care bits in a test pattern and an encoding is always guaranteed that eliminates the possibility of any degradation in test quality. The self-test scheme developed by utilizing NCR offers an efficient test with short test time. The NCR reduces the self-test hardware overhead up to half compared to the most efficient similar method.

1.3 Outline

Chapter 2 gives an overview of basic concepts that are necessary to better understand this work and describes the state of the art. In chapter 3 statistical analysis of different test sets generated for industrial circuits are presented. Chapter 4 describes the proposed compression methods. Chapter 5 is devoted to demonstrating the employment of RE in self-test in order to accomplish higher test quality with lesser test cost. Chapter 6 explains the development of a self-test scheme to efficiently achieve small on-chip area overhead using REWPD. Chapter 7 describes efficient self-test to attain short test time by employing NCR. Chapter 8 summarizes the whole work.

CHAPTER 2

Preliminaries and State of the Art

Testing is the process of applying a set of binary patterns to the inputs of the *circuit under test* (CUT) and analyzing the output responses. If the output responses match the expected responses then the circuit is considered good otherwise it is assumed to be defective. Each applied pattern is called a ***test pattern*** .

Defect in a circuit is the physical imperfection that causes a circuit to fail to perform in a required manner. A ***fault*** is the representation of a defect at the abstract function level. In order to assume that a combinational logic circuit with n inputs does not contain functional faults, all 2^n possible patterns have to be applied to its inputs. This approach is called ***exhaustive testing***. Since n is very large for modern circuits, exhaustive testing is not feasible. The ***structural testing*** is a practical approach where specific test patterns are generated based on the circuit's structural information and a set of fault models. A ***fault model*** reflects the behavior of defects in a circuit. The fault models are used for pattern generation because the diversity of real defects has made it difficult to generate patterns for defects. A combination of different fault models is often used for testing a circuit since no single fault model accurately reflects the behavior of all possible defects [Wang 06].

The ***stuck-at fault model*** is the most widely used fault model because it detects the majority of defects in a circuit. Here a circuit is assumed to be modeled as an interconnection of boolean gates. A stuck-at fault is assumed to affect only the interconnections between gates. A stuck-at fault transforms the correct value on the faulty connecting line to appear to be stuck at a constant logic value, either a logic *0* or a logic *1*, referred to as *stuck-at-0* or *stuck-at-1*, respectively [Abra 94, Bush 00, Wang 06]. In the example circuit of figure 2.1, a stuck-at-1 fault at the output of *NAND1* forces the output to remain *1* irrespective of the values at the gate inputs. Generally detecting each stuck-at fault only once provides sufficiently high defect coverage. However for critical applications the defect coverage is enhanced by detecting each stuck-at fault multiple times with different patterns [Ma 95, Redd 97, Chan 98, Grim 99, Pome 99]. The developed compression schemes could be used for storing test sets of any fault model cost effectively but for the sake of simplicity we shall assume only the stuck-at fault model in the rest of our discussion. A test pattern generated to detect a specific fault of a given fault model using the circuit's structural information is called a ***deterministic pattern***. Every bit of a deterministic pattern may take one of three values {*0,1,X*} where *0,1* are called *care bits* and *X* is a *don't care bit*. If no pattern can be found to test a fault, that fault is called ***undetectable fault***. The process of pattern generation for modern circuits is automated using *Automatic test pattern generation* (ATPG) [Roth 66, Goel 81, Fuji 83, Glas 95] tools.

To understand the process of pattern generation suppose we have the small circuit shown in figure 2.1. In order to generate the test pattern that detects the stuck-at-1 at *NAND1* output, the inputs of *NAND1* are set such that its output becomes *0*. This is only possible if both of its inputs are *1*. A fault in a circuit can only be detected if its expected response differs from the faulty response. For this the second input of *NAND3* must be *1*, which is justified by setting the inputs of *NAND2* as *0X*. Hence by applying the test pattern *110X* at the inputs of this example circuit, a *0* at the circuit output tells that the output of *NAND1* is stuck-at-1 and a 1 shows vice versa.

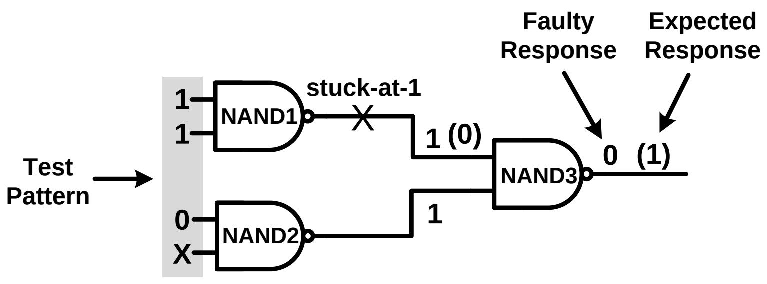


Figure 2.1: An example of stuck-at fault and the pattern generation

A set of deterministic patterns targeting the faults of a specific fault model is termed as ***deterministic test set***. In our discussion, unless specified otherwise, a test pattern and a test set would refer to a deterministic test pattern and deterministic test set respectively. The ***care bits density*** of a test set is the percentage of care bits among total bits of the test set. A test set with high care bits density is called a ***dense test set*** while the one with low density is a ***sparse test set***. Two patterns in a test set are called ***compatible patterns*** if there is no conflicting care bit at any position between them. For example, the test patterns p_1 and p_2 shown at left hand side in figure 2.2 are compatible with each other. The compatible patterns can be merged into a single pattern such that the resulting pattern carries all the care bits of both patterns. ***Test set compaction*** is the process of reducing number of patterns in a test set by merging pairwise compatible patterns into one pattern. It generally results in smaller but denser test sets. Since test set compaction has exponential algorithmic complexity, heuristic solutions [Kaji 94, Hamz 00, Lin 01, Galk 06] are often used to compact large test sets.

Figure 2.2 presents a small example of test set compaction. In the uncompacted test set, patterns p_1, p_2 and p_4 are pairwise compatible while p_3 is compatible with p_5. So p_1, p_2 and p_4 are merged to form the pattern p_1 of the compacted test set and the pattern p_2 of this test set is formed by merging p_3 and p_5. The test set size has been reduced from 5 to 2 due to compaction. This reduced test set still detects all the faults for which the uncompacted test set was generated. However the care bits density has significantly increased because of compaction.

Uncompacted test set

p_1	1	0	X	X	X	0	X	1
p_2	1	X	X	1	0	X	X	X
p_3	X	1	X	0	X	X	0	X
p_4	X	X	1	X	0	X	1	X
p_5	0	X	0	X	1	X	X	X

$\Rightarrow$

Compacted test set

p_1	1	0	1	1	0	0	1	1
p_2	0	1	0	0	1	X	0	X

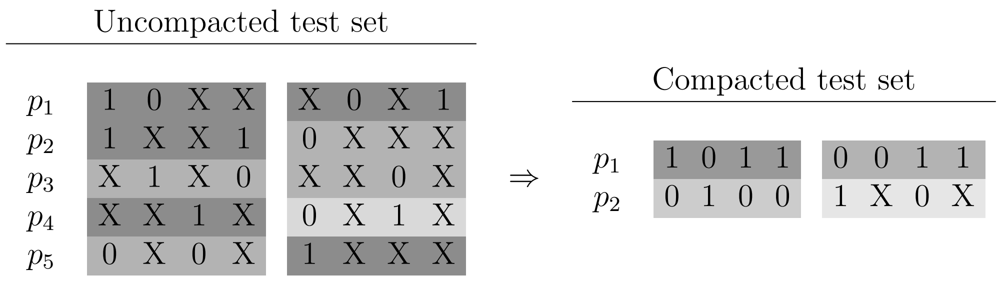
Figure 2.2: Examples of uncompacted and compacted test sets

A ***symbol*** is a continuous sequence of bits in a test set. Two symbols are compatible if they are of equal length and have no conflicting care bits at any position. ***Regularity*** in a test set is the repetition of pairwise compatible symbols. A test set may show regularity because of two main reasons:

- The care bits are sparse
- The care bits of the test set are biased toward *0* or *1*

In sparse test sets the majority of symbols are compatible with each other that results in ***strong regularity***. The compatibility between symbols is very low in dense test sets. However a weak regularity exists in dense test sets if their care bits are biased towards *0* or *1*. We shall refer such a regularity as ***weak regularity*** or ***biased regularity***.

The test sets without any regularity are named ***irregular*** test sets. Generally dense test sets without care bits biasness are irregular. Figure 2.3 shows the regularity in the example test sets of figure 2.2 after dividing the test sets into 4 bit symbols. The pairwise compatible symbols are represented with the similar background. The figure shows that compatible symbols appear repeatedly in the uncompacted test set while no symbol is repeated in the compacted test set.



Uncompacted test set

p_1	1 0 X X	X 0 X 1
p_2	1 X X 1	0 X X X
p_3	X 1 X 0	X X 0 X
p_4	X X 1 X	0 X 1 X
p_5	0 X 0 X	1 X X X

$\Rightarrow$

Compacted test set

p_1	1 0 1 1	0 0 1 1
p_2	0 1 0 0	1 X 0 X

Figure 2.3: Regularity in the uncompacted and compacted test sets

The fault detection capabilities of a given test set for a given fault list are measured in terms of *fault coverage* (FC) and *fault efficiency* (FE) which are defined as

$$\mathbf{FC} = \frac{\text{Number of detected faults}}{\text{Total number of faults}} \tag{2.1}$$

$$\mathbf{FE} = \frac{\text{Number of detected faults}}{\text{Total number of faults} - \text{Number of undetectable faults}} \tag{2.2}$$

Design for testability (DFT) enables structural test for complex circuits. ATPG and test of sequential circuits is hard due to the existence of numerous internal states that are difficult to set and check from external pins. ***Scan design*** [Eich 77], the most widely used DFT methodology, transforms the problem of testing a sequential circuit into the problem of testing a combinational logic. Every storage element is converted into a ***scan cell*** by adding one additional scan input port and one additional scan output port to provide them with external access. Scan cells are also termed as ***pseudo primary inputs*** and ***pseudo primary outputs***. Scan cells are connected together to form multiple shift registers, called ***scan chains***.

Conventionally a circuit is tested using an *automatic test equipment* (ATE). The ***test data*** containing deterministic test sets and expected responses is stored in the memory of tester. During test, the patterns are applied to the circuit and the responses are collected, which are then compared to the expected responses to detect potential faults. While the speed of circuits has increased in 100 folds over time due to reduction in feature sizes, the speed of ATE has increased only in 10 folds. Beside this testers have limited number of I/O pins to transfer data between ATE and CUT. The gap between the speeds of ATE and CUT and the limited number of tester pins results in bandwidth bottleneck. Specialized expensive ATE with large storage capacity and large number of I/O pins are required to test new generation circuits. In addition, in-field test and diagnosis are not possible using ATE.

The ***self-test*** schemes are used to reduce test costs and to improve test quality by eliminating the need of ATE and its related limitations. Here pattern generation, response evaluation and test control are implemented in hardware. Depending on required trade-offs in on-chip hardware overhead and test time, pattern generation, response evaluation and test control can be constructed either in a special chip or as part of the circuit itself. The former is termed as *built-out self-test* (BOST) and the latter as *built-in self-test* (BIST). Higher defect coverage, inherent at-speed testing for timing related faults and in-field test and diagnosis [Wang 08a] are additional advantages of self-test beside ATE and bandwidth bottleneck riddance.

Test compression is used to reduce the storage cost of applied patterns and expected responses. Test compression is of two types: ***the lossless compression*** and ***the lossy compression***. In lossless compression no test information is lost and the original test data can be regenerated from the compressed one. In lossy compression, commonly called ***compaction***, the original data can not be retrieved because the long sequences of the test data are converted into the short signatures. Lossless compression is used for test patterns in order to preserve the fault efficiency. A decoded pattern shifted into the scan chains exactly matches the original pattern in all the care bits. The output responses are compacted because it suffices for test response analysis that the signature of a faulty circuit differs from the signature of a good one. Since compaction extremely reduces the volume of expected responses its storage requirements are insignificant compared to compressed test sets. In the rest of our discussion the term test compression will solely be used for lossless compression of a test set.

Test compression presents an example of *test resource partitioning* (TRP) in ATE testing where test functionality is divided between circuit and the tester. Since test compression reduces the storage and the bandwidth requirements, a low cost tester with less memory and fewer I/O pins could be used. Figure 2.4 illustrates the idea of test compression in ATE testing for a circuit with scan design. Here test data is stored in ATE memory in compressed form. Additional hardware is added on-chip before the scan chains to decompress the patterns coming from the ATE and after the scan chains to compact the responses going to the ATE.

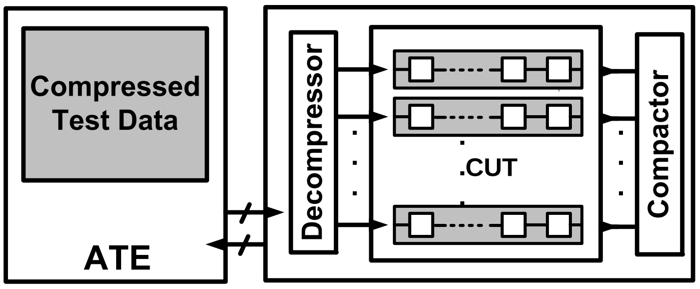


Figure 2.4: An example of test resource partitioning for a scan enabled circuit

A number of pattern generation approaches are adopted in self-test to minimize the hardware overhead needed to generate the input test data. In ***exhaustive self-test*** all 2^n possible patterns are applied to a circuit where n is the total number of primary and pseudo primary inputs. Any *n-bit* binary counter or a shift register that can cycle through all states could be used as exhaustive pattern generator [McCl 81, McCl 86, Wang 86]. This technique guarantees complete fault efficiency but the extremely long test time due to the large n has made this technique impractical for the modern day circuits.

A subset of 2^n test patterns is generated in ***pseudo random testing*** [Bard 82] in order to reduce the test length. The *self-test using MISR and parallel shift register sequence generator* (STUMPS) scheme [Bard 82] shown in figure 2.5 is the canonical approach of self-test where a *pseudo random pattern generator* (PRPG) is used as test pattern generator, and a *multiple input signature register* (MISR) is used as a response compactor. The storage elements as well as the primary inputs and the primary outputs of the circuit are converted into multiple scan chains. A PRPG produces, at each clock, a ***test vector*** of k bits, which are shifted into k scan chains in parallel. With t being the maximum scan chain length, t test vectors are loaded into the scan chains to form a complete test pattern. Once a complete pattern is shifted in, system clock is applied to capture the responses of the circuit. These responses are then transferred to MISR for compaction. The shifting in of the next pattern and the shifting out of the responses are done at the same time.

In the rest of our discussion, we shall assume that the circuit under test (CUT) complies with the STUMPS architecture. Pattern generation with PRPG offers minimum hardware overhead and an easy implementation, but fault coverage is often sacrificed due to the presence of the faults that are difficult to detect with random patterns. Such faults are called *random pattern resistant* (RP-resistant) faults. The faults that are detectable with random patterns are termed *random pattern testable* (RP-testable) faults.

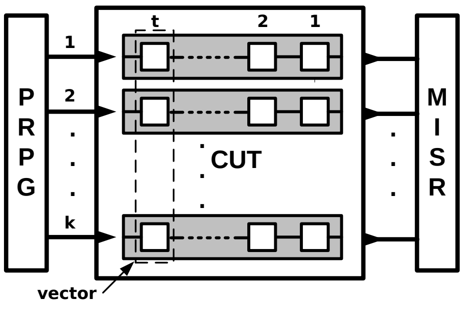


Figure 2.5: STUMPS scheme

Additionally achieving a certain level of fault efficiency is not possible in a reasonable amount of time with pseudo random testing [Bard 87, Rajs 98, Bush 00, Jha 03].

Better fault efficiency and shorter test time are achieved with ***weighted random pattern generation*** [Schn 75, Chin 84, Wund 87, Wund 90, Stro 91]. These patterns are generated by inserting a combinational circuit between the PRPG and the CUT to bias the random patterns with the help of stored weights such that the detection probability of RP-resistant faults increases. In many cases a large number of weights are required for complex circuits [Waic 89, Bers 93, Kapu 94, Lai 05] which limits its feasibility.

Pseudo exhaustive testing is another approach to shorten test length while retaining many advantages of exhaustive testing [McCl 81, Hell 90, Bush 00, Wang 08a]. When each output of the n-input CUT at most depends on w inputs, only 2^w patterns need to be generated and applied. The disadvantage of this approach is that if the value of w exceeds certain limits, hardware segmentation is required to ensure practical test time.

Deterministic self test schemes [Daeh 81, Dand 84] store precomputed deterministic patterns to guarantee desired fault efficiency. But very often as the number of required patterns increases with the size of the circuit under test, the required chip area to store these patterns becomes unaffordable.

The advantages of low cost of pseudo random testing and guaranteed coverage of deterministic testing are combined in ***mixed-mode self-test***. These schemes offer a trade-off between hardware overhead and test time while achieving desired fault efficiency. In mixed-mode self-test RP-testable faults are detected by applying a limited number of pseudo random patterns while the desired fault efficiency is achieved by targeting RP-resistant faults with deterministic patterns. The pseudo random patterns are generated cost effectively by running a counter or shift register in autonomous mode. To reduce the storage requirements of deterministic patterns different test compression schemes [Koen 91, Wund 96, Toub 96, Gher 04] are used.

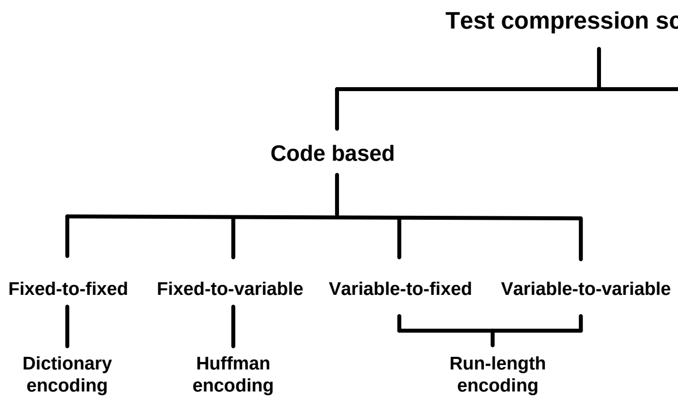


Figure 2.6: Overview of compression schemes

A large number of test compression schemes have been proposed because of their effectiveness in ATE testing and self-test. The chart in figure 2.6 gives a brief overview of test compression. Available compression schemes are broadly classified into two main categories namely ***code based*** schemes and ***linear decompressor*** based schemes. Code based schemes exploit the regularity in test sets. Here the test data is partitioned into symbols and every symbol is replaced with a codeword to form the compressed data. A decoder converts each codeword into the corresponding symbol to regenerate the original data. Two quantitative measures that are often used to merit the performance of a code based scheme are *compression percentage* (CP) and *compression ratio* (CR) which are defined as

$$\mathbf{CP} = \frac{\text{Bits in uncompressed test set} - \text{Storage bits}}{\text{Bits in uncompressed test set}} \times 100 \tag{2.3}$$

$$\mathbf{CR} = \frac{\text{Bits in uncompressed test set}}{\text{Storage bits}} \tag{2.4}$$

The CP of a compression scheme depends on the partition of test data into symbols and the frequency of occurrence of each symbol. The compression potential of a code based scheme is estimated by determining its ***entropy*** H that is computed using the formula

$$\mathbf{H} = -\sum_{i=1}^{n} \alpha_i \cdot \log_2 \alpha_i \tag{2.5}$$

where α_i is the probability of occurrence of symbol s_i in the test set and n is the total number of unique symbols. Since H gives the minimum average number of bits required for each code, the maximum compression percentage CP_{max} that can be achieved for these symbols is calculated as

$$\mathbf{CP}_{max} = \frac{\text{avg}(|s|) - H}{\text{avg}(|s|)} \times 100 \tag{2.6}$$

$$\text{where} \quad \text{avg}(|s|) = \sum_{i=1}^{n} \alpha_i \cdot |s_i| \quad \text{with } |s_i| = \text{length of symbol } s_i$$

A linear decompressor consists of XOR gates and flip-flops. In linear decompressor based schemes a deterministic pattern is encoded as the initial state, called seed, of the linear decompressor. The seed for a given pattern is computed by solving a system of linear equations where each equation corresponds to a care bit in the given pattern and each variable represents a seed bit. To decode the pattern, the linear decompressor is initialized with the seed and the output space of the decompressor gives the encoded pattern. The compression of a linear decompressor based scheme is independent of the regularity in a test set and depends on number of care bits in it. A measure used to indicate the performance of linear decompressor based schemes is *encoding efficiency* (EE) which is defined as

$$\mathbf{EE} = \frac{\text{Care bits in uncompressed test set}}{\text{Storage bits}} \tag{2.7}$$

The decompression time overhead and programmability are also important in judging overall performance of a compression scheme. Prolonged decompression time may limit the feasibility of a compression method while a resynthesis of the decompression hardware might be required after every change in test sets if the scheme is not programmable.

Code base schemes are further classified into four categories depending on whether the symbols and the codewords are of fixed or variable length. In fixed-to-fixed schemes, the test set is divided into fixed length symbols and each symbol is assigned a fixed length code. Here a small set of fully specified symbols is stored in the memory on-chip. These fully specified symbols are generated by merging the pairwise compatible unique symbols. The address of each symbol in the memory then becomes the codeword of all the unique symbols compatible to it. Fixed-to-fixed schemes are called ***dictionary encoding*** since each symbol in the memory can be viewed as an entry in a dictionary and the codeword as an index into the dictionary. If all the unique symbols occurring in the test set are compatible with one of the dictionary entry then it is called a ***complete dictionary*** otherwise it is a ***partial dictionary***. If a complete dictionary contains U entries of the length L_s, the code a is $a = \lceil log_2 U \rceil$ bits. After receiving a bits code, the decompressor regenerates L_s bits symbol in a single clock. The compression ratio using complete dictionary is $2^{L_s - a}$. A high compression ratio is achieved if a is much smaller than L_s.

A dictionary encoding scheme for the multiple scan chain architecture is proposed in [Redd 02] which is illustrated in figure 2.7. Each test vector is considered as a symbol and a complete dictionary is formed that is able to generate all the unique test vectors in the test set. This scheme does not causes any decompression time overhead. However if many conflicting test vectors exist in the test set, the size of dictionary could become prohibitively large.

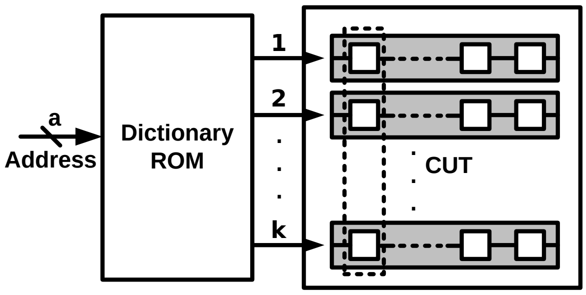


Figure 2.7: Overview of dictionary encoding

In [Li 03] this problem is solved by using a partial dictionary whose size is selected according to the available on-chip area for decompressor. A predefined number of most frequently occurring symbols are stored in the dictionary. A symbol is encoded as a fixed-length index of the dictionary if it could be generated by any of the dictionary entry otherwise it is left unencoded. A flag bit is used to distinguish between the encoded and the unencoded symbols during test. This scheme limits the hardware overhead caused by the dictionary but the compression of this scheme is affected by the fact that a large number of symbols are left unencoded because they do not match any dictionary entry.

