A Parallel Multilevel-Huffman Decompression Scheme for IP Cores with Multiple Scan Chains

X. Kavousianos[1], E. Kalligeros[2] and D. Nikolos[2]
[1]Computer Science Dept., University of Ioannina, 45110 Ioannina, Greece
[2]Computer Engineering & Informatics Dept., University of Patras, 26500 Patras, Greece

Abstract

Various efficient compression methods have been proposed for tackling the problem of increased test-data volume of contemporary, core-based Systems-on-Chip (SoCs). However, many of them cannot exploit the test-application-time advantage that cores with multiple scan chains offer, since they are not able to perform parallel decompression of the encoded data. For eliminating this problem, we present a new, low-overhead decompression scheme that can generate clusters of test bits in parallel. The test data are encoded using a recently proposed and very effective compression method called multilevel Huffman. Thus, apart from the significantly reduced test-application times, the proposed approach offers high compression ratios, as well as increased probability of detection of unmodeled faults, since the majority of the unspecified bits of the test sets are replaced by pseudorandom data. The time/space advantages of the proposed approach are validated by thorough experiments.

1. Introduction

In order to meet tight time-to-market constraints, contemporary systems embed pre-designed and pre-verified modules called IP (Intellectual Property) cores. The structure of IP cores is often hidden from the system integrator and as a result, neither fault simulation nor test pattern generation can be performed for them. IP cores are delivered along with a pre-computed test set and if they are not BIST-ready, proper test structures should be incorporated in the system.

Various methods have been proposed to cope with testing of IP cores. Some of them embed the pre-computed test vectors in longer pseudorandom sequences generated on chip [1], [10], [12]. The major drawback of such methods is their long test application time. Thus, many techniques, using various compression codes, have been proposed for direct test-data encoding. Golomb codes were proposed in [2], [3], [17], alternating run length codes in [4], FDR codes in [5], [14], statistical codes in [6], [8], [11], a nine-coded technique in [19], and combinations of codes in [15], [18]. An additional advantage of these approaches is that they can operate on fully compacted test sets thus allowing further test-time reductions. Some methods use dictionaries but impose high hardware overhead due to the large embedded RAMs required.

Usually, test sets, even if they are dynamically and/or statically compacted, include large numbers of 'x' values. Traditionally, ATPG tools fill these 'x' values randomly with logic 0 or 1, so as to increase the fortuitous detection of unmodeled faults. On the contrary, compression methods, in order to achieve high compression ratios, replace these 'x' values with the logic values 0 and/or 1, depending on the characteristics of the implemented code. Therefore, compression methods may adversely affect the coverage of unmodeled faults. The authors of [19] suggest that, if possible, at least a portion of a test set's 'x' values should be set randomly. This is why in [11] and [20] LFSRs are used for generating whole clusters of test data.

Another common problem of many compression methods is their inability of exploiting the test-application-time advantages that a core with multiple scan chains offers. In other words, if parallel decompression is not possible, a serial-in parallel-out register must be used for spreading the decoded data in the scan chains and as a result, no test-time savings, due to the multiple scan chains, are possible. The solution of using more than one decoder is too expensive and thus inapplicable. For that reason, in this paper we propose a test-data compression scheme that can generate whole clusters of decoded data in parallel. It will be shown that the proposed scheme manages to exploit most of the offered parallelism with low hardware overhead (comparable to that of single-scan-chain schemes). The test-data are compressed using the recently proposed and very effective multilevel Huffman encoding method of [11]. Thus, the proposed approach offers high compression ratios as well as increased probability of detection of unmodeled faults, since most of the test sets' 'x' bits are replaced by pseudorandom data.

The paper is organized as follows: since it is not straightforward, in Section 2 we explain how the multilevel Huffman approach can be applied to multiple-scan-chain cores. The required structures are also introduced. In Section 3 the proposed decompressor is discussed in detail, while in Section 4 the proposed technique is evaluated with experimental results and comparisons. Section 5 concludes the paper.

