Brief Contributions

Optimal Selective Huffman Coding for Test-Data Compression

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

Abstract—Selective Huffman coding has recently been proposed for efficient test-data compression with low hardware overhead. In this paper, we show that the already proposed encoding scheme is not optimal and we present a new one, proving that it is optimal. Moreover, we compare the two encodings theoretically and we derive a set of conditions which show that, in practical cases, the proposed encoding always offers better compression. In terms of hardware overhead, the new scheme is at least as low-demanding as the old one. The increased compression efficiency, the resulting test-time savings, and the low hardware overhead of the proposed method are also verified experimentally.

Index Terms—Embedded testing techniques, IP cores, selective Huffman coding, test-data compression.

1 Introduction

CORE-BASED design, i.e., the use of predesigned and preverified modules (cores) in modern Systems-on-a-Chip (SoCs), has become the dominant design strategy in the industry since it ensures the economic viability of electronic products. Although the utilization of cores shortens the design time of SoCs, the increased complexity of the latter introduces new challenges in the testing process. During testing, a large amount of data should be stored on the tester (Automatic Test Equipment—ATE), and then transferred at high rates deep into the chip. However, the limited channel capacity, memory, and speed of ATEs, as well as the reduced accessibility of some of the inner nodes of dense SoCs, make testing the bottleneck of the production cycle and, thus, the benefits from shortening the SoCs' design time are compromised.

For easing the burden of testing on ATEs and for providing the required accessibility to cores deeply embedded in a SoC, the ATE capabilities are combined with on-chip integrated test structures in an embedded testing framework. The test set of each core of an SoC is stored in the ATE's memory in a compressed form and, during testing, it is downloaded into the chip, where it is decompressed and applied to the core. For cores of known structure, Automatic Test Pattern Generation (ATPG) and fault simulation are utilized in order to minimize the data stored on ATE, as in SmartBIST [11], Embedded Deterministic Test [15], and DBIST [19]. However, this is not possible for Intellectual Property (IP) cores of unknown structure, the test sets of which are provided precomputed by the core vendor. In such cases, direct test set encoding is performed most of the time since it combines minimum test application time with high compression ratios. Various compression methods have been proposed toward this direction which use, among others, Golomb codes [1], alternating run-length codes [2], FDR codes [3], [13], statistical codes [4], [7], [8], [9], [10], a nine-coded technique [17], selective encoding of scan slices [18], and combinations of codes [12], [14], [16]. Due to their high efficiency, statistical codes have received increased attention and have been widely used in test data compression techniques. Their efficiency is justified by the inherent correlation of the test vectors belonging in the cores' test sets and by the existence of unknown ("x") values within them. Huffman is the most effective statistical code because it provably provides the shortest average codeword length among all uniquely decodable variable length codes [6]. The main problem of Huffman coding is the high hardware overhead of the required decompressors. To alleviate this problem, Selective Huffman coding was proposed in [8] which significantly reduces the decoder size by slightly sacrificing the compression ratio. Selective Huffman is a very attractive approach whose low hardware overhead allows the exploitation of the compression advantages of Huffman coding in the embedded testing framework.

In this paper, we show that the Selective Huffman approach of [8] is not optimal and we propose a new one which maximizes the compression ratio with similar or often smaller decompressor hardware overhead. We theoretically prove that the proposed Selective Huffman coding is optimal and we investigate under which conditions it provides better compression than that of [8] (otherwise, the achieved compression ratios are equal). Our analysis shows that, in practical cases, the proposed approach performs better than that of [8], a fact that is also verified experimentally.

The rest of the paper is organized as follows: Section 2 reviews the Selective Huffman encoding of [8], while Section 3 describes the proposed one and theoretically proves its optimality. A thorough theoretical analysis-comparison of the proposed approach and that of [8] is also provided. In Section 4, the effectiveness of the new encoding is evaluated with experimental results and comparisons, while Section 5 concludes the paper.

2 Selective Huffman Encoding Review

Statistical codes belong in the category of fixed-to-variable codes since they represent data blocks of fixed length using variable length codewords. Compression is achieved by encoding the most frequently occurring blocks with short codewords and the less frequently occurring ones with long codewords. Therefore, the efficiency of a statistical code depends on the frequency of occurrence of all distinct fixed-length blocks in a set of data. As mentioned above, Huffman is the statistical code that provably provides the shortest average codeword length (it is closer than that of any other statistical code to the theoretical entropy bound) and, thus, offers the best compression. Moreover, Huffman code is prefix-free (i.e., no codeword is the prefix of another one) and, therefore, its decoding process is simple and easy to implement.

Let T be the test set of an IP core. If we partition the test vectors of T into $|T|/l$ data blocks of length l ($|T|$ is the test set's size in bits), we get k distinct blocks, $b_1, b_2, \ldots, b_k$, with frequencies (probabilities) of occurrence $p_1 \geq p_2 \geq \ldots \geq p_k$, respectively. For example, if we partition T into 4-bit blocks ($l = 4$), then all possible distinct blocks will be $2^l = 2^4 = 16$ $(0000, 0001, \ldots, 1111)$. Each of the $|T|/l$ data blocks is equal to one of the distinct blocks. However, after T's partitioning, some of the 2^l distinct blocks may not appear (e.g., no data block is equal to distinct block 1011) and, as a result, $k \leq 2^l$. Hereafter, we will refer to blocks $b_1, b_2, \ldots, b_k$ with the term

- *X. Kavousianos is with the Computer Science Department, University of Ioannina, 45110 Ioannina, Greece. E-mail: kabousia@cs.uoi.gr.*
- *E. Kalligeros is with the Computer Science Department, University of Ioannina, 45110 Ioannina, Greece, and with the Computer Science and Technology Department, University of Peloponnese, Terma Karaiskaki, 22100 Tripoli, Greece. E-mail: kalliger@ceid.upatras.gr.*
- *D. Nikolos is with the Computer Engineering and Informatics Department, University of Patras, 26500 Patras, Greece. E-mail: nikolosd@cti.gr.*

Manuscript received 6 Sept. 2006; revised 19 Jan. 2007; accepted 21 Mar. 2007; published online 20 Apr. 2007.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0384-0906.
Digital Object Identifier no. 10.1109/TC.2007.1057.

