

Multilevel Huffman Coding: An Efficient Test-Data Compression Method for IP Cores

Xrysovalantis Kavousianos, *Member, IEEE*, Emmanouil Kalligeros, *Member, IEEE*, and Dimitris Nikolos, *Member, IEEE*

Abstract—A new test-data compression method suitable for cores of unknown structure is introduced in this paper. The proposed method encodes the test data provided by the core vendor using a new, very effective compression scheme based on multilevel Huffman coding. Each Huffman codeword corresponds to three different kinds of information, and thus, significant compression improvements compared to the already known techniques are achieved. A simple architecture is proposed for decoding the compressed data on chip. Its hardware overhead is very low and comparable to that of the most efficient methods in the literature. Moreover, the major part of the decompressor can be shared among different cores, which reduces the hardware overhead of the proposed architecture considerably. Additionally, the proposed technique offers increased probability of detection of unmodeled faults since the majority of the unknown values of the test sets are replaced by pseudorandom data generated by a linear feedback shift register.

Index Terms—Embedded testing techniques, Huffman encoding, intellectual property (IP) cores, linear feedback shift registers (LFSRs), test-data compression.

I. INTRODUCTION

THE INCREASED complexity of contemporary systems-on-a-chip (SoCs), the tight time-to-market constraints, and the lack of expertise in designing some building blocks are some of the reasons that make the use of predesigned and preverified modules (cores) during the development of an integrated circuit necessary. This high level of integration, although reducing the chips' production cost, makes their testing an increasingly challenging task. A large amount of data must be stored in the tester [automatic test equipment (ATE)] and then transferred as fast as possible deep into the chip.

Manuscript received February 8, 2006; revised June 12, 2006 and August 10, 2006. The research Project was co-funded by the European Union—European Social Fund (ESF) & National Sources, in the framework of the program “Pythagoras II” of the “Operational Program for Education and Initial Vocational Training” of the 3rd Community Support Framework of the Hellenic Ministry of Education. This paper was presented in part at the *Proceedings of the Design Automation and Test in Europe Conference*, March 2006. This paper was recommended by Associate Editor S. Hellebrand.

X. Kavousianos is with the Computer Science Department, University of Ioannina, 45110 Ioannina, Greece (e-mail: kabousia@cs.uoi.gr).

E. Kalligeros is with the Computer Science Department, University of Ioannina, 45110 Ioannina, Greece, with the Computer Science and Technology Department, University of Peloponnese, Terma Karaiskaki, 22100 Tripoli, Greece, and also with the Computer Engineering and Informatics Department, University of Patras, 26500 Patras, Greece (e-mail: kalliger@ceid.upatras.gr).

D. Nikolos are with the Computer Engineering and Informatics Department, University of Patras, 26500 Patras, Greece (e-mail: nikolosd@cti.gr).

Digital Object Identifier 10.1109/TCAD.2006.885830

However, the limited channel capacity, memory, and speed of ATEs, as well as the reduced accessibility of some of the inner nodes of dense SoCs, render the above requirements infeasible. Therefore, test-data volume, testing time, and test applicability are major concerns from a test economics point of view [19], [46].

Embedded testing has been proposed to ease the burden of testing on ATEs, as well as to provide the required accessibility to cores deeply embedded in a SoC's structure. It combines the ATE capabilities with on-chip integrated structures. Embedded testing techniques store a compressed version of the test set on the ATE, which is then downloaded and decompressed on chip. Various embedded testing approaches have been proposed in the literature, which use, among others, combinational continuous-flow linear decompressors [3], [38], the REusing Scan chains for test Pattern decompressIoN (RESPIN) architecture [10], the Illinois scan architecture [14], [20], linear feedback shift register (LFSR)-based decompressors [17], [30], [32], [34], [39], [53], folding counters [18], and weighted random pattern generators [26], [29]. Also, commercial tools exist, which automate the generation of embedded decompressors [2], [36], [47], [48].

Most of the aforementioned techniques need structural information of the core under test (CUT) or require the synergy of the automatic test pattern generation (ATPG) tool [34], [53]. From the cores incorporated in a SoC, some may be in-house designed reusable blocks, whereas others can be third-party intellectual property (IP) cores. The structure of IP cores is often hidden from the system integrator, and only a precomputed test set is provided by their vendor. Several methods have been proposed to cope with testing of IP cores of unknown structure. Some of them embed the precomputed test vectors in longer pseudorandom sequences, which are generated on chip [4], [33], [41], [50]. The main drawback of these techniques is their long test-application time. To reduce both the test-data volume and test-application time, many methods encode directly the test set using various compression codes. In this case, useless vectors are not applied to the CUT. Golomb codes were proposed in [5]–[7] and [49], alternating run-length codes in [8], frequency-directed run-length codes in [9] and [44], statistical codes in [12], [24], and [28], a nine-coded technique in [52], and combinations of codes in [37], [45], and [51]. Some techniques use dictionaries [35], [40], [54], [55] but suffer from increased hardware overhead due to the large embedded random access memories they require, and for that reason, they are not considered further in this paper.

Among the statistical codes used for test-data compression, Huffman are the most effective ones since they provably provide the shortest average codeword length [22]–[24], [28], [31]. Their main problem is the high hardware overhead of the required decompressors. For that reason, selective Huffman coding was proposed in [28], which significantly reduces the decoder size by slightly sacrificing the compression ratio.

Compression is sometimes performed in the difference vectors instead of the actual test vectors [5], [7], [9], [25]. This is motivated by the observation that test vectors are usually correlated (differ in a small number of bits), and therefore, their difference vectors will have long runs of 0s, which can be effectively compressed by run-length codes. However, when difference vectors are used, either cyclical shift registers, which increase the testing cost (especially for cores with a large number of scan cells), should be incorporated in the system, or the scan chains of other cores must be reused, if they are available and the power constraints of the system are not violated.

Moreover, there is a class of techniques that requires the preexistence of arithmetic modules or processors in the system [1], [11], [16], [21], [27], [43].

The high efficiency of all the aforementioned compression methods is mainly due to the large number of x values in the test sets. Traditionally, ATPG tools fill these x values randomly with logic 0 or 1, so as to improve the coverage of unmodeled faults. On the contrary, compression techniques, in order to achieve high compression ratios, replace all these x values with the same logic value (0 or 1), depending on the characteristics of the implemented code. For that reason, the unmodeled fault coverage may be reduced. In [52], it is suggested that, if possible, at least a portion of a test set's x values should be set randomly.

In this paper, we propose a statistical compression method based on Huffman coding, which fills the majority of a test set's x values randomly. This random filling is achieved by using a small LFSR. The proposed method improves the compression ratio by using multilevel Huffman coding (compression of different kinds of information with the same codewords) and, at the same time, requires a very simple decompressor with low hardware overhead. It also offers the ability of exploiting the tradeoff between compression ratio and area overhead. The proposed approach does not need any structural information of the CUT and is therefore proper for IP cores of unknown structure. Additionally, it does not require the incorporation of any special arithmetic modules, processors, or cyclical shift registers in the system and does not apply any useless vectors to the CUT. Note that although the combination of LFSRs with Huffman encoding has been used in the literature [37] (LFSR reseeding with seed compression), in this paper, a totally different approach is proposed.

The rest of this paper is organized as follows: Section II reviews Huffman coding, Section III describes the proposed compression method, and Section IV presents the decompression architecture. In Section V, we calculate the required test-application time and the achieved test-time reduction as compared to the no-compression case. Experimental results and comparisons are provided in Section VI, whereas conclusions are provided in Section VII.

II. HUFFMAN ENCODING PRELIMINARIES

Statistical codes represent data blocks of fixed length with variable-length codewords. The efficiency of a statistical code depends on the frequency of occurrence of all distinct fixed-length blocks in a set of data. The most frequently occurring blocks are encoded with short codewords, whereas the less frequently occurring ones are encoded with large codewords. In this way, the average codeword length is minimized. It is obvious however that, if all distinct blocks in a data set appear with the same (or nearly the same) frequency, then no compression can be achieved. Among all statistical codes, Huffman offer the best compression since they provably provide the shortest average codeword length. Another advantageous property of a Huffman code is that it is prefix free; i.e., no codeword is the prefix of another one. This makes the decoding process simple and easy to implement.

Let T be the fully specified test set of an IP core [a fully specified test set contains no don't care (x) bits]. Let us also assume that if we partition the test vectors of T into blocks of length l , we get k distinct blocks b_1, b_2, \dots, b_k with frequencies (probabilities) of occurrence p_1, p_2, \dots, p_k , respectively. The entropy of the test set is defined as $H(T) = -\sum_{i=1}^k p_i (\log_2 p_i)$ and corresponds to the minimum average number of bits required for each codeword. The average codeword length of a Huffman code is closer to the aforementioned theoretical entropy bound compared to any other statistical code. In practice, test sets have many don't care (x) bits. In a good encoding strategy, the don't cares must be assigned such that the entropy value $H(T)$ is minimized. In other words, the assignment of the test set's x values should skew the occurrence frequencies of the distinct blocks as much as possible. We note that the inherent correlation of the test cubes of T (test vectors with x values) favors the targeted occurrence frequency skewing and, consequently, the use of statistical coding.

To generate a Huffman code, we create a binary tree. A leaf node is generated for each distinct block b_i , and a weight equal to the occurrence probability of block b_i is associated with the corresponding node. The pair of nodes with the smallest weights is selected first, and a parent node is generated with weight equal to the sum of the weights of these two nodes. The previous step is repeated iteratively, selecting each time the node pair with the smallest sum of weights, until only a single node is left unselected, i.e., the root (we note that each node can be chosen only once). Starting from the root, we visit all the nodes once, and we assign to each left-child edge the logic 0 value and to each right-child edge the logic 1 value. The codeword of block b_i is the sequence of the logic values of the edges belonging to the path from the root to the leaf node corresponding to b_i . If c_1, c_2, \dots, c_k are the codeword lengths of blocks b_1, b_2, \dots, b_k , respectively, then the average codeword length is $C(T) = \sum_{i=1}^k p_i c_i$.

