Compression Based on Deterministic Vector Clustering of Incompatible Test Cubes

G. Mrugalski, N. Mukherjee, J. Rajski
Mentor Graphics Corporation
Wilsonville, OR 97070, USA

D. Czysz, J. Tyszer
University of Technology
60-965 Poznań, Poland

Abstract—The presented compression scheme is a novel solution that is based on deterministic vector clustering and encompasses three data reduction features in one on-chip decoding system. The approach preserves all benefits of continuous flow decompression and offers compression ratios of order 1000x with encoding efficiency much higher than 1.00.

1. Introduction

New test technologies witness increasing demands for high quality test driven by ever-expanding reliability needs combined with the ability to handle more complex and diverse designs. As circuits grow in size, however, it becomes increasingly expensive to maintain high level of test coverage. This is due to prohibitively large volumes of test data and long test application times. A method employed to reduce the amount of test data is, therefore, instrumental in maintaining the overall high efficiency of a testing scheme. On-chip test compression has already established itself as a mainstream DFT methodology [48] with a direct bearing on the manufacturing test cost.

Majority of existing test compression schemes, including the original idea known as LFSR coding [25], take advantage of low test cube fill rates. They treat external test data as Boolean variables used to create linear expressions filling conceptually all scan cells. Test patterns are then delivered, in a compressed form, by using tester channels to an on-chip LFSR (decoder) which expands them into the actual data loaded into internal scan chains. The LFSR coding was refined under the name of static LFSR reseeding in a number of approaches [17], [28], [31], [43], [58]. Typically, the encoding capability of these methods is limited by the LFSR size, unless a group of scan slices is encoded per seed [57], [29], [53]. Another drawback is that the loading of the seed and loading/unloading of the scan chains are done in two non-overlapping phases, resulting in inefficient utilization of the tester. This can be alleviated by employing additional shadow registers [52], [57]. One can also modify patterns produced by the LFSR to increase test coverage [10], [49], [54] while reducing the number of seeds [59]. Finally, as the number of specified bits may vary in successive test cubes, variable-length seeds were deployed to improve the encoding efficiency of the conventional static reseeding schemes [42].

In principle, static LFSR reseeding computes a seed for each test cube. Dynamic reseeding stands in a vivid contrast against its static counterpart. It continuously injects free variables into a decompressor as it loads the scan chains. This shift in compression paradigm, as proposed for the embedded deterministic test (EDT) [40], reduces test cost by providing one to two orders of magnitude reduction in scan test data volume and scan test time. Such a scheme is widely applicable and easy to deploy because it is based on the standard scan/ATPG methodology with a very simple flow. Reduction of both tester scan buffer data volume and scan test time is also accomplished in the schemes based on the On-Product MISR [1], [24], [27], where a bandwidth overhead for external test response compaction is reduced by on-chip signature generation while still supplying test stimuli from ATE.

Both test application time and test data volume can be reduced by broadcasting the same test vector to several scan chains through a single input as presented in [32], and then adopted by Illinois scan [14], [15], [19]. Here, a given test pattern must not contain contradictory values on corresponding cells in different chains loaded through the same input. This can be guaranteed by incorporating the compression-imposed constraints during test generation, e.g., by tying dependent inputs together in the circuit description provided to ATPG.

Other forms of compression are based on XOR networks [3], [4], [5], [30], [36], hybrid patterns [8], folding counters [18], [35], bit flipping [16], [55], non-linear decompressors [33], reconfigurable networks [46], and reuse of scan chains [9]. A Star-BIST reduces the amount of data by using clusters of correlated test patterns. A test generator stores a small number of vectors which serve as centers of clusters, and it applies each center test vector multiple times. Every time the center vector is shifted into the circuit, some of its positions are randomly [50] or deterministically [16] complemented. A separate group of techniques uses various forms of compression based on run-length [6], [11], [20], [21], [23], statistical [11], [20], [21], [22], constructive [52], and Golomb [7] codes. Though usable on any set of test cubes, the code-based methods are less than efficient at exploiting test cube low fill rates.

