DEVELOPMENT OF ENHANCED TEST DATA COMPRESSION SCHEME FOR SYSTEM ON CHIP

A Dissertation Submitted In Partial Fulfillment of the Required for the Degree of

MASTER OF TECHNOLOGY

In

VLSI Design

Submitted By

SHRUTI

Roll no. 601361025

Under the guidance of

Ms. Harpreet Vohra

Assistant Professor, ECED

T.U. Patiala

Department of Electronics and Communication Engineering

THAPAR UNIVERSITY

(Established under the section 3 of UGC Act, 1956)

PATIALA 147004 (PUNJAB)

July, 2015

DECLARATION

I hereby declare that the work which is being presented in dissertation titled **"DEVELOPMENT OF ENHANCED TEST DATA COMPRESSION SCHEME FOR SYSTEM ON CHIP"** in partial fulfilment of the requirement for the award of degree of Master of Technology in VLSI Design submitted in Electronics and Communication Engineering Department of Thapar University, Patiala is an authentic record of my study carried out under the guidance of **Ms. Harpreet Vohra,** Assistant Professor, ECED during 2013-2015.

Date: 10/7/15

SHRUTI

Roll no. 601361025

It is certified that the above statement made by the student is correct to the best of my Knowledge and belief.

Ms. Harpreet Vohra

Assistant Professor, ECED,
Thapar University,
Patiala, 147004

Countersigned by:

(Dr. Sanjay Sharma)
Professor and Head
ECED, Thapar University,
Patiala, 147004

Dr. S.S. Bhatia
Dean of Academics Affairs
Thapar University,
Patiala, 147004

ACKNOWLEDGEMENT

To discover, analyze and to present something new is to venture on an untrodden path towards and unexplored destination is an arduous adventure unless one gets a true torch bearer to show the way. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by various people. Words are often too less to reveals one's deep regards. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this thesis. I acknowledge with gratitude and humility my indebtedness to **Ms. Harpreet Vohra, Assistant Professor,** Electronics and Communication Engineering Department, Thapar University, Patiala, under whose guidance I had the privilege to complete this thesis. I wish to express my deep gratitude towards her for providing individual guidance and support throughout the thesis work.

I convey my sincere thanks to **Head of the Department, Dr. Sanjay Sharma** as well as **PG Coordinator, Dr. Amit Kumar Kohli, Associate Professor,** Electronics and Communication Engineering Department, entire faculty and staff of Electronics and Communication Engineering Department for their encouragement and cooperation.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the Almighty who bestowed self-confidence, ability and strength in me to complete this work for not letting me down at the time of crisis and showing me the silver lining in the dark clouds. I do not find enough words with which I can express my feelings of thanks to my dear friends for their help, inspiration and moral support which went a long way in successful competition of the present study.

(SHRUTI)

ABSTRACT

System on chips have turned out to be one of the viable solutions for keeping in pace with the increasing demand of adding more features in a single product. As the whole product lie on the functionality of the SOCs, thus they are required to be tested to make them fault free. But as the complexity of the core is increasing the amount of test data is also increasing. However, the ATEs available at present have not developed to that extent. So, huge test data of complex chips demands more and more ATE memory for storage. This requirement of excess memory adds to cost of ATE as ATEs have to be supported by external hardware. One solution to this is use of BIST, but for this, the IP cores should be BIST ready. So another promising solution is to reduce the test data. If the data is compressed then the pressure on ATE for more memory will be reduced.

This work presents analysis and the study of various test data compression techniques. These techniques include run length based codes, statistical codes and dictionary based codes. In the category of statistical codes, codes like selective Huffman and optimal Huffman have been analysed theoretically. Run length based techniques like Golomb coding, frequency directed run length code, extended FDR, alternate run length code, alternate variable code etc. have been compared experimentally. Along with the comparison of run length based codes this thesis report also proposes an efficient and enhanced compression code to improve the efficiency further. This is a multistage code with AVR and Golomb at stage one and 9C at stage two. The efficiency of this has been compared with the already discussed run length codes and it was concluded that it was more efficient than any of these. Also the weighted transition a technique to calculate the average power dissipation has been used.

These techniques reduce the test data volume, which in turn reduce memory requirement of the ATE. These also help to reduce the dependency of the cores to be BIST ready.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

VLSI	Very Large Scale Integration
CMOS	Complementary Metal Oxide Semiconductor
SOC	System On Chip
IP	Intellectual Property
ATE	Automatic Test Equipment
TAM	Test Access Mechanism
FSM	Finite State Machine
FDR	Frequency Directed Run Length
EFDR	Extended Frequency Directed Run Length
AFDR	Alternate Frequency Directed Run Length
9C	Nine Coded
CCPRL	Count Compatible Pattern Run Length
AVR	Alternate Variable
CUT	Circuit Under Test
DUT	Device Under Test
ICC	Integrated Compression Code
ECC	Enhanced Compression Code
TD	Test Data
CBS	Column Bit Stuffing
CBSDIFF	Column Bit Stuffing With Difference Vector
ZEROFILL	Zero Filling
MTFILL	Minimum Transition Filling
ATPG	Automatic Test Pattern Generation
RFILL	Random Fill
SD	Scan Data
TAT	Test Application Time

CHAPTER

1 INTRODUCTION

1.1 Role of Testing

With rapid strides in Semiconductor processing technology, the density of transistors on the die is increasing in correspondence with Moore's law [1] which in turn is increasing the complexity of the whole SoC design. With the emergence of cutting edge technology applications like set top boxes, HDTV, an increasingly evident need has been that of incorporating the SoC the whole system - on a single silicon i.e., Silicon On Chip (SoC) using standard IP-Cores. Keeping in mind crucial manufacturing yield and time-to-market schedules, for a SoC, it is important to select verification and analysis solutions that offer the best possible performance, while minimizing iteration time and test data volume. Once a product is designed and fabricated, it should be tested before it is sent to the market. If it fails the test, then it is clear that something might have gone wrong [2]. The causes can be either the test is incorrect or the fabrication process has some flaw or the design is inaccurate or the specification has some problem. A strategy is devised for a more streamlined approach in IP-core based SoC verification which helps in smooth transition from design to chip tape-out stage. The role of testing is to detect if anything wrong has happened and the part of diagnosis is to find out exactly what went wrong. Testing is done to search for the weak chips, i.e. the chips with the flaws. So, correctness and effectiveness is very important for quality products.

The advantages of testing are quality and economy. This implies, meeting the needs of the user at least possible cost. A good test process can remove all faulty products before they are sent to the market.

1.2 Test vector generation

To test a chip a simple method of comparing the response of CUT with the already stored responses of a good machine is followed. To generate the responses we need to have input stimulus. These are known as test patterns. Manual test pattern generation is expensive so we need automation. Thus, the equipment generating the test patterns is known as ATPG. There are various ATPG algorithms like D, Podem [1] etc. available for the generation of test patterns. ATPG algorithms introduce a fault in the circuit and they use a process to promote this fault inside the hardware and then examine it at the output. For the propagation of faults, the scan chain design is used.

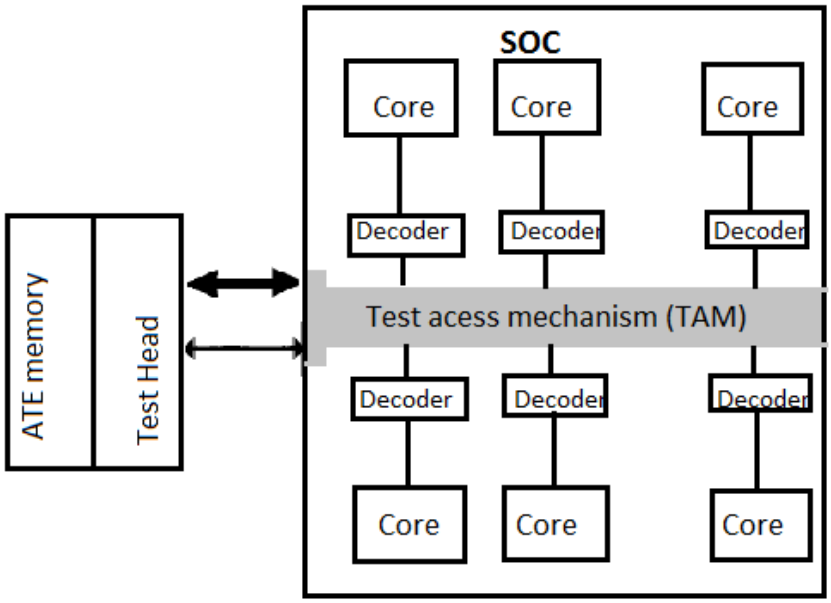

Figure 1.1: Architecture of test equipment [3]

The figure: 1.1 represents the conceptual architecture of test equipment. Here the test data is first stored in the ATE memory. This test data can also be the encoded test set. This is then decoded by the on chip decoder.[3]

The Level Sensitive Scan Design (LSSD) technique is used to observe and control all the flip flops. The figure: 1.2 shows the technique of linked positive edge triggered flip flops. These concatenated or linked flip flops are termed as scan chain [2]. The scan enable signal is used to convert the flip flops in a chain. If this enable signal is kept one then the flip flop work as a scan chain in which output of a previous signal is connected to the input of the next flip flop. The stimulus pattern is shifted in the scan

chain by the ATE with the help of scan input. Similarly, the ATE shifts the responses through the scan chain using scan chain.

Now days we even have built in self test (BIST) available within the chips. Thus in built tests are beneficial for maintenance and repair of the chip. But this demands for the additional hardware [4]. Moreover, it also wants the IP cores to be BIST ready[6]

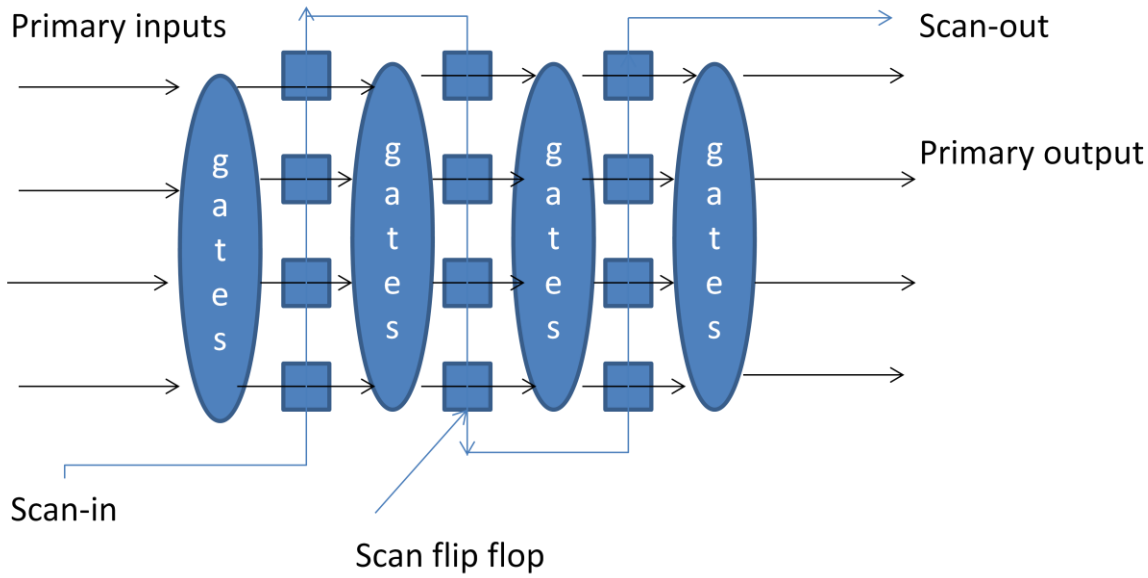

Figure 1.2: flip flops converted to scan chain

1.3 Challenges of testing

With the emergence of the CMOS process, core-based design method is intensively used to design System-on-Chip (SOC). As the technology is preceding towards ULSI, the no of transistors on a single chip are have increased to 10 millions. This has led to increment in test data, which has become vast now days. The test data is generally produced at work stations. The rising variations in the ASIC designs and reduction in production of individual types of ASICs demands more continuous downloads of the test data sets from workstations to ATE(Automatic Test Data equipment). Since the size of tests is very large, mostly of several gigabytes, the time implied in the downloading of the test sets is quite notable. There is network present which connects the workstations and ATE and this network supports the downloading, which generally takes several tens of hours.

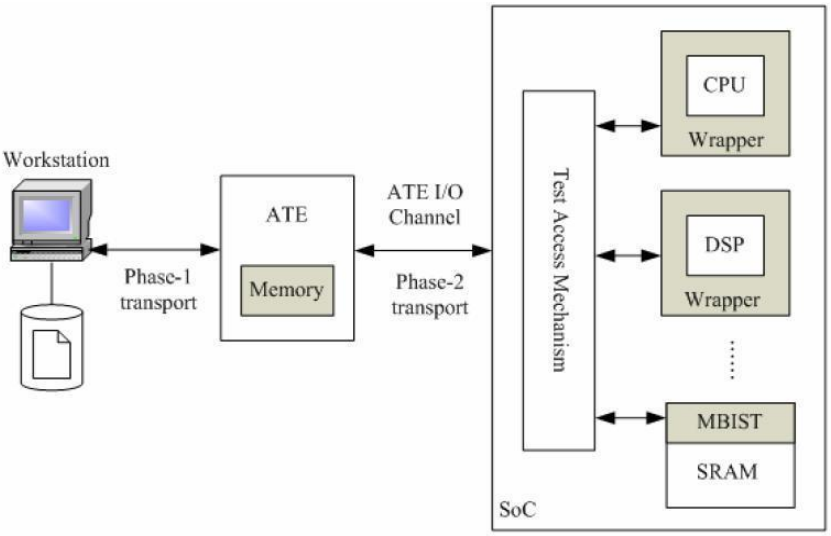

Figure 1.3: Test data being transported and test mechanism used for SOC testing [5]

Then this test set is transported from the workstation of the ATE to the main pattern memory with the help of high speed bus. This transfer takes several minutes. The entire process of transferring is shown in fig: 3. during the task of downloading the test data from work station to ATE, the ATE is sitting idle, and its speed and throughput is reduced. To polish the throughput of an ATE, it is important to decrease the test time of test data. A cost decreasing approach to achieve the solution to above problem is to compress the data.

