Correlation-Based Rectangular Encoding

Jinkyu Lee, *Member, IEEE*, and Nur A. Touba, *Fellow, IEEE*

***Abstract*—In this paper, a technique is presented for improving the compression achieved with any linear decompressor by adding a small nonlinear decoder that exploits bit-wise and pattern-wise correlations present in test vectors. The proposed nonlinear decoder has a regular and compact structure, and allows continuous-flow decompression. It has a very important feature, which is that its design does not depend on the test data. This simplifies the design flow and allows the decoder to be reused when testing multiple cores on a chip. Experimental results show that combining a linear decompressor with the small nonlinear decoder proposed here significantly improves the overall compression.**

***Index Terms*—Linear decompression, nonlinear decompression, rectangular encoding, test vector compression.**

I. Introduction

With the advent of system-on-a-chip (SoC) and 3-D ICs (3D-ICs), test data volume requirements have increased dramatically to achieve high test quality as size and complexity continue to grow [1]. To obtain high test quality, test data volume may exceed the memory capacity of the available automatic test equipment (ATE). Furthermore, the large amount of test data needs to be transferred from the ATE to a chip with limited tester bandwidth, which results in long test time. To overcome increased test memory requirements and tester data bandwidth requirements, test vector compression has become very important. Test vector compression provides a way of reducing both the tester memory requirement and the tester data bandwidth requirement. A number of test vector compression techniques have been proposed in the literature.

A special class of test vector compression schemes involves the use of a linear decompressor, which uses only linear operations to decompress the test vectors. This includes techniques based on linear feedback shift register (LFSR) reseeding and combinational linear expansion circuits consisting of XOR gates. Linear compression schemes are very efficient at exploiting don't care values in the test cubes to achieve large amounts of compression.

Linear decompressors expand seeds to deterministic test cubes. A seed is an initial state of the linear decompressor that is expanded by running the linear decompressor. Given a deterministic test cube, a corresponding seed can be computed by solving a set of linear equations (one equation for each specified bit) based on the feedback polynomial of linear decompressor. Typically, since only 1%–5% of the bits in a test vector are specified, most bits in a test cube do not need to be considered when a seed is computed because they are don't care bits. Therefore, the size of a seed is much smaller than the size of a test vector. Consequently, linear decompressors can significantly reduce test data storage and bandwidth.

The amount of compression that can be achieved with linear compression schemes depends directly on the number of specified bits in the test cubes. While linear decompressors are very efficient at exploiting don't cares in the test set, they cannot exploit correlations in the test cubes, and hence, they cannot compress the test data to less than the total number of specified bits in the test data. Nonlinear decompressors, on the other hand, can exploit correlations in the test cubes, but are not as efficient as linear decompressors in exploiting don't cares. Since test data are typically only 1%–5% specified with the rest as don't cares, linear decompressors are generally more effective overall. This fact coupled with the simple and compact design of linear decompressors is the main reason why they are used in commercial tools.

The approach taken in this paper is to combine linear and nonlinear compressions together to get the advantages of both. A nonlinear decompressor is used to exploit correlations in the specified bits to reduce the number of specified bits that the linear decompressor has to produce. Since the amount of compression achieved with a linear decompressor depends on the number of specified bits it needs to produce, this approach results in a much greater compression than what the linear decompressor could achieve by itself (preliminary results were presented in [6]).

A block diagram of the proposed scheme is shown in Fig. 1. A rectangular decoder (which is a sequential nonlinear decompressor) is placed between the linear decompressor and the scan chains. The rectangular decoder exploits bit-wise and pattern-wise correlations in the test cubes to reduce the number of specified bits. Consequently, the input data to the rectangular decoder have many fewer specified bits than the test cubes themselves. This makes the job of the linear decompressor easier since it now needs to produce significantly fewer specified bits. The number of bits required to be stored on the tester and transferred to the linear decompressor basically goes down linearly with the number of specified bits that it needs to produce, as can be seen in the data reported in [2], and [7]–[9].

II. Related Work

The first linear compression scheme that employs LFSR as a linear decompressor was introduced in [10], where it was shown that if $s_{\max}$ is the largest number of specified bits in any test cube, then for an LFSR of length $s_{\max} + 20$ bits, the probability of not being able to find a seed for some test cube is

Manuscript received January 14, 2009; revised April 30, 2009. First published September 22, 2009; current version published September 24, 2010.

J. Lee is with Intel Corporation, Austin, TX 78746 USA (e-mail: jinkyu_john@hotmail.com).

N. A. Touba is with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712 USA (e-mail: touba@ece.utexas.edu).

Digital Object Identifier 10.1109/TVLSI.2009.2025882

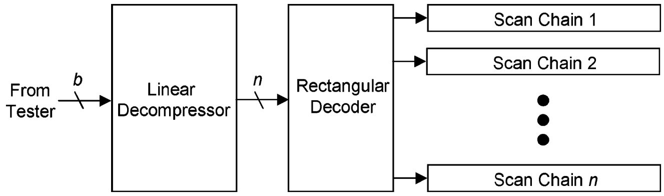


Fig. 1. Diagram of the proposed scheme.

less than 10^{-6}. Several techniques were proposed to improve the encoding efficiency of the basic scheme in [10]. Utilizing multiple-polynomial LFSRs was introduced in [11] and [12]. An LFSR reseeding scheme with variable-length seed was proposed in [13]. More recent work has focused on dynamic LFSR reseeding, where the seed is incrementally modified as the LFSR runs [2], [3], [7], [8]. In dynamic LFSR reseeding, the size of a seed does not depend on $s_{\max}$, and thus, can be even smaller than the size of an LFSR. The linear compression techniques that utilize linear decompressors receiving test data from ATE in a continuous flow were described in [2], [3], [7], [9], and [14]–[21]. The continuous-flow linear decompressor is directly connected to ATE, thereby receiving test data as fast as ATE operates. The test data compression ratios of the previous works that employ only linear decompressor are limited by the number of specified bits in test cubes. To overcome this limitation, it is proposed in this paper to employ nonlinear decompressor on top of linear decompressor. The nonlinear decompressor reduces the number of specified bits, thereby compressing the test data smaller than the number of specified bits in the original test cubes.

