Test-Data Compression Based on Variable-to-Variable Reusable Huffman Coding

X. Kavousianos[1], E. Kalligeros[1,2] and D. Nikolos[3]
[1]*Computer Science Dept., University of Ioannina, 45110 Ioannina, Greece*
[2]*Dept. of Computer Science & Technology, University of Peloponnese, Terma Karaiskaki, 22100 Tripoli, Greece*
[3]*Computer Engineering & Informatics Dept., University of Patras, 26500 Patras, Greece*

Abstract

A new efficient statistical test data compression method, suitable for IP cores of unknown structure with multiple scan chains is proposed. Huffman, which is a well known fixed-to-variable code, is used in this paper as a variable-to-variable code. The pre-computed test set of a core is partitioned into variable-length blocks, which are then compressed by an efficient Huffman-based encoding procedure with a limited number of codewords. For increasing the compression ratio, the same codeword can be reused for encoding compatible blocks of different sizes. Further compression improvements can be achieved by using two very simple test-set transformations. A low-overhead decompression architecture is also proposed.

1. Introduction

The extensive use of pre-designed and pre-verified cores in contemporary *Systems-on-a-Chip* (SoCs) makes their testing an increasingly challenging task. The testing problem is further exacerbated by the limited channel capacity, memory and speed of *Automatic Test Equipments* (ATEs). Embedded testing overcomes these difficulties by combining ATE capabilities with on-chip integrated structures. Specifically, the cores' test sets are stored compressed in the ATE and, during testing, are downloaded and decompressed on chip.

The structure of *Intellectual Property* (IP) cores is often hidden from the system integrator. In such cases, cores are only accompanied by a pre-computed test set, and neither Automatic Test Pattern Generation nor fault simulation can be performed for them. Several methods have been proposed for testing IP cores of unknown structure. [2, 13, 20] embed the pre-computed test vectors in long, on-chip generated, pseudorandom sequences, reducing that way the test data volume significantly, with the cost of long test application times. To reduce both test-data volume and test-application time, many methods encode directly the test set using various compression codes like Golomb [3, 4, 26], alternating run-length [5], FDR [6, 23], statistical codes [8, 11, 12, 14], nine-coded-based [29], and combinations of codes [24]. Compression can be performed on the difference vectors instead of the actual test vectors but expensive cyclical shift registers should be incorporated in the system, or the scan chains of other cores must be reused.

Test application time can be further reduced by exploiting the multiple scan chains of cores. Several approaches have been proposed towards this direction, which make use of combinational continuous flow decompressors [21], linear decompressors [1, 17], combinations of linear and non-linear decoders [16, 18, 30], the RESPIN architecture [7], the periodic alterable MUXs architecture [10], adaptive encoding [22], mutation encoding [25], and dictionaries [15, 19, 28, 31, 32]. [27], like [26, 5, 24] for single-scan-chain cores, reduces both test data volume and scan power.

Due to their high efficiency, statistical codes have received increased attention in the literature. Huffman is the most effective statistical code since it provably provides the shortest average codeword length. Its main problem is the high hardware overhead of the required decompressors. To alleviate this problem, selective Huffman coding was proposed in [12], which significantly reduces the decoder size by slightly sacrificing the compression ratio.

In this paper we propose a statistical compression method, which is based on Huffman coding of variable-length test-set blocks, and is suitable for IP cores of unknown structure with multiple scan chains (the applicability to single-scan-chain cores is straightforward). The encoding is performed in a selective manner, i.e., some blocks of the test set are left unencoded. Also, the generated codewords are reusable, in the sense that they can encode compatible blocks of different sizes. Two simple transformations are furthermore presented for improving the statistical properties of the test set before compression. The proposed approach is properly designed so as the hardware of its decompessors to be kept low.

2. Motivation

Let T be the test set of an IP core. T, which is of size $|T|$ (in bits), is partitioned into $|T|/l$ blocks of length l, called hereafter *test set parts* or *data parts*. Each data part contains 0, 1 and 'x' bits, and is compatible with one or more fully specified blocks generated by substituting its 'x' bits with all possible combinations of 0s and 1s. We call these fully specified blocks, *distinct blocks*. According to Selective Huffman coding [12], the m distinct blocks that are compatible with most of the data parts, are Huffman encoded (m is defined by the designer and is much smaller than the volume of all possible distinct blocks). Each data part is either encoded by the codeword generated for a compatible distinct block (if such a distinct block exists) or remains unencoded. In case that a data part is compatible with more than one encoded distinct blocks, the codeword of the most frequently occurring block is used for its encoding.

Assuming that the number of encoded distinct blocks m remains constant, the effectiveness of Selective Huffman coding is affected by block size l in two contradictory ways. As l increases, the test set is partitioned into fewer and larger data parts, and thus fewer codewords are needed for its encoding. On the other hand, the number of data parts that

The research Project is 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.

