An Efficient Test Compression Technique Based on Block Merging

Aiman H. El-Maleh

Department of Computer Engineering

King Fahd University of Petroleum & Minerals

P.O. Box 1063, Dhahran 31261

Saudi Arabia

Email: aimane@kfupm.edu.sa

Abstract

Test data compression is an effective methodology for reducing test data volume and testing time. In this paper, we present a new test data compression technique based on block merging. The technique capitalizes on the fact that many consecutive blocks of the test data can be merged together. Compression is achieved by storing the merged block and the number of blocks merged. It also takes advantage of cases where the merged block can be filled by all 0's or all 1's. Test data decompression is performed on chip using a simple circuitry that repeats the merged block the required number of times. The decompression circuitry has the advantage of being test data independent. Experimental results on benchmark circuits demonstrate the effectiveness of the proposed technique compared to other coding-based compression techniques.

I. INTRODUCTION

With recent advances in process technology, it is predicted that the density of integrated circuits will soon reach several billion transistors per chip [1]. The increasing density of integrated circuits has resulted in tremendous increase in test data volumes. Large test data volumes not only increase the testing time but may also exceed the capacity of tester memory. The cost of automatic test equipment (ATE) increases significantly with the increase in their speed, channel capacity, and memory. Having limited tester memory implies multiple time-consuming ATE loads ranging from minutes to hours [2].

An effective way to reduce test data volume is by test compression. The objective of test data compression is to reduce the number of bits needed to represent the test data. The compressed test set is stored in the tester memory and a decompression circuitry on chip is used to decompress the test data and apply it to the circuit under test. Test data compression results in reducing the required tester memory to store the test data and the test time. Test data compression techniques can be broadly classified into three categories [3]: Code-based schemes, Linear-decompression-based schemes and Broadcast-scan-based schemes.

Code-based compression schemes are based on encoding test cubes using test compression codes. Techniques in this category include run-length-based codes, statistical codes, dictionary-based codes and constructive codes. Run-

length code compression techniques are based on encoding runs of 0's such as Golomb codes [4], frequency-directed run-length (FDR) codes [5], alternating run-length coding using FDR (ALT-FDR) [6], or based on encoding both runs of 0's and runs of 1's such as extended frequency-directed run-length (EFDR) codes [7]. Statistical codes are mainly based on using Huffman coding to encode the test data such as selective Huffman coding (SHC) [8]. Some techniques combine the use of Huffman and run-length coding such as variable-input Huffman codes (VIHC) [9] which encodes runs of 0's, and mixed run-length and Huffman coding (RL-HC) [10] which encodes both runs of 0's and runs of 1's. In [11], LFSR generated pseudorandom sequences are combined with multilevel selective Huffman coding to improve compression and to enhance detection of non-modeled faults. In [12], a method of compression is proposed based on the use of 9 code words (9C) to encode the test data.

Dictionary-based coding techniques use a dictionary to store the encoded symbols, and achieve compression by using dictionary indices to represent encoded symbols. Examples of dictionary-based encoding techniques include [13] which uses a full-dictionary and [14] which uses a partial dictionary. Constructive codes-based compression techniques are based on the fact that test cubes contain few care bits and use codes to construct the care bits. This includes the geometric-shapes based compression technique [15] which uses primitive geometric shapes to encode the test data. Other techniques include [16] based on constructing a test cube by flipping values of a previous test cube, [17] based on starting with an all 0 or all 1 test cube and then copying the opposite bits, and multilayer data copy technique [18] based on using a multilayer decompression buffer to construct the test cube.

Code-based techniques often require synchronization with the tester and may have synchronization overhead that affects the test application time. In [19], a general framework is proposed that addresses the synchronization overhead problem and facilitates the use of low cost testers for manufacturing test.

Linear-decompression-based schemes are based on using pseudo random pattern generators (PRPGs) such as linear feed back shift registers (LFSRs) and XOR gates to encode the test data. The effectiveness of this approach relies on the ability to encode a test cube using a linear decompressor. These include techniques that rely on combinational linear decompressors which are based on XOR gates such as [20-22] or sequential linear decompressors which rely on LFSRs or ring generators such as [23, 24]. These techniques are often coupled with Automatic Test Pattern Generation (ATPG) to maximize test cube encoding. Recently, a technique is proposed in

[25] that can improve compression of linear-decompression-based schemes based on scan inversion and decompressor reconfiguration.

Broadcast-scan-based compression schemes are based on driving a large number of scan chains using a small number of tester channels by broadcasting the same values to multiple scan chains. The effectiveness of this approach is enhanced by using reconfiguration in which the tester channels used to drive the scan chains are reconfigured to allow the scan chains to be driven from different tester channels for different configurations. Examples of these techniques include [26-30].

Recently, a methodology is presented in [31] for improving the amount of compression achieved by continuous-flow decompressors by using multiple ratios of scan chains to tester channels. The methodology is applicable to both linear-decompression-based schemes and broadcast-scan-based compression schemes. To maximize compression, a number of hybrid test compression techniques have been proposed in [32-35] that combine different compression techniques.

While linear-decompression-based and broadcast-scan-based compression schemes achieve very high compression and do not have synchronization overhead with the tester, they require the availability of circuit structural information as they rely on automatic test pattern generation and/or fault simulation. This makes them inapplicable for Intellectual Property (IP) cores, which are the basic building blocks of systems on a chip (SOC).

Test compression techniques can be further classified as either being test-independent or test-dependent. The advantage of test-independent compression techniques is that the decompression circuitry does not need to be changed due to changes in the test set. Test-independent compression techniques include: Golomb [4], FDR [5], ALT-FDR [6] EFDR [7], 9-coded (9C) [12], and Geometric [15].