There has been some previous work that has also combined linear and nonlinear codings together, but in fundamentally different ways than what is done here. The inputs to the linear decompressor were encoded using a noniinear code in [22]. The objective in [22] was to select the seeds for the LFSR in such a way that they could be effectively compressed by a nonlinear code. In the proposed scheme, the inputs to the scan chains are encoded with a nonlinear code. The objective here is to reduce the number of specified bits that need to be produced by the linear decompressor. Whereas the method in [22] is only applicable for LFSR reseeding, where the seed is periodically loaded, and the proposed scheme is applicable for *any* linear decompressor including combinational and sequential continuous-flow decompressors (for which the method in [22] cannot be used). Dictionary coding and LFSR reseeding are combined in [23] such that either one or the other is used to load each scan slice. In the proposed method, rectangular coding is combined with a linear decompressor and both are used together for all scan bit slices, enabling a continuous-flow decompression with greater efficiency. A combinational statistical decoder is combined with linear decompression in [24]. The proposed method uses a sequential decoder that can exploit correlations across scan slices as well as across patterns. Results in [22]–[24] are compared with results in the proposed study in Section VI. Note that [25] also combines linear and nonlinear codings, but it is for a hybrid built-in self-test (BIST) application in which many more patterns are applied to the circuit under test (CUT) than the number of patterns in a deterministic test set.

In addition to the aforementioned differences, there are two additional key features that distinguish the proposed scheme from earlier work. The most important is that the design of the decoder for the proposed scheme is independent of the test set. In all the earlier test vector compression techniques that combine linear and nonlinear compressions, the design of the nonlinear decoder is customized for the test set. Having a fixed independent design for the nonlinear decoder is a major advantage as it simplifies the design flow, allows for late engineering changes to the test set, and allows the decoder to be reused when testing multiple cores on a chip. Furthermore, unlike the earlier techniques, which use a combinational nonlinear decoder, the proposed scheme uses a sequential nonlinear decoder that is able to exploit correlations across scan slices and across patterns, thereby making it more effective.

III. Rectangular Encoding

One well-known characteristic of the test data is that certain bit positions in test cubes tend to be correlated across many patterns. This arises from the fact that many faults in the circuit require similar input assignments to detect. A number of BIST schemes exploit this characteristic of test data including weighted pattern testing, STAR-BIST [26], and folding counters [27]. In weighted pattern testing, each weight set targets a subset of the test cubes that have highly correlated values in a subset of the bit positions. If one represents the test set as a test matrix in which each row corresponds to a test cube and each column corresponds to a bit position, then the correlations tend to exist in rectangles in this matrix. The idea with rectangular encoding is to encode these rectangles with a small number of specified bits, and then, have a simple decoder that decodes them. Since the rectangles have a regular structure, the decoder design is simple and independent of the test data.

A. Overview

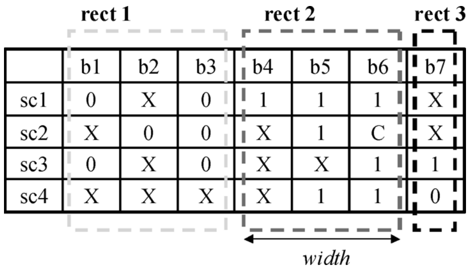
The first step in rectangular encoding is to partition the test cubes into clusters such that the pattern-wise correlation within a cluster is maximized. This is done by using a clustering algorithm that will be described in Section III-D. Each cluster is then encoded as one unit. If there are n scan chains, then each scan slice consists of the n bits that are shifted into the n scan chains in a clock cycle. A test cube with m bits consists of m/n scan slices. In rectangular encoding, the scan slices for a test cube are partitioned into a sequence of variable-length rectangles. All the test cubes within each test cube cluster are partitioned identically. So, in effect, the entire test matrix is partitioned into rectangles, where the height of each rectangle is determined by the number of test cubes in the test cube cluster it belongs to, and the width is determined by the scan slice partitioning for the test cube cluster it belongs to. A heuristic procedure will be described in Section III-B for partitioning the scan slices to maximize compression.

Within each rectangle, the largest set of scan chains that has compatible values is identified. This set of scan chains must have either a 1(0) or X for every scan slice across the width of the rectangle and every test cube across the height of the rectangle. A chain select mask is then defined for the rectangle that identifies which scan chains should be loaded from the linear decom-

t1	b1	b2	b3	b4	b5	b6	b7
sc1	0	X	X	1	1	1	X
sc2	X	0	0	X	1	**1**	X
sc3	0	X	X	X	X	1	1
sc4	X	X	X	X	1	1	0

t2	b1	b2	b3	b4	b5	b6	b7
sc1	X	X	0	1	X	1	X
sc2	X	0	0	X	1	**0**	X
sc3	0	X	0	X	X	X	1
sc4	X	X	X	X	1	1	0

Fig. 2. Example of test cubes.



	b1	b2	b3	b4	b5	b6	b7
sc1	0	X	0	1	1	1	X
sc2	X	0	0	X	1	C	X
sc3	0	X	0	X	X	1	1
sc4	X	X	X	X	1	1	0

Fig. 3. Example of rectangles.

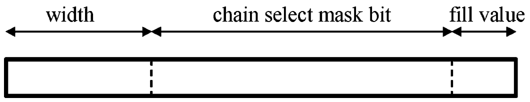


Fig. 4. Format of rectangle control data.

pressor and which scan chains should be filled with a specified fill value (0 or 1). So the information that is needed to decode each rectangle in a particular test cube cluster consists of three things. The width of the rectangle, the chain select mask, and the fill value. This is illustrated in the small example shown in Figs. 2–4.

In Figs. 2 and 3, there are seven scan slices (columns) and four scan chains (rows). For simplicity, it is assumed in this example that there are only two test cubes in a cluster, as shown in Fig. 2. In Fig. 3, the bit compatibility for the test cubes in the test cube cluster is shown, where a value of *X* indicates all the test cubes in the cluster have *X* for that scan cell, a value of 1(0) indicates that they all have either 1(0) or *X* for that scan cell, and a value of C indicates a conflict where both 1 and 0 are present. The scan slices are partitioned into three rectangles.

Fig. 4 shows the format for rectangular control data, and Fig. 5 shows the specific values for the three rectangles in Fig. 3. The width of the rectangle in terms of scan slices is encoded as a binary number. The maximum width of a rectangle is a user-defined parameter and determines the number of bits allocated for specifying the width (experimental data for different maximum widths is discussed in Section VI). The chain select mask contains 1 bit for each scan chain to indicate if the scan chain should be loaded from the linear decompressor or loaded with the fill value. The last bit indicates the fill value (either 1 or 0). In the second rectangle rect2, all the bits in $sc1$, $sc3$, and $sc4$ can be filled with 1 (this is not the case for $sc2$ because there is a conflict in scan slice 6). As can be seen in Fig. 5, the chain select mask in this case would be 1011 that would load all the scan chains except $sc2$ with the fill value of 1 ($sc2$ would be loaded from the linear decompressor). In rect1, all the scan chains except $sc4$ can be loaded with the fill value. Moreover, in this case, $sc4$ has only *X* values, and thus, it does not matter whether it is loaded with the fill value or from the linear decompressor, and therefore, the chain select mask is $111X$ in this case. The last rectangle rect3 is only one scan slice wide. For very narrow rectangles, it is generally more efficient to simply load them from the linear decompressor, and not bother specifying a chain select mask and fill value. For this reason, if the width of a rectangle is below some user-defined minimum threshold, the chain select mask is simply ignored and all the scan chains are filled from linear decompressor. The advantage of this is that as can be seen in Fig. 5, the chain select mask and fill value are simply don't cares for rect3.

rect 1	1	1	1	1	1	X	0
rect 2	1	1	1	0	1	1	1
rect 3	0	1	X	X	X	X	X

Fig. 5. Control data for three rectangles.

