Efficient Test-Data Compression for IP Cores Using Multilevel Huffman Coding*

X. Kavousianos[1], E. Kalligeros[1,2] and D. Nikolos[2,3]
[1]Computer Science Dept., University of Ioannina, 45110 Ioannina, Greece
[2]Computer Engineering & Informatics Dept., University of Patras, 26500 Patras, Greece
[3]Research Academic Computer Tech. Institute, N. Kazantzaki, Univ. Campus, 26500 Patras, Greece
kabousia@cs.uoi.gr, kalliger@ceid.upatras.gr, nikolsd@cti.gr

Abstract

In this paper we introduce a new test-data compression method for IP cores with unknown structure. The proposed method encodes the test data provided by the core vendor using a new, very effective compression scheme based on multilevel Huffman coding. Specifically, three different kinds of information are compressed using the same Huffman code, and thus significant test data reductions are achieved. A simple architecture is proposed for decoding on-chip the compressed data. Its hardware overhead is very low and comparable to that of the most efficient methods in the literature. Additionally, the proposed technique offers increased probability of detection of unmodeled faults since the majority of the unknown values of the test set are replaced by pseudorandom data generated by an LFSR.

1. Introduction

Due to the very tight time-to-market constraints, contemporary digital systems embed pre-designed and pre-verified modules, which are called IP (Intellectual Property) cores. The structure of IP cores is often hidden from the system integrator and a pre-computed test set is provided by the vendor. Therefore, neither fault simulation nor test pattern generation can be performed for such cores. In order for the required testability to be achieved, proper test structures should be incorporated in the system. Several methods have been proposed for coping with testing of IP cores. Some of them embed the pre-computed test vectors in longer pseudorandom sequences, which are generated on chip [1], [12], [13]. The main drawback of these techniques is their long test application time. Thus, many methods reduce the test data volume and test application time by directly compressing the test set, without applying any useless vectors to the core under test (CUT). Various compression codes have been used for encoding the test vectors of a CUT. Golomb codes were proposed in [2]-[4], [17], alternating run length codes in [5], FDR codes in [6], [15], statistical codes in [7], [11], a nine-coded technique in [19], and combinations of codes in [16], [18]. Some methods use dictionaries but since they impose high hardware overhead, due to the required large embedded RAMs, they are not considered further in this paper.

Compression is sometimes performed on the difference vectors instead of the actual test vectors [2], [4], [6], [10]. In this case either cyclical shift registers, which increase the testing cost, should be incorporated in the system, or the scan chains of other cores must be reused, if they are available.

Among the statistical codes used for test data compression, Huffman codes are the most effective ones, since they provably result in the shortest average codeword length [11]. Their main problem is the high hardware overhead of the required decompressors. For that reason, selective Huffman codes were proposed in [11], which significantly reduce the decoder size, by slightly sacrificing the compression ratio.

The high efficiency of all the aforementioned codes is 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 methods, 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. Therefore, compression methods may adversely affect the coverage of un-modeled faults. In [19] it is suggested that, if possible, at least a portion of a test set's 'x' values should be set randomly. This is why the authors of [20] try to match test-data blocks with LFSR sequences.

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 a single code), while, at the same time, requires a very simple decompressor with low hardware overhead. It also offers the ability of exploiting the trade-off between compression ratio and area overhead. The proposed approach does not need any structural information of the CUT, and therefore is 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.

2. Compression Method

The proposed compression method is based on Huffman coding with a limited number of codewords. The test cubes (test vectors with 'x' values) of the CUT are compared against the pseudorandom sequences generated by various cells of an LFSR, and if they match (i.e. they are compatible),

* This research was co-funded by the European Union 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, funded by 25% from national sources and by 75% from the European Social Fund (ESF).

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 as proposed in [11]. Direct test-data coding is, most of the times, used for portions of the test cubes with many defined bits, which are normally incompatible with the LFSR's pseudorandom sequences. On the other hand, the major part of the test data encoded by LFSR cells correspond to the test cubes' 'x'-bits sequences. Therefore, most of the cubes' 'x' values are replaced by pseudorandom data, increasing that way the probability of detection of unmodeled faults. Compared to the approach of [20], which also exploits LFSR-generated pseudorandom sequences, the proposed one compresses more information (three different kinds) with the same Huffman code. In the following we describe the proposed method assuming a single scan chain.

