Efficiently Utilizing ATE Vector Repeat for Compression by Scan Vector Decomposition

Jinkyu Lee and Nur A. Touba

Computer Engineering Research Center
University of Texas, Austin, TX 78712
{*jlee2, touba*}@ece.utexas.edu

Abstract

Previous approaches for utilizing ATE vector repeat are based on identifying runs of repeated scan data and directly generating that data using ATE vector repeat. Each run requires a separate vector repeat instruction, so the amount of compression is limited by the amount of ATE instruction memory available and the length of the runs (which typically will be much shorter than the length of a scan vector). In this paper a new and more efficient approach is proposed for utilizing ATE vector repeat. The scan vector sequence is partitioned and decomposed into a common sequence which is the same for an entire cluster of test cubes and a unique sequence that is different for each test cube. The common sequence can be generated very efficiently using ATE vector repeat. Experimental results demonstrate that the proposed approach can achieve much greater compression while using many fewer vector repeat instructions compared with previous methods.

1. Introduction

Test vector compression involves storing a deterministic test set on the automated test equipment (ATE) in a compressed form. One instruction that is commonly found in most ATEs is *vector repeat* which allows the ATE to repeat a sequence *n* times. Previous research has proposed ways to use this mechanism to reduce vector memory requirements for scan testing. In this paper, a new and more efficient approach is proposed for utilizing ATE vector repeat.

A test cube is a deterministic test vector in which the inputs that are not assigned during automated test pattern generation (ATPG) are left as don't cares. Typically only 1-5% of the bits in a test cube are specified while the rest are don't cares. Normally, *random fill* is performed where the don't cares are filled randomly with 1's and 0's to increase the chance of detecting additional faults. In [Barnhart 01], a methodology for utilizing ATE vector repeat was described. The idea is that instead of doing random fill of the test cubes, *repeat fill* is done where don't cares are filled by repeating the last specified bit within the same scan chain. This creates runs of repeated values in the scan chains. A scan slice is defined as the *n*-bits loaded in parallel from the tester into *n* scan chains each clock cycle. If multiple consecutive scan slices are identical, then ATE vector repeat can be used to generate them. Only one copy of the scan slice data needs to be stored in the ATE vector memory, and then a vector repeat instruction is stored in the ATE instruction memory. There is a limit to how much ATE vector repeat can be used based on the amount of instruction memory that is available. Consequently, ATE vector repeat is only used for the longest runs of repeated scan slices and not all runs.

In [Liu 02], ATE vector repeat is used for repeating full scan vectors during a transition test. For transition tests and other two-patterns tests, the same full scan vector may be repeated in the scan chain if it is used as the launch (V_2) vector for one two-pattern test and then as the initialization (V_1) vector for the next two-pattern test. In [Liu 02], transition test chains where only the very first and last vectors in a sequence are not repeated are formed to maximally utilize ATE vector repeat to reduce ATE storage.

In [Vranken 03], ATE vector repeat per pin-group is investigated. In [Barnhart 01], a single ATE vector repeat instruction applies to all pins. However, some testers provide the ability to specify pin-groups for which to apply the ATE vector repeat instruction. It was shown in [Vranken 03] that by using ATE vector repeat on smaller pin-groups, the length of the runs increases which allows greater compression.

In [Wang 05], ATE vector repeat is used in conjunction with a scan slice encoding scheme. Each scan slice is encoded into a sequence of smaller codewords, and an on-chip decoder is used to expand the codewords into the original scan slices. In this scheme, some codewords may be repeated, so ATE vector repeat is used for runs of repeated codewords.

