High-Quality Statistical Test Compression With Narrow ATE Interface

Vasileios Tenentes, *Student Member, IEEE,* and Xrysovalantis Kavousianos, *Member, IEEE*

***Abstract*—In this paper, we present a novel compression method and a low-cost decompression architecture that combine the advantages of both symbol-based and linear-based techniques and offer a very attractive unified solution that removes the barriers of existing test data compression techniques. Besides the traditional goals of high compression and short test application time, the proposed method also offers low shift switching activity and high unmodeled defect coverage at the same time. In addition, it favors multi-site testing as requires a very low pin-count interface to the automatic test equipment. Finally, contrary to existing techniques, it provides an integrated solution for testing multi-core system on chips (SoCs) as it is suitable for cores of both known and unknown structures that usually coexist in SoCs.**

***Index Terms*—Defect-oriented testing, design for testability, dft, IP cores, low-power scan-based testing, multi-core SoC, selective Huffman, test data compression, unmodeled defect coverage.**

I. Introduction

THE MAIN objective of test data compression (TDC) techniques is to compress large volumes of test data to small test sets that fit in the memory of automatic test equipment (ATE). In order to offer high compression, TDC techniques usually exploit the following inherent properties of test cubes (test cubes are vectors consisting of "0," "1," and x values).

1. The correlation between the specified "0" and "1" values that stem from the structural correlation of faults [1].
2. The large amounts of unspecified x values.

There are two mainstream TDC approaches: symbol based and linear based. Symbol-based techniques first divide the test data into several types of symbols and then they utilize codewords to encode these symbols [2]–[15]. An attractive technique is the selective Huffman code [3], [5] that offers low-cost decompressors and high compression at the same time. Symbol-based schemes are very effective in exploiting correlations in test cubes and they do not depend on the automatic test pattern generation (ATPG) process used. Consequently, they are very effective on precomputed (and usually precompacted and densely specified) test sets for intellectual property (IP) cores. However, they suffer from several serious drawbacks that prohibit their use in industrial designs: they do not exploit the low fill rate of test cubes; they impose long testing times as they cannot exploit the large number of scan chains; they require extensive interaction with the tester.

These shortcomings are tackled by linear compression methods. The most widely adopted linear compression method is the reseeding of linear feedback shift registers (LFSRs) [16]–[18]. LFSR reseeding exploits the low fill rate of test cubes. In [19], ring generators were proposed as an alternative to classical LFSRs and in [20], embedded deterministic test was presented. Other well known techniques have been presented in [21]–[28]. However, linear-based methods do not exploit the high correlation between test cubes' specified bits. In addition, they are ineffective for testing IP cores which are usually accompanied by precomputed and precompacted test sets.

A drawback of linear-based and symbol-based TDC techniques is that they elevate a switching activity beyond acceptable levels and thus degrade production yield [29]. A few symbol-based TDC techniques, such as [7]–[10], [15], inherently offer low shift power but they are not suitable for cores with multiple scan chains. Recently, linear decompressors that offer a low switching activity during testing emerged [30]–[32]. These techniques require additional data to control the switching activity. Specifically, in [33], a shadow register is utilized to offer low-power shift testing by repeating test data, while in [34], selective scan enable deactivation is used for low capture power. In [35], a TDC technique with narrow ATE-bandwidth requirements is presented. Even though the method proposed in [24] exploits similarities between test cubes, it is expected to be less effective in the case of IP cores where the test sets are highly compacted and test cubes usually exhibit many differences. Moreover, it requires new generation ATEs or imposes the use of circular buffers which result to large hardware overhead.

Another important issue in the nanometer era is the new types of defects that cause rapid growth of test data volume, test time, and inevitable power dissipation. Targeting all kinds of defects is a very difficult and costly solution, while there will always exist defects that known fault models cannot adequately cover. One solution for improving the defect coverage of test data without increasing their volume is to exploit the unspecified values (x) to increase the unmodeled defect cov-

Manuscript received November 25, 2012; revised January 27, 2013; accepted March 03, 2013. Date of current version August 16, 2013. A preliminary version of this work was presented in ICCAD 2011 with title "Test-Data Volume and Scan-Power Reduction With Low ATE Interface for Multi-Core SoCs." This work was supported by a collaboration between the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF)—Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund." This paper was recommended by Associate Editor X. Wen.

The authors are with the University of Ioannina, Ioannina 45221, Greece (e-mail: kabousia@cs.uoi.gr; tenentes@cs.uoi.gr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2013.2256394

0278-0070 © 2013 IEEE

erage of the test vectors. However, this target contradicts the main objective of TDC techniques, to exploit the unspecified values for improving the test data volume. Even worse, low-power TDC techniques consume all unspecified logic values to decrease the power dissipation and introduce high correlation in the decompressed test data which adversely affects the unmodeled defect coverage of the generated test vectors.

Even though there are many TDC techniques that partially target some of these objectives, to the best of our knowledge, there is no method to concurrently target all these objectives. In this paper, a novel unified approach is proposed that accomplishes all the aforementioned objectives. The main contributions of the proposed method are given below.

1) It exploits both the low fill rate and the correlations in the specified bits of test cubes and thus outperforms both symbol-based and linear-based encoding methods.
2) It offers low shift power during testing.
3) It supports very low pin–count interface as a single ATE channel suffices for fast downloading test data on-chip.
4) It offers short test application times (TATs) as it exploits the large number of scan chains of modern cores.
5) It does not require any kind of synchronization between the ATE and the circuit under test.
6) It is decoupled from the ATPG process and offers high compression even on highly compacted test sets of IP cores.
7) It is suitable for both IP and non-IP cores of modern system on chips (SoCs).
8) It exploits ATE's repeat command (wherever available) as an embedded feature of the encoding process to further decrease test data volume.

In addition, we show that statistical codes introduce significant correlation in the generated test vectors and thus offer low unmodeled defect coverage. To this end, a new technique is proposed that offers a tradeoff between test data volume and unmodeled defect coverage. This objective has not been targeted yet in the literature by any code-based TDC technique.

Finally, we present a low-cost decompression architecture that can be shared among multiple cores without compromising compression. In particular, it offers the potential to share hardware resources and test data, in order to cost effectively test multiple cores with different characteristics. The proposed decompressors can be shared even between IP and non-IP cores and thus they offer an additional advantage to achieve higher quality test solutions for SoCs at lower cost. Extensive experiments with largest ISCAS and a subset of large IWLS benchmark circuits [36] show the benefits of the proposed method as compared to already existing TDC methods.

II. Motivation

The Huffman code [37] is a fixed-to-variable code that uses short codewords to encode frequently occurring blocks and long codewords for the less-frequent ones. The optimal selective Huffman (OSH) code encodes only the m most frequent blocks [3], while the rest of the blocks remain unencoded and are distinguished by using an extra Huffman codeword [5].

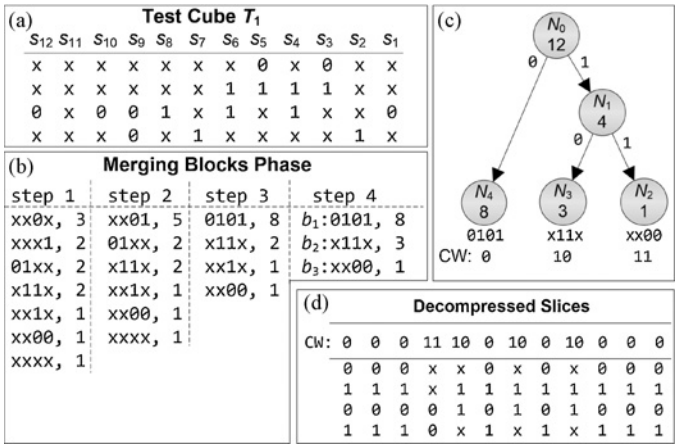

Fig. 1. Classical selective Huffman coding.

Let us assume a core with n scan chains of length r (we assume a balanced scan structure where the shorter scan chains are padded with x logic values). Each scan slice constitutes a single block (we will remove this restriction later). Let T be a set of test cubes and $|T|$ be its size in bits. T is partitioned into $|T|/l$ data blocks of size l. Among these blocks, the m most frequent distinct blocks, $b_1, b_2, ..., b_m$, with frequencies (probabilities) of occurrence $f_1 \geq f_2 \geq \cdots \geq f_m$, respectively, are stored in a dictionary and they are encoded using m Huffman codewords. The rest of the blocks with an aggregate frequency $f_{\text{un}} = f_{m+1} + f_{m+2} + \cdots$ remain nonencoded and a single codeword is used to precede them as (they are stored in a raw form in the compressed test data [5]). A binary tree is constructed beginning from the leaves and moving toward the root. For every dictionary entry b_i, a leaf node is generated and a weight equal to f_i is assigned to it. The pair of nodes with the smallest weights is selected first and a parent node is generated with a weight equal to the sum of the weights of both nodes. This is repeated iteratively, until the root is left unselected (each node can be selected only once). After the tree is constructed, each leaf node is assigned a codeword as follows: 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 block b_i is the sequence of the logic values of the edges on the path from the root to the leaf node corresponding to b_i.