At first, the CUT's test set is partitioned into clusters of fixed length. The LFSR is set to a random initial state and is let evolve for a number of cycles equal to the number of bits of the test set. Then all the clusters of the test set are compared against the normal and inverted pseudorandom sequences generated by each LFSR cell. 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 are selected in order to feed the scan chain(s) of the CUT through a multiplexer. Specifically, the multiplexer selection address of each cell is encoded using Huffman code. We call this type of encoding Cell encoding. All the clusters which are compatible with the sequences generated by the selected LFSR cells are encoded by those cells. The rest of the clusters are labeled as failed and are processed in a different way, as it will be explained later. Apart from encoding the selection address of the chosen LFSR cells according to the corresponding hit ratios, Cell encoding also associates a single Huffman codeword with all failed clusters, in order to distinguish them from the rest.

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

A drawback of the Huffman code is that it is a fixed-to-variable code, whereas variable-to-variable codes are more efficient [7]. In the proposed approach we try to eliminate this problem by allowing, if possible, consecutive clusters to be generated by the same LFSR cell (this is feasible due to the large number of 'x' values in the test sets). All these clusters are encoded using only one codeword, which succeeds the Cell- encoding codeword and indicates the number of consecutive clusters (cluster group) that will be generated. In order 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 experimentally selected to be equal to the powers of 2 in the interval [1, *max_length*), where *max_length* is the maximum number of consecutive clusters matched by the sequence of any of the selected cells. If, for example, *max_length* = 40, then the list of lengths L will be: $L = \{1, 2, 4, 8, 16, 32\}$. Each group of clusters is associated with the largest possible length in the list, which does not exceed the actual length of the group. Assuming the aforementioned list, a group of 30 consecutive clusters will be partitioned into a group of 16, a group of 8, a group of 4 and a group of 2 clusters. These list lengths are also encoded using Huffman code. This choice is justified 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 code is used for Cell and Length encoding, in order to keep the decoding cost low. Therefore the maximum number of potential list lengths is equal to the number of selected cells. In case that there are more selected cells than the volume of the powers of 2 in the interval [1, *max_length*), additional lengths are appended in the list according to the following rule: each additional length is selected iteratively as the mid point 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 mid point between 16 and 32 - distance = 32-16 =16). A Cell encoding codeword is always succeeded 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 with a selective Huffman code as proposed in [11]. We call this encoding Pure-Data encoding. According to the selective-Huffman approach, only the blocks with the highest probabilities of occurrence are encoded. Thus, some blocks remain un-encoded (we call them failed blocks) and are provided directly by the ATE. As in the case of failed clusters, a single Huffman codeword is associated with each failed block, while the other codewords are appointed to the most frequently occurring blocks. The data of an un-encoded block follow its codeword. In Pure-Data encoding the same Huffman code as in Cell and Length encoding is 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. Note that the failed data in both Cell and Pure-Data encoding are distinguished by using only Huffman codewords, in contrast to [11] where an extra bit is used in front of all codewords.

The major advantage of the proposed compression method is that the same Huffman decoder can be used for performing the three different decodings. The size of the Huffman decoder is determined by the number of the selected LFSR cells. Note that the number of selected cells is equal to the number of the list lengths in Length encoding and to the number of unique blocks encoded by Pure-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, to be generated. Thus the same codeword, depending on the mode of the decoding process, corresponds to 3 different kinds of information: to an LFSR cell (normal and/or inverted), to a cluster-group length or to a block of data. Always the first codeword in the code stream is considered as a Cell-codeword. If it does not indicate a failed cluster then the next codeword correspond to the length of the cluster group. If, on the other hand, it corresponds to a failed cluster then the next $\frac{cluster\ size}{block\ size}$ codewords are processed as Pure-Data codewords, where *cluster size* (*block size*) denotes the number of bits of each cluster (block). Each one of them may indicate a failed block or a Pure-Data block. In the first case the actual block of data follows in the code stream, or else the block of data is produced by the decompressor. This sequence is iteratively repeated starting always from a Cell encoding codeword.

