Reconfigurable Broadcast Scan Compression Using Relaxation Based Test Vector Decomposition

AIMAN H. EL-MALEH, MUSTAFA IMRAN ALI

AHMAD A. AL-YAMANI

aimane@kfupm.edu.sa, mustafa@ccse.kfupm.edu.sa, alyamani@kfupm.edu.sa

King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

Abstract

An effective reconfigurable broadcast scan compression scheme that employs test set partitioning and relaxation-based test vector decomposition is proposed. Given a constraint on the number of tester channels, the technique classifies the test set into acceptable and bottleneck vectors. The bottleneck vectors are then decomposed into a set of vectors that meet the given constraint. The acceptable and decomposed test vectors are partitioned into the smallest number of partitions while satisfying the tester channels constraint to reduce the decompressor area. Thus, the technique by construction satisfies a given tester channels constraint at the expense of increased test vector count and number of partitions, offering a tradeoff between test compression, test application time and test decompression circuitry area. Experimental results demonstrate that the proposed technique achieves better compression ratios compared to other test compression techniques.

1. Introduction

The increasing level of integration continues to add to the complexity of IC test. Scan-based test is the most widely used solution to achieve high levels of fault coverage in today's complex designs. However, scan-based test by itself does not address the increasing test data volume, test application time and power consumption problems. Upgrading the Automatic Test Equipments (ATE) with greater memory and channel capacity to cope with increasing test data volume is an expensive solution. Test data compression has therefore been gaining significant momentum to address the increasing cost of manufacturing test by reducing memory and channel capacity requirements on the ATE.

Many test compression techniques have been proposed in the literature, including some by major EDA tool vendors [19].

The variety of proposed approaches can be broadly classified into [36]: (i) code-based schemes [14, 35, 29, 17, 6, 9, 37], (ii) schemes using combinational or sequential linear decompressors [1, 22, 18], and (iii) broadcast scan schemes using static [34, 30, 13], dynamic [33] and no reconfiguration [31]. Although existing commercial tools incorporate the latter two categories [36], each technique has its pros and cons, offering different solutions in terms of area overhead, test application time, test volume reduction, and encoding complexity [33]. Characteristics such as dependence on structural information of the core-under-test (CUT) and whether the decoder hardware is test data dependent or independent, are also differentiating factors among the various techniques.

This work presents a test vector compression scheme based on reconfigurable broadcast scan approach in which N scan chains are driven using M tester channels ($N \gg M$). Using compatibility analysis [3], test vectors are classified into '*acceptable*' and '*bottleneck*' vectors. Acceptable vectors are those that can be driven by M tester channels while bottleneck vectors are those that cannot be driven by M channels. Acceptable vectors are partitioned into the smallest number of partitions in such a way that all test vectors in a partition can be driven by M tester channels. Each partition corresponds to a test configuration and thus minimizing the number of partitions reduces the decoder complexity. Bottleneck vectors are decomposed into a small subset of test vectors each satisfying the tester channel constraint M, using an efficient relaxation-based test vector decomposition technique [7]. Then, the decomposed test vectors are partitioned minimizing the number of partitions. The advantages of the proposed technique include:

1. ATPG independence,
2. being able to utilize compacted test sets to achieve high compression with relatively low test vector counts, and
3. maximizing compression under area overhead constraints and minimizing area overhead under compression ratio constraints.

This paper is organized as follows. In Section 2, a review of similar schemes is given and the proposed work is put into context. Section 3 explains the underlying concepts before the proposed compression scheme is presented in Section 4. Experiments and comparisons are given in Section 6 and Section 7 concludes with a brief mention of future work.

2. Related Work

Since the publication of the Illinois Scan Architecture (ILS) [11], a number of broadcast scan-based schemes have been proposed, utilizing multiple tester channels with reconfigurable networks [34, 30, 13, 33]. These schemes achieve greater compression than ILS through increased scan chains compatibility and by completely avoiding the serial mode of loading. A configuration is defined as a specific set of connections between the tester channels and internal scan chains and reconfigurability allows changing these connections. Reconfiguration can be static or dynamic [33, 36]. In static reconfiguration,

the configuration changes between test cubes or after a multiple of them, i.e., reconfiguration occurs on a per scan basis at most. Dynamic reconfiguration can even occur on a per shift basis. This work uses a static reconfiguration approach. Static reconfigurable networks have also been used by some other recently proposed schemes. [30] places multiplexers at the scan chain inputs to reconfigure the set of scan chains that each tester channel broadcasts to. It uses fault set partitioning as a means of identifying configurations. In [23], a reconfigurable interconnection network (RIN) is placed between the outputs of a pseudorandom pattern generator and the scan inputs of the circuit under test (CUT). Configurations are derived based on analysis of LFSR outputs. Omega networks are used in [34] to allow CUT-independent design of the reconfiguration network. [13] uses a reconfigurable MUX network and an iterative ATPG-based procedure for generating the test patterns during the compression.

The proposed technique uses direct compatibility [12] to compress multiple scan chains test data via input width reduction [3, 2] and tries to overcome the bottlenecks that prevent greater achievable compression. The main features of the proposed technique are:

- It partitions the test set into groups of scan vectors such that multiple scan chains of these vectors are maximally compatible amongst them and bottlenecks are decomposed using an efficient relaxation-based test vector decomposition technique [7] in an incremental manner. This leads to high compressions with relatively small test vector counts unlike other techniques [24, 27].

- The technique is parameterized in terms of number of tester channels so the tradeoff between the desired compression, hardware cost and final test set size can be explored. The goal of the proposed technique is to achieve the user specified compression target if such a solution exists for the given test data set.

- It relies only on fault simulations and is ATPG independent. This contributes to the efficiency of the technique in comparison to ATPG-based techniques. In addition, it reduces the constraints on the ATPG process to allow it to better cope with the increasing design complexity, possibly by using parallel ATPG techniques.

3. Preliminaries

The test set configuration used as input is shown in Figure 1(a). It consists of n_v test vectors, each configured into N scan chains of length L each. The compressed output is shown in Figure 1(b). It contains $n_{v'}$ test vectors, where $n_{v'} \geq n_v$, each encoded using M input channels (called representative chains) of length L, where in general $N \gg M$. These $n_{v'}$ vectors, called representative vectors, are distributed into n_p partitions. Figure 2 shows the block diagram of decompression hardware.

Before the actual algorithms are explained, the main concepts including compatibility based merging, partitioning of test vectors, and relaxation based decomposition of test vectors, are explained with simple examples.

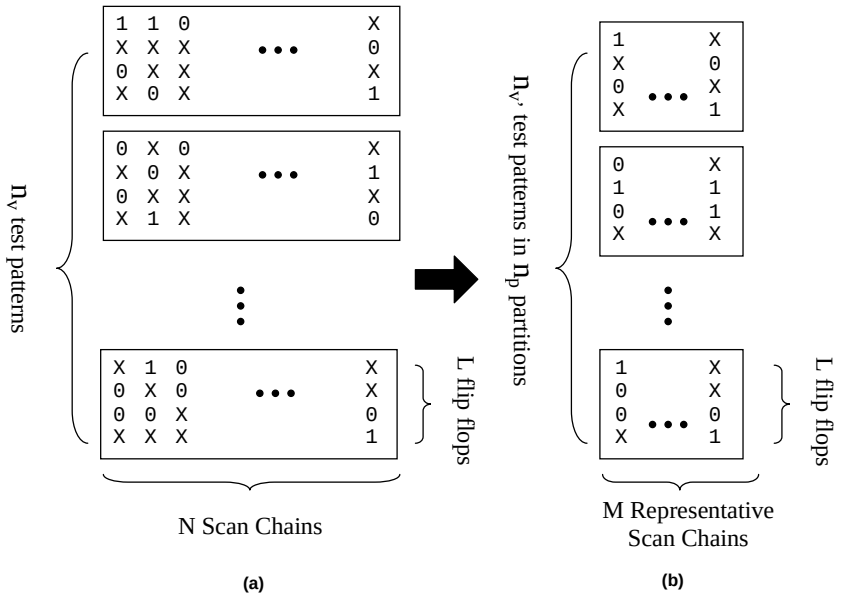

Figure 1. (a) Multiple scan chains test vectors configuration used by the compression algorithm, (b) Output of the compression algorithm.

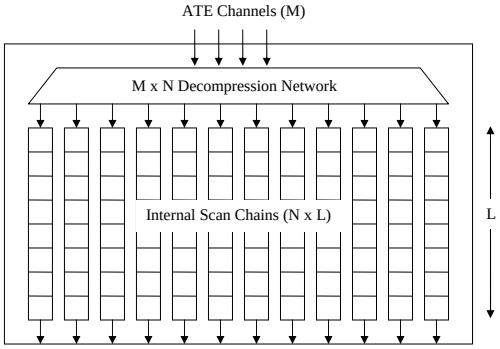

Figure 2. Block diagram of the decompression hardware and CUT with inputs and outputs.