2. Compression Method

Consider a core with N_{sc} balanced scan chains of W_{sc} scan cells each, as shown in Figure 1:

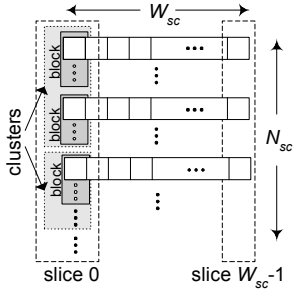

Figure 1. Scan chains, clusters and blocks

Each test cube (test vector with 'x' values) is partitioned into W_{sc} consecutive slices of N_{sc} bits, according to the scan-

chain structure of the core. In other words, a test slice consists of the test bits contained in the scan cells of a vertical cross-section of the scan chains. W_{sc} scan cycles are required for loading the scan chains. In case of a not perfectly balanced scan structure (scan chains are not equally sized), the short test slices are padded with 'x' values. Each test slice is partitioned into clusters of size *CS*. If N_{sc} is not divided exactly by *CS*, then the last cluster of all slices is shorter than the others. In other words each test cube is partitioned into $W_{sc} \cdot \lceil N_{sc}/CS \rceil$ test clusters.

The proposed encoding scheme is based on pseudorandom bit generation and multilevel Huffman coding. According to the multilevel Huffman approach, the same Huffman code is used for encoding different sets of information (three in our case, as it will be explained later). As pseudorandom generator we use a small LFSR and a phase shifter, which can produce pseudorandom clusters of size *CS*. The phase shifter is initially designed as proposed in [16]. However, since we want to be able to choose among different sequences of pseudorandom clusters for the same time period, we add an extra input to each XOR tree [9]. This extra input is driven, through a multiplexer, by various cells of the LFSR. For every different cell, a different sequence of pseudorandom clusters is generated at the outputs of the phase shifter. The pseudorandom generator is shown in Figure 2.

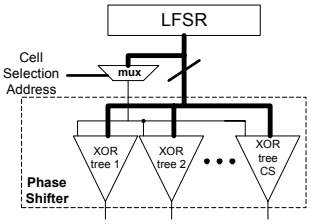

Figure 2. Pseudorandom Generator

At first, the LFSR is set to a random initial state and is let evolve for a number of cycles equal to the total number of the test clusters. Then all the test clusters are compared against the corresponding pseudorandom clusters of the cluster sequences generated when each LFSR-cell's output is fed to the phase shifter's extra input. If a test cluster is compatible with a pseudorandom cluster belonging in the sequence of cell *i*, a hit for cell *i* occurs. A designer-defined number of LFSR cells with the largest hit ratios are selected in order to feed the extra input of the phase shifter. The multiplexer selection address of each chosen cell is encoded using Huffman coding, i.e. each Huffman codeword is used for enabling an LFSR cell to drive the extra input of the phase shifter. We call this type of encoding *Cell encoding*. Since many test clusters have a large number of 'x' values, they are compatible with pseudorandom clusters generated when different LFSR cells feed the phase shifter's extra input. The proposed method associates each cluster with the cell that skews the cell-occurrence probabilities the most. Test clusters that are not compatible with any pseudorandom clusters are labeled as failed and a single Huffman codeword is used for distinguishing them from the rest. We note that both the normal and inverted outputs of the LFSR cells are considered during the aforementioned cell-selection procedure.

If consecutive test clusters can be generated by using the same LFSR cell, we encode them with only one codeword, which succeeds the Cell-encoding codeword and indicates the number of consecutive clusters (cluster-group length) that will be generated. In order to keep the cost low, the available lengths are chosen from a predetermined list of distinct lengths (group-length quantization). These distinct lengths are equal to the powers of 2 in the interval [1, *max_length*), where *max_length* is the maximum number of consecutive test clusters which are compatible with pseudorandom ones. We call this type of Huffman encoding, *Length encoding*. A cluster group with a length not included in the list, is partitioned into smaller groups of proper length. A Cell-encoding codeword is always followed by a Length-encoding codeword, if the encoded cluster is not a failed one.

