Test Data Compression Techniques for IP Core Based SoC: Time, Power and Area Optimization

A THESIS SUBMITTED TO

NIRMA UNIVERSITY

FOR THE DEGREE OF

Doctor of Philosophy

IN

TECHNOLOGY AND ENGINEERING

BY

Usha Sandeep Mehta

INSTITUTE OF TECHNOLOGY, NIRMA UNIVERSITY,

SARKHEJ-GANDHINAGAR HIGHWAY, AHMEDABAD-382 481

GUJARAT, INDIA

May 2011

Test Data Compression Techniques for IP Core Based SoC: Time, Power and Area Optimization

A Thesis

Submitted for the Degree of

Doctor of Philosophy

in

Technology and Engineering

By

Usha Sandeep Mehta

Under the guidance of

Dr. K. S. Dasgupta

Director, IIST, Thiruvananthapuram, Kerala

&

Dr. N. M. Devashrayee

PG-VLSI Coordinator, Nirma University, Ahmedabad

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

AHMEDABAD-382481

May 2011

CERTIFICATE

This is to certify that Ms. **Usha Sandeep Mehta** is registered as a student under Registration No.**07EXTPHDE17** for Doctoral programme under the Faculty of Nirma University and have fulfilled the requirement of **R.PH.D 6.4** to attend the Department for the work and coursework (for 90 days) as required.

Dr. N. M. Devashrayee

Name & Signature of Co-Guide

Dr. K. S. Dasgupta

Name & Signature of Guide

CERTIFICATE

This is to certify that I, **Usha Sandeep Mehta**, am registered as a student under Registration No. **07EXTPHDE017** for Doctoral programme under the Institute of Technology under the Nirma University. I have completed the research work and also the pre-synopsis seminar as prescribed under the regulation No. 3.5 for the Ph.D. study.

It is further certified that the work embodied in this thesis is original and contains the independent investigations carried out by me. The research is leading to ***the discovery of** new facts / **techniques** / correlation of scientific facts already known.

* (Please tick whichever is applicable)

Date : _____ Signature of the student

Forwarded by guide : _____ Co-Guide _____ Guide

Remarks of Head of the Department (if any) :

Date : **Signature :**_____

Remarks of Dean Faculty Concerned :

Date : **Signature:**_____

Remarks of Dean Faculty of Doctoral Studies & Research (if any) :

Date : **Signature :** _____

To,
The Executive Registrar
Nirma University

CERTIFICATE

This is to certify that the contents of this thesis entitled ***"Test Data Compression Techniques for IP Core Based SoC: Time, Power and Area Optimization"*** is the original research work of **Ms. Usha Sandeep Mehta** carried out under my supervision.

I further certify that the work has not been submitted either partly or fully to any other university or body – in quest of a degree, diploma or any other kind of academic award.

Date :
Place :

Name of Co-Guide: Dr. N. M. Devashrayee
Full Designation: Co-ordinator, PG-VLSI Design
Name of Institute: Institute of Technology, Nirma University, Ahmedabad, Gujarat

Place :

Name of Guide: Dr. K. S. Dasgupta
Full Designation: Director,
Name of Institute: Indian Institute of Space Science and Technology, Thiruvananthapuram, Kerala

Candidate's Statement

The work included in this thesis entitled ***"Test Data Compression Techniques for IP Core Based SoC: Time, Power and Area Optimization"*** is an independent investigation carried out by me under the guidance and supervision of **Dr. K. S. Dasgupta** and **Dr. N. M. Devashrayee**. In the thesis, references of the work done by others which have been used are cited at appropriate places.

I hereby declare that the work incorporated in the present thesis is original and has not been submitted to any other university or body – in quest of a degree, diploma or any other kind of academic award.

Date :
Place :

Name of Student: Usha Sandeep Mehta
Reg. No.: 07EXTEPHDE017

ACKNOWLEDGEMENTS

No work is ever the outcome of efforts or talent of single individual. This work is no exception. Many teachers, friends, well-wishers have contributed to this thesis work directly or indirectly and made it possible for me to present it in its current shape. Although it is not possible for me to name and thank them all individually, I must specially mention some of the personalities and acknowledge my sincere indebtedness to them and preserve gratitude to many others in my heart.

First and foremost, I express my deep and sincere gratitude to my guide Dr. K. S. Dasgupta for all his guidance, support, inspiration and seemingly endless patience. I also direct a special gratitude to my co-guide Dr. N. M. Devashrayee. Together with Dr. Dasgupta, he made a great team of advisors and has provided excellent guidance in how to obtain and present research results.

I am deeply indebted to Prof. Virendra Singh and Dr. D. J. Shah who have taken a lots of pain for regularly evaluating my work and provide very precise inputs which made the progress of this research work very fast. My thanks are also due to Prof. Nur Touba for providing technical help.

I am also thankful to Dr Ketan Kotecha, Prof. A. S. Ranade, my faculty friends and project students at Nirma University for their inspiration, support, help and contributions.

Any thanks or regards can not be enough for my mother who is the first and foremost teacher of my life. I don't have enough words to describe the sacrifices of my father who made me able in all adverse conditions of his life. I dedicate this work to both of them. The credit goes to my children Riddhi-Vandeet and in-laws for their patience, support and encouragement. Last but not least, thank you Sandeep for all the love and support you give to me. Thank you for being you.

- Usha Sandeep Mehta

April 2011

ABSTRACT

To handle design complexity and short time-to-market, it is increasingly common to use modular design approach in SoC. Such IP cores with hidden architecture have further exaggerated the two burning issues for fabrication testing of SoC: the test cost and test power. The cost of test is strongly related to the increasing test-data volumes which lead to longer test application times and larger tester memory requirement. The solution to this is test data compression.

The increasing test power leads to system reliability issues. The dynamic power during scan operations plays a major role in overall test power. This dynamic scan power is directly related to the number of transitions during scan-in and scan-out. Here, the 'test data compression' and 'switching activity reduction' issues in context of 'hidden structure of IP cores' are addressed.

In this thesis, various test data compression techniques are surveyed and it was observed that for ATPG generated binary data which contains large number of don't care bits, 'run length code' and 'statistical code' based methods are most suitable in context of IP cores. Similarly, the switching activity reduction methods for external testing are suitable to IP cores for power reduction.

It was inferred that if the 'don't care bit filling' and 'reordering' techniques are used in synergy to pre-process the test data, the compression can be increased and test power can be reduced without much on-chip area overhead.

In this thesis, mostly all existing run-length and statistical code methods are implemented and analyzed for compression, power and area overhead.

Using the 'don't care bit filling' and 'reordering' concepts, the test data processing techniques are proposed to further increase compression and reduce test power. The proposed 'Run Based Don't Care Bit Filling', 'HDR-CBF-DV', '2-D Reordering' and 'WTR-CBF-DV' test data processing techniques are used to improve the run length based test data compression codes. The same way proposed 'FDBAF' test data processing technique improves the results for statistical codes. To improve the overall test application time, the 'Modified Selective Huffman code' is proposed. Its effectiveness in increasing compression and reducing test application time without any extra area overhead is proved mathematically and also demonstrated with large amount of simulation results.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AFDR	Alternate Frequency Directed Run Length Code
AI	Artificial Intelligence
ASIC	Application Specific Integrated Circuits
ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
BIST	Built In Self Test
CUT	Circuit Under Test
DUT	Design Under Test
EFDR	Extended Frequency Directed Run Length Code
FDR	Frequency Directed Run Length Code
FDBAF	Frequency Directed Bit Appending and Filling
FPVL	Fixed Plus Variable Length Run length code
GA	Genetic Algorithm
HDR-CBF-DV	Hamming Distance Based Reordering - Column wise Bit Filling – Difference Vector
IC	Integrated Circuit
IP	Intellectual Property
ITRS	International Technology Roadmap for Semiconductors
MFDR	Modified Frequency Directed Run Length Code
MS-Huffman	Modified Selective Huffman Code
MT Fill	Minimum Transition Fill
SA	Simulated Annealing
SAFDR	Sifted Alternate Frequency Directed Run length code
SAR	Switching Activity Reduction
SoB	System-on-Board
SoC	System-On-Chip
TAT	Test Application Time
TDCE	Test Data Compression Environment

THD	Total Hamming Distance
TTM	Time To Market
VIHC	Variable Length Input Huffman Code
SVIHC	Split Variable Length Input Huffman Code
VLSI	Very Large Scale Integration
WSA	Weighted Switching Activity
WT	Weighted Transition
WTM	Weighted Transition Matrix

Chapter 1

Introduction

From the day in 1947, when Bardeen, Shockley and Brattain at Bell Lab completed their noble prize winning research on bipolar transistor, the use of transistor in terms of integrated circuits (ICs) has been increased at a very rapid rate. The first IC which contained only one transistor, three resistors and one capacitor available commercially was produced by Fairchild Semiconductor Corp. in 1961. From then, the everlasting improvements in semiconductor fabrication technology have led to ICs with billions of transistors. The VLSI industry annual survey shows that total 10^{18} transistors were manufactured in 2003. It means that 100 million transistors in form of integrated circuits were manufactured for each of the human on planet Earth during 2003. The ICs, informally known as chips are embedded nowadays in a wide range of products and systems, from consumer electronics and medical equipment to automotive and aviation systems which usually require high reliability and where the cost of failures can be immense.

The technology development in VLSI industry has made it possible to pack millions of transistor on a single chip. The earlier day's System-On-Board (SoB) which contained components like processors and peripheral devices including data transformation engines, data ports, controllers etc, can be now integrated on one single chip referred as System-On-Chip (SoC) [1].

The VLSI design and fabrication flow is extremely complex and time consuming. Addition to that, as described by Moore's law, the technology window for VLSI changes every two years which forces reduction in time-to-market (TTM). In order to meet short TTM requirements, the ready-made modules are preferred at system level design rather to design each block of system individually. Such ready-made i.e. pre-designed and pre-verified blocks of logic, generally called Intellectual Property (IP) cores are largely used in SoCs. The cores can be designed in-house or bought from core vendors; however, it is the task of the system integrator to integrate them into a system.

Even though the IC design is verified thoroughly and it is assured that the design meets all its intended requirements, because of the possible imperfection in fabrication process due to various physical reasons, many defects such as shorts to power or ground, opens, misaligned materials, extra materials, etc. may appear as faults and cause failures. Therefore, each manufactured IC needs to be tested individually [2]. This process is named as fabrication test. The same is applicable to SoC. The aim of fabrication test is to ensure that each fabricated SoC moving to customer is free from manufacturing defects. During the fabrication test, in general, the test stimuli are applied to Device-Under-Test (DUT) and the produced responses are compared against the expected (golden) ones.

1.1 Test Flow

The test process is divided into the two stages, the test generation and the test application (fabrication test). The physical defects like extra or missing material caused by dust particles on the mask, wafer surface or processing chemicals, can be detected by its resultant electrical (circuit) level failure modes, such as opens, shorts, and parameter degradations. Fault models are used to represent the effect of a failure. The effect of a failure will, at the logical level, appear as incorrect signal values. One of

the earliest and most popular fault models today, is the stuck-at fault model. According to the stuck-at fault model, a defect will cause the line in the design to permanently be stuck at logic value 0 (stuck-at 0) or 1 (stuck-at 1). A stuck-at 0 fault, present at a given fault location, is detected when the stimulus data applied is a 1. The produced response will be a 0 (since the fault location is stuck at 0), which will be different from the expected response which is a 1, hence the fault is detected.

During test generation, an automatic test pattern generator (ATPG) is usually used to generate test-data for the design, including test stimuli and expected responses. The netlist (layout) of the design is given as an input to the ATPG tool which uses sophisticated algorithms to analyze the design and generate test patterns for it. Examples of such test pattern generation algorithms are the D-algorithm and PODEM.

During test application (fabrication test), it is required that the test stimuli can be applied to any given location from the inputs and that the produced responses can be propagated from any given location to the outputs. Hence, two of the most important properties of test are the observability and the controllability. The controllability is the ability of controlling the logic value at a specific location in the IC design. The observability is the ability to observe a logical value at any part of the IC. The controllability is high for the locations close to the inputs while it is low for the locations close to the outputs.

To test an IC is a complex task, even for small ICs. In order to reduce this complexity, the controllability and observability of an IC can be increased during the design stages by adding testability features. This process is called DFT and is, usually, automatically performed using specialized design tools. During the test pattern generation stage, the test-data used to test the fabricated IC is developed. A fault simulator is used to verify the test patterns and to measure the fault coverage. If the fault coverage is low, DFT is repeated until acceptable fault coverage has been achieved. The general aim of DFT is to increase the testability of an

IC. Usually, DFT introduces a certain area and performance overhead. For example, it is possible to increase the observability and the controllability by inserting a direct connection, a so-called test point, between the hard-to-test fault location and an I/O pin. The test point DFT approach is straightforward, however, it does not scale as the number of hard-to-test fault locations is increased. A more scalable DFT-technique is to use scan chain insertion, which is a widely adopted DFT-technique. To make a design scanable, the FFs in the design are modified with one additional scan input, one additional scan output, and one scan enable input. The scan-modified FFs are then connected in shift registers which is known as scan chains.

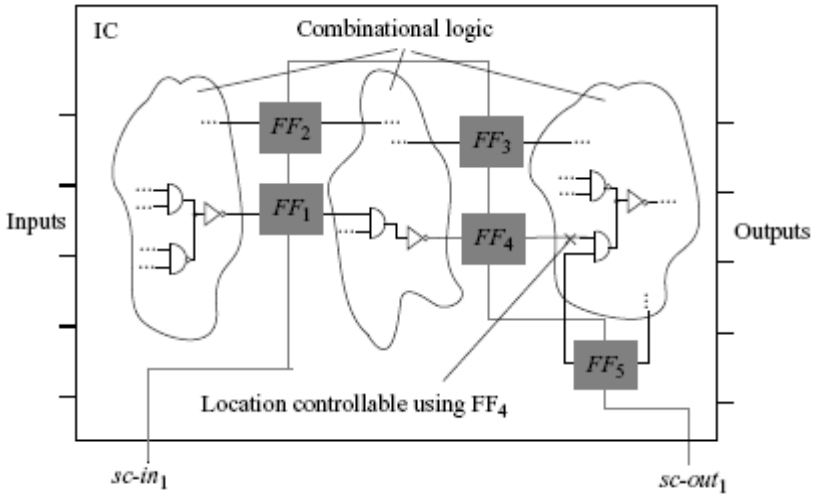

Figure 1.1: Scan Chain

In Figure 1.1, the 5 FFs in the design have been scan-modified and connected into one scan chain. (The scan enable is not illustrated for reasons of readability.) Two additional I/O pins, sc-in$_1$ and sc-out$_1$, are added for the test stimuli shift-in and the produced responses shift-out, respectively. Scan chain testing implies that the design has two modes: functional mode and test mode. The flow of a scan cycle is as follows:

1. Assert test mode, shift in test stimuli (scan-in phase) and set up the desired inputs.

2. Assert functional mode and apply one clock cycle. The produced responses are now captured in the FFs and at the outputs.
3. Assert test mode and shift out the produced responses (scan-out phase).

The test-data corresponding to the bits required for a full test stimuli shift-in, apply and capture, and shift-out of the produced responses is called a test pattern. For efficient test application, the test stimuli of the following test pattern are shifted in while the produced responses from the current test pattern are shifted out, that is, a concurrent scan-in and scan-out phase is performed. The scan test application is applied in Figure 1.2 using two test patterns, tp_1 and tp_2, which are applied to the IC design. The test application time for the two test patterns is 17 clock cycles. The test application time TAT_{sc} (number of clock cycles) for a test T used to test an IC with sc scan chains is as follows:

$$TAT_{sc} = (1 + ff) * l + ff \quad \text{.....(1.1)}$$

where l is the number of test patterns that are applied and ff is the length of the longest scan chain among the SC scan chains. The rate at which the test-data is shifted is given by the scan frequency, f_{scan}.

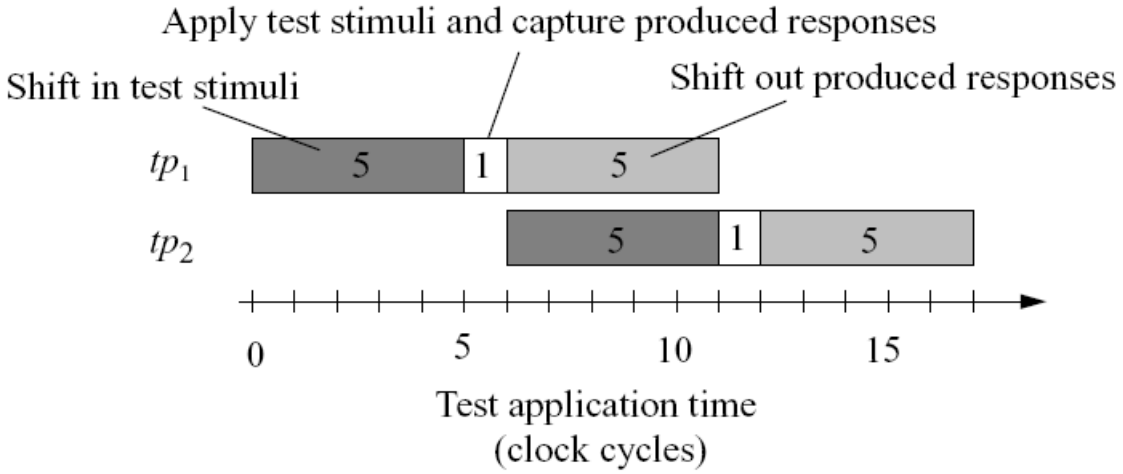

Figure 1.2: Scan Test Application

Fabrication test is usually performed using automatic test equipment (ATE). The test stimuli and expected responses are stored in the ATE memory. Testing is performed by applying test stimuli to the device under test, and by comparing the produced responses to the expected ones. A difference between the expected response and the produced ones indicates that a fault is present and that the device under test should be discarded. The rate at which the test-data is applied is given by the operating frequency of the ATE, f_{ATE}.

1.2 Test Cost

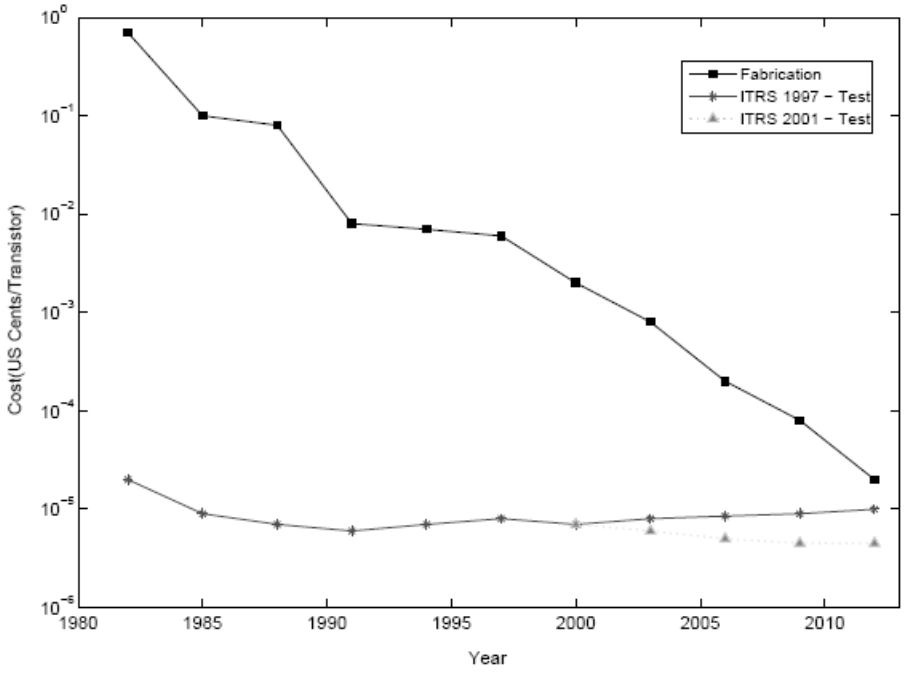

Figure 1.3: Manufacturing Cost versus Test Cost [Courtesy: ITRS]

The importance of reducing the cost of test is motivated by comparing the test cost with the fabrication cost. With the advances in fabrication process for IC, the fabrication cost has reduced drastically and now the testing cost is becoming a dominating part of the overall manufacturing

cost of an IC. Figure 1.3 adapted from International Technology Roadmap for Semiconductors (ITRS) 1999 [3] and ITRS 2001 [4] shows how the relative cost of test per transistor grows compared to the fabrication cost per transistor. It shows that the actual cost of test is almost constant while the cost of fabrication has been dramatically reduced over the years. Today, the overall manufacturing cost includes the cost of design, fabrication and test all together and the cost of test is a significant part of this overall manufacturing cost.

1.2.1 Test Time

As each IC is tested individually, the test cost per IC is directly proportional to the test application time per IC. Conventional external testing involves storing all test vectors and test response on an external tester-that is, Automatic Test Equipment (ATE). The overall test application time, in such case, involves time for transferring the test stimuli from tester, response generation time, time for transferring the generated response to tester and time for comparison of generated response with expected response. As per the ITRS 2009, the test application time per SoC will be increased 209.34 times in 2024 compared to it is today in 2009 [5].

1.2.2 Test Data Volume

As a result of the emergence of new fabrication technologies and design complexities, the number of tests and corresponding data volume and test time increase with each new fabrication process technology. As per the ITRS 2009, the test data volume which is 19Gb today, will be 488 times larger in 2024 [5]. The Figure 1.4 [6] shows the volume of test data with corresponding technology.

Just to maintain test quality requirements, new tests require: greater than 2X the test time to handle devices that double in gate count while maintaining the same number of scan channels, 3X to 5X the number of patterns to support at-speed scan testing for the growing population of timing defects at 130-nm and smaller fabrication processes, and 5X the number of patterns to handle multiple-detect and new DFM-based fault models.

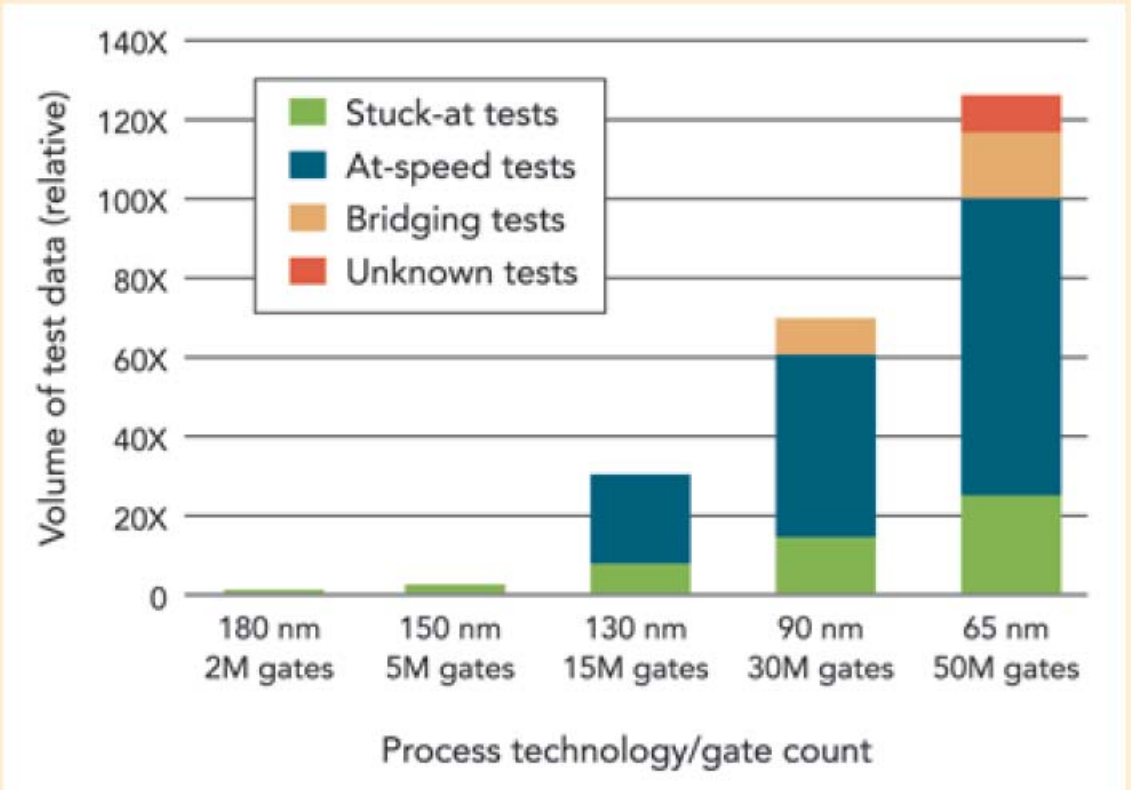

Figure 1.4: Test Data Volume [Courtesy: www.tmworld.com]

The ATEs have limited speed, memory, and I/O channels. The test data bandwidth between the tester and the chip is relatively small; in fact, it is often the bottleneck determining how fast you can test the chip.

$$test\ time\ \geq \frac{amount\ of\ test\ data\ on\ tester}{number\ of\ tester\ channels\ X\ tester\ clock\ rate} \quad \text{.....(1.2)}$$

Testing cannot proceed any faster than the amount of time required to transfer the test data So it can be said that the test time is directly proportional to test data volume (in bits). The test data volume contains test stimuli and expected response. In addition to that, larger test data demands large tester memory.

Three general approaches help to reduce the test time required by ATE per chip [7] are as following:

A. Stand-alone Built-In-Self-Test

Traditional stand-alone BIST involves using on-chip hardware to perform all test pattern generation and output response analysis. Stand-alone BIST eliminates the need for tester storage. This is very useful for performing self-test in the field when there is no access to a tester. However, achieving high fault coverage with stand-alone BIST generally requires considerable overhead because of random-pattern-resistant (RPR) faults, which have low detection probabilities. Detecting such faults requires either test points or deterministic-pattern-embedding logic. Other issues with BIST include the need for a BIST-ready design, a way to handle false and multicycle paths, and the need to keep non-deterministic values from corrupting the final signature.

B. Hybrid Built-In-Self-Test

If a particular chip design uses BIST only for manufacturing test, then hybrid BIST can be more cost-effective than stand-alone BIST. Hybrid BIST involves storing some data on the tester to help detect RPR faults. The simplest approach is to perform ATPG for RPR faults not detected by pseudorandom BIST to obtain a set of deterministic test patterns that "top up" the fault coverage to the desired level, and then store those patterns directly on the tester. More efficient hybrid BIST schemes store the deterministic top-up patterns on the tester in a compressed form,

then use the existing BIST hardware to decompress these patterns. Some schemes embed deterministic patterns by using compressed weight sets or by perturbing the pseudorandom sequence in some manner.

C. Test Data Compression

Test data compression Test data compression involves adding some additional on-chip hardware before and after the scan chains. This additional hardware decompresses the test stimulus coming from the tester; and also compacts the response after the scan chains and before it goes to the tester. This permits storing the test data in a compressed form on the tester. With test data compression, the tester still applies a precise deterministic (ATPG-generated) test set to the circuit under test (CUT).

1.2.3 Test Data Compression

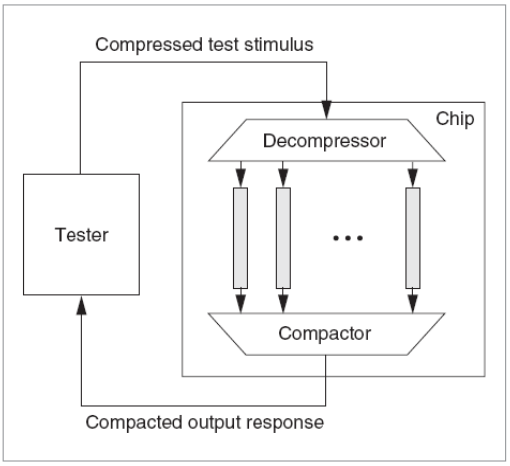

Figure 1.5: Test Data Compression

Considering above three methods, the stand alone BIST and hybrid BIST can be incorporated by the IP core designer however these schemes are not in the scope of system integrator as he cannot look into or modify the internal architecture of any IP core. The remaining method, test data

compression, is further explored for test time reduction in case of IP core based SoC in this thesis.

As shown in Figure 1.5, test-data compression will allow less numbers of bits to be transferred from ATE to DUT which in turn will require less data transfer time. Another advantage is that it offers is less test data storage requirement on ATE. The resurgence of interest in test data compression has also led to new commercial tools like OPMISR and SmartBIST tools from IBM [8] and the TestKompress tool from Mentor Graphics [9]. Even if, it is assumed that the size of ATE vector memory is assumed to increase as fast as DRAM bit size increases, the test data volume compression ratio required for today's SoC is 80 and it is further predicted that the requirement of this ratio will be as high as 104000 in 2024 [5]. So it is inferred that for test cost reduction, the test data compression techniques should be further explored.

1.3 Test Power

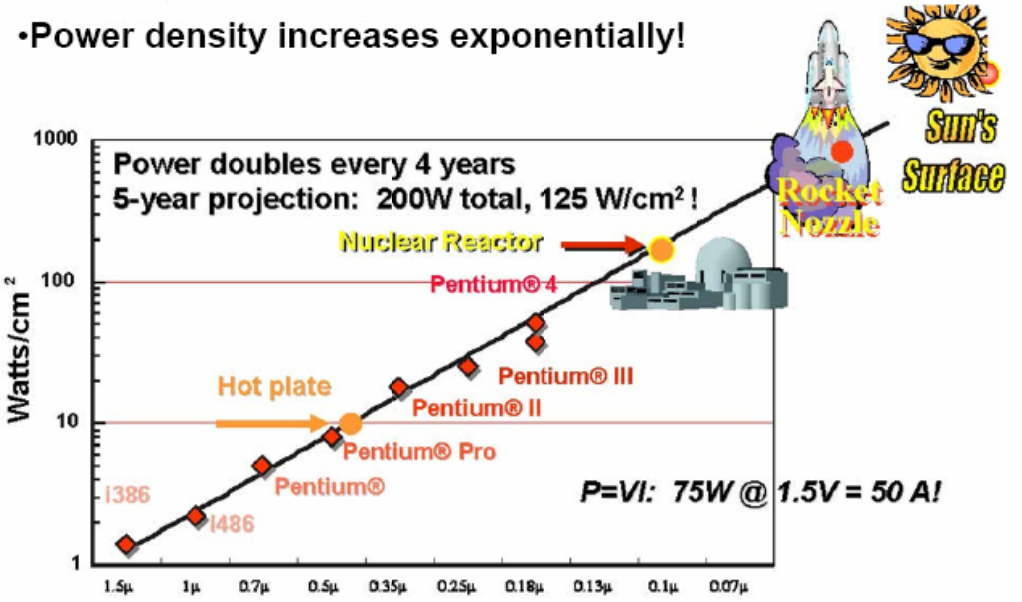

Figure 1.6: Power Density versus Channel Length [Courtesy: INTEL Corp.]

From last two decades, approach of VLSI industry has been to lower voltage and smaller geometry with each generation of IC process. As the result of lower voltage and shrunk geometry, the leakage current and energy increases contributing to higher power. As shown in Figure 1.6, the power density doubles every four year. The problem of power dissipation extends to power delivery, distribution and reliability issues [10]. Hence, low power VLSI design techniques are the major concern for current VLSI industry. Generally, a circuit in test mode may consume the power as high as twice the power consumed in the normal mode. The reasons are as follows. The switching activity of all nodes is often several times higher than the activity during normal operations. Parallel testing strategy adds to power consumption. While testing, the DFT circuits in addition to normal circuit will also consume power. These all reasons can cause significantly larger switching activity in the circuit during test than that during its normal operation. Since dynamic power dissipation in CMOS circuits is proportional to switching activity, this excessive switching activity during test may be responsible for cost, reliability, performance verification, autonomy and technology related problems [11].

The dynamic power has been the major part of overall power dissipation in CMOS circuits. Since this dynamic power is directly proportional to switching activity, the problem of test power reduction can be considered as problem of 'Switching Reduction'.

1.4 Issues Related to IP Core Based SoC

As discussed in section 1.2 and 1.3, the existing fabrication testing method faces two major problems: test cost and test power. As stated there, test cost problem can be addressed using test data compression and test power problem can be addressed using switching activity reduction. However, the current research on test data compression and

switching power reduction cannot be directly applied to the IP core based SoC.

The structure of IP cores are often hidden from system integrator. For this reason, neither any modification to its internal scan chain nor any DFT insertion is possible for IP cores. Further, any testing tools like Automatic Test Pattern Generator (ATPG) or fault simulation cannot be applied to it. Such cores are coming with ready to use test data. This test data is used to test the core when it is in isolation as well as when it is as a part of system after being integrated into system. It is usually assumed that the core is directly accessible and it becomes the task of the system integrator to ensure that the logic surrounding the core allows the test stimuli to be applied and the produced responses to be transported for evaluation.

In this scenario, only those test data compression and switching reduction schemes which do not require any modification in scan chain or internal structure are suitable to this application. Further, these schemes should not demand the help of any testing tool like ATPG or fault simulation and must be applicable to any kind of test data i.e. partially specified or fully specified.

1.5 Objective

The motivation here is to solve the issues like test cost and test power in context of hidden structure of IP core. The goal decided is to use test data compression for test cost, switching activity reduction for test power and ready-made test data in context of IP core based SoC. Figure 1.7 and 1.8 shows the motivation of the thesis and goal of the thesis respectively.

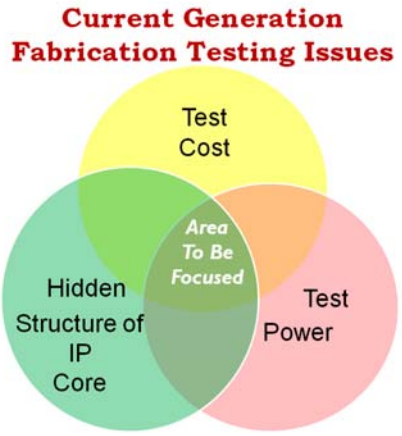

Figure 1.7: Motivation of the Thesis

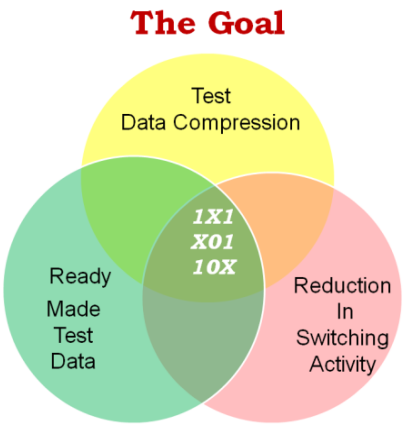

Figure 1.8: Goal of the Thesis

The objective of the thesis is defined as:

To design a test data processing or/and compression method specifically for IP core based SoC which

1. does not require any insertion or modification in internal structure of IP core
2. does not require any test development tools like ATPG or fault simulation
3. increases the compression
4. reduces the overall test application time (TAT)
5. reduces the switching activity during scan operation
6. does not increase the on-chip area overhead

compared to existing methods.

1.6 Adopted Methodology

In this thesis work, the methodology adopted is:

1. The need or issues related to current generation testing is studied.
2. The existing solutions for that issue are surveyed.

3. From the available solutions, the possible solutions for IP core based SoC are separated.
4. These solutions are implemented and analyzed.
5. The existing solutions are optimized.
6. For further improvement, new methods are proposed and proved mathematically supported by sufficient amount of experimental results.

1.7 Thesis Organization

The rest of the thesis work is divided into four chapters. The detail and publications related to each chapter is as follows:

Chapter 2 Literature Survey

In this chapter, the need and parameters related to test data compression is discussed in brief. The various test data compression schemes like linear decompression based schemes, broad cast scan based scheme and code based scheme with their suitability to IP cores is reviewed. After the survey on all four classifications of code based scheme, i.e. run-length based code, statistical code, dictionary codes and constructive code, the classifications suitable to IP core, `run length based code' and `statistical code' is explained in detail.
This chapter also includes the complete survey on various classifications of switching activity reduction methods in case of external testing, DFT and BIST techniques with criteria of the suitability to IP core. The selected external low power testing technique is further explored and 'don't care bit filling method' as well as 'test vector reordering method' is described in detail.

[Publications Related to This Chapter: Ref. J-3, C-1, C-8, C-9]

Chapter 3 Run Length Code Based Test Data Compression

This chapter contains the overview and analysis through implementation and simulation of existing run length based test data compression techniques like simple run length coding, Golomb coding, Frequency Directed Run length coding (FDR), extended FDR, modified FDR, alternating FDR, shifted alternating FDR, etc. The analysis of area overhead for various on-chip decoders is included. The proposed *`Run Based Bit Filling'* depending upon the classification of run types is described with necessary motivational example. The upper limit of compression based on entropy is estimated for the proposed 'Run Based Bit Filling'. The test data processing method "*Hamming Distance Based Reordering and Columnwise Bit Filling with Difference Vector (HDR-CBF-DV)*" is included in this chapter. This chapter also contains the power efficient test data processing technique "*2-D Reordering*" and the power efficient 'Weighted Transition Based Reordering (WTR)'. The effectiveness for each scheme is demonstrated with sufficient simulation results.