3.1. Multiple Scan Chains Merging Using Input Compatibility

Let $b(k, i, j)$ denote the data bit that is shifted into the j^{th} flip-flop of i^{th} scan chain for the k^{th} test vector. The compatibility of a pair of scan chains is defined as follows. Two scan chains i_1 and i_2 are mutually compatible if for any k, where $1 \leq k \leq n_{v'}$, and j, where $1 \leq j \leq L$, $b(k, i_1, j)$ and $b(k, i_2, j)$ are either equal to each other or at least one of them is a don't-care. A conflict graph is constructed from the test data that contains a vertex for each scan chain. If two scan chains are not compatible, the corresponding vertices are connected by an edge in G. Graph coloring (vertex coloring) of G yields the minimum number of ATE channels required to apply the test cubes with no loss of fault coverage (in the graph coloring solution a minimum number of colors is determined to color the vertices such that no two adjacent vertices are assigned the same color). Vertices in G that are assigned the same color represent scan chains that can be fed by the same ATE channel

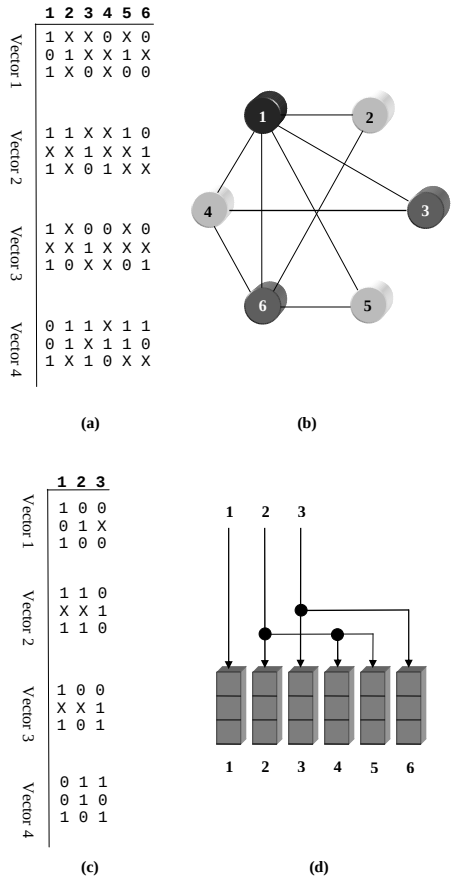

Figure 3. Width compression based on scan chain compatibilities; (a) Test cubes, (b) Conflict graph, (c) Compressed test data based on scan chain compatibility, (d) Fan-out structure.

via a fan-out structure. This procedure is illustrated using the example given in Figure 3. In this example, the number of scan chains $N = 6$, and the scan chain length $L = 3$. A conflict graph consisting of 6 vertices is constructed by analyzing the complete test set. The coloring solution for the conflict graph results in three colors used to color three sets of vertices: $\{1\}$, $\{2, 4, 5\}$, $\{3, 6\}$. Consequently, three ATE channels are needed to feed the test data for the 6 scan chains, and the fan-out structure is shown in Figure 3(d). Since the graph coloring problem is known to be NP-complete, a heuristic technique is used.

3.2. Partitioning of Test Vectors

Given a number of scan chains of a test vector, Section 3.1 explained how compatibility analysis is applied over the complete test set to obtain a reduced number of *representative* scan chains if the given scan chains are compatible amongst themselves. This is the basis of reduced data volume. Based on this, the following key observations can be made:

- The extent of compatibility achieved depends upon how many bits are specified per scan vector. The lesser the specified

bits, the lesser the conflicts, resulting in greater compatibility between the set of scan chains.

- The compatibility analysis can be applied to scan chains of a single vector or a set of vectors. Depending upon the extent of specified bits present, the amount of conflict tends to increase the longer the chains (or columns) are. The length of each column is determined by the number of bits per scan chain per vector multiplied by the total number of test vectors being considered for the analysis.

- Compatibility analysis per vector gives a lower bound on achievable *representative* scan chains for a given test set for a specified number of scan chains per vector.

If the compatibility analysis is done per vector, the resulting number of representative scan chains and the configuration of compatible groups may vary among the different vectors in a test set depending upon the amount of specified bits present and their relative positions in each vector. Even if most vectors individually lend themselves favorably to compression using compatibility analysis, the gains become less due to the increased conflicts when multiple vectors are considered together. Thus, when many vectors are being considered together, their combination can limit the compression of the test set and in the worst case this may allow no compression at all. From these observations, it is intuitively clear that greater compression can be achieved if the test vectors are partitioned into bins such that all vectors in a given bin satisfy a targeted M when considered together during compatibility analysis.

These concepts are illustrated with a simple example. The example shows the potential benefits of partitioning and how the bottleneck vectors are identified. Consider the same test set as shown in Figure 3(a) with the targeted $M = 2$ (called the *threshold*), which implies 66% compression. Consider the compatibility among the scan chains of the complete test set as shown in the conflict graph of Figure 3(b). It indicates that only $M = 3$ is possible. Next, each vector is considered separately and individual conflict graphs are presented for each vector as shown in Figures 4(a) - 4(d). It can be seen that vectors 1 and 2 satisfy the *threshold* individually (*acceptable* vectors) while vectors 3 and 4 require $M = 3$ (*bottleneck* vectors). Also, it is clear that vectors 1 and 2 cannot be combined in a single partition that will satisfy the *threshold* and hence each vector will need a separate partition. In this way we can partition the acceptable vectors to satisfy the *threshold*.The *bottleneck* vectors 3 and 4 will be dealt with as explained in Section 3.3.

3.3. Relaxation Based Decomposition of Test Vectors

It may be the case that many vectors exceed M (threshold) due to the relatively large number of specified bits present and their conflicting relative positions. The idea of test vector decomposition (TVD) [8] can be used to increase the number of unspecified bits per vector. TVD is the process of decomposing a test vector into its atomic components. An atomic component is a child test vector that is generated by relaxing its parent test vector for a single fault f. That is, the child test vector contains the assignments necessary for the detection of f. Besides, the child test vector may detect other faults in

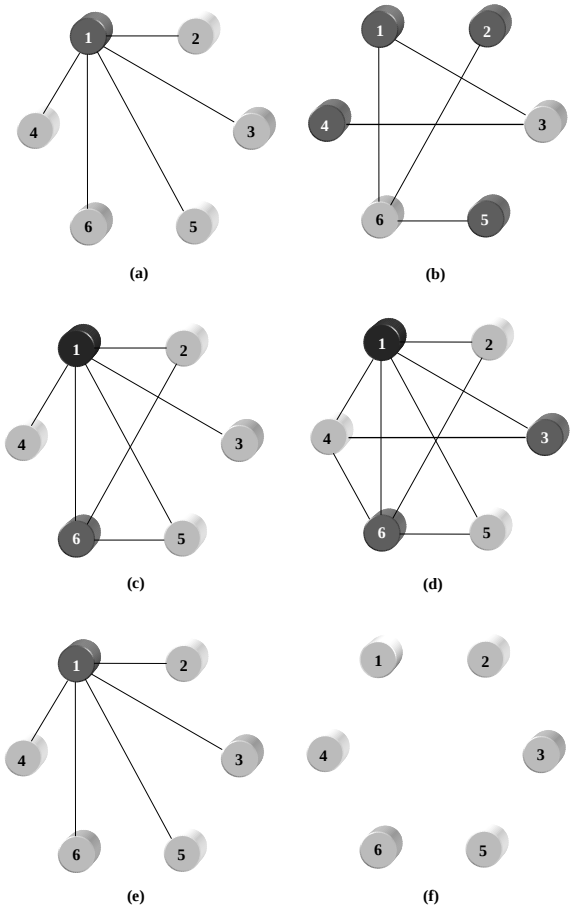

Figure 4. (a) Conflict graph for vector 1 (b) Conflict graph for vector 2 (c) Conflict graph for vector 3 (d) Conflict graph for vector 4 (e) Conflict graph for vector 3a (f) Conflict graph for vector 3b.

addition to f. For example, consider the test vector $t_p = 010110$ that detects the set of faults $F_p = \{f1, f2, f3\}$. Using the relaxation algorithm by El-Maleh and Al-Suwaiyan [7], t_p can be decomposed into three atomic components, which are $t_1 = (f1, 01xxxx)$, $t_2 = (f2, 0x01xx)$, and $t_3 = (f3, x1xx10)$. Every atomic component detects the fault associated with it and may accidentally detect other faults. An atomic component cannot be decomposed any further because it contains the assignments necessary for detecting its fault.

To achieve the desired compression ratio, a *bottleneck vector* can be decomposed into subvectors in the following manner. The atomic component for each fault detected by a *bottleneck vector* is obtained. These atomic components are then merged incrementally creating a new subvector from the parent bottleneck vector until this subvector doesn't exceed M. As many subvectors are created this way until all the faults detected by the parent vector are covered. These subvectors are made members of the existing partitions or new partitions are created to achieve the desired M as per compatibility analysis. The goal at this stage is to minimize the number of vectors resulting from this relaxation-based decomposition while also minimizing the total number of partitions. The role of TVD is explained with a simple example.

Vector	1	2	3	4	5	6
3	1	X	0	0	X	0
	X	X	1	X	X	X
	1	0	X	X	0	1
3a	1	X	0	0	X	0
	X	X	1	X	X	X
	1	0	X	X	0	X
3b	X	X	0	0	X	0
	X	X	1	X	X	X
	1	X	X	X	X	1

Figure 5. Decomposition of bottleneck vector 3 into two vectors 3a and 3b.

Consider the *bottleneck vector* 3 identified in the earlier example. Suppose that this vector is decomposed into the two vectors 3a and 3b as shown in Figure 5, each detecting a subset of faults detected by the original vector. The compatibility graphs of these vectors are shown in Figures 4(e) and 4(f). The decomposed vectors satisfy the *threshold* and can either fit in the existing partitions or new partitions can be created if required. In this example, vector 3a can fit in the same partition as vector 1 because of the identical compatibility classes while vector 3b can fit in any partition. Similarly, vector 4 can be decomposed and partitioned to satisfy the desired *threshold*.