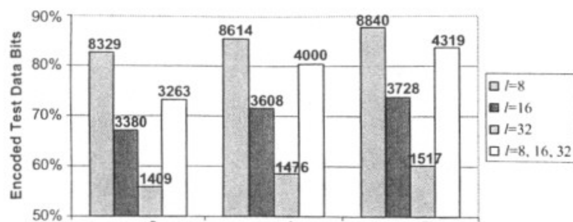

Figure 1. Encoded data statistics for s15850

remain unencoded increases (due to the large block size, fewer data parts are compatible with the m encoded distinct blocks). The block size which balances these two contradictory behaviors provides the best compression ratio.

For maximizing the volume of the encoded test set parts, keeping at the same time the total number of codewords low, variable length blocks can be utilized. For demonstrating this, we performed 4 series of experiments using as input the test set of s15850. In the first 3 series, we compressed the test set utilizing single-length distinct blocks of sizes (l) 8, 16 and 32 (for each experiment series, the encoded-distinct-block volume m was set to 3, 4 and 5). In the last series of experiments, variable-length distinct blocks were encoded. Specifically, the test set was initially partitioned into 32-bit data parts and the most frequently occurring distinct block of size 32 was selected for encoding. Then, the remaining 32-bit data parts (those that are not compatible with the selected distinct block) were partitioned into 16-bit parts for selecting the second distinct block (the most frequently occurring one of size 16). Finally, the remaining 16-bit data parts were partitioned into 8-bit parts for selecting the rest distinct blocks (i.e., for m=3, a 32-bit, a 16-bit and an 8-bit distinct block were encoded, for m=4, a 32-bit, a 16-bit and two 8-bit distinct blocks were encoded, etc.). Figure 1 presents the percentage of test data bits that were encoded at each experiment, as well as the number of codewords used (above each bar). It is obvious that when single-length distinct blocks are used, as l increases the total number of codewords drops rapidly, but so does the percentage of the encoded data bits. When three separate block sizes are used, the percentage of encoded data bits approaches that of the smallest single-length-block case, with less than half the codewords.

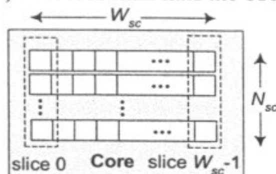

Figure 2. Parallel scan chains and slices

3. Proposed Encoding

3.1 Encoding Method

In the proposed approach, as test set part we consider a whole slice or a slice portion (see Figure 2, where N_{sc} denotes the number of scan chains, and W_{sc} the maximum scan-chain length). Each Huffman codeword encodes a distinct block of a specific size. In the following we present the way slices are partitioned into smaller portions of various sizes, as well as the way that distinct blocks are selected. For determining the proper assignment of the test set's 'x'-bits, so as the occurrence frequencies of the encoded distinct blocks to be as skewed as possible, we use the simple heuristic criteria proposed in [12].

Initially the test set is partitioned into slices according to the scan-chain structure of the core. Each slice is also called a P_0-part of the test set. All P_0-parts are considered for the selection of the first distinct blocks that will be encoded (P_0-blocks), with size equal to the number of scan chains N_{sc}. When a number of P_0-blocks have been selected, each one of the unencoded P_0-parts is partitioned into two portions of equal size, which are called P_1-parts. The size of each P_1-part is equal to $N_{sc}/2$. In case that P_0-parts cannot be partitioned into two equal parts, then half of the P_1-parts are 1-bit shorter than the rest. Again, a number of distinct P_1-blocks are encoded, the unencoded P_1-parts are partitioned into P_2-parts and some P_2-blocks are encoded, the unencoded P_2-parts are partitioned into P_3-parts and so on. Finally, each of the P_0, P_1, ..., P_{max}-blocks corresponds to some P_0, P_1, ..., P_{max}-parts respectively, where the max value is determined by the designer. P_{max}-parts are also called primitive parts, since they are the smallest test set parts (and P_{max}-blocks are the smallest encoded distinct blocks). Note that each P_i-part has size equal to either $\lceil N_{sc}/2^i \rceil$ or $\lfloor N_{sc}/2^i \rfloor$ bits.

According to the above procedure, when a test-set's P_i-part ($i \in [0, max]$, i=0 implies a whole slice) can be encoded (i.e., it is compatible with a distinct P_i-block), then the codeword corresponding to that P_i-block is appended to the compressed test set. If no encoding is possible, the P_i-part is partitioned into two P_{i+1}-parts, which are used for the selection of the P_{i+1}-blocks (and thus they may be encoded by P_{i+1}-block codewords). Although the association of each P_i-block with P_i-parts makes the decoding process easy to implement, it is very limiting. For that reason a slight modification of the encoding procedure is proposed, according to which a P_j-part can be encoded by the codeword of a longer P_i-block ($i < j$), provided that the first $\lceil N_{sc}/2^j \rceil$ or $\lfloor N_{sc}/2^j \rfloor$ bits of the P_i-block are compatible with the P_j-part (the rest are discarded during the decoding process). As an example, consider that each slice is eight bits long (N_{sc} = 8), and that one of the selected P_0-blocks is b = 00111010. The codeword of block b can be used for encoding P_1-part p = 0x11, since p is compatible with the first four bits of b.