Example. Assume a test set of 744 bits. For its encoding we use 4 LFSR cells and, consequently, 4 different cluster-group-list lengths and 4 different encode-able data blocks for each failed cluster. Let each cluster be 24 bit wide and each block 4 bit wide (6 blocks per cluster). Figure 1 presents the selected cells, the available list lengths and the most frequently occurring data blocks sorted in descending order according to their occurrence frequency. Each line of the table (i.e., the respective case for all three encodings) corresponds to a single codeword in the final encoded stream. Note that there are 12 groups of clusters matched by LFSR-cell sequences and 3 failed clusters which are partitioned into 18 blocks. Overall, there are 45 occurrences of encode-able data and 5 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 (45), generating the probability of occurrence of each distinct codeword, as shown in Figure 1. The encoded stream in Figure 1 shows the representation of the data stored in the ATE. The first codeword (0) corresponds to cell A and the next codeword (10) indicates the group length, which is 2. Therefore the scan chain is fed by cell A for the first two clusters. The next codeword (110) indicates that the next cluster is a failed one. According to the proposed compression scheme, cluster 3 is partitioned into 6 blocks. The next codeword (10) indicates that the first block is a failed one as well; therefore the actual data (0010) are not encoded and follow codeword 10. The codeword for the second block is 0 which correspond to the encoded block 0011 that will be shifted in the scan chain. This is repeated until all 6 blocks have been processed. The size of the encoded data stream is 109 bits.

	Cell Encoding		Length Encoding		Pure-Data Encoding			
	Cell	Occur.	List Length	Occur.	Data Block	Occur.	P	Code Word
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							

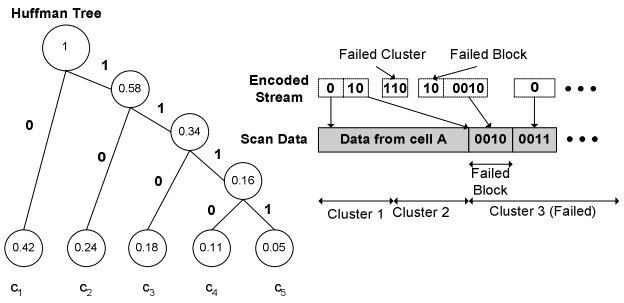

Figure 1. Proposed Encoding Example

3. Decompression Architecture

The block diagram of the proposed decompression architecture is shown in Figure 2. It consists of the following units (the functionality of the proposed architecture has been verified with extensive simulations. For convenience only the most important signals are reported):

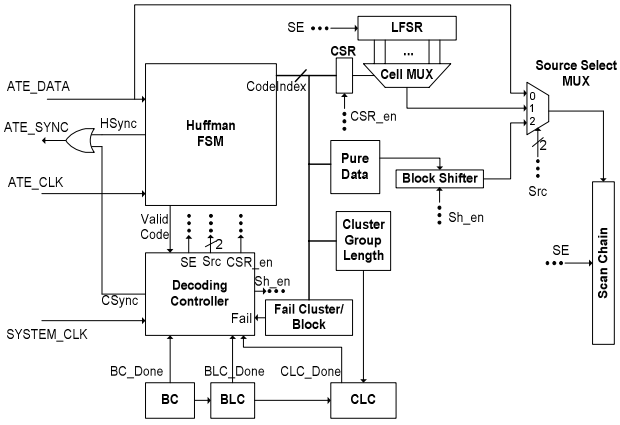

Figure 2. Decompression Architecture

Huffman FSM: This unit receives serially the data from the ATE (*ATE_DATA*) using the ATE clock (*ATE_CLK*). Upon reception of a codeword, the signal *HSync* is sent back to the ATE to stop the transmission until the decompressor is ready to receive the next one. 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*.