Test compression schemes are typically characterized by their encoding efficiency, i.e., a ratio of successfully encoded specified bits to the total number of deployed data bits. In particular, the encoding efficiency of plain reseeding-based schemes can reach, at their best, the value of 1.00 [2]. The encoding efficiency can be increased either by repeating certain patterns at the rates, which are adjusted to the requirements of test cubes [41], or by embedding certain bits into LFSR-generated sequences [12]. The latter technique requires additional memory to store compressed information necessary to handle separately test cubes that otherwise would require excessively large LFSRs to be encoded.
Further increase in the encoding efficiency is possible due to regularities occurring in test patterns. Dictionary-based approaches [34], [44], [45], [56] exploit the fact that certain vector values within test patterns are likely to be repeated. When they do occur, they may be encoded as pointers to on-chip memory locations storing vectors reused during test. The similar concept is used in packet-based encoding [57] and nine-coded compression [38], [47]. The recently proposed restrict encoding [13] combines LFSR reseeding with a pre-computed dictionary of test vector values which are injected at the same positions over multiple test patterns. The approach requires solving the clique covering and traveling salesman problems, though.

A new test data compression scheme presented in this paper further explores the occurrence of similar vectors in test stimuli. With this technique it is not necessary to resort to any form of dictionaries. Instead, its goal is to see that test cubes that feature many similar specified bits are merged and EDT-compressed even in the presence of conflicts, whose particulars are encoded efficiently, as well. As a result, our solution offers very high compression ratios, elevates the encoding efficiency well above the threshold of 1.00, and preserves all benefits of continuous flow decompression [40].

2. Motivation

Typically, the EDT-based compression uses cube merging to reduce a pattern count, and hence the ATE time and the amount of test data. It gradually expands a test pattern by incorporating successive compatible test cubes with appropriate values assigned to unspecified positions. Two cubes are compatible if in every position where one of the cubes has a value of 0 or 1, the other one either features the same value or don’t care. With the concerns over test compression and test application time, one needs to maximize the number of test cubes forming a given test pattern. Unfortunately, the cube merging process terminates when either no new compatible cube can be found in a reasonable amount of time, or the resultant pattern cannot be encoded anymore.

The encoding efficiency and the compression ratio could be significantly increased, if the cube merging process continued despite conflicts on certain positions. Clearly, such bits would not be the subject of the mainstream EDT-based compression, but they would have to be recovered in a different way to eventually load scan chains with patterns that feature the original test cubes. Furthermore, their quantity should be selected so as to not adversely affect the compression ratio. It is worth noting that B. Koenemann uses test cubes merging with conflicts for hybrid pseudo-random patterns [26]. A number of conflicting bits is small enough so that they can be assigned pseudo-random values to restore the required test cubes with a high probability and within a certain number of trial vectors. A.-W. Hakmi et al. [12] discuss somehow related scenario for a static LFSR reseeding-based compression. With that scheme, specified bits leading to an inconsistent set of equations are ignored and embedded by using a special flip vector generator. Our approach involves a deliberate attempt to increase the overall compression by merging a large number of cubes having a pre-determined number of mutually conflicting locations at most.

Consider, as a feasibility study, experiments in which test patterns are obtained by merging incompatible test cubes that were generated for several industrial designs. We will refer to such test vectors as parent patterns [50]. Accordingly, the objective is to estimate the amount of data that would become the subject of encoding assuming that a given test cube is regarded mergeable as long as the number of conflicts it causes is less than a pre-specified limit. Results shown in Table I provide the following information for each test case:

- the initial number of test cubes and the total number of their specified bits,
- the number of (arbitrarily) allowed conflicts between otherwise mergeable parent patterns and test cubes,
- the resultant number of parent patterns and the total number of their specified bits,
- the total number of incremental bits, i.e., bits whose values must be superposed on the parent patterns because of earlier cube merging conflicts.