The Automatic Test Equipment (ATE) tests chips by (1) applying stimulus patterns to chip inputs, (2) measuring the responses or the outputs (3) and then comparing these with already stored fault free circuit responses. Rising test data is the dominating factor leading to increase in the cost of the test data. A favourable proposal is to store and transmit compressed test data. For this we need and additional on chip circuitry to decompress the compressed test data. This additional circuitry is known as Design For Testability (DFT) circuitry.

1.4 TEST COST

The overall test cost [2] is given by the following equation:

$$C_T = C_P + C_E + C_S + C_{TQ} \quad (1.1)$$

Here, C_T represents total cost. C_P is test preparation cost, C_E is test execution cost, C_S is test cost related to silicon, and C_{TQ} is imperfect testing quality cost.

Test execution cost 'C_E' consists of test-related hardware cost. The hardware includes probe cards and cost incurred by tester use. Tester-use cost depends on various factors including time required to set up a tester, test execution time (as a function of die area

and yield), and capital cost of the tester (as a function of die area and number of I/O pins) and rate of capital equipment depreciation.

Test cost related to silicon C_S is the cost required to incorporate DFT features. This cost is modelled as a function of Wafer size, die size, yield, and the extra area required by DFT. The differences in parameters such as area, yield, and fault coverage between DFT and non-DFT designs are modelled by using appropriate multiplying or additive factors.

Imperfect testing quality cost 'C_{TQ}' is often ignored but is actually one of the most essential test cost components.

The test preparation cost 'C_P' further consists of test generation, tester program and DFT design. The test generation cost and tester program are related with ATE and are discussed in the following paragraph.

Although a (ATPG) with scan chains reduce the testing cost, the all-time rising number of gates results in all time rising number of patterns. The rising number of stimulus produces following problems[7]:

a) Restricted Bandwidth between workstation and ATE: the produced test data needs to be downloaded from workstation to ATE memory. Because of the restricted data bandwidth between ATE and Workstation, this will definitely take several hours. For this duration the ATE will sitting idle, thereby wasting its time.

b) Restricted ATE Memory: ATE with such a large memory may not be available in the market or will be very expensive. So, the test patterns are sometimes tarnished r, thereby reducing the quality of testing.

c) Restricted Bandwidth between ATE and DUT: The test stimulus stored in the ATE memory need to be applied to CUT. But because of the limited bandwidth between ATE and CUT, this may take a large amount of test time. The test cost increases with rising test time.

1.5 Reducing the Cost of Test

With the use of Built in Self Test (BIST) circuitry can decrease the cost of the test. This circuitry is capable of generating all stimulus patterns, applying the pattern to the circuit, evaluating all output data, and determines whether or not a faulty response was obtained [8]. The advantage of BIST is that no ATE is required. Hence, BIST can

also be used to test the device in field. But the area overhead of BIST can be quit notable. Additional requirement is that BIST design may depend on pattern sets. This demands the last minute changes in the design of BIST. Moreover there are many chips which are not BIST ready.

Thus test data compression provides a promising solution in the reduction of the test cost. The stimulus data is decompressed by test circuitry and response is compressed, as shown in fig. 3. On following this, there is a need to compress the precompiled test set of an IP-core, to a much smaller test set. This reduced test set is stored in the ATE memory. Then an on-chip decoder is used for decompression. This thesis focuses on the various algorithms used for test data comparison.

1.6 Matrices for Calculating Test Efficiency

Testing in test data compression environments (TDCE), means sending the compressed test data set from the ATE to the on-chip decoder, decompressing the test data set on chip and sending the decompressed test data to the circuit under test (CUT) [8]. There are two major components in TDCE: The compression method, implied to compress the test set off-chip, and the related decompression method, based on an on-chip decoder, used to restore the original test set on-chip. The on-chip decoder consists of two units: a unit to identify a compressed code and a unit to decompress it. Testing in TDCE can be characterized by the following three parameters:(a) compression ratio which identifies the performance of the compression method, the memory and channel capacity needs on the ATE; (b) area over head imposed by the on-chip decoder (dedicated hardware or on-chip processor); (c) test application time given by the time needed to transport and decode the compressed test set. There are a number of factors which influence the above parameters: the mapping and reordering algorithm, which prepares the test set for compression by mapping the "don't cares" in the test set to '0' or '1' and reordering the test set, the compression algorithm, the type of input patterns (fixed or variable length), the length of the pattern, and the type of the on-chip decoder (serial or parallel). Each of the three test parameters is influenced by these factors as follows:

Compression ratio: Compression algorithms that use various types of patterns and different lengths exploit different features of the test set. Using mapping and reordering of the initial test set, those features can be emphasized. Therefore, the

compression ratio is influenced firstly 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.

$$\text{Efficiency}=\frac{original\ test\ data-compressed\ test\ data}{original\ test\ data}\times 100\ \% \quad (1.2)$$

Decoder area overhead: Area overhead is influenced firstly by the nature of the decoder, and then by the type of the input pattern and the length of the pattern. If the decoder has a serial nature then synchronization between the two units(code identification and decompression) is already at hand. However, if the decoder has a parallel nature, the units have to synchronize one with each other, and hence increasing the area overhead. The type of the input pattern requires different type of logic to generate the pattern on-chip. For example, the use of fixed-length input patterns requires as shift register, while the use of variable-length input pattern (runs of 0s for example) requires a counter. On the other hand, the length of the pattern impacts the size of the decoding logic, and hence it influences area overhead.

Test application time (TAT) [10]: TAT is firstly influenced by the compression ratio, and then by the type of the on-chip decoder, the length of the pattern. To illustrate the factors that influence TAT, consider the ATE operating frequency as the reference frequency. If ATE frequency is f_{ATE} and scan frequency is f_{SCAN}, then f_{ATE} is always less than f_{SCAN} because f_{SCAN} depends on test data. If the test data becomes huge then f_{SCAN} also increases. The following equation gives the test application time reduction

$$\text{TR\%} = {t_{no\_comp} - t_{comp}}\Big/{t_{no\_comp}} \times 100 \quad (1.3)$$

Where, t_{no_comp} is the test application time required for applying uncompressed data and t_{comp} is the time required to apply compressed data. Minimum TAT (minTAT) is given by the size of the compressed test set in ATE clock cycles. But, this TAT can be obtained only when the on-chip decoder can process the currently compressed bit before the next one is sent by the ATE. In order to do so, the relation between the frequency, at which the on-chip decoder operates and the ATE operating frequency have to satisfy certain conditions. The frequency ratio can be defined as the ratio between the on-chip test frequency and the ATE operating frequency. Consider that

the minTAT is obtained when the frequency ratio is greater than an optimum frequency ratio. Since the minTAT is given by the size of the compressed test set, increasing the compression ratio would imply further reduction in TAT. However, this reduction happens only if the optimum frequency condition is met. Since real environments cannot always satisfy the optimum frequency ratio condition, then a obvious question is what happens if this condition is not met? TAT in this case is dependent on the type of on-chip decoder. Since the length of the pattern determines the optimum frequency ratio, consequently it also influences TAT.

Test power: Apart from the above discussed matrices for calculating test efficiency test power dissipation is another important parameter which needs to be taken care of. A circuit dissipates more power in test mode than in normal mode. Due to the higher consumption of power during testing, circuit can be degraded. Low-power testing is beneficial in surpassing the risk of damaging the CUT. Cost and time for testing are the major goals in the future development of VLSI design [30]. In scan testing, power consumption can be categorized into capture power and shift power. During testing a much larger percentage of the flip flops will change value in each clock cycle, which in return will increase the switching activity in the circuit. With the rapid increase in chip complexity, the problem of enlarged average power during scan testing is becoming an important issue in industry. Thus reducing the test cost is an important parameter to be taken care of to make the test cost more efficient.

1.7 Organization of Thesis Report

The organization of the dissertation is as follow

- Chapter 2 Presents the literature Review which includes discussion on the power reduction and compression techniques
- Chapter 3 Gives a detailed description of proposed algorithm for test data compression
- Chapter 4 Provides experimental results for ISCAS benchmark circuits
- Chapter 5 Conclusion of the proposed algorithm's efficiency in test data compression and future scope of the improvement in the techniques to improve the efficiency is discussed.

CHAPTER

LITERATURE REVIEW

Test data volume and its reduction for testing of a system has manifested itself as a significant problem. The testing time of a system mostly depends on the test data volume which is a function of number of test patterns for that input data and the number of stimulus and response bits per pattern. Therefore in order to minimize test data, there is a need to compress the input data. This chapter gives the description of the various schemes implied to compress the test data

For a large complex circuit, a designer or a test generator generates test patterns considering one or few sets at a time [11]. Thus, a block of test patterns usually examines a few modules of the circuit, while other sets are put to some static conditions. This means that logic value for only a subset change for block of test patterns, while other pins are kept stagnant logic value. Apart from this, another important observation is that the sequence of test patterns of an active pin often forms cycles. The literature survey has been broadly divided in two parts, namely test power and test compression.

2.1 Test Power Reduction

2.1.1 Weighted transition metric model

In a CMOS circuit, there is very little static power dissipation. The predominant fraction of the power is dissipated when the circuit elements switch from logic 1 to 0 or vice versa. Under test, the circuit is entirely controlled by the test vectors that are applied to it. The elements in the circuit-under-test (CUT) will switch when the primary inputs change value or when the scan flip flops change value.

For scan vectors, the dynamic power consumption during testing depends on the number of transitions that occur in the scan chain as well as on the number of circuit elements that switching during scan in and scan out operations. Power dissipation is

calculated based on Weighted Transition metric model based as introduced in [12]. The WTM models the fact that the scan in power for a given vector depends not only on the number of transitions in it but also on their relative positions. For example, consider a scan vector v1v2v3v4v5=01000. The 0 to 1 transition between v1 and v2 more switching activity in the scan chain than the 1 to 0 transition between v2 and v3. We use same model to estimate power during scan out operation.

If the peak power becomes more than a threshold value, it can cause structural damage to the silicon or to the package. Likewise, increased average power can also cause structural damage to the silicon, bonding wires, or the package. It also adds to the thermal load that must be transferred away from the device under test.

Scan in power depends on the manner in which the don‘t cares in T_D are mapped to binary values. While P_{avg} and P_{peak} can be reduced to minimal by choosing an appropriate mapping. However, such a mapping is not guaranteed to provide high compression along with reduced transitions during scan in. Even though scan out power cannot be directly address, the experiments with benchmark circuits prove that this approach reduces the number of transitions and the resulting WTM during scan out. This is an added advantage of using encoded test sets for scan testing.

Scan out vectors depend on the Scan in vectors and the way in which the two are related depends on the function implemented by the circuit under test (CUT). Determining the effect of filling don‘t cares cubes on the Scan out vectors need circuit simulation. The time required to compute the effect of each decision on the scan out vectors, cannot be afforded. Thus the only focus is on minimizing the scan-in power.

Some of the features of the test data which are helpful for the test power reduction and compression of the test patterns are discussed below.

2.1.2 Hamming Distance based reordering [13]

Given two bits i, j that belong to {0,1, X} , i and j are incompatible if i=1 and j=0 or vice versa. In all other circumstances they are compatible. The hamming distance between two scan frames is the is given by total number of corresponding incompatible bits. For instance, for two given frames F1=(100X01) and F2=(110110), the distance between F1 and F2 is d=3, as the second, fourth and fifth bits are incompatible.

2.1.3 Bit stuffing

Bit stuffing implies injection of bits inside the transmission unit as a way to provide signalling information to a receiver [12]. The receiver's function is to detect and remove the stuffed bits. Whereas testing is concerned, the bit stuffing means exchanging don't care terms with zeros or ones. This is done to increase the number of zeros or ones. Few techniques used for bit stuffing are discussed below:

a) Random fill (RFILL)

Random fill [13] is an older approach for the Xs in the test sets. RFILL demands replacing of the Xs randomly with 1s and 0s. The idea behind this approach is that it raises the chance of detecting the additional faults with a single test cube to hopefully remove the requirement for other test cubes, thus they will be removed from the test set when it is reverse fault simulated.

Though, RFILL is not good in terms of power, but random filling of Xs may result in lot of transition while scanning the vectors into the scan chain, which may result in lot of switching activity.

Another option for RFILL is available in all the present ATPG generates randomly specified compacted test data vectors for any given circuit.

As far as power is concerned it is better to use Minimum Transition Fill (MT Fill)

b) Minimum Transition Fill (MT Fill)

Minimum Transition Fill [14] means filling X's strings with the same value to minimize the transitions. For example, while filling the test cube 001XXXX1, it would be most appropriate to fill the strings of X with 1s i.e. 00111111.

For each chain of Xs in a test cube, if the specified bits on both sides of the string have the same the same value, then the chain of Xs should be filled with that value to make the number of transitions least. If there values are opposing, then it does not matter with what value the string is being filled. For instance when filling 0XX01X1X0, the first two Xs has be filled with 0, the third X with a 1 and the last X could be filled randomly with 0 or 1. Although MT-Fill minimizes the power, the drawback is that it may not be as effective as R-fill [14] for detecting additional faults.

As a result, the reverse fault simulation step may not drop as many test vectors from the test set when MT-Fill is implied instead of RFILL. If the initial test cubes are not

statistically compacted, but simply filled with MT-FILL, then the resulting test set will provide the minimum number of transitions per test vectors for that initial set of test cubes. When two test cubes are clubbed, the number of transitions in the resulting test cube is not less than the maximum number of transitions in either of the two original test sets after MTFILL. This happens because the resultant test cube can be formed by specifying Xs in either of the original test cube. There is no way to specify Xs in either test cube so that there are fewer transitions than MT-fill. Hence, the peak power is minimum for initial test cubes, and it monotonically increases as static compaction is performed, i.e., it can never decrease.

c) ZERO FILL

The demand of the run length based compression codes is longer runs of 0s for better compression results [12].

For example, filling the 01XXX1 with 0 will result in 010001, thereby increasing the runs of 0s. This will increase the compression ratio, but will adversely impact the switching activity which may depend on the nature of the test set generated by the ATPG.

d) COLUMN WISE BIT STUFFING

Column wise Bit Stuffing (CBS) [12] is a technique which when used with difference vector increases the compression ratio to a larger extent compared to other X's filling approaches discussed above.