Previously proposed approaches for utilizing ATE vector repeat are all based on identifying runs of repeated scan data and directly generating that data using ATE vector repeat. The amount of compression is limited by

IEEE COMPUTER SOCIETY

the amount of ATE instruction memory available and the length of the runs. This paper proposes a new and more efficient way of utilizing ATE vector repeat. The idea is to exploit the fact that many test cubes have similar input assignments due to the fact that they are structurally related in the circuit. The set of test cubes in a test set are partitioned into clusters that share many input assignments. The scan sequence is then decomposed into two components: the sequence of specified bits that is common across all the test cubes in a cluster, and the sequence of specified bits that is unique to each test cube. Two separate on-chip decompressors are then used to generate these two sequences as illustrated in Fig. 1. Since the common sequence is the same for all the test cubes in a cluster, the input stream to its decompressor can be generated using ATE vector repeat. Only one copy of this input stream needs to be stored in the ATE vector memory, and only one ATE vector repeat instruction needs to be stored in the ATE instruction memory for decompressing the entire test cube cluster. For the unique sequence corresponding to each test cube, it is generated with its own decompressor. The unique sequence will only contain very few specified bits because most of the specified bits will be generated by the common sequence decompressor. If a linear decompressor based on dynamic LFSR reseeding such as those described in [Krishna 01], [Konemann 01], and [Rajski 02] is used, the amount of compression depends only on the number of specified bits in the sequence. Note that the design of these decompressors is independent of the test set, so they can be reused when testing multiple cores in a system-on-chip (SOC) design. The diagram of the decompression hardware is shown in Fig. 1.

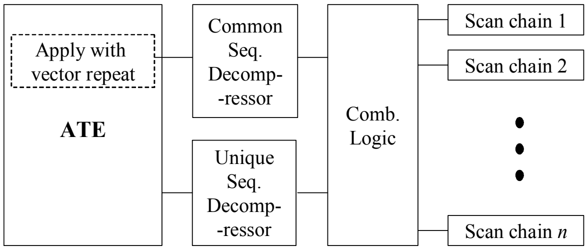


Figure 1. Diagram of Proposed Scheme

In comparing the proposed approach for using ATE vector repeat with the previous approaches, it has several advantages. Previous approaches can only generate runs of repeated values which typically will be much shorter than the length of a scan vector, and each run requires a separate vector repeat instruction. The proposed approach generates the common data component for a cluster of test cubes, and only one vector repeat instruction is needed for each test cube cluster. Consequently, the proposed approach is much more efficient in utilizing the ATE vector repeat instructions. For the limited ATE instruction memory available, the proposed approach will be able to achieve greater compression. The cost of the proposed approach is the need for the on-chip decompressors. However, the on-chip decompressors can efficiently exploit the large percentage of don't care bits in the test cubes to achieve very high compression. Compared with conventional linear decompression alone, the proposed use of ATE vector repeat provides a significant improvement in the amount of compression.

The concept of clustering test cubes to exploit similar input assignments has been previously investigated in the context of built-in self-test (BIST). STAR-BIST [Tsai 00] generates a parent pattern and then children patterns are generated by randomly flipping bits in the parent pattern. In [Liang 01], a folding counter is used to generate the children patterns. In [Li 05], frequently occurring sequences shorter than a full pattern are stored on-chip and used to embed deterministic patterns in a semi-random sequence. The underlying concept of these approaches is similar to what is proposed here, but there are a number of significant differences. The proposed approach generates a specific precise deterministic test set whereas the previous methods embed a deterministic test set into a much larger set of test vectors. The proposed approach decomposes the vectors into a common sequence and unique sequence, and these two sequences are combined in a fundamentally different manner than what is done in [Tsai 00], [Liang 01], and [Li 05].