[Publications Related to This Chapter: Ref. J-1, J-5, C-3, C-4, C-5, C-6]

Chapter 4 Statistical Code Based Test Data Compression

This chapter includes overview and analysis through implementation and simulation of existing statistical test data compression methods like Huffman code, selective Huffman code, optimal selective Huffman code, variable length input Huffman code (VIHC), Split VIHC, fixed plus variable length coding, RL-Huffman code, multilevel Huffman code etc. A test data compression code `Modified Selective Huffman Code (MS-Huffman Code)' with necessary decoder architecture is proposed. The improvement in compression and overall test application time compared

to existing method without any extra area overhead is proved mathematically as well as demonstrated with necessary simulation results. This chapter includes the test data processing technique `Frequency Dependant Bit Appending and Filling (FDBAF)'.

[Publications Related to This Chapter: Ref. J-2, J-4, C-2, C-7]

Chapter 5 Conclusion and Future Work

This chapter presents the conclusion of the thesis. It also uncovers the scope of future work in the same direction.

Chapter 2

Literature Survey

Because of the development in fabrication and design technology, IP Core based SoC has come up as the current state of art for VLSI design. While test cost and test power remain burning issues for testing, the hidden structure of IP core has further exaggerated the problem.

The test cost is directly proportional to test application time which is directly related to amount of test data. Hence it can be said that test cost problem can be converted in to test data compression problem.

Similar way, the dynamic power plays the significant role in overall test power consumption. The dynamic power is directly proportional to switching activity during test. It means that the overall test power can be directly controlled by reducing the switching activity during test.

The detailed survey on test data compression and switching activity reduction methods follows in subsequent sections.

2.1 Test Data Compression

The advantage of test data compression is that it generates the complete set of patterns applied to the CUT with ATPG, and this set of test patterns can be optimized with respect to the desired fault coverage. Test data compression is also easier to adopt in industry because it's compatible with the conventional design rules and test generation flows for scan testing. Test data compression has two advantages. The most

important is it can reduce the test time for a given test data bandwidth. The second advantage is it reduces the amount of data stored on the tester, which can extend the life of older testers that have limited memory.

Test data compression involves two methodologies: 1. for the single scan chain testing, it reduces the number of bits to be transferred from ATE to DUT and thus reduces the test data transfer time. At DUT, there is one on-chip decoder to expand the code word again in original form. This decoding time may further affect the overall reduction in test application time. 2. for multiple scan chain testing, the decompressor expand the data from n tester channels to fill greater than n scan chains. Increasing the number of scan chains shortens each scan chain, in turn reducing the number of clock cycles needed to shift in each test vector.

Test data compression must compress the test vectors lossless (i.e. it must reproduce all the care bits after decompression) to preserve fault coverage. Test vectors are highly compressible because typically only 1% to 5% of their bits are specified (care) bits. The rest are don't-cares, which can take on any value with no impact on the fault coverage. A test cube is a deterministic test vector in which the bits that ATPG does not assign are left as don't-cares (i.e. the ATPG does not randomly fill the don't-cares). In addition to containing a very high percentage of don't-cares, test cubes also tend to be highly correlated because faults are structurally related in the circuit. Both of these factors are exploitable to achieve high amounts of compression.

2.2 Test Data Compression Methods

There is variety of data compression algorithms available in literature used for multimedia, communication and such other applications which compresses the data either lossy or lossless. These algorithms can compress any type of data i.e. fully specified (without any don't care bits)

or partially specified (with don't care bits). From all these methods, only those methods which compress data lossless can be applied for compression of test vectors so that the fault coverage is retained.

The ATPG uncompacted test-data consists a large number of unspecified bits, so-called don't-care bits which, together with regularities in the test-data can be explored during the compression, such that a minimal amount of test-data needs to be stored in the tester memory [12]. The test application time can be reduced if decoders are placed on-chip, since the amount of test-data to be applied through the chip I/O pins is reduced [13].

The quality factors of test data compression techniques are:

1. The amount of compression possible
2. The area overhead because of decoding architecture. The on-chip decompression circuitry must be small so that it does not add significant area overhead. The properties of the code are chosen such that the decoder has a very small area.
3. The reduction in test time. Transferring compressed test vectors takes less time than transferring the full vectors at a given bandwidth. However, in order to guarantee a reduction in the overall test time, the decompression process should not add much delay (which would subtract from the time saved in transferring the test data). The properties of the code are chosen is guaranteed to be able to decode the test data as fast as the tester can transfer it.
4. The scalability of compression (does the compression technique work with various design sizes, with few or many scan channels, and with different types of designs?)
5. Power dissipation is an important factor in today's chip design.
6. The robustness in the presence of X states (can the design maintain compression while handling X states without losing coverage?)

7. The ability to perform diagnostics of failures when applying compressed patterns
8. Type of Decoder: data-independent decoder or data dependant decoder. In the former category, the on-chip decoder or decompression program is universal, i.e., it is reusable for any test set. In contrast, the decoder of a data-dependent technique can only decompress a specific test vector. They have difficulties in terms of size and organization for improved compression and often require large on-chip memory. Hence, data-independency is a preferable property

Various methods described in literature to compress specifically VLSI test data can be classified in following subsections.

1. Linear Decompression Based Test Data Compression Method

From a small number of bits (i.e. seed) stored on ATE, a large test data can be reproduce at DUT using only linear operations on seed with help of Linear Feedback Shift Register(LFSR) and XOR network. These schemes are called linear decompression based schemes. This methodology is effectively used in [14, 15, 16, 17, 18 and 19].

2. Broadcast Scan Based Test Data Compression Method

The second classification of these techniques which is called broadcast scan based method, is based on the idea of broadcasting the same value to multiple scan chains (a single tester channel drives multiple scan chains). This is actually a special degenerate case of linear decompression in which the decompressor consists of only fan-out wires. This methodology is widely used in [20, 21, 22, 23] etc.

3. Code Based Test Data Compression Method

The third classification is code based data compression method. It uses data compression codes to encode the test cubes. This involves partitioning the original data into symbols, and then replacing each symbol with a code word to form the compressed data. To perform decompression, a decoder simply converts each code word in the compressed data back into the corresponding symbol. Various code based data compression methods are evolved as described in next section of this chapter.

2.3 Compression Methods for IP Cores

The test data compression methods classified as above is widely accepted for ASIC testing. While considering its application for IP core based SoC testing, the suitability of such methods to IP must be analyzed.

From above three compression methods,

- Linear decompression method requires very simple controlling logic and is very efficient in compressing. The main drawback here is, it requires two stage ATPG. Also it is more convenient with partially specified test data only.
- The Broadcast scan methods are also efficient in compression and require a simple decoder. Like linear decompression, this method also requires one step ATPG and more suitable to partially specified test data with large amount of don't care bits.
- The code based test data compression method is not as efficient in exploiting don't care bits and requires a complex control logic. However it is suitable to any kind of test data and does not require any testing tool.

From above discussion, it can be inferred that the code based test data compression method is suitable to IP core based SoC. Hence, this method is selected as basic scheme for further consideration during this work. Before continuing the survey on code based compression methods, the parameters influencing the code based methods are discussed in the following section.

2.4 Test Data Compression Environment

Test data compression environment (TDCE) [24] comprising compression method and on-chip decoder, is defined and analyzed with respect to three TDCE parameters: compression ratio, area overhead and test application time. Testing in TDCE implies sending the compressed test data from the ATE to the on-chip decoder, decompressing the test data on-chip and sending the decompressed test data to the core under test (CUT). There are two main components in TDCE: the compression method, used to compress the test set off-chip, and the associated decompression method, based on an on-chip decoder, used to restore the initial test set on-chip. The on-chip decoder comprises two units: a unit to identify a compressed code and a unit to decompress it. If the two units can work independently (i.e., decompressing the current code and identifying a new code can be done simultaneously), then the decoder is called parallel. Otherwise, the decoder is referred to as serial.

Testing in TDCE is characterized by the following two parameters:

1. Compression Ratio: which identifies the performance of the compression method and the memory & channel capacity requirements of the ATE
2. Area Overhead: imposed by the on-chip decoder (dedicated hardware or on-chip processor)

There are a number of factors which influence the above parameters:

1. The mapping and reordering algorithm, which prepares the test set for compression by mapping the 'don't cares' in the test set to '0's or '1's, and by reordering the test set.
2. The compression algorithm, which based on a coding scheme, compresses the initial test set.
3. The type of input patterns used as input by the coding scheme, which can be of fixed or variable lengths.
4. The length of the pattern which is the maximum allowed input pattern length used in the coding scheme.
5. The type of the on-chip decoder, i.e., the on-chip decoder can be serial or parallel.

Compression Ratio

Using patterns of various types and various lengths, the compression algorithms exploit different features of the test set. Mapping and reordering the initial test set emphasizes these features. Therefore, the compression ratio is influenced first by the mapping and reordering algorithm, and then by the type of input patterns and the length of the pattern, and finally by the compression algorithm.

As a quantitative measure of test data compression, the following two expressions can be used:

$$test\ data\ compression\ ratio = \frac{compressed\ test\ data\ (\#\ of\ bits)}{original\ test\ data\ (\#\ of\ bits)} \quad \ldots\ldots\ldots\ldots\ldots\ldots (2.1)$$

$$test\ data\ compression\ in\ \% = \frac{original\ test\ data\ (\#\ of\ bits) - compressed\ test\ data\ (\#\ of\ bits)}{orginal\ test\ data\ (\#\ of\ bits)} X\ 100 \quad \ldots\ldots\ldots\ldots\ldots\ldots (2.2)$$

Area Overhead

Area overhead depends upon the decoder type and input pattern. If the decoder is serial then the synchronization between the two units (code identification and decompression) is already at hand. However, if the decoder is parallel, then the two units have to synchronize, which can lead to increased control complexity and consequently to higher area overhead. Depending on the type of the input pattern different types of logic are required to generate the pattern on-chip. For example, if the coding scheme uses fixed-length input patterns, then a shift register is required to generate the patterns, however, if variable-length input patterns (runs of '0's for example) are used, then counters can be employed to generate the patterns. Since the length of the pattern impacts the size of the decoding logic, it also influences the area overhead.

$$Area\ Overhead\ in\ \% = \frac{area\ of\ original\ circuit - area\ of\ decoder}{area\ of\ original\ circuit} X\ 100 \quad \ldots\ldots\ldots\ldots\ldots\ldots (2.3)$$

Test Application Time (TAT)

Test data compression will reduce the test time for transferring the data from ATE to DUT. However, the on-chip decoder will add decoding time to overall test application time. Hence, it should be taken care that the compression does not increase the overall test application time (TAT).

2.5 Code Based Test Data Compression Methods

The data compression codes are generally classified into four categories based on symbol size and codeword size. As the name suggests, in fixed to fixed coding schemes, the symbol size as well as codeword size is fixed. In fixed to variable coding schemes, the symbol size is fixed but codeword size is variable. In variable to fixed coding schemes, the symbol size is variable but codeword size is fixed. While in variable to variable coding schemes, the symbol as well as codeword size is variable. Based on the basic schemes and evolved variants, the code based compression methods are broadly divided into four different categories: Statistical Codes, Run-Length Codes, Dictionary Codes, and Constructive Codes.

All these categories are discussed in following subsections:

2.5.1 Statistical Code Based Methods

Statistical coding partitions the original data into n bit symbols and assigns variable length code words based on each symbol's frequency of occurrence. It assigns shorter code words to symbols that occur more frequently, and longer code words to those that occur less frequently. This strategy minimizes the average length of a code word.

The Huffman code is very widely used code in all lossless data compression applications as the Huffman code is an optimal statistical code that is proven to provide the shortest average codeword length among all uniquely decodable variable length codes. The only disadvantage is that when the Huffman hardware decoder is placed in chip, it requires a large amount of area overhead because its size grows exponentially with symbol size [25].

The improved approach is a selective Huffman coding [26] for which a very simple decoder can be constructed. It only codes the most frequently occurring blocks in-stead of all blocks using code words with small numbers of bits.

The Selective Huffman encoding required extra bit with each codeword. This constant overhead, although minimum for the unencoded data blocks, can be relatively high for the most frequently occurring code words. An optimal selective Huffman code [27] uses an additional Huffman codeword in front of only the unencoded data blocks, relieving in this way the most frequently occurring codewords from the extra-bit over-head.

An arithmetic coding based compression method is proposed in [28, 29]. Given a test set in which all the test vectors are fully specified, some primary input values may be changed to opposite logic values without losing fault coverage [30]. This helps in reducing the number of distinct blocks which later coded with Huffman coding [31].

There are number of techniques proposed by researchers which explore the advantages of statistical codes in combinations of other coding techniques.

In RL-Huffman coding [32], the don't care bits in test vector are first filled with either '0' or '1' such that the overall run length of 1s or 0s has been increased. Such runs with variable size are coded using Huffman coding.

The Multilevel Huffman test data compression method [33, 34] is based on Huffman coding with a limited number of codewords. The test cubes of the CUT are compared against the pseudorandom sequences generated by various cells of an LFSR and if they match, the Huffman coded index for each selected LFSR cell is chosen for feeding the scan chain of the CUT.

As per variable-to-variable Huffman coding [35], densely specified regions are the main sources of unencoded data, and, therefore, their compression is favored by the usage of small distinct blocks. On the

other hand, sparsely specified regions are more efficiently compressed using large distinct blocks, so to improve compression, the test sets should be partitioned into variable-length parts, which means that variable-length distinct blocks should be encoded. Apart from the variable-to-variable nature of the proposed approach, the generated codewords are reusable in the sense that they can encode compatible blocks of different sizes.

2.5.2 Run Length Code Based Methods

The first data compression codes that researchers explored for compressing scan vectors were encoded by runs of repeated values of zeroes.

The simplest scheme based on run-length codes that encoded runs of 0s using fixed-length code words was proposed in [36]. To increase the prevalence of runs of 0s, this scheme uses cyclic scan architecture to allow the application of difference vectors.

A technique based on Golomb codes [37, 38] encodes runs of 0s with variable-length code words. The codewords are divided into groups of equal size m where *m* is any power of 2. The use of variable-length code words allows efficient encoding of longer runs, although it requires a synchronization mechanism between the tester and the chip.

A Frequency Directed Run length code (FDR) similar to Golomb code is proposed in [39, 40] but the difference is the variable group size. It is based on the observation that the frequency of runs of 0s with run length less than 20 is high and even within the range of 0 to 20, the frequency of runs decreases rapidly with increase in run length. So test data compression can be more efficient if the runs of 0s with shorter run length are mapped to shorter codewords.

The FDR code is very efficient for compressing data that has few 1s and long runs of 0s but inefficient for data streams that are composed of both

runs of 0s and runs of 1s. Generally test vectors contain 0s and 1s in group i.e. there will be a run of 1s followed by run of 0s and vice versa. In an extension of FDR (EFDR) [41], the run of 0s followed by bit '1' and run of 1s followed by bit '0' are coded same way as FDR but adding an extra bit at beginning of FDR codeword.

Generally, the test set T is composed of alternating runs of zeros and run of ones. In alternating FDR coding (AFDR) style [42], instead of adding the extra bit to each run length code-word, only one bit will be added at beginning to indicate the type of first run length and then successive run length is automatically alternating type. So all run lengths are coded with normal FDR but at beginning, one bit will be added to indicate the first run type.

An evolution in alternating FDR is Shifted Alternating FDR (SAFDR) [43]. Here the runs are of either made of only ones or only zeroes. The both types of runs are placed alternating. So in such alternating FDR, no codeword is required for zero run length and each codeword will be shifted to one position higher run length. This helps in achieving higher compression compared to Alternating FDR.

Regardless of the good compression ratios the area overhead of FDR is a disadvantage. So a mix of Huffman and FDR is proposed which instead of only patterns of fixed-length uses patterns of variable-length as input to the Huffman algorithm (VIHC) [24]. Here the compression ratio is retained because of FDR and the area overhead is reduced using selective Huffman Coding.

Spilt VIHC [44] approach demonstrates that before going to the VIHC, if the test file is divided in to two or more equal parts and the vectors are reordered in a specific way, the compression ratio can be still improved.

One more scheme based on probability of 0s and FDR is Modified Frequency-Directed Run-length (MFDR)[45]. In this scheme, the groups of FDR is further modified such a way to give better compression ratio than FDR if the probability of 0s in the test set is greater than 0.8565.

A scheme based on selectively relaxing some bits of test vector before encoding it using FDR or Golomb code is proposed in [46]. Here by changing a specified bit with value 1 to a don't-care, two consecutive runs of 0s in the test sequence can be concatenated into a longer run of 0, thereby facilitating run-length coding. This procedure retains the fault coverage of the test set.

The data independent run length coding [47] explores the don't care bits in test patterns. It transmits the first segment of pattern as it is and then compares all other subsequent segments with first segment and decides either the next segment is same as first or complement of first segment.

Stuck at faults based test patterns can be reordered without any loss of fault coverage. The test patterns are reordered based on the minimum Hamming distance between them.

The run based reordering approach [48] is based on reordering the test patterns to give the bigger run lengths of 0s. The variable to fixed plus variable length coding [49] scheme divides code word into two parts: fixed length head section and variable-length tail section. Combined partial test vector reuse and FDR coding [50] finds such a vector, from which parts of each test vector from the different test sets can be sought. Based on this, a vector named overlapped vector which contains parts of each test vector and has shorter length than that of the sum of each test vector's length is decided.

2.5.3 Dictionary Code Based Methods

The basic idea behind dictionary codes is to take advantage of a number of commonly occurring sequences. This scheme partitions the original data into n-bit symbols and uses a dictionary to store each unique symbol. This technique compresses data by encoding each *n*-bits using a *b*-bit code word that corresponds to the symbol's index in the dictionary (*b* is less than *n* when all possible symbols do not occur in the data).

A scan vector compression scheme that uses a complete dictionary is proposed in [51]. The size of each index equals $[\log_2 n]$, where n is the number of distinct scan slices.

A partial-dictionary coding scheme [52] constrains the dictionary size based on the allocated area for the decompressor.

A method based on LZW, particularly LZ77 is proposed in [53, 54]. A partial dictionary along with a correction network that inputs bits to convert a dictionary entry into the desired scan slice is proposed in [55].

A hybrid test data compression method using scan chain compaction and dictionary based Scheme is proposed in [56].

2.5.4 Constructive Code Based Methods

Constructive codes exploit the fact that each n-bit scan slice typically contains relatively few care bits. It is possible to construct the scan slice by incrementally specifying all the care bits using a sufficient number of code words.

A scheme to construct the current scan slice from the previous scan slice by flipping bits is proposed in [57]. A constructive code method [58] first sets all bits in a scan slice to either 0 or 1 (whichever matches the largest number of care bits), and then incrementally loads the care bits with opposite value using either a single-bit or a group-copy mode.

2.6 Code Based Method for IP Cores

The dictionary codes and constructive codes have reported high compression ratio, however, their results in terms of consistency to achieve a high compression ratio, cost of memory demanding decoder, and scan-in power due to high bit transitions are not favoring them for further use [32]. A drawback of using a complete dictionary is that the dictionary size can become very large, resulting in too much overhead for

the decompressor. The drawback for constructive coding is very complex and data dependant decoder.
The remaining two methods: statistical coding and run length based coding are further selected for this thesis work on testing of IP core based SoC.

2.7 Test Power

Generally, a circuit in test mode may consume the power as high as three to four times the power consumed in the normal mode [10]. The main reasons of this extra power are parallel testing strategy and extra power drawn by DFT circuits in addition to normal circuit. This extra power consumption due to test application may give rise to severe hazards to the circuit reliability [11]. Moreover, it may be responsible for cost, performance verification as well as technology related problems.
The terminology related to power issues [59] is as follows:

Energy

The total switching activity generated during test application. Energy has impact on the battery lifetime during power up or periodic self-test of battery operated devices.

Peak Power

The peak power is the highest value of power at any given instant. The peak power determines the thermal and electrical limits of components and the system packaging requirements. If the peak power exceeds a certain limit, the correct functioning of the entire circuit is no longer guaranteed [11].
The peak power consumption corresponds to the maximum of the instantaneous power consumed during the test session. It therefore

corresponds to the highest energy consumed during one clock period, divided by T. More formally, it can be expressed by equation 2.4 [64]:

$$P_{peak} = max_k\left[P_{inst(Vk)}\right] = \frac{max_k[C_o \times V_{DD}^2 \times \sum_i S(i,k) \times F_i]}{2 \times T} \quad \text{.....(2.4)}$$

Average Power

The total distribution of power over a time period, which is generally the amount of power consumed during the application of a test. Elevated average power adds to the thermal load that must be vented away from the device under test. It may cause structural damage to the silicon (hot spots), to bonding wires or to the package.

The average power consumed during the test session is the total energy divided by the test time, and is given in equation 2.5:

$$P_{avg} = \frac{[C_o \times V_{DD}^2 \times \sum_i S(i,k) \times F_i]}{2 \times length_{test} \times T} \quad \text{..... (2.5)}$$

Where V_{DD} is the power supply voltage, C_0 is the minimum size parasitic capacitance of the circuit, $S\ (i,\ k)$ is the number of switching provoked by V_k at node i, $Length_{test}$ is the length of vector sequence, the fanout F_i is the fanout, S_i is the number of switching on node i and T is the clock period.

According to the above expressions of the power and energy consumption, and assuming a given CMOS technology and supply voltage for the circuit design, it is inferred in [59] :

1. The number of switching of a node i in the circuit is the only parameter that has impact on both, the peak power and the average power consumption.

2. The clock frequency used during testing has impact on both the peak power and the average power.
3. The test length the number of test patterns applied to the CUT, has impact only on the total energy consumption

Scan and Capture Power

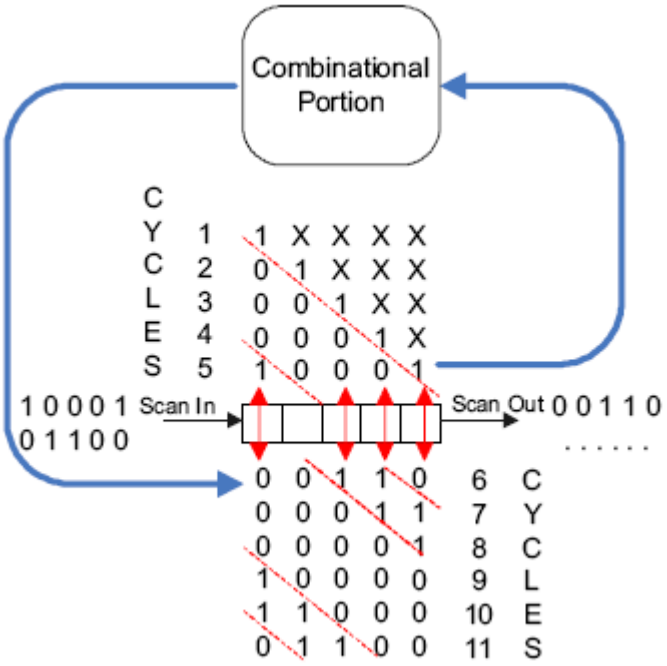

Figure 2.1: Shift and Capture Power during Scan Tests

Figure 2.1 [60] shows transitions in scan cells that cause shift and capture power consumption during scan tests. In this circuit, the first test vector 10001 is shifted into the scan chain in five clock cycles. After one capture cycle, the response vector 00110 is captured into the scan chain and scanned out while the next test vector 01100 is scanned in simultaneously. Each vector row in this figure represents states of the scan cells in one test cycle, the dash lines highlight where transitions happen. During the shift phase, transitions on the scan chain occur

when adjacent bits in test vectors have different logic values, and the number of transitions that it takes is determined by the position where it happens. It means, for scan based designs, each transition from '*1-to-0*' or '*0-to-1*' in the scan-in and scan-out vectors has multi-fold impacts on switching activities. In this regards, it can be inferred that to reduce the test power, the switching activities during scan operation must be reduced.

Differently, capture power is caused by transitions happened when scan cells have different values before and after capture.

The following is the detailed description of scan power.

Scan-in Power

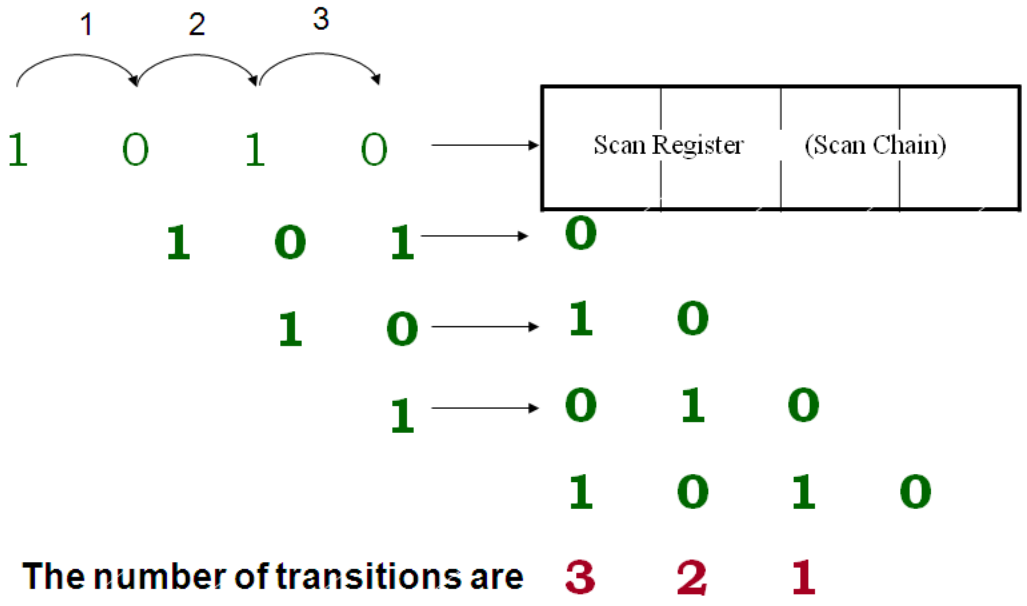

Figure 2.2: Weighted Transitions during Scan In

Figure 2.2 demonstrates how the position of transition in scan vector contributes to test power.

For a given test data set containing m vectors with n bits each, the Weighted Transition for each test vector is given by equation 2.6 and the total weighted transition during test is given by equation 2.7.

$$Weighted\ Transition\ for\ scan-in\ vector\ j = \sum_{i=1}^{n} \big(t(j,i) \oplus t(j,i+1)\big) * (n-i) \quad \text{.....(2.6)}$$

$$Total\ Weighted\ Transition\ during\ scan-in = \sum_{j=1}^{m}\sum_{i=1}^{n} \big(t(j,i) \oplus t(j,i+1)\big) * (n-i) \quad \text{.....(2.7)}$$

Scan-Out Power

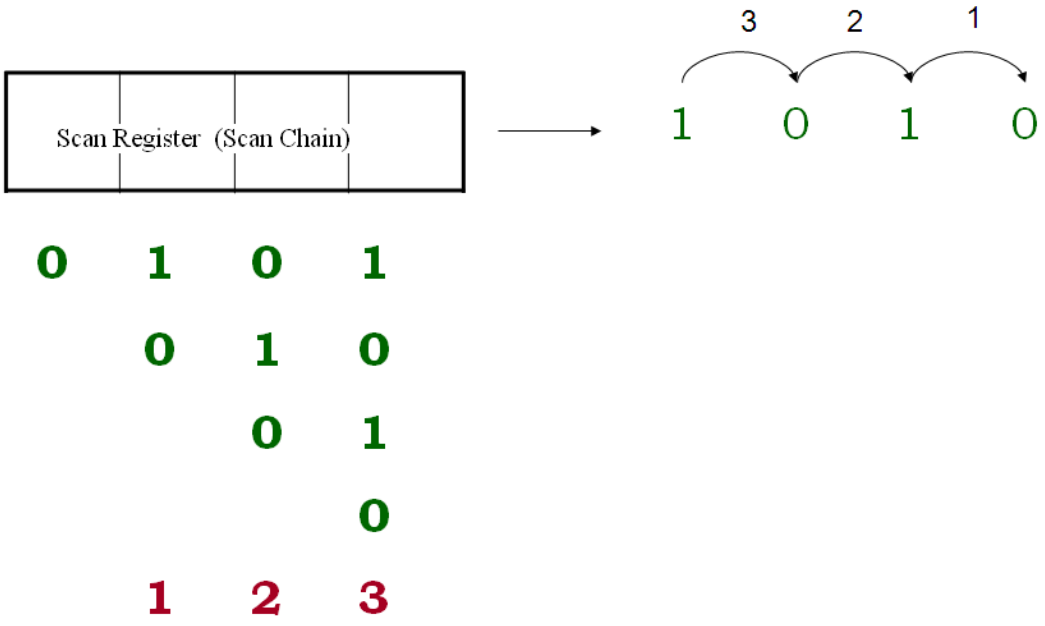

Figure 2.3: Weighted Transitions during Scan Out

Figure 2.3 demonstrates how the position of transition in scan-out vector (response) contributes to test power.

For a given test data set containing m vectors with n bits each, the Weighted Transition for each scan-out test vector is given by equation 2.8 and the total weighted transition during test is given by equation 2.9.

$$Weighted\ Transition\ for\ scan-out\ vector\ j = \sum_{i=1}^{n} \big(t(j,i) \oplus t(j,i+1)\big) * i \quad \text{.....(2.8)}$$

$$Total\ Weighted\ Transition\ during\ scan-out = \sum_{j=1}^{m}\sum_{i=1}^{n} \big(t(j,i) \oplus t(j,i+1)\big) * i \quad \text{.....(2.9)}$$

2.8 Switching Activity Reduction Methods

Various Switching Activity Reduction (SAR) methods described in literature can be classified as follow:

1. Methods dealing with Built-In-Self-Test (BIST)
2. Methods dealing with Design-For-Test(DFT)
3. Methods dealing with external testing or ATE.

2.8.1 SAR Techniques for Built-In-Self-Test

In these techniques, either the BIST architecture is made power conscious or some of the components of BIST are designed with power consideration.

The BIST architecture contains two major components: test pattern generator (TPG) and response checker [2]. Both of these components use Linear Feedback Shift Register (LFSR). The LFSR can be design to reduce the power consumption during test in following ways:

Reducing the Transitions in Test pattern Generator

Various methods for power reduction in case of BIST like a modified clock scheme for a low power BIST test pattern generator, POWERTEST tool : a tool for energy conscious weighted random pattern testing, LT-RTPG: a test-per-scan BIST TPG for low heat dissipation, etc. are discussed in [57]. A low power test pattern generator using the LFSR, called LP-TPG is presented in [61] which inserts intermediate patterns between the random patterns to reduce the transitional activities of primary inputs which eventually reduces the switching activities inside the circuit under test, and hence, power consumption. A polynomial-time algorithm that converts the test pattern generation problem into combinatorial problem called Minimum Set Covering is proposed in [62]. A low power BIST TPG scheme in [63] uses a transition monitoring window (TMW) that comprised of a TMW block and a MUX. The proposed technique suppresses transitions of patterns using the k-value which is a standard that is obtained from the distribution of TMW to observe over transitive patterns causing high power dissipation in a scan chain. A TPG based on Read Only Memory (ROM) which is carefully designed to store the test vectors with minimum area over the conventional ROM is presented in [64]. In [65], an approach to reconfigure the CUT's partial-acting-inputs into a short ring counter (RC), and keep the CUT's partial-freezing-inputs unchanged during testing is used. A low hardware overhead TPG [66] for scan-based built-in self-test (BIST) can reduce switching activity in circuits under test (CUTs) during BIST and also achieve very high fault coverage with reasonable lengths of test sequences. In [67], low-transition linear feedback shift register (LT-LFSR) technique reduces transitions in two dimensions: between consecutive patterns and between consecutive bits within a pattern. The proposed architecture increases the correlation among the patterns generated by LT-LFSR with negligible impact on test length. An algorithm to synthesize

a built-in TPG from low power deterministic test patterns without inserting any redundancy test vectors is proposed in [68].

Generation of the Useful Vectors Only

A significant amount of energy is wasted in the LFSR and in the CUT by useless patterns that do not contribute to fault dropping. LFSR tuning modifies the state transitions of the LFSR such that only the useful vectors are generated according to a desired sequence. This phenomenon is used in [69, 70].

Filtering of Unnecessary Vectors

There are some non detecting sequences generated by LFSR. By inhibiting such vectors during testing, over all switching can be reduced and hence power consumption can be reduced. A test-vector-inhibiting technique to filter out some non detecting subsequences of a pseudorandom test set generated by an LFSR was extended in [71] by the filtering action to all the nondetecting subsequences. A scan cell architecture that decreases power consumption and the total consumed energy [72] is based on the data compression.

Circuit Partitioning

The circuit is strategically partitioned in to sub-circuits to achieve the parallel testing. An efficient scan partitioning technique reduces average and peak power in the scan chain during shift and functional cycles. Main goal here is to partition the circuit in to sub circuit so that parallel testing can be achieved. A novel low-power BIST strategy based on circuit partitioning is proposed in [73]. This strategy partitions the original circuit into two structural sub circuits so that two different BIST sessions can successively test each sub circuit. A low-power virtual test

partitioning technique [74] where faults in the glue logic between subcircuits can be detected by patterns with low power dissipation that are applied at the entire circuit level, while the patterns with high power dissipation can be applied within a partitioned subcircuit without loss of fault coverage.

Separate Testing Strategy for Memory

Various RAM transition reduction techniques by reordering read and write access are described for low power consumption. In [59], the authors suggested the strategy on RAM transition reduction by reordering the read and write accesses and the address counting scheme. These results in decrease of the energy consumption while keeping test time the same, so they also minimize the average power. A row bank-based precharge technique based on the divided word line (DWL) architecture is proposed in [75] for low-power testing of embedded SRAMs. In low-power test mode, instead of precharging the entire memory array, only the current accessed row bank is precharged. This will result in significant power saving for the precharge circuitry. A low power memory BIST is proposed in [76].

2.8.2 SAR Techniques for Design-For-Test

Apart from internal and external techniques, in these categories, some DFT techniques are described for low power testing. Here main goal is to reduce switching activity by adding some hardware as DFT. Two design-for-testability (DFT) techniques based on clock partitioning and clock freezing to ease the test generation process for sequential circuits is proposed in [75]. In the first DFT technique, a circuit is mapped into overlapping pipelines by selectively freezing different sets of registers so that all feedback loops are temporarily cut. In the second DFT technique,

selective clock freezing is used to temporarily cut only the global feedback loops. In a Jump scan technique (or J-scan) for low power testing [78], the J-scan shifts two bits of scan data per clock cycle so the scan clock frequency is halved without increasing the test time. The experimental data shows that the proposed technique effectively reduces the test power by two thirds compared with the traditional MUX scan.

2.8.3 SAR Techniques for External Testing

This category contains various techniques adopted to reduce the power consumption during external testing by ATE. These methods depend on the number of transitions in test data set. Either the ATPGs are trained to generate the test vectors with minimum transitions or the generated test vectors are explored to minimize the transitions. Sometimes, the internal switching activities are controlled using primary input vectors, many times, the partially specified test data is filled with specified bits to give minimum transitions. It means, these schemes generally deal with test data rather than modification in circuit architecture.

2.9 SAR Techniques for IP Cores

The main focus here is IP core based SoC testing. The power reduction techniques for Built-In-Self-Test deals with the architecture design of BIST. Mostly, this category focuses on exploring the structure of components of BIST architecture like LFSR. It means that this category is in any case not suitable to IP core based SoC where IPs are not allowed for any change at system integration level. The same case is for power reduction techniques with help of specific DFT insertion. Neither the insertion nor modification in scan chain is possible. In this regards, the first two classifications are not applicable to this particular case. The

only option remains for the given application is 'power reduction techniques for external testing' which is further explored in next section.

2.10 SAR Techniques for External Testing

The power reduction techniques described in literature which are used with external testing are further classified as follows:

2.10.1 Low Power ATPG Algorithms

The current research in this field focuses on ATPG algorithm which not only gives maximum fault coverage but ensures the maximum fault coverage at lowest possible power dissipation. In [79], a heuristic method to generate test sequences is proposed which create worst-case power droop by accumulating the high and low-frequency effects using a dynamically constrained version of the classical D-algorithm for test generation.

A novel scan chain division algorithm [80] that analyzes the signal dependencies and creates the circuit partitions such that both shift and capture power can be reduced when using the existing ATPG flows.

In [81], ATPG technique simultaneously reduces capture and shift power during scan testing. This ATPG performs power reduction during dynamic test compaction so the test length overhead is very small. A low capture power ATPG and a power aware test compaction method is presented in [82].

2.10.2 Input Control

Here the idea is to identify an input control pattern such that by applying that pattern to the primary inputs of the circuit during the scan operation, the switching activity in the combinational part can be

minimized or even eliminated. The basic idea of input control technique with existing vector-or latch-ordering techniques that reduces the power consumption has been covered in [59]. In the same area, [83] presented a technique of gating partial set of scan cells. The subset of scan cells is selected to give maximum reduction in test power within a given area constraint.