Consider the following example:-

1011XX000XX010

110X0X0X10X1XX

0X101XX00X00X1

0XX0XX01XXX0X0

001X1X1X10X001

The don't care filling of first vector can be done in a way that it generates maximum zeroes to give maximum compression with run length based codes like Golomb, FDR. So for these cases, the don't care bits will be filled by zeroes. For run length based codes like EFDR and AFDR, the better compression can be achieved by

increasing the run length of 1s as well as 0s. For such case, the don‘t care bit will be replaced with the same value of that bit which is in the beginning of the don’t care string i.e. if the initial bit is 1 then the don‘t care will be refilled by 1 and if the initial bit is 0 then the don‘t care will be filled by 0.

For the second test patterns onwards, the don‘t care bit will be filled by the same value which its above vector has at the same bit position. The goal here is to get the maximum zeroes in difference vector. In given example, the first vector is 1011XX000XX010. Thus here all don‘t care bits are made zeroes and new bit stuffed vector will be 10110**0**00**000**010. Now for the second test pattern, the bit stuffing will be as follows:

FIRST PATTERN: 1 0 1 1 0 0 0 0 0 0 0 1 0

SECOND PATTERN: 1 1 0 X 0 X 0 X 1 0 X 1 X X

CBS RESULT : 1 1 0 **1** 0 **0** 0 **0** 1 0 **0** 1 **1 0**

The resulting vectors after column bit stuffing become: 10110000000010, 11010000100110, 11100000100011, 01100010100011 and 101010101000001

2.1.4 Difference Vector

If $T_D = \{t_1; t_2; \dots \dots t_n\}$ be the test set. T_{diff} is defined as : $T_{diff} = \{d_1; d_2; \dots \dots d_n\}$ = $\{t_1; t_1\ XOR\ t_2; \dots \dots; t_{n-1}\ XOR\ t_n\}$; 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 small number of bits [15]. Thus T_{diff} contains lesser number of 1s and it can be compressed effectively.

For example

VECTOR 1: 1011000001110

VECTOR 2: 1111001000110

DIFF VECTOR: 0100001001000

This example clearly shows that after using difference vector technique the run length of zeros has increased, which will further help in improving the compression

2.2 Test Data Compression techniques

Test data compression plays an important role in reducing test cost as already discussed in section 1.5, hence it becomes very important to discuss the test data compression techniques. Test vector compression schemes can be classified broadly into three categories [17]:

(i) *Code-based schemes*: test cubes are encoded using compression codes.

(ii) *Linear-decompression-based scheme*: the data is decompressed using only linear operations (that is LFSRs and XOR networks).

(iii) *Broadcast-scan-based schemes*: broadcasts the same values to multiple scan chains.

2.2.1 Code-based schemes:

Code-based schemes use data compression codes to encode the test cubes. This involves converting the original data into symbols, and then writing a code word for each symbol to form the compressed data. For decompression, a decoder converts each code word in the compressed data set again into the corresponding symbol. These can be further divided as run length based codes, statistical codes, dictionary based codes and constructive codes.

Linear-decompression based schemes and Broadcast-scan-based schemes are beyond the scope of this section and hence have not been discussed.

2.2.1.1 Statistical codes

Statistical coding divides the original data into *n*-bit symbols and assigns variable-length code words depending on each symbol's frequency of presence. It provides with smaller code words for the symbols which are present more frequently, and larger code words to those which are present less frequently. This kind of algorithm minimizes the average length of a code word. Jas et al. Described a scan vector compression scheme based on selective Huffman coding.[18] A Huffman code is statistical code that provides optimal reduction, but its decoder size grows exponentially as the symbol size increases. Selective Huffman coding encodes only the most commonly occurring symbols, leaving the rest as it is. This supports the use of longer symbol sizes, in turn providing greater compression than what equally sized,

full-Huffman decoder (with a shorter symbol size) could achieve. The examples of statistical codes are as follows:

- **Selective Huffman coding [24]** :

An important property of Huffman codes is that they are prefix-free. No codeword is a prefix of another codeword. This greatly simplifies the decoding process. The decoder can instantaneously recognize the end of a codeword uniquely without any look ahead. The amount of compression that can be achieved with statistical coding depends on how skewed the frequency of occurrence is for the different code words. If all of the code words occur with equal frequency, then no compression can be achieved.

- **Optimal selective Huffman coding[25]:**

The inefficiency of the Selective Huffman encoding of 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. This constant overhead, though minimum for the unencoded data blocks, can be relatively high for the most frequently occurring code words. A better approach would be to use an additional Huffman codeword in front of only the unencoded data blocks, alleviating 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 volume). Specifically, according to the proposed Selective Huffman approach, m þ 1 Huffman code words are used, m for encoding the m most frequently occurring distinct blocksand1 for indicating the unencoded blocks. The occurrence frequency of the latter codeword is equal to the sum of the occurrence frequencies of all the unencoded distinct blocks.

2.2.1.2 Dictionary based codes

Dictionary coding is yet another form of coding. It divides the original data into *n*-bit symbols and each unique symbol is stored using a dictionary [17]. This scheme compresses data by encoding every *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 disadvantage of using a complete dictionary is that the size of the

dictionary can become very large, which results in large overhead for the decompressor.

2.2.1.3 Constructive codes

Constructive codes works on the fact that each *n*-bit scan slice contains relatively few care bits. Each code word in a constructive code specifies a subset of the *n* bits in the scan slice. The scan slice can be constructed by increasingly specifying all the care bits by using a sufficient number of code words. There is a scheme that constructs the current scan slice from the previous scan slice by flipping bits [17]. A single bit is flipped by ever code word, so the number of code words used to actually construct each scan slice becomes equal to the number of care bits in the present scan slice that is different from those in the previous scan slice. This technique requires some control information to indicate when scan slice construction is complete and the slice is ready to be shifted into the scan chain.

2.2.1.4 Run length based schemes

The first data compression codes that researchers investigated for compressing scan vectors encoded runs of repeated values. It started with a scheme based on run-length codes that encoded runs of 0s using fixed-length code words.1 To increase the prevalence of runs of 0s, this scheme follows a cyclical scan architecture to allow the application of difference vectors where the difference vector between test cubes $t1$ and $t2$ is equal to $t1 \approx t2$. The ordering of the test cubes increases the number of 0s in the difference vectors, thus enhancing the effectiveness of run-length coding. Codes like Golomb, FDR, EFDR, CCPRL, AFDR etc are examples of run length based codes.In code based schemes the internal architecture of the IP cores is not changed hence they are most easy to use. The thesis is based on these schemes.

- **Golomb code[19]:**

It is variable –to-variable length code. Variable to variable means if the run of zeros varies then the code word also varies for it. Step (1) is to determine Golomb code parameter m, referred to as group size. *M* is generally taken in the power of 2. The run of zeros in T_{diff} are mapped to group of size m. A Codeword consists of group prefix and tail. The bit length of group prefix is given by L+1, this value also gives group number, L is a run of length, eg. When L=5, the group prefix and tail are 2 bits. The tail is a sequence of $\log_2$ m bits. To find out the optimum value of m, first of all the

probability p of runs of 0‘s is found out in the test data set. Then m is given by-

$m=\text{ceil}\left((1/\log_2(p))*\log_2(1/(1+p))\right)$ (7)

Table 2.1: Golomb code for *m* = 4

Group	Runs of zeros	Prefix	Tail	Codeword
A_1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
A_2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
A_3	8	110	00	110000
	9		01	110001
	10		10	110010
	11		11	110011
A_4	…..	…….	……..	……..

The block diagram of the Golomb decoder is presented in Figure 3, it has a 9-state finite state machine (FSM) and an *i* bit counter. Where value of *i* is given by $log_2 m$. The primary input is *bit-in*. When the signal *en* is high, the encoded test data is sent via *bit-in* to FSM. The *inc* is a count-up signal to increment the counter. The high condition of *rs* signal/line will show that the counter has finished counting.

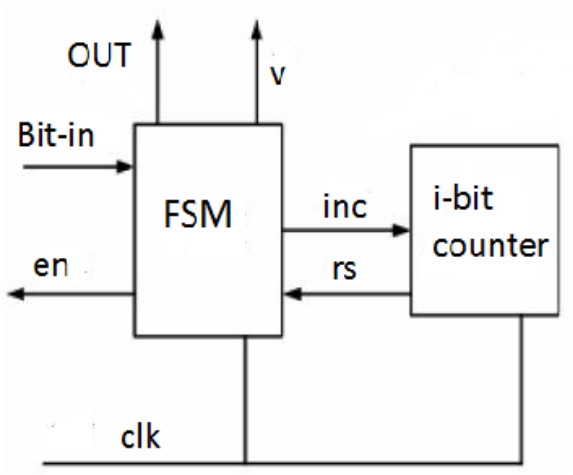

Figure 2.1: Decoder of Golomb code

The *out* is the output of the decoder and gives the decoded or the original data. When *v* is high, it indicates the output data is valid. The working of decoder is that the counter counts up to m cycles and enable the next input data is ready to be sent. Moreover, the *v* is high as the outputs m 0‘s while the counter is doing count-up. On the other hand, when *bit-in* is 0, the counter stops, thus, FSM follows the tail of codeword, and sends out few 0s and signal 1 but out, then the working of decompression would be competed in order.

- **Frequency directed run length code [20]:**

It is a modified form of Golomb code. There is a gap of two bits between every the code word of two successive groups. For any codeword, the prefix and tail are of equal length. For example, the prefix and the tail are each one bit long for A_1, two bits long for A_2 , etc. The length of the prefix for group $A_i$equals i and also there are 2^i members in each group. For example, the prefix is two bits long for group A_2. For any codeword, the prefix is identical to the binary representation of the run length corresponding to the first element of the group. For example, run length 8 is mapped to group A_3 and the first element of this group is run length 6. Hence, the prefix of the codeword for run length 8 is 110. The code word size increases by two bits as we move from group $A_i to\ A_{i+1}$.

Table 2.2: FDR code

Group	Runs of zeros	Prefix	Tail	Codeword
A_1	0 1	0	0 1	00 01
A_2	2 3 4 5	10	00 01 10 11	1000 1001 1010 1011
A_3	6 7 8 9 10 11 12 13	110	000 001 010 011 100 101 110 111	110000 110001 110010 110011 110100 110101 110110 110111
A_4	…..	…….	……..	……..

The block diagram of FDR decoder consists of three parts: 9-state FSM, *log_2 k-bit* counter (counter2) and *k-bit* serial input counter(counter1) as depicted in Figure 2.5, The *bit_ in* is an input pin which is used to enter encoded data when the *en* is high, and the *out* is an output pin used for getting decoded data. The output is valid only if the *v* pin is high. The *counter-in* is a line, used to send data to *k-bit* counter, and the *shift* signal takes control of the *counter-in*. The *dec1* is used to notice if the *k-bit* counter has starting count-down, and the *rs1* becomes high as the counting finishes. The *dec2* signals if *log_2 k-bit* counter has started doing count-down, *inc*, on the other hand, signals if it has started counting up. The *rs2* indicates when it has finished counting.

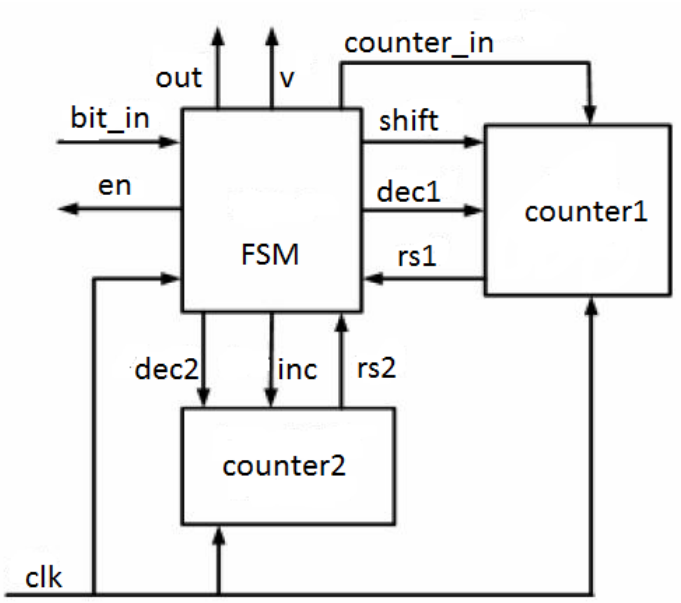

Figure 2.2 Decoder of FDR code

- **Extended FDR[21]:**

To encode both run's of 0's and 1's the FDR code can be extended based on adding an extra bit to the beginning of a code word to indicate the type of run. If the bit is 0, this indicates that the code word is encoding a run of type 0, otherwise it encodes a run of type 1. The rest of codeword stay the same. The code is called extended FDR. The code is shown in the following table. However, in this code there is no run length of size0. This is because encoding is done for both run's of 0's and run's of 1's. The extra bit must be located in MSB of codeword to inform the decoder whether it should invert the output data.

As shown in Figure 2.6, before propagating prefix data, the extra bit will enter first to enable XOR gate by FSM thorough the *mux* signal, if the *mux* is 1, the out is inverse the *fout* data, if it's 0, the out is equal *fout*, through the signal *mux* can decode codeword which encode by run 0s run-length or run 1s run-length. Rest of the working of decoder is same as discussed for FDR except *mux* and XOR gate

Table 2.3: EFDR Code

Group	Run length	Group Prefix	Tail	Code word in 0's	Code word in 1's
A1	1	0	0	000	100
	2		1	001	101
A2	3	10	00	01000	11000
	4		01	01001	11001
	5		10	01010	11010
	6		11	01011	11011
A3	7	110	000	0110000	1110000
	8		001	0110001	1110001
	9		010	0110010	1110010
	---		---	---	---

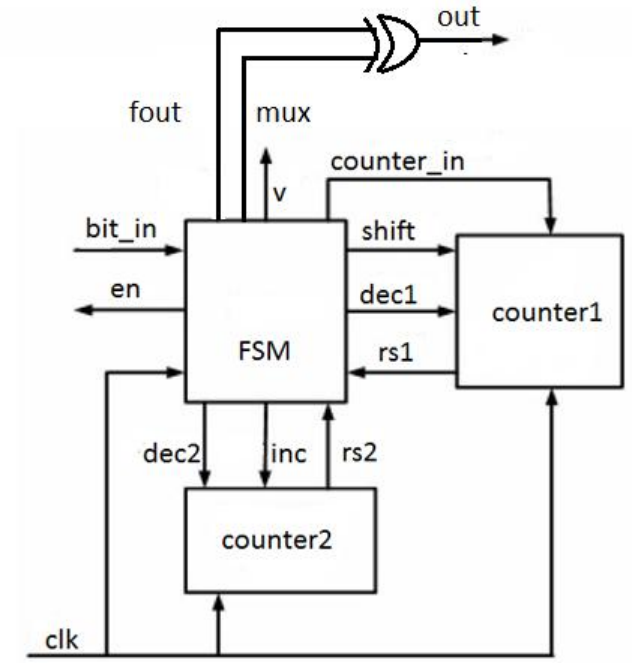

Figure2.3: Decoder of EFDR

- **Alternate run length code[22]:**

It is also variable to variable length code and consists of two parts group prefix and tail. The prefix identifies the group in which the run length lies and tail identifies the number within the group. An additional parameter associated with this code is the alternating binary variable *a*.

