Low Power Test Data Compression Based on LFSR Reseeding

Jinkyu Lee and Nur A. Touba

Computer Engineering Research Center
University of Texas at Austin, Austin, TX 78712-1084

Abstract

Many test data compression schemes are based on LFSR reseeding. A drawback of these schemes is that the unspecified bits are filled with random values resulting in a large number of transitions during scan-in thereby causing high power dissipation. This paper presents a new encoding scheme that can be used in conjunction with any LFSR reseeding scheme to significantly reduce test power and even further reduce test storage. The proposed encoding scheme acts as a second stage of compression after LFSR reseeding. It accomplishes two goals. First, it reduces the number of transitions in the scan chains (by filling the unspecified bits in a different manner), and second it reduces the number of specified bits that need to be generated via LFSR reseeding. Experimental results indicate that the proposed method significantly reduces test power and in most cases provides greater test data compression than LFSR reseeding alone.

1. Introduction

As the size and the complexity of systems-on-a-chip (SOC) continue to grow, test data volume has increased dramatically. Several commercial tools for test data compression have been introduced including TestKompress by Mentor Graphics [Rajski 02] and SmartBIST by Cadence [Koenemann 01]. All of the commercial tools that have been introduced so far are based on LFSR reseeding [Koenemann 91].

The basic idea in LFSR reseeding is to generate deterministic test cubes by expanding seeds. A seed is an initial state of the LFSR that is expanded by running the LFSR in autonomous mode. Since typically 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, reseeding can significantly reduce test data storage and bandwidth.

Several reseeding schemes have been proposed to reduce test storage. The first was introduced in [Koenemann 91]. Several techniques were proposed to improve the encoding efficiency of the basic scheme in [Koenemann 91] including using multiple-polynomial LFSRs [Hellebrand 92], using test cube concatenation [Hellebrand 95], and using variable-length seeds [Zacharia 95]. More recent work has focused on dynamic LFSR reseeding where the seed is incrementally modified as the LFSR runs [Koenemann 01], [Krishna 01], [Rajski 02]. 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.

While reseeding is a very powerful method for test data compression, it is not good for power consumption. The don't care bits in each test cube get filled with random values thereby resulting in excessive switching activity when they are shifted into a scan chain. When scanning in test vectors where 95% to 99% of the bits have been filled with random values, a very large percentage of the flip-flops will make transitions thereby resulting in excessive power consumption during test. The chip may be designed to only handle the power consumption during normal operation, and thus the excessive power consumption during test can result in overheating. One solution to this problem is to simply reduce the scan frequency, however this results in longer test times. Many techniques for reducing power consumption during scan testing have been presented and are summarized in [Girard 02]. While most work has focused on reducing test power for general scan testing, only recently has work been done on considering together the problems of test data compression and low power test. Research in this direction has been presented in [Sankaralingam 00], [Chandra 01, 02], and [Rosinger 02] and is summarized in Sec. 2.

In this paper, we present a new encoding algorithm that can be used in conjunction with any LFSR reseeding scheme to significantly reduce power consumption during test. A key feature of the proposed approach is that it reduces the number of specified bits and the number of transitions at the same time. Since the amount of compression for LFSR reseeding depends on the number of specified bits, the proposed approach exploits this property. Experimental results indicate that the proposed method significantly reduces test power and in most cases provides greater test data compression than LFSR reseeding alone.

2. Related Work

The idea of considering together the problems of test data compression and low power test has been previously investigated in a few papers. In [Sankaralingam 00], a procedure for directing the static compaction process in a manner that reduces test power and test data was described. In [Chandra 01], an encoding algorithm that reduces both test storage and test power was presented. The test cubes

are encoded using a Golomb code which is a run-length code. All don't care bits are mapped to 0 and the Golomb code is used to encode runs of 0's. The Golomb code efficiently compresses the test data, and the mapping of the don't cares to all 0's reduces the number of transitions during scan-in and thus power. One drawback of a Golomb code is that it is very inefficient for runs of 1's. In fact, the test storage can even increase for test cubes that have many runs of 1's.