***Source Select Mux*:** Selects the source (an LFSR cell, *Pure-Data,* or a failed block from ATE) that will feed the scan chain, by setting bus *Src* to 01, 10 and 00 respectively.
Cell Mux: Selects the cell that will feed the scan chain.
CSR (Cell Select Register): Stores the address of the selected cell when *CSR_en*=1 and holds it during scan loading.
LFSR: It is the Linear Feedback Shift Register. It is enabled by the signal *SE* (Scan Enable) every time the scan chain is loaded with any kind of test data. When pure or unencoded data are fed in the scan chain, the LFSR data are simply ignored.
Pure-Data / Cluster Group Length: Combinational blocks (or Lookup Tables) which receive *CodeIndex* and return the pure data / group length respectively.
Block Shifter: Shifts the data block received by the *Pure-Data* unit in the scan chain. It is controlled by *Sh_en*.
Fail Cluster/Block: Sets *Fail*=1 when a codeword corresponds to a failed cluster or a failed block.
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*=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 into the scan chain.
Decoding Controller: This is a finite state machine which synchronizes the operation of all units. The state diagram of this machine is shown on Figure 3. At *WAIT_CHANNEL*, *WAIT_LENGTH* and *WAIT_FAILED_CLUSTER* states the controller waits for a codeword of Cell, Length and Pure-Data encoding respectively. At states *SHIFT_LFSR_DATA*, *SHIFT_PURE_DATA* and *WAIT_FAILED_BLOCK* the controller loads the scan chain with *LFSR* sequences, Pure-Data encoded blocks and data of failed blocks from ATE, respectively. As for, *CLUSTER_DONE?* state, it checks if all the blocks of a failed cluster have been processed or not. During *WAIT_FAILED_BLOCK* the controller sends the signal *CSync* to the ATE to enable the transmission of an un-encoded block. It also samples *ATE_CLK* and shifts each data bit received from the ATE in the scan chain of the CUT (using signal SE). During this state, *BC* counter is triggered by *ATE_CLK*. Each time a vector is loaded in the scan chain, a counter (not shown in Figure 2) is decreased by one. This counter initially contains the number of test vectors which must be applied to the core. When the counter reaches zero, the end of the test session is indicated.

As it will be shown in Section 4, the efficiency of the proposed encoding approach depends mainly on the number of selected cells, which determine the number of codewords of the Huffman code. The same decompressor can be used for two or more cores by just replacing the units *Cell Mux*, *Pure-Data*, *Cluster Group Length* and *Fail Cluster/Block*, which occupy only a small portion of the area of the decompressor. Moreover, if the *Pure-Data* and *Cluster Group Length* units are implemented as Lookup Tables, they need to be loaded with the specific data of 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. An issue that will be also clarified in Section 4, is how effectively the same Huffman codewords can be used for different cores. As it will be demonstrated, in most cases the reduction in the compression ratio, when using the same codewords, is marginal. This is easily explained if we take into account that, for the same number of cells (same number of codewords) and for relatively skewed frequencies of occurrence, the Huffman trees are not much different and thus the encoding, if not optimal, will be very close to the optimal one. Note that, regardless of 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.

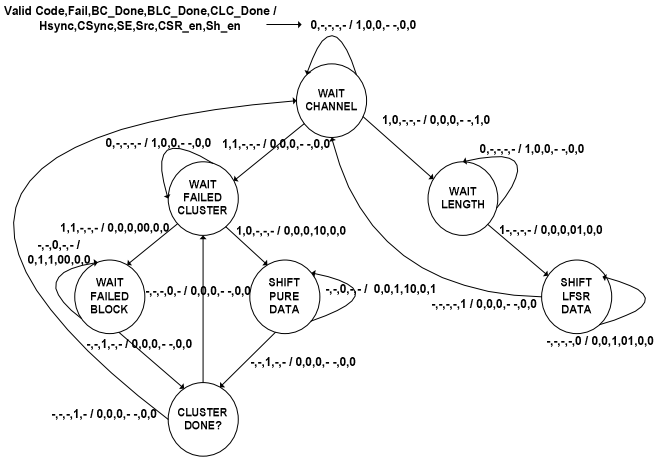

Figure 3. Decoding Controller State Diagram

For applying the proposed scheme to multiple scan chains architectures, a shift register with width equal to the number of scan chains is required. The shift register is loaded by the decompressor and then feeds the scan chains in parallel [19].

