Improving Compression ratio using Code based test data compression methods

[1]HEMALATHA.K , [2]GOVINDARAJ.V

PG STUDENT (M.E., VLSI DESIGN), K.P.R Institute of Engineering and Technology, Coimbatore,India.

[2]ASSISTANT PROFESSOR, Department of Electronics and Communication Engineering, K.P.R Institute of engineering and technology,Coimbatore,India.

ABSTRACT - Modern System on chip (SOC) designs integrate various modules and intellectual property (IP) cores into a single chip, in which a rapidly growing number of transistors are used. Test data volume in the SOC has been considered as a major contributor to the cost of manufacturing testing for integrated circuits. Test data compression is most widely used approach, which reduces test data volume to a smaller amount to save the memory of the ATE. Pattern run length is a compression method based on encoding runs of equal or inversely compatible patterns. Test data in the test set is divided into a number of equal segments. Each segment is constituted by a series of 2^{1n1} compatible patterns or inversely compatible patterns. In the proposed work pattern run length is combined with the Dictionary based compression method for better compression ratio To demonstrate the effectiveness of the proposed work, experiments are conducted on ISCAS89 benchmark circuits. Simulation results on ModelsimSE 6.3 shows that this work can achieve significant compression ratio of about 27%.

Keywords - Automated test equipment (ATE), System on chip (SOC), Pattern Run length (PRL), test data compression.

I. INTRODUCTION

A major problem in VLSI testing is test data volume which increases the testing time and the memory of the automatic test equipment (ATE).Several intellectual property (IP) blocks are present in the system on chip (SOC), each of which must be exercised by a large number of precompute test patterns. The increasing high volume of SOC test data is not only exceeding the memory but also lead to excessive testing time. The amount of data required to test ICs is growing in each technology generation. Increase in the integration density results in larger designs with more scan cells and faults. Moreover, achieving high test quality in ever smaller geometries requires more test patterns targeting delay faults and other fault models beyond stuck- at faults. Existing external testing involves storing all test vectors and test response on an external tester called, ATE. But these testers have disadvantages such as limited memory,speed and I/O channels.

Test data compression is a promising approach to reduce the increasing test data volume in the SOC designs. Test datacompression consists of test vector compression on the input side and response compaction on the output side. The advantage of test data compression is that it generates the complete set of patterns applied to the CUT with desired fault coverage. Test data compression is classified into three categories: Code based scheme, linear decompressor based scheme and broadcast scan based scheme. Code-based schemes use data compression codes to encode the test cubes. In this case the original data is partitioned into symbols, and then each symbol is replaced with a code word to form the compressed data. These code based scheme involves run length codes, dictionary codes, statistical codes and constructive codes. This code based compression is popularly used scheme and this paper deals with the run length based and dictionary based schemes.

II. RELATED WORKS

The problem of reducing the test data volume and testing time for core based SOCs has been reviewed by various literatures studies. Hamzaoglu et al., proposed the test data compaction technique [7], which reduces the volume of test data transferred from the workstation to the ATE. While these approaches effectively reduce the volume of test data, they do not reduce the ATE bandwidth requirements.Embedded deterministic test (EDT) is a novel test-data volume-compression methodology, which reduces manufacturing test cost by reduction in scan test data volume and scan test time was proposed [13]. But these approaches may suffer from inefficient tester channel usage.

Iyengar et al., proposed a new built-in scheme for testing sequential circuits based on statistical coding but this method is suitable only for circuits with small number of primary inputs. The selective Huffman coding [8] eliminates the disadvantage of statistical coding. This method splits the test vectors into fixed-length input patterns of size b (where b is a block size), and applies Huffman coding .But these method increase the computational complexity and large area overhead. Since the selective Huffman coding was not optimal, Kavousianos et al., proposed the optimal Selective Huffman coding [11] with low hardware overhead and better compression. The Golomb coding assigns a variable-length Golomb code, of group size mg, to a run of ’0’s as described in [1]. But these golomb codes are optimum only for a particular pattern distribution. An approach called,

 All Rights Reserved © 2014 IJSETR

"Frequency-directed run-length (FDR) coding was proposed [2] from two parts, a prefix and a tail, where both parts have the same length. But the FDR code requires a more complicated decoder with fixed area overhead. Gonciari et al., proposed the variable–length huffmann coding [6].This method decreases the ATE memory and channel capacity requirements by obtaining good compression ratios.