2. Decomposing Scan Data

In the proposed scheme, the set of test cubes in a test set are partitioned into clusters that share many input assignments. The scan data is then decomposed into two components: the sequence of specified bits that is common across all the test cubes in a cluster, and the sequence of specified bits that is unique to each test cube. An example is shown in Fig. 2 to illustrate how the scan data is decomposed. Assume that the eight test cubes shown in Fig. 2 are included in one cluster. Each bit position in a test cube cluster can be classified as either being a don't care if no test cube has a specified value in that bit position, having "common data" if all test cubes have compatible values in that bit position, or having "unique data" if two or more test cubes have conflicting specified values. In the example in Fig. 2, the last bit position is a don't care. The 1st and 3rd bit positions have compatible value across all of the eight test cubes and thus are common data. The common data can be generated by the common sequence decompressor that operates based on ATE vector repeat since it is the same for each test cube. The 2nd, 4th, 5th, 6th, and 7th bit positions have conflicting values and thus are unique data. They must be generated by the unique sequence generator. The common data for the test cube cluster is shown in Fig. 2 along with the unique data for each test cube.

Because the scan data is decomposed into common data and unique data, a control signal is required to indicate if a bit position should be filled from the common data or the unique data. This is illustrated in Fig. 3. The control signal is a don't care for any don't care bit position in a cluster (in the example in Fig. 2, only the last bit position is a don't care), and it has a specified value for all other bit positions. A key property is that the same control sequence can be used when decompressing all test cubes in a cluster and thus it is a "common control". This means that the control signal can be generated by the common sequence generator using ATE vector repeat and thus the storage required for the common control is amortized across all the test cubes in the cluster. The common control sequence for the example test cube cluster is shown in Fig. 2.

Original								
Test cube 1	0	1	1	1	1	0	1	x
Test cube 2	0	0	1	1	1	0	1	x
Test cube 3	0	1	1	1	1	0	0	x
Test cube 4	0	1	1	0	0	1	1	x
Test cube 5	0	0	x	1	1	0	0	x
Test cube 6	0	1	1	0	1	1	1	x
Test cube 7	x	1	1	1	0	1	1	x
Test cube 8	x	1	1	1	1	1	1	x
Encoded								
Common control	1	0	1	0	0	0	0	x
Common data	0	x	1	x	x	x	x	x
Unique data 1	x	1	x	1	1	0	1	x
Unique data 2	x	0	x	1	1	0	1	x
Unique data 3	x	1	x	1	1	0	0	x
Unique data 4	x	1	x	0	0	1	1	x
Unique data 5	x	0	x	1	1	0	0	x
Unique data 6	x	1	x	0	1	1	1	x
Unique data 7	x	1	x	1	0	1	1	x
Unique data 8	x	1	x	1	1	1	1	x

Figure 2. Example of proposed encoding scheme

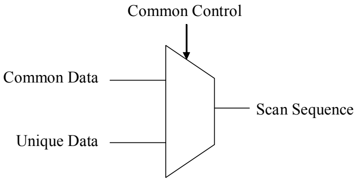


Figure 3. Example of proposed encoding scheme

Typically a large number of specified bits can be included in the common data because many test cubes have similar input assignments due to the fact that they are structurally related in the circuit. The clustering procedure described in Sec. 4 selects the clusters to maximize the amount of compression for the cluster. During decompression, the common sequence generator produces both the common data and common control. Only one copy of the input stream for the common sequence generator needs to be stored in the ATE vector memory since it can be applied using the ATE vector repeat instruction for all the test cubes in the cluster. Therefore, a large reduction in storage requirements can be achieved.

In the example in Fig. 2, the number of specified bits in the original test cubes is 53, and the number of specified bits in the encoded test cubes is 49 (2 specified bits in the common data, 40 specified bits in the unique data, and 7 specified bits in the common control). The reduction in the example shown in Fig. 2 is small, but in real cases, the number of test cubes in a cluster is much greater than the number of test cubes in this example, so the number of specified bits generated from the common data is much higher, thereby making the reduction in the total number of specified bits larger.