Example 1 (Huffman Encoding): Consider the test set shown in column 1 of Table I, which consists of four test vectors of length 16 (64 bits overall). If we partition the test vectors into blocks of length 4, we get 16 occurrences of five distinct blocks. Columns 2 and 3 present those five blocks and their frequencies of occurrence, respectively. Column 4 presents the codeword of

TABLE I
HUFFMAN ENCODING EXAMPLE

Test Set T	Block	Occ. Frequency	Codeword
	1010	7/16	0
0000 1010 1111 1010	0000	5/16	10
1111 0000 1010 0001	1111	2/16	110
1010 0000 0010 1010	0001	1/16	1110
0000 1010 0000 1010	0010	1/16	1111

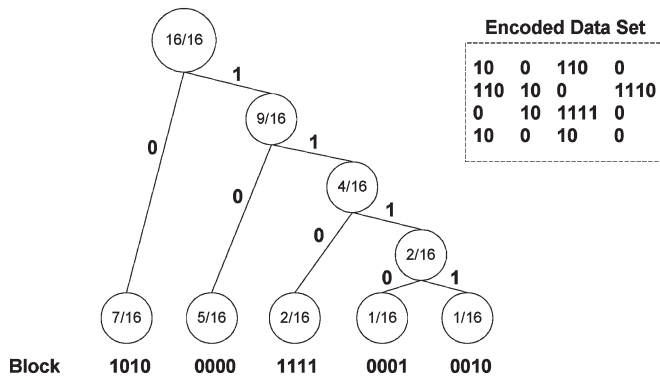


Fig. 1. Huffman tree of Example 1.

each block (the corresponding Huffman tree and the encoded data set are shown in Fig. 1). The size of the encoded data set is 31 bits, and the average codeword length is

$$C(T) = 1 \cdot \frac{7}{16} + 2 \cdot \frac{5}{16} + 3 \cdot \frac{2}{16} + 4 \cdot \frac{1}{16} + 4 \cdot \frac{1}{16} = 1.9375.$$

Note that the entropy is equal to

$$\begin{aligned} H(T) &= -\left(\frac{7}{16} \cdot \log_2 \frac{7}{16} + \frac{5}{16} \cdot \log_2 \frac{5}{16} \right. \\ &\quad \left. + \frac{2}{16} \cdot \log_2 \frac{2}{16} + \frac{1}{16} \cdot \log_2 \frac{1}{16} + \frac{1}{16} \cdot \log_2 \frac{1}{16} \right) \\ &= 1.9218. \end{aligned}$$

The size of a Huffman decoder depends on the number of distinct blocks that are encoded. Increased encoded-block volumes lead to big decoders due to the big size of the corresponding Huffman tree. For that reason, a selective Huffman approach was adopted in [28], according to which only the most frequently occurring blocks are encoded, whereas the rest are not. An extra bit is appended to every block, indicating if it is encoded or not.

The encoding procedure of this paper combines LFSR-generated pseudorandom sequences with multilevel selective Huffman coding. As far as the selective Huffman part is concerned, the proposed approach differs from that of [28] in two main points.

- 1) The utilized Huffman code is specified by taking into account the occurrence frequency of multiple kinds of information and not only that of the data blocks as in [28]. As a consequence, the proposed compression scheme is more sophisticated and hence much more effective (higher compression is achieved since the same Huffman

codewords are used for encoding three different kinds of information and not only data blocks).

- 2) The unencoded data blocks are indicated by using a separate Huffman codeword instead of appending an extra bit to every block either encoded or not. This separate codeword precedes only each unencoded block.

III. COMPRESSION METHOD

As mentioned above, the proposed compression method exploits pseudorandom LFSR sequences and performs multilevel Huffman coding with a limited number of codewords (selective). The test cubes of the CUT are compared against the bit sequences generated by various cells of an LFSR, and if they match (i.e., they are compatible), an appropriate cell is chosen for feeding the scan chain(s) of the CUT. What is actually coded is an index for each selected LFSR cell; i.e., each Huffman codeword is used for enabling a specific LFSR cell to feed the scan chain(s). If no match with an LFSR cell sequence can be found, then the test data are directly encoded using a selective Huffman code. Direct test-data coding is most of the times used for portions of the test cubes with many defined bits, which are expected to be incompatible with the LFSR's pseudorandom sequences. On the other hand, the major part of the test data encoded by LFSR cells corresponds to the test cubes' x -bit sequences. Therefore, most of the cubes' x values are replaced by pseudorandom data, which increases the probability of detection of unmodeled faults. In the following, we describe the proposed compression method assuming a single scan chain.

A. Encoding Procedure Overview

At first, the CUT's test cubes are concatenated so as to form a single bit stream, which is partitioned into clusters of fixed length. All the test set's clusters (referred also as test clusters or simply clusters) are then compared against the normal and inverted pseudorandom sequences generated by each cell of a randomly initialized LFSR. The pseudorandom sequences have the same length as the bit stream created by the concatenation of the test cubes, and they are also partitioned into fixed-length (pseudorandom) clusters. By considering multiple LFSR cells and, consequently, multiple pseudorandom sequences, we increase the number of test clusters that are compatible with the corresponding clusters of the LFSR sequences. In other words, if a test cluster cannot be matched by the corresponding pseudorandom cluster of an LFSR cell, it may be compatible with the respective cluster of another cell. Moreover, LFSRs with internal XOR gates are used because, due to the internal XOR gates, the pseudorandom sequences generated by their cells are less correlated than the sequences generated by the cells of external feedback LFSRs.

When a cluster of test data is compatible with the respective cluster of an LFSR cell sequence, a cell hit occurs. A predetermined (defined by the designer) number of LFSR cells with the largest hit ratios is selected to feed the scan chain of the CUT through a multiplexer. Specifically, the multiplexer selection address of each cell is Huffman encoded. We call this type of

encoding “cell encoding.” All the clusters that are compatible with the pseudorandom sequences of the selected cells are generated by the LFSR (and are consequently encoded). The rest of the clusters are labeled as failed and are processed afterward in a different way, as it will be explained later. We should note that a single Huffman codeword is associated with all failed clusters to distinguish them from the rest. This codeword notifies the decoder that the next cluster to be processed is a failed one.

Since many clusters have a large number of x values, they are compatible with the sequences generated by more than one LFSR cell. The proposed method associates each cluster with the LFSR cell, which skews the cell occurrence (matching) frequencies the most. In other words, if for a cluster cl more than one hit from different LFSR cells occurs, then cl is appointed to the most frequently used cell. The construction of the Huffman tree is done later, taking into account the matching frequencies of the selected cells as well as the frequency of occurrence of the failed clusters.

For exploiting the advantages of variable-to-variable coding (Huffman is a fixed-to-variable code), we allow, if possible, consecutive clusters to be generated by the same LFSR cell. The number of consecutive clusters (cluster group length) that will be generated is encoded using only one codeword, which follows the cell encoding codeword in the encoded data stream. Due to the large number of x values in the test sets, cluster grouping, as was verified experimentally, can be performed in many cases. To keep the hardware overhead low, we allow the length of each group of clusters to be among a predetermined list of distinct lengths (group length quantization). These distinct lengths are chosen to be equal to $2^0, 2^1, \dots, 2^{r-1}$, where $2^{r-1} < \text{max_length}$ and max_length is the maximum number of consecutive clusters that are compatible with LFSR-generated pseudorandom clusters. If, for example, $\text{max_length} = 40$, then the list of lengths L will be $L = \{1, 2, 4, 8, 16, 32\}$. In the case that the length of a cluster group is not equal to a length in L , then a number of clusters is removed from the end of the group, so its length becomes equal to the largest possible length in the list. Group lengths are also Huffman encoded. This choice is motivated by the lengths’ occurrence probabilities, which are normally skewed (large lengths are expected to occur less frequently than short lengths). We call this type of Huffman encoding “length encoding.” As it will be explained later, the same Huffman codewords are used for cell encoding and length encoding to keep the decoding cost low. Therefore, the maximum number of potential list lengths is equal to the number of selected cells. In the case that the number of selected cells is greater than r , additional lengths are appended in the list according to the following rule: Each additional length is selected iteratively as the midpoint of the greatest distance between two successive lengths in the list. For example, if an extra list length could be appended to list L , this would be equal to 24 (the midpoint between 16 and 32). A cell encoding codeword is always followed in the code stream by a length encoding codeword when the encoded cluster is not a failed one.

In the case of a failed cluster, a different approach is adopted. The cluster is partitioned into equally sized blocks, and each block is encoded directly using selective Huffman coding. We

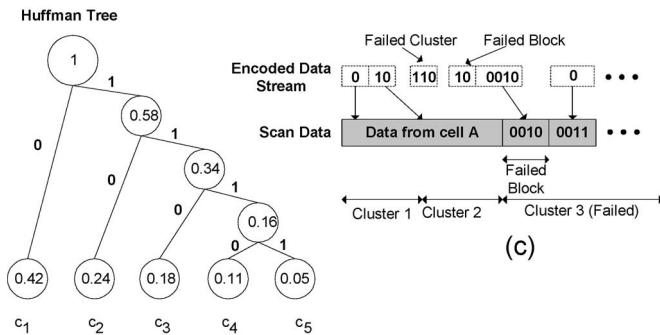
call this encoding “block-data encoding.” According to the selective Huffman approach, only the blocks with the highest probabilities of occurrence are encoded. Thus, some blocks remain unencoded (we call them failed blocks) and are provided directly by the ATE. As in the case of failed clusters, a single Huffman codeword (not necessarily the same as for failed clusters) is associated with all failed blocks. The actual data of an unencoded block follow that codeword. In block-data encoding, the same Huffman codewords as in cell encoding and length encoding are used. Therefore, a number of distinct blocks equal to the number of the selected LFSR cells and to the number of potential list lengths can be encoded.

In terms of hardware overhead, the major advantage of the proposed compression method is that the same Huffman decoder can be used to implement the three different decodings. The size of the Huffman decoder is determined by the number of the selected LFSR cells (which is equal to the number of the list lengths in length encoding and to the number of distinct blocks encoded by block-data encoding). The Huffman tree is constructed by summing the corresponding occurrence probabilities of all three cases so as a single Huffman code, covering all three of them, is generated. Thus, the same codeword, depending on the mode of the decoding process, corresponds to three different kinds of information (and hence is decoded differently): 1) to an LFSR cell output (normal or inverted); 2) to a cluster group length; or 3) to a block of data. The first codeword in the encoded data stream is always considered as a cell codeword. If it does not indicate a failed cluster, then the next codeword corresponds to the length of the cluster group. If, on the other hand, a failed cluster is indicated, then the next “cluster size/block size” codewords are processed as block-data codewords, where cluster size (block size) denotes the number of bits of each cluster (block). Each one of them may indicate a block-data encoded block or a failed block. In the first case, the data block is generated by the decompressor, or else, the actual failed block follows in the encoded stream. This codeword sequence is repeated iteratively starting always from a cell encoding codeword.

