INTERNATIONAL JOURNAL OF ADVANCED RESEARCH IN ENGINEERING AND TECHNOLOGY (IJARET)

ISSN 0976 - 6480 (Print)
ISSN 0976 - 6499 (Online)
Volume 5, Issue 1, January (2014), pp. 24-35
© IAEME: www.iaeme.com/ijaret.asp
Journal Impact Factor (2013): 5.8376 (Calculated by GISI)
www.jifactor.com

IJARET
©IAEME

ERROR RESILIENCE IN CASE OF TEST DATA COMPRESSION TECHNIQUE FOR IP CORE BASED SOC

Usha S Mehta, Rakesh G Trivedi, Nandish B Thaker

EC Dept, Institute of Technology, Nirma University, Ahmedabad, India

ABSTRACT

The fault coverage is the major quality criteria for any test data. Any bit flip in test data can reduce the fault coverage and hence yield. As per recent trends, use of preprocessed and compressed test data has been common for testing of IP core based SoC. The bit flip in such data can be more dangerous to fault coverage. Further, the hidden structure of IP core does not allow the system integrator to analyze the effect of bit flip on fault coverage. In this scenario, it is utmost necessary to make the test data compression method error resilient. In this paper, we have proposed a method to make test data error resilience. Its effect on % compression and test application time is calculated using highly cited ISCAS89 benchmark circuits. Also the on-chip decoder required for error resilient test data is proposed and corresponding results for area-overhead is shown.

Keywords: Error Resilience; Test Data Compression, IP core based SoC, EFDR.

I. INTRODUCTION

This section includes the need of error resilience for compressed test data set used for SoC testing.

A. Bit Flips and Need of Error Resilience in Test Data

While storing the test data in the ATE head or transferring the test data from ATE to DUT, there are many reasons which may cause the bit flips in test data. As per [1], the major reasons of such bit flips are: 1. the noise of loadboard which can affect transmission line between the ATE pins and the DUT 2. The generation of test inputs as pin waveforms to the DUT. In general, as testing requires very high speed (tens of millions of ATE cycles over hundreds of pins), few bit-flips must

This research work is supported by the GUJCOST-MRP on "New Testing Method at Design Level for Improvement in Performance of VLSI in nanometer region"

be considered to be unavoidable in ATE environments. During manufacturing test process, a single bit-flip can change data such that the fault coverage of the erroneous test set (as affected by bit-flips) can be reduced. Reduction in fault coverage causes a lower yield and increase in defects per million for products being shipped to customers. Hence it is necessary to make the test data error resilient.

B. Error Resilience in case of SoC

In today's era, the use of Intellectual Property (IP) cores have been very common for SoCs and NoCs [2]. Such IP Core comes with ready – to – use test data. System Integrates has no knowledge of its internal structure and hence if there is any error/bit flip in this ready – to – use test data, its effect on fault coverage cannot be known. Further, considering a very huge SoC test data set made up of individual IP test data, use of test data compression method has been the common practice for SoCs [3].

As it is very much clear that any bit flip in test data damages the fault coverage, it must be noted that bit flip in compressed test data is more dangerous because a single bit flip in compressed test data may result in a large number of bit flips in decompressed test data when erroneous test data is decompressed on chip. The coverage lose may be up to 30% in worst case for various compression approach [1].

Current research publications on test data compression scheme focuses on its compression efficiency, on-chip decoder area and overall test application time. Only a little work is found on error-resilience for compressed test data.

In this paper, it has been emphasized that the error resilience is one of major quality criteria for compressed test data in case of hidden structure of IP cores used in SoCs.

II. PREVIOUS WORK

During literature survey on error resilience, it is found that the work done by Hashempour H and et.al in nearly 2005 is the remarkable.

First of all, let us understand the effect of a bit flip in compressed data on the corresponding decompressed data set and fault coverage as described in [4].

(a) Constant size codeword and symbol

For the test data compression, if the coding style is with constant size codeword and constant size symbol, than there will be minimum effect of bit flip as only one code word will be replace by another erroneous code word and corresponding bits in decompressed data set may be affected.

(b) Variable size codeword and symbol

To get maximum compression efficiency, the current trends has focused on using variable length of symbols as well as variable length of code words. Some of the examples of variable length coding styles are Golomb[5], FDR[6], EFDR[7], MFDR[8], Huffman Coding[9], Selective Huffman Code[10], Modified Selective Huffman Code[11] etc.

