King Fahd University of Petroleum & Minerals

DEPARTMENT OF COMPUTER ENGINEERING
DHAHRAN 31261, SAUDI ARABIA

FINAL REPORT
SUBMITTED TO THE RESEARCH COMMITTEE
PROJECT No. FT2000/07

ENTITLED

A GEOMETRIC-PRIMITIVES-BASED COMPRESSION SCHEME FOR TESTING SYSTEMS-ON-A-CHIP

Period
December, 2000 to June, 2002

Submitted By:
DR. Aiman H. El-Maleh (Principal Investigador)

March, 2005

CONTENTS

List of Figures

List of Tables

Abstract

The increasing complexity of systems-on-a-chip with the accompanied increase in their test data size has made the need for test data reduction imperative. In this work, we introduce a novel lossless compression technique for testing systems-on-a-chip based on geometric shapes. The technique exploits reordering of test vectors to minimize the number of shapes needed to encode the test data. After sorting the test vectors, the test set is partitioned into blocks and each block is encoded separately. For testing a chip, the compressed test data is transferred from the automatic test equipment to the chip where it gets decompressed. Test data decompression is performed on chip and is performed either in hardware using a decoding circuitry or in software using an embedded processor on chip. In both cases, test decompression requires the availability of memory to store the decoded block. In this work, we have deomnstrated both cases. The effectiveness of the technique in achieving high compression ratio is demonstrated on the largest ISCAS85 and full-scanned versions of ISCAS89 benchmark circuits. The proposed technique achieved significantly higher compression ratio in comparison to other test compression techniques. Frequency-directed run-length (FDR) code is a variable-to-variable code based on encoding runs of 0's. In this work, we demonstrate that higher test data compression can be achieved based on encoding both runs of 0's and 1's. We propose an extension to the FDR code (EFDR) and demonstrate by experimental results its effectiveness in achieving higher compression ratio. In the Geometric-Primitives-Based Compression technique, some of the blocks are encoded by storing the real test data because the encoded block size is larger than the actual test data block size. Reducing the number of these blocks could result in higher test data compression. In this work, we propose hybrid test data compression techniques that exploit the use of either FDR or EFDR codes to reduce the number of blocks that are encoded by storing the real test data. Based on experimental results, we demonstrate the effectiveness of the proposed hybrid compression techniques in increasing the test data compression ratios over those obtained by the Geometric-Primitives-Based compression technique.

Key Words

Testing, System-on-a-Chip, Test Compression, Run-length Coding

ملخص البحث

ان زيادة التعقيد في الشرائح الحاوية لأنظمة متكاملة والذي يتطلب زيادة حجم بيانات الاختبارات الخاصة بهذه الشرائح يجعل الحاجة الى تقليل حجم هذه البيانات أمرا مهما. في هذا البحث، نقدم طريقة جديدة وفعالة لضغط بيانات اختبارات الشرائح الحاوية لأنظمة متكاملة بدون فقد أي معلومات باستخدام الأشكال الهندسية الأساسية ثنائية الأبعاد. هذه الطريقة تعتمد على اعادة ترتيب متجهات الاختبار بهدف التقليل من الأشكال اللازمة لترميز البيانات. وبعد ترتيب المتجهات، تقسم البيانات الى قوالب متساوية الحجم ويطبق خوارزم الترميز على القوالب كلا على حدة. ولاختبار شريحة ما، تنقل البيانات مرمزة من جهاز الاختبار الى داخل الشريحة حيث يتم فك رموز البيانات. يمكن فك رموز البيانات باستخدام برنامج أو جهاز خاص. وفي كلتا الحلتين، يحتاج فك الرموز الى كمية من الذاكرة المؤقتة لتخزين قطعة من القوالب التي تم فك رموزها. في هذا البحث، تم تنفيذ كلتا الطريقتين السابقتين لفك رموز البيانات. نتائج التجارب على الدوائر القياسية (ISACS85 & ISCAS89) أظهرت فعالية الطريقة المقترحة في الحصول على نسبة ضغط عالية جدا. لقد تمكنت هذه الطريقة المقترحة من الحصول على نتائج أفضل بكثير مقارنة مع طرق أخرى لضغط بيانات الاختبار. ان شفرة طول الجري الموجّه بالتّكرار (FDR) شفرة متغيّر لمتغيّر على أساس تشفير جري ال٠ . في هذا العمل, نبّين أن ضغط بيانات اختبار أعلى يمكن أن يُحَقق على أساس تشفير كلا الجريين ال٠ و ١ . نقترح امتدادًا إلى شفرة FDR (EFDR) و نبّين بالنّتائج التّجريبيّة فاعليّتها في تحقيق نسبة انضغاط أعلى . في تكنيك الضّغط المبني على استخدام الأشكال الهندسية الأساسية ثنائية الأبعاد, بعض من القوالب تُشَفّر بتخزين بيانات الاختبار الحقيقيّة لأنّ حجم الكتلة المشفّر أكبر من حجم كتلة بيانات الاختبار الفعليّ . تقليل عدد هذه الكتل يمكن أن يتسبّب في ضغط بيانات اختبار أعلى. في هذا العمل, نقترح تكنيكات ضغط بيانات اختبار هجين التي تستغلّ الاستعمال لإمّا شفرات FDR أوEFDR لتقليل عدد الكتل التي تُشَفّر بتخزين بيانات الاختبار الحقيقيّة. بناء على النتائج التجريبيّة, نبّين فاعليّة تكنيكات ضغط الهجين المقترحة في زيادة نسب ضغط بيانات الاختبار على هؤلاء المحصولين بتكنيك الضّغط المبني على استخدام الأشكال الهندسية الأساسية ثنائية الأبعاد .

1. Introduction

With today's technology, it is possible to build complete systems containing millions of transistors on a single chip. Systems-on-a-chip (SOC) are comprised of a collection of 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 [1], [2]. To test a certain chip, the entire set of test vectors, for all the cores and components inside the chip, has to be stored in the tester memory. Then, during testing, the test data must be transferred to the chip under test and test responses collected from the chip to the tester as illustrated in Figure 1.

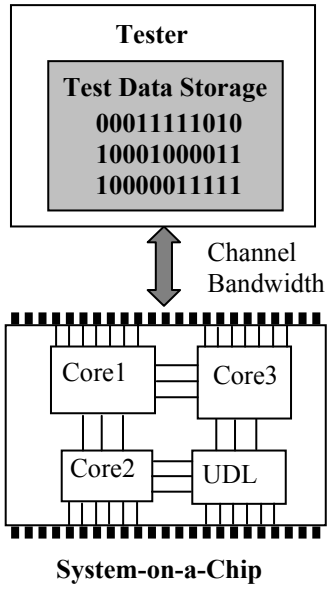


Figure 1 Test data transfer between the tester and the circuit under test.

One of the challenges in testing SOC is dealing with the large size of test data that must be stored in the tester and transferred between the tester and the chip. The amount of time required to test a chip depends on the size of test data that has to be transferred from the tester to the chip and the channel capacity.

The cost of automatic test equipment (ATE) increases significantly with the increase in their speed, channel capacity, and memory. As testers have limited speed, channel bandwidth, and memory, the need for test data reduction becomes imperative. To achieve such reduction, several compaction and lossless compression schemes were proposed in the literature.

The objective of test set compaction is to generate the minimum number of test vectors that achieve the desired fault coverage. There are two main types of compaction, static compaction and dynamic compaction. In static compaction, the number of test vectors is reduced after they have been generated. Examples of

static compaction algorithms include reverse order fault simulation [3], forced pair merging [4], N_by_M [5], and redundant vector elimination (RVE) [6]. In dynamic compaction, the number of test vectors is minimized during the automatic test pattern generation (ATPG) process. Examples of dynamic compaction algorithms include COMPACTEST [7], and bottleneck removal [8].

In test data compression, the objective is to reduce the number of bits needed to represent the test data. For test data compression, it is essential that the compression is lossless. Run length coding, Huffman codes, Lempel-Ziv algorithms, and arithmetic codes are examples of lossless compression [9].

Several test data compression/decompression techniques were proposed in the literature. These 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.

Examples of BIST-based compression techniques are test width compression [10], variable length reseeding [11], and Design For High Test Compression (DFHTC) [12].

Deterministic test compression techniques take advantage of the high correlation between test vectors. One of these techniques is proposed in [13] and uses Burrows-wheeler (BW) transformation and a modified version of run-length coding to encode the test data. This technique has been improved in [14] by applying the GZIP compression scheme to strings that are not effectively compressed by run-length coding. Another technique proposed in [15] uses what is called variable-to-block run-length coding. In this technique, a codeword is used to encode a block of data based on the number of zeros followed by a one in that block. This technique is used for compressing fully-specified test data that feeds a cyclical scan chain. A cyclical scan chain is used to decompress this data and transfer it to the "test scan chain". Golomb code is a variable-to-variable run-length code that is used in [16] 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. Another enhancement to the works done in [15] and [16] was proposed in [17]. It uses frequency-directed run-length (FDR) codes, which is another variable-to-variable coding technique. FDR code outperforms Golomb code based on the observation that the frequency of runs decreases with the increase in their lengths. Hence, assigning smaller codewords to runs with small lengths and larger codewords to those with larger lengths will decrease the overall cost. In [18], statistical coding is used for encoding deterministic test data. The technique uses a modified version of Huffman coding as to minimize the bits needed for codewords. Although this technique has less compression ratio than Huffman coding, the hardware implementation of the decoder is simpler. Another technique was proposed in [19] which performs decompression of test data based on an embedded processor. The technique is based on storing the differing bits between two test vectors. It divides each test vector into blocks and stores those blocks that are different from the preceding vector. The use of variable length input Huffman codes for SOC test data compression has been proposed in [20]. Techniques for

resuing scan chains from other cores in a SOC to increase the test data bandwidth have been described in [21], and automatic test patern generation techniques for producing test cubes that are suitable for encoding using the above technique have been described in [22]. A fault-simulation-based technique to reduce the entropy of the test vector set by pattern transformation is described in [23]. Such transformations increase the amount of compression that can be achieved on the transformed test set using statistical coding. ATPG algorithms for producing test vectors that can more effectively be compressed using statistical codes have been described in [24]. Several dictionary-based compression compression methods have recently been proposed to reduce SOC test data volume. A dictionary with fixed-length indices is used to generate all the distinct output vectors in [25]. A test data compression technique based on LZ77 and LZW methods are proposed in [26] and [27], respectively. The former uses a dynamic dictionary and the latter uses a memory in the on-chip decoder. A compression technique using dictionary with fixed-length indices is proposed in [28]. Commercial test data compression tools that can provide high compression for industrial designs have been recently proposed in [29-31].

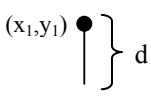
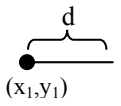
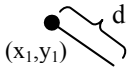
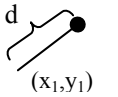
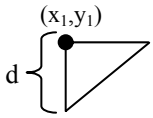
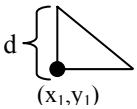
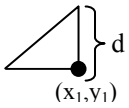
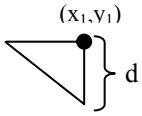
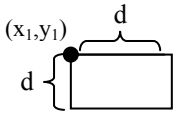
In this work, we introduce a novel compression scheme for deterministic testing of SOCs based on geometric shapes. This scheme is designed based on test cubes to maximize the compression ratio. Test vector decompression is performed on chip and is implemented either in hardware or software. For hardware decompression option, a decoding circuitry is placed on the chip to perform the decompression algorithm. However, for software decompression option, an embedded core is used to 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 the chip.

The techniques in [15-17] are all based on encoding only runs of 0's. This was motivated based on the idea that encoding the difference vectors instead of the actual test vectors may reduce the number of 1's in the encoded data. However, it was demonstrated in [17] that, in general, better test data compression results are achieved, based on both FDR and Golomb codes, by encoding the actual test vectors. Based on test data analysis, we have observed that the frequency of runs of 1's is as significant as runs of 0's, for many of the circuits. This suggests that encoding both runs of 0's and 1's could result in higher test data compression. In this work, we propose an extension to the FDR code to encode the test data based on encoding both types of runs. Furthermore, we propose hybrid test data compression techniques that exploit the use of either FDR or EFDR codes to reduce the number of blocks that are not encoded by the geometric shapes and encoded by storing the real test data. We demonstrate based on experimental results the effectiveness of the proposed hybrid compression technique in increasing the test data compression ratios over those obtained by the Geometric-Primitives-Based compression technique.

2. The Proposed Encoding Algorithm

The proposed encoding algorithm is based on encoding the 0's or the 1's in a test set by geometric shapes. In this work, we limited those primitive shapes to the basic four, namely: point, line, triangle, and rectangle as shown in Table 1. These shapes are the most frequently encountered shapes in any test set. For rectangles, a point and two distances are needed to encode the shape which costs $4*\log_2 N$, where N is the block dimension. However, lines and triangles can be represented by a point and a distance *d* and this reduces the number of bits needed to encode them by $(\log_2 N)-2$ in comparison to encoding them by two

Table 1 The used primitive geometric shapes.

	Type 1	Type 2	Type 3	Type 4
Point	(x_1,y_1) Code=00	X	X	X
Lines	 Code=0100	 Code=0101	 Code=0110	 Code=0111
Triangles	 Code=1000	 Code=1001	 Code=1010	 Code=1011
Rectangle	 Code=11	X	X	X

points. Two bits are used to determine the type of line or the type of triangle encoded.

Figure 2 shows the algorithm of the encoder, which consists of the following main steps:

2.1 Test Set Sorting

Sorting the vectors in a test set is crucial and has a significant impact on the compression ratio. In this step, we aim at generating clusters of either 0's or 1's in such a way that it may partially or totally be fitted in one or more of the geometric shapes shown in Table 1. Several sorting scenarios have been considered and investigated. In this work, we used a simple correlation-based sorting technique. The sorting may be

```
Encoder (N)
    Sort_Test_Set ();
    Partition_Test_Set (N);
    For i = 1 to # of segments
       For j = 1 to # of blocks in i
          Extract_Shapes (1, j);
          α1 = Encode_Shapes ();
          Extract_Shapes (0, j);
          α0 = Encode_Shapes ();
          B = # of bits in j + 2;
          E = min (α0, α1,B);
          Store_Encoded_Bits ();
          E_total += E;
End Encoder;

Extract_Shapes(b, j)
  For each bit x in block j {
    If x = b Then {
       Find the largest line of each type started at x
       Find the largest triangle of each type such that x is the vertix of the right angle
       Find the largest rectangle such tha x is its up-left corner
    }
  }
   Solve a covering problem to find the best group of shapes covering all bits b in block j.
End Extract_Shapes;
```

Figure 2 Test vectors encoding algorithm.

with respect to 0's (*0-sorting*), to 1's (*1-sorting*) or to both 0's and 1's (*0/1-sorting*). The technique is based on finding the distance D between two vectors A and B that maximizes the clusters of 0's and 1's.

The distance D may be computed with respect to 0's (*0-distance*), to 1's (*1-distance*) or to 0's and 1's (*0/1*-distance) as follows:

$$D(A, B) = \sum_{i=0}^{k-1} W(A_i, B_{i-1}) + W(A_i, B_i) + W(A_i, B_{i+1})$$

where k is the test vector length and $W(A_i, B_i)$ is the weight between bits A_i and B_i. Table 2, Table 3 and Table 4 specify the weights used in computing the 0-distance, the 1-distance, and the 0/1-distance between two vectors, respectively. Note that for i = 0, $W(A_i, B_{i-1}) = 0$ and for i = k-1, $W(A_i, B_{i+1}) = 0$.

The assignment of a 0.25 weight for an 'x' to each of its immediate neighbors be it an 'x' or the sorted bit ('0' for 0-sorting, '1' for 1-sorting, and '0' and '1' for 0/1-sorting) 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.

Table 2 Weights for the 0-distance between two test vectors.

	0	1	x
0	1.0	0.0	0.25
1	0.0	0.0	0.0
x	0.25	0.0	0.25

Table 3 Weights for the 1-distance between two test vectors.

	0	1	x
0	0.0	0.0	0.0
1	0.0	1.0	0.25
x	0.0	0.25	0.25

Table 4 Weights for the 0/1-distance between two test vectors.

	0	1	x
0	1.0	0.0	0.25
1	0.0	1.0	0.25
x	0.25	0.25	0.25

In Table 5, we show a simple example to illustrate the impact of sorting on test vector compression. As can be seen, sorting the vectors based on the 0-distance requires the encoding of two triangles to encode the 0's. However, sorting the vectors based on the 1-distance requires the encoding of one triangle and two lines to encode the 1's. Thus, for this example sorting based on the 0-distance results in higher compression.

The sorting algorithm requires $O(VT^2)$ time; where T is the number of vectors in the test set and V is the vector length.

2.2 Test Set 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 set 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 $\lceil C/N \rceil$. A *segment* consists of K blocks. In other words, the test set is segmented into L segments each contains K blocks. For test vectors whose columns and/or rows are not divisible by the predetermined block dimension 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 number of vectors, the vector length, and the block dimension, the number of bits used

to encode the coordinates of the geometric shapes can be less than $log_2 N$. The decoder recognizes those special cases and decodes them properly. The partitioning step requires constant time; i.e. it runs in O(1) time.

Table 5 An example of test vector sorting.

Original Vectors	*v1*	0	0	1	X	1	0	X	X
	v2	0	X	1	1	0	0	0	1
	v3	1	1	X	1	1	X	0	1
Sorted Vectors (0-dist.)	*v2*	0	X	1	1	0	0	0	1
	v1	0	0	1	X	1	0	X	X
	v3	1	1	X	1	1	X	0	1
Sorted Vectors (1-dist.)	*v3*	1	1	X	1	1	X	0	1
	v2	0	X	1	1	0	0	0	1
	v1	0	0	1	X	1	0	X	X

2.3 Encoding process

As mentioned earlier, the encoding process will be applied on each block independently. The procedure *Extract_Shapes(b)* will find the best group of shapes that cover the bits that are equal to *b* as shown in the algorithm. *Encode_Shapes* determines the number of bits, α, needed to encode this group of shapes. There are two cases that may occur:

(a) The block contains only 0's and x's or only 1's and x's. In this case, the block can be encoded as a rectangle. However, instead of this, it is encoded as "01" (indicating that the block can be filled by 0's or 1's) followed by the bit that fills the block. Hence, the number of bits to encode the block $\alpha = 3$. We call such blocks *filled blocks*.

(b) The block needs to be encoded by a number of shapes. We call such a block *encoded block*. In this case, we need the following:

- 2 bits to indicate the existence of shapes and the type of bit encoded. If the encoded bit is 0, then the code is 10, otherwise it is 11.
- $P = (2*\text{Log}_2 N - 3)$ bits to encode the number of shapes, S. If the number of shapes exceeds 2^P, then the number of bits needed to encode the shapes is certainly greater than the total number of bits in the block. In this case, the block is not encoded and the real data is stored. Therefore, selecting $N = 4$ or less is not effective in our technique because the maximum possible number of shapes in this case = $2^P = 2^{2*2-3} = 2^1 = 2$ shapes. Hence, we have experimented with 8x8, 16x16, and 32x32 block sizes.
- $\sum_{i=1}^{S} L_i$ bits; where L_i is computed as follows
  - If shape i is a point, $L_i = 2 + 2*\log_2 N$ (shape type + 2 coordinates).