Example 1: In Fig. 1(a), the test cube T_1 is presented that is partitioned into $r = 12$ scan slices $s_1, s_2, ..., s_{12}$. The scan slice s_i is the part of T_1 that simultaneously loads the n scan chains $SC_1, SC_2, ..., SC_n$ at the clock cycle c_i (the scan slices are loaded from right to left, i.e., the scan slice s_1 is loaded first, s_2 is loaded second, etc.). The number of scan chains is $n = 4$ and the block size is $l = 4$ (each scan slice is considered as a test data block). The test data blocks and their numbers of occurrences are shown in the first column of Fig. 1(b). As proposed in [3], the two most frequent compatible blocks are merged (two blocks are compatible when they do not differ at any bit position where they are both specified). The merging provides a new block that has the same specified bits with the two blocks at the same bit positions with them. This is iteratively applied until no further merging of blocks is

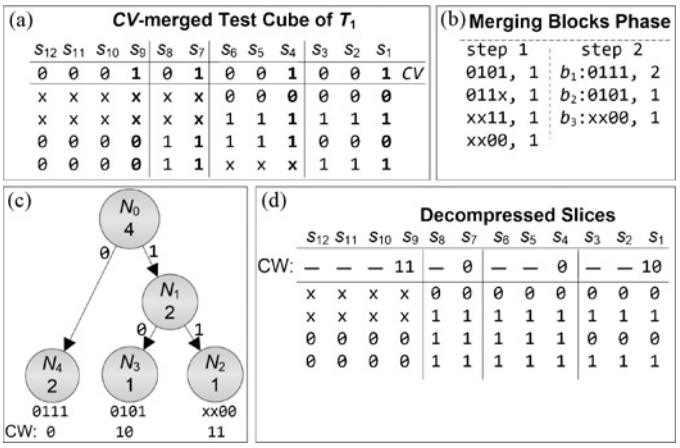

Fig. 2. Selective Huffman coding with premerged blocks.

possible. Columns 2–4 of Fig. 1(b) present the merged blocks generated after the most frequent blocks are merged each time (next to each block, its frequency of occurrence is reported). The final encoded distinct blocks are presented in the fourth column of Fig. 1(b) and they constitute the dictionary of the OSH. The Huffman tree that belongs to the dictionary example of Fig. 1(a) is shown in Fig. 1(c). The codewords assigned to entries "0101," "$x11x$," and "$xx00$" are "0," "10," and "11," respectively. The compressed test set has a size of 16 bits and it is shown in the first row of the decompressed slices in Fig. 1(d) (one codeword per slice). The decompressed test data are shown below that row. ■

Despite the fact that there are many blocks in a test set consisting mostly (or even entirely) of x values, each and every one of them has to be encoded using a separate codeword. However, the blocks corresponding to scan slices s_1, s_2, s_3 of the test cube are compatible and thus they can be merged into a single block. Another block can be generated by merging scan slices s_4, s_5, s_6, s_7, s_8 and another one by merging $s_9, s_{10}, s_{11}, s_{12}$. If we encode these three blocks instead of the blocks shown in the fourth column of Fig. 1(b), then we can use the first block for loading the scan chains for the first three cycles, the second block for the next five cycles, and the third block for the last four cycles. This way, we use only three codewords to encode the test cube. In order to know during each scan-in cycle if the last block is repeated or not, we need one control bit per block. If the next block is the repetition of the previous one, this bit is set to logic value "0"; otherwise, it is set to logic value "1." The control bits of all slices of a test cube comprise the *control vector* (CV). The CV that corresponds to the merging of scan slices $s_1 \ldots s_3$, $s_4 \ldots s_8$, and $s_9 \ldots s_{12}$ is CV = 000100001001 (the rightmost bit corresponds to s_1 and the leftmost to s_{12}). This CV indicates that 1) blocks $s_1 \ldots s_3$, $s_4 \ldots s_8$, and $s_9 \ldots s_{12}$ are merged, 2) the resulting blocks are loaded at the first, fourth, and ninth clock cycles and they are repeated for another two, four, and three clock cycles, respectively. The test cube formed after merging its blocks according to CV is called *CV-merged*.

The storage of CV along with the two codewords results to worse compression than the original encoding shown in Example 1. However, for sparsely specified test cubes (i.e., test cubes with many x), there are many different CVs that can be used. Even for the test cube at hand, which consists of 33.3% specified bits and is rather densely specified, there are many CVs that can be used, like for example 000100001001, 000100000011, 000100100101, etc. (as shown in many studies [20], [25], the vast majority of test cubes of industrial designs consists of less than 5% of specified test bits). Each different CV implies a different merging process. We exploit this property to generate pseudorandomly the CV vectors based on probabilistic properties of test cubes. CVs are generated prior to the encoding and thus apply a block pre-encoding merging phase which minimizes the number of blocks which will be encoded using OSH. This process can be better illustrated by means of an example.

Example 2: Let us assume that for the test cube T_1 of Example 1 the CV = 000101001001 has been pseudorandomly generated. Fig. 2(a) presents the CV-merged T_1 for this CV. There are four groups of merged blocks, while only the first block of each group is encoded using OSH (these blocks are shown in bold and correspond to the logic value "1" in CV). The rest of the blocks are simply repeated versions of the first block in each group. The CV constitutes the guide for the encoding process as shown in Fig. 2(b). The number of occurrences of the blocks is calculated based on the number of times each block has to be encoded (note that only the blocks corresponding to logic values "1" in the CV, i.e., the first one of each group, are encoded and thus the number of occurrences of each block is equal to 1). The generated code and its tree are presented in Fig. 2(c). Finally, Fig. 2(d) shows the codeword used for each block corresponding to the logic value "1" in the vector CV, as well as the decompressed blocks loaded in the scan chains. It is obvious, since CV is not stored, that only six bits are needed to compress the given test cube.■

The compressed bits in the above example are much less than the specified bits of the test cube. Linear methods cannot reduce the size of the compressed test set below the volume of its specified bits unless methods like [24] are employed. At the same time, the unspecified values of test cubes are compressed adequately. In addition, the encoding of the most frequently occurring blocks decreases the size of the encoded test data and thus exploits the correlation between specified test bits. Moreover, loading the same values in successive scan slices reduces also the shift power during scan-in (and consequently the scan-out shift power as shown in [38]). Further optimization can be achieved by employing techniques like [24] on top of the proposed technique.

The idea of grouping scan slices for the purpose of low-power shift-in has been proposed earlier in [30] and [31]. This grouping requires the encoding of additional control data and specifically one bit per test slice. Every control bit is encoded together with the test data bits. Depending on the number of scan chains, the volume of control bits can be high compared to the average number of specified bits for a test set. Assume that the 10K scan cells of the largest circuit in our suite, the ethernet, are organized into 100 scan chains (100 test slices per vector), 100 control bits must be encoded for any test cube. For an average fill rate of 1% (i.e., 100 specified bits per test cube), along with the 100 specified bits for each test cube, we

need to encode additional 100 control bits. In this paper, we show that control data can be completely avoided and can still achieve very high shift power reduction.

III. Encoding Method

A. Generation of CV

The encoding method and the generation of the pseudorandom vector CV are strongly interdependent processes. Each CV implies a specific merging of the blocks of test cubes. Some of the test cubes will have their respective blocks compatible and thus they can be generated using this particular CV (we call hereafter those test cubes CV-mergeable, or mergeable for the CV). The probability that a given CV can be used to premerge a given test set depends on two factors.

1) The volume of "0" logic values of CV: a large volume of "0" logic values imposes extensive compatibility restrictions between successive test slices of test cubes and decreases the possibility of an arbitrary test cube t to be mergeable for this CV. The opposite happens when the CV consists of many "1" logic values. For example, every test cube is mergeable for the all-ones CV as every slice is encoded independently of the rest of the slices.
2) The volume of x values of test cubes: the higher is this volume, the higher is expected to be the number of compatible slices of the test cubes and thus a larger population of CVs can be used for such test cubes.

CVs with many "0" values can be used for merging sparsely specified test cubes, while CVs with many "1" values are needed for the densely specified test cubes. However, from the compression and power perspective, CVs with large volumes of "0" values are more effective than CVs with large volumes of "1" values (in the first case, less blocks are encoded and less transitions occur during the scan-in as more blocks are repeated versions of their preceding blocks). A good practice is to begin from CVs with a small volume of "0" values to merge sparsely specified test cubes and then gradually increase this volume to merge the remaining densely specified cubes.

A vector CV is generated pseudorandomly as a signal that is set to a logic value "1" with a probability P_{CV}. P_{CV} is set equal to various discrete probability values $p_1, p_2, \ldots, p_b$ ($0 \leq p_1 < p_2 < \cdots < p_b = 1$). P_{CV} is initially set to p_1 and then it is gradually increased to p_2, p_3, etc. For each value of P_{CV}, many CVs are generated and each one is used to load one test vector in the scan chains. Every CV is generated by a pseudorandom unit which will be described in the next section. This unit is synthesized prior to the encoding process and thus the exact CV sequences are known during the encoding. Note that if we set $P_{\mathrm{CV}} = 1$ throughout the whole encoding process, then all blocks are encoded using traditional OSH [5]. Therefore, this approach is a generalization of OSH.

In order to show the effectiveness of pseudorandomly generated CVs to encode large test sets, we performed an experiment using the ethernet circuit of the IWLS suite [36]. This circuit consists of more than 10 000 scan cells, so it is more representative of realistic industrial designs than the rest of the IWLS benchmark circuits. Fig. 3 depicts the percentage of test cubes that are mergeable for various CVs

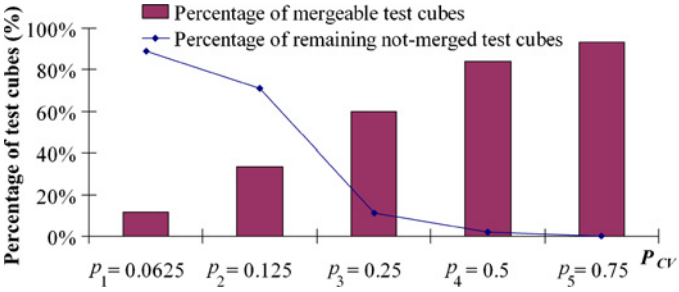

Fig. 3. Percentage of mergeable test cubes for ethernet.

generated pseudorandomly (note that the generated test cubes achieve 100% coverage of stuck-at faults). The x-axis presents P_{CV} values used in this experiment for generating CVs. For each P_{CV} value, 100 different CVs were generated and the percentage of test cubes which are mergeable for at least one of them is reported by means of bars. As the value P_{CV} increases, more test cubes become mergeable for the generated CVs. The curve shows the test cubes which remain nonmerged at the end of each step (test cubes which are mergeable for any CV generated are immediately dropped in this case). Note that more than 80% of the test cubes are mergeable for CVs consisting of less than 25% logic values "1." Also, less than 2% of the test cubes require CVs with 50–75% of the logic values being equal to "1." As a result, the vast majority of blocks do not have to be encoded at all (for the above experiment, more than 70% of blocks are generated as repeated versions of other encoded blocks and thus they require no test data). In conclusion, the effectiveness of CV depends on the fill rate of test sets, which is fairly low in large circuits, and not on the size or amount of test cubes. Therefore, the proposed method is scalable to very large test sets.

CVs also affect static and/or dynamic compaction of test cubes. Note that the term compaction refers to a different process than encoding or compression. In particular, static compaction is the process of merging all compatible test cubes to be encoded later (static compaction precedes encoding), while dynamic compaction is the process of merging test cubes during the encoding (after encoding the first test cube, additional compatible test cubes are encoded in the same test vector generated by the decompressor). These processes, applied during (or prior to) the encoding process, decrease the volume of test vectors generated and offer additional compression and test time benefits. Both types of compaction can be applied in the proposed method. In fact, they can be applied even more aggressively than in linear-based methods, reducing the volume of applied vectors.