It is worth mentioning here that the majority of compression techniques rely on the fact that test sets contain a large number of X's [36, 37]. Efficient test relaxation techniques for combinational and sequential circuits have been recently proposed in [36-38]. It is shown in [36] that for highly compacted test sets by Mintest [39] a large percentage of X's is extracted for most circuits. Some of the proposed compression techniques, such as [4-7], are based on replacing existing don't-cares with zeros or ones to achieve higher compression ratio. Such compression may adversely affect the fault coverage of non-modeled faults. Therefore, techniques such as [12] that leave at

least a portion of don't-cares unchanged may be preferred. These leftover don't-cares can be replaced randomly to help detect non-modeled faults.

In this paper, we present an efficient test-independent compression technique based on block merging. The technique is based on merging consecutive compatible blocks of the test data. The merged block and the number of merged blocks are stored in the encoded test data. In cases where the merged block can be filled by all 0's or all 1's, it is encoded by 2 bits, otherwise its content is stored. Test data decompression is performed on chip using a simple circuitry that repeats the merged block the required number of times. It should be observed that this technique has an advantage over run-length coding techniques [4-7] in the possibility of encoding a stream of test data having a mix of 0 and 1 patterns by a single code word. Furthermore, it leaves a portion of the don't cares unchanged, which can be randomly filled to help detect non-modeled faults. Our proposed technique compresses the test data without requiring any structural information of the circuit. Thus, it is applicable for test compression of intellectual property cores in SoCs.

The rest of the paper is organized as follows. In Section II, we present the proposed block merging compression technique. The design of the test decompression circuitry of the block merging compression technique is described in Section III. Experimental results and a comparison with other code-based test compression techniques are presented in Section IV. Finally, Section V concludes the paper.

II. BLOCK MERGING COMPRESSION TECHNIQUE

The Block Merging (BM) compression technique is based on partitioning the test set into blocks of size *b* bits and then merging consecutive compatible blocks. Two blocks are considered compatible if every two corresponding bits in the two blocks are compatible. Two bits are compatible if they have the same value or any one of them is an X. The technique merges all compatible consecutive blocks into one merged block and then stores the merged block along with a count indicating the number of blocks merged. It encodes the merged block in two different ways depending on whether the merged block can be filled by one value (i.e. 0 or 1) or it contains both values. If the merged block contains both values, then it will be stored as is, otherwise its content is encoded.

In order to reduce the number of bits used for representing the merged blocks count, the encoded merged blocks are grouped into six groups, where a prefix is used to represent each group group. This is motivated by the observation that the frequency of merged blocks count is the highest at low values and decreases with increasing values. Table I shows the six groups with their different encoding schemes. It should be observed that *b* is the block size and it is assumed to be between 4 and 10 bits. The first group, B=1, represents the case when a block cannot be merged with consecutive blocks and in this case it will be represented by a prefix 0 and *b* bits for storing the bits of the block. The second group, B=2, represents the case when only two blocks are merged and in this case a prefix 10 is used to represent the group. If the next bit is 0, this indicates that the block contains both 0's and 1's and hence its content needs to be stored using *b* bits. However, if the block can be filled by all 1's, then the two bits 11 are used, otherwise the bits 10 are used to indicate that the block is filled with all 0's. The next group represents the case when the number of merged blocks is between 3 and 6. In this case, the prefix 110 is used to represent this group. This is followed by 2 bits to represent the number of blocks merged. The next bit indicates whether the block is a filled block with all 0's or all 1's or a block containing both 0's and 1's. The same policy applies for the remaining groups. Thus according to this code, the maximum number of merged blocks supported by this code is 62. It should be observed that, for groups B>2, the number of 1's in the prefix code of each group is the same as the number of bits used for storing the merged blocks count code.

The proposed test compression technique shown in Table I was designed based on test data analysis of the frequency of occurrence of merged blocks count. Part of this analysis is shown in Table II and Figure 1. The numbers between brackets in the table represent the percentage of filled blocks. The block sizes used in the analysis are the ones that produced the highest compression ratio for each circuit. As can be seen from the results, the most frequently occurring merged blocks counts are less than 16. However, there are instances of merged blocks counts along the whole range. It can be also observed that the frequency of occurrences is the highest at the lowest merged blocks count and decreases with increased count. It is worth mentioning that the percentage of filled merged blocks ranges from 13% to 51%, which justifies the benefit obtained from encoding filled blocks.

The next example illustrates the encoding of the block merging technique for a block size of 5:

Input data: X0X1X 101XX XX111 1XX11 0X0X0 XX000 110XX (35 bits).

Encoded data: 001 11001010111 1010 0110XX (24 bits).

Note that the first 3 bits are for encoding the block size. It is assumed that block sizes allowed are in the range of 4 to 10 and hence 3 bits are used to encode them. Thus, 001 encodes a block size of 5. The first four blocks are compatible and are encoded by the bit stream 110 01 010111. The next two blocks are compatible and result in a merged block that can be filled by 0's and hence are encoded by the bit stream 10 10. Finally, the last block is stored as is and is encoded by the bit stream 0 110XX.

III. BLOCK MERGING DECODER DESIGN

In this section, we present the design of the Block Merging decoder describing its datapath and the control unit modeled as a finite state machine (FSM).

A. Datapath Design

The datapath of the Block Merging decoder is composed of eleven main components as shown in Fig. 2. The main functionality of each component is described below:

Counter1: This counter is a 5-bit counter and is used to store the code prefix (without the 0). It will receive the code prefix from the FSM and shift it in while SHIFT1 is set. The counter would start to decrement as soon as the DEC1 signal is set and when it reaches zero it will set RST1 signal indicating the end of this operation. When this counter receives five consecutive ones, it will set the signal MAX so that the FSM would stop sending more inputs to the counter.

