# *An Efficient Test Vector Compression Technique Based on Block Merging*

*Aiman El-Maleh*
Department of Computer Engineering
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
aimane@ccse.kfupm.edu.sa

*Abstract*—**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 previous approaches.**

## I. INTRODUCTION

With recent advances in process technology, it is predicted that the density of integrated circuits will reach billion transistors per chip. 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.

One way to reduce the 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. Test data compression techniques can be classified into two categories; one is based on BIST and Pseudo-Random Generators (PRG) and the other is based on deterministic test compression.

Deterministic test compression techniques can be further classified into test-independent and 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. Several test-independent compression techniques have been proposed in the literature: Golomb[1], FDR[2], EFDR[3], Geometric[4], and 9C[5].

In this paper, we introduce 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 blocks merged is 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.

## 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 all two bits in corresponding positions 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 if the block can be filled by only 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 the 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 to 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

This work is supported by King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia.

TABLE I. BLOCK MERGING ENCODING SCHEME

| Groups | Type | Codeword |
|---|---|---|
| **B=1** | No merging | **0**+b bits |
| **B=2** | 1's & 0's | **100**+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 |

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 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 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.

## III. BLOCK MERGING DECODER

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. 1. 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). When it receives five consecutive ones, it sets 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.

*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.

*3-bit Shift Register*: Stores the 3-bit block size.

*4-bit Counter*: 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 3-bit 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. The configuration is done based on the block size stored in the 3-bit 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 3-bit 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.

*2 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.

*Size decoder*: Decodes the block size from 3 bits to 4 bits and its output is connected to the 4-bit counter.

### B. FSM Design

The Block Merging FSM is composed of fourteen states with 6 inputs and 16 outputs as shown in Fig. 2. 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 twice 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

TABLE II. BLOCK MERGING COMPRESSION RESULTS

| Circuit | #VEC | B=3 | B=4 | B=5 | B=6 | B=7 | B=8 | B=9 | B=10 | B=11 |
|---|---|---|---|---|---|---|---|---|---|---|
| **S13207** | 700 | 81.78 | 83.19 | 84.70 | 84.24 | 83.45 | 84.69 | **84.89** | 84.74 | 83.97 |
| **S15850** | 611 | 65.97 | 68.05 | 68.82 | 68.60 | **69.49** | 68.76 | 68.48 | 67.94 | 66.62 |
| **S35932** | 1763 | 77.20 | 77.19 | **78.35** | 77.95 | 76.96 | 74.47 | 73.81 | 73.39 | 72.77 |
| **S38417** | 1664 | 59.08 | **59.39** | 58.87 | 58.31 | 58.21 | 56.90 | 57.43 | 54.84 | 54.68 |
| **S38584** | 1464 | 63.86 | 65.53 | 66.47 | **66.86** | 66.80 | 66.36 | 66.08 | 66.24 | 65.40 |
| **S5378** | 214 | 50.49 | 52.27 | 53.50 | **54.98** | 53.51 | 52.99 | 53.25 | 52.47 | 52.75 |
| **S9234** | 247 | 46.32 | 48.99 | **51.19** | 49.31 | 50.75 | 49.55 | 49.25 | 48.24 | 46.45 |

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.

## IV. EXPERIMETAL RESULTS

In order to demonstrate the effectiveness of our scheme, we have performed experiments on a number of the largest full-scanned versions of ISCAS89 benchmark circuits. We have used the Mintest[6] test sets generated using the dynamic compaction option.

Table II presents the compression ratios achieved by the proposed technique for different block sizes. The *compression ratio* is computed as (# Orig. bits - #Comp. bits)/# Orig. bits X 100. As can be seen, the block sizes achieving the highest compression ratios vary depending on the test set.

Table III compares the proposed technique to both FDR and EFDR techniques in terms of the compression ratio and the number of clock cycles needed to decompress the test sets. The number of clock cycles is obtained based on VHDL models of the decoders of the three techniques. As can be seen, the block merging compression technique achieves higher compression than the FDR technique for all the circuits and achieves higher compression than the EFDR technique in five out of seven circuits. On average, it achieves around 15% higher compression than FDR and 2% higher compression that EFDR per circuit. Furthermore, the average number of clock cycles needed to decompress the test sets is less than both FDR and EFDR correlating with the compression ratios achieved.