4. Proposed Compression Algorithms

The objective of the proposed algorithms is to compress a given test set using M tester channels while minimizing the increase in the number of test vectors (due to decomposition) and total partitions, and maintaining the original fault coverage (%FC). The increase in test vectors increases the test application time and beyond a certain point decreases the compression ratio, while the increase in number of partitions increases the area overhead of the decoder. To minimize the decomposition needed, the approach used is to minimize the number of undetected faults associated with each subsequently decomposed bottleneck vector as it directly affects the amount of decomposition required: the fewer the undetected faults, the lesser the decomposition required to derive new acceptable test vectors. Since a representative vector derived from broadcasted scan values is more specified than the original test vector, it detects more faults. To benefit from this fact, all acceptable vectors present in the input test set are partitioned and faults detected by the representative vectors obtained after partitioning are dropped. Then, during bottleneck vector decomposition, each derived subvector is partitioned and its representative vector is fault simulated to drop newly detected faults before any further decomposition. However, in this approach the fault coverage depends upon the representative vectors and if they are modified, the fault coverage changes. This may happen because a partition changes as new vectors are made members of existing partitions. If a partition attains a different configuration of compatibility classes to accommodate a new vector, all the representative vectors previously created for existing members of this partition need to be updated. The consequence is that some faults that are detected by the old set of representative vectors may become undetected. This can happen for faults that are essential in the original test set and detected by a

bottleneck vector. When such faults are covered by some other representative vector, they are dropped and not considered during bottleneck vector's decomposition. However, the representative vector may be modified after the bottleneck vector has been decomposed, making the fault undetected, unless it is detected surreptitiously during the remaining pass by some representative vector. Surreptitious detection is more likely for non-essential faults. Two approaches can be used to deal with this problem: (i) either not to allow any previous essential fault detection to change while partitioning, or (ii) allow faults detection to be disturbed but address it by creating new vectors if needed. These two approaches can give different solutions in terms of total partitions created and the final test vectors count. The first approach tends to create more partitions as a vector will not be included in an existing partition if there is disturbance of any previous essential fault detection. On the other hand, the second approach may create more new vectors but has a potential to give fewer total partitions as a new vector is most likely to be included in an existing partition because of high percentage of don't cares present. Two basic variations have been proposed based on these two strategies to explore the solutions space that results due to the choice between creating a new vector or a new partition. The difference among these variations is in the partitioning step when decomposing the bottleneck vectors and specifically in the choice that needs to be made between creating a new partition for a newly created vector or using an existing partition with a chance of reduced fault coverage and then creating new vector(s) to make up for it. It should be noted that the initial partitioning of acceptable vectors does not have this issue since acceptable vectors are not recreated while partitioning.

Having discussed the basis of the technique and its variations, the main algorithm is given in Figure 6 and the details of the critical substeps and the algorithm's variations are elaborated subsequently in the referenced subsections. The algorithm does not perform any merging-based compaction or random fill of don't cares in the representative vectors because the compatibility constraints can be violated.

4.1. Initial Partitioning

If there are acceptable vectors present in the test set, an initial partitioning is performed that will allow representative vectors to be created and fault simulated so that faults can be dropped before handling the bottleneck vectors. The heuristic used for partitioning initial acceptable vectors is give in Figure 7. Test vectors with higher representative counts are attempted first as they tend to be more conflicting with other test vectors and have less degree of freedom. However, test vectors with lesser representative counts have more X's and have higher chances of fitting in existing partitions before any new partitions are created, thus leading to fewer total partitions.

4.2. Bottleneck Vector Decomposition and Partitioning

Bottleneck test vectors are decomposed and partitioned according to the algorithm given in Figure 8.

A subvector is first tested for inclusion in a partition by applying the existing compatibility graph of a partition to the

Compression Algorithm

1. *Fault simulate test set to mark essential and non-essential faults.*
2. *Analyze the compatibility of each test vector to get its* representative *scan chains (*representative count*).* M *is the maximum* representative *count that is acceptable, called the* threshold.
3. *Include vectors with* representative count $> M$ *in set* bottleneck*, otherwise in set* acceptable.
4. *Partition acceptable vectors using the initial partitioning algorithm (Section 4.1)*
5. *Get representative test vectors for all partitioned test vectors, fault simulate them and drop all detected faults.*
6. *Incrementally decompose each bottleneck vector into subvector(s) for all its undetected faults. Partition each subvector and fault simulate its representative vector to drop faults before any further decomposition of the bottleneck vector (Section 4.2).*
7. *Fault simulate the set of all representative vectors to check %FC. If %FC $<$ original, generate atomic components for all undetected faults and attempt merging them with existing partitions (Section 4.3).*
8. *For remaining undetected faults, atomic components for these faults are merged into the smallest set of subvectors satisfying the threshold and are partitioned without disturbing existing fault detection.*

Figure 6. The main algorithm.

Initial Partitioning Algorithm

1. *The* acceptable *vectors, are sorted on their* representative count *in descending order.*
2. *The first vector in the sorted list is made a member of the first (default) partition.*
3. *The compatibility of the next test vector is analyzed together with members of existing partition(s), testing the available partitions list in order. If the test vector and the existing test vectors in a partition can be* M *colored, it is included in the partition. Otherwise, the next partition is tested.*
4. *If the test vector fails to be partitioned with any of the existing partitions, a new partition is created.*
5. *Steps 3-4 are repeated until all acceptable test vectors are processed.*

Figure 7. The initial partitioning algorithm.

Bottleneck Vector Decomposition & Partitioning Algorithm

1. *Select an undetected fault from the fault list of the bottleneck vector and add its atomic component to a new subvector.*
2. *Select the next undetected fault from the list and merge its atomic component with the subvector. Determine the representative count of the subvector.*
3. *If* M *is not exceeded and there are undetected faults remaining, go to step 2.*
4. *Undo the last merge if the threshold was exceeded and perform partitioning of the created subvector.*
5. *Get the representative vectors of the modified partition and fault simulate them to drop all detected faults.*
6. *If the current bottleneck vector has remaining undetected faults, goto step 1.*

Figure 8. Bottleneck vectors processing.

subvector. If it succeeds, current fault detection is not disturbed. If this fails with all partitions, partitioning with recoloring is attempted in a similar manner as explained in section 4.1. But in this case, the problem of fault coverage loss (Section 4) has to be addressed. As mentioned earlier, two variations are proposed to address this issue. Before details of these variations are given, some terms related to fault detection need to be mentioned in order to clearly explain and justify the approach taken.

Since the algorithm relies on representative vectors created during partitioning for providing the required fault coverage, at any instant, the current test set is composed of all representative vectors and the unprocessed bottleneck vectors, if any. The term *disturbed* is used for a fault to imply that it was detected before the latest change to the set of representative vectors and is no longer detected by any vector in any partition. Furthermore, since the test set is being recreated using representative vectors, the essential and non-essential status of each fault keeps changing dynamically as each new representative vector is created. However, it should be noted that the algorithm relies on the initial essential and non-essential status of each fault, i.e., based on the initial test set. Based on this, all originally non-essential faults are treated as easy to detect faults and are not cared for during decomposition and partitioning if being disturbed. This is done because these faults are assumed to have a high probability of surreptitious detection by representative vectors and this fact is used to minimize any new vectors and partitions created to deal with these faults.

The details of the implemented variations are now given under the headings Algorithm I and II.

4.2.1 Algorithm I

This variation uses the approach of not disturbing the faults that are essential and whose detecting vector has been processed. If including the newly created subvector in a partition results in such a disturbance, another partition is tried. Finally, a new partition is created if no existing partition satisfies the conditions. The case of essential faults of currently being processed bottleneck vector needs special mention. In this variation, the currently being processed bottleneck vector is treated as 'partially processed' in terms of all faults that were checked in step 2 of bottleneck vector decomposition (Section 4.2). In other words, those essential faults detected by the currently being processed bottleneck vector that have already been checked are not allowed to be disturbed. A slight variation to this approach can be that currently being processed bottleneck vector is marked 'unprocessed' until all its faults have been covered. In other words, any fault that is detected by the currently being processed bottleneck vector is allowed to be disturbed. Since, empirically, this slight change didn't lead to any useful difference, it has not been employed.

Non-essential faults on the other hand are ignored as they have a high probability of surreptitious detection. However, in case of non-detection of any non-essential fault, the last two steps in the algorithm (Figure 6) can handle these faults as explained in Section 4.3. This variation minimizes the number of new vectors created at the expense of more partitions.

Merging Algorithm

1. *Get atomic component for the undetected fault and locate all partitioned subvector(s) that are derived from the same parent vector as this component.*
2. *Merge the atomic component with a subvector and recolor that partition.*
3. *If M is exceeded try next available subvector. Otherwise, regenerate representative vectors according to the new compatibility configuration of the partition and check for any fault detection disturbance.*
4. *If there is fault detection disturbance, try next available subvector.*

Figure 9. Atomic components merging.

4.2.2 Algorithm II

This variation is essentially an unrestricted version of Algorithm I. It allows any fault to be disturbed when partitioning subvectors of bottleneck vectors. In this algorithm, all partitions are tested and the partition that gives the minimum disturbance is selected. All the disturbed faults that are not covered even after processing of all bottleneck vectors are taken care of in the last two steps, i.e., component merging (Section 4.3) or by creating new vectors. Algorithm II attempts to minimize partitions by relaxing the partitioning criteria at the expense of creating more vectors.

4.3. Merging of Atomic Components with Partitioned Subvectors

To restore fault coverage without creating any new vectors and partitions, for each undetected fault the steps given in Figure 9 are performed. When merging fails, step 8 in the test compression algorithm (Figure 6) is performed by grouping the atomic components of undetected faults into the smallest set of subvectors satisfying the threshold. Then, these subvectors are partitioned without any fault detection disturbance.