Let us now calculate the test application time reduction of the proposed encoding scheme. Suppose that $|D|$, $|E|$ is the size in bits of the uncompressed and compressed test data respectively. The compression ratio is given by the formula $CR=(|D|-|E|)/|D|$. Let f_{ATE}, f_{SYS} be the ATE and system clock frequencies respectively, with $f_{SYS}=m \cdot f_{ATE}$, and *CS*, *BS* be the cluster and block sizes respectively. Also, in the encoded stream, let G_i be the number of occurrences of the cluster group with length L_i ($i = 1...n$, where n is the number of codewords), and F_c, F_b be the number 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 $t_{red}=(t_D-t_E)/t_D$, where t_E is the test application time of the compressed test set. t_E consists of three main parts:
t_1: The time for downloading the data stream from ATE to the core. The data stream consists of codewords which enter the Huffman FSM unit and un-encoded data (failed blocks) which are shifted directly from the ATE to the scan chain. Therefore the application time of this part is $t_1=|E|/f_{ATE}$.

t_2: The time required for loading the scan chain with LFSR sequences of length equal to the number of bits of the decoded cluster groups; therefore, $t_2 = \frac{CS}{f_{SYS}} \sum_{i=1}^{n} G_i L_i$.

t_3: The time for loading the scan chain with Pure-Data encoded blocks (not failed blocks); thus, $t_3 = \frac{F_c \cdot CS - F_b \cdot BS}{f_{SYS}}$.

The total time required for the compressed test set is $t_E = t_1 + t_2 + t_3$ and it can be easily proven that

$$t_{red} = CR - \frac{CS}{|D| \cdot m} \sum_{i=1}^{n} G_i L_i - \frac{(F_c \cdot CS - F_b \cdot BS)}{|D| \cdot m}$$

4. Experimental results

The proposed compression method was implemented in C programming language. We conducted experiments on a Pentium PC for the largest ISCAS '89 benchmarks circuits, assuming full scan and a single scan chain. As input we used the dynamically compacted test sets generated by Mintest [8]. The same test sets were used in [2], [3], [5]-[7], [11], [14], [15] and [17]-[19]. The run time of the compression method is a few seconds for each benchmark circuit. After extensive experiments we deduced that the size and the characteristic (primitive) polynomial of the LFSR, as well as the initial seed used affect the compression ratio marginally. Thus, in our experiments, we utilized an internal-XOR LFSR of size 15. Note that for each LFSR cell, apart from its normal sequence, the inverted one is also considered (normal and inverted cell outputs are considered as different cells).

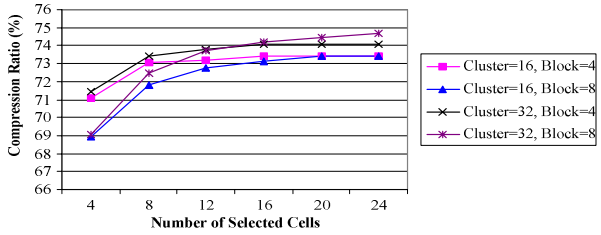

Figure 4. Varying Number of Cells for s15850

In Figure 4 we present the effect of the selected-cells volume on the compression ratio, for several cluster and block sizes. We can see that an increase in the number of cells leads to compression-ratio improvements. The saturation value of each curve depends on the cluster- and block-size values. We also observe that when the number of cells is very small, the proposed method achieves better compression with smaller block sizes.

Table 1. Compression results

	Cells = 4		Cells = 12		Cells = 24	
Circuit	**ENC**	**C, B**	**ENC**	**C, B**	**ENC**	**C, B**
s5378	10416	18, 6	9686	16, 8	**9358**	20, 10
s9234	17449	16, 4	16268	20, 4	**15511**	20, 10
s13207	23021	40, 10	19335	40, 10	**18384**	30, 10
s15850	21738	20, 4	19901	32, 8	**18926**	32, 8
s38417	65545	20, 5	61602	40, 8	**58785**	48, 8
s38584	62955	20, 4	58437	32, 8	**55200**	20, 10

In Table 1 the compression results of the proposed method for 4, 12 and 24 cells are presented. For each cell-volume case, various cluster- and block-size cases were examined. The best results are shown in Table 1. Columns labeled "C, B" report the utilized cluster and block sizes, while the "ENC" columns show the encoded data volumes. We see that compression improves as the number of cells increases.

Table 2. Comparisons with [11] and [19]

	Size				Red. % of prop. over		
Circ.	**Mintest**	**[11]**	**[19]**	**Prop.**	**Mintest**	**[11]**	**[19]**
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

In Table 2 we compare the proposed method against the approach of [11], which is based on selective Huffman coding, and that of [19], 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 volumes of the encoded data of [11], [19] and the proposed method are reported. The reductions achieved over Mintest, [11] and [19] are presented in columns 6-8. It is obvious that the proposed scheme offers better compression results than both [11] and [19]. No comparisons are provided against the approach of [20], which also exploits LFSR-generated pseudorandom sequences, since its ATPG-synergy requirement renders it unsuitable for IP cores of unknown structure.

