World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:9, No:3, 2015

Test Data Compression Using a Hybrid of Bitmask Dictionary and 2^n Pattern Runlength Coding Methods

C. Kalamani, K. Paramasivam

***Abstract*—**In VLSI, testing plays an important role. Major problem in testing are test data volume and test power. The important solution to reduce test data volume and test time is test data compression. The Proposed technique combines the bit mask-dictionary and 2^n pattern run length-coding method and provides a substantial improvement in the compression efficiency without introducing any additional decompression penalty. This method has been implemented using Mat lab and HDL Language to reduce test data volume and memory requirements. This method is applied on various benchmark test sets and compared the results with other existing methods. The proposed technique can achieve a compression ratio up to 86%.

***Keywords*—**Bit Mask dictionary, 2^n pattern run length code, system-on-chip, SOC, test data compression.

I. Introduction

A circuit or system consumes more power in test mode than in normal mode, this leads to severe hazard in circuit reliability, instant circuit damage, increases cost, increases verification problems and decrease in overall yield. Some of Test challenges are Escalating transistor counts, increasing chip's complexity, while maintaining its size. Testing is one of the most expensive and problematic aspects in a circuit design cycle. Test efficiency correlates with toggle rate. In test mode, switching activity of all nodes is often several times higher than normal operation. Often parallel testing is used in system-on-chip (SOC) to reduce test application time, which might result in excessive energy consumption and power dissipation. DFT circuit is designed to reduce test complexity, which is often idle during normal operation, but might be intensively used in test mode. Correlation between successive functional inputs may be significant; however, for test patterns it is generally very low. Switching activity during test leads to increase in both hardware costs and time. As a result of new fabrication technologies and design complexities, standard stuck-at scan tests are no longer sufficient. Each new fabrication process technology increases the number of tests as well as corresponding data volume. Higher circuit densities in system-on-chip (SOC) designs have led to drastic increase in test data volume. Larger test data size demands higher memory requirements as well as an increase in testing time. Test data compression solved this problem by reducing the test data volume without affecting the overall system performance. 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; it also compact 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. It is also easier to adopt in industry because its compatibility with the conventional design rules and test generation flows for scan testing. Test data compression provides two benefits. First, it reduces the memory requirements on the tester, which can extend the life of older testers that have limited memory. Secondly, it applies even for testers with plenty of memory it can reduce the test time for a given test data bandwidth because less test data has to be transferred between the tester and the chip.

C. Kalamani and K. Paramasivam are with the ECE Department, Dr. Mahalingam College of Engineering and Technology, Karpagam College of Engineering, Coimbatore, Tamilnadu, India (e-mail:kalamec18@gmail.com, kp_sivam@yahoo.com).

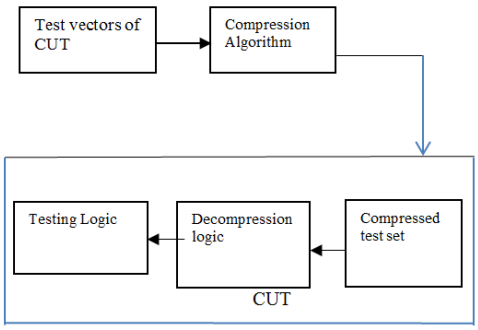

Fig. 1 A process of test data compression and decompression for cut

Test vectors are highly compressible because typically only 1% to 5% of their bits are specified (care) bits. To solve this problem, various test data compression techniques have been proposed. There are three types of test compression techniques such as, linear-decompression scheme, broadcast-scan scheme and coding scheme [1]-[3]. Coding scheme is widely used one and it doesn't require any structural information of IP cores. In this scheme, a given test input is compressed by data compression techniques and stored in VLSI tester memory. While a CUT on a chip is tested, the compressed test input set is transported to a decompression on the chip and then it is restored. The compressed test set can achieve the reduction of the time for test transportation, not just the size of the test storage device. The overview of traditional test data compression/decompression process method is shown in Fig. 1. The original test vector is compressed and stored in

the memory. Thus, the memory size is significantly reduced. An on-chip decoder decodes the compressed test data from the memory and delivers the original uncompressed set of test vectors.