Table 2.4: AFDR code

Group	Runlength of 0/1	prefix	Tail	codeword
A_1	0 1	0	0 1	00 01
A_2	2 3 4 5	10	00 01 10 11	1000 1001 1010 1011
A_3	6 … 13	110	000 111	110000 110111

The encoding produced by the alternating run length code for a given run length depends on the value of *a*. If *a*=0, the run length is treated as run's of 0's. On the other hand if *a*=1,the run length is treated as run of 1's.

The circuit block diagram of AR decoder is depited in Figure 6. There is an addition of a T Flip-Flop which is negatively triggered and a XOR gate in the circuit of FDR decoder. The operation of additional part is to provide with a signal feedback and to provide negative trigger to control the point t, whenever t=1, the output data of out is invert of fout, when t=0, the output of *out* and fout is equal. The operation of the rest part is the same as that of FDR decoder.

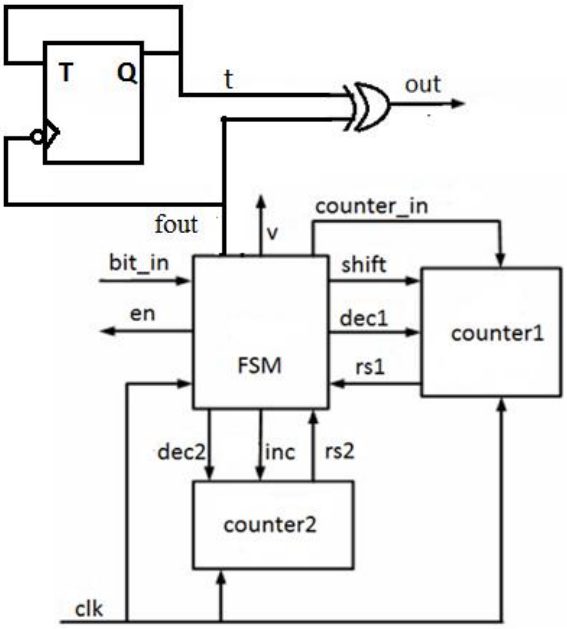

Figure 2.4 Decoder of AFDR

Input data stream: 000000011111111111000000001 (26- bits)

FDR encoded data: 110001000000000000000000110010(30-bits)

AR encoded data: 110001110011110010 (18-bits)

| run 0 | run 1 | run 01 |

Figure 2.5: Comparison of FDR coding and AFDR coding [21]

- **Fixed length pattern run length (PRL) coding[23]:**

The compression is performed by simply retaining one copy of a pattern and using"10"or"11" for specifying whether the other patterns are equal to or complement of the retained pattern. A "0" is used to terminate a round. For example, the vector101*x* 101*x*10*xx*010*x* 01*xxx*1*xx* would be compressed into 101*x*10101111110. In this compressed sequence the first 4 bits specify the retained pattern101*x*. The next "10" specifies the fact that the next 4bits are equal to the retained pattern.

- **Nine coded compression[26] :**

The proposed technique is based on fixed-length-input test data compression. Suppose that the fixed-length input blocks is of the size *K*. The input test vectors are partitioned into groups of *K* bits and encoding is performed on each *K*-bit block. Table shows the proposed coding for *K*=8. As shown in the table, each K-bit block is divided into two *K* 2 halves. As shown, there are overall nine codewords. For the rows 1 to 4, each half K-bit is matched with all 0's or 1's. Rows 5-8 show the mismatched bits, which have to be sent along with the codeword (UUUU is called a mismatch meaning this half has mix of 0 and 1 and perhaps don't-care X). Row 9 shows that each half is a mismatch and the entire K-bit block has to be sent along with the codeword. The third column shows a symbol for each case. The sixth column shows the decoder input which can be only a codeword (in the first four rows) or

Table 2.5: 9C coding

Case (i)	InputBlock	Symbol	Description	Codeword (*Ci*)	DecoderInput	Size (bits)
1	0	0	All0's	0	0	1
2	11111111	11	All1's	10	10	2
3	1111	1	Lefthalf0,righthalf1	11000	11000	5
4	11110000	10	Lefthalf1,righthalf0	11001	11001	5
5	1111UUUU	1U	Lefthalf1,righthalfmismatch	11010	11010UUUU	5+*K*/2=9
6	UUUU1111	U1	Lefthalfmismatch,righthalf1	11011	11011UUUU	5+*K*/2=9
7	0000UUUU	0U	Lefthalf0,righthalfmismatch	11100	11100UUUU	5+*K*/2=9
8	UUUU0000	U0	Lefthalfmismatch,righthalf0	11101	11101UUUU	5+*K*/2=9
9	UUUUUUUU	UU	Allmismatch	1111	1111UUUUUUUU	4+*K*=12

codeword plus the mismatch portion (in the last five rows). Finally, the last column shows the final code size for each case.

The decoder does the decoding of the 9 prefix free code words. These prefix free test sets are independent of previously generated test data set.

The decoder is made up of *counter_1*, *FSM* (finite_state_machine) and *reg_*$^{k}/_{2}$ register. The first few bits are made to enter the *FSM* through *data_in* and are then sent into the *reg_*$^{k}/_{2}$ register. Thus this *k* is used by decoder for the whole test data set. $^{k}/_{2}$ is sent only because 9c technique detects each half of a *k-bit* block separately for each codeword.

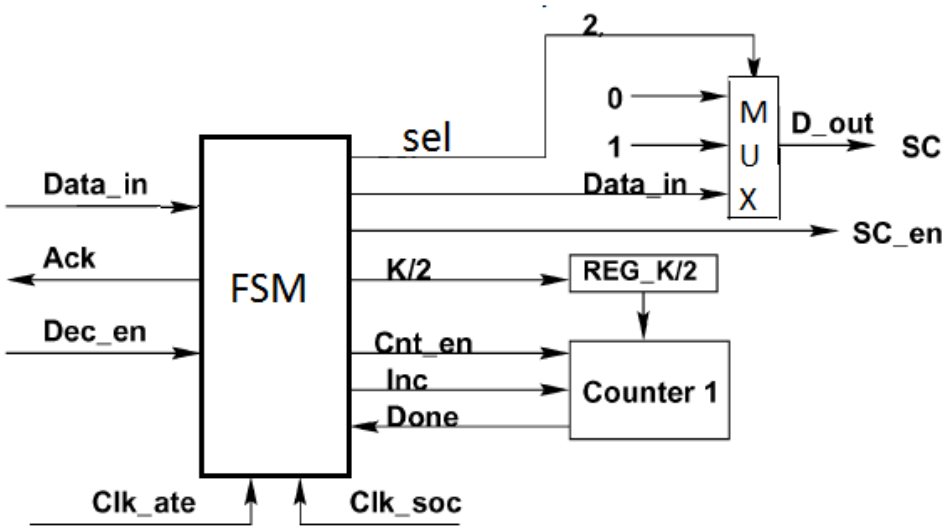

Figure 2.6: Decoder of 9C decoding

- **The Count Compatible Pattern Run-Length (CCPRL) [27]:**

Coding compression method is proposed to further improve the compression ratio. Firstly, a segment of pattern in the test set is retained. Secondly, don't-care bits are filled so as to make subsequent patterns compatible with the retained pattern for as many times as possible until it can no longer be made compatible. Thirdly, the compatible patterns are represented by symbol "0" (equal) and symbol"1" (contrary) in the codeword. In addition, the number of consecutive compatible patterns is counted and expanded into binary which indicates when the codeword ends.

The architecture of the decompressor is shown in the figure 9 it is simple and straight forward. The output of the FSM is coutn_in and is used to shift the retained pattern inside the k-bit counter, till $flag_1$=1. Shift signal is used to transfer the data in, dec_1

is used to decrement the counter and res_1 is used to mark the set state of the counter. The counter is now n_bit. Inc signal obviously increases the counter and dec signal as the name suggests decreases the counter. The finishing of the counter is marked by rs_2. If $flag_2 = 1$, it indicates that counter is in R mode else it is in C mode. If $flag_3 = 1$ the the register is in input mode else is in output mode. Finally, the signal out and signal contrary being opposite to each other gives the output of the decoder

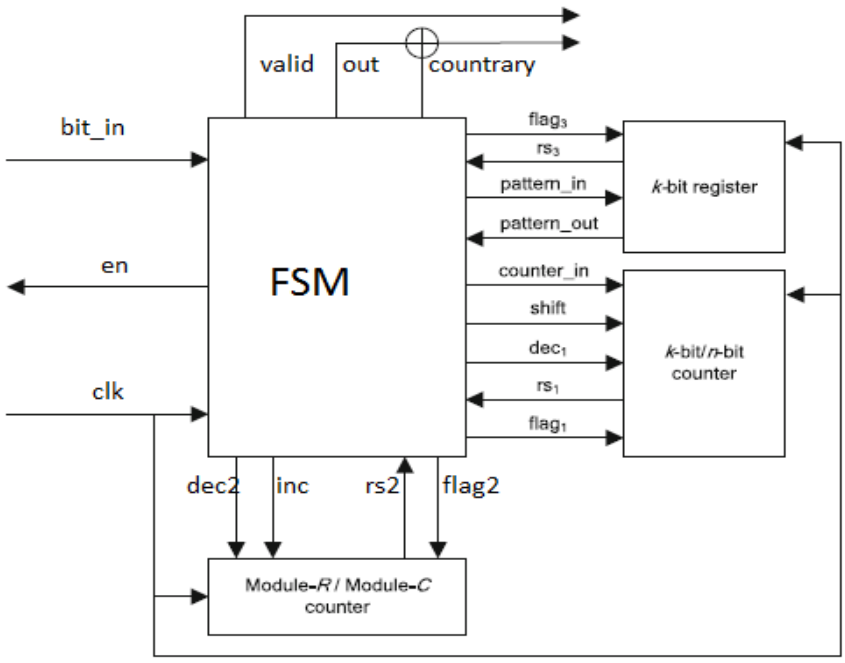

Figure 2.7: Decoder of CCPRL

- **Alternate variable runlength code [28]**

The alternating variable run-length code is also a variable to variable length code, consisting of two parts – the group prefix and the tail. The prefix determines the group in which the present run length lies and the tail tells us about the number of ones or zeros (runlength) within the group. The test data can b classified into runs of zero's ending with a one and runs of one's ending with a zero. Extra parameter 'a' was used to whether the run length being decoded was of '0'or '1'. This parameter was also used during decoding to produce the correct run length.

Table 2.6: AVR code

Group	Run length	Group prefix	Tail	Codeword
A_1	1	0	0	010
	2		1	011
	3	1	0	100
	4		1	101
A_2	5	00	00	00100
	6		01	00101
	7		10	00110
	8		11	00111
	9	11	00	11000
	10		01	11001
	11		10	11010
	12		11	11011
A_3	13	0001	000	0001000
	14		001	0001001
	..		….	………..
	20		111	0001111
	21	1110	000	1110000
	22		001	1110001
	….		….	……

Similar to the ALT-FDR decoder, an on-chip decoder decompresses the encoded test set $\boldsymbol{T_E}$ and produces the original test set $\boldsymbol{T_D}$. The decoder diagram can be drawn by making a small change to the decoder for alternate FDR. The testing time was decreased because test pattern decompression could be carried out on-chip at higher clock frequencies. The designing of decompression architecture is simple, and it is independent of the preciously computed test set and the circuit under test (CUT). $\boldsymbol{L_{max}}$ is considered to be the longest run of 0s or 1s in$\boldsymbol{T_D}$, and *k* is given by, *k*= $\boldsymbol{\lceil log_2(L_{max}+4)-2\rceil}$. The decoder is made up of a (k+1)-bit counter, a finite-state machine (FSM), a $\boldsymbol{log_2(K+1)}$-bit counter, a T flip-flop and an exclusive OR gate.

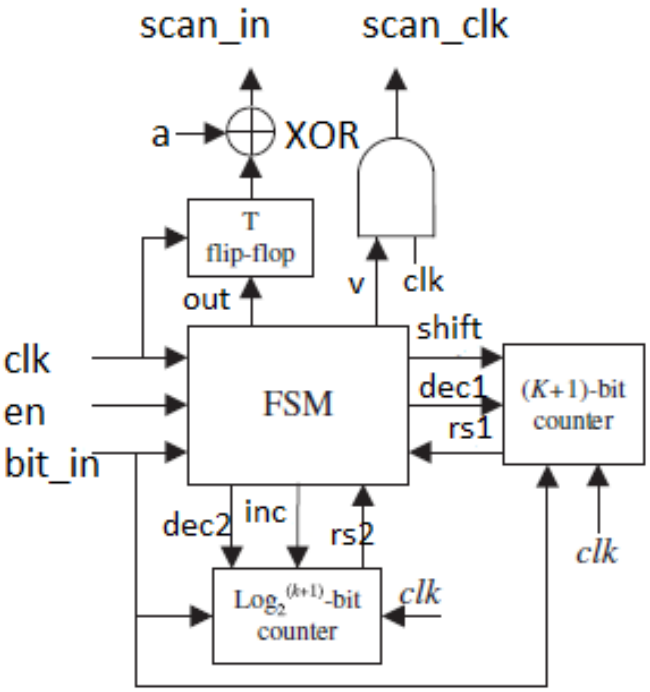

Figure 2.8: Decoder alternate variable code

CHAPTER

3

ENHANCED COMPRESSION CODE

The various runlength based codes discussed in chapter 2 are analysed with the help of an example. The analysis is shown in table 3.1 and the drawback of the Golomb code and FDR is that they do not compress the number of ones can be clearly observed. Best part of the Golomb code is that it provides shortest code for a large number of zeros. On comparing AFDR and AVR it is found that AVR provides with shorter codes, thus providing better compression. The drawback of CCPRL is that it fails when the successive blocks are not compatible or incompatible. 9c provides best compression among these but its decoder is complicated. This observation is best suited by the following example.