2.10.3 Ordering Techniques

In this technique, the test vectors are reordered to reduce the number of transitions and hence the test power. The problem of test vector reordering has been mapped into finding the Hamiltonian path in a fully-connected weighted graph which is similar to the traveling salesman problem (TSP). As there exists no polynomial time algorithm for TSP, approximation methods of solution have been used. In [84], a greedy algorithm is proposed to decrease test power consumption without modifying the initial fault coverage. A second technique based on Simulated Annealing (SA) has been proposed in which the greedy solution is used as initial solution and it shows a considerable average power reduction during test application. Another work [85] has also considered the Hamming distance minimization between adjacent vectors to reduce the dynamic power dissipation during testing. In [86], reduction of power dissipation during test application has been studied both for scan designs and for combinational circuits tested using built-in self-test (BIST). They have shown that heuristics with good performance bounds can be derived for combinational circuits tested using BIST and a post-ATPG phase has been proposed for reducing power dissipation during test application in full-scan circuits and for pure combinational circuits. They have shown that scan-latch ordering along with test-vector reordering can give considerable improvement in power dissipation and considerable savings can be obtained by repeating some of the test

vectors. In [85], an evaluation of different heuristic approaches has been done in terms of execution time and quality. In [87], a GA-based formulation is proposed to solve the problem of generating a test pattern set such that it has high fault coverage and low power consumption.
Hamming distance based reordering is described in [89]. In [90], another method based on Artificial Intelligence (AI) is proposed to order the test vectors for combinational circuits in an optimal manner to minimize switching activity during testing.

2.10.4 Don't Care Bit Filling

ATPG generated uncompacted test data contains a large number of don't care bits. In test vectors, only 1% to 5% of their bits are specified (care) bits. Such don't care bits in test data can be manipulated to reduce the test power. An automatic test pattern generation (ATPG) scheme for low power launch-off-capture (LOC) transition test is proposed in [91]. In [92], a Genetic Algorithm based heuristic to fill the don't cares is used. A segment-based X-filling to reduce test power and keep the defect coverage is presented in [93]. Based on the operation of a state machine, [94] elucidates a comprehensive frame for probability-based primary-input dominated X-filling methods to minimize the total weighted switching activity (WSA) during the scan capture operation. In [95, 96], the effect of don't care filling of the patterns generated via automated test pattern generators is described. It presents a trade-off in the dynamic and static power consumption.

2.11 External Testing SAR Techniques for IP Cores

Reviewing the characteristics of low power external testing techniques, the following points need to be considered.

Improved ATPG Algorithms

The implementation of this method

- deals with generation of new test set rather than available test sequence or test architecture
- requires the netlist of the design. It cannot be directly applicable to hard core
- requires the knowledge of internal details of design

Input Control

The implementation of this method

- deals with test architecture rather than test sequence
- requires the knowledge of internal details of design
- requires the additional hardware to modify test pattern sequence
- requires modification in internal structure

Ordering Technique

The implementation of this method

- deals with test sequence rather than test architecture
- requires the well defined test sequence i.e. test data set
- does not requires the knowledge of internal details of design
- requires the additional hardware to reorder test pattern sequence
- does not requires modification in internal structure

Exploring Don't Care Bits

The implementation of this method

- deals with test data bit sequence rather than test vector sequence or test architecture
- requires the well defined test sequence i.e. test data set
- does not require the knowledge of internal details of design
- does not require any additional hardware
- does not requires modification in internal structure

Out of the above mentioned four classifications, the 'low power ATPG algorithms' and 'input control' classifications require the internal details i.e. netlist of the design under test. Generally such details are not available with IP cores and hence, in context of IP core based SoC, these two classifications are not suitable. The remaining options for switching reduction in case of IP core based SoC testing are "ordering techniques" and "don't care bit filling" methods.

2.12 Methodology for Experiments

This section contains the methodology adopted for experimental work in this thesis and also the experimental results for test data compression with contemporary tool i.e. TestKompress from MentorGraphics.

2.12.1 Results for the Proposed Methods in This Thesis

The experiments are conducted on a workstation containing a 2.0 GHz Core2duo T5750 processor with 2.99 GB of RAM. For various methods described in thesis, the mathematical models are implemented using

MATLAB7.0 and then final implementation is done in C language for experimental work.

Unless specified differently, for the quantitative measurement of test data compression, test application time and number of transitions (for power calculations), the experiments were performed on test data compression for test cubes generated by MINTEST for the widely used six largest ISCAS'89 benchmark circuits. The author is thankful to Prof. Nur A. Touba for providing this test cubes generated by MINTEST for the widely used six largest ISCAS'89 benchmark circuits. The experiment results in mostly all reference papers refereed for result comparison in this thesis have used the same test sets. The details of these test sets for the six largest ISCAS benchmark circuits are given in Table 2.1.

Table 2.1: Details of MINTEST Generated Test Set for ISCAS'89 Circuits

ISCAC Circuit	# of Test Vectors	# of Bits/Vector	# of Total Bits	% of 0s	% of 1s	% of Xs
S5378	111	214	23754	12.4947	14.8901	72.6151
S9234	159	247	39273	12.2629	14.7302	73.0069
S13207	236	700	165200	3.8093	3.0387	93.1519
S15850	126	611	76986	9.5186	6.922	83.5593
S38417	99	1664	164736	14.0279	17.891	68.0811
S38584	136	1464	199104	9.2781	8.4448	82.2771

The equation 2.2 is used for all analysis and comparison of different methodologies as this definition of compression ratio is widely used in literature.

For the quantitative measurement of test power reduction, the total number of transitions (i.e. from '0-to-1' or '1 to 0') is considered for all analysis and comparisons.

For on-chip decoder area comparison, the decoder FSMs are implemented using VHDL. The EDA tools used are Mentor graphics HDL

designer for design entry, Modelsim for simulation and Leonardo spectrum for synthesis. The library used for synthesis is TSMC 0.35u (Taiwan Semiconductor Machine Corporation). For the results for area overhead of on-chip decoder, the parameters like equivalent NAND gates, nets, ports and maximum clock frequency are taken from synthesis report.

2.12.2 Comparisons of Results with Contemporary Profession Tool

In this thesis, the work is done on a ready to use test data for a given IP core where the internal structure is hidden and no ATPG or fault simulation is possible. Still to have a comparison of results with contemporary professional tool, the results from TestKompress from Mentror Graphics are presented here.

Table 2.2: Details of TestKompress Generated Test Set for ISCAS'89 Circuits

	# of Test Vectors	# of Bits/Vector	# of Total Bits
S5378	111	160	17760
S9234	80	135	10800
S13207	131	483	63273
S38417	109	1463	159467
S38584	133	1260	167580

TestKompress uses the Embedded Deterministic Test (EDT) technology [9]. In this technology, the compression algorithm is tightly integrated with the dynamic compaction of the ATPG engine. The linear equation solver works iteratively with ATPG to maximize compression. The Table 2.2 presents the size of test set generated by TestKompress. However, it should be noted that the TestKompress involves the use of ATPG and

requirement of netlist which is against the objective of this research work. Further, the test data generated by TestKompress is fully specified i.e. without any don't care bit.

2.13 Inferences

From the above literature survey, it could be inferred that

- The test data compression reduces the test time which results in test cost reduction. Out of wide range of test data compression methods, "run length code" and "statistical code" based compression methods are suitable to IP core based SoC.
- The same way, out of wide range of switching activity reduction techniques, "reordering of test vectors" and "don't care bit filling" are suitable to IP core based SoC.

For a common solution to test cost, test time and test power problems, a "run length code" or "statistical code" based test data compression method involving "test vector reordering" and "don't care bit filling" should be developed and it should be further optimized for overall test application time and on-chip area overhead.

Chapter 3

Run Length Code Based Test Data Compression

The ATPG generates binary test data with a large number of don't care bits. These don't care bits can be converted into either zeros or ones. Because of that the test data can form long runs of ones or zeros and hence can be efficiently compressed with a run-length code. This chapter includes the simple run-length code and all its variants for test data compression. The proposed test data processing methods to enhance the compression in case of run length based codes are included in this chapter.

3.1 Existing Run Length Codes

This section includes the overview and analysis of various run length based test data compression methods.

3.1.1 Overview of Existing Run Length Codes

This section includes the detailed descriptions of various run length codes like simple, Golomb code, Frequency Directed Run length (FDR) code, Extended FDR (EFDR) code, Modified FDR (MFDR) code,

Alternating Run Length code, Shifted Alternating Run Length coding, etc.

Simple and Modified Simple Run Length Code

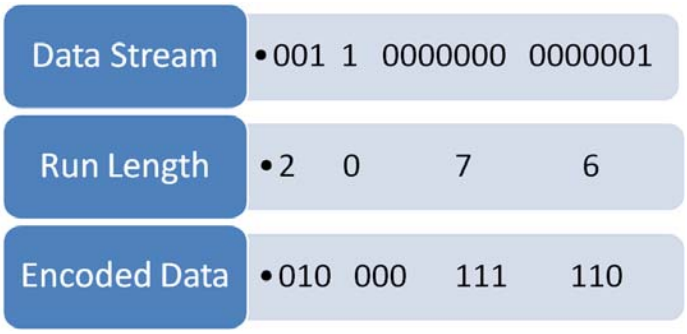

Figure 3.1: Example of Simple Run Length Coding

The example of a variable-to-block run-length code is shown in Table 3.1. A variable number of bits are encoded by a fixed number of bits. The example shown in Figure 3.1 follows the coding table where the fixed number of bits is 3. The simple run length code is inefficient when runs of 1s occur. To overcome this inefficiency, the modified simple run length code is proposed in [36] as shown in Table 3.1.

Table 3.1: Modified 3 - Bit Run length Code

Code Word	Symbol For 3–Bit Run Length Code	Symbol For Modified 3–Bit Run Length Code
000	1	10
001	01	11
010	001	01
011	0001	001
100	00001	0001
101	000001	00001
110	0000001	000001
111	0000000	000000

Golomb Code

The simple or modified run length codes are very efficient when probability of occurring '*0*' is very high compared to probability of

occurring '*1*' i.e. for long runs of '0s'. Unlike that scan test data, the probability of occurring 0 or 1 is same. Moreover, in such test data, the runs are generally short. So to make run length code more efficient for scan test data, the Golomb code [37, 38] is proposed. The coding table is shown in Table 3.2, Golomb codes encode runs of 0s with variable-length code words. The codewords are divided into groups of equal size *m* (*m* is any power of 2). Each group A_k is assigned a group prefix *(k -1)* 1s followed by a *'0'* and as each group contains uniquely identifiable symbols, the final code word consists of a group prefix and a tail of *n* bit which identifies the member in group. The Figure 3.2 demonstrates this coding method.

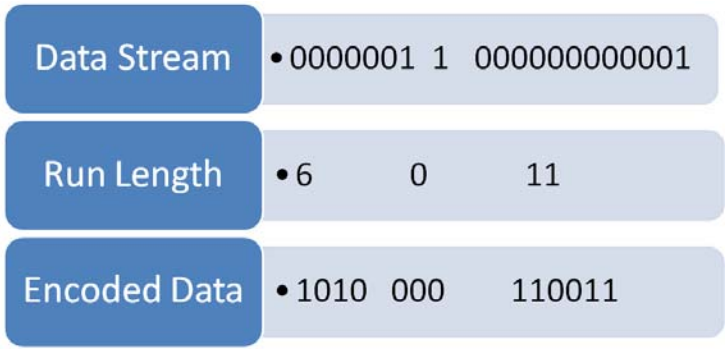

Figure 3.2: Example of Golomb Coding

Table 3.2: Golomb Code for m = 4

Group	Run Length	Group Prefix	Tail	Code word
A_1	0	0	00	000
	1		01	001
	3		11	011
A_2	4	10	00	1000
	5		01	1001
	-		--	----
	7		11	1011
A_3	8	110	000	110000
	9		001	110001

Frequency Directed Run Length Code (FDR)

Table 3.3: Frequency Directed Run Length Code

Group	Run Length	Group Prefix	Tail	Code word
1	0	0	0	00
	1		1	01
2	2	10	00	1000
	--		--	----
	5		11	1011
3	6	110	000	110000
	7		001	110001
	----		---	----

A new scheme 'Frequency Directed Run length coding' [39, 40] is proposed based on the observation that the frequency of runs of 0s with run length less than 20 is high and even within the range of 0 to 20, the frequency of runs of length l decreases rapidly with increasing l. Test data compression can be more efficient if the runs of 0s with shorter run length are mapped to shorter codewords. The FDR is similar to Golomb code but the difference is the variable group size. The size of the i^{th} group is equal to 2^i i.e. that group contains 2^i members. Table 3.3 demonstrates the codewords for different run length. The example in Figure 3.3 demonstrates this coding format.

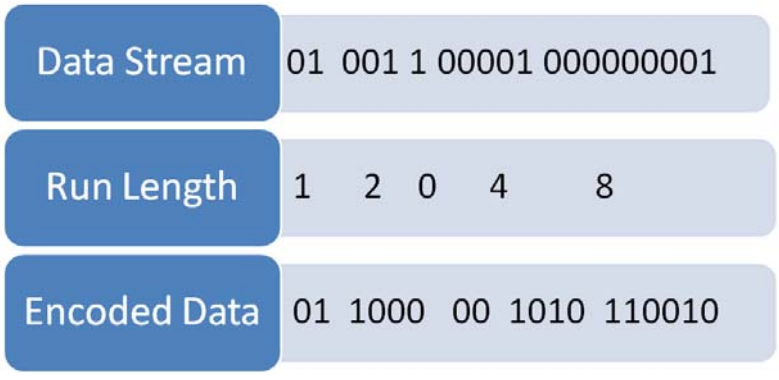

Figure 3.3: Example of Frequency Directed Run length Coding

Extended FDR Code (EFDR)

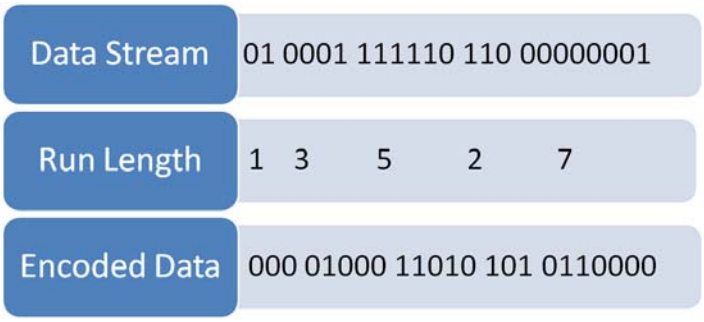

Figure 3.4: Example of Extended FDR Coding

The FDR code is very efficient for compressing data that has few 1s and long runs of 0s but inefficient for data streams that are composed of both runs of 0s and runs of 1s. Generally test vectors contain 0s and 1s in group i.e. there will be a run of 1s followed by run of 0s and vice versa. An extension of FDR is EFDR [41]. Here the '*run of 0s followed by bit 1*' and '*run of 1s followed by bit 0*' are coded same way as FDR but adding an extra bit at beginning of FDR codeword. Codewords for this method are shown in Table 3.4. The sample of encoded data is given in Figure 3.4.

Table 3.4: Extended Frequency Directed Run Length Code

Group	Run Length	Group Prefix	Tail	Code word Runs of 0s	Code word Runs of 1s
1	1	0	0	000	100
	2		1	001	101
2	3	10	00	01000	11000
	4		01	01001	11001
	5		10	01010	11010
	6		11	01011	11011
3	7	110	000	0110000	1110000
	8		001	0110001	1110001
	----		---	----	----
	14		111	0110111	1110111

Alternating FDR Code

The FDR code is very efficient for compressing data that has few 1s and long runs of 0s. However, for data streams that are composed of both runs of 0s and runs of 1s, the FDR code is rather inefficient. In fact, the sizes of encoded test sets obtained for such test sets were larger than the sizes of uncompressed test sets. The alternating FDR (AFDR) [42] follows the same coding scheme as FDR but it uses the runs are of both types, i.e. runs of '0s followed by one 1' and runs of '1s followed by one 0'. It is assumed that the data starts with run of 0 and then it keeps alternating between runs of 0 and 1. If the first run is not of 0 than a code word of indicating 0 run length should be prefixed. It has been made clear in Figure 3.5.

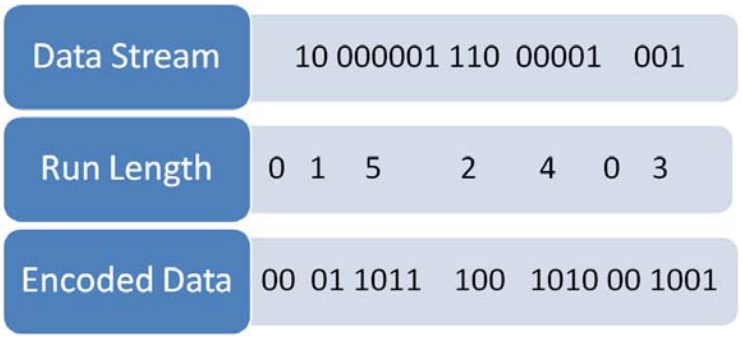

Figure 3.5: Example of Alternating FDR Coding

Shifted Alternating FDR Code (SAFDR)

Table 3.5: Shifted Alternating Frequency Directed Run Length Code

Group	Run Length	Group Prefix	Tail	Code word
1	1	0	0	00
	2		1	01
2	3	10	00	1000
	4		01	1001
	--		--	----
	6		11	1011
3	7	110	000	110000

The AFDR is inefficient for data containing the large number of short runs [43]. The evolution in AFDR is Shifted Alternating Run Length based FDR (SAFDR) [43].

In this coding format, the runs contain only 0s or only 1s. It does not contain any post-amble bit. Moreover, the data can start with any type of run. To indicate the type of first run, the encoded data will be preceded by a special bit. This bit will be 0 (or 1) if the first run type is '0s followed by 1' (or '1s followed by 0'). This coding format follows FDR coding format as shown in Table 3.3. As, in SAFDR, there will be no run-length of 0 sizes, the code word of 0 of FDR scheme is assigned to run length size 1 and so on each code word is shifted to one position higher as shown in Table 3.5. This helps in achieving higher compression compared to AFDR which is demonstrated in Figure 3.6 by encoding same data used in Figure 3.5.

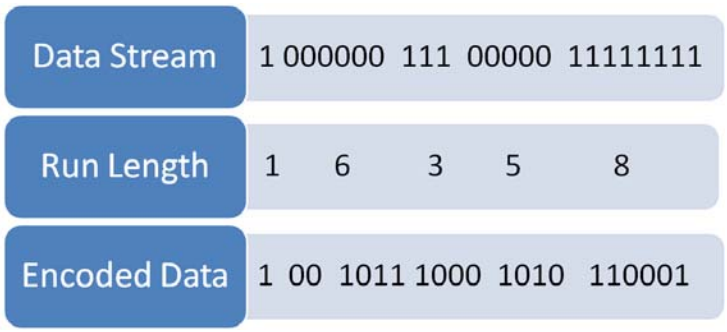

Figure 3.6: Example of Shifted Alternating FDR Coding

Modified FDR Code (MFDR)

In MFDR (Modified Frequency-Directed Run-length), groups of FDR is further modified to give better compression ratio than FDR if the probability of 0s in the test set is greater than 0.8565 [45].

Table 3.6: Modified Frequency Directed Run Length Code

Group	Run Length	Group Prefix	Tail	Code word
A_1	0	01	00	000
	1		01	001
	3		11	011
A_2	4	10	00	1000
	-		--	----
	7		11	1011
A_3	8	001	00	11000
	--		--	--
	11		11	11011
A_4	12	110	000	110000
	-		-	-
	19		111	110111
A_5	20	0001	000	0001000
	-		-	-

Variable Length Input Huffman Code

Few other variants of run length codes are developed using a combination of run length and statistical code. The FDR code requires a complicated decoder with higher area overhead. Therefore, regardless of the good compression ratios the area overhead of FDR is a disadvantage. A mix of Huffman and FDR 'Variable Length Input Huffman Code (VIHC)' is proposed which instead of fixed-length pattern uses variable-length pattern as input to the Huffman algorithm [24]. In this method, the compression ratio is retained because of FDR and the area overhead is reduced using selective Huffman Coding. Spilt VIHC (SVIHC) [44] code demonstrates that before going to the VIHC, if the test data set is divided in to two or more equal parts and the vectors are reordered in a certain way, the compression ratio can be still improved.

Fixed Plus Variable Length Run Length Code (FPVL)

A test data compression based on Fixed-Plus-Variable-Length (FPVL) coding [49] divides code word into two parts: fixed length head section and variable-length tail section. The value presenting tail is the length of

runs in the original test data plus *'2'*. In order to obtain further compression, the highest bit of the tail section is reduced from the code words because all of the highest bits in the tail section of the tail are '1'. The Table 3.7 shows codewords.

Table 3.7: Fixed Plus Variable Length Run Length Code

Group	Run Length	Head (K=2)	Tail	Code word
1	0	00	10	000
	1		11	001
2	2	01	100	0100
	-		---	----
	5		111	0111
3	6	10	1000	10000
	----		----	----
	13		1111	10111
4	14	11	10000	110000
	----		-----	-----
	29		11111	111111

3.1.2 Analysis of Existing Run Length Codes

The existing run length codes are verified by the implementation of the codes using MATLAB. The corresponding on-chip decoder area overhead is derived using the VHDL coding and Leonardo Spectrum synthesizer with TSMC 0.35u library. The compression, on-chip area overhead and test power in terms of number of switching is as follows:

Compression

For the various ISCAS'89 bench mark circuits, the % compression is shown in Table 3.8. These results clearly advocate the Extended FDR code. For cases where the probability of 0s in the test set is very high, MFDR gives the better results.

Table 3.8: Comparison of % Compression for Run Length Based Codes

ISCAS'89 Circuits	Original Test Data	Golomb Code	FDR	EFDR	Alternating FDR	MFDR	FPVL
S5378	23754	40.70	48.02	51.93	-NA-	51.47	52.15
S9234	39273	43.34	43.59	45.89	44.96	57.74	45.82
S13207	165200	74.78	81.30	81.85	80.23	83.42	81.58
S15850	76986	47.11	66.22	67.99	65.83	66.93	67.70
S38417	164736	44.12	43.26	60.57	60.55	57.95	43.06
S38584	199104	47.71	60.91	62.91	61.13	59.32	32.29

On-chip Decoder

Table 3.9 shows the comparison of hardware requirement. Being the simplest one, the Golomb decoder requires minimum area overhead.

Table 3.9: Comparison of On-chip Decoder Area for Run Length Codes

Coding scheme	# of Equivalent NAND gates	# of ports	# of nets	Maximum Clock Frequency (MHz)
GOLOMB	580	3	361	29.7 MHz
FDR	1104	3	797	24.6 MHz
EFDR	1078	3	757	13.9 MHz
ASFDR	1111	3	799	11.3 MHz
MFDR	2199	3	1493	11.0 MHz

3.2 The Proposed Run Based Bit Filling Method

This section contains the proposed optimizations for existing run length codes.

3.2.1 Problem Formulation

During the survey and analysis of various run length codes discussed in Section 3.1, it is observed that the existing run length codes can be

broadly classified into five categories on the basis of types of runs formed. These five categories are listed below:

Type-A. These codes consider the runs of ‘zeros followed by one’ like simple run length code, Golomb Code, FDR Code, MFDR Code

For example: 001 01 000001 . . .

Type-B. These codes consider the runs of ‘ones followed by zero’. This is a proposed case. No such code is proposed in literature but we have taken this case for comparison purpose. This code gives very promising results for data with probability of occurrence of bit '1' is much higher compared to probability of occurrence of `0'can be more efficient

For example: 1110 10 111110 10. . .

Type-C. These codes consider runs of ‘zeros followed by one’ as well as runs of ‘ones followed by zero’ like Extended FDR

For example: 001 1110 10 000001 01 11110 1110. . .

Type-D. These codes consider the alternating runs of ‘only zeros’ and ‘only ones’ (with-out any post-ambled bit) like Shifted Alternating FDR

For example: 00 1111 0000 11 00000 111 0 . . .

Type-E. These codes consider alternating runs of ‘zeros followed by one’ and runs of ‘ones followed by zeros’ like Alternating FDR

For example: 001 1110 0001 10 00001 110 . . .

Here it should be noted that even though the different authors have followed the different types of runs, the don't care bit filling method adopted in majority of the cases is replacing don't care bits with ‘0’.

To get the maximum compression, the don't care bits should be filled such that it supports the type of run and elongates the run. So instead of filling the don't care bits blindly with ‘0’ only, the don't care bits should be filled with 1s or 0s depending up on the different type of run.

So here the following five different methods for don't care bit filling on the basis of run type are proposed.

Method: A *X Filling for Codes Considering Runs of '0s followed by 1'*

For codes like Golomb, FDR and MFDR, the symbols are made of runs of '0s followed by 1'. So a simple technique for replacing all don't care bits with 0s is adopted. It will increase the run length of 0s resulting into increase in the data compression

Method: B *X Filling for Codes Considering Runs of '1s followed by 0'*

This is a new case introduced to analyze the compression results. The symbols are made of runs of '1s followed by 0'. Here all Xs are replaced by 1s. So the runs of 1s will increase.

Method: C *X Filling for Codes Considering Runs of '0s followed by 1' as well as Runs of '1s followed by 0'*

The code EFDR is a case which accepts both the types of runs. The X filling must be done in such a way that it should not only maximize the run length but it should not introduce any new symbol also. While filling the X, the logic applied is that if the run has ended just before the position of X, the X should be filled with reference to next run. But if there is a continuous run going on at position of X, X should be filled such that it increases the run length of ongoing run. Proposed algorithm needs to do forward tracking as well as backward tracking.

Method: D *X Filling for Codes Considering Alternating Runs of 'only 0s' and 'only 1s'*

For code like SAFDR, the runs are made of alternating runs of 0s and 1s without any post-ambled bit. So in this case, all the Xs are replaced with last non-X value i.e. don't cares

following a run of zeros are mapped to zeros, and don't cares following a run of ones were mapped to ones to minimize the number of runs. If test set starts with don't care bits, all such initial don't care bits are replaced with the first specified (fixed) bit.

Method: E *X Filling for Codes Considering Alternating Runs of '0s followed by 1' and Runs of '1s followed by 0'*

For code like AFDR, the symbols are made of alternating runs of '0s followed by 1' and '1s followed by 0'. Here the first run must be of zero type. So all the initial 'Xs' before the first specified bit comes, will be replaced by 0s. If first bit is '1', because of the absence of first '0s followed by 1', it is considered that the first run is of zero length. After that, all the X bits are filled with last non-X value.

3.2.2 Entropy Based Maximum Compression Limit

For any data compression method, the entropy is an important concept used to calculate the maximum possible limit of % compression that can be achieved by that compression code. The entropy of a symbol *E(S)* is the minimum number of bits needed to encode that symbol. The entropy of the test set is calculated from the probabilities of occurrence of unique symbols using the formula in equation 3.1

$$Entropy\ of\ a\ data\ set\ E(S) = \sum_{i=1}^{n} P_i * log_2 \frac{1}{P_i} \quad \text{..... (3.1)}$$

where *Pi* is the probability of occurrence of symbol Xi in the test set and *n* is the total number of unique symbols.

In case of test data compression methods for which symbol length is fixed, the formula for maximum compression based on entropy in percentage that can be achieved is given by equation 3.2 [96].

$$Max.Compression\ (for\ fixed\ symbol\ length) = \frac{Symbol\ length - entropy}{symbol\ length}\ X\ 100$$

..... (3.2)

and in case of test data compression methods for which where symbol length is variable, the formula for entropy based maximum compression in percentage that can be achieved is given by equation 3.3

$$Max.Compression\ (for\ variable\ symbol\ length) = \frac{Average\ Symbol\ length - entropy}{Average\ symbol\ length}\ X\ 100$$

.....(3.3)

$$Average\ Length = \sum_{i=1}^{n} P_i\ *|X_i|$$

.....(3.4)

where *Pi* is the probability of occurrence of symbol *Xi*. $|Xi|$ is the length of symbol *Xi* and *n* is the total number of unique symbols.

Mathematically it can be proved that the formula in equation 3.5 is a generic formula for maximum compression and is valid for any type of symbol length.

$$Max.Possible\ Compression = \frac{T - (S * E)}{T}\ X\ 100$$

.....(3.5)

where *T* is the total number of bits in original uncoded test data, *S* is the total number of symbols needed to be encoded and *E* is the entropy. For

all further discussion, the formula in equation 3.5 is used for estimation of maximum limit of % compression based on entropy.

3.2.3 Experimental Results

Implementation Steps

For this proposed bit filling algorithm, the following steps are applied:

----------Algorithm For Don't Care Bit Filling----------

Step: 1 Filling the unspecified bit corresponding to specific run type

Step: 2 Finding the difference vector for successive test vectors (where ever necessary for comparison with existing results)

Step: 3 Calculating the length of runs and frequency of occurrence of each distinct run length.

Step: 4 Calculating the total number of runs i.e. symbols S.

Step: 5 Calculating entropy and hence maximum limit of % compression using equations 3.1 and 3.5 successively.

The max possible compressions in Table 3.10 are calculated from the exact values of entropy that were generated after the X filling.

Table 3.10: % Compression for Proposed Bit Filling Methods

ISCAS'89 Circuit	Test Data Bits	% Compression for Various X Filling Methods A, B, C, D, E				
		A	B	C	D	E
S5378	23754	52.36	54.33	**56.38**	40.73	38.94
S9234	39273	47.80	44.34	**53.37**	35.92	35.33
S13207	165200	83.65	81.12	**85.55**	79.72	79.59
S15850	76986	68.18	62.49	**71.90**	60.72	60.32
S38417	164736	54.50	57.44	**65.84**	55.77	55.14
S38584	199104	62.49	60.09	**66.67**	54.65	54.21

Table 3.11: Total # of Symbol for Proposed Bit Filling Methods

ISCAS'89 Circuits	Total No. of Symbols Needed to be Encoded				
	A	B	C	D	E
S5378	3538	2969	**2094**	3129	3130
S9234	4817	5786	**3216**	4904	4905
S13207	5021	6294	**3550**	5427	5428
S15850	5330	7329	**3647**	5628	5629
S38417	29473	23110	**9548**	13751	13752
S38584	16814	18474	**10771**	16275	16275

Table 3.12: Total # of Distinct Symbols for Bit Proposed Bit Filling Methods

ISCAS'89 Circuits	Total No. of Distinct Symbols				
	A	B	C	D	E
S5378	79	87	**141**	131	136
S9234	74	86	**141**	133	134
S13207	211	178	**331**	314	314
S15850	173	145	**234**	230	229
S38417	121	154	**236**	233	238
S38584	214	191	**345**	333	333

Table 3.13: Entropy for Proposed Bit Filling Methods

ISCAS'89 Circuits	Entropy				
	A	B	C	D	E
S5378	3.20	3.65	**4.95**	4.50	4.63
S9234	4.26	3.78	**5.69**	5.13	5.17
S13207	5.38	4.95	**6.72**	6.17	6.21
S15850	4.60	3.94	**5.93**	5.37	5.42
S38417	2.54	3.03	**5.89**	5.29	5.37
S38584	4.44	4.30	**6.16**	5.54	5.60

Table 3.14: Scan-In Power for Proposed Bit Filling Methods

ISCAS Circuit	Peak Power				Average Power			
	A	B	C	D	A	B	C	D
S5378	12085	12375	**11732**	11522	4300	4087	**3524**	3526
S9234	15395	15640	**14092**	14103	6706	6521	**4002**	4022
S13207	110129	126820	**94879**	94886	12318	1453	**8073**	7887
S15850	84360	88794	**70875**	70894	19448	25636	**13611**	13659
S38417	514716	539019	**437884**	437935	194843	193140	**118100**	118080
S38584	530464	533975	**481158**	481171	133320	142220	**86135**	86305

3.2.4 Observations and Analysis

For the ISCAS'89 benchmark circuits, Table 3.10 and 3.11 compares the % compression and the total number of symbols for various X filling method respectively. For method C, the total number of symbols needed to be encoded is minimum compared to remaining methods, hence % compression is maximum in this case. However, as it can be shown in Table 3.13, for the same method C, the entropy is maximum compared to other methods. The reason of higher entropy is the higher number of distinct symbols shown in Table 3.13. It means, in spite of higher number of distinct symbols and higher value of entropy compared to other method, the method C results in to maximum compression as the number of total runs are minimum compared to remaining methods A, B, D and E.

As seen in Table 3.10, the maximum % Compression is achieved when the bit filling is done with Type C and the minimum % compression is achieved when the bit filling is done with Type E. The % improvement with Type C compared to Type E is shown in Table 3.15.

Table 3.15: Improvement in % Compression for Run Based Bit Filling Methods

ISCAS'89	% Compression		% Improvement in % Compression
Circuit	Type C	Type E	
S5378	56.38	38.94	44.79
S9234	53.37	35.33	51.06
S13207	85.55	79.59	7.49
S15850	71.9	60.32	19.20
S38417	65.84	55.14	19.41
S38584	66.67	54.21	22.98

The Type C method consider runs of 'zeros followed by one' as well as runs of 'ones followed by zero'. Here, the runs are not necessarily in alternating sequence. While Type E method consider 'zeros followed by one' and runs of 'ones followed by zeros' in alternating sequence only.

For example:

The test data set 000001111011111000001001 will be interpreted by Type C and Type E methods as follow:

In case of Type C:

Runs:	000001	1110	111110	00001	001
Run Type :	0	1	1	0	0
Run length:	5	3	5	4	2

In case of Type E:

Runs:	000001	1110	*	111110	00001	*	001
Run Type:	0	1	**0**	1	0	**1**	0
Run length:	5	3	**0**	5	4	**0**	2

The run formation in Type C method is such that there can not be any run with length zero. Hence, the code word for zero run length need not be added in compressed data.

While the run formation in Type E method is such that it also need to introduce the runs of length zero to maintain the alternating sequence (as shown with bold letters in the above example). For such runs with length zero, the corresponding code word related to length zero needs to be added in compressed data. It means that the runs with zero length in Type E cause the code word overhead and reduce the % compression. Such reduction in % compression is appreciably large when the test data is having a large number of stand alone “zero” sandwiched between “ones” (e.g. 111101111) or stand alone “one” sandwiched between “zeros”(e.g. 00000010000).

3.3 The Proposed HDR-CBF-DV Method

This section introduces the proposed ‘Hamming Distance Based Reordering - Columnwise Bit Filling – Difference Vector’. The following subsections describe its various aspects.

3.3.1 Problem Formulation

From the analysis in section 3.1, it can be seen that the run length codes like Simple, Golomb, FDR, EFDR, MFDR and SAFDR uses the runs of 0s. If for a given test set, we can generate long run of 0s, it would result into higher test data compression and more scan-power reduction also. To increase the runs of 0s, the very first approach is to fill all the don't care bits with 0s. The maximum achievable test data compression based on entropy using this approach for various ISCAS'89 circuit is given in column-A of Table 3.10. To go beyond this limit is not possible for the given data. However, if the don't care bits are filled differently and data is reordered, it is another form of the same data, and for such data, the compression can be increased. It means that, the test data must be processed differently compared to existing methods.

The proposed method "Hamming Distance Based Reordering (HDR) and Columnwise Bit Filling (CBF) with Difference Vector (DV)" is the method to prepare the test data set for long run of 0s before the run length based codes are applied to it. In this thesis, it is sometimes referred as 1-Dimensional (1-D) method of reordering also because it reorders the positions of test vectors only. Here the bits within the test vectors are not reordered.

3.3.2 Hamming Distance Based Reordering

The following observations are considered for reordering:

a. The ATPG derived test set for given DUT contains the test patterns with a large amount of don't care values.
b. Stuck at faults based test patterns can be reordered without any loss of fault coverage. Here only thing that needs to be taken care is: the corresponding fault free outputs that are stored in ATE as golden references must be also reordered in the same sequence.

To calculate the Hamming distance between two test vectors, the following definitions are used.

Incompatible & Compatible

Given two bits $i, j \varepsilon \{0, 1, X\}$, i and j are incompatible if $i = 0$ and $j = 1$ or vice versa. All other circumstances are compatible as shown in Table 3.16.

Table 3.16: Compatibility of Bits

Type of Bit	1	0	X (don't care)
1	Compatible	Incompatible	Compatible
0	Incompatible	Compatible	Compatible
don't care(X)	Compatible	Compatible	Compatible

Distance

The distance between two scan vectors is equal to the number of corresponding incompatible bits. This definition is similar to Hamming distance with extension of don't-care bits

Total Hamming Distance (THD)

Total Hamming Distance (THD) is defined as sum of Hamming distance between successive test vectors in the sequence. Let Hamming distance $d[t_i, t_j]$ be the total number of changes between i^{th} and j^{th} test vector. The Total Hamming Distance (THD) for the whole test vector set is calculated by the following relation.

$$THD = \sum_{i=1}^{n} d[t_i, t_{i+1}]$$

.....(3.6)

where n represents total number of test vectors in the whole set.

