An Efficient Test Vector Compression Technique Based on Geometric Shapes

Saif al Zahir[1], Aiman El-Maleh[2], and Esam Khan[2]

[1] University of British Columbia, ECE Dept., Vancouver, B.C., Canada
[2] King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia
Email: saif_zahir@yahoo.com, {aimane,esamkhan}@ccse.kfupm.edu.sa

Abstract

One of the prime challenges of testing a system-on-a-chip (SOC) is to reduce the required test data size. In this paper, we introduce a novel compression / decompression scheme based on geometric shapes that substantially reduces the amount of test data and reduces test time. The proposed scheme is based on ordering the test vectors in such a way that enables the generation of geometric shapes that can be highly compressed via perfect lossless compression. Experimental results on ISCAS benchmark circuits demonstrate the effectiveness of the proposed technique in achieving very high compression ratio. Compared to published results, our technique achieves significantly higher compression ratio.

1. Introduction

Due to the rapid advancement in VLSI technology, it is possible to build very large systems containing millions of transistors on a single integrated circuit. This has resulted in a new paradigm for the design of integrated circuits where a system-on-a-chip (SOC) is constructed based on pre-designed and pre-verified cores and user defined logic (UDL). As the complexity of systems-on-a-chip continues to increase, the difficulty and cost of testing such chips is increasing rapidly [6], [7].

One of the challenges in testing SOC is dealing with the large amount of test data that must be transferred between the tester and the chip. The amount of time required to test a chip depends on the size of test data and the channel speed of data transfer. The cost of automatic test equipment increases significantly with the increase in their speed, channel capacity, and memory. Thus, reducing test storage and test time is one of the challenges for testing SOCs.

Applying lossless compression techniques can reduce test storage and test time, which is the objective of this work. Lossless compression techniques provide for the exact reconstruction of the original data from its compressed version. Run length coding, Huffman codes, Lempel-Ziv algorithms, and arithmetic codes are examples of lossless compression [8]. Several compression/decompression techniques are proposed in the literature to reduce test memory requirements and test time. All the proposed compression techniques are lossless and most of them attempt at utilizing either Huffman coding, run-length coding, or variations of these methods. Some sort of vector sorting to facilitate higher compression ratio precedes the implementation of these techniques. In [1], Burrows-Wheelers (BW) transformation is applied on the test data to produce longer and fewer runs, and then run length coding is applied to compress the transformed data. In [4], a statistical compression scheme is proposed that is based on variable length codewords to encode fixed length blocks of bits in test data. In [3], a compression scheme is proposed that uses careful ordering of the test data and formations of cyclical scan chains to achieve compression with run-length codes. In this scheme, a codeword is used to encode a block of data based on the number of zeros followed by a one in that block. Golomb code is used in [2], which is a variable-to-variable run-length code, to enhance the scheme described above. It divides the runs into groups, each is of size *m*. The number of groups is determined by the length of the longest run, and the group size *m* is dependent on the distribution of test data. In [9], a compression scheme using an embedded processor on a SOC is proposed. This scheme is based on generating the next test vector from the previous one by storing only the information about how the vectors differ. In [5], a different approach is proposed to design a core that can be tested with fewer number of test vectors.

In this paper, we introduce a novel and very efficient compression scheme for deterministic testing of SOCs based on geometric shapes. This scheme is designed based on test cubes to maximize the compression ratio. Test vector decompression is performed on chip and is implemented either in hardware or software. For hardware decompression option, a decoding circuitry is

placed on the chip to perform the decompression algorithm. However, for software decompression option, the compressed data is loaded into an embedded core. The embedded core will then execute the decompression algorithm and decompress the test data, which is then applied to the circuit under test. The decompression algorithm can be stored in a ROM on chip. This approach can reduce both the amount of test data that must be stored on the tester and the test time.

2. The Proposed Encoding/Decoding Algorithm

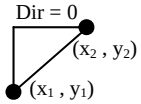
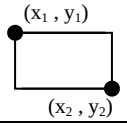
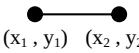
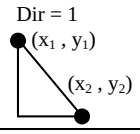
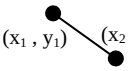
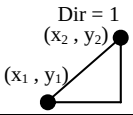
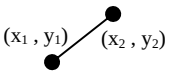
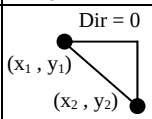
The proposed encoding/decoding algorithm is based on geometric shapes. In this work, we limited those shapes to the basic four namely: point, line, triangle, and rectangle as shown in Table 1. The choice of those shapes is made based on the following: (i) those shapes are bounded by a maximum of two point coordinates that can be encoded with a small number of bits; (ii) they are the most frequently encountered shapes in the test sets.