All failed clusters are partitioned into blocks of size *BS*, and the blocks with the highest probabilities of occurrence are encoded using a selective Huffman code as proposed in [8]. We call this encoding *Block encoding*. Some blocks remain un-encoded (failed blocks) and are embedded in the compressed test set. As in the case of failed clusters, a single Huffman codeword is associated with each failed block, contrary to [8] where an extra bit is used in front of all codewords.

The advantage of the proposed compression method is that the same Huffman decompressor can be used for implementing the three different decodings (Cell, Length and Block). Note that one codeword is used for each cell. As the same codewords are used for all three encodings, the number of selected cells is equal to the number of list lengths in *Length encoding* and to the number of unique blocks encoded by *Block encoding*. The Huffman tree is constructed by summing the corresponding occurrence probabilities of the 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 encoded-data stream is considered as a Cell-codeword. If it does not indicate a failed cluster, then the next codeword determines the length of the cluster group. If, on the other hand, it corresponds to a failed cluster, the next *CS/BS* codewords are processed as Block codewords. Each of the Block codewords may indicate a failed block or a Huffman encoded block. In the former case the actual block of data is embedded in the encoded-data stream, while in the later case the block of data is generated by the decompressor. This sequence is iteratively repeated starting always from a Cell encoding codeword.

Example. Consider a core with 48 scan chains and a test set of 768 bits. For its encoding we use 4 LFSR cells and thus 4 cluster-group-list lengths and 4 encode-able data blocks. Let each cluster be 24 bit wide (2 clusters per slice) and each block 4 bit wide (6 blocks per cluster). Figure 3 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) corre-

sponds to a single codeword in the final encoded stream. Note that there are 13 groups of clusters matched by LFSR-cell sequences and 3 failed clusters which are partitioned into 18 blocks. Overall, there are 47 occurrences of encodeable 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 (47), generating the probability of occurrence of each distinct codeword, as shown in Figure 3. The encoded stream in Figure 3 corresponds to the data stored in the ATE. Initially (t_0) the first slice is undefined (u). The first codeword (0) corresponds to cell A, and the next codeword (10) indicates the group length, which is 2 according to the table. Therefore the first slice is filled by using cell A (Cluster 1 in t_1 and Cluster 2 in t_2). The next codeword (110) indicates that the next cluster is a failed one. According to the proposed compression scheme, this cluster 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 in the code stream. The codeword for the second block is 0 which correspond to the encoded block 0011. This is repeated until all 6 blocks have been processed. The size of the encoded data is 111 bits.

	Cell Encoding		Length Encoding		Block Encoding			
	Cell	Occur.	List Length	Occur.	Data Block	Occur.	P	Code Word
c_1	A	6	1	7	0011	8	(6+7+8)/47	0
c_2	B	4	2	3	Fail	4	(4+3+4)/47	10
c_3	Fail	3	4	2	0000	3	(3+2+3)/47	110
c_4	C	2	8	1	1111	2	(2+1+2)/47	1110
c_5	D	1			0001	1	(1+1)/47	1111
	SUM	16		13		18		