The dictionary encoding proposed in [Wurt 04] allows a fixed number of corrections in the dictionary entries to limit the size of the dictionary. A symbol is encoded with the status bit, the dictionary index and the positions of required corrections. A correction logic is used in between the decoder and the scan chains to modify the dictionary entry before shifting it into the scan chains. This scheme reduces the dictionary size but the savings due to the reduced dictionary size are diminished by the correction information storage. Better compression is achieved with hybrid dictionary encoding in [Kim 08] but the decompression hardware of this scheme is not flexible and depends on the test set.

In fixed-to-variable encoding schemes, the fixed length symbols in the test set are assigned variable length codes. The basic idea behind variable length codes is to minimize the average length of codewords by assigning shorter codewords to those symbols that occur most frequently in the test set. The ***Huffman encoding*** [Huff 52] is an example of the fixed-to-variable encoding that is proven to provide the shortest average codeword length among all other schemes of this category. The codes are assigned by constructing a Huffman tree according to the frequency of occurance of each symbol as described in [Huff 52]. The Huffman encoding provides the optimum compression but the decoder size increases exponentially with the increasing size of the symbols. In [Jas 03, Kavo 07] the hardware overhead of decoder is limited by using selective Huffman codes. Only a limited number of most frequently occurring symbols are encoded with Huffman codes while others are left unencoded. An extra bit is used with each code to differentiate between the encoded and the unencoded symbols. Though the decoder size grows linearly in these scheme, the compression ratio is sacrificed. Moreover, a hardwired decoder is the common drawback in all the test compression scheme employing Huffman codes.

The ***run-length encoding*** is another famous example of code based schemes. In run-length encoding the test set is divided into variable length symbols such that consecutive 0's and 1's form a single symbol. The symbols are then assigned a fixed [Jas 98] or variable length [Chan 01, Gonc 02, Chan 03, El M 08] code. The run-length encoding performs better for the biased regular test sets. In [Jas 98], the test information is made biased by ordering the test patterns such that similar test patterns come after each other. A difference pattern is then computed between two adjacent test patterns which is more likely to be biased towards 0. This difference pattern is encoded instead of the test patterns. A cyclic scan architecture is used during the test to regenerate the test patterns from the difference pattern. In cyclic scan architecture, an XOR operation is performed between the test data currently being shifted in and the previous test pattern.

In [Chan 01] it has been shown that the compression ratio significantly improves if the run-length symbols are encoded with the variable length Golomb code [Golo 66]. The efficiency of run-length encoding was further improved in [Chan 03, Ruan 07, El M 08] by proposing *frequency directed* (FDR), *matching pattern* and *extended frequency directed* (EFDR) run-length codes respectively. A more optimized run-length encoding scheme is *variable input Huffman code* (VIHC) presented in [Gonc 02]. Since Huffman codes [Huff 52] are used, the decompression architecture of this scheme is test set dependent. The decompression architecture of a run-length scheme mainly consists of a *finite state machine* (FSM) and a counter. The FSM detects the codewords while the counter is used to generate a continuous stream of 0s or 1s according to the length of encoded symbol. The run-length encoding is useful for the test patterns showing biased regularity however since the run-length symbols are regenerated serially with counters, it results in significant decompression time overhead.

Other examples of code based compression schemes include packet-based compression [Volk 02], nine-coded compression [Tehr 05], multilevel Huffman coding [Kavo 08] and selective scan slice encoding [Wang 08b]. Code based schemes are advantageous in exploiting the regularities in test sets. However if the test set is irregular then the compression ratio is poor. In addition some of these schemes may also result in significant decompressor area and decompression time penalties.

Linear decomprssor based schemes are efficient in encoding irregular test sets. The *linear feedback shift register* (LFSR) reseeding is an example of it. Figure 2.8(a) shows the process of LFSR seed computation. Here a 12 bit pattern with 4 care bits is encoded with a 4 bit LFSR. To compute the seed, each of the seed bit is assumed as a free variable and the linear equations of these variables corresponding to each care bit in the pattern are generated. The resulting system of equations is then solved and the solution gives the seed. Figure 2.8(b) shows this seed along with the decoded pattern. If no solution exists for the employed system of linear equations, that pattern can not be encoded using the given LFSR. If c_{max} represents the maximum number of care bits in a pattern of a test set, it has been shown in [Koen 91] that by keeping the LFSR size equal to $c_{max} + 20$, the probability of not finding a seed is less than 10^{-6}. The scheme in [Hell 95] maintains this small probability of failure with a $c_{max} + 1$ bit LFSR using multiple polynomials. Further reductions in storage were achieved using variable-length seeds [Rajs 98], partial-reseeding [Kris 01], continuous reseeding [Volk 03], seed encoding [Al Y 05] and state skip LFSRs [Tene 08].

Other linear decompressor based schemes include scan chain concealment [Bayr 01], adjustable linear decompressor [Kris 03], ring generators [Mrug 04] and align encode [Sina 08]. Linear decompressor based schemes offer a small and simple decoder. In addition, the linear decompressors can generate a complete test vector at each clock (figure 2.9) for random or deterministic testing. Due to this reason a linear decompressor is an integral part of every self-test scheme. However linear decompressor based schemes

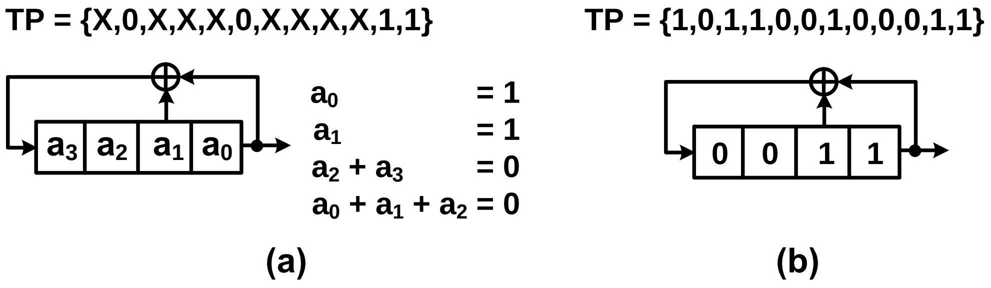


Figure 2.8: Example of LFSR seed computation

can not exploit the regularities in a test set and their compression is limited by the number of care bits in it. In general, the maximum encoding efficiency that could be achieved using linear decompressor based schemes is 1 [Wang 06]. Moreover, though by keeping linear decompressor large enough the probability of failure could be kept quite small, but still the encoding of a complete test set is not guaranteed. This results either in the degradation of test quality or the repetition of ATPG process. The repetition of ATPG process is not possible in core based designs where structural information of the circuits is not known.

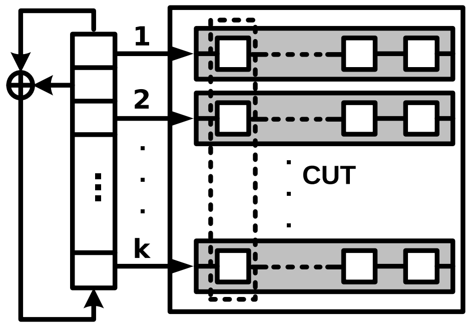


Figure 2.9: Pattern generation with LFSR

Test compression schemes significantly reduce the storage requirements for the ATE and the self-test. Mixed-mode self-test schemes are more attractive since only a subset of the deterministic patterns need to be stored to achieve the complete fault efficiency. The storage requirements are further reduced by storing the deterministic patterns in compressed form. In available mixed-mode schemes one of the two approaches known as reseeding [Koen 91] and test set embedding [Wund 96, Toub 96, Gher 04] are used to compress the deterministic patterns.

In reseeding, the deterministic patterns are encoded as the seeds of the same linear decompressor that is used to generate the random patterns. The seeds are stored in a memory on-chip. After pseudo random testing finishes; the linear decompressor is reseeded to generate the desired patterns. Main advantage of this approach is its programmability. If the test set is changed due to last minute design changes, new seeds could be computed and stored in the same memory. The disadvantages of this approach are that the storage requirements remain significant and the encoding of a pattern is not guaranteed which may affect fault coverage.

In test set embedding, the deterministic patterns are embedded into the pseudo random sequence by changing some bits in useless pseudo random patterns. The embedding information is stored on-chip as a combinational logic called bit flipping [Wund 96, Gher 04] or bit fixing [Toub 96] logic and its area overhead is reduced using logic minimization techniques. The advantage of this approach is less on-chip area overhead compared to the seeds memory [Wund 96, Gher 04] while the disadvantage is that the test information is stored hardwired which makes it test set dependent, introducing serious challenges in the design flow.

CHAPTER 3

Test Set Analysis

A test set is a set of test patterns that are applied in a sequence to achieve a desired fault efficiency. The fault efficiency for STUMPS complying circuits is independent of the order in which the patterns are applied. In STUMPS (figure 2.5), each test pattern is a set of test vectors. A complete test vector is shifted into scan chains at each clock. To initialize the storage elements correctly, the test vectors must be shifted into the scan chains in a specific order. For the rest of our discussion we define few notations:

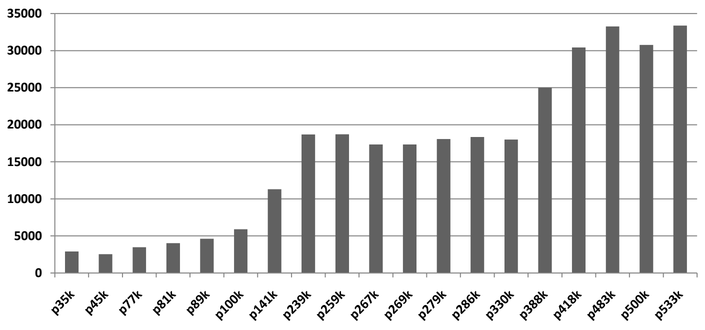
Let k represent the number of scan chains and t the maximum scan chain length. Then a test vector v is k bits long containing care and don't care bits. The bit position i of v is represented as $v(i)$. The number of care bits in vector v is noted as $|v|$. A test vector where $|v| > 0$ is called a ***care vector*** while a test vector with $|v| = 0$ is a ***don't care vector***. Two vectors v_i, v_j are *compatible* ($v_i \sim v_j$), if there is no conflicting care bit at any position. Compatible vectors can be *merged* $v = v_i + v_j$ so that v carries all the care bits of both v_i and v_j. If a vector set V contains only pairwise compatible vectors, the whole set can be merged into one vector which is noted as $v = \Sigma V$.A *pattern* $p = (v_1, v_2, \ldots, v_t)$ is a sequence of vectors. Let P be a test set with $|P| = q$, the test vector at position j of pattern p_i is noted as $p_i(j) = v_{ij}$. Figure 3.1 shows an example test set for STUMPS with $k = 4$, $t = 3$ and $|P| = 3$.

To analyze the statistical attributes of test sets we consider few industrial circuits. Figure 3.2 shows some important characteristics of these circuits. The name of each circuit represents the number of nets in it. For example $p35k$ means that the circuit contains 35 thousand nets. In figure 3.2(a) total number of inputs and pseudo primary inputs are presented while the test vector size k and test vectors per pattern t are shown

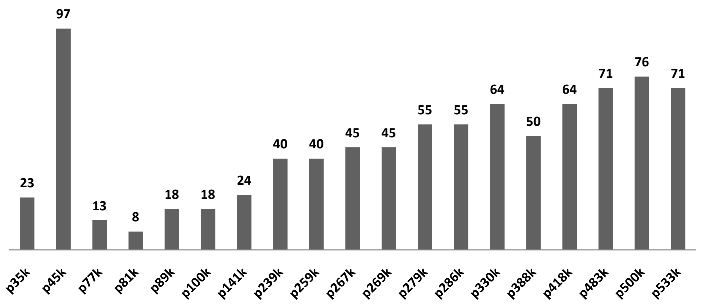
p_3			p_2			p_1		
v_{33}	v_{32}	v_{31}	v_{23}	v_{22}	v_{21}	v_{13}	v_{12}	v_{11}
1	X	0	X	X	X	0	1	X
X	1	1	1	X	1	0	1	1
1	X	1	1	X	X	X	X	0
1	0	X	0	X	1	X	1	X

Figure 3.1: An example test set for STUMPS

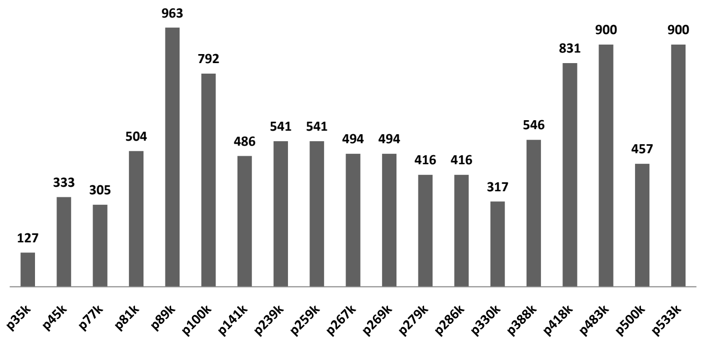
in figures 3.2(b) and 3.2(c) respectively.



(a) Number of primary+pseudo primary inputs



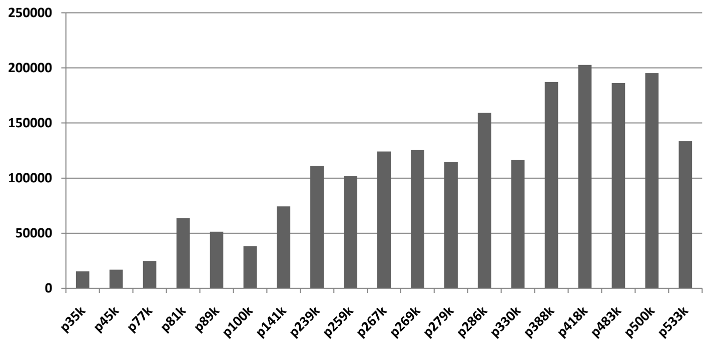
(b) Number of scan chains (k)



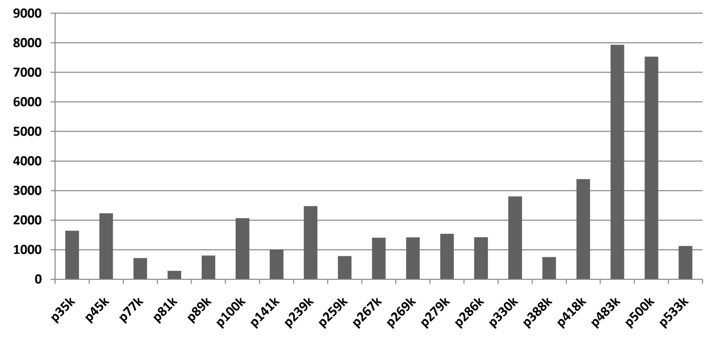
(c) Maximum scan chain length (t)

Figure 3.2: Characteristics of experimented circuits

For each circuit wo test sets achieving complete fault efficiency were generated for each circuit using a commercial ATPG tool. The test set compaction by merging compatible patterns was turned off for the first test set P_1 while the merging efforts were set to high for the second test set P_2. As a result P_1 is an uncompacted test set where each pattern detects at least one such fault not detected by any other pattern. The test set P_2 is a fully compacted test set containing the smallest number of patterns needed to achieve complete fault efficiency. The number of patterns in each test set are shown in figure 3.3.



(a) Uncompacted test set P_1



(b) Compacted test set P_2

Figure 3.3: Number of patterns in each test set

The comparison of figures 3.3(a) and 3.3(b) reveals that pattern merging highly reduces the size of a test set. However the reduction in test set size is not similar for all circuits. The compaction ratio for each circuit is shown in figure 3.4 which is calculated by dividing $|P_1|$ with $|P_2|$. The figure tells that the compaction ratio fluctuates between 8 to 248 times. The high fluctuation in compaction ratio shows that the test sets of various circuits are quite different from each other.

The most important attribute of a test set with respect to test data compression is its care bits density. The care bits density of P_1 and P_2 is presented in figure 3.5. The graphs reveal that the care bits density not only significantly differs between uncompacted and compacted test sets but it also varies largely within both type of test sets. The density of uncompacted test sets lies in the range of 0.02 to 1.47%. In general it could be said

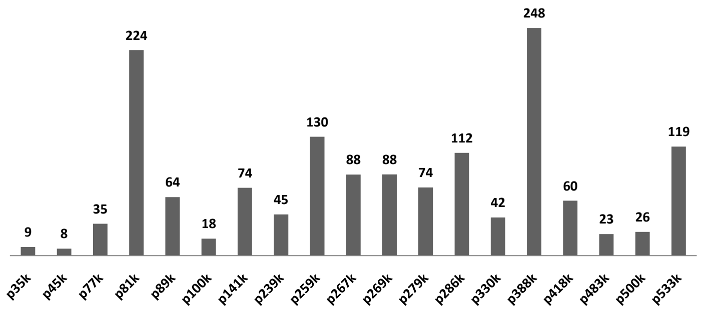
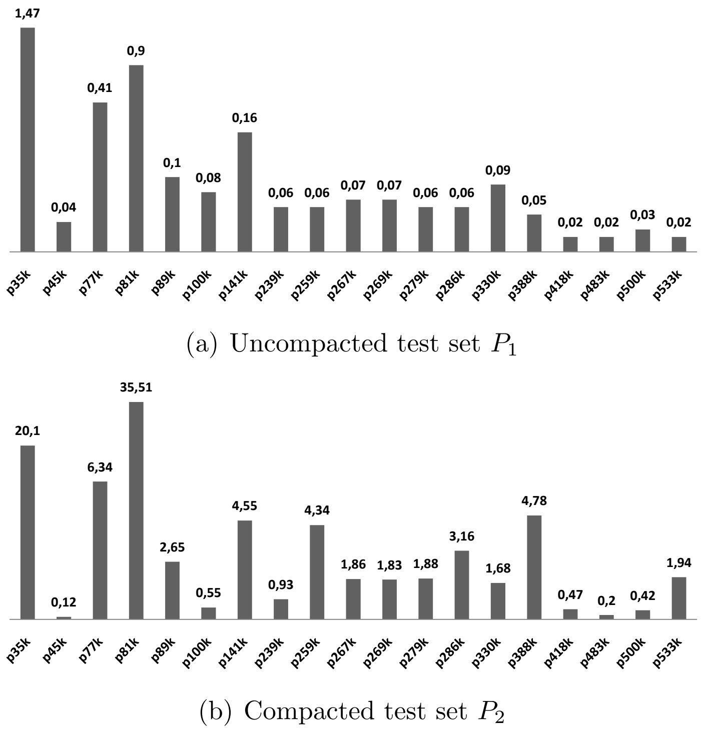


Figure 3.4: Compaction ratio ($|P_1|/|P_2|$)

for P_1 that the care bits density is lower for the larger test sets. In compacted test sets the variation in density is higher and it ranges from 0.12 to 35.5%. This is since different compaction ratios are achieved for each test set.

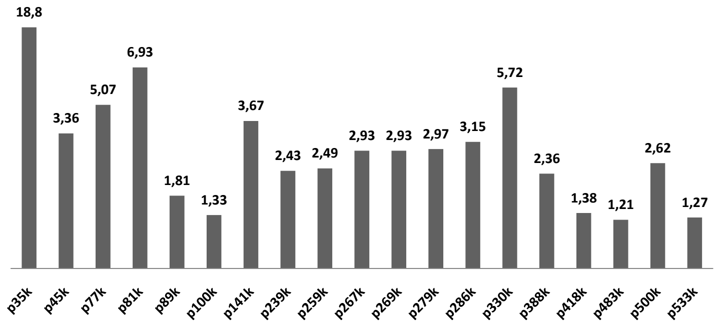


(a) Uncompacted test set P_1

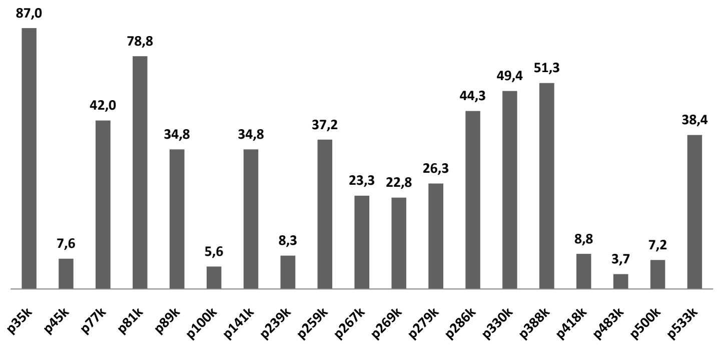
(b) Compacted test set P_2

Figure 3.5: Care bits density in each test set

In STUMPS a complete test vector of k bits is shifted into scan chains in parallel. Therefor we analyze test sets for the presence of k bit symbols. For fixed length symbols each care vector presents a complete symbol. Figure 3.6 shows the percentages of care vectors in P_1 and P_2. The figure demonstrates that in uncompacted test sets up to 18.8% test vectors are care vectors while they reach up to 87% in compacted test sets. The percentage of care vectors in a test set depends on its care bits density and test vector size k. For example k is 40 for $p239k$ and $p259k$. The care bits density of both circuits is similar in P_1, so their percentages of care vectors in P_1 are also similar. However, care bits density of $p259k$ is higher than $p239k$ in P_2 that results in higher percentage of care vectors for $p259k$. On the other hand the care bits density of $p286k$ is equal to the density of $p259k$ in P_1 and it is less than the density of $p259k$ in P_2, but for both P_1 and P_2, the care vectors percentage of $p286k$ is much higher than $259k$ due to its larger value of k.



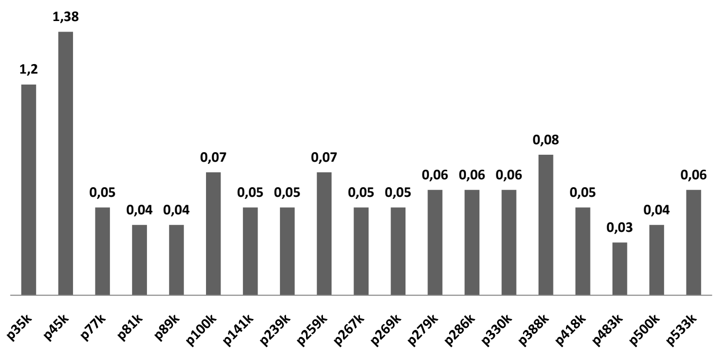
(a) Uncompacted test set P_1



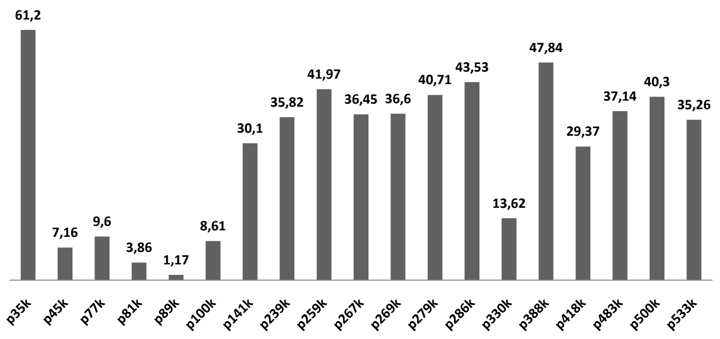
(b) Compacted test set P_2

Figure 3.6: Percentage of care vectors in each test set

The regularity in a test set depends on how many times a symbol repeats itself and also on the compatibility among the symbols. The repetition of symbols in a test set could be estimated by computing the percentage of unique symbols among all the symbols. Since each care vector represents a symbol, the percentage of unique test vectors among total care vectors is computed and shown in figure 3.7. The very low unique vectors percentage in P_1 of all circuits shows that each symbol is repeated many times in sparse test sets. This percentage is high in most of the compacted test sets with high care bits density. The 61.2% for $p35k$ indicates that a large number of symbols occur only once in the test set. It should be noted that despite highest care bits density in $p81k$, its unique vectors percentage is low. It is because of the very small test vector size in $p81k$ that is 8. The number of maximum possible symbols that may occur in a test set depends on symbols size.