For example given below, two vectors t_1 = *(10XX01)* and t_2 = *(001X11)*, the distance $d(t_1, t_2)$ is 2 because the first and the fifth corresponding bits in the vectors are incompatible.

Selection of First Test Vector of Reordered Set

For the selection of first test pattern, the heuristic that is applied in this scheme is '*Hardest Path First*'. The test pattern with minimum don't cares will be selected as the first test pattern of reorder list. The reason for selecting the test pattern with minimum don't care bits is that there is a minimum flexibility for bit filling. If more than one test pattern have minimum don't care bits than any test pattern with minimum don't care bits can be selected. Following example illustrates the selection procedure. The test set T contains five test patterns of 14 bits each.

	Test patterns	# of X
O_1	1 X 1 0 0 X X 0 1 X 0 0 X 1	5
O_2	1 1 1 X 0 X 0 X 1 0 X 0 X X	6
O_3	1 0 1 1 0 X 0 0 X X X 0 1 0	4
O_4	0 X X 0 X X 1 0 X X X 0 X X	9
O_5	1 0 1 X 1 X 1 X 1 0 X 0 0 X	5

Here the third test pattern from original list (O_3) is selected as first test pattern of reordered new list (N_1).

Hamming Distance Based Reordering

From remaining test patterns, the pattern with minimum Hamming distance from first reordered pattern will be selected as next reordered pattern. The logic behind the selection of pattern with minimum Hamming distance as the next reordered pattern is: when the further bit filling and difference vector will be done, this new sequence will generate maximum zeroes.

If each test pattern is taken as a vertex in a complete undirected graph G, and the distance between two patterns as the weight of an edge, then the problem of reordering the test pattern is similar to Hamilton problem, which is NP-hard and solved by various greedy algorithms.

The simplest pure greedy algorithm is: choosing as the next pattern in a path the one that is closest to the current pattern, provided it hasn't been visited yet. It seems that the Hamilton path of G is the solution to this reordering problem.

Now from the first pattern of new list (N_1), the Hamming distances for remaining all vectors O_1, O_2, O_4 and O_5 from N_1 are 2, 1, 3 and 2 respectively. As the O_2 has minimum Hamming distance from N_1, we will put O_2 at second position of reordered set as N_2. Now remaining vectors will be compared with N_2 and the vector with minimum Hamming distance will be placed as N_3. This process will be continue still the original test vector list is empty. The reordered test set is as follow:

N_1 1 0 1 1 0 X 0 0 X X X 0 1 0

N_2 1 1 1 X 0 X 0 X 1 0 X 0 X X

N_3 1 X 1 0 0 X X 0 1 X 0 0 X 1

N_4 0 X X 0 X X 1 0 X X X 0 X X

N_5 1 0 1 X 1 X 1 X 1 0 X 0 0 X

The overall procedure to minimize the difference bits in consecutive vectors during testing is as follows:

----------Algorithm for Hamming Distance Based Reordering----------

Step: 1 Consider a digital circuit with *f* scan flipflops, *p* inputs and *q* outputs.

Step: 2 Generate all the test vectors to detect all the single stuck at faults of the circuit. Let the number of test vectors be *n*.

Step: 3 Find the Hamming distance between each and every test vectors and load the same in array H_d of size *n x n*. Let $H_d[i][j]$

be the array elements which gives Hamming distance between i^{th} and j^{th} test vectors.

Step: 4 Apply reordering algorithm to find the reordered test vector sequence with minimum total Hamming distance.

The reordering algorithm used in step 4 is as follows:

---------- Reordering Algorithm----------

The various parameters used in the algorithms are as follows:

$t_1, t_2, \ldots\ldots t_n$ be n test vectors with m bits in each vector.

$T=\{ t_1, t_2, \ldots t_k \ldots t_n\}$ where k represents k^{th} position in the vector set generated by ATPG.

R is a set to store reordered test vector sequence. *bits_init[ ]* is an array containing the value of number of don't care bits in each test vector. Q is a set to store T-R.

Step: 1 Select a test vector x such that *bits_init[x]* is minimum in the array *bits_init* which contains the number of don't care bits in each vector.

Step: 2 Add x to set R.

Step: 3 Select a test vector y such that $H_d[x][y]$ is minimum in the array.

Step: 4 Add y to R; $Q \leftarrow T$-R; $x \leftarrow y$.

Step: 5 From the array $H_d[x][j]$ when j varies as in Q, find y so that $h_d[x][y]$ is the smallest value.

Step: 6 Go to step 3.

Step: 7 In the step 4, if $H_d[x][j]$ has more than one smallest value, then select any one value randomly.

Finally the set R will have reordered test vector sequence which will produce maximum zeroes after difference vector.

3.3.3 Columnwise Bit Filling

The bit filling of first vector will be done depending up on the type of run length code going to be used with the test data. For the don't care bit filling, method 'A' will be used for the codes like Golomb, FDR or MFDR; the method 'C' will be used for EFDR, and method 'D' will be used for SAFDR.

For the second test patterns and onwards, the don't care bit will be filled with the same value either '1' or '0' which its upper vector has at the same bit position. In given example, the first vector '10110X00XXX010' will be '10110000000010' after bit filling.

Now for the second test pattern, the bit filling will be like this.

Bit filled 1st vector : 1 0 1 1 0 0 0 0 0 0 0 0 1 0

2nd Vector : 1 1 1 X 0 X 0 X 1 0 X 0 X X

Bit Filled 2nd Vector : 1 1 1 1 0 0 0 0 1 0 0 0 1 0

After bit filling, the vectors for given example are as follow:

Reordered Bit Filled Test Set N_{RB}

1 0 1 1 0 0 0 0 0 0 0 0 1 0

1 1 1 1 0 0 0 0 1 0 0 0 1 0

1 1 1 0 0 0 0 0 1 0 0 0 1 1

0 1 1 0 0 0 1 0 1 0 0 0 1 1

1 0 1 0 1 0 1 0 1 0 0 0 0 1

3.3.4 Difference Vector

The next step is to take the difference vector of two consecutive vectors. This will further increase the numbers of zeroes and hence data compression. Let $T_D = \{t_1; t_2; ....; t_n\}$ be the ordered precomputed test set. T_{diff} is defined as follows: $T_{diff} = \{d_1; d_2; . . . ; d_n\} = \{ t_1; t_1 \oplus t_2; t_2 \oplus t_3 ;.....;$

$t_{n-1} \oplus t_n$ }; where a bit-wise exclusive-or operation is carried out between patterns t_i and t_{i+1}. The successive test patterns in a test sequence often differ in only a small number of bits. Therefore, T_{diff} contains few 1s and it can be efficiently compressed using any run length code.

For given example: The first vector will be as it is: 10110000000010

The next vector will be difference of first and second vector.

1st vector	: 1 0 1 1 0 0 0 0 0 0 0 0 1 0
2nd Vector	: 1 1 1 1 0 0 0 0 1 0 0 0 1 0
Difference Vector	: 0 1 0 0 0 0 0 0 1 0 0 0 0 0

The difference vector set for given example is as follow:

Reordered Bit Filled Difference Test Set N_{RBD}

1 0 1 1 0 0 0 0 0 0 0 0 1 0

0 1 0 0 0 0 0 0 1 0 0 0 0 0

0 0 0 1 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 1 0 0 0 0 0 0 0

1 1 0 0 1 0 0 0 0 0 0 0 1 0

3.3.5 On-Chip Decoder

While considering the on-chip area overhead in case of proposed method, it should be noted that the comparison is between 'simple data encoded using a run length code' and 'proposed HDR-CBF-DV method based preprocessed data encoded using the same run length'.

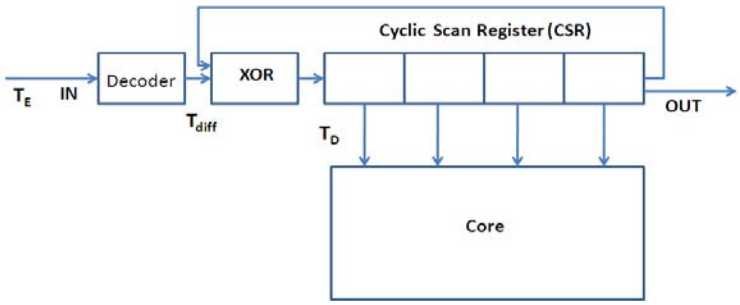

Figure 3.7: On-Chip Decoder for Difference Vector

It means that the same on-chip silicon decoder used for that particular run length code can be used in case of proposed method also. The only area overhead in case of the proposed scheme is because of difference vector. The decoder needs an extra mechanism for difference vector [36]. This will add a little area penalty only i.e. extra area for one XOR gate and feed-back mechanism as shown in Figure 3.7.

3.3.6 Experimental Results

After the test data set is processed with the proposed 'HDR_CBF_DV', this preprocess data is applied various code based scheme like FDR, Golomb, EFDR, MFDR and SAFDR. The detailed description of codes is available in Section 3.1 of this chapter.

The Table 3.17 and Table 3.18 compare the compressions of following three cases:

1. % compression claimed in literature for corresponding run length method i.e. FDR, EFDR, SAFDR, etc.
2. Maximum Possible % Compression based on entropy for unprocessed test data
3. % compression in case of test data processed with HDR-CBF-DV method.

Table 3.17: % Compression of FDR, Golomb, MFDR and HDR-CBF-DV method

ISCAS'89 Circuits	% Compression						
	XOR +FDR [40]	HDR-CBF-DV +FDR	Golomb [38]	HDR-CBF-DV +Golomb	MFDR	HDR-CBF-DV +MFDR [45]	Max. Limit for Unprocessed Data
S5378	48.02	**62.33**	40.70	**52.97**	51.47	**57.26**	***52.36***
S9234	43.59	**61.06**	43.34	**56.05**	57.74	**60.58**	***47.80***
S13207	81.30	**87.47**	74.78	70.03	83.42	**87.80**	***83.65***
S15850	66.22	**72.84**	47.11	**62.55**	66.93	**72.82**	***68.18***
S38417	43.26	**66.18**	44.12	**56.09**	57.95	**62.46**	***54.50***
S38584	60.91	**64.79**	47.71	**55.87**	59.32	**61.83**	***62.49***

Table 3.18: % Compression of EFDR, SAFDR and HDR-CBF-DV Method

ISCAS'89 Circuits	% Compression					
	EFDR[41]	Max. Limit for Unprocessed Data (Method C)	HDR-CBF-DV +EFDR	SAFDR[43]	Max. Limit for Unprocessed Data (Method D)	HDR-CBF-DV +SAFDR
S5378	51.93	56.32	**60.03**	49.95	51.44	**57.36**
S9234	45.89	53.37	**57.56**	45.14	47.26	**53.23**
S13207	81.85	85.54	**86.40**	80.11	82.45	**85.16**
S15850	67.99	71.90	**70.43**	65.63	67.25	**67.53**
S38417	60.57	65.84	**65.67**	60.52	62.93	**62.18**
S38584	62.91	66.67	**63.10**	61.09	62.23	**59.29**

The same comparisons are shown graphically in Figure 3.8 and 3.9.

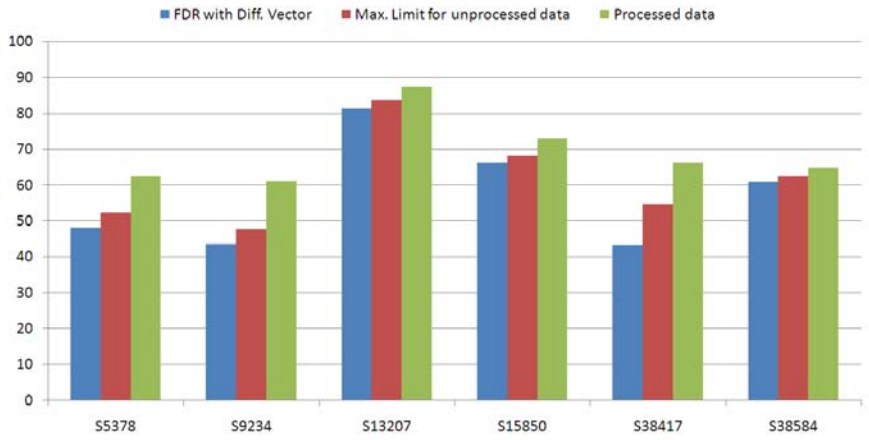

Figure 3.8: Improvement in % Compression in FDR by Proposed HDR-CBF-DV

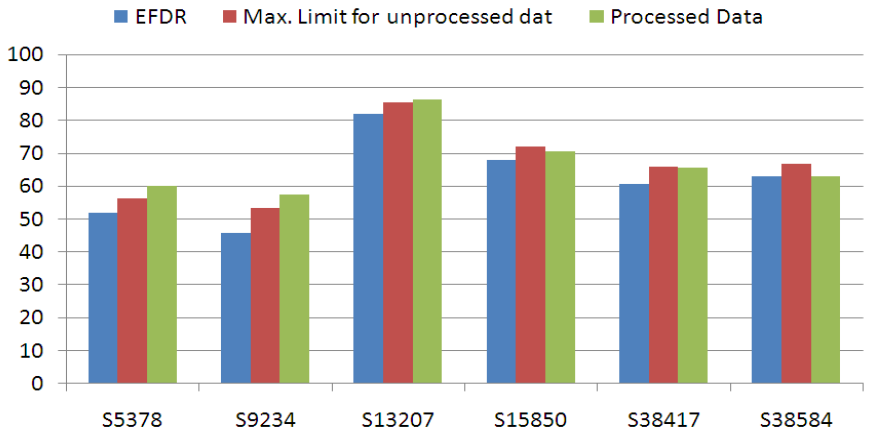

Figure 3.9: Improvement in % Compression in EFDR by Proposed HDR-CBF-DV

3.3.7 Observations and Analysis

From the experimental results in Tables 3.17 and 3.18, it can be observed that the proposed HDR-CBF-DV method of the test data processing helps in increasing the compression of test data. For the test data processed with the proposed method, the % compression is more than the entropy based maximum possible compression predicted for unprocessed data.

For unprocessed data, as shown in Table 3.17 and 3.18, the maximum % compression is achieved for EFDR coding and minimum % compression is achieved for Golomb coding. Table 3.19 represents the improvement in compression when the test data is preprocessed with the proposed HDR-CBF-DV scheme. The Column II of Table 3.19 shows the relative % improvement in % compression when the data is processed with HDR-CBF-DV scheme and compressed with FDR coding compared to unprocessed row data compressed with FDR coding. Similarly, Columns III to VI represent the relative % improvement for Golomb, MFDR, EFDR and SAFDR coding. Table 3.19 shows that the proposed test data processing scheme enhances the % compression for each of the coding method like FDR, Golomb, MFDR, EFDR and SAFDR.

Table 3.19: Relative % Improvement in % Compression with HDR-CBF-DV Method

ISCAS'89 Circuits	Relative % Improvement in % Compression with Proposed HDR-CBF-DV Test Data Processing Scheme compared to Unprocessed Data				
	FDR	Golomb	MFDR	EFDR	SAFDR
S5378	29.8	30.15	11.25	13.49	14.83
S9234	40.08	29.33	4.92	20.27	17.92
S13207	7.59	6.35	5.25	5.27	6.30
S15850	10.00	32.77	8.80	3.46	2.90
S38417	52.98	27.13	7.78	7.77	2.74
S38584	6.37	17.10	4.23	0.30	6.87

3.4 The Proposed 2-D Reordering Method

This section contains the various aspects of the prosed '2 – D Reordering' Method.

3.4.1 Problem Formulation

The 1-Dimension reordering (HBR-CBF-DV) method gives very good result for compression but does not address the issue of scan-power reduction. The next method proposed is 2-Dimensional reordering which not only increases the % compression but also reduces the test power. This method 'Hamming Distance Based 2-Dimensional Reordering with Power Efficient Don't Care Bit Filling' is an enhancement to HDR-CBF-DV (1-Dimensional) method. The partially specified test set from ATPG is two-fold reordered with the Hamming distance parameter. Test set is considered as a matrix of [m X n] where m is the total number of vectors in each set and n is the number of bits per vector. The first reorder is of row i.e. vectors. The m vectors are reordered on the basis of Hamming distance between them. The second reorder is of column. All n columns i.e. bits in the row-wise reordered set are reordered using Hamming distance. Then the unspecified bits in this 2-D reordered test set are mapped with power efficient bit mapping method.

3.4.2 Row-wise First Reordering

The approach used for the first reordering of partially specified uncompacted test set from ATPG, the heuristic for selection of first test vector and then remaining test vector is same as described in section 3.3.1. To avoid the repetition, it is not described here again. The same motivational example used in case of HDR-CBF-DV method will be used here to explain the next coming subsections.

3.4.3 Columnwise Second Reordering

For second reorder, the location of bits will be interchanged in each test vector. The same test data T as a matrix of *m* X *n* where *m* is the number of test pattern and *n* is the number of bits per test vector is considered. T_R is the row-wise reordered data set. As we want to cluster the 0s and 1s in each test vectors, the columns with matched bits should be placed nearby. For this, the transpose matrix T_{TR} of the T_R is used. Here the first raw of T_{TR} is kept unshifted and for remaining rows, the same Hamming Distance based reordering which is described in subsection 3.3.1 is applied. At the end of second reorder, T_{TDR}, i.e. the reordered T_{TR} will be prepared. Transpose matrix of T_{TDR} will be the required 2-dimensionally reordered test data set T_{DR}.

The 2-D reordered test set T_{DR} for given example is as follow:

t_1: 1 1 X X 0 0 X 1 1 0 X X 0 0

t_2:1 1 X 1 X 0 0 X 1 1 X 0 0 X

t_3:1 1 X 0 X 0 0 0 X X 1 X 0 0

t_4: 0 X X X 0 X 1 X X X X X 0 0

t_5:1 1 X 0 X 1 1 X 1 X 0 0 0 X

3.4.4 Power Efficient Bit Filling

The don't care bit mapping of test vector will be done using minimum transition fill (MT-fill) algorithm. Each unspecified (don't care) bit will be mapped by its predecessor specified bit. If the first test vector starts with unspecified bit(s), this unspecified bit(s) will be replaced by the first specified bit in the vector. Here, test data after bit mapping will be like:

t_1:1 1 1 1 0 0 0 1 1 0 0 0 0 0

t_2:1 1 1 1 1 0 0 1 1 1 1 0 0 1

t_3:1 1 1 0 1 0 0 0 1 1 1 1 0 0

t_4:0 0 0 0 0 0 1 0 0 0 0 0 0 0

t_5:1 1 1 0 0 1 1 0 1 0 0 0 0 0

3.4.5 On-Chip Decoder

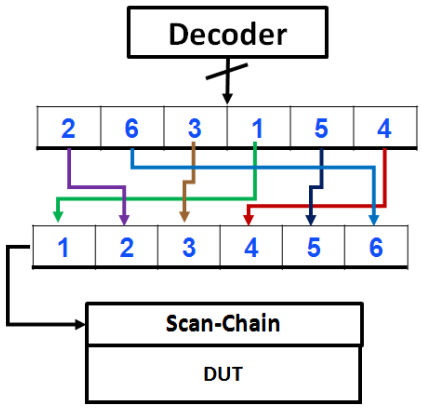

Figure 3.10: On-Chip Decoder Architecture for 2-D Method

The proposed 2-D test data processing method when applied with any of run length code, demands the basic decoder of the respective code only. This approach totally avoids the use of difference vector method. So it does not require any CSR area overhead and does not introduce any delay because of CSR. However, it needs a different routing for scan-in. Suppose for a given scan chain made of scan flip flops f_1, f_2, f_3, f_4 and f_5, the scan test data has four scan-in test vectors t_1, t_2, t_3, t_4. In each test vector there are five bits b_1, b_2, b_3, b_4 and b_5 corresponding to f_1, f_2, f_3, f_4 and f_5. During the first reorder, the test vectors are reordered in t_4, t_1, t_3, t_2 sequence. During the second reorder, the bits are reordered in b_1, b_3, b_4, b_5, b_2. This bit sequence will be then encoded by ASFDR (may be EFDR also). Let the encoded data for t_4 be e_1, e_2. This encoded data will serially move from ATE to on-chip decoder. The decoder will regenerate the b_1, b_3, b_4, b_5, b_2 bit sequence from the encoded data sequence e_1, e_2. The routing of scan flip-flops is done according to the second reordering i.e. f_1, f_3, f_4, f_5 and f_2, then the bits will automatically reach to proper flip-flops. The Figure 3.10 shows the architecture for 2D reordered data decoder.

3.4.6 Experimental Results

This subsection contains the experimental results of '2-D reordering' methods for various ISCAS'89 benchmark circuits.

Test Data Compression

The Table 3.20 and 3.21 compares the compressions of following three cases:

1. % compression in case of simple EFDR (or SAFDR)
2. Maximum % compression predicted by Entropy from Table 3.10
3. % compression achieved when test data is processed with 2-D reordering and applied EFDR (or SAFDR).

Table 3.20: % Compressions of EFDR with Proposed 2-D Method

ISCAS'89 Circuits	% Compression		
	EFDR[41]	Max. Limit for Unprocessed Test Data (Method C)	2-D +EFDR
S5378	51.93	56.32	**53.47**
S9234	45.89	53.37	**51.76**
S13207	81.85	85.54	**82.39**
S15850	67.99	71.90	**70.15**
S38417	60.57	65.84	**57.74**
S38584	62.91	66.67	**67.91**

Table 3.21: % Compressions of SAFDR with Proposed 2-D Method

ISCAS'89 Circuits	% Compression		
	SAFDR[43]	Max. Limit for Unprocessed Test Data (Method D)	2-D +SAFDR
S5378	-NA-	51.44	**54.00**
S9234	44.96	47.26	**52.03**
S13207	80.23	82.45	**82.41**
S15850	65.83	67.25	**70.24**
S38417	60.55	62.93	**57.79**
S38584	61.13	62.23	**68.02**

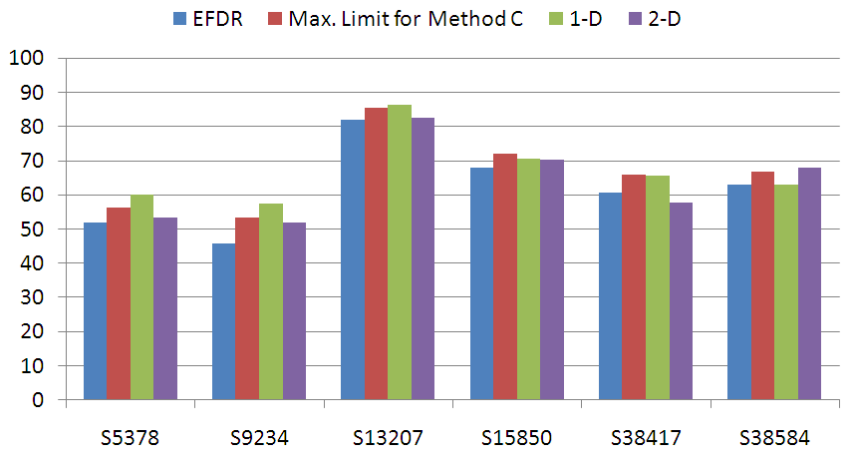

Figure 3.11: % Compression for EFDR, Max. Limit (C), HDR-CBF-DV and 2-D

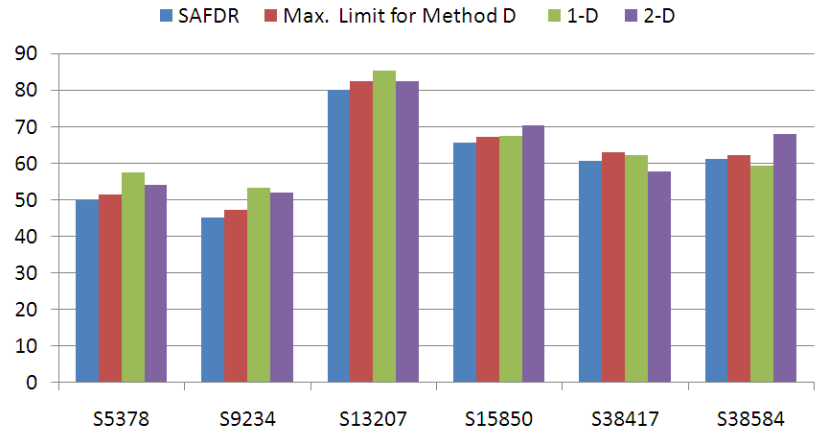

Figure 3.12:% Compression for SAFDR, Max. Limit (D), HDR-CBF-DV and 2-D

Scan-In Test Power

The Table 3.22 and 3.23 compares the scan-in peak power and average power in terms of total number of weighted transitions of following three cases:

a. Power in case of don't care bits mapped to '0' [47]
b. Power in case of don't care bits mapped to minimize transitions
c. Power in case of Hamming distance based 2-D reordered test set with power efficient bit filling, i.e. don't care bits mapped to minimize the transitions in 2-D reordered test set
d. % reduction in power

Table 3.22: Comparison of Peak Power

ISCAS'89 Circuits	Peak Power (# of Weighted Transitions)			
	Uncompacted test sets with don't cares mapped to 0s [47]	Uncompacted test sets with don't cares mapped to minimize WT [47]	2-D	% Reduction
S5378	10127	9531	**5010**	50.53
S9234	12994	12060	**6667**	48.69
S13207	101127	97606	**37972**	62.45
S15850	81832	63478	**34974**	57.26
S38417	505295	404617	**205989**	59.23
S38584	531321	479530	**182769**	65.79
Average % Reduction in Peak Power				**57.32**

Table 3.23: Comparison of Average Power

ISCAS'89 Circuits	Peak Power (# of Weighted Transitions)			
	Uncompacted test sets with don't cares mapped to 0s [47]	Uncompacted test sets with don't cares mapped to minimize WT [47]	2-D	% Reduction
S5378	3336	2435	**1581**	52.61
S9234	5692	3466	**2369**	58.38
S13207	12416	7703	**5140**	58.60
S15850	20742	13381	**9257**	55.37
S38417	172665	112198	**106040**	35.02
S38584	136634	88298	**52494**	61.58
Average % Reduction in Average Power				**53.59**

As it clearly shown in graph of Figure 3.12 and 3.13, the scan-in power is reduced drastically for each ISCAS'89 circuit. The average reduction in peak power is 57.32% and in average power is 53.59%.

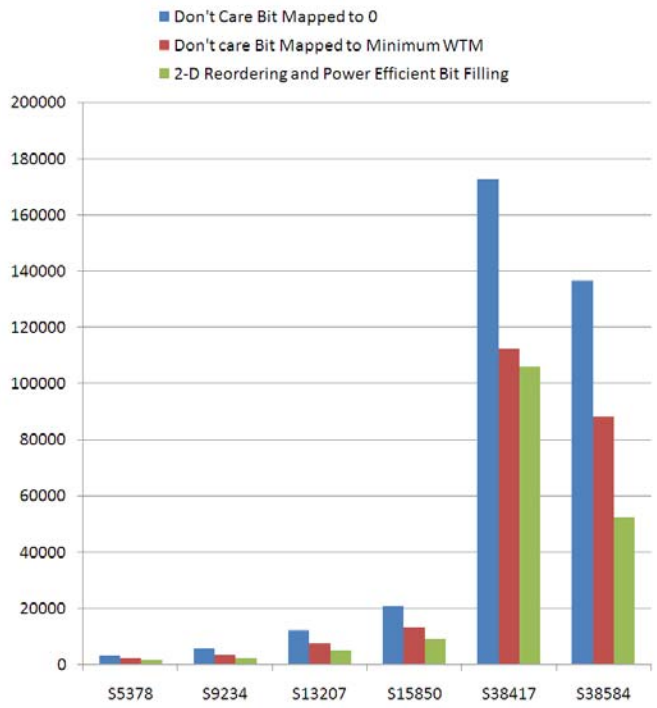

Figure 3.13: Comparison of Average Power for 2-D Method

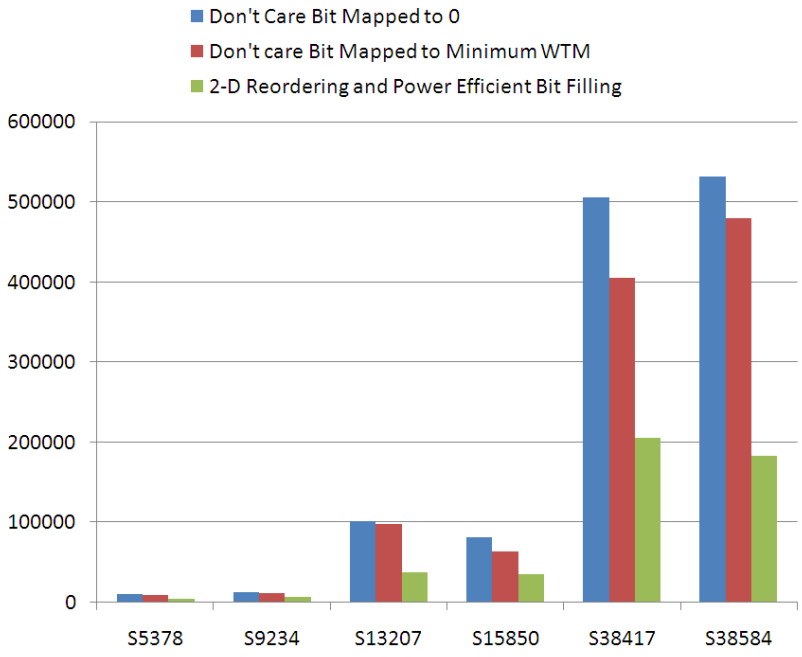

Figure 3.14: Comparison of Peak Power for 2-D Method

3.4.7 Observations and Analysis

Table 3.24: Relative % Improvement in % Comparison and Power in case of the Proposed 2-D Reordering Method

ISCAS'89 Circuits	Relative % Improvement in compression and Power for 2-D Reordering Method compared to the unprocessed test data			
	Compression		Power	
	EFDR	SAFDR	Average Power	Peak Power
S5378	2.97	-NA-	50.53	52.61
S9234	12.79	15.73	48.69	58.38
S13207	0.66	2.72	62.45	58.6
S15850	3.17	6.70	57.26	55.37
S38417	4.67	4.56	59.23	35.02
S38584	7.95	11.27	65.79	61.58

For unprocessed data, as shown in Table 3.17 and 3.18, the compression is large for EFDR and SAFDR compared to FDR, MFDR and Golomb coding. Table 3.24 represents the relative % improvement in compression when the test data is preprocessed with the proposed 2-D test data processing method. The Colum II of Table 3.24 represents the relative % improvement in compression when the data is processed with 2-D processing (reordering) method and EFDR scheme is applied. It is compared with unprocessed data directly applied the EFDR. Similarly, Column III represents the relative % improvement in compression when the data is processed with 2-D processing (reordering) method and applied the SAFDR compared to unprocessed data directly applied the SAFDR. From above analysis (Ref. Table 3.24), it can be shown that the proposed 2-D test data processing method applied on test data further improves the % compression in case of EFDR and SAFDR. In addition, the proposed 2-D processing (reordering) method provides additional % improvement in reduction of peak power and average power at the cost of the larger on-chip area overhead.

3.5 The Proposed WTR-CBF-DV Method

The earlier proposed Hamming Distance based Reordering - Columnwise Bit Filling and Difference Vector (HDR-CBF-DV) is taken as the basic scheme for this method.

3.5.1 Problem Formulation

During the reordering process for various circuits, the authors found that very frequently, the test set contains more than one test vectors with same Hamming distance. This tie should be broken in favor of power reduction. So this proposed method applies both 'the Hamming distance and weighted transitions' as the criteria for reordering. The Hamming distance improves the compression and weighted transition improves power reduction.

3.5.2 Weighted Transition Based Reordering

This section contains the steps of weighted transition based reordering like selection of first vector, reordering and bit filling of remaining vectors, run length coding etc.

Selection of the First Vector

Same as HDR-CBF-DV, for the selection of first test pattern, the heuristic applied in this scheme is "Hardest Path First". The test pattern with minimum don't cares will be selected as the first test pattern of reordered list. Unlike random selection in HDR-CBF-DV, in WTR-CBF-DV, if there are more than one test vectors with minimum number of don't care bits, each will be MT filled and then evaluated for its weighted transitions and the vector with minimum weighted transition will be selected as the first

test vector of the reordered test set. Further in HDR-CBF-DV, the first vector is kept unfilled until all the test vectors are reordered but in this proposed scheme, the selected first vector is MT filled immediately before continue reordering to make the overall testing and selection of remaining test vectors power aware.

Reordering and Bit Filling of Remaining Test Vectors

For reordering of the remaining test patterns in HDB-CBF-DV, the pattern with minimum Hamming distance from first pattern of reordered set will be placed next to first pattern of reordered set. It is decided to take the next vector with minimum Hamming distance because when the further columnwise bit filling and difference vector will be done, this reordered vector sequence will generate maximum zeroes such that run length and hence the compression will increase. After completing the reordering of all the vectors, for the second test patterns and onwards, the don't care bit will be replaced by the same value which its upper vector has at the same position. Unlike this approach of HDR-CBF-DV, in the WTR-CBF-DV, while selecting the next vector of the reordered test set, if there are more than one vectors with the same Hamming distance from the last selected vector of reordered set, the weighted transition will be taken into consideration. All this equidistance vectors will be considered for applying columnwise bit filling and their weighted transitions are calculated. The vector with the minimum weighted transitions will be selected as next vector of reordered test set. The don't care bits in this vector will be replaced by the same value which its upper vector has at the same position before selecting the next vector for reordered set. The same procedure will be repeated until all the vectors are reordered.

3.5.3 Difference Vector

The next step is to take the difference vector of two consecutive vectors in the reordered set same as described in HDR-CBF-DV. This will further increase the numbers of zeroes and hence data compression.

3.5.4 Run Length Code for Compression

For the proposed method, the test data will be first pre-processed by WTR-CBF-DV scheme and then, the Frequency Directed Run Length code (FDR) [28] will be applied to pre-process data. The example in Figure 3.3 and Table 3.3 demonstrates this coding format.

3.5.5 Algorithm for WTR-CBF-DV

----------WTR-CBF-DV----------

Step: 1 Consider a digital circuit with n scan flipflops, p inputs and q outputs. The ATPG generated partially specified test set with m scan-in test-vectors each of n bits is the input to this algorithm.

Step: 2 Find the test vector with minimum number of don't care bits in the given test set.

Step: 3 If there are more than one vector with minimum don't care bits,

a. Apply MT fill to each vector
b. Calculate weighted transition for each vector.
c. Select the MT filled vector with minimum WT as first vector of reordered set

Step: 4 Find the Hamming distance of remaining each vector from the first vector of reordered set.

Step: 5 Select the vector with minimum Hamming distance as next vector.

Step: 6 If there are more than one vector with minimum Hamming distance, than

a. Apply columnwise bit filling to each vector i.e. replace the don't care bit of the vector with the same position bit value of last selected vector.

b. Calculate weighted transition for each vector.

c. Select the columnwise bit filled vector with minimum WT as next vector of reordered set

Step: 7 Repeat step-6 until all the vectors are reordered.

Step: 8 Apply difference vector mechanism.

Step: 9 First vector of reordered set is kept unchanged.

Step: 10 From the second vector onward, if the same position bits in last vector and current vector are same, replace the bit by 0 else 1.

Step: 11 Apply frequency directed run length code.

3.5.6 Motivational Example

Considering the following test data for example:

Test Vector Set

1 X 1 0 0 X X 0 1 X 0 0 X 1

1 1 1 X 0 X 0 X 1 0 1 0 X X

1 0 1 1 0 X 0 0 X X X 0 1 0

0 X X 0 X X 1 0 X X X 0 X X

1 0 1 X 1 X 1 X 1 0 X 0 0 X

1 1 1 1 0 X 0 0 X X X X 0 0

Selection of First Test Vector of Reordered Set

In the above test data, vector V_3 has the minimum number of don't care bits i.e. 4. Now this vector is MT filled as shown below:

V_3 : 1 0 1 1 0 0 0 0 0 0 0 0 1 0

Its corresponding weighted transition as per equation 2.4 is 38.

Reordering Remaining Test Vector with Columnwise Bit Filling

After placing the V_3 at first place and V_1 shifted to position of vector 3, the test set after first vector selected and filled is:

Test Vectors after First Vector Reordered

R_1 1 0 1 1 0 0 0 0 0 0 0 0 1 0
V_2 1 1 1 2 0 2 0 2 1 0 1 0 2 2
V_3 1 2 1 0 0 2 2 0 1 2 0 0 2 1
V_4 0 2 2 0 2 2 1 0 2 2 2 0 2 2
V_5 1 0 1 2 1 2 1 2 1 0 2 0 0 2
V_6 1 1 1 1 0 2 0 0 2 2 2 2 0 0

Now the Hamming distance of remaining each test vector V_2, V_3, V_4, V_5 and V_6 from first reordered vector R_1 is 3, 3, 3, 4 and 2. Hence, V_6 is selected as next vector of reordered set. After placing V_6 as R_2, the columnwise bit filling is done.