3. Decompression Hardware

The proposed scheme requires relatively simple decompression hardware (two sequential linear decompressors and one MUX per scan chain). The proposed decompression hardware is shown in Fig. 4. There are two types of memory in the ATE, instruction memory and vector memory. The instruction memory stores the ATE instructions including the vector repeat instructions, and the vector memory stores the data. The data is transferred to the decompressors based on the ATE instructions. There are two sequential linear decompressors in Fig. 4. One decompressor (the upper one in Fig. 4) operates with ATE vector repeat and the other decompressor operates without the ATE vector repeat. The decompressor that operates with vector repeat generates the common data and common control signals for each scan chain. Only one copy of the input stream for generating the common data and common control for all the test cubes in a cluster is stored in the ATE and applied repeatedly for each test cube in the cluster using an ATE vector repeat instruction. The other decompressor (the lower one in Fig. 4) generates the unique data and operates without vector repeat. This decompressor operates in the same way as the decompressor in conventional linear compression schemes. A 2-to-1 multiplexer is placed between the decompressors and each scan chain.

One very nice property of sequential linear decompressors is that regardless of how many outputs signals they generate or how long of a sequence they generate, the number of input bits that are required for the decompressor depends only on the total number of specified bits that it needs to generate (the rest of the bits are essentially filled with random data). Thus the architecture can be easily scaled to any number of scan chains limited only by the rate at which data from the

ATE can be transferred to the sequential linear decompressors relative to the number of specified bits that the decompressor needs to generate. Note that the decompression hardware does not depend on the circuit or test set, which makes it possible to reuse it when testing multiple cores in a system-on-chip (SOC) design.

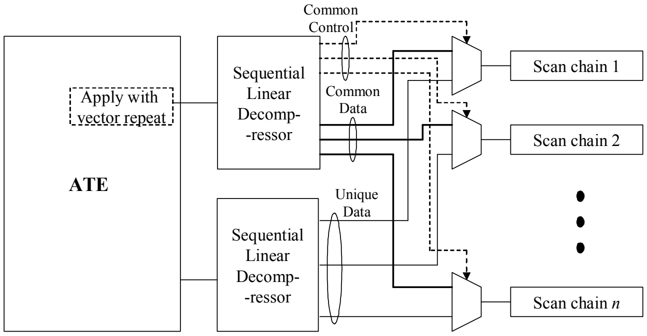


Figure 4. Decompression hardware

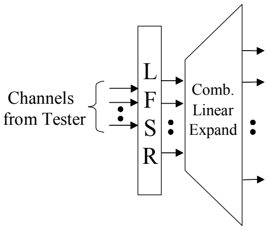


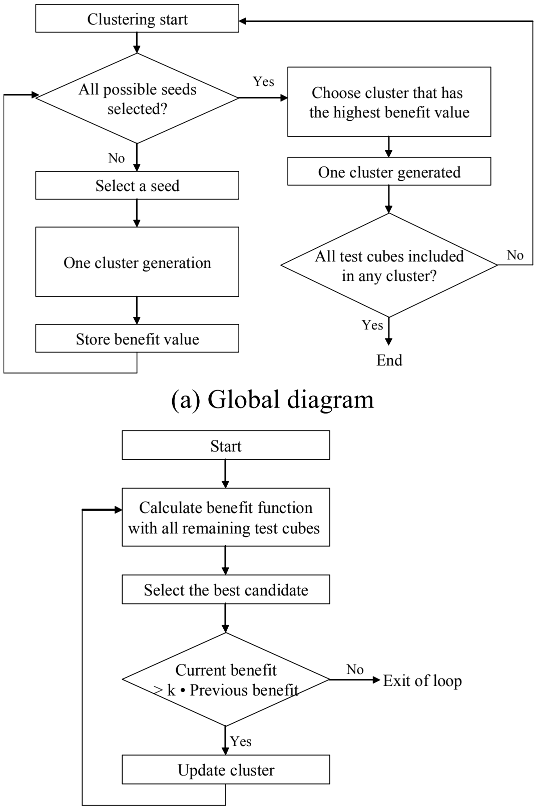
Figure 5. Example of sequential linear decompressor

The sequential linear decompressors that are used for this scheme could be any of the ones described in [Krishna 01], [Konemann 01], or [Rajski 02]. An example of the sequential linear decompressor is shown in Fig. 5.