4.4. Essential Faults First Approach (Algorithm III)

The idea behind this variation is to begin with more relaxed vectors by processing initially the essential faults in case of both the acceptable and bottleneck vectors. Intuitively, this can lead to fewer partitions due to the greater flexibility in partitioning owing to fewer specified bits in each vector being partitioned. In this algorithm, instead of starting with the given test vectors, new vectors are created that detect only the essential faults. From these new vectors, the acceptable and bottleneck vectors are identified as already explained in Section 4. Acceptable vectors are partitioned first followed by bottleneck vectors. The decomposition of bottleneck vectors and the associated subvector partitioning only considers the essential faults. The coverage of non-essential faults is left to surreptitious detection by the more specified representative vectors initially. However, the undetected non-essential faults are taken care of by using merging (Section 4.3) or by additional vectors creation at the end. During the decomposition and partitioning of bottleneck vectors for essential faults, Algorithms

I is used as the goal is to ensure detection of essential faults and minimize vectors created and the associated partitions.

4.5. Illustrative Example

A simple example is presented to illustrate the various steps of the algorithm and highlight the different outcomes that may result when different variations are used, specifically Algorithm I and II. Suppose that the number of test vectors are 4, the number of scan chains is $N = 8$ and the targeted M is 3. The test set and the compatibility analysis details are given in Table 1. The acceptable vectors 1 and 2 are partitioned using the initial partitioning algorithm resulting in a single partition. The representative vectors created and the faults dropped are shown in Table 2. The bottleneck vector 3 is decomposed to create the new subvector 3a. A new partition needs to be created for this subvector as it cannot fit in partition 1. No further decomposition of vector 3 is required because all its other faults have already been detected by the set of all representative vectors. Table 3 shows the subvector 3a, its representative vector and the faults detected. The bottleneck vector 4 is now decomposed for all undetected faults into the subvector 4a as shown in Table 4. This vector can be partitioned in partition 1 with recoloring while partition 2 cannot accommodate it. If partition 1 is selected, the new partition configuration and the faults detected are shown in Table 5. However, it can be seen that with this partitioning, the essential fault $f9$ is now disturbed and its detecting vector 3 has already been processed. Algorithms I does not allow this, hence, if Algorithm I is being used, partition 2 is tested and since this partition cannot accommodate the subvector 4a, a new partition 3 is created with the subvector 4a. All the faults have been detected and no further decomposition is required. On the other hand, if algorithm II is employed, the disturbed essential fault $f9$ needs to be covered. In this case, first atomic component for fault $f9$ is generated and its merging is attempted with the partitioned subvector 3a. This fails as the threshold (M) is exceeded. So, a new subvector 3b is created that partitions in partition 2 without recoloring due to a high percentage of Xs present. Thus, more vectors are created when Algorithm II is employed compared to Algorithm I in this case, while Algorithm I creates more partitions.

Table 1. The test set and compatibility analysis details for the example.

VECTORS	COLORS	COMPATIBILITY CLASSES	FAULTS DETECTED	ACCEPTABLE
0 X 1 X 0 1 0 1 X 1 X X X 1 1 x 1 1 0 1 X X x x	2	{1,2,4,5,7}, {3,6,8}	f1,f2,f3, f4	Yes
0 1 X X X 0 X X X X 1 1 0 X X 1 1 X X X X 1 1 X	2	{2,5,7}, {1,3,4,6,8}	f1,f4,f5, f6	Yes
0 0 1 X 1 0 1 1 1 0 1 0 X 0 X X X 0 X 1 0 X 0 1	4	{1},{2,6}, {3,5,7}, {4,8}	f1,f3,f7, f8,f9,f10, f11,f13	No
X 1 0 1 1 X 0 1 0 0 X 1 0 1 0 X 0 1 1 0 0 0 0 1	5	{1,5},{2,8}, {3},{4,6}, {7}	f5,f11,f12 ,f13,f14, f15,f16	No

Table 2. Details of partition 1.

REPRESENTATIVE VECTORS 1 AND 2	COMPATIBILITY CLASSES	COLORS	FAULTS COVERED
0 X 1 X 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0	{1,5,7},{2,4}, {3,6,8}	3	f1, f2, f3, f4, f5, f6, f7, f9
0 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 1 X 1 X 1 1 1 1			

Table 3. Details of partition 2.

SUBVECTOR 3A & ITS REP. VECTOR	COMPATIBILITY CLASSES	COLORS	FAULTS COVERED
X 0 1 X 1 0 1 1 1 X 1 0 X 0 X X X 0 X 1 X X 0 1	{1,3,7},{2,6}, {4,5,8}	3	f1, f3, f7, f8, f10, f11, f13
1 0 1 1 1 0 1 1 1 0 1 0 0 0 1 0 0 0 0 1 1 0 0 1			

Table 4. Decomposition of bottleneck vector 4.

SUBVECTOR 4A AND ITS REP. VECTOR	COMPATIBILITY CLASSES	COLORS	FAULTS COVERED
X 1 0 1 1 X X X X 0 X 1 X X 0 X X X 1 X 0 X X 1	{1,2,5,6,7},{4}, {3,8}	3	f5, f12, f14, f15, f16
1 1 0 1 1 1 1 0 0 0 X 1 0 0 0 X 0 0 1 X 0 0 0 1			

Table 5. Details of modified partition 1.

REPRESENTATIVE VECTORS 1,2 & 4A	COLORS	COMPATIBILITY CLASSES	FAULTS DETECTED
0 0 1 0 0 1 0 1 X 1 1 X 1 1 1 1 1 1 0 1 1 0 1 0	3	{2,5,7}, {3,6,8}, {1,4}	f1, f2, f3, f4, f5, f6, f7, f8, f12, f14, f15, f16
0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1			
1 1 0 1 1 0 1 0 1 0 X 1 0 X 0 X X 0 1 X 0 1 0 1			

5. Decompression Hardware

The decompression hardware for compatibility based compression is simply a fan-out structure as shown in Figure 3(d). The decompression hardware required to support partitioning of the test set can be realized by MUXs. Essentially, since each partition has a different set of compatibility classes, it requires it's own fan-out structure. The MUXs allow different fan-out structures to be switched in as required. Thus, the number of MUXs required is equal to the number of fan-out scan chains

feeding the core, i.e., N. The number of data inputs on each of these MUXs is equal to the number of partitions i.e. n_p and are connected to one of the M ATE inputs. The MUXs' select lines are controlled by a partitions counter that increments according to partitions' changes. This counter's enable input is controlled by a patterns' counter that is loaded with the partition's size to support the non-uniform partition sizes. The patterns' counter is loaded with the size of partition when the first vector of a partition is being scanned in. It then decrements with each input vector. When this counter reaches to zero, the enable signal of partition's counter is asserted. The overall hardware structure is shown in Figure 10. It also shows how the MUXs are connected for implementing the partitioning. Thus, in this arrangement, the cost of hardware is proportional to n_p times N. Since N is mostly a design parameter, the n_p required to achieve the desired compression determines the cost. However, it should be noted that the actual MUX sizes can be optimized by the synthesis tool, e.g., by utilizing the common tester channel inputs within a single MUX or across different MUXs.

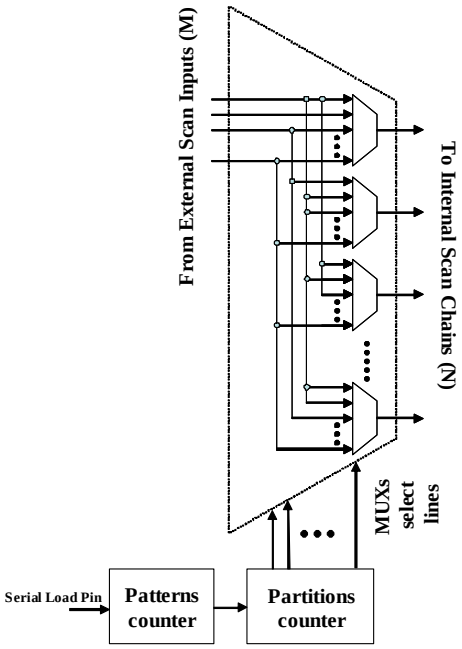

Figure 10. The decompression hardware structure

6. Experiments

In this section, experimental results are discussed in detail followed by a comparison with other work.

6.1. Experimental Setup

The compression algorithms were written in C and compiled under Linux. HOPE fault simulator [21] and the DSATUR graph coloring implementation by Joseph Culberson [4] have been used. To evaluate the proposed algorithms, full-scan

Table 6. Details of ISCAS-89 benchmarks and test sets used

Test Case	#FFs	Static compacted			Dynamic compacted		
		#TV	%DC	T_D	#TV	%DC	T_D
s5378	214	97	74.14	20758	111	72.62	23754
s9234	247	105	70.29	25935	159	73.01	39273
s13207	700	233	93.36	163100	236	93.15	165200
s15850	611	94	80.96	57434	126	83.56	76986
s35932	1763	12	36.68	21156	16	35.3	28208
s38417	1664	68	67.36	113152	99	68.08	164736
s38584	1464	110	80.72	161040	136	82.28	199104

versions of seven largest ISCAS-89 benchmark circuits have been used with MINTEST-generated [10], static and dynamic compacted test sets. The static compacted test sets were relaxed using a bit-wise relaxation approach [7] to obtain don't cares and have been used for detailed results. The characteristics of test sets are given in Table 6. The columns #FF, #TV, %DC and T_D indicate the number of scan flip-flops, the total number of test vectors, percentage of don't care bits and test data volume in bits, respectively. These tests achieve 100% coverage of all detectable faults.

As discussed in section 3, M and N are required input parameters. With a large N, better compression is obtained in general due to less conflicts in the compatibility analysis. However, actual constraints on routing dictate the maximum allowed N value for a given circuit [30]. The experimentation results are presented with N approaching 100 and 200 scan chains for the largest five circuits and 64 for the smaller circuits. These values enable comparison with most other work. For a given N, the desired M value is varied within a range that shows the behavior of the algorithm in terms of number of partitions created and final test vector count over a range of compression targets for each test case. There is, however, a limit on the minimum value of M permitted with a given test set and chosen N. This is determined by the minimum number of colors required to color an atomic component of any fault in the list.