TABLE 1
Test Data Partitioning and Occurrence Frequencies

Test Set T	Distinct Blocks	Occur. Freq.
1010 0000 1010 1111	1010	9/20
1111 0000 1010 0001	0000	5/20
1010 0000 0010 1010	1111	3/20
0000 1010 1010 0000	0001	2/20
1010 1111 1010 0001	0010	1/20

distinct blocks in order to explicitly distinguish them from the $|T|/l$ *data blocks* that result from the partitioning of test set T.

In order to generate a Huffman code, a binary tree is constructed, beginning with the leaves and moving toward the root. For every distinct block b_i, a leaf node is generated, and a weight equal to p_i is assigned to it (p_i is the occurrence frequency of block b_i). The pair of nodes with the smallest weights is selected first and a parent node is generated with weight equal to the sum of the weights of these two nodes. This step is repeated iteratively until only a single node is left unselected, the root (we note that each node can be selected only once). Then, starting from the root, all nodes are visited once and the logic "0" ("1") value is assigned to each left (right)-child edge. The codeword of distinct block b_i is the sequence of the logic values of the edges belonging to the path from the root to the leaf node corresponding to b_i.

As already mentioned, the major problem that prevents us from using Huffman coding for embedded testing is the large hardware overhead of the required decompressors. Specifically, a linear increase in the block size l causes an exponential increase in the distinct-block volume k and, consequently, in the number of codewords that should be decoded. For that reason, the approach of [8] encodes only a few of the distinct blocks. Specifically, among the k distinct blocks $b_1, b_2, \ldots, b_k$ of the test set, only the m most frequently occurring ones, $b_1, b_2, \ldots, b_m$ $(m < k)$, are Huffman encoded, while the rest, $b_{m+1}, \ldots, b_k$, remain unencoded (i.e., the compressed test set contains some unencoded data blocks). For distinguishing between a codeword and an unencoded data block in the compressed test set, an extra bit is used before either of them (each unencoded data block is preceded by the "0" bit, while each codeword is preceded by the "1" bit). This way, a new prefix-free code, called Selective Huffman, is generated.

Example 1. Consider the test set shown in column 1 of Table 1, which consists of five test vectors of length 16 (80 bits overall). By partitioning the test vectors into 4-bit blocks, we get 20 occurrences of five distinct blocks ($k = 5$) shown in Column 2. Column 3 presents the occurrence frequency of each distinct block. Suppose that the test set is compressed by using Selective Huffman coding with $m = 3$ encoded distinct blocks. Then, the first three distinct blocks, namely 1010, 0000, and 1111, are encoded as shown in Fig. 1, while blocks 0001 and 0010 remain unencoded. During the construction of the compressed test set, an extra bit (shown underlined in Fig. 1) is appended before either each codeword (the "1" bit) or each unencoded data block (the "0" bit). The compressed data volume is equal to 57 bits (compression ratio = 28.8%).

Usually, test sets include many unspecified ("x") values which are exploited in order to maximize the compression ratio. In [8], two procedures have been proposed which replace these "x" values with 0s or 1s in order to maximize the skewing of the occurrence frequencies of the distinct blocks that will be encoded. The great number of "x" values even in compacted test sets, as well as the inherent correlation of test vectors, guarantee the high efficiency of Selective Huffman coding since, even if a few distinct blocks are selected for encoding, they will be compatible with the vast majority of the data blocks (and, thus, these data blocks will be compressed).

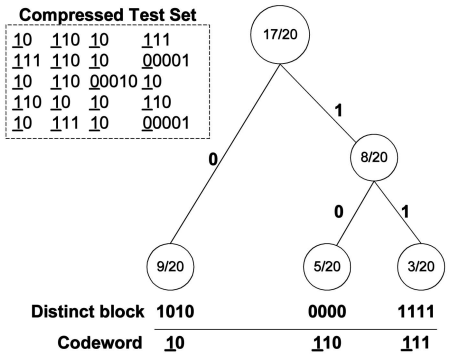


Fig. 1. The Huffman tree of Example 1 and Selective Huffman encoding according to [8].

3 Proposed Selective Huffman Encoding

The inefficiency of the Selective Huffman encoding of [8] stems from the fact that the required extra bit equally lengthens all data in the compressed test set (encoded or not), irrespective of their occurrence frequency. This constant overhead, although minimum for the unencoded data blocks, can be relatively high for the most frequently occurring codewords. A better approach would be to use an additional Huffman codeword in front of only the unencoded data blocks, relieving in this way the most frequently occurring codewords from the extra-bit overhead. Thus, the gain from shortening the codewords that appear very often in the compressed test set will overbalance the loss from using a whole codeword, instead of a single bit, in front of the unencoded data blocks (the unencoded data blocks are usually a small fraction of the compressed test set). Specifically, according to the proposed Selective Huffman approach, $m + 1$ Huffman codewords are used, m for encoding the m most frequently occurring distinct blocks and 1 for indicating the unencoded blocks. The occurrence frequency of the latter codeword is equal to the sum of the occurrence frequencies of all the unencoded distinct blocks.