As per [12], in variable length code words, the bit flip may cause the effect "propagation and shift" which means either of these two conditions may arise: 1. the affected codeword is broken into one or more valid code words 2. The affected codeword is broken into valid code words and a "dangling suffix" which is not a valid codeword. Considering the example from [12], the set of Huffman code words given by {0, 10, 110, 111}. A bit-flip affecting codeword 0 results into a zero valid codeword and a dangling suffix of 1 which is not a valid codeword. A bit-flip affecting the first bit of codeword 10, results into 00, i.e., two valid code words. A bit-flip at the second bit of codeword 111, results into 101, i.e., one valid codeword and a dangling suffix of 1 which is not valid.

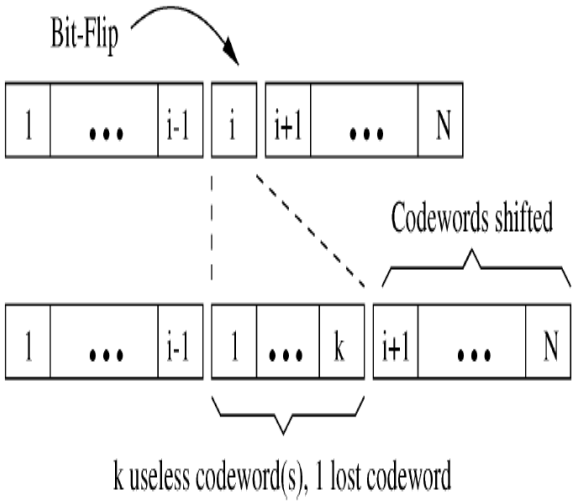

Fig.1. The shift effect of bit flip [12]

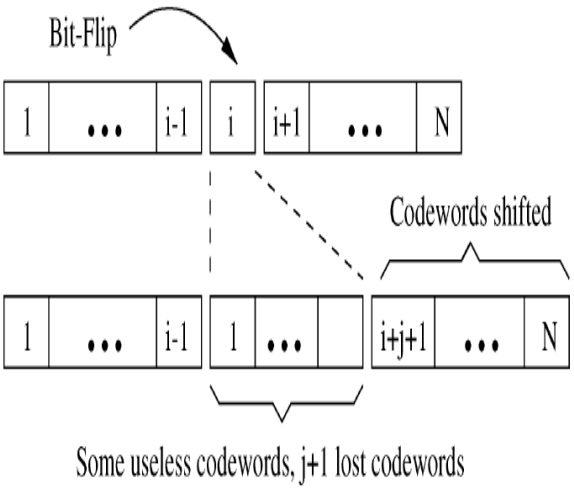

Fig. 2 The Propagation effect due to bit flip [12]

From above discussion, it can be concluded that because of a bit flip in codeword, one or more code words can be lost and remaining code words are shifted by at least one codeword. This contributes to synchronization loss. After decompression, it is important that the unaffected sequence of bits appears exactly in the same location within the uncompressed test stream as it would if there was no bit-flip. In the presence of bit-flips, this sequence is preceded by a corrupt sequence of corrupt bits is different from the number of otherwise correct bits in the uncompressed stream, then there is a loss of synchronization, i.e., the upcoming unaffected sequence of bits will be misplaced. Synchronization loss acts as if the unaffected sequence of bits is a random set of bits.

With a goal to reduce the negative effect of bit flips in test data on fault coverage, the effect of bit flip in a specific test vector is constrained within that vector only by using bit padding and vector padding[13]. Bit padding and vector padding avoids the propagation of error to next vectors.

Still avoiding propagation does not provide the error free testing. So, in this paper, a method is proposed for error free testing in case of SOC.

III. PROPOSED METHOD

A very popular approach which corrects the negative effects of bit-flips in data communication is based on the use of parity checks, for example, the data being transmitted is padded with the even or odd parity bit. At receiver side the parity is checked and if the parity of transmitted and received data does not match, the receiver requests the retransmission of data to the transmitter.

The same concept of parity bit addition and checking we have adopted in this paper for error resilience. The parity check ensures accurate data transmission between nodes during communication. A parity bit is appended to the original data bits tocreate an even or odd bit number with value one. The source then transmits this data via a link, and bits are checked and verified at the destination. Data is considered accurate if the number of bits (even or odd) matches the number transmitted from the source. Parity checking eliminates data communication errors and it is a simple method of network data verification and has an easy and understandable working mechanism.

As an example, if the original code word is 1001001, there are three 1s. When even parity checking is used, a parity bit with value 1 is added to the data's right side to make the number of 1s even; transmitted codeword becomes 10010011. However, if odd parity checking is used, then parity bit value is zero; 10010010.