Nourani et al., proposed RL-Huffman coding [12] .This method mixes two encoding techniques to reduce power dissipation test data volume, and test pattern delivery time in scan test applications. Eventhough this method is applicable only when the number of don't care bits is high. A new test-data compression technique that uses exactly nine code words was presented in [5]. The technique capitalizes on the fact that many consecutive blocks of the test data can be merged together was presented in [3]. Compression is achieved in this scheme by storing the merged block and the number of blocks merged.

III. Pattern Run Length (PRL) Encoding

Pattern Run Length (PRL) is a code based compression method based on encoding compatible patterns within the test data. Two patterns of the same length are compatible or inversely compatible if every bit pair at the same position has the same or opposite value. This run length method initially partitions the test data into fixed length (L-bit) segments where L has a power of 2. After the partition, compression is conducted on $2^{|n|}$ compatible or inversely compatible patterns where n is a signed integer. PRL is classified into two ways: Internal 2^n PRL ($n<0$) and External 2^n PRL ($n\geq0$), characterized by the sign of the exponent n.

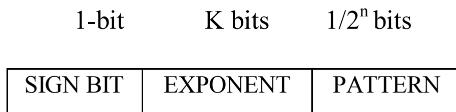


Fig 1 Internal 2^n PRL code word format.

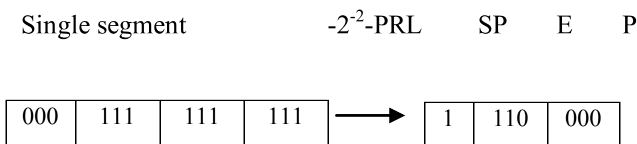


Fig 2 Encoding flow of internal 2^n PRL (Inversely compatible)

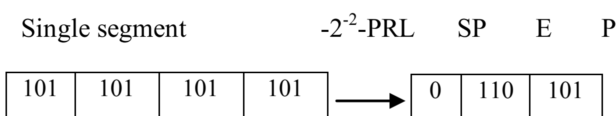


Fig 3 Encoding flow of internal 2^n PRL (compatible)

When $2^{|n|}$ runs of compatible or inversely compatible subs-segments inside a single segment is compressed, then it is termed as Internal 2^n PRL. For example, in this we consider a 12-bit segment "000111111111." If 12-bit segment is divided into four sub segments, each of which has a length of 12 x 2^{-2} = 3 bits. The first sub segment "000" is found inversely compatible with the last three sub segments "111". Hence, the segment can be encoded by 2^{-2} PRL, where the negative sign before the radix indicates that the first subsegment is inversely compatible with the following sub-segments and the exponent value −2 represents that this segment is divided into $1/2^{-2} = 4$ sub segments.

For internal PRL the code word is formed by three components as shown in Figure 1 where sign (S) is the sign before the radix (S="1" for negative and S= "0" for positive), exponent (E) is the K-bit exponent field value and pattern (P) is the first sub-segment data. As an example, the above 12-bit segment can be encoded into code word 1 110 000 as shown in Figure 2.

When 2^n consecutive compatible segments are compressed into a into a short one then it is termed as External 2^n PRL. For example, if we consider two consecutive segments, Si = "0x1xx0000xx0" and Si+1 = "01xx00000000," are compatible with the current reference segment, Sref = "011000000000," Si and Si+1 can be encoded using $+2^1$ PRL, here the exponent value 1 indicates that the number of consecutive segments got encoded is $2^1 = 2$.

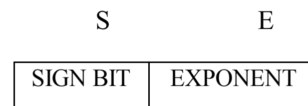


Fig 4 External 2^n PRL Code word format

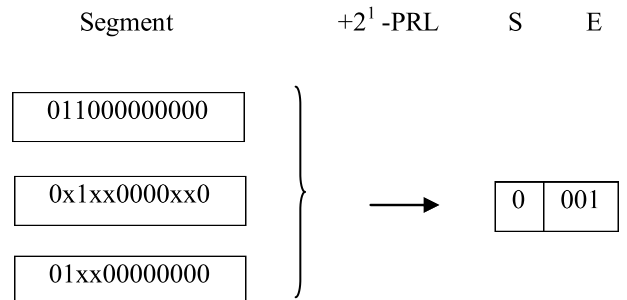


Fig 5 Encoding flow of external 2^n PRL

In Figure.5, where "(S)" stands for sign and "(E)" stands for exponent. Hence, segments Si and Si+1 can be encoded into code word 0001, as shown in Figure 4.in Figure 5,where the reference segment is compatible with the next two segments we denote the sign bit with '0'.