Compression ratio is defined as the ratio between the original test vector Size and compressed test vector size:

$$\text{Compression Ratio} = \frac{\text{compressed test vector Size}}{\text{Original test vector Size}}$$

The test data compression ratio can serve as a measure of the complexity of a test data set.

II. Existing Method

In test data compression, lossless data compression is used to compress the test data. In lossless data compression, the integrity of the data is preserved. The original data and the data after compression and decompression are exactly the same because, in these methods, the compression and decompression algorithms are exact inverses of each other. No part of the data is lost in the process. Redundant data is removed in compression and added during decompression. Lossless compression methods are normally used when it cannot afford to lose any data. Various compression techniques have been proposed over the years to reduce the test data volume and test time. Many coding schemes have been proposed for code-based scheme. Some of Huffman coding based methods were proposed, however, this method suffered from high area overhead [4]. To cure this problem, the selective Huffman encoding technique was proposed [5]. It only encodes the symbols with higher occurrence frequencies. The size of the decompression circuit can be reduced substantially. Hence, the optimal selective Huffman coding technique was proposed [6]. It reduces the test data. A 9C technique uses exactly nine code words aiming at pre-computed data of intellectual property cores in SOC. It is flexible in utilizing both fixed-length and variable-length blocks [7]. In addition, run-length methods can make a good trade-off between test data compression ratio and area overhead was improved using variable-to-variable encoding techniques, Golomb coding and FDR coding consist of a prefix and a tail with same size. It requires complicated decoder and inefficient for long run of 1's [8], [9]. An enhanced coding scheme named Extended Frequency-Directed Run-length (EFDR) was proposed [10]. This method took advantage of both runs of 0's and runs of 1's and outperformed the other coding techniques that are based on only runs of 0's. RL-HC combines the run length based Huffman coding for scan testing to reduce the test data volume [11]. PRL is also an efficient approach. The compression is data-independent and the program for decompression is very small and simple, thereby allowing fast and high throughput to minimize test time [12]. Some other methods were proposed, Tunstall coding [13]. LZW coding [14], 9-coded technique [15], heterogeneous compression technique [16], multilevel Huffman coding [17] have shown test data reduction using an on-chip pattern decompression scheme. These methods produce a moderate compression. In order to increases Compression ratio further this paper propose the combination of the bitmask-dictionary [18], [19] and 2^n PRL [20]. The rest of the paper is organized as follows; section III describes the background of this method. Section IV describes the proposed methods. Section V describes the results and comparison with existing. Section VI concludes the paper.

III Background

A. Bitmask Based Dictionary Compression

The bitmask based dictionary compression technique is proposed [18], [19]. It is a promising technique that illustrates better compression ratio over dictionary based compression technique. This technique uses extra bits to record the differences in unmatched words from the dictionary entries and encode them as usual dictionary indices. This technique uses limited dictionary to match many words with small bit changes. A bitmask makes use of the fact that binary numbers are made up of 1's and 0's, each digit in a binary number being equivalent to one bit. It easily checks the state of individual bits regardless of the other bits. The bit masking approach tries to incorporate maximum bit changes using mask patterns without adding significant cost (extra bits) such that the compression ratio is improved. Bit masking compression technique also ensures that the decompression efficiency is improved. Fig. 2 illustrates the encoding scheme used by the bit masking compression technique. This encoding format can store information for each pattern; it stores the mask type, the location of mask, and the mask pattern. The encoding scheme can be optimized, only the types and sizes of the bitmask combinations are determined.

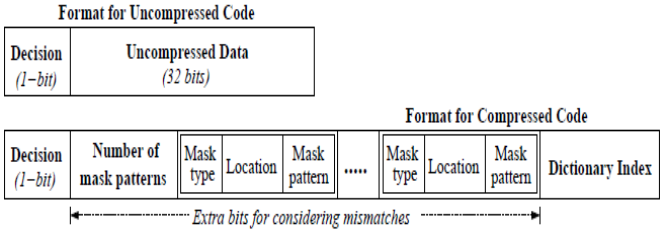

Fig. 2 Encoding format for bit masking compression technique

The bitmask can be applied on different places on a vector and the number of bits required for indicating the position varies depending on the bitmask type. For instance, if consider a 32-bit vector, an 8-bit mask is applied on only byte boundaries require 2-bits, since it can be applied on four locations. It will require 5 bits to indicate any starting position on a 32-bit vector. Bitmask-based compression is an enhancement on the dictionary-based compression scheme, which helps to get more matching patterns. In dictionary-based compression, each vector is compressed only if it completely matches with a dictionary entry. There are three major challenges in bitmask-based test data compression.