The decompression circuitry for the three techniques were synthesized using Xilinx Spartan2 XC2S100-6tq144 FPGA optimized for area. The estimated gate count was 1105, 1176, and 1818 gates for the FDR, EFDR and BM compression techniques, respectively.

## V. CONCLUSIONS

In this work, we have presented an efficient test compression technique based on merging consecutive blocks. The merged block along with merged blocks count is stored in the compressed test set. The merged block is either stored as is or encoded if it is a filled block. The technique achieves higher compression ratios in comparison to other test-independent compression techniques.

TABLE III. COMPARISON WITH OTHER TECHNIQUES

| Circuit | Compression Ratio | | | # CLK Cycles | | |
|---|---|---|---|---|---|---|
| | *FDR* | *EFDR* | *BM* | *FDR* | *EFDR* | *BM* |
| **S13207** | 81.3 | 81.85 | **84.89** | 691169 | 680804 | **673145** |
| **S15850** | 66.21 | 67.99 | **69.49** | 337633 | 326109 | **307725** |
| **S35932** | 19.36 | **80.31** | 78.35 | 135665 | **115581** | 122339 |
| **S38417** | 43.37 | **60.57** | 59.39 | 765365 | **704633** | 713101 |
| **S38584** | 47.98 | 51.93 | **54.98** | 110081 | 102560 | **93125** |
| **S5378** | 43.61 | 45.89 | **51.19** | 184331 | 173087 | **159861** |
| **S9234** | 60.93 | 62.91 | **66.86** | 885537 | 848914 | **811571** |
| **Average** | 51.82 | 64.49 | **66.45** | 444254 | 421670 | **411552** |

## ACKNOWLEDGMENT

The author would like to thank Mr. Mohammad Kalkattawi for his help in the implementation of this work.

## REFERENCES

[1] A. Chandra and K. Chakrabarty, "Test data compression and decompression based on internal scan chains and Golomb coding," IEEE Trans. Computer Aided Design of Integrated Circuits and Systems, Vol. 21, No. 6, pp. 715-722, June 2002.

[2] 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 Trans. Comp., Vol. 52, pp. 1076-1088, 2003.

[3] A. El-Maleh and R. Al-Abaji, "Extended frequency-directed run length code with improved application to system-on-a-chip test data compression" Proc. 9th IEEE Int. Conf. Elect., Circuits and Systems, pp. 449-452, 2002.

[4] A. El-Maleh, S. Al-Zahir and E. Khan,"A geometric-primitives-based compression scheme for testing systems-on-a-chip," 19'th IEEE VLSI Test Symposium , pp. 54-59, 2001.

[5] M. Tehranipoor, M. Nourani, and K. Chakrabarty, "Nine-coded compression technique for testing embedded cores in SoCs," IEEE Transactions on Very Large Scale Integration Systems, Vol. 13, No. 6, 719 – 731, June 2005.

[6] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits'', Proc. Int. Conf. Computer-Aided Design, pp. 283-289, 1998.