Total Sum = 47

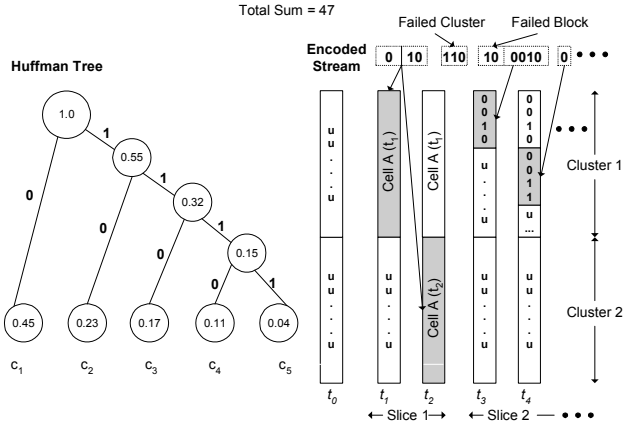

Figure 3. Proposed Encoding Example

3. Decompression Architecture

The block diagram of the proposed decompression architecture is shown in Figure 4. The Input Buffer receives the encoded data from the ATE channels (*ATE_DATA*) with the frequency of the ATE clock (*ATE_CLK*), shifts them into Huffman FSM unit with the frequency of the system clock, and activates signal *Empty* to notify ATE so as to send the next test data. When the Huffman FSM recognizes a codeword, it informs Input Buffer to stop sending test data and, assuming that the implemented code consists of *N* codewords, it places on the bus *CodeIndex* a binary value between 0 and *N*-1, which is used as the *Cell Selection Address* (Figure 2). It also sets *Valid Code=1*. Units Block and Cluster Group Length which are combinational blocks (or lookup tables) return respectively the encoded block and group length that correspond to *CodeIndex*. As it was shown in Section 2, a failed cluster is partitioned into blocks and each block is either Huffman encoded or not encoded (embedded as is into the compressed data). These two cases are distinguished by the signal *Select Huffman/ATE* and the selected data are driven through multiplexing logic to the Source Select unit. This unit receives pseudorandom clusters and blocks of failed clusters and, depending on the *Select Cluster/Block* signal, it constructs the slice that will enter the scan chains.

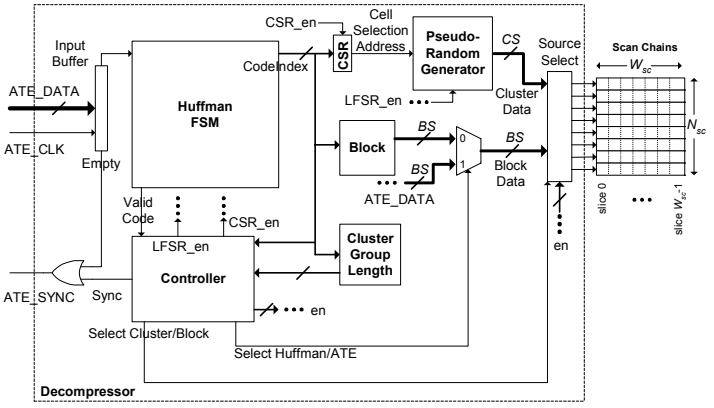

Figure 4. Decompression Architecture

The controller is a finite state machine which synchronizes the operation of all units. Its state diagram is shown in Figure 5 (the most important signals are presented). Initially the controller waits for the first codeword to be received (*Valid Code=1*). If this codeword indicates a non-failed cluster (this is determined by the value of bus *CodeIndex*), the controller sets *CSR_en*=1 so as to store the cell address to the CSR resister and proceeds to *WAIT_LENGTH* state. Otherwise, it reaches state *WAIT_FAILED_CLUSTER*. At *WAIT_LENGTH* state the controller waits for the next codeword and upon reception stores the data returned by the Cluster Group Length unit (length of the group) and sets *Select Cluster/Block=0* in order to enable pseudorandom data to enter the Source Select unit. Then it proceeds to the state *LOAD_CLUSTER_GROUP* where it remains for a number of clock cycles equal to the length of the group. During these cycles the LFSR evolves (*LFSR_en*=1) and the produced pseudorandom clusters are loaded into the Source Select unit. Every time a whole test slice is ready, it is loaded into the scan chains. After the end of these cycles the state machine returns to *WAIT_CHANNEL* state for the next iteration.

In the *WAIT_FAILED_CLUSTER* state the controller waits for the next codeword. If this codeword corresponds to an encoded block, the controller sets *Select Huffman/ATE*=0 and *Select Cluster/Block*=1 in order to drive the output of the Block unit (which is the decoded data block) into the Source Select unit, and proceeds to the *CLUSTER_DONE?* state. On

the other hand if the received codeword corresponds to a failed block then the controller proceeds to the *WAIT_FAILED_BLOCK* state and sets *Select Huffman/ATE*=1, *Select Cluster/Block*=1 and *Sync*=1 to enable the ATE to send the data block. Then it samples the *ATE_CLK* and when the data from the ATE are available, they are driven directly to the Source Select unit (Input Buffer is bypassed). From the *WAIT_FAILED_BLOCK* state the controller proceeds to the *CLUSTER_DONE?* state. If all blocks of the failed cluster have been handled, the LFSR is let evolve for one clock cycle (*LFSR_en*=1) and the next state is *WAIT_CHANNEL*. Otherwise the controller proceeds to state *WAIT_FAILED_CLUSTER* in order to process the next block.