- If shape i is a line or a triangle, $L_i = 2 + 2 + 3*\log_2 N$ (shape type + type of line or triangle + 2 coordinates + distance).
- If shape i is a rectangle, $L_i = 2 + 4*\log_2 N$ (shape type + 2 coordinates + 2 distances)

Therefore, $\alpha = 2 + P + \sum_{i=1}^{S} L_i$

For partial blocks, the encoder will output the needed bits and the decoder will take care of that. If α_0 (number of bits needed to encode shapes with 0) and α_1 (number of bits needed to encode shapes with 1) are greater than B which equals (N*N+2), then it is better not to encode the block. Instead, the real data is stored after a 2-bit code (00). We call such blocks *real-data blocks*. The procedure *Store_Encoded_Bits* will decide which case is the best (encoding 0's, encoding 1's, or storing the real data) based on E, which is the minimum of α_0, α_1, and B.

There are N^2 bits in a block; where N is the dimension of the block. Extracting each type of shape covering a bit requires $O(N^2)$ time at most (for example, the rectangle). Since we have constant number of shapes, the time complexity of extracting all shapes for each block is $O(N^4)$. Then, a covering step is performed to select the best group of shapes. The maximum number of shapes for any block (before selecting) is $10*N^2$; where 10 is the number of shape types. Therefore, this step requires $O(N^2)$. Hence, the encoding algorithm for each block requires $O(N^4)$ time. There are L*K blocks; where $L = \left\lceil \frac{T}{N} \right\rceil$ and $K = \left\lceil \frac{V}{N} \right\rceil$. Therefore, the total time complexity of the encoding algorithm is $O(LKN^4) = O(TVN^2)$. Since the maximum value of N in our algorithm is 32, then $N^2 = 1024$ at most, which means that N^2 is constant. Hence, the time complexity of the algorithm is O(TV), which means that the algorithm runs in linear time with respect to the size of the test set. The (N^2) term gives an indication that the time needed by the encoding algorithm increases with the increase in the block size.

3. Decoding Process

One of the main issues when designing a compression scheme for testing data is the implementation of the decompressor (or the decoder). The decoder of any compression scheme must be simple enough to achieve two requirements, minimizing the time needed for decompression and minimizing hardware overhead. Decoders of the compression schemes described in the literature can be classified into three main categories:

1. The scan chains available in the SOC are exploited to implement the decoder with possibly some additional logic.

2. An FSM is used to decompress the test data. Sometimes, additional hardware is needed.
3. If there exists an embedded processor in the SOC, a microcode is loaded to this processor and used to decode the compressed data.

In our work, we have implemented the decoder using both the second category, called the *hardware decoder,* and the third category, called the *software decoder.*

3.1 Software Decoder

Most of the SOC's have embedded processors and some amount of memory inside the chip. In this case, the decoder can be implemented as a microcode executed by the processor to output the original test vectors. In our scheme, some amount of temporary memory is needed to store the blocks one after the other until a whole segment is decoded. Then, the test vectors of that segment are applied to the scan chains in order.

Figure 3 shows the pseudo-code of the decoding algorithm. It first reads the arguments given by the encoder and computes the parameters needed for the decoding process. These parameters include the number of segments, the number of blocks in a segment and the dimensions of the partial blocks. In order to reconstruct the vectors, each segment has to be stored before sending its vectors to the circuit under test. For each segment, its blocks are decoded one at a time. The first two bits indicate the status of the block as follows:

- 00: the block is not encoded and the following N*N bits are the real test data.
- 01: fill the whole block with 0's or 1's depending on the following bit.
- 10: There are shapes that are filled with 0's.
- 11: There are shapes that are filled with 1's.

For those blocks that have shapes, the procedure *Decode_Shapes* is responsible for decoding these shapes. It reads the number of shapes in the block and then for each shape it reads its type and based on this it reads its parameters and fills it accordingly.

Based on the arguments read first, the decoder can determine the number of bits needed for each variable (e.g. the coordinates and the distances). These are used for the partial blocks when only one block of each segment remains and when the last segment is being decoded.

After all the blocks in a segment have been decoded, the segment is output to the circuit under test, vector by vector.

Similar to the complexity analysis for the encoding algorithm, we can conclude that the time required by the software decoder is O(VT). This means that it runs in linear time with respect to the test set size. It should be noted here that this algorithm is much simpler than the encoding algorithm because it does not

```
Decoder ()
   Read (N, # of segments (L), # of blocks  per segment (K),
row remainder (R), column remainder (C) );
    For i = 1 to # of segments {
     For j = 1 to # of blocks in i {
       b1b0 = Read_Bits (2);
       Case b1b0
           00 : Read_Bits (N* N);
           01 : b_type = Read_Bits (1);
                Fill_Block (j, b_type);
         10 : Decode_Shapes (0);
         11 : Decode_Shapes (1);
       End Case;
     }
    Output_Segment ();
  }
End Decoder;

Decode_Shapes (b)
   Num_Shapes = Read_Bits (2log2 N -3);
   For j = 1 to Num_Shapes
      Shape_type = Read_Bits (2);
      Case Shape_type
           00 : c = Get_Coordinate ();
                Fill_Point (b,c);
         01 : t = Get_Type ();
                c = Get_Coordinate ();
                d = Get_Distance ();
                Fill_Line(b,t,c,d);
         10 :  t = Get_Type ();
                 c = Get_Coordinate ();
                d = Get_Distance ();
                Fill_Triangle(b, t, c,d);
         11 : c = Get_Coordinate ();
                d1 = Get_Distance ();
                d2 = Get_Distance ();
                 Fill_Rectangle (b,c1,d1,d2);
End Decode_Shapes;
```

Figure 3 Test vectors decoding algorithm.

require extracting shapes; i.e. the (N^2) term found in the analysis of the encoding algorithm does not exist here.

3.2 Hardware Decoder

If there exists no embedded processor on chip, some additional hardware is needed for decoding the test data. The hardware decoder is implemented using an FSM controlling the data path which consists of some counters, registers and some basic gates. The hardware decoder has been designed and then modeled and verified using VHDL [32] .

3.2.1 Data Path Implementation

The data path is shown in Figure 4 and consists mainly of some registers and counters. The registers are:

- A shift register *I* is used to hold the input data before loading it to the corresponding register or counter. So, the size of this register is the maximum size of all registers and counters, which is 12 bits. A shift-left signal (*SHL I*) is used to shift a bit from the input data to the LSB of *I* and a clear signal (*CLR I*) is used to reset the register.
- Another shift register (*code*) is used to save the type of the shape that is currently decoded (point, line, …etc) and the type of that shape if it is line or triangle. The size of this register is 4 bits. Only one signal is needed to control this register which is (*SHL code*) that shifts a bit from the input data to the LSB of *code*.
- A 1-bit register (B) to save the bit with which the current block is filled. This FF can be loaded from either the input data or from I_0. So, a MUX is needed to select between these two inputs. The signals needed here are *Load B* and the select signal.
- A 2-bit register (*N*) is used to save the block dimension (8, 16 or 32). We need to get the actual size *N* from two bits given by the encoder as follows:
  - 00 ➔ *N* = 8 = 00111 (we start counting from 0).
  - 01 ➔ *N* = 16 = 01111.
  - 11 ➔ *N* = 32 = 11111.

  Let the two bits given by the encoder be I_1 and I_0 and the needed 5 bits be $N_4N_3N_2N_1N_0$. Then, we find that $N_4 = I_1$, $N_3 = I_0$, and $N_2 = N_1 = N_0 = 1$. The last three bits can be stored as wires connected to VDD. Therefore, the only hardware added here is a two-bit register connected directly to the least significant two bits of the input shift register *I* as shown in Figure 4. For this register, only a *Load* signal is required.
- Another two 5-bit registers are used to save the row remainder *R* and the column remainder *C*. These two registers will be loaded directly from the input register *I*. For these two registers, only a *Load* signal is required.

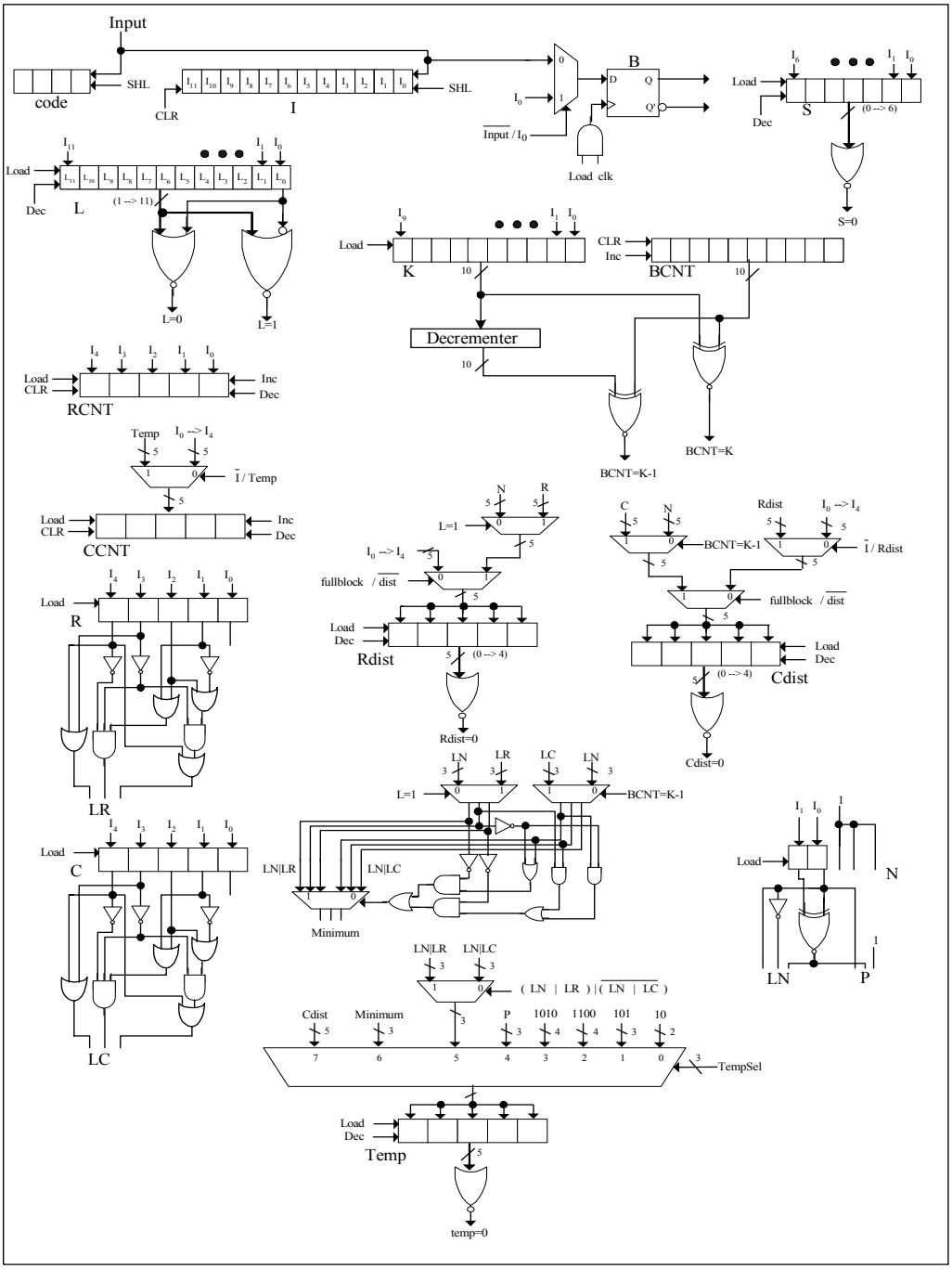


Figure 4 Data path implementation of the decoder.

- In order to know how many bits are to be read for each dimension (for the coordinates and the distances), $\log_2$ of the current dimension (*N, R,* or *C*) is required. In addition, we need to know how many bits are

needed to store the number of shapes. This number (P) depends on the dimension of the block N such that $P = 2*\log_2 N - 3$. All these can be obtained from the N, R and C registers using some combinational logic. We illustrate this as follows:

First, we want to get $\log_2 N$ (LN) as follows:

- N = 00111 ➔ LN = 011 ($\log_2 8 = 3$).
- N = 01111 ➔ LN = 100 ($\log_2 16 = 4$).
- N = 11111 ➔ LN = 101 ($\log_2 32 = 5$).

We can notice that $LN_2 = N_3$, $LN_1 = \overline{N_3}$ and $LN_0 = N_4$ XNOR N_3.

We can get the value of P from N as follows:

- 00111 (N = 8) ➔ 011 ($2*\log_2 8 - 3 = 3$).
- 01111 (N = 16) ➔ 101 ($2*\log_2 16 - 3 = 5$).
- 11111 (N = 32) ➔ 111 ($2*\log_2 32 - 3 = 7$).

Notice that $P_2 = N_3$, $P_1 = N_4$ XNOR N_3 and $P_0 = 1$.

For the partial blocks, the dimensions range between 1 and 31. We need to get $\log_2$ of these dimensions to know how many bits need to be read for the coordinates and distances in these blocks. Let the input (the dimension given by the encoder) be $A_4A_3A_2A_1A_0$ and the output (the bits needed) be $B_2B_1B_0$, then the following truth table is obtained:

A_4 A_3 A_2 A_1 A_0	$B_2B_1B_0$	
1 x x x x	1 0 1	5 bits needed for 16 to 31
0 1 x x x	1 0 0	4 bits needed for 8 to 15
0 0 1 x x	0 1 1	3 bits needed for 4 to 7
0 0 0 1 x	0 1 0	2 bits needed for 2 and 3
0 0 0 0 1	0 0 1	1 bit needed for 1

Using K-map technique, we get the following equations for B_2, B_1, and B_0:

- $B_2 = A_4 + A_3$.
- $B_1 = \overline{A_4} . \overline{A_3} . (A_2 + A_1)$.
- $B_0 = A_4 + \overline{A_3} . (A_2 + \overline{A_1})$.

- The last register needed in the data path is K which holds the number of blocks in a segment. In our implementation, we assume that the maximum vector length is 8K. Therefore, the maximum number of blocks in a segment is 1K blocks (when the dimension of a block = 8) and hence K is a 10-bit register.

Now, let us discuss the counters used in the data path:

(1) The first counter needed is *L* which is initially loaded with the number of segments in the test set. We assume that the maximum number of vectors in a test set is 32K vectors. Therefore, the maximum number of segments = 32K / 8 = 4K. So, the size of *L* = 12 bits. Whenever a segment is decoded, it is sent to the scan chains and *L* is decremented. When *L* = 0, the process is terminated. This condition can be checked by NORing all bits of *L*. Another condition that has to be checked is when *L* = 1 whereby the last segment is to be decoded. This can be checked also by NORing all bits of *L* with inverting L_0. The signals that we need here are *Load* and *Dec* (Decrement).

(2) Another counter (*BCNT*) is required to keep track of the block number within the current segment. The size of this counter equals the size of register *K* which is 10 bits. This counter must start counting from 0 because it is used for addressing the memory (as will be explained shortly). Therefore, we did not use this counter as in the case of the number of segments *L*. Instead, we added some comparators to check for the last block and to check if all blocks in a segment have been decoded. For each segment, *BCNT* is cleared first and then incremented for every block decoded until it equals *K*, which means that all blocks in the current segment have been decoded. This condition can be checked by XNORing every bit of *BCNT* with the corresponding bit of *K* then ANDing the results. To know when the last block of the current segment is to be decoded, *BCNT* is compared with *K*-1, which is obtained by decrementing the content of *K* and XNORing the result with *BCNT*. The signals needed to control *BCNT* are *CLR* (Clear) and *Inc* (Increment).

(3) In each block, there may be some shapes encoded. To know how many shapes are in the block, a counter *S* is used. The size of this counter = 7 bits ($2*\log_2 32 - 3$) which is the maximum possible for all block sizes. For each block that has shapes, *S* is loaded with the number of shapes. Whenever a shape is decoded, *S* is decremented until it reaches 0. Also here we need to check for 0 (similar to *L*). The signals needed are *Load* and *Dec.*

(4) Four 5-bit counters are used for decoding shapes and writing them to memory. These are *RCNT, CCNT, Rdist* and *Cdist. RCNT* and *CCNT* are used to address the bit to be written within the current block in the form (row, column), respectively. They are loaded with the coordinate of a shape and then incremented or decremented according to the direction of writing. *Rdist* and *Cdist* are used for the length of writing in each direction. They are loaded with the distance and then decremented until they reach 0. Hence, a check for 0 is needed for each. The loading can be from *N, R,* or *I* for *Rdist* and from *N, C, I* or *Rdist* for *Cdist.* This depends on the block number, on whether a full block is to be filled or only a portion of it and on the type of the shape (line, triangle, or rectangle). For *RCNT* and *CCNT*, the signals needed are *Load, CLR, Inc* and *Dec.* For *Rdist* and *Cdist*, the signals are *Load, Dec* and the select signals.

(5) The last counter is a temporary counter (*temp*) that is used mainly to decide the number of bits to be read from input data. Since there are many cases, the value that is loaded to *temp* must be selected depending on the parameter to be read. The values *LN|LR, LN|LC,* and *Minimum* are used to select between full blocks and partial blocks. Note that *Minimum* is used to determine the minimum number of bits needed to encode a coordinate or a distance. This is used in the case of a partial block. Each value is selected in a certain case as shown in Figure 5. For reusing resources, *temp* is used as a temporary register in the case of decoding a triangle. In this case, it is loaded from *Cdist.* In all cases, we need to know when *temp* = 0 to stop reading data. So, a check for 0 is required. The signals required to control *temp* are *Load, Dec* and the select signals.

For hardware implementation as well as for software implementation, some amount of memory is required to store a segment before applying its vectors to the CUT. The size of this memory is equal to the size of the scan chain times the number of vectors per segment, which is in our case equal to 32 as maximum. For the hardware decoder to be simple and fast, we need to address this memory bit-wise. This can be achieved by dividing the address into three fields:

1) Block #: this specifies the block to be decoded among the blocks of the current segment. The size of this field = $\lceil \log_2 K \rceil$; where K = the number of blocks per segment.
2) Row #: this indicates the row of the current block. The size of this field = $\log_2 N = 5$ as maximum (when N=32).
3) Column #: this indicates the column of the current block. The size of this field = $\log_2 N = 5$ as maximum (when N=32).

The three counters *BCNT, RCNT*, and *CCNT* are used to decide the address of the bit to be written.
As we mentioned before, the maximum vector length is assumed to be 8K. Therefore, the maximum memory size required = 32*8K = 256 Kbit. This needs an address of 18 bits. Since we have 5+5+10 = 20 bits in the three counters, we need to select the bits to represent the address in each case (*N*=8, 16, or 32) using multiplexers. The outputs of the multiplexers are connected directly to the memory address bus.

3.2.2 Implementation of the FSM

The FSM controlling the decoding process is shown in Figure 5. It consists of 62 states, which means that 6 FFs plus some combinational logic are enough to implement it. This FSM is designed to decode the whole test set, not only one segment or one block. The FSM can be summarized in the following:

- The decoding process is activated at state S_0 when a starting signal *start* = 1.