Partially Reordered Test Vectors

R_1 1 0 1 1 0 0 0 0 0 0 0 0 1 0
R_2 1 1 1 1 0 0 0 0 0 0 0 0 0 0
V_3 1 2 1 0 0 2 2 0 1 2 0 0 2 1
V_4 0 2 2 0 2 2 1 0 2 2 2 0 2 2
V_5 1 0 1 2 1 2 1 2 1 0 2 0 0 2
V_6 1 1 1 2 0 2 0 2 1 0 1 0 2 2

Now the Hamming distance of V_3, V_4, V_5 and V_6 from R_2 is calculated as 3, 3, 4 and 2. So V_6 is selected as R_3

Partially Reordered Test Vectors

R_1 1 0 1 1 0 0 0 0 0 0 0 0 1 0

R_2 1 1 1 1 0 0 0 0 0 0 0 0 0 0

R_3 1 1 1 1 0 0 0 0 1 0 1 0 0 0

V_5 0 2 2 0 2 2 1 0 2 2 2 0 2 2

V_6 1 0 1 2 1 2 1 2 1 0 2 0 0 2

V_7 1 2 1 0 0 2 2 0 1 2 0 0 2 1

Now the remaining all three V_4, V_5 and V_6 vectors have equal Hamming distance 3 from R_3. Now each vector will be tried for its weighted transition as if columnwise bit filling is applied to it.

R_3 1 1 1 1 0 0 0 0 1 0 1 0 0 0

V_4 0 1 1 0 0 0 1 0 1 0 1 0 0 0

Applying the equation 2.4, the weighted transition is equal to 57 for this case.

R_3 1 1 1 1 0 0 0 0 1 0 1 0 0 0

V_5 1 0 1 1 1 0 1 0 1 0 1 0 0 0

and

R_3 1 1 1 1 0 0 0 0 1 0 1 0 0 0

V_6 1 1 1 0 0 0 0 0 1 0 0 0 0 1

The possible weighted transitions for V_5 and V_6 are 67 and 23 respectively as shown above. V_6 is selected as the next test vector. Repeating the reordering process until the last vector is reordered, the final reordered set is as follow:

Reordered Test Vectors

R_1 1 0 1 1 0 0 0 0 0 0 0 0 1 0

R_2 1 1 1 1 0 0 0 0 0 0 0 0 0 0

R_3 1 1 1 1 0 0 0 0 1 0 1 0 0 0

R_4 1 1 1 0 0 0 0 0 1 0 0 0 0 1

R_5 0 1 1 0 0 0 1 0 1 0 0 0 0 1

R_6 1 0 1 0 1 0 1 0 1 0 0 0 0 1

Difference Vector

The next step is to take the difference vector of two consecutive vectors in reordered set to increase the numbers of zeroes and hence data compression. The difference vector set is as shown below.

Difference Test Vectors

R_1 1 0 1 1 0 0 0 0 0 0 0 0 1 0

D_2 0 1 0 0 0 0 0 0 0 0 0 0 1 0

D_3 0 0 0 0 0 0 0 0 1 0 1 0 0 0

D_4 0 0 0 1 0 0 0 0 0 0 1 0 0 1

D_5 1 0 0 0 0 0 1 0 0 0 0 0 0 0

D_5 1 1 0 0 1 0 0 0 0 0 0 0 0 0

Run Length Coding

The difference vector set is applied the frequency directed run length coding as described in subsection 3.5.3.

Comparison

In Table 3.25, the comparison of % compression, peak power and average power for various test data processing scheme with FDR coding applied to test data of motivational example is given.

Table 3.25: Comparison of Test Data Processing Methods Applied with FDR Coding

	MT Fill	0 Fill - DV	HDR-CBF-DV	WTR-CBF-DV
% Compression	-2.38	7.14	16.67	16.67
Peak Power (# of Weighted Transitions)	38	81	82	82
Average Power (# of Weighted Transitions)	23.83	36.83	42.17	38.67

The column II in Table 3.25 shows the results when test data is applied with MT filling and FDR coding. The peak power and average power is minimum in this case but the compression is negative. Column III is for test data where don't care bits are filled with 0s and without reordering, the difference vectors are created and FDR is applied. The column IV and V represents the results for HDR-CBF-DV and WTR-CBF-DV. As it is seen from these results, the % compression is maximum in case of HDR-CBF-DV and WTR-CBF-DV. While average power is comparable in case of difference vector only, HDR-CBF-DV and WTR-CBF-DV.

3.5.7 On-Chip Decoder

The WTR-CBF-DV is a test data processing method applied in conjunction with FDR coding. The same FDR decoder described in [28] as shown in Figure 3.7 is used for this method. As in this approach, the difference vector is already used, the proposed WTR-CBF-DV does not require any extra on-chip area overhead.

3.5.8 Experimental Results

The Table 3.26, 3.27 and 3.28 shows the comparison of % compression, average power and peak power for various ISCAS'89 circuits' test data with FDR coding when test data is applied the following processing prior to FDR coding:

1. Don't care bits are MT filled but no reordering is applied.
2. Don't care bits are filled with on the basis of run type but no reordering is applied (Here the maximum possible compression is taken into consideration).
3. Don't care bits are filled with 0s, difference vector is applied [27].
4. HDR-CBF-DV applied.
5. 2-D Reordering is applied.
6. WTR-CBF-DV applied.

Table 3.26: % Compression for WTR-CBF-DV Method with FDR Coding

	MT Fill	0 Filling +XOR	Run Based Bit Filling Max. Limit	HDR CBF DV	2-D	WTR CBF DV
s5378	-12.31	48.02	52.36	62.33	59.66	62.35
s9234	-20.67	43.59	47.80	61.06	61.35	63.31
s13207	6.16	81.30	83.65	87.47	88.22	88.04
s15850	-17.91	66.22	68.18	72.84	73.96	73.33
s38417	-20.39	43.26	54.50	66.18	65.13	66.38
s38584	-8.90	60.91	62.49	64.79	66.08	65.20

Table 3.27: Average Power for WTR-CBF-DV Method

	MT Fill	0 Filling +XOR	Run Based Bit Filling Max. Limit	HDR CBF DV	2-D	WTR CBF DV
s5378	3433	3526	3526	11133	7934	10344
s9234	3958	4022	4022	14382	13329	13492
s13207	7735	7887	7887	113890	78856	103400
s15850	13514	13659	13659	82421	71015	64275
s38417	117540	118080	118080	452860	486000	443030
s38584	85656	86305	86305	410240	423260	329110

Table 3.28: Peak Power for WTR-CBF-DV Method

	MT Fill	0 Filling +XOR	Run Based Bit Filling Max. Limit	HDR CBF DV	2-D	WTR CBF DV
s5378	3433	3526	3526	11133	7934	10344
s9234	3958	4022	4022	14382	13329	13492
s13207	7735	7887	7887	113890	78856	103400
s15850	13514	13659	13659	82421	71015	64275
s38417	117540	118080	118080	452860	486000	443030
s38584	85656	86305	86305	410240	423260	329110

3.5.9 Observations

The proposed test data processing scheme 'WTR-CBF-DV' improves the % compression compared to earlier methods described in literature. Moreover, this method increases the compression beyond the limit of maximum possible compression in case if data is filled with run type basis. The peak power and average power is controlled using weighted transition based reordering. The proposed scheme demands no extra on-chip area overhead compared to earlier methods in literature.

3.6 Inferences

In this chapter, first of all, the current run length based test data compression techniques are implemented and analyzed. From the comparison of % compression in case of each technique, it is inferred that the Extended FDR (EFDR) method gives the maximum compression.
It was observed that the various types of runs used in run length codes can be classified in to four categories only. On the basis of this observation, five different bit filling methods were proposed. The maximum test data compression limit based on entropy is calculated for the proposed run based bit filling methods.
It was enforced that to achieve the compression improvement beyond the maximum possible limits in case of run type based filling, the test data must be further processed before applying the coding.
The proposed 'Hamming Distance Based Reordering-Columnwise Bit Filling-Difference Vector (HDR-CBF-DV or 1-D) is the data processing method which helps the % compression to increase beyond the maximum limit that was predicted for unprocessed data.
While considering HDR-CBF-DV method of data processing, only test data compression was taken in to consideration. The scan-in power, on-chip area overhead and test data compression are trade off. In the next

proposed '2-Dimensional Reordering with Power Efficient Bit Filling' method for data processing, scan-in power is considered with test data compression. The results of 2-D reorder method show that with use of this approach, the scan-in power reduces drastically. The compression is also higher compared to the base run length code and bit filling method. However, the trade off is larger on-chip area for decoder.

The 'Weighted Transition Based Reordering – Columnwise Bit Filling – Difference Vector (WTR-CBF-DV)' reduces the scan-in power without scarifying on-chip area.

From all above observations, finally it is concluded that in case of run length based test data compression method, if the WTR-CBF-DV is applied to prepare the data for long runs of zeros and then FDR coding is applied, it gives the best results in terms of compression, scan-in power and area overhead.

Chapter 4

Statistical Code Based Test Data Compression

The statistical codes are fixed-to-variable length codes. In this method, the original test cubes are partitioned into n-bit blocks to form the symbols. These symbols are then encoded using variable-length codewords which represent fixed-length blocks of bits in a data set. The code words are assigned based on each symbol's frequency of occurrence. It assigns shorter code words to symbols that occur more frequently, and longer code words to those that occur less frequently. This strategy minimizes the average length of a code word.

4.1 Existing Statistical Codes

The overview of the existing statistical codes used for test data compression followed by their analysis is given in this section.

4.1.1 Overview of Existing Statistical Codes

This section contains the detailed description of Huffman code and its variants like selective Huffman code, optimal selective Huffman code, RL Huffman code etc..

4.1.1.1 Huffman Code

The Huffman code is very widely used statistical code in all lossless data compression applications as the Huffman code is an optimal statistical code that is proven to provide the shortest average codeword length among all uniquely decodable variable length codes. The only disadvantage is that when the Huffman hardware decoder is placed in chip, it requires a large amount of area overhead because decoder size grows exponentially with symbol size.

4.1.1.2 Selective Huffman Code

A scan vector compression scheme based on selective Huffman coding is described in [26]. The idea is to only code the most frequently occurring blocks instead of all blocks using codewords with small numbers of bits. Consider the case where the test set is divided into fixed-length blocks of *n* bits. The first bit of each codeword will be used to indicate whether the following bits are coded or not. If the first bit of the codeword is a 0, then the next *n* bits are not coded and can simply be passed through the decoder as it is (hence, the complete codeword has *n+1* bit). If the first bit of the codeword is a '1', then the next variable number of bits forms a prefix-free code that will be translated by the decoder into an *n*-bit block. Even though, this method gives less compression ratio compare to simple Huffman coding, it simplifies the complexity of decoder and hence requires less on chip area overhead.

4.1.1.3 Optimal Selective Huffman Code

The inefficiency of the Selective Huffman encoding stems from the fact that the required extra bit equally lengthens all data in the compressed test set (encoded or not), irrespective of their occurrence frequency [27]. This constant overhead, although minimum for the unencoded data

blocks, can be relatively high for the most frequently occurring codewords. A better approach would be to use an additional Huffman codeword in front of only the unencoded data blocks, relieving in this way the most frequently occurring codewords from the extra-bit overhead. Thus, the gain from shortening the codewords that appear very often in the compressed test set will overbalance the loss from using a whole codeword, instead of a single bit, in front of the unencoded data blocks (the unencoded data blocks are usually a small fraction of the compressed test set). This approach is used in optimal selective Huffman code [27]. Table 4.1 demonstrates the example of the Selective Huffman Code and optimal selective Huffman code where the number of distinct symbols to be coded (n) is 3.

Table 4.1: Selective and Optimal Selective Huffman Codes

Distinct Patterns	Freq	Selective Huffman Code (n=3)	Optimal Selective Huffman Code (n=3)
S0- 1010	9/20	**1**0	**0**
S1- 0000	5/20	**1**10	**10**
S2- 1111	3/20	**1**11	**110**
0001	2/20	**0**0001	**111**0001
0010	1/20	**0**0010	**111**0010

4.1.1.4 RL-Huffman Code

In RL-Huffman Coding [32], the don't care bits in test vector are first filled with either '0' or '1' such that the overall run length of 1s or 0s has been increased but still the run lengths are in alternating form. i.e. runs of 0s and 1s are alternating. Now each run length will have some occurrence of frequency. Based on occurrence of frequency, they are coded using Huffman coding. In final codeword, the first bit will be the indication of type of first run length and then followed by Huffman codes indicating the size of alternating run length block.

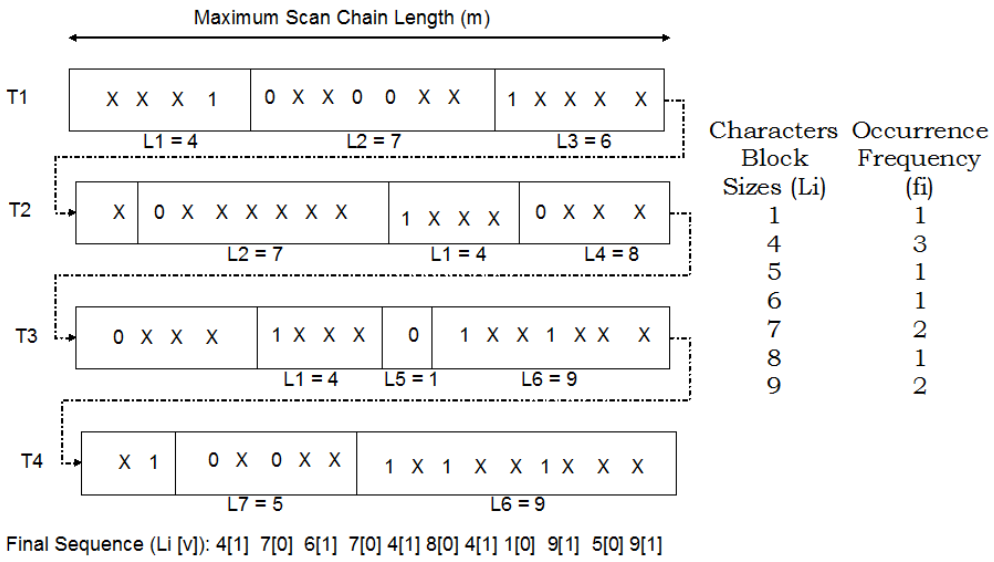

Figure 4.1: Example for RL-Huffman Code[32]

Table 4.2: RL-Huffman Code for Given Example

Run Length	Occurrence of Frequency	Huffman Code	Saving
4	3	00	+6
9	2	010	+12
7	2	011	+8
5	1	100	+2
8	1	101	+5
6	1	110	+5
1	1	111	-2
Total savings (in bits)			+34

4.1.1.5 Multilevel Huffman Code

The Multilevel Huffman test data compression method [33, 34] is based on Huffman coding with a limited number of codewords. The test cubes (test vectors with 'X' values) of the CUT are compared against the pseudorandom sequences generated by various cells of an LFSR, and if they match (i.e. they are compatible), an appropriate cell is chosen for feeding the scan chain(s) of the CUT. What is actually coded is an index for each selected LFSR cell, i.e. each Huffman codeword is used for enabling a specific LFSR cell to feed the scan chain(s). If no match with

an LFSR-cell sequence can be found, then the test data are directly encoded using a selective Huffman code.

4.1.1.6 Variable Length Input Huffman Coding

This method uses patterns of variable-length as input to the Huffman algorithm [24], allowing an efficient exploitation of test sets which exhibit long runs of '0s'. Table 4.3 demonstrates this coding method.

Table 4.3: % Compression for VIHC (Group size=4)

Symbol	Pattern	Frequency	Code
L0	1	4	00
L1	01	3	011
L2	001	2	010
L3	0001	5	10
L4	0000	7	11

4.1.1.7 Variable-To-Variable Huffman Code

In [35], the well-known characteristic that, in every test set, there are regions with many defined bits (i.e. densely specified) and regions with many x bits (i.e., sparsely specified) is explored. Densely specified regions are the main sources of unencoded data, and, therefore, their compression is favored by the usage of small distinct blocks. On the other hand, sparsely specified regions are more efficiently compressed using large distinct blocks, since, this way, many test set parts, despite their big size, are compatible with the encoded distinct blocks due to the great number of x bits that they contain. To improve compression, the test sets should be partitioned into variable-length parts. It means that variable-length distinct blocks should be encoded. Apart from the variable-to-variable nature of the proposed approach, the generated codewords are reusable in the sense that they can encode compatible blocks of different sizes [35].

4.1.1.8 Split Variable Input Huffman Code (Split-VIHC)

Spilt VIHC [44] approach demonstrates that before going to the VIHC, if the test data sete is divided in to two or more equal parts and the vectors are reordered in a certain way. Then applying VIHC coding scheme on both file, the compression can be improved compare to VIHC. The same test data used in Table 4.3 is divided into two equal halves. One half contains the first two vectors and other half the remaining two. Now if the VIHC coding is applied separately on both files as shown in Table 4.4 and 4.5, then the total compression length becomes 26+14=40. Then compression ratio is 37.5% which is higher the VIHC coding scheme.

Table 4.4: Split VIHC for Test Set 1

Symbol	Patterns	Frequency	Code
L0	1	4	00
L1	01	1	0111
L2	001	2	010
L3	0001	1	0110
L4	0000	4	1

Table 4.5: Split VIHC for Test Set 2

Symbol	Patterns	Frequency	Code
L0	1	2	11
L1	0001	4	0
L2	0000	3	10

4.1.2 Analysis of Existing Statistical Codes

Various statistical codes are reverified by their implementation using MATLAB and C language. This section contains the analysis of compression, power and area based on the experimental results of these implementations.

Test Data Compression

For analysis of % compression, various statistical codes described in literature are implemented. Table 4.6 represents % compression in case of various statistical codes.

Table 4.6: % Compression for Various Huffman Codes

Circuit	Selective Huffman Code B= 8, N=8	Optimal Selective Huffman Code B= 8, N=8	VIHC	Split VIHC
s5378	26.7	42.6	43.8	44.9
s9234	30.2	50.3	44.6	45.7
s13207	45.1	69.2	67.1	69.5
s15850	35.1	57.6	57.1	58.7
s38417	30.3	49.6	54.7	55.6
s38584	36.2	57.8	55.2	56.4

On-Chip Area Comparison

The on-chip area overhead of existing statistical code is described inthis subsection.

Huffman Decoder

The encoded test data is read out one bit at a time during decoding. The patterns can be decoded from the contents of encoded data using a simple finite-state machine (FSM) as described in Figure 4.2 [25].

The FSM receives a single-bit input from encoded data set and produces an *n*-bit output for the test patterns, where *n* is the length of each pattern. The decoder also produces a single-bit control output VALID. The control output is enabled only when a valid pattern is generated by the decoder. The prefix-free property of the Huffman code, on which the design is based, ensures that no short codeword is duplicated as the beginning of a longer codeword. The use of the VALID signal ensures that the sequence of D is preserved and no additional patterns are output

from the decoder. The FSM decoder for Huffman code with a symbol size of n bits contains total 2^n states.

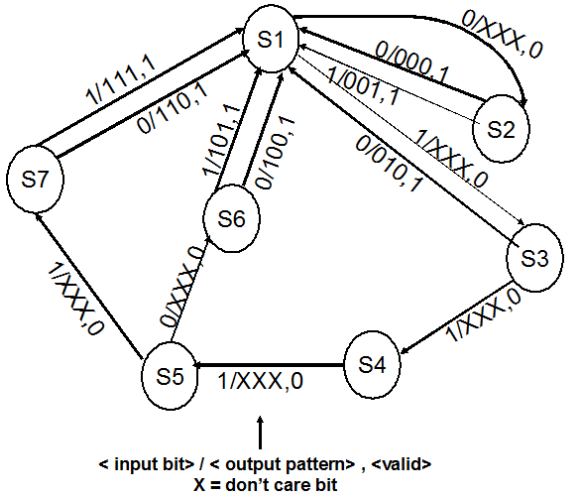

Figure 4.2: FSM for Huffman Decoder [25]

Selective Huffman Decoder

The selective Huffman decoder [26] is shown in Figure 4.3.

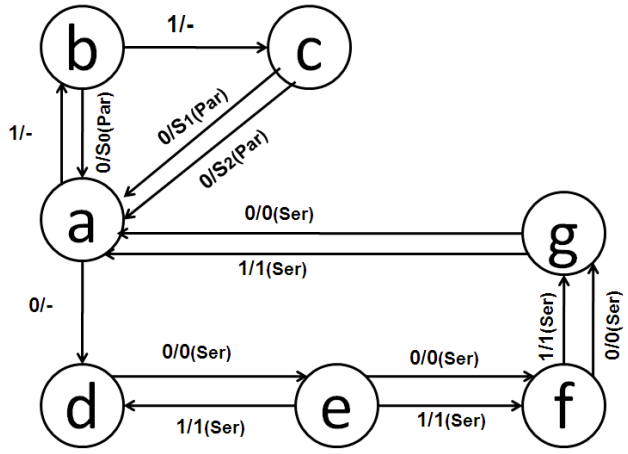

Figure 4.3: FSM Decoder for Selective Huffman Code

Optimal Selective Huffman Decoder

For optimal selective Huffman code, only coding scheme is described in [27]. The FSM for optimal selective Huffman code is proposed in Figure 4.4. Figure 4.4 describes the example of an implementation of the decoder using simple FSM for a specific case of 3 bits per symbols and 4

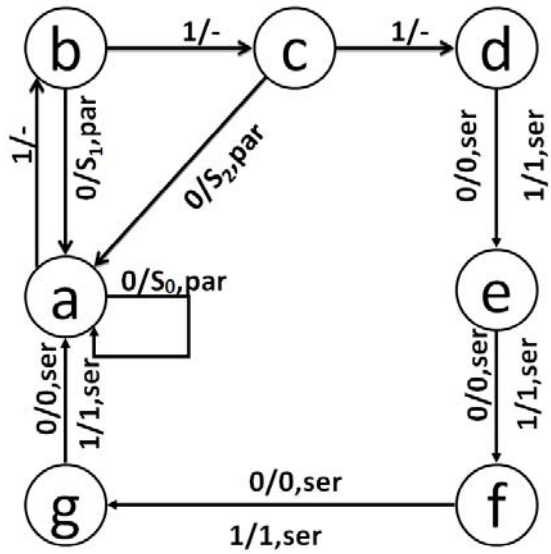

Figure 4.4: FSM Decoder for Optimal Selective Huffman Code

symbols to be encoded. There are two inputs to the decoder, one is the tester clock and the other is the serial input from the tester channel. For a symbol size of *m*, the decoder has *m* data outputs and two control outputs. The two control outputs are: parallel load (*Par*) and serial load (*Ser*). These two signals control the buffering and loading of data into the serializer when the data has been decoded. On arrival of the first bit of the codeword, the decoder branches on each bit one at a time until it reaches the end of the codeword at which point it does a parallel loading of the appropriate *m*-bit block into the register. The codeword '*111*' indicates that the next *m* bits are original unencoded data bits. So *Ser* signal will be high for next *m* cycles and will load next *m* bits serially from coded data.

Area overhead for on chip decoder is directly proportion to the states of decoder FSM and the memory requirement to store the symbol words. To make the discussion technology independent, the total bits required to implement the FSM as well as memory is used here for comparison purpose. After implementing the FSMs for various symbol size and number of symbols to be coded, it is observed that if *N* is the number of distinct symbols to be encoded, and *M* is the number of bits per symbol,

the number of states in FSM will be N+M as shown in equation 4.1. For on-chip storage of N distinct symbols with size M bits each, the memory requirement is approximately $N\ X\ M$ bits. So the total hardware requirement for selective or optimal selective Huffman decoder can be approximated as shown in equation 4.2.

$$\#\ of\ FSM\ States\ S_{fsm} = \#\ of\ Coded\ Symbols\ (N) + \ \#\ ofBits/Symbol(M) \quad \ldots\ldots\ldots\ldots\ldots\ (4.1)$$

$$On - Chip\ \ Area\ for\ Optimal\ Decoder\ (\#\ of\ equivalent\ bits) = \ log_2(N+M) + \ (N*M) \quad \ldots\ldots\ldots\ldots\ldots\ldots(4.2)$$

VIHC Decoder

The details of block diagram and other units of the VIHC decoder are given in Figure 4.5, 4.6 and 4.7 from [32]. The decoder comprises a Huffman decoder (Huff-decoder) and a Control-and-Generator-Unit (CGU). The Huff-decoder is a finite state machine (FSM) which detects a Huffman code and outputs the corresponding binary code.

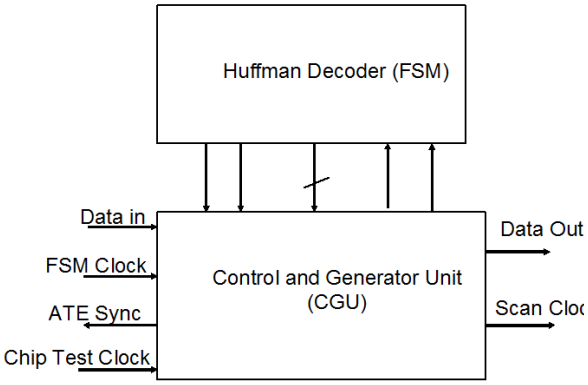

Figure 4.5: VIHC Decoder [24]

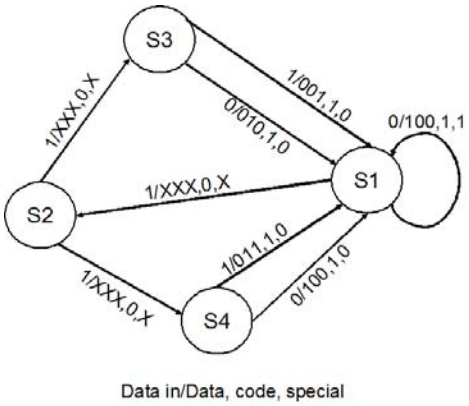

Figure 4.6: FSM for VIHC Decoder [24]

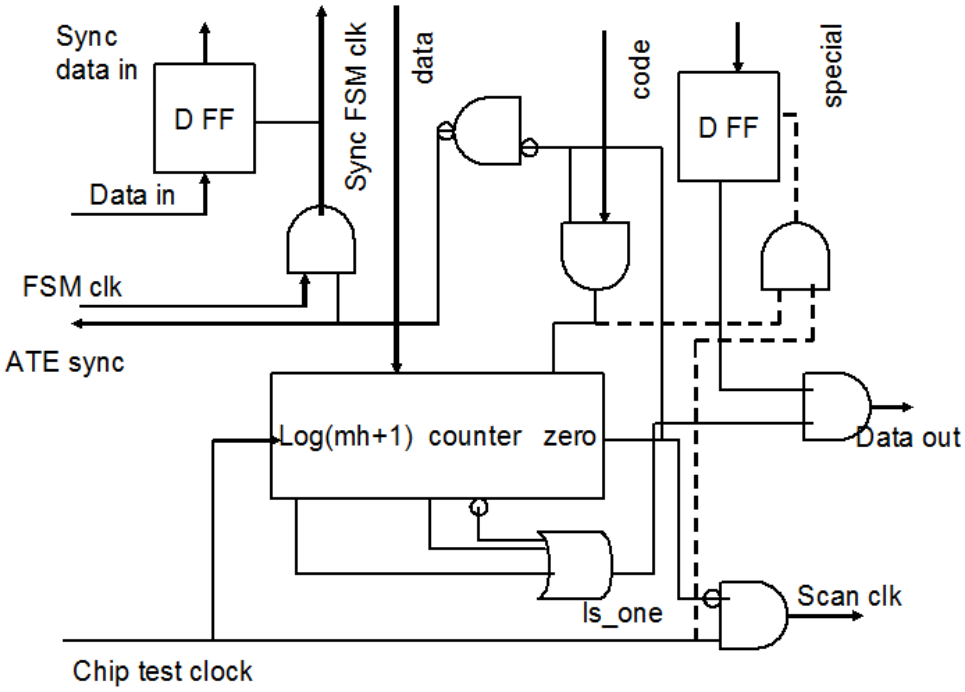

Figure 4.7: CGU for VIHC Decoder [24]

The CGU is responsible for controlling the data transfer between the ATE and the Huff-decoder, generate the initial pattern and control the scan clock for the CUT. The data in line is the input from the ATE synchronous with the external clock (ATE clock). When the Huff-decoder detects a codeword, the code line is high and the binary code is output on the data lines. The special input to the CGU is used to differentiate between the two types of patterns composed of '0's only (Lm_h) and runs of '0's ending in '1' (L_0. . . . . .Lm_{h-1}).

Split VLIHC Decoder

The decoder is similar to VLIHC decoder. It consists of two parallel units: a finite state machine (FSM) and a code generation unit (CGU). The interface between the two units is also similar to the VLIHC decoder as shown in Figure 4.7 [24]. The FSM provides data, code and special bit.

The data is the binary index of the word to be output by the CGU, code identifies the time when data is valid. Special bit distinguishes an all zero block from the block having only last bit as 1. In Split VLIHC controller, only a single FSM is used to detect code words from two files. The FSM for this is shown in Figure 4.8 [44].

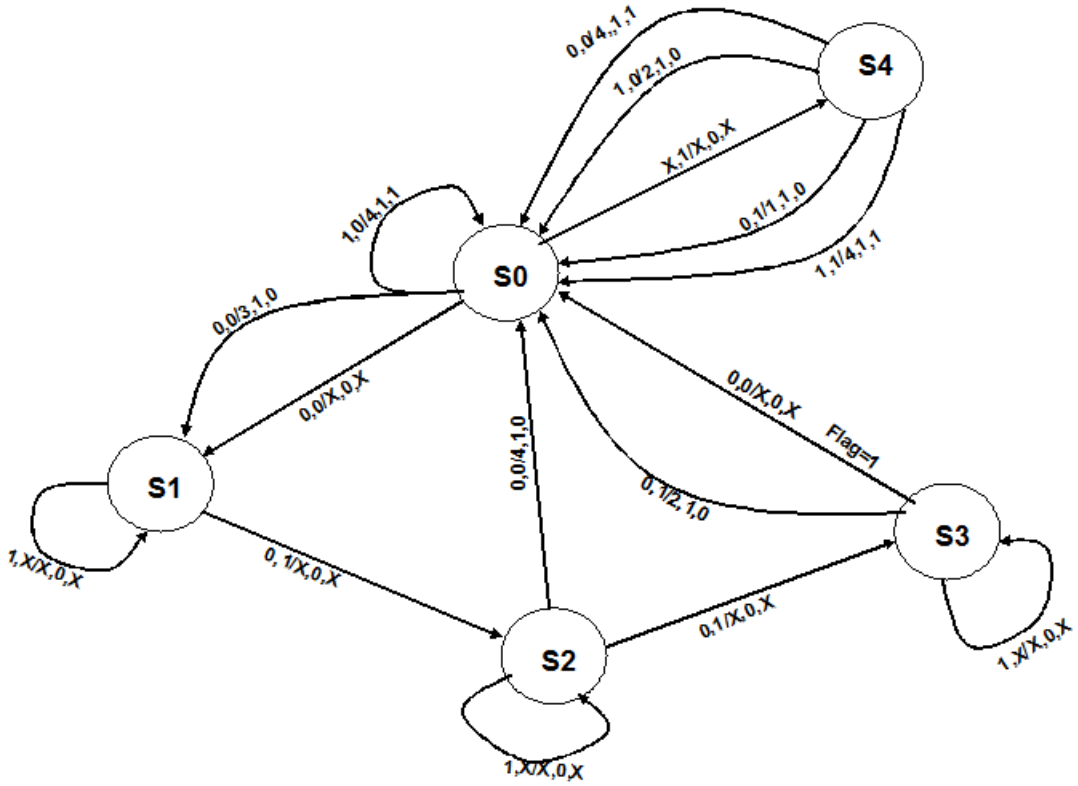

Figure 4.8: FSM for Split VIHC Decoder [44]

4.1.3 Observations

The Figure 4.9 represents the comparison of % compression and Figure 4.10 represents the comparison of area overhead for various statistical codes. Even though the % compression for optimal Huffman code, VLIHC and Split VLIHC is nearly same, the area overhead is very less in case of optimal selective coding style. For further development, the optimal selective code is chosen as the reference code.

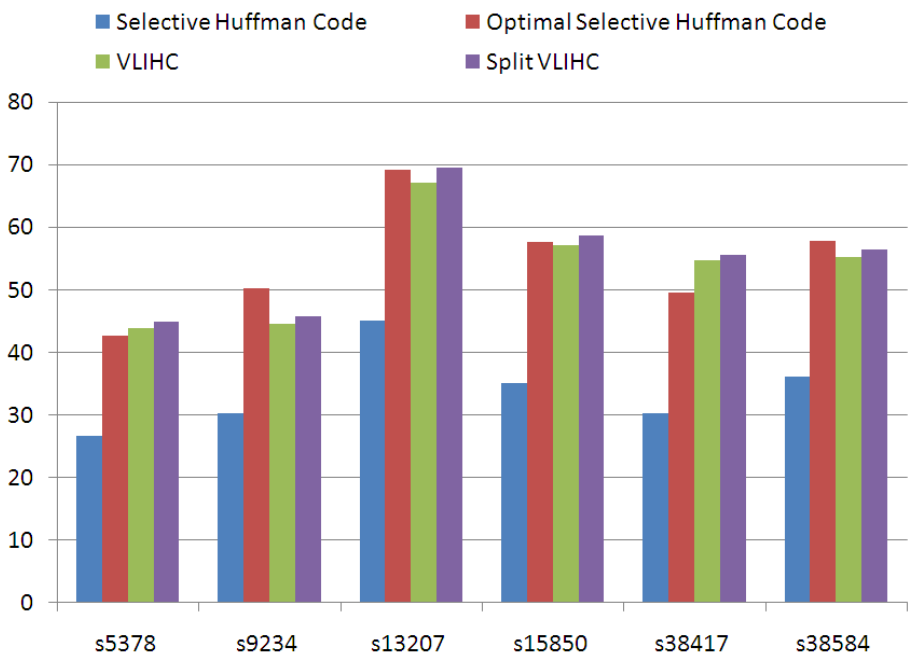

Figure 4.9: % Compression for Various Statistical Codes

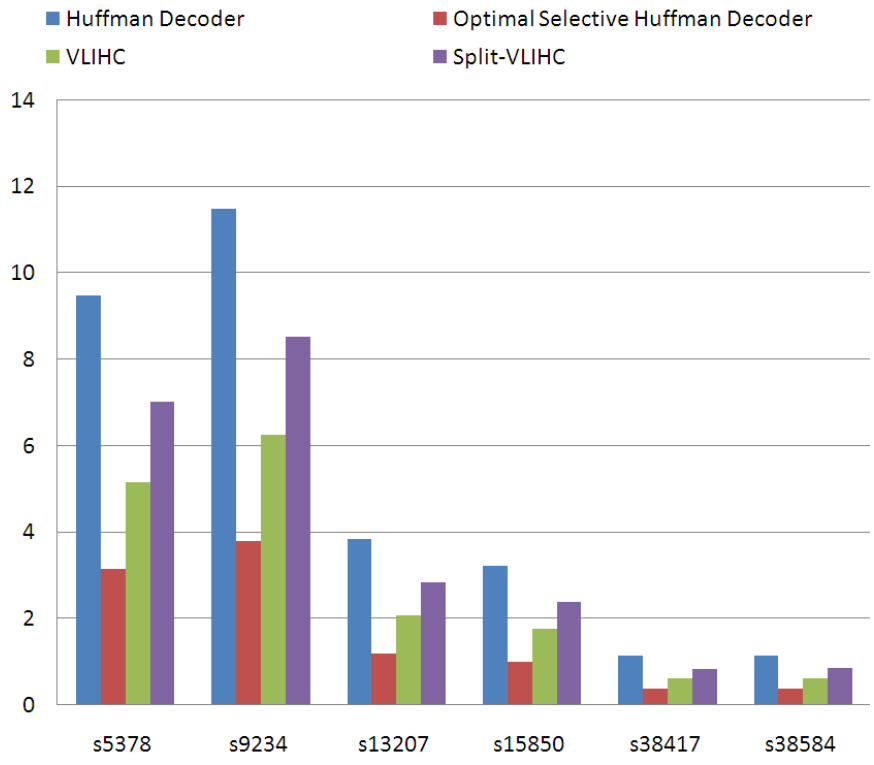

Figure 4.10: % Area Overhead for Various Statistical Decoders

4.2 The Proposed MS-Huffman Code

This section describes the various aspects of the proposed Modified Selective Huffman (MS-Huffman) code.

4.2.1 Problem Formulation

In section 4.1, it has been shown that, compared to selective Huffman code, in case of optimal selective Huffman code, the compression has increased to large extent with a marginal increase in the on-chip area overhead.

However, for optimal efficiency in terms of test application time, the unencoded blocks must be directly transferred from the ATE to serializer or scan chain in serial mode without any extra routing. The weakness of optimal selective Huffman code is that it places one code word in front of each unencoded symbol. Before the uncoded symbol bits are transferred from ATE to serializer, these bits will have to pass through a large number of FSM states of decoder at an ATE clock speed which is quite slow in normal case. Thus this extra codeword preamble to each uncoded symbol is an overhead not only to compression ratio but to overall test application time also.