Counter2: This counter is a 5-bit counter and is used to store the merged blocks count, and it will not start unless *Counter1* finishes.

Counter3: A 3-bit counter used to count the number of 1's in the prefix code to determine the number of bits of the merged blocks count to be read in *Counter2*. For each input shifted to *Counter1*, the counter is incremented.

However, for each input shifted to *Counter2*, the counter is decremented. The end of this operation is indicated by setting the RST4 signal.

Shift Register: This 3-bit register stores the block size and is loaded during the first three states of the FSM.

Counter4: A 4-bit counter that stores the decoded 4-bit block size and gets its input from the block size decoder. It is decremented during the generation of the block. Once the counter reaches zero it will set RST3 and reload itself with the data again from the *Shift Register* through the *Block Size Decoder*.

4/10-bit Shift Register: Configured to be used as a 4-bit shift register to a 10-bit shift register. It is composed of 14 multiplexers and 10 flip-flops controlled by 3 signals from the FSM and the output of the *3-8 Decoder* as shown in Fig. 3. The configuration is done based on the block size stored in the *Shift Register* through the *3-8 Decoder*. The register receives data through either the serial input or the least significant flip-flop for repeating the stored block. This is controlled by the REP signal. The serial input gets data either from the FSM directly or the latch based on the FILL signal.

3-8 Decoder: Used to configure the 4/10-bit shift register according to the block size (4 bits to 10 bits). It decodes the content of the *Shift Register* considering that code 000 represents a block size of 4, and code 110 represents a block size of 10.

A latch: Used to store the fill bit in case of a filled merged block. The FSM sets or resets the latch according to the filling bit.

Two Multiplexers: One multiplexer is inserted between the *4/10-bit Shift Register* and the FSM in order to choose the serial input either from the *latch* or the FSM directly. The other multiplexer is inserted at the output stage in order to drive the scan chain either directly from the serial input or from the output of the *4/10-bit Shift Register*.

Block Size Decoder: Decodes the block size from 3 bits to 4 bits and its output is connected to *Counter4*.

B. FSM Design

The Block Merging FSM is composed of fourteen states with 6 inputs and 16 outputs as shown in Fig. 4. The behavior of the Block Merging FSM is summarized as follows. S0 sets the EN signal to the tester indicating its readiness for receiving the next encoded bit. S1-S3 read the next 3 bits representing the block size and store it in the *3-bit Shift Register*. S4 checks the next bit to determine whether the number of merged blocks is greater than one or not. If it is greater than one, it goes to state S5, otherwise it goes to state S10. S5 checks the next bit to determine whether the number of merged blocks is two or not. S6 reads the prefix code of the codeword and stores it in *Counter1*. S7 reads the merged blocks count and stores it in *Counter2*. S8 checks the next bit to determine whether the merged block is a filled block (=1) or not. S9 checks the next bit for determining the filling bit and sets or rests the *latch* accordingly. It also outputs the first bit of the filled block. S10 reads and outputs the merged block. S11 outputs the remaining bits of the filled block. S12 outputs the merged block a number of times equal to *Counter1* which is the prefix code without the 0. S13 outputs the merged block a number of times equal to *Counter2*, which is the merged blocks count code. It should be observed that the number of times a block is repeated is equal to the sum of the number of times stored in *Counter1* and *Counter2*.

Like other coding techniques, the block merging technique requires synchronization with the tester and has a synchronization overhead that may impact the test application time. However, the general framework proposed in [19] can be applied to reduce the impact of synchronization overhead.

IV. EXPERIMENTAL RESULTS

In order to demonstrate the effectiveness of the proposed block merging test compression technique, we have performed experiments on a number of the largest full-scanned versions of ISCAS89 benchmark circuits. We have used the Mintest [39] test sets generated using the dynamic compaction option.

Table III presents the compression ratios achieved by the proposed technique for different block sizes. The *compression ratio* is computed as:

$$Comp.Ratio = \frac{\#Original\ Bits - \#Compressed\ Bits}{\#Original\ Bits} X\,100$$

According to the results shown in Table III, the block sizes achieving the highest compression ratios vary depending on the test set. The highest compression for circuit s38417 is achieved with a block size of 4, while the highest compression for circuit s13207 is achieved with a block size of 9.

Table IV compares the proposed technique to a number of coding-based compression techniques including: Golomb [4], FDR [5], ALT-FDR [6] EFDR [7], SHC [8], VIHC [9], RL-HC [10], and 9C [12]. It should be noted that SHC [8], VIHC [9] and RL-HC [10] are test-data dependent while the other techniques are test independent. As can be seen, the proposed block merging compression techniques achieves the highest average compression in comparison to all compared techniques. It also achieves the highest compression for 3 out of the 6 compared benchmark circuits.

Table V compares the proposed block merging technique to both FDR [5] and EFDR [7] techniques in terms of the number of clock cycles needed to decompress the test sets, which corresponds to the test application time, and the estimated area of the synthesized decompression circuitry. The number of clock cycles and the estimated area are obtained based on VHDL models of the decompression circuitry of the three techniques. The average number of clock cycles needed to decompress the test sets is less than both FDR and EFDR techniques, correlating with the compression ratios achieved. This indicates that the block merging compression technique will result in less test application time.

The decompression circuitry for the three techniques were synthesized using Xilinx Spartan2 XC2S100-6tq144 FPGA optimized for area. The estimated gate count for the block merging decompression circuitry is higher than that for the FDR and EFDR compression techniques. However, the area overhead for the three techniques is considered small in comparison with the circuit size.