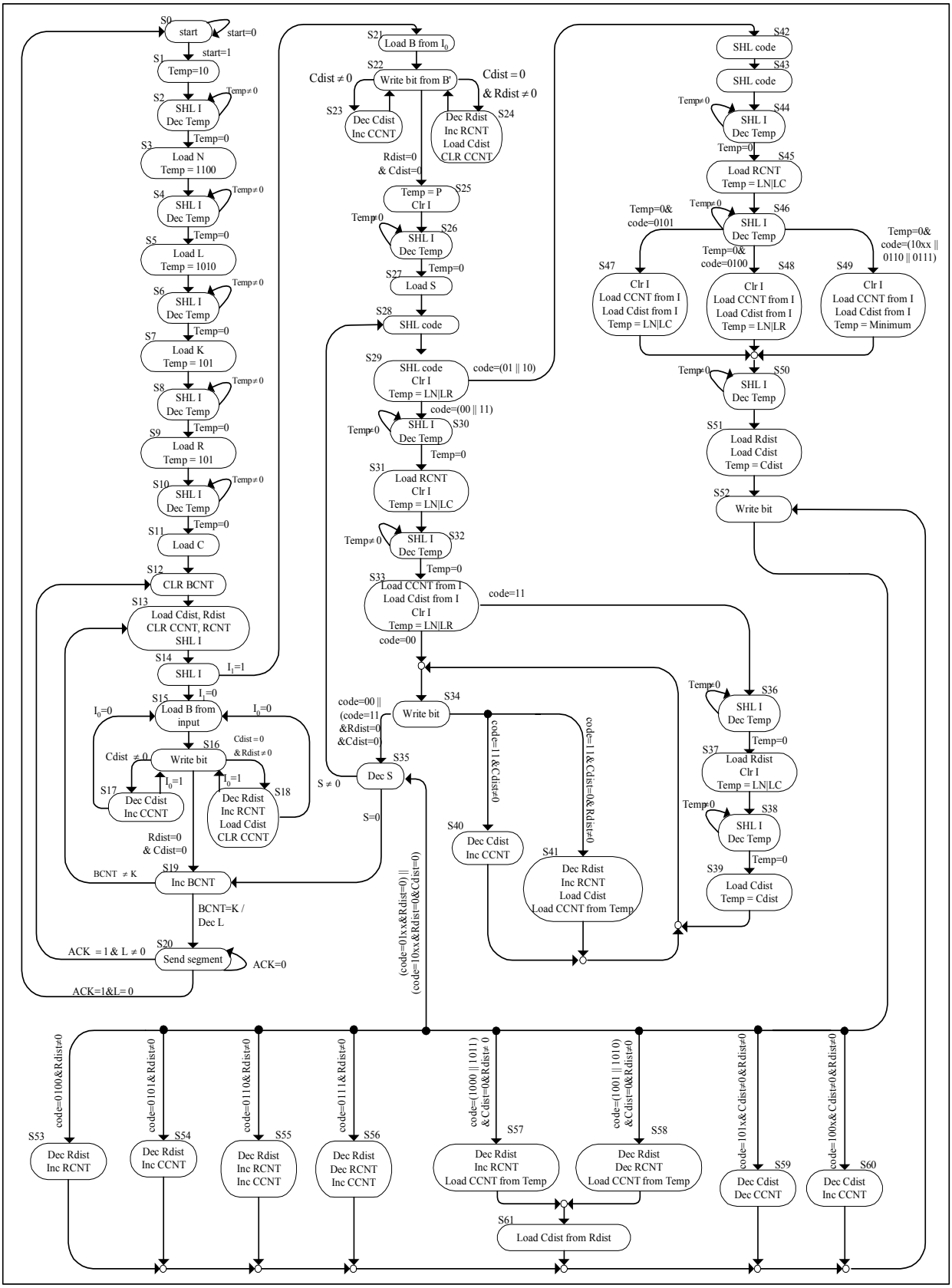


Figure 5 The FSM of the decoder.

- From S_1 to S_{11}, the five parameters (block dimension (N), # of segments (L), # of blocks per segment (K), row remainder (R), and column remainder (C)) are read and loaded to the appropriate counters and registers. Then, the counters used for addressing the memory are initialized in states S_{12} and S_{13} and this is done for each block.
- In S_{14}, there are two possibilities:
  (i) There are no shapes to be decoded: in this case the whole block will be filled with either 1 bit (0 or 1) or filled with real data. In the former case, S_{15} is visited only once to initialize the bit with which the block is filled and then the process goes through states S_{16}, S_{17} and S_{18}. In the latter case, S_{15} is visited for each bit read.
  (ii) There are shapes to be decoded: in this case the process goes to state S_{21}.
- States S_{21} to S_{24} initialize the block with the complement of the bit with which all shapes are encoded. This is important to make sure that all bits in the block have the correct values. When the shapes are decoded, the corresponding bits will be overwritten.
- The number of shapes is read in states S_{25} to S_{27}.

For each shape, S_{28} is visited to read the first bit in that shape.

- The type of the shape is determined in S_{29}. If it is a point or a rectangle, the process continues in S_{30}. Otherwise, it goes to S_{42}.
- States S_{30} to S_{33} read the coordinate of the point and the rectangle shapes. If the shape is a point, it is written in S_{34}. If, on the other hand, it is a rectangle, the process goes to states S_{36} to S_{39} to initialize the counters and then goes through S_{34}, S_{40} and S_{41} until the whole rectangle is written.
- If the shape is a line or a triangle, the process goes through states S_{42} to S_{51} to initialize the counters and determine the type of the shape. Then according to the type of the shape and the status of the counters, the process goes to one of the states S_{53} to S_{60}. Then, the process repeats until the shape is written.
- After every shape is decoded, the number of shapes is decremented in S_{35}. If there are other shapes, the process goes back to S_{28}. Otherwise, it goes to S_{19} in which the number of blocks (*BCNT*) is incremented. If there are still other blocks, the process goes to S_{13}; otherwise it goes to S_{20} if all blocks of the current segment have been decoded.
- In S_{20}, the segment just decoded is sent to the scan chains and the process waits for an acknowledgment to proceed. If there are other segments, the process goes to S_{12}. Otherwise, the process is terminated and goes back to the initial state. This is the only case where a mealy output is required. Therefore, we can say that our FSM is almost Moore.

3.3 Decoder Interface

In this section, we outline the interface between the decoder and the tester and between the decoder and the scan chains. First, we discuss the interface of the software decoder and then we discuss the interface of the hardware decoder.

3.3.1 Interface of the software decoder

Figure 6 shows the interface between the software decoder and the tester. The decoding program is stored in a ROM on chip. When the tester starts sending the encoded data to the processor, the processor reads the instructions from the ROM and executes them in order to decode the test data. Then, it writes the decoded data to the memory. After a whole segment is decoded, the processor will send a signal to the controller to start applying the test vectors to the scan chains. It should be stated here that there must be some synchronization mechanism between the processor and the tester in order to avoid overflow.

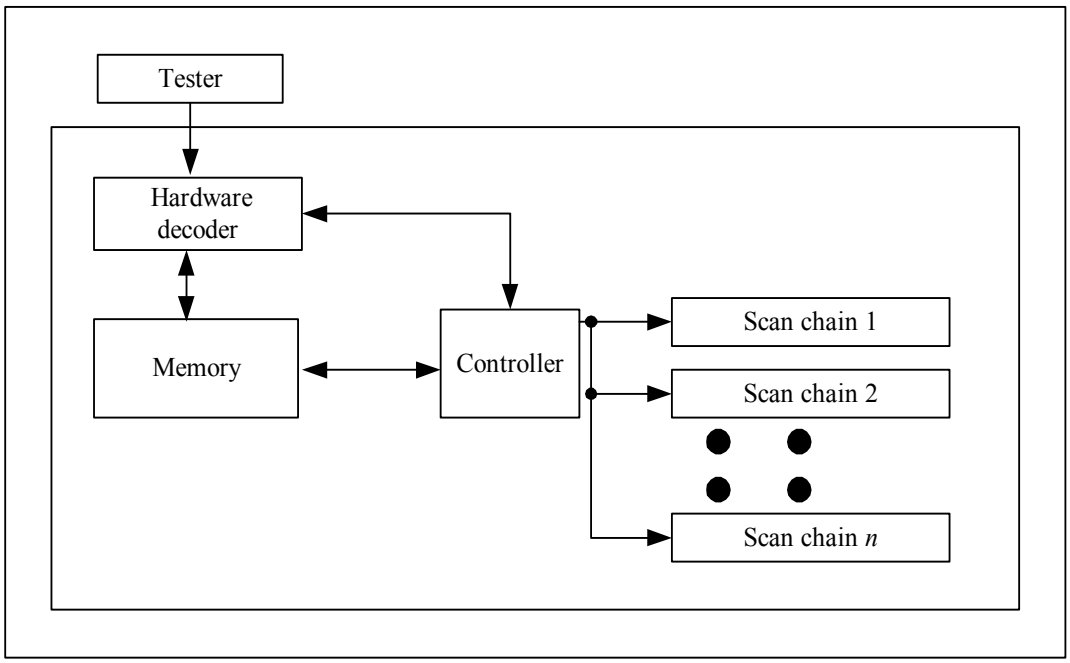


Figure 6 Interface of the software decoder.

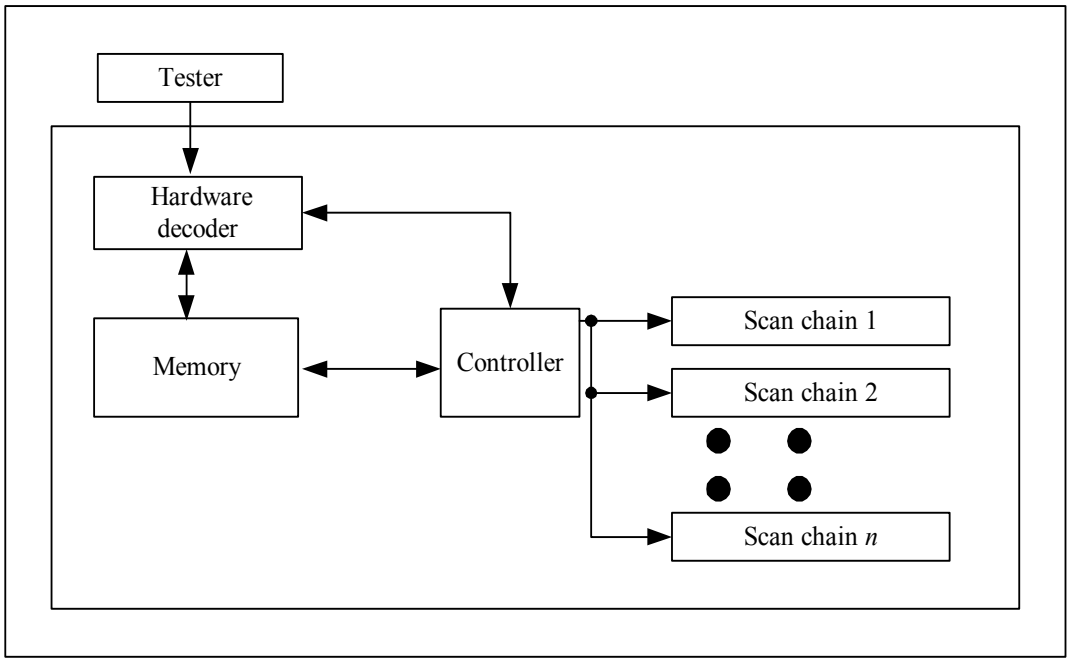


Figure 7 Interface of the hardware decoder.

3.3.2 Interface of the hardware decoder

Similar to the interface explained above for the software decoder, the hardware decoder can be interfaced to the tester in place of the processor. This is shown in Figure 7. Here, the hardware decoder reads the encoded data from the tester and writes the decoded data to the memory. After decoding a complete segment, the decoder sends a signal to the controller to apply the test vectors and waits for the acknowledgement to start decoding another segment. Also here, we need some synchronization mechanism between the tester and the decoder.

4. Frequency-Directed Run-Length (FDR) Code

Many of the test data compression techniques are based on run-length coding. A run is a consecutive sequence of equal symbols. A sequence of symbols can be encoded using two elements for each run; the repeating symbol and the number of times it appears in the run. Frequency-directed run-length (FDR) code is a variable-to-variable coding technique based on encoding runs of 0's. In FDR code, the prefix and the tail of any codeword are of equal size. In any group A_i, the prefix is of size i bits. The prefix of a group is the binary representation of the run length of the first member of that group. When moving from group A_i to group A_{i+1}, the length of the code words increases by two bits, one for the prefix and one for the tail.

Runs of length i are mapped to group A_j, where $j = \lceil \log_2(i+3) \rceil - 1$. The size of the i'th group is equal to 2^i, i.e., group A_i contains 2^i members. The FDR code for the first three groups is shown in Table 6.

Table 6 FDR code.

Group	Run Length	Group Prefix	Tail	Code Word
A1	0	0	0	00
	1		1	01
A2	2	10	00	1000
	3		01	1001
	4		10	1010
	5		11	1011
A3	6	110	000	110000
	7		001	110001
	8		010	110010
	9		011	110011
	10		100	110100
	11		101	110101
	12		110	110110
	13		111	110111

4.1 Test Data Analysis

Based on test data analysis, it has been observed that test sets contain a large number of runs of 1's in addition to runs of 0's. By considering both types of runs, the total number of runs will decrease, which could result in higher test data compression.

To support this observation, we have analyzed test data for the largest ISCAS 85 and full-scanned versions of ISCAS 89 circuits. We have used the test sets generated by MinTest [6], using both static and dynamic compaction. Test sets generated by dynamic compaction option have the letter *d* appended in their name. All the test sets used achieve 100% fault coverage of the detectable faults in each circuit. Test sets generated based on static compaction were relaxed, as this has the advantage of keeping unnecessary assignments as X's, which enables higher compression.

Given a relaxed test set, techniques based on encoding only runs of 0's fill all the X's by 0's to reduce the number of runs that need to be encoded. However, to encode both runs of 0's and 1's in a test set, X's are filled by 1's if they are bounded by 1's from both sides, otherwise they are filled by 0's. This results in a reduction in the total number of runs that need to be encoded.

Table 7 shows the analysis of the number of runs on the used test sets. The first column indicates the circuit name. The second column shows the number of runs of 0's in the test set assuming that only runs of 0's will be encoded. The third, fourth, and fifth columns indicate the number of runs of 0's, runs of 1's, and the total number of runs, respectively, assuming that both types of runs will be encoded. As can be seen from the table, for most of the circuits, the number of runs of 1's is as significant as the number of runs of 0's. For all the circuits, the total number of runs decreases and for some circuits the reduction is significant.

Table 7 Analysis of number of runs in test data.

		Encoding 0 Runs	Encoding 0 and 1 Runs		
Circuit	Original Bits	0 Runs	0 Runs	1 Runs	Total Runs
c2670	10252	1677	505	414	919
c5315	6586	1628	561	454	1015
c7552	15111	2695	652	1111	1763
s13207	163100	4804	2615	1157	3772
s15850	57434	4635	2514	1106	3620
s35932	21156	7554	1236	1071	2307
s38417	113152	20970	5331	3761	9092
s5378	20758	2915	1072	806	1878
s9234	25939	3843	1770	980	2750
s13207d	165200	5021	2581	1210	3791
s15850d	76986	5329	2644	1202	3846
s35932d	28208	10018	235	346	581
s38417d	164736	29473	5773	4834	10607
s38584d	199104	16814	7585	4074	11659
s5378d	23754	3537	1237	1001	2238
s9234d	39273	4816	2347	1212	3559

Figures 8, 9, and 10 show the frequency of both runs of 0's and runs of 1's for test sets of the circuits: s15850, s9234, and s35932d, respectively. As can be seen from the figures, the frequency of runs of 1's follow a similar shape to that of runs of 0's, although with a smaller magnitude. For the circuit in Figure 8, it can be observed that there are more runs of 1's than 0's for run length < 5, but for run length > 5 there are more runs of 0's. For the circuit in Figure 9, we can see that runs of 0's with any length are on the average more that the runs of 1's with the same length. For the circuit in Figure 10, it can be observed that runs of 1's of small and large run length are more than those of 0's. But for middle run length ranges, the number of both 0 and 1 runs is comparable.

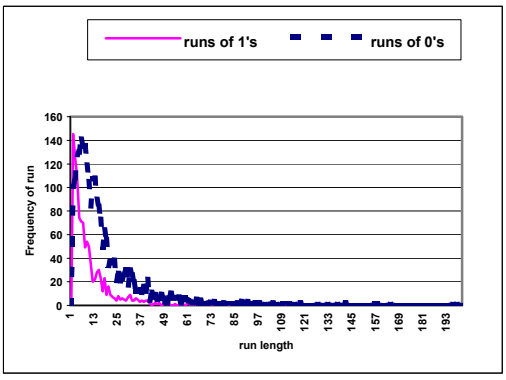


Figure 8 Distribution of runs of 0's and 1's for circuit s15850.

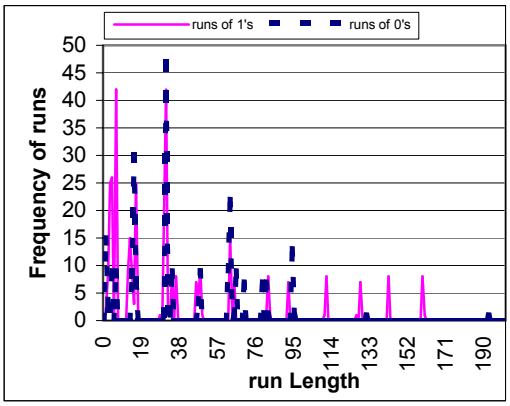


Figure 9 Distribution of runs of 0's and 1's for circuit s35932d.

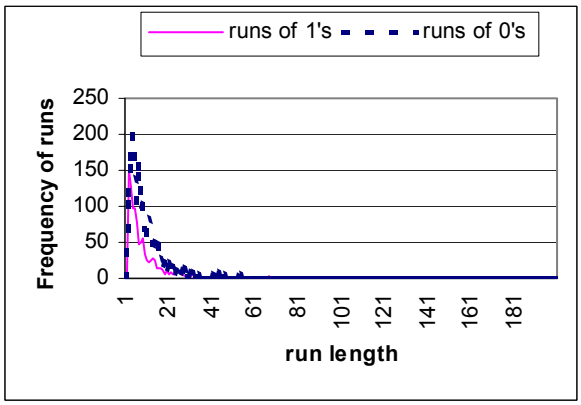


Figure 10 Distribution of runs of 0's and 1's for circuit s9234.

5. Extended FDR (EFDR) Code

To encode both runs of 0's and 1's, we extend the FDR code based on adding an extra bit to the beginning of a code word to indicate the type of run. If the bit is 0, this indicates that the code word is encoding a run of type 0, otherwise it encodes a run of type 1. This code, called Extended FDR (EFDR), is shown in Table 8. It should be observed that this code is a direct extention to the FDR code shown in Table 6. However, in this code we do not have run length of size 0. This is because we are encoding both runs of 0's and runs of 1's. Note that runs of 0's are strings of 0's followed by a 1, while runs of 1's are strings of 1's followed by a 0, i.e. runs of 1's of length i are the complement of runs of 0's of the same length, and vice versa. As with FDR code, in this code when moving from group A_i to group A_{i+1}, the length of code words increases by two bits, one for the prefix and one for the tail. Runs of length i are mapped to group A_j, where $j = \lceil \log_2(i+2) \rceil - 1$. The size of the i'th group is equal to 2^{i+1}, i.e., group A_i contains 2^{i+1} members.

Table 8 Extended FDR (EFDR) code.