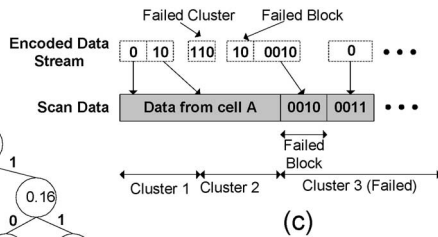
Example 2: Assume a test set of 744 bits. Let each cluster be 24 bits wide and each block 4 bits wide (six blocks per cluster). For the encoding of the test set, we use four LFSR cells and, consequently, four different cluster group list lengths and four distinct encode-able data blocks for each failed cluster. Consider the encoding scenario presented in Fig. 2(a), where the selected cells, the available list lengths, and the most frequently occurring data blocks are reported in descending order according to their frequency of occurrence. Each line of the table (i.e., the respective case for all three encodings) corresponds to a single codeword in the final encoded data stream. Note that there are 12 groups of clusters matched by LFSR cell sequences and three failed clusters that are partitioned into 18 blocks. Overall, there are 45 occurrences of encode-able information and five unique codewords that will be used for encoding them. The occurrence volumes in each line of the table are summed and divided by the total number of occurrences (i.e., 45), generating the probability of occurrence of each distinct codeword. Note that the descending order at each column guarantees that the occurrence frequencies of the

	Cell Encoding		Length Encoding		Block-Data Encoding		P	Code Word
	Cell	Occur.	List Length	Occur.	Data Block	Occur.		
c ₁	A	5	1	6	0011	8	(5+6+8)/45	0
c ₂	B	4	2	3	Fail	4	(4+3+4)/45	10
c ₃	Fail	3	4	2	0000	3	(3+2+3)/45	110
c ₄	C	2	8	1	1111	2	(2+1+2)/45	1110
c ₅	D	1			0001	1	(1+1)/45	1111
SUM		15		12		18		
Total Sum = 45								

(a)



(b)



(c)

Fig. 2. Proposed encoding example.

codewords will be as skewed as possible. The encoded data stream in Fig. 2(c) is a representation of the data stored in the ATE. The first codeword (i.e., 0) corresponds to cell A, and the next codeword (i.e., 10) indicates the group length, which is 2 [see the corresponding line in the table of Fig. 2(a)]. Therefore, the scan chain is fed from cell A for the first two clusters. The next codeword (i.e., 110) indicates that the next cluster is a failed one. According to the proposed compression scheme, each failed cluster is partitioned into six blocks. The next codeword (i.e., 10) indicates that the first block of the third cluster is a failed one as well; therefore, the actual data (i.e., 0010) are not encoded and follow codeword 10. The codeword for the second block is 0, which corresponds to the encoded block 0011 that will be loaded in the scan chain. This is repeated until all six blocks of the failed cluster have been processed. The size of the encoded stream is 109 bits.

B. Encoding Algorithm

The encoding process consists of five main steps.

Step 1—Generation of the Pseudorandom Sequences and Concatenation of the Test Cubes Into a Single Bit Stream: At the first step, the LFSR is initialized randomly and is allowed to evolve for a number of cycles equal to the number of bits of the test set. The pseudorandom sequences generated by each LFSR cell (normal and inverted) are partitioned into clusters of fixed length. Then, the CUT’s test cubes are concatenated to form a single test bit stream, which is also partitioned into clusters of the same length. The concatenation is performed in a number of iterations equal to the number of the test cubes. At each iteration, a new test cube is selected and appended at the end

of the test bit stream. For choosing the appropriate test cube, a cube weight is calculated as follows: Each cube is partitioned into test clusters, which are compared against the corresponding pseudorandom clusters generated by all LFSR cells. For every match of a test cluster with a pseudorandom cluster, the number of the defined bits of the test cluster is added to the weight of the cube. We select the cube with the maximum weight, i.e., the one that contains the maximum number of defined bits that can be generated by most of the LFSR cells. Note that in the subsequent steps (after the generation of the test bit stream), every test cluster can be examined against $2 \cdot \text{len}$ pseudorandom clusters (len is the LFSR length), one for each pseudorandom sequence generated by the $2 \cdot \text{len}$ normal and inverted outputs of the LFSR cells.

Step 2—Selection of the LFSR Cells for the Encoding: At the second step of the encoding process, a predetermined number of LFSR cells is iteratively selected. During each iteration, a weight is calculated for each cell, and the cell with the largest weight is selected. The weight of each cell is calculated as the sum of the defined-bit volumes of all clusters of the test bit stream, which are compatible with the corresponding pseudorandom clusters generated by that cell. The cell with the largest weight is selected, and the clusters of the test bit stream, which are compatible with the corresponding pseudorandom clusters of that cell, are not further considered. The same procedure is applied again to the remaining clusters of the test bit stream, for the rest LFSR cells, until the predetermined number of LFSR cells is selected.

Step 3—Determination of the LFSR Cell That Will Encode Each Test Cluster: At the next step, each test cluster is associated with one LFSR cell in a greedy iterative fashion as follows: At each iteration, the pseudorandom sequence of every selected (from Step 2) LFSR cell is separately examined against the whole test bit stream, and all successive pseudorandom clusters that are compatible with the corresponding test clusters are grouped. The size of the largest group determines the value of max_length (see Section III-A), and thus, the list of group lengths is generated. Then, if the length of any of the formed groups is not equal to one of the distinct lengths in the list, the largest distinct length that does not exceed the length of the group is selected, and the last clusters of the group are removed. In this way, the length of the group becomes equal to the selected distinct length. The largest group is then selected, and if there is more than one maximum sized group generated by different cells, the one produced by the cell that is already associated with the greatest number of groups is selected. The clusters of the test bit stream matched by the pseudorandom clusters of the selected group are not further considered in the remaining iterations.

Step 4—Partitioning of Failed Clusters Into Blocks and Selection of the Encode-Able Blocks: At the next step of the encoding process, the test clusters that cannot be generated by any of the selected LFSR cells are labeled as failed, and they are partitioned into blocks of the same size. Among them, a number of blocks (equal to the number of selected LFSR cells) with the highest probabilities of occurrence are selected.

Step 5—Generation of the Codewords and Encoding of the Test Bit Stream: At the final step of the encoding process, the

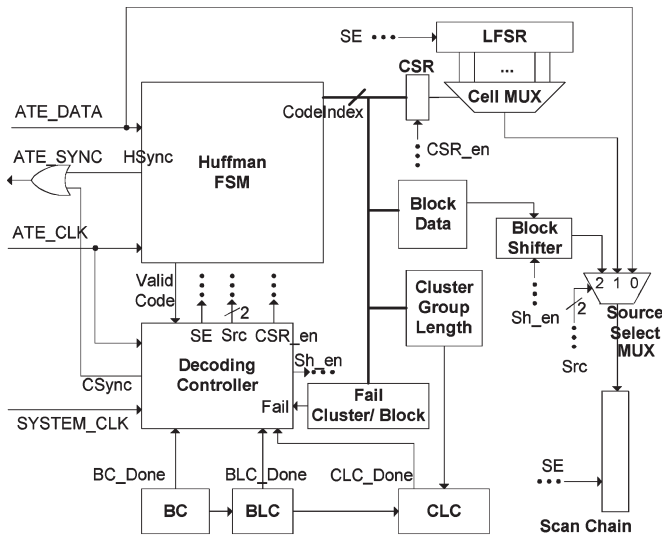


Fig. 3. Decompression architecture.

Huffman codewords are generated, and the test bit stream is encoded. Specifically, three different occurrence lists of encodable information are created.

- 1) Cell occurrence list. It consists of: 1) one entry for every selected LFSR cell. The value of each entry is equal to the number of groups that will be generated by the corresponding cell and 2) one entry for all failed clusters with value equal to the failed-cluster volume.
- 2) Group length occurrence list. It consists of one entry for every distinct length of the list of group lengths. The value of each entry is equal to the number of cluster groups with the corresponding length.
- 3) Block occurrence list. It consists of: 1) one entry for each block to be encoded with value equal to the number of its occurrences and 2) one entry for all failed blocks with value equal to the failed-block volume.

The above lists are sorted in descending order according to the values of their entries, and then, these values are added rowwise to create a single list. Specifically, the values of the first entries of all lists (largest ones) are added to create the first entry of the new list, the values of the second entries (second largest) are added to create the second entry of the new list, etc. The new list is used for the construction of the Huffman tree, and the resulting codewords are used for encoding each piece of information of the initial lists.

IV. DECOMPRESSION ARCHITECTURE

The block diagram of the proposed decompression architecture is presented in Fig. 3. It consists of the following units.¹

- 1) *Huffman FSM*. This unit receives serially the data from the ATE (ATE_DATA) with the frequency of ATE clock (ATE_CLK). Upon reception of a codeword, the signal HSync, which is sent back to the ATE, is set to 0 to stop the transmission until the decompressor is ready to

receive the next codeword. At the same time, the FSM places on the bus CodeIndex a binary index indicating which codeword has been received and notifies the Decoding Controller with the signal Valid Code. The aforementioned index is a binary value between 0 and $N - 1$, assuming that the implemented code consists of N codewords, and it also serves as the selection address of the Cell Mux.