(a) Uncompacted test set P_1

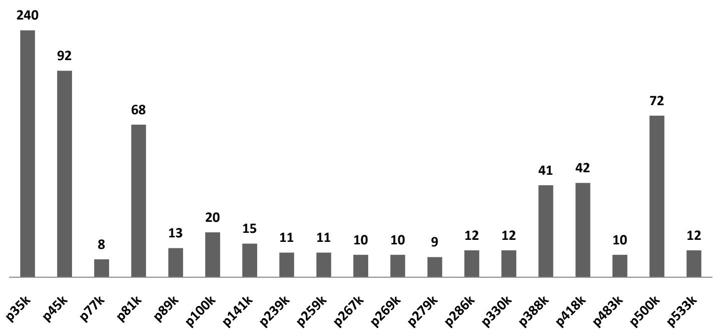


(b) Compacted test set P_2

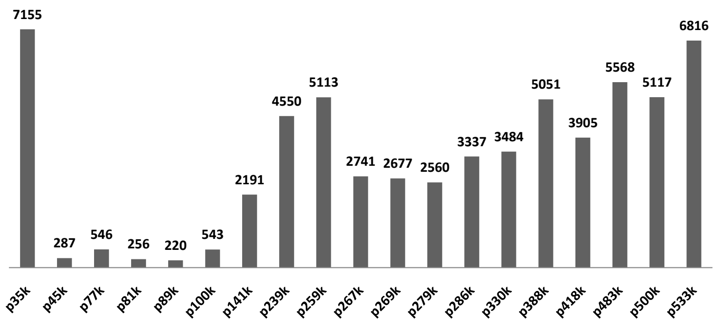
Figure 3.7: Percentage of unique vectors among care vectors

A complete dictionary for a test set is the minimum cover for all of its unique vectors. The number of dictionary vectors could be considered a quantitative measure to judge compatibility between test vectors. The number of entries in a complete dictionary for P_1 and P_2 are shown in figure 3.8. Since finding minimum set of vectors that can generate all the unique vectors has an exponential complexity, a greedy algorithm based on clique partitioning was used. The unique vectors were divided into maximum size cliques of pairwise compatible vectors and a dictionary vector was generated by merging all the vectors in a clique. It could be seen from figure that the test sets with low care bits density could be generated from very few dictionary vectors irrespective of their test vector size. Depending on care bits density and test vector size the number of dictionary entries is high for dense test sets. The dictionary of $p81k$ contains all possible 256 entries for 8 bit long symbols.

Looking at unique vectors percentage and the number of dictionary vectors we can conclude that the test sets with low care bits density contain very strong regularity. This regularity decreases with an increase in care bits density. The larger the symbol size, the higher the impact of care bits density. The test sets with high care bits density are irregular since the symbols rarely repeat and the compatibility among different symbols



(a) Uncompacted test set P_1

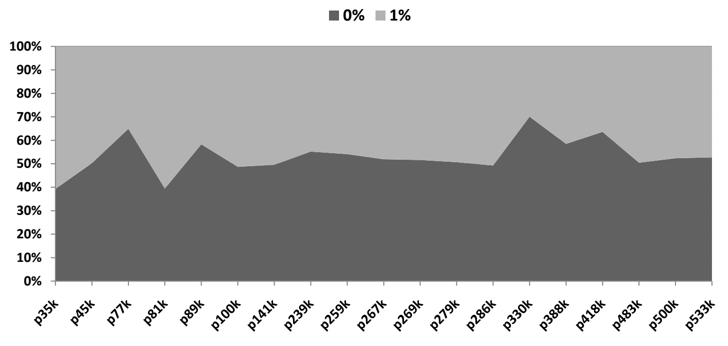


(b) Compacted test set P_2

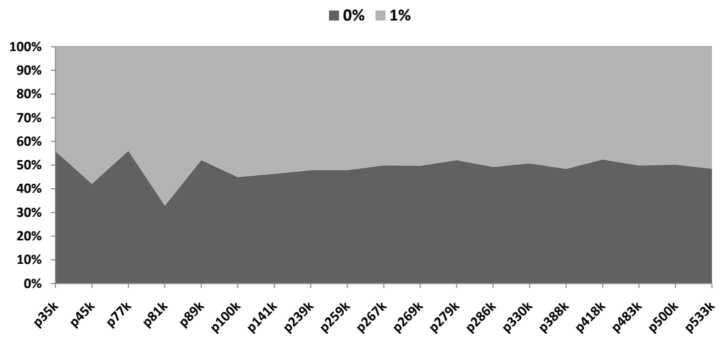
Figure 3.8: Number of dictionary entries

is low.

A closer look into dense test sets reveals that if the 0 and 1 are in unequal proportions then the symbols containing only majority bits repeat quite often. We call it a weak regularity. The distribution of 0 and 1 among the care bits of experimented test sets are shown in figure 3.9. As can be seen from the figure 3.9(b), among dense test sets $p35k$ is biased 6% toward 0 and $p81k$ is biased 17% toward 1. The percentages of majority and minority bit test vectors among total care vectors are computed for both circuits using different values of k. These percentages are shown in figure 3.10. The figure reveals that significant portion of care vectors contain majority bit vectors and the percentage of majority bit vectors is always higher than the percentage of minority bit vectors. In $p35k$ that is slightly biased toward 0, the percentage of 0 vectors decreases with the increasing test vector size. This is not the case in $p81k$ that is highly biased toward 1. In $p81k$ the varying test vector size has no significant impact on the percentage of 1 vectors.

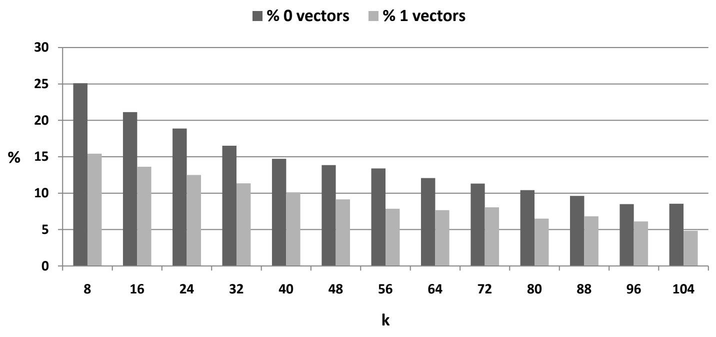


(a) Uncompacted test set P_1

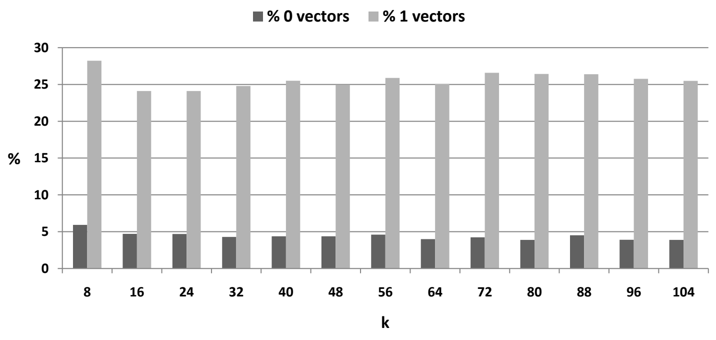


(b) Compacted test set P_2

Figure 3.9: Care bits distribution in each test set



(a) $p35k$



(b) $p81k$

Figure 3.10: Comparison of 0 and 1 vectors percentages

CHAPTER 4

Efficient Test Data Compression

In this chapter we present basics of the three novel test compression methods that offer optimized efficiency for test sets with strong, weak and no regularities. Section 4.1 explains encoding of a test set with strong regularities. The compression of a test set with weak regularities is described in section 4.2 while section 4.3 is devoted to explain the encoding of irregular test sets. These methods will be explained in detail in the subsequent chapters by implementing a self-test scheme using each method.

4.1 Restrict Encoding

The proposed method *restrict encoding* (RE) minimizes storage requirements for strongly regular test sets. It is based on the observation that in a strongly regular test set portions of the test set could be identified where long sequences of test vectors at similar vector positions are pairwise compatible. By restricting vectors in a sequence to a single value, all care bits in the sequence are generated. Such a sequence of test vectors is called a *restrict* and the value injected continuously for all the test vectors in the sequence is called a *restrict vector*. Due to high compatibility among the vectors of a strongly regular test set, all the restricts are generated by storing a very small set of restrict vectors.

The restrict encoding could be considered an instance of dictionary encoding where the restrict vectors storage represents a dictionary. However unlike conventional dictionary encoding schemes [Redd 02, Li 03, Wurt 04] a dictionary index is stored for a restrict instead of individual vectors. The encoding of a restrict contains its starting position, ending position and address of the associated restrict vector.

The cost to store a restrict is constant and does not depend on the number of test vectors or care bits in it. A vast majority of test vectors are part of long restricts therefore most of the test vectors in the test set are encoded very efficiently. This efficiency is further improved by carefully reordering the test patterns in order to increase the lengths of restricts. The portions where regularity does not appear in consecutive vectors are encoded with seeds.

Suppose figure 4.1 shows the first 4 patterns of a test set. Each pattern consists of 4 vectors. The first two restricts start with the first pattern and continuously inject restrict vectors v_a and v_b into the vector positions 1 and 2 respectively. The third restrict starts at pattern p_2 replacing vector position 4 with the restrict vector v_a. The care-bits in unrestricted vector positions $(v_{14}, v_{13}, v_{23} \dots)$ are encoded with seeds.

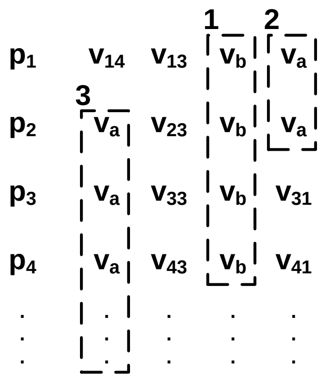


Figure 4.1: Start of a pattern application with three restricts

The total cost of restrict encoding consists of the restrict vectors, the restricts information and the seeds storage. The cost of storing the restrict vectors *dcost* is estimated as

$$dcost = |RV| \cdot k$$

where $|RV|$ is the total number of restrict vectors that need to be stored to generate all the restricts.

The cost of storing a restrict is constant. Let l be the number of bits to be stored for one restrict and R the total number of restricts, then the cost of storing restricts *rcost* would be

$$rcost = l \cdot R$$

The seeds storage is proportional to the number of encoded care bits. If e represents the encoding efficiency of reseeding and c the number of care bits encoded with the seeds then the cost of storing seeds *rscost* is estimated as

$$rscost = \frac{c}{e}$$

The total cost of encoded test set P_E then becomes

$$\begin{aligned} P_E &= dcost + rcost + rscost \\ &= |RV| \cdot k + l \cdot R + \frac{c}{e} \end{aligned}$$

The smallest achievable size of encoded test set $P_{E,min}$ is estimated by assuming that all the test vectors at each vector position form a single restrict. In this case the test set will contain a total of t restricts and no seeds need to be stored. So $P_{E,min}$ becomes

$$P_{E,min} = |RV| \cdot k + l \cdot t$$

In the worst case no restricts are found and as a result the test set is encoded with seeds. In this case encoding is an instance of conventional reseeding and the maximum size of encoded test set $P_{E,max}$ will be

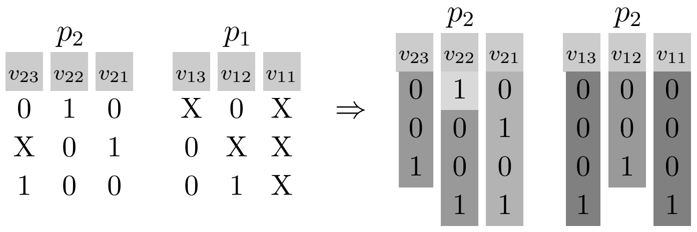
$$P_{E,max} = \frac{c}{e}$$

A built-in self-test scheme implementing restrict encoding will be presented in chapter 5.

4.2 Run-length Encoding with Parallel Decompression

A weak regularity exists in dense test sets if the care bits are biased towards 0 or 1. The proposed *run-length encoding with parallel decompression* (REWPD) is a variable-to-variable encoding scheme that offers efficient compression and decompression of weakly regular test sets. The test set is divided into run-length symbols. Each test vector is considered separately in this process. It keeps test control simple for STUMPS complying circuits and limits the decompression hardware size as well. The Xs in the test set are filled with majority bits. Every test vector is seen as a sequence of majority bit runs, each of which stops with a minority bit. If a test vector ends with a majority bit, an extra minority bit is assumed at its end. This extra bit does not add any overhead in the communication or storage. Rather it reduces the maximum number of run-length symbols from $2k$ to $k+1$ and provides a way to regenerate a complete run-length symbol within a single clock cycle. If all the bits in a test vector are majority bits then we have just one symbol of length $k+1$ and if all are minority bits then we have k symbols each of length 1.

Figure 4.2 shows an example of dividing a test set with $k = 3$, $t = 3$ and $|P| = 2$ into run-length symbols. In this example test set, 0 is majority bit and 1 is minority bit. After replacing Xs with 0s, four test vectors v_{11}, v_{13}, v_{21} and v_{22} end with 0. So a 1 is assumed at $k+1$ bit position to ensure that every run-length ends with a 1. By assuming this extra bit, maximum possible run-length symbols have been reduced from 6 to 4.

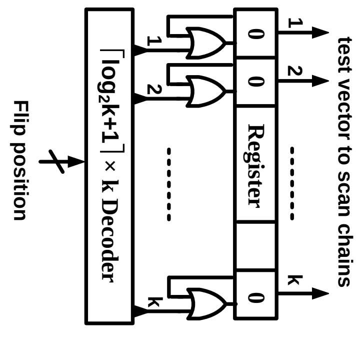


p_2
v_{23} v_{22} v_{21}
0 1 0
X 0 1
1 0 0
p_1
v_{13} v_{12} v_{11}
X 0 X
0 X X
0 1 X
$\Rightarrow$
p_2
v_{23} v_{22} v_{21}
p_2
v_{13} v_{12} v_{11}

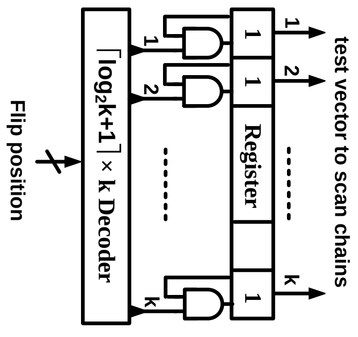
Figure 4.2: Dividing a test set into run-length symbols

Starting from a code of length 1, the symbols are assigned variable length binary codes according to their frequency of occurrences. In every test set a large number of test vectors are don't care vectors. Among the care vectors of a biased test set, a vast majority of the care vectors contain only majority bits. The don't care vectors and the vectors with only majority bits form a single symbol of length $k+1$. In practice this symbol has the highest frequency and is assigned a 1 bit code.

When it is known in advance that every run-length symbol ends with a minority bit, the only information needed to regenerate a symbol is its length. Figure 4.3 shows the basic idea of regenerating a run-length symbol in parallel. A k bit register is reset to majority bits and after knowing the symbol length l, the l^{th} bit with reference to previously generated symbol is flipped in the register. When $k + 1^{th}$, bit is needed to be flipped, no flip operation is performed. Once a test vector is completely generated, the register values are shifted into scan chains and the register is reset to majority bits. This way as soon as the length of a symbol is known, it is regenerated in a single clock. To regenerate and retain 0 run-length symbols (figure 4.3(a)) OR gates are used. The output of the binary decoder is 1 for the flip position and 0 for the rest. In case of 1 run-length symbols AND gates are used. Here the decoder outputs 0 for flip position and 1 for others.



(a) Regeneration of 0 run-length symbols



(b) Regeneration of 1 run-length symbols

Figure 4.3: Regeneration of run-length symbols

Using the proposed scheme, the size of encoded test set P_E is estimated as:

$$P_E = \sum_{i=1}^{k+1} f_i \cdot l_i$$

where f_i is the frequency of occurrence of the symbol s_i and l_i is the length of codeword assigned to s_i.

In proposed parallel regeneration of run-length symbols, a single clock overhead is caused by every minority bit appearing at any of the bit positions $1, 2, \ldots, k-1$ in the vectors of a test set. So the total overhead of symbol regeneration O_r in terms of clock cycles is determined as:

$$O_r = \text{Number of minority bits in test set} - \text{Number of minority bits at bit position k}$$

The total number of clocks T_P needed to regenerate the complete test set becomes:

$$T_P = (t \times |P|) + O_r$$

These regeneration clocks are $t \times |P|$ for STUMPS and $t \times |P| \times k$ for conventional run-length regenerators [Jas 98, Chan 01, Gonc 02, Chan 03, El M 08]. Since O_r is very small for the biased test sets, the test patterns regeneration time of the proposed schemes is far less than available methods and it is very close to STUMPS.

Implementation of the proposed run-length compression would be demonstrated in chapter 6 by developing a built-out self-test scheme.

4.3 Nearly Complete Reseeding

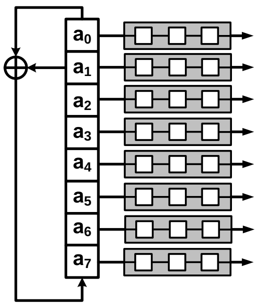
We present a novel linear decompressor based scheme named *nearly complete reseeding* (NCR) that offers higher encoding efficiency for irregular test sets while guaranteeing complete encoding. This has been achieved by discovering the regularity in the encoding capabilities of a linear decompressor. The NCR is based on the observation that ignoring few inconsistent bits during seed computation enables to encode significantly large number of care bits in the seed of a linear decompressor in a regular way.

To demonstrate the basic idea of NCR with a small example, suppose we have the circuit in figure 4.4(a) where an 8-bit LFSR feeds 8 scan chains. Here test vector size is 8 and a test pattern is formed by three vectors. Figure 4.4(b) shows seed computation for an encodable pattern p_1 of this circuit. In order to compute a seed for p_1, we represent each care bit of p_1 with an equation in terms of the seed variables $a_0, a_1, \ldots, a_7$. The solution of the obtained system of equations results in the seed s_1.

Some equations may cause the linear equation system to become unsolvable because of the linear dependency which prevents certain patterns from being encoded. The pattern p_2 in figure 4.4(c) is an example of it. As can be seen in the figure, it is not possible to satisfy the first two equations simultaneously. Same is true for the last three equations. So this system of equations can not be solved and hence no seed exists that can encode p_2. If the equations $a_3 = 0$ and $a_1 + a_2 = 1$ in the equation system of p_2 are ignored, the system becomes consistent and a seed can be computed.

The ignored bits are separately stored and embedded into the LFSR generated patterns during the test. In the following we analytically estimate the encoding efficiency of the presented idea and compare it with existing methods.

In the sequel, we use the probabilistic model developed in [Hell 95, Toub 96] in order to estimate the impact of ignoring a certain number of equations. The probability $P_{seed}(l, c, i_{max})$ of encoding a pattern with c care bits using an l-bit LFSR while allowing i_{max} bits to be ignored can be estimated by considering the process of equation generation as a Markov chain $(X_\tau)_{1 \leq \tau \leq c}$ over a set of states $\{0, 1, \ldots, l\}$. Suppose $X_{e,d,i}$ represents the state of Markov chain at time τ and is interpreted as, $e + i$ equations have been



(a) Example Circuit

$$
\begin{array}{ccc}
\underline{p_1} \\
\mathrm{X\ X\ 0} \\
\mathrm{X\ X\ X} \\
\mathrm{0\ X\ 1} \\
\mathrm{X\ X\ X} \\
\mathrm{X\ X\ X} \\
\mathrm{X\ X\ 0} \\
\mathrm{X\ 1\ X} \\
\mathrm{1\ 0\ X}
\end{array}
\Rightarrow
\begin{array}{ll}
a_0 & = 0 \\
a_2 & = 1 \\
a_4 & = 0 \\
a_5 & = 0 \\
a_7 & = 1 \\
a_0 + a_1 & = 0 \\
a_1 + a_2 & = 1
\end{array}
\Rightarrow
\begin{array}{c}
\underline{s_1} \\
0 \\ 0 \\ 1 \\ \mathrm{X} \\ 0 \\ 0 \\ \mathrm{X} \\ 1
\end{array}
$$

(b) Seed computation

$$
\begin{array}{c}
\underline{p_2} \\
\mathrm{0\ X\ X} \\
\mathrm{1\ X\ 0} \\
\mathrm{X\ 1\ X} \\
\mathrm{1\ X\ 0} \\
\mathrm{X\ X\ 0} \\
\mathrm{0\ 1\ X} \\
\mathrm{X\ X\ 1} \\
\mathrm{1\ 1\ X}
\end{array}
\Rightarrow
\begin{array}{ll}
a_3 & = 0 \\
a_3 & = 1 \\
a_4 & = 0 \\
a_5 & = 1 \\
a_6 & = 1 \\
a_7 & = 0 \\
a_0 + a_1 & = 1 \\
a_1 & = 0 \\
a_2 & = 0 \\
a_1 + a_2 & = 1
\end{array}
\Rightarrow
\begin{array}{c}
\underline{s_2} \\
\text{No Seed}
\end{array}
$$

(c) Unsolvable system of equations

Figure 4.4: The seed computation using NCR

generated up to τ, e of them are solvable with rank d while i equations have been ignored in order to make the system of e equations solvable. Generating a new equation at interval $\tau + 1$ has one of the following consequences

$$\begin{tikzcd}
& X_{e,d,i} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
X_{e+1,d+1,i} & X_{e+1,d,i} & X_{e,d,i+1}
\end{tikzcd}$$

Either the new equation increases the rank d of the system and the system remains solvable or the rank does not increase but the system is still solvable. The third possibility is that the system becomes inconsistent and the currently generated equation is ignored to make it solvable. The transition probabilities for above three cases can be derived as follows

Since the rank is d in state $X_{e,d,i}$ there are in total $2^d - 1$ equations that are dependent on the $e + i$ equations. $2^d - 1 - e - i$ of them are not in the current system and the total number of equations that are not contained in the system are $2^l - 1 - e - i$. Assuming that the $2^d - 1 - e - i$ equations are uniformly distributed within $2^l - 1 - e - i$ possible equations, the fraction of them that corresponds to the linearly dependent equations in the current m bits long pattern will be:

$$(m - e - i) \cdot \frac{2^d - 1 - e - i}{2^l - 1 - e - i}$$

Let

$$\begin{aligned} \Lambda &= m - e - i \\ \Gamma &= 2^d - 1 - e - i \\ \Upsilon &= 2^l - 1 - e - i \end{aligned}$$

The number of linearly independent equations in the test pattern then becomes

$$\Lambda - \Lambda \cdot \frac{\Gamma}{\Upsilon}$$

Hence the probability to generate a linearly independent equation out of Λ possible equations is:

$$\frac{\Lambda - \Lambda \cdot \frac{\Gamma}{\Upsilon}}{\Lambda} = \frac{2^l - 2^d}{\Upsilon}$$

With this probability the new equation will increase the rank of the system and a solution is guaranteed.