In order to keep the compressed-data volume low, no information is stored about the size of the actual P_j-part encoded by the P_i-block codeword. Therefore, for determining the P_j-part univocally, the following condition must be satisfied during the encoding process:

"*a P_i-block codeword can be used for encoding a P_j-part, with j>i, if the P_j-part* ***is not the first*** *of a larger test-set part*"

In other words, each codeword is assumed to encode the largest possible test-set part. That is, if we receive the codeword of block b at the beginning of a slice (N_{sc} = 8), then all of its bits will be used and a whole slice will be generated. If, on the other hand, we receive the same codeword at the beginning of a P_1-part, **which is not the first of a P_0-part** (i.e., after having decoded the first half of a slice), then the first four bits of b will be generated. Similarly, the decoding of b's codeword at the beginning of a P_2-part, **which is not the first of a P_0-part or a P_1-part,** will produce the first two bits

of b. Thus, the same codeword may lead to the generation of different number of test bits by the decompression logic, depending on the stage of the decoding process. Note that the decoded data parts do not contain 'x' bits, since the encoded distinct blocks are fully specified.

More formally, the size of the slice portion that has already been decoded and the size of the distinct block corresponding to the received codeword, determine univocally the size of the next data part to be decoded. Note that each slice can be partitioned into at most 2^{max} primitive parts (P_{max}-parts), and thus the size of a P_0-part (slice) or P_0-block is equal to that of 2^{max} primitive parts, the size of a P_1-block is equal to that of 2^{max-1} primitive parts, etc. Let the size of the slice portion that has been decoded be equal to that of s primitive parts ($0 \leq s < 2^{max}$) and consider a received codeword corresponding to a P_i-block. If s is divided exactly by 2^{max-i} then the largest data part that can be decoded has size equal to that of the P_i-block corresponding to the received codeword. Otherwise, if s is divided exactly by $2^{max-i-1}$, then the size of the largest part that can be decoded is equal to the (upper) half of the P_i-block corresponding to the received codeword. In other words, the smallest positive integer $j \in [i, max]$ is found, such that 2^{max-j} divides s exactly (this is always possible, at least for $j = max$). Note that s is divided exactly by 2^q, when the q least significant bits of s are equal to 0. Therefore the above decoding is very simple to implement in hardware.

Primitive Parts	s	# of 0 LS bits	P_0 (max-i=3)	P_1 (max-i=2)	P_2 (max-i=1)	$P_3 \equiv P_{max}$ (max-i=0)
0	000	3	2^3 / 0-7	2^2 / 0-3	2^1 / 0-1	2^0 / 0
1	001	0	2^0 / 1	2^0 / 1	2^0 / 1	2^0 / 1
2	010	1	2^1 / 2-3	2^1 / 2-3	2^1 / 2-3	2^0 / 2
3	011	0	2^0 / 3	2^0 / 3	2^0 / 3	2^0 / 3
4	100	2	2^2 / 4-7	2^2 / 4-7	2^1 / 4-5	2^0 / 4
5	101	0	2^0/ 5	2^0 / 5	2^0 / 5	2^0 / 5
6	110	1	2^1 / 6-7	2^1 / 6-7	2^1 / 6-7	2^0 / 6
7	111	0	2^0 / 7	2^0 / 7	2^0 / 7	2^0 / 7

8 bits; Slice; max=3; # of decoded P_{max}-parts / actual decoded P_{max}-parts

Figure 3. Example

Example. Consider a circuit with N_{sc}=64 scan chains. Each slice of the test set is 64 bits long and it is partitioned into 2^3=8 primitive parts. Hence max = 3 and each primitive part consists of 8 bits. The test set of the circuit is encoded using P_0-, P_1-, P_2-, and P_3-blocks, of size equal to that of 8, 4, 2 and 1 primitive parts, respectively. In Figure 3 we present a slice partitioned into 8 P_{max}-parts (0-7), as well as the values of s, which indicate the next primitive part of the slice that will be decoded (note that when s=000, no P_{max}-part has been decoded yet). Suppose that during the decoding process, a codeword encoding a P_1-block is received (i=1). If s=0 (000) or s=4 (100), then s is divided exactly by 2^{max-i}=2^2, and therefore the whole P_1-block is loaded in the positions of the next 2^2 primitive parts (0-3 when s=000, or 4-7 when s=100). If s=2 (010) or 6 (110), then s is not divided exactly by 2^{max-i}, but by $2^{max-i-1}$=2^1, and therefore the first half of the P_1-block is loaded in the positions of the next 2^1 primitive parts (2-3 when s=010, or 6-7 when s=110). Finally, when s=001, 011, 101 or 111, s is divided exactly only by $2^{max-i-2}$=2^0, and thus the first quarter of the P_1-block is loaded in the position of the next 2^0 primitive part (1, 3, 5 or 7, when s=001, 011, 101 or 111 respectively). Note that the portion of the P_1-block that will be decoded can be determined by examining the volume of consecutive least significant bits of s that are 0 (if 2 LS bits are 0 then 2^2 P_{max}-parts will be decoded, if 1 LS bit is 0 then 2^1 P_{max}-parts will be decoded, etc). In Figure 3, for every possible combination of s-i, we show the number of P_{max}-parts, as well as the actual P_{max}-parts that are decoded. ■