Let the test vector T_D = 101011100001101, number of bits= 16. The following table gives the comparison of all the codes.

Table 3.1: Analysis of various codes

Name of the code	Result after compression	Number of bits after compression
Golomb	000 001 001 000 000 000 1000 000 001	28
FDR	00 01 01 00 00 00 1010 00 01	20
EFDR	101 101 11001 11000 101 101	22
AFDR	01 00 00 00 1001 1001 01 00	20
9C	11011 11100	10
CCPRL	100 10100 111111 11010	19
AVR	010 011 101 101 010 010	18

The drawback of the Golomb code as already seen is that it is beneficial only for the run's of zeros. This was evident from the experimental results. Hence there is needed to encode runs of ones also. As we have seen in the above comparison that Golomb code is good to use only when we have larger runlength of zeros and when *m* size is kept 8 or 16 then it is best to use. Again from the above comparison it is important to note that AVR is most efficient in encoding ones but when moved from one group to another it increments two bits in the code word.

There is a way in which we can use both these codes. If we think of making such a code in which we use Golomb code for zeros and AVR for ones, then lets see if our compression improves for the above example. On following this idea we get following result:-

T_E= 010 011 101 1000 010 010, number of bits=19

3.1 Integrated compression code

We have seen that we have achieved no improvement in the compression. Rather, because of using Golomb code with *m*= 4, we have decreased the compression. So, we have to work on two new things, first is thinking about cases when run length is 0 or 1 and secondly using Golomb code with higher valued *m*. Further if for lower run length we use only AVR code and is we surpass run length 0 or 1, lets see what the result is

T_E = 1010 101 101 101, number of bits = 13

Thus with the above discussed improvements, we find that we have improved the impression. The following paragraph gives the summery of the above discussed changes and newly developed code.

The two codes are merged first. The code providing minimum number of bits to a particular run length is used. For runs of ones, only AVR code is used and for runs of zeros, the code providing least bits in the codeword is used. Also it is observed that if the run length 0 and 1 of whether one or zero are not encoded the bits are further reduced. Hence a *bypass mode* is used to bypass the coding of run length 0 and 1. This scheme is named as ICC.

Table 3.2: Integrated Compression Code

Run length	Code to be used	No. of bits in codeword
0	Bypass mode	0
1	Bypass mode	1
2-4	AVR 0/1	3
5-15	Golomb for 0/ AVR for 1	5-7
16-31	Golomb for 0/ AVR for 1	6-7

The Golomb code for *m*=16 is used above.

For example:

$T_{D=}$0000001 11111111111000010001000000011111100001

No. of bits =45

$T_{E=}$ 00110 11011 011 011 0111 101 100

No. of bits =26

Here we can see that on using Integrated Compression code (ICC), we get better results. Although here only one bit difference can be observed if coding is done through AVR but when the test data will be huge then the difference will b remarkable. Also there is no area overhead in the on-chip decoder. This is discussed later.

3.2 Enhanced compression code

Even after compressing the bits with ICC there are still more bits available which can be compressed. The compression can be thought of as the bits are still available as strings of ones and zeros. We can try and use nine coded compression (9c) [18] on the already compressed data to improve the compression further. This type of coding is known as multistage code, as the coding is done for two stages. Taking the above example

$T_{E=}$ 00110 11011 011 011 0111 101 100 No. of bits =26

$T_{E1=}$ 1111 1111 11110 No. of bits =13

After using ICC the compressed number of bits is 26 and then on implying 9c on this compressed data string, the number of bits is further reduced to 13. Thus improving the compression by 50% at the second stage. At the end of the multistage coding we achieve the compression = 71.11 %. If only 9c was used then compressed bits achieved are 29. Which are even lesser than above proposed ICC. Thus ECC is more efficient than all of these.

3.3 Decompression architecture

The decoder of Enhanced compression code has been explained in the following sections. Firstly the decoder of ICC has been explained, then in next part using this decoder we have implemented the decoder of the Enhanced compression code scheme.

3.3.1 Decoder of ICC

The decoder architecture is similar to the on chip decoder architecture of Golomb code[8] and AVR code[13] with minute additional signals. The decoder decodes the encoded test data to produce the original test data (T_D). The testing time has also been reduced as test pattern compression could be carried out on chip at higher clock frequencies. l_{max} is the longest run of zeros or 1s in T_D, and k=$\lceil log_2(l_{max}+4)-2 \rceil$. The decoder is made up of a k+1- bit counter, a $log_2(k+1)$ bit counter, a T filp flop and an exclusive or gate. Fig.1 shows the view of decoder architecture. It has following signals.

- The *select* signal tells us which code was used for encoding. If it is 0 then AVR has been used and if it is 1 then Golomb has been used
- *Bypass* signal reflects the bypass mode if it is high
- *Bit_in* as the name suggest feeds the FSM with the input bits and an enable signal *en* has been used to control the input when the decoder was ready
- *Shift* signal controlled the prefix and the tail of the codeword to shift in the k+1 bit counter. *Rs1* and *RS2* told about the reset stats of the two counters
- Another $log_2(k+1)$ counter has been used to count prefix and tail, which helped in identifying the group.
- The out of the FSM controlled the toggle of the T f/f, and indicated that it has decoded the runs of 0's and 1's according to the binary parameter 'a'. Single *v* signifies the validity of the output

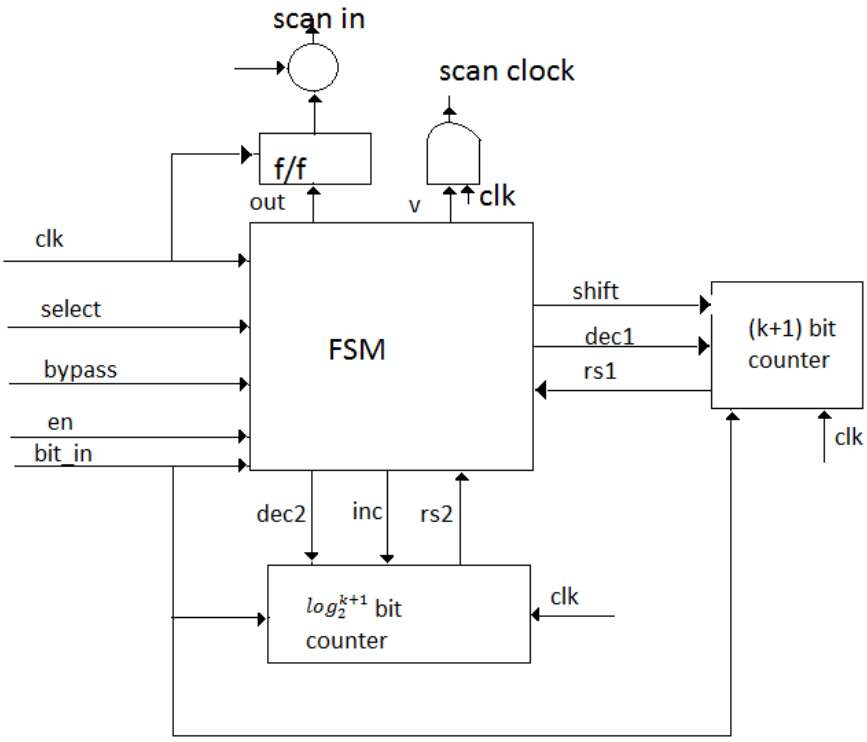

Figure 3.1: Decoder of ICC

The operation of the decoder can be understood as follows

Firstly the T f/f was reset and *v* was set to 0. Signal *en* if is low means that it is busy counting number of zeros and if it is high means that it has finished counting. So initially it has been set to 1. Now the FSM is ready to receive the data.

The function of the k+1 bit counter was to identify the prefix with the help of separator .this is valid for both Golomb code and AVR. The signals *en*, *shift* and *inc* were kept high until 0 or 1 was received.

The FSM outputs, 0s and 1s, decremented the k+1 bit counter and made the signal *dec1* high. It continued it until *rs1* is high. The *v* signal showed whether the output was high or not.

The tail part was again shifted to k+1 bit counter, but was under the control of $log_2(k+1)$ bit counter. It controlled the length of the control word.

Fig.2 shows FSM. It has 6 states. The S0 state is starting state as well as bypass state. For AVR code S0-S1-S3 states are used for decoding prefix starting with 0 and S0-S2-S4 states are used for decoding prefix starting with 1. For Golomb code the prefix decoding is done by S0-S3-S4. The tail decoding is done by states S4-S4.

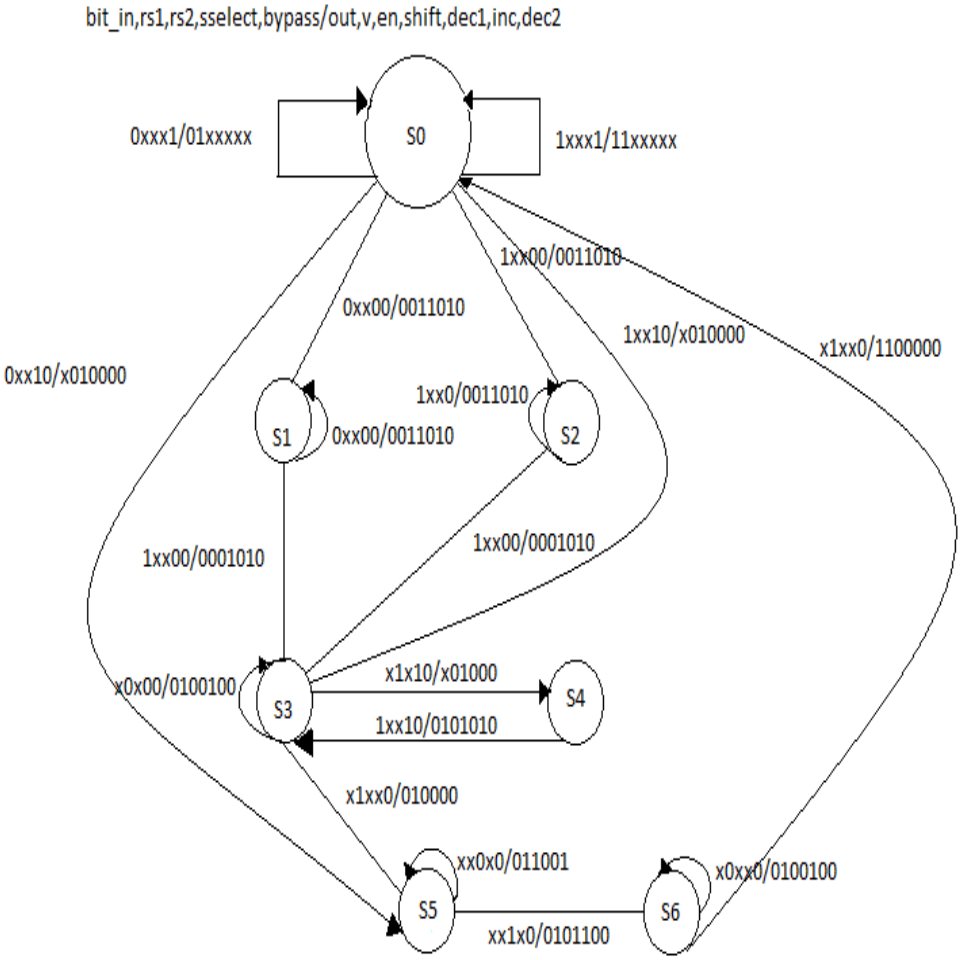

Figure 3.2 FSM of ICC

3.3.2 Decoder of ECC

The decoder contains two FSM, one counter, one synchronization circuit, a multiplexer and control signals. The decoder operates with two clocks, one external i.e. ATE_Clk and the internal clock SOC_Clk. FSM* is of 9C and FSM# is of ICC. FSM of 9C has been explained in [18]. FSM* receives the data from DATA_IN. This data is compressed data. Firstly the compressed data is passed to FSM* which will remove the 9C coding and then this data is passed to FSM# through Data_in_1to decode it further. The signals have the same value as discussed for the ICC decoder architecture.