- 2) *Source Select Mux*. Selects the source (an LFSR cell, Block Data, or a failed block from ATE) that will feed the scan chain according to the value of bus Src (01, 10, and 00, respectively). Src is an output of the Decoding Controller.
- 3) *Cell Mux*. Selects the cell that will feed the scan chain.
- 4) *Cell Select Register (CSR)*. Stores the address of the selected cell (CodeIndex value) when CSR_en is set to 1 (by the Decoding Controller) and holds it during scan chain loading.
- 5) *LFSR*. It is enabled by the scan-enable (SE) signal every time the scan chain is loaded with any kind of test data. In this way, the pseudorandom sequence generated by each LFSR cell has size equal to the size of the bit stream loaded in the scan chain. When blocks (encoded or not) are fed in the scan chain, the generated LFSR data are simply ignored.
- 6) *Block Data and Cluster Group Length*. Combinational blocks that receive CodeIndex and return the block data and the group length, respectively. They can be also implemented as lookup tables, with CodeIndex used for their addressing.
- 7) *Block Shifter*. Shifts the data block received by the Block Data unit into the scan chain. Decoding Controller controls the shifting with signal Sh_en.
- 8) *Fail Cluster/Block*. A very small combinational circuit that sets Fail = 1 when a codeword corresponds to a failed cluster or a failed block.
- 9) *Bit Counter (BC), Block Counter (BLC), and Cluster Counter (CLC)*. Count, respectively, the number of bits, blocks, and clusters that enter the scan chain. BC_Done is set to 1 when a whole block has been shifted in the scan chain, BLC_Done = 1 when all the blocks of a cluster have been shifted in the scan chain, and CLC_Done = 1 when all the clusters of a group have been shifted in the scan chain.
- 10) *Decoding Controller*. This is a finite state machine that synchronizes the operation of all units. The state diagram of this state machine is presented in Fig. 4 (only the most important input and output signals are shown). Initially, the controller waits for the first codeword to be received by the Huffman FSM unit. After its reception (Valid Code = 1), the controller checks if it encodes an LFSR cell (Fail = 0) or indicates a failed cluster (Fail = 1). In the former case, the controller sets CSR_en = 1 (the cell selection address, i.e., CodeIndex, is stored in the CSR register) and proceeds to the WAIT_LENGTH state. In the latter case, it proceeds to the WAIT_FAILED_CLUSTER state. At the WAIT_LENGTH state, the controller waits for the next codeword, which is a

¹The decompression architecture has been verified with extensive simulations. For the convenience of the reader, only the most important signals are shown.

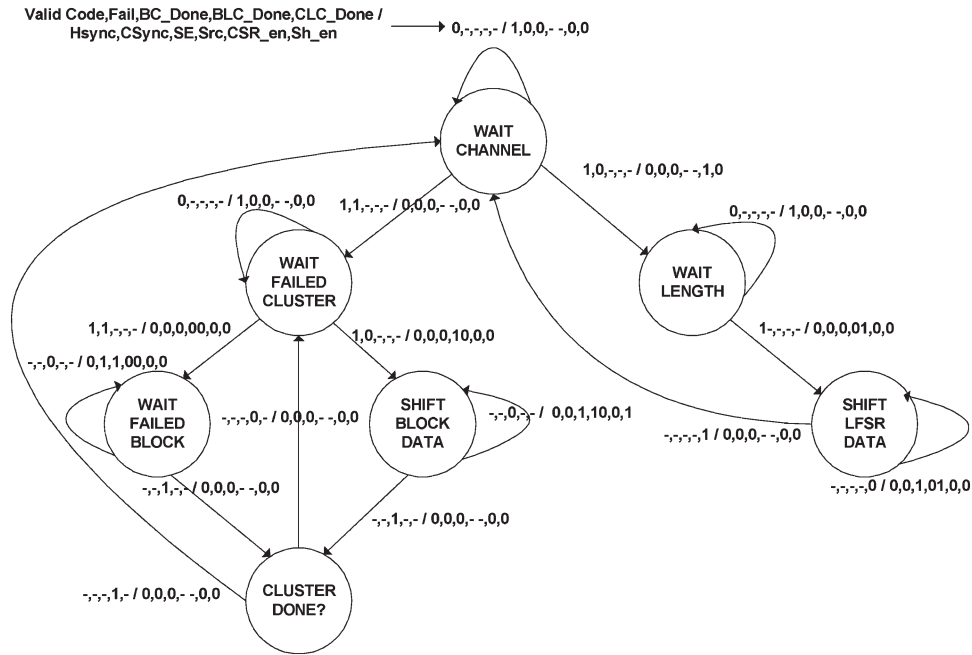


Fig. 4. Decoding Controller state diagram.

group length codeword and, after its reception, enables the loading of CLC with the output of the Cluster Group Length unit (it is the binary representation of the length of the group). Also, the controller initializes the BC and BLC, sets $\text{Src} = 01$ (the selected LFSR cell is driven to the output of the Source Select Mux) and proceeds to the **SHIFT_LFSR_DATA** state. At this state, it activates the $\text{SE} = 1$ signal, and the LFSR begins to load the scan chain with data from the selected cell until Cluster Counter reaches zero ($\text{CLC_Done} = 1$). Then, the state machine returns to the **WAIT_CHANNEL** state. In the case of a failed cluster, the controller enters the **WAIT_FAILED_CLUSTER** state and waits for the next codeword. If it corresponds to an encoded data block ($\text{Fail} = 0$), then the controller stores the output of the Block Data unit (which is the decoded data block) to the Block Shifter, sets $\text{Src} = 10$ (the Block Shifter's output is driven to the output of the Source Select Mux), and proceeds to the **SHIFT_BLOCK_DATA** state, where it remains until it is signaled by Bit Counter ($\text{BC_Done} = 1$). At that state, the data of the Block Shifter are serially loaded in the scan chain. On the other hand, if the codeword received at the **WAIT_FAILED_CLUSTER** state corresponds to a failed block ($\text{Fail} = 1$), the controller sets $\text{Src} = 00$ (the ATE channel is driven to the output of the Source Select Mux) and proceeds to the **WAIT_FAILED_BLOCK** state, where it remains until it is signaled by Bit Counter ($\text{BC_Done} = 1$). During **WAIT_FAILED_BLOCK**, Bit Counter is enabled once at each ATE_CLK cycle. The controller sets signal CSync to enable the transmission of an unencoded block, it samples ATE_CLK and sends each data bit received from the ATE to the scan chain of the CUT (using signal SE). From both **SHIFT_BLOCK_DATA** and **WAIT_FAILED_BLOCK** states, the controller proceeds

to the **CLUSTER_DONE?** state, in which it is checked if all the blocks of a failed cluster have been processed or not. If Block Counter has reached 0 ($\text{BLC_Done} = 1$), then all the blocks have been processed, and thus, the next state is **WAIT_CHANNEL**. Otherwise, the next state is **WAIT_FAILED_CLUSTER**.

The whole test-generation process is controlled by two small counters, which are not shown in Fig. 3. The first one indicates when a test vector has been loaded in the scan chain and is ready for application. When this happens, the function of the decoder is suspended for one clock cycle (the capture cycle). When all the vectors have been applied to the CUT, the second counter indicates the end of the test session.

The proposed decompression architecture, like the decoders of many other compression techniques, suffers from the problem of synchronization with the ATE. One way to solve this problem is to use a first-in first-out buffer between the decoder and the ATE [13]. For simplicity, such a buffer was omitted from our description.

As it will be shown in Section VI, the efficiency of the proposed encoding approach depends mainly on the number of selected cells, which determines the number of codewords of the Huffman code. The same decompressor can be used for two or more cores by implementing for each core only the units Cell Mux, Block Data, Cluster Group Length, and Fail Cluster/Block, which occupy only a small portion of the area of the decompressor. Moreover, if the Block Data and Cluster Group Length units are implemented as lookup tables, they just need to be loaded with the specific data for each core only at the beginning of the test session. Therefore, the decompressor can be easily reused for different cores with almost zero area penalty. Note that the unit Huffman FSM is implemented only once, and thus, the same codewords are used for all cores. As it will be demonstrated in Section VI, in most cases, the reduction in the compression ratio when using the same codewords for

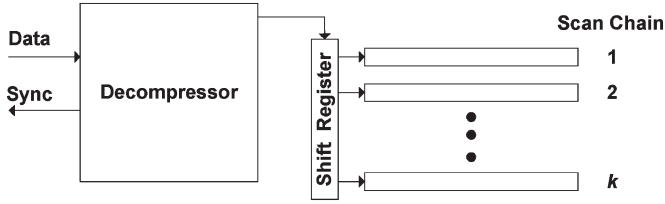


Fig. 5. Application to multiple-scan-chain architectures.

multiple cores is only marginal. This can be easily explained if we take into account that, for the same number of cells (same number of Huffman codewords) and skewed frequencies of occurrence, the Huffman trees are not much different, and thus, the achieved compression will be very close to the one of a dedicated Huffman code. Note that, despite the fact that the same Huffman FSM unit is used, the selected cells, the cluster size, and the block size do not have to be the same for different cores.

The proposed scheme can be applied to multiple-scan-chain architectures by using a shift register with width equal to the number of scan chains (Fig. 5). The shift register is loaded by the decompressor and then feeds the scan chains in parallel, as proposed in [52]. Alternatively, the scan output of chain i can be connected to the scan input of chain $i + 1$ ($i \in [1, k - 1]$), thus forming a single scan chain. Obviously, in both cases, the test-application time advantage that multiple scan chains offer is canceled. However, the tradeoff between test-application time and hardware overhead can be explored if the set of scan chains is partitioned into groups and each group is driven by a single decompressor. In this case, all groups will be loaded in parallel, and the test-application time will be reduced by a factor equal to the number of groups. Of course, the hardware overhead, due to the multiple decompressors, will increase.

V. TEST-APPLICATION TIME CALCULATION

Let us now calculate the test-application time reduction of the proposed encoding scheme. Suppose that $|D|$ and $|E|$ are the sizes in bits of the uncompressed and compressed test sets, respectively. The compression ratio is given by the formula $CR = (|D| - |E|)/|D|$. Let f_{ATE} and f_{SYS} be the ATE and system clock frequencies, respectively, with $f_{SYS} = m \cdot f_{ATE}$ ($m \geq 1$), and CS and BS be the cluster and block sizes, respectively. Also, in the encoded data set, let G_i be the number of occurrences of the cluster group with length L_i ($i = 1, \dots, n$, where n is the number of codewords), and F_c and F_b be the volumes of failed clusters and failed blocks, respectively. The test-application time of the uncompressed test set is $t_D = |D|/f_{ATE}$, and the test-application time reduction is given by the formula $TR = (t_D - t_E)/t_D$, where t_E is the test-application time of the compressed test set. t_E consists of three main parts.