If the original codeword contains an even number of 1s (1101001), then parity bit of value 1 is added to the data's right side to make the number of 1s odd, if odd parity checking is used and data transmitted becomes 11010011. In case data is transmitted incorrectly, the parity bit value becomes incorrect; thus, indicating error has occurred during transmission.

At first glance, this method may not be appraised considering that it increases the redundant bits so data volume is increased and further, it demands the retransmission so transmission time overhead may increase.

But considering the following facts, we are proposing the same parity bit method for error resiliency in test data:

1. For maximum possible compression efficiency, majority of the test compression methods [14, 15] prefer to preprocess the test data using reordering and differentiation. Such preprocessed test data is highly sensitive to bit flip. Even the bit padding or vector padding cannot stop the effect of error propagation to next vector in case of reordering and difference vector.

2. Further, as stated earlier, when there is no scope of use of fault simulation, rather than knowing the damage on fault coverage, the only option remains is to avoid the damage on fault coverage.

3. Reduction in compression ratio because of redundant parity bit is comparable to the reduction in compression caused by other methods to enhance error resilience like bit padding or vector padding.

4. The extra time required to retransmit the codeword can be a trade off against the error resilience and fault coverage.

5. The extra area overhead in on-chip decoder because of parity checking may be negligible considering the today's million gate SoC.

Here, in this method, we are adding one parity bit at the end of each codeword and transmit the data with parity bit itself.

For example the original test vector is 000001110000000010001. We have selected here the Hamming distance based reordering, column wise bit stuffing and difference vector as a preprocessing method for data because it guarantees comparatively maximum compression [15]. Further, as per [16], the compression method which uses Runs of '0s followed by 1' as well as Runs of '1s followed by 0' ensures the minimum number of symbols for a given test data set. If the number of symbols will be minimum, the parity bits added per symbol will be less for given test data set. Hence, we have used here EFDR test data compression method. Let's consider an example of the EFDR coding method for given sequence for runs like: 000001110000000010001. The code words will be like this: first bit of codeword will be to indicate the type of run followed by the bits indicating the length of run.

run 000001 : type of run :0 code for run length of 5: 1010

EDFR codeword : 01010

run 110 : type of run :1 code for run length of 2: 01

EFDR codeword: 101

run 00000001 : type of run :0 code for run length of 7: 1110000

EFDR codeword: 01110000

run 0001 : type of run :0 code for run length of 3: 1000

EFDR codeword: 01000

Hence coded data sequence will be 01010 101 01110000 01000. In the proposed method we will add each parity bit to coded data word. Here, in this method, odd parity bit checking is used. So each code word now appended with parity bit as follows:

codeword: 01010, parity bit: 1
codeword: 101, parity bit: 1
codeword: 01110000 , parity bit: 0
codeword: 01000, parity bit: 0

Hence, now the data sequence will be 010101 1011 011100000 010000. This sequence of code words appended with parity bits will now transfer from ATE to DUT. At DUT, the on chip decoder first of all checks for parity of code word. If parity does not match, it is assumed that the code word received has some error in it. So decoder will request retransmission of codeword. When the parity check is correct, the codeword is decoded and the decompressed data sequence is applied to scan chain for further testing.

IV. EXPERIMENTAL RESULTS

A. Bit Overhead because of Error Resilience

The proposed method of error resilience for IP Core Based SoC is implemented using MATLAB language. The experiments are conducted on a workstation with ai5 333os CPU, 2.70 GHz and 4 GB of memory. The seven largest ISCAS89 full-scan circuits have been considered for this experiment. For all ISCAS89 circuits, the test sets (with don't care) obtained from the Mintest ATPG program is used. The test sets were pre-processed with HDR-CBS-DV scheme and then applied EFDR test data compression and then parity bit was added to each codeword. Table I presents the experimental results for the widely cited seven ISCAS 89 bench mark circuits. It contains the total number of test patterns, total number of bits per vector and total number of bits in original complete test set, the total number of symbols considering EFDR method, the %compression achieved by EFDR, the % compression achieved in case of test data is pre-processed using HDR+CBS+DV method and % compression when error resilience is added to test data. For each of the case, the % compression without the application of proposed scheme from the original IEEE papers is shown and the % compression after application of the proposed scheme is also clearly shown. The amount of % compression obtained is computed as per the formula of % compression given ahead. In this table, all the reference cases against which comparisons are made used exactly similar source and conditions of test data i.e. all the reference cases have used the ISCAS89 circuits, the test sets (with don't care) obtained from the Mintest ATPG program. The original test data size is matching perfectly. Also the % bit overhead because of error resilience is calculated for each benchmark circuit.