One more drawback of optimal selective Huffman is because of the number of codewords being used. Optimal selective Huffman code uses one more code word compared to the number of codewords being used in case of selective Huffman code. Suppose there are n distinct symbols to be coded, the selective Huffman code requires n codewords. However for the same case, optimal selective Huffman code requires $n+1$ codewords. The bit size of n^{th} codeword in case of optimal selective Huffman is large than the size of n^{th} codeword in case of selective Huffman code. As shown in the Table 4.1, for the third block S_2, the size of codeword is 2 for selective Huffman and 3 for optimal selective Huffman scheme. Hence,

the optimal selective code requires more transfer time compared to selective Huffman code during the minimum frequency encoded symbol and the remaining unencoded symbols.

4.2.2 The Modified Selective Huffman Code

The modified selective Huffman code is proposed here on the basis of the observation that during scan-in operations, only few input ports of the device-under-test (DUT) are used and remaining are idle. In case of single scan-chain, one normal data input port may be multiplexed with scan-in data input port to load the scan-in data serially to scan chain. In case of multiple scan-chains also, the number of input ports multiplexed for shifting the scan data is equal to number of chains. In any case, there are still a large number of idle input ports of DUT available during scan-in operation.

Further, for scan-in process, the encoded test data is being transferred serially from automatic test equipment (ATE) to DUT. In case of single scan chain, such shifting requires only one ATE channel and in case of multiple scan-chain, depending upon the coding style, it may demand more than one ATE channel. In this scenario, a large number of ATE channels are also idle during scan-in operations.

The main objective of this proposed modified selective Huffman code is to effectively use these idle input ports of DUT as well as idle channels of ATE during scan-in operations for increasing test data compression and hence effectively reduce the TAT.

In the selective Huffman code or optimal selective Huffman code, the information related to block i.e. ether it is encoded or unencoded, being transmitted serially with the encoded data itself. Unlike these schemes, in proposed code, such information is being transmitted parallel to the test data itself. Here, one signal E/N' is used to indicate the status of test data being shifted. If E/N' is 1 then, the test data being transferred is

encoded and if E/N' is 0, then the data being transferred is without encoded.

Referring to given example in Table 4.4, Table 4.7 shows the comparison of the selective Huffman code, optimal selective Huffman code and MS-Huffman code. As shown in Table 4.7, in the case of MS-Huffman also, same as other two techniques, three distinct blocks are encoded. Here when the encoded blocks S_0, S_1 and S_3 are being transmitted, the signal E/N' is high. When blocks 0001 and 0010 are transmitted without coding, the signal E/N' is low.

Table 4.7: Modified Selective Huffman Code

Distinct Patterns	Freq	Selective Huffman Code n=3	Optimal Selective Huffman Code n=3	MS-Huffman Code	
				Test data	E/N'
S0- 1010	9/20	**1**0	**0**	**0**	1
S1- 0000	5/20	**1**10	**10**	**10**	1
S2- 1111	3/20	**1**11	**110**	**11**	1
0001	2/20	**0**0001	**111**0001	**0001**	0
0010	1/20	**0**0010	**111**0010	**0010**	0
# Bits in Encoded Test data	80 (original)	**57**	**49**	**37**	
% compression		**28.75**	**38.75**	**53.75**	

Here it should be noted that all the three approach encode the same distinct blocks. The extra signal E/N' in case of MS-Huffman plays the same role as adding an extra bit to code word in selective Huffman or using one more codeword in optimal Huffman code. Moreover, while four Huffman codewords are used in case of optimal selective Huffman code, only three codewords are used for modified selective Huffman code. This has made the difference in average codeword length. In optimal Huffman codeword, the length of S_2 codeword is three while in case of MS-Huffman, it is only two. In general, if the symbols S_1, S_2,. . . .S_m are the symbols arranged in descending order of its frequency of occurrence,

than the m^{th} codeword of this sequence will have larger codeword length in case of optimal selective Huffman code compared to MS-Huffman code.

4.2.3 Mathematical Analysis

Consider two different data sets D_1 and D_2 are applied Huffman coding. The number of distinct blocks (m) is same for both the set. For the first set D_1: $p_1, p_2, \ldots. p_m$, are the occurrence frequencies of m distinct blocks $b_1, b_2, \ldots, b_m$. ($p_1 \geq p_2 \geq \ldots \geq p_m$). l_i , $i \varepsilon [1, m]$ is the length of the Huffman codeword of the each encoded distinct block b_i , $i \varepsilon [1, m]$ for set D_1. For the another set D_2 : $p'_1, p'_{2,\ldots..}, p'_m, p'_{m+1}$, are the occurrence frequencies of m+1 distinct blocks $b'_1, b'_2, \ldots, b'_m, b'_{m+1}$. l'_i , $i \varepsilon [1, m+1]$ is the length of the Huffman codeword of the each encoded distinct block b'_i , $i \varepsilon [1, m+1]$ for set D_2.

It is observed that the length of first (m-1) codewords for both the data set is same irrespective of their frequency of occurrence.

$$l_i = l'_i \; for \; i < m$$

.....(4.3)

For m^{th} codeword, the length is different by 1 bit for both the data set.

$$l_i + 1 = l'_i \; for \; i = m$$

.....(4.4)

The $(m+1)^{th}$ codeword length in set D_2 is same as m^{th} codeword length in set D_2.

$$l_m' = l_{m+1}' \; for \; i < m$$

.....(4.5)

Now the test data set T of an IP core is considered. Here the same terminology is used. It is assumed that p_1, $p_{2,\ldots\ldots}$, p_m, are the occurrence frequencies of m distinct blocks b_1, b_2, . . ., b_m that will be encoded and that $p_u = p_{m+1} + p_{m+2} + \ldots \ldots + p_k$ is the sum of the occurrence frequencies of all the remaining k-m unencoded blocks. ($p_1 \geq p_2 \geq . . . \geq p_m$).

The average block length of the proposed encoding, which is the average length of the codeword and the unencoded data blocks in the compressed test set is given by the relation

$$W_{MS-Huffman} = [l_1 p_1 + l_2 p_2 + \ldots\ldots\ldots + l_{m-1} p_{m-1}] + l_m p_m + l_b p_u \quad \ldots\ldots\ldots\ldots\ldots\ldots (4.6)$$

where l_i , $i \varepsilon$ [1, m] is the length of the Huffman codeword of the each encoded distinct block b_i , $i \varepsilon$ [1, m] and l_b is the length of the original unencoded data block.

Now for the same test set T, if optimal selective Huffman code is applied then the average block length of the optimal selective Huffman code is the average length of the codeword and the unencoded data blocks.

$$W_{optimal-Huffman} = l_1' p_1 + l_2' p_2 + \ldots\ldots\ldots + l_m' p_m + (l'_{m+1} + l_b) * p_u \quad \ldots\ldots\ldots\ldots\ldots\ldots (4.7)$$

It means

$$W_{optimal-Huffman} = l_1' p_1 + l_2' p_2 + \ldots\ldots\ldots + l_{m-1}' p_{m-1}] + l_m' p_m + l'_{m+1} p_u + l_b p_u \quad \ldots\ldots\ldots\ldots\ldots\ldots (4.8)$$

where l'_i , $i \varepsilon$ [1, m] is the length of the Huffman codeword of the each encoded distinct block b_i , $i \varepsilon$ [1, m], l'_{m+1} is the length of the codeword corresponding to the unencoded data blocks and l_b is the length of the original unencoded data block.

Here the term $l_b p_u$ is common in Equations 4.6 and 4.8. The terms in brackets in 4.6 and 4.8 are also same as per Equation 4.3. The term l_m p_m in Equation 4.6 should be compared with $l'_m p_m + l'_{m+1}\ p_u$ in Equation 4.8.

From Equation 4.4,

$$l_m' p_m + l_{m+1}' p_u = (l_m + 1) p_m + (l_m + 1) p_u \quad \text{.....(4.9)}$$

$$= l_m p_m + l_m p_u + p_m + p_u \quad \text{.....(4.10)}$$

As l_m, p_u *and* $p_m > 0$, it can be said that

$$l_m' p_m + l_{m+1}' p_u > l_m p_m \quad \text{.....(4.11)}$$

It means that

$$W_{optimal-Huffman} > W_{MS-Huffman} \quad \text{.....(4.12)}$$

It proves that W_{opt} is larger by $W_{MS\text{-}Huffman}$. From that, it can be concluded that MS-Huffman gives the better compression than optimal selective Huffman code as well as selective Huffman code. If the difference between the number of encoded data bits in case of optimal selective Huffman and MS- Huffman is denoted by T_{diff} then

$$T_{diff} = l_m p_u + p_m + p_u \quad \text{.....(4.13)}$$

As discussed earlier, the length of m^{th} codeword l_m is m-1 bits,

$$T_{diff} = (l_m + 1) p_u + p_m \quad \text{.....(4.14)}$$

$$T_{diff} = mp_u + p_m$$

.....(4.15)

4.2.4 On Chip Decoder

The proposed scheme involves statistical coding for compression of data to be stored on ATE. The decompression will be done at chip using an on-chip decoder. The compressed data from the ATE will be transferred serially to chip using a single scan-in input. Figure 4.11 illustrates the block diagram of architecture of on-chip decoder for the proposed code.

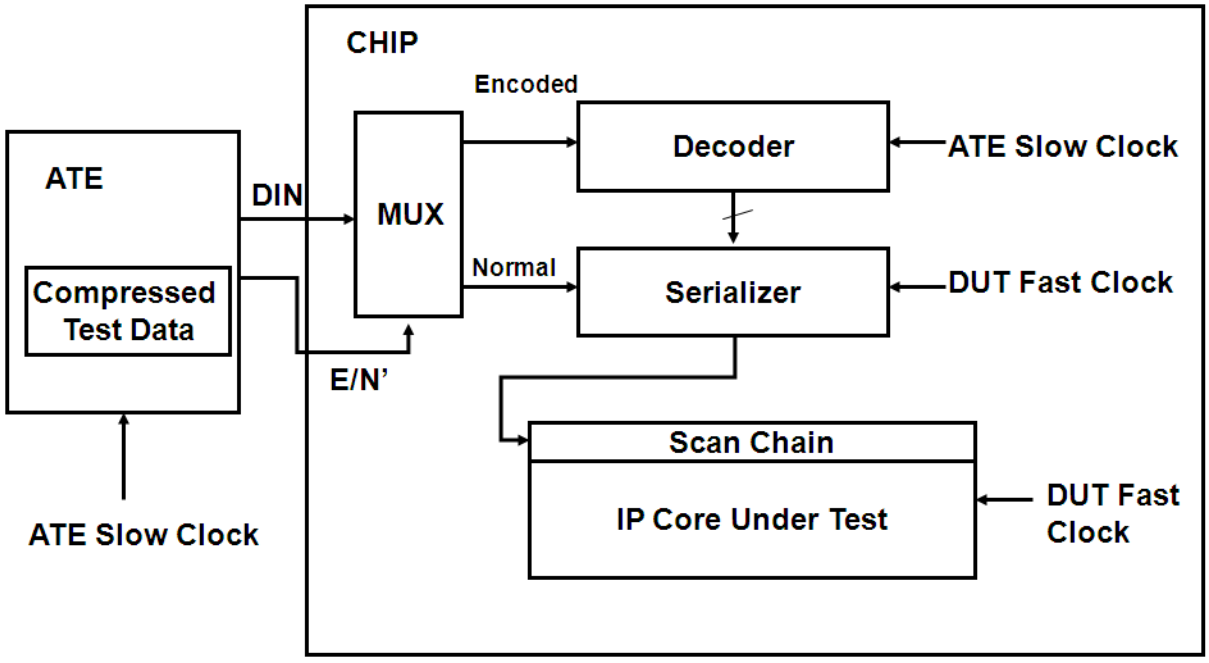

Figure 4.11: Architecture of On-chip Decoder for MS-Huffman Code

The tester channel D_{IN} shifts a stream of variable size codewords (corresponding to encoded blocks of fixed length) or fixed length unencoded blocks in serial fashion. One more channel E/N' from ATE indicates the status of the bit stream on D_{IN} channel. E/N' is high when the data bits on D_{IN} channel corresponds to codewords of corresponding

blocks. The *E/N'* is low when the data bits on D_{IN} corresponds to unencoded data blocks. *E/N'* is applied to chip through any input port of chip. The data being loaded through D_{IN} is applied to a 2-to-1 multiplexer. The selection line of this multiplexer is connected to *E/N'* signal.

If *E/N'* is low, the data on D_{IN} is normal (i.e. without coding) and hence should be applied directly to serializer bypassing the decoder. The signal *serial* will go to high indicating that the serializer is being loaded serially. If the block length is *B* bits, it takes *B* clock cycles of ATE clock to serially load the block to serializer.

If the *E/N'* is high, the data on D_{IN} is coded and should be applied to decoder which decodes the data bits. If the length of codeword is *L* bits then decoder will take *L* clock cycles of ATE clock to decode the data and then loads the corresponding block of B bits to serializer in parallel mode. The signal parallel is high indicating that the serializer is being loaded parallel. Here the size of serializer is B bits.

Once the serializer is full with valid codeword, it will have to shift the B bits to scan chain. This shifting will take B clock cycles of DUT clock. Generally, the DUT clock is faster than the ATE clock. For simplicity of diagram, control signals are not shown in Figure 4.11.

Figure 4.12 shows the FSM decoder for MS-Huffman code. Here the number of distinct blocks being coded is three: S_0, S_1 and S_2. '0', '10' and '11' are the corresponding codewords. Compared to the FSM decoder for optimal selective Huffman code shown in Figure 4.4, the FSM of Figure 4.12 is quite simple. So it can be easily shown that in spite of addition of one 2-to-1 multiplexer, the overall decoder complexity is very less in proposed architecture.

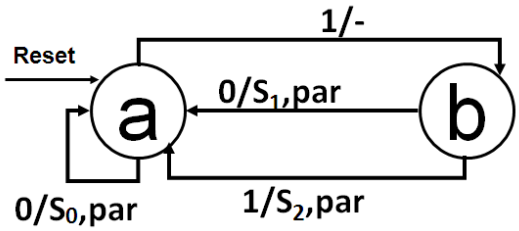

Figure 4.12: FSM Decoder for MS-Huffman Code

Figure 4.12 shows the FSM decoder for MS-Huffman code. Considering the same example given in Table 4.7, the symbol S_0 (1010) is coded with codeword '0', the symbol S_1 (0000) is coded with codeword '10' and symbol S_2 (1111) is coded with codeword '11'. When the E/N' is '1', as the bit stream of test data contains the codewords of corresponding symbols and such bit stream is directed towards the FSM. The '0' in bit stream indicates the codeword for symbol S_0 and hence corresponding symbol will be loaded to serializer. The same way, the detection of sequence '10' and '11' indicate the corresponding symbols S_1 and S_2. Compared to the FSM decoder for optimal selective Huffman code shown in Figure 4.4, the FSM of Figure 4.12 is quite simple. So it can be easily shown that in spite of addition of one 2-to-1 multiplexer, the overall decoder complexity is very less in proposed architecture.

Here it should be noted that to achieve a minimum limit of % data compression for any data set, depending up on the frequency distribution, the number of encoded symbols are to be selected. The decoder FSM is directly dependant on the number of encoded symbols.

4.2.5 Test Application Time (TAT)

The overall test application time (TAT) includes the time to transfer the encoded test data from ATE to chip as well as the time to decode the codewords of encoded data. The complexity of decoder plays a vital role in overall test application time. If the decoder takes a large amount of

time to decode the data, even if the compression is very high and data transfer time from ATE to chip is small, decoder consumes a large amount of time and effectively, there is not much reduction in TAT. Generally, the ATE operating frequency is slower compared to chip test clock.

Consider the code words which are not encoded and directly transferred to chip. While this type of blocks are transferred, E/N' is low. The data is fed to multiplexer and multiplexer route this data to serializer directly. In this case, the time required to load the data into scan chain is equal to the time required to serially load the data in to serializer and then pass it to scan chain. This transfer takes $B+1$ cycles of ATE clock. Here B is the block size. If the block is encoded, then E/N' is high and compressed data is routed to decoder FSM via multiplexer. Here the transfer time is equal to L_i ATE clock cycles where L_i is the length of the corresponding code word. It should be noted that compared to selective or optimal selective Huffman, in the proposed method, the unencoded block need not to unnecessarily pass through decoder FSM and hence the overall TAT is reduced significantly. The TAT analysis for the proposed method is analyzed with respect to the ratio of on chip test frequency and ATE operating frequency. If $\alpha = f_{chip}/f_{ATE}$ then for the proposed method, the lower bound on the size of codeword L_{min} that will allow maximum reduction in TAT is same as [26, 27]

$$L_{min} = \frac{B}{\alpha} \qquad \text{..... (4.16)}$$

After a Huffman code is identified, α-1 internal clock cycles from the current ATE cycle can be used to generate the pattern [97].

For the proposed scheme, the TAT in terms of ATE clock cycle for each distinct block of size B is given by formula:

$$TAT_{MS-Huffman_i} = p_i\,(B+1). \ . \ . \text{ if the i}^{\text{th}} \text{ block is unencoded} \quad \ldots\ldots\ldots\ldots\ldots\ldots(4.17)$$

$$= p_i\,(L_i+1). \ . \ . \text{ if for i}^{\text{th}} \text{ block, } \ L_i > B/\alpha \quad \ldots\ldots\ldots\ldots\ldots\ldots(4.18)$$

$$= p_i\,(B+1)/\alpha. \ . \ . \text{ if for i}^{\text{th}} \text{ block, } \ L_i < B/\alpha \quad \ldots\ldots\ldots\ldots\ldots\ldots(4.19)$$

where p_i is the occurrence frequency of the distinct block and L_i is the size of codeword for that distinct block.

The total test application time for compressed test data set made of total D distinct blocks is given by

$$TAT_{total_{MD-Huffman}} = \sum_{i=1}^{D} TAT_{MS-Huffman_i} \quad \ldots\ldots\ldots\ldots\ldots\ldots(4.20)$$

Considering the similar analysis for optimal selective Huffman code, the TAT in terms of ATE clock cycle for each distinct block of size B is given by formula in Equation 4.21:

$$TAT_{optimal_i} = p_i\,(B+L_i+1). \ . \ . \text{ if the i}^{\text{th}} \text{ block is unencoded but a codeword of length } L_i \text{ is prefixed to it} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(4.21)$$

$$= p_i\,(L_i+1). \ . \ . \ldots\ldots\ldots\ldots\ldots\ldots\text{if for i}^{\text{th}} \text{ block, } \ L_i > B/\alpha \quad \ldots\ldots\ldots\ldots\ldots\ldots(4.22)$$

$$= p_i\,(B+1)/\alpha. \ . \ . \ldots\ldots\ldots\ldots\ldots\ldots\text{if for i}^{\text{th}} \text{ block, } \ L_i < B/\alpha \quad \ldots\ldots\ldots\ldots\ldots\ldots(4.23)$$

The total test application time for compressed test data set made of total D distinct blocks in case of optimal selective Huffman code is given by

$$TAT_{total_{optimal}} = \sum_{i=1}^{D} TAT_{optimal_i} \quad \ldots\ldots\ldots\ldots\ldots\ldots(4.24)$$

4.2.6 Experimental Results

This section contains the experimental results for compression, test time analysis and on-chip area overhead.

Test Data Compression

Table 4.8: % Compression for Selective, Optimal Selective and MS- Huffman Code (N=8)

Circuit	**Encoded Blocks N = 8**											
	B=4			**B=8**			**B=12**			**B=16**		
	A	B	**MS**	A	B	**MS**	A	B	**MS**	A	B	**MS**
S5378	28.9	47.1	**54.2**	50.1	54.7	**61.2**	53.0	54.9	**58.7**	50.3	51.1	**55.0**
S9234	30.0	51.4	**55.5**	50.4	55.3	**61.7**	50.7	51.8	**55.9**	46.1	46.1	**49.1**
S13207	45.6	69.9	**70.3**	69.2	80.2	**81.5**	76.6	83.4	**84.3**	78.6	83.1	**84.3**
S15850	38.8	62.1	**64.2**	60.0	67.9	**71.4**	63.6	67.3	**69.4**	61.6	63.6	**66.8**
S38417	34.9	55.9	**59.9**	55.3	60.9	**67.5**	56.9	59.3	**63.8**	54.4	55.2	**60.9**
S38584	37.8	60.4	**61.9**	58.5	65.4	**70.9**	62.2	65.2	**70.1**	61.7	63.3	**67.3**

Table 4.9: % Compression for Selective, Optimal Selective and MS- Huffman Code (N=16)

Circuit	**Encoded Blocks N=16**								
	B=8			**B=12**			**B=16**		
	A	B	**MS**	A	B	**MS**	A	B	**MS**
S5378	50.2	55.8	**61.5**	55.1	57.5	**60.5**	53.8	55.0	**58.3**
S9234	50.2	57.7	**61.4**	54.2	56.8	**59.1**	51.0	51.9	**54.1**
S13207	69.2	80.6	**81.4**	77.1	84.3	**84.8**	79.7	84.5	**85.3**
S15850	59.9	69.3	**71.2**	65.6	70.4	**71.3**	64.8	67.2	**69.4**
S38417	55.5	62.8	**67.8**	58.9	61.8	**66.1**	57.3	58.6	**63.6**
S38584	58.5	66.9	**70.8**	63.9	68.0	**71.6**	64.1	66.1	**69.8**

Table 4.8 and 4.9 show the % compression for ISCAS'89 benchmark circuit test set for the number of encoded blocks is 8 and 16 respectively. For each of the case, the % compression for selective Huffman code [26], optimal selective Huffman code [27] and modified selective Huffman code is placed in column A, column B and column C respectively. The experiment result clearly shows that the % compression for MS-Huffman code has increased significantly for each of the benchmark circuit.

On-Chip Area Overhead

Table 4.10: Area Overhead for Selective, Optimal Selective and MS-Huffman Code

	Selective Decoder	Optimal Selective Decoder	MS-Huffman Decoder
# of equivalent NAND gates	388	389	362
# ports	5	5	6
# nets	225	226	217
Max clock (MHz)	1506.2	1242.4	1927.2

Table 4.10 shows the comparison in terms of NAND gates, nets and ports. Total number of equivalent NAND gates and nets is minimum in case of MS-Huffman scheme. The maximum allowable frequency is also highest in case of proposed scheme. As expected from architecture, the port requirement is one higher than the others.

Test Application Time

To compare the overall test application time in case of optimal selective Huffman compression scheme and the proposed scheme, a simulator was implemented using analysis for both the compression schemes as shown in Equations 4.13 and 4.21.

Table 4.11 shows the circuit, the compression method and the TAT obtained for five different frequency ratios: α = 1, 2, 4, 16, 32. The block size B is kept same (i.e. B = 16) for each circuit. The number of encoded distinct blocks *N* is also same for each circuit (i.e. N=8). The frequency ratios are selected to get the range of B/α from ½ (i.e. B/ α < 1) to 16 (i.e. B/ α = 1). The second column of Table has the value of α is 1. i.e. ATE clock and chip test clock has the same frequency. Even though the ATE clock will be slower compared to chip test clock but for the worst case consideration, this ratio (α = 1) has been considered. For B/α = 1 case, the maximum reduction in TAT is achieved. For any value of B/ α < 1, as expected, the TAT is same as when B/ α = 1.

Table 4.11: Test Application Time for Selective, Optimal Selective and MS-Huffman Code

ISCAS'89 Circuit	TAT (# of ATE clock cycles)									
	B/α = 16		B/α = 8		B/α = 4		B/α = 1		B/α = ½	
	α = 1		α = 2		α = 4		α = 16		α = 32	
	OPT	MS	OPT	MS	OPT	MS	OPT	***MS***	OPT	MS
S5378	25449	23760	18073	16384	14610	12846	13600	***11608***	13600	11608
S9234	41344	39280	29960	27896	25268	22704	24901	***21414***	24901	21414
S13207	168404	165200	92212	89008	55936	51961	39510	***35535***	39510	35535
S15850	80589	76992	51685	48088	38562	34556	33922	***29143***	33922	29143
S38417	174591	164736	118503	108648	93051	82167	83495	***71407***	83495	71407
S38584	199104	207888	131760	122976	98681	87927	87125	***74603***	87125	74603

The tables 4.12 and Table 4.13 describes the test application time for block size B = 32 and block size B = 8 respectively for various ISCAC'89 circuits test data

Table 4.12: TAT for Selective, Optimal Selective and MS-Huffman Code for Block Size = 32

ISCAS '89 Circuit	TAT (# of ATE clock cycles)									
	Block Size = 32 No. of Symbols to be coded = 8									
	B/α = 32		B/α = 16		B/α = 4		B/α = 1		B/α = ½	
	α = 1		α = 2		α = 8		α = 32		α = 64	
	OPT	MS	OPT	MS	OPT	MS	OPT	***MS***	OPT	MS
S5378	24652	23776	19772	18896	16256	15316	16147	**14966**	16147	14966
S9234	41074	39296	35650	33872	31764	29895	31832	**29715**	31832	29715
S13207	167388	165216	96364	94192	44021	41551	37773	**34385**	37773	34385
S15850	78958	76992	56190	54224	39750	37535	38788	**35399**	38788	35399
S38417	170252	164736	132012	126496	104447	98456	104046	**96140**	104046	96140
S38584	204206	199104	145470	140368	103187	97309	102358	**93585**	102358	93585

Table 4.13: TAT for Selective, Optimal Selective and MS-Huffman Code for Block Size = 08

ISCAS' 89 Circuit	TAT (# of ATE clock cycles)									
	Block Size = 8 No. of Symbols to be coded = 4									
	B/α = 8		B/α = 4		B/α = 2		B/α = 1		B/α = ½	
	α = 1		α = 2		α = 4		α = 8		α = 16	
	OPT	MS	OPT	MS	OPT	MS	OPT	***MS***	OPT	MS
S5378	26220	23760	17620	15160	14506	11567	14506	***11567***	14506	11567
S9234	43324	39280	29076	25032	25297	19814	25297	**19814**	25297	19814
S13207	169532	165200	91264	86932	55831	50837	55831	**50837**	55831	50837
S15850	81894	76992	49934	45032	37993	31383	37993	**31383**	37993	31383
S38417	177954	164736	113210	99992	89452	72404	89452	**72404**	89452	72404
S38584	212274	199104	130282	117112	100439	82330	100439	**82330**	100439	82330

Table 4.14: Test Application Time for Various Compression Methods

Benchmark Circuit	TAT (ATE clock cycles)					
	Statistical Code	Golomb Code	FDR	VIHC	Optimal Selective	MS-HUFFMAN
S5378	15491	11892	12968	**11516**	13600	11608
S9234	25324	**17735**	21381	**17735**	24901	21414
S13207	36784	47781	41989	36065	39510	**35535**
S15850	46067	30339	33767	30264	33922	**29143**
S38417	103604	75758	81578	74943	83495	**71407**
S38584	124011	85668	96677	85668	87125	**74603**

Table 4.14 shows the comparison of TAT for a wide spectrum of compression schemes including various run length based schemes and statistical schemes. For each compression scheme, the minimum TAT (in ATE clock cycles) out of presented TAT [97] is selected for comparison. Table 4.14 shows that out of six circuits, for four circuits, the proposed scheme gives minimum TAT. For one circuit, the proposed scheme gives a comparable TAT.

4.2.7 Observations

For the proposed modified selective Huffman code, it is proved that the proposed approach gives better test data compression compared to recently proposed selective Huffman compression scheme and optimal selective Huffman scheme. It is also verified and demonstrated with experiment results that the proposed approach does not only improve the test data compression but also reduces the overall test application time and on-chip decoder area overhead compared to recently proposed schemes.

4.3 The Existing Bit Filling Methods

When the bit filling is being done with reference to data compression, one of the advantages of implementing this Huffman encoding scheme on test cubes is that the unspecified bits can be filled with 1's and 0's in a way that the frequency distribution of the patterns becomes skewed.
This algorithm is based on bit filling method from [26]. When the block size is sufficiently small, an exact analysis can be done by considering all binary combinations (minterms) contained in the unspecified blocks. This algorithm is illustrated with an example in the following paragraphs. For a given test data set of length 12, and the block size be b = 4, let the given test cubes partitioned into a set of 9 four-bit blocks are: B = {10X1;

XX10; 1XXX; X011; 10X1; 10X1; 0X10; 101X; 1XXX}. Each unspecified block can contain from 1 (if fully specified) to 2^4 = 16 (if completely unspecified) possible binary combinations (minterms). For each of the 16 possible minterms for a block, the frequency of occurrence is determined by seeing

how many of the unspecified blocks (in set B) contain that minterm. For example, the minterm *1111* is contained in two of the unspecified blocks in the set B, while the minterm *0000* is not contained in any of the unspecified blocks. The minterm that occurs most frequently (i.e., is contained in the largest number of unspecified blocks in set B) is selected first. The Xs in each unspecified block that contains the most frequent minterm are specified so that it matches that minterm, and the unspecified block is then removed from the set B. The frequency of occurrence for each of the remaining minterms is then recomputed since the set B has been changed, and the procedure repeats until the set B is empty. This procedure maximizes the frequency of occurrence of the codewords thereby increasing the encoding efficiency of the statistical encoding. For given example, the most frequently occurring minterm is 1011. Seven of the unspecified blocks in B contain 1011, so after the first iteration, the set B will contain only XX10 and 0X10. The most frequently occurring minterm in the second iteration is 0010, which is contained in both remaining unspecified blocks. This algorithm provides an exact analysis of the frequency distribution of the minterms by considering all possibilities. However, this comes at a cost in terms of the runtime of the algorithm. It is easy to see that the algorithm is exponential in block size *b* and, hence, the use of this algorithm should be limited to small block sizes only.

4.4 The Proposed FDBAF Method

The proposed 'Frequency Dependant Bit Appending and Filling' method is described in this section with necessary example.

4.4.1 Problem Formulation

During the reverification of existing Huffman code for this thesis work, it was observed that for any Huffman based codes, when the length of test vector is a prime number, the on-chip area overhead increases to a large amount. As per Equation 4.2, the on-chip area requirement for optimal selective Huffman decoder depends on the symbol size i.e. m bits/symbol. For test vectors with length of l bits where l is a prime number, the vector cannot be divided into small symbols and hence the symbol size is too large which in turn requires large FSM and larger memory storage. The proposed approach here is to convert this undividable test vector in to divisible vectors so it can be cut it in to smaller symbols. The scheme proposed here adds some extra bits at the end of test vector such that the total number of bits per test vector is divisible by a preferred number (i.e. m bits/symbol).

The simplest approach is to append the necessary number of zeros at the end of each test vector. But when the zeroes are appended to test vectors with prime number of bits, it is observed that the total no of distinct symbols increases. Hence frequency distribution for distinct symbols is widened and the compression reduces. This had affected the selection criteria for number of symbols to be encoded and finally the area overhead and % compression.

The main concern here is that the bits appended to test vectors should not widen the frequency distribution but it should narrow down the distribution. For the solution of this problem, a scheme based on Frequency Dependant Bit Appending and Filling (FDBAF) is proposed

here. The concept is to append the bits in such a way that it does not add any new symbol but it creates the symbol same as any one of the existing symbols with the highest possible frequency. In this case, each new symbol created by bit appending will strengthen the frequency distribution of symbols and improves the overall compression. The following steps describe the proposed scheme for software implementation.

4.4.2 The FDBAF Algorithm

.....FDBAF Algorithm.....

Step: 1 Calculate the total number of test vectors (t) and total number of bits per each test vector (l).

Step: 2 For m = 2 to $\left\lceil \frac{l}{2} \right\rceil$, where m is the symbol size, do the following steps

Step: 3 Is the bits/ test vector is divisible by m? If yes, go to step:6

Step: 4 Append the m-r don't care bits at the end of test vector. Where r = rem(l, m)

Step: 5 Divide each vector into symbols of m bits

Step: 6 Separate the symbols in two types: 1. fully specified symbols and 2. partially specified symbols

Step: 7 Find out the total number of distinct symbols for both cases i.e. in case of fully specified symbols and partly specified symbols.

Step: 8 Prepare the list of distinct symbols in ascending order of frequency.

Step: 9 Take the partially specified symbol with highest frequency of occurrence and compare it with one by one all symbols in fully specified list arranged in ascending order of frequency. If the partially specified symbol gets a match for it in fully

specified symbol list, merge the partially specified symbol with that fully specified symbol and add the frequency of partially specified symbol to fully specified symbol.

Step: 10 If there is no match of partially specified symbol in the fully specified symbol list, add this partially specified symbol to fully specified list as new member of the list.

Step: 11 Rearrange the fully specified list in ascending order of frequency.

Step: 12 Repeat from step: 10 to step:13 until the partially specified symbol list is vacant.

Step: 13 If there is any symbol with don't care bit (it would be added from partially specified list because of no match available) in fully specified list, the don't care bits in the symbol should be MT filled

.....

Table 4.15: Relation between % Compression and # Bits/Symbol for FDBAF Method

# of Distinct Symbols Coded (n)	# of Bits/symbol (m)					
	2	3	4	5	6	11*
2	12.6	12.6	18.6	13.0	16.2	-1.2
3	16.6	24.9	24.1	21.3	24.1	2.0
4	21.3	27.7	27.7	26.5	28.5	5.1
5		29.6	30.8	27.3	31.6	7.9
6		30.8	31.2	27.7	35.2	10.7
7		32.0	31.6	28.1	35.6	13.4
8			32.0	28.1	36.8	16.2
9			32.4	28.5	37.5	18.6
10			33.2	29.2	38.7	20.9
11			34.8	30.0	39.9	23.3
12				30.8	41.5	25.7
13				32.0	**43.9**	28.1
14				34.0		30.4
15						32.8
16						35.6
17						38.3
18						41.1
19						**43.9**

In Table 4.15, the % compression for various values of number of bits per symbol (*m*) and number of symbols coded (*n*) in case of optimal selective Huffman code for the ISCAS'89 benchmark circuit 74L85 is presented. Column 7 of the Table indicates the % compression when the symbol size is equal to no. of bits per vector. i.e. no appending is done. While column 2 to 6 is the % compression for different symbol size i.e. appending is done here. For the case of bit appending, the maximum possible data compression is 43.9% for which the number of bits per symbol (*m*) is 6 and number of symbols encoded (*n*) is 13. Now for the nearly same % compression (coincidently in this case, both are exactly same) in case of no bit appending is done (i.e. in column 7), the number of bits per symbol (*m*) is 11 and number of symbols encoded (*n*) is 19.

4.4.3 On-Chip Decoder

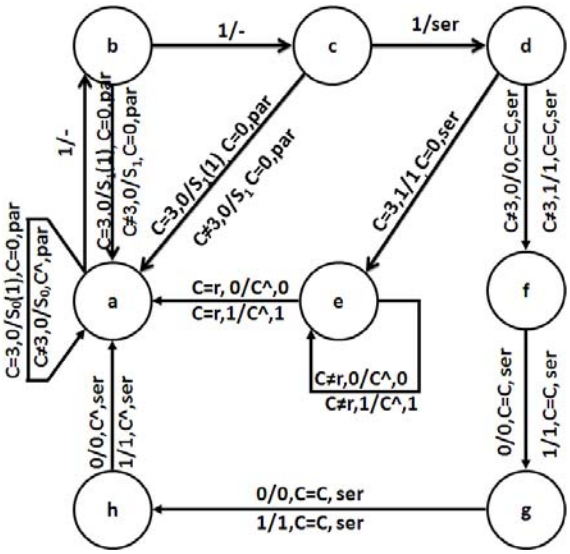

Figure 4.13: FSM Decoder for FDBAF Algorithm

Figure 4.13 describes an implementation of the decoder using simple FSM for a specific case of 3 bits/symbols and 4 symbols to be encoded. The inputs and outputs are same as in FSM for optimal selective

Huffman decoder. Only one signal count (*C*) is added to indicate the index of symbol being decoded. Counter counts up to total symbols per vector.

The concept is divided in to two parts.

Part: 1 If *C* is less than 3, it means the code word is not corresponding to last symbol of test vector. On arrival of the first bit of the codeword, the decoder branches on each bit one at a time until it reaches the end of the codeword at which point it does a parallel load of the appropriate m-bit block into the parallel and *C* is incremented by 1 to indicate that one codeword is successfully decoded and corresponding symbol is loaded. The codeword "111" indicates that the next *m* bits are original uncoded data bits. So *ser* signal will be high for next *m* cycles and will load next m bits serially from coded data.

Part: 2 If *C* is equal to 3, the code word is corresponding to last symbol of test vector. On arrival of the first bit of the codeword, the decoder branches on each bit one at a time until it reaches the end of the codeword at which point it does a parallel load of the appropriate *r* bits from *m* bits symbol into the parallel, *C* is set to 0 to indicate that one complete test vector is successfully decoded and corresponding symbol is loaded.. The codeword "111" indicates that the next *m* bits are original uncoded data bits. Only *r* bits (1 bit in this case) should be passed to DUT through serial load. Remaining *m-r* bits should be discarded. The same counter is used to count up to *r* and when it reaches to *r*, FSM moves to initial state.