Example 2. Consider the test set of Table 1 and that $m = 3$, that is, 0001 and 0010 are, as in Example 1, the unencoded blocks. The sum of the occurrence frequencies of 0001 and 0010 is equal to 2/20 + 1/20 = 3/20. The proposed Selective Huffman encoding as well as the compressed test set are given in Fig. 2. We observe that now, for encoding distinct blocks 1010 and 0000, only 1 and 2 bits are, respectively, required (instead of 2 and 3 in Example 1), while the cost for distinct block 1111 remains

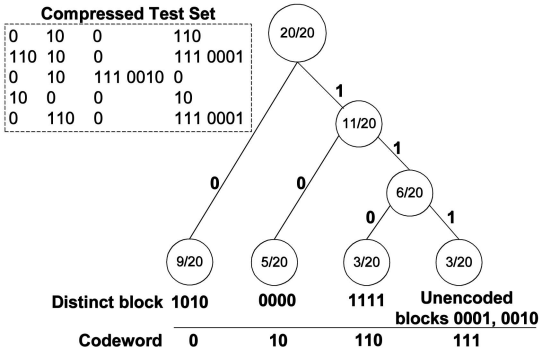


Fig. 2. Proposed Selective Huffman encoding.

constant (3 bits). Only before the unencoded data blocks, a 3-bit codeword, instead of a single bit, is utilized. The compression ratio in this case is equal to 38.8 percent, which is significantly higher than that of Example 1.

Note that both approaches encode the same distinct blocks, i.e., the same part of the original test set, since the additional Huffman codeword of the proposed method is not used for distinct-block encoding but plays the same role as the extra "0" bit in the approach of [8] (it indicates the unencoded blocks). Equivalently, the extra "0" bit of the method of [8] can be regarded as the codeword that precedes the unencoded data blocks.

Example 2 also serves as a counterexample that proves the nonoptimality of the Selective Huffman approach of [8]. Since, for the presented case, the proposed encoding offers better compression (for the same number of encoded distinct blocks), then the code of [8] is not an optimal Selective Huffman code.

Let T be the test set of an IP core and assume that $p_1, p_2, \ldots, p_m$ are the occurrence frequencies of the m distinct blocks $b_1, b_2, \ldots, b_m$ that will be encoded and that $p_u = p_{m+1} + p_{m+2} + \ldots + p_k$ is the sum of the occurrence frequencies of all the k-m unencoded distinct blocks $b_{m+1}, b_{m+2}, \ldots, b_k$ ($p_1 \geq p_2 \geq \ldots \geq p_k$).

Theorem 1. *The proposed Selective Huffman encoding is optimal.*

Proof. The average block length of the proposed encoding, which is the average length of the codewords and the unencoded data blocks in the compressed test set, is given by the relation

$$w_{pr} = l_1 \cdot p_1 + l_2 \cdot p_2 + \ldots + l_m \cdot p_m + (l_u + l_b) \cdot p_u$$

or, equivalently,

$$w_{pr} = [l_1 \cdot p_1 + l_2 \cdot p_2 + \ldots + l_m \cdot p_m + l_u \cdot p_u] + l_b \cdot p_u, \qquad (1)$$

where l_i, $i \in [1, m]$, is the length of the codeword of each encoded distinct block b_i ($i \in [1, m]$), l_u is the length of the codeword corresponding to the unencoded data blocks, and l_b is the length of each unencoded data block. We note that all codewords of the proposed encoding are Huffman codewords.

Assume now that, for the same test set T, there is another Selective Huffman code, which encodes the same distinct blocks b_i, $i \in [1, m]$, and leads to smaller average block length w than that of the proposed one. That is,

$$w = l'_1 \cdot p_1 + l'_2 \cdot p_2 + \ldots + l'_m \cdot p_m + (l'_u + l_b) \cdot p_u$$

or, equivalently,

$$w = [l'_1 \cdot p_1 + l'_2 \cdot p_2 + \ldots + l'_m \cdot p_m + l'_u \cdot p_u] + l_b \cdot p_u \qquad (2)$$

with $w < w_{pr}$. We observe that the term $l_b \cdot p_u$ is common in relations (1) and (2) and that the term in the brackets in (1) gives the average length of $m+1$ Huffman codewords with occurrence frequencies $p_1, p_2, \ldots, p_m$ and p_u, while the term in the brackets in (2) gives the average length of the $m+1$ codewords of a uniquely decodable variable length code with the same occurrence frequencies as in (1). Then, the assumption that $w < w_{pr}$ leads to the erroneous conclusion that a uniquely decodable variable length code exists, which provides shorter average codeword length than that of a Huffman code. Therefore, $w_{pr} \leq w$, which means that the proposed Selective Huffman encoding is optimal. □

Corollary 1. *Assuming that w_J is the average block length of the encoding proposed in [8], then $w_{pr} \leq w_J$.*

We will now investigate under which conditions the proposed encoding performs better than that of [8] (i.e., when $w_{pr} < w_J$). The code proposed in [8] is equivalent to the one of the Huffman-like tree shown in Fig. 3 (Q-tree). This tree is constructed as follows: The root's left child (N_u) is the node corresponding to all unencoded distinct blocks, while the root's right child is the Huffman tree generated for encoding the m most frequently occurring distinct blocks. Hence, as explained above, the "0" bit is the codeword preceding every unencoded data block (N_u), while every distinct-block codeword is preceded by the "1" bit (subtree S).