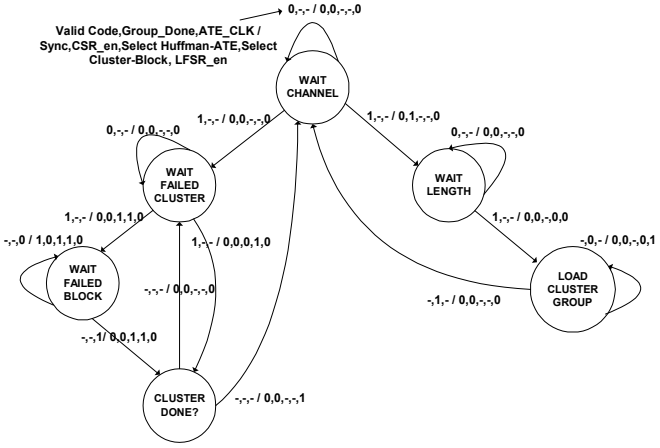

Figure 5. Controller State Diagram

The Source Select unit is shown in Figure 6a. It receives cluster data produced by the Pseudorandom Generator (encoded clusters - *Cluster Data* bus), as well as block data either by the Block unit (Huffman encoded blocks - *Block Data* bus) or by the ATE (failed blocks). The received data are stored in a buffer (Scan Buffer) of size equal to that of a test slice (N_{sc}). This buffer consists of $\lceil N_{sc}/BS \rceil$ registers with size equal to *BS* each, grouped into $k=\lceil N_{sc}/CS \rceil$ groups of $w=CS/BS$ registers. All Buffer Groups are loaded in a round robin fashion (Buffer Group *i* is loaded after Buffer Group *i*-1). When *SelectCluster/Block*=0 the *Cluster Data* bus (of width *CS*) loads, through the Mux unit, all registers of a group simultaneously (in a single clock cycle), while when *SelectCluster/Block*=1 the *Block Data* bus (of width *BS*) is driven to every register (*w* clock cycles are needed for loading a whole group). This operation is handled by the controller through the use of *w* enable signals $en_{iw}...en_{(i+1)w-1}$, one for each register in the group (Buffer Group *i* is shown in Figure 6b). Totally, $k \cdot w$ enable signals are generated for the whole Scan Buffer. In order for a cluster to be loaded into Buffer Group *i* by the Pseudorandom Generator, all *w* enable signals of this group are activated. When a failed cluster is loaded into Buffer Group *i*, group's *i* registers are enabled one after the other, until all the blocks of the failed cluster are loaded into the corresponding registers (the enable signals are one-hot encoded). When the whole Scan Buffer is full the scan chains are loaded.

The Scan Buffer can be avoided if the core is equipped with a separate scan enable or clock signal for each scan chain. Then the scan chains can be loaded directly without the interference of the buffer, using the enable signals for driving the scan enables, or for gating the clock of each scan chain.

