SOC Test Compression Scheme Using Sequential Linear Decompressors with Retained Free Variables

Sreenivaas S. Muthyala and Nur A. Touba

Computer Engineering Research Center
University of Texas, Austin, TX 78712
sreenivaas@utexas.edu, touba@utexas.edu

Abstract

A highly efficient SOC test compression scheme which uses sequential linear decompressors local to each core is proposed. Test data is stored on the tester in compressed form and brought over the TAM to the core before being decompressed. Very high encoding efficiency is achieved by providing the ability to share free variables across test cubes being compressed at the same time as well as in subsequent time steps. The idea of retaining unused non-pivot free variables when decompressing one test cube to help for encoding subsequent test cubes that was introduced in [Muthyala 12] is applied here in the context of SOC testing. It is shown that in this application, a first-in first-out (FIFO) buffer is not required. The ability to retain excess free variables rather than wasting them when the decompressor is reset avoids the need for high precision in matching the number of free variables used for encoding with the number of care bits. This allows greater flexibility in test scheduling to reduce test time, tester storage, and control complexity as indicated by the experimental results.

1. Introduction

System-on-chip (SOC) designs are composed of reusable cores each of which must be thoroughly tested. Cores can be synthesizable designs (i.e., soft cores) in which the design-for-test (DFT) architecture can be customized (e.g., number and length of scan chains, etc.), or they can be layouts (i.e., hard cores) in which the DFT architecture is fixed. For intellectual property cores (IP cores), it may not be possible to perform ATPG in which case a set of *test cubes* (i.e., test vectors in which the unassigned inputs are left as don't cares) is provided that must be applied during test.

SOC testing typically involves designing wrappers to go around the cores, providing test access mechanisms (TAMs) for transporting data from the tester pins to the cores, and developing a test schedule for which cores are being tested at different times. Many techniques for designing wrappers, TAMs, and test schedules have been developed, see [Xu 05] for a survey.

One way to reduce test time is through the use of test compression [Touba 06]. Several techniques for incorporating test compression in SOC testing have been proposed. Early techniques were based on coding using frequency-directed run-length (FDR) codes [Iyengar 05], nine codewords [Tehranipoor 05], and XOR networks [Gonciari 05]. More recent work has been based on sequential linear decompression which provides higher encoding efficiency. In [Wang 07], a single sequential linear decompressor is used to expand the tester channels to drive a larger number of TAM lines. The drawback of performing decompression before the TAM lines is that uncompressed data needs to be transported to the cores which requires more TAM bandwidth. Two existing schemes as well as the scheme proposed here are based on having sequential linear decompressors local to each core which allows compressed data to be transported over the TAM lines. The two existing schemes with local decompressors are described in further detail in the next two paragraphs.

In [Kinsman 10], the compressed data coming from the tester in each clock cycle (which will be referred to here as a *tester slice*) is loaded into one of the k decompressors which is selected by $log_2(k)$ control bits. Since the control bits are needed each clock cycle, $log_2(k)$ tester channels must be allocated for providing the control bits. The fine grain control over how many tester slices are used to encode each test cube for a core helps to match the number of *free variables* (i.e., bits stored on the tester that can be assigned either a 0 or 1) with the number of care bits in the test cube thereby helping to improve encoding efficiency. This comes at the cost of having a lot of control data which subtracts from the encoding efficiency and offsets the gains to some degree.

An approach based on using Embedded Deterministic Test (EDT) [Rajski 04] with dynamic channel allocation is described in [Kassab 10], [Janicki 11, 12]. The key idea in this approach is to dynamically allocate tester channels to decompressors in a way that allows test cubes across multiple cores to be decompressed simultaneously. Test scheduling is performed on a test cube basis to determine which channels are allocated to which decompressors. So control information is needed on a per test cube basis as opposed to a per slice basis as in [Kinsman 10]. Consequently, the control information can be loaded using the same tester channels that load the data which eliminates the need for allocating tester channel to be used only for control data. The number of free variables used for encoding each test cube is controlled by

978-1-4673-5543-8/13/$31.00 ©2013 IEEE

how many tester channels are allocated to the decompressor for that test cube.