- 1) t_1 : The time for downloading the data stream from the ATE to the core. It is equal to $t_1 = |E|/f_{ATE}$.
- 2) t_2 : The time required for loading the scan chain with pseudorandom LFSR sequences of length equal to the number of bits of the decoded cluster groups. The number of all clusters is $\sum_{i=1}^n G_i L_i$, and therefore, $t_2 = CS/f_{SYS} \sum_{i=1}^n G_i L_i$.

- 3) t_3 : The time for loading the scan chain with encoded data blocks (not failed blocks). The total number of blocks is equal to $F_c \cdot CS/BS$; thus, the number of the successfully encoded blocks is $(F_c \cdot CS/BS) - F_b$, and their total size in bits is $F_c \cdot CS - F_b \cdot BS$. Therefore, $t_3 = (F_c \cdot CS - F_b \cdot BS)/f_{SYS}$.

The total time required for the compressed test set is

$$t_E = t_1 + t_2 + t_3$$

$$= \frac{|E|}{f_{ATE}} + \frac{1}{f_{SYS}} \left(CS \cdot \sum_{i=1}^n G_i \cdot L_i + F_c \cdot CS - F_b \cdot BS \right)$$

and the test-application time reduction is equal to

$$TR = 1 - \frac{|E|}{|D|} - \frac{f_{ATE}}{|D| \cdot f_{SYS}} \times \left(CS \cdot \sum_{i=1}^n G_i L_i + F_c \cdot CS - F_b \cdot BS \right)$$

or equivalently

$$TR = CR - \frac{1}{|D| \cdot m} \left(CS \cdot \sum_{i=1}^n G_i L_i + F_c \cdot CS - F_b \cdot BS \right).$$

VI. EVALUATION AND COMPARISONS

The proposed compression method was implemented in C programming language. We conducted experiments on a Pentium PC for the largest IEEE International Symposium on Circuits and Systems (i.e., ISCAS 1989) benchmark circuits, assuming a single (full) scan chain. We used the dynamically compacted test sets generated by Mintest [15] for stuck-at faults. The same test sets were used in [5], [6], [8], [9], [12], [28], [42], [44], [49], [51], and [52]. The runtime of the compression method is a few seconds for each benchmark circuit. The compression ratio is calculated by the formula

$$CR (\%) = \frac{\text{data bits} - \text{compressed bits}}{\text{data bits}} \cdot 100.$$

Primitive polynomial LFSRs with internal XOR gates were used in the experiments. Note that for each LFSR cell, apart from its normal sequence, the inverted one is also considered (normal and inverted LFSR cell outputs are considered as different cells).

At first, experiments have been performed using the test sets of several ISCAS 1989 benchmark circuits so as to study how the various algorithm parameters influence the compression ratio. In the following, we present results only for s15850. The results for the rest benchmark circuits are similar.

Our first set of experiments studies the effect of the LFSR size, polynomial, and seed on the compression ratio. Various LFSRs were used, and the number of selected cells in all experiments was 8, whereas the cluster size and block size were set to 16 and 4, respectively. Ten random primitive polynomial seed pairs were utilized for each of the ten examined LFSR sizes between 15 and 60 (with step = 5). Among them, the maximum and the minimum compression ratios achieved for

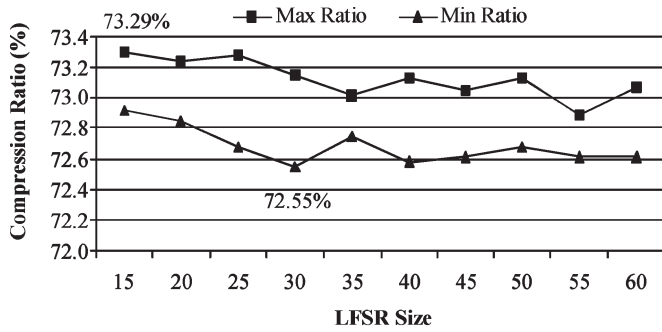


Fig. 6. LFSR size experiments for s15850.

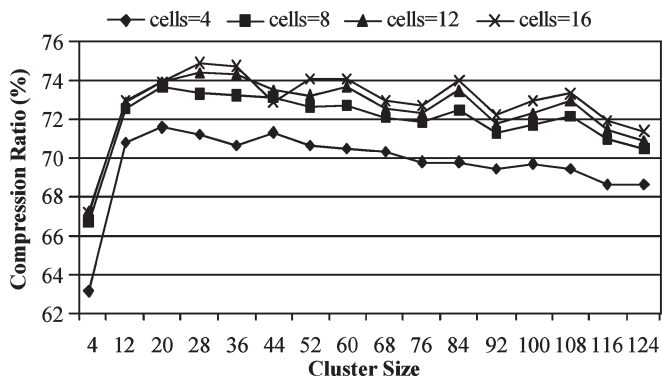


Fig. 7. Varying cluster size for s15850.

each LFSR size are reported in Fig. 6 (Max Ratio curve and Min Ratio curve, respectively). Both curves do not seem to be affected by the LFSR size, whereas the distance between them depends mainly on the use of different polynomials and seeds. We therefore conclude that the size of the LFSR does not affect the compression ratio, whereas the choice of polynomials and seeds slightly affects the compression results. The maximum compression ratio variation, calculated as the difference of the global maximum ratio (73.29%) minus the global minimum ratio (72.55%), is 0.74%, which is very small. Consequently, we deduce that the LFSR size, polynomial, and seed affect the proposed compression method only marginally. Hence, in the following experiments, an LFSR of size 15, a single random primitive polynomial, and a single random seed are used.

The second set of experiments demonstrates the effect of the cluster size on the compression ratio. For the test set of s15850, we applied the proposed method for cluster sizes between 4 and 124 bits (with step = 8) and for 4, 8, 12, and 16 selected cells. As mentioned earlier, the same polynomial and the same seed were used in all experiments. For each case, various block sizes were examined, and the block size value that maximized the compression ratio was selected. It is obvious from Fig. 7 that, initially, as the cluster size increases, the compression ratio improves. For a specific value of the cluster size, which depends on the number of selected cells, the compression ratio reaches a peak value and then drops. Two things, with contradicting effects on the compression ratio, happen as the cluster size increases: The total number of clusters and, as a result, the bits required for their encoding are reduced, while on the other hand, the number of failed clusters increases. The cluster size, which balances this behavior, leads to the maximum compression

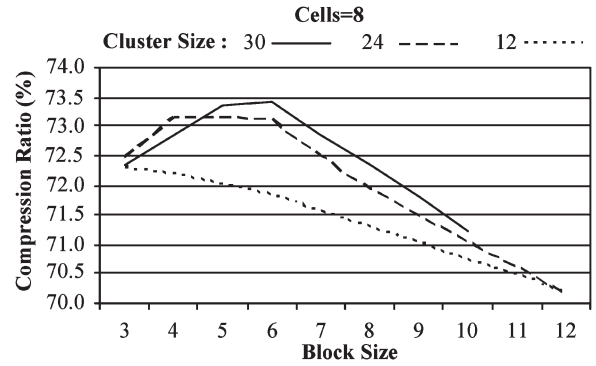


Fig. 8. Varying block size for s15850.

ratio. Note that as the selected-cell volume increases, larger cluster sizes are required for achieving the best compression. This is an expected behavior since as the cell volume increases, the probability of generating a cluster from one of the selected cells is higher, and thus, the number of failed clusters drops. The variation of the compression ratio values in these experiments is 8.4% for 4 cells, 6.9% for 8 cells, 7.1% for 12 cells, and 7.7% for 16 cells.

The third set of experiments shows the effect of different block sizes on the compression ratio. For the test set of s15850, we varied the block size between 3 and 12 bits (with step = 1) using a single polynomial, a single seed, eight selected cells, and cluster sizes equal to 12, 24, and 30. Note that the block size affects only the encoding of failed clusters (Block Data encoding). The results are shown in Fig. 8. For small block sizes, many codewords are required for encoding each failed cluster since each cluster contains many blocks. On the other hand, large block sizes affect the compression achieved by Block Data encoding in a negative manner. This is a consequence of the selective Huffman nature of Block Data encoding, according to which only a portion of all distinct blocks is encoded. Specifically, as the block size increases, the number of distinct blocks grows exponentially, whereas the number of encoded blocks remains constant; thus, the percentage of the failed blocks increases. The compression ratio variation in this set of experiments reached 2.1% for cluster size 12, 3% for cluster size 24, and 2.2% for cluster size 30.

We next study the effect of the selected-cell volume on the compression ratio. The following parameter values were used (for s15850): cluster size equal to 16 and 32 and block size equal to 4 and 8. For each experiment, we varied the number of cells between 4 and 24 with step = 4 (for one polynomial and one seed). The results are shown in Fig. 9(a). We can see that an increase in the number of selected cells leads to compression ratio improvements, independently of the other parameters. For small selected-cell volumes, the compression ratio is mainly affected by the block size. This is due to the large number of failed clusters, which increases the percentage of the data encoded by Block Data encoding. When the number of cells increases, the compression ratio is mainly affected by the cluster size. In this case, the number of failed clusters decreases, and thus, more test data are encoded by pseudorandom LFSR sequences (cell encoding and length encoding). This is shown in Fig. 9(b), where the percentages of clusters encoded by the

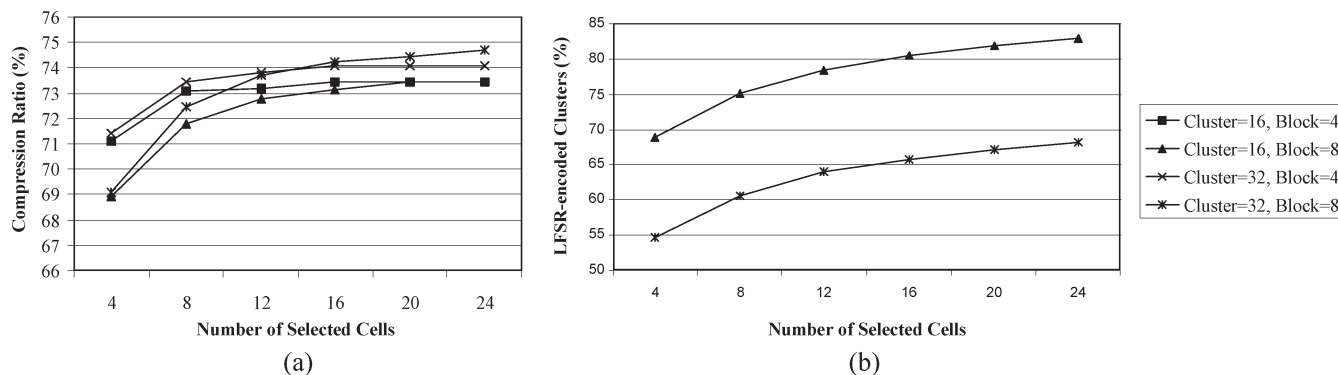


Fig. 9. Varying cell volume for s15850.