Group	Run Length	Group Prefix	Tail	Code Word Runs of 0's	Code Word Runs of 1's
A1	1	0	0	000	100
	2		1	001	101
A2	3	10	00	01000	11000
	4		01	01001	11001
	5		10	01010	11010
	6		11	01011	11011
A3	7	110	000	0110000	1110000
	8		001	0110001	1110001
	9		010	0110010	1110010
	10		011	0110011	1110011
	11		100	0110100	1110100
	12		101	0110101	1110101
	13		110	0110110	1110110
	14		111	0110111	1110111

To illustrate the use of this code, let us consider an example. Consider the test T={01100011111111000000001}, of size 22 bits. The number of 0 runs in this test is 10. However, the number of both 0 and 1 runs is 5. Encoding this test using FDR code results in the encoded test T_{FDR}={01 00 1001 00 00 00 00 00 00 110010} of size 26 bits. Thus, for this example the number of bits needed to encode the test data using FDR code is more than the actual size of the original test data. However, encoding this test using EFDR code, we obtain the encoded test T_{EFDR}={000 100 001 11011 0110000}, of size 21 bits. Obviously, for this example EFDR code outperforms FDR code. Note that FDR code suffers whenever we have runs of 1's, as each 1 bit will be encoded by a separate 0 run of length 0.

6. Hybrid Test Compression Scheme

As it was mentioned before, in the Geometric-Primitives-Based compression technique there are some blocks which are encoded by storing the real test data. This is because the size of these blocks when they are encoded is larger than thier original size. So, no compression is achieved for such blocks. In order to reduce the number of these blocks, we propose to combine the Geometric-Primitives-Based compression technique with either the FDR or the EFDR compression techniques. In this case, the FDR or EFDR techniques are applied to encode a block. The block is encoded with these techniques if its encoding size is less than the encoding size with geometric shapes. The block encoding format for the hybrid technique combining the geometric and FDR compression techniques, called GFDR, is shown in Table 9. Note that the difference between this encoding scheme and the Geometric encoding scheme is in the header code starting with 00. So, blocks that will still be encoded with real test data will have an extra bit in the header. The other blocks have exactly the same format. The block encoding format for the hybrid technique combining the

Table 9 Geometric-FDR (GFDR) & Geometric-EFDR (GEFDR) compression block encoding format.

Header Code	Encode Block
000	with real test data
001	with FDR (EFDR) codes
010	as filled with 0's
011	as filled with 1's
10	with geometric shapes covering 0's
11	with geometric shapes covering 1's

geometric and EFDR compression techniques, called GEFDR, is similar to GFDR with the difference of using EFDR instead of FDR.

Test data decompression will be done on chip and the decoded test will then be applied to the chip under test. The decoders for the proposed hybrid techniques are a direct combination of the decoders for the Geometric, FDR[17], and EFDR techniques.

6. Experimental Results

In order to demonstrate the effectiveness of our scheme, we have performed experiments on a number of the largest ISCAS85 and full-scanned versions of ISCAS89 benchmark circuits. The experiments were run on a Pentium II processor with a speed of 350 MHz and a 32 Mbyte RAM. We have used the test sets generated by MinTest [6], which are highly compacted test sets, that achieve 100% fault coverage of the detectable faults in each circuit. Test cubes were generated from each test set as this has the advantage of keeping unnecessary assignments as x's, which enables higher compression. Then, the test vectors were sorted to maximize the compression. In this work, test vectors were sorted based on a greedy algorithm. Test vectors sorting based on the 0-distance, the 1-distance, and the 0/1-distance was performed. For both the 0-distance and 0/1-distance sorting, the test vector with more 0's was selected as the first vector. However, for the 1-distance sorting, the vector with more 1's was selected as the first vector. Table 10 shows the compression ratio of the three sorting criteria on the 8x8 block size. As shown in the table, the 0/1 sorting gives the best results most of the time. Therefore, we used this sorting criterion for our experiments.

The *compression ratio* is computed as:

$$Comp.Ratio = \frac{\#Original\ Bits - \#Compressed\ Bits}{\#Original\ Bits} \quad X100$$

Table 10 Compression results of the proposed scheme for different sorting criteria.

Circuit	1-sorting	0-sorting	0/1-sorting
s13207	84.952	84.724	**85.561**
s15850	69.646	69.782	**70.188**
s35932	65.177	**65.889**	62.231
s38417	61.84	61.677	**62.226**
s38584	65.203	65.186	**65.594**
s5378	55.805	55.658	**57.94**
s9234	54.989	55.921	**57.22**

Table 11 Compression results of the proposed scheme for various block sizes.

Circuit	Scan Size	No. Vec	8x8	16x16	32x32	CPU (sec)
s13207	700	236	85.561	**86.628**	85.316	29
s15850	611	126	**70.188**	69.253	65.776	10
s35932	1763	16	62.231	74.688	**78.123**	5
s38417	1664	99	**62.226**	59.304	54.245	54
s38584	1464	136	**65.594**	65.085	61.13	33
s5378	214	111	**57.94**	52.854	48.657	4
s9234	247	159	**57.22**	55.789	52.148	14

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 11. The second column in the table shows the scan size, which is basically the width of a test vector. The third column indicates the number of test vectors in the test set. As can be seen, the effectiveness of the proposed encoding algorithm is clearly demonstrated as high compression ratio was obtained for all the circuits. A block size of 8x8 achieves the best results for most of the circuits. The last column in Table 11 shows the total CPU time used for compressing the test vectors based on the three block sizes, i.e. the total CPU time used to produce the best result, which is highlighted in the table. Based on the compression results in Table 11, our technique achieves an average compression ratio of around **68%** based on highly compacted tests.

In order to demonstrate the effectiveness of our technique, we compare it with the techniques proposed in [16] and [17]. The comparison is performed based on the test sets generated using the dynamic compaction option of MinTest [6]. Table 12 shows the comparison among the three techniques. As can be seen from the table, for all the compared circuits, our technique achieves significantly higher compression ratio than the other techniques.

Next, we compare the compression ratio obtained for two test sets with different sizes. The two test sets shown in Table 13 have different sizes. The first set test is generated using the dynamic compaction option

Table 12 Comparison with the techniques of [17] (FDR) and of [16] (Golomb).

Circuit	Geometric	FDR	Golomb
s5378f	57.94	48.02	37.11
s9234f	57.22	43.59	45.25
s13207f	86.628	81.3	79.74
s15850f	70.188	66.23	62.82
s35932f	78.123	19.37	0
s38417f	62.226	43.26	28.37
s38584f	65.594	60.91	57.17
average	68.27414	51.81143	44.35143

Table 13 Compression results of the proposed scheme for different test sizes.

	Test set 1			Test set 2		
Circuit	Original Bits	Comp. Ratio	Comp. Bits	Original Bits	Comp. Ratio	Comp. Bits
s5378	23754	57.94	**9991**	20758	51.551	**10057**
s9234	39273	57.22	16801	25935	43.451	14666
s13207	165200	86.628	**22091**	163100	85.012	**24445**
s15850	76986	70.188	22952	57434	60.32	22790
s35932	28208	78.123	**6171**	21156	25.78	**15702**
s38417	164736	62.226	62228	113152	46.497	60540
s38584	199104	65.594	68504	161040	65.944	54844

of MinTest [6] while the second is generated using the static compaction option of MinTest [6]. The first set is larger in size than the second. The effect of the test set size cannot be shown if we consider only the compression ratio. However, if we look at the number of bits after compression, we can see that some of the circuits ended up with a smaller number of bits although the size of the original test set is larger. These circuits are shaded in Table 13. From this observation, we can conclude that adding some redundancy to the test set may help in achieving higher compression.

We next show some statistics on the type of block encoding. As explained before, there are three possibilities for encoding a block. The first is to encode the block as filled by either 0's or 1's. The second is to encode the block using geometric shapes. The third is to store the real data if the number of bits needed to encode the block is greater than the actual number of bits in that block. We call the first type of blocks *filled blocks*, the second type of blocks *encoded blocks* and the last type of blocks *real-data blocks*. The cost of each filled block is only 3 bits, while the cost of each real-data block is the size of the block + 2. The cost of an encoded block depends on the shapes in that block.

Table 14, Table 15, and Table 16 show the percentage of these types of blocks for the benchmark circuits used for block size 8x8, 16x16, and 32x32, respectively. From these tables, we can notice the following:

Table 14 Statistics on block encoding (8x8 blocks).

	8x8						
circuit	total # of blocks	# of real blocks	%	# of filled blocks	%	# of encoded blocks	%
s13207f	2640	68	2.5758	2041	77.31061	531	20.113636
s15850f	1232	82	6.6558	614	49.83766	536	43.506494
s35932f	442	2	0.4525	56	12.66968	384	86.877828
s38417f	2704	189	6.9896	1068	39.49704	1447	53.513314
s38584f	3111	347	11.154	1180	37.92993	1584	50.916104
s5378f	378	78	20.635	143	37.83069	157	41.534392
s9234f	620	60	9.6774	150	24.19355	410	66.129032
average			8.3057		39.89559		51.798686

Table 15 Statistics on block encoding (16x16 blocks).

	16x16						
circuit	total # of blocks	# of real blocks	%	# of filled blocks	%	# of encoded blocks	%
s13207f	660	22	3.3333	356	53.93939	282	42.727273
s15850f	312	27	8.6538	75	24.03846	210	67.307692
s35932f	111	1	0.9009	0	0	110	99.099099
s38417f	728	41	5.6319	162	22.25275	525	72.115385
s38584f	828	98	11.836	145	17.51208	585	70.652174
s5378f	98	17	17.347	13	13.26531	68	69.387755
s9234f	160	17	10.625	7	4.375	136	85
average			8.3325		19.34043		72.327054

Table 16 Statistics on block encoding (32x32 blocks).

	32x32						
circuit	total # of blocks	# of real blocks	%	# of filled blocks	%	# of encoded blocks	%
s13207f	176	0	0	44	25	132	75
s15850f	80	6	7.5	3	3.75	71	88.75
s35932f	56	1	1.7857	0	0	55	98.214286
s38417f	208	2	0.9615	31	14.90385	175	84.134615
s38584f	230	44	19.13	21	9.130435	165	71.73913
s5378f	28	5	17.857	2	7.142857	21	75
s9234f	40	7	17.5	0	0	33	82.5
average			9.2478		8.56102		82.191147

1) The percentage of filled blocks decreases with the increase in block size while the percentage of real-data blocks does not change much. This shows why the 8x8 block size gives the best

results most of the time followed by the 16x16 block size. From this point, we can notice the advantage of partitioning the test set into blocks.

2) Some of the circuits have high percentage of real-data blocks. This shows that there is a room for improvement if these blocks are encoded using another compression scheme.

All the compressed test sets were decoded and verified by fault simulation. The decoding algorithm is very fast and the decoding time for each test set was in fractions of a second. We have modeled the hardware decoder using VHDL and based on simulation counted the clock cycles needed to complete decoding each circuit. If we assume a certain clock rate, then we can find the time required by the decoder by dividing the number of clock cycles by the clock rate. Table 17 shows the results for a clock rate of 500 MHz and for different block sizes. The time given in the table is in μ seconds. We should indicate here that this timing is for the decoding process only and does not include the test application time.

Table 17 Timing performance of the hardware decoder.

	8x8		16x16		32x32	
circuit	clock cycles	time (μs)	clock cycles	time (μs)	clock cycles	time (μs)
s13207f	366506	733.012	373039	746.078	406249	812.498
s15850f	191193	382.386	200540	401.08	220739	441.478
s35932f	83120	166.24	80880	161.76	77967	155.934
s38417f	438157	876.314	464892	929.784	516920	1033.84
s38584f	509046	1018.092	542300	1084.6	562884	1125.768
s5378f	61748	123.496	67458	134.916	72380	144.76
s9234f	107482	214.964	113818	227.636	118152	236.304
average	251036	502.072	263275.286	526.55057	282184.4286	564.36886

Based on the results given in Table 17 and Table 11, we can notice that the number of clock cycles needed to decode a test set increases with the decrease in the compression ratio. The only exception in this trend is in the case of circuit s13207f, where the highest compression ratio is for the 16x16 block size while the smallest number of clock cycles is for the 8x8 block size. The reason for this is that the compression ratios for the two block sizes are very close to each other while the percentage of real-data blocks is higher for the case of 16x16 block size. Since the real-data blocks need more time for decoding (because they require more reading cycles), the number of clock cycles increases with the increase in the percentage of real-data blocks.

Table 18 compares the compression results using the FDR and EFDR code. The first column shows the circuit name and the second column shows the size of the test set in bits. The third and fourth columns show the number of compressed bits using FDR and EFDR codes, respectively. The last two columns indicate the respective compression ratios. As can be seen from the table, significant improvements in the compression

Table 18 Compression results of FDR & EFDR.

Circuit	Original Bits	FDR Bits	EFDR Bits	FDR CR	EFDR CR
c2670	10252	5760	4807	43.82	**53.11**
c5315	6586	5238	4700	20.47	**28.64**
c7552	15111	9500	8843	37.13	**41.48**
s13207	163100	34608	33637	78.78	**79.38**
s15850	57434	24992	25105	**56.49**	56.29
s35932	21156	20312	11502	3.99	**45.63**
s38417	113152	70536	53914	37.66	**52.35**
s5378	20758	11032	10210	46.85	**50.81**
s9234	25939	16912	16127	34.80	**37.83**
s13207d	165200	30880	29992	81.31	**81.85**
s15850d	76986	26016	24643	66.21	**67.99**
s35932d	28208	22746	5554	19.36	**80.31**
s38417d	164736	93452	64962	43.27	**60.57**
s38584d	199104	77798	73853	60.93	**62.91**
s5378d	23754	12356	11419	47.98	**51.93**
s9234d	39273	22148	21250	43.61	**45.89**

ratio are obtained for some of the circuits. Consider for example the circuit s35932. For the first test set of this circuit, the compression ratio improves from 3.99% using FDR to 45.63% using EFDR code. For the second test set of the same circuit, the compression ratio increases from 19.36% using FDR to 80.31% using EFDR code. This result is not surprising as based on the statistics for this circuit given in Table 7, the total number of runs reduces significantly when both types of runs are used versus using only 0 runs. Similarly, significant increase in the compression ratio is obtained for the test sets c2670, c5315, s38417, and s38417d. For all the test sets except one, using EFDR code achieves higher compression ratio.

For test data decompression based on EFDR code, the decoder design follows a direct extention of the FDR decoder proposed in [17].

Table 19 shows the compresion ratios obtained for five compression schemes namely, geometric, FDR, EFDR, GFDR, and GEFDR, respectively. The best result from the three block sizes (8x8, 16x116, 32x32) is reported for each case.

As can be seen from the table, the two hybrid compression techniques, GFDR and GEFDR, both improved the compression ratio over the Geometric compression technique for all the circuits. However, the GEFDR compression scheme achieved better results and improved the compression ratio on average from 59.06% to 62.13%. Among the five compared compression schemes, the GEFDR compression scheme achieved the best results in 9 out of 14 test sets. However, the GFDR compression scheme achieved the best results in 3 out of the 14 test sets. The best compression ratio for the remaining test sets is achived by the EFDR compression technique.

Table 20 shows a detailed analysis of the number blocks encoded by the different encoding formats for the Geometric, GFDR, and GEFDR compression schems. This analysis is shown for an 8x8 block size. The

Table 19 Compression results of Geometric, FDR, EFDR, GFDR, and GEFDR techiques.

Circuit	Test Set Size	Geometric CR	FDR CR	EFDR CR	GFDR CR	GEFDR CR
c2670	10252	51.85	43.82	53.11	54.14	**54.56**
c5315	6586	27.88	20.47	28.64	29.03	**29.21**
s13207	163100	85.01	78.78	79.38	**85.48**	85.40
s15850	57434	60.32	56.49	56.29	**61.70**	61.43
s35932	21156	25.78	3.99	**45.63**	26.27	44.93
s38417	113152	46.50	37.66	**52.35**	48.37	51.45
s5378	20758	51.55	46.85	50.81	53.12	**53.18**
s13207d	165200	86.63	81.31	81.85	87.60	**87.74**
s15850d	76986	70.19	66.21	67.99	71.21	**71.42**
s35932d	28208	78.12	19.36	80.31	78.12	**81.71**
s38417d	164736	62.23	43.27	60.57	63.09	**65.23**
s38584d	199104	65.59	60.93	62.91	66.30	**67.03**
s5378d	23754	57.94	47.98	51.93	58.32	**58.62**
s9234d	39273	57.22	43.61	45.89	**58.39**	57.87
AVG		59.06	46.48	58.40	60.08	**62.13**

Table 20 Detailed analysis of block encodings for Geometric, GFDR, and GEFDR compression.

Circuit	Geometric				GFDR			GEFDR		
	#Blocks	#Filled	#Shapes Encoded	#Real	#Shapes Encoded	#FDR Encoded	#Real	#Shapes Encoded	#EFDR Encoded	#Real
c2670	180	56	99	25	70	33	21	68	40	16
c5315	115	8	61	46	52	22	33	51	23	33
s13207	2640	1671	963	6	895	72	0	906	62	1
s15850	924	127	787	10	677	120	0	698	97	2
s35932	442	76	191	175	182	54	130	129	196	41
s38417	1872	252	1484	136	1214	348	58	1059	534	27
s5378	351	73	220	58	193	46	39	185	63	30
s13207d	2640	2041	531	68	487	76	36	487	87	25
s15850d	1232	614	536	82	470	110	38	481	112	25
s35932d	442	56	384	2	384	0	2	334	50	2
s38417d	2704	1068	1447	189	1290	220	126	1129	453	54
s38584d	3111	1180	1584	347	1413	284	234	1442	332	157
s5378d	378	143	157	78	142	24	69	134	43	58
s9234d	620	150	410	60	367	71	32	375	60	35
AVG(%)		28.97	55.87	15.16	48.78	12.23	10.02	46.16	17.50	7.37

second column shows the total number of encoded blocks. The third column shows the number of blocks encoded as a block filled with either 0 or 1. The fourth and fifth columns show the number of blocks encoded by geometric shapes and those encoded by the real test data, respectively for the Geometric compression scheme. The sixth, seventh and eightth columns show the number of blocks encoded by geometric shapes, those encoded by FDR codes, and those encoded by the real test data, respectively for the GFDR compression scheme. Similarly, the last three columns show the number of blocks encoded by geometric shapes, those encoded by EFDR codes, and those encoded by the real test data, respectively for the GEFDR compression scheme. As can be seen from the table, both the GFDR and GEFDR compression schemes reduce the number of blocks encoded by the real test data and hence improve the compression ratio. For the circuits considered, the average number of real blocks is 15.16% for the Geometric compression scheme, 10.02% for the GFDR compresion scheme, and 7.37% for the GEFDR technique. Thus, the GEFDR

compression technique reduces the number of real blocks by more than 50%. As indicated by the results, there is still a percentage of blocks that achieve no compression and are encoded by storing the real test data. The average number of blocks encoded by FDR codes in the GFDR technique is 12.23% and the average number of blocks encoded by EFDR codes in the GEFDR technique is 17.5%. This indicates that these blocks achieve better compression if encoded by these codes rather than by geometric shapes, which adds to the benefit of the proposed hybrid compression schemes.

7. Conclusions

In this work, a novel compression/ decompression scheme for testing systems-on-a-chip has been presented. The technique is based on encoding the test data by geometric shapes. The test data is partitioned into blocks and then each block is encoded separately. To increase the compression ratio, the scheme exploits test vectors reordering, the block size, the type of bit to be encoded, and whether or not to encode the block. Experimental results on ISCAS85 and full-scanned versions of ISCAS89 benchmark circuits have demonstrated the effectiveness of the technique in achieving high compression ratio. In comparison to other test compression techniques, our technique achieves significantly higher compression ratio. We have demonstrated that starting with a test set with a larger size could result in higher test data compression. Thus, adding some redundancy to the test set may help in achieving higher compression. Based on statistics on the type of block encoding, we have demonstrated that some of the circuits have high percentage of real-data blocks. This shows that there is a room for improvement if these blocks are encoded using another compression scheme.