4. Forming Test Cube Clusters

In the proposed scheme, test cubes are grouped into clusters. Each cluster requires the use of an ATE vector repeat instruction for generating the common sequence for the cluster. Because the ATE instruction memory is limited, the number of clusters cannot exceed the amount of ATE instruction memory available. For that reason, the clustering algorithm used here tries to maximize the correlation in each cluster (to reduce the number of specified bits, thereby minimizing the tester storage), while at the same time generating a small number of clusters (to minimize the number of ATE repeat instructions required).

Test cube clustering has been previously studied and some nice algorithms can be found in [Alleyne 94]. For the proposed scheme, a special benefit function is needed to account for the both the control and data bits required to encode each cluster.


Clustering start
All possible seeds selected?
Yes
No
Select a seed
One cluster generation
Store benefit value
Choose cluster that has the highest benefit value
One cluster generated
All test cubes included in any cluster?
No
Yes
End
Start
Calculate benefit function with all remaining test cubes
Select the best candidate
Current benefit > k • Previous benefit
No
Exit of loop
Yes
Update cluster

(a) Global diagram

(b) Sub-diagram of one cluster generation

Figure 6. Flow diagram of clustering algorithm

In order to maximize the compression achieved for each cluster, 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 number of clusters is reduced. This has the benefit of minimizing the number of the ATE instructions required for the vector repeat, but there is a tradeoff as more bit positions are likely to have conflicts thus reducing the effectiveness of each repeat instruction. A greedy clustering procedure that takes this tradeoff into consideration is described here.

The flow diagram of the proposed clustering algorithm is shown in Fig. 6 and the benefit function that is used to assign a value to a cluster is shown in the below:

$$Benefit = \frac{total\_spec}{compatible\_pos + spec\_pos + total\_unique\_spec}$$

where *total_spec* is the total number of specified bits in the cluster, *compatible_pos* is the number of bit positions that are compatible in the cluster, *spec_pos* is the number of bit positions that have specified bits, and *total_unique_spec* is the total number of specified bits not in compatible bit positions. Essentially the numerator corresponds to the uncompressed storage requirements, and the denominator corresponds to the compressed storage requirements (the common data has a specified bit for each compatible bit position in the cluster, the

15th Asian Test Symposium (ATS'06)
0-7695-2628-4/06 $20.00 © 2006 IEEE

Table 1. Results for proposed scheme on benchmark circuits

Circuit	Num. Test cubes	Original Specified Bits	Num. Clusters	Num. Specified in Unique Seq.	Num. Specified in Common Seq.	Encoded Specified Bits	Reduction
s13207	266	9389	18	5612	1881	7493	20.2%
			40	4551	3049	7600	19.1%
s15850	269	10944	15	6335	2103	8438	22.9%
			34	4650	3579	8229	24.8%
s38417	376	30669	19	12427	7207	19634	35.9%
			31	11044	8131	19175	37.5%
s38584	296	26185	18	17505	5849	23354	10.8%
			98	11293	10642	21935	16.2%

common control has a specified bit for each bit position that has one or more specified bits in the cluster, and the unique data has one specified bit for each specified bit not in a compatible bit position in the cluster). The larger the benefit value, the larger the amount of compression that is achieved when encoding the cluster. Note that while a greedy clustering procedure is described here, any clustering procedure in the literature can be used to maximize the benefit function defined here.

One issue with the benefit function is that it may generate too many clusters in some cases. Note that one ATE repeat instruction is required in the proposed scheme for each cluster. Thus, there is a limit on how many clusters can be used based on the amount of ATE instruction memory that is available. To provide a mechanism for reducing the number of clusters generated by the clustering procedure, a tuning variable, *k*, is added to the algorithm as shown in Fig. 6-(b). To increase the number of test cubes in each cluster, the condition for adding a test cube into a cluster can be loosened by making the value of *k* lower than 1. By lowering the value of *k*, the number of clusters reduces with a reasonable sacrifice in the number of specified bits. Note that if the value of *k* is reduced too much, at some point the number of specified bits in the encoded test cubes approaches the number of specified bits in original test cubes and hence no compression is achieved. In Sec. 5, experimental results are shown for different values of *k* to illustrate the tradeoffs.