TABLE II
COMPRESSION RESULTS

Circuit	Cells = 4		Cells = 8		Cells = 12		Cells = 16		Cells = 20		Cells = 24	
	ENC	C, B	ENC	C, B	ENC	C, B	ENC	C, B	ENC	C, B	ENC	C, B
s5378	10416	18, 6	9795	16, 8	9686	16, 8	9431	20, 10	9373	16, 8	9358	20, 10
s9234	17449	16, 4	16441	16, 4	16268	20, 4	15993	20, 10	15697	20, 10	15511	20, 10
s13207	23021	40, 10	19943	50, 10	19335	40, 10	18799	40, 10	18559	30, 10	18384	30, 10
s15850	21738	20, 4	20174	20, 4	19901	32, 8	19356	32, 8	19269	20, 10	18926	32, 8
s38417	65545	20, 5	62765	36, 6	61602	40, 8	60180	40, 8	59341	40, 8	58785	48, 8
s38584	62955	20, 4	58657	24, 6	58437	32, 8	56630	32, 8	55586	32, 8	55200	20, 10

TABLE III
COMPARISONS AGAINST [28] AND [52]

Circuit	Size			Red. (%) of prop. over			
	Mintest	[28]	[52]	Prop.	Mintest	[28]	[52]
s5378	23754	10666	10511	9358	60.6	12.3	11.0
s9234	39273	17987	17763	15511	60.5	13.8	12.7
s13207	165200	37996	24450	18384	88.9	51.6	24.8
s15850	76986	26175	22126	18926	75.4	27.7	14.5
s38417	164736	67542	61134	58785	64.3	13.0	3.8
s38584	199104	71478	62897	55200	72.3	22.8	12.2

LFSR are presented for the experiments of Fig. 9(a) (note that block size was set to eight since it does not affect the number of LFSR-encoded clusters). It is obvious that as the number of selected cells increases, the percentage of LFSR-encoded clusters increases too (or, equivalently, the failed-cluster percentage decreases), and thus, the compression ratio improves. After a specific number of cells, the number of clusters encoded by the LFSR, and consequently the compression ratio reach a saturation point. The saturation value of each curve in Fig. 9(a) depends on the cluster and block sizes (it generally gets higher as cluster and block sizes increase). The maximum variation of the compression ratio in these experiments is about 5.7% for the case of cluster size 32 and block size 8.

So far, we have seen that the LFSR size does not affect the compression ratio, whereas the influence of the utilized polynomial and initial seed on it is very limited. The most important parameter is the number of selected cells, and for each cell volume, there is one cluster and one block size that maximize the compression ratio. This behavior is depicted in Table II, where the compression results of the proposed method for 4–24 cells (with step = 4) are presented. For every cell-volume case, various cluster and block sizes were examined. Among them, the best results are shown in Table II. The same LFSR (of size 15) and ten random initial seeds were used in

each experiment. Columns labeled “C, B” report the utilized cluster and the block sizes, whereas columns labeled “ENC” present the encoded test sets’ sizes (the sizes of the original test sets are shown in Table III). It is obvious that in all cases, the compression improves as the number of cells increases.

In Table III, we compare the proposed method against the approach of Jas *et al.* [28], which is based on selective Huffman coding, and that of Tehranipour *et al.* [52], which is the most effective compression method proposed so far in the literature. In columns 2–5, the sizes of the original Mintest test sets, as well as the encoded data volumes of [28] and [52] and the proposed approach, are reported. The reduction percentages of the proposed method over Mintest, [28] and [52] are presented in columns 6–8. To calculate them, we utilized the formula

$$\text{Red}(\%) = \frac{\text{Size}(\text{Method}) - \text{Size}(\text{Proposed})}{\text{Size}(\text{Method})} \cdot 100 \quad (1)$$

where Size(Method) denotes the data volume of Mintest, [28], or [52]. It is obvious that the proposed scheme offers better compression results than both [28] and [52]. We note that no comparisons are provided against approaches that need structural information of the CUT or require ATPG synergy. Also, we do not compare against the techniques of [37] and [38], since they do not provide results for the Mintest test sets, and that of [10], since several conditions have to be satisfied by a core nearby the CUT, so as the former is used as decompressor.

In Table IV, we present the comparisons of the proposed method against other compression techniques in the literature, which impose similar hardware overhead to the CUT and report results for the Mintest test sets. Specifically, we have used (1) for calculating the compressed-data reduction percentages achieved by the proposed method against each one of the other

TABLE IV
COMPRESSED-DATA REDUCTION PERCENTAGES (IN PERCENT) OF THE
PROPOSED METHOD AGAINST OTHER TECHNIQUES

Circuit	[5]	[6]	[8]	[9]	[12]	[42]	[44]	[49]	[51]
s5378	-	-	20.0	24.2	18.3	34.2	18.0	36.0	14.8
s9234	30.3	31.0	28.2	30.0	25.1	48.5	27.0	35.2	24.6
s13207	55.9	47.7	43.7	40.5	32.5	12.4	38.7	51.6	36.4
s15850	53.5	38.1	28.1	27.2	23.3	24.7	23.2	39.6	24.7
s38417	36.1	35.5	9.5	37.1	23.5	31.0	9.5	20.0	0.4
s38584	47.0	38.6	28.7	29.1	26.5	3.4	25.3	36.1	26.3

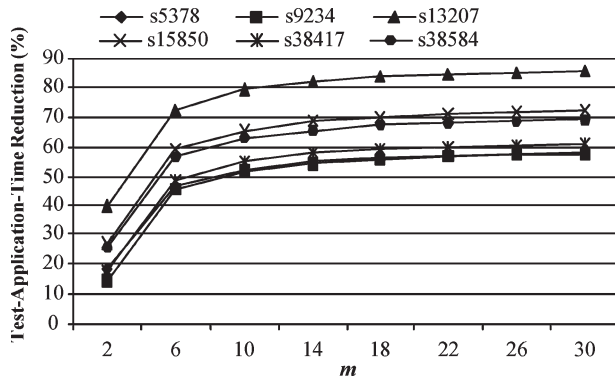


Fig. 10. Test-application time reductions for 24 cells.

TABLE V
DECOMPRESSOR HARDWARE OVERHEAD FOR s15850

Number of cells	Area Overhead (Gate Equivalents)						Total
	Huffman FSM	Group Length	Block Data	Fail Cl./Bl.	Cell Mux	Rest Units	
4	34	3	3	1	5	157	203
8	63	9	10	2	10	169	263
12	88	9	26	2	15	174	314
16	110	11	35	2	20	177	355
20	128	11	53	2	24	179	397
24	142	11	58	2	30	189	432

methods. It can be seen that the proposed approach performs better than all the rest.

As far as the test-application time is concerned, it is obvious that as $m = f_{\text{SYS}}/f_{\text{ATE}}$ increases, greater test-application time gain is achieved. The test-application time reductions for the experiments presented in Table II for 24 cells are shown in Fig. 10 and range from 14.7% for $m = 2$ to 85.6% for $m = 30$.

For calculating the hardware overhead of the proposed technique, we synthesized six different decompressors for 4–24 cells (step = 4), cluster size = 16 bits, and block size = 8 bits, using Leonardo Spectrum (Mentor tools). The Block Data and Cluster Group Length units were implemented as combinational circuits. The area overhead depends strongly on the number of selected cells, which affects the area of the Huffman FSM unit. The block size affects the hardware overhead in a limited way since only the area of the Block Data unit depends on it. Cluster size does not affect the hardware overhead at all. Table V shows the area overhead of the decompressor in gate equivalents (a gate equivalent corresponds to a two-input NAND gate). The decompressors were synthesized for the test set of s15850 (note that the decompressor area does not depend on the test set but on the architectural parameters mentioned above). Columns labeled Huffman FSM,

Group Length, Block Data, Fail Cluster/Block, and Cell Mux present the area of the corresponding units. Column “Rest Units” reports the area overhead of the rest units of the proposed architecture. It is obvious that the area overhead depends mainly on the area of Huffman FSM, which becomes larger as the number of cells (and thus the number of Huffman codewords) increases. The overhead imposed by the Group Length, Block Data, Fail Cluster/Block, and Cell Mux units is only a small portion of the total area overhead. The area occupied by the remaining units is almost constant. In the column “Total,” the total area overhead of the decompressor is reported.

The hardware overhead, in gate equivalents, of the most efficient methods in the literature is 125–307 for [5] (as reported in [12]), 320 for [8], 136–296 for [12], and 416 for [52]. In [28], the hardware overhead is provided as a percentage of the benchmark circuit area and cannot be directly compared to the above methods. However, it is larger than that of [12]. As can be seen, the hardware overhead imposed by the proposed decompressors is comparable to that of the other techniques.

The hardware overhead can be reduced if the same decompressor is used for testing one after the other several cores of a chip. Units Huffman FSM, Decoding Controller, BC, BLC, CLC, CSR, LFSR, and Source Select Mux of the decompressor can be implemented only once on the chip. On the other hand, units Block Data, Group Length, Fail Cluster/Block, and Cell Mux should be implemented for every core that will be tested. The area overhead occupied by the latter units is equal to 5.9%, 11.8%, 16.6%, 19.2%, 22.6%, and 23.4% of the total area of the decompressors for 4, 8, 12, 16, 20, and 24 selected cells, respectively. Therefore, only a small amount of hardware should be implemented for every additional core. The main part of the decompressor is implemented only once on the chip and is used for testing several cores. Note that the area overhead of the Group Length and Block Data units can be completely avoided if lookup tables are used. These lookup tables will be loaded only at the beginning of the test session of each core.