| Design | Test cubes | Specified bits | Allowed conflicts | Parent patterns | Specified bits in parents | Incremental bits | Incremental bit sites | EDT-based | Ratio |
|--------|------------|----------------|-------------------|----------------|---------------------------|-----------------|-----------------------|------------|
| D1     | 5,453      | 180,606        | 8                 | 90             | 16,347                    | 36,212          | 3,373                 | 171,192    | 2.37 |
|        |            |                | 16                | 19             | 6,987                     | 68,272          | 2,593                 | 1,408,878  | 2.02 |
| D2     | 24,818     | 1,443,818      | 8                 | 312            | 239,828                   | 172,008         | 24,966                | 1,728,581  | 2.04 |
|        |            |                | 16                | 136            | 121,043                   | 317,001         | 25,237                | 1,728,581  | 2.04 |
| D3     | 28,732     | 1,736,264      | 16                | 64             | 145,134                   | 395315          | 33,994                | 1,833,154  | 2.39 |
| D4     | 30,191     | 5,151,031      | 16                | 77             | 233,913                   | 393,894         | 18,536                | 5,127,107  | 5.82 |
|        |            |                | 32                | 35             | 112,986                   | 615,701         | 17,143                | 6,084,081  | 3.62 |
| D5     | 49,135     | 6,154,237      | 8                 | 3627           | 639,910                   | 366,143         | 35,227                | 6,084,081  | 4.33 |
|        |            |                | 16                | 1816           | 324,896                   | 718,599         | 35,548                | 1,833,154  | 2.39 |
| D6     | 33,825     | 1,846,202      | 16                | 466            | 149,305                   | 449,335         | 20,283                | 1,833,154  | 2.04 |
3. Cube merging and compression

We now discuss the new compression scheme in more detail. As mentioned earlier, it is based on an atypical form of vector clustering which merges groups of deterministic test cubes while allowing some degree of their incompatibility. Consequently, each cluster contains one parent pattern and a number of its derivatives obtained by imposing some extra bits on the parent pattern. Clearly, there is no need to store the entire clusters. Instead, one has to keep some extra bits on the parent pattern. Clearly, there is no need to store the entire clusters. Instead, one has to keep some extra bits on the parent pattern. These extra bits are counted twice, as they correspond to sites that should not be affected by the process of embedding incremental bits, and such information must also be taken into account. The content of the last column clearly indicates that for a certain number of allowed conflicts, the amount of bits that would become the subject of encoding is a fraction of data that the conventional EDT would need to handle. Indeed, the ratio between specified bits that EDT has to encode and the corresponding data for parent patterns and their derivates varies from 2.02 to 5.97. Much the same can be said about other designs examined in similar experiments not reported here. Note that the results gathered in Table I do not represent the actual test data needed to encode all test relevant information. However, by assuming in both cases the same encoding efficiency, these numbers may serve as a baseline in estimating expected compression gains provided an appropriate form of data encoding is employed.

The compression algorithm involves three major steps.
1. Given degree $\delta$ of allowed conflicts, determine the set of parent patterns and all incremental bits associated with every parent pattern.
2. Determine a single control pattern and all incremental patterns for each parent pattern.
3. Encode the parent patterns, the control patterns, and the incremental patterns. This tri-modal encoding process is further described in Section 5.

Bits of the parent pattern may assume one of the following four values: 0, 1, $x$, or $C$. Clearly, the Boolean values of 0 and 1 represent the specified bits assuming these exact values. $x$ is don’t care, while $C$ is assigned to positions on which at least one conflict has been recorded. A conflict occurs in two cases: (1) two corresponding bits have the opposite specified values, (2) one bit is specified while the other is already labeled as $C$.

The first step of the algorithm is essentially a cube merging process, and it proceeds as follows. Pick a test cube to instantiate a parent pattern $P$. For every remaining test cube $\tau$ determine the number of specified bits that this cube has in common with the parent pattern. Also, check $P$ and $\tau$ to determine if their number of conflicting bits is greater than $\delta$. If it is, delete $\tau$ from the list of cubes that
could be merged with \( P \). Moreover, check all test cubes that have already been integrated with \( P \) to determine the number of conflicting bits they would have with \( P \), if \( \tau \) was a part of \( P \). Again, if at least one of these numbers is greater than \( \delta \), delete \( \tau \) from the list.

Subsequently, scan the entire list of candidate test cubes and pick a few of them with the highest degree of similarity with the current parent pattern (the largest number of common specified bits). Merge the first selected test cube with \( P \) and try to encode a newer version of \( P \) with \( C \)-bits being ignored. If the encoding fails, verify whether \( P \) can be compressed with some of newly added specified bits discarded and treated as conflicting bits. This can only be done provided the total number of conflicts does not exceed \( \delta \). Otherwise, take the next candidate from the short list. The process of forming the parent pattern terminates when none of the short-listed test cubes can be used. If a test cube is successfully added to the parent pattern, all statistics regarding the remaining test cubes are updated and the process continues until no test cubes can be chosen due to incompatibility constrains imposed by factor \( \delta \). One of the unmerged test cubes becomes now an initial form of a new parent pattern, and the algorithm proceeds as shown above. The process finally terminates when no new parent patterns can be formed.