1) *Dictionary Selection: A profitable dictionary is to be selected which takes into account the bit savings due to frequency matching as well as bitmasks.*
2) *Bitmask Selection: Appropriate number and type of bitmasks are to be selected for compression.*
3) *Don't Care Resolution: It is necessary to selectively replace each don't care with "0" or "1".*

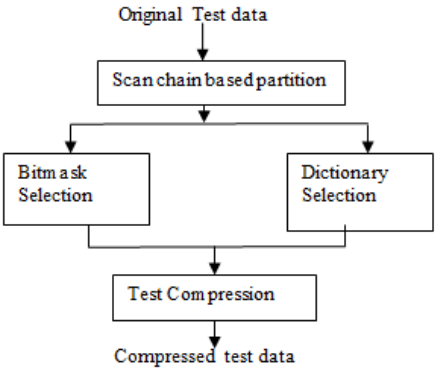

Fig. 3 Bitmask based code compression technique

Fig. 3 demonstrates the bitmask based code compression techniques. The first step divides the uncompressed test data set into equal length slices for compression. After getting the input test data, the next task would be to divide them into scan chains of predetermined length. Let us assume that the test data consists of n test patterns. Divide the scan elements into m scan chains in the best-balanced manner possible. This results in each vector being divided into m sub-vectors, each of length L. Dissimilarity in the lengths of the sub-vectors are resolved by padding don't cares at the end of the shorter sub-vectors. Thus, all the sub-vectors are of equal length. The m bit data which is present at the same position of each sub-vector constitutes an m bit slice. If there are n vectors at the beginning, obtain a total of n ×L m bit slices, which is the uncompressed data set that needs to be compressed. Two types of bitmask are present. A fixed bitmask is one which can be applied to fixed locations. However, sliding bitmasks can be applied anywhere in the test vector. Since the fixed bitmasks can be applied only to fixed locations, the number of positions where they can be applied is significantly less compared to sliding bitmasks. Hence, the number of bits needed to represent them is less than sliding bitmasks. The number of bitmasks selected, which depends on both the test vector length and the dictionary. The dictionary selection algorithm is a critical part in bitmask- based test data compression which is shown in Fig. 4.

Bits in two vectors are said to be compatible if they meet any one of the following two requirements: 1) for all positions, the corresponding characters in the two vectors are either equal or one of them is a don't care; or 2) two vectors can be matched by predetermined profitable bitmasks. Each edge contains weight information. The weight is determined based on the number of bits that can be saved by using that edge. Three dictionary selection techniques are used. They are:

1. Two-step method (TSM)
2. Method using compatible edges (MCE) without edge weights
3. MCE with edge weights (MEW).

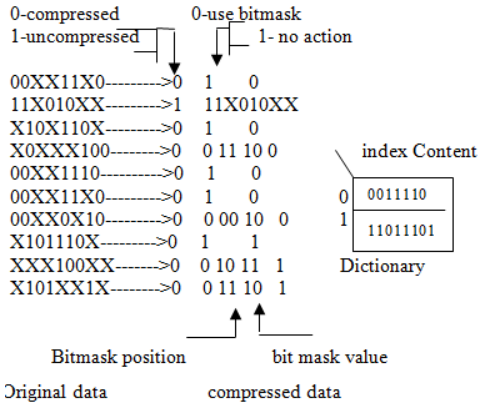

Fig. 4 Compression using bitmask and dictionary

Each of these techniques uses a variant of well-known clique partitioning algorithm. In TSM, consider only edges that are formed by direct matching. The graph will not have any edges corresponding to bitmask-based matching. Then a clique partitioning algorithm is performed on the graph. This is a heuristics-based procedure that selects the node with the largest connectivity and is entered as the first entry to a clique. Now, the nodes connected with it are analyzed, and the node having the largest connectivity among these (and not in the entire graph) is selected. This process is repeated until no node remains to be selected. The entries of the clique are deleted from the graph. The algorithm is repeated until the graph becomes empty. The clique partitioning algorithm is used in MCE and MEW as well. The number of cliques selected may be greater than the predefined number of entries or vice versa. Method Using Compatible Edges (MCE) Without Edge Weights*:* In MCE, weight of all the edges (direct or bitmask-based match) is considered equal. MCE with Edge Weights (MEW) is same as MCE except that it considers edge weights. As indicated earlier, the edge weight is determined based on the number of bits saved if that edge is used for direct or bitmask-based matching.