The use of the same Huffman FSM unit for several cores implies that the codewords, which correspond to LFSR cells, list lengths, and data blocks, are the same for each core, whereas the actual cells, list lengths, and data blocks do not have to be the same. The question is how much this common-FSM choice affects the compression efficiency. For investigating this, we conducted a series of experiments, in which the test set of s15850 was reencoded using the codewords generated for the rest benchmarks for various selected-cell volumes. For each cell-volume case, six different experiments were conducted. During the first one, the best code for s15850, assuming a dedicated decompressor, was generated. The parameters of this experiment are identical to those of Table II and correspond to the best compression achieved for s15850 (for the respective cell volume). The compression ratio of this experiment is presented in Fig. 11 by the columns labeled “Ded.” Each of the other five experiments was a reencoding of the test set of s15850 with different codewords. The utilized codewords in these experiments are those generated for each of the five remaining benchmarks in the best compression cases reported in Table II (for the same cell volume as in the first experiment). The compression ratios are reported in Fig. 11 in the form of

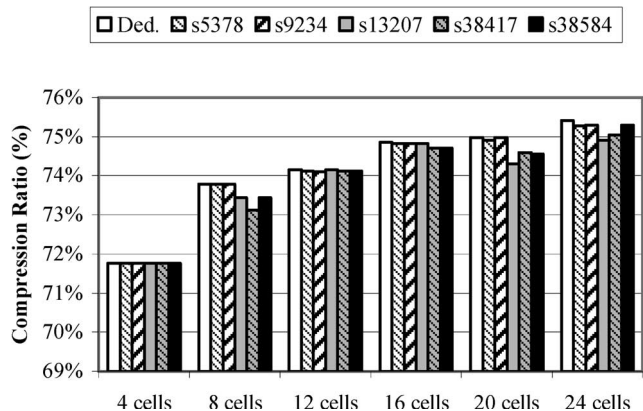


Fig. 11. Compression ratios for s15850 using Huffman codewords generated for other circuits.

columns labeled with the name of the corresponding benchmark circuit. It is obvious that independently of the selected-cell volume, the drop of the compression ratio is negligible (0%–0.7%).

As we have already shown in Fig. 9(a) and Table II, the compression ratio improves (up to a saturation value) as the volume of selected cells increases. The cell volume also affects the implementation cost as shown in Table V. Therefore, in an area-constrained multicore design, the cell volume for each core is adjusted according to the available area. To save silicon, the designer may implement one decompressor for several cores, taking in mind that the same codewords will be used for encoding all test sets. In this case, it is preferable to implement a dedicated decompressor for the core that requires the largest number of cells and then reuse it for the rest cores. The advantages of this architectural decision are obvious (Fig. 11); in almost all cases, an increase in the number of cells leads to compression ratio improvements, independently of the codewords used. Thus, comparing the two scenarios, in a multicore environment where either: 1) a dedicated small decompressor (low cell volume) can be used for each core or 2) a common large decompressor (high cell volume) can be used for all cores, scenario 2 is preferable. Even if the encoding is not the best possible for some of the cores, the gain in compression ratio from the increased cell volume is higher than the loss due to the use of a less effective code for them. Moreover, since only one decompressor is implemented, significant area savings are achieved. To justify this, we conducted the following experiment: Assuming that all benchmarks are embedded in a system, we generated the decompressor for s38417 for the 24-selected-cell case, and for each of the rest benchmarks, we generated only the Block Data, Group Length, Fail Cluster/Block, and Cell Mux units. We compare this common-decompressor approach with that of implementing the dedicated decompressors that correspond to the results of Table II for 4 and 24 selected cells, respectively. The comparisons are shown in Table VI. Columns 2 and 3 present the area overhead and the compression ratio, assuming that a dedicated decompressor with four selected cells is used for each core. Columns 4 and 5 present the area overhead and the compression ratio when a dedicated decompressor with 24 selected cells is implemented for each core (these decompressors correspond to the best results of Table II). Columns 6 and 7 present the area overhead and the

TABLE VI
HARDWARE OVERHEAD IN A SYSTEM WITH MULTIPLE CORES

Circuit	Dedicated Decompressor				Common Decompressor	
	4 cells		24 cells		Area	Compr. Ratio
	Area	Compr. Ratio	Area	Compr. Ratio		
s5378	204	56.1%	444	60.6%	102	60.3%
s9234	190	55.6%	428	60.5%	101	59.9%
s13207	224	86.0%	462	88.9%	121	88.5%
s15850	195	71.8%	440	75.4%	114	75.1%
s38417	207	60.2%	467	64.3%	467	64.3%
s38584	206	68.4%	477	72.3%	129	72.0%
Total/Avg.	1226	66.4%	2718	70.3%	1034	70.0%

compression ratio when a common decompressor is used for all cores. In column 6, the area of the dedicated decompressor of s38417, as well as the area of only the aforementioned required units for the rest benchmarks, is reported. Column 7 shows the compression ratio achieved for each core. In the last row of Table VI, the total area overhead of the necessary decompressors and the average compression ratios are presented. As it was expected, when dedicated decompressors with four selected LFSR cells are used, the total hardware overhead is small, but the average compression ratio is not very high. On the contrary, when dedicated decompressors with 24 selected cells are utilized, the total hardware overhead and the average compression ratio are high. When a common decompressor with 24 selected cells for all cores is employed, less hardware overhead is required than both previous cases. Moreover, the average compression ratio is almost the same with that achieved when a dedicated decompressor with 24 selected LFSR cells is implemented for each core. Note that the area gain in the case of using a common decompressor, compared to the 24-selected-cell dedicated-decompressor case, is 62%. The total area of the common decompressor is equal to 6.5% of the total area of the six benchmarks. In the case of using lookup tables, the total area of the decompressor is equal to 3.9% of the whole system. Note that the decompressor area does not depend on the size of the cores, and therefore, for larger cores, which are common in contemporary systems, this percentage is much lower.

Instead of generating the codewords considering the occurrence lists of a single core and then reusing them for the rest cores, an even better approach is to generate the codewords, taking into account the occurrence lists of all cores. In this case, the average compression ratio is expected to be even higher. However, as shown in Table VI, the expected improvement from this scenario is rather limited since the (average) maximum compression when using a dedicated decompressor with 24 selected cells for each core is only 0.3% higher than that achieved by the utilization of the decompressor of s38417 for all cores. On the other hand, minimum hardware overhead will be imposed if except for the codewords, the encoded data are common for all cores as well (i.e., the same codeword encodes the same data for all test sets). Then, all units can be shared among the different cores, and the area overhead will be minimized. However, in this case, the encoded data (selected LFSR cells, cluster group lengths, encoded blocks, cluster size, and block size) will not be optimized for each core, and consequently, the average compression ratio is expected to drop.

VII. CONCLUSION

In this paper, a new test-data compression method, which is based on multilevel Huffman coding, was presented. The proposed method is suitable for IP cores of unknown structure. Three different kinds of information are encoded using the same Huffman codewords, and thus, improved compression results can be achieved. The area overhead of the required decompressor is very low and comparable to that of the most efficient methods in the literature. Furthermore, in a system with multiple cores, the major part of the decompressor can be implemented only once and shared among different cores. In this way, significant area savings can be achieved. In addition, most of the test sets' x values are filled with pseudorandom data generated by an LFSR, leading to increased probability of detection of unmodeled faults.

ACKNOWLEDGMENT

The authors would like to thank Prof. K. Chakrabarty for providing the Mintest test sets that were used in the experiments.