However, the encoding of, let say, a P_3-part by a P_0-block codeword right from the beginning, may prevent the encoding of a larger data part (a P_1- or P_2-part) that includes the aforementioned P_3-part, in a subsequent step. For avoiding this, every P_{i+1}-part ($i \in [0, max$-1]) is allowed to be encoded by the codeword of a P_0-, P_1-, ..., P_i-block, only at the beginning of the selection process of P_{i+1}-blocks. The selection process of P_i-blocks stops and that of P_{i+1}-blocks starts when the number of bits of the P_i-parts ($TestBits_i$) that are compatible with next P_i-block to be chosen, are fewer by a factor F than the bits of all P_{i+1}-parts ($TestBits_{i+1}$) that can be encoded by the codeword of the first P_{i+1}-block to be selected, as well as by the codewords of the already chosen P_0-, P_1-, ..., P_i-blocks (i.e., when $TestBits_{i+1} \geq F \cdot TestBits_i$). The value of factor F is determined by the designer.

When all P_0, P_1, ..., P_{max}-blocks have been selected (their total number m is defined by the designer), some of the P_{max}-parts remain unencoded. Such parts are labeled as failed and a separate Huffman codeword is assigned to all of them. In the compressed test set, these P_{max}-parts are embedded unencoded, preceded by the aforementioned codeword. The Huffman tree is constructed when all P_i-blocks have been selected, so as all occurrence frequencies to be known. An overview of the proposed algorithm is presented in Figure 4.

1. Set i = 0.
2. Find the next best P_i-block (the one compatible with most P_i-parts) and calculate $TestBits_i$.
3. If i=max encode all possible P_i-parts using the P_i-block and go to 5.
4. Find the best P_{i+1}-block and calculate $TestBits_{i+1}$. If $TestBits_{i+1} \geq F \cdot TestBits_i$, select the P_{i+1}-block, and encode all possible P_{i+1}-parts using the P_{i+1}-block and the already chosen P_0-,P_1-,...,P_i-blocks. Also, set i=i+1. Else, select the P_i-block and encode all possible P_i-parts using the P_i-block.
5. If the number of selected distinct blocks does not exceed a predetermined upper limit, go to Step 2.
6. Label all unencoded P_{max}-parts as failed.
7. Generate the Huffman code and produce the compressed test set.

Figure 4. Proposed encoding algorithm

3.2 Statistical Improvement of Test Data

In this section, we propose two test-set transformations, which can be optionally applied before compressing the test data, for improving their statistical properties (no structural information of the core is required). This is achieved by increasing the difference between 0s and 1s in the test set. Specifically, all the bits of selected scan chains (transformation **T**$_1$) and/or the values of selected scan cells (transformation **T**$_2$) can be inverted. The transformed test set is then compressed and during decompression the original test set is restored by removing on the fly the transformations. For example, suppose that for a test set, the 0-bit volume is higher than the 1-bit volume. According to **T**$_1$, the scan chains with more 1s than 0s (considering all test vectors) can be inverted, for favoring the 0s count. Similarly, **T**$_2$ can be used for inverting a predefined number of scan cells with the highest difference

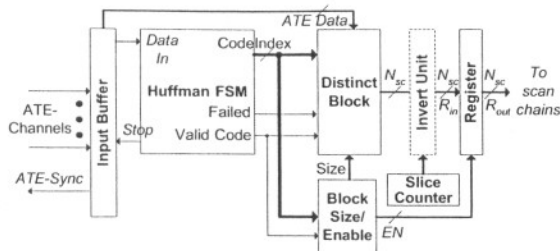

Figure 5. Proposed decompression architecture

of 0-bits from 1-bits for all test vectors. Both transformations improve, in most cases, the compression ratio, whereas they impose very small hardware overhead.

4. Proposed Architecture

The proposed decompression architecture is presented in Figure 5. Input Buffer receives the encoded data in parallel from the ATE and shifts them serially into the Huffman FSM. Upon recognition of a codeword, and assuming that the implemented code consists of N codewords, the Huffman FSM places on the bus *CodeIndex* a binary index (value) between 0 and N-1. This index indicates which codeword has been received. Also, signal *Valid Code* is set to 1, and Input Buffer is disabled until the received codeword is processed (*Stop*=1). In the particular case that the codeword corresponds to an unencoded data part, signal *Failed* is set to 1.