Test data decompression can be performed in software by an embedded processor or in hardware. In this work, we have demonstrated both options.

In this work, we have also proposed an extension to the recently proposed FDR code, namely Extended FDR (EFDR) code. The proposed technique is based on encoding both runs of 0's and 1's as opposed to encoding only runs of 0's. Based on experimental results on ISCAS benchmark circuits, it has been demonstrated that the proposed EFDR code outperformed FDR code and resulted in significant increase in test data compression ratio for several circuits, improving the compression ratio from 19.36% to 80.31% for one of the benchmark circuits. Furthermore, we have proposed two hybrid compression schemes that combine the Geometric and FDR compression schemes (GFDR), and the Geometric and EFDR compression schemes (GEFDR). The objective of these schemes is to reduce the number of blocks in the Geometric compression scheme that are encoded with the actual test data. Based on experimental results on ISCAS benchmark circuits, it has been demonstrated that the proposed hybrid compression schemes improved the

test data compression ratio for all the circuits over those obtained by the Geometric compression scheme. The GEFDR technique achieved the best results and improved the compression ratio on average from 59.06% to 62.13% over the Geometric compression scheme.

8. Project Objectives vs. Accomplishments

The principle objectives of this work as submitted in the proposal are as follows:

1. Study and examine the existing lossless test vectors compression/decompression schemes.
2. Propose a new compression/decompression scheme that is two-dimensional and geometric-primitives-based capable of compressing test vectors with higher compression ratio and that is simple to implement.
3. Implement the proposed scheme and conduct experiments on benchmark circuits to determine the quality of the proposed scheme and compare it with existing schemes.
4. Investigate a hardware implementation of the decoding algorithm of the proposed scheme.
5. Publish the results of this research in refereed international conference on testing and a refereed journal.

All the mentioned above objectives have been achieved. A summary of the accomplished work is given below:

1. **Literature Survey**: We have performed extensive literature survey of all compression and compaction techniques for combinational and sequential circuits. Detailed information about the conducted survey is not included in the report to make the report more concise.
2. **Generating compacted test vectors for benchmark circuits**: we have used the automatic test pattern generation tool (HITEC) and the fault simulators (HOPE & PROOFS) to generate and compact the generated test sets before compressing them. Test compaction was achieved by implementing the reverse order fault simulation algorithm. These test sets were used in the initial phase of the project but later we used the test sets generated by Mintest for comaprison purposes with published results.
3. **Implementing test relaxation algorithm**: In order to have effective test data compression, it is important to identify those bits whose value is an X. We have implemented test relxation based on a brute force method by changing every bit to an X and doing fault simulation to see whether the relaxation is possible or not.

4. **Test Vector Reordering**: We have developed several test vector ordering algorithms and implemented them. The algorithms were based on a gready ordering and near-optimal ordering. Several weight functions were experimented with and the one that gave the best results was chosen.
5. **Developing & Implementing the encoding algorithm**: we have developed the encdoding algorithm and implemented it. Many improvements in the encoding algoithm were made based on analysis of experimenta results.
6. **Developing & Implementing the decoding algorithm in software**: we have developed and implmented the decoding algorithm in software to be run utilizing on-chip processor. The results of the decoding algorithm were verified by fault simulation.
7. **Comparison with Published Results**: We have compared our results with two recently published and found that our technique achieves the best compression ratios. We have experimented with our technique on different test sets and we demonsrated its effectiveness.
8. **Design of the decoding algorithm in hardware**: This step is necessary in case the chip does not have a processor and the decoding circuitry has to be inserted. We have designed the decoding circuitry in hardware and modeled it in VHDL and verified its correctness.
9. **Improved FDR Compression teschnique**: Recently an effective test compression technique has been proposed with the advantage of simple decoding circuitry. Based on test data analysis, we found that we could improve the compression effectiveness of this technique without increasing the decoder complexity. We have extended this technique and called it EFDR and implemented it. The technique has resulted in significant improvement in test compression.
10. **Hybrid compression technique**: We have proposed hybrid compression techniques that combine the Geometric compression technique with either FDR or EFDR techniques and demonstrated improvement in the compression ratio.
11. **Publications**: we have published four conference papers and submitted one paper for journal publication.

It should be noted here that the accomplished work in 9 and 10 was more than was planned in the proposal.

List of Publications Resulting from this Work

AimanEl-Maleh, *Saif Al-Zahir, and Esam Khan,"A Geometric-Primitives-Based Compression Scheme for Testing Systems-on-a-Chip," 19'th* ***IEEE VLSI Test Symposium (VTS)****, pp. 54-59, 2001.*

Saif Al-Zahir, ***Aiman El-Maleh****, and Esam Khan, "An Efficient Test Vector Compression Technique Based on Geometric Shapes," the 8th* ***IEEE International Conference on Electronics, Circuits and Systems (ICECS 2001),*** *pp. 1561-1564, 2001.*

Aiman El-Maleh *and Raslan Al-Abaji, "Extended Frequency-Directed Run Length Code with Improved Application to System-on-a-chip Test Data Compression" Proc. of the 9th I****EEE International Conference on Electronics, Circuits and Systems****, pp. 449-452, Sep. 2002.*

Aiman El-Maleh*, "A Hybrid Test Compression Technique for Efficient Testing of Systems-on-a-Chip," **10th IEEE International Conference on Electronics, Circuits and Systems****, pp. 599-602, December 2003.*

The following paper was also submitted and it is still under processing:

AimanEl-Maleh*, Saif Al-Zahir, and Esam Khan,"A Geometric-Primitives-Based Compression Scheme for Testing Systems-on-a-Chip," submitted to IEEE Transactions on Computer-Aided Design..*

It is also worth mentioning that our work in this project has inspired us to the importance of relaxing a test set and has led to do work in this direction that has resulted in the following publication:

Aiman El-Maleh *and Ali Al-Suwaiyan, "An Efficient Test Relaxation Technique for Combinational and Full-Scan Sequential Circuits" Proc. of the 20'th* ***IEEE VLSI Test Symposium (VTS)****, pp. 53-59, 2002.*

Aiman El-Maleh *and Ali Al-Suwaiyan, "An Efficient Test Relaxation Technique for Combinational Circuits Based on Critical Path Tracing" Proc. of the 9th I****EEE International Conference on Electronics, Circuits and Systems****, pp. 461-465, Sep. 2002.*

Aiman El-Maleh *and Khaled Al-Utaibi, "An Efficient Test Relaxation Technique for Synchronous Sequential Circuits" Proc. of the 21'th* ***IEEE VLSI Test Symposium (VTS)****, pp. 179-185, 2003.*

Aiman El-Maleh *and Khaled Al-Utaibi, "An Efficient Test Relaxation Technique for Synchronous Sequential Circuits" Proc. of the* ***IEEE International Symposium on Circuits and Systems (ISCAS)****, pp.* V-545 - V-548, *2003.*

In addition, our publications that resulted from this project have been cited by several references as shown below, which clearly indicate the usefulness of the results obtained in this project:

AimanEl-Maleh*, Saif Al-Zahir, and Esam Khan,"A Geometric-Primitives-Based Compression Scheme for Testing Systems-on-a-Chip," 19'th* ***IEEE VLSI Test Symposium (VTS)****, pp. 54-59, 2001.*

Citations:

- *Bayraktaroglu, I, Orailoglu, A.,* "Concurrent application of compaction and compression for test time and data volume reduction in scan designs" IEEE Transactions on Computers, ,Volume: 52 , Issue: 11 , Pages:1480 – 1489, Nov. 2003.
- *Lei Li; Chakrabarty, K.;"*Test set embedding for deterministic BIST using a reconfigurable interconnection network" IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume: 23 , Issue: 9 , Pages:1289 – 1305, Sept. 2004.
- A. Chandra and K. Chakrabarty, "Analysis of Test Application Time for Test Data Compression Methods Based on Compression Codes", JOURNAL OF ELECTRONIC TESTING: Theory and Applications 20, pp. 199–212, 2004.
- M. Tehranipour, M. Nourani, and K. Chakrabarty, "Nine-Coded Compression Technique with Application to Reduced Pin-Count Testing and Flexible On-Chip Decompression," Proceedings of the 2004 Design and Test in Europe conference, pp. 1284-1289, Feb. 2004.
- M. Flottes, R. Poirier, and B. Rouzeyre, "An Arithmetic Structure for Test Data Horizontal Compression," Proceedings of the 2004 Design and Test in Europe conference, pp. 428-433, Feb. 2004.

- Lei Li and K. Chakrabarty, "Deterministic BIST Based on a Reconfigurable Interconnect Network", IEEE Int. Test Conf., Oct. 2003.
- A. Chandra and K. Chakrabarty, " A unified Approach to Reduce SOC Test Data Volume, Scan Power and Testing Time", IEEE Trans. Computer-Aided Design, Vol. 22, No 3, pp. 352-362, Macrh 2003.
- P. Gonciari, B. Al-Hashimi, and N. Nicolici, "Variable-Length Input Huffman Coding for System-on-a-Chip Test", IEEE Trans. CAD, Vol. 22, No 6, pp. 783-796, June 2003.
- P. Gonciari, B. Al-Hashimi, and N. Nicolici, "Improving Compression Ratio, Area Overhead, and Test Application Time for System-on-a-Chip Test Data Compression/Decompression", IEEE Design Automation and Test in Europe Conf., Feb. 2002.
- Lei Li and K. Chakrabarty, "Test Data Compression Using Dictionaries with Fixed-Length Indices", IEEE VLSI Test Symposium, pp. 219-224, April 2003.
- Lei Li, K. Chakrabarty and Nur A. Touba "Test Data Compression Using Dictionaries with Selective Entries and Fixed-Length Indices", ACM Trans. on Design Automation of Electronic Systems, Vol. 8 , No 4, pp. 470-490, Oct. 2003.
- A. Jas, J. Ghosh-Dastidar, and N. Touba, "An Efficient Test Vector Compression Scheme Using Selective Huffman Coding", IEEE Trans. Computer-Aided Design , Vol. 22, No 6, pp. 797-806, June 2003.
- A. Jas, and N. Touba, "Deterministic Test Vector Compression/Decompression for Systems-on-a-Chip Using an Embedded Processor", Journal of Electronic Testing: Theory and Applications, 18, pp. 503-514, 2002.
- K. Balakrishnan and N. Touba, "Deterministic Test Vector Decompression in Software Using Linear Operations", IEEE VLSI Test Symposium, pp. 225-231, April 2003.
- K. Balakrishnan and N. Touba, "Matrix-Base Test Vector Decompression using and Embedded Processor", IEEE Symp. on Defect and Fault Tolerance, pp. 159-165, 2002.
- Ichihara, H.; Kinoshita, K.; Isodono, K.; Nishikawa, S.,"Channel width test data compression under a limited number of test inputs and outputs," Proceedings. 16th International Conference on VLSI Design, pp. 329 – 334, Jan. 2003.
- Ichihara, H.; Shintani, M.; Ohara, T.; Inoue, T., "Test response compression based on Huffman coding," 12th Asian Test Symposium, Page(s): 446- 449, Nov. 2003.
- Schafer, L.; Dorsch, R.; Wunderlich, H.-J., "RESPIN++ - deterministic embedded test," Proceedings of the Seventh IEEE European Test Workshop (ETW'02), pp. 37- 44, 2002.

- V Iyengar, A Chandra **, "**A Unified SOC Test Approach Based on Test Data Compression and TAM Design," Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03).

Aiman El-Maleh *and Raslan Al-Abaji, "Extended Frequency-Directed Run Length Code with Improved Application to System-on-a-chip Test Data Compression" Proc. of the 9th I****EEE International Conference on Electronics, Circuits and Systems****, pp. 449-452, Sep. 2002.*

Citations:

- *Lei Li; Chakrabarty, K.; Kajihara, S.; Swaminathan, S.;* "Efficient Space/Time Compression to Reduce Test Data Volume and Testing Time for IP Cores" 18th International Conference on VLSI Design, Pages:53 - 58 Jan. 2005
- Chandra and K. Chakrabarty, "Analysis of Test Application Time for Test Data Compression Methods Based on Compression Codes", JOURNAL OF ELECTRONIC TESTING: Theory and Applications 20, pp. 199–212, 2004.
- Lei Li and K. Chakrabarty, "Test Data Compression Using Dictionaries with Fixed-Length Indices", IEEE VLSI Test Symposium, pp. 219-224, April 2003.
- Lei Li and K. Chakrabarty, "Test Data Compression Using Dictionaries with Selective Entries and Fixed-Length Indices", ACM Trans. On Design Automation of Electronic Systems, Vol. 8 , No 4, pp. 470-490, Oct. 2003.
- Kajihara S., Yasumi Doi Lei Li, Chakrabarty K., *"*On combining pinpoint test set relaxation and run-length codes for reducing test data volume," Proceedings. 21st International Conference on Computer Design, pp. 387 - 392 Oct. 2003.
- Kay D., Mourad S., *"*Interactive built-in self-test compression for testing a system-on-a-chip," IEE Proceedings- Computers and Digital Techniques, Vol. 150, No. 4, pp. 189 – 200, July 2003.
- M. Tehranipour, M. Nourani, and K. Chakrabarty, "Nine-Coded Compression Technique with Application to Reduced Pin-Count Testing and Flexible On-Chip Decompression," Proceedings of the 2004 Design and Test in Europe conference, pp. 1284-1289, Feb. 2004.

- Xiaoyun Sun, Larry Kinney, Bapiraju Vinnakota, “Combining dictionary coding and LFSR reseeding for test data compression “, Proceedings of the 41st annual conference on Design automation, pp. 944 - 947, 2004.

Contribution of this project to stutdents:

1. Master Thesis: Esam Khan, MsC in Computer Engineering "A Two-Dimensional Geometric-Primitives-Based Compression Scheme for Testing Systems-on-a-Chip".

2. Graduate Student Training: The following two students were trained by working on this project:

- Mr. Esam Khan and his work on this project has resulted in 2 conference publications.
- Mr. Raslan Al-Abaji and his work on this project has resulted in 1 conference publication.

3. Undergraduate Student Training: Two students were trained by doing his senior design project on issues related to the project:

- Mr. Faisal Ba-Haidarah, Senior Design Project, “Hybrid Test Compression”.
- Muhammad Kalkattawi, Senior Design Project, "Test Data Compression/Decompression for System on Chip".

ملخّص نتائج المشروع

١. اقتراح طريقة جديدة وفعالة لضغط بيانات اختبارات الشرائح الحاوية لأنظمة متكاملة بدون فقد أي معلومات باستخدام الأشكال الهندسية الأساسية ثنائية الأبعاد.
٢. تصميم دائرة كهربائية لفك ظغط بيانات اختبارات الشرائح المضغوطة باستخدام الأشكال الهندسية الأساسية ثنائية الأبعاد.
٣. اقتراح امتداد إلى شفرة إف دي آر (إي إف دي آر)على أساس تشفير كلا الجريين ال٠ و ١ وتبيّن بالنّتائج التّجريبيّة فاعليّتها في تحقيق نسبة انضغاط أعلى.
٤. اقتراح تكنيكات ضغط بيانات اختبار هجين التي تستغلّ الاستعمال لإمّا شفرات إف دي آر أو إي إف دي آر لتقليل عدد الكتل التي تُشَفر بتخزين بيانات الاختبار الحقيقيّة باستخدام طريقة الضغطالمبنية على استخدام الأشكال الهندسية الأساسية ثنائية الأبعاد.

Acknowledgment

This work is supported by King Fahd University of Petroleum & Minerals under project FT2000/07. The author would like to thank Dr. Saif Alzahir, Mr. Esam Khan, Mr. Raslan Al-Abaji, Mr. Faisal Ba-Haidarah, and Mr. Muhammad Kalkattawi for their contribution in this project. Special thanks are also to Dr. Alaaeldin Amin for his useful coments on the hardware impementation of the decompression circuitry for the Geometric compression technique.

References

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

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

[3] M. Schulz, E. Trischhler, and T. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Trans. Computer-Aided Design*, pp. 126-137, Jan. 1988.

[4] J. Chang and C. Lin, "Test Set Compaction for Combinational Circuits," *IEEE Trans. Computer Aided Design*, pp. 1370-1378, Nov. 1995.

[5] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. Reddy, "Cost-Effective Generation of Minimal Test sets for Stuck-at Faults in Combinational Circuits," *IEEE Trans. Computer Aided Design*, pp. 1496-1504, Dec. 1995.

[6] I. Hamzaoglu and J. H. Patel, "Test Set Compaction Algorithms for Combinational Circuits'', *Proc. Int. Conf. Computer-Aided Design*, pp. 260-267, Nov. 1998.

[7] I. Pomeranz, L. Reddy, and S. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits," *Proc. of Int. Test Conference*, pp. 194-203, 1991.

[8] S. Chakradhar and A. Raghunathan, "Bottleneck Removal Algorithm for Dynamic Compaction in Sequencial Circuits," *IEEE Trans. Computer-Aided Design*, Vol. 16, No. 10, pp. 1157-1172, Oct. 1997.

[9] G. Gibson, Toby Berger, David Lindbergh, Richard L., III Baker, *Digital compression for multimedia*, (Morgan Kaufmann Publishers)., 1998.

[10] K. Chakrabarty, B.T. Murray, J. Liu, and M. Zhu, "Test Width Compression for Built-In Self-Testing," *Proc. of International Test Conference,* pp. 328-337, 1997.

[11] J. Rajski, J. Tyszer, and N. Zaccharia, "Test Data Decompression for Multiple Scan Designs with Boundary Scan," *IEEE Trans. Computers,* pp. 1180-1200, Nov. 1998.

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

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

[14] M. Ishida, D.S. Ha, and T. Yamaguchi, "COMPACT: A Hybrid Method for Compression Test Data," *Proc. of VLSI Test Symposium,* pp. 62-69, 1998.

[15] 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.

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

[17] A. Chandra, and K. Chakrabarty, "Frequency-Directed Run-Length (FDR) Codes with Application to Systems-on-a-Chip Test Data Compression," *Proc. of IEEE VLSI Test Symposium,* pp. 42-47, 2001.

[18] A. Jas, J.G. Dastidar and N.A. Touba, "Scan Vector Compression/ Decompression Using Statistical Coding," *Proc. of Int. Test Conference,* pp. 202-207, 1994.

[19] A. Jas and N.A. Touba, "Using an Embedded Processor for Efficient Deterministic Testing of System-on-a-Chip," *Proc. of IEEE Int. Conf. on Computer Design (ICCD),* pp. 418-423, 1999.

[20] P. Gonciari, B. Al-Hashimi, and N. Nicolici, "Improving Compression Ratio, Area Overhead, and Test Application Time for System-on-a-Chip Test Data Compression/Decompression", IEEE Design Automation and Test in Europe Conf., Feb. 2002.

[21] R. Dorsch and H.-J. Wunderlich, "Resuing scan chains for test pattern decompression", European Test Workshop, pp. 124-132, 2001.

[22] R. Dorsch and H.-J. Wunderlich, "Tailoring ATPG for embedded testing", Int. Test Conf., pp. 530-537, 2001.

[23] H. Ichihara, K. Kinoshita, I. Pomeranz, and S. M. Reddy, "Test transformation to improve compaction by statistical coding", Int. Conf. VLSI Design, pp. 294-299, 2000.