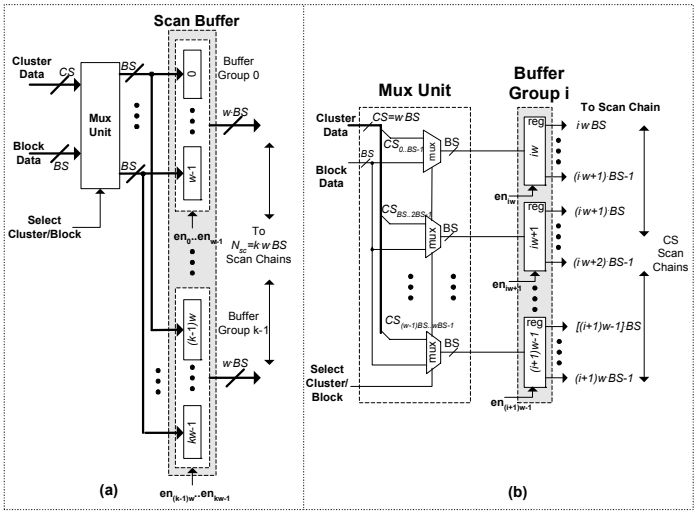

Figure 6. Source Select unit

As it will be shown, the encoding efficiency of the proposed method 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 changing only the units Block and Cluster Group Length, as well as the multiplexer in the Pseudorandom Generator, which occupy only a small portion of the total area. Moreover, if the Block 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. In [11] it was shown that the compression ratio reduction in the case of utilizing the same decompressor for multiple cores, due to the use of the same codewords, is only marginal. This is easily explained if we take into account that, for the same number of cells (same number of Huffman codewords) and 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 utilized, the selected cells, list lengths, encoded blocks, cluster size and block size do not have to be the same for different cores.

Let us now calculate the test application time. Suppose that $|D|$, $|E|$ is the size in bits of the uncompressed and compressed test set 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 N_{ch} be the number of channels available for downloading the test data from the ATE. Also, let G_i be the number of occurrences of cluster groups with length L_i 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 $t_D=|D|/(N_{ch} \cdot f_{ATE})$ and the reduction is given by the formula

Table 1. Compression Results

core	Min-Test	20 scan chains			40 scan chains			80 scan chains			100 scan chains			Red. (%)
		8 cells	16 cells	24 cells	8 cells	16 cells	24 cells	8 cells	16 cells	24 cells	8 cells	16 cells	24 cells	
s5378	23754	9597	9420	**9247**	9702	9470	9261	9713	9427	9338	10029	9697	9521	61,1
s9234	39273	16595	16056	15787	16746	16201	**15722**	17095	16330	15860	16995	16358	15923	60,0
s13207	165200	21865	20258	19400	20363	18973	18543	20319	18840	18381	19512	18593	**18153**	89,0
s15850	76986	20844	20143	19630	20715	19754	19326	20687	19763	**19313**	20921	20162	19329	74,9
s38417	164736	65372	63725	62227	63569	60585	59078	63420	60593	59026	63184	60140	**58706**	64,4
s38584	199104	62770	61891	59750	62449	59699	**57801**	62989	60776	58381	62412	60584	58518	71,0

$t_{red}=(t_D-t_E)/t_D$, where t_E is the test application time of the compressed test set. t_E consists of four parts:

t_1. The time required for downloading the data (codewords and failed blocks) from the ATE to the core: $t_1=|E|/(N_{ch}f_{ATE})$.

t_2. The time for the serialization of codewords by the Input Buffer (note that the compressed test set contains also failed blocks which do not require serialization): $t_2=(|E|-F_b \cdot BS)/f_{SYS}$.

t_3. The time required for loading the scan chains with pseudorandom sequences of length equal to the number of the decoded cluster groups (each cluster requires a single clock cycle so as to be loaded): $t_3 = \frac{1}{f_{SYS}} \sum_i G_i L_i$.

t_4. The time required for loading the scan chains with failed clusters. Each cluster is partitioned into *CS*/*BS* blocks and a single clock cycle is required for loading each block (either from the Block unit or the ATE). We note that the time required for downloading the failed blocks from the ATE has been taken into account in t_1. Thus: $t_4=F_b \cdot CS/(BS \cdot f_{SYS})$.

The total time required for the compressed test set is $t_E=t_1+t_2+t_3+t_4$ and it can be easily proven that

$$t_{red} = CR - \frac{N_{ch}}{|D| \cdot m}\left(|E| - F_b \cdot BS + \frac{F_c CS}{BS} + \sum_i G_i L_i\right)$$

4. Evaluation and comparisons

The proposed compression method was implemented in C programming language. We conducted our experiments on a Pentium PC for the largest ISCAS '89 benchmarks circuits using the dynamically compacted test sets generated by the Mintest ATPG program [7]. The same test sets were also used in [2]-[6], [8], [13], [14], [17]-[19]. The run time of the compression method is a few seconds for each benchmark circuit. In the experiments that we will present, a primitive-polynomial LFSR with internal XOR gates and size 20 was used. Each XOR tree of the phase shifter consists of 3 gates. Block size (*BS*) was considered equal to N_{ch} and ranged from 5 to 10, while cluster-size values (*CS*) ranged from 20 to 50. Note that normal and inverted LFSR-cell outputs can be selected as different cells.