Encoding efficiency in sequential linear decompressors is defined as the number of care bits in the test data divided by the number of free variables used to encode them (i.e., the number of bits stored on the tester). If too few free variables are used to encode a test cube, then it becomes unlikely to be able to solve the linear equations to encode it. So the goal is to use enough free variables to be able to reliably encode all test cubes, but not use more than necessary. What makes the problem difficult is the fact that the number of care bits in each test cube varies considerably for a single core. Examples of test cube profiles for industrial circuits showing how the percentage of care bits varies can be found in [Kassab 10]. The strategy for improving encoding efficiency in the previous schemes is to try to match the number of free variables used to encode each test cube to the number of care bits in the test cube. This is done by regulating the number of free variables sent to the decompressor for each test cube. In [Kinsman 10], this is done by selecting the number of tester slices loaded in the decompressor, and in [Kassab 10], [Janicki 11, 12], this is done by selecting the number of tester channels allocated to the decompressor. In both cases, any unused free variables (i.e., free variables that were not used as pivots when solving the linear equations) are wasted when the decompressor is reset before decompressing the next test cube. The resolution in matching the number of free variables to the number of care bits in a test cube during decompression is limited by the quanta of free variables in each slice for [Kinsman 10] or channel for [Kassab 10], [Janicki 11, 12] as well as the number of care bits in the available test cubes that can be paired with it. Moreover, even if there was precise control of the number of free variables used, it is still necessary for the number of free variables to be larger than the number of care bits to increase the probability of being able to reliably solve the linear equations. For all these reasons, a considerable number of free variables are generally wasted each time the decompressor is reset.

The proposed scheme for SOC test compression uses the ideas described in [Muthyala 12] for retaining unused non-pivot free variables for one test cube to help with encoding the next test cube without significantly increasing the computational complexity of solving the linear equations. This approach avoids the need for high precision in matching the number of free variables to the number of care bits during decompression because excess free variables are not wasted, but can be retained for encoding subsequent test cubes. This simplifies the test scheduling problem and reduces the number of control bits required to specify how the decompression is performed. A simple scheme for retaining non-pivot free variables without the need for a first-in first-out (FIFO) buffer (as is required in [Muthyala 12]) is utilized in the context of SOC testing. The proposed scheme offers high encoding efficiency to reduce tester storage, greater flexibility in test scheduling to reduce test time, and simple control complexity.

2. Retaining Non-Pivot Free Variables

Encoding test cubes with a sequential linear decompressor involves solving a system of linear equations. There is one equation for each care bit which gives its dependence on the free variables. An example is shown in Fig. 1 where a test cube with 5 care bits (corresponding to the rows) is encoded using 10 free variables (corresponding to the columns). The system of linear equations are solved by performing Gauss-Jordan elimination to obtain a set of pivots (one per care bit). The non-pivots represent free-variables that can be assigned any value. Depending on the values assigned to the non-pivots, the pivots can always be assigned appropriate values to solve the system of linear equations. A more detailed explanation of the encoding process can be found in [Könemann 91], [Krishna 01], and [Wang 06].

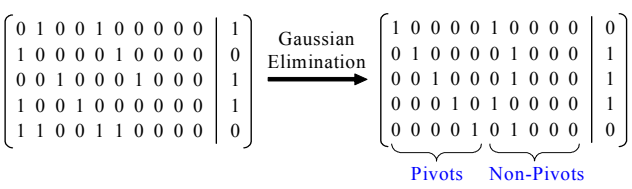

Figure 1. Example of solving system of linear equations for a particular test cube

Conventional methods reset the sequential linear decompressor before decompressing the next test cube to decouple the linear equations. If the decompressor is not reset, then the size of the linear equations grows as more free variables arrive. The complexity for solving the linear equations is $O(n^3)$ for n free variables, so the runtime can quickly become prohibitive if the free variable dependence becomes too large.

