Combining Linear and Non-Linear Test Vector Compression Using Correlation-Based Rectangular Encoding

Jinkyu Lee and Nur A. Touba

Computer Engineering Research Center
University of Texas, Austin, TX 78712

Abstract

A technique is presented here for improving the compression achieved with any linear decompressor by adding a small non-linear decoder that exploits bit-wise and pattern-wise correlation present in test vectors. The proposed non-linear 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 non-linear decoder proposed here significantly improves the overall compression.

1. Introduction

A special class of test vector compression schemes involves using a linear decompressor which uses only linear operations to decompress the test vectors. This includes techniques based on 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.

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. Non-linear 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. Because test data is 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 are the main reasons why they are used in commercial tools.

The approach taken in this paper is to combine linear and non-linear compression together to get the advantages of both. A non-linear 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 much greater compression than what the linear decompressor could achieve by itself.

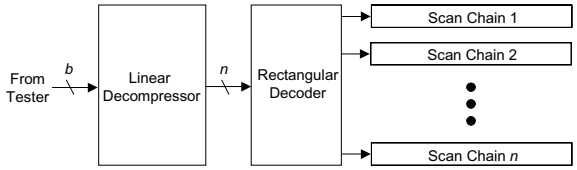


Figure 1. Diagram of Proposed Scheme

A block diagram of the proposed scheme is shown in Fig. 1. A rectangular decoder (which is a sequential non-linear 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 has 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 [Rajski 02] and [Krishna 01, 04].

There has been some previous work that has also combined linear and non-linear coding together, but in fundamentally different ways than what is done here. In [Krishna 02], the inputs to the linear decompressor were encoded using a non-linear code. The objective in [Krishna 02] was to select the seeds for the LFSR in such a way that they could be effectively compressed by a non-linear code. In the proposed scheme, the inputs to the scan chains are encoded with a non-linear 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 [Krishna 02] is only applicable for LFSR reseeding where the seed is periodically loaded, the proposed scheme is applicable for *any* linear decompressor including combinational and sequential continuous-flow decompressors (for which the method in [Krishna 02] cannot be used). In

Proceedings of the 24th IEEE VLSI Test Symposium (VTS'06)
0-7695-2514-8/06 $20.00 © 2006 IEEE

[Sun 04], dictionary coding and LFSR reseeding are combined 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. In [Ward 05], a combinational statistical decoder is combined with linear decompression. The proposed method uses a sequential decoder that can exploit correlations across scan slices as well as across patterns. Note that [Liang 01] also combines linear and non-linear coding, but it is for a hybrid 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 differences mentioned above, 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 of the earlier test vector compression schemes that combine linear and non-linear compression, the design of the non-linear decoder is customized for the test set. Having a fixed independent design for the non-linear 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. The second distinguishing feature of the proposed scheme is that unlike the earlier techniques which use a combinational non-linear decoder, the proposed scheme uses a sequential non-linear decoder that is able to exploit correlations across scan slices and across patterns thereby making it more effective.

2. Rectangular Encoding

One well-known characteristic of 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 [Tsai 00], [Rajski 03], and folding counters [Hellebrand 00]. 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.

2.1 Overview

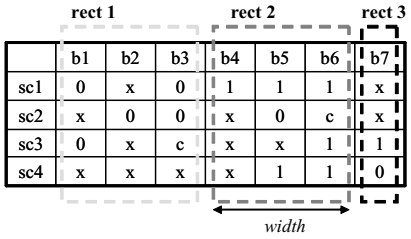
The first step in rectangular encoding is to partition the test cubes into clusters such that pattern-wise correlation within a cluster is maximized. This is done using a clustering algorithm that will be described in Sec. 2.4. Each cluster is then encoded as one unit. If there are n scan chains, then 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 Sec. 2.2 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 which identifies which scan chains should be loaded from the linear decompressor and which scan chains should be filled with a specified fill value (*1* or *0*). 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 and 3.

In Fig. 2, there are 7 scan slices (columns) and 4 scan chains (rows). The bit compatibility for a 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 3 rectangles.