6.2. Results and Discussion

Before discussing the results of the proposed algorithms, analysis of the MINTEST static compacted test sets using different scan chain lengths is presented. Figures 11 and 12 show the number of bottleneck vectors with decreasing M using different scan chain lengths for the test cases s5378 and s38417, respectively, obtained through compatibility analysis. The compression ratio (CR%) is defined as:

$$CR\% = \frac{n_v \times L \times N - n_{v'} \times L \times M}{n_v \times L \times N} = 1 - \frac{M}{N} \times \frac{n_{v'}}{n_v}$$

Note that decreasing M and increasing N corresponds to increasing the compression ratio. It is observed that the bottleneck vectors increase with both increasing M and scan chain length, thus requiring more decomposition for targeting a smaller

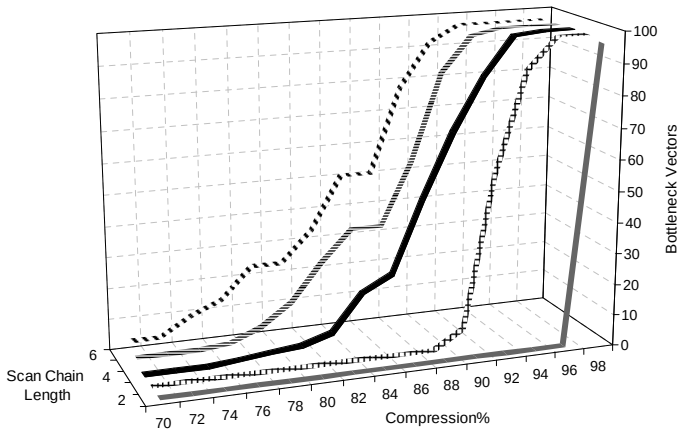

Figure 11. Bottleneck vectors with respect to scan chain length and desired compression for s5378.

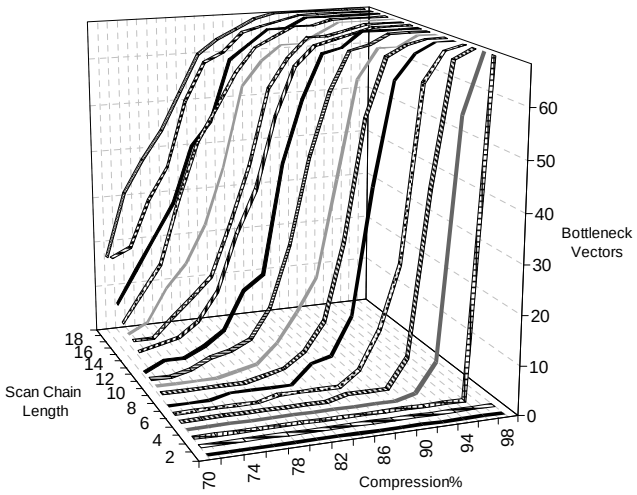

Figure 12. Bottleneck vectors with respect to scan chain length and desired compression for s38417.

M.

To show the impact of partitioning, Figure 13 shows the achievable compression using only compatibility analysis of all MINTEST static compacted tests without partitioning at different scan chain lengths. It can be seen that little or no compression is achieved except at very small scan chain lengths. Figure 14 shows the achievable compression with partitioning at different scan chain lengths with M set equal to the maximum representative count of the test set so that no test vector decomposition is required. The required number of partitions are shown in Figure 15. Significant compression is achieved by using partitioning, especially at small scan chain lengths. By using test vector decomposition, compression can be further improved as shown next.

Since it is found experimentally that the overall trend is similar in all proposed algorithm variations, detailed results for Algorithm I are given in Tables 7 - 9 and this is followed by a discussion of the differences in the results obtained by all

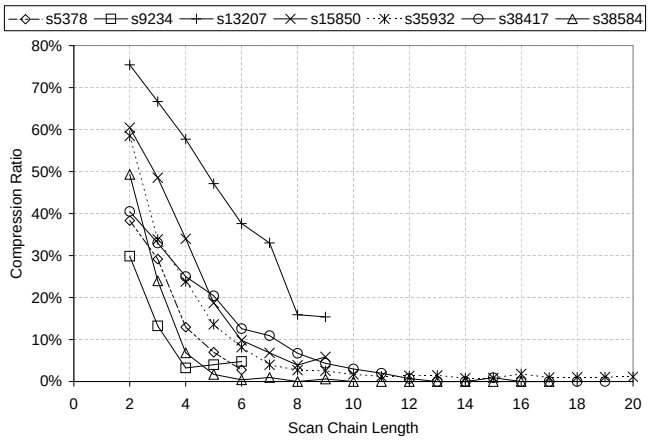

Figure 13. Maximum compression achieved without partitioning at different scan chain lengths.

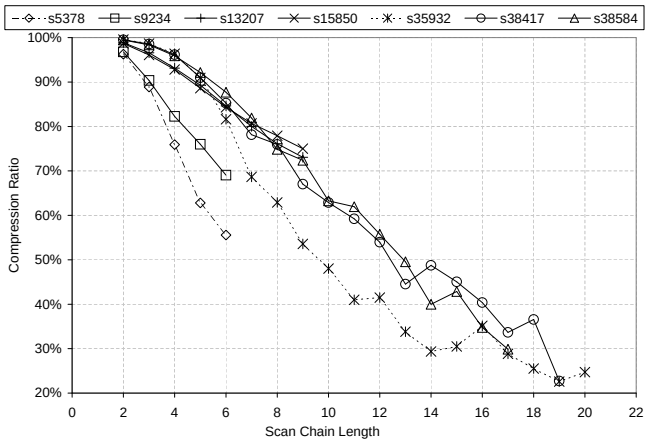

Figure 14. Maximum compression achieved with partitioning at different scan chain lengths without any decomposition required.

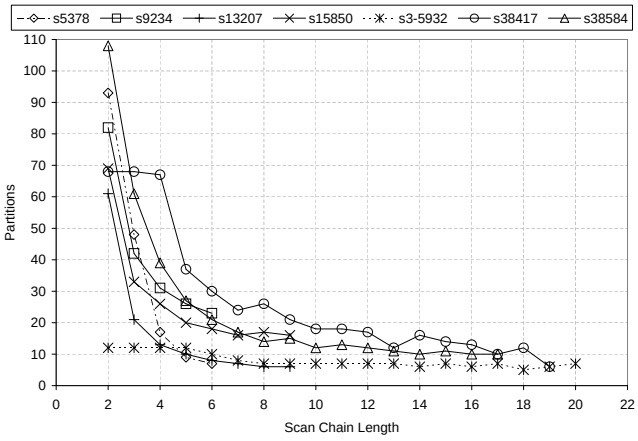

Figure 15. Partitions required to support compression shown in Figure 14.

algorithm variations. In these tables, n_B, n'_v, n_p and $CR\%$ give the number of bottleneck vectors, the final test vector count, the number of partitions and the compression ratio, respectively. The highest compression that is achieved without any increase in test vectors is shown underlined while the overall highest compression achieved is shown in bold. With increasing compression, there is invariably an increase in the number of partitions required because of increased conflicts among the vectors when colored together, owing to a stricter constraint on the desired number of colors. Furthermore, as decomposition creates more vectors, these may require new partitions to satisfy the constraint. However, it should be noted that the test vectors count does not increase till a certain point even though the decomposition of bottleneck vectors takes place. This happens when a bottleneck vector(s) is decomposed only into a single vector because a portion of its faults are already covered by other representative vectors. It can also be observed that with decreasing M, the resulting compression increases until the increase in test set size overcomes any gains from reduced M. The test case s35932 shows an irregular change in compression due to the small number of total vectors such that any increase in vectors due to decomposition has a pronounced effect on overall compression achieved. After the point where the decrease in M overcomes the effect of increasing test vectors, the compression increases steadily. Comparing between the two different scan chain lengths selected for the larger five benchmarks, it can be seen that a higher compression is achieved at similar or lower counts of both vectors and partitions with smaller chain lengths. Compared to other test cases, s35932 and s38417 have a relatively small percentage of Xs and consequently a much larger number of bottleneck vectors. Hence, the compression ratios for these test cases are low, especially when smaller number of scan chains are used. Since the percentage of don't cares in current industrial designs are much higher, these two test cases are exceptions rather than indicative of the algorithm's performance.