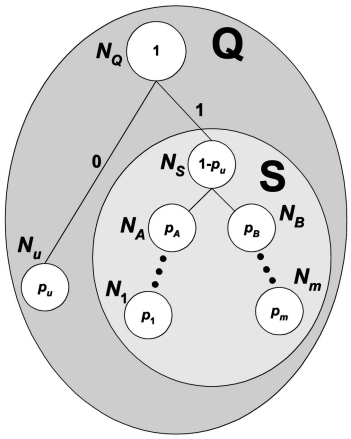


Fig. 3. Equivalent tree for the Selective Huffman encoding proposed in [8].

Let N_A, N_B be the children of node N_s (root of subtree S in Fig. 3), with weights p_A, p_B. Note that

$$p_A + p_B = 1 - p_u = p_1 + p_2 + \ldots + p_m.$$

The assumption that N_S has two children is reasonable since, nearly always, at least two distinct blocks are encoded (if only one distinct block was encoded, a case that is rather impractical, the two examined coding schemes would be equivalent).

Lemma 1. *The proposed encoding and that of [8] are equivalent if and only if $p_u \geq p_A$ and $p_u \geq p_B$. In any other case, the proposed encoding is more efficient than that of [8].*

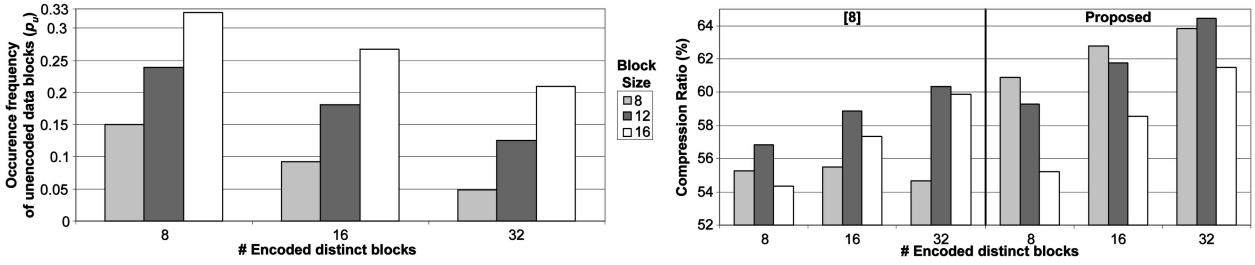
Proof. If $p_u \geq p_A$ and $p_u \geq p_B$, then, for constructing a Huffman tree, among nodes N_A, N_B, and N_u, nodes N_A and N_B would be selected as children of node N_s since they have the smallest weights. Therefore, in this case, the Q tree of Fig. 3 is a Huffman tree and, thus, the two encodings are equivalent.

When at least one of the inequalities $p_u \geq p_A$ and $p_u \geq p_B$ is not valid, we have that $p_u < p_A$ and/or $p_u < p_B$. In order to prove that the proposed encoding is more efficient than that of [8], it suffices to prove that, if $p_u < p_A$ or $p_u < p_B$, the Q tree of Fig. 3 is not a Huffman tree. Assume that $p_u < p_A$. Then, among the pairs of nodes (N_A, N_u), (N_A, N_B), and (N_B, N_u), pair (N_A, N_B) does not have the smallest sum of weights since $p_u + p_B < p_A + p_B$. Hence, either node-pair (N_A, N_u) or (N_B, N_u) has the smallest sum of weights and, therefore, in a Huffman tree, node N_u would not be a child of the root, but it would belong to the S subtree. Consequently, the Q tree is not a Huffman tree. For $p_u < p_B$ the proof is similar. □

The above lemma will help us determine the range of p_u values for which the proposed encoding is more efficient than that of [8].

Theorem 2. *If $p_u \geq 1/2$, then the proposed encoding and that of [8] are equivalent.*

Proof. We have that $p_A + p_B + p_u = 1$ or, equivalently, $p_u = 1 - p_A - p_B$. Taking into account that $p_u \geq 1/2$, we get $p_A + p_B \leq 1/2$. Therefore, $p_A \leq 1/2$ and $p_B \leq 1/2$ and,



(a) (b)

Fig. 4. (a) Occurrence frequencies of unencoded data blocks (p_u) and (b) compression ratios for s38417.

consequently, $p_u \geq p_A$ and $p_u \geq p_B$. Then, according to Lemma 1, the two encodings are equivalent. □

Theorem 3. *If $p_u \in [0, 1/3)$, then the proposed encoding is more efficient than that of [8].*

Proof. Since $p_u = 1 - p_A - p_B$ and $p_u < 1/3$, then $p_A + p_B > 2/3$. Therefore, at least one of the inequalities $p_A \geq 1/3$ and $p_B \geq 1/3$ is true and, consequently, $p_u < p_A$ or $p_u < p_B$. Then, according to Lemma 1, the proposed encoding is more efficient than that of [8]. □