V. CONCLUSIONS

In this work, we have presented an efficient test-independent compression technique based on merging consecutive blocks. The merged block, along with the merged blocks count, are stored in the compressed test set. The merged block is either stored as is or encoded as a filled block. The proposed technique leaves a portion of the don't cares unchanged, which can be randomly filled to help detect non-modeled faults. Furthermore, it compresses the test data without requiring any structural information of embedded cores. Thus, it is applicable for test compression of intellectual property cores in SoCs. Based on experimental results on ISCAS benchmark circuits, the proposed technique achieves higher compression ratios in comparison to other coding-based compression techniques.

ACKNOWLEDGMENT

The author would like to thank Mr. Mohammad Kalkattawi for his help in the implementation of this work. This work is supported by King Fahd University of Petroleum & Minerals.

REFERENCES

[1] Semiconductor Industry Association. International Technology Roadmap for Semiconductors, 2001 Edition. http://public.itrs.net/Files/2001ITRS/Home.htm.

[2] Vranken, H., Hapke, F., Rogge, S., Chindamo, D., and Volkerink, E. : "ATPG padding and ATE vector repeat per port for reducing test data volume," Proc. Int. Test Conf., Charlotte, NC, Sep. 2003, pp. 1069–1078.

[3] Touba, N. A.: "Survey of test vector compression techniques," IEEE Design & Test of Computers, 2006, 23, (4), pp. 294–303.

[4] Chandra, A., and Chakrabarty, K.: "System-on-a-chip data compression and decompression architecture based on Golomb codes," IEEE Trans. Computer Aided Design, 2001, 20, (3), pp. 355–368.

[5] 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, 2003, 52, (8), pp. 1076–1088.

[6] Chandra, A., and Chakrabarty, K.: “A unified approach to reduce SoC test data volume, scan power, and testing time,” IEEE Trans. Computer Aided Design, 2003, 22, (3), pp. 352–363.

[7] El-Maleh, A., and Al-Abaji, R.: “Extended frequency-directed run length code with improved application to system-on-a-chip test data compression” Proc. of the 9th IEEE International Conference on Electronics, Circuits and Systems, Dubrovnik, Croatia, September 2002, pp. 449-452.

[8] Jas, A., Gosh-Dastidar, J., Ng, M., and Touba, N.: “An efficient test vector compression scheme using selective Huffman coding,” IEEE Trans. Computer Aided Design, 2003, 22, (6), pp. 797–806.

[9] Gonciari, P., Al-Hashimi, B., and Nicolici, N.: “Improving compression ratio, area overhead, and test application time for system-on-a-chip test data compression/decompression,” Proc. Design Automation Test in Europe, Paris, France, March 2002, pp. 604–611.

[10] Nourani, M., and Tehranipour, M.: “RL-Huffman encoding for test compression and power reduction in scan application,” ACM Transactions on Design Automation of Electronic Systems, 2005, 10, (1), pp. 91–115.

[11] Kavousianos, X., Kalligeros, E., and Nikolos, D.: “Multilevel Huffman coding: An efficient test-data compression method for IP cores,” IEEE Trans. Computer Aided Design, 2007, 26, (6), pp. 1070–1083.

[12] Tehranipoor, M., Nourani, M., and Chakrabarty, K.: “Nine-coded compression technique for testing embedded cores in SoCs, ” IEEE Transactions on VLSI Systems, 2005, 13, (6), pp. 719-731.

[13] Reddy, S. M., Miyase, K., Kajihara, S., and Pomeranz, I.: “On test data volume reduction for multiple scan chain designs,” Proc. VLSI Test Symp., Monterey, CA, April 2002, pp. 103-108.

[14] Li, L., Chakrabarty, K., and Touba, N.A.: “Test data compression using dictionaries with selective entries and fixed-length indices,” ACM Trans. Design Automation Electrical Systems, 2003, 8, (4), pp. 470-490.

[15] El-Maleh, A., Al Zahir, S., and Khan, E.: “A geometric-primitives-based compression scheme for testing system-on-chip,” Proc. VLSI Test Symp., Marina' Del Rey, CA, April 2001, pp. 54–59.

[16] Reda, S., and Orailoglu, A.: “Reducing test application time through test data mutation encoding,” Proc. Design, Automation, and Test in Europe, Paris, France, March 2002, pp. 387-393.

[17] Wang, Z., and Chakrabarty, K.: “Test data compression for IP embedded cores using selective encoding of scan slices,” Proc. Int. Test Conf., Austin, Texas, November 2005, pp. 581-590.

[18] Lin, S.-P., Lee, C.-L., Chen, J.-E., Chen, J.-J., Luo, K.-L., and Wu, W.-C.: “A multilayer data copy test data compression scheme for reducing shifting-in power for multiple scan design,” IEEE Transactions on VLSI Systems, 2007, 15, (7), pp. 767-776.

[19] Gonciari, P. T., Al-Hashimi, B. and Nicolici, N.: “Synchronization overhead in SoC compressed test,” IEEE Transactions on VLSI Systems, 2005, 13, (1), pp. 140-153.

[20] Bayraktaroglu, I., and Orailoglu, A.: “Concurrent application of compaction and compression for test time and data volume reduction in scan designs,” IEEE Trans. Computers, 2003, 52, (11), pp. 1480-1489.

[21] Mitra S., and Kim K. S.: “XPAND: An efficient test stimulus compression technique, ” IEEE Trans. Computers, 2006, 55, (2), pp. 163-173.