In Table 1 the compression results of the proposed method for 20, 40, 80 and 100 scan chains, and 8, 16 and 24 cells are presented. For each cell-volume case, various cluster and block sizes were examined. Among them the best results are reported. In column 2 the sizes of the original Mintest test sets are shown. It is obvious that the compression improves as the number of cells increases. Last column presents the reduction achieved by the best results of the proposed method (boldfaced) over Mintest.

Table 2. Compression improvement (%) vs. other methods

Circ.	[2]	[3]	[4]	[5]	[6]	[8]	[13]	[14]	[17]	[18]	[19]
s5378	-	-	20.9	25.1	19.3	13.3	35.0	19.0	36.7	15.8	12.0
s9234	29.3	30.1	27.3	29.0	24.1	12.6	47.8	26.0	34.3	23.6	11.5
s13207	56.4	48.3	44.4	41.2	33.4	52.2	13.5	39.5	52.2	37.2	25.8
s15850	52.6	36.8	26.6	25.7	21.8	26.2	23.2	21.6	38.3	23.2	12.7
s38417	36.2	35.6	9.6	37.2	23.6	13.1	31.1	9.6	20.1	0.5	4.0
s38584	44.5	35.7	25.3	25.7	23.0	19.1	-1.2	21.7	33.1	22.8	8.1

In Table 2 we present the comparisons of the proposed method against other compression techniques in the literature that have reported results for the Mintest test sets. It can be seen that the proposed approach performs better than all the other methods except of the case of s38584 of [13] which provides a marginally better result. Compared to the single-scan-chain Multilevel Huffman approach of [11] the compression results are similar and therefore are not appended. We note that 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.

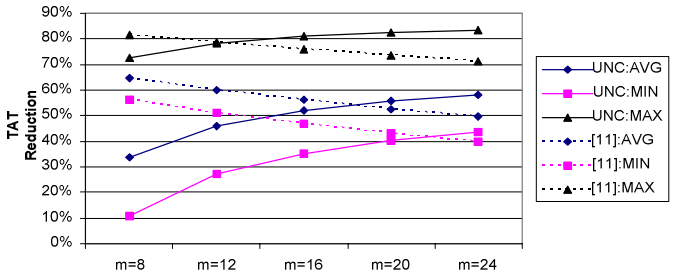

Figure 7. TAT reduction.

For assessing the test application time (TAT) improvements of our method we performed two sets of experiments, for the boldfaced cases of Table 1. In the first one we study the reduction of the test application time achieved against the case in which the test set is downloaded uncompacted (UNC) to the core, using the same number of channels. Figure 7 presents the average (UNC:AVG), minimum (UNC:MIN) and maximum (UNC:MAX) improvement for various values of $m=f_{SYS}/f_{ATE}$ for all benchmarks. It is obvious that as *m* increases, the test-time gain becomes greater. In the second set of experiments we compare the test application time of the proposed method against the single-scan-chain Multilevel Huffman approach of [11]. Since [11] considers only one channel for downloading data from ATE, we recalculated its test application time for the channel volumes used in this paper (an input buffer is appended in [11] too). The best results of the proposed method and [11] have been compared and the average ([11]:AVG), minimum ([11]:MIN) and maximum ([11]:MAX) improvement for various values

of *m* for all benchmarks, are shown in Figure 7. It is obvious that the test application time gain is very high in all cases (40%-81.6%). However, although the test-time gain attributed to the parallel loading of multiple scan chains is constant, the serialization of the decoder input data is carried out faster as *m* increases and thus the test-time reduction drops.