In [Muthyala 12], a scheme for retaining the non-pivot free variables from one test cube and using them to help encode the next test cube is described. It involves using a FIFO to store the last q tester slices when decompressing one test cube and then loading the free variables from the FIFO into the decompressor while decompressing the next test cube. The Gaussian elimination can be ordered to first try to use the early free-variables as pivots as much as possible, and only create pivots in the free variables coming in the last q clock cycles when necessary. This approach maximizes the number of non-pivot free-variables that get retained in the FIFO for encoding the next test cube and achieves nearly the same benefit as encoding the two test cubes together without resetting the sequential linear decompressor. It only misses out on non-pivots that occur before the last q clock cycles which would generally be few. If the number of free variables retained in the FIFO is 10% of the total number of free

variables, it was shown that the computation complexity for solving the two test cubes together increases only by a factor of 1.15 over the conventional approach where no free variables are shared.

In SOC testing, it is possible to retain non-pivot free variables without the need for a FIFO. The next sections describe a scheme for accomplishing this and optimizing test scheduling and control information by exploiting the additional flexibility that is provided.

3. Proposed SOC Test Architecture

The proposed scheme uses the broadcast architecture illustrated in Fig. 2 which allows a single tester slice to be simultaneously loaded into multiple decompressors in a clock cycle. The set of decompressors that load the tester slices is fixed throughout the decoding of a test cube, so the control information is needed only on a per test cube basis similar to [Kassab 10], [Janicki 11, 12]. This eliminates the need to allocate tester channels for carrying control data as is needed in [Kinsman 10]. When decompressing a test cube for a core, if there are more free variables than necessary, then test cubes for other cores can be decompressed at the same time. Let the set of cores that are decompressing test cubes at the same time be denoted as *core-set-1* and the set of test cubes (one corresponding to each core in *core-set-1*) be denoted as *testcube-set-1*. These sets are selected under the constraint that there are enough free variables to simultaneously solve for all care bits. The tester slices are broadcast to all cores in *core-set-1* and a system of linear equations containing an equation for each care bit in *testcube-set-1* must be solved when encoding these test cubes. The unused non-pivot free variables can be retained as described in Sec. 2.

Let the next set of cores that are decompressing test cubes at the same time be denoted as *core-set-2* and the set of test cubes be denoted as *testcube-set-2*. As long as *core-set-1* and *core-set-2* are disjoint, then the unused non-pivot free variables from encoding *testcube-set-1* can be retained to help in encoding *testcube-set-2*. This is done by broadcasting the last q tester slices when decompressing *testcube-set-1* to the decompressors for *core-set-2* (with scan shifting disabled in the cores in *core-set-2*). This pre-loads the decompressors in *core-set-2* with the last q tester slices worth of free variables for *testcube-set-1*. By ordering the Gaussian elimination process to try to create pivots in the earlier free variables first when encoding *testcube-set-1*, most of the non-pivot free variables will be in the last q tester slices worth of free variables and hence will be available to help encode *testcube-set-2*. After decompression of *testcube-set-1* is completed, then decompression of *testcube-set-2* is performed.

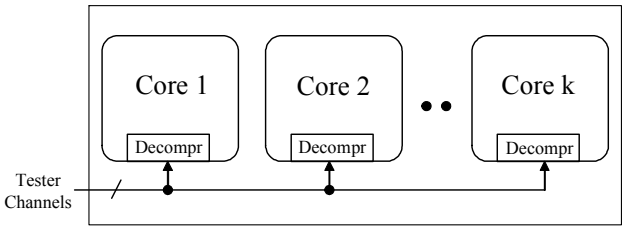

Figure 2. Proposed SOC Test Architecture

Due to the partial overlap of free variables used to encode *testcube-set-1* and *testcube-set-2*, the linear equations must be solved altogether. The structure of the linear equations will look as shown in Fig. 3. Note that there is some overlap of the free variables used to solve for *testcube-set-1* and *testcube-set-2*. The amount of overlap is equal to the number of free variables in q tester slices which is equal to q times the number of tester channels, c. If the percentage of overlap is 10%, then as shown in [Muthyala 12], the computational complexity for solving for the two test cube sets together increases only by a factor of 1.15 over the conventional case where there is no overlap.