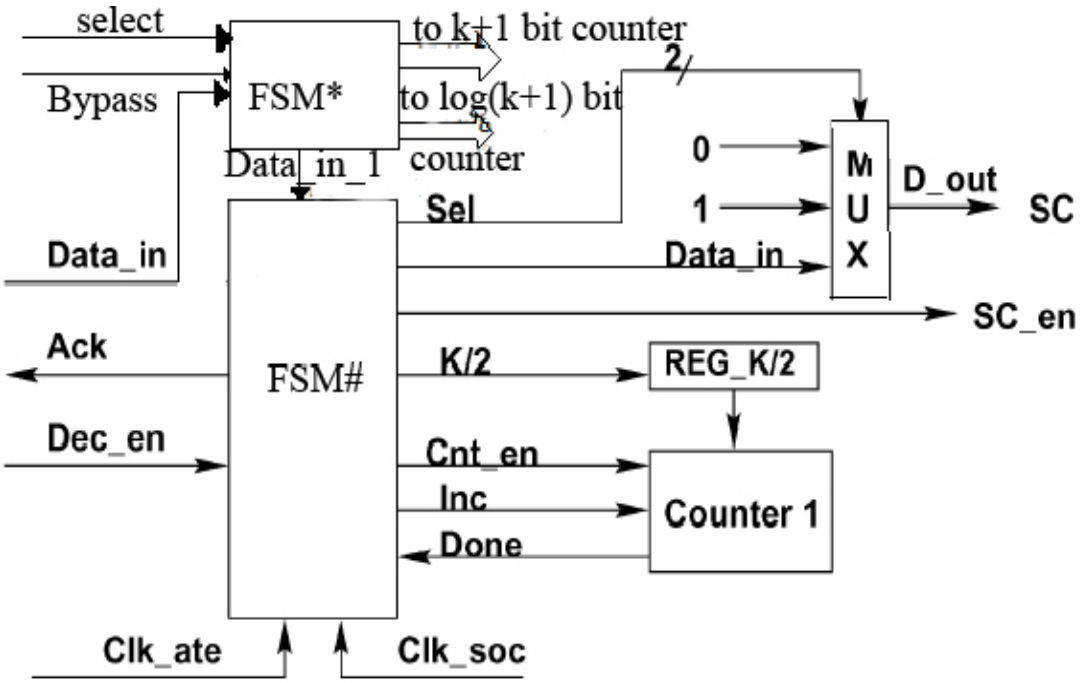

Figure 3.3: Decoder of ECC

3.4 Power Analysis

Power dissipation is an important problem for the circuit under test. A huge amount of power is dissipated when the circuit elements switched from logic 1 to 0 and vice versa. During testing this activity is dominantly controlled by test vectors [9].The power dissipation of the applied input and its response can be calculated using the weighted transitions in the vector. Scan vectors having the higher value of weighted transition metrics will dissipate more power in the circuit under test(CUT). The equation used for calculating weighted transitions is

$$Weighted\_transions=\Sigma(size\_of\_scan\_chain-position\_of\_transition) \quad (3)$$

Table 3.3: Power reduction using WTM

Scan vector with don't care terms	0000110xxxx1001xxxx0
Filling with WTM	00001100000100111110=20bit Golomb code length = 29 bits, AVR code length= 17bits, ICC code length= 15, WTM= 53
Filling with zeros	00001100000100100000 =20 bits, Golomb code length =21 bits, AVR code length= 15 bits, ICC code length= 13, WTM=58

Weighted transition metric has been discussed in details in [17] to estimate the power dissipation of the scan data. It told that power dissipation does not only depend on number of transitions but also on the relative positions of the vectors. If vector 'z' had more WTM than the vector 'y', then 'z' is said to dissipate more power than 'y'.

Let us say that a scan chain of length t is being dealt with and a scan vector $l_j = l_{j,1}l_{j,2}\ldots\ldots l_{j,t}$, and $l_{j,1}$ is scanned before l_{j2}. As shown in [15], we can calculate the value of the WTM for inputs as well as their responses by the following equation

$$WTM_j=\sum_{i=1}^{t-1}(t-i)(l_{j,i} \oplus l_{j,i+1}) \quad (3.1)$$

With the help of the above equation we can also calculate the peak(P_{peak}) and the average(P_{avg}) power as follows

$$P_{avg} = \frac{\sum_{j=1}^{n} \sum_{i=1}^{t-1}(t-i)(l_{j,i} \oplus l_{j,i+1})}{n} \quad (3.2)$$

$$P_{peak}=max_{j\in(1,2..n)}\left\{\sum_{i=1}^{t-1}(t-i)(l_{j,i} \oplus l_{j,i+1}\right\} \quad (3.3)$$

These equations were used to calculate the average and peak power for the four different test vector set for benchmark circuits.

There are different techniques that are useful to reduce the switching power dissipation in the scan chain. These can be minimum transmission fill[16],column bit stuffing[14], zero filling, random filling[16]. These techniques combined with the proposed ICC provided excellent results for power dissipation and test data volume reduction.

CHAPTER

4

EXPERIMENTAL RESULTS

We have used columnwise bit stuffing with difference vector (CBSTDIFF) [14], column bit stuffing(CBS) [14] , zero filling[16] and minimum transition filling [16] to fill the don't care terms of the test vectors generated by the Mintest ATPG program. The ISCAS'89 benchmark circuits used were s298, s400t, s1494, s1196t. The efficiency was used to compare the different codes with each other, where

$$Efficiency = \frac{T_D - T_E}{T_D} \times 100 \quad (4.1)$$

Where T_D is the original test data and T_E is the data achieved after compression. The coding and bit stuffing was done in the C programming language on the work station of corei3 and 4GB memory.

4.1 Compression and power results for circuit s298

Table 4.1 : Average and Peak Power for circuit s298

	Power average	Power peak
CBSTD	57.4419	96
CBSTDDIFF	20.9612	57
ZERO FILL	25.5039	57
MTFILL	14.47	53

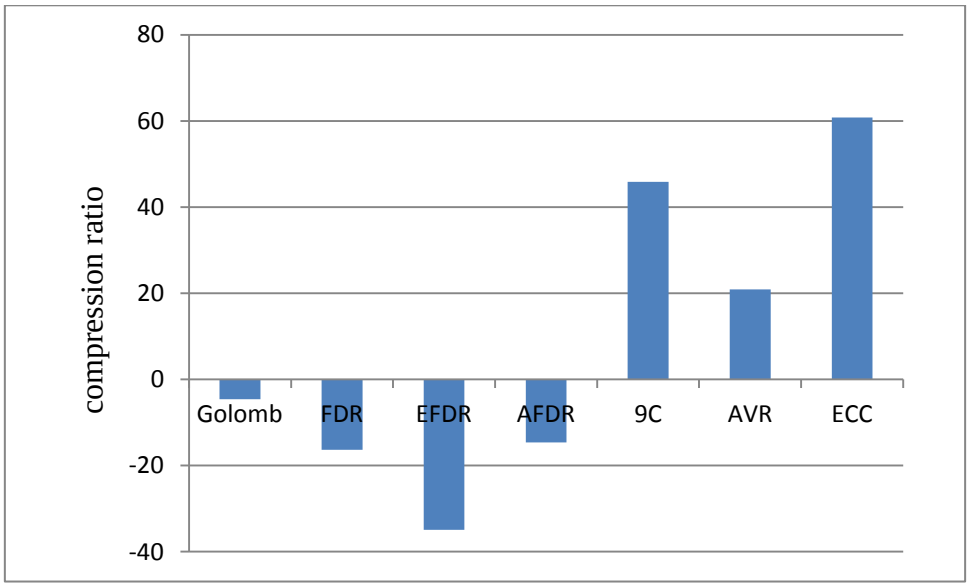

Figure 4.1: comparison for CBSTD for different codes for s298

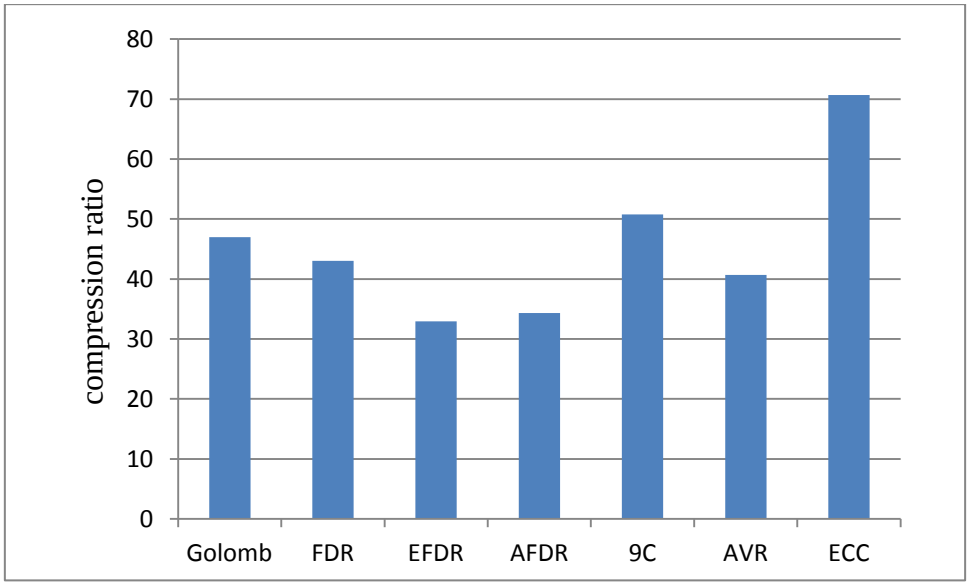

Figure 4.2: comparison for CBSTDDIFF for different codes for s298

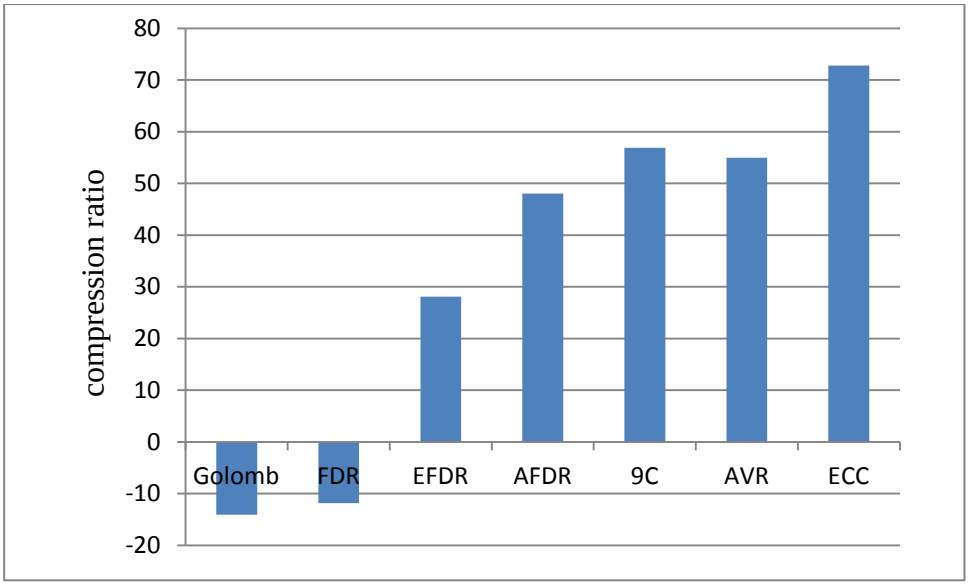

Figure 4.3: comparison for ZEROFILL for different codes for s298

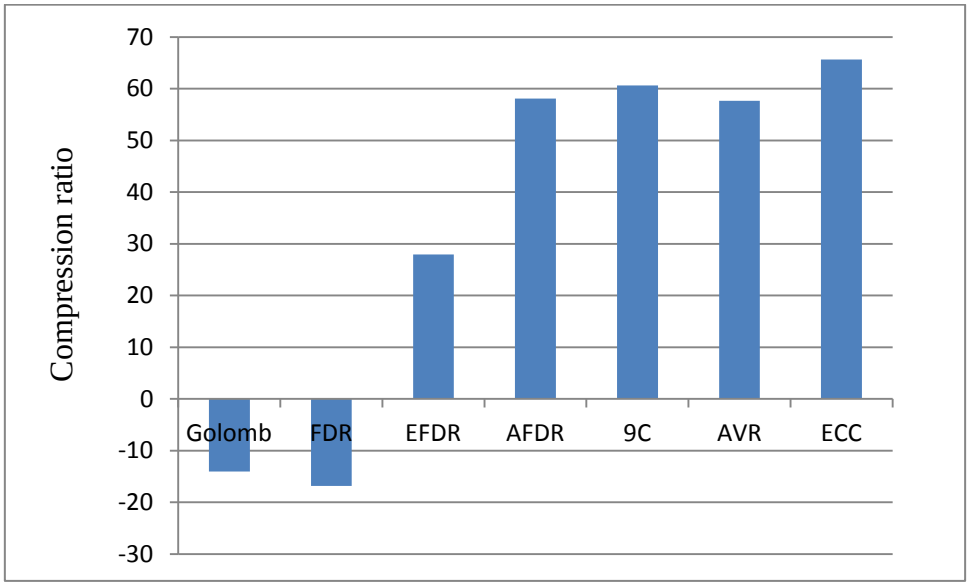

Figure 4.4: comparison for MTFILL for different codes for s298

4.2 Compression and power results for circuit s400

Table 4.2 : Average and Peak Power for circuit s400

	Power average	Power peak
CBSTD	134.3566	224
CBSTDDIFF	37.8005	111
ZERO FILL	46.8775	120
MTFILL	32.33	95