5. Experimental Results

Experiments were performed on the four largest ISCAS-89 benchmark circuits. In Table 1, the number of test cubes and the original number of specified bits in the deterministic test sets are shown in the second and third columns. The fourth column shows the number of clusters generated by the clustering algorithm described in Sec. 4. For each circuit, results are shown for two different numbers of clusters. One was obtained with a value of *k=1*, and the other was obtained by adjusting the value of *k*. The fifth and sixth columns show the number of specified bits in the unique sequence and the number of specified bits in common sequence. The seventh column shows the total number of specified bits in the encoded test cubes. The last column shows the reduction in the number of specified bits. Note that the maximum number of clusters shown in Table 1 is 98, which means that the maximum number of ATE vector repeat instructions that have to be stored in the ATE instruction memory is only 98 or less for these circuits. In most circuits, the number of clusters is below 40. Of course, the number of clusters can also be reduced if necessary by lowering *k*. Since the number of specified bits with the proposed scheme is reduced, the number of free-variables that are needed to encode the data using the linear decompressors will also reduce correspondingly.

Table 2. Results for using linear decompressor in [Krishna 01] alone versus using it with proposed scheme

Circuit	[Krishna 01]		Proposed			
	Num. Spec.	Vector Mem.	Num. Spec.	Vector Mem.	Reduc.	Comp.
s13207	9389	9872	7493	7750	21.5%	95.3%
s15850	10944	11322	8229	8694	23.2%	88.4%
s38417	30669	31245	19175	20769	33.5%	86.8%
s38584	26185	28312	21935	24268	14.3%	86.7%

Table 3. Results comparing with [Vranken 03]

Circuit	[Vranken 03]		Proposed			
	Num. Repeat Inst.	Vector Mem.	Num. Repeat Inst.	Vector Mem.	Reduction	
					Inst. Mem.	Vector Mem.
s13207	976	8464	18	7750	98.1%	8.4%
s15850	894	15974	34	8694	96.2%	45.6%
s38417	2518	41506	31	20769	98.8%	50.0%
s38584	3264	53952	98	24268	96.9%	55.1%

To get results for the actual tester storage requirements using the proposed approach, we did experiments using the linear decompressor described in [Krishna 01] (other linear decompressors could also be used). There results are shown in Table 2. The same test sets are compressed using the linear decompressor in [Krishna 01] by itself, and using it in conjunction with the proposed scheme to utilize ATE vector repeat. The results show that the amount of data that needs to be stored in the ATE vector memory is significantly reduced with the proposed scheme. The second to last column shows the vector memory reduction compared with [Krishna 01], and the last column shows the percentage of compression achieved comparing the compressed test set with a highly compacted test set generated with COMPACTEST [Pomeranz 93].

In Table 3, the proposed scheme is compared with the scheme described in [Vranken 03] for utilizing ATE vector repeat. In [Vranken 03], the best results were obtained when a sequencer controls two pins, so we also assumed that a sequencer controls two pins. And as suggested in [Vranken 03], only the vectors that can be repeated at least 16 times are encoded by the ATE vector repeat to reduce the number of the repeat instructions. Using this criteria, we generated experimental results on our test sets in the manner described in [Vranken 03] (note that results published in [Vranken 03] were for test sets that are not publicly available). The number of ATE repeat instructions is shown in the second and the fourth columns, and the amount of data stored in the vector memory is shown in the third and fifth columns. As can be seen, much larger reductions in the vector memory can be obtained with the proposed approach using an order of magnitude fewer ATE repeat instructions compared with [Vranken 03]. Of course, it should be pointed out that the method in [Vranken 03] does not require any on-chip hardware, whereas the proposed method requires two on-chip linear decompressors. However, note that the linear decompressors used in the proposed scheme will require a very small amount of area with current chip densities, and they can be reused when testing multiple cores.