B. Partitioning Scan Slices Into Rectangles

The reduction in the number of specified bits that the linear decompressor has to produce (and hence, the amount of compression achieved) for each rectangle depends on the number of control bits that need to be specified for decoding the rectangle versus the number of specified bits in the test cubes that get covered with the fill value (and hence do not need to be generated by the linear decompressor). The goal in partitioning the scan slices for a test cube cluster into rectangles is to achieve the greatest overall reduction in the number of specified bits. A greedy heuristic procedure for this is described in this section.

The first step is to generate a compatibility cube for the test cube cluster. This is illustrated in Fig. 3 and has been explained earlier. From the compatibility cube, the rectangle that provides the largest reduction in specified bits is identified. This is done by considering each scan slice as a starting point for a rectangle and considering all possible rectangle widths (up to the user-defined maximum rectangle width) from that starting point. Once the best rectangle is identified, it is marked as selected and the procedure repeats, taking into consideration that rectangles cannot overlap. Rectangles continue to be selected in a greedy manner until all scan slices have been included in a rectangle.

C. Reducing Size of Chain Select Mask

The amount of control data that needs to be specified for decoding the rectangles is typically dominated by the bits for the chain select mask. One way to reduce these data is that instead of using 1 bit in the chain select mask for each scan chain, 1 bit can be used per k scan chains. This reduces some of the flexibility since now all k scan chains controlled by the same bit in the chain select mask need to be compatible in order to use the fill value. However, it generally provides greater compression

t1	x	0	0	0
t2	x	0	0	1
t3	0	x	0	x
t4	0	x	1	x

Fig. 6. Example of clustering.

since the control data are significantly reduced. Experimental results are shown in Section VI for different values of k.

D. Forming Test Cube Clusters

In rectangular encoding, the test cubes are partitioned into clusters, and each cluster is then divided into rectangles. Some effective algorithms for this type of clustering were described in [28]. A similar approach is taken here, but using a different benefit function to maximize correlation within a cluster and also minimize the number of clusters.

In order to maximize the compression achieved for each rectangle, it is important that the test cubes in each cluster have many bit positions with compatible values. As more test cubes are added to a cluster, the height of each rectangle increases. This has the benefit of amortizing the control bits required for decoding each rectangle over more test cubes, but there is a tradeoff as more bit positions are likely to have conflicts (thereby increasing the number of Cs in the compatibility cube), and thus reducing the effectiveness of each rectangle. A greedy clustering procedure that takes this tradeoff into consideration is described here.

One test cube is used as a seed for the cluster. All other test cubes are then considered as candidates to add to the cluster. The heuristic that is used to measure the optimality of a cluster is the total number of specified bits that are present in each compatible bit position of the cluster. This value forms a benefit function for the cluster. It is computed by considering each compatible bit position and adding up the number of test cubes that have a specified value in that bit position. Consider the test cubes in Fig. 6. A cluster consisting of test cubes $t1$, $t2$, and $t3$ is compatible in the first 3 bit positions, and the total number of specified bits in these 3 bit positions is 6 (the *X*s are not counted). The change in this benefit function is computed for adding each candidate test cube. The one that gives the greatest improvement in the benefit function is added to the cluster. This continues until a point is reached, where no positive improvement in the benefit function can be obtained by adding another test cube to the cluster. For example, in Fig. 6, if the test cube $t4$ was added to the cluster, then the benefit function would actually decrease because bit position 3 would no longer be compatible. Since the final cluster is very dependent on the initial seed, all test cubes are used as seeds and the best resulting cluster is selected. This process is repeated iteratively for the remaining test cubes until all test cubes are members of a cluster.

Note that while a greedy clustering procedure is described here, any clustering procedure can be used to maximize the benefit function defined before. The flow diagram of the proposed clustering algorithm is shown in Fig. 7. The time complexity is $O(n^2)$, assuming n is the number of test cubes. The CPU time with a 1.3-GHz machine for the overall encoding algorithm is shown in Fig. 8.

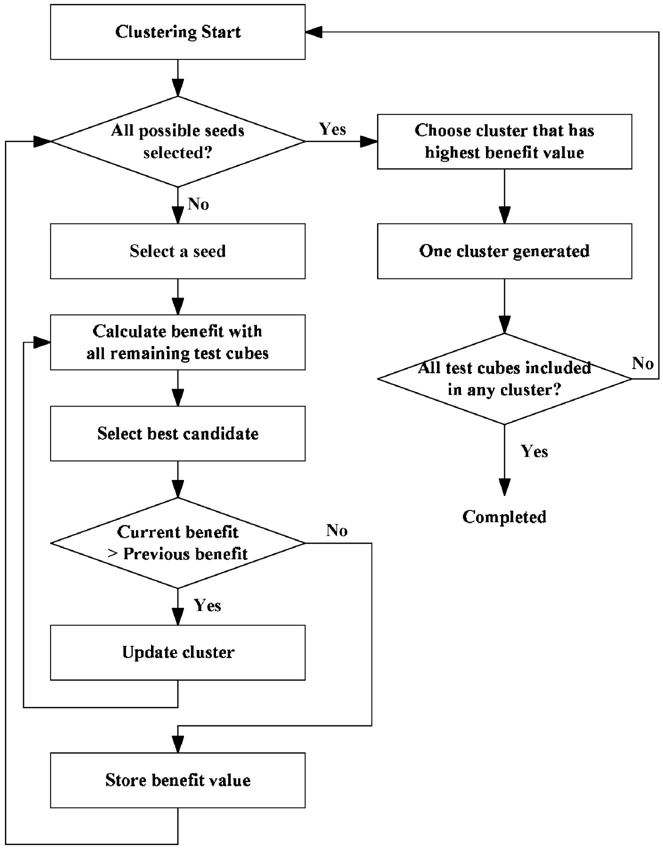

Clustering Start
All possible seeds selected?
Yes
Choose cluster that has highest benefit value
No
Select a seed
One cluster generated
Calculate benefit with all remaining test cubes
All test cubes included in any cluster?
No
Select best candidate
Yes
Completed
Current benefit > Previous benefit
No
Yes
Update cluster
Store benefit value

Fig. 7. Flow diagram of the clustering algorithm.