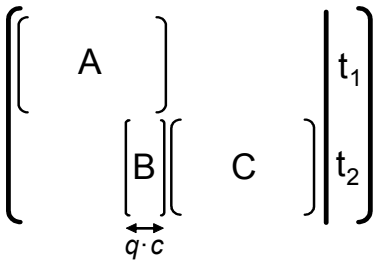

Figure 3. Structure of Linear Equations when Solving *testcube-set-1* (t_1) and *testcube-set-2* (t_2) with $q \cdot c$ Free Variables Retained

If desired, this process could be repeated to encode a third set of test cubes, *testcube-set-3*. In this case, *core-set-3* needs to be disjoint from *core-set-2*, but could include cores that were present in *core-set-1*, because at this point the decompressors in *core-set-1* are finished decompressing *testcube-set-1* and hence are available to be pre-loaded for decompressing *testcube-set-3*. The last q tester slices when decompressing *core-set-2* is broadcast to the decompressors for *core-set-3* to pre-load them. This allows non-pivot free variables from decompressing *testcube-set-1* and *testcube-set-2* to be used for decompressing *testcube-set-3*. In this case, all three testcube sets need to be solved together. If the percentage of overlap is 10%, then as shown in [Muthyala 12], the computational complexity for solving for the three test cube sets together increases only by a factor of 1.35 over the conventional case where there is no overlap.

Let m be the number of test cube sets that are encoded together. Then $m=1$ is the conventional case where there is no overlap of free variables when encoding test cubes

sets, *m=2* is the case where the non-pivot free variables for *testcube-set-1* are used to help encode *testcube-set-2*, and *m=3* is the case where the non-pivot free variables for *testcube-set-1* are used to help encode *testcube-set-2* and the non-pivot free variables for *testcube-set-2* are used to help encode *testcube-set-3*. Note that there is a diminishing marginal return from encoding additional test cubes sets together (as can be seen in the experimental results in Sec. 6). So encoding more than 3 test cube sets together would generally not be worthwhile.

Each set of test cubes that are encoded together are encoded using some particular *decompression mode*. A decompression mode is defined by *core-set-1*, *core-set-2*, and *core-set-3*, where some of these core sets would be empty if *m* is less than 3. For example, for a decompression mode where *m=2*, then *core-set-3* would be empty. So the decompression process works as follows. First some control data is provided to establish the decompression mode (methods for providing the control data will be discussed in Sec. 5). Then the cores in *core-set-1* load the tester slices for a number of clock cycles equal to the longest scan length in any core in *core-set-1*. If *m > 1* for the current decompression mode, then in the last *q* clock cycles, the decompressors in *core-set-2* also load the tester slices. After the scan chains in *core-set-1* are finished loading, the cores in *core-set-1* perform a capture cycle, and then the scan chains in *core-set-2* are loaded, and when that completes, they receive a capture cycle. If *m > 2*, then the decompressors in *core-set-3* would be loaded in the last *q* clock cycles when *core-set-2* is loaded, and the scan chains for *core-set-3* would be loaded when *core-set-2* is finished. After completing one decompression in the current decompression mode, then control information is provided to establish the decompression mode for the next decompression, and the process repeats until all test cubes have been applied.

Note that output compaction is not addressed in this paper, however, there are a number of existing output compaction schemes that can be employed in this scenario including something similar to what is used in [Janicki 11].

4. Test Scheduling Procedure

Given the set of test cubes that needs to be applied to each core, the test scheduling problem involves deciding which test cubes should be decompressed together and which decompression mode should be used when they are decompressed. The larger the number of unique decompression modes that are used, the more control bits are required to specify the decompression mode. So there is a tradeoff in terms of how many different decompression modes are used versus how efficient the encoding is. To manage this tradeoff, the test scheduling procedure described here uses a user-supplied threshold parameter, *thresh*, to determine when a new mode should be created and when an existing mode should be used. If the improvement in encoding efficiency for some decompression with a newly created mode exceeds the encoding efficiency of using an existing mode by an amount greater than *thresh*, then the new mode is created, otherwise the existing mode is used.

The proposed test scheduling procedure is as follows:

Step 1: Select the test cube with most care bits from the core with most remaining test cubes left to encode.