B. Pre-encoding Merge Process

The pre-encoding merge process is a step-by-step process that is applied before the OSH encoding. The objective of this process is to reduce the volume of test cubes to be encoded as much as possible and thus decrease both the test data volume (less test slices to be encoded) and the test sequence length (TSL) (less test vectors to be generated). At the beginning, all test cubes are appended in a set TS and the probability value of CV is set to the minimum discrete value p_1. Then,

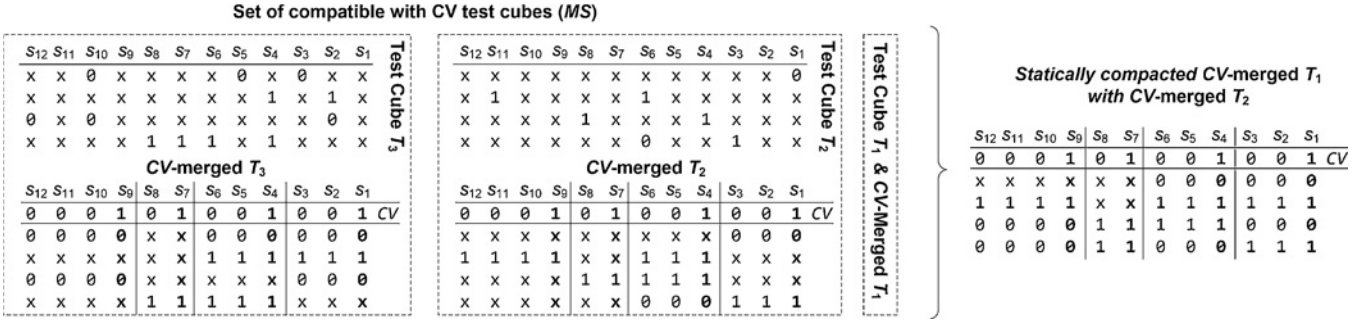

Fig. 4. Premerging example.

an iterative process begins and at each iteration, a new CV is generated based on the value of P_{CV} (note that one CV is a sequence of r logic values which are needed for loading one test vector into the scan chains). Then, the test cubes of set TS that are mergeable for the generated CV are identified and are moved to an empty set MS. If no test cube is mergeable for the current CV, then, the probability P_{CV} is increased to the next higher discrete value, and the iteration starts over by generating a new CV. One test cube of set MS is selected and is merged using the CV. Then, all test cubes of set MS that cannot be statically compacted with t_1 are removed from the set MS and are appended again back to the set TS. Out of the remaining test cubes in MS, one test cube is selected, let say t_2, and it is statically compacted with t_1. This is iteratively applied until the set MS becomes empty. While TS is not empty, the process continues, until TS becomes empty, by generating the next CV for the current P_{CV} value.

At each iteration, among the test cubes of MS, we select first the hardest-to-merge test cubes, i.e., those test cubes that are less likely to be mergeable by CVs generated using low values of P_{CV}. This is done in order to increase the number of test cubes that are merged at the early stages for low P_{CV} values; the ones that offer better compression and shift power than higher P_{CV} values. To this end, we rank the test cubes using a measure which is representative of this likelihood. Let t be a test cube and $i \in [1, n]$ be a scan chain of the core. The volume of incompatibilities, INC(t, i), of scan chain i for test cube t, is the number of successive test slices of t with complementary logic values at their positions corresponding to scan chain i. Note that test cubes consist also of x logic values which affect this measure based on the way they are filled. To alleviate this problem we adopt an approximation according to which every x logic value shifted into the scan chain is considered to be equal to the last specified logic value "0" or "1" which was encountered during the loading of this scan chain for cube t. Note that the x values of test cubes are not actually filled and thus test cubes remain unaffected by this process. For example, for the test cube t of Fig. 1(a) we have $INC(t, 1) = 0$, $INC(t, 2) = 0$, $INC(t, 3) = 2$, and $INC(t, 4) = 1$. This approximation is reasonable as the proposed method fills the x values in a similar manner, therefore there is a high probability that the x values will be eventually filled in this manner (e.g., this is the case for the cube when it is encoded as shown in Fig. 2). Finally, we calculate the volume of incompatibilities $INC(t)$ of test cube t as the sum of the incompatibility values $INC(t, i)$ of all scan chains $i \in [1, n]$. The test cube with the highest value of $INC(t)$ is considered as the hardest-to-be-merged test cube and it is always the first one selected from set MS.

Example 3: Consider the test cube T_1 of Example 1 and the CV shown in Fig. 2, and assume that T_1 along with test cubes T_2 and T_3 form test set TS shown in Fig. 4 (T_1 is omitted in Fig. 4). T_1, T_2, T_3 are all mergeable for the CV shown in Fig. 2 and thus they are moved to set MS. The incompatibility values are $INC(T_1) = 3$, $INC(T_2) = 1$, $INC(T_3) = 0$ and thus T_1 is first selected to be encoded [the CV-merged version of T_1 was shown in Fig. 2(a)]. We can see that the the CV-merged versions of T_2, T_3 (shown below T_2, T_3 in Fig. 4) are both compatible with the CV-merged version of T_1. Since $INC(T_2) > INC(T_3)$ cube T_2 is selected next and the CV-merged versions of T_1, T_2 are statically compacted as shown at the right of Fig. 4. The CV-merged cube T_3 is no longer compatible with the resulting CV-merged test cube and thus it is moved back to set TS to be encoded using a new CV. ■

C. Slice Partitioning

So far, we have assumed that a whole slice is encoded as a single Huffman block. This is realistic when the size of a slice (and thus the volume of scan chains) is small (we remind that OSH achieves good compression for relatively small sized blocks [3], [5]). For cores with many scan chains, we partition the set of scan chains into groups of equal size and each group is encoded separately. Let l be the required block size and n be the number of scan chains. Then, the set of scan chains is partitioned into $k = \lceil n/l \rceil$ groups $G_1, G_2, \ldots, G_k$. Each group is assigned its own $CV_1, CV_2, \ldots, CV_k$, which is generated by its own properly selected probability P_{CV_j}. This partitioning process is applied before the merging phase and the same groups are retained for the whole test period.

Scan chains with similar volumes of incompatibilities (as they were calculated in the previous section) are appended to the same group. Then, groups with low volume of incompatibilities are assigned lower initial probability P_{CV} than those groups with higher volume of incompatibilities. This increases the probability of test cubes to be mergeable with the generated CVs. In order to keep the encoding process (and the decompression process) simple, each time that a set $CV_1, CV_2, \ldots CV_k$ fails to encode a test cube (i.e., no test cube is mergeable for this set of CVs), then each one of the probabilities $P_{CV_1}, P_{CV_2}, \ldots P_{CV_k}$ is increased to its next higher

1: TS: set of test cubes, n: number of scan chains, l: size of block
2: Partition n scan chains into $k = \lceil n/l \rceil$ groups
3: Calculate P_{CV_j} value for each group G_j.
4: **while** TS is not empty **do**
5: Generate $CV_1, \ldots CV_k$ of r bits each using $P_{CV_1}, \ldots P_{CV_k}$.
6: Move test cubes which are mergeable for the CVs into MS.
7: **if** set MS is empty **then**
8: increase all $P_{CV_1}, P_{CV_2}, \ldots P_{CV_k}$ values to the next higher discrete value and go to the next iteration
9: **else**
10: Statically compact and drop test cubes of MS.
11: **end if**
12: Move test cubes remaining in MS back to set TS.
13: **end while**
14: Encode the resulting blocks using repeat-friendly OSH.

Fig. 5. Encoding process.

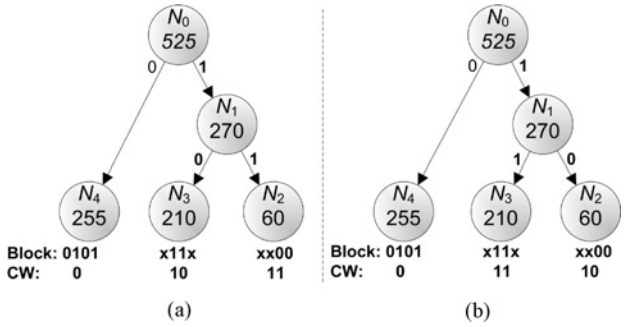

Fig. 6. Swap procedure on node N_1 for repeat-friendly (RF) code.

discrete value until all of them reach the highest discrete values (i.e., those that are equal to p_1 are increased to p_2, those that are equal to p_2 are increased to p_3, etc.).

First, we present how the scan chains are partitioned into disjoint groups of size l each. Let i be a scan chain. We define the incompatibility load $L(i)$ of scan chain i as the sum of the $INC(t, i)$ values of all test cubes t of test set T, i.e., $L(i) = \sum_{t \in T} INC(t, i)$. The incompatibility load of each scan chain is then normalized by the worst load, i.e., the highest value $L_{\max} = \max\{L(i)\}$ found for any scan chain i, using formula $NL(i) = L(i)/L_{\max}$. Then, the n scan chain are appended in a list of ascending order of their $NL(i)$ values. The first l scan chains of this list comprise the first group G_1, the next l scan chains comprise the next group G_2, etc. Finally, we define the normalized load $NL^G(j)$ of group G_j as the maximum $NL(i)$ value of its members i, with $0 \leq NL^G(1) \leq NL^G(2) \leq \cdots \leq NL^G(k) = 1$.

After partitioning the scan chains into groups, the initial probability P_{CV_j} used for generating CV_j for each group G_j is determined. This probability depends on two factors: 1) the relation between the normalized load values of different groups and 2) the tradeoff between compression, power, and TAT as set by the test engineer. The test engineer selects the initial probability, let say a, for generating CV_k that is the CV of the group with the largest load. The rest of the groups with lower load than G_k are automatically assigned an initial probability which is lower than a proportionally to their NL^G value. Specifically, for each group G_j a probability value pg_j is calculated using the formula $pg_j = a \times NL^G(j)$ (we remind that $NL^G(j) \leq 1$). In that way group G_k is assigned the probability a (note that $NL^G(k) = 1$) and the rest of the groups are assigned lower probabilities. The higher is the value of a, the lower is the TSL as many test cubes are mergeable right from the beginning and higher levels of static compaction can be reached. However, a large value of a usually results to less compression and higher average shift power. The opposite trend is observed when a small value of a is used. The encoding process is shown in Fig. 5.