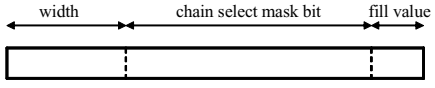
Figure 3(a) shows the format for rectangular control data, and Fig. 3(b) shows the specific values for the three rectangles in Fig. 2. The width of the rectangle in terms of scan slices is encoded as a binary number. The maximum width of a rectangle is a user-defined parameter and determines the number of bits allocated for specifying the width (experimental data for different maximum widths is discussed in Sec. 4). The *chain select mask* contains one 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. 3(b), the *chain select mask* in this case would be 1011 which 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 *sc3* can be loaded with the fill value. Moreover, in this case *sc4* has only *X* values and thus it doesn't matter whether it is loaded with the fill value or from the linear decompressor, and therefore the *chain select mask* is *110X* 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 with from linear decompressor. The advantage of this is that as can be seen in Fig. 3, the *chain select mask* and fill value are simply don't cares for *rect3*.



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

Figure 2. Example of Rectangles



(a) Data format

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

(b) Control data for three rectangles

Figure 3. Format for Rectangular Control Data

2.2 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 subsection.

The first step is to generate a *compatibility cube* for the test cube cluster. This is illustrated in Fig. 2 and has been explained earlier. From the compatibility cube, the rectangle which 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.

2.3 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 this data is that instead of using one bit in the *chain select mask* for each scan chain, one 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 since he control data is significantly reduced. Experimental results are shown in Sec. 4 for different values of *k*.

2.4 Forming Test Cube Clusters

In rectangular encoding, the test cubes are partitioned into clusters and each cluster is then divided into rectangles. Some nice algorithms for this type of clustering were described in [Alleyne 94]. 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 *C*'s 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. 4. 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 those 3 bit positions is 7 (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. 4, if 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 above.

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

Figure 4. Example of Clustering

3. Rectangular Decoder

Decoding the rectangles is done with a sequential non-linear decoder that is placed between a linear decompressor and the scan chains. A block diagram for the rectangular decoder is shown in Fig. 5. It consists of a controller which 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. 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. 5 for sake of readability.

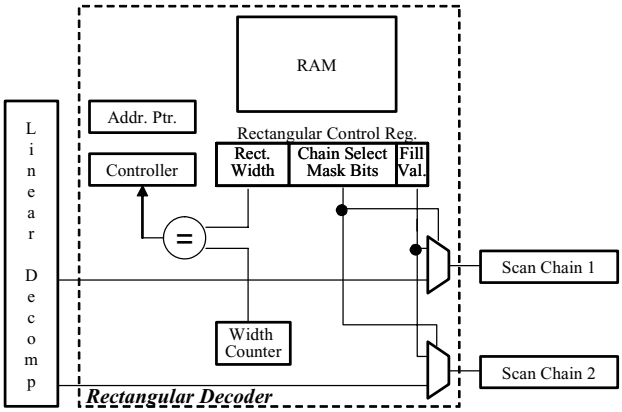


Figure 5. Block Diagram for Rectangular Decoder

The RAM holding the rectangular control data can be loaded 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 is transferred either directly from the tester or through the linear decompressor to the RAM. In this case, the RAM must be large enough to store all the rectangular 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 needs 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.

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 tell 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 was 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 is to be loaded incrementally, it is done at this point (only the data needed for this cluster). If the rectangular control data was 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,

Table 1. Results for Proposed 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

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 nice 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.

4. Experimental Results

Experiments were performed on the four largest ISCAS89 circuits. In Table 1, the number of test cubes and the number of specified bits in deterministic test sets are shown in the second and third column. The fourth column shows the number of clusters obtained with the pattern clustering algorithm described in Sec. 2.3. 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. Note that the graph on the left side of Fig. 6 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 on the right side of Fig. 6 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.

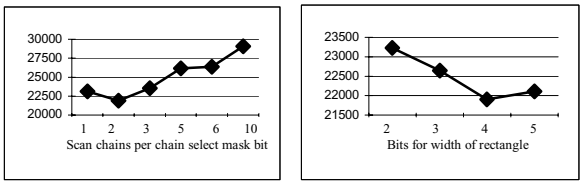


Figure 6. Specified bits vs. width and *k* value for *s38417*

The total number of specified bits that the linear decompressor has to produce when using the proposed non-linear decoder is shown in the tenth 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.

Results for combining the proposed scheme with an actual linear decompressor are shown in Table 2. 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 [Krishna 01] alone and using it with the proposed scheme are shown in Table 2. 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.