It remains to examine the case that $p_u \in [1/3, 1/2)$. Since, as already mentioned, at least two distinct blocks are encoded, we have that $p_A > 0$ and $p_B > 0$. Replacing p_B with $1 - p_u - p_A$ in the latter inequality, we get $p_A < 1 - p_u$. Hence,

$$0 < p_A < 1 - p_u. \tag{3}$$

According to Lemma 1, the proposed encoding and that of [8] are equivalent when $p_u \geq p_A$ and $p_u \geq p_B$. Again, by replacing p_B with $1 - p_u - p_A$, we get $p_A \geq 1 - 2p_u$ and, thus,

$$1 - 2p_u \leq p_A \leq p_u. \tag{4}$$

Inequality (4) determines the range of p_A values for which the two compared encodings are equivalent. For that reason we will call this value range of p_A the *equivalence range*. The wider that range is, the greater the probability is that the Q tree is a Huffman tree (since (4) is satisfied for more values of p_A). In the interval of interest, i.e., $p_u \in [1/3, 1/2)$, when $p_u \to 1/3$, the p_A equivalence range shrinks and tends to become equal to $[1/3, 1/3]$. Since the Q tree is a Huffman tree for fewer values of p_A, the probability that the two encodings are equivalent reduces (for $p_u = 1/3$, equivalence exists only if $p_A = 1/3$). On the other hand, when $p_u \to 1/2$, the p_A equivalence range expands and tends to become equal to $(0, 1/2)$.[1] Note that the range of p_A values according to (3) is $(0, 1 - p_u)$ and, as $p_u \to 1/2$, this range tends to $(0, 1/2)$. Since the p_A equivalence range tends to become equal to the range of p_A values as $p_u \to 1/2$, we conclude that the probability that the two encodings are equivalent increases.

Concluding the above analysis, we can say that, for $p_u \geq 1/2$, the proposed encoding is equivalent to that of [8], for $1/3 \leq p_u < 1/2$, the probability that the proposed encoding is better than that of [8] increases as p_u gets closer to $1/3$, while, for $p_u < 1/3$, the proposed encoding is always more efficient. However, in practical cases, p_u is relatively small since, due to the inherent test vectors' correlation and the existence of many "x" values within them, the few encoded distinct blocks are compatible with the majority of the test set's data blocks (this is why high compression ratios are achieved). That is, in practice, the theoretical p_u's upper bound of $1/3$, under which the proposed encoding is always better than that of [8], is high enough to ensure the superiority of the proposed approach (this is also verified experimentally in the following section).

1. p_A cannot become equal to 0 or $1/2$ since $p_u < 1/2$.

4 Evaluation and Comparisons

The effectiveness of Selective Huffman coding has been verified with thorough experimental results and comparisons in [8]. Here, for demonstrating the advantages of the proposed encoding over that of [8], we implemented them both in C programming language and we conducted experiments on a Pentium PC for the largest ISCAS '89 benchmarks circuits. For the examined circuits, we assumed full scan with a single scan chain, while, as input, we used the dynamically compacted test sets generated by Mintest [5] for stuck-at faults.

We will first discuss the relation between p_u and compression ratio, for the two compared approaches

$$[Compression\ ratio\ (\%) = (1 - Compression\ bits/Mintest\ bits) \cdot 100].$$

In Fig. 4a and Fig. 4b, we present, respectively, the values of p_u and the compression ratios for benchmark circuit s38417, for 8, 16, and 32 encoded distinct blocks, and block sizes equal to 8, 12, and 16 bits (note that p_u is the same for the two compared methods and, thus, Fig. 4a refers to both of them). p_u depends on the number of distinct blocks that are encoded as well as on the block size. Specifically, as shown in Fig. 4a, p_u reduces as the encoded-distinct-block volume increases and/or as block size decreases. From Fig. 4b, we observe that, in general, both approaches yield better compression as p_u reduces. This is an expected behavior since smaller p_u values mean that fewer data blocks of the original test set remain unencoded. However, very small block sizes lead to many data blocks after the partitioning of the original test set, which can negatively affect the compression ratio. This is why, in four out of six cases of Fig. 4b, the best compression is not achieved for block size = 8 (although, by setting block size = 8, we get the smallest p_u). Thus, moderate block size values are preferable. We also observe that, for constant block size, the proposed method always benefits from greater encoded-distinct-block volumes. However, this is not true for the approach of [8] (see the corresponding bars for block size = 8) and, as explained by its authors, the reason is the extra-bit overhead imposed on all codewords. On the contrary, the proposed approach, which eliminates this overhead, can better exploit the advantage of more encoded distinct blocks. The above described behaviors can also be verified for the rest of the benchmark circuits in the results reported in Table 2.

In Table 2, we present the compression ratios achieved by the approach of [8] (columns 3-12) and by the proposed one (columns 13-22), for all examined benchmark circuits, using 8, 16, and 32 encoded distinct blocks, and block size equal to 4, 8,