B. 2^n Pattern Run Length Coding

2^n PRL (pattern run length) [20], relies on encoding compatible patterns within the test data. Two patterns of the same length are compatible (inversely compatible) if every bit pair at the same position has the same (opposite) value after properly filling the values of don't-cares. 2^n PRL first partitions test data into fixed-length (L-bit) segments where L has a power of 2. Compression is then conducted on $2^{|n|}$ compatible patterns where *n* is a signed integer. Depending on whether encoding is performed within a segment or across several adjacent segments, test data can be encoded in either of the following two ways which are shown in Fig. 5.

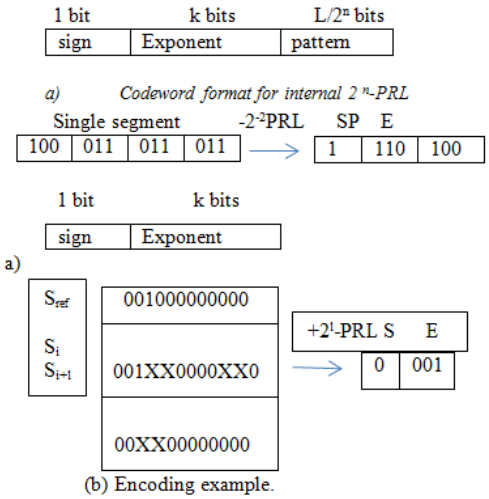

Fig. 5 Code word format for 2^N PRL and its example

Internal 2^n -PRL ($n < 0$) compresses $2^{|n|}$ runs of compatible (inversely compatible) sub segments of single segments and code word for this method consists of three components as sign(S)(S=0 means positive and S=1 means negative which is inversely compatible) and exponent (E) of K bit and pattern(p), if don't care bits are properly filled to make compatible with corresponding sub segments and external 2^n -PRL ($n \geq 0$), compresses 2^n consecutive compatible segments into shorter one. The following steps are used to perform further compression, Bitmask compressed data taken as an input to the 2^n pattern run length coding. The input data is split into fixed length (L) segments and set the exponent. The data can be encoded either of two types and exception type (non-compressible segment). The encoding process updates the reference segment during three conditions. First, when a segment is encoded by internal 2^n -PRL the corresponding segment becomes reference segment. Secondly, when several segments, inversely compatible with the reference segments are encoded by external 2^n -PRL. The inversely compatible becomes the reference segments. Third condition occurs when segment is encoded by exception type. This becomes the reference segment.

IV. Proposed

In the proposed method, combines the bitmask and dictionary method and 2^n PRL (pattern run length) methods. In the first stage of compression is performed using bitmask and dictionary method using two-step methods [19]. Second stage of compression is performed using 2^n PRL (pattern run length) methods. The proposed method shows better compression than the above methods.

A. Compression Algorithm

The compression algorithm is developed in two stages. The first stage of implementation, concept of bitmask-dictionary method is used and the second stage, concept of 2^n PRL coding is used. Test vector is given at the first stage which is cascaded with the second stage and finally the compressed test data is obtained:

Algorithm is as follows:

Steps:

1. Divide the uncompressed test data based on number of scan chains;
2. Select Bitmasks and Dictionary
3. Perform Test compression using selected dictionary and bitmask
4. Store the compressed data and dictionary.
5. The compressed data is split into compatible or incompatible if two pattern of the same length with bit pair at the same position has same value or opposite value after careful filling of don't cares bits.
6. Partitions test data into fixed length (L) segments where L has a power of 2 and set exponent value.
7. Check whether the type of encoding are internal 2^n PRL ($n < 0$) or external 2^n PRL ($n > 0$) or exception and based on the result, set the reference segment, encode the data.
8. Find the length of the bit pattern and calculate the compression efficiency.