Distinct Block unit receives *CodeIndex* and returns the distinct block (or a portion of it) that corresponds to the received codeword. Specifically, if the encoded distinct block has size equal to that of 2^q primitive parts (this size is provided by the Block Size/Enable unit), then at the outputs of the Distinct Block unit, 2^{max-q} identical copies of the first $2^q \cdot Size(P_{max})$ bits of the encoded distinct block are generated. The enable signals activated by the Block Size/Enable unit ensure that only the proper positions of the Register will be loaded with the generated data. Similarly, in case of a failed P_{max}-part, its bits, which are received directly from the ATE (through the Input Buffer unit), are repeated 2^{max} times at the outputs of the Distinct Block unit, and are loaded in the proper positions of the Register.

According to the proposed approach, even if a codeword corresponds to a P_i-block, it may be utilized for encoding P_{i+1}, P_{i+2}, ..., P_{max}-parts (with sizes equal to that of $2^{max-i-1}$, $2^{max-i-2}$, ..., 2^0 primitive parts). The proper decoded-part size is determined in the Block Size/Enable unit, by a small combinational logic that examines the value (s) of a *max*-bit binary counter (s ranges from 0 to 2^{max}-1). s points to the next primitive part of a slice that has not been decoded yet (see Example in Section III). When the q least significant bits of s are equal to 0, then the largest part that can be decoded is a P_{max-q}-part. If the received codeword encodes a P_i-block and $max-q \leq i$, then the whole P_i-block is decoded. Otherwise, the first bits of the P_i-block that form a P_{max-q}-part are decoded. When s reaches value 2^{max}-1, a whole slice has been loaded in the Register and is then transferred into the scan chains.

When transformations $\mathbf{T_1}$ and/or $\mathbf{T_2}$ are applied to the test set, the Invert unit is placed between the Distinct Block unit and the Register. It consists of at most N_{sc} gates (one for each scan chain), which can be either inverters (for inverting all test data entering a scan chain - $\mathbf{T_1}$) or XOR/ XNOR gates (for selectively inverting specific bits entering a scan chain - $\mathbf{T_2}$). If $\mathbf{T_2}$ is applied, the Invert unit also incorporates a decoding logic.

5. Evaluation & Comparisons

The proposed compression method was implemented in C programming language. As input we used the dynamically compacted test sets generated by Mintest [9] for stuck-at faults. The same test sets were also used in [3-6, 8, 12, 14, 15, 19, 21, 23-26, 28, 29, 31, 32]. The run time of our method is a few seconds for each benchmark circuit.

Table 1. Results of the proposed encoding (# bits)

Circuit	16 Scan Chains			64 Scan Chains			128 Scan Chains		
	No Transf.	T_1 + 50 T_2	T_1 + 100 T_2	No Transf.	T_1 + 50 T_2	T_1 + 100 T_2	No Transf.	T_1 + 50 T_2	T_1 + 100 T_2
s5378	9776	9183	8955	8736	8267	8384	9277	7611	**7611**
s9234	15792	14514	14049	14565	13265	13164	14493	12765	**12765**
s13207	24039	23053	21623	17911	16748	16366	16193	15092	**14578**
s15850	22113	20685	20016	19570	18161	17595	18611	17354	**16562**
s38417	**58663**	61241	59748	59411	64262	63165	59492	64963	63833
s38584	60291	60101	59809	56890	57309	56854	55612	55965	**55353**

In Table 1 we present the test-data compression results (# bits) of the proposed method for 16, 64 and 128 scan chains and 24 selected distinct variable-length blocks (the bit volumes of the original Mintest test sets can be found in Table 2). The size of primitive parts (P_{max}) was in all cases equal to 8 bits. Three different compression cases are presented for each scan-chain volume: a) compression without any transformations ("No Transf."), b) compression with transformations $\mathbf{T_1}$ and $\mathbf{T_2}$ with 50 selected cells ("$\mathbf{T_1}$ + 50 $\mathbf{T_2}$"), and c) compression with transformations $\mathbf{T_1}$ and $\mathbf{T_2}$ with 100 selected cells ("$\mathbf{T_1}$ + 100 $\mathbf{T_2}$"). In almost all cases the compression improves as the number of scan chains increases. Also, compared to the "no transformations" case, we get better compression when $\mathbf{T_1}$ + 50 $\mathbf{T_2}$ are applied, whereas further increase in the number of cells inverted by $\mathbf{T_2}$ does not provide significant improvements. Since the hardware overhead imposed by $\mathbf{T_1}$ is negligible, and that of $\mathbf{T_2}$ depends on the number of selected cells, the total hardware overhead of the Invert unit is limited (50-100 inverted cells are relatively few).