REFERENCES

- [1] K. J. Balakrishnan and N. A. Touba, "Matrix-based test vector decompression using an embedded processor," in *Proc. 17th IEEE Int. Symp. DFT VLSI Syst.*, 2002, pp. 159–165.
- [2] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann, "OPMISR: The foundation for compressed ATPG vectors," in *Proc. ITC*, 2001, pp. 748–757.
- [3] I. Bayraktaroglu and A. Orailoglu, "Concurrent application of compaction and compression for test time and data volume reduction in scan designs," *IEEE Trans. Comput.*, vol. 52, no. 11, pp. 1480–1489, Nov. 2003.
- [4] K. Chakrabarty, B. Murray, and V. Iyengar, "Deterministic built-in test pattern generation for high-performance circuits using twisted-ring counters," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 5, pp. 633–636, Oct. 2000.
- [5] A. Chandra and K. Chakrabarty, "System-on-a-chip test-data compression and decompression architectures based on Golomb codes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 3, pp. 355–368, Mar. 2001.
- [6] A. Chandra and K. Chakrabarty, "Test data compression and decompression based on internal scan chains and Golomb coding," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 6, pp. 715–772, Jun. 2002.
- [7] A. Chandra, K. Chakrabarty, and R. A. Medina, "How effective are compression codes for reducing test data volume?" in *Proc. 20th IEEE VTS*, 2002, pp. 91–96.
- [8] A. Chandra and K. Chakrabarty, "A unified approach to reduce SOC test data volume, scan power and testing time," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 3, pp. 352–363, Mar. 2003.
- [9] A. Chandra and K. Chakrabarty, "Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes," *IEEE Trans. Comput.*, vol. 52, no. 8, pp. 1076–1088, Aug. 2003.
- [10] R. Dorsch and H.-J. Wunderlich, "Tailoring ATPG for embedded testing," in *Proc. ITC*, 2001, pp. 530–537.
- [11] A. Dutta, T. Rodrigues, and N. A. Touba, "Low cost test vector compression/decompression scheme for circuits with a reconfigurable serial multiplier," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2005, pp. 200–205.
- [12] P. T. Gonciari, B. Al-Hashimi, and N. Nicolici, "Variable-length input Huffman coding for system-on-a-chip test," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 6, pp. 783–796, Jun. 2003.
- [13] P. T. Gonciari, B. Al-Hashimi, and N. Nicolici, "Synchronization overhead in SOC compressed test," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 1, pp. 140–152, Jan. 2005.
- [14] I. Hamzaoglu and J. H. Patel, "Reducing test application time for full scan embedded cores," in *Proc. FTCS*, Jun. 1999, pp. 260–267.
- [15] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 8, pp. 957–963, Aug. 2000.
- [16] H. Hashempour and F. Lombardi, "Compression of VLSI test data by arithmetic coding," in *Proc. 19th IEEE Int. Symp. DFT VLSI Syst.*, 2004, pp. 150–157.
- [17] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 223–233, Feb. 1995.
- [18] S. Hellebrand, H.-G. Liang, and H.-J. Wunderlich, "A mixed mode BIST scheme based on reseeding of folding counters," *J. Electron. Test.: Theory Appl.*, vol. 17, no. 3/4, pp. 341–349, Jun. 2001.
- [19] G. Hetherington *et al.*, "Logic BIST for large industrial designs: Real issues and case studies," in *Proc. ITC*, 1999, pp. 358–367.
- [20] F. Hsu, K. Butler, and J. Patel, "A case study on the implementation of the Illinois scan architecture," in *Proc. ITC*, 2001, pp. 538–547.
- [21] S. Hwang and J. A. Abraham, "Test data compression and test time reduction using an embedded microprocessor," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 5, pp. 853–862, May 2003.
- [22] H. Ichihara, K. Kinoshita, I. Pomeranz, and S. M. Reddy, "Test transformation to improve compaction by statistical encoding," in *Proc. 13th Int. Conf. VLSI Des.*, 2000, pp. 294–299.
- [23] H. Ichihara, A. Ogawa, T. Inoue, and A. Tamura, "Dynamic test compression using statistical coding," in *Proc. 10th ATS*, 2001, pp. 143–148.
- [24] V. Iyengar, K. Chakrabarty, and B. T. Murray, "Deterministic built-in pattern generation for sequential circuits," *J. Electron. Test.: Theory Appl.*, vol. 15, no. 1/2, pp. 97–114, Aug.–Oct. 1999.
- [25] A. Jas and N. A. Touba, "Test vector decompression via cyclical scan chains and its application to testing core-based designs," in *Proc. ITC*, 1998, pp. 458–464.
- [26] A. Jas, K. Mohanram, and N. A. Touba, "An embedded core DFT scheme to obtain highly compressed test sets," in *Proc. ATS*, 1999, pp. 275–280.
- [27] A. Jas and N. A. Touba, "Deterministic test vector compression/decompression for systems-on-a-chip using an embedded processor," *J. Electron. Test.: Theory Appl.*, vol. 18, no. 4/5, pp. 503–514, Aug. 2002.
- [28] A. Jas, J. Ghosh-Dastidar, M.-E. Ng, and N. A. Touba, "An efficient test vector compression scheme using selective Huffman coding," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 6, pp. 797–806, Jun. 2003.
- [29] A. Jas, C. V. Krishna, and N. A. Touba, "Weighted pseudorandom hybrid BIST," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 12, pp. 1277–1283, Dec. 2004.
- [30] A. Jas, B. Pouya, and N. A. Touba, "Test data compression technique for embedded cores using virtual scan chains," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 7, pp. 775–781, Jul. 2004.
- [31] S. Kajihara, K. Taniguchi, K. Miyase, I. Pomeranz, and S. M. Reddy, "Test data compression using don't-care identification and statistical encoding," in *Proc. ATS*, 2002, pp. 67–72.
- [32] E. Kalligeros, X. Kavousianos, and D. Nikolos, "Multiphase BIST: A new reseeding technique for high test data compression," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 10, pp. 1429–1446, Oct. 2004.
- [33] D. Kaseridis, E. Kalligeros, X. Kavousianos, and D. Nikolos, "An efficient test set embedding scheme with reduced test data storage and test sequence length requirements for scan-based testing," in *Proc. IEEE ETS Inf. Papers Dig.*, 2005, pp. 147–150.
- [34] A. Khoche, E. Volkerink, J. Rivoir, and S. Mitra, "Test vector compression using EDA-ATE synergies," in *Proc. 20th IEEE VTS*, 2002, pp. 97–102.
- [35] M. J. Knieser, F. G. Wolff, C. A. Papachristou, D. J. Weyer, and D. R. McIntyre, "A technique for high ratio LZW compression," in *Proc. DATE Conf. and Exhib.*, 2003, pp. 116–121.
- [36] B. Koenemann *et al.*, "A SmartBIST variant with guaranteed encoding," in *Proc. IEEE ATS*, 2001, pp. 325–330.
- [37] C. V. Krishna and N. A. Touba, "Reducing test data volume using LFSR reseeding with seed compression," in *Proc. ITC*, 2002, pp. 321–330.
- [38] C. V. Krishna and N. A. Touba, "Adjustable width linear combinational scan vector decompression," in *Proc. IEEE/ACM ICCAD*, 2003, pp. 863–866.
- [39] C. V. Krishna and N. A. Touba, "3-Stage variable length continuous-flow scan vector decompression scheme," in *Proc. 22nd IEEE VTS*, 2004, pp. 79–86.
- [40] L. Li, K. Chakrabarty, and N. A. Touba, "Test data compression using dictionaries with selective entries and fixed-length indices," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 8, no. 4, pp. 470–490, Oct. 2003.
- [41] L. Li and K. Chakrabarty, "Test set embedding for deterministic BIST using a reconfigurable interconnection network," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 9, pp. 1289–1305, Sep. 2004.

- [42] L. Li, K. Chakrabarty, S. Kajihara, and S. Swaminathan, "Efficient space/time compression to reduce test data volume and testing time for IP cores," in *Proc. 18th Int. Conf. VLSI Des.*, 2005, pp. 53–58.
- [43] A. El-Maleh, S. al Zahir, and E. Khan, "A geometric-primitives-based compression scheme for testing systems-on-a-chip," in *Proc. 19th IEEE VTS*, Apr./May 2001, pp. 54–59.
- [44] A. El-Maleh and R. Al-Abaji, "Extended frequency-directed run-length code with improved application to system-on-a-chip test data compression," in *Proc. 9th ICECS*, 2002, vol. 2, pp. 449–452.
- [45] M. Nourani and M. H. Tehranipour, "RL-Huffman encoding for test compression and power reduction in scan applications," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 10, no. 1, pp. 91–115, Jan. 2005.
- [46] J. Rajski, "DFT for high-quality low cost manufacturing test," in *Proc. ATS*, 2001, pp. 3–8.
- [47] J. Rajski *et al.*, "Embedded deterministic test for low-cost manufacturing," *IEEE Des. Test Comput.*, vol. 20, no. 5, pp. 58–66, Sep./Oct. 2003.
- [48] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded deterministic test," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 776–792, May 2004.
- [49] P. Rosinger, P. T. Gonciari, B. Al-Hashimi, and N. Nicolici, "Simultaneous reduction in volume of test data and power dissipation for systems-on-a-chip," *Electron. Lett.*, vol. 37, no. 24, pp. 1434–1436, Nov. 2001.
- [50] S. Swaminathan and K. Chakrabarty, "On using twisted-ring counters for test set embedding in BIST," *J. Electron. Test.: Theory Appl.*, vol. 17, no. 6, pp. 529–542, Dec. 2001.
- [51] M. Tehranipour, M. Nourani, K. Arabi, and A. Afzali-Kusha, "Mixed RL-Huffman encoding for power reduction and data compression in scan test," in *Proc. ISCAS*, 2004, vol. 2, pp. II-681–II-684.
- [52] M. Tehranipour, M. Nourani, and K. Chakrabarty, "Nine-coded compression technique for testing embedded cores in SoCs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 6, pp. 719–731, Jun. 2005.
- [53] E. H. Volkerink, A. Khoche, and S. Mitra, "Packet-based input test data compression techniques," in *Proc. ITC*, 2002, pp. 154–163.
- [54] F. G. Wolff and C. Papachristou, "Multiscan-based test compression and hardware decompression using LZ77," in *Proc. ITC*, 2002, pp. 331–339.
- [55] A. Wurtenberger, C. S. Tautermann, and S. Hellebrand, "Data compression for multiple scan chains using dictionaries with corrections," in *Proc. ITC*, 2004, pp. 926–935.



Emmanouil Kalligeros (M'06) received the Diploma degree, the M.Sc. degree, and the Ph.D. degree in embedded testing from the Computer Engineering and Informatics Department, University of Patras, Patras, Greece, in 1999, 2001, and 2005, respectively.

He is currently teaching hardware description languages with the Computer Science and Technology Department, University of Peloponnese, Tripoli, Greece, and computer architecture with the Computer Engineering and Informatics Department, University of Patras. His research interests include test data compression, built-in self-test, low-power testing, and delay testing.

Dr. Kalligeros is a member of the Technical Chamber of Greece.



Dimitris Nikolos (M'95) received the B.Sc. degree in physics, the M.Sc. degree in electronics, and the Ph.D. degree in computer science from the University of Athens, Athens, Greece.

Since 1999, he has been a Full Professor with the Computer Engineering and Informatics Department, University of Patras, Patras, Greece, and the Head of the Technology and Computer Architecture Laboratory. He has authored or coauthored more than 150 scientific papers. He is the holder of one U.S. patent. His main research interests include fault-

tolerant computing, computer architecture, VLSI design, and test and design for testability.

Dr. Nikolos has served as Program Co-Chairman of the IEEE International On-Line Testing Workshops (1997–2001). He also served on the Program Committees for the IEEE International On-Line Testing Symposiums (2002–2006), the IEEE International Symposiums on Defect and Fault Tolerance in VLSI Systems (1997–1999), the Third and Fourth European Dependable Computing Conferences, the International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS; 2005–2006), and the DATE (2000–2006) Conferences. He was a Guest Co-Editor of the June 2002 special issue of *Journal of Electronic Testing, Theory and Applications* devoted to the 2001 IEEE International On-Line Testing Workshop. He was a co-recipient of the Best Paper Award for his work "Extending the viability of I_{DDQ} testing in the deep submicron era," which was presented at the Third IEEE International Symposium on Quality Electronic Design (ISQED 2002).



Krysovalantis Kavousianos (S'97–M'02) received the Diploma degree and the Ph.D. degree in on-line testing from the Computer Engineering and Informatics Department, University of Patras, Patras, Greece, in 1996 and 2000, respectively.

He is currently a Lecturer with the Computer Science Department, University of Ioannina, Ioannina, Greece. His main interests are in the fields of very large scale integration design and digital testing. He is currently working in the areas of test data compression, built-in self-test, low-power testing, and online testing.

Dr. Kavousianos is a member of the Technical Chamber of Greece.