Here the % compression and % bit overhead because of error resilience is calculated as follows:

$$\%\, dc = \frac{(tb - cb) * 100}{or\_c}$$

Where,

% dc = % data compression
tb = total # of bits in original test data
cb = # of bits in compressed data
or_c = total # of bits in original data

$$\%\, bo = erb * \frac{100}{tb}$$

Where,

% bo = % bit overhead
erb = # of error resilience bits added to compressed test set
tb = # total bits in test set

TABLE I: ERROR RESILIENCE FOR EDFR COMPRESSION METHOD

ISCAS 89 Benchmark Circuits	# Test vector	# Bits/ vector	# Total Bits	# of Symbols	% Compression for EFDR [7]	% Compression for HDR-CBS-DV + EFDR [15]	% Compression for HDR-CBS-DV + EFDR + Error Resilience Bit	% Bit Overhead Because of Error Resilience
S5378	111	214	23754	1732	51.93	60.03	52.7406	7.2914
S9234	159	247	39273	2802	45.89	57.56	50.4290	7.1347
S13207	236	700	165200	3132	81.85	86.40	84.5133	1.8959
S15850	126	611	76986	3623	67.99	70.43	65.7288	4.7061
S35932	16	1763	28208	860	Not Available	74.48	71.4407	3.0488
S38417	99	1664	164736	10077	60.57	65.67	59.5535	6.1171
S38584	136	1464	199104	12773	62.91	63.10	56.6849	6.4152

B. Test Time Overhead because of Error Resilience

Table II compares error resilience technique in terms of the number of clock cycles needed to decompress the test sets, which corresponds to the test application time. The number of clock cycles is obtained based on VHDL models of the decompression circuitry and the simulation results based on Modelsim simulators. Here the reduction in clock cycles in spite of the added error resilience bits is because of the effective modelling of FSM of on-chip decoder and reduction in FSM states as discussed in next section.

TABLE II. PERFORMANCE EVALUATION IN TERMS OF CLOCK CYCLES

ISCAS 89 Benchmark Circuits	# of clock cycles		% Performance Improvement in clock cycles
	EFDR	Error Resilience EFDR	
S5378	102210	40179	60.69
S9234	171961	66385	61.40
S13207	679540	130058	80.86
S15850	325607	102001	68.67
S35932	115489	36805	68.13
S38417	702936	249615	64.49
S38584	846395	305053	63.96
AVG	420591	132871	68.41

V. ON CHIP DECODER

Fig. 3 shows the proposed on-chip decoder for error resilient EFDR test data compression method. The decoder accepts EFDR encoded code words with maximum run length of 2000. The codeword is checked for even parity and if parity checking is successful then required run is generated otherwise a request of retransmission of codeword is sent to ATE.