Bottleneck vectors can be dealt with either through test vector decomposition, as illustrated in this approach, or through serial scan of the test vectors. It is assumed that in a serial scan approach, a test vector will be scanned by loading M scan chains (equal to the number of tester channels) in parallel. Thus, a test vector will require $\lceil \frac{N}{M} \rceil$ scan chain loads to load a bottleneck vector. Tables 10 and 11 shows the test application times (in clock cycles) and compression ratio versus M and the bottleneck vectors using Algorithm I. The columns TAT and TAT' are the test application times with bottleneck test vector decomposition vs. serial load of bottleneck vectors, respectively. Similarly, the columns CR and CR' give the compression

Table 7. Results of Algorithm I with $N \leq 64$ for s5378 and s9234.

s5378 ($N = 54$)					**s9234** ($N = 62$)				
M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%
11	2	22	97	79.44	10	1	34	105	83.81
10	3	29	97	81.31	9	2	40	105	85.43
9	6	36	97	83.18	8	3	45	105	<u>87.04</u>
8	18	37	97	<u>85.05</u>	7	8	54	107	88.45
7	23	50	99	86.65	6	13	67	108	90.01
6	46	57	108	**87.51**	5	37	81	129	**90.05**
5	67	52	148	85.74	4	70	92	166	89.76

Table 8. Results of Algorithm I with N approaching 100 for the largest five test cases.

s13207 ($N = 100$)					**s15850** ($N = 88$)					**s35932** ($N = 98$)					**s38417** ($N = 98$)					**s38584** ($N = 98$)				
M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%
19	1	8	233	81.0	13	1	22	94	85.1	68	1	8	12	30.6	32	7	40	68	67.3	39	5	18	110	60.0
11	1	14	233	89.0	12	2	24	94	86.2	67	2	8	13	25.9	30	9	44	68	69.3	24	7	26	110	75.4
10	2	16	233	90.0	11	3	26	94	87.4	38	7	14	18	41.8	28	11	48	68	71.4	19	10	34	112	80.2
9	2	18	233	91.0	10	11	29	99	87.9	33	8	15	19	46.6	26	18	52	70	72.7	17	12	37	112	82.3
8	4	20	233	92.0	9	12	32	100	89.0	22	9	17	24	55.1	24	27	57	73	**73.7**	15	13	43	112	84.3
7	6	22	233	93.0	8	15	36	103	89.9	10	12	25	34	71.1	22	37	62	80	73.6	12	21	52	115	87.1
6	9	28	235	93.9	7	20	42	106	90.9	8	12	25	37	74.8	20	46	64	88	73.6	10	25	59	115	89.3
5	15	35	239	94.9	6	26	50	112	91.8	6	12	29	43	78.0	18	56	66	102	72.4	8	20	80	117	91.3
4	26	48	248	95.7	5	37	62	125	92.4	4	12	28	56	80.9	16	61	69	117	71.9	4	107	132	203	92.4
3	47	67	264	**96.6**	4	54	76	144	**93.0**	3	12	26	65	**83.4**	15	63	69	124	72.1	3	109	164	257	**92.8**

Table 9. Results of Algorithm I with N approaching 200 for largest five test cases.

s13207 ($N = 175$)					**s15850** ($N = 153$)					**s35932** ($N = 196$)					**s38417** ($N = 185$)					**s38584** ($N = 183$)				
M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%
9	1	17	233	94.9	10	1	27	94	93.4	79	2	8	12	56.7	35	2	41	68	81.1	32	2	20	110	82.5
8	2	20	233	95.4	9	2	30	94	94.1	15	12	18	23	85.3	32	3	46	68	82.7	29	4	21	110	84.1
7	2	23	233	96.0	8	5	34	96	94.6	13	12	18	24	86.7	29	6	53	68	84.3	21	7	29	110	88.5
6	3	27	233	96.6	7	9	38	98	95.2	11	12	18	24	88.8	23	19	62	68	**87.6**	14	10	43	110	92.3
5	6	34	233	97.1	6	16	46	102	95.7	9	12	21	27	89.7	21	38	66	76	87.3	10	19	57	113	94.4
4	20	48	239	97.7	5	27	57	112	96.1	7	12	20	30	91.1	18	53	67	94	86.5	8	27	71	114	95.5
3	45	72	256	98.1	4	45	72	132	**96.3**	5	12	20	35	92.5	14	65	72	133	85.2	6	47	96	121	**96.4**
2	78	164	295	**98.5**	3	69	87	180	96.2	3	12	19	44	**94.4**	11	67	71	180	84.2	4	104	123	186	96.3

ratio. The values TAT, TAT' and CR' are calculated using the following expressions:

$$TAT' = (n_v - n_B) \times L + n_B \times \lceil \frac{N}{M} \rceil \times L$$

$$TAT = n'_v \times L$$

Table 10. Test application times and compression ratios with and without bottleneck vector decomposition for benchmarks s5378, s9234 and s13207.

s5378 ($N = 64$)						**s9234** ($N = 52$)						**s13207** ($N = 100$)					
M	n_B	TAT'	TAT	CR'	CR	M	n_B	TAT'	TAT	CR'	CR	M	n_B	TAT'	TAT	CR'	CR
12	1	404	388	77.0	77.6	10	1	444	420	83.1	83.8	11	1	1694	1631	88.6	89.0
11	2	420	388	78.0	79.4	9	2	468	420	83.9	85.4	10	2	1757	1631	89.2	90.0
10	3	448	388	79.0	81.3	8	3	504	420	84.6	87.0	9	2	1785	1631	90.2	91.0
9	6	508	388	78.2	83.2	7	8	676	428	82.0	88.4	8	4	1967	1631	90.4	92.0
8	18	820	388	69.4	85.0	6	13	940	432	79.1	90.0	7	6	2219	1631	90.6	93.0
7	23	1032	396	66.4	86.6	5	37	2196	516	59.5	90.1	6	9	2639	1645	90.4	93.9
6	46	1860	432	46.7	87.5	4	70	4620	664	31.2	89.8	5	15	3626	1673	88.9	94.9
5	67	3068	592	28.1	85.7							4	26	5999	1736	85.3	95.7
												3	47	12488	1848	77.4	96.6

Table 11. Test application times and compression ratios with and without bottleneck vector decomposition for benchmarks s15850, s35932, s38417 and s38584.

s15850 ($N = 88$)						**s35932** ($N = 98$)						**s38417** ($N = 98$)						**s38584** ($N = 98$)					
M	n_B	TAT'	TAT	CR'	CR	M	n_B	TAT'	TAT	CR'	CR	M	n_B	TAT'	TAT	CR'	CR	M	n_B	TAT'	TAT	CR'	CR
13	1	700	658	84.3	85.1	68	1	234	216	28.1	30.6	32	40	1513	1156	60.4	67.3	52	1	1665	1650	46.5	46.7
12	2	756	658	84.5	86.3	63	2	252	234	29.8	30.3	30	44	1615	1156	60.2	69.4	48	2	1710	1650	50.1	50.8
11	3	805	658	84.7	87.4	38	7	468	324	25.5	41.8	28	48	1717	1156	59.9	71.4	24	8	2130	1650	70.0	75.4
10	11	1274	693	78.3	87.9	33	8	504	342	22.1	46.7	27	51	1972	1156	55.4	72.4	22	9	2190	1665	71.2	78.7
9	12	1414	700	78.3	89.0	28	9	702	378	17.9	50.0	26	52	2074	1190	54.0	72.7	20	10	2250	1680	72.4	79.1
8	15	1708	721	76.4	90.0	23	9	864	468	19.1	49.1	24	57	2992	1241	45.5	73.7	16	13	2820	1695	73.8	83.2
7	20	2338	742	72.5	91.0	18	10	1116	450	13.6	61.7	22	62	3672	1360	35.4	73.6	12	21	4170	1725	71.0	87.1
6	26	3206	784	67.4	91.8	13	11	1602	522	7.2	67.9	20	64	4284	1496	25.8	73.6	8	30	7050	1755	66.8	91.3
5	37	5061	875	57.2	92.4	8	12	2808	666	0	74.8	18	66	5916	1734	14.4	72.4	4	107	40170	3045	2.6	92.4
4	54	8596	1008	40.6	93.0	3	12	7128	1170	0	83.4	16	69	7378	1989	8.6	71.9	3	109	53970	3855	0.9	92.8

Table 12. Highest compression obtained with and without increment in test vectors for all variations with N near 100.

Test Case	Incr. Allowed	**Algorithm I**					**Algorithm II**					**Algorithm III**				
		M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%
s5378	no	8	18	37	97	85.05	8	18	37	97	85.05	8	18	37	97	85.05
	yes	6	46	57	108	**87.51**	6	46	56	109	**87.40**	6	46	57	110	**87.28**
s9324	no	8	3	45	105	87.04	8	3	45	105	87.04	8	3	50	105	87.04
	yes	5	37	81	129	**90.05**	6	13	66	108	**90.01**	5	37	85	130	**89.97**
s13207	no	7	6	22	233	93.00	7	6	22	233	93.00	7	6	23	233	93.00
	yes	3	47	67	264	**96.60**	3	47	66	265	**96.59**	3	47	66	269	**96.54**
s15850	no	11	3	26	94	87.40	11	3	26	94	87.40	12	2	26	94	86.25
	yes	4	54	76	144	**92.98**	4	54	75	145	**92.93**	4	54	79	142	**93.08**
s35932	no	68	1	8	12	30.57	68	1	8	12	30.57	67	2	8	12	31.59
	yes	3	12	26	65	**83.41**	3	12	23	81	**79.33**	3	12	29	65	**83.41**
s38417	no	27	16	51	68	72.42	28	11	47	68	71.39	27	13	47	68	72.42
	yes	24	27	57	73	**73.68**	24	27	57	73	**73.68**	22	32	62	78	**74.22**
s38584	no	24	8	26	110	75.41	41	2	16	110	57.99	22	9	31	110	77.46
	yes	3	109	164	257	**92.82**	3	109	159	281	**92.15**	3	109	163	225	**92.87**

Table 13. Highest compression obtained with and without increment in test vectors for all variations with N near 200.