**Example.** Consider the following parent pattern:

\[
P = \{x_{0} x_{1} x_{3} x_{4} x_{5} x_{6} x_{7} x_{8} x_{9} x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}\}
\]

This particular parent pattern was obtained after merging the following three test cubes:

1. \( x_{0} x_{1} x_{3} x_{4} x_{5} x_{6} x_{7} x_{8} x_{9} x_{10} x_{11} = 0 \)
2. \( x_{0} x_{1} x_{3} x_{4} x_{5} x_{6} x_{7} x_{8} x_{9} x_{10} x_{11} = 1 \)
3. \( x_{0} x_{1} x_{3} x_{4} x_{5} x_{6} x_{7} x_{8} x_{9} x_{10} x_{11} = 0 \)

Let the allowed level of conflicts be two. The process of merging considers now other six test cubes as possible candidates for merging. They are listed below. Each test cube is accompanied by its degree of compatibility \( \chi \) (the number of common specified bits with the parent pattern) as well as the number \( \Delta \) of conflicts with the parent pattern:

\[
\begin{array}{c|c|c|c}
\chi & \Delta \\
1 & 0 & 2 \\
2 & 2 & 0 \\
3 & 1 & 2 \\
4 & 3 & 1 \\
5 & 0 & 3 \\
6 & 3 & 2 \\
\end{array}
\]

The fifth test cube must be rejected as its number of conflicts with the parent pattern (in this particular case 3) is too high. The best candidate seems to be the fourth test cube. It has the same three specified bits as the parent pattern. Also, it features only one conflict. If merged, it would increase the number of conflicts for one of already merged cubes by one, so this is acceptable. It remains to be seen, however, whether the new parent pattern is compressible.

If so, the new parent pattern is accepted, statistics for the remaining test cubes are updated, and the process continues.

Once all parent patterns and the incremental bits are known, we can determine the corresponding control data. Let \( p_i \) and \( c_i \) denote the \( i \)-th bit of the parent pattern and the control pattern, respectively. Bits of the control pattern are then defined as follows:

\[
p_i = 0 \text{ or } p_i = 1 \Rightarrow c_i = 0, \\
p_i = C \Rightarrow c_i = 1, \\
p_i = x \Rightarrow c_i = x.
\]

The control pattern assumes the value of 0 every time the parent pattern features a specified value. Its value of 1 indicates that the parent pattern has a conflicting bit, and therefore the test data should be provided by the incremental pattern. This pattern, in turn, features the specified bits of 0 and 1 on some of the positions where \( p_i = C \).

**Example.** Let the parent pattern be as follows:

\[
\begin{align*}
x_{0} x_{1} x_{3} x_{4} x_{5} x_{6} x_{7} x_{8} x_{9} x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} &= 0 \\
x_{0} x_{1} x_{3} x_{4} x_{5} x_{6} x_{7} x_{8} x_{9} x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} &= 1 \\
\end{align*}
\]

The corresponding control pattern is then of the form:

\[
\begin{align*}
x_{0} x_{1} x_{3} x_{4} x_{5} x_{6} x_{7} x_{8} x_{9} x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} &= 0 \\
x_{0} x_{1} x_{3} x_{4} x_{5} x_{6} x_{7} x_{8} x_{9} x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} &= 1 \\
\end{align*}
\]

One of the incremental patterns may look like the following vector (the specified bits are chosen here arbitrarily just for the sake of illustration; note also that only a subset of \( C \)-bits is typically covered by a single incremental pattern as the control pattern serves all incremental patterns associated with a given parent pattern):

\[
\begin{align*}
x_{0} x_{1} x_{3} x_{4} x_{5} x_{6} x_{7} x_{8} x_{9} x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} &= 0 \\
x_{0} x_{1} x_{3} x_{4} x_{5} x_{6} x_{7} x_{8} x_{9} x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} &= 1 \\
\end{align*}
\]