Table 3. Improvement (%) versus other methods

Circuit	**[2]**	**[17]**	**[3]**	**[15]**	**[5]**	**[6]**	**[7]**	**[18]**	**[14]**
s5378	-	36.0	-	18.0	20.0	24.2	18.3	14.8	34.2
s9234	30.3	35.2	31.0	27.0	28.2	30.0	25.1	24.6	48.5
s13207	55.9	51.6	47.7	38.7	43.7	40.5	32.5	36.4	12.4
s15850	53.5	39.6	38.1	23.2	28.1	27.2	23.3	24.7	24.7
s38417	36.1	20.0	35.5	9.5	9.5	37.1	23.5	0.4	31.0
s38584	47.0	36.1	38.6	25.3	28.7	29.1	26.5	26.3	3.4

In Table 3 we compare the proposed method against other compression techniques for IP cores of unknown structure, which impose similar hardware overhead to the CUT and have reported results for the Mintest test sets. Techniques compressing difference vectors have been excluded from the comparisons, since they require cyclical shift registers. Again, the proposed approach performs better than the rest methods.

As far as the test application time is concerned, it is obvious that as $m = f_{SYS} / f_{ATE}$ increases, greater test application time reductions are achieved. Specifically, for the experiments presented in the sixth column of Table 1, the test time reduction ranges from 14.7% for m=2 to 85.6% for m=30.

For calculating the hardware overhead of the proposed technique, we synthesized three different decompressors for 4, 12 and 24 cells, with cluster size =16 bits and block size = 8 bits, using Leonardo Spectrum (Mentor tools). The *Pure-Data* and *Cluster Group Length* units were implemented as combinational circuits. The resulted area overhead is 203, 314 and 432 gate equivalents respectively (a gate equivalent

corresponds to a 2-input NAND gate). The hardware overhead, in gate equivalents, for the most efficient methods in the literature is: 416 for [19], 320 for [5], 136-296 for [7] and 125-307 for [2] (as reported in [7]). In [11] the hardware overhead is provided as a percentage of the benchmark circuits' area and cannot be directly compared to the above methods. However, it is greater than that of [7]. As can be seen, the hardware overhead imposed by the proposed decompressors is comparable to that of the rest techniques.

The hardware overhead of the proposed method can be reduced if the same decompressor is used for testing, one after the other, several cores of a chip. The units *Huffman FSM*, *Decoding Controller*, *BC*, *BLC*, *CLC*, *CSR*, *LFSR* and the *Source Select Mux* of the decompressor can be implemented only once on the chip. On the other hand the units *Pure-Data*, *Cluster Group Length*, *Fail Cluster/Block* and *Cell Mux* should be implemented for every core under test. The area occupied by the latter units is equal to 5.9%, 16.6% and 23.4% of the total decompressor area for 4, 12 and 24 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.

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 every core, while the actual cells, list lengths and data blocks do not have to be the same. The question is if this common-FSM choice can affect the compression efficiency. As we have already seen in Figure 4, the compression improves (up to a saturation value of course) as the number of selected cells (which is defined by the designer) increases. The cell volume affects also the implementation cost. Therefore, if we want to use one decompressor for several cores, it is preferable to implement the decompressor for the core requiring the largest number of cells and then reuse it for the rest cores. For assessing the influence of the utilization of the same codewords for different cores, we generated for the test set of s15850, the codes for 8 selected cells and various cluster and block sizes, and among them we selected the cluster and block size providing the maximum compression. The same procedure was followed for the rest benchmarks circuits. We then compressed their test sets again using the pre-generated code of s15850. The results showed that the reduction in compression ratio varies between 0% and 0.45%. Thus, we conclude that the difference in compression is very small.

5. Conclusion

In this paper a test-data compression method based on multilevel Huffman coding was presented. Different kinds of information are encoded using the same Huffman code, and thus improved compression results can be achieved. The area overhead of the required decompressor is very low. Furthermore, most of the test set's 'x' values are filled with pseudo-random data generated by an LFSR, which leads to increased probability of detection of unmodeled faults.