Table 2. Results combined with partial reseeding

Circuit	Test Cube	[Krishna 01]		Proposed		
		Specified	Storage	Specified	Storage	Reduction
s13207	266	9389	9872	7231	7678	22.2%
s15850	269	10944	11322	6726	7176	36.6%
s38417	376	30669	31245	21102	21780	30.3%
s38584	296	26185	28312	21652	22199	21.6%

Table 3. Results compared with [Ward 05]

Circuit	Num. of Specified Bits	
	[Ward 05]	Proposed
s13207	7499	6900
s15850	8333	6726
s38417	22277	21102
s38584	23254	21652

We also compared the number of specified bits in the proposed scheme to the number of specified bits in [Ward 05] in Table 3. Note that [Ward 05] reports the best tester storage results among compression schemes that use both linear and non-linear techniques. In all the cases shown in [Ward 05], 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 non-linear compression techniques (i.e., [Krishna 02], [Sun 04], and [Ward 05]), it also allows continuous flow decompression and the design of the decoder is independent of the test data.

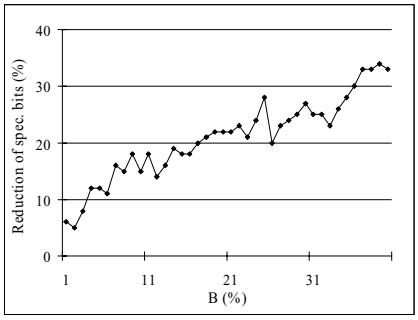


Figure 7. Results for test set having few specified bits

The percentage of specified bits in the test sets for the ISCAS89 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%* which 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 also be specified in the same bit position and a *B%* chance of having the same specified value as the previous test cube. Figure 7 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 this data suggests the proposed method can be quite effective.

5. Conclusions

The proposed scheme harnesses the power of linear and non-linear decompression 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.

Acknowledgements

This material is based on work supported in part by the National Science Foundation under Grant No. CCR-0306238.

References

[Alleyne 94] Alleyne, R., "Clustering of Test Cubes: A Procedure for the Efficient Encoding of Complete Test Sets Based on the Intelligent Reseeding of LFSRs", Masters Thesis, McGill University, 1994.

[Hellebrand 00] Hellebrand, S., H.-G. Liang, and H.-J. Wunderlich, "A Mixed-Mode BIST Scheme Based on the Reseeding of Folding Counters," *Proc. of Int. Test Conf.*, pp. 778-784, 2000.

[Krishna 01] Krishna, C.V., A. Jas, and N.A. Touba, "Test Vector Encoding Using Partial LFSR Reseeding", *Proc. of IEEE International Test Conference*, pp. 885-893, 2001.

[Krishna 02] Krishna, C.V., and N.A. Touba, "Reducing Test Data Volume Using LFSR Reseeding with Seed Compression ", *Proc. of IEEE International Test Conference*, pp. 321-330, 2001.

[Krishna 04] Krishna, C.V., and N.A. Touba, "3-Stage Variable Length Continuous-Flow Scan Vector Decompression Scheme", *Proc. of IEEE VLSI Test Symposium,* pp. 79-86, 2004.

[Liang 01] Liang, H.-G., S. Hellebrand, and H.-J. Wunderlich, "Two-Dimensional Test Data Compression for Scan-Based Deterministic BIST," *Proc. Int. Test Conf.*, pp. 894-902, 2001.

[Rajski 02] Rajski, J., *et al.,* "Embedded Deterministic Test for Low Cost Manufacturing Test," *Proc. of Int. Test Conf.*, pp. 301-310, 2002.

[Rajski 03] Rajski, J., "Method for Clustered Test Pattern Generation", *US. Patent No. 6,662,327*, Dec. 9, 2003.

[Sun 04] Sun, X., L. Kinney, and B. Vinnakota, "Combining Dictionary Coding and LFSR Reseding for Test Data Compression," *Proc. Design Autom. Conf.*, pp. 944-947, 2004.

[Tsai 00] Tsai, K.-H., J. Rajski, and M. Marek-Sadowska, "Star Test: The Theory and Its Applications," *IEEE Trans. on Computer-Aided Design*, Vol. 19, Issue 9, pp. 1052-1064, Sep. 2000.

[Ward 05] Ward. I. S., Schattauer, C., and N. A. Touba, "Using Statistical Transformations to Improve Compression for Linear Decompressors," *Proc. Defect Fault Tol. Symp.*, pp. 42-50, 2005.