D. Repeat-Friendly Huffman Code

Even though Huffman is a very effective code, further improvements can be achieved by exploiting certain ATE utilities, like the repeat command [39]. Using the repeat command, multiple successive identical logic values can be stored only once in the ATE-channel memory and they can be repeatedly transmitted over the ATE channel in successive cycles. Huffman code optimizes the length of the codewords, but the codewords are not repeat friendly. For example, the codewords "10" and "11" of Fig. 2(c) require 2 and 1 bits, respectively, when the repeat command is used (note that the first codeword has no identical successive logic values and thus the repeat command has no effect, whereas the second codeword has only one logic bit repeated two times and thus the repeat command eliminates the need to store the last bit of this codeword each time it is used). It is obvious that higher gain can be achieved by assigning RF codewords to frequently occurring blocks.

As noted in Section II, after the tree is constructed each leaf node is assigned a codeword by assigning the logic value "0" ("1") to each left (right)-child edge. In order to provide RF codewords, we propose a very simple modification of the edge-assignment process. At the beginning, the edge to the left (right) child of the root is arbitrarily assigned the logic value "0" ("1"). Then we visit the rest of the nodes starting from these two nodes and moving toward the leafs. Every node is processed only when the edge connecting the node to its parent has been assigned a logic value. Let node A be one of these nodes. We find the child of A with the highest weight and we assign at the edge connecting node A with this child the same logic value that is assigned to the edge connecting node A with its own parent. The opposite logic value is assigned to the edge connecting node A to its other child with the smallest weight. This way, the more frequently occurring blocks are assigned more RF codewords.

Example 4: Fig. 6(a) presents the Huffman tree of Fig. 1(c) enhanced with more realistic block frequencies. The memory space required for storing the compressed test data using the repeat command (ignoring logic-bit repetitions between different codewords) is equal to $255 \times 1 + 210 \times 2 + 60 \times 1 = 735$ bits. The modified Huffman tree is shown in Fig. 6(b). Even though codewords' length are the same with Fig. 6(a), the leaf nodes are assigned different codewords. In Fig. 6(b) the edge connecting nodes N_1 and N_3 is assigned the logic value "1" and the codeword for b_2 is "11" instead of "10" that was in Fig. 6(a) Huffman tree. Memory space required in this case is $255 \times 1 + 210 \times 1 + 60 \times 2 = 585$ bits, which is considerably lower than that required by the encoding of Fig. 6(a). ■

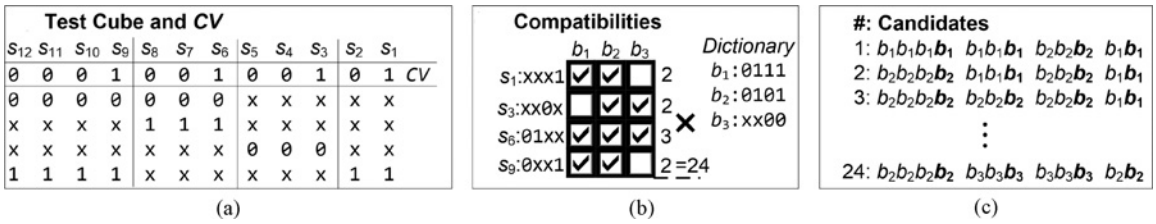

Fig. 7. Blocks replacement and candidates generated.

IV. Unmodeled Defect Coverage Improvement

The repetitive loading of identical test data into successive slices and the biased encoding of the blocks toward the most frequent blocks induces correlation, which adversely impacts the unmodeled defect coverage of the generated test vectors [40]. Unmodeled defect coverage can be improved by decreasing the correlation between test vectors and also by adopting effective quality metrics to assess the test quality of each vector. The correlation of test vectors can be reduced by relaxing the tight objective of the encoding process to use the most frequent dictionary entries for encoding test data blocks. The use of a small minority of test data blocks repeatedly is responsible for biasing the generated test vectors toward similar patterns of specified bits.

Test quality can be assessed using output deviations [41]. Test patterns with high output deviations tend to be more effective for defect detection. Output deviations are probability measures at primary outputs and pseudooutputs that reflect the likelihood of error detection at these outputs. Intuitively, the deviation for an input pattern is a measure of the likelihood that the circuit outputs are incorrect for that pattern. An efficient metric based on output deviations, which considers also the structure of the core was proposed in [42]. The output-deviation-based metric proposed in [43] further increases the unmodeled defect coverage of the selected test vectors by considering both timing-independent and timing-dependent defects. In this section, we propose a cost-effective process to generate candidate test vectors and evaluate them using the output-deviation based metric proposed in [43]. Based on this evaluation, the encoding process selects the best test vectors in terms of unmodeled defect coverage.

The block substitution technique can be applied after step 14 of the encoding process (not shown in Fig. 5). Candidate test vectors are generated by exploiting the potential of sparsely specified test data blocks to be encoded using multiple dictionary entries. Specifically, according to OSH, every test data block is encoded using the most frequent dictionary entry in order to be favored by the shortest possible codeword length. If we remove the requirement of encoding at the minimum cost, then sparsely specified blocks can be encoded using compatible less-frequent dictionary entries.

Example 5: To understand how the blocks replacement technique works let us again consider the Huffman code of Fig. 2. We also consider the *CV*-merged test cube presented in Fig. 7(a) that needs to be encoded using this code. Based on the *CV* shown on the top of this test cube the blocks to be encoded are the three blocks corresponding to slices s_1, s_3, s_6 and s_9 shown in Fig. 7(a). The block $xxx1$ is compatible with entries b_1 and b_2 of the dictionary (Fig. 2) and thus both codewords "0" and "10" can be used to encode this block. In each case a different test vector will be generated which is compatible to the encoded test cube (note that these two encodings are actually two different ways of filling some of the unspecified values of the test cube). The two encodings are not equally effective in terms of test data volume because codeword "10" is more expensive than codeword "0." The same can be done for the rest of the slices. In Fig. 7(b) we present all possible compatibilities between the slices s_1, s_3, s_6, and s_9 of Fig. 7(a) and dictionary entries b_1, b_2, b_3. Based on this table, we can generate all possible candidates that are equal to $2 \times 2 \times 3 \times 2 = 24$ and are shown in Fig. 7(c) in ascending order of test data volume (bold entries correspond to the encoded blocks, nonbold to repeated blocks). ■

In most cases, the encoding cost is expected to increase as we can only use less-frequent dictionary entries for each block than those used by the original encoding. The highest overhead is imposed by the extreme case: when an encoded by a dictionary entry block is left unencoded and it is preceded by the codeword corresponding to the unencoded blocks (OSH provides this option [5]). Even though this is always an available option for any block (any block can be simply left unencoded), it is very expensive and should be wisely and rather rarely used.

Block substitution changes the dictionary block frequencies and thus codeword lengths generated for the initial frequencies will not be any further optimal for the new frequencies. The additional overhead can be moderated if the codewords are regenerated to properly reflect the new frequencies of the dictionary entries. To this end, the tree is generated again for the new frequencies resulting after block substitution and a new codeword is assigned to each dictionary entry. In order to further reduce the additional overhead of this process, the proposed method bounds the volume of the blocks that are involved in this substitution process. This is achieved through the use of a predetermined probability P, called hereafter as probability of blocks change. For example, when $P = 10\%$ only a 10% of randomly selected blocks will be encoded by suboptimal entries. Higher values of P increase the test data volume (TDV) cost but also the gain in unmodeled defect coverage.

V. Decompression Architecture

The proposed decompression architecture is shown in Fig. 8. It consists of four main units: the selective Huffman decoder (SHD), the signal probabilities generation (SPG)

unit, the CV generation (CVG) unit and scan-registers $SR_1, SR_2, \ldots, SR_k$ which load the scan chains. It operates as follows: the SHD unit receives the compressed data from ATE and decodes the codewords. It loads one scan register at a time with the decoded block. This register is selected by the unit CVG that generates the CVs $CV_1, \ldots, CV_k$. When CV_i is asserted, scan register SR_i is loaded with the block decoded at the output of the SHD unit. Signals $CV_1, \ldots, CV_k$ are generated from pseudorandom signals produced at the SPG unit. When all registers SR_i (which have their CV_i signals asserted) are loaded with new test data, the slice is shifted into the scan chain and the decompression proceeds to the next scan slice (the rest of the registers hold their contents). Let us see each unit in detail.

The *selective Huffman decoder unit* (SHD) loads serially the compressed test data from a single ATE channel and provides the decoded blocks of size l each. It consist of a finite state machine (FSM) which decodes the codewords and a small dictionary which stores the distinct block encoded by each codeword. It generates two signals namely *DecodedBlock* and *BlockReady*. It asserts signal *BlockReady* when a new block is available at the *DecodedBlock* output.

The *scan registers unit* (SR) consists of k, l-bit registers, $SR_1, \ldots, SR_k$ which correspond to groups $G_1, \ldots, G_k$, respectively. Each scan register SR_i is controlled by signal $Load_i$ which is asserted whenever CV_i is asserted and the respective test data are available at the *DecodedBlock* output. When $Load_i$ is not asserted then the register holds its contents.

The *signal probabilities generation unit* (SPG) is shown in Fig. 9. It consists of a small LFSR that is initially loaded with a random seed, and a very small combinational logic which generates pseudorandom signals of various probabilities in the range of $[0, 1]$. The operation of this unit is very simple: each LFSR output has probability of 50% to receive logic value "1." A 2-input AND gate driven by two signals with probability 50% provides at its output a signal with probability $P_{\text{out}} = 25\%$. By using various combinational gates with various numbers of inputs, signals with many discrete probabilities can be generated. In the proposed scheme the eight probabilities shown in Fig. 9 are implemented by just few 2–4 input gates. Since an independent CV has to be generated for each group, a different group of signals $p_1, \ldots, p_8$ is needed for each CV ($p_1^1, p_2^1, \ldots p_8^1$ are used for CV_1, $p_1^2, p_2^2, \ldots, p_8^2$ for CV_2, etc). In order to eliminate correlations between signals $CV_1, CV_2, \ldots, CV_k$ phase shifters are inserted between the LFSR and the combinational logic. SPG unit is controlled by the CVG unit with signal *SliceLoaded*. This signal is asserted whenever the scan chains are loaded with the current slice and it enables the LFSR to move to its next state and to generate signals $p_1^j, p_2^j, \ldots, p_8^j$ $(j = 1 \ldots k)$ for the next slice.