References

[1] K. Chakrabarty et al., "Deterministic Built-In Test Pattern Generation for High-Performance Circuits using Twisted-Ring Counters", *IEEE Trans. on VLSI Systems*, vol. 8, no. 5, pp. 633-636, 2000.

[2] A. Chandra and K. Chakrabarty, "System-on-a-Chip Test-Data Compression and Decompression Architectures Based on Golomb Codes", *IEEE Trans. on CAD*, vol. 20, no. 3, pp. 355-368, 2001.

[3] Chandra A. and Chakrabarty K., "Test Data Compression and Decompression Based on Internal Scan Chains and Golomb Coding", *IEEE Trans. on CAD*, vol. 21, no. 6, pp. 715-72, 2002.

[4] A. Chandra et al., "How Effective are Compression Codes for Reducing Test Data Volume?", Proc. of IEEE VTS, pp. 91-96, 2002.

[5] A. Chandra and K. Chakrabarty, "A Unified Approach to Reduce SOC Test Data Volume, Scan Power and Testing Time", *IEEE Trans. on CAD*, vol. 22, no. 3, pp. 352-363, 2003.

[6] 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. on Computers*, vol. 52 , no. 8, pp. 1076-1088, 2003.

[7] P.T. Gonciari, B.M. Al-Hashimi and N. Nicolici, "Variable-Length Input Huffman Coding for System-On-A-Chip Test", *IEEE Trans. on CAD*, vol. 22 , no. 6, pp. 783-796, 2003.

[8] I. Hamzaoglu and J. H. Patel, "Test Set Compaction Algorithms for Combinational Circuits", *IEEE Trans. on CAD*, vol. 19, no. 8, pp. 957-963, 2000.

[9] G. Hetherington et al., "Logic BIST for Large Industrial Designs: Real Issues and Case Studies", Proc. of ITC, pp. 358-367, 1999.

[10] A. Jas and N. A. Touba, "Test Vector Decompression via Cyclical Scan Chains and Its Application to Testing Core-Based Designs", Proc. of ITC, pp. 458-464, 1998.

[11] A. Jas, J. Ghosh-Dastidar, M. -E. Ng and N. A. Touba, "An Efficient Test Vector Compression Scheme Using Selective Huffman Coding", *IEEE Trans. on CAD*, vol.22, no.6, pp.797-806, 2003.

[12] D. Kaseridis et al., "An Efficient Test Set Embedding Scheme with Reduced Test Data Storage and Test Sequence Length Requirements for Scan-based Testing", Inf. Papers Dig. of IEEE ETS, pp. 147-150, 2005.

[13] L. Lei and K. Chakrabarty, "Test Set Embedding for Deterministic BIST Using A Reconfigurable Interconnection Network", *IEEE Trans. on CAD*, vol.23, no. 12, pp. 1289-1305, 2004.

[14] Lei Li et al., "Efficient space/time compression to reduce test data volume and testing time for IP cores", Proc. of Int. Conf. on VLSI Design, pp. 53-58, 2005.

[15] A.H. El-Maleh and R.H. Al-Abaji, "Extended Frequency-Directed Run-Length Code with Improved Application to System-On-A-Chip Test Data Compression" Proc of IEEE ICECS, vol. 2, pp. 449-452, 2002.

[16] M. Nourani and M. H. Tehranipour, "RL-huffman Encoding for Test Compression and Power Reduction in Scan Applications", *ACM Trans. on Des. Aut. of El. Syst.*, vol.10, no.1, pp. 91-115, 2004.

[17] P. Rosinger et al., "Simultaneous Reduction in Volume of Test Data and Power Dissipation for Systems-On-A-Chip", *Electr. Letters*, vol. 37, no. 24, pp. 1434-1436, 2001.

[18] M. Tehranipour et al., "Mixed RL-Huffman Encoding for Power Reduction and Data Compression in Scan Test", Proc. of ISCAS, vol. 2, pp. II- 681-4, 2004.

[19] M. Tehranipour, M. Nourani and K. Chakrabarty, "Nine-Coded Compression Technique for Testing Embedded Cores in SoCs", *IEEE Trans. On VLSI Systems*, vol. 13, no.6, pp. 719-731, 2005.

[20] E.H. Volkerink, A. Khoche and S. Mitra, "Packet-Based Input Test Data Compression Techniques", Proc. ITC, pp. 154-163, 2002.