Table 4. Results comparing with [Wang 05]

Circuit			[Wang 05]	Proposed		
Name	Test Cubes	Scan Cells	Vector Memory	Repeat Inst.	Vector Memory	Reduction
ckt-4	1529	43414	3264850	121	1518409	53.5%
				404	1341943	58.9%
ckt-5	4900	26970	6079410	283	2579402	57.6%
				841	2268261	62.7%

In Table 4, a comparison is made with the scheme described in [Wang 05] that also utilizes ATE vector repeat. Here results are shown for the exact same test cube files for two of the industrial circuits that were used in [Wang 05] (the others were not publicly available). The circuit information is shown in the first, second and third columns. The fourth column shows the best results in terms of vector memory requirements reported in [Wang 05]. For the proposed method, results are shown for two different numbers of ATE vector repeat instructions. Note that the number of repeat instructions used in [Wang 05] is not reported in the paper and thus is not shown in Table 4. The vector memory required is shown in the sixth column. The last column shows the reduction in the vector memory required. There is a substantial reduction. A lot of the reduction comes from the fact that the proposed scheme is based on linear decompression. A major advantage of the proposed scheme is that it is compatible with linear decompression which is known to be highly efficient.

6. Conclusions

The proposed scheme provides a way to utilize ATE vector repeat to achieve additional compression on top of the compression achieved using a linear decompressor. The design of the decompressor for the proposed scheme is independent of the test set or CUT and thus can be reused when testing multiple cores.

Acknowledgements

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

References

[Alleyne 94] Alleyne, Ronald, “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.

[Barnhart 01] Barnhart, C., Brunkhorst, V., Distler, F., Farnsworth, O., Keller, B., and Koenemann, B., “OPMISR: the Foundation for Compressed ATPG Vectors,” *Proc. of Int. Test Conf.*, pp. 748-757, 2001.

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

[Könemann 01] Könemann, B., “A SmartBIST Variant with Guaranteed Encoding” *Proc. of Asian Test Symposium*, pp. 325-330, 2001.

[Li 05] Li, L., and K. Chakrabarty, “Hybrid BIST Based on Repeating Seequences and Cluster Analysis,” *Proc. of Design, Automation, and Test in Europe (DATE)*, pp. 1142-1147, 2005.

[Liang 01] Liang, H.-G., S. Hellebrand, and H.-J. Wunderlich, “Two-Dimensional Test Data Compression for Scan-Based Deterministic BIST,” *Proc. of International Test Conference*, pp. 894-902, 2001.

[Liu 02] Liu, X., Hsiao, M., Chakravarti, S., and Thadikaran, P.J., “Techniques to Reduce Data Volume and Application Time for Transition Test,” *Proc. of Int. Test Conf.*, pp. 983-992, 2002.

[Pomeranz 93] Pomeranz, I., L.N. Reddy, and S.M. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 12, No. 7, pp. 1040-1049, Jul. 1993.

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

[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.

[Vranken 03] Vranken, H., Hapke, F., Rogge, S., Chindamo, D., and Volkerink, E., “Atpg Padding and ATE Vector Repeat per Port for Reducing Test Data Volume,” *Proc. of Int. Test Conf.*, pp. 1069-1078, 2003

[Wang 05] Wang, Z., and Chakrabarty, K., “Test Data Compression for IP Embedded Cores Using Selective Encoding of Scan Slices,” *Proc. of Int. Test Conf.*, pp. 581-590, 2005.

15th Asian Test Symposium (ATS'06)
0-7695-2628-4/06 $20.00 © 2006 IEEE