Circuit	Test Cube	CPU time
s13207	266	24m
s15850	269	20m
s38417	376	2h 25m
s38584	296	1h 43m

Fig. 8. CPU time for the proposed encoding scheme.

IV. Rectangular Decoder

Decoding of the rectangles is done with a sequential nonlinear decoder that is placed between a linear decompressor and the scan chains. A block diagram for the rectangular decoder is shown in Fig. 9. It consists of a controller that is a small finite-state machine, a RAM that stores the rectangular control data, a RAM address pointer that points to the control data for the next rectangle, a width counter, and a rectangular control register that stores the control data for the current rectangle (rectangle width, chain select mask, and fill value). Note that a RAM that is present for functional purposes can be utilized in the rectangular decoder (it is not necessary to add an extra RAM). A MUX is placed in front of each scan chain. The select line to the MUX is the bit in the chain select mask that corresponds to that scan chain. Note that if $k > 1$, then one bit in the chain select mask will fan out to k MUXes. Depending on the corresponding value in the chain select mask, each scan chain will either be loaded with the fill value or be loaded from the linear decompressor. Note that if the rectangle width is below the user-defined threshold, then the scan chain is loaded from the linear decompressor regardless of the value of the chain select mask.

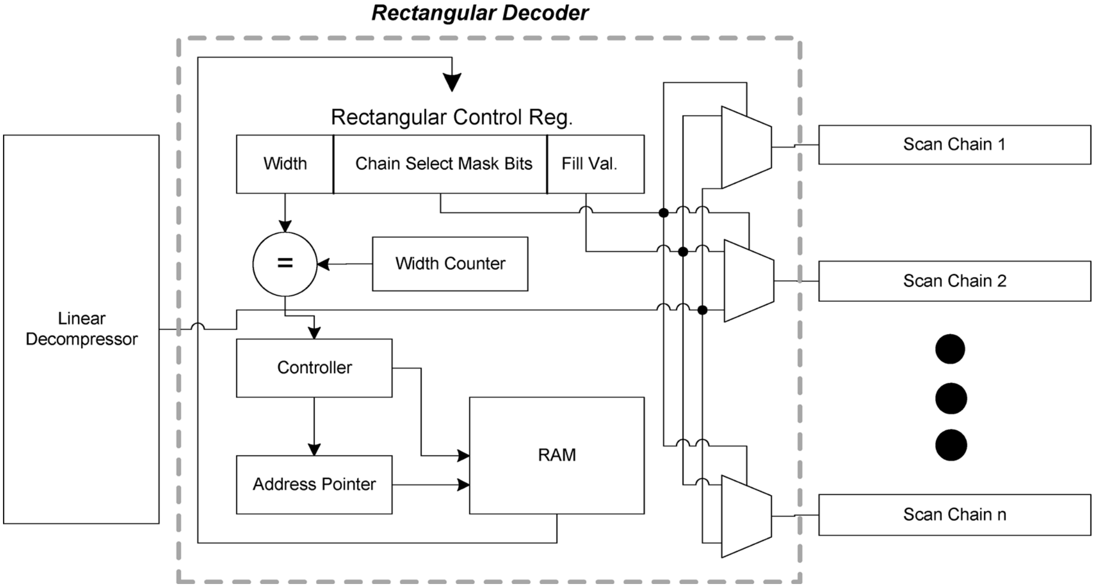


Fig. 9. Block diagram for rectangular decoder.

This is implemented by adding another MUX whose select line comes from a less-than comparator that checks the value of rectangle width. Note that this is not shown in Fig. 9 for sake of readability.

The RAM holding the rectangular control data can be loaded through the linear decompressor from the tester, either all at the beginning of the test session or incrementally during the test session. If it is all loaded at the beginning, the entire rectangular control data are transferred through the linear decompressor to the RAM. In this case, the RAM must be large enough to store all the rectangular control data. And one additional bit per test cube is required to indicate whether the current test cube is in the same cluster as the previous test cube or not. Note that the number of outputs of a phase shifter should be larger than or equal to the size of the rectangular control register to load the rectangular control data into the RAM efficiently. As explained in Section III-C, one bit in the chain select mask can be used per k scan chains. By increasing the value of k, the size of the rectangular control register can be reduced, which can be also helpful to increase compression ratio by reducing the amount of the control data.

The other option is to incrementally load the rectangular control data each time a new test cube cluster is started. In this case, only the rectangular control data for one cluster need to be stored on-chip at a time. Thus, the required RAM size would only depend on the maximum number of rectangles in any cluster. Note that if configuring the RAM for storing test data is not possible, additional registers that can store the rectangular control data for one cluster can substitute for the RAM. The size of the additional register depends on the maximum number of rectangles in any cluster.

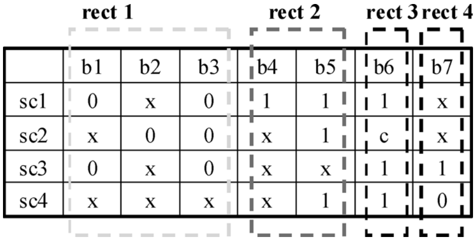
The test set is ordered so that all the test cubes in a cluster come in succession. An extra clock cycle is added at the start of each test cube in which the linear decompressor generates one specified bit to indicate to the controller whether or not this is the start of a new test cube cluster. If it is not the start of a new cluster, then the same rectangular data that were used for the previous test cube is used for this one (the RAM pointer is simply reset back to the first rectangle for this cluster). If it is the start of a new cluster, then there are two cases. If the rectangular control data are to be loaded incrementally, it is done at this point (only the data needed for this cluster). If the rectangular control data were all loaded into the RAM at the start, then the RAM address pointer is incremented to point to the start of the rectangular control data for this new cluster.

After this, each rectangle is decoded one at a time as the test cube is shifted into the scan chains. For each rectangle, the controller loads the rectangular control data from the RAM into the rectangular control register, and the width counter is reset to 0. As each scan slice is loaded into the scan chains, the width counter is incremented. When it becomes equal to the rectangle width, then the next rectangle is loaded from the RAM into the rectangular control register and the RAM pointer is incremented. This process repeats until the entire test cube has been shifted in.

As can be seen, the rectangular decoder is simple, compact, and regular. A very proficient feature is that it does not depend on the actual test data. It can be designed so that it is capable of decoding any set of rectangles. This simplifies the design flow since there is no need to have the test data when implementing the decoder.