The *control vector generation unit* (CVG) is responsible for both vectors $CV_1, CV_2, ..., CV_k$ and signals $Load_1, Load_2, \ldots, Load_k$. For each CV_j signal one multiplexer 8-to-1, namely P-MUX$_j$, is used to select one of the $p_1^j, p_2^j, \ldots, p_8^j$ signals generated by SPG unit. p_1^j is connected to the first input of P-MUX$_j$, p_2^j is connected to the second input, etc. The selection inputs of P-MUX$_j$ are connected to counter GC$_j$. This counter is initialized before decompression

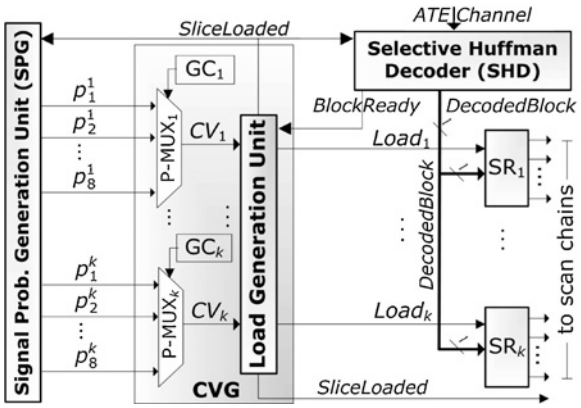

Fig. 8. Proposed decompression architecture.

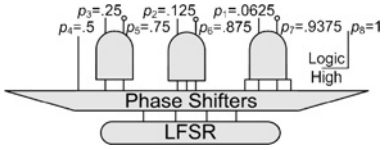

Fig. 9. Signal probabilities generation unit.

process begins with the value corresponding to the initial P_{CV_j} calculated at the beginning of the encoding process. Each time P_{CV_j} has to be increased to the next higher discrete value counter GC$_j$ is triggered once to count up. This way it selects the next input of P-MUX$_j$ which is already connected to the pseudorandom signal with the next higher probability. When *BlockReady* is asserted CVG unit loads the decoded block to the first scan register that has its respective CV signal asserted and waits until the next test data block is decoded. When all scan registers with logic value "1" at their CVs for the current slice are loaded then current slice is shifted into the scan chains and decoding continues to the next slice.

All GC counters are simultaneously triggered during the decoding process whenever P_{CV} values are increased. This is done at the most seven times as the minimum possible initial value for P_{CV} is equal to p_1 and it can be potentially increased up to p_8. Each triggering has to be done at a specific vector in the test sequence which is determined during the encoding (Fig. 5). The whole vector sequence is controlled by the means of a vector counter (not shown in Fig. 8 for simplicity). We use seven registers (also not shown in Fig. 8) which are loaded before the decompression process begins with the specific vector-counter values that should trigger each time the GC counters. Thus, each time the vector counter reaches the value stored in any of these registers all the the GC counters are triggered once.

The rate at which the blocks are decoded at the output of the SHD unit depends on the length of the codewords (long codewords need more cycles to be decoded). As a result, there might be cases that the loading of the scan registers has to wait for the next block to be decoded and vice versa. The first case is handled by the CVG unit which controls the loading of the shift registers and stalls both this loading and the scan-in operation when it is necessary. For the second case there are two solutions. First, a FIFO can be used at the output of the SHD unit to hold all blocks which are decoded early (usually a

very small FIFO suffices to store all such blocks). The second solution is to let the SHD hold the last decoded block and ignore any additional test data sent by the ATE. If the repeat command is available then it can be used to eliminate these data by repeating the last useful bit for as many cycles as needed without incurring additional overhead. When the ATE-repeat command is not available both techniques can be used at the same time to offer a tradeoff between hardware overhead and test data storage. As a result, no handshaking is required between the ATE and the decompressor.

The SHD unit of the proposed architecture is test set dependent. SHD unit can be designed in a test-set-independent way if 1) the dictionary is implemented using a small RAM that is loaded from the ATE before the decompression process begins and 2) the FSM is designed to be generic (similar to [44] and [45])—provided of course that it decodes a predetermined number of codewords (the exact codewords can be determined at a later step of the design process). As it has been shown in [3] and [5], a very low number of blocks, 8–16, suffices to provide high compression efficiency. Even in the case that the SHD is designed to be test set dependent, last minute design changes are neither expected to affect the Huffman decoder nor the dictionary entries (the same decompressor can be still used even at the cost of a marginal reduction of the compression achieved). Only in the case of extensive design modifications (that cause also extensive changes on the test sets of the SoC's cores) the SHD must be redesigned to reflect these changes.

Multiple cores residing in the same SoC require in many cases decompressors tuned to different parameter values. This requires developing a dedicated decompressor for each core which is an expensive approach. In order to tackle this issue we propose the development of a low-cost reconfigurable decompressor which can adjust its characteristics to the requirements of multiple cores at the expense of a slight increase in test data volume. This decompressor can be shared among multiple cores for decreasing hardware cost, without sacrificing compression. Specifically, we assume a single decompression unit for multiple cores which uses a common FSM for the cores but a separate dictionary for each one (note that the hardware implementation of the dictionary can be also common if it is implemented as a RAM that is loaded with the particular contents of each core before it is tested). Using this technique, the same codewords are used for the cores that share the decompressor, but each codeword corresponds to a different entry that is found at a separate dictionary for each core. In that way, the FSM, which is the major contributor to the overhead of the decompressor, is shared among multiple cores while at the same time the dictionaries which occupy very limited space and offer great compression benefits are dedicated and optimized to the characteristics of each core.

The multi-core decompressor is presented in Fig. 10. To account for different scan chain configurations among different cores we use the same block size l for those cores that share the decompressor, and we equip the decompressor with the maximum number of SRs used by any of the cores (we remind that each core requires a number of SRs that is equal to the number of scan chains divided by the block size). While testing each core only the necessary SRs are activated.

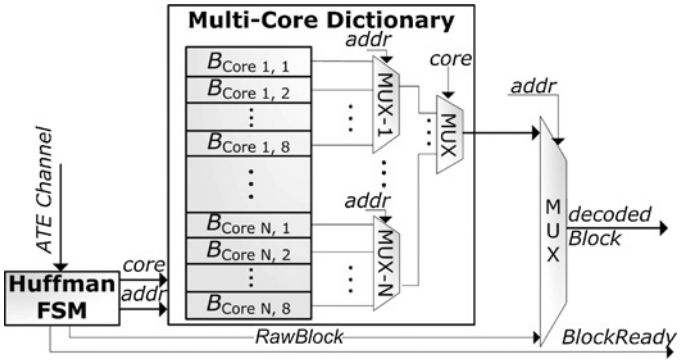

Fig. 10. Selective Huffman decoder.

The use of common codewords for a group of cores requires proper adjustments of the encoding process. In particular, the RF encoding (i.e., step 14 at Fig. 5) cannot be applied for any core until the blocks corresponding to the dictionary entries of every core are generated. The aggregate number of occurrences of these blocks are used to generate common Huffman codewords. Since in each core the most frequently met block will be encoded using the shortest codeword, the aggregation of the frequencies is done in such a way as to bias the frequencies of the common codewords. In particular, the number of occurrences of the most frequent blocks are summed to provide the frequency of the most frequent codeword of the group, and these blocks are stored in the first entry of each dictionary. Then, the same process is applied to the second most frequent block of each core, etc.

Example 6: Let us assume an SoC consisting of two cores with four dictionary entries and number of occurrences (in descending order) $f_1^1 = 30$, $f_2^1 = 22$, $f_3^1 = 20$, $f_4^1 = 18$ for the first core and $f_1^2 = 20$, $f_2^2 = 15$, $f_3^2 = 10$, $f_4^2 = 9$, for the second core. The aggregate frequencies for both cores become $f_1^{1+2} = 50$, $f_2^{1+2} = 37$, $f_3^{1+2} = 30$, $f_4^{1+2} = 27$. Based on the aggregate frequencies a Huffman tree is constructed and it is optimized with the repeat friendly optimization method presented in Section III-D. The resulting codewords are used to build a common FSM for the cores. ■

By properly grouping cores and allocating one decompressor at each group, the area cost can be retained low. In addition, cores that share a common decompressor should be located close in the floorplan to minimize the routing overhead. Finally, the ability of the proposed decompressors to work well with both IP and non-IP cores enables the sharing of decompressors among both of these types of cores and offers a further degree of freedom during the grouping process and the allocation of decompressors to each group. Note that scheduling techniques can be applied to further decrease the TAT but they are out of the scope of this paper.

VI. Experimental Results

We implemented the proposed encoding scheme using the C++ programming language and we synthesized the decompression logic using commercial tools. The proposed method was applied on the largest ISCAS'89 and a subset of the large IWLS benchmark circuits [36]. We used a commercial tool to generate test sets for 100% stuck-at fault coverage. Unless

TABLE I
BENCHMARKS INFORMATION

Circuit	Gates	PPI/PPO	Scan cells	$n \times r$
s5378	4285	86	214	16 × 14
s9234	4190	77	247	16 × 16
s13207	10 103	154	700	32 × 22
s15850	11 919	103	611	32 × 20
s38417	30 460	136	1664	64 × 26
s38584	26 864	292	1464	48 × 31
ac97_ctrl	28 554	132	2253	32 × 71
mem_ctrl	11 440	267	1194	48 × 25
pci_bridge	45 055	369	3517	128 × 28
tv80	14 223	46	372	32 × 12
usb_funct	27 081	249	1858	32 × 59
ethernet	157 520	211	10 647	128 × 84

otherwise stated, the parameters used for the proposed method were set equal to $m = 8$ codewords and $l = 8$ bits block size. The running time for the largest circuit, the Ethernet, was less than 3 min. Table I contains information on the benchmarks and in particular the size in combinational gates, the pseudorandom primary input/output pins count (PPI/PPO), the scan cells count and scan structure represented as number of scan chains (n) multiplied by the length of scan chains (r).

We implemented the state-of-the-art low-power dynamic reseeding (LPDR) proposed in [46] using ring generators with sizes in the range [40, 150]. In this case we used a single shadow register to favor the TDV measurements of this method. The shadow register was implemented using both techniques proposed in [30] and [46] (internal XOR tap or one additional ATE channel) and the best result is reported in every case. In the case of LPDR, the repeat command was utilized to further reduce the compressed test data. Moreover, as suggested in [20] and [46] the unsolved variables were filled in a RF way to improve further the TDC of LPDR. Even though both the ATPG and fault simulation steps can be straightforwardly embedded in both the LPDR method and the proposed encoding, we omitted these steps as they cannot be applied in the case of IP cores (their internal structure is unknown). We also compare our method to various code-based methods [5], [6], [47], and [25]. For all methods we use the minimum number of ATE channels, that is one channel for the proposed method and code-based methods, and two channels for LPDR (a very small number of channels is highly desirable in a multi-site test environment).