The following steps summarize the encoding process of the proposed algorithm:

Step 1. Test vectors sorting:

This step is crucial and has a significant impact on the compression ratio as inappropriate sorting may cause lower compression ratio. In this step, we aim at generating clusters of either zeros or ones in such a way that it may partially or totally be fitted in one or more of the geometric primitives shown in Table 1. Several sorting scenarios have been considered and investigated. In this work, we used a simple correlation-based sorting technique. This technique works as follows: At first, we chose the vector with the maximum number of zeros to become the first vector in the sorted vector set. Although this choice may not produce the optimal sorting of vectors, it was found to be a good heuristic based on experimentation. To determine the second vector, the "distance" of each of the remaining vectors to this vector is calculated and the vector that generates the maximum distance, i.e., most correlated, is chosen to be the second vector in the sorted set and so on. The "distance" between two vectors can be computed based on either the 0's, referred to as the zero-distance, or the 1's, referred to as the one-distance. For example, to compute the zero-distance between two vectors, *v1* and *v2*, we do the following. For each '0' in *v1*, we assign a weight of 1.0 to each of its immediate (vertical and diagonal) '0' neighbor, 0.25 to each of its immediate 'X' neighbor, and 0.0 to each of the immediate '1' neighbor in *v2*. Furthermore, for each 'X' in *v1*, we assign a weight of 0.25 to each of its immediate '0' or 'X' neighbor, and 0.0 to each of the immediate '1' neighbor in *v2*. A weight of 0.0 is given for all other cases. The

Table 1. The primitive geometric shapes.

	Lines	Triangles	Rectangle
Type1			
Type 2			X
Type3			X
Type 4			X

assignment of a 0.25 weight for an 'X' to each of its immediate neighbor be it an 'X' or a '0' is chosen due to the following reasons. First, this weight may help in completing or generating additional geometric shapes that can lead to a better solution. Second, this can help in generating blocks filled by 'X's which can be minimally encoded. Different weights have been experimented with, and a weight of 0.25 has been found to produce better results in most of the cases. The one-distance can be calculated similarly.

Since the first vector chosen is the one with the largest number of zeros, we performed sorting based on the zero-distance. In Table 2, we show a simple example to illustrate this sorting procedure. Let *v1*, *v2*, and *v3* be three test vectors to be sorted using the zero-distance approach. Then, zero-distance(*v1*,*v2*) = (0.0 + 0.25) + (0.0 + 0.25 + 0.25) + (0.25 + 0.25 + 0.25) + (0.25 + 0.25 + 1) + (0.25 + 1) = 4.25, and zero-distance (*v1*,*v3*)= (1.0 + 1.0) + (0.25 + 0.25 + 0.0) + (1.0 + 0.0 + 1.0) + (0.0 + 1.0 + 0.25) + (1.0 + 0.25) = 7.0. Based on the calculated distances, the sorting scheme will choose the order *(v1, v3, v2)*, as shown in Table 2. Note that this sorting produces geometric shapes that can be encoded efficiently, as shown in the table. However, if the vectors are sorted using the order (*v1, v2, v3)*, more shapes would have been needed to cover the same number of 0s. This sorting scheme produced good results in most cases compared to other scenarios.

Table 2. An example of test vector sorting.

Original Vectors	*v1*	0	X	0	0	0
	v2	1	X	X	X	0
	v3	0	0	1	0	X
Sorted Vectors	*v1*	0	X	0	0	0
	v3	0	0	1	0	X
	v2	1	X	X	X	0

Step 2. Test Data partitioning

A set of sorted test vectors, *M*, is represented in a matrix form, *RxC*, where *R* is the number of test vectors and *C* is the length of each test vector. The test data is segmented into *LxK* blocks each of which is *NxN* bits, where *L* is equal to $\lceil R/N \rceil$ and K is equal to $\lceil C/N \rceil$. For test vectors whose columns and/or rows are not divisible by the predetermined size of block *N*, a partial block will be produced at the right end columns and/or the bottom rows of the test data. Since the size of such partial blocks can be deduced based on the vector length and the block size, the number of bits used to encode the coordinates of the primitives can be less than *log N*. The decoder recognizes those special cases and decodes them properly.