Table 1 shows the encoding table for a 3-bit exponent. Sixteen encoding types ranging from $\pm2^4$ PRL to $\pm2^{-4}$ PRL are presented along with their corresponding code word sizes given in the last column.

Table 1.Encoding table for a 3-bit exponent

2^nPRL	Type	S	E	Type	S	E	Size (bits)
EXT	$+2^4$	0	100	-2^4	1	100	4
	$+2^3$	0	011	-2^3	1	011	4
	$+2^2$	0	010	-2^2	1	010	4
	$+2^1$	0	001	-2^1	1	001	4
	$+2^0$	0	000	-2^0	1	000	4
INT	$+2^{-1}$	0	111	-2^{-1}	1	111	4+L/2
	$+2^{-2}$	0	110	-2^{-2}	1	110	4+L/4
	$+2^{-3}$	0	101	-2^{-3}	1	101	4+L/8

IV. Decompression Architecture Of 2^N PRL

The decompression architecture of 2^n-PRL consists of a control and generator unit (CGU) and finite-state machine (FSM). The CGU is responsible for data transmission control between the FSM and the ATE, controlling the scan chain and test pattern generation. The FSM is generally responsible for the code word identification. The decompressor is descibed as follows.

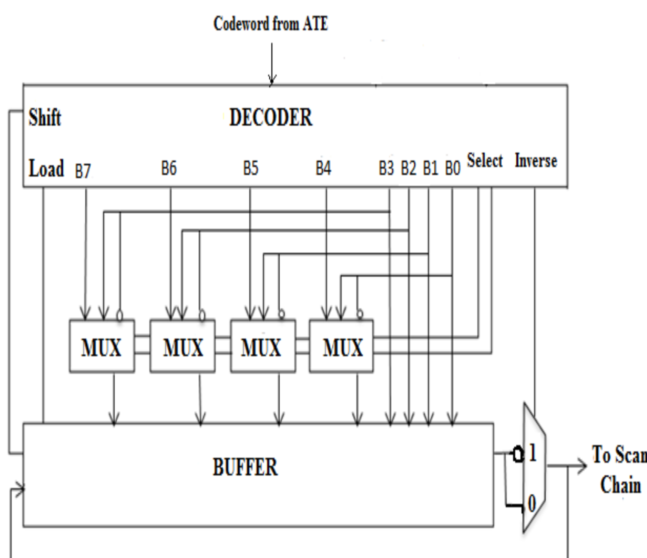


Fig 6 Decompression Architecture

A. Decoder

The decoder in the decompression architecture code word identification. It also generates control signals such as Inverse, Load, and Shift. Inverse signal decides whether the buffer content should be inverted due to decompressing inversely compatible segments. Load signal enables loading a test pattern which is decoded from the multiplexer array into the buffer when a segment is encoded by internal 2^n-PRL or an exception type. Shift enables shifts the test data in the buffer into the scan chain

B. Multiplexer Array

An array of multiplexers is present to produce test patterns. The number of multiplexers equals L−L/2K−1. So for K = 2 and L = 8 we need four multiplexers. These multiplexers are connected to the decoder and buffer as in figure 6. Here an 8-bit segment is partitioned into two 4-bit sub segments. Hence, the data inputs B0, B1, B2, and B3 are the first 4 bits of a code word. B4, B5, B6, and B7 are the last 4 bits of the code word if a segment is encoded using an exception type.

C. Buffer

The buffer is responsible to decompressed test data storage. It also sends those decompressed values to the scan chain. In case of external 2^n PRL, the data content in the buffer corresponds to a reference segment during compression. Hence, if the underlying code word is encoded as a compatible external 2^n PRL, Inverse from the decoder selects the buffer's content to be fed into the scan chain. On the other hand, if the underlying code word is encoded as a inversely compatible external 2^n PRL, then Inverse signal from the decoder selects the inverted value of buffer's content.