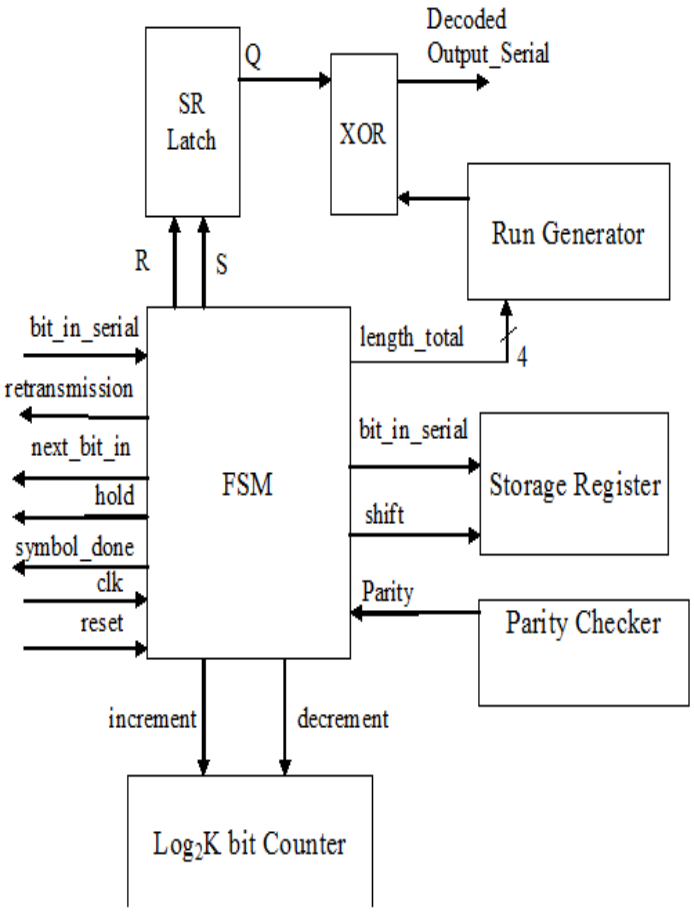

Fig. 3 Proposed Error Resilient EFDR On-chip Decoder

The proposed decoder consists of FSM, SR Latch, XOR, run generator, storage register and Log_2K bit counter. The description of various signals is as follow:

Parity: Parity signal checks for parity of 1s in given codeword.

clk: global clock to every block.

bit_in_serial: Serial input from encoder.

next_bit_in: shows readiness to accept next incoming bit. The next bit is accepted only if this bit is high.

Reset: to reset the FSM.R or S : this signal is used to store the run type.

Increment: to increment the Log_2K bit counter for every head bit.

decrement: to decrement the Log_2K bit Counter for every head bit.

Count: to keep feed backing the FSM about the state of count in Log_2K bit Counter.

Length_total: to give total length of input code word to Run Generator so that it can compute the run length encoded in code word.
Hold: To temporary hold transmission.

symbol_done: It shows when a symbol is successfully decoded.

The function of various blocks is as follows:

SR Latch :If run type is '0', **r** input is made high and for run type of '1' **s** input is made high.

Storage Register:On encounter of every bit, it is moved down into storage register and the total length is computed which helps to determine the total length of run.

Log_2K bit Counter:This counter is used to detect the end of codeword. It will be incremented for every input head bit and decremented for every input tail bit.

Run Generator:It will generate the total length for run. It will always generate run of 0s'. The output of this is given as input to the XOR gate. Now, if run type is '1', then Q = '1' and run of 0s' generated by this block will be XORed with Q which will generate run of 1s' bit by bit on every clock.

Parity Checker:It calculates parity and returns whether parity is odd or even.

Six states of FSM are as below:

S0: This is the **reset** state. Also this state accepts the first input bit and determines whether the run will be “run of 1s'” or “run of 0s'”.

S1: It will accept the first bit of code word to determine whether the run is of length is 1 or 2 or more.

S2:

1. If second input bit or first bit of code word is a '0' then it is clear that the run is either of length 1 or 2. So, FSM will reside in this state for one other cycle to accept next bit. After accepting this bit, it again in the same S2 state.

2. Now, the next likely input bit is parity bit. The FSM will compute the parity and if parity is even, run of required length will be generated in the same state.

3. Otherwise, it will send a retransmission signal and the FSM will come into its reset state i.e. S0 again.

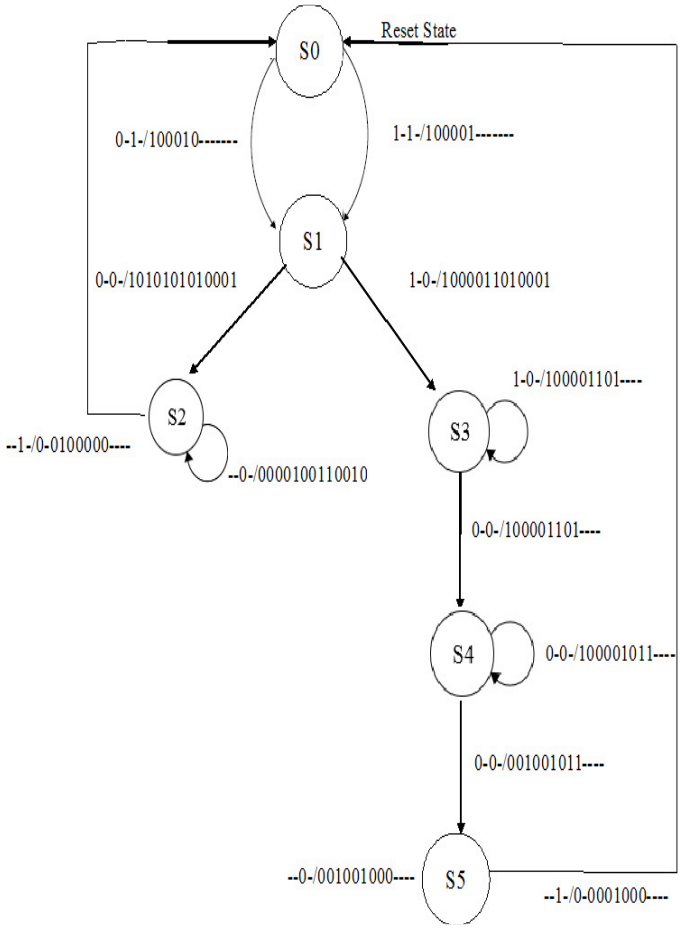

Fig. 4 FSM for Proposed Error Resilient EFDR On-chip Decoder

S3:

1. If run is having length of more than 2, FSM will come down to this state.
2. It will reside in this state and keep on accepting next bits till it encounter a '0' which is an indication of end of the head.
3. Meanwhile at each encounter of head bit the Log_2K counter is incremented.
4. On encounter of '0', it will come in the next state.

S4: Here, it will reside till all the tail bits are accepted and saved in the Storage Register. It will accept the parity bit and move down to state S5.

S5: If parity is even then run of required length is generated. Otherwise, the retransmission signal is made high and FSM gets reset.

A. Operation of other blocks

If run type is '0', **r** input to SR Latch is made high and for run type of '1' **s** input is made high. On encounter of every bit, it is moved down into storage register and the total length is computed which helps to determine the total length of run. Log_2K bit counter is used to detect the end of codeword. It will be incremented for every input head bit and decremented for every input tail bit.

Run Generator will generate the total length for run. It will always generate run of 0s'. The output of this is given as input to the XOR gate. Now, if run type is '1', then Q = '1' and run of 0s' generated by run generator block will undergo the following logical operation

Decoded_Output = (Run Generator o/p) xor Q

Which will generate run of 1s' bit by bit on every clock. Same operation will follow if run type is zero.

VI. CONCLUSION

It has been shown that the error resilience is a very essential quality of test data particularly when the test data is preprocessed and compressed and to be used with IP core based SoC. By using the current available techniques, the propagation of effect of bit flip to the next vector can be avoided but cannot be corrected. In proposed technique, the effect of bit flip is corrected. The corresponding overhead in terms of test data volume and testing time is also calculated. Further, the on-chip decoder for the proposed scheme is developed and its area overhead is shown. In short, this paper focuses on making the preprocessed-compressed test data more robust against bit flip error.

REFERENCES

[1] HamidrezaHashempour, et al, "Evaluation, Analysis, and Enhancement of Error Resilience for Reliable Compression of VLSI Test Data," IEEE Transactions on Instrumentation and Measurement, Vol. 54, No. 5, October, 2005.

[2] U. Mehta, K. Dasgupta, and N. Devashrayee, "Survey of test data compression techniques emphasizing code based schemes," Proceedings of 12th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 09). Patras, Greece: IEEE, August 2009, pp. 617-620.

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

[4] H. Hashempour, L. Schiano, and F. Lombardi, "Error resilient test data compression using Tunstall codes," in *Proc. IEEE Int. Conf. Defect and* Fault Tolerance in VLSI Systems, 2004, pp. 316–323.

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

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

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

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

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

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

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

[12] H. Hashempour, L. Schiano, and F. Lombardi, "Evaluation of error resilience for reliable compression of test data," in *Proc. Design and Test in Eur. (DATE) Conf.*, 2005, pp. 1284–1289.

[13] P. Tunstall, "Synthesis of noiseless compression codes," Ph.D. thesis, Dept. Elect. Comput. Eng., Georgia Inst. Technology, Atlanta, GA, 1967.

[14] U. Mehta, K. Dasgupta, and N. Devashrayee, "Weighted transition Based Reordering, Colum wise Bit Filling and Difference Vector –A Power Aware Test Data Compression Method," An Open Access International Journal "VLSI Design", Special issue on "CAD for Gigascale SoC Design and Verification Issues".

[15] U. Mehta, K. Dasgupta, and N. Devashrayee, "Hamming distance based reordering and column wise bit stuffing with difference vector: A better scheme for test data compression with run length based codes," Proceedings of 23rd International Conference on VLSI Design (VLSID 10). Bangalore: IEEE, 2010, pp. 33-38.

[16] U. Mehta, K. Dasgupta and N. Devashrayee, "Combining unspecified test data bit filling methods and run length based codes to estimate compression, power and area overhead", IEEE Asia Pacific 33rd Conference on Circuits and Systems (APCCS-10), Malaysia, ISBN: 978-0-7695-4076-4 , December 2010.