[22] Krishna, C.V., and Touba, N.A.: “Adjustable width linear combinational scan vector decompression,” Proc. Int’l Conf. Computer-Aided Design, San Jose, CA, Nov. 2003, pp. 863-866.

[23] Wohl, P., Waicukauski, J.A., Patel, S., DaSilva, F. , Williams, T.W., and Kapur, R. : “Efficient compression of deterministic patterns into multiple PRPG seeds,” Proc. Int. Test Conf., Austin, Texas, November 2005, pp. 916-925.

[24] Rajski, J., Tyszer, J., Kassab, M., and Mukherjee, N.: “Embedded deterministic test,” IEEE Trans. on Computer-Aided Design, 2004, 23, (5), pp. 776-792.

[25] Balakrishnan, K. J., and Touba, N. A.: “Improving linear test data compression, ” IEEE Trans. on Computer-Aided Design, 2006, 14, (11), pp. 1227-1237.

[26] Samaranayake, S., Gizdarski, E., Sitchinava, N., Neuveux, F., Kapur, R., Williams, T.W.: “A reconfigurable shared scan-in architecture,” Proc. 21th VLSI Test Symp., Napa Valley, CA, April 2003, pp. 9-14.

[27] Wang, L.-T. , Xiaoqing, Wen, Furukawa, H., Fei-Sheng, Hsu, Shyh-Horng, Lin, Sen-Wei, Tsai, Abdel-Hafez, K.S., Shianling, Wu: "VirtualScan: A new compressed scan technology for test cost reduction," Proc. Int. Test Conf., Charlotte, NC, October 2004, pp. 916-925.

[28] El-Maleh, A., Ali, M. I., and Al-Yamani, A.: "A Reconfigurable broadcast scan compression scheme using relaxation based test vector decomposition," Proc. of Asian Test Symposium, Beijing, China, October 2007, pp. 91-94.

[29] Han, Y., Li, X., Swaminathan, S., Hu, Y., and Chandra, A.: "Scan data volume reduction using periodically alterable MUXs decompressor," Proc. of Asian Test Symposium, Calcutta, India, December 2005, pp. 372– 377.

[30] Shi, Y., Togawa, N., Kimura, S., Yanagisawa, M., and Ohtsuki, T.: "FCSCAN: An efficient multiscan-based test compression technique for test cost reduction, " Proc. Conf. on Asia South Pacific Design Automation, New York, NY, January 2006, 653–658.

[31] Putman, R., and Touba, N.: "Using multiple expansion ratios and dependency analysis to improve test compression, " Proc. IEEE VLSI Test Symp., Berkeley, CA, May 2007, pp. 211-218

[32] El-Maleh, A.: "A hybrid test compression technique for efficient testing of systems-on-a-chip," Proc. IEEE Int. Conf. on Electronics, Circuits and Systems, Dubai, UAE, December 2003, pp. 599- 602.

[33] Cho, S., Song, J., Yi, H. and Park, S.: "Hybrid test data compression technique for SOC scan testing," Proc. IEEE Int. SOC Conf., New Beach, CA, November 2005, pp. 69-72.

[34] Lingappan, L., Ravi, S., Raghunathan, A., Jha, N. K., and Chakradhar, S. T.: "Test-volume reduction in systems-on-a-chip using heterogeneous and multilevel compression techniques," IEEE Trans. on Computer-Aided Design, 2006, 25, (10), pp. 2193-2206.

[35] Arai, M., Fukumoto, S., Iwasaki, K., Matsuo, T., Hiraide, T.: "Test data compression of 100x for scan-based BIST, " Proc. Int. Test Conf., Santa Clara, CA, October 2006, pp. 1-10.

[36] El-Maleh, A., and Al-Suwaiyan, A.: "An efficient test relaxation technique for combinational and full-scan sequential circuits" Proc. VLSI Test Symp., Monterey, CA, April 2002, pp. 53-59.

[37] Miyase, K., and Kajihara, S.: "Don't care identification of test patterns for combinational circuits," IEEE Trans. Computer Aided Design, 2004, 23, (2), pp. 321-326.

[38] El-Maleh, A., and Al-Utaibi, K.: "An efficient test relaxation technique for synchronous sequential circuits" Proc. VLSI Test Symp., Napa Valley, CA, April 2003, pp. 179-185.

[39] Hamzaoglu, I., and Patel, J. H.: “Test set compaction algorithms for combinational circuits’’, Proc. Int. Conf. Computer-Aided Design, San Jose, CA, November 1998, pp. 283-289.

Table I Block merging encoding scheme.

Groups	Type	Codeword
B=1	No merging	**0**+b bits
B=2	1's & 0's	**10**0+b bits
	Fill with 1s	**10**11
	Fill with 0s	**10**10
B =3 to 6	1's & 0's	**110**+2 bits+0+b bits
	Fill with 1s	**110**+2 bits+11
	Fill with 0s	**110**+2 bits+10
B=7 to 14	1's & 0's s	**1110**+3 bits+0+b bits
	Fill with 1s	**1110**+3 bits+11
	Fill with 0s	**1110**+3 bits+10
B=15 to 30	1's & 0's	**11110**+4 bits+0+b bits
	Fill with 1s	**11110**+4 bits+11
	Fill with 0s	**11110**+4 bits+10
B=31 to 62	1's & 0's	**11111**+5 bits+ 0+ b bits
	Fill with 1s	**11111**+5 bits+11
	Fill with 0s	**11111**+5 bits+10

Table II Percentage of occurrence of merged block counts.