At first we compare the proposed approach against methods that also exploit the multiple scan chains of a core in order to reduce test application time. In Table 2, comparisons against Mintest, selective Huffman [12] (re-implemented here for 64 scan chains), [21] and [25] are presented. We do not compare the proposed method against [7], since several conditions have to be satisfied by a core nearby the CUT, so as the former to be used as decompressor. Also, no comparisons are provided against approaches that use different test sets from those used in our experiments [1, 10, 16-18, 22, 27, 30]. In column 2, the sizes of the original Mintest test sets are provided. For the selective Huffman approach, the number of selected distinct blocks was set to 24 and the block size to 8, 16 and 32. The best result for each core is reported in column 3 of Table 2. In columns 4-5 the best results of [21] and [25] are presented. We note that in case of [25], apart from the test data reported in column 4, an additional significant amount of control data is required to be stored in the ATE

Table 2. Comparisons against compression methods for cores with multiple scan chains

Circuit	Mintest (# bits)	Select. Huff. (# bits)	[21] (# bits)	[25] (# bits)	Proposed (# bits)		Reduction %* (No Transf.) over:				Reduction %* (T_1 + 100 T_2) over:			
					No Transf.	T_1 + 100 T_2	Min-test	Select. Huff.	[21]	[25]	Min-test	Select. Huff.	[21]	[25]
s5378	23754	11433	14220	-	8736	**7611**	63.2	23.6	38.6	-	68.0	33.4	46.5	-
s9234	39273	19168	30144	-	14493	**12765**	63.1	24.4	51.9	-	67.5	33.4	57.7	-
s13207	165200	28328	20988	74423	16193	**14578**	90.2	42.8	22.8	78.2	91.2	48.5	30.5	80.4
s15850	76986	26873	25140	26021	18611	**16562**	75.8	30.7	26.0	28.5	78.5	38.4	34.1	36.4
s38417	164736	70954	85225	**45003**	58663	59748	64.4	17.3	31.2	-30.4	63.7	15.8	29.9	-32.8
s38584	199104	71315	57120	73464	55612	**55353**	72.1	22.0	2.6	24.3	72.2	22.4	3.1	24.7

*Reduction % = [1 - (# bits of proposed / # bits of: Mintest, Select. Huff., [21], [25])] · 100

and then transferred to the core. Although these additional data increase the volume of stored bits, they cannot be taken into account in Table 2, since they have not been reported by the authors of [25]. In columns 6-7 the best results of the proposed method are provided when transformations $\mathbf{T_1}$ and $\mathbf{T_2}$: a) are not applied and b) are both applied ($\mathbf{T_2}$ for 100 selected cells). In columns 8-11 (12-15) the reduction percentages of the proposed method without (with) test-set transformations over the rest methods are reported. As we can see, in all but one case (s38417 of [25], for which no control data have been reported) the proposed technique performs better than the others, with or without transformations.

Table 3. Compr. ratios of the prop. and dictionary-based methods

Circuit	Prop.	[15]	[19]	[28]	[29]	[31]	[32]
s5378	63.2	-	73.3	90.0	63.2	61.0	51.2
s9234	63.1	70.3	70.7	88.8	61.0	55.0	51.9
s13207	90.2	81.7	94.8	95.4	89.2	81.5	80.8
s15850	75.8	76.3	82.0	92.6	75.9	79.0	64.0
s38417	64.4	70.6	61.8	70.5	71.4	61.6	48.5
s38584	72.1	75.1	73.2	87.9	73.8	60.0	67.2

In Table 3 we provide the compression ratios of the proposed and dictionary-based methods that are applicable to multiple-scan-chain cores (Compression ratio (%) = [1 - (Compressed bits / Mintest bits)] · 100). In some cases, our method performs better than the rest, whereas in other cases it does not. However, as we will show later, the hardware overhead of these methods is prohibitively large.

Table 4. Prop. vs. methods for single-scan-chain cores (reduct. %)

Circuit	[3]	[4]	[5]	[6]	[8]	[12]	[14]	[23]	[24]	[26]	[29]
s5378	-	-	34.9	38.4	33.5	28.6	18.7	33.3	30.7	47.9	27.6
s9234	42.6	43.3	40.9	42.4	38.4	29.0	17.7	39.9	38.0	46.7	28.1
s13207	65.0	58.5	55.3	52.8	46.5	61.6	20.7	51.4	49.5	61.6	40.4
s15850	59.3	45.8	37.0	36.3	32.9	36.7	12.5	32.8	34.1	47.1	25.1
s38417	36.3	35.6	9.7	37.2	23.6	13.1	0.2	9.7	0.6	20.1	4.0
s38584	46.8	38.4	28.5	28.9	26.3	22.6	-0.3	25.0	26.1	35.9	12.0

In Table 4 we present the compressed-data reduction percentages of the proposed method against techniques applicable to cores with a single scan chain. The compression achieved by the proposed approach is higher than that of the rest methods, except for the s38584 case of [14]. Note that all these methods require long test application times due to their inability to exploit the parallelism of the scan chains.