It is worth noting that test application time in the proposed method depends on how many times each parent pattern is applied. This is determined by the number of test cubes that have been merged with a given parent pattern despite conflicts (the compatible test cubes do not require any further action besides just a single application of their parent pattern). Interestingly, compatible incremental vectors of the same parent pattern can be merged in an attempt to further reduce test application time. This can be easily achieved by inspecting successive lists of incremental patterns to determine maximal sets of pair-wise compatible vectors. One has to make sure, however, that the combined incremental patterns can be still encoded by a method adopted to handle these particular vectors.
4. Decompressor architecture

Fig. 2 shows the general structure of a new test data decompressor unit employing the approach presented in Section 3. Essentially, it comprises three modules designated to decompress the parent patterns, the incremental patterns, and the control patterns, respectively. In principle, all decomposition techniques adopted here follow the EDT-based continuous flow decompression [40], [41] modus operandi. However, due to particulars characterizing patterns this tri-modal decompressor is to handle, each block is designed in a slightly different way, as described below.

As can be seen, a 2-input multiplexer is placed in the front of each scan chain. These devices are intended to route decompressed test data from one of two sources: (1) the basic EDT-like decompressor that primarily handles the parent patterns, (2) another decompressor that consists of ring generator 2 with phase shifter 2, which takes care of the incremental patterns. The actual source is determined by applying appropriate select signals to the multiplexers. These signals are provided by the third decompression logic employed here to handle the control patterns.

Ring generator 1 and phase shifter 1 constitute the first part of our new decompression logic. This is virtually a sequential continuous flow decompressor whose logic synthesis and corresponding compression algorithms have already been presented elsewhere [37], [39], [41]. As mentioned earlier, this part of decompressor is deployed to decode the parent patterns which are subsequently fed to the scan chains unless the control signals decide otherwise.

As demonstrated in Section 3, the control patterns feature a relatively large number of zeros (corresponding to the specified bits of the parent patterns) with the sparse presence of ones (corresponding to C-bits). To encode such patterns it suffices to target only a small subset of 0-bits as long as they occur in sequences no interspersed with 1-bits. For example, a sequence

\[ x \quad x \quad 0 \quad x \quad x \quad x \quad 0 \quad x \quad 0 \quad x \quad x \quad x \quad 0 \quad x \quad x \quad x \quad x \quad 1 \quad x \quad 0 \quad x \quad x \quad 0 \quad x \]

• • •

• • •

can be efficiently encoded by taking into account only indicated bits, and by assuming that the decompressor outputs are sustained for more than a single clock cycle to deliver the identical test data (here to the multiplexer select inputs) for a number of shift cycles. With such a mechanism, the number of specified bits that need to be encoded is drastically reduced. The same rule applies to 1-bits, though their continuous appearance in control patterns is less frequent.

In order to implement the above idea, we use a shadow register to keep the decompressor outputs unchanged. It is placed between ring generator 1 and phase shifter 3. The shadow register captures and saves, for a number of cycles, a desired state of ring generator 1, while the generator itself keeps advancing to the next states in order to encode upcoming specified bits. Note that we reuse ring generator 1 as its encoding capabilities are sufficient to handle very low control pattern fill rates.

Interestingly, the decompressor input channels also facilitate the operation of the shadow register by using seed variables to deliver a load enable signal, as shown in Fig. 2. Small buffers placed in parallel with the decompressor inputs drive an XOR tree which computes a parity of input variables, including not only data currently entering the ring generator, but also those used in previous cycles. If this signal is 1, then the shadow register is reloaded with the current content of ring generator 1 before new seed variables enter the generator (and it reaches its next state). Otherwise, the content of the register remains unchanged.

The third part of the decompressor is used to decode the incremental patterns. It consists of ring generator 2 and phase shifter 2. As the incremental test patterns feature extremely sparse specified bits (corresponding to C-bits), they do not require all variables, which would be injected within the conventional framework of EDT scheme. Hence, injections of new test data occur regularly in rare and predetermined scan shift cycles only. Such an approach further elevates the compression ratio and adheres conveniently to the ATE ability of repeating the same patterns multiple times. Alternatively, every set of new variables can be injected once, and then buffered, for a requested number of cycles, as shown at the top of Fig. 2.