Step 2: Consider each existing decompression mode that contains the selected core. For each such mode, choose test cubes for the other cores present in the decompression mode that maximize encoding efficiency while still having a solution for the overall set of linear equations.

Step 3: Create one new candidate decompression mode for each value of *m* for the test cube selected in step 1 which maximizes encoding efficiency.

Step 4: If there is no existing decompression modes that use the selected core, then use the new candidate decompression mode from Step 3. Otherwise, compare the encoding efficiency of the best candidate decompression mode from Step 3 with the encoding efficiency of the best existing decompression mode from Step 2, and if it exceeds the value of *thresh*, then use the new candidate decompression mode, otherwise, use the existing decompression mode.

Step 5: Remove all the test cubes covered by the decompression and loop back to Step 1 while more test cubes remain.

If the number of decompression modes that result from running the test scheduling procedure is too high, the procedure can be rerun with a higher value for *thresh*. The test scheduling procedure is a greedy procedure and will find a good test schedule but is not guaranteed to find the optimum test schedule.

Note that in some cases where unwrapped glue logic between cores is tested, it may be necessary to apply some test cubes to groups of cores at the same time. This can be handled in the proposed framework, but would require some constraints on the test scheduling process to ensure that the decompression mode for applying such test cubes includes the necessary cores.

5. Control Data

As mentioned earlier, control data is required to establish the decompression mode used for each decompression. There are a number of different ways for generating the control data. One approach would be to load the control data with one or more tester slices right at the beginning of each decompression. The minimum number of control bits would be $log_2(number\ of\ modes)$ to select the decompression mode to use. On-chip hardware would then decode the mode and generate the approach control signals during the decompression. Another approach would be to order the decompressions so that all the ones that use mode 1 come first, followed by those that use mode 2, and then mode 3, etc. In this way, one bit

could signal whether the mode should be incremented or not. It is also possible to design an on-chip custom FSM for a particular test schedule that controls the mode, in which case no control data would be required on the tester.

If it is desirable to have a generic controller design that is not customized to any particular set of modes, then more control data would need to be brought in from the tester. For example, the value of *m* (i.e., number of core-sets) for the mode could be loaded followed by a vector with one bit per core indicating which cores are included in each core-set. In this case, there would be no need to minimize the number of modes during test scheduling since any number of modes could be applied at equal cost.

6. Experimental Results

Two sets of experiments were performed with the proposed scheme on different SoC designs with different number of cores. One is where the number of scan chains in each core is optimized to maximize compression (results are shown in Table 1). The other is where the scan architecture for each core is not changed as would be the case for hard cores (results are shown in Table 2). In both cases, the number of tester channels used is 16, and the test time and amount of data stored on the tester is shown for the case where the cores are tested serially one at a time, with the total test time being the sum of the clock cycles required for testing each core. Next, the test scheduling procedure in Sec. 4 is used without retaining unused free variables (i.e., *m=1*). Then results are shown for the *m=2* case where the unused free variables when encoding test cubes in *testcube-set-1* are retained and used to help encode test cubes in *testcube-set-2*. Similarly, results are shown for the *m=3* case where unused free variables are retained between *testcube-set-1* and *testcube-set-2*, and well as between *testcube-set-2* and *testcube-set-3*. In all cases, the percentage improvement in test compression is calculated in comparison to the conventional case where the cores are tested serially.

As can be seen in the results, retaining free variables provides more flexibility during test scheduling to allow greater efficiency resulting in significantly better compression. The improvement is less when the scan architecture is optimized in Table 1 because conventional test compression does better to begin with, but it is still a significant improvement. In Table 2 where the scan architecture is not optimized, the greater flexibility in test scheduling is a big help in improving compression, and the final numbers for test time and tester data for *m=3* in Table 2 get close to those in Table 1.

The effect of changing the number of tester channels on the test time is shown in the graph in Fig. 4 for Design *B*. The test time is plotted versus the number of tester channels for serial testing as well as the *m=1, m=2,* and *m=3* cases using the proposed scheme. The effect on the amount of data stored on the tester (i.e., the amount of compression) is shown in Fig. 5. Note that the overall amount of compression is relatively constant regardless of the number of tester channels, although it tends to degrade slightly as the number of channels is increased.