Circuit	B=1	B=2	B=3	B=4	B=5	B=6	B=7	B=8	%Filled
s5378	54.3(6.7)	12.6(2.6)	5.9(0.8)	4.6(0.7)	3.5(0.3)	3.9(1.2)	2.9(0.3)	1.3(0.0)	13.1
s9234	37.8(11.9)	20.8(5.4)	12.6(2.4)	7.3(2.0)	5.6(1.1)	3.9(0.6)	2.1(0.6)	2.1(0.3)	25.6
s13207	35.3(6.5)	14.7(2.3)	7.5(0.4)	5.2(0.8)	3.4(0.4)	3.6(0.1)	1.0(0.2)	1.5(0.3)	13.2
s15850	42.8(9.4)	19.1(2.9)	11.1(1.8)	6.0(0.7)	4.1(0.7)	3.2(0.3)	2.0(0.3)	1.1(0.1)	19.0
s35932	38.1(2.8)	9.8(4.8)	6.5(5.8)	1.0(0.1)	5.0(5.0)	12.4(11.9)	2.3(1.3)	1.1(1.1)	51.2
s38417	45.3(12.6)	18.1(10.2)	7.0(4.1)	5.0(3.2)	2.7(1.7)	2.5(1.7)	1.7(1.3)	1.8(1.0)	43.8
s38584	40.8(8.3)	16.3(4.0)	9.6(2.1)	6.8(1.6)	4.1(0.9)	3.1(0.6)	2.8(0.5)	2.5(0.6)	21.1

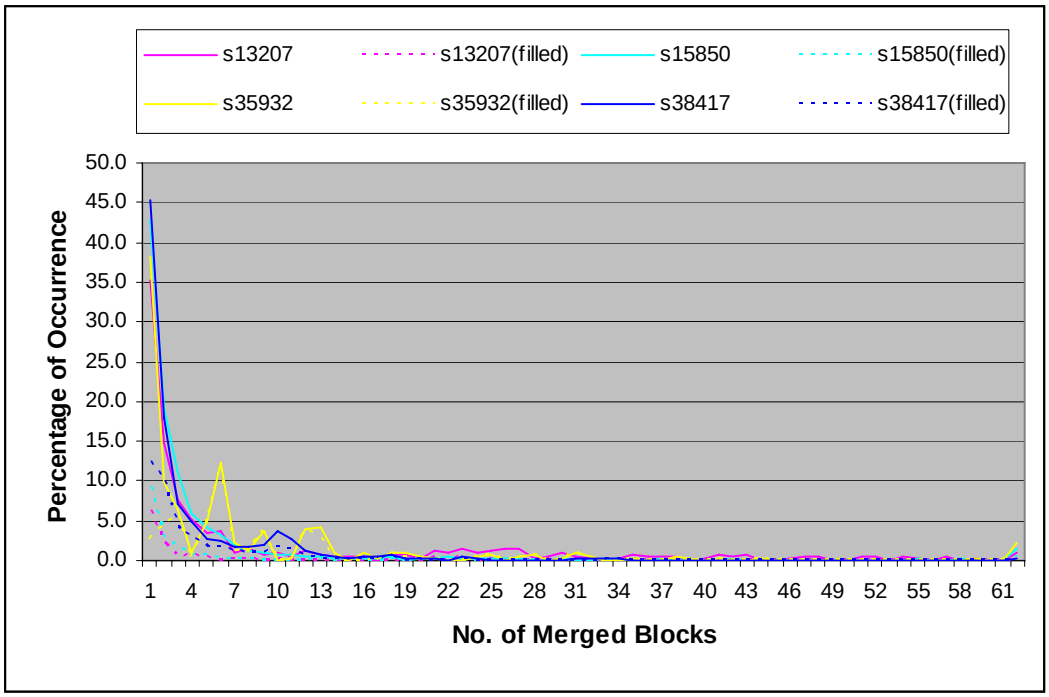

Figure 1 Test data analysis on frequency of occurrence of merged blocks count.