The probability that the new equation will not increase the rank of the system will be:

$$1 - \frac{2^l - 2^d}{\Upsilon} = \frac{\Gamma}{\Upsilon}$$

If the rank of the current system does not increase, the new equation may or may not cause the system of equations to become inconsistent. The probability that the system becomes unsolvable in this case is considered as:

$$\frac{1}{2} \cdot \frac{\Gamma}{\Upsilon}$$

Any combination of i out of $e + i$ equations can be ignored to make the system solvable. The number of all the combinations of i bits is

$$\varsigma = \frac{(e + i)!}{i! \cdot e!}$$

So the probability that we can not find a combination of i equations such that the remaining e equations become solvable will be

$$\left(\frac{1}{2}\cdot\frac{\Gamma}{\Upsilon}\right)^{\varsigma}$$

The probability that the new equation does not increase the rank d but the system of equations still remains solvable becomes

$$\frac{\Gamma}{\Upsilon}-\left(\frac{1}{2}\cdot\frac{\Gamma}{\Upsilon}\right)^{\varsigma}$$

If $i < i_{max}$, the probability that current system of equations is made solvable by ignoring the new equation is

$$\left(\frac{1}{2}\cdot\frac{\Gamma}{\Upsilon}\right)^{\varsigma}$$

Assuming the initial probability distribution of Markov chain as $P(X_{1,1,0}) = 1$, the transition probabilities can be summarized as

$$P(X_{e+1,d+1,i}|X_{e,d,i}) = \begin{cases} \frac{2^l-2^d}{\Upsilon} & \text{if } 1 < d+1 \leq l, \\ 0 & \text{otherwise} \end{cases}$$

$$P(X_{e+1,d,i}|X_{e,d,i}) = \begin{cases} \frac{\Gamma}{\Upsilon} - (\frac{1}{2}\cdot\frac{\Gamma}{\Upsilon})^{\varsigma} & \text{if } d > 0 \text{ \& } \tau+1+i \leq 2^d-1 \text{ \& } i \leq i_{max}, \\ 0 & \text{otherwise} \end{cases}$$

$$P(X_{e,d,i+1}|X_{e,d,i}) = \begin{cases} (\frac{1}{2}\cdot\frac{\Gamma}{\Upsilon})^{\varsigma} & \text{if } d > 0 \text{ \& } \tau+1+i \leq 2^d-1 \text{ \& } i < i_{max}, \\ 0 & \text{otherwise} \end{cases}$$

The total probability of finding an l-bit seed to encode a pattern with c care bits while allowing to ignore i_{max} bits can be computed as

$$P_{seed}(l,c,i_{max}) = \sum_{d=\lceil log_2(c+1)\rceil}^{min(l,c)} \sum_{i=0}^{i_{max}} P(X_{c-i,d,i})$$

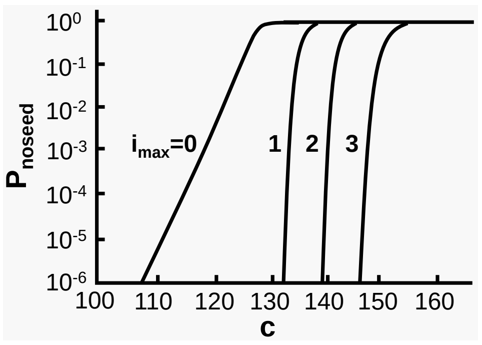


Figure 4.5: Values of $P_{noseed}(l, c, i_{max})$ for $l = 128$

Where $P(X_{e,d,i})$ is computed with the recursive function

$$\begin{aligned} P(X_{e,d,i}) &= P(X_{e-1,d,i)} \cdot P(X_{e,d,i}|X_{e-1,d,i}) + \\ &\quad P(X_{e-1,d-1,i}) \cdot P(X_{e,d,i}|X_{e-1,d-1,i}) + \\ &\quad P(X_{e,d,i-1}) \cdot P(X_{e,d,i}|X_{e,d,i-1}) \end{aligned}$$

Figure 4.5 shows the set of curves $P_{noseed}(l, c, i_{max})$ as the function of c with parameters $l = 128$ and $i_{max} = \{0, 1, 2, 3\}$. Here, $P_{noseed}(128, c, 0)$ represents the probability for the conventional reseeding in [Koen 91]. Obviously, by ignoring a small number of equations, a seed can be computed for a significantly larger number of care bits compared to conventional reseeding. The largest gain is achieved by ignoring a single equation and the coding efficiency increases almost linearly if further equations are ignored.

In this example, we are able to save approximately 20 bits of a seed compared with conventional reseeding if we are allowed to ignore one equation. However, the ignored bits have to be encoded by their position in the scan chain. Hence, a rough estimation of the savings in this case is $20 - log(m)$, where m is the pattern length.

In practice, additional savings are obtained for two reasons:

- For the ignored equations, the corresponding value in the pattern generated by the LFSR will assume the opposite value. In some cases, this pattern still detects the target fault, an effect that is exploited in test set compaction [Pome 91], and the LFSR pattern may remain unchanged.
- For circuits complying with STUMPS, the embedding information can even be shared between the scan chains, thereby further increasing coding efficiency.

A built-in self-test scheme implementing nearly complete reseeding will be presented in chapter 7

CHAPTER 5

Self-Test with High Defect Coverage

In this chapter we develop a built-in self-test scheme that achieves high defect coverage with small hardware overhead. The hardware overhead is reduced by applying restrict encoding to efficiently encode large test sets. The problem of having high defect coverage self-test with small hardware overhead is explained in section 5.1. The overview of self-test scheme employing restrict encoding is given in section 5.2. A heuristic to find minimum number of restrict vectors is described in section 5.3. Section 5.4 explains the generation of restrict candidates while the section 5.5 presents a heuristic to reorder test patterns such that the length of restrict candidates is maximized. An analysis procedure to recognize the restrict efficient candidates is described in section 5.6. Section 5.7 gives the encoding of restrict information and section 5.8 evaluates the efficiency of restrict encoding using experimental results.

5.1 Introduction

Many circuits are used in life critical and mission critical applications. A malfunction in such a circuit may lead to substantial loss of human life or wealth. Therefore it is necessary to ensure that such circuits are free of any defects. The defect coverage of a chip is enhanced by detecting each stuck-at fault multiple times with different patterns [Ma 95, Redd 97, Chan 98, Grim 99, Pome 99]. The defect coverage is increased with multiple detections because each fault is generally targeted in several different ways, increasing the probability to activate a particular defect when the observation path to the fault site opens up [Wang 08a]. In order to ensure high reliability and defect coverage for life critical and mission critical ICs, a self-test with multiple detection of stuck-at faults is required. The size of a test set detecting each fault multiple times is large and its storage on-chip requires significant chip area.

In general, multiple detect test sets have low care bits density. These test sets show strong regularity for two main reasons:

1. Due to low care bits density many test vectors are compatible to each other.
2. Some input assignments appear repeatedly in many patterns at the same positions. This is because certain circuit parts get sensitized by these assignments, and multiple patterns that test the faults in these parts keep these assignments constant.

Dictionary encoding is suitable for test sets with strong regularity. However the available dictionary based encoding schemes are not able to fully exploit the regularity in large test sets since an index of the dictionary is required to encode each care and don't care vector. A large test set contains large number of don't care vectors. Due to the encoding cost of don't care vectors, employing available dictionary encoding schemes for large test sets may result in storage requirements worse than reseeding.

In this chapter we develop a built-in self-test scheme that minimizes the storage requirements of large test sets using *restrict encoding* (RE). The next section gives an overview of realizing restrict encoding in BIST.

5.2 Overview of the Proposed Scheme

The *restrict encoding* combines reseeding of an LFSR with a small dictionary of previously calculated restrict values. Figure 5.1 shows the basic structure of restrict encoding. The LFSR is provided with seed information and the restrict vector (RV) ROM is addressed by a bits. For each test vector, either the output of the LFSR, or a restrict value from the ROM is selected using the *Slct* signal. The ROM addresses and the selection signal for each test vector are read from a register file which holds the current states of all the vector positions. The registers are updated using the instructions stored in test program (TP) memory.

Figure 5.2 gives an overview of the status registers . Each of the t registers is $a + 1$ bits long and a register is updated with new information if the write–enable signal *Wen*

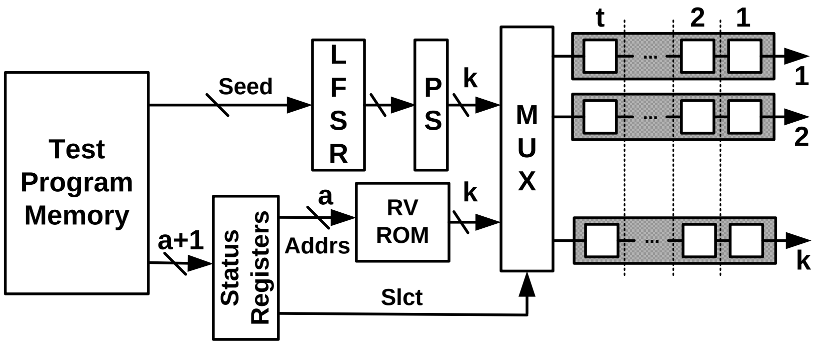


Figure 5.1: BIST with restrict encoding

is 1. By addressing the status registers with a modulo t counter, the same sequence of addresses and selection signals is generated for each pattern. Therefore, restrict values are continuously injected at the same vector position over multiple patterns. The injection starts and ends by updating the values of the status register accordingly. This way a single dictionary index is stored for a restrict rather than individual test vectors.

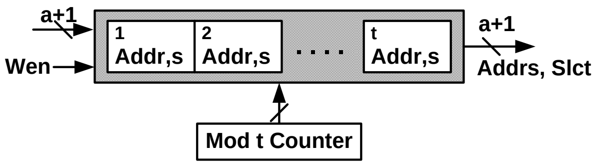


Figure 5.2: Status register for generating restrict information

Maximizing the gain for restrict encoding leads to the following optimization goals:

- Construct a small RV ROM with restrict values that can be injected as often as possible.
- Maximize the run lengths of the restricts so that the required updates in the status register and hence the size of the test program memory are minimized.

5.3 Generation of the Restrict Vector ROM

Let P be a test set with $|P| = q$. The *vector set* V_P of the test set P is the set of all unique test vectors present in $p_1, \ldots, p_q \in P$. A vector v at position i in V_P is expressed as $V_P(i) = v_i$. The *cardinality* of a vector v_i with respect to P is $c_P(v_i) = n$ with n being the number of vectors in $p_1, \ldots, p_q \in P$ that are equal to v_i, and the *weight* of v_i

is $w_P(v_i) = c_P(v_i) \cdot |v_i|$. The *vector slice set* V_i of the pattern set P at vector position $1 \leq i \leq t$ is equal to $\{p(i)|p \in P\}$.

A set of 5 patterns for a circuit with $t = 6$ and $k = 4$ is shown in figure 5.3. For easier demonstration of restrict encoding concepts, it is assumed that scan chains are vertical. In rest of the chapter same presentation of test patterns will be followed. In this figure V_1, V_2, V_3 and V_4 represent vector slice sets for all 4 vector positions. The corresponding unique vector set V_P is shown in figure 5.3 along with the *cardinality* and *weight* of $v_i \in V_P$. There are 4 instances of v_1 in the example test set so its cardinality is 4 and weight is 12 since it contains 3 care bits.

	V_1	V_2	V_3	V_4
P_1	X 1 1 X X 0	0 X 1 X 0 1	1 X 1 0 X X	X 0 0 1 X X
P_2	1 X 0 1 0 X	X 0 0 X X 1	1 X 1 0 X X	1 X X 0 0 X
P_3	0 X 1 X 0 1	X 0 1 1 X X	X 1 1 X X 0	X 1 1 X X 0
P_4	X 0 0 1 X X	X 0 0 1 X X	1 X X 0 0 X	X 0 0 X X 1
P_5	X 0 0 X X 1	1 X 0 1 0 X	1 X X 0 0 X	X 1 1 X X 0

Figure 5.3: Example test set with $k = 6$, $t = 4$ and $|P| = 5$

	V_P						$c_P(v_i)$	$w_P(v_i)$
v_1	X	1	1	X	X	0	4	12
v_2	0	X	1	X	0	1	2	8
v_3	1	X	1	0	X	X	2	6
v_4	X	0	0	1	X	X	3	9
v_5	1	X	0	1	0	X	2	8
v_6	X	0	0	X	X	1	3	9
v_7	1	X	X	0	0	X	3	9
v_8	X	0	1	1	X	X	1	3

Figure 5.4: Unique vectors in example test set

In restrict encoding only those sequences of consecutive test vectors are restricted where restricting offers a gain over reseeding. So it is not required to have a complete dictionary rather a set of fully specified vectors is needed that can produce all the restricted vectors. In this case the goal is to find a minimum set RV of fully specified restrict vectors, so that a large number of care bits can be encoded by replacing the original vectors with compatible restrict vectors.

More formally, the sum over the weights of vectors restrictable with a set RV

$$\sum_{v \in V_P} w(v)[\exists v_r \in RV \text{ with } v \sim v_r]$$

should be large. The final restrict vector set is determined in a two-step process. The first step constructs a candidate set RV_c of restrict vectors. The second step selects the final restrict vectors $RV \subseteq RV_c$ as will be described in the next section.

To construct restrict candidate set RV_c, we represent unique vector set V_P as a node–weighted *compatibility graph* $G(E, V_P, w)$ with

$$(v_i, v_j) \in E \Leftrightarrow v_i \sim v_j \qquad \forall v_i, v_j \in V_P$$

The *compatibility graph* $G(E, V_P, w)$ is partitioned into cliques. Each clique contains pairwise compatible vectors which are merged into a single restrict vector candidate $v_r \in RV_c$. If a v_r contains don't cares, they are filled according to the compatible vectors with maximum weight from other cliques.

The clique partitioning problem has exponential complexity and the following simple heuristic is used:

COMPUTE-RESTRICTS:

1. Let $RV_c = \emptyset$.
2. C=FIND-NEXT-CLIQUE.
3. Let $V_P = V_P - C$ and add $v_r = \Sigma C$ to RV_c.
4. Unless $V_P = \emptyset$ go to step 2.
5. Return RV_c.

FIND-NEXT-CLIQUE:

1. Choose a $v \in V_P$ with $w(v)$ maximum, let $C = \{v\}$.
2. Let $N = \{v' | v' \in V_P - C, (v, v') \in E \ \forall v \in C\}$ be the set of all common neighbors. Unless $N = \emptyset$, add v' with largest $w(v')$ to C and repeat step 2.

3. Return C.

Figure 5.5 shows an example of clique partitioning for the unique vectors shown in figure 5.3. Each node is labeled with the vector index and its weight. The compatibility between two vectors is represented by connecting them with an edge. Initially, being the highest weight node, a clique is started with v_1. Then v_7 is added in the clique because it has maximum weight amongst the neighboring nodes. As v_3 is the only common neighbor of v_1 and v_7, the clique is completed by adding v_3 in it. The process continues after removing v_1, v_3 and v_7 from the graph. As a result the graph in the figure is partitioned in three cliques which are shown next to it. The unique vectors in each clique are merged together to form restrict candidate set RV_c. This restrict candidate set is shown in figure 5.3 for our example.

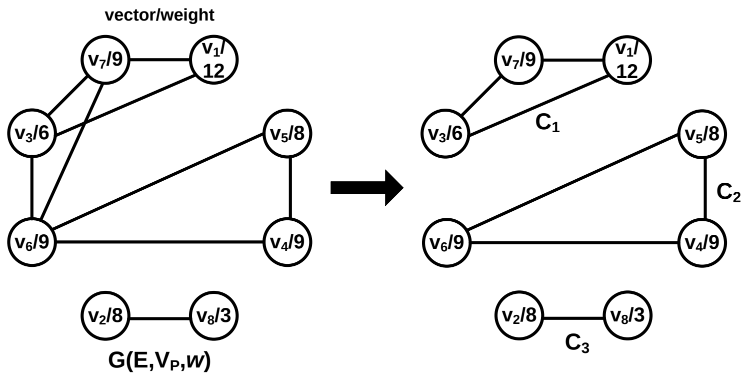


Figure 5.5: Example of clique partitioning

After clique partitioning, every $v \in V_P$ is compatible with at least one of the $v_r \in RV_c$. The next section will use some of these candidates for restricting and therefore determine the subset of RV_c, that has to be stored in a ROM.

	RV_c					
v_{r1}	1	1	1	0	0	0
v_{r2}	1	0	0	1	0	1
v_{r3}	0	0	1	1	0	1

Figure 5.6: Candidates of restrict vectors

5.4 Generation of the Restrict Candidates

A set R_c of *restrict candidates* is generated by scanning through the vector slice sets V_i. A restrict candidate $r = (s, e, i, v_r) \in R_c$ is a four tuple consisting of a starting index s, an ending index e, a vector position i and the restrict vector v_r. The following procedure scans through the list of vectors at a certain position $1 \leq i \leq t$ and generates restrict candidates in a greedy manner. The candidates are determined by successively computing the intersections of the sets of restrict values compatible to the considered vectors.

1. Let starting index $s = 1$.
2. Let ending index $e = s$.
3. Let T contain all the restrict values compatible with the i^{th} vector in the s^{th} pattern:
   $T = \{v_r \in RV_c | p_s(i) \sim v_r\}$.
4. Intersect T with the restrict values compatible with the i^{th} vector of the next pattern:
   $T' = T \cap \{v_r \in RV_c | p_{e+1}(i) \sim v_r\}$
5. If $T' \neq \emptyset$ then $T = T'$, $e = e + 1$ and go to step 4.
6. Now, T holds the last non–empty intersection. Add the restrict $(s, e, i, v_r \in T)$ to R_c.
7. Set $s = e + 1$ and if $s < |P|$, go to step 3.
8. Return R_c.

Figure 5.7 highlights restrict candidates for our example test set in figure 5.3. As every test vector is compatible with only a single restrict value in our example, consecutive vectors in a vector slice set form a single restrict candidate if they belong to the same restrict value and multiple restrict candidates otherwise. Vector slice set V_3 contains a single restrict candidate because all the vectors are compatible with v_{r1} while V_1 consists of 4 restrict candidates because only last two consecutive vectors belong to the same restrict value.

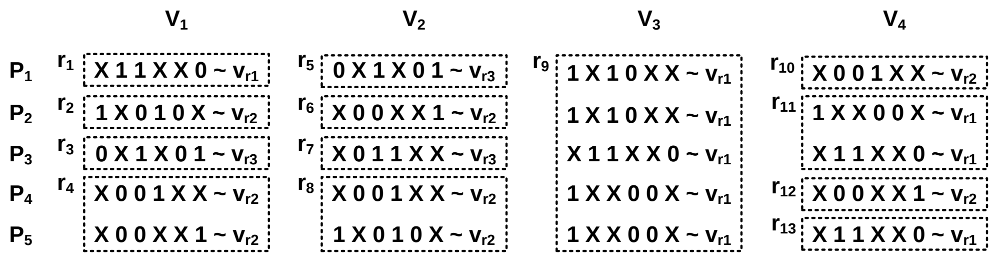


Figure 5.7: Example test set before reordering

5.5 Test Set Ordering

In the proposed scheme a single restrict vector from ROM is used to encode all the test vectors in a restrict candidate. This is similar to static test pattern compaction [Kaji 94, Hamz 00, Lin 01] where multiple compatible patterns are merged to shorten the test set. In this approach, multiple compatible vectors are jointly encoded with minimum information to improve coding efficiency. The gain of restrict encoding increases if large number of care bits are covered with minimum number of restrict candidates. This can be achieved by ordering the patterns such that the total number of restrict candidates decreases in the test set.

To order the patterns, a gain function is defined that expresses the benefit of sorting two patterns adjacent to each other. This benefit increases with the number of compatible restrict vectors shared by these two patterns. However, each vector v can be compatible to multiple restrict vector candidates $v_r \in RV_c$ and it is not yet clear, which restrict vector is likely to be used at the end. The following procedure determines for each vector v a single restrict value v_r that is most capable in restricting large amounts of care bits.

Each vector position $1 \leq i \leq t$ is considered separately. The weight of a restrict vector with respect to a vector position i is defined as the overall weight of its compatible vectors in V_i:

$$w_i(v_r) = \sum_{v \in V_i} |v|[v_r \sim v]$$

Figure 5.5 shows the weights of restrict vectors in figure 5.3 for all 4 vector slice sets. As no vector is compatible with v_{r1} in vector slice set V_2, its weight is 0 with respect to V_2 and it is 15 with respect to V_3 because it is compatible with all the test vectors containing a total of 15 care bits.

	V_1	V_2	V_3	V_4
v_{r1}	3	0	15	9
v_{r2}	10	10	0	6
v_{r3}	4	7	0	0

Figure 5.8: Weights of the restrict candidate vectors for the vector slice sets

The more care bits a certain restrict vector can cover at position i, the higher is the weight. The restrict vectors with a higher weight should have priority over restrict vectors with a lower weight because it is more likely, that they are able to cover more care bits in longer runs. Thus, each vector $v \in V_i$ is associated with the compatible restrict vector v_r of highest weight $w_i(v_r)$. This single restrict vector associated with v is noted as $r(v)$. Each pattern now has t restrict vectors associated with it; one at each vector position.

The benefit gained by putting two patterns p_m, p_n in the pattern set adjacent to each other depends on the following factors:

- The number of vector positions for which these two patterns share the same restrict vector $(r(p_n(i)) = r(p_m(i)))$.
- The amount of care bits covered by these common restrict vectors.

Based on these factors, the *similarity* of a pair of patterns $p_m, p_n \in P$ is defined as:

$$s(p_m, p_n) = \sum_{i=1}^{t} (|p_m(i)| + |p_n(i)|) \cdot y(p_m(i), p_n(i)) \quad \text{with}$$

$$y(v_j, v_k) = \begin{cases} 2 & \text{if } |v_j| \cdot |v_k| > 0 \text{ and } r(v_j) = r(v_k), \\ 1 & \text{if } |v_j| \cdot |v_k| = 0, \\ -1 & \text{otherwise.} \end{cases}$$

The first term weights the outcome of y by the number of care bits at each position. y is positive only if the two vectors are compatible. It evaluates to 2, if both vectors have care bits and are associated with the same restrict vector.

For a given pattern set P and with the metric defined above, a similarity graph $S(P, s)$ is constructed. This graph is edge weighted, undirected, and complete. Each node represents a pattern and the edges between the patterns are weighted with the similarity of the two adjacent nodes. The patterns are ordered by finding a maximum weight path in S such that every node is visited only once. It is an instance of the travelling salesman problem, so the best solution cannot be found in polynomial time. The following greedy heuristic however, provides sufficient results.

1. Let $s(p_0, p_1)$ is maximum in S
2. Let $L = (p_0, p_1)$ be a pattern list and $p = p_1$.
3. Find a $p' \in P$ with $s(p, p')$ maximum. Let $p = p'$, $P = P - \{p\}$ and append p to list L.
4. Unless $P = \emptyset$, go to step 2
5. Return L.

Figure 5.9 shows similarity graph for our example. The traversal starts by selecting the pattern pair (P_2, P_5) because of its maximum similarity. Amongst the remaining patterns P_5 has the maximum similarity with P_4, so it is selected next. This way traversal yields the pattern order $(P_2, P_5, P_4, P_1, P_3)$. The ordered patterns along with the new restrict candidates are shown in figure 5.10. The total number of restrict candidates have been reduced from 13 to 9 because of the new pattern order. Please note that either we select P_2 as the first pattern or P_5, it yields the same total similarity as well as the total number of restrict candidates.