A method based on an alternating run-length code was presented in [Chandra 02] to improve on the encoding efficiency of a Golomb code. While a Golomb code only encodes runs of 0's, an alternating run-length code can encode both runs of 0's and runs of 1's. However, the code becomes inefficient when a pattern is encoded where the runs are short.

While both Golomb codes and alternating run-length codes are good for reducing test power, they are not as efficient as LFSR reseeding for compressing test data. With LFSR reseeding, only the specified bits, which generally account for only 1-5% of the test data, need to be considered.

One previous method has been proposed in [Rosinger 02] for reducing test power for LFSR reseeding. Two LFSRs are used. The main LFSR generates the test cube through conventional reseeding. An extra "masking" LFSR is used to generate a set of mask bits. If the number of 1's in a test cube is less than the number of 0's, then the output of the two LFSRs are ANDed together and the mask cube will have a 1 for each specified 1 in the test cube and an X for each specified 0 or X in the test cube. If the number of 0's in the test cube is less than the number of 1's, then the output of the two LFSRs are ORed together and the mask cube will have a 0 for each specified 0 in the test cube and an X for each specified 1 or X in the test cube. A seed is computed for the extra "masking" LFSR so that it generates the mask cube. Thus the effective number of specified bits that must be generated using this method is equal to the original number of specified bits in the test cube plus the number of specified bits in each mask cube (which is equal to the minimum of the number of 0's or 1's in each test cube). The size of the main LFSR is the same as for conventional reseeding, and the size of the extra "masking" LFSR depends on the maximum number of specified bits in any mask cube. Test power is reduced because the output of the two LFSRs are ANDed or ORed, thus reducing the transition probability. However, the test data compression for this scheme is greatly reduced compared with conventional LFSR reseeding because it requires storing an extra set of seeds for the extra "masking" LFSR. Results in the paper indicate that the test storage was increased by 21% to 54% compared with conventional LFSR reseeding while the transition count was reduced by about 24%.

The proposed method addresses the problem of reducing test power for LFSR reseeding. However, a different approach is taken which does not compromise the amount of test data compression. Moreover, the number of transitions is reduced much more significantly than in [Rosinger 02]. The experimental results for the proposed method are compared with the previous methods in Sec. 5.

The proposed scheme has some similarity to the dynamic scan scheme presented in [Samaranayake 02], however there are a number of significant differences. Both schemes use the fact that different test cubes have compatible values for a significant number of scan elements. The dynamic scan scheme identifies scan chain segments that have compatible values across a set of test cubes and bypasses those segments in order to reduce test time and test storage. This approach also serves to reduce power as well, although that is not the focus of the paper. The proposed scheme also takes advantage of compatible scan segments, but in a different way. The proposed scheme exploits the property that the number of transitions in a test cube is smaller than the number of specified bits. By dividing the scan chains into blocks and identifying blocks that do not contain transitions, the proposed approach is able to fill those blocks with constant values. The compatibility of blocks across different test cubes is exploited in reducing control information and not through bypassing. In [Samaranayake 02], scan chain reconfiguration is necessary which requires inserting design-for-test (DFT) logic in the scan chains themselves to provide the bypass capability, whereas for the proposed scheme, DFT logic is inserted only at the inputs of the scan chains and is thus compatible with conventional scan chains. The proposed scheme can be used for hard cores and firm cores.

3. Encoding Algorithm

Let a transition in a test cube be defined as a specified 0 (1) followed by zero or more X's followed by a specified 1 (0). The key idea of the proposed encoding algorithm is to take advantage of the fact that number of transitions in a test cube is always less than the number of specified bits in a test cube. Thus, rather than using LFSR reseeding to directly encode the specified bits as in conventional LFSR reseeding, the proposed encoding algorithm divides the test cube into blocks and only uses LFSR reseeding to produce the blocks that contain transitions. For the blocks that do not contain transitions, the logic value fed into the scan chain is simply held constant. This approach reduces the number of transitions in the scan chains and in most cases also reduces the total number of specified bits that must be generated by the LFSR as compared with conventional LFSR reseeding.