TABLE 2
Compression Ratios (%) Achieved by [8] and the Proposed Encoding

Circuit	Mintest test sets (# bits)	[8]										Proposed encoding									
		8 enc. dist. blocks				16 enc. dist. blocks			32 enc. dist. blocks			8 enc. dist. blocks				16 enc. dist. blocks			32 enc. dist. blocks		
		block size				block size			block size			block size				block size			block size		
		4	8	12	16	8	12	16	8	12	16	4	8	12	16	8	12	16	8	12	16
s5378	23754	28.9	50.1	**53.0**	50.3	50.2	**55.1**	53.8	49.8	56.6	**56.9**	47.1	54.7	**54.9**	51.1	55.8	**57.5**	55.0	57.0	**59.6**	58.4
s9234	39273	30.0	50.4	**50.7**	46.1	50.2	**54.2**	51.0	49.0	**55.7**	54.4	51.4	**55.3**	51.8	46.1	**57.7**	56.8	51.9	58.9	**60.0**	56.2
s13207	165200	45.6	69.2	76.6	**78.6**	69.2	77.1	**79.7**	68.9	77.2	**80.2**	69.9	80.2	**83.4**	83.1	80.6	84.3	**84.5**	80.8	84.7	**85.3**
s15850	76986	38.8	60.0	**63.6**	61.6	59.9	**65.6**	64.8	59.2	66.5	**66.8**	62.1	**67.9**	67.3	63.6	69.3	**70.4**	67.2	69.9	**72.2**	69.8
s38417	164736	34.9	55.3	**56.9**	54.4	55.5	**58.9**	57.3	54.7	**60.3**	59.9	55.9	**60.9**	59.3	55.2	**62.8**	61.8	58.6	63.9	**64.5**	61.5
s38584	199104	37.8	58.5	**62.2**	61.7	58.5	63.9	**64.1**	57.7	64.5	**65.5**	60.4	**65.4**	65.2	63.3	66.9	**68.0**	66.1	67.3	**69.5**	68.2

12, and 16. Note that we do not provide compression ratio results for block size = 4 when 16 and 32 distinct blocks are encoded since there are no unencoded data blocks in these cases ($p_u = 0$). Also, we should mention that block sizes greater than 16 do not improve the compression ratio. We can see that the proposed approach always provides better compression than that of [8], except for the case of s9234 for eight encoded distinct blocks and block size = 16, for which the compression ratios of both methods are equal. Moreover, for every encoded-distinct-block volume in Table 2, we mark (in bold) the highest compression ratio that has been achieved for each benchmark circuit. In most cases, the best result of the proposed encoding (which is always better than that of [8]) is obtained with smaller block size (in the rest of the cases, the best compression is achieved with equal block sizes). This can be explained as follows: The proposed approach is favored more than that of [8] by smaller values of p_u (i.e., by smaller block sizes for the case at hand) since, as p_u decreases, more encoded-distinct-block codewords and fewer unencoded data blocks participate in the compressed test set (recall that the advantage of the proposed method is that it requires fewer bits than [8] for encoding the selected distinct blocks). Taking into account that, for both encodings, there is a minimum block-size value, below which the compression ratios worsen instead of improving, it is expected that the increasing gain of the proposed method over that of [8] as block size decreases would allow it to keep improving for smaller block size values than the minimum value of [8] (for the same number of encoded distinct blocks). In other words, compared to the approach of [8], the proposed one can "tolerate" smaller block sizes without a performance drop due to its smaller average length of the encoded-distinct-block codewords. This behavior also leads to reduced decompressor hardware overhead, as we will see shortly.

TABLE 3
Compression Improvement (%) of the Proposed Encoding over [8]

Circuit	8 encoded dist. blocks				16 encoded dist. blocks			32 encoded dist. blocks		
	block size				block size			block size		
	4	8	12	16	8	12	16	8	12	16
s5378	25.6	9.4	4.1	1.5	11.3	5.3	2.5	14.4	7.0	3.6
s9234	30.6	9.9	2.1	0.0	15.1	5.7	1.9	19.3	9.8	4.1
s13207	44.7	35.5	29.1	21.1	37.1	31.4	23.6	38.1	32.9	25.8
s15850	38.0	19.6	10.0	5.1	23.5	13.9	6.7	26.3	17.0	9.0
s38417	32.2	12.6	5.6	1.9	16.4	7.0	2.9	20.3	10.4	4.0
s38584	36.3	16.7	8.0	4.1	20.4	11.4	5.8	22.7	14.1	7.8

The compression improvements of the proposed encoding over that of [8], for all the compression cases of Table 2, are shown in Table 3 and are calculated by the formula:

$$Compression\ Improvement\ (\%) = \left(1 - \frac{Compressed\ bits\ of\ proposed}{Compressed\ bits\ of\ [8]}\right) \cdot 100.$$

We see that, in most cases, the proposed scheme provides considerably better compression than that of [8]. As expected, better improvements are achieved for greater encoded-distinct-block volumes and/or for smaller block sizes (i.e., for smaller values of p_u). This is illustrated in Fig. 5 for s38417.

In Table 4, we present the comparisons between the experiments with the highest compression ratio of the two approaches, for each encoded-distinct-block volume. We compare both the achieved compression ratios and the imposed area overhead and we report the improvement percentages in the corresponding columns ("Compr. Impr." and "Area Impr.," respectively). As far as the compression improvements are concerned, we observe that they are significant. This is justified by Fig. 6, which shows the distribution of p_u for the experiments compared in Table 4. Fig. 6 confirms our assertion that the p_u's upper bound of $1/3$ is high enough to ensure the superiority of the proposed encoding in practical cases since it is obvious that p_u is always below $1/3$. Also, as explained above, increased encoded-distinct-block volumes lead to smaller p_u values and, thus, to greater compression improvements, a trend that is clear in Table 4.