Step 3. Encoding process

As mentioned earlier, the encoding process will be applied on each *NxN* block independently. The procedure of encoding is as follows:
(i) Extraction of shapes: Let the type of the bit to be encoded be *b* (*b* is either 0 or 1), then for each bit *b*, the largest shape covering bit *b* is extracted for each primitive geometric shape type (shown in Table 1). For example, in the sorted vectors in Table 2, extraction of shapes covering the first 0 produces a line of type 1, a line of type 2, a line of type 3, a rectangle of type 1, a triangle of type 2, and a triangle of type 4.
(ii) Covering problem: A covering problem is then solved based on the extracted shapes in *(i)* to identify the shapes covering all the bits to be encoded, with the smallest number of bits.
(iii) Steps *(i)* and *(ii)* are performed once for covering the zeros and another time for covering the ones. The block is then encoded based on the one that produces better results.

The format for encoding the shapes in a block is done as illustrated in Figure 1. For each block, if the number of bits needed to encode the shapes is larger than the number of bits in the block, then such a block is not encoded and the same test data is used. Otherwise, the block is encoded. If the block can be encoded with one rectangle covering all bits in the block, then such a block is marked as a block that is filled with either 0s or 1s. In this case, two bits are sufficient to encode the block instead of encoding it as a rectangle. Otherwise, the block is encoded with the geometric primitives. When encoding a block that contains geometric shapes, the number of shapes is encoded first followed by the encoding for each shape.

For this scheme, the decoding process is simple and straightforward. In this work, it is assumed that an embedded processor on chip will implement the decoder.

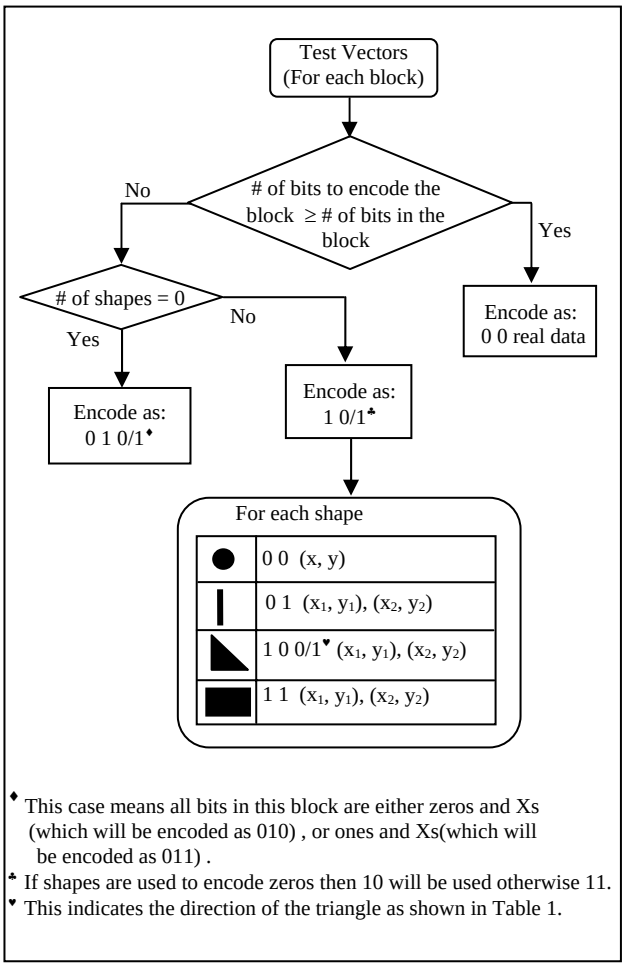


Figure 1. Schematic diagram of the encoding format.

A framework illustrating the details of how the test vectors can be transferred from the embedded processor to the tested parts of the chip has been outlined in [9]. A similar framework can be used for our decoding algorithm.

3. Experimental 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 test cubes obtained using the Mintest program [10] with dynamic compaction.

The test vectors were sorted to maximize the compression. In this work, test vectors were sorted greedily based on the zero-distance measure starting with the test vector with the largest number of 0s. The test sets were partitioned into blocks of sizes 8x8, 16x16, and 32x32 respectively. Then, the proposed encoding algorithm was applied for each case separately as shown in Table 3. The second column in the table shows the scan size, which is basically the width of a test vector. The *compression ratio* is computed as:

Table 3. Compression results of the proposed scheme.