For evaluating the unmodeled defect coverage we used a surrogate fault model, i.e., a fault model that is not targeted by the generated test sets. That fault model is the transition delay fault (TDF) model. For detecting transition faults each test vector generated by the decompressors is applied on the circuit using two capture cycles according to launch-on-capture (LOC) technique. Note that similar approaches were adopted in many techniques (e.g., [40], [41], [48]).

The measurements of average switching activity (ASA) were done using the normalized weighted transitions metric (WTM) [30].

A. Impact of Parameters a and l

In Fig. 11 we present the TDV, ASA, TSL, and TAT results of the proposed method for the s13207 benchmark circuit

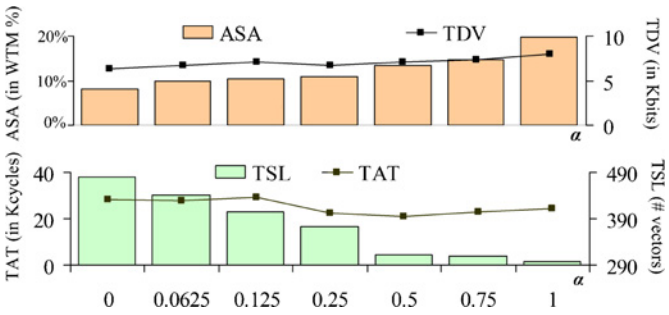

Fig. 11. Tradeoffs for a value for s13207.

for various values of the parameter a (a is used to generate the starting signal probabilities of the CVs as shown in Section III-C). It consists of two parts that are aligned on a common x-axis (shown at the bottom) that presents the selected values for parameter a. The top part presents the TDV curve (1 kbit = 1000 bits) and the ASA measures (bars) of the proposed method, while the bottom bar presents the TAT (line) and the TSL measures (bars). The smaller is the value of parameter a, the more sparse are the generated CVs in terms of their "1" logic values and thus the more power efficient are the generated test vectors ("0" logic values in the CVs load the scan chains with the same test data and thus reduce the shift power dissipation). Higher values of parameter a cause the test vectors to become less power efficient as they increase the number of update operations on the shadow register. At the same time, as a increases the encoding process is less constrained by the shift power objectives and thus more test cubes are encoded into each test vector during the premerging process. As a result TSL drops considerably but the shift power (ASA) increases.

An interesting property of the proposed method is that TAT is not affected by the value of a as much as TSL is affected. TSL depends on the number of vectors applied to the core, while TAT depends mostly on the codeword decoding process and thus on the TDV which does not depend much on the value of a. Note that the loading of the scan chains is done in parallel with the decoding of incoming data from the ATE. When the value of a is high many update operations occur in a short time and the Load Generation Unit has to stall in these cases for new test data to be decoded through the SHD. Even though the TSL is low (and consequently the overall number of test slices loaded into the scan chains is low), when the value of a is high, there is a high number of update operations that render the SHD unit the bottleneck of the test generation process. As the value of a drops, the overall number of test slices loaded into the scan chains increases (due to the increase of TSL) but most of the additional test slices are repeated versions of their previous ones and are generated without incurring any additional decoding cost (no test data need to be decoded for those slices). As a result, the load generation unit stalls less frequently and the decoding process is very well parallelized with loading the scan chains using mostly repeated test data. Therefore, we conclude that a low value of a is more preferable from both TDV and power perspectives, while the additional test vectors applied due to the increased TSL in that case, can

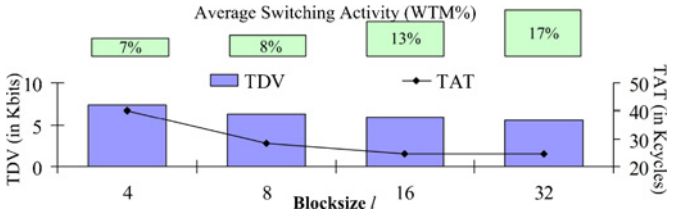

Fig. 12. TDV, TAT, and ASA for various blocksize l values on s13207.

be exploited to increase the unmodeled defect coverage with a very small impact on TAT.

Another important parameter that affects both the ASA and the TAT of the proposed method is the block size. The larger the block size is, the smaller is the number of scan chain groups and thus the lower is the number of blocks that need to be decoded for every scan slice. However, as the number of scan chain groups decreases, the benefits on reducing shift power reduces (the probability that a block can be repeatedly loaded into the scan chains drops as the size of the block increases). Fig. 12 presents results for TDV, TAT, and ASA for various blocksize values on s13207 benchmark circuit. As block size l increases, both TDV and TAT drop while ASA increases. Beyond a certain block size (i.e., equal to 32 in the case at hand) TAT increases again while TDV saturates. Therefore, depending on the power budget of the design, an optimal block size exists for every circuit which can be easily found due to proposed method's short CPU time.

B. Impact of Parameter P

The impact of parameter P effect is shown in Fig. 13 where the lines correspond to TDF and the bars to TDV results. We present results for the proposed defect-aware encoding for $a = 0.125$ and various values of P labeled as $P = 0.01$, $P = 0.1$, $P = 0.5$, and the proposed defect-unaware encoding labeled as DU. We also present results for the LPDR method labeled as LPDR. The x-axis presents the number of vector pairs applied using the LOC scheme for all the methods and the y-axis presents the TDF measures for each technique. Although the proposed defect unaware (DU) technique is superior in terms of TDV as compared to the LPDR method (7.0 kbits over 20.8 kbits), it is inferior to LPDR in terms of coverage on the surrogate transitions-delay faults. This is the effect of the increased correlation of the test slices that results from the biasing of the encoding process toward a small number of frequently occurring test data blocks. However, the defect aware proposed scheme improves considerably the TDF coverage. Even for very small values of P (i.e., the case labeled as $P = 0.01$) which correspond to the case that only a very small percentage of blocks are substituted for increasing test quality, TDF becomes 59.8% and almost reaches that of LPDR, with only a slight increase of the test data (they become 7.8 kbits from 7.0 kbits). If we further increase P to the value of $P = 0.1$ and $P = 0.5$ the TDF of the proposed technique reaches higher values than that of LPDR.

C. Repeat-Friendly Huffman Code: TDV Improvement

We run 10 different experiments for each of the 11 benchmark circuits for the proposed method by varying the

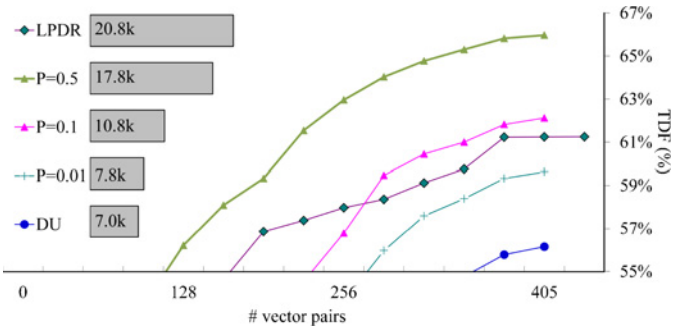

Fig. 13. Tradeoffs for blocks substitution probability P on s13207.

blocks substitution probability P from the set of values [0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.25, 0.5, 1]. In each case, the TDV improvement TDV_{impr} offered by using the RF version of the proposed method over the original not optimized encoding (Orig) was calculated by using the formula $TDV_{Impr} = (TDV_{Orig} - TDV_{RF})/TDV_{Orig}$ (we note that in both RF and "Orig" cases the compressed test data were further encoded using the ATE repeat command). In the 100 out of the 110 cases, the TDV improved using RF and the improvement was as high as 17.5%. The average and median improvement was 5% and 6.18%, respectively. In the rest five cases the TDV slightly increased and the increment was in the range [0%–2.3%]. We note that the repeat friendly encoding optimizes the internal characteristics of codewords (intracodeword) but it does not consider the sequence of codewords (inter-codeword) which also affect the TDV. This may cause a slight reduction of the TDV benefits and in those rare cases that the gains from intra-codeword improvements are not high, it slightly increases the overall TDV.

D. Comparisons

In Table II the TDV, TSL, and ASA comparisons of the proposed method (labeled as "Prop.") against LPDR and OSH [5] are presented. For this comparison we used uncompacted test sets because LPDR is very efficient with uncompacted test sets. In all cases the proposed method offers the lowest TDV among all methods. The improvement ranges between $1.5x$ and $4.3x$ as compared to the LPDR method and between $2.6x$ and $8.8x$ as compared to the OSH method. The TSL of the proposed method is lower than that of the LPDR method and higher than that of the OSH. We note that the TSL greatly depends on the static and/or dynamic compaction policy followed during the encoding (in our case this is the premerge process and, as noted in Section III, it belongs to the static compaction processes). Static and/or dynamic compaction in linear encoding methods is generally constrained by the size of the linear decompressor, the number of variables injected into the decompressor and the power constraints. Specifically, the number of free variables available during the encoding of every test cube must be higher than the number of specified bits of the test cube. Since every additional encoded test cube consumes variables for its encoding, it is rather unlike that all compatible test cubes can be encoded together at the same test vector using linear encoding, unless a large number of variables are injected at each clock cycle.