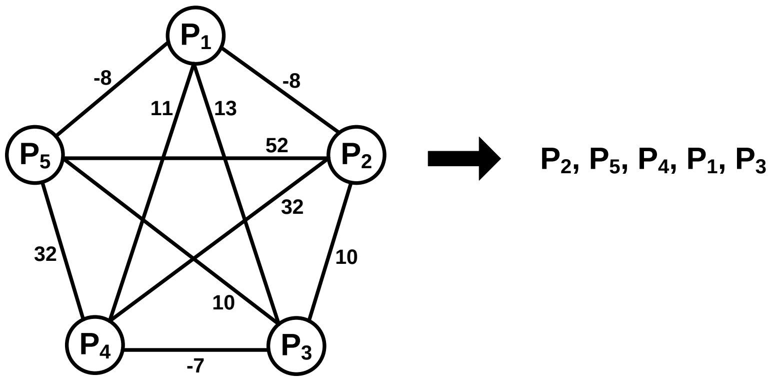


Figure 5.9: Similarity graph for example test set

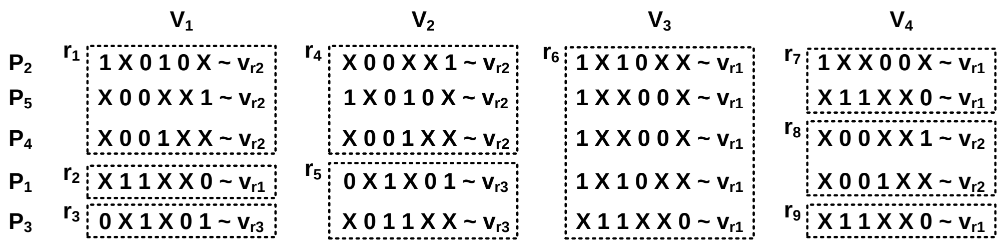


Figure 5.10: Example test set after reordering

5.6 Restrict Candidates Analysis

The goal of restrict candidates analysis is to distinguish between restrict efficient and reseed efficient candidates. For this purpose restrict and reseed storage cost of every candidate $r_i \in R_c$ is estimated and the candidates for whom restricting offers a gain over reseeding are selected as final restricts.

During the test an action indicates a required update of status register or LFSR seed. The status register needs to be updated to indicate the start of a restrict or start of seed encoding while LFSR needs to be reseeded whenever a test vector raises a conflict with the current seed. Every action needs to be executed at a certain shift cycle which could be determined by the pattern indices and the vector positions. This information is stored efficiently if encoded as the number of shift cycles to the next action, called delay. The number of bits required to store this delay information is estimated as $d = \lceil log_2(max\_delay) \rceil$. Since three different types of actions are required, a 2-bit flag is needed to distinguish between them. So total storage cost to indicate an action is $S_{action} = 2 + d$ bits.

The address of associated restrict value needs to be stored with a restrict start action while the cost of storing LFSR seeds should be considered for the candidates encoded with seeds. The address of a restrict value is $a = \lceil log_2(|RV_c|) \rceil$ bits long, which fixes the total cost of restricting a candidate to

$$S_{restrict} = S_{action} + a$$

The seeds storage of a candidate is always proportional to the number of care bits in it. Let $|r_i|$ denote the number of care bits in a candidate and e the expected encoding efficiency of LFSR then the ratio $S_{seed} = \frac{|r_i|}{e}$ gives an estimate of the seeds storage for r_i.

The total cost of encoding a candidate with seeds then becomes

$$S_{seedenc} = S_{action} + S_{seed}$$

The action information S_{action} is needed only at the beginning of first candidate if a group of candidates adjacent to each other is encoded with seeds. So in the first step of our analysis we only consider the minimum cost S_{seed} that has to be paid for every candidate encoded with the seeds. The seeds cost S_{seed} is estimated for each candidate and the candidates for whom $S_{seed} > S_{restrict}$ are selected to be encoded with restricts. In the second step undecided consecutive candidates are considered together. Let $G = \{r_1, r_2, \dots, r_n\}$ denotes a group of adjacent restrict candidates then the seed encoding cost of this group is estimated as

$$S_{seedenc,G} = S_{action} + \sum_{i=1}^{n} S_{seed,i}$$

while the restrict cost for G becomes

$$S_{restrict,G} = S_{restrict} \cdot n$$

If $S_{restrict,G} < S_{seedenc,G}$ then all the candidates in G are marked as restricted otherwise they are encoded with the seeds.

In our example (figure 5.10) maximum delay between two actions is 7 which makes $d = 3$ and $S_{action} = 5$. There are three restrict values in RV_c, hence $a = 2$ and $S_{restrict} = 7$. Maximum encoding efficiency that could generally be achieved with an LFSR seed is 1 [Bala 07]. Assuming $e = 1$ yields

$$S_{seed} = 10_{r_1}, 3_{r_2}, 4_{r_3}, 10_{r_4}, 7_{r_5}, 15_{r_6}, 6_{r_7}, 6_{r_8}, 3_{r_9}$$

In the first step candidates r_1, r_4, r_6 are selected as final restricts because their minimum cost of seed encoding is higher than restrict cost which is 7. In the second step remaining candidates are collected in groups $G_1 = \{r_2, r_3\}, G_2 = \{r_5\}, G_3 = \{r_7, r_8, r_9\}$. The seed encoding cost of first group $S_{seedenc,G_1}$ is 12 which includes 5 bits to store action information and 7 bits as cumulative seed storage cost for r_2 and r_3. The restricting cost for the same group is 14 because it contains 2 candidates. Since seed encoding cost of G_1 is less than its restricting cost, r_2 and r_3 are encoded with the seeds. Same way G_2 is selected for restricting and G_3 for seed encoding. Figure 5.11 shows the final restricted and the seed encoded portions of our example patterns. The vectors written in black text are encoded with restrict values while the vectors with white text are encoded with the seeds. The dark Grey and black backgrounds of vectors represent start of a restrict and seed encoding respectively. The status register is updated at these positions using the information stored in the memory.

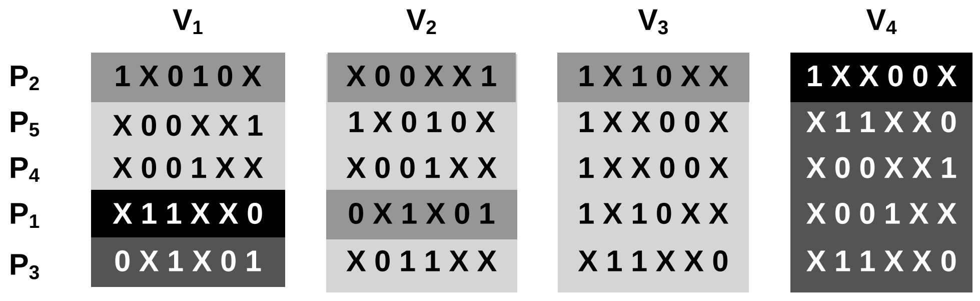


Figure 5.11: Restricted and reseeded portions of example patterns

5.7 Test Program Encoding and Decompression

The compressed test data consists of a sequence of actions. An action is either an update for the status register (fig. 5.2) or a reseeding of the LFSR. Each action corresponds to a command in a test program. Each command consists of the action to be performed and the number of shift cycles to the next command, called *delay*. A dictionary address or an LFSR seed is also part of the command if the required action is a restrict or LFSR reseed respectively.

Three actions namely restrict start, seed encoding start and LFSR reseeding are possible. A two bit prefix is used to specify the required action. The length of delay field is determined by maximum delay between two consecutive actions making delay field $\lceil log_2(max\_delay)\rceil$ bits wide while $\lceil log_2(|RV|)\rceil$ bits encode the address of RV ROM. The length of seed data is equal to the size of LFSR. Figure 5.12 shows the format of commands to express the actions of starting seed encoding, starting a restrict and the need to reseed LFSR.

These commands are stored in a bit addressable memory and a small decoder is used to decode these commands sequentially. Initially decoder reads $2 + \lceil log_2(max\_delay)\rceil$ bits. If the 2 bit prefix indicates that the required action is a restrict start then next $\lceil log_2(|RV|)\rceil$ bits are read and if the action is reseeding of LFSR then the bits equal to LFSR size are read. The decoder works independently and sets *ready* signal high

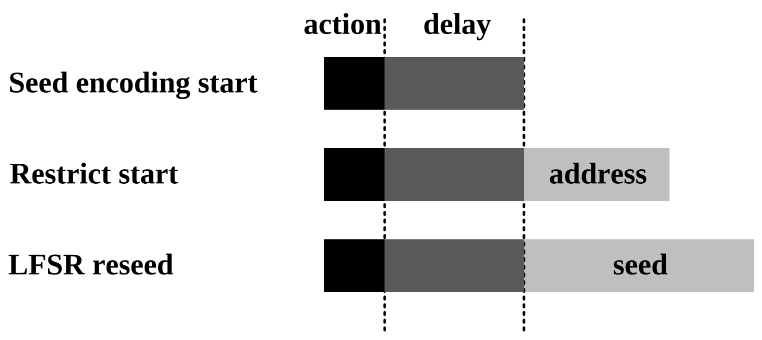


Figure 5.12: Commands for restrict encoding

whenever the next instruction is ready to be executed. A pipeline is used to ensure that after the execution of a command the control unit does not has to wait the decoding of the next command to continue its operation. Since mostly there is a large delay between two consecutive actions a single level pipeline suffices to avoid any noticeable degradation in the test time. The chances of any possible degradation in the test time can further be reduced by increasing the depth of the pipeline.

Suppose two 11 bit seeds *10101010101* and *01010101010* encode the unrestricted vectors in our example and a reseed action is required at shift cycle 0 and shift cycle 11. Figure 5.13 shows the resulting test program for our example, stored in a 16 bits wide memory. It is assumed that the status registers are reset to seed encoding before starting the test. The first two commands in the test program indicate the loading of LFSR seed and setting the status register for the first restrict prior starting scan shift. Since the first restrict is encoded with the second value in the RV ROM, the address field contains *01*. The third and the fourth commands update the status register, after one shift cycle each, to start the second and the third restrict. If seed encoding for some vector position starts from first pattern, no action is required because the status register is set to seed encoding in the beginning. That is why no command is stored to indicate the start of seed encoding from the fourth vector position of the first pattern in our example (figure 5.11). The last three commands represent LFSR reseeding, start of seed encoding and start of a restrict respectively.

5.8 Experimental Evaluation

Experiments were performed on considerably large industrial circuits. The experimental setup and the results are explained in appendix A.1. The experiments reveal that in large

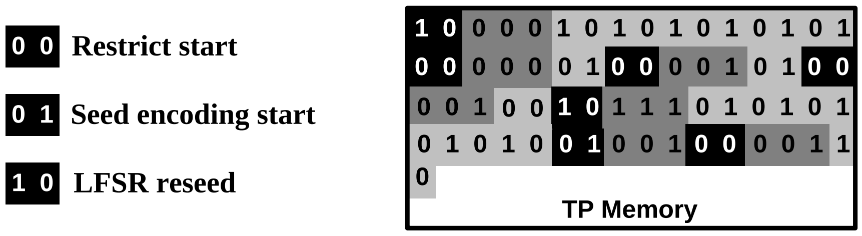


Figure 5.13: Test program for the example

test sets with low care bits density, on average, 80% care bits are encoded with restricts and only 20% with the seeds. The encoding efficiency of care bits encoded with restricts is two to four times higher than the encoding efficiency of remaining care bits that are encoded with seeds. The cumulative encoding efficiency of restrict encoding for all the experimented circuits always remains higher than 1. Among the test data compression methods published so far, no compression method has ever achieved encoding efficiency greater than 1. It is first time in the literature that such high encoding efficiency is achieved. It is made possible only by efficiently exploiting strong regularities in sparse test sets.

The results of restrict encoding are compared with continuous reseeding [Volk 03] which offers the best compression for low care bits density test sets compared to other methods. The comparison reveals that restrict encoding offers an improvement in encoding efficiency ranging from 62 to 186 percent. The average improvement remains higher than 100% reducing the storage requirements to more than the half. Considering the large sizes of high defect coverage test sets for nanometer ICs, this reduction in storage results in great savings in test cost. The most important advantage of restrict encoding over continuous reseeding is that it offers an encoding efficiency higher than the maximum theoretical limit of reseeding. The encoding efficiency of continuous reseeding is as high as 0.9 in our experiments showing that any new reseeding scheme can at most achieve an improvement of 11% only. While in the proposed method an improvement up to 186% has been achieved.

Comparisons are also made with available dictionary encoding schemes [Redd 02, Li 03, Wurt 04, Kim 08]. The comparisons show that restrict encoding offers many times better compression. It is due to the fact that in dictionary encoding an index of dictionary has to be stored for each test vector irrespective of the number of care bits in it and employing dictionary encoding for large test sets with low care bits density results in very high storage requirements. While in the proposed scheme a single index is shared by a large number of test vectors and if it is not possible to encode significantly large number of care bits with dictionary index, reseeding is applied to avoid degradation in the compression.

CHAPTER 6

Self-Test with Minimal Hardware Overhead

This chapter demonstrates the use of *run-length encoding with parallel decompression* (REWPD) by developing a new built-out self-test (BOST) scheme for deterministic testing. This external deterministic self-test minimizes the on-chip test hardware overhead while keeping the test time in the built-in self-test (BIST) range. Section 6.1 defines the problem. The proposed scheme is based on bit-flipping deterministic logic BIST (DLBIST) [Wund 96]. An overview of DLBIST along with the target architecture is presented in section 6.2. The detailed implementation of REWPD compression and decompression are explained in section 6.3. To simplify the test at board level IEEE 1149.1 JTAG boundary scan standard is used as communication protocol for sending data from the test chip to the circuit under test (CUT). Section 6.4 describes this communication protocol while section 6.5 gives an overview of the test configuration. The outcome of experiments performed on large industrial circuits is presented in section 6.6.

6.1 Introduction

In many applications small dimensions of a chip and lowest fabrication cost are desired. For such applications reducing on-chip area overhead to minimal is of prime concern. Deterministic BIST [Daeh 81, Dand 84] required to ensure high quality of fabricated chips need an amount of on-chip area that is expensive. Because of this reason mixed-mode deterministic BIST schemes [Koen 91, Hell 95, Wund 96, Toub 96, Gher 04] are more popular to get desired fault efficiency with lesser area overhead. Though hardware overhead of mixed-mode BIST is much smaller compared to a pure deterministic BIST, still the test hardware may take up to 30% of overall chip area [Gher 04] with an ever increasing tendency. For area critical applications this amount of test hardware is not affordable.

Built-out self-test (BOST) schemes [Bard 82, Eich 83, Hell 90, Stro 91] try to combine the advantages of external testing and BIST by implementing test resources in a special chip. Figure 6.1 shows basic structure of BOST. Here a dedicated test chip replaces the expensive tester and pattern generation, response evaluation and test control are performed by the test chip. The main challenge in the design of a test chip is to keep test information storage as well as communication overhead under acceptable limits. The BOST is attractive for area critical applications because it reduces the on-chip test hardware overhead to a minimal. If the test chip is programmable, a single test chip could be shared by several ICs at board (figure 6.2). Test chips producing random [Bard 82, Eich 83], weighted random [Stro 91] and pseudo-exhaustive [Hell 90] patterns are already presented. Due to high storage requirements and communication overhead a test chip producing deterministic patterns has not been considered feasible up to now.

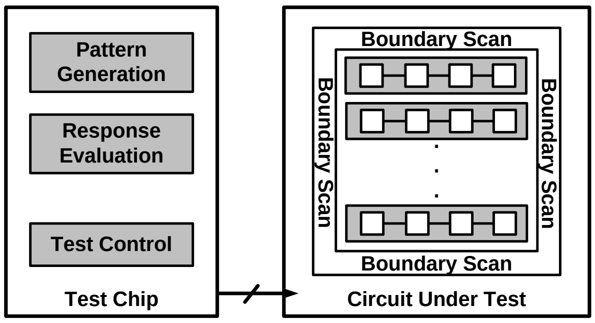


Figure 6.1: The BOST Scheme

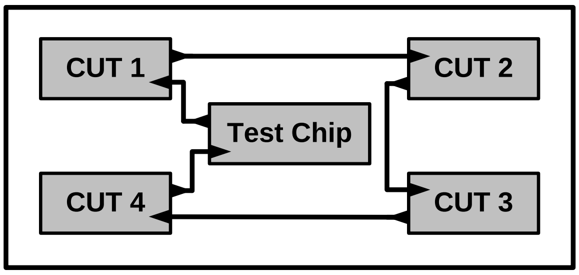


Figure 6.2: BOST at board level

While a pure deterministic BOST solution seems expensive, a mixed-mode deterministic BOST scheme may be developed to achieve complete fault efficiency with very small on-chip area overhead. By adapting *test resource partitioning* (TRP) and test data compression for built-out self-test, the communication overhead and the storage requirements for test chip are minimized. The design of a test chip with mixed-mode deterministic pattern generation poses the following challenges:

- Partition the self-test circuitry between off-chip and on-chip such that the best trade-off is found between the area overhead and the test time
- Minimize the storage requirements for the test chip
- Maintain the test time close to mixed-mode BIST
- Keep the test chip programmable
- Keep the on-chip test hardware independent of the test set
- Use a standard communication interface between the test chip and the CUT

A test chip meeting aforementioned challenges is developed based on bit-flipping DLBIST [Wund 96, Gher 04]. The overview of bit-flipping DLBIST along with the overview of the proposed method will be explained in the next section.

6.2 Overview of the Proposed Scheme

The bit-flipping DLBIST requires less on-chip area compared to programmable mixed-mode deterministic BIST solutions [Wund 96]. Figure 6.3 shows this DLBIST architecture. Here an LFSR is used to generate the pseudo random patterns while an XOR gate is used in front of each scan chain to embed the deterministic patterns into the LFSR generated pseudo random patterns. The XOR gates are controlled by a combinational logic called bit-flipping logic (BFL). The bit-flipping logic (BFL) changes a few bits of the random vectors generated by the LFSR in order to produce precomputed deterministic patterns. In [Wund 96] it is shown that random vectors can be found which leave most of the bits of the BFL output not specified, the remaining bits are mostly 0 as they match with the random vectors, and only very few bits must be 1. The LFSR, the XOR gates and the BFL form the pattern generator of this scheme.

At each clock the LFSR produces a *random vector* v_r and the BFL generates an *embedding vector* v_e. Both r and e are k bits long. A *deterministic vector* v_d, shifted into the scan chains at each clock, is obtained as $v_d(i) = v_r(i) \oplus v_e(i)$ for $1 \leq i \leq k$. Figure

6.4 shows an example of random (P_R), embedding (P_E) and deterministic (P_D) test sets for a circuit with $k = 4, t = 3$ and $|P| = 3$.

The test application is managed by a finite state machine called test control unit (TCU). The TCU mainly contains a vector counter (VC) and a pattern counter (PC). The VC is used to control the shifting of a new pattern into the scan chains while the PC is used to control the length of the test set.

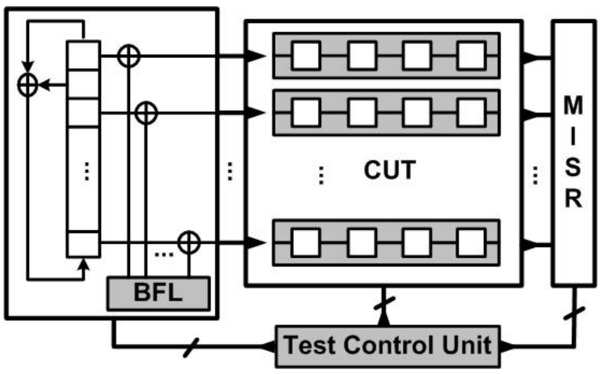


Figure 6.3: Bit-flipping DLBIST

P_R										P_E										P_D								
p_3			p_2			p_1				p_3			p_2			p_1				p_3			p_2			p_1		
v_{33}	v_{32}	v_{31}	v_{23}	v_{22}	v_{21}	v_{13}	v_{12}	v_{11}		v_{33}	v_{32}	v_{31}	v_{23}	v_{22}	v_{21}	v_{13}	v_{12}	v_{11}		v_{33}	v_{32}	v_{31}	v_{23}	v_{22}	v_{21}	v_{13}	v_{12}	v_{11}
0	0	0	0	1	0	0	1	0	$\oplus$	X	X	0	X	X	X	X	0	X	$=$	X	X	0	X	X	X	X	1	X
1	1	0	1	0	1	1	0	0		X	0	X	X	X	0	0	1	X		X	1	X	X	X	1	1	1	X
1	0	1	0	0	1	1	1	1		X	X	1	X	X	0	1	X	X		X	X	0	X	X	1	0	X	X
0	0	0	1	1	1	0	1	1		X	X	1	X	X	1	X	0	X		X	X	1	X	X	0	X	1	X

Figure 6.4: An example of random, embedding and deterministic test sets

In order to find the best partitioning between off-chip and on-chip, major parts of bit-flipping DLBIST and their inputs and outputs are analyzed. The analysis reveals that the LFSR, the XOR gates, the MISR and the TCU do not add much hardware overhead on-chip while their implementation externally would require extensive communication and will result in long test time. So these parts are implemented on-chip. The significant area overhead of DLBIST is caused by the bit-flipping logic that represents hardwired storage of the compressed deterministic test set. So deterministic test set storage is moved from on-chip to the test-chip. To make the test chip programmable we have to store the compressed deterministic patterns in memory. Unlike deterministic patterns with no regularity, the embedding information of DLBIST shows weak regularity due to biasness towards 0. So we compress bit flip information instead of the deterministic patterns.

Figure 6.5 shows an overview of the targeted self-test scheme. The embedding information is compressed and stored in an external chip memory. The on-chip BFL is replaced by a decoder while other test circuitry is still on-chip. The test chip also contains some control logic along with the memory to regulate the test operations. The encoded embedding information is read from the memory bit by bit and sent to the on-chip decoder. The decoder detects the codeword and regenerates the embedding information, which is then applied to the XOR inputs. The on-chip control unit manages the synchronization between the LFSR and the decoded embedding information.

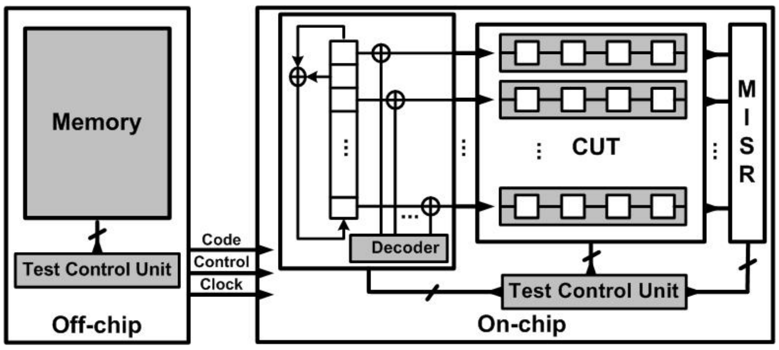


Figure 6.5: External test based on bit-flipping

Challenges like minimizing the on-chip decoder size, keeping the decoder independent of the test set, reducing the off-chip storage and maintaining the test time under throughput all depend upon the compression and the decompression of the embedding patterns. Using our proposed test compression method for weak regularity test sets "run-length encoding with parallel decompression", all these challenges are successfully met. In the next section, the application of the compression method and the decompression architecture are described in detail.