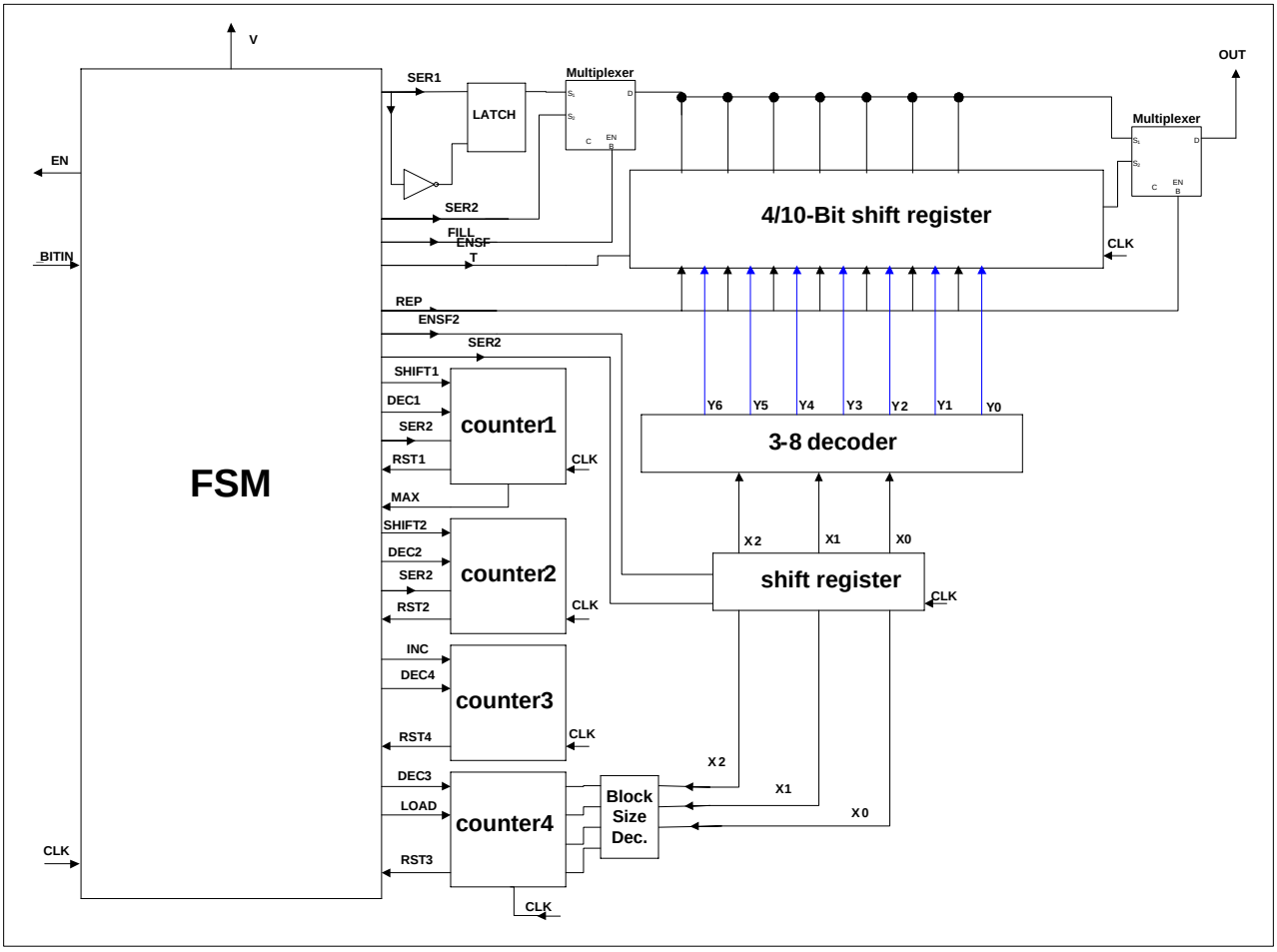

Figure 2 Block merging decoder design.

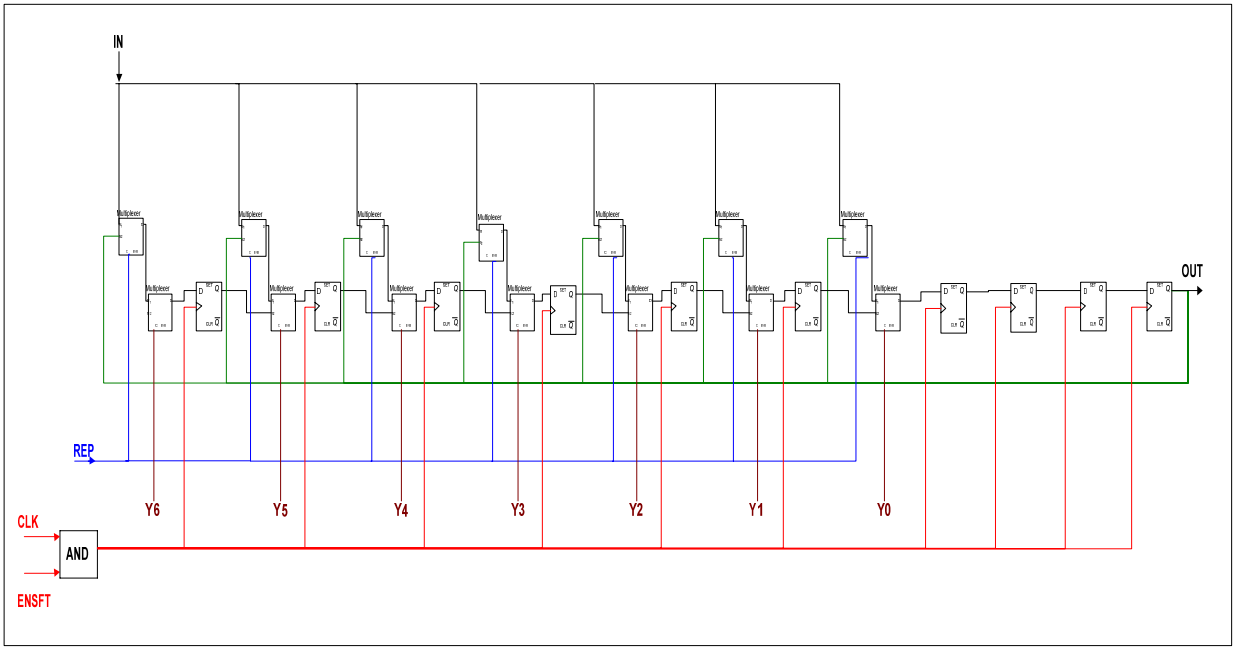

Figure 3 4/10-bit shift register.