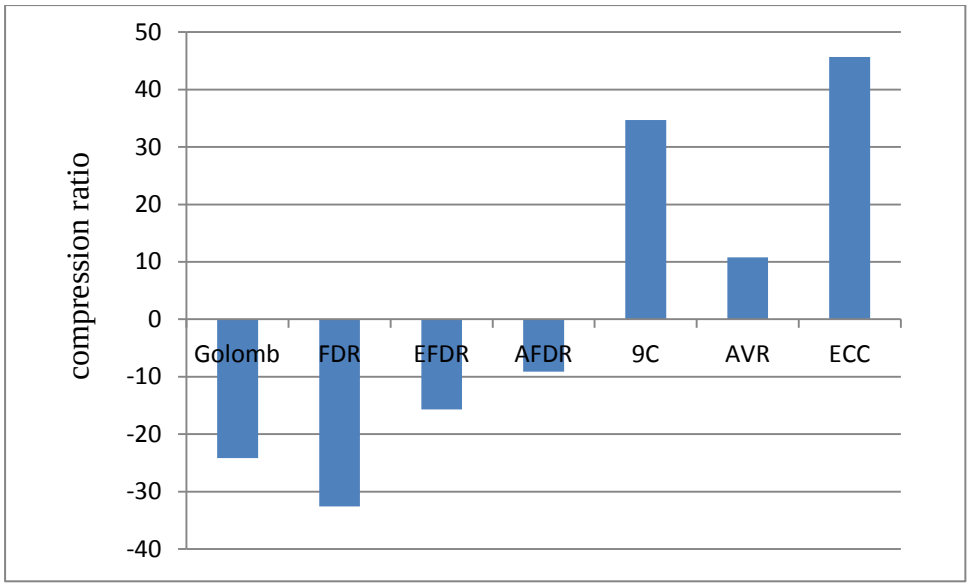

Figure 4.5: comparison for CBSTD for different codes for s400

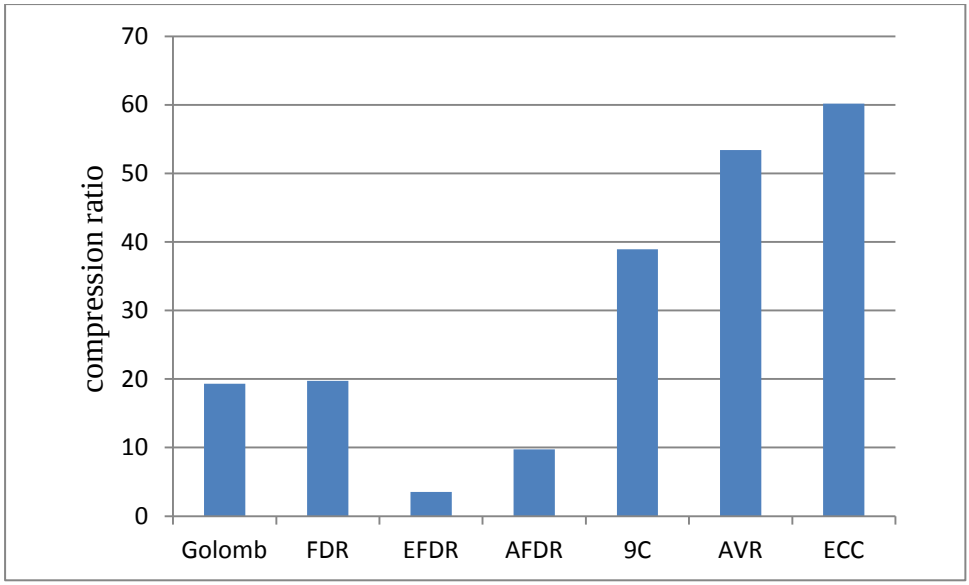

Figure 4.6: comparison for CBSTDDIFF for different codes for s400

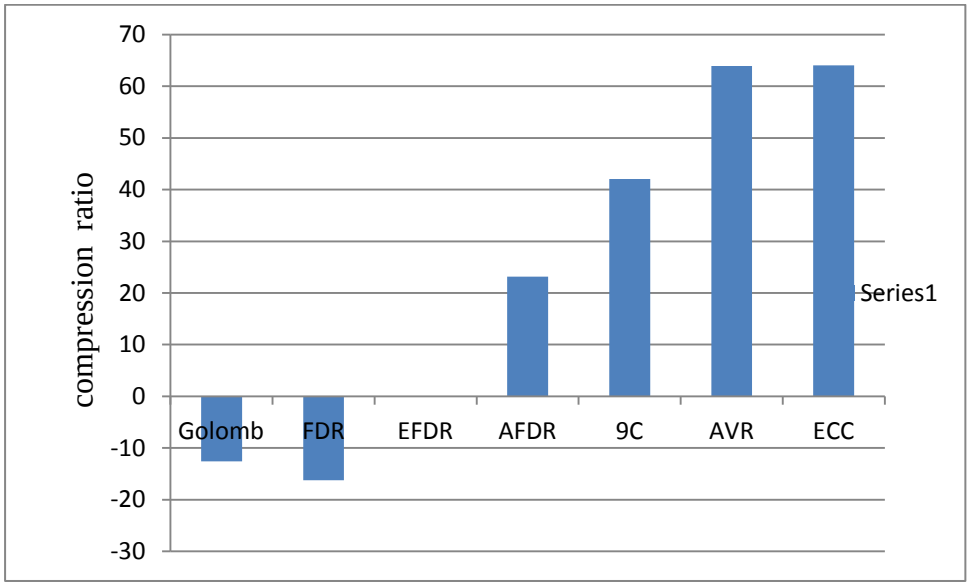

Figure 4.7: comparison for ZEROFILL for different codes for s400

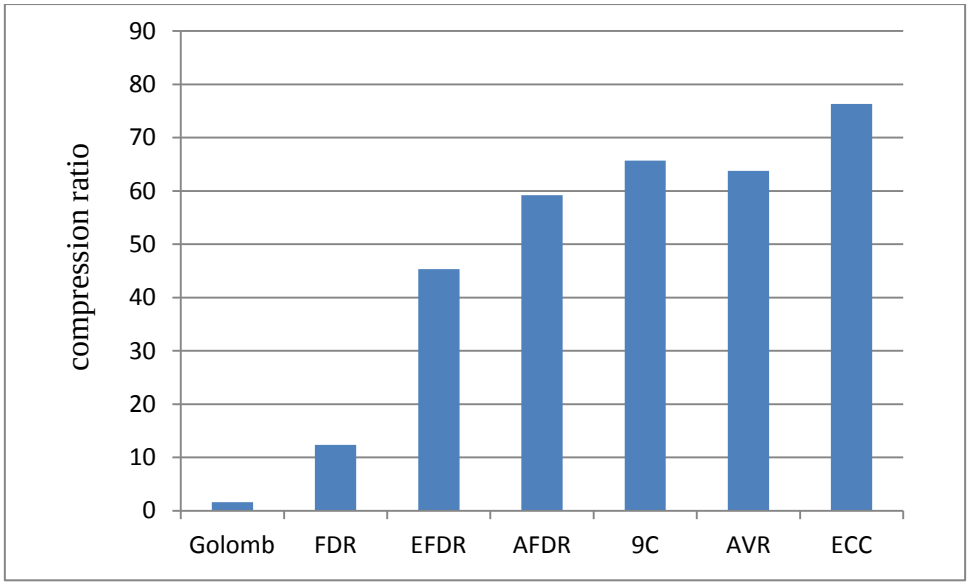

Figure 4.8: comparison for MTFILL for different codes for s400

4.3. Compression and power results for circuit s1494

Table 4.2 : Average and Peak Power for circuit s1494

	Power average	Power peak
CBSTD	42.940	87
CBSTDDIFF	20.09	61
ZERO FILL	28.16	66
MTFILL	19.45	60

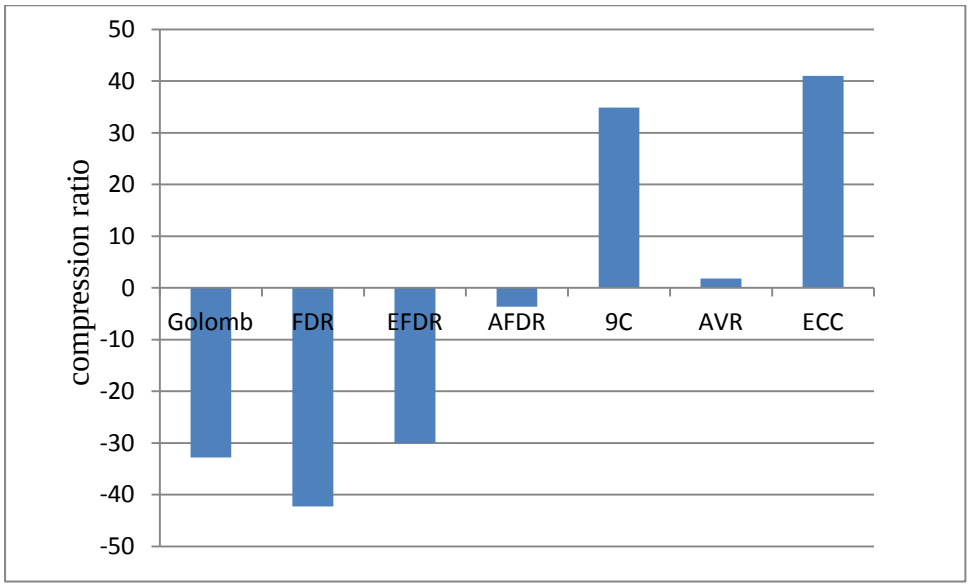

Figure 4.9: comparison for CBSTD for different codes for s1494

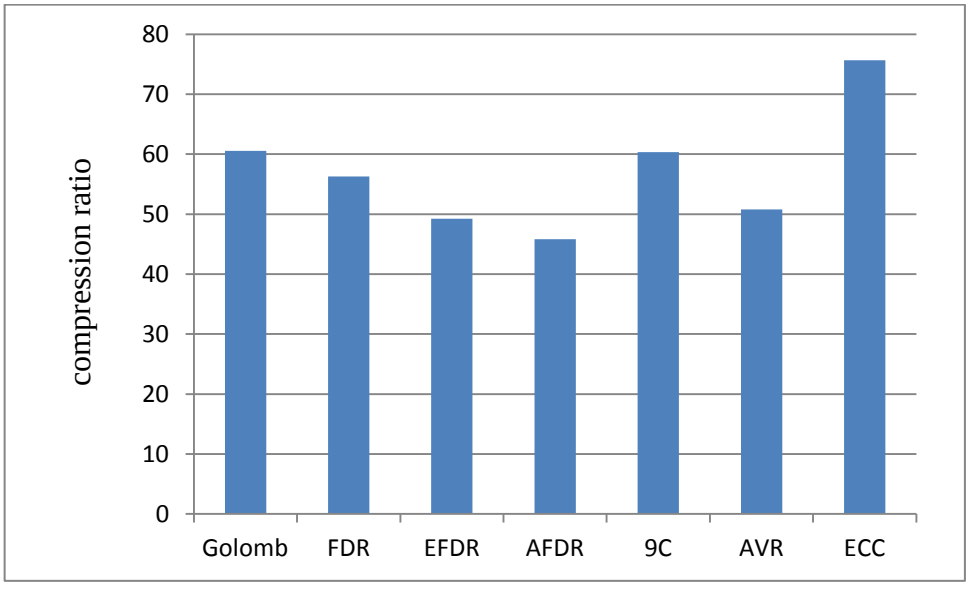

Figure 4.10: comparison for CBSTDIFF for different codes for s1494

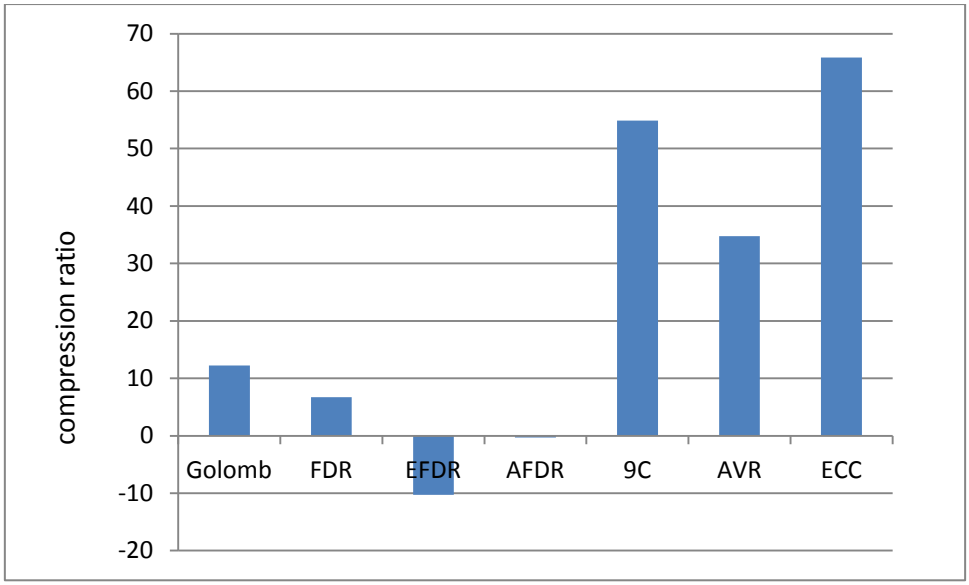

Figure 4.11: comparison for ZEROFILL for different codes for s1494

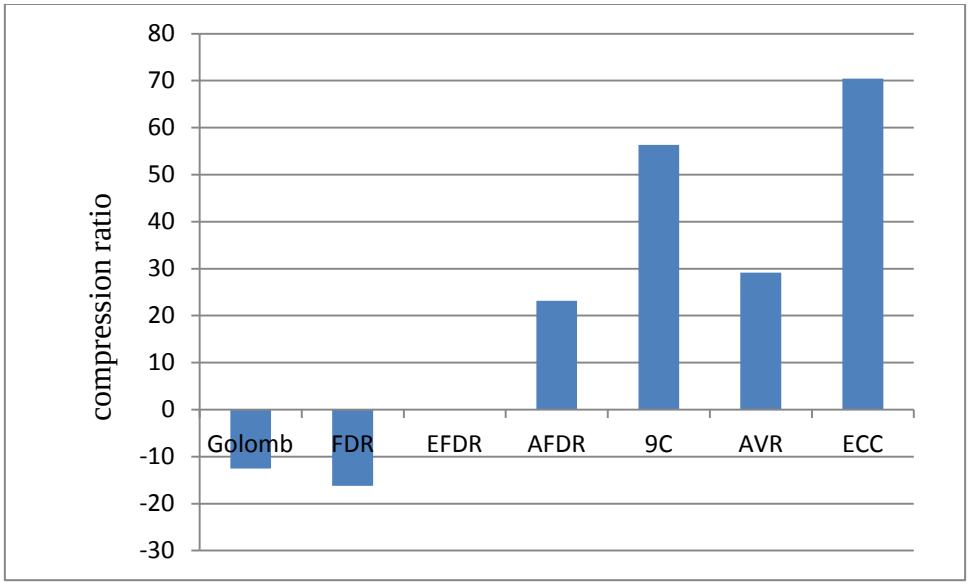

Figure 4.12: comparison for MTFILL for different codes for s1494

4.4. Compression and power results for circuit s1196t

Table 4.2 : Average and Peak Power for circuit s1196t

	Power average	Power peak
CBSTD	87.8049	150
CBSTDDIFF	44.26	118
ZERO FILL	70.12	131
MTFILL	48.61	130