6.3 Compression/Decompression of Embedding Information

6.3.1 Compression Algorithm

Figure 6.6 shows the compression algorithm. The algorithm starts by replacing Xs in P_E with 0s since 0 is the majority bit in P_E. Then an extra bit, that is always 1, is assumed at the end of all the test vectors which end with a 0. After this, assuming maximum symbol length of $k+1$, the test vectors are divided into the runs of 0. If all the bits in a test vector are 0 then we have just one symbol of the length $k+1$ and if all are 1 then we have k symbols each of length 1.

```
1. Fill Xs in $P_E$ with 0
2. Let $v(k+1) = 1 \ \forall \ v_{mn} \in P_E \ \& \ v_{mn}(k) = 0$
3. Let $s_l$ is 0 runlength symbol with $|s_l| = l \ \& \ s(l) = 1$
4. Let maximum symbol length $l_{max} = k+1$
5. Divide each $v_{mn} \in P$ into $s_l$ with $|s_l| \leq l_{max}$
6. Generate a dictionary $D$ with all $s_l$ and their frequency $f$ in $P$
7. Sort $D$ in descending order of $f$
8. Let $D$ starting index $i = 1$
9. Let $D$ ending index $j = k+1$
10. Let code $c = 0$
11. Assign $c$ to $i$
12. Set $i = i+1$
13. Set $c = 1c$
14. Go to step 11 until $i < j$
15. Truncate least significant bit from $c$
16. Assign $c$ to $i$
```

Figure 6.6: Compression Algorithm of REWPD

In the next step a dictionary D of these 0 run-length symbols along with their frequencies of occurrence in P_E is built. The dictionary is sorted in the descending order of the symbol frequency. Then binary codes are assigned to all the symbols in a way that the symbol with the highest frequency gets the shortest code. So the first symbol in D is assigned code 0. The subsequent symbols, except the last one, are assigned a code by appending a 1 in the code of the previous symbol. This way the second symbol in the dictionary will get the code 10 and the third one will get 110. The code of the last symbol is generated by replacing the 0 with 1 in the code of previous symbol. Although the Huffman codes might offer better compression they are not used as they require a decoder that depends on test set. Figure 6.7 shows the application of the proposed compression method on embedding test set P_E from figure 6.4.

p_3			p_2			p_1		
v_{33}	v_{32}	v_{31}	v_{23}	v_{22}	v_{21}	v_{13}	v_{12}	v_{11}
X	X	0	X	X	X	X	0	X
X	0	X	X	X	0	0	1	X
X	X	1	X	X	0	1	X	X
X	X	1	X	X	1	X	0	X

$\Rightarrow$

p_3			p_2			p_1		
v_{33}	v_{32}	v_{31}	v_{23}	v_{22}	v_{21}	v_{13}	v_{12}	v_{11}
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0	0
1	1		1	1		1	1	1

$\Rightarrow$

Symbol	Frequency	Code
00001	5	0
001	3	10
01	2	110
0001	1	1110
1	1	1111

Figure 6.7: An example of REWPD compression algorithm

6.3.2 Decompression Architecture

The block diagram of the decompression architecture is shown in figure 6.8. It mainly consists of a binary decoder, a small memory and a generate unit. The binary decoder receives a codeword serially and produces the address of the memory word that stores the length of the encoded symbol. The length is passed to the generate unit that regenerates the run-length symbol. When the test vector is complete it is shifted into the scan chains. Detection of a binary code, reading of a memory word and generation of a run-length symbol are performed in parallel.

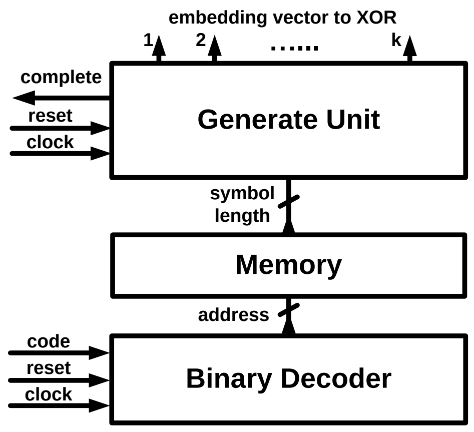


Figure 6.8: Block diagram of REWPD decoder

The binary decoder is a simple finite state machine consisting of at most k states and implemented by $\lceil log_2 k \rceil$ flip-flops. It receives a code bit by bit through the code line, and when the code is complete it generates an address of a memory word. This decoder is designed in a way that on the detection of the shortest code (i.e. 0) it gives the address of the first memory word, on the second shortest (i.e. 10) the address of the second memory word and so on. If the code is not complete, the address of the last memory word appears at the decoder output. This decoder is only dependent on k i.e. number of scan chains in the design and is independent of the test set.

The memory is used to store the lengths of the symbols according to the descending order of their frequencies. So the length of the symbol with the highest frequency and the shortest code (i.e. 0) is always stored in the first word of the memory, the length of the symbol with the second shortest code (i.e. 10) is in the second word and so on. The last memory word, the address of which is produced during the reception of a code, always stores 0. The length of each memory word is $\lceil log_2(k+1) \rceil$ and the maximum number of required words is $k+2$. Therefore the size of the maximum memory required becomes $(k + 2) * \lceil log_2(k + 1)\rceil$ bits. If the test set is changed and hence the frequencies of the different 0 run-length symbols, the memory words are initialized according to the new frequencies. In this way the decoder is independent of the test set. Figure 6.9 shows the state diagram of the binary decoder and the memory contents for the example in figure 6.7.

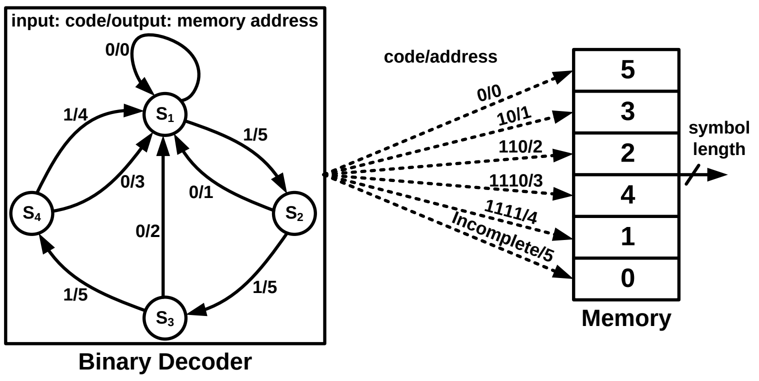


Figure 6.9: Example of the binary decoder and the memory contents

The detailed design of the generate unit is shown in figure 6.10. This architecture is similar as was presented in 4.3(a) except few additional components. The *Register*1 is used to store the total number of generated bits and is $\lceil log_2(k + 1)\rceil$ bits wide. Initially all the registers are set to 0. The length of an encoded symbol is read from the memory and passed to the adder. It adds this length to the contents of the *Register*1 and stores back the results in the *Register*1 while flipping the corresponding bit from 0 to 1 in the *Register*2. This process continues until the comparator detects that the value of the *Register*1 is greater or equal to k i.e. complete test vector has been generated. At this point the generated vector is shifted into the pipeline register and the registers are set back to 0.

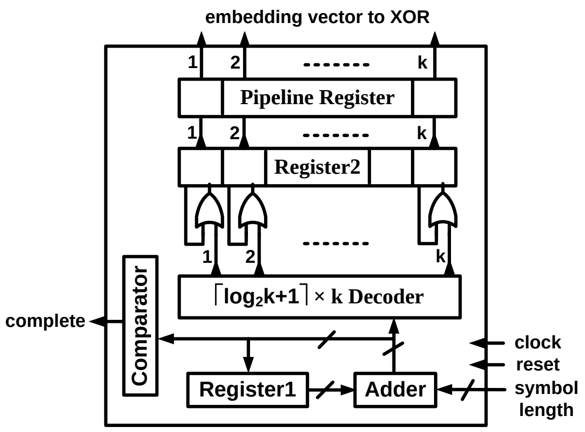


Figure 6.10: Design of the generate unit

6.4 Communication Protocol

The IEEE 1149.1 JTAG boundary scan standard defines a test access protocol and a boundary-scan architecture for digital ICs. Since its approval by the IEEE as a test standard in 1990, the standard has been employed by most electronics companies when building large chips. Today, almost all general-purpose CPU, DSP, and FPGA and many application-specific designs comply with the 1149.1 standard. Because boundary scan provides a simple and efficient protocol for data communication, this standard has also been employed in many other applications, including power management, embedded instrumentation control, clock/PLL control, debugging/diagnosis, verification, and chip reconfiguration [Rear 05].

Figure 6.11 shows a chip with the boundary-scan architecture. It mainly contains a test access port (TAP), a TAP controller, instruction register (IR) and several test data registers. The TAP consists of four mandatory pins called test data input (TDI), test data output (TDO), test mode select (TMS) and test clock (TCK). The TAP controller is a 16-state finite state machine that controls each step of the boundary scan operations. The instruction register is serially loaded with an instruction through TDI, which enables various different operation modes of the test hardware. Several instruction modes are mandatory, others are optional, and user-defined instructions can be added.

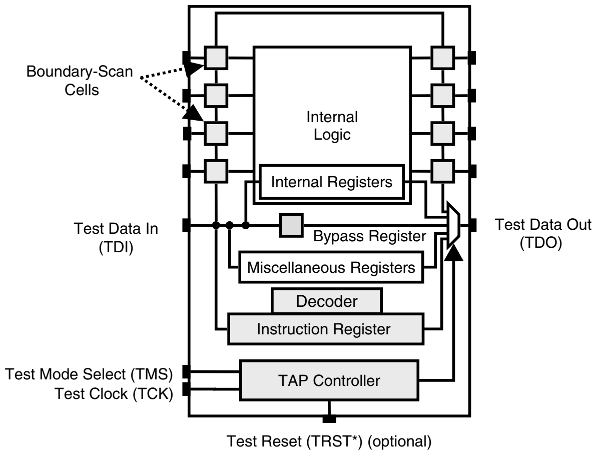


Figure 6.11: Boundary scan architecture [Wang 06]

Easy integration of the test chip is facilitated by adopting boundary scan standard to transmit codes from the off-chip to the on-chip. For the external deterministic self-test only one extra instruction named BFTEST is added. Whenever this instruction is loaded into the instruction register and the TAP controller reaches Run-Test state, it sends a start signal to the on-chip test controller and the TDI is connected with the binary decoder.

6.5 Test Configuration

The testing procedure starts by setting the start test input of the external chip. When this input is set, the external chip controller transfers the BFTEST instruction into the instruction register through instruction scan cycle and then changes the TAP controller mode to Run-Test. When the TAP controller reaches this state, transformation of the codes from the test chip starts. The codes are read bit by bit from the test chip memory and are sent to the binary decoder of the CUT through TDI. The binary decoder detects a codeword and provides the address of the memory word containing the length of the encoded block. This length is passed to the generate unit that regenerates this block using the procedure described in section 6.3.2. When the embedding vector is complete, the comparator sends a signal to the on-chip controller that sets the scan line to shift XOR outputs into scan chains. The controller also decrements the bit counter and changes the state of the LFSR. When bit counter reaches 0, the pattern counter is decremented by 1 and one system clock is applied. Responses of the circuit are captured again in scan chains. Shift in new pattern and shift out responses go in parallel. This process continues until the pattern counter reaches 0.

6.6 Experimental Evaluation

The experimental setup for external deterministic self-test is described in appendix A.2. The experimental results reveal that the proposed compression algorithm offers significantly higher compression percentage for embedding information compared to the most efficient known run-length encoding [El M 08] scheme. The achieved compression is close to the maximum theoretical compression achievable using run-length encoding. The test time comparison with DLBIST demonstrates that the proposed decompression architecture successfully achieves the goal of keeping test time close to DLBIST. Compared to other state of the art run-length encoding schemes [Chan 01, Gonc 02, Chan 03, El M 08] the speed up achieved due to the novel decompression architecture is significant. While minimum 637 million cycles are required using conventional run-length decompressors to complete a test for one of the industrial circuits, only 11 million cycles are required with the proposed method which is close to 8 million cycles required in DLBIST.

Comparing the size of the on-chip test hardware with state of the art deterministic logic BIST (DLBIST) [Gher 04] reveals that significant savings in on-chip test hardware are achieved using the proposed built-out self-test. The on-chip area overhead caused by proposed method could be considered insignificant for large circuits. The tables show that the test hardware of proposed scheme only depends upon the number of scan chains in the circuit and is fully independent of the circuit size or the test set.

CHAPTER 7

Self-Test with Short Test Application Time

In this chapter a built-in self-test scheme aimed to achieve short test time with minimum hardware overhead on-chip is developed by employing *nearly complete reseeding* (NCR). The problem of achieving short test time with smaller hardware overhead is described in section 7.1. An overview of the target BIST scheme realizing the idea of NCR is presented in section 7.2. Section 7.3 describes the seed computation algorithm for "nearly complete reseeding". The compression and the decompression of NCR embedding information is explained in section 7.4 while efficiency of the proposed scheme is evaluated in section 7.5 using the experimental results.

7.1 Introduction

In today's highly competitive semiconductor industry, time-to-market is crucial for many products. The test application time is one of the major contributors in time-to-market. Since the test time is directly proportional to the number of applied patterns, the rapid increase in the test sets sizes of nanometer ICs is posing a serious challenge in speeding up time-to-market within affordable costs.

The size of a test set is shortened by applying full compaction in order to reduce the test time. In built-in self-test, this test set is stored on-chip. The on-chip hardware overhead caused by test set storage is scaled down using the test data compression. The available compression schemes yield poor compression for a fully compacted test set due to high care bits density. This poor compression results in higher hardware cost needed to achieve short test time.

In general, such test sets are irregular so the code based compression schemes are not appropriate for them. The linear decompressor based schemes are more suitable for the test sets without regularity. However, the compression of available linear decompressor based schemes is limited by the number of care bits encoded per seed. In addition, possible failure in finding the seeds of few patterns results in fault coverage loss.

In this chapter we demonstrate the use of *nearly complete reseeding* in built-in self-test to overcome these problems. The use of NCR significantly increases the efficiency of an irregular test set while guaranteeing complete fault efficiency. The next section presents an overview of the BIST scheme implementing nearly complete reseeding.

7.2 Overview of the Proposed Scheme

Figure 7.1 shows the realization of nearly complete reseeding in BIST. A complete test vector is shifted into the scan chains in parallel in STUMPS. During seed computation of nearly complete reseeding one or more bits of a test vector may raise conflicts and will need to be ignored. Addressing the ignored bits of a test vector individually may introduce significant overhead in the test time and the hardware. Hence, we address a complete vector and ignore it if any of its bits causes a conflict during seed computation. The seeds for a test set are computed by allowing to ignore a fixed number of vectors. The positions of ignored vectors are stored along with the seeds.

When decoding an ignored test vector, the LFSR will generate the desired values at the compatible bit positions, and the opposite values at the conflicting bit positions. In order to retain the values at compatible bit positions and to flip the opposite values, a *flip vector* is assigned to each ignored position. The seeds, the ignored positions and the flip vectors are stored in the memory. In order to reduce the memory requirements, ignored positions are encoded as the distance between two ignored positions, and flip vectors are compressed using a compression method similar to [Wurt 04]. A small decoder is then used to generate the flip vectors. An XOR operation is performed at all the ignored positions between LFSR generated vector and the associated flip vector in order to get the desired values.

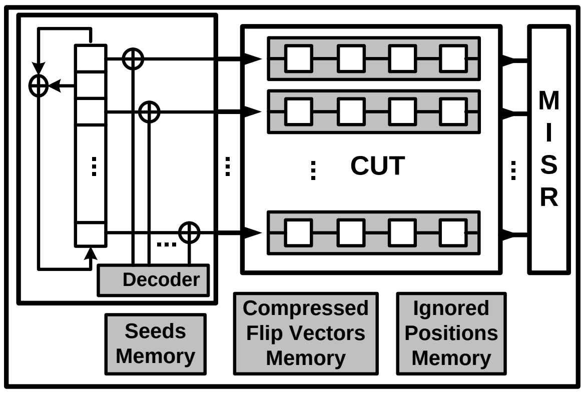


Figure 7.1: Overview of BIST employing NCR

Two main approaches to seed computation are popular in current literature. Either a complete pattern is encoded in a single seed [Koen 91, Hell 95, Rajs 98, Kris 01, Al Y 05], or continuous subsequences of a single or multiple patterns are encoded in each seed [Volk 03]. For the latter approach, it was shown that storage requirements are improved and shorter LFSRs can be employed [Volk 03]. Although nearly complete reseeding can be employed with any of the two reseeding techniques, the second approach has been selected here, as multiple patterns can be encoded before the number of ignored vectors exceeds a predefined limit. Furthermore, control information for the start of the next seed can be derived implicitly by starting the next seed just after the last ignored position associated with a seed. This control information had to be stored separately in [Volk 03].

In the next section we describe the seed computation algorithm for the proposed scheme.

7.3 Seeds Computation

Let P be a test set with $|P| = q$, then P could be viewed as a sequence of $t * q$ test vectors. A v at position j in P is represented as $P(j) = v_j$. If a care vector v_j is considered during seed computation it is called an *encoded vector* otherwise it is termed as *ignored vector* and its position j as *ignored position.* For a seed S, V_E and V_I represents the sets of encoded and ignored vectors respectively. For each $v_j \in V_I$ a *flip vector* f is assigned such that $v_j = r_j \oplus f$ where r_j is the random vector generated by LFSR at position j.

During seed computation, a complete test vector is ignored if any of its bits raises conflicts. A limit I is set that defines the number of vectors that are allowed to be ignored for each seed. Figure 7.2 shows the seeds computation algorithm of nearly complete reseeding. Seeds computation is done in an iterative manner. It starts with empty sets of encoded and ignored vectors. A *care vector* is added to the set of encoded vectors, the linear equation system is generated and it is established whether or not a solution exists. If it does, the process continues with the next care vector. Otherwise, the current vector is moved from the encoded set to the ignored set first. The process of adding a *care vector* into the encoded or ignored set continues until the number of ignored vectors exceeds the defined limit I.

```
1. Let $j = 0$
2. Let encoded vector set $V_E = \emptyset$
3. Let ignored vector set $V_I = \emptyset$
4. Until $v_j$ is don't care vector    $j++$
5. $V_E = V_E + v_j$
6. Find seed for $V_E$
7. Unless $V_E$ becomes unsolvable go to step 4
8. $V_E = V_E - v_j$
9. $V_I = V_I + v_j$
10. Unless $|V_I| \leq I$ go to step 4
11. Unless FIND-OPTIMIZED-IGNORE-POSITIONS fails go to step 4
12. Store seed, ignored positions and compute flip vectors go to step 2
```

Figure 7.2: Seeds computation algorithm for NCR

When the addition of a certain *care vector* makes the linear equation system unsolvable, most times, the current vector is not the only choice to be ignored in order to make the set solvable again. The ignored vector can be chosen from a number of possible candidates and choosing another vector instead of the most recent one, might result in fewer conflicts later on. But as the best choice of ignored positions might change with every new vector, the current vector is chosen as the victim until the allowed limit is crossed. If n is the total number of vectors considered so far, then as soon as the defined limit I is exceeded, the ignored set contains $I + 1$ vectors and the encoded set contains $n - I - 1$ vectors. At this point, a search is executed for a combination of I vectors such that ignoring these I instead of $I + 1$ vectors, causes the other $n - I$ vectors to become encodable.

If a subset of vectors is found for which the system of linear equations is consistent, the sets of encoded and ignored vectors are updated and the process continues. If no such subset exists, the current vector is used to delimit the start of the next seed, which means that $I + 1$ vectors are actually ignored for each seed instead of I vectors . The $I + 1^{th}$ ignored position is different from the first I ignored positions in that no new vector is encoded after it, but a new seed is started. Before starting the next iteration with empty sets of encoded and ignored vectors, the flip vectors are derived from the ignored vectors and are stored together with the seed and ignored positions. By simulating the unchanged pattern (i.e. without flip vectors applied), we can determine if this information is required for fault detection. The fault simulation is done only w.r.t the target faults of the pattern and does not add significant computational overhead.

Figure 7.3 demonstrates seeds computation for NCR with the help of an example: suppose 5 test patterns (7.3(a)) for the circuit in 4.4(a) have to be encoded and we are allowed to ignore 2 care vectors per seed. The seed computation procedure starts by adding the 1st vector of p_1 (right most vector) to the encoded set. The equations are generated and it is determined that this equation system has a solution. Then, the 2^{nd} vector is added and equations for both vectors in the set are generated to observe solvability. The encoded set remains solvable until the 4^{th} vector is added. As the limit is not yet exceeded this vector is moved to the set of ignored vectors. Then the 6^{th} and the 8^{th} vectors are found to cause the equations to be unsolvable, so they are moved to the ignored set, too. By ignoring the 8^{th} vector, the size of the ignored set becomes 3, which is greater than allowed. Now it is attempted to optimize the selection of ignored vectors so that instead of three vectors only two vectors have to be ignored and the rest of the vectors become solvable. For this, we generate equations and check solvability of all the possible 6 out of 8 vectors. We find that none of these subsets are solvable, so the seed of the encoded set $1, 2, 3, 5, 7$ is saved. In order to derive flip vectors for the ignored set, the location of the bits related to the conflicting equations are determined and the flip vector is assigned 0 for matching bits, 1 for conflicting bits, and X for the rest. The remaining seven vectors are encoded by the second seed using the same method. Figure 7.3(b) shows both seeds along with their ignored positions and associated flip vectors.

7.4 Compression/Decompression of Embedding Information

The embedding information contains flip positions and flip vectors associated with them. The flip positions are efficiently encoded by encoding the distances between two consecutive flip positions instead of their absolute numbers. These distances are used during the test to identify the ignored vectors and start of the next seed. For the flip vectors, a compression method similar to [Wurt 04] is used. The embedding information of NCR is different in statistical attributes compared to embedding information of

p_5			p_4			p_3			p_2			p_1		
v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1
X	X	1	X	X	1	1	X	X	0	X	X	X	X	0
1	1	X	0	0	1	1	0	1	1	X	0	X	X	X
X	X	0	X	0	X	X	0	0	X	1	X	0	X	1
1	0	X	1	0	X	X	1	1	1	X	0	X	X	X
X	X	1	0	1	X	0	X	X	X	X	0	X	X	X
X	X	0	0	X	X	0	X	X	0	1	X	X	1	0
0	0	X	X	X	X	X	X	X	X	X	1	X	1	X
X	X	0	1	X	1	0	X	1	1	1	X	1	0	X

(a) Set of 5 patterns

s_2	f_2			s_1	f_1		
	7	5	2		8	6	4
1	X	1	0	0	X	0	X
1	1	X	0	0	0	0	0
1	X	1	X	1	1	X	X
0	1	X	X	1	1	1	1
0	X	1	X	0	X	X	1
0	X	0	X	0	X	0	X
1	0	X	X	1	X	X	0
0	X	0	1	1	X	1	X

(b) Computed seeds and flip vectors

Figure 7.3: Example of seed computation with NCR

DLBIST. In DLBIST a deterministic pattern is mapped to a random pattern after searching a random pattern that matches it in majority of care bits. So the embedding information of DLBIST is biased towards 0. However in NCR an ignored deterministic test vector has to be mapped to a random vector generated at ignored cycle. The number of matching and conflicting bits between two vectors is random. So the embedding information of NCR is not biased.

The number of flip vectors is very small compared to the total number of test vectors. The flip vectors show strong regularity and most of the flip vectors are compatible with each other or differ at very few bit positions. This regularity is exploited to develop a compression method and decompression architecture that offers reduction in storage of embedding information without prolonging test time.

7.4.1 Compression Algorithm

The main idea in the compression of flip vectors is that a reference vector is computed with which the flip vectors have a minimum number of conflicts. In most cases, a large number of the flip vectors are fully compatible with the reference vector, and the ignored position and the flip vector is encoded as a tuple $(i, 0)$ where i is the address of the ignored vector relative to the previous ignored. Otherwise, the encoding is $(i, 1, (p_0, 0) \dots (p_{n-1}, 0), (p_k, 1))$, where $p_0 ... p_k$ are the positions in which the flip-vector differs from the reference vector.

The reference vector contains a 1 at a certain bit position, if the flip vectors have more 1s than 0s, otherwise it contains a 0. Figure 7.4 shows the compression of the flip vectors computed in the previous example. Each flip vector is a column of a matrix, and we determine which of 0 or 1 has the highest frequency in each row. This value is entered in the reference vector RV. In the example, we find that the first row has more 0s than 1s, so the first bit of the reference vector RV is assigned a 0. The complete reference vector is computed this way and is shown in the column next to the flip vectors. In our experiments a single reference vector was sufficient for all the flip vectors. But if the number of flip vectors is too large, the flip vectors can be divided into the groups of consecutive test vectors and a reference vector is computed for each subsequence.