As for the area comparisons, we can see from Table 4 that, when, for the same encoded-distinct-block volume, the two compared best-compression results are obtained with the same block size (the cases in the nonshaded cells), the decompressors of both schemes occupy nearly the same area (the improvement ranges from -3.1 percent to +1.8 percent). This is explained by the fact that the proposed approach does not use more codewords than [8], but it just modifies the codewords' length (it shortens some of the codewords that encode distinct blocks and lengthens

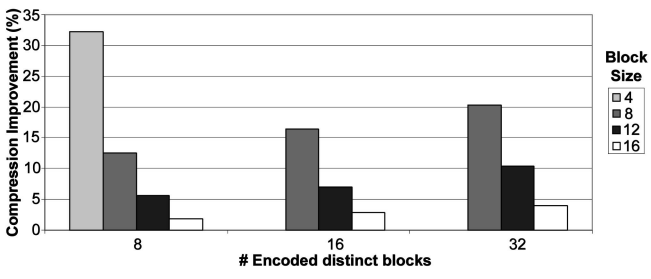


Fig. 5. Compression improvement for s38417.

TABLE 4
Compression and Area Comparisons (Improvement %) between Best Experiments of Table 2

Circuit	8 encoded dist. blocks		16 encoded dist. blocks		32 encoded dist. blocks	
	Compr. Impr.	Area Impr.	Compr. Impr.	Area Impr.	Compr. Impr.	Area Impr.
s5378	4.1	-1.9	5.3	-0.6	6.4	7.9
s9234	9.3	14.8	7.8	16.0	9.8	-2.7
s13207	22.3	7.6	23.6	-1.9	25.8	0.5
s15850	11.6	17.1	13.9	1.8	16.2	2.9
s38417	9.4	15.1	9.5	13.5	10.4	-3.1
s38584	8.6	15.6	11.0	5.3	11.5	5.4

the codeword that precedes the unencoded data blocks). However, since, in most cases, the highest compression ratio of the proposed encoding is obtained with smaller block size (shaded cells), the imposed hardware overhead is smaller than that of [8] (from +2.9 percent to +17.1 percent).

Fig. 7 illustrates the average test application time (TAT) improvements of the proposed method over: 1) the case of storing the uncompressed (Mintest) test sets in the ATE and 2) the approach of [8], for all of the experiments presented in Table 2 and for various values of r (r is the ratio of the frequency of an SoC's internal clock to the frequency of the ATE clock). TAT improvements are calculated the same way as compression improvements. In Fig. 7a, we assume that (as previously) each benchmark circuit has a single scan chain and that one ATE channel is used for transferring the test data to the SoC. Due to the utilization of a single ATE channel, the test data transfer time dominates TAT and, thus, improvements are only marginally affected by r. As a consequence, the average TAT improvements are close to the average test-data improvements, which are significant.

In Fig. 7b, we present the average TAT improvements assuming 32 scan chains for every circuit and four ATE channels for transferring the test data (compression and area comparisons for circuits with multiple scan chains are similar to those for the single-scan-chain circuits). Both the proposed method and [8] load the scan chains in parallel, with one decoded data block in every clock cycle (i.e., the number of cycles for loading each slice is equal to: $\lceil Number\ of\ scan\ chains/block\ size \rceil$). For being able to do this, either a separate scan-enable signal should be provided for every group of block size scan chains or a separate 32-bit register should be utilized. This register is first loaded with the decoded blocks and then its data are transferred to the scan chains in parallel. Note that, when Huffman coding is used, the decoded data blocks are generated in parallel (e.g., using a lookup table) and, thus, the parallelism offered by the multiple scan chains can be straightforwardly exploited. As for the uncompressed test set case, the test data are transferred to the SoC and loaded in the scan chains in groups of 4 bits, through the four ATE channels. The use of four ATE channels in the comparisons of Fig. 7b reduces the test data transfer time significantly, in this way increasing the influence of the decoding time (and, hence, r) on TAT. When r increases, the decoding time reduces and, thus, the TAT improvement of the proposed encoding over the uncompressed test set case gets higher. On the other hand, the decoding process of [8] is also performed faster as r increases and, therefore, the average TAT improvement over [8] is practically constant.

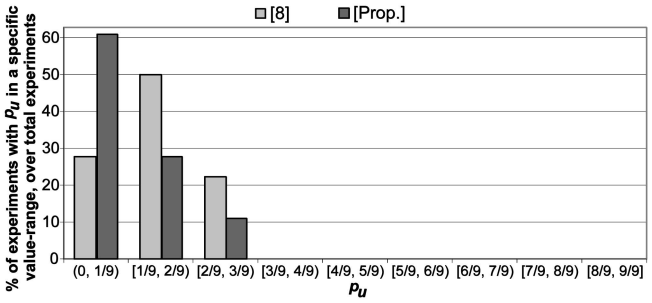
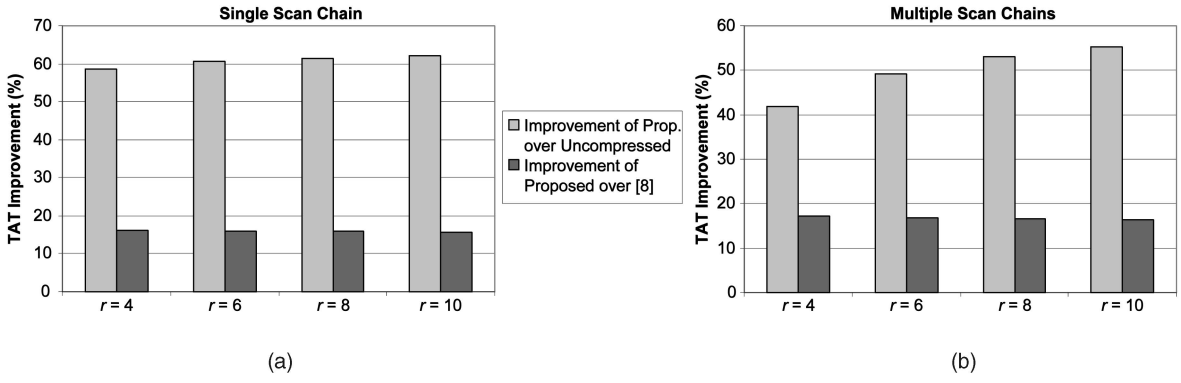


Fig. 6. Distribution of p_u for the experiments compared in Table 4.