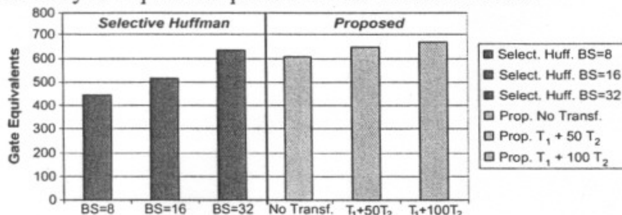

Figure 6. Hardware overhead for 24 selected distinct blocks

For assessing the hardware overhead of the proposed method, we synthesized three different decompressors for the test set of s9234, by applying: a) no transformations, b) $\mathbf{T_1}$ and $\mathbf{T_2}$ for 50 selected cells and c) $\mathbf{T_1}$ and $\mathbf{T_2}$ for 100 selected cells. We also synthesized the decoder of the implemented parallel selective Huffman approach, with (single) block size (BS) equal to 8, 16 and 32. In all experiments, the number of scan chains was set to 64, and the number of selected distinct blocks to 24 (note that the decompressor size does not depend on the compressed test set but on architectural parameters like the number of scan chains, the number of selected distinct blocks etc). The results are shown in Figure 6 in gate equivalents (a gate equivalent corresponds to a 2-input NAND gate). Observe that the transformations marginally increase the hardware overhead of the proposed method. Compared to the selective Huffman approach, the proposed one imposes slightly higher hardware overhead. As far as the approaches of [21] and [25] are concerned, their hardware overhead is low, but, as shown earlier, their compression ratios are much smaller than those of the proposed method.

Compared to the dictionary-based methods the hardware overhead of the proposed approach is very small. The size of the required dictionary in [29] is in the range of 1,187 to 9,152 bits, which is comparable to that of [19] and smaller than those of [15] and [31]. The size of the dictionary in [32] is in the range of 6,264 to 43,712 bits, while that of [28] is equal to 25,600 bits. Apart from the dictionaries, extra area will be also occupied by the decompressors of the above techniques. Therefore, we conclude that dictionary-based methods impose prohibitively large hardware overhead.

The hardware overhead of the single-scan-chain methods, in gate equivalents, is: 125-307 for [3] (as reported in [8]), 320 for [5], 136-296 for [8], 203-432 for [14], 551-769 for the main part of the decompressor in [24] and 416 for [29]. The hardware overhead of [12] is greater than that of [8]. Although cheaper in hardware, these techniques require much longer test application times and much more data to be stored in the ATE, compared to the proposed approach.

Table 5. Average reduction percentages of test application time

r	no Transf. – avg. red. % over:			T_1+100 T_2 – avg. red. % over:		
	Single Scan Chain		Mintest with Multiple Scan Chains	Single Scan Chain		Mintest with Multiple Scan Chains
	Prop.*	[12]		Prop.*	[12]	
2	72.4	70.8	38.2	73.9	72.8	42.4
10	54.8	57.6	62.4	56.7	60.6	65.1

*: Single-Scan-Chain version of the proposed method

As far as the test application time (TAT) is concerned, the average reduction-percentages (per circuit) of the proposed method, assuming 5 ATE channels, are shown in Table 5 (we

considered the best cases of Table 1). We present results for $r = 2$ and $r = 10$, where $r = f_{SYS} / f_{ATE}$. In order to illustrate the advantage of exploiting the multiple scan chains of a core we compare against: a) the single-scan-chain version of the proposed method ("Prop.*" columns) and b) the approach of [12], which is applicable to cores with a single scan chain ("[12]" columns). In both cases the TAT gain is significant. Note that the TAT gain depends on many parameters, like the number of utilized ATE channels, the P_{max}-parts' size, r, etc. The TAT benefits from the application of the proposed compression method are illustrated in columns "Mintest with Multiple Scan Chains", where we compare the proposed technique against the parallel-loading case of the original, uncompressed test sets (with 5 ATE channels).

6. Conclusions

In this paper we proposed an efficient compression method, suitable for multiple-scan-chain IP cores of unknown structure. Huffman is used as a variable-to-variable code for compressing variable length blocks. For increasing the compression ratio, codeword reusability as well as two transformations that improve the statistical properties of the original test set, were also introduced. A simple and low-overhead architecture was finally proposed for performing the decompression.

7. References

[1] K. Balakrishnan et al., "PIDISC: Pattern Independent Design Independent Seed Compression Technique", in *Proc. 19th VLSID*, 2006, pp. 811-817.

[2] K. Chakrabarty et al., "Deterministic Built-In Test Pattern Generation for High-Performance Circuits using Twisted-Ring Counters", *IEEE Trans. VLSI Systems*, vol. 8, pp. 633-636, Oct. 2000.

[3] A. Chandra and K. Chakrabarty, "System-on-a-Chip Test-Data Compression and Decompression Architectures Based on Golomb Codes", *IEEE Trans. CAD*, vol. 20, pp. 355-368, March 2001.