Figure 7.4 also illustrates the encoding of the ignored positions and the flip vectors. A comparison of the flip vectors with the reference vector reveals that the first 4 flip vectors are fully compatible, while the last two vectors have conflicts. So, the first 4 flip vectors are encoded by storing a single status bit 0 with the ignored positions. For the second last vector, the bit number 0 and the bit number 7 have conflicts with the reference vector. This vector is encoded as $(3, 1, (0, 0), (7, 1))$ which indicates that the reference vector can be used as the flip vector after modifying the bits 0 and 7. The status bit 1 in $(7, 1)$ means that the reference vector is ready to be used after this modification.

Using the proposed technique the test set of figure 7.3(a), containing 5 patterns and 53 care bits, can be encoded with a total of 54 bits where 16 bits are required to store the seeds, 12 bits to store the ignored positions and 26 bits to store the flip vectors. The same patterns would have required 210 bits and 90 bits in case of conventional [Koen 91] and variable pattern length [Volk 03] reseeding techniques respectively.

f_2			f_1			RV
7	5	2	8	6	4	
X	1	0	X	0	X	0
1	X	0	0	0	0	0
X	1	X	1	X	X	1
1	X	X	1	1	1	1
X	1	X	X	X	1	1
X	0	X	X	0	X	0
0	X	X	X	X	0	0
X	0	1	X	1	X	1

$$C_1 = \{(4,0),(2,0),(2,0)\}$$
$$C_2 = \{(2,0),(3,1,(0,0),(7,1)),(2,1,(1,1))\}$$

Figure 7.4: Flip vectors compression example

7.4.2 Decompression Architecture

To decode the encoded flip vectors, we need the reference vector, the information whether the flip vector of an ignored position is compatible with the reference vector, and the conflicting positions. Figure 7.5 shows the architecture of the flip vector generator. It contains a register to store the reference vector (RV Register), a modify unit to generate incompatible flip vectors, and a multiplexer to select between the reference and the incompatible flip vector depending on the status bit of the ignored position.

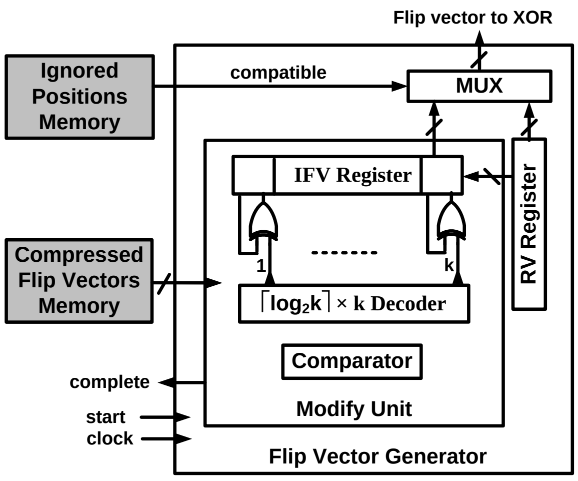


Figure 7.5: Decompression Hardware of NCR

The modify unit is similar to the generate unit of "run-length encoding with parallel decompression". It creates flip vectors serially during shifting by using the reference vector in the RV Register and the encoded conflict positions in the compressed flip vectors memory. On receiving the *start* signal, the incompatible flip vector register (IFV Register) is reset to the reference vector and the conflicting positions are read from the memory and flipped in the IFV Register until the comparator detects that the status bit of a conflicting position is '1'. The *complete* signal is set to high at this point, indicating that the flip vector is ready to be used. The number of clocks needed to produce a complete incompatible vector is equal to the number of conflicts between an incompatible flip vector and the reference vector.

Here it should be noted that the modify unit is completely independent and it does not need to wait until the ignored position of an incompatible flip vector is met. It produces the next incompatible flip vector in advance and waits for its use. As ignored positions are randomly distributed and very few flip vectors require modification, the decompression does not cause any noticeable speed degradation in general. Some cycles could be wasted only in the case if there are two consecutive flip positions and both of them have incompatible flip vectors, an extremely rare case. Moreover, the flip vector generator only depends on the number of scan chains in the circuit and is independent of the test set or size of the circuit.

7.5 Experimental Evaluation

In order to validate the efficiency of the presented algorithm and test architecture, experiments were performed on industrial circuits. The experimental setup and the results are explained in appendix A.3. Evaluating the encoding efficiency as a function of the number of ignored vectors per seed reveals that coding efficiency increases linearly with the number of ignored vectors. The gain achieved by increasing the number of ignored vectors is higher for first few ignores. It is because the linear dependence is low in the beginning and by ignoring few inconsistent equations many subsequent equations can be encoded with the same seed. While the achieved gain per ignored vector decreases if more and more vectors per seed are ignored, the computation time increases significantly. It was found that for majority of the circuits 2 and 3 are good values to achieve significantly higher encoding efficiency within a reasonable time.

The results are compared to the variable pattern length per seed technique known as continuous reseeding [Volk 03], which is the most efficient single polynomial technique published so far and is equivalent to evaluating the presented algorithm using 0 allowed ignore per seed. The comparison shows that the proposed scheme offers an increase in encoding efficiency ranging from 35 to 118 percent. For majority of the experimented circuits the gain remains higher than 50 percent offering significant reduction in storage requirements.

The major draw back of reseeding is that encoding of a pattern is not guaranteed. The size of LFSR have to be kept larger than the maximum number of care bits in a pattern or vector in order to ensure the high probability of encoding. In nearly complete reseeding, any standard size LFSR could be employed. If some vectors are left unencoded by LFSR, they are encoded as flip vectors. In our experiments, a standard 64 bit LFSR was used for NCR while in case of continuous reseeding larger LFSR had to be employed for some circuits.

In nearly complete reseeding a small decoder is required to regenerate compressed flip vectors. Experimental results reveal that the hardware overhead associated with this decoder is negligible compared to the size of the circuit under test. In addition this decoder only depends on the number of scan chains in the circuit and is independent of the test set or size of the circuit.

CHAPTER 8

Conclusions

8.1 Summary

In this work novel test solutions offering optimized reductions in test cost have been presented. Explosive increase in test sets volume is causing a rapid growth in test cost and Test data compression techniques are used to curtail it. Since the compression of a test set depends on its statistical attributes, our work started with the analysis of various test sets generated for the large industrial circuits. Using the findings of the analysis the test sets are categorized into three classes based on their statistical attributes namely strongly regular, weakly regular and irregular test sets. Optimized compression is obtained by examining each class differently and hence proposing three novel compression methods and decompression architectures. The decompression hardware of each proposed method is independent of the encoded test set. Three low cost programmable deterministic self-test schemes offering different trade-offs are developed by employing each method.

A large number of repeating sequences of bits are found in the test sets with low care bits percentages. We have classified such test sets as strongly regular test sets. The strong regularity in such test sets is exploited by restricting the successive repeating sequences to a single precomputed value called *restrict value.* The irregular portions of the test set are encoded with the seeds. This compression method is named *restrict encoding* (RE). Minimum number of restrict values are found using a clique partitioning heuristic while length of successive sequences is maximized by solving traveling salesman problem. A built-in self-test scheme achieving high quality with very less storage is developed using *restrict encoding.* The experimental results showed that *restrict encoding* always offers an encoding efficiency greater than 1.

The test sets with high percentages of care bits but biased toward *0* or *1* are categorized as weakly regular test sets. A variable-to-variable run-length encoding method is proposed for such test sets that achieves a compression ratio close to the entropy limit. A major drawback of conventional run-length decoders is the large decompression time overhead due to their serial regeneration of run-length symbols. This problem is overcome by presenting a novel run-length decoder that is able to regenerate a complete run-length symbol in a single clock. This encoding method is named *run-length encoding with parallel decompression* (REWPD). A built-out self-test scheme minimizing the on-chip hardware overhead is developed by employing REWPD.

The repeating sequences of care bits are rare in the unbiased test sets with high percentages of care bits. Such test sets are classified as irregular test sets. The probabilistic analysis as well as various experiments showed that a significant number of care bits could be encoded in the seed of a linear decompressor if a small number of bits causing inconsistency are ignored during seed computation. This finding is used in the proposed compression method for irregular test sets called *nearly complete reseeding* (NCR). In NCR the ignored bits are separately encoded and stored in memory. During the test the ignored bits are embedded into the generated patterns. The built-in self-test scheme implemented by using NCR tries to achieve short test time with minimal storage requirements.

8.2 Future Work

In our work we have assumed a core based design style where structural information of the circuits is not available. For the test environments where structural information is known, the *restrict encoding* and the *run-length encoding with parallel decompression* could be extended by embedding the test compression algorithms into the ATPG and guiding the pattern generation process such that the encoding efficiency is maximized.

One interesting issue that is not covered in the scope of this thesis is the evaluation of *restrict encoding* for low power test. In restrict encoding, a vast majority of the test vectors are generated during test using very few *restrict vectors* and a single *restrict vector* is injected continuously for long runs of consecutive test vectors. This results in significant reductions in switching activity. A thorough study should be performed to quantify the power savings achieved with *restrict encoding*.

Bibliography

[Abra 94] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital systems testing and testable design.* IEEE Press, Piscataway, NJ, 1994.

[Al Y 05] A. Al-Yamani, S. Mitra, and E. McCluskey. "Optimized reseeding by seed ordering and encoding". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 2, pp. 264–270, Feb. 2005.

[Bala 07] K. J. Balakrishnan and N. A. Touba. "Relationship between entropy and test data compression". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 26, No. 2, pp. 386–395, Feb. 2007.

[Bard 82] P. H. Bardell and W. H. McAnney. "Self-testing of multichip logic modules". In: *Proceedings of International Test Conference (ITC)*, pp. 200–204, 1982.

[Bard 87] P. H. Bardell, W. H. McAnney, and J. Savir. *Built-in test for VLSI: pseudorandom techniques.* John Wiley & Sons, 1987.

[Bayr 01] I. Bayraktaroglu and A. Orailoglu. "Test volume and application time reduction through scan chain concealment". In: *Proceedings of Design Automation Conference (DAC)*, pp. 151–155, 2001.

[Bers 93] M. Bershteyn. "Calculation of multiple sets of weights for weighted random testing". In: *Proceedings of International Test Conference (ITC)*, pp. 1031–1040, 1993.

[Bush 00] M. L. Bushnell and V. D. Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits.* Springer Science, New York, 2000.

[Chan 01] 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*, Vol. 20, No. 3, pp. 355–368, 2001.

[Chan 03] A. Chandra and K. Chakrabarty. “Test data compression and test resource partitioning for systom-on-a-chip using frequency-directed run-length (FDR) codes”. *IEEE Transactions on Computers*, Vol. 52, No. 8, pp. 1076–1087, 2003.

[Chan 98] J. T.-Y. Chang, C.-W. Tseng, C.-M. J. Li, M. Purtell, and E. J. McCluskey. “Analysis of pattern-dependent and timing-dependent failures in an experimental test chip”. In: *Proceedings of International Test Conferece (ITC)*, pp. 184–193, 1998.

[Chin 84] C. K. Chin and E. J. McCluskey. “Weighted pattern generation for built-in self-test”. In: *Technical report (CRC TR) No. 84-7, Center for Reliable Computing, Stanford University*, p. , August 1984.

[Daeh 81] W. Daehn and J. Mucha. “Hardware test pattern generation for built-in testing”. In: *Proceedings of International Test Conference (ITC)*, pp. 110–120, 1981.

[Dand 84] R. Dandapani, J. H. Patel, and J. A. Abraham. “Design of test pattern generators for built-in test”. In: *Proceedings of International Test Conference (ITC)*, pp. 315–319, 1984.

[Eich 77] E. B. Eichelberger and T. W. Williams. “A logic design structure for LSI testability”. In: *Proceedings of Design Automation Conference (DAC)*, pp. 462–468, 1977.

[Eich 83] E. B. Eichelberger and E. Lindbloom. “Random-pattern coverage enhancement and diagnosis for LSSD logic self-test”. *IBM Journal of Research and Development*, Vol. 27, No. 3, pp. 265–272, 1983.

[El M 08] A. El-Maleh. “Test data compression for system-on-a-chip using extended frequency-directed run-length code”. *IET Computers and Digital Techniques*, Vol. 2, No. 3, pp. 155–163, May 2008.

[Fuji 83] H. Fujiwara and T. Shimono. “On the acceleration of test generation algorithms”. *IEEE Transactions on Computers*, Vol. 32, pp. 1137–1144, 1983.

[Galk 06] C. Galke, U. Gätzschmann, and H. T. Vierhaus. “Scan-based SOC test using space/time pattern compaction schemes”. In: *Ninth Euromicro Conference on Digital System Design (DSD)*, pp. 433–438, 2006.

[Gher 04] V. Gherman, H.-J. Wunderlich, H. P. E. Vranken, F. Hapke, M. Wittke, and M. Garbers. “Efficient pattern mapping for deterministic logic BIST”. In: *Proceedings of International Test Conference (ITC)*, pp. 48–56, 2004.

[Glas 95] U. Gläser and H. T. Vierhaus. “FOGBUSTER: an efficient algorithm for sequential test generation”. In: *European Design Automation Conference (EURO-DAC)*, pp. 230–235, 1995.

[Goel 81] P. Goel. “An implicit enumeration algorithm to generate tests for combinational logic circuits”. *IEEE Transactions on Computers*, Vol. 30, No. 3, pp. 215–222, 1981.

[Golo 66] S. W. Golomb. “Run-length encoding”. *IEEE Transactions on Information Theory*, Vol. IT-12, pp. 399–401, December 1966.

[Gonc 02] 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: *Proceedings of Design, Automation and Test in Europe (DATE)*, pp. 604–611, 2002.

[Grim 99] M. R. Grimaila, S. Lee, J. Dworak, K. M. Butler, B. Stewartn, H. Balachandran, B. Houchins, V. Mathur, J. Park, and M. R. Mercer. “REDO - Random Excitation and Deterministic Observation - First commercial experiment”. In: *Proceedings of VLSI Test Symposium (VTS)*, pp. 268–294, 1999.

[Hakm 05] A. W. Hakmi, H.-J. Wunderlich, V. Gherman, M. Garbers, and J. Schlöffel. “Implementing a scheme for external deterministic self-test”. In: *Proceedings of VLSI Test Symposium (VTS)*, pp. 101–106, 2005.

[Hakm 07] A. W. Hakmi, H.-J. Wunderlich, C. G. Zoellin, A. Glowatz, , J. Schlöffel, F. Hapke, and L. Souef. “Programmable deterministic built-in self-test”. In: *Proceedings of International Test Conference (ITC)*, p. Paper 18.1, 2007.

[Hakm 09] A. W. Hakmi, S. Holst, H.-J. Wunderlich, J. Schlöffel, F. Hapke, and A. Glowatz. “Restrict encoding for mixed-mode BIST”. In: *Proceedings of VLSI Test Symposium (VTS)*, pp. 179–184, 2009.

[Hamz 00] I. Hamzaoglu and J. H. Patel. “Test set compaction algorithms for combinational circuits”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 19, No. 8, pp. 957–963, Aug 2000.

[Hell 90] S. Hellebrand, H.-J. Wunderlich, and O. F. Haberl. “Generating pseudo-exhaustive vectors for external testing”. In: *Proceedings of International Test Conference (ITC)*, pp. 670–679, 1990.

[Hell 95] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois. “Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers”. *IEEE Transactions on Computers*, Vol. 44, No. 2, pp. 223–233, 1995.

[Huff 52] D. A. Huffman. “A method for the construction of minimum redundancy codes”. In: *Proceedings of IRE*, 1952.

[ITRS 07] ITRS. “Test and test equipment”. In: *International technology roadmap for semiconductors*, pp. 1098–1101, 2007.

[Jas 03] A. Jas, J. Ghosh-Dastidar, M. 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*, Vol. 22, No. 6, pp. 797–806, 2003.

[Jas 98] A. Jas and N. Touba. “Test vector decompression via cyclical scan chains and its application to testing core-based designs”. In: *Proceedings of International Test Conference (ITC)*, pp. 458–464, 1998.

[Jha 03] N. Jha and S. Gupta. *Testing of digital systems.* Cambridge University Press London, 2003.

[Kaji 94] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. Reddy. “On compacting test sets by addition and removal of test vectors”. In: *Proceedings of VLSI Test Symposium (VTS)*, pp. 202–207, 1994.

[Kapu 94] R. Kapur, S. Patil, T. J. Snethen, and T. W. Williams. “Design of an efficient weighted random pattern generation system”. In: *Proceedings of International Test Conference (ITC)*, pp. 491–500, 1994.

[Kavo 07] X. Kavousianos, E. Kalligeros, and D. Nikolos. “Optimal selective Huffman coding for test-data compression”. *IEEE Transactions on Computers*, Vol. 56, No. 8, pp. 1146–1152, Aug. 2007.

[Kavo 08] X. Kavousianos, E. Kalligeros, and D. Nikolos. “Multilevel Huffman test data compression for IP cores with multiple scan chains”. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 16, No. 7, pp. 926–931, July 2008.

[Kim 08] T. Kim, S. Chun, Y. Kim, M.-H. Yang, and S. Kang. "An effective hybrid test data compression method using scan chain compaction and dictionary-based scheme". In: *Proceedings of Asian Test Symposium (ATS)*, pp. 151–156, 2008.

[Koen 91] B. Koenemann. "LFSR-coded test patterns for scan designs". In: *Proceedings of European Test Conference (ETC)*, pp. 237–242, 1991.

[Kris 01] C. V. Krishna, A. Jas, and N. A. Touba. "Test vector encoding using partial LFSR reseeding". In: *Proceedings of International Test Conference (ITC)*, pp. 885–893, 2001.

[Kris 03] C. Krishna and N. Touba. "Adjustable width linear combinational scan vector decompression". In: *Proceedings of International Conference on Computer Aided Design (ICCAD)*, pp. 863–866, 2003.

[Lai 05] L. Lai, J. H. Patel, T. Rinderknecht, and W. T. Cheng. "Hardware efficient LBIST with complementary weights". In: *Proceedings of International Conference on Computer Design (ICCD)*, pp. 479–481, 2005.

[Li 03] L. Li, K. Chakrabarty, and N. A. Touba. "Test data compression using dictionaries with selective entries and fixed-length indices". *ACM Transactions on Design Automation of Electronic Systems*, Vol. 8, No. 4, pp. 470–490, 2003.

[Lin 01] X. Lin, J. Rajski, I. Pomeranz, and S. Reddy. "On static test compaction and test pattern ordering for scan designs". In: *Proceedings of International Test Conference (ITC)*, pp. 1088–1097, 2001.

[Ma 95] S. C. Ma, P. Franco, and E. J. McCluskey. "An experimental chip to evaluate test techniques experiment results". In: *Proceedings of International Test Conferece (ITC)*, pp. 663–672, 1995.

[McCl 81] E. McCluskey and S. Bozorgui-Nesbat. "Design for autonomous test". *IEEE Transactions on Computers*, Vol. 30, No. 11, pp. 866–875, 1981.

[McCl 86] E. J. McCluskey. *Logic design principles: with emphasis on testable semiconductor circuits.* Prentice Hall, Englewood Cliffs, NJ, 1986.

[Mrug 04] 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*, Vol. 23, No. 9, pp. 1306–1320, Sept. 2004.

[Pome 91] I. Pomeranz, L. N. Reddy, and S. M. Reddy. "COMPACTEST: A method to generate compact test sets for combinatorial circuits.". In: *Proceedings of International Test Conference (ITC)*, pp. 194–203, 1991.

[Pome 99] I. Pomeranz and S. M. Reddy. "On n-Detection test sets and variable n -Detection test sets for transition faults". In: *Proceedings of VLSI Test Symposium (VTS)*, pp. 173–179, 1999.

[Rajs 98] J. Rajski, J. Tyszer, and N. Zacharia. "Test data decompression for multiple scan designs with boundary scan". *IEEE Transactions on Computers*, Vol. 47, No. 11, pp. 1188–1200, 1998.

[Rear 05] J. Rearick, B. Eklow, K. Posse, A. Crouch, and B. Bennetts. "IJTAG (internal JTAG): A step toward a DFT standard". In: *Proceedings of Internatioanl Test Conference (ITC)*, p. Paper 32.3, 2005.

[Redd 02] S. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz. "On test data volume reduction for multiple scan chain designs". In: *Proceedings of VLSI Test Symposium (VTS)*, pp. 103–108, 2002.

[Redd 97] S. Reddy, I. Pomeranz, and S. Kajihara. "Compact test sets for high defect coverage". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16, No. 8, pp. 923–930, Aug 1997.

[Roth 66] J. P. Roth. "Diagnosis of automata failures: A calculus and a method". *IBM Journal of Research and Development*, Vol. 10, No. 4, pp. 278–291, 1966.

[Ruan 07] X. Ruan and R. Katti. "Data-independent pattern run-length compression for testing embedded cores in SoCs". *IEEE Transactions on Computers*, Vol. 56, No. 4, pp. 545–556, April 2007.

[Schn 75] H. D. Schnurmann, E. Lindbloom, and R. G. Carpenter. "The weighted random test-pattern generator". *IEEE Transactions on Computers*, Vol. 24, No. 7, pp. 695–700, 1975.

[Sina 08] O. Sinanoglu. "Scan architecture with align-encode". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 27, No. 12, pp. 2303–2316, Dec. 2008.

[Stro 91] A. Ströle and H.-J. Wunderlich. "TESTCHIP: A chip for weighted random pattern generation, evaluation and test control". *IEEE Journal of Solid State Circuits*, Vol. 26, No. 7, pp. 1056–1063, 1991.

[Tehr 05] M. Tehranipoor, M. Nourani, and K. Chakrabarty. "Nine-coded compression technique for testing embedded cores in SoCs". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 13, No. 6, pp. 719–731, 2005.

[Tene 08] V. Tenentes, X. Kavousianos, and E. Kalligeros. "State skip LFSRs: bridging the gap between test data compression and test set embedding for IP cores". In: *Proceedings of Design, Automation and Test in Europe (DATE)*, pp. 474–479, 2008.

[Toub 96] N. A. Touba and E. J. McCluskey. "Altering a pseudo-random bit sequence for scan-based BIST". In: *Proceedings of International Test Conference (ITC)*, pp. 167–175, 1996.

[Volk 02] E. H. Volkerink, A. Khoche, and S. Mitra. "Packet-based input test data compression techniques". In: *Proceedings of International Test Conference (ITC)*, pp. 154–163, 2002.

[Volk 03] E. H. Volkerink and S. Mitra. "Efficient seed utilization for reseeding based compression". In: *Proceedings of VLSI Test Symposium (VTS)*, pp. 232–240, 2003.

[Waic 89] J. A. Waicukauski, E. Lindbloom, E. B. Eichelberger, and O. P. Forenza. "WRP: A method for generating weighted random test patterns". *IBM Journal of Research and Development*, Vol. 33, No. 2, pp. 149–161, 1989.

[Wang 06] L.-T. Wang, C.-W. Wu, and X. Wen. *VLSI test principles and architectures: design for testability.* Morgan Kaufmann Publishers, 2006.

[Wang 08a] L.-T. Wang, C. E. Stroud, and N. A. Touba. *System-on-chip test architectures: nanometer design for testability.* Morgan Kaufmann Publishers, 2008.

[Wang 08b] Z. Wang and K. Chakrabarty. "Test data compression using selective encoding of scan slices". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 16, No. 11, pp. 1429–1440, Nov. 2008.

[Wang 86] L.-T. Wang and E. McCluskey. "Complete feedback shift register design for built-in self-test". In: *Proceedings of International Conference on Computer Aided Design (ICCAD)*, pp. 56–59, 1986.

[Wund 87] H.-J. Wunderlich. "Self test using unequiprobable random patterns". In: *Digest of Papers, Fault-Tolerant Computing Symp.*, pp. 258–263, 1987.