Test Case	Incr. Allowed	**Algorithm I**					**Algorithm II**					**Algorithm III**				
		M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%	M	n_B	n_p	n'_v	CR%
s13207	no	5	6	34	233	97.14	5	6	33	233	97.14	5	9	34	233	97.14
	yes	2	78	164	295	**98.55**	2	78	164	295	**98.55**	2	80	166	302	**98.52**
s15850	no	9	2	30	94	94.11	9	2	30	94	94.11	9	2	32	94	94.11
	yes	4	45	72	132	**96.32**	4	45	72	132	**96.32**	3	72	86	178	**96.28**
s35932	no	79	2	8	12	59.67	79	2	8	12	59.67	71	4	8	12	63.76
	yes	3	12	19	44	**94.38**	3	12	15	56	**92.85**	3	12	22	44	**94.38**
s38417	no	23	19	62	68	87.56	23	19	62	68	87.56	25	12	60	68	86.48
	yes	23	19	62	68	**87.56**	23	19	62	68	**87.56**	21	33	66	70	**87.81**
s38584	no	14	10	43	110	92.35	21	7	28	111	88.42	21	7	29	110	88.52
	yes	6	47	96	121	**96.39**	6	47	94	121	**96.39**	6	47	94	120	**96.42**

$$CR' = \frac{n_v \times L \times N - ((n_v - n_B) \times L \times M + n_B \times L \times N)}{n_v \times L \times N}$$

The benefit of incremental decomposition using the fault dropping approach is clearly visible. As can be observed, the proposed approach achieves significantly lower test application time compared to the serial approach especially for high compression ratios with increasing bottleneck vectors count.

As discussed in Section 4, the approach used to decompose bottlenecks and partition the subvectors determine the final vectors count, number of partitions and ultimately the actual compression ratio. This is the differentiating factor in the proposed algorithm variations. The emphasis in the comparison among the algorithm variations is with respect to the number partitions and the number of vectors created. Tables 12 and 13 show the highest compression obtained with and without allowing increment in test vectors for each test case for all variations. The results of the first two variations are nearly identical in most cases except for slight differences at low M values when increment is allowed. Algorithm I achieves the lowest final test vector counts while Algorithm II tradeoffs test vector counts for smaller number of partitions, though the differences in number of vectors and partitions are not drastic. On the other hand, Algorithm III results in less compression compared to Algorithm I and II in some cases. This happens when many new vectors need to be created to handle the non-essential faults. Furthermore, these new vectors may not fit in the existing partitions, resulting in more new partitions. However, Algorithm III achieves the best results in test cases s38417 and s38584, in some cases giving the highest compression with the least partitions. In these test sets, it is observed that a very large percentage of bottleneck vectors are present compared to other test cases at similar compression levels. By focusing on essential faults only, Algorithm III obtains less vectors and, consequently smaller number of partitions as many of the non-essential faults are covered by the representative vectors. Thus, in these test cases, focusing on essential faults first gives two-fold gains of greater compression with fewer partitions due to the less decomposition required.

6.3. Comparison with Other Work

For a fair comparison with other work, the test sets used, number of internal scan chains, and test vector count has to be taken into account. The total test vectors count affects the test application time, so it is important to be taken into account in the comparison. Finally, the cost of implementing decompression hardware is also compared with those works that report it. For each test case, the best result of the three variations is used in the comparison according to the comparison criteria.

Table 14 presents comparison with a MUXs network based decompressor [13]. This scheme also uses a static reconfiguration approach. ATLANTA [20] ATPG tool has been used to generate the test sets and 100 scan chains are used for all circuits. Separate comparisons are given according to M value, total vectors count, and the highest achieved compression. The columns T_E and TV indicate compressed test data volume in bits and final test vector count, respectively. The results

show that the proposed technique gives greater compression in three out of five test cases. It should be noted that due to different test sets used, the distribution and percentage of don't cares bits is not identical, which has an impact on the results obtained. Furthermore, the compared scheme in [13] relies on an iterative test pattern generation procedure per fault during the compression, which is a much more expensive process compared to the relaxation procedure used in this work.

Table 14. Comparison with MUXs network based decompressor.

Test	[13]				**At Similar M**				**At Similar TV**				**Highest Compression**			
Case	M	#TV	n_P	T_E	M	#TV	n_P	T_E	M	#TV	n_P	T_E	M	#TV	n_P	T_E
s13207	10	248	22	17360	10	233	15	16310	4	248	48	6944	3	264	67	5544
s15850	13	108	11	9828	13	94	22	8554	7	106	42	5194	4	142	79	3976
s35932	11	15	24	2970	11	29	24	5742	62	13	8	14508	3	65	26	3510
s38417	20	74	28	27676	20	88	64	29920	22	78	62	29172	22	78	62	29172
s38584	15	122	26	27450	15	112	43	25200	10	115	59	17250	3	255	163	3825

Next, comparisons are presented with a variety of recent techniques based on different approaches. Table 15 presents a comparison with approaches that use the MINTEST test sets with dynamic compaction (see Table 6). For the proposed scheme, the values indicate the highest compression obtained without any increment in test vectors with N near 100 and 200. The values are the compressed test data volumes in bits. The schemes compared with include block merging (BM) [5], arithmetic coding (AC) based compression [14], dictionary based nine-coded compression (D9C) [35], pattern-based runlength (PRL) compression [29], multi-level Huffman (MLH) code compression [17], a modified Huffman (MH) codes based technique [16], a dictionary with corrections (DC) scheme [37], and another dictionary-based scheme using fixed length indices (DFL) [26]. Schemes [37] and [26] use multiple scan chains varying between 16 to 200 and the best results are given. The last two columns show the percent improvements over the best results (shown in bold) among all compared schemes while the bottom two rows show the average improvements over each individually compared scheme calculated by averaging the results for all test cases. The individual percentage improvements average varies from 38% to 96% and from 106% to 307% in case of N near 100 and 200, respectively. The improvements over best results is a significant 91% with N near 200 and 3.24% with N near 100. It should be noted that much greater compression can be achieved by allowing even a little increase in test vector count. For instance, in case of s38584 by allowing an increment of only one more test vector, 56.6% compression can be achieved compared to only 28.9% without any increment, using near 100 scan chains. This is because a single vector can be the bottleneck in achieving higher compression ratios.

In Table 16, a comparison with some multiple scan chains based techniques is given. Results for the proposed scheme are given with and without test vector increase over the MINTEST static compacted test set. The T_E values with increment in test vector counts also indicate the associated vector counts. For the smallest two circuits near 64 scan chains are used while for the larger five circuits both near 100 and 200 scan chains are used. The scan chains and test sets used in other schemes are as follows: Shi et al. [32] (FCSCAN) use a commercial ATPG tool and compression is reported with 200 scan chains, Hayashi et al. [15] (SDI) use X-maximal program for test set generation with scan chains lengths varying between 16 and

Table 15. Comparison with other schemes using the MINTEST dynamic compacted test sets.

Test Case	**# TV**	**Near 100**	**Near 200**	BM [5]	AC [14]	D9C [35]	PRL [29]	MLH [17]	MH [16]	DC [37]	DFL [26]
s5378	111	4440*	N/A	10694	10861	8732	9530	9358	11644	11592	**6345**
s9234	159	5724*	N/A	19169	16235	15309	15674	15511	17877	18908	**11498**
s13207	236	19824	6608	24962	26343	17809	18717	18384	31818	31772	**8517**
s15850	126	11466	4032	23488	24762	18523	19801	18926	25459	27721	**13873**
s38417	99	62271	27621	66899	72044	**47147**	70721	58785	71874	84896	62939
s38584	136	142800	42432	65983	85984	52225	**51429**	55200	70443	65396	53287
Avg. %Imp. ($N \approx 100$)				76.69	75.50	37.96	50.86	45.43	86.98	95.93	7.69
Avg. %Imp. ($N \approx 200$)				239.50	269.07	155.67	187.90	172.63	284.79	307.45	106.60

* near 64 scan chains used

64 in powers of 2, Tang et al. [34] (ONC) do not specify the specifics of test sets and all results are reported with 256 scan chains except for s15850, where 128 are used. [30] (RSS) uses Synopsys TetraMax with near 100 scan chains while EDT [28] does not give the specifics of test sets and scan chains. The best results among the strategies compared with are shown in bold. The proposed technique achieves higher compression in all test cases except for s35932 at comparable or much lower vector counts, even though a conservative number of scan chains are used as compared to some of these schemes [34].

Table 16. Comparison with other multiple scan chain schemes.

	Proposed						FCSCAN [32]		SDI [15]		ONC [34]		RSS [30]		EDT [28]
	No incr.		**With Incr.**												
Test	**100**	**200**	**100**		**200**										
Case	T_E	T_E	**#TV**	T_E	**#TV**	T_E	**#TV**	T_E	**#TV**	T_E	**#TV**	T_E	**#TV**	T_E	T_E
s5378	3104*	N/A	*108*	2592*	*N/A*	N/A	*N/A*	N/A	*99*	5748	*N/A*	N/A	*N/A*	N/A	**5676**
s9234	3360*	N/A	*129*	2580*	*N/A*	N/A	*N/A*	N/A	*110*	**8872**	*N/A*	N/A	*N/A*	N/A	9534
s13207	11417	4660	*248*	6944	*239*	3824	*251*	10920	*233*	13114	*415*	**4980**	*296*	6512	10585
s15850	7238	3384	*144*	4032	*132*	2112	*148*	**7072**	*97*	11372	*386*	7720	*448*	8960	9805
s35932	14472	7668	*34*	4896	*34*	1530	*35*	8045	*12*	7252	*45*	**1260**	*N/A*	N/A	N/A
s38417	31212	14076	*78*	29172	*73*	13797	*183*	29550	*86*	30404	*692*	**19376**	*781*	21868	31458
s38584	36300	12320	*197*	11820	*121*	5808	*288*	21020	*111*	28140	*537*	**12888**	*636*	16536	18568

* near 64 scan chains used

6.4. Hardware Cost Comparison

Unlike compression results, extensive comparison of hardware cost with previous work is not always possible because either the actual hardware cost and the specific implementation library is not reported, or the schemes are based on a single scan chain. To compare hardware cost, two schemes based on multiple scan chains are considered that report hardware cost using the lsi_10k library provided with the Synopsys Design Compiler. Of these, one is a dictionary based scheme [26] with variable length indices [25] while the other uses width compression along with variable length indices dictionary compression [25]. Table 17 reports the hardware cost obtained with the proposed algorithms for each test case for two compression values. These two results correspond to compression obtained with and without increment in test vector counts using near 100 scan chains. Since the hardware cost is dependent upon partitioning, these values give a an idea about the

difference in hardware cost for different compression values. It is observed that compared to the pure dictionary scheme [26], much greater compressions are obtained at lower or similar hardware costs. In case of the work in [25], the reported hardware costs are smaller but compression is significantly lesser compared to the proposed scheme in all test cases except s35932.