Note that in these experiments, the set of test cubes is fixed and must be encoded as is (i.e., *static encoding*). An alternative (which is used in EDT [Rajski 04]) is to incorporate the encoding process into the automatic test pattern generation (ATPG) (i.e., *dynamic encoding*). The proposed scheme can also be used with dynamic encoding in the manner described in [Muthyala 12].

Table 1. Results Using the Proposed Scheme when the Number of Scan Chains in Individual Cores is Optimized

Design	Cores	Scan Cells	Serial Testing		m = 1			m = 2			m = 3		
			Test Time	Tester Data	Test Time	Tester Data	Percent Reduction	Test Time	Tester Data	Percent Reduction	Test Time	Tester Data	Percent Reduction
A	34	95322	139381	2230096	116101	1857616	16.7%	105805	1692880	24.1%	83169	1330704	40.3%
B	20	57923	71366	1141856	62006	992096	13.1%	52050	832800	27.1%	44481	711696	37.7%
C	16	45375	59963	959408	54725	875600	8.7%	45003	720048	25.0%	39313	629008	34.4%
D	23	78821	92318	1477088	75173	1202768	18.6%	64011	1024176	30.7%	58450	935200	36.7%
E	28	86379	128365	2053840	106795	1708720	16.8%	97054	1552864	24.4%	78660	1258560	38.7%

Table 2. Results Using the Proposed Scheme when the Number of Scan Chains in Individual Cores is Not Changed

Design	Cores	Scan Cells	Serial Testing		m = 1			m = 2			m = 3		
			Test Time	Tester Data	Test Time	Tester Data	Percent Reduction	Test Time	Tester Data	Percent Reduction	Test Time	Tester Data	Percent Reduction
A	34	95322	169680	2714880	143000	2288000	15.7%	98380	1574080	42.0%	82760	1324160	51.2%
B	20	57923	92660	1482560	82000	1312000	11.5%	49880	798080	46.2%	41920	670720	54.8%
C	16	45375	71560	1144960	65600	1049600	8.3%	37280	596480	47.9%	31980	511680	55.3%
D	23	78821	111920	1790720	94300	1508800	15.7%	56320	901120	49.7%	51700	827200	53.8%
E	28	86379	142780	2284480	121420	1942720	15.0%	79960	1279360	44.0%	71540	1144640	49.9%