A decompressor comprising three separate modules has also been proposed in [51]. This scheme combines a 3-weight pseudorandom testing with LFSR reseeding. It em-
ploys a pseudo-random test pattern generator to feed scan chains and handle conflicts. Moreover, if certain test cubes feature the same specified bits, then such bits are provided by additional two LFSRs that produce a value and indicate a scan shift cycle when this value should replace data arriving from the PRPG. Both LFSRs have a reseeding ability and are controlled by an external tester.

In order to reduce the ATE channels bandwidth, compressed parent patterns can be delivered to a circuit under test only once, and then stored on chip by deploying either a small memory or certain scan chains acting as circular buffers. The second approach takes advantage of low fill rates allowing several scan chains to be specified bits-free, and, moreover, not affected by fault propagation sites. Indeed, many experimental evidences indicate that a small number of scan chains suffice to accommodate all compressed parent and control patterns. Once the parent pattern is stored, the bandwidth requirements scale down as the incremental patterns need only a fraction of the original tester interface throughput. Possible underutilization of external channels opens new research directions that one may explore in order to further enhance the overall efficiency of deterministic test data compression.

5. Encoding algorithm

The decompressor architecture is tightly coupled with the encoding algorithm, where all specified bits are represented by linear functions of variables injected into the decompressor. A compressed pattern is then determined by solving the system of linear equations in the Galois field modulo 2. In particular, this technique applies to the parent patterns, whose corresponding equations are formed based on a feedback polynomial implemented by ring generator 1 and structure of phase shifter 1. Recall that C-bits occurring in the parent patterns are not the subject of encoding.

As mentioned earlier, we employ a shadow register to address a crucial feature the control patterns exhibit: 0-bits are predominant specified values while 1-bits occur sparsely. This property is a key factor in reducing the volume of test data as one may deliver the identical data to the multiplexers for a number of scan shift cycles. Consequently, the encoding algorithm partitions a given control pattern into several blocks comprising certain number of consecutive slices such that there are no scan chains with both 0’s and 1’s inside the same block. This allows one to repeat a given combination many times in succession by using the shadow register storing a state that the ring generator entered at the beginning of a block. Such a technique gives the ring generator enough time to compensate for fading encoding ability by collecting new input variables. They will facilitate successful compression during next steps once the shadow register is reloaded.

The ability of a decompressor to decode data within boundaries of the block determines its size. Hence, the following four rules govern the encoding process:

- it begins with a block and the corresponding state of a ring generator which should be applied first, and it gradually moves towards the end of a control pattern,
- whenever specified bits of the same value are repeated many times in succession (possibly interspersed with don’t care bits) within the same block and the same scan chain, there is no need to encode all of them but the first one,
- for each scan shift cycle there is an associated equation representing a request to store, if needed, the content of the ring generator in the shadow register before new variables change the state of the generator during the next cycle,
- an equation representing the first specified bit in the current block and in a given scan chain is expressed in terms of variables injected until this block.

The last rule is needed as a ring generator state which is to cover a given bit has to be completely determined before it is moved to the shadow register during the first cycle of a new block. This is equivalent to conceptually moving this specified bit to the beginning of the block. As long as such bits can be encoded, the algorithm works by repeatedly increasing the size of the block, and by adding new equations, if necessary. At some point a solution may not exist anymore. This particular time frame is then assigned a new block, and the procedure continues.

Example. Consider a 2-input decompressor employing a shadow register controlled by 4-input XOR gate whose inputs comprise the last two variables injected through each input of the ring generator (Fig. 2). Suppose we need to encode a control pattern shown in Fig. 3a (white dots denote 0’s while black dots correspond to 1-bits; bits to be considered first are located on the right). The input variables $a_0, b_0, a_1, b_1, ...$ are provided in pairs. Fig. 3b illustrates a hypothetical partition of the control pattern into blocks with those specified bits that need to be encoded highlighted. By relocating these specified bits to the border lines of the blocks, we get locations corresponding to linear equations that will be used to perform actual encoding (Fig. 3c). Note that a new block is formed (with the shadow register reloaded) anytime a given scan frame has a conflict with the previous slice. Furthermore, one has to remember that a set of linear equations used by the compression procedure must include one additional equation per every scan slice (time frame) of the following form:

$$a_k + b_k + a_{k-1} + b_{k-1} = r,$$

where $r$ is equal to 1, if the shadow register is to be reloaded during the $k$-th scan shift cycle, and it assumes the value of 0, otherwise. In general, the number of variables used in the above equation depends on the number of decompressor external channels and the size of input buffers, as shown in Fig. 2.
two tester channels drive a decompressor. Consider the following equation:

\[ a_0 + b_0 + a_1 + a_2 + b_3 = 1. \]