For calculating the hardware overhead of the proposed technique, we synthesized three different decompressors using Leonardo Spectrum (Mentor tools) for 8, 16 and 24 cells, assuming 10 ATE channels, 40 scan chains, $CS = 20$ bits and $BS = 10$ bits. The Block and Cluster Group Length units were implemented as combinational circuits. The resulted area overhead is 377, 473 and 582 gate equivalents respectively (a gate equivalent corresponds to a 2-input NAND gate). In this overhead we have not considered the Scan Buffer which can be avoided and is not considered in the other methods too. The hardware overhead, in gate equivalents, for the most efficient methods in the literature is: 416 for [19], 320 for [4], 136-296 for [6], 125-307 for [2] (as reported in [6]) and 203-432 for [11], while the hardware overhead of [8], although not reported directly, is greater than that of [6]. As can be seen, the hardware overhead of the proposed decompressor is comparable to that of the rest techniques, even though all of them do not exploit the advantages of multiple scan chains (i.e., perform serial decoding which is a simpler and less hardware intensive case).

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. Units Huffman FSM, Controller, CSR, Source Select, as well as the LFSR and the phase shifter can be implemented only once on the chip. On the other hand, units Block, Cluster Group Length and the multiplexer of the Pseudorandom Generator must be implemented for every core under test. The area occupied by the latter units is equal to 7.7%, 14% and 19.7% of the total area of the decompressor for 8, 16 and 24 cells respectively. Therefore, only a small amount of hardware should be added for each additional 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, while the actual cells, list lengths and data blocks can be different. As shown in [11], in such a case, the compression ratio suffers only a marginal decrease.

5. Conclusion

A test-data compression method that can exploit the existence of multiple scan chains in a core in order to reduce the test-application time has been presented. Multilevel Huffman coding, properly adapted to the multiple-scan-chains case, is used for compressing the test data, while a low-overhead decompressor capable of generating whole clusters of test bits in parallel is also introduced. The proposed method offers reduced test-application times, high compression ratios and increased probability of detection of unmodeled faults, since most of the test sets' 'x' bits are replaced by pseudorandom values.

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*, pp. 633-636, Oct. 2000.

[2] A. Chandra, 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., Chakrabarty K., "Test Data Compression and Decompression Based on Internal Scan Chains and Golomb Coding", *IEEE Trans. on CAD*, vol. 21, pp. 715-72, June 2002.

[4] A. Chandra, 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.

[5] A. Chandra, 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.

[6] P.T. Gonciari, B.M. Al-Hashimi, 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.

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

[8] 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.

[9] E. Kalligeros et al., "Multiphase BIST: A New Reseeding Technique for High Test-Data Compression", *IEEE Trans. on CAD*, vol. 23, no. 10, pp. 1429-1446.

[10] 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.

[11] X. Kavousianos et al., "Efficient Test-Data Compression for IP Cores Using Multilevel Huffman Coding", DATE 06, *to appear*.

[12] L. Lei, K. Chakrabarty, "Test Set Embedding for Deterministic BIST Using A Reconfigurable Interconnection Network", *IEEE Trans. on CAD*, vol.23, pp. 1289- 1305, Dec. 2004.

[13] 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.

[14] A. El-Maleh, R. 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.

[15] M. Nourani, M. H. Tehranipour, "RL-huffman Encoding for Test Compression and Power Reduction in Scan Applications", *ACM Trans. on Design Automation of Electronic Systems*, vol. 10, no. 1, pp. 91 – 115, 2004.

[16] J. Rajski, N. Tamarapalli, and J. Tyszer, "Automated synthesis of phase shifters for built-in self-test applications," IEEE Trans. Computer-AidedDesign, vol. 19, pp. 1175–1188, Oct. 2000.

[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, K. Chakrabarty, "Nine-Coded Compression Technique for Testing Embedded Cores in SoCs", *IEEE Trans. On VLSI Systems*, vol. 13, pp. 719-731, June 2005.

[20] E.H. Volkerink, A. Khoche, S. Mitra, "Packet-Based Input Test Data Compression Techniques", Proc. ITC, pp. 154-163, 2000.