V. Enhanced Encoding Scheme

In addition to the scheme described so far, an enhancement has been developed to maximize the advantage of rectangular encoding. The total number of specified bits required in the proposed scheme depends on two factors: the number of specified bits in the encoded test cubes and the number of specified bits in the control data. This section describes how to reduce both the number of specified bits in the encoded test cube and the number of specified bits in the control data further.



	b1	b2	b3	b4	b5	b6	b7
sc1	0	x	0	1	1	1	x
sc2	x	0	0	x	1	c	x
sc3	0	x	0	x	x	1	1
sc4	x	x	x	x	1	1	0

Fig. 10. Rectangles in secondary encoding.

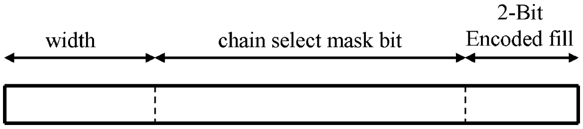


Fig. 11. Format of rectangle control data in secondary encoding.

rect 1	1	1	x	x	x	x	1	0
rect 2	1	0	x	x	x	x	1	1
rect 3	x	1	1	0	1	1	0	0
rect 4	x	x	x	x	1	0	0	1

Fig. 12. Control data for four rectangles in Fig. 8.

A. Secondary Encoding for Control Data

The secondary encoding scheme for control data is motivated by the following two factors.

1) There are many rectangles that have both ‘‘fill-0’’ and ‘‘fill-1.’’ If a 1 bit fill value is employed, only one binary value, either 0 or 1, can be filled, and the other value should be loaded by linear decompressor even though the bit position may have high pattern-wise correlation.
2) To fill more specified bits in the test cubes by the nonlinear decompressor, a larger number of rectangles should be employed. However, the more rectangles, the more specified bits in the total control data. Therefore, it is important to reduce the number of specified bits per rectangle, thereby allow more rectangles without a significant increase in the number of specified bits in total control data.

The rectangles for the example in Fig. 3 are shown in Fig. 10. The format of the rectangle control data in the secondary encoding is illustrated in Fig. 11. Fig. 12 shows the control data for the four rectangles in Fig. 10, using the format in Fig. 11.

As shown in Fig. 11, instead of only a 1-bit fill value, a 2-bit encoded fill value is utilized to resolve the aforementioned two issues. There are four modes and each mode is explained as follows.

Fill 1: The 2-bit fill value is ‘‘11.’’ All the bits in the corresponding rectangle are filled with ‘‘1.’’ Therefore, all the chain select mask bits are don’t care bits. Only the width is specified. rect2 is an example of this mode in Fig 12.

Fill 0: The 2-bit fill value is ‘‘10.’’ All the bits in the corresponding rectangle are filled with ‘‘0.’’ There are no specified bits in the chain select mask bits. Only the width is specified. rect1 is in an example of this mode in Fig. 12.

Fill 0/1: The 2-bit fill value is ‘‘01.’’ Some bit positions in the rectangle are filled with ‘‘0’’ and some of the other bit positions are filled with ‘‘1.’’ However, there is no conflict in any bit position in the rectangle. rect4 is an example of this mode in Fig. 12. In this mode, the chain select mask bit contains fill values for each scan chain. Different from what is described in Section II, the chain select mask bits in this case do not indicate which value is loaded (i.e., from the linear decompressor or from the fill value) into scan chains. In this mode, since it is ensured that there are no conflicts in any bit position and all the bit positions can be fixed to either 0 or 1, the chain select mask bits are utilized to indicate which value can be fixed for each scan chain. Note that the width of the rectangle with this mode is always set to 1, which means that the rectangle is composed of one scan slice. This makes the width bits don’t care bits in this mode, thereby reducing the number of specified control bits.

Fill with c: The 2-bit fill value is ‘‘00.’’ There is a conflict in some bit position. rect3 is an example of this mode in Fig. 12. The bit position that has a conflict is loaded from the linear decompressor. The chain select mask bits state which scan chain is loaded from the linear decompressor (where the chain select mask bit is 0) and which scan chain is loaded by filling (where the chain select mask bit is 1), as described in Section II. In this mode, the width of the rectangle is always set to 1 similar to the ‘‘01’’ mode. Except for MSB, all the other bits in the width control data are assigned to don’t care bits. The MSB is set to the bit that is filled into scan chains, where corresponding chain select mask bits have 1.

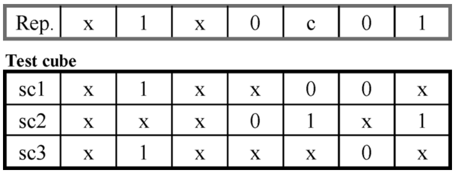
The secondary encoding increases the number of rectangles for each cluster; however, it also increases the number of specified bits in the test cube that are filled by the fill value. The number of control bits per rectangle is significantly decreased compared with the use of a 1-bit fill value. Note that the number of rectangles increases significantly with the secondary encoding method. However, since the increased number of rectangles makes the number of bits that must be specified by the linear decompressor minimal, the overall number of specified bits is reduced compared to use of a 1-bit fill value. According to the experimental results, except for two cases (10 scan chains and 20 scan chains in s15850), the number of total specified bits is reduced in all the other cases. The secondary encoding scheme adds only a very small amount of extra combinational decoding logic (a few additional MUXes) to the rectangular decoder shown in Fig. 9.

B. Test Cube Clustering Based on Scan Chain Information

The clustering algorithm described in Section II-D has one form of the clusters regardless of the number of scan chains for a test set. It maximizes pattern-wise and bit-wise correlations in each rectangle. However, the clustering algorithm does not count for the correlation among scan chains in a rectangle. A rectangle contains bits in multiple scan chains, and in order to fix all the bits over all scan chains in the rectangle, the correlation among the bits in the scan chains is as important as pattern-wise

TABLE I
RESULTS FOR RECTANGULAR ENCODING SCHEME

Circuit	Test Cubes	Original Spec. Bits	Num. Clusters	Scan Chains	Num. Rect.	Rect. Control			Data Spec. Bits	Control Spec. Bits	Total Spec. Bits	Reduction (%)	RAM (bits)
						W	C	T					
s13207	266	9389	8	10	86	4	5	10	5774	1126	6900	26.5	130
				20	75	4	10	15	5665	1391	7056	24.8	150
				30	54	4	15	20	6217	1346	7563	19.5	180
s15850	269	10944	9	10	75	4	5	10	5707	1019	6726	38.5	100
				20	56	4	10	15	6190	1109	7335	33.0	135
				30	49	4	15	20	6602	1249	7851	28.3.	160
s38417	376	30669	7	20	107	4	10	15	19880	1981	21861	28.7	255
				30	71	4	15	20	19306	1796	21102	31.2	280
				40	64	4	20	25	19735	1976	21711	29.2	250
s38584	296	26185	8	20	135	3	10	15	19331	2321	21652	17.3	300
				30	108	3	15	20	19693	2348	21941	16.2	320
				40	101	3	20	25	19508	2720	22228	15.1	375



Rep.	x	1	x	0	c	0	1

Test cube

sc1	x	1	x	x	0	0	x
sc2	x	x	x	0	1	x	1
sc3	x	1	x	x	x	0	x

Fig. 13. Representative cube generation.

correlation and bit-wise correlation. The improved clustering algorithm described in this section considers the scan-chain-wise correlation in addition to the pattern-wise and bit-wise correlations. This can be accomplished by simply adding one more step in the clustering algorithm described in Section II-D.