If the algorithm is to replace variables injected during the second and fourth steps (i.e., \( a_1, b_1, a_3, \) and \( b_3 \)) with variables injected during the first and third cycles (i.e., \( a_0, b_0, a_2, \) and \( b_2 \)), then we clearly have:

\[ a_0 + b_0 + a_0 + a_2 + b_2 = b_0 + a_2 + b_2 = 1. \]

To accomplish this action in a time-efficient manner, we use an array of bits to represent variables. The part of each equation corresponding to omitted variables is then added modulo 2, in a bit-wise fashion, to the replacement part. If we continue this way until all variables are examined, the resulting equation will have exactly the same form as the one obtained by symbolic simulation of the decompressor with certain variables repeatedly injected several times. The eliminated variables are eventually replaced with 0s.

6. Experimental results

The proposed compression scheme was tested on several industrial designs. This section reports results for some of them, ranging in size from 220K to 2.2M gates. The basic data regarding the designs: the number of gates and number of scan cells are listed in the left part of Table II. All reported experiments were performed for both stuck-at tests and launch-off-capture transition tests. Consequently, the next two entries to the table report the resultant number of test patterns and the total number of specified bits these patterns feature. Finally, for the sake of reference, the best results obtained for the conventional EDT-based compression scheme are reported in the last two columns. They include the amount of compressed test data and the resultant test data volume compression.

<table>
<thead>
<tr>
<th>Design</th>
<th>Gates</th>
<th>Scan cells</th>
<th>Test patterns</th>
<th>Specified bits</th>
<th>EDT test data</th>
<th>EDT compress</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>220K</td>
<td>13K</td>
<td>1,573</td>
<td>171,192</td>
<td>201,344</td>
<td>98</td>
</tr>
<tr>
<td>D2</td>
<td>545K</td>
<td>45K</td>
<td>3,223</td>
<td>1,408,878</td>
<td>2,301,222</td>
<td>63</td>
</tr>
<tr>
<td>D3</td>
<td>1.3M</td>
<td>86K</td>
<td>4,214</td>
<td>1,728,581</td>
<td>2,098,572</td>
<td>174</td>
</tr>
<tr>
<td>D4</td>
<td>1.1M</td>
<td>70K</td>
<td>10,466</td>
<td>5,127,107</td>
<td>19,048,120</td>
<td>39</td>
</tr>
<tr>
<td>D5</td>
<td>260K</td>
<td>14K</td>
<td>37,050</td>
<td>6,084,081</td>
<td>10,448,100</td>
<td>50</td>
</tr>
<tr>
<td>D6</td>
<td>2.2M</td>
<td>144K</td>
<td>24,137</td>
<td>1,833,154</td>
<td>26,792,070</td>
<td>129</td>
</tr>
</tbody>
</table>

The results of the experiments are summarized in Table III. The first part of the table consists of three columns, each entry of which specifies the scan configuration and the decompressor architecture. Note that columns “Ring 1” and “Ring 2” are used to designate the size of ring generator 1 and ring generator 2, respectively. Information regarding the number of inputs, i.e., the number of decompressor channels connecting ATE with both ring generators, as shown in Fig. 2, is given in the same columns.