3.1 Basic concepts

The proposed encoding scheme encodes each test cube with two kinds of data: *hold flags* and *data bits*.

Block	Block1					Block2					Block3					Block4				
Original		0	X	X	1		X	1	1	1		1	X	1	X		X	X	X	X
Encoded	**0**	0	X	X	1	**1**	-	-	-	-	**1**	-	-	-	-	**X**	X	X	X	X

Figure 1. Example of encoding test data

Block	Block1					Block2					Block3					Block4				
Original		X	0	1	X		X	0	X	0		X	X	X	X		1	1	1	X
Encoded	**0**	X	0	1	0	**1**	-	-	-	-	**0**	X	X	X	1	**1**	-	-	-	-

Figure 2. Example of conversion procedure (last bit of blocks 1 and 3 are specified to convert blocks 2 and 4 into non-transitons blocks)

Each test cube is divided into several blocks and each block has a one-bit hold flag. The hold flag indicates whether a transition occurs in a block. There are three types of blocks:

1) Transition block (Hold flag = 0)

One or more transitions exist in the block. Either both 0 and 1 are present in the block (e.g., XX1X0X), or only 0 or 1 is present but the last specified bit from a previous block was opposite.

2) Non-transition block (Hold flag = 1)

No transition occurs in current block. Only 0 or 1 is present in the block, and the last specified bit from a previous block is same (e.g., X0XX0X).

3) Don't care block (Hold flag = X)

No specified bits occur in the block, all are don't cares.

If the hold flag for a block is 1, then the data bits in the block are simply held constant from the last data bit in the previous block. If the hold flag is 0, then the data bits are loaded directly from the LFSR. If the hold flag is X, then it can be either treated as a non-transition block or as a transition block with all X data. Both the hold flags and the data bits are generated from a single LFSR using reseeding.

An example of the proposed encoding is shown in Fig. 1. The test sequence in the example is composed of 4 blocks and each block has 1 hold flag and 4 data bits. The hold flags are shown in bold in the "Encoded" bit sequence row. In Fig. 1, the original test cube contains 7 specified bits. However, using the proposed encoding scheme, the encoded data has only 3 specified hold flags and 2 specified data bits giving a total of only 5 specified bits. Thus, the proposed encoding scheme reduces the number of specified bits that need to be generated using LFSR reseeding. As shown in Fig. 1, the 1's in block 2 and block 3 don't need to be generated directly by the LFSR, but are rather generated as a by-product of the fact that the hold flags keep the input to the scan chain held constant at 1. Thus, test data compression can be achieved in this way. Moreover, no transitions will occur when generating block 2 and block 3 because the hold flags are 1 thus keeping all the bits in the blocks constant. This would not be the case in conventional LFSR reseeding where the X's in blocks 1 and 2 get filled with random data which may results in many more transitions. Thus, a reduction in the number of transitions can be achieved in this way.

3.2 Conversion procedure

It is possible to increase the number of non-transition blocks by converting some transitions blocks into non-transition blocks. There are two requirements that must be satisfied in order to convert a transition block into a non-transition block. The first is that it cannot contain both specified 0's and specified 1's. The second is that the last bit of the previous block must be an X. Two examples of this are shown in Fig. 2. Block 2 is initially a transition block even though it only contains specified 0's because the last specified bit in block 1 was a 1. However, the very last bit of block 1 is a don't care, so a *conversion procedure* can be used to specify that don't care as a 0 and thereby convert block 2 into a non-transition block. Even though this conversion required adding an extra specified data bit, the net result is still a reduction in the total number of specified bits because now block 2 is a non-transition block and thus none of its data bits need to be generated by the LFSR. This same conversion procedure can also be used to convert block 4 in Fig. 2 into a non-transition block.