			Block 8x8	Block 16x16	Block 32x32
Circuit	Scan Size	Original Bits	Cmp. Ratio	Cmp. Ratio	Cmp. Ratio
s5378f	214	23754	**54.69**	46.99	39.84
s9234	247	39273	**54.51**	50.2	42.03
s13207	700	165200	84.36	**84.86**	84.09
s15850	611	76986	**68.96**	65.90	62.38
s35932	1763	28208	65.0	73.49	**77.85**
s38417	1664	164736	**60.55**	58.42	52.67
s38584	1464	199104	**64.17**	59.89	53.01

Table 4. Comparison with Golomb codes [2].

	Proposed Technique		Golomb [2]		% Reduction Comp. Bits
Circuit	Comp. Ratio	Comp. Bits	Comp. Ratio	Comp. Bits	
s5378f	54.69	10763	40.7	14086	23.59
s9234	54.51	17865	43.34	22252	19.72
s13207	84.86	25011	74.78	41664	42.37
s15850	68.96	23897	47.11	40718	41.31
s35932	77.85	6248	0.0	28208	77.85
s38417	60.55	64988	44.12	92055	29.40
s38584	64.17	71339	47.71	104111	31.48

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

As can be seen from the table, the best compression ratio obtained is dependent on the block size used. However, for most of the cases a block size of 8x8 produces the best results (which are highlighted in the table). The effectiveness of the proposed encoding algorithm is clearly demonstrated as very high compression ratio was obtained for all the circuits (over 54%). The encoding algorithm is very fast as the CPU time for encoding each test set, for the three block sizes, was less than a minute. Since the encoding algorithm is fast and since the size of the block that produces the best results is dependent on the test set, encoding can be performed for the three block sizes and the best result is chosen.

In Table 4, a comparison between our technique with the one proposed in [2] is shown. The last column shows the percentage reduction in the number of compressed bits obtained by our technique relative to what is obtained in [2]. As can be seen from the table, for all the circuits, our technique achieves significantly higher compression ratio. Our technique reduces the size of compressed bits by 20%-78% more than the size of compressed bits in [2]. It is interesting to observe that for the circuit s35932, while the technique in [2] achieved 0.0% compression, our technique achieved 77.85% compression.

4. Conclusions

In this paper, a fast compression/decompression scheme based on geometric shapes has been presented. The test data is first sorted to minimize the number of geometric shapes to be encoded and maximize the compression ratio. Then, it is partitioned into blocks and each block is encoded separately. The scheme exploits the block size, the type of bits to be encoded, and whether or not to encode the block. Based on experimental results, our technique achieves a very high compression ratio. Compared to compression based on Golomb codes, our technique reduced the size of compressed bits by 20-78%. In this work, we assumed that the decompression algorithm is implemented in software and will be executed by an embedded processor on chip.

Acknowledgment

This work is supported by King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia, under project #FT/2000-07.

References

[1] T. Yamaguchi, M. Tilgner, M. Ishida, and D.S. Ha, "An Efficient Method for Compressing Test Data," *Proc. Int. Test Conf.*, pp. 191-199, 1997.

[2] A. Chandra and K. Chakrabarty, "Test Data Compression for System-On-a-Chip using Golomb Codes," *Proc. of IEEE VLSI Test Symp., pp. 113-120,* 2000.

[3] A. Jas and N.A. Touba, "Test Vector Decompression via Cyclical Scan Chains and its Application to Testing Core-Based Designs," *Proc. of Int. Test Conf.*, pp. 458-464, 1998.

[4] A. Jas, J.G. Dastidar and N.A. Touba, "Scan Vector Compression/ Decompression Using Statistical Coding," *Proc. of Int. Test Conf.*, pp. 458-464, 1998.

[5] A. Jas, K. Mohanram, and N.A. Touba, "An Embedded Core DFT Scheme to Obtain Highly Compressed Test Sets," *Proc. of IEEE Asian Test Symp.*, pp. 275-280, 1999.

[6] R. Chandramouli, and S. Pateras, "Testing Systems on a Chip," *IEEE Spectrum, pp. 42-47, Nov. 1996.*

[7] Y. Zorian, E.J. Marinissen, and S. Dey, "Testing Embedded-Core Based System Chips," *Proc. of Int. Test Conf.*, pp. 130-143, 1998.

[8] G. Gibson et-al, *Digital Compression for Multimedia*, Morgan Kaufmann Publishers, Inc., 1998.

[9] A. Jas and N. Touba, "Using an Embedded Processor for Efficient Deterministic Testing of Systems on a Chip", *IEEE Int. Conf. On Computer Design*, pp. 418-423, 1999.

[10] I. Hamzaoglu and J. H. Patel, "Test Se Compaction Algorithms for Combinational'', *Proc. Int. Conf. Computer-Aided Design*, pp. 283-289, 1998.