B. Decompression Mechanism

The decompression architecture is shown in Fig. 6. It consists of Finite State Machine (FSM) for identifying the code words, Control and Generation Unit (CGU) is used for data transmission control and generating test pattern. Decompression logic generates the bitmask data which is XOR with the dictionary in parallel. So, it reduces the additional penalty. The decompression is worked based on the algorithm given below.

Algorithm Steps:

1. Compressed data is partitioned into 8 bit segment
2. Find the encoding type of segment whether compatible, uncompatible or exception
3. Find the code word based on encoding type
4. Repeat the above steps for all segments.
5. The decompression output is sent to bitmask-dictionary decompression stage.
6. Let p be the first bit of the compressed string
7. If p= =1 the remaining bits are the uncompressed string go to step14
8. Let q be the second bit of the compressed string
9. If q==1 the remaining bits correspond to dictionary index, it is a direct match, read the dictionary entry go to step 14.
10. Read respective bits for mask values and positions.
11. Compose final bitmask using the data from step 10.
12. Read the dictionary entry based on dictionary index.
13. Find XOR of dictionary entry with final bitmask to generate the uncompressed string.
14. Send uncompressed string to DUT.

V. Implementation Result and Discussion

Compression algorithm was implemented using MATLAB-13 and VHDL and simulated and experimented on various ISCAS benchmark circuits. Compression Ratio is used to identify the efficiency of the compression technique. The proposed method is implemented in two stages, in the first stage Bitmask and dictionary method [19] and in the second stage is 2^n PRL compression technique are used [20]. Implementation results of two stages are shown in Table I. It shows that 2^n PRL performs well compared with proposed

methods for small circuits and proposed methods outperforms for large circuits. Table II and Fig. 7 shows the comparison of proposed method with existing methods that the proposed Combination of bitmask-dictionary and 2^n PRL provides better compression efficiency than 2^n PRL but 1% and 15 % higher than [17] for s9234 and s38417 test sets respectively. Bit mask dictionary technique shows better compression for s15850 test sets than proposed methods because of more number of *don't care* bits. This technique provides good compression efficiency when compared with the existing methods which is shown in Table II. The proposed method is implemented using VHDL and simulated using Xilinx 9.2 –model-sim simulator which is shown in Table III. The MATLAB implementation of compression is simple and easy, it outperforms than VHDL Implementation. The decompression was implemented using VHDL is simple and easy.