[24] H. Ichihara, A. Ogawa, T. Inoue, and A. Tamara, "Dynamic test compression using statistical coding", Asian Test Symp., pp. 143-148, 2001.

[25] S. M. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz, "On test data reduction for multiple scan chain designs", VLSI Test Symp., pp. 103-108, 2002.

[26] F. G. Wolff and C. Papachristou, "Multiscan-based test compression and hardware decompression using LZ77", Int. Test Conf., pp. 331-339, 2002.

[27] M. Knieser, F. G. Wolff, C. Papachristou, D. Weyer and D. McIntyre, "A technique for high ratio LZW compression", Design Automation and Test in Eyrope Conf., pp. 116-121, 2003.

[28] Lei Li and K. Chakrabarty, "Test Data Compression Using Dictionaries with Fixed-Length Indices", IEEE VLSI Test Symposium, pp. 219-224, April 2003.

[29] C.Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Konemann, "OPMISR: The foundation for compressed ATPG vectors", Int. Test Conf., pp. 748-757, 2001.

[30] B. Konemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A SmartBIST variant with guaranteed encoding", Asian Test Symp., pp. 325-330, 2001.

[31] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, H. Tsai, A. Hertwing, N. Tamarapalli, G. Mrugalski, G. Eide, and J. Qang, "Embedded deterministic test for low cost manufacturing test", Int. Test Conf., pp. 301-310, 2002.

[32] L. Baker, *VHDL Programming with Advanced Topics*, John Wiley & Sons, 1993.

APPENDIX (Puplished Papers)

A Geometric-Primitives-Based Compression Scheme for Testing Systems-on-a-Chip

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

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

Abstract

The increasing complexity of systems-on-a-chip with the accompanied increase in their test data size has made the need for test data reduction imperative. In this paper, we introduce a novel and very efficient lossless compression technique for testing systems-on-a-chip based on geometric shapes. The technique exploits reordering of test vectors to minimize the number of shapes needed to encode the test data. The effectiveness of the technique in achieving high compression ratio is demonstrated on the largest ISCAS85 and full-scanned versions of ISCAS89 benchmark circuits. In this paper, it is assumed that an embedded core will be used to execute the decompression algorithm and decompress the test data.

1. Introduction

With today's technology, it is possible to build complete systems containing millions of transistors on a single chip. Systems-on-a-chip (SOC) are comprised of a collection of 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 [11], [12]. To test a certain chip, the entire set of test vectors, for all the cores and components inside the chip, has to be stored in the tester memory. Then, during testing, the test data must be transferred to the chip under test and test responses collected from the chip to the tester as illustrated in Figure 1.

One of the challenges in testing SOC is dealing with the large size of test data that must be stored in the tester and transferred between the tester and the chip. The amount of time required to test a chip depends on the size of test data that has to be transferred from the tester to the chip and the channel capacity.

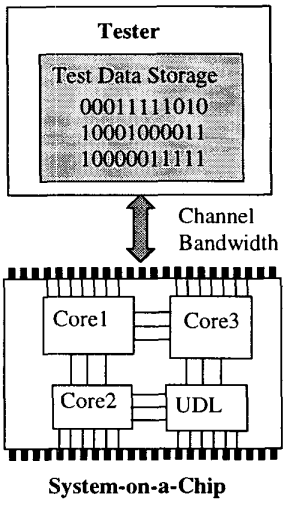


Figure 1. Test data transfer between the tester and the circuit under test.

The cost of automatic test equipment (ATE) increases significantly with the increase in their speed, channel capacity, and memory. As testers have limited speed, channel bandwidth, and memory, the need for test data reduction becomes imperative. To achieve such reduction, several compaction and lossless compression schemes were proposed in the literature.

The objective of test set compaction is to generate the minimum number of test vectors that achieve the desired fault coverage. There are two main types of compaction, static compaction and dynamic compaction. In static compaction, the number of test vectors is reduced after they have been generated. Examples of static compaction algorithms include reverse order fault simulation [15], forced pair merging [16], N_by_M [18], and redundant vector elimination (RVE) [14]. In dynamic compaction, the number of vectors is

1093-0167/01 $10.00 © 2001 IEEE

minimized during the automatic test pattern generation (ATPG) process. Examples of dynamic compaction algorithms include COMPACTEST [17], and bottleneck removal [6].

In test data compression, the objective is to reduce the number of bits needed to represent the test data. For test data compression, it is essential that the compression is lossless. Run length coding, Huffman codes, Lempel-Ziv algorithms, and arithmetic codes are examples of lossless compression [13].

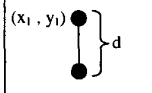
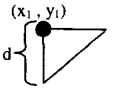
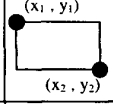
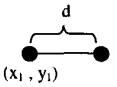
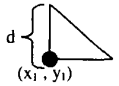
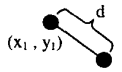
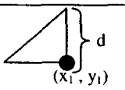
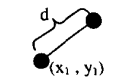
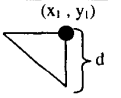
Several test data compression/decompression techniques were proposed in the literature. These techniques can be classifies into two categories; one is based on BIST and Pseudo-Random Generators (PRG) and the other is based on deterministic compression.

Examples of BIST-based compression techniques are test width compression [2], variable length reseeding [5], and Design For High Test Compression (DFHTC) [10].

Deterministic compression techniques take advantage of the high correlation between test vectors. One of these techniques is proposed in [1] and uses Burrows-wheeler (BW) transformation and a modified version of run-length coding to encode the test data. This technique has been improved in [3] by applying the GZIP compression scheme to strings that are not effectively compressed by run-length coding. Another technique proposed in [8] uses what is called variable-to-block run-length coding. In this technique, a codeword is used to encode a block of data based on the number of zeros followed by a one in that block. This technique is used for compressing fully-specified test data that feeds a cyclical scan chain. A cyclical scan chain is used to decompress this data and transfer it to the "test scan chain". Golomb code is a variable-to-variable run-length code that is used in [4] 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], statistical coding is used for encoding deterministic test data. The technique uses a modified version of Huffman coding as to minimize the bits needed for codewords. Although this technique has less compression ratio than Huffman coding, the hardware implementation of the decoder is simpler. Another technique was proposed in [7] which performs decompression of test data based on an embedded processor. The technique is based on storing the differing bits between two test vectors. It divides each test vector into blocks and stores those blocks that are different from the preceding vector.

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, an embedded core is used to 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 the chip.

Table 1. The used primitive geometric shapes.

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

2. The Proposed Encoding Algorithm

The proposed encoding algorithm is based on encoding the 0's or the 1's in a test set by geometric shapes. In this work, we limited those primitive shapes to the basic four, namely: point, line, triangle, and rectangle as shown in Table 1. These shapes are the most frequently encountered shapes in any test set. For the rectangles, two points are needed to encode the shape and each point costs $2*\log_2 N$, where N is the block dimension. However, lines and triangles can be represented by a point and a distance d and this reduces the number of bits needed to encode them by $(\log_2 N)-2$ in comparison to encoding them by two points. Two bits are used to determine the type of line or the type of triangle encoded.

Figure 2 shows the algorithm of the encoder, which consists of the following main steps:

(i) Test Set Sorting

Sorting the vectors in a test set is crucial and has a significant impact on the compression ratio. In this step, we aim at generating clusters of either 0's or 1's in such a way that it may partially or totally be fitted in one or more of the geometric shapes shown in Table 1. Several sorting scenarios have been considered and investigated. In this work, we used a simple correlation-based sorting technique. The sorting may be with respect to 0's (*0-*

```
Encoder (N)
    Sort_Test_Set ();
    Partition_Test_Set (N);
    For i = 1 to # of segments
      For j = 1 to # of blocks in i
        Extract_Shapes (1, j);
        α1 = Encode_Shapes ();
        Extract_Shapes (0, j);
        α0 = Encode_Shapes ();
        B = # of bits in j + 2;
        E = min (α0, α1,B);
        Store_Encoded_Bits ();
        E_total += E;
End Encoder;

Extract_Shapes(b, j)
  For each bit x in block j {
    If x = b Then {
      Find the largest line of each type started at x
      Find the largest triangle of each type such that
        x is the vertix of the right angle
      Find the largest rectangle such tha x is its up-
        left corner
    }
  }
  Solve a covering problem to find the best group of
    shapes covering all bits b in block j.
End Extract_Shapes;
```

Figure 2. Test vectors encoding algorithm.

sorting), to 1's (*1-sorting*) or to both 0's and 1's (*0/1-sorting*). The technique is based on finding the distance D between two vectors A and B that maximizes the clusters of 0's and 1's.

The distance D may be computed with respect to 0's (*0-distance*), to 1's (*1-distance*) or to 0's and 1's (*0/1-*distance) as follows:

$$D = \sum_{i=0}^{k-1} W(A_i,B_{i-1}) + W(A_i,B_i) + W(A_i,B_{i+1})$$

where k is the test vector length and $W(A_i, B_i)$ is the weight between bits A_i and B_i. Table 2, Table 3 and Table 4 specify the weights used in computing the 0-distance, the 1-distance, and the 0/1-distance between two vectors, respectively. Note that for i = 0, $W(A_i, B_{i-1}) = 0$ and for i = k-1, $W(A_i, B_{i+1}) = 0$.

The assignment of a 0.25 weight for an 'x' to each of its immediate neighbors be it an 'x' or the sorted bit ('0' for 0-sorting, '1' for 1-sorting and '0' and '1' for 0/1-sorting) is chosen due to the following reasons. First, this weight may help in completing, integrating, 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.

Table 2. Weights for the 0-distance between two test vectors.

	0	1	x
0	1.0	0.0	0.25
1	0.0	0.0	0.0
x	0.25	0.0	0.25

Table 3. Weights for the 1-distance between two test vectors.

	0	1	x
0	0.0	0.0	0.0
1	0.0	1.0	0.25
x	0.0	0.25	0.25

Table 4. Weights for the 0/1-distance between two test vectors.

	0	1	x
0	1.0	0.0	0.25
1	0.0	1.0	0.25
x	0.25	0.25	0.25

Table 5. An example of test vector sorting.

Original Vectors	*v1*	0	0	1	X	1	0	X	X
	v2	0	X	1	1	0	0	0	1
	v3	1	1	X	1	1	X	0	1
Sorted Vectors (0-dist.)	*v2*	0	X	1	1	0	0	0	1
	v1	0	0	1	X	1	0	X	X
	v3	1	1	X	1	1	X	0	1
Sorted Vectors (1-dist.)	*v3*	1	1	X	1	1	X	0	1
	v2	0	X	1	1	0	0	0	1
	v1	0	0	1	X	1	0	X	X

In Table 5, we show a simple example to illustrate the impact of sorting on test vector compression. As can be seen, sorting the vectors based on the 0-distance requires the encoding of two triangles to encode the 0's. However, sorting the vectors based on the 1-distance requires the encoding of one triangle and two lines to encode the 1's. Thus, for this example sorting based on the 0-distance results in higher compression.

(ii) Test Set 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 set 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 $\lceil C/N \rceil$. A *segment* consists of K blocks. In other words, the test set

is segmented into L segments each contains K blocks. For test vectors whose columns and/or rows are not divisible by the predetermined block dimension 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 number of vectors, the vector length, and the block dimension, the number of bits used to encode the coordinates of the geometric shapes can be less than $log_2 N$. The decoder recognizes those special cases and decodes them properly.

(iii) Encoding process

As mentioned earlier, the encoding process will be applied on each block independently. The procedure *Extract_Shapes(b)* will find the best group of shapes that cover the bits that are equal to *b* as shown in the algorithm. *Encode_Shapes* determines the number of bits, α, needed to encode this group of shapes. There are two cases that may occur:

(a) The block contains only 0's and x's or 1's and x's. In this case, the block can be encoded as a rectangle. However, instead of this it is encoded by the code 01 followed by the bit that fills the block. Hence, the number of bits to encode the block α = 3.

(b) The block needs to be encoded by a number of shapes. In this case, we need the following:

- 2 bits to indicate the existence of shapes and the type of bit encoded. If the encoded bit is 0, then the code is 10, otherwise it is 11.
- P = $(2*\text{Log}_2 N - 3)$ bits to encode the number of shapes, S. If the number of shapes exceeds 2^P, then the number of bits needed to encode the shapes is certainly greater than the total number of bits in the block. In this case, the block is not encoded and the real data is stored.
- $\sum_{i=1}^{S} L_i$; where L_i is computed as follows
  - If shape i is a point, $L_i = 2 + 2*\log_2 N$ (shape type, coordinates).
  - If shape i is a line or a triangle, $L_i = 4 + 3*\log_2 N$ (shape type, type of line or triangle, point and distance)
  - If shape i is a rectangle, $L_i = 2 + 4*\log_2 N$ (shape type, 2 points)

Therefore, $\alpha = 2 + P + \sum_{i=1}^{S} L_i$

If α_0 and α_1 are greater than B (N*N+2), then it is better not to encode the block. Instead, the real data is stored after a 2-bit code (00). The procedure *Store_Encoded_Bits* will decide which case is the best (encoding 0's, encoding 1's, or storing the real data) based on E, the minimum of α_0, α_1, and B.

```
Decoder ()
  Read (# of Vectors (R), Vector_Length (C), N);
  Compute_Parameters ();
  For i = 1 to # of segments {
    For j = 1 to # of blocks in i {
      b1b0 = Read_Bits (2);
      Case b1b0
        00 : Read_Bits (N* N);
        01 : b_type = Read_Bits (1);
             Fill_Block (j, b_type);
        10 : Decode_Shapes (0);
        11 : Decode_Shapes (1);
      End Case;
    }
   Output_Segment ();
  }
End Decoder;

Decode_Shapes (b)
  Num_Shapes = Read_Bits (2*log2 N -3);
  For j = 1 to Num_Shapes
    Shape_type = Read_Bits (2);
    Case Shape_type
      00 : c = Get_Coordinate ();
           Fill_Point (b,c);
      01 : t = Get_Type ();
           c = Get_Coordinate ();
           d = Get_Distance ();
           Fill_Line(b, t, c,d);
      10 : t = Get_Type ();
           c = Get_Coordinate ();
           d = Get_Distance ();
           Fill_Triangle(b, t, c,d);
      11 : c1 = Get_Coordinate ();
           c2 = Get_Coordinate ();
           Fill_Rectangle (b,c1,c2);
End Decode_Shapes;
```

Figure 3. Test vectors decoding algorithm.

3. Decoding Process

The decoding process is simple and straightforward. In this work, we assume that an embedded processor on a chip will implement the decoding algorithm. 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 [7]. A similar framework can be used for our decoding algorithm.

Figure 3 shows the algorithm of the decoder. It first reads the arguments given by the encoder and computes the parameters needed for the decoding process. These parameters include the number of segments, the number

Table 6. Compression results of the proposed scheme for various block sizes.

			Block 8x8 Cmp. Ratio			Block 16x16 Cmp. Ratio			
Circuit	Scan Size	No. Vec	1-distance	0-distance	0/1-distance	1-distance	0-distance	0/1-distance	CPU (sec)
c7552	207	73	**37.873**	37.35	37.754	28.661	30.66	33.618	3
c2670	233	44	49.815	50.39	**51.853**	45.416	46.635	47.444	3
s5378	214	97	50.496	49.961	**51.551**	41.418	42.61	44.19	4
s9234	247	105	42.834	42.803	**43.451**	38.249	38.442	38.905	3
s15850	611	94	59.778	**60.898**	60.32	58.81	59.301	59.632	15
s13207	700	233	83.703	83.518	84.145	84.497	84.566	**85.012**	51
s38417	1664	68	46.114	**46.552**	46.497	42.788	43.024	42.47	29

Table 7. Comparison with the techniques by Jas and Touba [7] and Chandra and Chakrabarty [4].

	Proposed Scheme			**Jas and Touba [8]**			**Chandra and Chakrabarty [4].**		
Circuit	Org. Bits	Cmp. Ratio	Cmp. Bits	Org. Bits	Cmp. Ratio	Cmp. Bits	Org. Bits	Cmp. Ratio	Cmp. Bits
c7552	15111	37.873	9388	62721	**42.39**	36134	-	-	-
c2670	10252	51.853	4936	35183	**58.45**	14619	-	-	-
s5378	20758	**51.551**	10057	29850	39.0	18209	23754	40.70	14086
s9234	25935	**43.451**	14666	48906	26.6	35897	39273	43.34	22252
s15850	57434	**60.898**	22458	86151	46.65	45962	76986	47.11	40717
s13207	163100	**85.012**	24446	186200	73.32	49678	165200	74.78	41664
s38417	113152	46.552	60478	247936	**59.06**	101505	164736	44.12	92055

of blocks in a segment and the dimensions of the partial blocks. In order to reconstruct the vectors, each segment has to be stored before sending its vectors to the circuit under test. For each segment, its blocks are decoded one at a time. The first two bits indicate the status of the block as follows:

- 00: the block is not encoded and the following N*N bits are the real data.
- 01: fill the whole block with 0's or 1's depending on the following bit.
- 10: There are shapes that are filled with 0's.
- 11: There are shapes that are filled with 1's.

For those blocks that have shapes, the procedure *Decode_Shapes* is responsible for decoding these shapes. It reads the number of shapes in the block and then for each shape it reads its type and based on this it reads its parameters and fills it accordingly.

After all the blocks in a segment have been decoded, the segment is output to the circuit under test vector by vector.

4. Experimental Results

In order to demonstrate the effectiveness of our scheme, we have performed experiments on a number of the largest ISCAS85 and full-scanned versions of ISCAS89 benchmark circuits. The experiments were run on a Pentium II processor with a speed of 350 MHz and a 32 Mbyte RAM. We have used the test sets generated by MinTest [14], which are highly compacted test sets, that achieve 100% fault coverage of the detectable faults in each circuit. Test cubes were generated from each test set as this has the advantage of keeping unnecessary assignments as x's, which enables higher compression. Then, the test vectors were sorted to maximize the compression. In this work, test vectors were sorted based on a greedy algorithm. Test vectors sorting based on the 0-distance, the 1-distance, and the 0/1-distance was performed. For both the 0-distance and 0/1-distance sorting, the test vector with more 0's was selected as the first vector. However, for the 1-distance sorting, the vector with more 1's was selected as the first vector.

The test sets were partitioned into blocks of sizes 8x8 and 16x16, respectively. Then, the proposed encoding algorithm was applied for each case separately as shown in Table 6. The second column in the table shows the scan size, which is basically the width of a test vector. The third column indicates the number of test vectors in the test set. The *compression ratio* is computed as:

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

As can be seen, the effectiveness of the proposed encoding algorithm is clearly demonstrated as high compression ratio was obtained for all the circuits. For most of the circuits, sorting based on the 0/1-distance on an 8x8 block size produced the best results.

The last column in Table 6 shows the total CPU time used for compressing the test vectors based on the two block sizes and based on the three types of distance sorting, i.e. the total CPU time used to produce the best result, which is highlighted in the table.

Based on the compression results in Table 6, our technique achieves an average compression ratio of around **54%** based on highly compacted tests. In Table 7, we compare the compression ratio obtained by our technique with that obtained by the techniques proposed in [7] and [4]. It is important to point out that although the test sets used in our work are different from those used in [7] and [4], they are considerably smaller. As can be seen from the table, for all the compared circuits, our technique achieves significantly higher compression ratio than the technique in [4]. Furthermore, in four of the circuits, out of seven, our technique achieves higher compression ratio than the technique in [7]. It should be observed here that for the three circuits where the technique in [7] achieves higher compression ratio, their original test sets are significantly larger, i.e. they contain much more redundancy, which leads to higher compression ratio. For example, the original test set used in [7] for the circuit c7552 is more than four times larger than the original test set we used.