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

[5] A. Chandra and K. Chakrabarty, "A Unified Approach to Reduce SOC Test Data Volume, Scan Power and Testing Time", *IEEE Trans. CAD*, vol. 22, pp. 352-363, March 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. Computers*, vol. 52, pp. 1076-1088, Aug. 2003.

[7] R. Dorsch and H. -J. Wunderlich, "Tailoring ATPG for Embedded Testing", in *Proc. ITC*, 2001, pp. 530-537.

[8] P. Gonciari et al., "Variable-Length Input Huffman Coding for System-On-A-Chip Test", *IEEE Trans. CAD*, vol. 22, pp. 783-796, June 2003.

[9] I. Hamzaoglu and J. Patel, "Test Set Compaction Algorithms for Combinational Circuits", *IEEE Trans. CAD*, vol. 19, pp. 957-963, Aug. 2000.

[10] Y. Han et al., "Scan Data Volume Reduction Using Periodically Alterable MUXs Decompressor", *in Proc. 14th ATS*, 2005, pp. 372-377.

[11] V. Iyengar et. al., "Deterministic Built-In Pattern Generation for Sequential Circuits", *J. Electron. Test.: Theory Applicat.*, vol. 15, pp. 97-114, Aug.-Oct. 1999.

[12] A. Jas et al., "An Efficient Test Vector Compression Scheme Using Selective Huffman Coding", *IEEE Trans. CAD*, vol. 22, pp. 797-806, June 2003.

[13] 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. Pap. Dig. IEEE ETS*, 2005, pp. 147-150.

[14] X. Kavousianos et al., "Efficient Test-Data Compression for IP cores Using Multilevel Huffman Coding", in *Proc. DATE* 2006, pp. 1033-1038.

[15] M.J. Knieser et al., "A Technique for High Ratio LZW Compression", in *Proc. DATE* 2003, pp.116-121.

[16] C. Krishna and N. Touba, "Reducing Test Data Volume Using LFSR Reseeding with Seed Compression", in *Proc. ITC*, 2001, pp. 321-330.

[17] C. Krishna and N. Touba, "Adjustable Width Linear Combinational Scan Vector Decompression", in *Proc. IEEE/ACM ICCAD*, 2003, pp. 863-866.

[18] J. Lee and N. Touba, "Combining Linear and Non-Linear Test Vector Compression Using Correlation-Based Rectangular Encoding", in *Proc. 24th VTS*, 2006, pp. 252-257.

[19] L. Li, K. Chakrabarty and N. A. Touba, "Test Data Compression Using Dictionaries with Fixed-Length Indices", in *Proc. 21th VTS*, 2003, pp. 219-224.

[20] L. Li and K. Chakrabarty, "Test Set Embedding for Deterministic BIST Using a Reconfigurable Interconnection Network", *IEEE Trans. CAD*, vol.23, pp. 1289-1305, Sept. 2004.

[21] L. Li et al., "Efficient space/time compression to reduce test data volume and testing time for IP cores", in *Proc. 18th Int. Conf. on VLSI Des.*, 2005, pp. 53-58.

[22] S. Lin et al., "Adaptive Encoding Scheme For Test Volume/Time Reduction in SOC Scan Testing", in *Proc. 14th ATS*, 2005, pp. 324-329.

[23] 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. ICECS*, 2002, vol. 2, pp. 449-452.

[24] M. Nourani and M. H. Tehranipour, "RL-huffman Encoding for Test Compression and Power Reduction in Scan Applications", *ACM Trans. Des. Autom. of Electr. Syst.*, vol. 10, pp. 91-115, Jan. 2005.

[25] S. Reda and A. Orailoglu, "Reducing Test Application Time Through Test Data Mutation Encoding", in *Proc. DATE* 2002, pp. 387-393.

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

[27] Y. Shi et al., "Low Power Test Compression Technique for Designs with Multiple Scan Chains", in *Proc. 14th ATS*, 2005, pp. 386-389.

[28] X. Sun et al., "Combining Dictionary and LFSR Reseeding for Test Data Compression", in *Proc. DAC*, 2004, pp. 944-947.

[29] M. Tehranipour et al., "Nine-Coded Compression Technique for Testing Embedded Cores in SoCs", *IEEE Trans. VLSI Systems*, vol. 13, pp. 719-731, June 2005.

[30] S. Ward et al., "Using Statistical Transformations to Improve Compression for Linear Decompressors", in *Proc. IEEE Int. Symp. on DFT*, 2005, pp. 42-50.

[31] F.G. Wolff and C. Papachristou, "Multiscan-Based Test Compression and Hardware Decompression Using LZ77", in *Proc. ITC*, 2002, pp. 331-339.

[32] A. Wurtenberger et al., "Data Compression for Multiple Scan Chains using Dictionaries with Corrections", in *Proc. ITC*, 2004, pp. 926-935.