[Wund 90] H.-J. Wunderlich. “Multiple distributions for biased random test patterns”. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 9, No. 6, pp. 584–593, 1990.

[Wund 96] H.-J. Wunderlich and G. Kiefer. “Bit-flipping BIST”. In: *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 337–343, 1996.

[Wurt 04] A. Wurtenberger, C. S. Tautermann, and S. Hellebrand. “Data compression for multiple scan chains using dictionaries with corrections”. In: *Proceedings of International Test Conference (ITC)*, pp. 926–935, 2004.

Index

APPENDIX A

Tables with Experimental Results

The experiments were performed on large industrial circuits. Same set of circuits was used to evaluate the efficiency of the three developed schemes. A commercial ATPG tool was used to generate required test sets. The algorithms were implemented in JAVA while the hardware was modeled in VHDL. The efficiency of each scheme was compared with the most efficient similar methods. The storage requirements are given in terms of bits assuming that the compressed test sets are stored in a bit addressable memory. The hardware size is presented in two input NAND equivalent, estimated by using a commercial tool and a free library.

Table A.1 presents some of the properties of the employed circuits and the generated test sets. In the first column name of the circuit corresponds to the number of nets in the circuit. Second column with header k reports the number of scan chains in circuits. The original scan chain configurations were used. Next column named t tells the maximum length of scan chains. In the subsequent three columns the test set size and care bits density are given for the test sets encoded with restrict encoding (RE), run-length encoding with parallel decompression (REWPD) and nearly complete reseeding (NCR) respectively. Here the first sub-column labeled $|P|$ reports the number of patterns in each test set while the sub-column named $\%cb$ tells the care bits density. The test sets for RE and REWPD were generated for RP-resistant faults after applying 10,000 random patterns while the test set for NCR was generated for all faults. Pattern merging efforts were set to low for RE and high for REWPD and NCR.

Circuit	k	t	RE		REWPD		NCR	
			$\|P\|$	$\%cb$	$\|P\|$	$\%cb$	$\|P\|$	$\%cb$
p35k	23	127	8506	4.88	1291	21.75	1681	21.5
p45k	97	333	3406	0.06	2042	0.10	2102	0.14
p77k	13	305	5883	0.85	294	7.46	490	7.54
p81k	8	504	30706	1.23	204	30.15	273	31.81
p89k	18	963	13514	0.23	807	2.08	1220	2.25
p100k	18	792	3437	0.14	228	1.27	2048	0.55
p141k	24	486	10984	0.37	496	4.55	675	7.43
p239k	40	541	6034	0.12	321	1.50	508	3.85
p259k	40	541	8482	0.11	439	1.43	637	4.37
p267k	45	494	14123	0.18	639	2.23	963	2.65
p269k	45	494	14615	0.18	588	2.37	963	2.63
p279k	55	416	17575	0.13	567	2.33	739	3.83
p286k	55	416	25645	0.13	862	2.08	1082	3.77
p330k	64	317	17322	0.24	1705	1.32	1765	2.13
p388k	50	546	14109	0.11	298	2.96	462	6.67
p418k	64	831	33704	0.05	718	0.97	798	1.92
p483k	71	900	22168	0.03	180	1.41	264	4.55

Table A.1: Statistics of the experimented circuits and the test sets

A.1 Restrict Encoding

Table A.2 shows the results of restrict encoding. The first part of the table deals with the restricts only. The first sub-column labeled *%cb* shows the percentage of care bits in each test set encoded with the restrict vectors. Then, the total number of restricts are given in sub-column *#rest* followed by the total number of commands in sub-column *#com*. Sub-column *rcost* shows the amount of storage spent on these commands while sub-column *dcost* represents the size of the RV ROM in bits. The next sub-column *scost* reports the size of status register in bits. The second last sub-column of the first part named *tcost* tells the total cost of storing the restrict information while the encoding efficiency for the care bits covered by restricts is given in the final sub-column *EE*. On average 80% of the care bits in a test set are encoded with the restricts. Encoding efficiencies ranging from 1.8 to 3.9 are achieved for the care bits encoded with the restricts. These efficiencies are much higher than the maximum efficiency achievable by a linear decompressor.

The second part of the table shows the results for encoding the remaining care bits with reseeding. For reseeding, a 128 bit LFSR was used in combination with a randomly generated phase shifter having a single XOR gate for every scan chain. Here sub-column

%cb reports the percentages of care bits encoded with the seeds. Sub-column *rscost* shows the amount of seed information stored to encode the unrestricted care bits and sub-column *EE* gives the encoding efficiency for the care bits encoded with reseeding. The *rscost* is comprised of the seeds bits and the additional delay information in each seeding command. As expected, encoding efficiency for reseeding alone is below 1. But on average only 20% of the care bits in test sets need to be encoded in this way. The overall encoding efficiency of restrict encoding remains greater than 1 in all cases as shown in last part of the table.

Table A.3 compares the performance of restrict encoding with continuous reseeding [Volk 03] which is the most efficient LFSR reseeding scheme published so far and among available compression schemes offers the best compression for a test set with low care bits density. Similar 128-bit LFSR was used in both methods. The results of continuous reseeding are presented in the second column while in the third column the results of proposed scheme are shown. The sub-columns *Storage* tell the total storage requirements in terms of bits and the sub-columns *EE* report the encoding efficiencies. The last column shows the improvement in encoding efficiency achieved using restrict encoding. The encoding efficiency of restrict encoding is always higher than the maximum theoretical limit of reseeding. Compared to the experimented encoding efficiency of the most efficient reseeding method, an improvement ranging from 62 to 186 percent is achieved. This results in reducing the storage requirements up to one third. Considering the very large test set sizes of nanometer ICs, this reduction in storage offers great savings in test cost.

In table A.4 the results of proposed method are compared with the available dictionary encoding schemes [Redd 02, Li 03, Wurt 04, Kim 08]. The best case storage for all available schemes is similar. In both columns *Dictionary Encoding* and *Restrict Encoding*, sub-column *Storage* reports the total storage while sub-column *CR* tells the achieved compression ratio. The *CR* is computed using the equation 2.4 at page 14. As can be seen from the table, the compression ratio of restrict encoding is much higher compared to dictionary encoding. This difference is due to the fact that in dictionary encoding an index of the dictionary has to be stored for each test vector while in the proposed scheme a single index of the RV ROM is shared by large number of test vectors.

	Restricts								Reseeding			Total	
Circuit	%cb	#rest	#com	rcost	dcost	scost	tcost	EE	%cb	rscost	EE	Storage	EE
p35k	90.1	22561	30092	616806	12075	1397	628881	1.73	9.9	226446	0.53	856724	1.41
p45k	73.8	850	1264	23624	1164	1665	24788	2.27	26.2	23472	0.86	49925	1.54
p77k	83.5	2307	3568	46169	91	1220	46260	3.6	16.5	41700	0.79	89180	2.23
p81k	93.1	30828	49489	636863	64	2016	636927	2.24	6.9	117594	0.90	756537	2.02
p89k	79.8	6913	11277	223725	108	3852	223833	1.92	20.2	130232	0.84	357917	1.51
p100k	69.5	939	1358	19113	108	3168	19221	2.62	30.5	24220	0.91	46609	1.55
p141k	84.3	6906	10826	146710	312	2430	147022	2.76	15.7	84651	0.90	234103	2.06
p267k	86.1	9281	14469	283097	405	2470	283502	1.8	13.9	95265	0.87	381237	1.55
p269k	86.4	10433	15078	197157	360	1976	197517	2.63	13.6	91740	0.90	291233	2.06
p279k	77.0	8058	13047	240984	825	2080	241809	1.78	23.0	154656	0.83	398545	1.4
p286k	78.4	10248	16432	231992	990	2496	232982	2.67	21.6	197797	0.86	433275	1.83
p330k	85.2	15921	27080	469884	704	1585	470588	1.56	14.8	159874	0.80	632047	1.36
p388k	73.5	5467	9057	130552	450	2730	131002	2.54	26.5	136360	0.88	270092	1.68
p418k	84.6	11448	17478	348948	512	3324	349460	2.46	15.4	180164	0.86	532948	1.9
p483k	75.5	4027	6069	84909	426	3600	85335	3.9	24.5	118860	0.91	207795	2.12

Table A.2: Restrict encoding results

	Continuous Reseeding		Restrict Encoding		Improvement
Circuit	Storage	EE	Storage	EE	(%)
p35k	2411094	0.5	856724	1.41	182
p45k	85960	0.89	49925	1.54	73
p77k	254380	0.78	89180	2.23	186
p81k	2138360	0.71	756537	2.02	185
p89k	611452	0.88	357917	1.51	72
p100k	79947	0.9	46609	1.55	72
p141k	667635	0.72	234103	2.06	186
p267k	777615	0.76	381237	1.55	104
p269k	786216	0.76	291233	2.06	172
p279k	655900	0.85	398545	1.4	65
p286k	918400	0.86	433275	1.83	113
p330k	1016400	0.84	632047	1.36	62
p388k	530442	0.85	270092	1.68	98
p418k	1194930	0.85	532948	1.9	124
p483k	491746	0.89	207795	2.12	138

Table A.3: Comparison of RE with continuous reseeding

A.2 Run-length Encoding with Parallel Decompression

In bit-flipping DLBIST [Wund 96, Gher 04] the deterministic patterns to detect RP-resistant faults are embedded into long sequence of pseudo random patterns by applying the embedding patterns. In [Gher 04] it is suggested that 10,000 is a good number to detect majority of the RP-testable faults in reasonable time. Consequently the length of pseudo random phase was set to 10,000 patterns. For RP-resistant faults a fully compacted test set was generated with the help of a commercial tool. Then every pattern

	Dictionary Encoding		Restrict Encoding	
Circuit	Storage	CR (times)	Storage	CR (times)
p35k	10802620	2	856724	29
p45k	4536792	24	49925	2203
p77k	5382945	4	89180	261
p81k	46427472	2	756537	163
p89k	39041946	6	357917	654
p100k	8166312	6	46609	1051
p141k	21352896	6	234103	547
p267k	27907048	11	381237	823
p269k	21659430	15	291233	1115
p279k	29244800	13	398545	1008
p286k	53341600	11	433275	1354
p330k	21964296	16	632047	556
p388k	30814056	12	270092	1426
p418k	84024072	21	532948	3363
p483k	59853600	23	207795	6816

Table A.4: Comparison of RE with available dictionary encoding schemes

in the generated test set was mapped to one of the useless random patterns such that total number of conflicts were minimized. The corresponding embedding information was stored.

Table A.5 compares the compression of the proposed algorithm with the maximum theoretical limit and state of the art run-length encoding scheme. The column with header CP_{max} tells the maximum compression that can be achieved for the embedding information of each circuit. This theoretical limit is computed using the equation 2.6 at page 15. Next column labeled *EFDER* reports the compression achieved using extended frequency directed run-length coding (EFDR) [El M 08]. In the last column compression of the proposed algorithm is shown. For all circuits compression of the proposed algorithm is significantly higher compared to state of the art run-length encoding scheme while it is very close to theoretical limit for majority of the circuits.

Table A.6 shows the gain of the proposed decompression architecture in test application time. It presents the number of clocks in millions that are required by different techniques to complete a test containing 10,000 patterns. It is assumed that on-chip and external chip frequencies are identical. The column named DLBIST represents the number of clocks required in the DLBIST approach [Wund 96, Gher 04]. The column with header *Conventional* shows the number of clocks if the embedding information is decompressed by using available run length compression/decompression methods [Chan 01, Gonc 02, Chan 03, El M 08]. Only the ideal case for these schemes is consid-

Circuit	CP_{max} (%)	EFDR (%)	Proposed (%)
p35k	89.28	72.48	84.19
p45k	99.88	92.74	98.82
p77k	98.54	61.10	91.49
p81k	96.68	49.06	85.90
p89k	98.78	71.88	93.93
p100k	99.75	72.17	94.30
p141k	98.34	78.69	94.24
p239k	99.59	84.87	97.10
p259k	99.47	84.84	96.97
p267k	98.96	86.29	96.45
p269k	98.99	86.30	96.50
p279k	99.02	88.75	96.62
p286k	98.72	88.63	95.98
p330k	98.50	88.46	96.02
p388k	99.31	87.77	97.15
p418k	99.42	88.84	97.66
p483k	99.76	90.06	98.27

Table A.5: Compression ratio comparison for REWPD

ered when there is no data transfer overhead. The lower bounds for all these schemes is equal because their decoders regenerate the test vectors serially using counters. The last column reports the number of clocks for the proposed scheme. It is evident from the table that the presented compression method and decompression architecture makes the external self test significantly faster compared to available techniques and external self-test time is close to the internal BIST schemes.

Table A.7 compares the on-chip area overhead of the proposed scheme with state of the art deterministic logic BIST (DLBIST) [Gher 04]. The cell area of the proposed decompression architecture as well as of the circuit is shown in two input NAND equivalent. The second column named *Decoder Area* shows the area required by the proposed decompression hardware. This includes the area of the binary decoder, the memory and the generate unit. The third column labeled *Circuit Area* reports the total cell area of the circuit while the fourth column with header *Decoder Overhead* tells the overhead that would be caused in circuit size if the proposed decoder is added. The last column presents the average area overhead caused by the bit-flipping logic (BFL) as reported in [Gher 04]. Here only average area overhead is reported because the experimented circuits are different in both cases. The numbers in the table demonstrate that the area overhead of the proposed decompression architecture is very small compared to size of the circuit and compared to DLBIST significant savings in hardware are achieved.

Circuit	DLBIST	Conventional	Proposed
p35k	1.26	29.18	4.61
p45k	3.33	323.01	3.78
p77k	3.03	39.47	3.35
p81k	4.98	39.84	5.61
p89k	9.59	172.69	10.47
p100k	7.92	142.56	8.11
p141k	4.84	116.54	6.7
p239k	5.39	215.68	6.24
p259k	5.39	215.64	6.52
p267k	4.92	221.45	7.85
p269k	4.91	221.03	7.72
p279k	4.13	227.97	7.69
p286k	4.13	228.09	9.15
p330k	3.16	202.65	8.05
p388k	5.38	269.42	7.65
p418k	8.27	529.87	12.35
p483k	8.97	637.08	11.0

Table A.6: Test time comparison for REWPD in million cycles

Moreover the size of the proposed decoder is constant with respect to the number of scan chains and does not depend on the circuit complexity. For example, p418k is more complex than p330k but both contain the same number of scan chains (64) and so the same size of the decoder (1203). This also explains why the overhead of p45k is the largest. The p45k is one of the smallest in size but contains the largest number of scan chains. Because of this, decoder size of p45k is larger than the decoder size of the biggest circuit p483k.

A.3 Nearly Complete Reseeding

A modular 64-bit LFSR was used for test sets encoding in NCR. A randomly generated phase shifter containing single XOR gate for each scan chain was used in conjunction with each LFSR. The encoding efficiency of proposed scheme is evaluated as a function of the number of ignored vectors per seed in table A.8. The total storage requirements and encoding efficiency are given for $I = 1, 2, 3, 4$ where I represents the number of ignored vectors per seed. Sub-column *Storage* reports the total storage while the sub-column labeled *EE* tells the encoding efficiency. The total storage includes the amount of memory required to store seeds and the embedding information. Fixed length data fields are used to encode the embedding information.

Circuit	Decoder Area	Circuit Area	Decoder Overhead (%)	Avg. BFL Overhead (%)
p35k	707	65,228	1.08	
p45k	1456	61,486	2.37	
p77k	504	105,545	0.48	
p81k	396	179,131	0.22	
p89k	632	132,292	0.48	
p100k	632	135,339	0.47	
p141k	722	245,365	0.29	
p239k	979	370,672	0.26	
p259k	979	500,625	0.19	14
p267k	1027	352,755	0.29	
p269k	1027	354,903	0.29	
p279k	1132	405,666	0.27	
p286k	1132	537,518	0.21	
p330k	1203	498,815	0.24	
p388k	1088	712,576	0.15	
p418k	1203	594,629	0.20	
p483k	1294	682,568	0.18	

Table A.7: Hardware overhead comparison

The table reveals that except p100k, encoding efficiency increases with the increasing number of ignored vectors per seed. The highest improvement in encoding efficiency is achieved by incrementing I from 1 to 2. For the larger values of I the gain per ignored vector keeps decreasing. The reason for this decreasing gain is that when more and more care bits are encoded with the same seed, dependence among linear equations increases. With the increasing linear dependence, the chances of encoding significantly large number of care bits that can offset the cost of ignored bits storage decreases. The gain in encoding efficiency turns negative with the increasing number of ignored vectors if the gain achieved by ignoring a vector fails to offset embedding information cost. The circuit p100k is an example of this. Because of the significant increase in test time with the increasing number of ignored vectors, $I = 2$ and $I = 3$ seems a good choice that offer significantly higher encoding efficiency within affordable time.

Table A.9 compares the storage requirements of nearly complete reseeding with continuous reseeding [Volk 03]. In this table second and third columns labeled *Continuous Reseeding* and *Proposed* report the storage requirements and the encoding efficiency of continuous reseeding and nearly complete reseeding respectively. For some circuits, continuous reseeding was not able to encode the complete test set using 64-bit LFSR. Sufficiently larger LFSRs were used in this case for continuous reseeding. For the proposed scheme number of vectors ignored per seed were limited to 3. The last column

	I=1		I=2		I=3		I=4	
Circuit	Storage	EE	Storage	EE	Storage	EE	Storage	EE
p35k	2,241,845	0.47	2,023,947	0.52	1,873,293	0.56	1,751,832	0.60
p77k	252,430	0.57	244,906	0.59	239,817	0.61	235,519	0.62
p81k	708,761	0.49	622,380	0.56	586,713	0.59	561,024	0.62
p89k	827,764	0.56	773,055	0.62	737,058	0.65	731,210	0.66
p100k	305,248	0.52	285,230	0.56	288,538	0.55	291,315	0.54
p141k	1,162,899	0.50	1,053,054	0.55	974,888	0.60	962,794	0.62
p239k	758,168	0.55	691,155	0.61	652,920	0.64	627,714	0.66
p259k	1,059,826	0.56	957,819	0.62	893,107	0.67	862,320	0.69
p267k	1,255,887	0.45	1,182,622	0.48	1,143,952	0.49	1,131,678	0.50
p269k	1,238,287	0.45	1,168,761	0.48	1,127,298	0.50	1,110,524	0.51
p279k	1,203,978	0.53	1,119,299	0.58	1,090,302	0.60	1,072,617	0.61
p286k	1,650,841	0.56	1,551,156	0.60	1,504,748	0.62	1,460,569	0.63
p330k	1,254,112	0.61	1,161,974	0.65	1,105,879	0.67	1,080,213	0.68
p388k	1,450,873	0.57	1,341,627	0.62	1,281,657	0.65	1,255,324	0.66
p418k	1,779,436	0.46	1,639,122	0.49	1,568,934	0.51	1,542,219	0.52

Table A.8: Impact of increasing ignored vectors per seed on encoding efficiency

named *Improvement* shows the improvement achieved over continuous reseeding. The table reveals that using the proposed scheme a gain up to 118 percent can be achieved in encoding efficiency which in turn reduces the storage requirements more than half as compared to state of the art reseeding method. Most importantly the gain is higher for the difficult to test circuits having the test sets with more care bits.

	Continuous Reseeding		Proposed		Improvement
Circuit	Storage	EE	Storage	EE	%
p35k	3,034,584	0.34	1,873,293	0.56	64.7
p77k	293,175	0.50	239,817	0.61	22.0
p81k	1,259,925	0.27	586,713	0.59	118.5
p89k	982,832	0.48	737,058	0.65	35.4
p100k	430,430	0.37	285,230	0.56	51.4
p141k	1,647,825	0.35	974,888	0.60	71.4
p239k	1,255,444	0.33	652,920	0.64	93.9
p259k	1,808,268	0.33	893,107	0.67	103.0
p267k	1,703,692	0.33	1,143,952	0.49	48.5
p269k	1,684,236	0.33	1,127,298	0.50	51.5
p279k	1,638,150	0.39	1,090,302	0.60	53.8
p286k	2,249,475	0.41	1,504,748	0.62	51.3
p330k	1,842,374	0.41	1,105,879	0.67	63.4
p388k	2,224,800	0.37	1,281,657	0.65	75.7
p418k	2,603,370	0.31	1,568,934	0.51	64.5

Table A.9: Comparison of NCR with continuous reseeding

Table A.10 shows the on-chip area overhead of the proposed scheme in two input NAND equivalent. The second column labeled *Decoder Area* shows the area required by the proposed decompression architecture. This includes the area of the flip vector generator, the flipping logic and the memory access mechanism to write and read data from memory. The third column reports the total cell area of the circuit while the fourth column tells the overhead that would be caused in circuit size with the addition of the proposed decoder. The numbers in the table demonstrate that the area overhead of the proposed decompression architecture is very small compared to the size of the circuit. Moreover the size of the proposed decoder is constant with respect to the number of scan chains and does not depend on the circuit complexity. For example, p418k is more complex than p330k but both contain the same number of scan chains (64) and so the same size of the decoder (2094).

Circuit	Decoder Area	Circuit Area	Decoder Overhead (%)
p35k	1016	65,228	1.56
p77k	847	105,545	0.80
p81k	608	179,131	0.34
p89k	968	132,292	0.73
p100k	968	135,339	0.72
p141k	1028	245,365	0.42
p267k	1603	352,755	0.45
p269k	1603	354,903	0.45
p279k	1973	405,666	0.49
p286k	1973	537,518	0.36
p330k	2094	498,815	0.42
p388k	1865	712,576	0.26
p418k	2094	594,629	0.35

Table A.10: Decompression hardware overhead for NCR

APPENDIX B

Published Papers

Conference Proceedings

1. A. W. Hakmi, S. Holst, H.-J. Wunderlich, J. Schlöffel, F. Hapke, and A. Glowatz. "Restrict encoding for mixed-mode BIST". In: Proceedings of VLSI Test Symposium (VTS), pp. 179-184, 2009.
2. A. W. Hakmi, H.-J. Wunderlich, C. G. Zoellin, A. Glowatz, , J. Schlöffel, F. Hapke, and L. Souef. "Programmable deterministic built-in self-test". In: Proceedings of International Test Conference (ITC), Paper 18.1, 2007.
3. A. W. Hakmi, H.-J. Wunderlich, V. Gherman, M. Garbers, and J. Schlöffel. "Implementing a scheme for external deterministic self-test". In: Proceedings of VLSI Test Symposium (VTS), pp. 101-106, 2005.

Workshop Contributions

1. A. W. Hakmi, H.-J. Wunderlich, C. G. Zoellin, A. Glowatz, , J. Schlöffel, F. Hapke, and L. Souef. "Programmable deterministic built-in self-test". In: 19th ITG/GI/GMM Workshop Testmethoden und Zuverlässigkeit von Schaltungen und Systemen, pp. 61-65, 2007.
2. A. W. Hakmi, H.-J. Wunderlich, V. Gherman, M. Garbers, and J. Schlöffel. "Im-

plementing a scheme for external deterministic self-test". In: 17th ITG/GI/GMM Workshop Testmethoden und Zuverlässigkeit von Schaltungen und Systemen, pp. 27-31, 2005.

APPENDIX C

Short Presentation of the Author

Abdul-Wahid Hakmi received the B.Sc. degree in Computer Science from International Islamic University Islamabad, Pakistan, in 2001, and M.Sc. degree in Information Technology from the University of Stuttgart, Germany, in 2003. From 2004 to 2010, he was working as research assistant and pursuing the doctor degree at the Institute of Computer Architecture and Computer Engineering, University of Stuttgart, Germany. He was involved in the projects AZTEKE and MAYA supported by the German Federal Ministry of Education and Research (BMBF).

Abdul-Wahid Hakmi is member of IEEE.

His research interests include self-test, test compression and logic diagnosis.