TABLE II
COMPARISONS OF TDV, TSL, AND ASA

circuit	TDV (in kbits)			TSL (no. of vectors)			ASA (WTM %)		
	LPDR	OSH	Prop.	LPDR	OSH	Prop	LPDR	OSH	Prop.
s5378	10.0	15.7	5.1	305	155	277	5.8	21.3	14.2
s9234	20.9	29.9	11.7	504	259	438	11.6	21.0	18
s13207	20.8	39.5	10.5	419	255	405	5.4	19.9	12.4
s15850	26.8	43.6	11.2	552	243	523	7.0	24.7	12.9
s38417	97.5	150.9	49.5	1548	267	1094	6.2	31.2	13.5
s38584	89.7	112.3	33.9	1179	245	591	7.0	32.4	13.2
ac97_ctrl	57.5	60.8	15.5	2161	77	206	1.3	34.4	11.1
mem_ctrl	115.5	205.7	37.7	2466	884	1581	5.0	14.7	7.4
pci_bridge	233.0	418.3	120.6	3435	654	1062	20.6	21.8	20.4
tv80	72.5	131.8	48.1	2330	1662	2115	10.8	35.8	19.0
usb_funct	98.2	156.6	49.9	2340	231	763	1.1	35.2	15.5
ethernet	612.4	1257.0	143.1	3115	1100	1811	2.7	10.4	15.4

TABLE III
COMPARISON OF TDF

circuit	TSL (no. of vectors)		TDV (in kbits)				TDF (%)			
	LPDR	Prop.	LPDR	Proposed			LPDR	Proposed		
				DU	ME	He		DU	ME	HE
s5378	305	277	10.0	**4.3**	4.9	5.1	62.0	60.2	62.4	**63.4**
s9234	504	438	20.9	**10.1**	11.2	11.7	49.8	45.6	49.1	**49.9**
s13207	419	405	20.8	**7.0**	13.7	17.8	61.3	56.2	63.5	**66.2**
s15850	552	523	26.8	**10.5**	11.2	13.7	54.7	54.3	56.4	**57.2**
s38417	1548	1629	97.5	**53.0**	64.5	77.2	87.8	83.2	86.3	**87.9**
s38584	1179	1130	89.7	**44.1**	44.1	45.2	67.0	65.4	68.2	**68.4**
ac97_ctrl	2161	2136	57.5	**31.1**	36.7	36.4	53.4	51.3	56.3	**58.8**
mem_ctrl	2469	2357	115.5	**32.3**	74.4	81.3	**43.7**	33.1	36.8	39.3
pci_bridge	3435	2774	233.0	**105.3**	169.7	230.7	81.9	65.5	81.0	**83.2**
tv80	2330	2426	72.5	**45.2**	54.7	74.9	59.9	55.2	60.4	**62.3**
usb_funct	2340	2379	98.2	**52.6**	75.8	94.9	72.7	61.8	71.5	**72.9**
ethernet	3115	1811	612.4	**143.1**	182.9	252.6	55.3	49.1	57.7	**64.4**

On the contrary, code-based decompressors (like for example the Huffman decompressor in the OSH case) do not suffer from this restriction and thus permit the application of very aggressive static compaction processes before the compression which reduce the TSL a lot. Even though this is also the case for the proposed method, the power objectives implemented through the use of CVs introduce additional constraints into the static compaction process (premerging) that do not permit the TSL to drop as much as it does in the OSH method. However, due to the inherent property of the proposed method to encode the vast majority of test slices using repeating test data, the proposed method offers much lower TAT than the OSH approach which counterbalances the increase in TSL.

As far as the ASA measures are concerned, the LPDR gives the lowest WTM values. In most of the cases this is also related to the long TSL of this technique, that is even 10 times higher in one case than that of the proposed method, and that permits higher repetition of the test data loaded into the scan chains. However, the WTM values of the proposed method are very low and much lower than those of OSH technique. Note that, as it was shown in Fig. 11, ASA can be considerably reduced by using lower values of α parameter, which constitute a favorable selection for the proposed method.

In Table III, we present the unmodeled defect coverage comparisons between the proposed method and LPDR. We consider three different instances of the proposed method: 1) the DU instance where the test quality improvement technique was not applied; 2) the defect aware encoding with the values of parameter P selected from the range $[0.1, 0.25]$ denoted as medium effort (ME) encoding, and 3) the defect aware encoding with the values of parameter P selected from the range $[0.5, 1]$ denoted as high effort (HE) encoding. We have to note that the unmodeled defect coverage depends a lot on the number of test vectors applied. Thus, for providing a fair comparison, we applied restrictions on the premerging phase of the proposed method to increase its TSL (note that the proposed method offers considerably lower TSL than LPDR). In the first two columns the TSL results are presented. The next four columns present the TDV comparisons. Note that the TDV of the proposed method is lower than that of LPDR for all values of P in almost all cases (the best results are bolded). The last four columns present the TDF comparisons (the highest TDF entries are bolded). Note, that DU has relatively low TDF that however is considerably improved by using the proposed test quality improvement process. In all but one case, when the effort for increasing defect coverage is set to high, the proposed method offers the highest TDF, and still retains lower TDV than LPDR. In the vast majority of the cases, the proposed method offers higher TDF than LPDR even when the effort is set to medium. So, we conclude that the proposed method offers much lower TDV than LPDR and at the same time it offers a trade-off between the TDV improvements and the unmodeled defect coverage, yielding higher unmodeled defect coverage than LPDR for higher values of P.

TABLE IV
COMPARISONS WITH TDC TECHNIQUES (IN KBITS)

Circuit	[6]	[47]	[25]	[15]	Proposed
s9234	12.8	30	–	20.6	**10.2**
s13207	14.6	21	74	28.9	**6.3**
s15850	16.6	25	26	25.1	**10.5**
s38417	58.7	85	45	59.0	**44.2**
s38584	55.4	57.1	74	74.9	**24**

TABLE V
HARDWARE OVERHEAD COMPARISONS

n	Dynamic Reseeding HO					OSH	Proposed HO				
	d=48	d=64	d=96	d=128	d=150*		k=1	k=2	k=4	k=8	k=16
32	626	774	1072	1370	1574	344	560	622	746	994	1490
64	805	954	1251	1549	1754	417	633	695	819	1067	1563
128	1164	1313	1610	1908	2112	563	779	841	965	1213	1709

*150 is the size of ring generator used for *ethernet* in Table II.

In Table IV, we compare the proposed method against some of the best TDC techniques in the literature in terms of TDV. The methods in [6], [47], and [25] focus on test time and TDV optimization and method [15] is a TDC method that targets average power reduction too. In the case of the proposed method we set the value of parameters $l = 8$, $P = 0$, and $a = 0.0625$. For this comparison we used compacted test sets because these methods perform better with them. It is obvious that the proposed method offers the lowest TDV.

In Table V, we present the area overhead of the decompressors for LPDR, OSH and the proposed method for various scan chain volumes n, ring generator sizes d, and number of groups k. The hardware overhead is measured in terms of gate

equivalents, where one gate equivalent corresponds to the area of a 2-input NAND gate. The area overhead strongly depends on the values of the parameters used and it is relatively low in all methods. In general the hardware overhead of the proposed method is larger than OSH but lower than dynamic reseeding.

We have to note that the hardware overhead of the proposed decompressors does not depend on the size of the core under test, but on the dictionary size, the number of codewords, the block size, and the number of scan chains n and scan chain groups k. As it was shown in previous studies (like [3] and [5]) a relatively small block size in the range $[8-16]$ and a small number of codewords and dictionary entries in the range [8–32] suffice for high compression. In addition, the value of k is decided by the test engineer and it can be always in the range [1–16] (the value of $k = 16$ is already very high). In addition, the overhead increases linearly with k, n. Therefore, the hardware overhead of the decompressors is not expected to increase for even larger circuits than those reported in Table V. In addition, as it is shown in Tables II and III the largest circuit, the Ethernet, which is almost one order of magnitude larger than most of the rest circuits, gives the best results. Therefore the proposed encoding method is expected to scale very well even for larger circuits.

E. IP-Cores and Multi-Core Experiments

In order to show the effectiveness of the proposed method for precomputed test sets of IP cores, we applied it on a precompacted test set of the largest circuit, the Ethernet. Note that in the case of IP-cores, precomputed and most likely precompacted test sets are provided to the test engineer of the SoCThe size of the compacted test set was 11.6 Mbits (1 Mbit = 10^6 bits) and after applying the proposed method the TDV dropped by a factor higher than $50x$ and reached the value of 221.7 kbits. The number of specified bits of the initially generated (and highly compacted) test set is 263.8 kbits which clearly show that the proposed method succeeded to reduce the test data volume below this number which is a lower bound for most of the compression techniques. The TSL was 1100 and the WTM value was 9.2% which are both very low. The TDV of the OSH was found to be more than five times higher than that of the proposed method, and specifically it is equal to 1246.7K. We note that the LPDR technique is not applicable in this case as the large variation of the specified bits in the test set requires the use of unrealistically large (in the range of thousand of cells) ring generators. It is also worth noting that despite the fact that this is a very compacted test set and thus the proposed premerging process has no effect, the compressed TDV of the proposed method is very close to that shown in Table II which was computed using noncompacted test sets that offer higher degrees of freedom in the encoding process. So, we conclude that the proposed method is very efficient for both IP and non-IP cores.

In the last experiment, we study the performance of the proposed method in a multi-core SoC. To this end we synthesized a hypothetical SoC consisting of all the cores presented in Table II. For simplicity, we assume that all cores are tested in a nonoverlapping manner while concurrency can be straightforwardly applied if multiple decompressors are available. In order to show the effectiveness of the proposed technique even in the extreme case that only one decompression unit is available for the entire SoC, we used a single decompression unit for all the cores and we kept the FSM of the decompression architecture common in all cases, i.e., the same codewords were used for all cores (note that any test scenario with multiple decompressors will give an even better solution in terms of compression and TAT). For each of the cores we assumed a different dictionary which was optimized to the particular characteristics of the core. The overall TDV is equal to 529.9 kbits. The overhead in that case was found to be less than 0.5% of the overhead of the SoC. In the case that a different decompressor is used for every core (optimizing thus the FSM and TDV for each particular core separately) the hardware overhead increases to the 3% of the SoC. Therefore, the shared decompressor offers very high TDV benefits at a very small area cost, and thus offers a very compelling solution for testing multi-core SoCs.

VII. Conclusion

In this paper, we presented a unified TDC approach for very low pin-count interface of multi-core SoC that is very effective for cores of both known and unknown structure as it offers the combined advantages of symbol-based and linear-based techniques. In addition, the proposed scheme offers an effective low-cost solution (in terms of area overhead) for testing multi-core SoCs. Therefore, we conclude that the proposed method can serve as an attractive alternative to the widely adopted solution of linear-based encoding.

References

[1] N. A. Touba, "Survey of test vector compression techniques," *IEEE Des. Test*, vol. 23, no. 4, pp. 294–303, Apr. 2006.

[2] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolici, "Variable-length input Huffman coding for system-on-a-chip test," *IEEE Trans. Comput.-Aided Des.*, vol. 22, no. 6, pp. 783–796, Jun. 2003.