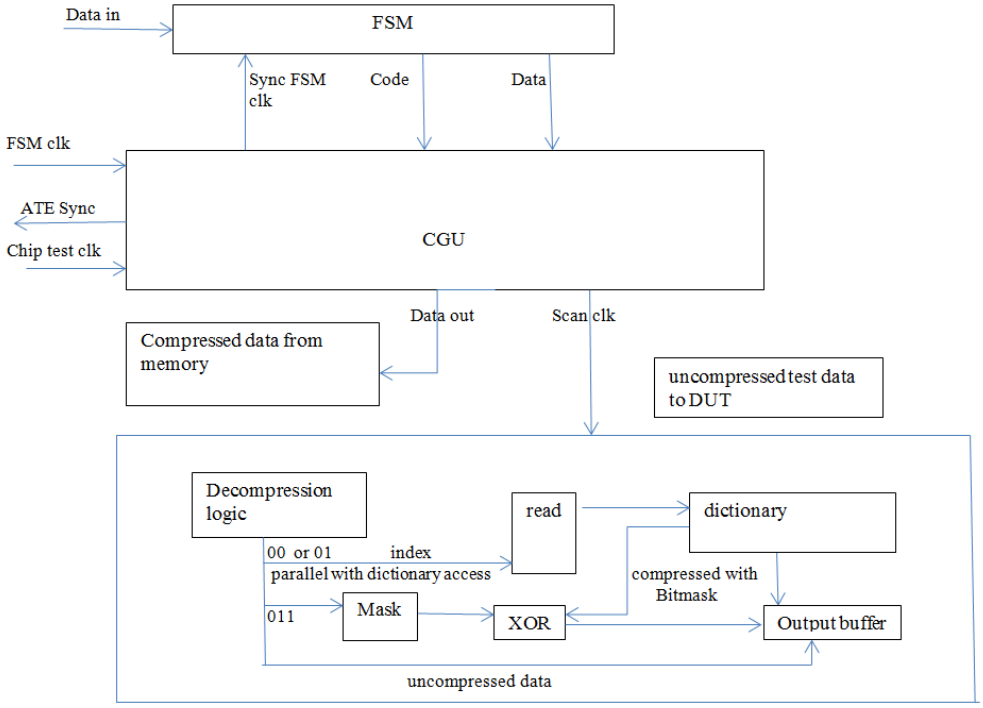

Fig. 6 Decompression architecture

VI. Conclusion

Testing power and large data volume are the most important issues in testing a VLSI circuit. Code compression technique offers an efficient solution for this issue. This Paper proposed an efficient code compression technique by combining bitmask-dictionary and 2^n pattern run length coding and implemented to reduce the large test data volume. The proposed algorithm applied on various benchmarks circuits and compared the results with existing test compression technique which shows that the proposed method presents a better compression ratio.

TABLE I
Implementation Results of Proposed Method and 2N PRL and Bitmask-Dictionary

Circuits	2^n PRL [20]	Bitmask-Dictionary [19]	Proposed Method
C432	87.1	78.1	85.67
C499	86.19	77.47	85.08
C880	87.5	78.22	85.92
S5378	54.94	77.10	85.27
S9234	57.72	76.90	88.05
S38417	72.44	76.93	86.52
S15850	74.29	78.30	85.48

TABLE II
Performance of Different Compression Schemes Using Mintest Test Data

Compression ratio (%)							
Existing methods							
Circuits	SHC [5]	VIHC [7]	RL-HC [11]	MD-PRL [21]	2^n PRL [20]	Bitmask & dictionary [19]	Proposed Method
S5378	55.1	51.78	53.75	54.6	54.2	-	85.27
S9234	54.2	47.25	47.59	53.2	57.7	87	88.05
S38417	59	53.36	64.2	55.4	58.3	71	86.52
S15850	66	67.94	67.34	69.9	74.3	89	85.48

TABLE III
VHDL Implementation Results of Proposed Method

Circuits	Original size	Compressed bits with Dictionary		Compression Ratio (%)	
		(8-bit)	(16 bit)	(8bit)	(16 bit)
S5378	20758	3537	3259	83	84.3
S9234	25935	4408	3648	83.01	85.9
S13207	163100	24465	20174	85	87.6
S15850	57434	9189	8092	84.01	85.9
S35932	21156	2961	2618	86.01	87.6

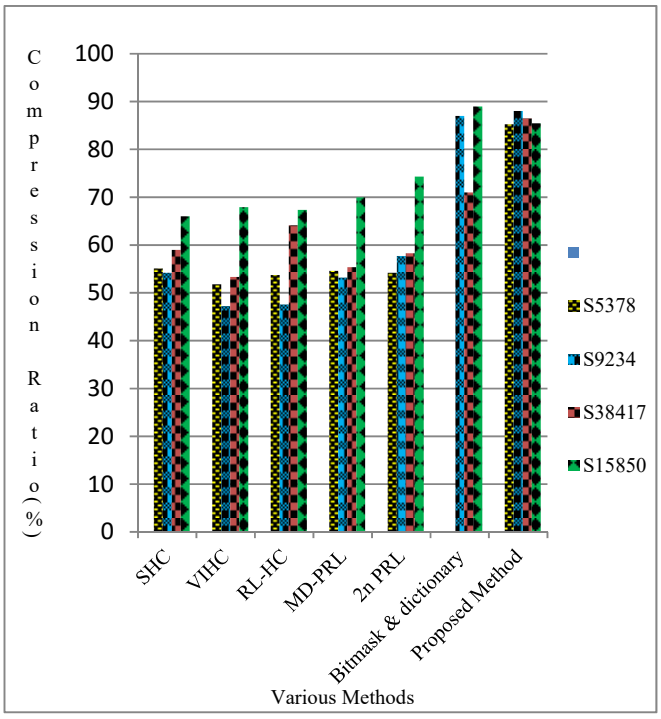

Fig. 7 Comparison of compression ratio of proposed with existing methods

REFERENCES

[1] Laung-Terng Wang, Cheng-Wen Wu, and Xiaoqing Wen, "*VLSI Test Principles and Architectures: Design for Testability",* Morgan Kaufmann; Academic Press; Newnes (781)- 2006-313-4732.