V. Dictionary Based Test Data Compression

Dictionary based code compression techniques provide compression efficiency as well as fast decompression mechanism. The basic idea is to take the advantage of commonly occurring instruction sequences by using a dictionary. The repeating occurrences are replaced by a code word that points to the index of the dictionary that contains the pattern. The compressed program consists of both code words and uncompressed instructions. The compression ratio is calculated using Equation (1).

$$\text{Compression ratio} = \frac{\text{compressed program size}}{\text{original program size}} \quad \ldots\ldots (1)$$

Table 2 Dictionary entries

Index	Dictionary content
00	00000000
01	11000010
10	01000010
11	01110010

 All Rights Reserved © 2014 IJSETR

Table 1 shows the dictionary enteries. These dictionary entries are generally compressed into code words. The codeword contains the information of thecompressed data along with the a single bit expressing whether the test data is compressed or uncompressed.

Table 2 shows the entries of the dictionary with the index value. Table 3 shows the dictionary based selection method. In that the first column shows the original data. If the data is matched with the entries of the dictionary then the data is compressed to specific codes. The second column shows whether the data is compressed or decompressed. If the bit is compressed it will be encoded as '1'. If the bit is not compressed it will be encoded as '0' and the original bit is obtained as output in in case of uncompressed data.

Table 3. Dictionary based selection

Original data	Compressed or uncompressed bit	Compressed Data
00000000	0	00
11000010	0	01
10001010	1	10001010
01000010	1	10
10001011	1	10001011
11000000	1	11000000
11000000	1	11000000
01110010	0	11
01000110	1	01000110
01000011	1	01000011

VI. Results And Discussion

The experimental results are simulated using ModelsimSE 6.3. Here an input circuit is taken as an example.For n input we need 2^n test patterns. Hence for 8 bit input we need to generate 2^8 test patterns (i.e. 256). So in order to have reduction in test data volume 2^n pattern run length is applied. By applying this compression method, compression ratio of 10% is obtained. We will take v as an input and generate the test patterns like a counter circuit. The v input will take 00000000 as initial input when the clock signal is given. Two outputs are taken, where data_out is the uncompressed output when the pattern is not encoded and data_out1 is compressed output when the patterns are encoded with 2^n compatible or inversely compatible patterns. This method is combined with dictionary for better compression.

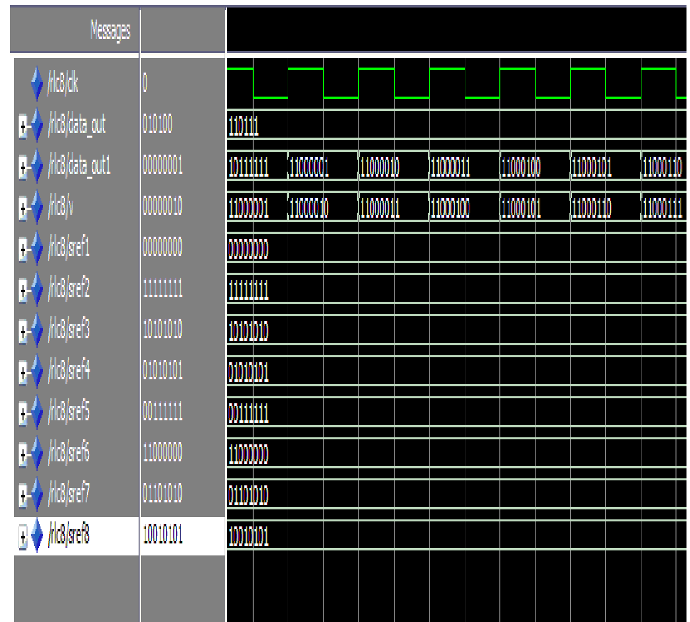
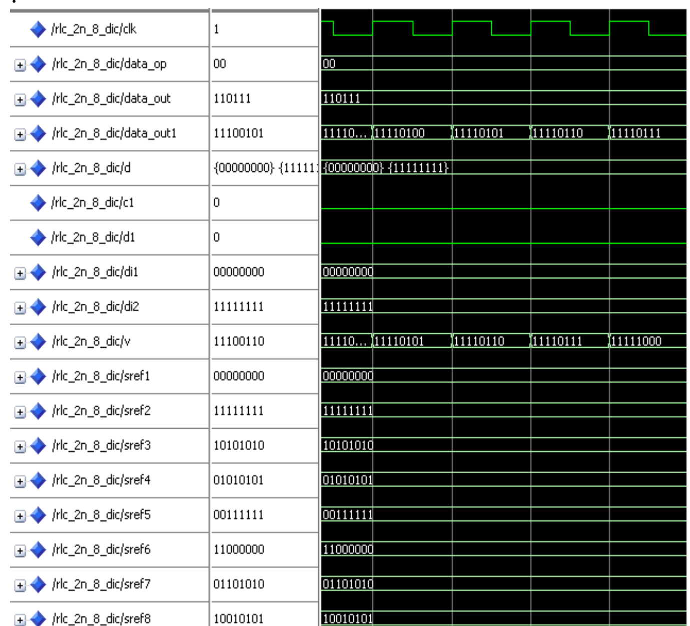