Summing up, our experimental analysis verifies that, independent of what is more critical, area or compression, the proposed encoding is a better solution compared to [8]. Specifically, in a highly area-constrained design, the number of encoded distinct blocks should remain low (the size of the decompressor mainly depends on the encoded-distinct-block volume). As we have seen, although p_u increases as the number of encoded distinct blocks drops, it remains smaller than $1/3$ even in the case where only eight distinct blocks are encoded. This guarantees the superiority of the proposed approach. Further hardware overhead reduction can be achieved by using smaller block sizes, a strategy that favors the proposed encoding (since p_u reduces). On the other hand, in a less area-constrained design, higher compression ratios can be achieved through the utilization of more encoded distinct blocks, which leads to reduced p_u values, again rendering the proposed encoding the best choice. Finally, the improved compression of the proposed method results in smaller test application times compared to [8].

5 Conclusion

In this paper, it was shown that the Selective Huffman coding scheme of [8] is not optimal and an optimal one, with similar or



(a) (b)

Fig. 7. Average test application time (TAT) improvements.

often smaller hardware overhead, was presented. A theoretical analysis was also performed, demonstrating the superiority of the proposed encoding in practical cases. The improved compression efficiency, the low hardware overhead, and the reduced test application time of the proposed approach were verified by thorough experiments.

ACKNOWLEDGMENTS

The research project is cofunded by the European Union—European Social Fund (ESF) and National Sources, in the framework of the program "Pythagoras II" of the "Operational Program for Education and Initial Vocational Training" of the Third Community Support Framework of the Hellenic Ministry of Education.

REFERENCES

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

[2] A. Chandra and K. Chakrabarty, "A Unified Approach to Reduce SOC Test Data Volume, Scan Power and Testing Time," *IEEE Trans. Computer-Aided Design,* vol. 22, pp. 352-363, Mar. 2003.

[3] 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, no. 8, pp. 1076-1088, Aug. 2003.

[4] P.T. Gonciari, B. Al-Hashimi, and N. Nicolici, "Variable-Length Input Huffman Coding for System-on-a-Chip Test," *IEEE Trans. Computer-Aided Design,* vol. 22, pp. 783-796, June 2003.

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

[6] D.A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," *Proc. IRE,* vol. 40, pp. 1098-1101, 1952.

[7] V. Iyengar, K. Chakrabarty, and B.T. Murray, "Deterministic Built-In Pattern Generation for Sequential Circuits," *J. Electronic Test: Theory and Applications,* vol. 15, pp. 97-114, Aug./Oct. 1999.

[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. Computer-Aided Design,* vol. 22, pp. 797-806, June 2003.

[9] X. Kavousianos, E. Kalligeros, and D. Nikolos, "Efficient Test-Data Compression for IP Cores Using Multilevel Huffman Coding," *Proc. Design, Automation and Test in Europe Conf. and Exhibition,* pp. 1033-1038, Mar. 2006.

[10] X. Kavousianos, E. Kalligeros, and D. Nikolos, "A Parallel Multilevel-Huffman Decompression Scheme for IP Cores with Multiple Scan Chains," *Proc. IEEE European Test Symp. Inf. Papers Digest,* pp. 164-169, May 2006.

[11] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A SmartBIST Variant with Guaranteed Encoding," *Proc. 10th Asian Test Symp.,* pp. 325-330, Nov. 2001.

[12] C.V. Krishna and N.A. Touba, "Reducing Test Data Volume Using LFSR Reseeding with Seed Compression," *Proc. Int'l Test Conf.,* pp. 321-330, Oct. 2002.

[13] 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. Ninth Int'l Conf. Electronics, Circuits, and Systems,* vol. 2, pp. 449-452, Sept. 2002.

[14] M. Nourani and M.H. Tehranipour, "RL-Huffman Encoding for Test Compression and Power Reduction in Scan Applications," *ACM Trans. Design Automation of Electronic Systems (TODAES),* vol. 10, pp. 91-115, Jan. 2004.

[15] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded Deterministic Test," *IEEE Trans. Computer-Aided Design,* vol. 23, pp. 776-792, May 2004.

[16] M. Tehranipour, M. Nourani, K. Arabi, and A. Afzali-Kusha, "Mixed RL-Huffman Encoding for Power Reduction and Data Compression in Scan Test," *Proc. Int'l Symp. Circuits and Systems,* vol. 2, pp. II-681-4, May 2004.

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

[18] Z. Wang and K. Chakrabarty, "Test Data Compression for IP Embedded Cores Using Selective Encoding of Scan Slices," *Proc. Int'l Test Conf.,* pp. 1-10, Nov. 2005.

[19] P. Wohl, J.A. Waicukauski, S. Patel, and M.B. Amin, "Efficient Compression and Application of Deterministic Patterns in a Logic BIST Architecture," *Proc. Design Automation Conf.,* pp. 566-569, June 2003.