[2] N.A. Touba, "Survey of Test Vector Compression Techniques", *IEEE Design & Test Magazine,* Vol. 23, Issue 4, Jul. 2006, pp-294-303.

[3] Kalamani. C and Dr. K. Paramasivam, "Survey of Low Power Testing Using Compression Techniques", *International Journal of Electronics & Communication Technology ,* Vol.4, Issue 4, Oct-Dec 2013,pp. 13-18.

[4] V. Iyengar, K. Chakrabarty, and B. T. Murray, "Huffman encoding of test sets for sequential circuits", *IEEE Transactions on Instrumentation and Measurement*, vol. 47, February 1998, pp. 21-25.

[5] A. Jas, and N. A. Touba, et al, "An efficient Test vector compression scheme using selective Huffman coding",IEEE Trans Comput-Aided Des Integr, Circuits Syst.,vol.22,no.6,jun.2003, pp.797-806.

[6] X. Kavousianos, E. Kalligeros and D. Nikolos, "Optical selective Huffman Coding for test data compression", IEEE Trans comput, vol.56, no.8, Aug.2007, pp.1146-1152.

[7] Gonciari. P. T., "Variable length input Huffman coding for system-on-a-chip test", *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, no.6, Jun.2003, pp. 783–796.

[8] Chandra and K. Chakrabarty, "System-on-a-chip data compression and decompression architecture based on Golomb codes," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* vol. 20, no. 3, Mar. 2001, pp. 355–368.

[9] A. Chandra and K. Chakrabarty, "Test Data Compression and Test Resource Partitioning for system on chip using Frequency Directed Run length coding*", IEEE Trans. Comput.,* vol.52,no8, Mar 2003, pp. 352-363.

[10] Aiman El-Maleh et al, "Test Data Compression for System-on-a-Chip using Extended Frequency-Directed Run-Length (EFDR) Code," IET Computers & Digital Techniques, vol. 2, No. 3, 2008, pp. 155–163.

[11] M. Nourani and M. Tehranipour, "RL-Huffman encoding for test compression and power reduction in scan application", *ACM Trans.Des. Automat Electron Syst*., vol.10, no.1, 2005, pp. 91-115.

[12] Lung-Jen Lee, Wang-Dauh Tseng, and Rung-Bin Lin, "An Internal Pattern Run-Length Methodology for Slice Encoding", *ETRI Journal*, Volume 33, Number 3, June 2011.

[13] H. Hashempour, L. Schiano, and F. Lombardi, "Error-resilient test data compression using Tunstall codes", in *Proc. IEEE Int. Symp. DefectFault Tolerance VLSI Syst.*, 2004, pp. 316–323.

[14] M. Knieser, F. Wolff, C. Papachristou, D. Weyer, and D. McIntyre, "A technique for high ratio LZW compression", in *Proc. Des., Autom., Test Eur.*, 2003, pp. 10116.

[15] M. Tehranipour, M. Nourani, and K. Chakrabarty, "Nine-coded compression technique for testing embedded cores in SOCs", *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, Jun. 2005, pp. 719–731.

[16] L. Lingappan, S. Ravi, et al, "Test-volume reduction in systems-on-a-chip using heterogeneous and multilevel compression techniques", *IEEE Trans.Comput.-Aided Des Integr. Circuits Syst.*, vol. 25, no. 10, Oct. 2006, pp.2193–2206.

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

[18] Seok-Won Seong and Prabhat Mishra, "Bitmask-Based Code Compression for Embedded Systems", *IEEE Transactions on computer-aided design of integrated circuits and systems*, 2008.

[19] Kanad Basu, Prabhat Mishra, "Test Data Compression Using Efficient Bitmask and Dictionary Selection", IEEE *Transactions on Very Large Scale Integration (VLSI) systems,* vol. 18, no. 9, September 2010.

[20] Lung-Jen Lee, et al, "2^n Pattern Run-Length for Test Data Compression", *IEEE transactions on computer- aided design of integrated circuits and systems*, vol. 31, no. 4, April 2012.

[21] Wang-Dauh Tseng & Lung-Jen Lee, "A Multidimensional Pattern Run Length method for test data compression", *in proc. Asian Test Symp*, 2009, pp. 111-116.