Before running the clustering algorithm, each test cube is translated into a representative cube. The representative cube contains correlation information among scan chains. In the representative cube, each bit position indicates the specified bit in each scan slice. Fig. 13 shows an example of the representative for a test cube. If a scan slice has 1 and *X* only, then the corresponding bit position is a ''1.'' If there are both 0 and 1 in a scan slice, the corresponding bit position in the representative cube has ''*c*,'' which means that the scan slice is not correlated (i.e., it has a conflict).

The representative cubes for all test cubes in a test set are created and given to the clustering algorithm described in Section II-D instead of the test cubes. Since the representative cube can be different based on the number of scan chains, the configuration of clusters depends on the number of scan chains. Using the representative cubes, the clustering method can consider the correlation among scan chains as well as bit-wise and pattern-wise correlations, thereby providing more efficient clusters for the rectangular encoding scheme.

VI. Experimental Results

Experiments were performed on the four largest ISCAS-89 circuits. The test cubes used in the experiments were generated in the following way. *Atalanta* generated uncompacted test cubes, and then, bit stripping was performed [29]. Finally, the test cubes were merged to minimize the number of specified bits. In Table I, the number of test cubes and the number of specified bits in deterministic test sets are shown in the second and third columns. The fourth column shows the number of clusters obtained with the pattern clustering algorithm described in Section II-D. Results for the proposed method were generated for three different numbers of scan chains. In the sixth column, the number of total rectangles across all clusters is shown. The next three columns show the size of a rectangle control data required for each rectangle. ''W'' is the number bits used for the rectangle width, ''C'' is the number of chain select mask bits, and ''T'' is the number of total bits per rectangle (which is equal to $W + C$ plus 1 for the fill value). Note that in all cases, $k = 2$ (i.e., each chain select mask bit controlled two scan chains), and thus, C is equal to the number of scan chains divided by 2 in all cases.

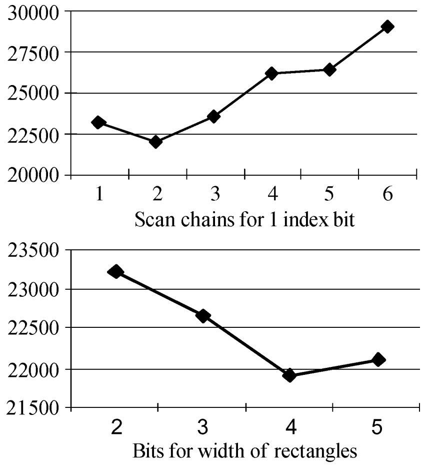


Fig. 14. Specified bits versus width and k value in s38417

Note that the graph in Fig. 14 (top) shows the total number of specified bits with different k values for $s38417$. As can be seen, the best result occurs for $k = 2$. The graph in Fig. 14 (bottom) shows the total number of specified bits using different numbers of bits for specifying the rectangle width (i.e., using different maximum rectangle widths) for $s38417$. The best result is observed when using 4 bits for the width. In all of the circuits except for $s38584$, the best result is observed using 4 bits, while for $s38584$ it is observed for 3 bits.

The total number of specified bits that the linear decompressor has to produce when using the proposed nonlinear

TABLE II
RESULTS FOR ENHANCED RECTANGULAR ENCODING SCHEME

Circuit	Test Cube	Original Spec. Bits	Num. Clusters	Scan Chain	Num. Rect.	Rect. Control			Data Spec. Bits	Cont. Spec. Bits	Total Spec. Bits	Reduction (%)	RAM (bits)
						W	C	T					
s13207	266	9389	14	10	415	4	10	16	4186	1822	6008	36.0	560
			24	20	502	3	20	25	4449	2307	6756	28.0	575
			27	30	509	3	30	35	5129	2302	7431	20.9	770
s15850	269	10944	16	10	478	4	10	16	6454	1468	7922	13.4	720
			41	20	780	3	20	25	5913	2712	8625	21.2	475
			48	30	801	3	30	35	5201	2512	7713	29.5	805
s38417	376	30669	38	20	1645	4	20	26	14807	6204	21011	31.5	1508
			68	30	2610	3	30	35	9757	11346	21103	31.2	1575
			87	40	3050	3	40	45	8906	12232	21138	31.1	1665
s38584	296	26185	38	20	1544	4	20	26	15736	4625	20361	22.2	1066
			73	30	2254	3	30	35	12073	8174	20247	22.7	1225
			77	40	2396	3	40	45	11594	10178	21772	16.9	1665

TABLE III
AREA OVERHEAD FOR CONTROL LOGIC

Circuit	Original Area	Width Counter (bits)	Num. 2-to-1 MUX	Rect. Control Register	RAM (bits)	Control Logic Area w/o Register	Area Overhead (%)	Control Logic Area w. Register	Area Overhead (%)
s13207	20751.1	4	10	10	130	260.0	1.25	1716.0	8.27
			20	15	150	394.0	1.90	2074.0	9.99
			30	20	180	528.0	2.54	2544.0	12.26
s15850	22856.0	4	10	10	100	260.0	1.14	1380.0	6.04
			20	15	135	394.0	1.73	1906.0	8.34
			30	20	160	528.0	2.31	2320.0	10.15
s38417	56701.7	4	20	15	255	355.0	0.63	3211.0	5.66
			30	20	280	450.0	0.79	3586.0	6.32
			40	25	250	545.0	0.96	3345.0	5.89
s38584	55327.4	3	20	15	300	338.5	0.61	3698.0	6.68
			30	20	320	433.5	0.78	4017.5	7.26
			40	25	375	528.5	0.96	4728.5	8.54

decoder is shown in the 12th column (this includes all the specified rectangular control data as well as the extra bit for each test cube to indicate if it is the start of a new cluster.) The percentage reduction in specified bits is shown in the next column. As can be seen, the number of specified bits that the linear decompressor has to produce is significantly reduced. This reduction in the specified bits is a very powerful result because it means that in most cases, up to an additional 30% or more, compression can be achieved on top of the best possible compression that is currently available for any linear decompression scheme. If the test data bandwidth is held constant, this translates to an equivalent reduction in test time. As the number of scan chains increases, the number of specified bits required for the proposed scheme increases slightly, but not much. The last column shows the size (in number of bits) of the RAM required to store the rectangular control data if it is incrementally loaded. Note that it is very small.