Table 17. Comparison of H/W costs with some other schemes.

	Proposed		[26]		[25]	
Test Case	H/W	T_E	H/W	T_E	H/W	T_E
s5378	1536/1518	3104/2592	2636	6124	628	11180
s9234	2225/2717	3360/2592	3701	11388	1397	18410
s13207	1711/2228	11417/5544	4293	6093	1270	14087
s15850	2120/3178	7238/3976	3908	12947	1469	15907
s35932	987/1388	14688/5742	3026	1040	36	3308
s38417	4172/4787	32368/29172	2382	58397	74	69274
s38584	2717/4559	41250/17250	5036	52612	1320	54878

7. Conclusions and Future Work

An effective test vector compression technique has been proposed in this work that uses test set partitioning and bottleneck test vector decomposition through relaxation. The technique targets a user specified number of ATE channels to achieve test data compression and it can explore tradeoffs among compression ratio and area overhead. The technique relies on an efficient test relaxation algorithm and can work with compacted test sets to achieve high compression with much lower vector counts, thus minimizing test application time. The concept of constrained incremental relaxation can be used with other compression techniques as well. The results clearly show that the proposed technique achieves significantly greater compression compared to other recent work. Further improvements may be achieved by directing test relaxation to avoid conflicting bit positions. This is being explored as a potential enhancement to this work.

References

[1] A. A. Al-Yamani, S. Mitra, and E. J. McCluskey. Optimized Reseeding by Seed Ordering and Encoding. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(2):264–270, February 2005.

[2] K. Chakrabarty and B. T. Murray. Design of Built-In Test Generator Circuits Using Width Compression. *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, 17(10):1044–1051, October 1998.

[3] C. Chen and S. K. Gupta. Efficient BIST TPG Design and Test Set Compaction via Input Reduction. *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, 17(8):692 –705, August 1998.

[4] J. Culberson. Graph Coloring Page.

[5] A. El-Maleh. An Efficient Test Vector Compression Technique Based on Block Merging. In *ISCAS '06: Proceedings of the International Symposium on Circuits and Systems*, 2006.

[6] A. El-Maleh and R. Al-Abaji. Extended Frequency-directed Run-length Codes with Improved Application to System-on-a-Chip Test Data Compression. In *ICECS'02: Int. Conf. on Electronic Circuits Systems*, pages 449–452, 2002.

[7] A. El-Maleh and A. Al-Suwaiyan. An Efficient Test Relaxation Technique for Combinational & Full-Scan Sequential Circuits. In *VTS '02: Proceedings of the 20th IEEE VLSI Test Symposium*, page 53, Washington, DC, USA, 2002. IEEE Computer Society.

[8] A. El-Maleh and Y. E. Osais. Test Vector Decomposition-Based Static Compaction Algorithms for Combinational Circuits. *ACM Transactions on Design Automation of Electronic Systems*, 8(4):430–459, October 2003.

[9] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolici. Variable-Length Input Huffman Coding for System-on-a-Chip Test. *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, 22(6):783–796, June 2003.

[10] I. Hamzaoglu and J. H. Patel. Test Set Compaction Algorithms for Combinational Circuits. In *Proc. Int. Conf. Comput.-Aided Des.*, pages 283–289, 1998.

[11] I. Hamzaoglu and J. H. Patel. Reducing Test Application Time for Full Scan Embedded Cores. pages 260–267. FTC'99: IEEE International Symposium on Fault Tolerant Computing, 1999.

[12] I. Hamzaoglu and J. H. Patel. Reducing Test Application Time for Built-in-Self-Test Test Pattern Generators. In *VTS'00: Proceedings of the 18th IEEE VLSI Test Symposium*, page 369, Washington DC, USA, 2000. IEEE Computer Society.

[13] Y. Han, X. Li, S. Swaminathan, Y. Hu, and A. Chandra. Scan Data Volume Reduction Using Periodically Alterable MUXs Decompressor. In *ATS '05: Proceedings of the 14th Asian Test Symposium*, pages 372– 377, 2005.

[14] H. Hashempour and F. Lombardi. Application of Arithmetic Coding to Compression. *IEEE Trans. Comput.*, 54(9):1166–1177, September 2005.

[15] T. Hayashi, H. Yoshioka, T. Shinogi, H. Kita, and H. Takase. Test Data Compression Technique Using Selective Don't-Care Identification. In *ASP-DAC '04: Proceedings of The 2004 Conference on Asia South Pacific Design Automation*, pages 230–233, Piscataway, NJ, USA, 2004. IEEE Press.

[16] L. Jieyi, F. Jianhua, Z. Iida, X. Wenhua, and W. Xinan. A New Test Data Compression/Decompression Scheme to Reduce SOC Test Time. In *ASICON 2005. 6th International Conference On ASIC, 2005*, volume 2, pages 685–688, 2005.

[17] X. Kavousianos, E. Kalligeros, and D. Nikolos. Efficient Test-Data Compression for IP Cores Using Multilevel Huffman Coding. In *DATE '06: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1033–1038, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[18] C. V. Krishna, A. Jas, and N. A. Touba. Achieving High Encoding Efficiency with Partial Dynamic LFSR Reseeding. *ACM Transactions on Design Automation of Electronic Systems*, 9(4):500–516, October 2004.

[19] M. Lange. Adopting the Right Embedded Compression Solution. *EE-Evaluation Engineering*, pages 32–40, May 2005.

[20] H. K. Lee and D. S. Ha. On the Generation of Test Patterns for Combinational Circuits. Technical Report 12-93, Dept. of Electrical Eng., Virginia Polytechnic Institute and State University, 1993.

[21] H. K. Lee and D. S. Ha. HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(9):1048–1058, September 1996.

[22] J. Lee and N. A. Touba. Low Power Test Data Compression Based on LFSR Reseeding. In *ICCD04: Proceedings of the IEEE International Conference on Computer Design*, 2004.

[23] L. Li and K. Chakrabarty. Test Set Embedding for Deterministic BIST Using a Reconfigurable Interconnection Network. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(9):1289–1305, September 2004.

[24] 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 *VLSID05: Proceedings of the 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design*, 2005.

[25] L. Li, K. Chakrabarty, S. Kajihara, and S. Swaminathan. Three-Stage Compression Approach to Reduce Test Data Volume and Testing Time for IP Cores in SOCs. *IEE Proc. Comput. Digit. Tech.*, 152(6):704–712, November 2005.

[26] L. Li, K. Chakrabarty, and N. A. Touba. Test Data Compression Using Dictionaries with Selective Entries and Fixed-Length Indices. *ACM Transactions on Design Automation of Electronic Systems*, 8(4):470–490, October 2003.

[27] I. Pomeranz and S. M. Reddy. Reducing the Number of Specified Values Per Test Vector by Increasing the Test Set Size. *IEE Proceedings - Computers and Digital Techniques*, 153(1):39–46, 2006.

[28] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee. Embedded Deterministic Test. *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, 23(5):776–792, May 2004.

[29] X. Ruan and R. Katti. An Efficient Data-Independent Technique for Compressing Test Vectors in Systems-on-a-Chip. In *ISVLSI '06: Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, page 153, Washington, DC, USA, 2006. IEEE Computer Society.

[30] S. Samaranayake, E. Gizdarski, N. Sitchinava, F. Neuveux, R. Kapur, and T. Williams. A Reconfigurable Shared Scan-in Architecture. In *VTS'03: Proceedings of the 21st IEEE VLSI Test Symposium*, April 2003.

[31] M. A. Shah and J. H. Patel. Enhancement of the Illinois Scan Architecture for Use with Multiple Scan Inputs. In *Proc. IEEE Ann. Symp. on VLSI*, pages 167–172, 2004.

[32] Y. Shi, N. Togawa, S. Kimura, M. Yanagisawa, and T. Ohtsuki. FCSCAN: An Efficient Multiscan-Based Test Compression Technique for Test Cost Reduction. In *ASP-DAC '06: Proceedings of the 2006 Conference on Asia South Pacific Design Automation*, pages 653–658, New York, NY, USA, 2006. ACM Press.

[33] N. Sitchinava, S. Samaranayake, R. Kapur, E. Gizdarski, F. Neuveux, and T. W. Williams. Changing the Scan Enale design Shift. In *VTS 2004: Proceedings of the 22nd IEEE VLSI Test Symposium*, 2004.

[34] H. Tang, S. M. Reddy, and I. Pomeranz. On Reducing Test Data Volume and Test Application Time for Multiple Scan Chain Designs. In *ITC '03: Proc. of the Intl. Test Conf.*, pages 1079–1087, 2003.

[35] M. Tehranipoor, M. Nourani, and K. Chakrabarty. Nine-Coded Compression Technique for Testing Embedded Cores in SOCs. *IEEE Trans. Very Large Scale Integr. Syst.*, 13(6):719–731, June 2005.

[36] N. A. Touba. Survey of Test Vector Compression Techniques. *IEEE Design & Test of Computers*, pages 294–303, 2006.

[37] A. Wurtenberger, C. S. Tautermann, and S. Hellebrand. Data Compression for Multiple Scan Chains Using Dictionaries with Corrections. In *ITC '04: Proceedings of the International Test Conference on International Test Conference*, pages 926–935, Washington, DC, USA, 2004. IEEE Computer Society.