All the compressed test sets were decoded and verified by fault simulation. The decoding algorithm is very fast and the decoding time for each test set was in fractions of a second.

5. Conclusions

In this paper, a fast and very efficient compression/ decompression scheme for testing systems-on-a-chip has been presented. The technique is based on encoding the test data by geometric shapes. The test data is partitioned into blocks and then each block is encoded separately. To increase the compression ratio, the scheme exploits test vectors reordering, the block size, the type of bit to be encoded, and whether or not to encode the block. Experimental results on ISCAS85 and full-scanned versions of ISCAS89 benchmark circuits demonstrate the effectiveness of the technique in achieving high compression ratio. An average of 54% compression ratio is achieved on highly compacted test sets. In this work, we assumed that the decompression of test data is performed in software by an embedded processor. Hardware implementation of the decompression algorithm will be investigated in future work.

Acknowledgment

The authors would like to thank King Fahd University of Petroleum & Minerals for support.

References

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

[2] K. Chakrabarty, B.T. Murray, J. Liu, and M. Zhu, "Test Width Compression for Built-In Self-Testing," *Proc. of International Test Conference,* pp. 328-337, 1997.

[3] M. Ishida, D.S. Ha, and T. Yamaguchi, "COMPACT: A Hybrid Method for Compression Test Data," *Proc. of VLSI Test Symposium,* pp. 62-69, 1998.

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

[5] J. Rajski, J. Tyszer, and N. Zaccharia, "Test Data Decompression for Multiple Scan Designs with Boundary Scan," *IEEE Trans. Computers,* pp. 1180-1200, Nov. 1998.

[6] S. Chakradhar and A. Raghunathan, "Bottleneck Removal Algorithm for Dynamic Compaction in Sequencial Circuits," *IEEE Trans. Computer-Aided Design,* 1997.

[7] A. Jas and N.A. Touba, "Using an Embedded Processor for Efficient Deterministic Testing of System-on-a-Chip," *Proc. of IEEE Int. Conf. on Computer Design (ICCD),* 1999.

[8] 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.

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

[10] A. Jas, K. Mohanram, and N.A. Touba, "An Embedded Core DFT Scheme to Obtain Highly Compressed Test Sets," *Proc. of IEEE Asian Test Symposium,* 1999.

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

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

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

[14] I. Hamzaoglu and J. H. Patel, "Test Set Compaction Algorithms for Combinational Circuits", *Proc. Int. Conf. Computer-Aided Design,* Nov. 1998.

[15] M. Schulz, E. Trischhler, and T. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Trans. Computer-Aided Design,* pp. 126-137, Jan. 1988.

[16] J. Chang and C. Lin, "Test Set Compaction for Combinational Circuits," *IEEE Trans. Computer Aided Design,* pp. 1370-1378, Nov. 1995.

[17] I. Pomeranz, L. Reddy, and S. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits," *Proc. of Int. Test Conference,* pp. 194-203, 1991.

[18] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. Reddy, "Cost-Effective Generation of Minimal Test sets for Stuck-at Faults in Combinational Circuits," *IEEE Trans. Computer Aided Design,* pp. 1496-1504, Dec. 1995.

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 Wisconsin, Computer Science and Information Systems Department., WI, USA
[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 geometric shapes based compression / decompression scheme that substantially reduces the amount of test data and hence reduces test time. The proposed scheme is based on reordering 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 gates 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

0-7803-7057-0/01/$10.00 ©2001 IEEE.

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

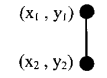
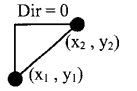
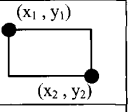
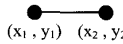
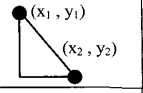
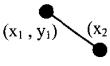
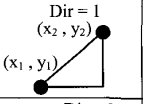
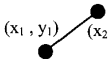
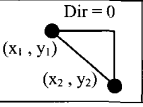
The proposed encoding/decoding algorithm is based on geometric shapes. In this work, we limited those shapes to the basic four namely: point, line, triangle, and rectangle as shown in Table 1. The choice of those shapes is made based on the following: (i) those shapes are bounded by a maximum of two 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, referre d 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 imme diate '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 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.

Table 1. The primitive geometric shapes.

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

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.

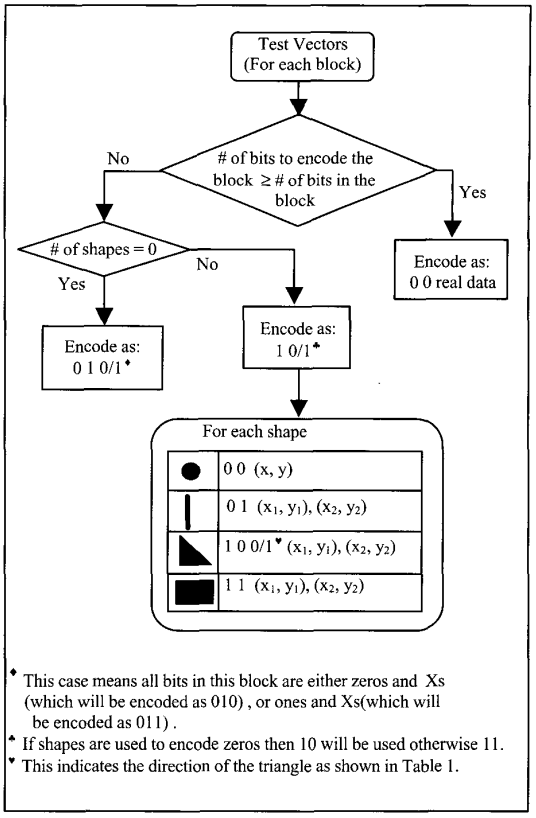


Figure 1. Schematic diagram of the encoding format.

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

$$Comp.Ratio = \frac{\#Original\ Bits - \#Compressed\ Bits}{\#Original\ Bits} X100$$

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 new fast geometric-shapes based compression/decompression scheme has been presented. In this scheme, the test data is first sorted so that we generate the minimum number of geometric shapes to be encoded in order to maximize the compression ratio. Then, the sorted data 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, the proposed technique achieved a very high compression ratio. Compared to compression results based on Golomb codes, our technique reduced the size of compressed bits by 20-78% as shown in Table 4. 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, "Tes ting 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.

Extended Frequency-Directed Run-Length Code with Improved Application to System-on-a-Chip Test Data Compression

Aiman H. El-Maleh and Raslan H. Al-Abaji

King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia
Email: {aimane, raslan}@ccse.kfupm.edu.sa

ABSTRACT

One of the major challenges in testing a System-on-a-Chip (SOC) is dealing with the large test data size. To reduce the volume of test data, several test data compression techniques have been proposed. Frequency-directed run-length (FDR) code is a variable-to-variable run length code based on encoding runs of 0's. In this work, we demonstrate that higher test data compression can be achieved based on encoding both runs of 0's and 1's. We propose an extension to the FDR code and demonstrate by experimental results its effectiveness in achieving higher compression ratio.

1. INTRODUCTION

Advances in VLSI technology have resulted in a change in the design paradigm where complete systems containing millions of transistors are integrated on a single chip. As the complexity of systems-on-a-chip continues to increase, the difficulty and cost of testing such chips is increasing rapidly [1-2]. One of the challenges in testing system-on-a-chip is dealing with the large size of test data that must be stored in the tester memory and transferred between the tester and the chip under test. The cost of automatic test equipment (ATE) increases significantly with the increase in their speed, channel capacity, and memory. Thus, to reduce the testing time and cost, it is necessary to reduce the volume of test data.

Test data reduction can be achieved by both test compaction [3-7] and compression [8-14]. Several test data compression techniques have been proposed in the literature. In [8], statistical coding is used for encoding test data based on a modified version of Huffman coding. In [9], efficient test data compression is achieved based on partitioning the test data into two-dimensional blocks and encoding each block separately based on geometric shapes. Another technique proposed in [10] uses what is called variable-to-block run-length coding. In this technique, a code word is used to encode a block of data based on the number of zeros followed by a one in that block. This technique is used for compressing fully specified test data that feeds a cyclical scan chain. A cyclical scan chain is used to decompress this data and transfer it to the "test scan chain". Golomb code is a variable-to-variable run-length code that is used in [11] 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. Another enhancement to the work done in [10] and [11] was proposed in [12]. It uses frequency-directed run-length (FDR) code, which is another variable-to-variable coding technique. It is designed based on the observation that the frequency of runs decreases with the increase in their lengths. Hence, assigning smaller code words to runs with small length and larger code words to those with larger length could result in higher test data compression.

The techniques in [10-12] are all based on encoding only runs of 0's. This was motivated based on the idea that encoding the difference vectors instead of the actual test vectors may reduce the number of 1's in the encoded data. However, it was demonstrated in [12] that, in general, better test data compression results are achieved, based on both FDR and Golomb codes, by encoding the actual test vectors. Based on test data analysis, we have observed that the frequency of runs of 1's is as significant as runs of 0's, for many of the circuits. This suggests that encoding both runs of 0's and 1's could result in higher test data compression. In this work, we propose an extension to the FDR code to encode the test data based on encoding both types of runs.

2. FREQUENCY-DIRECTED RUN-LENGTH (FDR) CODE

Many of the test data compression techniques are based on run-length coding. A run is a consecutive sequence of equal symbols. A sequence of symbols can be encoded using two elements for each run; the repeating symbol and the number of times it appears in the run. Frequency-directed run-length (FDR) code is a variable-to-variable coding technique based on encoding runs of 0's. In FDR code, the prefix and the tail of any codeword are of equal size. In any group A_i, the prefix is of size i bits. The prefix of a group is the binary representation of the run length of the first member of

0-7803-7596-3/02/$17.00 ©2002 IEEE

Table 1. FDR code.

Group	Run Length	Group Prefix	Tail	Code Word
A1	0	0	0	00
	1		1	01
A2	2	10	00	1000
	3		01	1001
	4		10	1010
	5		11	1011
A3	6	110	000	110000
	7		001	110001
	8		010	110010
	9		011	110011
	10		100	110100
	11		101	110101
	12		110	110110
	13		111	110111

that group. When moving from group A_i to group A_{i+1}, the length of the code words increases by two bits, one for the prefix and one for the tail. Runs of length i are mapped to group A_j, where $j = \lceil \log_2(i+3) \rceil - 1$. The size of the i'th group is equal to 2^i, i.e., group A_i contains 2^i members. The FDR code for the first three groups is shown in Table 1.

3. TEST DATA ANALYSIS

Based on test data analysis, it has been observed that test sets contain a large number of runs of 1's in addition to runs of 0's. By considering both types of runs, the total number of runs will decrease, which could result in higher test data compression.

To support this observation, we have analyzed test data for the largest ISCAS 85 and full-scanned versions of ISCAS 89 circuits. We have used the test sets generated by MinTest [5], using both static and dynamic compaction. Test sets generated by dynamic compaction option have the letter *d* appended in their name. All the test sets used achieve 100% fault coverage of the detectable faults in each circuit. Test sets generated based on static compaction were relaxed, as this has the advantage of keeping unnecessary assignments as X's, which enables higher compression.

Given a relaxed test set, techniques based on encoding only runs of 0's fill all the X's by 0's to reduce the number of runs that need to be encoded. However, to encode both runs of 0's and 1's in a test set, X's are filled by 1's if they are bounded by 1's from both sides, otherwise they are filled by 0's. This results in a reduction in the total number of runs that need to be encoded.

Table 2 shows the analysis of the number of runs on the used test sets. The first column indicates the circuit name. The second column shows the number of runs of 0's in the test set assuming that only runs of 0's will be encoded. The third, fourth, and fifth columns indicate the number of runs of 0's, runs of 1's, and the total number of runs, respectively, assuming that both types of runs

Table 2. Analysis of number of runs in test data.

		Encoding 0 Runs	Encoding 0 and 1 Runs		
Circuit	Original Bits	0 Runs	0 Runs	1 Runs	Total Runs
c2670	10252	1677	505	414	919
c5315	6586	1628	561	454	1015
c7552	15111	2695	652	1111	1763
s13207	163100	4804	2615	1157	3772
s15850	57434	4635	2514	1106	3620
s35932	21156	7554	1236	1071	2307
s38417	113152	20970	5331	3761	9092
s5378	20758	2915	1072	806	1878
s9234	25939	3843	1770	980	2750
s13207d	165200	5021	2581	1210	3791
s15850d	76986	5329	2644	1202	3846
s35932d	28208	10018	235	346	581
s38417d	164736	29473	5773	4834	10607
s38584d	199104	16814	7585	4074	11659
s5378d	23754	3537	1237	1001	2238
s9234d	39273	4816	2347	1212	3559

will be encoded. As can be seen from the table, for most of the circuits, the number of runs of 1's is as significant as the number of runs of 0's. For all the circuits, the total number of runs decreases and for some circuits the reduction is significant.

Figures 1, 2, and 3 show the frequency of both runs of 0's and runs of 1's for test sets of the circuits: s15850, s9234, and s35932d, respectively. As can be seen from the figures, the frequency of runs of 1's follow a similar shape to that of runs of 0's, although with a smaller

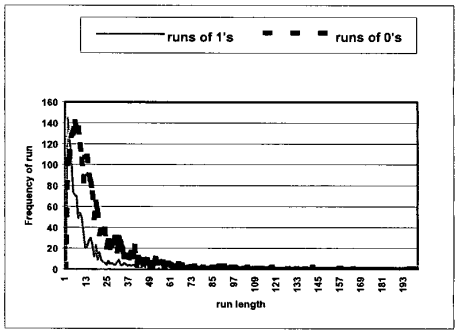


Figure 1. Distribution of runs of 0's and 1's for circuit s15850.

magnitude. For the circuit in Figure 1, it can be observed that there are more runs of 1's than 0's for run length < 5, but for run length > 5 there are more runs of 0's. For the circuit in Figure 2, we can see that runs of 0's with any length are on the average more that the runs of 1's with the same length. For the circuit in Figure 3, it can be observed that runs of 1's of small and large run length are more than those of 0's. But for middle run length ranges, the number of both 0 and 1 runs is comparable.

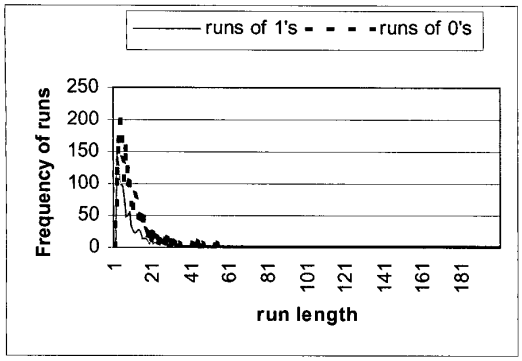


Figure 2. Distribution of runs of 0's and 1's for circuit s9234.

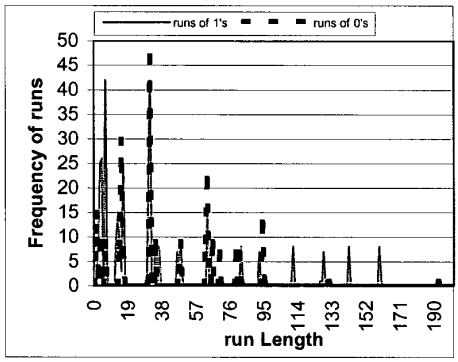


Figure 3. Distribution of runs of 0's and 1's for circuit s35932d.

4. EXTENDED FDR (EFDR) CODE

To encode both runs of 0's and 1's, we extend the FDR code based on adding an extra bit to the beginning of a code word to indicate the type of run. If the bit is 0, this indicates that the code word is encoding a run of type 0, otherwise it encodes a run of type 1. This code, called Extended FDR (EFDR), is shown in Table 3. It should be observed that this code is a direct extention to the FDR code shown in Table 1. However, in this code we do not have run length of size 0. This is because we are encoding both runs of 0's and runs of 1's. Note that runs of 0's are strings of 0's followed by a 1, while runs of 1's are strings of 1's followed by a 0, i.e. runs of 1's of length i are the complement of runs of 0's of the same length, and vice versa. As with FDR code, in this code when moving from group A_i to group A_{i+1}, the length of code words increases by two bits, one for the prefix and one for the tail. Runs of length i are mapped to group A_j, where $j = \lceil \log_2(i+2) \rceil - 1$. The size of the i'th group is equal to 2^{i+1}, i.e., group A_i contains 2^{i+1} members.

To illustrate the use of this code, let us consider an example. Consider the test T={01100011111111000000001}, of size 22 bits. The number of 0 runs in this test is 10. However, the number of both 0 and 1 runs is 5. Encoding this test using FDR code results in the encoded test T_{FDR}={01 00 1001 00 00 00 00 00 00 110010} of size 26 bits. Thus, for this example the number of bits needed to encode the test data using FDR code is more than the actual size of the original test data. However, encoding this test using EFDR code, we obtain the encoded test T_{EFDR}={000 100 001 11011 0110000}, of size 21 bits. Obviously, for this example EFDR code outperforms FDR code. Note that FDR code suffers whenever we have runs of 1's, as each 1 bit will be encoded by a separate 0 run of length 0.

Table 3. Extended FDR (EFDR) code.

Group	Run Length	Group Prefix	Tail	Code Word Runs of 0's	Code Word Runs of 1's
A1	1	0	0	000	100
	2		1	001	101
A2	3	10	00	01000	11000
	4		01	01001	11001
	5		10	01010	11010
	6		11	01011	11011
A3	7	110	000	0110000	1110000
	8		001	0110001	1110001
	9		010	0110010	1110010
	10		011	0110011	1110011
	11		100	0110100	1110100
	12		101	0110101	1110101
	13		110	0110110	1110110
	14		111	0110111	1110111

5. EXPERIMENTAL RESULTS

Table 4 compares the compression results using the FDR and EFDR code. The first column shows the circuit name and the second column shows the size of the test set in bits. The third and fourth columns show the number of compressed bits using FDR and EFDR codes, respectively. The last two columns indicate the respective compression ratios. The *compression ratio* is computed as:

$$Comp.Ratio = \frac{\#Original\ Bits - \#Compressed\ Bits}{\#Original\ Bits} X100$$

As can be seen from the table, significant improvements in the compression ratio are obtained for some of the circuits. Consider for example the circuit s35932. For the first test set of this circuit, the compression ratio improves from 3.99% using FDR to 45.63% using EFDR code. For the second test set of the same circuit, the compression ratio increases from 19.36% using FDR to 80.31% using EFDR code. This result is not surprising as based on the statistics for this circuit given in Table 2, the total number of runs reduces significantly when both types of runs are used versus using only 0 runs. Similarly, significant increase in the compression ratio is obtained for the test sets c2670, c5315, s38417, and s38417d. For

Table 4. Compression results of FDR & EFDR codes.

Circuit	Original Bits	FDR Bits	EFDR Bits	FDR CR	EFDR CR
c2670	10252	5760	4807	43.82	**53.11**
c5315	6586	5238	4700	20.47	**28.64**
c7552	15111	9500	8843	37.13	**41.48**
s13207	163100	34608	33637	78.78	**79.38**
s15850	57434	24992	25105	**56.49**	56.29
s35932	21156	20312	11502	3.99	**45.63**
s38417	113152	70536	53914	37.66	**52.35**
s5378	20758	11032	10210	46.85	**50.81**
s9234	25939	16912	16127	34.80	**37.83**
s13207d	165200	30880	29992	81.31	**81.85**
s15850d	76986	26016	24643	66.21	**67.99**
s35932d	28208	22746	5554	19.36	**80.31**
s38417d	164736	93452	64962	43.27	**60.57**
s38584d	199104	77798	73853	60.93	**62.91**
s5378d	23754	12356	11419	47.98	**51.93**
s9234d	39273	22148	21250	43.61	**45.89**

all the test sets except one, using EFDR code achieves higher compression ratio.