Fig 7. Simulation result for S208 Benchmark circuit using PRL



\

Fig.8 Simulation result for S208 Benchmark circuiusing PRL and Dictionary method

VII. CONCLUSION

Test data compression is an efficient methodology in reducing large test data volume for system on chip. This proposed method presents a new pattern run-length compression method whose decompressor is simple and easy

to implement. It encodes $2^{|n|}$ runs of compatible or inversely compatible patterns, either inside a single test data segment or across multiple test data segments. Experimental results on ISCAS89 benchmark circuit show that it can achieve a compression ratio of 10%. This proposed method is combined with the dictionary based approach for better compression ratio and have got about 27% compression.

VIII. REFERENCES

[1] Chandra, A. and Chakrabarthy, k. “System-on-a-chip data compression and architecture based on Golomb codes,” IEEE Trans.Computer Aided Des.Integr. Circuits Syst., vol.20, no.3, pp.335-368, Mar 2001.

[2] Chandra, A. and Chakrabarty, K. “Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run- length (FDR) codes,” IEEE Trans. Computer., vol. 52, no. 8, pp. 1076– 1088, Aug. 2003

[3] El-Maleh.A.H. “Efficient test compression technique based on block merging,” IET Computer. Digit. Tech., vol. 2, no. 5, pp. 327–335, 2008.

[4] El-Maleh, A. H. “Test data compression for system-on-a-chip using extended frequency-directed run-length (EFDR) code,” IET Comput. Digit. Tech., vol. 2, no. 3, pp. 155163, May 2008.

[5] Tehranipoor, M. Nourani, M. and Chakrabarty, K. “Nine-coded compression technique for testing embedded cores in SoCs,” IEEE Trans. Very Large Scale Integr. Syst., vol. 13, no. 6, pp. 719–731, Jun. 2005.

[6] Gonciari, P. T. Al-Hashimi, B. and Nicolici, N. “Variable-length input Huffman coding for system-on-a-chip test,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 22, no. 6, pp. 783–796, Jun. 2003.

[7] Hamzaoglu, I. and Patel, J. H. “Test set compaction algorithms for combinational circuits,” IEEE Trans. Computer-Aided Des. Integr. Circuits Syst., vol. 19, no. 8, pp. 957–963, Aug. 2000.

[8] Jas, A. Ghosh-Dastidar, J. Mom-Eng, J. and Touba, N. A. “An efficient test vector compression scheme using selective Huffman coding,” IEEE Trans. Computer-Aided Des. Integr. Circuits Syst., vol. 22, no. 6, pp. 797– 806, Jun. 2003.

[9] Kanad Basu, Phabhat Mishra, “Test data compression using efficient bitmask and dictionary methods” IEEE Trans. On VLSI systems, vol.18, no.9, Sep. 2010.

[10] Kavousianos, V. Kalligeros, V. and Nikolos, D. “Optimal selective Huffman coding for test-data compression,” IEEE Trans. Comput., vol. 56, no. 8, pp. 1146–1152, Aug. 2007.

[11] Mitra, S. and Kim, K. S. “X-Compact: An efficient response compaction technique,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 23, no. 3, pp. 421–432, Mar. 2004.

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

[13] Rajski, J. Tyszer, J. Kassab, M. and Mukherjee, N. “Embedded deter- ministic test,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 23, no. 5, pp. 776–792, May 2004

 All Rights Reserved © 2014 IJSETR