The remaining parts of Table III list various performance-related statistics of the proposed compression.
scheme. In particular, for each test case the following information is provided:

- the maximal number of allowed conflicts when merging original test cubes,
- the number of parent and incremental patterns (recall that the number of control patterns is the same as that of the parent patterns),
- the total number of bits that have to be encoded; this quantity includes the total number of specified bits occurring in the parent patterns, the total number of incremental bits as well as the total number of control bits – the latter value is equal to the number of parent specified bits plus the number of all incremental bit sites as seen by a given parent pattern (compare a detailed breakdown of such data presented in Table I),
- the total number of bits encoding parent, and control patterns, and, separately, incremental patterns,
- the total test data volume used to encode all specified bits listed above,
- the encoding efficiency of the tri-modal scheme, i.e., the ratio of the total number of specified bits as presented (for each design) in Table I and the number of bits (test data) reported in the previous column,
- the effective test data volume compression resulting from using our scheme; this quantity is obtained as a ratio of the following two values: the number of scan cells multiplied by the number of test patterns and the amount of test data reported in column “Total test data” of Table III.

As indicated by data in the last column of Table III, application of our scheme produces remarkable compression levels. In all examined test cases, the proposed solution yields significantly higher compression than that of the conventional EDT-based scheme. The observed increase in test data compression varies from 3.5 to almost 36 times (or well above 4,500x in absolute numbers). Note that the test coverage stays unaffected in each case while keeping the required added test logic on the circuit under test to a minimum. In particular, it appears that a single scan channel suffices in all test cases to operate the decompressor designated to handle incremental patterns.

The proposed approach compares favorably to earlier test data reduction schemes as far as compression and encoding efficiency are concerned. For instance, the “restrict encoding” scheme of [12] is capable of delivering 1.58 coding efficiency on the average with the corresponding maximum value reaching level of 2.0. As shown in Table III, the average encoding efficiency computed across industrial designs examined in this work is equal to 4.08 with the maximal efficiency elevated to the value of 7.93. Clearly, these numbers are much higher than those of conventional static and dynamic reseeding-based compression schemes. In fact, to the best of our knowledge, this is the first test data volume compression scheme achieving such extreme encoding efficiency. Deterministic encoding, moreover, allows the proposed method to handle any number of conflicts (provided it does not compromise compression), as opposed to simply applying pseudo-random patterns to conflicting bits as done in [26] at the cost of significantly increased test application time.

For each design, there exists a “sweet spot” where a combination of several factors, including primarily the number of allowed conflicting bits, the resultant number of parent patterns, incremental bits as well as their locations,
leads to a particularly suitable solution in terms of encoding efficiency and compression levels. Indeed, as one may expect, with the increasing number of conflicts, the number of parent patterns decreases. This trend is clearly counterbalanced by the increase in incremental bits (and then patterns). Both quantities (plus the corresponding control patterns) must eventually be encoded by using certain amounts of test data in each case. Their sum can be regarded as a cost function. By proper selection of the maximal number of conflicting bits, the decompressor size (mainly ring generator 1), and the number of scan channels, one can search for optimal sets of test data such that the cost function is minimized, or alternatively the compression is maximized.

It is worth noting that information listed in column “Incremental patterns” essentially estimates test application time. Recall that initial incremental patterns obtained in parallel with the process of forming successive parent patterns can be further merged provided they are mutually compatible. Eventually, tests are delivered by repeatedly applying successive parent patterns, every time using a next incremental pattern, i.e., a derivative of the current parent. The incremental pattern count is therefore indicative of the expected test time. Interestingly, test application time in the case of designs D3, D4, D5, and D6 is either virtually the same as that of the conventional EDT or even shorter (see the number of test patterns in Table II).

7. Conclusion

One of the key requirements for the future compression scheme is to further and significantly reduce the amount of test data. With the size of test patterns growing at alarming rates, the new approach presented in this paper provides a coherent way to generate and apply extremely compressed test patterns in a cost effective manner, for any fault model used today and for any fault model or defect-based testing scenario of the future. Similarly to the EDT scheme, the proposed scheme is non-intrusive as it requires no modifications to the core logic, such as test points or X-bounding logic. And still, it offers a substantial increase in coding efficiency. Moreover, it integrates seamlessly with a test logic synthesis flow and fits into the core-based design paradigm, where only a few external pins can be employed to drive a large number of internal scan chains in the cores. Finally, as this new compression preserves all benefits of continuous flow decompression, it provides a smooth migration path to the next-generation test data compression solutions while maintaining very high quality of test and minimizing design and manufacturing cost impacts.

8. Acknowledgment

The work of D. Czysz and J. Tyszer was supported in part by the SRC under Grant 2007-TJ-1584.

9. References