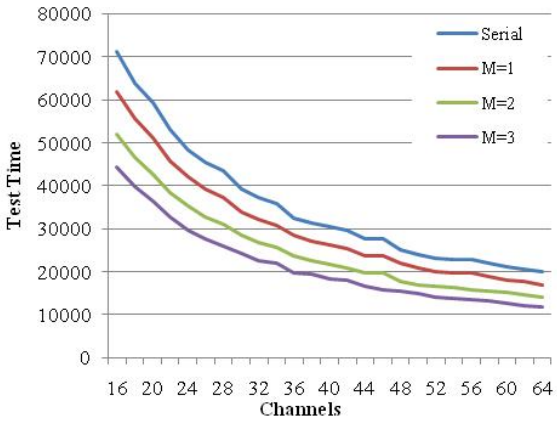

Figure 4. Test Time versus Number of Tester Channels for Design *B*

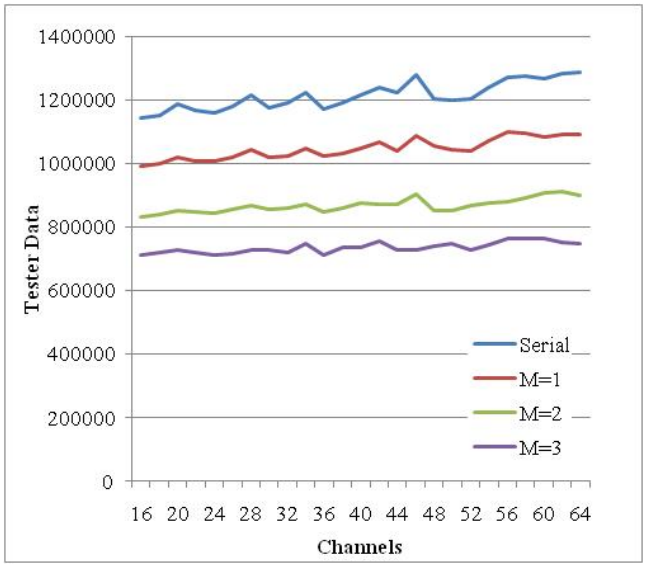

Figure 5. Tester Data versus Number of Tester Channels for Design *B*

7. Conclusions

By retaining unused free variables, the proposed scheme provides greater flexibility in test scheduling for SoC testing resulting in greater test compression. No additional hardware is required to retain the free variables. The computational complexity and control complexity can be adjusted as desired.

Acknowledgements

This research was supported in part by the National Science Foundation under Grant No. CCF-1217750.

References

[Gonciari 05] P.T. Gonciari, P. Rosinger, and B.M. Al-Hashimi, "Compression Considerations in Test Access Mechanism Design", *IEE Proc. Computers & Digital Techniques*, Vol. 152, Issue 1, pp. 89-96, Jan. 2005.

[Iyengar 05] V. Iyengar, and A. Chandra, "Unified SOC Test Approach based on Test Data Compression and TAM Design", *IEE Proc. Computers & Digital Techniques*, Vol. 152, Issue 1, pp. 82-88, Jan. 2005.

[Janicki 11] J. Janicki, J. Tyszer, A. Dutta, M. Kassab, G. Mrugulski, N. Mukherjee, and J. Rajski, "EDT Channel Bandwidth Management in SoC Designs with Pattern-Independent Test Access Mechanism", *Proc. of International Test Conference,* Paper 14.1, 2011.

[Janicki 12] J. Janicki, J. Tyszer, G. Mrugulski, and J. Rajski, "Bandwidth-Aware Test Compression Logic for SoC Designs", *Proc. of European Test Symp.*, 2012.

[Kassab 10] M. Kassab, G. Mrugalski, N. Mukherjee, J. Rajski, J. Janicki, and J. Tyszer, "Dynamic Channel Allocation for Higher EDT Compression for SoC Designs," *Proc. of International Test Conference*, Paper 9.2, 2010.

[Kinsman 10] A.B Kinsman and N Nicolici "Time-Multiplexed Compressed Test of SOC Designs", *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol.18, no.8, pp 1159-1172, Aug 2010

[Könemann 91] B. Könemann, "LFSR-Coded Test Patterns for Scan Designs", *Proc. of European Test Conference*, pp. 237-242, 1991.

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

[Muthyala 12] S.S. Muthyala and N.A. Touba, "Improving Test Compression by Retaining Non-Pivot Free Variables in Sequential Linear Decompressors," *Proc. of International Test Conference,* Paper 9.1, 2012.

[Rajski 04] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded Deterministic Test", *IEEE Trans. on Computer-Aided Design*, Vol. 23, Issue 5, pp. 1306-1320, May 2004.

[Tehranipoor 05] M. Tehranipoor, M. Nourani, and K. Chakrabarty, "Nine-Coded Compression Techniques for Testing Embedded Cores in SoCs", *IEEE Trans. on VLSI Systems*, Vol. 13, No. 6, pp. 719-722, Jun. 2005.

[Touba 06] N.A. Touba, "Survey of Test Vector Compression Techniques", *IEEE Design & Test Magazine,* Vol. 23, Issue 4, pp. 294-303, Jul. 2006.

[Xu 05] Q. Xu and N. Nicolici, "Resource-Constrained System-on-a-Chip Test: A Survey", *IEE Proc. Computers & Digital Techniques*, Vol. 152, Issue 1, pp. 67-81, Jan. 2005.

[Wang 06] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability*, Morgan Kaufmann, 2006.

[Wang 07] Z. Wang., K. Chakrabarty, S. Wang, "SoC Testing Using LFSR Reseeding, and Scan-Slice-Based TAM Optimization and Test Scheduling", *Design, Automation & Test in Europe Conference & Exhibition,2007. DATE '07,* pp 1-6.