Table II shows results with the enhanced rectangular encoding scheme. The greatest difference between the original rectangular scheme and the enhanced scheme is that the number of rectangles employed in the enhanced scheme increases drastically compared to the number of rectangles in the original scheme (Table I). The large number of rectangles contributes to reducing the number of specified data bits even more than the original rectangular encoding. On the other hand, the number of specified control bits increases due to the large number of rectangles. As explained in Section V, the secondary encoding scheme is employed to minimize the increase in the number of specified control bits. As a result, more reduction in the number of specified bits can be achieved by the enhanced rectangular scheme, as shown in the 13th column in Table II.

TABLE IV
RESULTS COMBINED WITH PARTIAL RESEEDING

Circuit	LFSR Size	[8]		Proposed		
		Specified	Storage	Specified	Storage	Reduction
s13207	64	9389	9872	6900	7243	26.6%
s15850	66	10944	11322	6726	7176	36.6%
s38417	120	30669	31245	21102	21780	30.3%
s38584	120	26185	28312	21652	22199	21.6%

The area overhead for the control logic in the original rectangular encoding scheme is calculated based on standard cell area and shown in Table III. For the overhead calculation, the standard library for a TSMC 0.18-$\mu\mathrm{m}$ process [30] has been utilized. The second column actually shows the sum of widths of all the standard cells in each ISCAS-89 benchmark circuits. The height for all the cells is 5 $\mu\mathrm{m}$. Table III shows major components in the control logic from the third column to the sixth column. The RAM memory required for the proposed scheme is shown in the sixth column. Note that the RAM memory is not additional overhead if a functional RAM can be configured to be utilized as proposed. As shown in the eighth column, the area overhead in this case is very small (0.61% $\sim$ 2.54%). If a functional RAM cannot be used during test, then additional registers whose size

TABLE V
RESULTS COMPARED TO [22]–[24]

Circuit	Test Cube	[23]			Test Cube	[22]			[24]			Proposed		
		Specified	Storage	Comp.		Specified	Storage	Comp.	Specified	Storage	Comp.	Specified	Storage	Comp.
s13207	236	11313	11402	93.1%	266	9389	11285	93.9%	9389	9872	94.7%	7231	7678	95.9%
s15850	126	12657	7240	90.6%	269	10944	12438	92.4%	10944	11322	93.1%	6726	7176	95.6%
s38417	99	52582	51739	68.6%	376	30669	34767	94.4%	30669	31245	95.0%	21102	21780	96.5%
s38584	136	35287	30752	84.6%	296	26185	29397	93.2%	26185	28312	93.5%	21652	22199	94.9%

is equal to value in the sixth column are required to store rectangular control data for one cluster. In the scenario where a functional RAM cannot be used for test data decompression, the additional registers become a dominant component in the hardware overhead.

Results for combining the proposed scheme with an actual linear decompressor are shown in Table IV. The number of test patterns in the test set and the number of specified bits that need to be generated using the linear decompressor in [8] alone and using it with the proposed scheme are shown in Table IV. As can be seen, the reduction in test storage is very closely related to the reduction in specified bits. Note that the proposed scheme can be used with any linear decompressor.

We also compared the number of specified bits and tester storage required in the proposed scheme to the number of specified bits and tester storage required in [22]–[24] in Table V. The same test set is employed in [22], [24], and the proposed study, while [23] does not ([23] employs Mintest test set). As shown in Table V, the proposed scheme achieves the highest compression ratio in all the circuits, which is enabled by reduction in the number of specified bits. Compared with [23], more compression ratio is observed in the proposed study. Another important advantage over [23] is that the decoder in the proposed scheme is independent of circuit or test data, while [24] requires a dictionary that is dependent on circuit. Note that the Mintest test set in [23] has a smaller number of test cubes and more inconsistent in terms of the number of specified bits than the test set used in the proposed study.

Note that [24] reports the best tester storage results among compression schemes that use both linear and nonlinear techniques. In all the cases shown in [24], the proposed scheme reduces the number of specified bits more. Not only does the proposed scheme provide greater compression than previous schemes that combine linear and nonlinear compression techniques (i.e., [22]–[24]), it also allows continuous flow decompression and the design of the decoder is independent of the test data.

The percentage of specified bits in the test sets for the ISCAS-89 circuits (around 5%–20%) is typically much higher than what is reported for industrial circuits. Experiments were performed to see how effective the proposed scheme would be with much lower percentages of specified bits. A test set that has 2% specified bits was randomly generated with different degrees of correlation. The amount of correlation was controlled by a variable B% that determines both the bit-wise correlation and the pattern-wise correlation. Each bit has a 2% probability of being specified and 98% probability of being a don't care. If a bit is specified, then it has B% chance of having the same specified value as the previous specified bit in the test cube. Pattern-wise correlation is generated in the following way. If the previous test cube was specified in some bit position, then there is a 50% chance for the current test cube to be also specified in the same bit position and a B% chance of having the same specified value as the previous test cube. Fig. 15 shows how the percentage reduction in the number of specified bits varies with the amount of correlation. Test sets typically have quite a bit of correlation, so these data suggest that the proposed method can be quite effective.

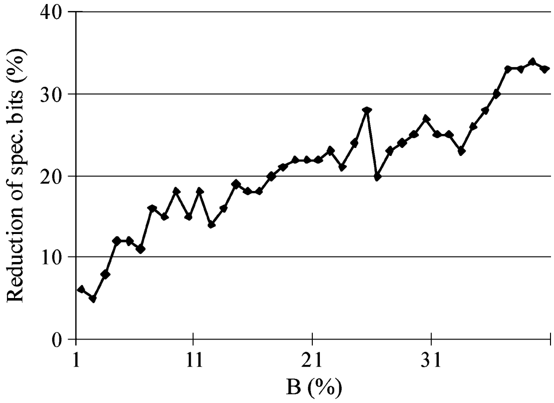


Fig. 15. Results for test set having few specified bits.

VII. CONCLUSION

The proposed scheme harnesses the power of linear and nonlinear decompressions together using a simple and compact decoder whose design is independent of the test set. Note that the compression could be significantly improved if scan chain reordering was employed along with the proposed scheme to increase bit-wise correlation.

REFERENCES

[1] A. Khoche and J. Rivoir, "I/O bandwidth bottleneck for test: Is it real," in *Proc. Int. Workshop Test Resour. Partitioning*, 2000, pp. 2.3-1–2.3-6.

[2] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, R. Tompson, T. Kun-Han, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eider, and G. Jun, "Embedded deterministic test for low cost manufacturing test," in *Proc. Int. Test Conf.*, 2002, pp. 301–310.