Here the extra hardware requirement compared to optimal Huffman decoder (shown in Figure 4.3) is one counter of $count = \left\lceil \frac{l}{m} \right\rceil$ bits to indicate that the symbol created by decoding of codeword is a complete symbol or it contains few of the appended bits which later on needs to be removed.

4.4.4 Experimental Results

Table 4.16 compares the % compression in case of test data is processed with MT fill algorithm and FDBAF algorithm respectively for ISCAS'89 circuits.

Table 4.16: % Compression for Proposed FDBAF Method

ISCAS'89 Circuits	% Compression (MS Huffman Code, Block Size = 8 Bits)	
	MT Fill Algorithm	FDBAF Algorithm
s5378	52.11	**60.53**
s9234	45.17	**56.34**
s13207	72.17	**78.85**
s15850	60.34	**64.11**
s38417	58.41	**60.54**
s38584	56.88	**65.61**

4.4.5 Observations

The proposed FDBAF algorithm prepares the test data to increase the compression in case of statistical codes. With MS-Huffman code it gives comparatively very high results for compression.

4.5 Inferences

After study of existing statistical codes, considering the area overhead and complexity of decoder, it was found that the inefficiency of optimal selective Huffman code [27] is that it prefixes a Huffman codeword to each of uncoded symbol which not only affects the compression but also

highly affects overall test application time. This inefficiency is removed in the proposed modified selective Huffman code in which the Huffman code words are used for symbol to be coded and they are not preamble to uncoded symbol. It is proved that this modified selective Huffman code reduces the overall test application time compared to selective Huffman and optimal selective Huffman code because here the coded data has to pass through a very simple FSM which results into less clock cycles for decoding compared to others.

The simplicity of FSM decoder for modified selective Huffman code also gives the better result in terms of on-chip area overhead. The 'Frequency Dependant Bit Appending and Filling' in combination with MS-Huffman code further improves the compression.

Chapter 5

Conclusion and Future Work

This chapter concludes the thesis and discusses the possible future work.

5.1 Conclusion

There are two major concerns for the existing methodology for fabrication testing of IC: test cost and test power.

With the advances in fabrication process for IC, the testing cost is becoming a dominating part of the overall cost of an IC. The testing cost is directly proportional to test application time which again depends on test data volume. This way, one of the solutions to problem of lowering test cost can be test data compression.

Smaller geometry and lower power supply has made 'test power' a major concern. Test power is strongly related to switching activity during test.

To reduce TTM, the Intellectual Property (IP) cores are widely used in SoCs. The hidden structures of IP cores have further exaggerated the testing issues like test data compression and test power. The test data compression methods and test power reduction methods which are been specifically developed for ASIC, can not be directly applicable to SoC with IP cores.

In this thesis, the techniques for increasing test data compression and reducing the switching activity are developed in context of IP core based SoC i.e. these techniques require neither the internal details of IP core nor the test development tools. The effectiveness of these methods is proved through the extensive simulation results using MATLAB and C. In addition, it is also demonstrated that these methods do not add significant area overhead through the adequate synthesis results using VHDL, Modelsim and LeonardoSpectrum. One test data compression method is proposed and it is proved mathematically that this method reduces the overall test application time (i.e. test data transfer time plus decoding time) compared to existing methods.

The main contributions of this thesis are as follow:

In Chapter 2, various test data compression methods like linear decompression based compression methods, broadcast scan based compression methods and code based compression methods are examined for the suitability to IP cores. After surveying the requirements of linear decompression and broadcast scan based methods, it is inferred that they are not suitable to hidden structure of IP cores. On the basis of its suitability to IP cores, the code based method is further selected for this thesis work [publication ref. C-1]. After classifying the code based methods into four categories: statistical coding, run length coding, dictionary coding and constructive coding; it was observed that the run length code and statistical code based test data compression methods are efficient in terms of compression and on-chip area.

The same way, during the literature survey on test power in Chapter 2, the various switching activity reduction techniques applicable to ASIC is studied. During this survey, these techniques are classified into three categories: switching reduction applicable with BIST [publication ref. C-8], switching reduction applicable with DFT and switching reduction

applicable with external testing. It is observed that the switching reduction technique applicable to external testing can be further explored for IP cores and through a detailed survey [publication ref. J-3], it was inferred that out of the available variants of switching reduction techniques applicable to external testing, the 'test vector reordering' and 'don't care bit filling' methods can be adopted to reduce the test power.

In short, Chapter 2 infers that the synergy of 'reordering' and 'don't care bit filling' can be used to solve the test cost and test power together. The thesis work deals with the concepts of 'test vector reordering' and 'don't care bit filling' methods in conjunction with run length codes and statistical codes to increase test data compression and to reduce the test power as well as on-chip area overhead.

In Chapter 3, existing run length codes are analyzed through implementing them using MATLAB and C code and extensive simulation results. On the basis of this analysis result, the run based bit filling method is proposed [publication ref. C-4]. This method effectively demonstrates that instead of filling the don't care bits simply with '0' irrespective of considering the type of runs used for coding, if the don't care bit filling is done considering the type of run used, it enhances the run length and the compression is increased [publication ref. J-1]. The proposed test data processing method 'Hamming Distance Based Reordering-Columnwise Bit Filling-Difference Vector (HDR-CBF-DV)' further improves the compression by reordering the test vectors, applying don't care bit filling corresponding to difference vector mechanism and using the difference vectors [publication ref. C-3]. The next proposed '2-Dimensional Reordering with Power Efficient Bit Filling' method for data processing, improves the scan-in power with test data compression [publication ref. C-5]. However, the trade off is larger on-chip area for decoder [publication ref. C-6]. The 'Weighted Transition Based Reordering – Columnwise Bit Filling – Difference Vector (WTR-CBF-DV)' reduces the scan-in power without scarifying on-chip area [publication ref. J-5].

From the proposed methods for run length based test data compression method in Chapter 3, it can be said that if the WTR-CBF-DV is applied in combination with FDR coding, it gives the better results in terms of compression, scan-in power and area overhead.

In Chapter 4, existing statistical codes are analyzed through implementing them using MATLAB and C code and extensive simulation results [publication ref. C-7 and J-4]. On the basis of this analysis result, the 'Frequency Dependant Bit Appending and Filling' is proposed to strengthen the frequency distribution of symbols and hence to improve the compression [publication ref. C-2]. To effectively reduce the test application time and to increase the compression without adding any area overhead, the 'Modified Selective Huffman (MS-Huffman) code' is proposed [publication ref. J-2]. Its effectiveness in improvement in compression compared to very recently proposed selective Huffman code [26] and optimal selective Huffman code [27] is proved mathematically and demonstrated with sufficient experimental results. In chapter 4 it is clearly shown that the use of FDBAF method in combination with Modified Selective Huffman code gives the better results in terms of test time, test power and on-chip area compared to other methods available in literature.

"In Chapter 3, the run length code based method WTR-CBF-DV is proposed and it has been shown that it gives the more compression and less power compared to all other existing run length based codes without any extra area overhead compared to existing methods. In Chapter 4, the statistical code based MS-Huffman code is proposed and it has been shown that the MS-Huffman code gives the enhanced compression, less test application time (TAT) and less area overhead compared to the existing methods like Selective Huffman and Optimal Selective Huffman codes.

The trade off for both of the above cases is the marginal computational load added because of test data processing schemes. It should be further

noted that this computational load is for one time only before the test data is stored in ATE and the test data need not to be processed every time the chip is being tested. Hence this one time computational load is negligible.

The comparison of these two proposed methods shows that the run length code based proposed “WTR-CBF-DV method in combination with FDR coding” gives higher compression result compared to the statistical code based proposed “FDBAF in combination with MS-Huffman coding”. The trade off here is area penalty. The WTR-CBF-DV with FDR code requires more area compared to FDBAF with MS-Huffman code. For both the methods the average and peak power are nearly same.

5.2 Future Work

There are several possible extensions to the work presented in this thesis. In this section possible extensions, directly related to the work are presented. A few possible directions of future work that are beyond the scope of this thesis will also be discussed.

Here follows a list of possible extensions for each of the problems described in this thesis:

- Here various proposed methods for compression, reordering and bit filling are analyzed for % compression, on-chip area overhead, dynamic power and overall test application time. The compression of test data may cause the adverse effect on error-resilience, i.e. the capability of a test data stream which is transferred from automatic test equipment (ATE) to the device under test (DUT) to tolerate errors. These proposed methods can be further analyzed in terms of error-resilience property.
- Here the fault coverage is considered using stuck-at faults only. The don't care bit filling does not affect the stuck-at fault coverage. It is true that the stuck-at faults plays a significant role in overall

fault coverage but more precise considerations, other faults such as transitions fault, transistor stuck open or stuck-short, bridging faults etc. need to evaluates. The don't care bit filling can affect to such kind of fault coverage.

- The test data compression reduces the transfer time for scan-in test vector from ATE to DUT. The same way the lossy compaction is used to reduce the test data transfer time of output response from DUT to ATE. The proposed test vector reordering methods can also affect this compaction. This effects need to be further analyzed.
- Even though, the don't care bit filling in scan vector reduces the scan-power, the type of bit filling (i.e. either filled by 1 or filled by 0) can have the different affect on its capture power. While considering overall test power, this change in capture power should be considered.

References

[1] H. Chang *et al., Surviving The SoC Revolution : A Guide to Platform based Design.* Kluwer Academic Publishers, 1999.

[2] M. Abramovici *et al., Digital Systems Testing and Testable Design.* Jacoba, 1997.

[3] "International technology roadmap for semiconductors (ITRS),"Semiconductor Industry Association, 1999.

[4] *"International technology roadmap for semiconductors (ITRS),"* Semiconductor Industry Association, 2001.

[5] *"International technology roadmap for semiconductors (ITRS),"* Semiconductor Industry Association, 2009.

[6] R. Press and J. Jahangir, "Test compression", Test and Measurement World web site at www.tmworld.com, p. article ID CA6375810, January 2006.

[7] N. Touba, *"Survey of test vector compression techniques,"* IEEE Design and Test of Computers, July 2006.

[8] J. Rajaski *et al., "A SmartBIST variant with guaranteed encoding,"* Proceedings of the Asian Test Symposium, 2001.

[9] J. Rajaski, *"Embedded deterministic test for low cost manufacturing test,"* Proceedings of the International Test Conference, pp. 301-310, 2002.

[10] F. Pollack, *"New microarchitecture challenges in the coming generations of CMOS process technologies,"* Proceedings of the 32nd annual IEEE international Symposium on Microarchitecture, pp. 167-172, 1999.

[11] Y. Bonhomme *et al., "Test power: A big issue in large SoC designs,"* Proceedings of the First IEEE Workshop on Electronic Design, Test And Applications, pp. 447-449, 2002.

[12] S.Kajihara *et al., "Test data compression using don't care identification and statistical encoding,"* Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications (DELTA 02), January 2002.

[13] A. Larsson, *"Test Optimization for Core Based System on Chip".* Department of Computer and Information Science, Linkpings Universitet Ph. D. Thesis, 2008.

[14] E. Volkerink and S. Mitra, *"Efficient seed utilization for reseeding based compression,"* Proceedings of the 21st VLSI Test Symposium (VTS 03), pp. 232-237, 2003.

[15] B. Koenemann, *"LFSR coded test patterns for scan designs,"* Proceedings of the European Test Conf. (ETC 91), pp. 237-242, 1991.

[16] I. Bayraktaroglu and A. Orailoglu, *"Concurrent application of compaction and compression for test time and data volume reduction in scan designs,"* IEEE Transactions on Computers, Volume 52, Issue 11, pp. 1480-1489, November 2003.

[17] S. Mitra and K. Kim, *"XPAND: An efficient test stimulus compression technique,"* IEEE Transactions on Computers, Volume 55, Issue 02, pp. 163-173, February 2006.

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

[19] J. Lee and N.Touba, *"Combining linear and non-linear test vector compression using correlation-based rectangular coding,"* Proceedings of the 24th VLSI Test Symposium (VTS 06), pp. 252-257, 2006.

[20] K. Lee et al., *"Using a single input to support multiple scan chains,"* Proceedings of the International Conference Computer Aided Design (ICCAD 98), pp. 74-78, 1998.

[21] K. Lee, *"Enhancement of the Illinois scan architecture for use with multiple scan inputs,"* Proceedings of the IEEE Computer Society Annual Symposium VLSI (ISVLSI 04), pp. 167-172, 2004.

[22] L. Wang *et al., "VirtualScan: A new compressed scan technology for test cost reduction,"* Proceedings of the 22nd VLSI Test Symposium (VTS 04), pp. 916-925, 2004.

[23] L. Wang, *"UltraScan: Using time division demultiplexing multiplexing with VirtualScan for test cost reduction,"* Proceedings of the 23rd VLSI Test Symposium (VTS 05), pp. 946-953, 2005.

[24] P. Gonciari et al., *"Variable length input Huffman coding for system on a chip test,"* IEEE Transactions on Computer Aided Design , Volume 22 Issue 6, pp. 783-796, June 2003.

[25] K. C. Vikram Iyengar, *"An efficient finite-state machine implementation of Huffman decoders,"* Information Processing Letters 64, 1997.

[26] X. Kavousianos, *"An efficient test vector compression scheme using selective Huffman coding,"* IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Volume 22, Issue 6, June 2003.

[27] X. Kavousianos *et al., "Optimal selective Huffman coding for test data compression,"* IEEE Transactions on Computers, Volume 56 Issue 8, August 2007.

[28] Hashempour and F.Lombardis, *"Application of arithmetic coding to compression of VLSI test data,"* IEEE Transactions on Computers, Volume 54, Issue 9, September 2005.

[29] Hashempour and F.Lombardis, *"Compression of VLSI test data by arithmetic coding,"* Proceedings of 19th IEEE International Symposium on the Defect and Fault Tolerance in VLSI Systems, October 2004.

[30] S.Kajihara and K. Miyase, *"On identifying don't care inputs of test patterns for combinational circuits,"* Proceedings of the IEEE International Conference on Computer Aided Design (ICCAD 01), November 2001.

[31] S.Kajihara *et al., "Test data compression using don't care identification and statistical encoding,"* Proceedings of the 11th Asian Test Symposium, Novembaer 2002.

[32] M. Nourani and M.Tehranipour, *"RL Huffman encoding for test compression and power reduction in scan applications,"* Transactions on Design

Automation of Electronic Systems (TODAES) , Volume 10 Issue 1, January 2005.

[33] X. Kavousianos et al., *"Efficient test data compression for IP cores using multilevel Huffman coding,"* Proceedings of the conference on Design, automation and test in Europe, March 2006.

[34] X. Kavousianos et al., *"Multilevel Huffman test data compression for IP cores with multiple scan chains,"* IEEE Transactions on Very Large Scale Integration (VLSI) Systems , Volume 16 Issue 7, July 2008.

[35] X. Kavousianos et al., *"Test data compression based on variable-to-variable Huffman encoding with codeword reusability,"* IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems , Volume 27 Issue 7, July 2008.

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

[37] A. Chandra and K. Chakrabarty, *"Test data compression for system on a chip using Golomb codes,"* Proceedings of the 18th IEEE VLSI Test Symposium, (VTS 00), 2000.

[38] A. Chandra and K. Chakrabarty, *"Efficient test data compression and decompression for system on a chip using internal scan chains and Golomb coding,"* Proceedings of the conference of Design, automation and test in Europe (DATE 01), March 2001.

[39] A. Chandra and K. Chakrabarty, *"Frequency directed run length (FDR) codes with application to system on a chip test data compression,"* Proceedings of the 19th IEEE VLSI Test Symposium (VTS 01), March 2001.

[40] A. Chandra and K. Chakrabarty, *"Test data compression and test resource partitioning for system on a chip using frequency directed run length (FDR) codes,"* IEEE Transactions on Computers , Volume 52 Issue 8, August 2003.

[41] A. E. Maleh and R. A. Abaji, *"Extended frequency directed run length code with improved application to system on chip test data compression,"* Proceedings of the International Conference on Electronics, Circuits and Systems, pp. 449-452, September 2002.

[42] A. Chandra and K. Chakrabarty, *"Reduction of SoC test data volume, scan power and testing time using alternating run length codes,"* Proceedings of the 39th conference on Design Automation, June 2002.

[43] S. Hellebrand and A. Wrtenberger, *"Alternating run length coding: A technique for improved test data compression,"* Handouts 3rd IEEE International Workshop on Test Resource Partitioning, Baltimore, MD, USA, October 2002.

[44] C. Giri et al., *"Test Data Compression by Split-VIHC (SVIHC),"* Proceedings of the International Conference on Computing: Theory and Applications (ICCTA 07), March 2007.

[45] J. Feng and G. Li, *"A test data compression method for system on chip,"* Proceedings of 4th IEEE International Symposium on Electronic Design, Test and Applications (DELTA 08), 2008.

[46] S. Kajihara et al., *"On combining pinpoint test set relaxation and runlength codes for reducing test data volume,"* Proceedings of 4th IEEE

International Symposium on Electronic Design, Test and Applications (DELTA 08), 2008.

[47] X. Ruan and R. Katti, *"Data independent pattern run length compression for testing embedded cores in SoCs,"* IEEE Transactions on Computers , Volume 56 Issue 4.

[48] H. Fang et al., *"RunBasedReordering: A novel approach for test data compression and scan power,"* Proceedings of IEEE International Conference on Asia South Pacific design automation (ASP DAC 07), January 2007.

[49] W. Zhan et al., *"Test data compression scheme based on variable to fixed plus variable length coding,"* Journal of Systems Architecture: the EUROMICRO Journal , Volume 53 Issue 11,, November 2007.

[50] G. Ma et al., *"Combined partial test vector reuse and FDR coding for two dimensional SoC test compression,"* Proceedings of International Conference on Internet Computing in Science and Engineering - Volume 00 , Volume 00 (ICICSE 08), January 2008.

[51] X. Sun et al., *"Combining dictionary coding and LFSR reseeding for test data compression,"* Proceedings of the 41st annual conference on Design automation, June 2004.

[52] K. Basu and P. Mishra, *"A novel test data compression technique using application aware bitmask and dictionary selection methods,"* Proceedings of the 18th ACM Great Lakes symposium on VLSI (GLSVLSI 08), May 2008.

[53] F. Wolf and Papachristou, *"Multiscan based test compression and hardware decompression using LZ77,"* Proceedings of the International Test Conference (ITC 02), pp. 331-339, April 2002.

[54] M. Knieser et al., *"A technique for high ratio LZW compression,"* Proceedings of the conference Design, Automation and Test in Europe (DATE 03), pp. 116- 121, April 2003.

[55] A. Wrtenberger et al., *"Data compression for multiple scan chains using dictionaries with corrections,"* Proceedings of the International Test Conference (ITC04), pp. 926-935, April 2004.

[56] T. Kim et al., "An effective hybrid test data compression method using scan chain compaction and dictionary-based scheme," Proceedings of the 17th IEEE Asian Test Symposium, 2008.

[57] S. Reda and A. Orailoglu, *"Reducing test application time through test data mutation encoding,"* Proceedings of Design, Automation, and Test in Europe (DATE 02), pp. 387-393, 2002.

[58] Z. Wang and K. Chakrabarty, *"Test data compression for IP embedded cores using selective encoding of scan slices,"* Proceedings of the International Test Conference (ITC 05), 2005.

[59] P. Girard, *"Survey of Low power testing of VLSI circuits,"* IEEE Journal on Design and Test of Computers Volume 19, pp. 80-90, 2002.

[60] Y. H. Jia LI, Qiang XU and X. LI, *"On reducing both shift and capture power for scan-based testing,"* In the proceedings of IEEE Asia and South Pacific Design Automation Conference ASPDAC, pp. 653-658, January 2008.

[61] N. Ahmed et al., *"Low power pattern generation for BIST architecture,"* Proceedings of IEEE International Symposium on Circuits And Systems, pp. 689-692, 2004.

[62] N. Ahmed et al., *"Low power test pattern generator design for BIST via non uniform cellular automata,"* Proceedings of International Symposium on VLSI Design, Automation And Test, pp. 212-215, 2005.

[63] Y. Kim et al., *"A new low power test pattern generator using a transition monitoring window based on BIST architecture,"* Proceedings of 14th Asian Test Symposium, pp. 230-235, 2005.

[64] K. Gunavathi et al., *"A novel BIST TPG for testing of VLSI circuits,"* Proceedings of First International Conference on Industrial And Information Systems, pp. 106-114, 2006.

[65] B. Zhou et al., *"A new low power test pattern generator using a variable length ringcounter,"* Proceedings of IEEE Conference on Quality of Electronic Design, pp. 248-252, 2009.

[66] S. W. Wang et al., *"A BIST TPG for low power dissipation and high fault coverage,"* IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Volume 15, pp. 777-789, 2007.

[67] B. Zhou et al., *"Low transition test pattern generation for BIST based applications,"* IEEE Transactions on Computers Volume 57, pp. 303-315, 2007.

[68] C. Bei et al., *"A low power deterministic test pattern generator for BIST based on cellular automata,"* Proceedings of 4th IEEE International Symposium on Electronic Design, Test And Applications, pp. 266-269, 2008.

[69] H. LiGang and P. XiaoHong, *"A low power dynamic pseudo random bit generator for test pattern generation,"* Proceedings of 9th International Conference on Solid State And Integrated Circuit Technology, pp. 2079-2082, 2008.

[70] P. Girard et al., *"Low energy BIST design impact of the LFSR TPG parameters on the weighted switching activity,"* Proceedings of IEEE International Symposium on Circuits and Systems, 1999.

[71] S. Manich et al., *"Low power BIST by filtering non detecting vectors,"* Proceedings of the IEEE European Test Workshop, pp. 317-319.

[72] S. Hatami et al., *"A low power scan path architecture,"* Proceedings of International Symposium on Circuits And Systems, pp. 5278-5281, 2005.

[73] S. Bhunia et al., *"Power reduction in test per scan BIST with supply gating and efficient scan partitioning,"* Proceedings of Sixth International Symposium on Quality of Electronic Design, pp. 453-458, 2005.

[74] X. Qiang et al., *"Pattern directed circuit virtual partitioning for test power reduction,"* Proceedings of IEEE International Test Conference, pp. 1-10, 2007.

[75] S. Lu et al., *"Low power built in self test techniques for embedded SRAMs,"* VLSI Design Archive, Volume 2007, Issue 2, pp. 1-7, 2007.

[76] W. Yuejian and I. Andre, *"Low power SoC memory BIST,"* Proceedings of 21st IEEE International Symposium on Defect And Fault Tolerance In VLSI Systems, pp. 197-205, 2006.

[77] Y. Xiaoming and M. Abramovici, *"Sequential circuit ATPG using combinational algorithms,"* IEEE Transactions on Computer Aided Design of Integrated Circuits And Systems, Volume 24, pp. 1294-1310, 2005.

[78] C. MinHao and M. Li, *"Jump scan: A DFT technique for low power testing,"* Proceedings of 23rd IEEE VLSI Test Symposium, pp. 277-282, 2005.

[79] I. Polian et al., *"Power droop testing,"* Proceedings of the IEEE International Conference on Computer Design (ICCD 06), pp. 243-250, 2006.

[80] H. Fai and N. Nicolici, *"Automated scan chain division for reducing shift and capture power during broadside at speed test,"* IEEE Transactions on Computer Aided Design of Integrated Circuits And Systems, Volume 27, pp. 2092-2097, 2008.

[81] J. C. H. Lin, T. Li, *"Simultaneous capture and shift power reduction test pattern generator for scan testing,"* IET Journal on Computers & Digital Techniques, Volume 2, pp. 132-141, 2008.

[82] S. Wang et al., *"Low peak power ATPG for n detection test,"* Proceedings of IEEE International Symposium on Circuits And Systems, pp. 1993-1996, 2009.

[83] M. Elshoukry et al., *"Partial gating optimization for power reduction during test application,"* Proceedings of 14th Asian Test Symposium, pp. 242-247, 2005.

[84] P. Girard et al., *"Reducing power consumption during test application by test vector ordering,"* Proceedings of IEEE International Symposium on Circuits and Systems, p. CD ROM, 1998.

[85] H. N. J. M. P. Flores, J. Costa and J. Marques-Silva, *"Assignment and reordering of incompletely specified pattern sequences targeting minimum power dissipation,"* Proceedings of the VLSID, p. 3741, 1999.

[86] I. P. V. Dabholkar, S. Chakravarty and S. Reddy, *"Techniques for minimizing power dissipation in scan and combinational circuits during test application,"* IEEE TCAD, p. 13251333, 1998.

[87] H. Hashempour and F. Lombardi, *"Evaluation of heuristic techniques for test vector ordering,"* Proceedings of the Great Lake Symposium on VLSI (GLSVLSI-04), pp. 96-99, January 2004.

[88] S. Chattopadhyay and N. Choudhary, *"Genetic algorithm based approach for low power combinational circuit testing,"* Proceedings of the VLSID, vol. pp. 552559, 2003.

[89] K. Paramasivam et al., *"Algorithm for low power combinational circuit testing,"* Proceedings of IEEE Region 10 Conference, TENCON, pp. 336-339, 2004.

[90] S. Roy et al., *"Artificial intelligence approach to test vector reordering for dynamic power reduction during VLSI testing,"* Proceedings of IEEE Region 10 Conference, TENCON, pp. 1-6, 2008.

[91] S. Wang et al., *"Low capture power test generation for launch off capture transition test based on don't care filling,"* Proceedings of IEEE International Symposium on Circuits And Systems, pp. 3683-3686, 2007.

[92] S. Kundu and S. Chattopadhyay, *"Efficient don't care filing for power reduction during testing,"* Proceedings of IEEE International Conference on

Advances In Recent Technologies In Communication And Computing, pp. 319-323, 2009.

[93] Z. Chen et al., *"Scan chain configuration based X filling for low power and high quality testing,"* IET Journal on Computers and Digital Techniques, Volume 4, pp. 1-13, 2009.

[94] J. Yang and Q. Xu, *"State sensitive x filling scheme for scan capture power reduction,"* IEEE Transactions on Computer-Aided Design of Integrated Circuits And Systems, Volume 27, pp. 1338-1343, 2008.

[95] T. Maiti and S. Chattopadhyay, *"Don't care filling for power minimization in VLSI circuit testing,"* Proceedings of IEEE International Symposium on Circuits And Systems, pp. 2637-2640, 2008.

[96] K. Balakrishnan and N. Touba, *"Relating entropy theory to test data compression,"* Proceedings of the European Test Symposium (ETS 04), May 2004.

[97] P. Gonciari et al., *"Improving compression ratio, area overhead, and test application time for system on a chip test data compression decompression,"* Proceedings of the International Conference on Design, automation and test in Europe, March 2002.

List of Publications

Papers in Refereed Journals

[J-1] U. Mehta, K. Dasgupta, and N. Devashrayee, “Run length based test data compression techniques: How far from entropy and power bounds," An Open Access International Journal "VLSI Design", vol. 2010, article ID 670476, pp. 1-8, January 2010.

[J-2] U. Mehta, K. Dasgupta, and N. Devashrayee, “Modified selective Huffman coding for optimization of test data compression, test application time and area overhead," International Journal of Electronics Testing: Theory and Application (JETTA), Springer Publications, vol. 26, no. 6, pp. 679-688, December 2010.

[J-3] U. Mehta, K. Dasgupta, and N. Devashrayee, “Suitability of various low-power testing techniques for IP core-based SoC: A survey," An Open Access International Journal "VLSI Design", vol. 2011, article ID 948926, pp. 1-7, January 2011.

[J-4] U. Mehta and K. Bhavasar, “Analysis of Statistical Code Based Techniques for Optimization of Test Data Compression and Area Overhead," International Journal of Computer Application (ISBN: 978-93-80747-95-7), vol. 18, no. 3, pp. 30-34, April, 2011.

[J-5] U. Mehta, K. Dasgupta, and N. Devashrayee, “Weighted transition Based Reordering, Columwise Bit Filling and Difference Vector –A Power Aware Test Data Compression Method," An Open Access International Journal "VLSI Design", Special issue on “CAD for Gigascale SoC Design and Verification Issues”, (in press).

Presentations in Conferences

[C-1] U. Mehta, K. Dasgupta, and N. Devashrayee, “Survey of test data compression techniques emphasizing code based schemes," Proceedings of 12th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 09). Patras, Greece: IEEE, August 2009, pp. 617-620.

* cited by 5 as per IEEExplore

[C-2] U. Mehta, K. Dasgupta, and N. Devashrayee, “Frequency dependant bit appending: An enhancement to statistical codes for test data compression," Proceedings of 3rd IEEE India Conference (INDICON 09). DAIICT, Gandhinagar, IEEE, December 2009, pp. 301-304.

*cited by 4 as per IEEExplore

[C-3] U. Mehta, K. Dasgupta, and N. Devashrayee, “Hamming distance based reordering and column wise bit stuffing with difference vector: A better scheme for test data compression with run length based codes," Proceedings of 23rd International Conference on VLSI Design (VLSID 10). Bangalore: IEEE, 2010, pp. 33-38.

*cited by 1 as per IEEExplore

[C-4] U. Mehta, K. Dasgupta, and N. Devashrayee, “Combining unspecified test data bit filling methods and run length based codes to estimate compression, power and area overhead," IEEE Asia Pacific 33rd Conference on Circuits and Systems (APCCS-10). Kuala-Lumpur, Malaysia: IEEE, December 2010, pp. (on CD).

[C-5] U. Mehta, K. Dasgupta, and N. Devashrayee, “Hamming distance based 2-D reordering with power efficient don’t care bit filling optimizing the test data compression method." 9th International Symposium on System-on-Chip (SoC 2010), Tampare, Finland, 2010, pp. 1-7.

[C-6] U. Mehta, K. Dasgupta, N. Devashrayee, and K. Choksi, “Hamming distance based distributed scan chain reordering for test power optimization," Proceedings of 4th IEEE India Conference (INDICON 10). Jadhavpur University, Kolkatta, IEEE, December 2010, p. on CD.

[C-7] U. Mehta, K. Dasgupta, N. Devashrayee and K. Bhavsar, “Analysis of test data compression methods using Huffman code." Proceedings of International Conference on Current Trends in Technology, NUiCONE-10, December 2010, p. on CD.

[C-8] U. Mehta, K. Dasgupta, N. Devashrayee and H. Parmar, “Low power testing architecture for Built-in-Self-Test." Proceedings of International Conference on Current Trends in Technology, NUiCONE-10, December 2010, pp. (on CD).

[C-9] U. Mehta, K. Dasgupta, and N. Devashrayee, “Benchmark circuits required for EDA tools used with VLSI testing: A complete survey." Proceedings of International Conference on Wireless Networks and Embedded Systems (WECON 08), October 2008, pp. 364-368.

Annexure-I

Synthesis Reports

Decoders for Run Length Codes

Golomb Decoder

****

Cell: golomb View: xyz Library: chintan_and1_lib

****

Cell	Library	References	Total Area
PadInC	tsmc035_typ	1 x	
PadOut	tsmc035_typ	2 x	
and02	tsmc035_typ	4 x	1 5 gates
and03	tsmc035_typ	2 x	2 3 gates
aoi21	tsmc035_typ	5 x	1 6 gates
aoi22	tsmc035_typ	1 x	1 1 gates
buf02	tsmc035_typ	21 x	1 21 gates
dff	tsmc035_typ	9 x	5 43 gates
fake_gnd	tsmc035_typ	1 x	
inv01	tsmc035_typ	34 x	1 26 gates
inv02	tsmc035_typ	14 x	1 11 gates

inv04 tsmc035_typ 2 x 1 2 gates

latch tsmc035_typ 58 x 2 144 gates

latchr tsmc035_typ 18 x 3 49 gates

nand02 tsmc035_typ 19 x 1 19 gates

nand03 tsmc035_typ 1 x 1 1 gates

nand04 tsmc035_typ 5 x 1 7 gates

nor02_2x tsmc035_typ 4 x 1 4 gates

nor02ii tsmc035_typ 71 x 1 88 gates

nor03_2x tsmc035_typ 5 x 1 6 gates

nor04 tsmc035_typ 9 x 1 13 gates

oai21 tsmc035_typ 4 x 1 5 gates

or02 tsmc035_typ 2 x 1 2 gates

or04 tsmc035_typ 1 x 2 2 gates

xnor2 tsmc035_typ 32 x 2 61 gates

xor2 tsmc035_typ 28 x 2 59 gates

Number of ports : 3

Number of nets : 361

Number of instances : 353

Number of references to this view : 0

Total accumulated area :

Number of gates : 580

Number of accumulated instances : 353

Clock Frequency Report

Clock : Frequency

---

clk : 29.7 MHz

FDR Decoder

****

Cell: fdr_1 View: xyz Library: chintan_and1_lib

****

Cell	Library	References	Total Area
PadInC	tsmc035_typ	1 x	
PadOut	tsmc035_typ	2 x	
and02	tsmc035_typ	7 x 1	9 gates
and03	tsmc035_typ	1 x 2	2 gates
and04	tsmc035_typ	8 x 2	14 gates
ao21	tsmc035_typ	3 x 2	5 gates
ao22	tsmc035_typ	7 x 2	14 gates
ao32	tsmc035_typ	23 x 2	46 gates
aoi21	tsmc035_typ	10 x 1	12 gates
aoi22	tsmc035_typ	17 x 1	25 gates
aoi221	tsmc035_typ	1 x 2	2 gates
aoi222	tsmc035_typ	1 x 2	2 gates
aoi32	tsmc035_typ	4 x 2	7 gates
aoi321	tsmc035_typ	2 x 2	4 gates
aoi322	tsmc035_typ	1 x 2	2 gates
buf02	tsmc035_typ	41 x 1	42 gates

buf04 tsmc035_typ 6 x 1 7 gates

buf16 tsmc035_typ 12 x 3 35 gates

dff tsmc035_typ 8 x 5 38 gates

fake_gnd tsmc035_typ 1 x

inv01 tsmc035_typ 108 x 1 82 gates

inv02 tsmc035_typ 100 x 1 76 gates

latch tsmc035_typ 17 x 2 42 gates

latchr tsmc035_typ 43 x 3 116 gates

mux21 tsmc035_typ 32 x 2 59 gates

mux21_ni tsmc035_typ 22 x 2 40 gates

nand02 tsmc035_typ 71 x 1 71 gates

nand02_2x tsmc035_typ 7 x 1 7 gates

nand03 tsmc035_typ 1 x 1 1 gates

nand03_2x tsmc035_typ 2 x 1 2 gates

nand04 tsmc035_typ 4 x 1 6 gates

nand04_2x tsmc035_typ 1 x 2 2 gates

nor02_2x tsmc035_typ 29 x 1 29 gates

nor02ii tsmc035_typ 73 x 1 91 gates

nor03_2x tsmc035_typ 4 x 1 5 gates

nor04 tsmc035_typ 2 x 1 3 gates

oai21 tsmc035_typ 13 x 1 16 gates

oai22 tsmc035_typ 22 x 1 33 gates

oai221 tsmc035_typ 1 x 2 2 gates

oai222 tsmc035_typ 1 x 2 2 gates

oai32 tsmc035_typ 1 x 2 2 gates

oai321 tsmc035_typ 2 x 2 4 gates

oai322 tsmc035_typ 1 x 3 3 gates

or02 tsmc035_typ 6 x 1 7 gates

or03 tsmc035_typ 3 x 2 5 gates

or04 tsmc035_typ 1 x 2 2 gates

xnor2 tsmc035_typ 63 x 2 120 gates

xor2 tsmc035_typ 4 x 2 8 gates

Number of ports : 3

Number of nets : 797

Number of instances : 790

Number of references to this view : 0

Total accumulated area :

Number of gates : 1104

Number of accumulated instances : 790

Clock Frequency Report

Clock : Frequency

---

clk : 24.6 MHz

EFDR Decoder

****

Cell: efdr_1 View: xyz Library: chintan_and1_lib

****

Cell	Library	References	Total Area
PadInC	tsmc035_typ	1 x	
PadOut	tsmc035_typ	2 x	
and02	tsmc035_typ	5 x 1	6 gates
and04	tsmc035_typ	8 x 2	14 gates
ao32	tsmc035_typ	29 x 2	57 gates
aoi21	tsmc035_typ	11 x 1	14 gates
aoi22	tsmc035_typ	17 x 1	25 gates
aoi222	tsmc035_typ	1 x 2	2 gates
buf02	tsmc035_typ	48 x 1	49 gates
buf04	tsmc035_typ	4 x 1	5 gates
buf16	tsmc035_typ	6 x 3	18 gates
dff	tsmc035_typ	9 x 5	43 gates
fake_gnd	tsmc035_typ	1 x	
inv01	tsmc035_typ	89 x 1	68 gates
inv02	tsmc035_typ	70 x 1	53 gates
inv04	tsmc035_typ	1 x 1	1 gates
latch	tsmc035_typ	19 x 2	47 gates
latchr	tsmc035_typ	44 x 3	119 gates
mux21	tsmc035_typ	32 x 2	59 gates
mux21_ni	tsmc035_typ	26 x 2	47 gates
nand02	tsmc035_typ	77 x 1	77 gates
nand02_2x	tsmc035_typ	3 x 1	3 gates

nand03 tsmc035_typ 2 x 1 2 gates

nand04 tsmc035_typ 3 x 1 4 gates

nor02_2x tsmc035_typ 26 x 1 26 gates

nor02ii tsmc035_typ 82 x 1 102 gates

nor03_2x tsmc035_typ 3 x 1 4 gates

nor04 tsmc035_typ 4 x 1 6 gates

oai21 tsmc035_typ 11 x 1 14 gates

oai22 tsmc035_typ 20 x 1 30 gates

oai222 tsmc035_typ 3 x 2 7 gates

oai32 tsmc035_typ 1 x 2 2 gates

oai321 tsmc035_typ 1 x 2 2 gates

oai322 tsmc035_typ 2 x 3 5 gates

oai332 tsmc035_typ 1 x 3 3 gates

or02 tsmc035_typ 8 x 1 10 gates

or03 tsmc035_typ 2 x 2 3 gates

or04 tsmc035_typ 2 x 2 4 gates

xnor2 tsmc035_typ 67 x 2 128 gates

xor2 tsmc035_typ 9 x 2 19 gates

Number of ports : 3

Number of nets : 757

Number of instances : 750

Number of references to this view : 0

Total accumulated area :