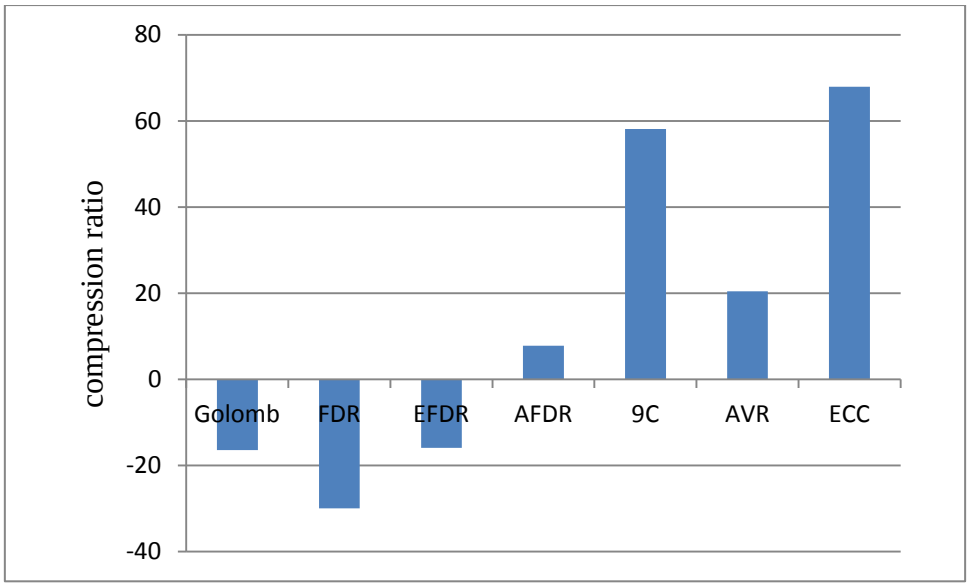

Figure 4.12: comparison for CBSTD for different codes for s1196t

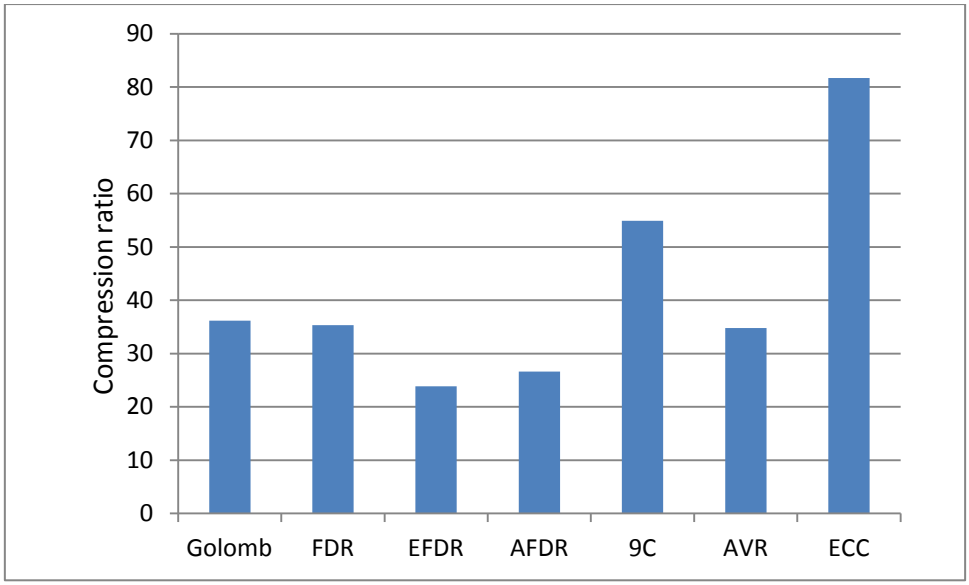

Figure 4.14: comparison for CBSTDIFF for different codes for s1196t

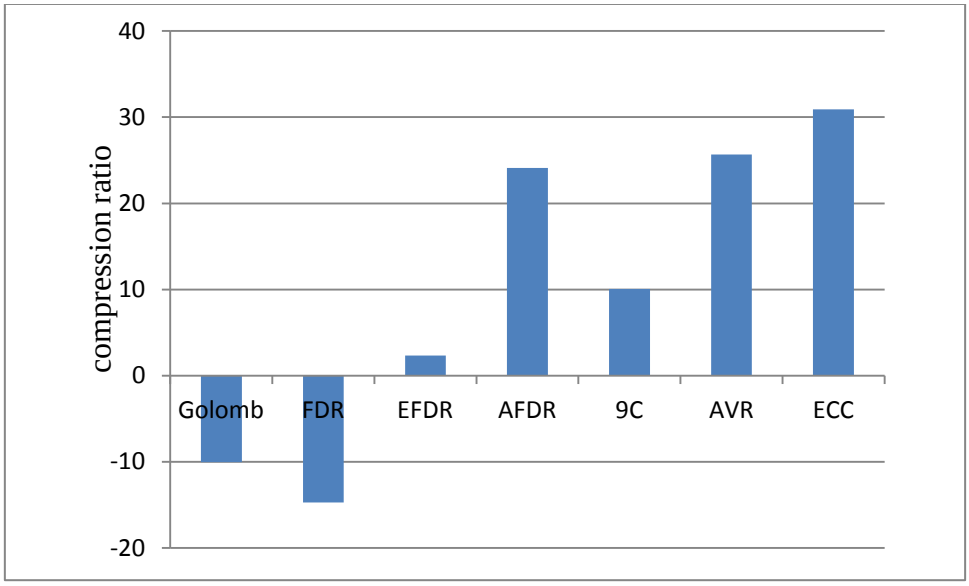

Figure 4.15: comparison for ZEROFILL for different codes for s1196t

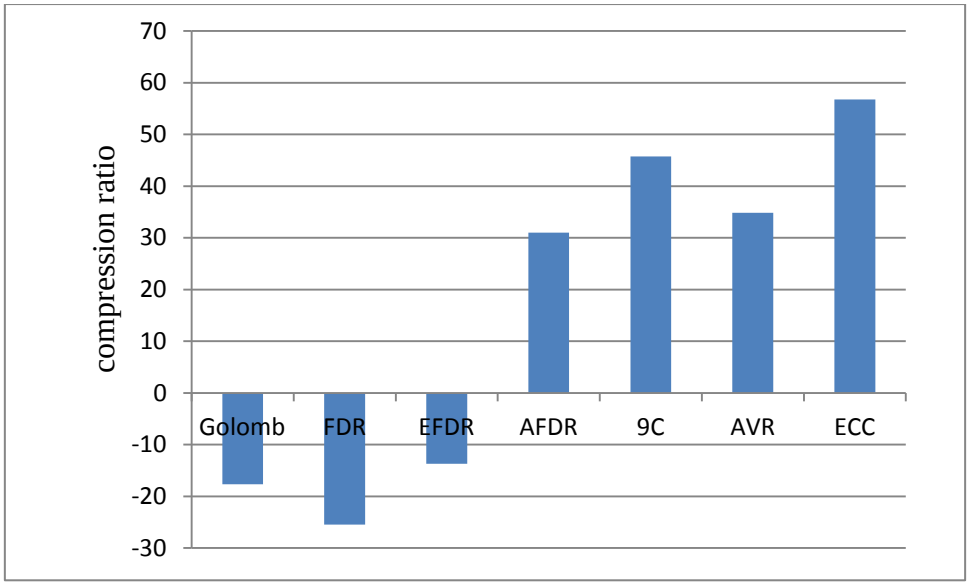

Figure 4.16: comparison for MTFILL for different codes for s1196t

CHAPTER

CONCLUSION AND FUTURE SCOPE

This thesis report presents the comparison of various runlength based compression schemes like Golomb code, FDR, AFDR, AVR and 9C. The coding for these schemes has been in C language. Also along with the comparison, a new code called Enhanced Compression Code has been developed. The Enhanced compression code or ECC is a multi stage compression scheme with two stages. At stage one it uses Integrated Compression Code, which is obtained after merging Golomb and AVR codes. Then at stage two it uses 9C coding scheme to further improve the compression ratio. 9C has been implied because it provides best compression and when it is implied at stage two then compression is improved by almost 50%.

Another advantage of ECC is that it involves the usage of Minimum Transition Fill for bit stuffing, which is best technique to reduce the average and peak power. Also decoder and FSM of ECC have been implemented by adding a few states in the FSM of AVR code.

Experimental results for ISCAS89 benchmark circuit show that enhanced compression scheme along with don’t care filling technique gives better results than any other compression code previously discussed.

The future work is to further integrate heterogeneous codes to obtain higher compression ratio and to study the timing specifications for the proposed circuit.

References

[1] Robert R. Schaller, "Moore's law: past, present and future", IEEE Spectrum, June 1997

[2] M.L. Bhushnell and V.D. Agarwall, Essentials of Electronic Testing For Digital Memory and Mixed signals VLSI circuits, Kluwer Academic Publishers, London, 2000.

[3] Pranab K. Nag, Anne Gattiker, Sichao Wei, "Modeling the Economics of Testing: A DFT : Prospective", in IEEE Design & Test of Computers,2002

[4] S. Hellebrand, H. Linag, and H.-J. Wunderlich, "A mixed-mode BIST scheme based on reseeding of folding counters," in Proc. Int. Test Conf.(ITC'00*),* pp. 778–784,2000

[5] M. Nourini, M. Tehranipoor, N. Ahmed, "Low –transition test pattern generation for BIST application", IEEE Trans. Comput. 57(3) pp 303-315,2008

[6] L. M. Denq, Y. T,Hsing, C. W. Wu, "Hybrid BIST scheme for multiple heterogeneous embedded memories", IEEE Des. Test Comput.26(2) pp. 64-72,2009

[7] C.W. Wu, J.F. Li and C.-T. Hung, "Core-Based System-on-Chip Testing: Challenges and Opportunities", in Journal of The Chinese Institute of Electrical Engineering, vol. 8, no. 4, pp. 335-353, 2001

[8] S. Lei, X. Hou. Z. Shao, F. Liang, "A class of SIC circuits: Theory and application in BIST design", IEEE Transaction Circuit system II 55(2) pp.161-165,2008

[9] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolici, ―Improving Compression Ratio, Area Overhead, and Test Application Time for System-on-a-Chip Test Data Compression/Decompression‖, in Proc. Design, Automation and Test in Europe Conference and Exhibition, pp. 604 – 611, March 2002.

[10] Lei Li, Krishnendu Chakarbarty, Seiji Kajihara and Shivakumar Swaminathan, "Efficient space/Time Compression to Reduce Test Data Volume and Test Time for IP cores" in 18th conference on VLSI design IEEE ,2005

[11] Czysz,d., Kassab, M., Murgalski G., "Low power scan shift nad capture in EDT environment", IEEE International Test Conference, 2008

[12] Usha S. Mehta, Kankar S. Dasgupta, Niranjan M. Devashrayee, "Hamming distance based reordering and columnwise bit stuffing with difference vector: A better scheme for test data compression with run length based codes", 23rd Intenational Conference on VLSI design,2010

[13] Ranganathan Sankaralingam, Rama RaoOruganti, and Nur A. Touba, "Static Compaction Techniques to Control Scan Vector Power Dissipation" VTS 2000

[14] W.D. Tseng, "Scan Chain ordering technique for switching activity reduction during scan test" published in Computer and Digital Techniques, IEEE proceedings, 2005

[15] A. Jas, J. and N. A Touba, "Test Vector Decompression via Cyclical Scan Chains and Its Application to Testing Core-Based Designs", in Test International Conference, pp.458-464, 1998

[16] M. Ishida, D. S. Ha, and T. Yamaguchi, "COMPACT: A hybrid method for compressing test data," in Proc. IEEE VLSI Test Symposium

[17] N. A. Tauba, "Survey of Test Vector Compression Techniques", IEEE transaction Design & Test of Computers, 2006

[18] Deepika Sharma, Debbrat Ghosh and Harpreet Vohra, "Test Data Volume Minimization using Double Hamming Distance Reordering with Mixed RL Huffman based compression scheme for System-on-chip,"2012 Nirma University International Conference On Engineering, Nuicone-2012, 06-08December, 2012

[19] A.Chandra and K.Chakarbarty," System-on-a-Chip Test-Data Compression and Decompression Architecture Based on Golomb Codes,"IEEE Trans. Computer-Aided Design, vol.20, pp. 355-368, Mar. 2001,

[20] A. Chandra and K. Chakarbarty, "Test Data Compression and Test Resource Partitioning for System-on-Chip Using Frequency-Directed Run-Length(FDR) Codes," IEEE Trans. Computers, vol. 52, pp.1076-1088, Aug 2003

[21] Aiman H. El-Maleh and Raslan H. Al-Abaji, "Extended Frequency-Directed Run-Length Code with Improved Application to System-on-a-Chip Test Data Compression" Int. Conf: on Electronics, Circuits and Systems, 2:449-452,Sep 2002

[22] A. Chandra and K. Chakarbarty, "Reduction of SOC Test Data Volume, Scan Power and Testing Time Using Alternating Runlength Codes", DAC'02: Proceeding of the 39th conference on Design automation, June 2002

[23] Lung-Jen Lee, Wang-Dauh Tseng, Rung-Bin Lin, and Cheng-Ho Chang, "2^nPattern Run-Length for Test Data Compression", IEEE Transaction on Computer –Aided Design of Integration Circuits and System vol. 31, no. 4, April 2012

[24] Abhijit Jas, Jayabrata Ghosh-Dastidar, Mom-Eng Ng, and Nur A. Touba, "An Efficient Test Vector Compression Scheme Using Selective Huffman Coding" IEEE Transactions on Computer-Aided Design of Integrated Circuits And Systems, Vol. 22, No. 6, June 2003

[25] Xrysovalantis Kavousianos, Emmanouil Kalligeros, and Dimitris Nikolos,''Optimal Selective Huffman Coding for Test-Data Compression" IEEE Transactions on Computers, Vol. 56, No. 8, August 2007

[26] Mohammad Tehranipour, Mehrdad Nourani, Krishnendu Chakrabarty, "Nine-Coded Compression Technique with Application to Reduced Pin-Count Testing and Flexible On-Chip Decompression", Proceedings of the Design, Automation and Test in Europe Conference 04 2004

[27] Haiying Yuan, Jiaping Mei, Hongying Song , Kun Guo, "Test data compression for System-on-a-Chip using Count Compatible Pattern Run length coding" J Electron Test 30:237–242, 2014

[28] Bo Ye, Qian Zhao, Duo Zhou, Xiaohua Wang, Min Luo, "Test data compression using alternate variable run length code", Integration, the VLSI journal 44, pp. 103-110, 2011

[29] S. Sivananthan, M. Padmavahy, P.S. Mallik, J. Raja Paul Perinbam,

"Enhancement of Test Data Compression with Multistage Encoding" published in Integration, the VLSI journal, vol. 47, pp. 499-509,2014

[30] S. Sivananthan, M. Padmavahy, P.S. Mallik, J. Raja Paul Perinbam, "Reduction of Test Power and Test Data Volume by Power Aware Compression Scheme", International Conference on Advances in Computing and Communications, 2012

List of Publications

[1]	Shruti, Vohra Harpreet, "Enhanced Compression Code for SOC Test Data Volume Reduction", published in International Journal of Computational Engineering and Management, vol.18, issue3, May 2015