For test data decompression based on EFDR code, the decoder design follows a direct extention of the FDR decoder proposed in [12].

6. CONCLUSION

In this work, we have proposed an extension to the recently proposed FDR code, namely Extended FDR (EFDR) code. The proposed technique is based on encoding both runs of 0's and 1's as opposed to encoding only runs of 0's. Based on experimental results on ISCAS benchmark circuits, it has been demonstrated that the proposed EFDR code outperformed FDR code and resulted in significant increase in test data compression ratio for several circuits, improving the compression ratio from 19.36% to 80.31% for one of the benchmark circuits.

ACKNOWLEDGMENT

This project is supported by King Fahd University of Petroleum & Minerals under project FT2000/07.

REFERENCES

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

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

[3] M. Schulz, E. Trischhler, and T. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Trans. Computer-Aided Design*, pp. 126-137, Jan. 1988.

[4] I. Pomeranz, L. Reddy, and S. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits," *Proc. of Int. Test Conference*, pp. 194-203, 1991.

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

[6] J. Chang and C. Lin, "Test Set Compaction for Combinational Circuits," *IEEE Trans. Computer Aided Design*, pp. 1370-1378, Nov. 1995.

[7] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. Reddy, "Cost-Effective Generation of Minimal Test sets for Stuck-at Faults in Combinational Circuits," *IEEE Trans. Computer Aided Design*, pp. 1496-1504, Dec. 1995.

[8] A. Jas, J.G. Dastidar and N.A. Touba, "Scan Vector Compression/ Decompression Using Statistical Coding," *Proc. of IEEE VLSI Test Symp., pp. 114-120,* 1999.

[9] A. El-Maleh, S. Al-Zahir, and E. Khan, "A Geometric-Primitives-Based Compression Scheme for Testing Systems-on-a-Chip," *Proc. of IEEE VLSI Test Symp.*, pp. 54-59, 2001.

[10] 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.

[11] 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.

[12] Chandra, A. and Chakrabarty, K., "Frequency-Directed Run-Length (FDR) Codes with Application to Systems-on-a-Chip Test Data Compression," *Proc. of IEEE VLSI Test Symp.,* pp. 42-47, 2001.

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

[14] A. Jas and N.A. Touba, "Using an Embedded Processor for Efficient Deterministic Testing of System-on-a-Chip," *Proc. of IEEE Int. Conf. on Computer Design (ICCD),* 1999.

A Hybrid Test Compression Technique for Efficient Testing of Systems-on-a-Chip

Aiman H. El-Maleh

King Fahd University of Petroleum & Minerals, P.O. Box 1063, Dhahran, 31261, Saudi Arabia
Email: aimane@ccse.kfupm.edu.sa

ABSTRACT

One of the major challenges in testing a System-on-a-Chip (SOC) is dealing with the large test data size. To reduce the volume of test data, several efficient test data compression techniques have been recently proposed. In this paper, we propose hybrid test compression techniques that combine the Geometric-Primitives-Based compression technique with the frequency-directed run-length (FDR) and extended frequency-directed run-length (EFDR) coding techniques. Based on experimental results, we demonstrate the effectiveness of the proposed hybrid compression techniques in increasing the test data compression ratios over those obtained by the Geometric-Primitives-Based compression technique.

1. INTRODUCTION

With today's technology, it is possible to build complete systems containing millions of transistors on a single chip. One of the major challenges in testing SOC is dealing with the large size of test data that must be stored in the tester and transferred between the tester and the chip [1]. The cost of automatic test equipment (ATE) increases significantly with the increase in their speed, channel capacity, and memory. Thus, to reduce the testing cost, the need for test data reduction becomes imperative. To achieve such reduction, several test compaction and lossless test compression schemes are used.

In test data compression, the objective is to reduce the number of bits needed to represent the test data. Several test data compression techniques have been proposed in the literature [2-6]. One of these techniques is proposed in [2] and uses what is called variable-to-block run-length coding. In this technique, a code word is used to encode a block of data based on the number of zeros followed by a one in that block. This technique is used for compressing fully specified test data that feeds a cyclical scan chain. A cyclical scan chain is used to decompress this data and transfer it to the "test scan chain". Golomb code is a variable-to-variable run-length code that is used in [3] to enhance the scheme described in [2]. 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. Another enhancement to the work done in [2] and [3] was proposed in [4]. It uses frequency-directed run-length (FDR) codes, which is another variable-to-variable coding technique. It is designed based on the observation that the frequency of runs decreases with the increase in their lengths. Hence, assigning smaller code words to runs with small length and larger code words to those with larger length could result in higher test data compression.

The techniques in [2-4] are all based on encoding only runs of 0's. This was motivated based on the idea that encoding the difference vectors instead of the actual test data may reduce the number of 1's in the encoded data. However, it was demonstrated in [4] that, in general, better test data compression results are achieved, based on both FDR and Golomb codes, by encoding the actual test data. Based on test data analysis in [6], it was observed that, in the actual test data, the frequency of runs of 1's is as significant as runs of 0's, for many of the circuits. Hence, to encode the test data based on both types of runs, extended frequency directed run-length (EFDR) codes were proposed in [6], that result in higher test data compression.

Recently, an efficient test data compression technique that gives high compression ratios, the Geometric-Primitives-Based compression technique [5], has been proposed. It is based on partitioning the test data into two-dimensional blocks and encoding each block separately based on geometric shapes. In the Geometric-Primitives-Based Compression technique, some of the blocks are encoded by storing the real test data because the encoded block size is larger than the actual test data block size. So, reducing the number of these blocks could result in higher test data compression.

In this paper, we propose hybrid test data compression techniques that exploit the use of either FDR or EFDR codes to reduce the number of blocks that are not encoded by the geometric shapes and encoded by storing the real test data. We demonstrate based on experimental results the effectiveness of the proposed hybrid compression technique in increasing the test data compression ratios over those obtained by the Geometric-Primitives-Based compression technique.

2. GEOMETRIC-BASED COMPRESSION

The Geometric-Primitives-Based Compression technique is based on encoding the 0's or the 1's in a test set by geometric shapes. In this technique, the number of shapes are limited to the basic four shaps, namely: point, line, triangle, and rectangle as shown in Table 1. For the rectangles, two points are needed to encode the shape and each point costs $2*\log_2 N$ bits, where N is the block dimension. However, lines and triangles are represented by a point and a distance d. Two bits are used to determine the type of line or the type of triangle encoded. In this technique, the test vectors are first sorted to generate clusters of either 0's or 1's in such a way that it may partially or totally be fitted in one or more of the geometric shapes shown in Table1. Then, the test set is partitioned into blocks each of which is NxN bits. For test vectors whose columns and/or rows are not divisible by the predetermined block dimension 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 number of vectors, the vector length, and the block dimension, the number of bits used to encode the coordinates of the geometric shapes can be less than $log_2 N$. The decoder recognizes those special cases and decodes them properly.

0-7803-8163-7/03/$17.00 © 2003 IEEE

Table 1. The used primitive geometric shapes.

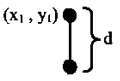
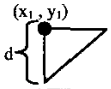
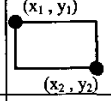
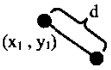
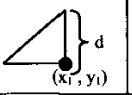
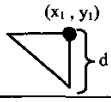
	Lines	Triangles	Rectangle
Type 1			
Type 2			X
Type 3			X
Type 4			X

Table 2. Geometric compression block encoding format.

Header Code	Encode Block
00	with real test data
010	as filled with 0's
011	as filled with 1's
10	with geometric shapes covering 0's
11	with geometric shapes covering 1's

The encoding process is then applied on each block independently. There are three cases that may occur:

(i) The block contains only 0's and X's, or 1's and X's. In this case, the block is encoded by the code 01 followed by the bit that fills the block.

(ii) The block needs to be encoded by a number of shapes. In this case, two bits are needed to indicate the existence of shapes and the type of bit encoded. If the encoded bit is 0, then the code is 10, otherwise it is 11.

(iii) The number of bits needed to encode the shapes is greater than the total number of bits in the block. In this case, the block is encoded by storing the real data. The real data is stored after a two-bit code (00).

The block encoding format is summarized in Table 2.

3. FDR COMPRESSION

Frequency-directed run-length (FDR) code [4] is a variable-to-variable coding technique based on encoding runs of 0's. In FDR code, the prefix and the tail of any codeword are of equal size. In any group A_i, the prefix is of size i bits. The prefix of a group is the binary representation of the run length of the first member of the group. When moving from group A_i to group A_{i+1}, the length of the code words increase by two bits, one for the prefix and one for the tail. The FDR code for the first three groups is shown in Table 3. Since the FDR technique is based on encoding only runs of 0's, all the X's in a test set will be filled by 0's to reduce the number of runs that need to be encoded.

Table 3. FDR code.

Group	Run Length	Group Prefix	Tail	Code Word
A1	0	0	0	00
	1		1	01
A2	2	10	00	1000
	3		01	1001
	4		10	1010
	5		11	1011
A3	6	110	000	110000
	7		001	110001
	.....		.....	.....
	13		111	110111

Table 4. Extended FDR (EFDR) code.

Group	Run Length	Group Prefix	Tail	Code Word Runs of 0's	Code Word Runs of 1's
A1	1	0	0	000	100
	2		1	001	101
A2	3	10	00	01000	11000
	4		01	01001	11001
	5		10	01010	11010
	6		11	01011	11011
A3	7	110	000	0110000	1110000
	8		001	0110001	1110001
	.....		.....	.....	.....
	14		111	0110111	1110111

4. EXTENDED FDR (EFDR) COMPRESSION

Based on test data analysis in [6], it has been observed that test sets contain a large number of runs of 1's as well as runs of 0's. By encoding both types of runs, the total number of runs will decrease, which could result in higher test data compression. To encode both runs of 0's and 1's, the extended FDR technique [6] is used by adding an extra bit to the beginning of a code word to indicate the type of run. If the bit is 0, this indicates that the code word is encoding a run of type 0, otherwise it encodes a run of type 1. The EFDR code for the first three groups is shown in Table 4. Unlike the FDR code, the EFDR code does not have run length of size 0. This is because both runs of 0's and 1's are encoded. Runs of 0's are strings of 0's followed by a 1, while runs of 1's are strings of 1's followed by a 0. Since the EFDR technique encodes both types of runs, the X's are filled with 0 except when the X's are bounded by 1 from both sides, they are filled with 1.

5. HYBRID TEST COMPRESSION SCHEME

As it was mentioned before, in the Geometric-Primitives-Based compression technique there are some blocks which are encoded by storing the real test data. This is because the size of these blocks when they are encoded is larger than their original size. So, no compression is achieved for such blocks. In order to reduce the number of these blocks, we propose to combine the Geometric-Primitives-Based compression technique with either the FDR or the EFDR compression techniques. In this case, the

Table 5. Geometric-FDR (GFDR) & Geometric-EFDR (GEFDR) compression block encoding format.

Header Code	Encode Block
000	with real test data
001	with FDR (EFDR) codes
010	as filled with 0's
011	as filled with 1's
10	with geometric shapes covering 0's
11	with geometric shapes covering 1's

FDR or EFDR techniques are applied to encode a block. The block is encoded with these techniques if its encoding size is less than the encoding size with geometric shapes. The block encoding format for the hybrid technique combining the geometric and FDR compression techniques, called GFDR, is shown in Table 5. Note that the difference between this encoding scheme and the Geometric encoding scheme is in the header code starting with 00. So, blocks that will still be encoded with real test data will have an extra bit in the header. The other blocks have exactly the same format. The block encoding format for the hybrid technique combining the geometric and EFDR compression techniques, called GEFDR, is similar to GFDR with the difference of using EFDR instead of FDR.

Test data decompression will be done on chip and the decoded test will then be applied to the chip under test. The decoders for the proposed hybrid techniques are a direct combination of the decoders for the Geometric [5], FDR[4], and EFDR [6] techniques.

6. EXPERIMENTAL RESULTS

In order to demonstrate the effectiveness of the proposed hybrid compression schemes, we have performed experiments on a number of the largest ISCAS85 and full-scanned versions of ISCAS89 benchmark circuits. We have used the test sets generated by MinTest [7], using both static and dynamic compaction. Test sets generated by the dynamic compaction option have the letter *d* appended in their name. All the test sets used achieve 100% fault coverage of the detectable faults in each circuit. Test sets generated based on static compaction were relaxed, as this has the advantage of keeping unnecessary assignments as X's, which enables higher compression.

The test sets were partitioned into 8x8, 16x16, and 32x32 blocks, respectivly. Then, the hybrid compression schemes, GFDR and GEFDR, are applied for each case separetly. Table 6 shows the compression ratios obtained for five compression schemes namely, geometric, FDR, EFDR, GFDR, and GEFDR, respectively. The best result from the three block sizes is reported for each case.

The *compression ratio* is computed as:

$$Comp.\ Ratio = \frac{\#\ Original\ Bits - \#\ Compressed\ Bits}{\#\ Original\ Bits} X100 \quad (1)$$

As can be seen from the table, the two hybrid compression techniques, GFDR and GEFDR, both improved the compression ratio over the Geometric compression technique for all the circuits. However, the GEFDR compression scheme achieved better results and improved the compression ratio on average from 59.06% to 62.13%. Among the five compared compression schemes, the GEFDR compression scheme achieved the best results in 9 out of 14 test sets. However, the GFDR compression scheme achieved the best results in 3 out of the 14 test sets. The best compression ratio for the remaining test sets is achived by the EFDR compression technique.

Table 7 shows a detailed analysis of the number blocks encoded by the different encoding formats for the Geometric, GFDR, and GEFDR compression schems. This analysis is shown for an 8x8 block size. The second column shows the total number of encoded blocks. The third colums shows the number of blocks encoded as a block filled with either 0 or 1. The fourth and fifth columns show the number of blocks encoded by geometric shapes and those encoded by the real test data, respectively for the Geometric compression scheme. The sixth, seventh and eighth columns show the number of blocks encoded by geometric shapes, those encoded by FDR codes, and those encoded by the real test data, respectively for the GFDR compression scheme. Similarly, the last three columns show the number of blocks encoded by geometric shapes, those encoded by EFDR codes, and those encoded by the real test data, respectively for the GEFDR compression scheme. As can be seen from the table, both the GFDR and GEFDR compression schemes reduce the number of blocks encoded by the real test data and hence improve the compression ratio. For the circuits considered, the average number of real blocks is 15.16% for the Geometric compression scheme, 10.02% for the GFDR compresion scheme, and 7.37% for the GEFDR technique. Thus, the GEFDR compression technique reduces the number of real blocks by more than 50%. As indicated by the results, there is still a percentage of blocks that achieve no compression and are encoded by storing the real test data.

The average number of blocks encoded by FDR codes in the GFDR technique is 12.23% and the average number of blocks encoded by EFDR codes in the GEFDR technique is 17.5%. This indicates that these blocks achieve better compression if encoded by these codes rather than by geometric shapes, which adds to the benefit of the proposed hybrid compression schemes.

7. CONCLUSION

In this work, we have proposed two hybrid compression schemes that combine the Geometric and FDR compression schemes (GFDR), and the Geometric and EFDR compression schemes (GEFDR). The objective of these schemes is to reduce the number of blocks in the Geometric compression scheme that are encoded with the actual test data. Based on experimental results on ISCAS benchmark circuits, it has been demonstrated that the proposed hybrid compression schemes improved the test data compression ratio for all the circuits over those obtained by the Geometric compression scheme. The GEFDR technique achieved the best results and improved the compression ratio on average from 59.06% to 62.13% over the Geometric compression scheme.

ACKNOWLEDGMENT

This work is supported by King Fahd University of Petroleum & Minerals under project FT2000-07. The author would like to thank Mr. Faisal Ba Haiderah for his help in the implmentation of this work.

Table 6. Compression results of Geometric, FDR, EFDR, GFDR, and GEFDR techiques.

Circuit	Test Set Size	Geometric CR	FDR CR	EFDR CR	GFDR CR	GEFDR CR
c2670	10252	51.85	43.82	53.11	54.14	**54.56**
c5315	6586	27.88	20.47	28.64	29.03	**29.21**
s13207	163100	85.01	78.78	79.38	**85.48**	85.40
s15850	57434	60.32	56.49	56.29	**61.70**	61.43
s35932	21156	25.78	3.99	**45.63**	26.27	44.93
s38417	113152	46.50	37.66	**52.35**	48.37	51.45
s5378	20758	51.55	46.85	50.81	53.12	**53.18**
s13207d	165200	86.63	81.31	81.85	87.60	**87.74**
s15850d	76986	70.19	66.21	67.99	71.21	**71.42**
s35932d	28208	78.12	19.36	80.31	78.12	**81.71**
s38417d	164736	62.23	43.27	60.57	63.09	**65.23**
s38584d	199104	65.59	60.93	62.91	66.30	**67.03**
s5378d	23754	57.94	47.98	51.93	58.32	**58.62**
s9234d	39273	57.22	43.61	45.89	**58.39**	57.87
AVG		59.06	46.48	58.40	60.08	62.13

Table 7. Detailed analysis of block encodings for Geometric, GFDR, and GEFDR compression schemes.

Circuit	Geometric				GFDR			GEFDR		
	#Blocks	#Filled	#Shapes Encoded	#Real	#Shapes Encoded	#FDR Encoded	#Real	#Shapes Encoded	#EFDR Encoded	#Real
c2670	180	56	99	25	70	33	21	68	40	16
c5315	115	8	61	46	52	22	33	51	23	33
s13207	2640	1671	963	6	895	72	0	906	62	1
s15850	924	127	787	10	677	120	0	698	97	2
s35932	442	76	191	175	182	54	130	129	196	41
s38417	1872	252	1484	136	1214	348	58	1059	534	27
s5378	351	73	220	58	193	46	39	185	63	30
s13207d	2640	2041	531	68	487	76	36	487	87	25
s15850d	1232	614	536	82	470	110	38	481	112	25
s35932d	442	56	384	2	384	0	2	334	50	2
s38417d	2704	1068	1447	189	1290	220	126	1129	453	54
s38584d	3111	1180	1584	347	1413	284	234	1442	332	157
s5378d	378	143	157	78	142	24	69	134	43	58
s9234d	620	150	410	60	367	71	32	375	60	35
AVG(%)		28.97	55.87	15.16	48.78	12.23	10.02	46.16	17.50	7.37

REFERENCES

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

[2] 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.

[3] 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.

[4] A. Chandra and K. Chakrabarty, "Frequency-Directed Run-Length (FDR) Codes with Application to Systems-on-a-Chip Test Data Compression," *Proc. of IEEE VLSI Test Symp.,* pp. 42-47, 2001.

[5] A. El-Maleh, S. Al-Zahir, and E. Khan, "A Geometric-Primitives-Based Compression Scheme for Testing Systems-on-a-Chip," *Proc. of IEEE VLSI Test Symp.*, pp. 54-59, 2001.

[6] A. El-Maleh and R. Al-Abaji, "Extended Frequency-Directed Run-Length Codes with Improved Application to System-on-a-Chip Test Data Compression", *Proc. of IEEE Int. Conf. Electronics, Circuits and System, pp. 449-452, 2002.*

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