Number of gates : 1078

Number of accumulated instances : 750

Clock Frequency Report

Clock : Frequency

---

state(0) : N/A

clk : 13.9 MHz

ASFDR Decoder

****

Cell: asfdr_1 View: xyz Library: chintan_and1_lib

****

Cell	Library	References	Total Area
PadInC	tsmc035_typ	1 x	
PadOut	tsmc035_typ	2 x	
and02	tsmc035_typ	4 x 1	5 gates
and03	tsmc035_typ	14 x 2	21 gates
and04	tsmc035_typ	9 x 2	16 gates
ao21	tsmc035_typ	1 x 2	2 gates
ao22	tsmc035_typ	1 x 2	2 gates
ao221	tsmc035_typ	1 x 2	2 gates
ao32	tsmc035_typ	16 x 2	32 gates
aoi21	tsmc035_typ	34 x 1	42 gates
aoi22	tsmc035_typ	2 x 1	3 gates
aoi221	tsmc035_typ	4 x 2	8 gates
aoi32	tsmc035_typ	17 x 2	29 gates
aoi33	tsmc035_typ	1 x 2	2 gates

buf02 tsmc035_typ 50 x 1 51 gates

buf04 tsmc035_typ 2 x 1 2 gates

buf16 tsmc035_typ 2 x 3 6 gates

dff tsmc035_typ 8 x 5 38 gates

fake_gnd tsmc035_typ 1 x

inv01 tsmc035_typ 111 x 1 84 gates

inv02 tsmc035_typ 67 x 1 51 gates

latch tsmc035_typ 16 x 2 40 gates

latchr tsmc035_typ 46 x 3 124 gates

mux21 tsmc035_typ 23 x 2 42 gates

mux21_ni tsmc035_typ 72 x 2 130 gates

nand02 tsmc035_typ 75 x 1 75 gates

nand02_2x tsmc035_typ 2 x 1 2 gates

nand03 tsmc035_typ 1 x 1 1 gates

nand04 tsmc035_typ 3 x 1 4 gates

nor02_2x tsmc035_typ 47 x 1 47 gates

nor02ii tsmc035_typ 33 x 1 41 gates

nor03_2x tsmc035_typ 2 x 1 2 gates

nor04 tsmc035_typ 4 x 1 6 gates

oai21 tsmc035_typ 40 x 1 50 gates

oai22 tsmc035_typ 4 x 1 6 gates

oai32 tsmc035_typ 1 x 2 2 gates

oai322 tsmc035_typ 1 x 3 3 gates

oai33 tsmc035_typ 1 x 2 2 gates

or02 tsmc035_typ 4 x 1 5 gates

or03 tsmc035_typ 16 x 2 25 gates

or04 tsmc035_typ 1 x 2 2 gates

xnor2 tsmc035_typ 27 x 2 52 gates

xor2 tsmc035_typ 25 x 2 53 gates

Number of ports : 3

Number of nets : 799

Number of instances : 792

Number of references to this view : 0

Total accumulated area :

Number of gates : 1111

Number of accumulated instances : 792

Clock Frequency Report

Clock : Frequency

---

clk : 11.3 MHz

MFDR Decoder

****

Cell: mfdr View: xyz Library: mfdr_lib

****

Cell Library References Total Area

PadInC tsmc035_typ 1 x

PadOut tsmc035_typ 2 x

and02 tsmc035_typ 62 x 1 78 gates

and04 tsmc035_typ 9 x 2 16 gates

ao21 tsmc035_typ 3 x 2 5 gates

ao32 tsmc035_typ 2 x 2 4 gates

aoi21 tsmc035_typ 20 x 1 25 gates

aoi22 tsmc035_typ 2 x 1 3 gates

aoi221 tsmc035_typ 1 x 2 2 gates

aoi32 tsmc035_typ 17 x 2 29 gates

aoi321 tsmc035_typ 1 x 2 2 gates

buf02 tsmc035_typ 105 x 1 107 gates

buf04 tsmc035_typ 2 x 1 2 gates

dff tsmc035_typ 11 x 5 53 gates

fake_gnd tsmc035_typ 1 x

inv01 tsmc035_typ 163 x 1 124 gates

inv02 tsmc035_typ 171 x 1 130 gates

inv16 tsmc035_typ 1 x 2 2 gates

latch tsmc035_typ 65 x 2 161 gates

latchr tsmc035_typ 91 x 3 246 gates

mux21 tsmc035_typ 21 x 2 38 gates

mux21_ni tsmc035_typ 157 x 2 284 gates

nand02 tsmc035_typ 59 x 1 59 gates

nand02_2x tsmc035_typ 1 x 1 1 gates

nand03 tsmc035_typ 4 x 1 5 gates

nand04 tsmc035_typ 10 x 1 15 gates

nor02_2x tsmc035_typ 34 x 1 34 gates

nor02ii tsmc035_typ 91 x 1 113 gates

nor03_2x tsmc035_typ 5 x 1 6 gates

nor04 tsmc035_typ 17 x 1 25 gates

oai21 tsmc035_typ 31 x 1 38 gates

oai22 tsmc035_typ 63 x 1 93 gates

oai221 tsmc035_typ 3 x 2 6 gates

oai222 tsmc035_typ 1 x 2 2 gates

oai32 tsmc035_typ 41 x 2 71 gates

oai321 tsmc035_typ 2 x 2 4 gates

oai33 tsmc035_typ 1 x 2 2 gates

or02 tsmc035_typ 20 x 1 25 gates

or03 tsmc035_typ 2 x 2 3 gates

or04 tsmc035_typ 1 x 2 2 gates

xnor2 tsmc035_typ 141 x 2 269 gates

xor2 tsmc035_typ 53 x 2 112 gates

Number of ports : 3

Number of nets : 1493

Number of instances : 1488

Number of references to this view : 0

Total accumulated area :

Number of gates : 2199

Number of accumulated instances : 1488

Clock Frequency Report

Clock : Frequency

---

clk : 11.0 MHz

Decoders for Statistical Codes

Selective Huffman Decoder

```
LeonardoSpectrum Level 3 - 2008a.5 (Release Production Release, compiled Jun 30 2008 at 14:33:46)
Copyright 1990-2008 Mentor Graphics.  All rights reserved.
Portions copyright 1991-2008 Compuware Corporation

--
-- Welcome to LeonardoSpectrum Level 3
-- Run By temp@NIT-106
-- Run Started On Fri Mar 12 14:41:39 India Standard Time 2010
--
Info, Working Directory is now 'H:/ATE with decoder/selective decoder/vhdl'
Info, History file moved to new working directory
Info, Log file moved to new working directory
Reading library file `D:\LeoSpec_LS2008a_5\\lib\tsmc018_typ.syn`...
Library version = v3.1 Release : Patch (a) : (Aug 26, 2005)
Delays assume: Process=typical Temp=  0.0 C  Voltage=1.80 V
Info: setting encoding to auto
-- Reading file D:\LeoSpec_LS2008a_5\\data\standard.vhd for unit standard
-- Loading package standard into library std
-- Reading vhdl file H:/ATE with decoder/selective decoder/vhdl/serializer3.vhd into library work
-- Reading file D:\LeoSpec_LS2008a_5\\data\std_1164.vhd for unit std_logic_1164
-- Loading package std_logic_1164 into library ieee
-- Loading entity serializer3 into library work
-- Loading architecture xyz of serializer3 into library work
-- Reading vhdl file H:/ATE with decoder/selective decoder/vhdl/ate3.vhd into library work
-- Searching for SYNOPSYS package STD_LOGIC_ARITH..
-- Reading file D:\LeoSpec_LS2008a_5\\data\syn_arit.vhd for unit STD_LOGIC_ARITH
-- Loading package std_logic_arith into library ieee
-- Searching for SYNOPSYS package STD_LOGIC_UNSIGNED..
-- Reading file D:\LeoSpec_LS2008a_5\\data\syn_unsi.vhd for unit STD_LOGIC_UNSIGNED
-- Loading package STD_LOGIC_UNSIGNED into library ieee
-- Loading entity ate3 into library work
-- Loading architecture Behavioral of ate3 into library work
-- Reading vhdl file H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd into library work
-- Loading entity decoder3 into library work
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 21: Info, Enumerated type fsm_state
with 7 elements encoded as onehot.
Encodings for fsm_state values
       value    fsm_state[6-0]
==============================
          a0  ------1
           a  -----1-
           b  ----1--
           c  ---1---
           d  --1----
           e  -1-----
           f  1------
```

-- Loading architecture Behavioral of decoder3 into library work
-- Compiling root entity ate3(Behavioral)
"H:/ATE with decoder/selective decoder/vhdl/ate3.vhd",line 34: Warning, initial value for abc is ignored for synthesis.
-- Compiling entity decoder3(Behavioral)
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 22: Warning, initial value for p_state is ignored for synthesis.
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 22: Warning, initial value for n_state is ignored for synthesis.
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 45: Info, conditions are mutually exclusive; resolve without priority.
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 45: Info, else part is never selected for synthesis.
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 50: Warning, s0 should be declared on the sensitivity list of the process.
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 50: Info, conditions are mutually exclusive; resolve without priority.
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 50: Info, else part is never selected for synthesis.
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 55: Warning, s1 should be declared on the sensitivity list of the process.
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 56: Warning, s2 should be declared on the sensitivity list of the process.
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 55: Info, conditions are mutually exclusive; resolve without priority.
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 55: Info, else part is never selected for synthesis.
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 11: Warning, serializer is not always assigned. Storage may be needed..
"H:/ATE with decoder/selective decoder/vhdl/decoder3.vhd",line 23: Warning, s is not always assigned. Storage may be needed..
-- Compiling entity serializer3(xyz)
"H:/ATE with decoder/selective decoder/vhdl/serializer3.vhd",line 12: Warning, initial value for Q is ignored for synthesis.
"H:/ATE with decoder/selective decoder/vhdl/serializer3.vhd",line 12: Warning, initial value for Q1 is ignored for synthesis.
"H:/ATE with decoder/selective decoder/vhdl/ate3.vhd",line 33: Warning, serialreg is not always assigned. Storage may be needed..
-- Boundary optimization.
-- Boundary optimization.
-- Start pre-optimization for design .work.decoder3.Behavioral_unfold_1137
-- Start pre-optimization for design .work.serializer3.xyz_unfold_704
-- Start pre-optimization for design .work.ate3.Behavioral
"H:/ATE with decoder/selective decoder/vhdl/ate3.vhd", line 89:Info, Inferred counter instance 'abc' of type 'counter_up_sclear_clock_32'
-- Start pre-optimization for design .work.decoder3.Behavioral_unfold_1137
-- Start pre-optimization for design .work.serializer3.xyz_unfold_704
-- Start pre-optimization for design .work.ate3.Behavioral
Info, Instances dissolved by autodissolve in View .work.ate3.Behavioral
"H:/ATE with decoder/selective decoder/vhdl/ate3.vhd", line 39: X1 (decoder3)
"H:/ATE with decoder/selective decoder/vhdl/ate3.vhd", line 40: X2 (serializer3)
-- Optimizing netlist .work.ate3.Behavioral
-- Matching combinational logic..
-- Matching non-combinational logic..
-- Covering..

```
-- CPU Time used :   00:00 Mapping
Re-checking DRC after adjustments
-- Final Design Rule Check..
Re-checking DRC after adjustments
-- Start timing optimization for design .work.ate3.Behavioral
Starting Timing Characterization...
Starting Timing Analysis...
Timing analysis done, time = 0 CPU secs.
Timing characterization done, time = 0 CPU secs.

*******************************************************

Cell: ate3     View: Behavioral     Library: work

*******************************************************


 Cell       Library  References     Total Area

and02       tsmc018_typ    18 x      1     23 gates
and03       tsmc018_typ    13 x      2     20 gates
and04       tsmc018_typ     1 x      2      2 gates
aoi21       tsmc018_typ     4 x      1      5 gates
buf02       tsmc018_typ     3 x      1      3 gates
dff         tsmc018_typ    43 x      4    190 gates
dffsr_ni    tsmc018_typ     1 x      6      6 gates
fake_gnd    tsmc018_typ     1 x      1      1 fake_gnd
inv01       tsmc018_typ    26 x      1     20 gates
latch       tsmc018_typ     4 x      2     10 gates
latchr      tsmc018_typ     1 x      3      3 gates
mux21_ni    tsmc018_typ     1 x      2      2 gates
nand02      tsmc018_typ    18 x      1     18 gates
nand03      tsmc018_typ     3 x      1      4 gates
nand04      tsmc018_typ     2 x      1      3 gates
nor02       tsmc018_typ    15 x      1     15 gates
nor02ii     tsmc018_typ    21 x      1     26 gates
nor03       tsmc018_typ    17 x      1     21 gates
nor04       tsmc018_typ     5 x      1      7 gates
oai21       tsmc018_typ     1 x      1      1 gates
oai32       tsmc018_typ     1 x      2      2 gates
oai33       tsmc018_typ     1 x      2      2 gates
or02        tsmc018_typ     1 x      1      1 gates
or04        tsmc018_typ     2 x      2      3 gates
xnor2       tsmc018_typ     1 x      2      2 gates

 Number of ports :                  5
 Number of nets :                 225
 Number of instances :            204
 Number of references to this view :     0

Total accumulated area :
 Number of fake_gnd :               1
 Number of gates :                388
 Number of accumulated instances :     204
Info: setting report_delay_slack_threshold to 0.000000
```

Clock Frequency Report

Clock	: Frequency
ate_clk	: 1506.2 MHz
sys_clk	: 248.0 MHz

Critical Path Report

Critical path #1, (unconstrained path)

NAME	GATE	ARRIVAL	LOAD
clock information not specified			
delay thru clock network		0.00 (ideal)	
reg_abc(1)/Q	dff	0.00 0.43 up	0.05
ix452/Y	nand04	0.11 0.54 dn	0.02
ix113/Y	nor02ii	0.15 0.69 up	0.02
ix484/Y	nand02	0.09 0.78 dn	0.02
ix145/Y	nor02ii	0.15 0.92 up	0.02
ix502/Y	nand02	0.09 1.01 dn	0.02
ix177/Y	nor02ii	0.15 1.16 up	0.02
ix530/Y	nand02	0.09 1.24 dn	0.02
ix209/Y	nor02ii	0.15 1.39 up	0.02
ix539/Y	nand02	0.09 1.48 dn	0.02
ix241/Y	nor02ii	0.15 1.62 up	0.02
ix560/Y	nand02	0.09 1.71 dn	0.02
ix273/Y	nor02ii	0.15 1.86 up	0.02
ix580/Y	nand02	0.09 1.94 dn	0.02
ix305/Y	nor02ii	0.15 2.09 up	0.02
ix594/Y	nand02	0.09 2.18 dn	0.02
ix337/Y	nor02ii	0.15 2.33 up	0.02
ix622/Y	nand02	0.09 2.41 dn	0.02
ix369/Y	nor02ii	0.15 2.56 up	0.02
ix638/Y	nand02	0.09 2.64 dn	0.02
ix401/Y	nor02ii	0.15 2.79 up	0.02
ix660/Y	nand02	0.09 2.88 dn	0.02
ix433/Y	nor02ii	0.15 3.03 up	0.02
ix678/Y	nand02	0.09 3.11 dn	0.02
ix465/Y	nor02ii	0.15 3.26 up	0.02
ix700/Y	nand02	0.09 3.35 dn	0.02
ix497/Y	nor02ii	0.15 3.49 up	0.02
ix716/Y	nand02	0.09 3.58 dn	0.02
ix529/Y	nor02ii	0.18 3.76 up	0.03
ix724/Y	xnor2	0.08 3.84 dn	0.01
ix533/Y	nor02ii	0.12 3.95 up	0.01
reg_abc(31)/D	dff	0.00 3.95 up	0.00
data arrival time		3.95	
data required time		not specified	

data required time not specified
data arrival time 3.95

---

unconstrained path

Info: setting novendor_constraint_file to FALSE
AutoWrite args are : decoder3_0.vhd
-- Writing file decoder3_0.vhd

Optimal Selective Huffman Decoder

LeonardoSpectrum Level 3 - 2008a.5 (Release Production Release, compiled Jun 30 2008 at 14:33:46)

--

-- Welcome to LeonardoSpectrum Level 3

-- Run By temp@NIT-106

-- Run Started On Fri Mar 12 14:38:52 India Standard Time 2010

--

Info, Working Directory is now 'H:/ATE with decoder/optimal selective decoder/vhdl'

Info, History file moved to new working directory

Info, Log file moved to new working directory

Reading library file `D:\LeoSpec_LS2008a_5\\lib\tsmc018_typ.syn`...

Library version = v3.1 Release : Patch (a) : (Aug 26, 2005)

Delays assume: Process=typical Temp= 0.0 C Voltage=1.80 V

Info: setting encoding to auto

-- Reading file D:\LeoSpec_LS2008a_5\\data\standard.vhd for unit standard

-- Loading package standard into library std

-- Reading vhdl file H:/ATE with decoder/optimal selective decoder/vhdl/serializer2.vhd into library work

-- Reading file D:\LeoSpec_LS2008a_5\\data\std_1164.vhd for unit std_logic_1164

-- Loading package std_logic_1164 into library ieee

-- Loading entity serializer1 into library work

-- Loading architecture xyz of serializer1 into library work

-- Reading vhdl file H:/ATE with decoder/optimal selective decoder/vhdl/ate2.vhd into library work

-- Searching for SYNOPSYS package STD_LOGIC_ARITH..

-- Reading file D:\LeoSpec_LS2008a_5\\data\syn_arit.vhd for unit STD_LOGIC_ARITH

-- Loading package std_logic_arith into library ieee

-- Searching for SYNOPSYS package STD_LOGIC_UNSIGNED..

-- Reading file D:\LeoSpec_LS2008a_5\\data\syn_unsi.vhd for unit STD_LOGIC_UNSIGNED

-- Loading package STD_LOGIC_UNSIGNED into library ieee

-- Loading entity ate1 into library work

-- Loading architecture Behavioral of ate1 into library work

-- Reading vhdl file H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd into library work

-- Loading entity decoder1 into library work

"H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd",line 21: Info, Enumerated type fsm_state with 7 elements encoded as onehot.

Encodings for fsm_state values

value fsm_state[6-0]

==========

a ------1

b -----1-

c ----1--

d ---1---

e --1----

f -1-----

g 1------

-- Loading architecture Behavioral of decoder1 into library work

-- Compiling root entity ate1(Behavioral)

"H:/ATE with decoder/optimal selective decoder/vhdl/ate2.vhd",line 34: Warning, initial value for abc is ignored for synthesis.

-- Compiling entity decoder1(Behavioral)

"H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd",line 45: Warning, s0 should be declared on the sensitivity list of the process.

"H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd",line 45: Info, conditions are mutually exclusive; resolve without priority.

"H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd",line 45: Info, else part is never selected for synthesis.

"H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd",line 50: Warning, s1 should be declared on the sensitivity list of the process.

"H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd",line 50: Info, conditions are mutually exclusive; resolve without priority.

"H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd",line 50: Info, else part is never selected for synthesis.

"H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd",line 55: Warning, s2 should be declared on the sensitivity list of the process.

"H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd",line 55: Info, conditions are mutually exclusive; resolve without priority.

"H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd",line 55: Info, else part is never selected for synthesis.

"H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd",line 23: Warning, s is not always assigned. Storage may be needed..

"H:/ATE with decoder/optimal selective decoder/vhdl/decoder2.vhd",line 11: Warning, serializer is not always assigned. Storage may be needed..

-- Compiling entity serializer1(xyz)

"H:/ATE with decoder/optimal selective decoder/vhdl/serializer2.vhd",line 12: Warning, initial value for Q is ignored for synthesis.

"H:/ATE with decoder/optimal selective decoder/vhdl/serializer2.vhd",line 12: Warning, initial value for Q1 is ignored for synthesis.

"H:/ATE with decoder/optimal selective decoder/vhdl/ate2.vhd",line 33: Warning, serialreg is not always assigned. Storage may be needed..

-- Boundary optimization.

-- Boundary optimization.

-- Start pre-optimization for design .work.decoder1.Behavioral_unfold_1135

-- Start pre-optimization for design .work.serializer1.xyz_unfold_702

-- Start pre-optimization for design .work.ate1.Behavioral

"H:/ATE with decoder/optimal selective decoder/vhdl/ate2.vhd", line 89:Info, Inferred counter instance 'abc' of type 'counter_up_sclear_clock_32'

-- Start pre-optimization for design .work.decoder1.Behavioral_unfold_1135

-- Start pre-optimization for design .work.serializer1.xyz_unfold_702

-- Start pre-optimization for design .work.ate1.Behavioral

Info, Instances dissolved by autodissolve in View .work.ate1.Behavioral

"H:/ATE with decoder/optimal selective decoder/vhdl/ate2.vhd", line 39: X1 (decoder1)

"H:/ATE with decoder/optimal selective decoder/vhdl/ate2.vhd", line 40: X2 (serializer1)

-- Optimizing netlist .work.ate1.Behavioral

-- Matching combinational logic..

-- Matching non-combinational logic..

-- Covering..

-- CPU Time used : 00:00 Mapping

Re-checking DRC after adjustments

-- Final Design Rule Check..

Re-checking DRC after adjustments

-- Start timing optimization for design .work.ate1.Behavioral

Starting Timing Characterization...

Starting Timing Analysis...

Timing analysis done, time = 0 CPU secs.

Timing characterization done, time = 0 CPU secs.

****

Cell: ate1 View: Behavioral Library: work

****

Cell	Library	References	Total Area
and02	tsmc018_typ	17 x 1	21 gates
and03	tsmc018_typ	13 x 2	20 gates
and04	tsmc018_typ	1 x 2	2 gates
ao221	tsmc018_typ	1 x 2	2 gates
aoi21	tsmc018_typ	4 x 1	5 gates
buf02	tsmc018_typ	3 x 1	3 gates
dff	tsmc018_typ	43 x 4	190 gates
dffsr_ni	tsmc018_typ	1 x 6	6 gates
fake_gnd	tsmc018_typ	1 x 1	1 fake_gnd
inv01	tsmc018_typ	26 x 1	20 gates
latch	tsmc018_typ	4 x 2	10 gates
latchr	tsmc018_typ	1 x 3	3 gates
mux21_ni	tsmc018_typ	1 x 2	2 gates
nand02	tsmc018_typ	19 x 1	19 gates
nand03	tsmc018_typ	3 x 1	4 gates
nand04	tsmc018_typ	2 x 1	3 gates
nor02	tsmc018_typ	15 x 1	15 gates
nor02ii	tsmc018_typ	20 x 1	25 gates
nor03	tsmc018_typ	17 x 1	21 gates
nor04	tsmc018_typ	5 x 1	7 gates
oai21	tsmc018_typ	2 x 1	2 gates

```
oai32       tsmc018_typ     1 x      2      2 gates
oai33       tsmc018_typ     1 x      2      2 gates
or04        tsmc018_typ     2 x      2      3 gates
xnor2       tsmc018_typ     1 x      2      2 gates
 Number of ports :                  5
 Number of nets :                 226
 Number of instances :            204
 Number of references to this view :     0
Total accumulated area :
 Number of fake_gnd :               1
 Number of gates :                389
 Number of accumulated instances :     204
Info: setting report_delay_slack_threshold to 0.000000
                    Clock Frequency Report
          Clock              : Frequency
      -----------------------------------
          ate_clk            : 1242.4 MHz
          sys_clk            : 248.0 MHz
                    Critical Path Report
Critical path #1, (unconstrained path)
NAME                         GATE           ARRIVAL           LOAD
-------------------------------------------------------------------------------
clock information not specified
delay thru clock network                        0.00 (ideal)
reg_abc(1)/Q                  dff        0.00  0.43 up           0.05
ix452/Y                       nand04     0.11  0.54 dn           0.02
```

ix123/Y	nor02ii	0.15	0.69	up	0.02
ix484/Y	nand02	0.09	0.78	dn	0.02
ix155/Y	nor02ii	0.15	0.92	up	0.02
ix502/Y	nand02	0.09	1.01	dn	0.02
ix187/Y	nor02ii	0.15	1.16	up	0.02
ix532/Y	nand02	0.09	1.24	dn	0.02
ix219/Y	nor02ii	0.15	1.39	up	0.02
ix540/Y	nand02	0.09	1.48	dn	0.02
ix251/Y	nor02ii	0.15	1.62	up	0.02
ix560/Y	nand02	0.09	1.71	dn	0.02
ix283/Y	nor02ii	0.15	1.86	up	0.02
ix580/Y	nand02	0.09	1.94	dn	0.02
ix315/Y	nor02ii	0.15	2.09	up	0.02
ix594/Y	nand02	0.09	2.18	dn	0.02
ix347/Y	nor02ii	0.15	2.33	up	0.02
ix622/Y	nand02	0.09	2.41	dn	0.02
ix379/Y	nor02ii	0.15	2.56	up	0.02
ix640/Y	nand02	0.09	2.64	dn	0.02
ix411/Y	nor02ii	0.15	2.79	up	0.02
ix658/Y	nand02	0.09	2.88	dn	0.02
ix443/Y	nor02ii	0.15	3.03	up	0.02
ix678/Y	nand02	0.09	3.11	dn	0.02
ix475/Y	nor02ii	0.15	3.26	up	0.02
ix698/Y	nand02	0.09	3.35	dn	0.02
ix507/Y	nor02ii	0.15	3.49	up	0.02
ix714/Y	nand02	0.09	3.58	dn	0.02

ix539/Y	nor02ii	0.18	3.76	up	0.03
ix722/Y	xnor2	0.08	3.84	dn	0.01
ix543/Y	nor02ii	0.12	3.95	up	0.01
reg_abc(31)/D	dff	0.00	3.95	up	0.00

data arrival time 3.95

data required time not specified

---

data required time not specified

data arrival time 3.95

---

unconstrained path

Info: setting novendor_constraint_file to FALSE

AutoWrite args are : decoder2_0.vhd

-- Writing file decoder2_0.vhd

Modified Selctive Huffman Decoder

LeonardoSpectrum Level 3 - 2008a.5 (Release Production Release, compiled Jun 30 2008 at 14:33:46)

-- Welcome to LeonardoSpectrum Level 3

-- Run By temp@NIT-106

-- Run Started On Fri Mar 12 14:37:30 India Standard Time 2010

Info, Working Directory is now 'H:/ATE with decoder/huffman decoder/vhdl'

Reading library file `D:\LeoSpec_LS2008a_5\\lib\tsmc018_typ.syn`...

Library version = v3.1 Release : Patch (a) : (Aug 26, 2005)

Delays assume: Process=typical Temp= 0.0 C Voltage=1.80 V

Info: setting encoding to auto

-- Reading file D:\LeoSpec_LS2008a_5\\data\standard.vhd for unit standard

-- Loading package standard into library std

-- Reading vhdl file H:/ATE with decoder/huffman decoder/vhdl/serializer1.vhd into library work

-- Reading file D:\LeoSpec_LS2008a_5\\data\std_1164.vhd for unit std_logic_1164

-- Loading package std_logic_1164 into library ieee

-- Loading entity serializer1 into library work

-- Loading architecture xyz of serializer1 into library work

-- Reading vhdl file H:/ATE with decoder/huffman decoder/vhdl/ate1.vhd into library work

-- Searching for SYNOPSYS package STD_LOGIC_ARITH..

-- Reading file D:\LeoSpec_LS2008a_5\\data\syn_arit.vhd for unit STD_LOGIC_ARITH

-- Loading package std_logic_arith into library ieee

-- Searching for SYNOPSYS package STD_LOGIC_UNSIGNED..

-- Reading file D:\LeoSpec_LS2008a_5\\data\syn_unsi.vhd for unit STD_LOGIC_UNSIGNED

-- Loading package STD_LOGIC_UNSIGNED into library ieee

-- Loading entity ate1 into library work

-- Loading architecture Behavioral of ate1 into library work

-- Reading vhdl file H:/ATE with decoder/huffman decoder/vhdl/decoder1.vhd into library work

-- Loading entity decoder1 into library work

"H:/ATE with decoder/huffman decoder/vhdl/decoder1.vhd",line 26: Info, Enumerated type fsm_state with 2 elements encoded as binary.

Encodings for fsm_state values

  value   fsm_state[0]

========================

        a 0

        b 1

-- Loading architecture Behavioral of decoder1 into library work

-- Compiling root entity ate1(Behavioral)

"H:/ATE with decoder/huffman decoder/vhdl/ate1.vhd",line 36: Warning, initial value for abc is ignored for synthesis.

-- Compiling entity decoder1(Behavioral)

"H:/ATE with decoder/huffman decoder/vhdl/decoder1.vhd",line 51: Warning, s0 should be declared on the sensitivity list of the process.

"H:/ATE with decoder/huffman decoder/vhdl/decoder1.vhd",line 55: Warning, s1 should be declared on the sensitivity list of the process.

"H:/ATE with decoder/huffman decoder/vhdl/decoder1.vhd",line 56: Warning, s2 should be declared on the sensitivity list of the process.

"H:/ATE with decoder/huffman decoder/vhdl/decoder1.vhd",line 55: Info, conditions are mutually exclusive; resolve without priority.

"H:/ATE with decoder/huffman decoder/vhdl/decoder1.vhd",line 55: Info, else part is never selected for synthesis.

"H:/ATE with decoder/huffman decoder/vhdl/decoder1.vhd",line 28: Warning, s is not always assigned. Storage may be needed..

"H:/ATE with decoder/huffman decoder/vhdl/decoder1.vhd",line 27: Warning, n_state is not always assigned. Storage may be needed..

"H:/ATE with decoder/huffman decoder/vhdl/decoder1.vhd",line 16: Warning, serializer is not always assigned. Storage may be needed..

-- Compiling entity serializer1(xyz)

"H:/ATE with decoder/huffman decoder/vhdl/serializer1.vhd",line 12: Warning, initial value for Q is ignored for synthesis.

"H:/ATE with decoder/huffman decoder/vhdl/serializer1.vhd",line 12: Warning, initial value for Q1 is ignored for synthesis.

"H:/ATE with decoder/huffman decoder/vhdl/ate1.vhd",line 34: Warning, serialreg is not always assigned. Storage may be needed..

-- Boundary optimization.

-- Boundary optimization.

-- Start pre-optimization for design .work.decoder1.Behavioral_unfold_1135

-- Start pre-optimization for design .work.serializer1.xyz_unfold_702

-- Start pre-optimization for design .work.ate1.Behavioral

"H:/ATE with decoder/huffman decoder/vhdl/ate1.vhd", line 93:Info, Inferred counter instance 'abc' of type 'counter_up_sclear_clock_32'

-- Start pre-optimization for design .work.decoder1.Behavioral_unfold_1135

-- Start pre-optimization for design .work.serializer1.xyz_unfold_702

-- Start pre-optimization for design .work.ate1.Behavioral

Info, Instances dissolved by autodissolve in View .work.ate1.Behavioral

"H:/ATE with decoder/huffman decoder/vhdl/ate1.vhd", line 40: X1 (decoder1)

"H:/ATE with decoder/huffman decoder/vhdl/ate1.vhd", line 41: X2 (serializer1)

-- Optimizing netlist .work.ate1.Behavioral

-- Matching combinational logic..

-- Matching non-combinational logic..

-- Covering..

-- CPU Time used : 00:00 Mapping

Re-checking DRC after adjustments

-- Final Design Rule Check..

Re-checking DRC after adjustments

-- Start timing optimization for design .work.ate1.Behavioral

Starting Timing Characterization...

Starting Timing Analysis...

Timing analysis done, time = 0 CPU secs.

Timing characterization done, time = 0 CPU secs.

****

Cell: ate1 View: Behavioral Library: work

****

Cell	Library	References	Total Area
and02	tsmc018_typ	19 x 1	24 gates
and03	tsmc018_typ	13 x 2	20 gates
ao32	tsmc018_typ	1 x 2	2 gates

aoi21 tsmc018_typ 3 x 1 4 gates

buf02 tsmc018_typ 2 x 1 2 gates

dff tsmc018_typ 36 x 4 159 gates

dffr tsmc018_typ 1 x 5 5 gates

dffsr_ni tsmc018_typ 1 x 6 6 gates

fake_gnd tsmc018_typ 1 x 1 1 fake_gnd

inv01 tsmc018_typ 28 x 1 21 gates

latch tsmc018_typ 4 x 2 10 gates

latchr tsmc018_typ 1 x 3 3 gates

latchsr_ni tsmc018_typ 1 x 3 3 gates

mux21_ni tsmc018_typ 2 x 2 4 gates

nand02 tsmc018_typ 17 x 1 17 gates

nand03 tsmc018_typ 1 x 1 1 gates

nand04 tsmc018_typ 4 x 1 6 gates

nor02 tsmc018_typ 15 x 1 15 gates

nor02ii tsmc018_typ 21 x 1 26 gates

nor03 tsmc018_typ 16 x 1 20 gates

nor04 tsmc018_typ 5 x 1 7 gates

oai21 tsmc018_typ 1 x 1 1 gates

or04 tsmc018_typ 2 x 2 3 gates

xnor2 tsmc018_typ 1 x 2 2 gates

Number of ports : 6

Number of nets : 217

Number of instances : 196

Number of references to this view : 0

Total accumulated area :

Number of fake_gnd : 1

Number of gates : 362

Number of accumulated instances : 196

Info: setting report_delay_slack_threshold to 0.000000

Clock Frequency Report

Clock : Frequency

---

serial_data : N/A

ate_clk : 1927.2 MHz

sys_clk : 247.5 MHz

en : N/A

Critical Path Report

Critical path #1, (unconstrained path)

NAME	GATE	ARRIVAL	LOAD
clock information not specified			
delay thru clock network		0.00 (ideal)	
reg_abc(1)/Q	dff	0.00 0.44 up	0.05
ix449/Y	nand04	0.11 0.55 dn	0.02
ix51/Y	nor02ii	0.15 0.70 up	0.02
ix480/Y	nand02	0.09 0.78 dn	0.02
ix83/Y	nor02ii	0.15 0.93 up	0.02
ix500/Y	nand02	0.09 1.02 dn	0.02
ix115/Y	nor02ii	0.15 1.17 up	0.02
ix534/Y	nand02	0.09 1.25 dn	0.02
ix147/Y	nor02ii	0.15 1.40 up	0.02
ix544/Y	nand02	0.09 1.49 dn	0.02
ix179/Y	nor02ii	0.15 1.63 up	0.02
ix563/Y	nand02	0.09 1.72 dn	0.02
ix211/Y	nor02ii	0.15 1.87 up	0.02

ix578/Y nand02 0.09 1.95 dn 0.02

ix243/Y nor02ii 0.15 2.10 up 0.02

ix592/Y nand02 0.09 2.19 dn 0.02

ix275/Y nor02ii 0.15 2.33 up 0.02

ix618/Y nand02 0.09 2.42 dn 0.02

ix307/Y nor02ii 0.15 2.57 up 0.02

ix634/Y nand02 0.09 2.65 dn 0.02

ix339/Y nor02ii 0.15 2.80 up 0.02

ix654/Y nand02 0.09 2.89 dn 0.02

ix371/Y nor02ii 0.15 3.03 up 0.02

ix671/Y nand02 0.09 3.12 dn 0.02

ix403/Y nor02ii 0.15 3.27 up 0.02

ix690/Y nand02 0.09 3.35 dn 0.02

ix435/Y nor02ii 0.15 3.50 up 0.02

ix706/Y nand02 0.09 3.59 dn 0.02

ix467/Y nor02ii 0.18 3.77 up 0.03

ix716/Y xnor2 0.08 3.84 dn 0.01

ix471/Y nor02ii 0.12 3.96 up 0.01

reg_abc(31)/D dff 0.00 3.96 up 0.00

data arrival time 3.96

data required time not specified

---

data required time not specified

data arrival time 3.96

unconstrained path

Info: setting novendor_constraint_file to FALSE

AutoWrite args are : decoder1_0.vhd