[3] A. Jas, J. Ghosh-Dastidar, M.-E. Ng, and N. A. Touba, "An efficient test vector compression scheme using selective Huffman coding," *IEEE Trans. Comput.-Aided Des.*, vol. 22, no. 6, pp. 797–806, Jun. 2003.

[4] X. Kavousianos, E. Kalligeros, and D. Nikolos, "Multilevel Huffman coding: An efficient test-data compression method for IP cores," *IEEE Trans. Comput.-Aided Des.*, vol. 26, no. 6, pp. 1070–1083, Jun. 2007.

[5] X. Kavousianos, E. Kalligeros, and D. Nikolos, "Optimal selective Huffman coding for test-data compression," *IEEE Trans. Comput.*, vol. 56, no. 8, pp. 1146–1152, Aug. 2007.

[6] X. Kavousianos, E. Kalligeros, and D. Nikolos, "Test data compression based on variable-to-variable Huffman encoding with codeword reusability," *IEEE Trans. Comput.-Aided Des.*, vol. 27, no. 7, pp. 1333–1338, Jul. 2008.

[7] A. Chandra and K. Chakrabarty, "System-on-a-chip test-data compression and decompression architectures based on Golomb codes," *IEEE Trans. Comput.-Aided Des.*, vol. 20, no. 3, pp. 355–368, Mar. 2001.

[8] A. Chandra and K. Chakrabarty, "A unified approach to reduce soc test data volume, scan power and testing time," *IEEE Trans. Comput.-Aided Des.*, vol. 22, no. 3, pp. 352–363, Mar. 2003.

[9] A. Jas, J. Ghosh-Dastidar, and N. A. Touba, "Scan vector compression/decompression using statistical coding," in *Proc. VTS*, 1999, pp. 114–120.

[10] A. Chandra and K. Chakrabarty, "Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes," *IEEE Trans. Comput.*, vol. 52, no. 8, pp. 1076–1088, Aug. 2003.

[11] 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," in *Proc. ICECS*, 2002, pp. 449–452.

[12] M. H. Tehranipoor, M. Nourani, and K. Chakrabarty, "Nine-coded compression technique for testing embedded cores in SOCS," *IEEE Trans. Very Large Scale Integr.*, vol. 13, no. 6, pp. 719–731, Jun. 2005.
[13] K. Basu and P. Mishra, "Test data compression using efficient bitmask and dictionary selection methods," *IEEE Trans. Very Large Scale Integr.*, vol. 18, no. 9, pp. 1277–1286, Sep. 2010.
[14] G. Wolff and C. Papachristou, "Multiscan-based test compression and hardware decompression using LZ77," in *Proc. ITC*, 2002, pp. 331–339.
[15] M. Nourani and M. H. Tehranipour, "Rl-Huffman encoding for test compression and power reduction in scan applications," *ACM Trans. Des. Autom. Electr. Syst.*, vol. 10, pp. 91–115, Jan. 2005.
[16] B. Konemann, "LFSR-coded test patterns for scan designs," in *Proc. ETC*, 1991, pp. 237–242.
[17] C. V. Krishna, A. Jas, and N. A. Touba, "Test vector encoding using partial lfsr reseeding," in *Proc. ITC*, 2001, pp. 885–893.
[18] C. V. Krishna, A. Jas, and N. A. Touba, "Achieving high encoding efficiency with partial dynamic lfsr reseeding," *ACM Trans. Des. Autom. Electr. Syst.*, vol. 9, no. 4, pp. 500–516, Oct. 2004.
[19] G. Mrugalski, J. Rajski, and J. Tyszer, "Ring generators—new devices for embedded test applications," *IEEE Trans. Comput.-Aided Des.*, vol. 23, no. 9, pp. 1306–1320, Sep. 2004.
[20] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded deterministic test," *IEEE Trans. Comput.-Aided Des.*, vol. 23, no. 5, pp. 776–792, May 2004.
[21] K. J. Balakrishnan and N. A. Touba, "Improving linear test data compression," *IEEE Trans. Comput.-Aided Des.*, vol. 14, no. 11, pp. 1227–1237, Nov. 2006.
[22] I. Bayraktaroglu and A. Orailoglu, "Test volume and application time reduction through scan chain concealment," in *Proc. DAC*, 2001, pp. 151–155.
[23] D. Czysz, G. Mrugalski, N. Mukherjee, J. Rajski, and J. Tyszer, "On compaction utilizing inter and intra-correlation of unknown states," *IEEE Trans. Comput.-Aided Des.*, vol. 29, no. 1, pp. 117–126, Jan. 2010.
[24] D. Czysz, G. Mrugalski, N. Mukherjee, and J. R. J. Tyszer, "Compression based on deterministic test vector clustering of incompatible test cubes," in *Proc. ITC*, 2009, pp. 1–10.
[25] S. Reda and A. Orailoglu, "Reducing test application time through test data mutation encoding," in *Proc. DATE*, 2002, pp. 387–393.
[26] V. Tenentes, X. Kavousianos, and E. Kalligeros, "Single and variable-state-skip LFSRS: Bridging the gap between test data compression and test set embedding for IP cores," *IEEE Trans. Comput.-Aided Des.*, vol. 29, no. 10, pp. 1640–1644, Oct. 2010.
[27] G. Zeng and H. Ito, "Concurrent core test for SOC using shared test set and scan chain disable," in *Proc. DATE*, 2006, pp. 1–6.
[28] Q. Zhou and K. Balakrishnan, "Test cost reduction for SOC using a combined approach to test data compression and test scheduling," in *Proc. DATE*, 2007, pp. 1–6.
[29] C. Shi and R. Kapur, "How power-aware test improves reliability and yield," *EE Times EDA News Online*, Sep. 15, 2004.
[30] D. Czysz, M. Kassab, X. Lin, G. Mrugalski, J. Rajski, and J. Tyszer, "Low-power scan operation in test compression environment," *IEEE Trans. Comput.-Aided Des.*, vol. 28, no. 11, pp. 1742–1755, Nov. 2009.
[31] D. Czysz, G. Mrugalski, J. Rajski, and J. Tyszer, "Low-power test data application in EDT environment through decompressor freeze," *IEEE Trans. Comput.-Aided Des.*, vol. 27, no. 7, pp. 1278–1290, Jul. 2008.
[32] J. Lee and N. A. Touba, "LFSR-reseeding scheme achieving low-power dissipation during test," *IEEE Trans. Comput.-Aided Des.*, vol. 26, no. 2, pp. 396–401, Feb. 2007.
[33] D. Czysz, G. Mrugalski, N. Mukherjee, J. Rajski, P. Szczerbicki, and J. Tyszer, "Low power compression of incompatible test cubes," in *Proc. ITC*, 2010, pp. 1–10.
[34] D. Czysz, G. Mrugalski, N. Mukherjee, J. Rajski, P. Szczerbicki, and J. Tyszer, "Deterministic clustering of incompatible test cubes for higher power-aware EDT compression," *IEEE Trans. Comput.-Aided Des.*, vol. 30, no. 8, pp. 1225–1238, Aug. 2011.
[35] J. Tyszer, D. Czysz, G. Mrugalski, N. Mukherjee, and J. Rajski, "On deploying scan chains for data storage in test compression environment," *IEEE Des. Test*, Early access article, Mar. 7, 2013.
[36] IWLS'05 Circts. (2005) [Online]. Available: http://www.iwls.org/iwls2005/benchmarks.html
[37] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.
[38] A. Chandra and R. Kapur, "Bounded adjacent fill for low capture power scan testing," in *Proc. VTS*, 2008, pp. 131–138.
[39] H. Vranken, F. Hapke, S. Rogge, D. Chindamo, and E. Volkerink, "ATPG padding and ATE vector repeat per port for reducing test data volume," in *Proc. ITC*, 2003, pp. 1069–1078.
[40] S. Balatsouka, V. Tenentes, X. Kavousianos, and K. Chakrabarty, "Defect aware x-filling for low-power scan testing," in *Proc. DATE*, 2010, pp. 873–878.
[41] Z. Wang and K. Chakrabarty, "Test-quality/cost optimization using output-deviation-based reordering of test patterns," *IEEE Trans. Comput.-Aided Des.*, vol. 27, no. 2, pp. 352–365, Feb. 2008.
[42] X. Kavousianos and K. Chakrabarty, "Generation of compact test sets with high defect coverage," in *Proc. DATE*, 2009, pp. 1130–1135.
[43] X. Kavousianos, V. Tenentes, K. Chakrabarty, and E. Kalligeros, "Defect-oriented lfsr reseeding to target unmodeled defects using stuck-at test sets," *IEEE Trans. Very Large Scale Integr.*, vol. 19, no. 12, pp. 2330–2335, Dec. 2011.
[44] M. Rudberg and L. Wanhammar, "High speed pipelined parallel Huffman decoding," in *Proc. ISCAS*, 1997, pp. 2080–2083.
[45] C.-H. Lin and C.-W. Jen, "Low power parallel Huffman decoding," *Electron. Lett.*, vol. 34, no. 3, pp. 240–241, Feb. 1998.
[46] G. Mrugalski, J. Rajski, D. Czysz, and J. Tyszer, "New test data decompressor for low power applications," in *Proc. DAC*, 2007, pp. 539–544.
[47] L. Li, K. Chakrabarty, S. Kajihara, and S. Swaminathan, "Efficient space/time compression to reduce test data volume and testing time for IP cores," in *Proc. ICVD*, 2005, pp. 53–58.
[48] Z. Wang, H. Fang, K. Chakrabarty, and M. Bienek, "Deviation-based lfsr reseeding for test-data compression," *IEEE Trans. Comput.-Aided Des.*, vol. 28, no. 2, pp. 259–271, Feb. 2009.

Vasileios Tenentes (S'07) received the Bachelor degree in computer science from the University of Piraeus, Piraeus, Greece, in 2003, and the M.S. degree from the Department of Computer Science, University of Ioannina, Ioannina, Greece, in 2007. He is currently pursuing the Ph.D. degree in embedded testing architectures from the same university.

His current research interests include test data compression, low-power architectures and testing, interactive distributed optimization, and computational geometry algorithms.

Xrysovalantis Kavousianos (S'97–M'02) received the Diploma from the Department of Computer Engineering and Informatics, University of Patras, Patras, Greece, in 1996, and the Ph.D. degree from the same university in 2000.

He is currently an Assistant Professor of computer science at the University of Ioannina, Ioannina, Greece. His current research interests include testing, design-for-testability of integrated circuits, low-power design and testing, and on-line testing. He has published over 65 papers in journals and refereed conference proceedings.