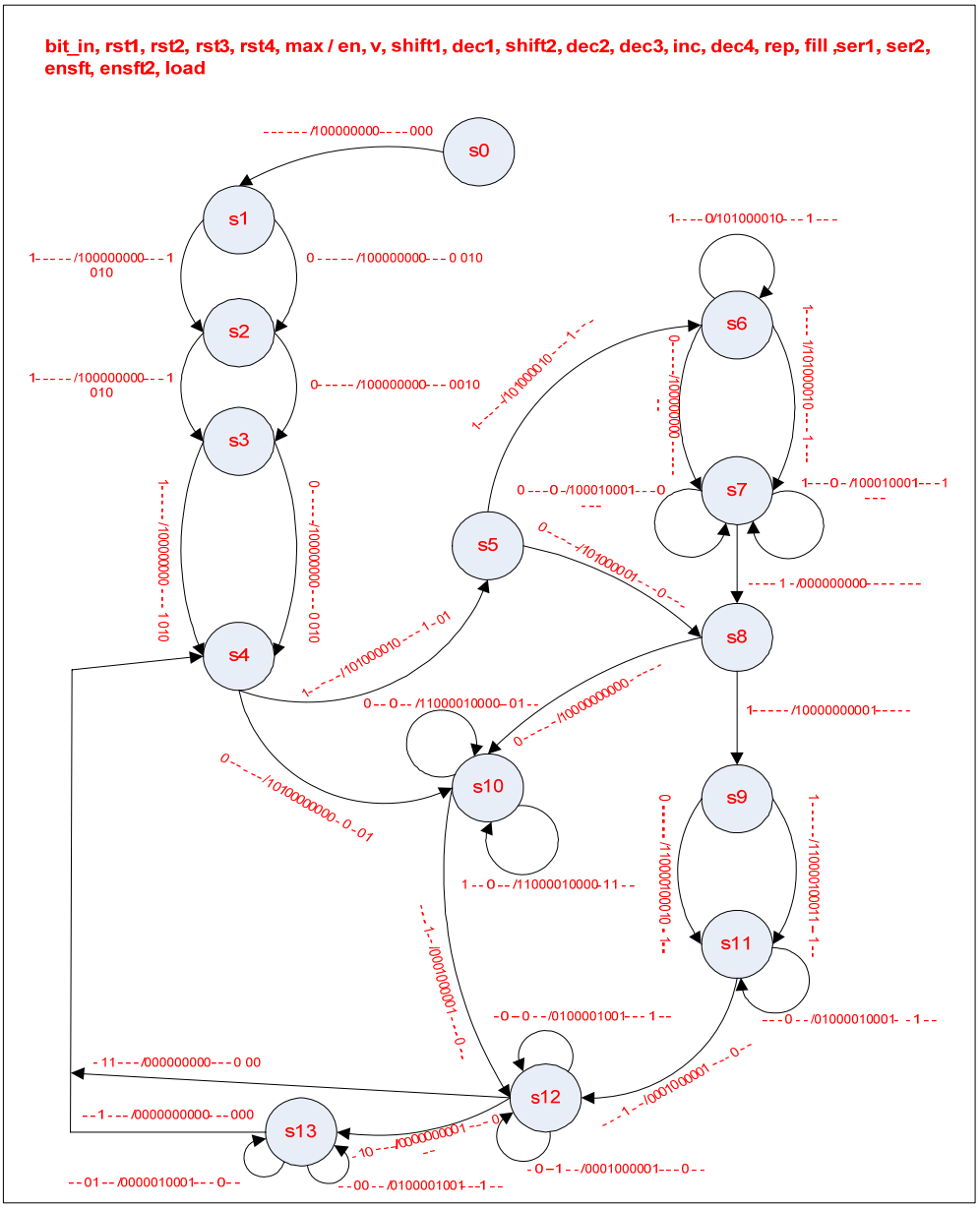

Figure 4 Block merging decoder FSM design.

Table III Block merging compression results.

Circuit	#VEC	Test Size	B=3	B=4	B=5	B=6	B=7	B=8	B=9	B=10	B=11
s5378	214	23754	50.49	52.27	53.50	**54.98**	53.51	52.99	53.25	52.47	52.75
s9234	247	39273	46.32	48.99	**51.19**	49.31	50.75	49.55	49.25	48.24	46.45
s13207	700	165200	81.78	83.19	84.70	84.24	83.45	84.69	**84.89**	84.74	83.97
s15850	611	76986	65.97	68.05	68.82	68.60	**69.49**	68.76	68.48	67.94	66.62
s35932	1763	164736	77.20	77.19	**78.35**	77.95	76.96	74.47	73.81	73.39	72.77
s38417	1664	199104	59.08	**59.39**	58.87	58.31	58.21	56.90	57.43	54.84	54.68
s38584	1464	28208	63.86	65.53	66.47	**66.86**	66.80	66.36	66.08	66.24	65.40

Table IV Compression ratio comparison with other coding-based compression techniques.

Circuit	GOLUMB[4]	FDR[5]	ALT-FDR[6]	EFDR[7]	SHC[8]	VIHC[9]	RL-HC[10]	9C[12]	BM
s5378	37.11	47.98	50.77	51.93	**55.10**	51.52	53.75	51.64	54.98
s9234	45.25	43.61	44.96	45.89	54.20	**54.84**	47.59	50.91	51.19
s13207	79.74	81.3	80.23	81.85	77.00	83.21	82.51	82.31	**84.89**
s15850	62.82	66.21	65.83	67.99	66.00	60.68	67.34	66.38	**69.49**
s38417	28.37	43.37	60.55	60.57	59.00	54.51	**64.17**	60.63	59.39
s38584	57.17	60.93	61.13	62.91	64.10	56.97	62.40	65.53	**66.86**
Average	51.74	57.23	60.58	61.86	62.57	60.29	62.96	62.90	**64.47**

Table IV Test application time and area comparison with FDR and EFDR techniques.

Circuit	# CLK Cycles			Estimated Area (Gates)		
	FDR [5]	EFDR [7]	BM	FDR [5]	EFDR [7]	BM
s5378	184331	173087	**159861**	**1380**	1457	1931
s9234	885537	848914	**811571**			
s13207	691169	680804	**673145**			
s15850	337633	326109	**307725**			
s38417	765365	**704633**	713101			
s38584	110081	102560	**93125**			
Average	495686	472685	**459755**			