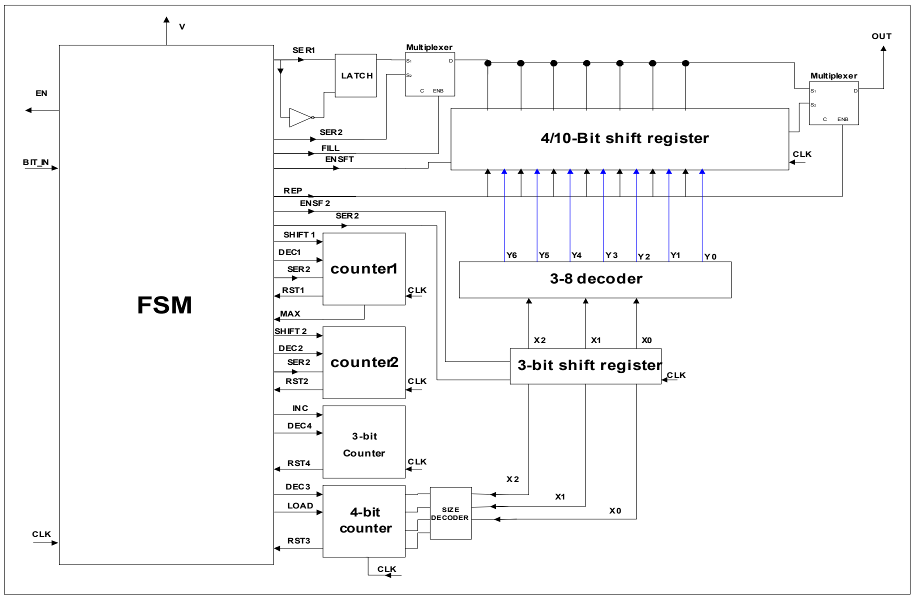

Figure 1. Block merging decoder design.

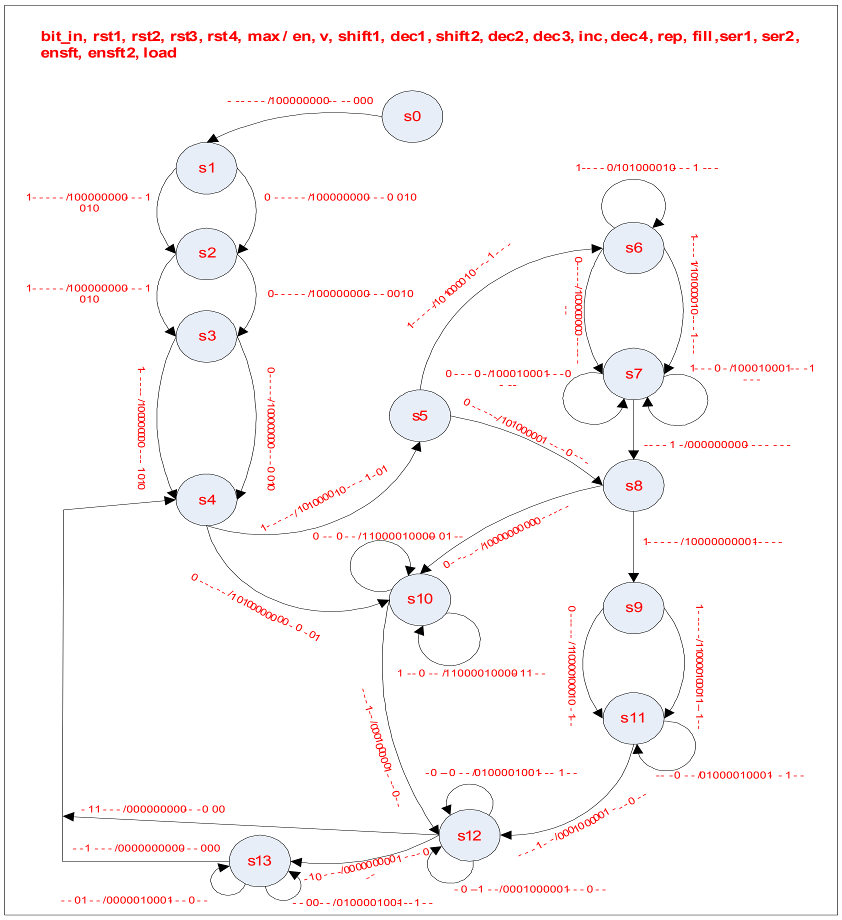
bit_in, rst1, rst2, rst3, rst4, max / en, v, shift1, dec1, shift2, dec2, dec3, inc, dec4, rep, fill, ser1, ser2, ensft, ensft2, load
s0
s1
s2
s3
s4
s5
s6
s7
s8
s9
s10
s11
s12
s13

Figure 2. Block merging decoder FSM design.