[3] B. Könemann, "A SmartBIST variant with guaranteed encoding," in *Proc. Asian Test Symp.*, 2001, pp. 325–330.

[4] M. Chandramouli, Synopsys, "How to implement deterministic logic Built-In Self-Test (BIST)," *Complier: Mon. Mag. Technol. Worldwide*, Jan. 2003.

[5] L.-T. Wang, X. Wen, H. Furukawa, F.-S. Hsu, S.-H. Lin, S.-W. Tsai, K. S. Abdel-Hafez, and S. Wu, "VirtualScan: A new compression scan technology for test cost reduction," in *Proc. Int. Test Conf.*, 2004, pp. 916–925.

[6] J. Lee and N. A. Touba, "Combining linear and non-linear test vector compression using correlation-based rectangular encoding," in *Proc. IEEE VLSI Test Symp*, 2006, pp. 251–257.

[7] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded deterministic test," *IEEE Trans. Comput.-Aided Des. Integr. Circuit Syst.*, vol. 23, no. 5, pp. 776–792, May 2004.

[8] C. V. Krishna, A. Jas, and N. A. Touba, "Test vector encoding using partial LFSR reseeding," in *Proc. IEEE Int. Test Conf.*, 2001, pp. 885–893.

[9] C. V. Krishna and N. A. Touba, "3-stage variable length continuous-flow scan vector decompression scheme," in *Proc. IEEE VLSI Test Symp.*, 2004, pp. 79–86.

[10] M. Könemann, "LFSR-coded test patterns for scan designs," in *Proc. Eur. Test Conf.*, 1991, pp. 237–242.

[11] S. Hellebrand, S. Tarnuck, J. Rajski, and B. Courtois, "Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers," in *Proc Int. Test Conf.*, 1992, pp. 120–129.

[12] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 223–233, Feb. 1995.

[13] N. Zacharia, J. Rajski, and J. Tyszer, "Decompression of test data using variable-length seed LFSRs," in *Proc. VLSI Test Symp.*, 1995, pp. 426–433.

[14] I. Hamzaoglu and J. H. Patel, "Reducing test application time for full scan embedded cores," in *Proc. Int. Symp. Fault Tolerant Comput.*, 1999, pp. 260–267.

[15] I. Bayraktaroglu and A. Orailoglu, "Test volume and application time reduction through scan chain concealment," in *Proc. Des. Autom. Conf.*, 2001, pp. 151–155.

[16] S. Mitra and K. Kim, "XMAX: X-tolerant architectures for maximal test compression," in *Proc. Int. Conf. Comput. Des.*, 2003, pp. 326–330.

[17] C. Krishna and N. Touba, "Adjustable width linear combinational scan vector decompression," in *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, 2003, pp. 863–866.

[18] A. Jas, B. Pouya, and N. Touba, "Virtual scan chains: A means for reducing scan length in cores," in *Proc. VLSI Test Symp.*, 2000, pp. 73–78.

[19] B. Könemann, "A SmartBIST variant with guaranteed encoding," in *Proc. Asian Test Symp.*, 2001, pp. 325–330.

[20] E. Volkerink and S. Mitra, "Efficient seed utilization for reseeding based compression," in *Proc. VLSI Test Symp.*, 2003, pp. 232–237.

[21] W. Rao, I. Bayraktaroglu, and A. Orailoglu, "Test application time and volume compression through seed overlapping," in *Proc. Des. Autom. Conf.*, 2003, pp. 732–737.

[22] C. V. Krishna and N. A. Touba, "Reducing test data volume using LFSR reseeding with seed compression," in *Proc. IEEE Int. Test Conf.*, 2001, pp. 321–330.

[23] X. Sun, L. Kinney, and B. Vinnakota, "Combining dictionary coding and LFSR reseding for test data compression," in *Proc. Des. Autom. Conf.*, 2004, pp. 944–947.

[24] I. S. Ward, C. Schattauer, and N. A. Touba, "Using statistical transformations to improve compression for linear decompressors," in *Proc. Defect Fault Tolerant VLSI Syst.*, 2005, pp. 42–50.

[25] H.-G. Liang, S. Hellebrand, and H.-J. Wunderlich, "Two-dimensional test data compression for scan-based deterministic BIST," in *Proc. Int. Test Conf.*, 2001, pp. 894–902.

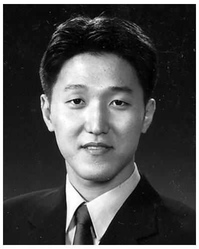
[26] K.-H. Tsai, J. Rajski, and M. Marek-Sadowska, "Star test: The theory and its applications," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 9, pp. 1052–1064, Sep. 2000.

[27] S. Hellebrand, H.-G. Liang, and H.-J. Wunderlich, "A mixed mode BIST scheme based on reseeding of folding counters," in *Proc. IEEE Int. Test Conf.*, 2000, pp. 778–784.

[28] R. Alleyne, "Clustering of test cubes: A procedure for the efficient encoding of complete test sets based on the intelligent reseeding of LFSRs," M.S. thesis, Dept. Elect. Comput. Eng., McGill University, Montreal, QC, Canada, 1994.

[29] R. Sankaralingam and N. A. Touba, "Controlling peak power during scan testing," in *Proc. IEEE VLSI Test Symp.*, 2002, pp. 153–159.

[30] Artisan Components, Sunnyvale, CA, "TSMC 0.18 $\mu\mathrm{m}$ process 1.8-Volt SAGE-X standard cell library databook," 2001.



Jinkyu Lee (M'09) received the B.S. degree from Yonsei University, Seoul, Korea, in 2001 and the M.S. and Ph.D. degrees from the University of Texas at Austin, Austin, TX, in 2004 and 2006, respectively, all in electrical and computer engineering.

He is currently a Component Design Engineer with Intel, Austin, TX.

Nur A. Touba (SM'05–F'09) received the B.S. degree from the University of Minnesota, Minneapolis, in 1990, and the M.S. and Ph.D. degrees from Stanford University, Stanford, CA, in 1991 and 1996, respectively, all in electrical engineering.

He is currently a Professor with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin.

Dr. Touba was a recipient of the National Science Foundation Early Faulty CAREER Award in 1997, the Best Paper Award at the 2001 VLSI Test Symposium, and the 2008 Defect and Fault Tolerance Symposium. He was elevated to IEEE Fellow in 2009. He served as program chair for the 2008 International Test Conference and general chair for the 2007 Defect and Fault Tolerance Symposium. He currently serves on the program committee for the Design Automation and Test in Europe Conference, International On-Line Test Symposium, European Test Symposium, Asian Test Symposium, Defect and Fault Tolerance Symposium, and International Test Synthesis Workshop.