By increasing the number of non-transition blocks, the conversion procedure can help to reduce both test storage (since it can reduce the total number of specified bits) as well as test power (since it can reduce the number of transitions by enabling all the X's in the converted non-transition block to be filled with the same logic value).

3.3 Partitioning into hold cube compatible sets

The test storage for LFSR reseeding depends on the number of specified bits. For each block that is not a don't care block, the hold flag for that block is specified. If the number of specified hold flags becomes larger than the number of the specified test data bits that are reduced by using the proposed encoding scheme, then the encoding scheme would be reducing test power dissipation at the cost of test storage. The test storage would increase because the total number of specified data bits plus specified hold bits would exceed the total number of specified bits in the original test cubes. However, in this section, a method for reducing the number of specified hold flags is introduced.

The key idea is to take advantage of the fact that many test cubes may have compatible assignments in their corresponding hold flags. We will denote the set of hold

flags for one test cube as a *hold cube* since each hold flag can be either a 1, 0, or don't care (X). If several consecutive test cubes have the same hold cube, it is not necessary to change any of the hold flags. Thus, the hold flags could be loaded once and then reused when applying subsequent test cubes. The hold cubes for a pair of test cubes are compatible if they do not conflict in any specified bit positions. In other words, for every bit position where one hold cube has a specified value, the other hold cube has either the same specified value or a don't care (and vice versa). Let a *hold cube compatible set* be defined as a set of test cubes with mutually compatible hold cubes. Since typically only around 1-5% of the data bits in a test cube are specified, the corresponding hold cube will typically have a large number of don't cares. Thus the test cubes can generally be partitioned into a relatively small number of hold cube compatible sets. The test cubes can then be ordered so that the test cubes in each hold cube compatible set will occur in succession. Thus, the hold flags only need to be loaded once for each hold cube compatible set. One extra bit per test cube is required to indicate if the hold flags for the current test cube needs to be updated or not.

Test Cube	Hold cube					
1	1	X	X	1	0	X
2	X	1	X	X	1	1
3	1	0	X	1	X	X
4	X	0	0	1	0	X
5	X	X	X	0	1	1

(a) Hold cube before partitioning

Test Cube	Update Flag	Hold cube					
1	1	1	0	0	1	0	X
3	0	X	X	X	X	X	X
4	0	X	X	X	X	X	X
2	1	X	1	X	0	1	1
5	0	X	X	X	X	X	X

(b) Hold cube after partitioning

Figure 3. Hold cube regeneration and reordering

Figure 3 shows an example of partitioning a test set into hold cube compatible sets. The original hold cubes for each test cube are shown in Fig. 3-(a). Originally, they require 16 specified bits. They are then grouped into 2 hold cube compatible sets. The first set contains test cubes 1, 3, and 4, and the second contains test cubes 2 and 5. The test cubes are reordered so that the hold cube compatible sets are grouped together. An extra update flag bit is added to each test cube to indicate if the hold flags need to be updated. This flag is set only for the first test cube in each hold cube compatible set. In this example, the total number of specified bits (including the added update flag bits) is reduced to 14. In our experiments (shown in Sec. 5), we found that in most cases, this encoding scheme was capable of reducing the total number of specified bits (including data bits, hold flags, and update flags) to below that of the original test cubes.

4. Hardware implementation

The hardware implementation for the proposed scheme is shown in Fig. 4. Each scan chain is divided into one or more blocks. Let B be the number of blocks per scan chain. Each scan chain has a *hold flag shift register (HF-SR)* whose size is equal to B. LFSR reseeding is used to generate all of the data for each test cube which consists of three components: update flag, hold flags, and test data. The format for the data coming out of the LFSR for each test cube is shown in Fig. 5.

Figure 4. Hardware implementation

There is a small finite state machine (FSM) controller that controls where the data coming out from the LFSR is stored. In the first clock cycle, the LFSR generates a single bit which is the update flag. If the update flag is 1, then in the next B clock cycles, the LFSR generates the hold flags for each of the scan chains which are shifted into the HF-SRs. If the update flag is 0, then the HF-SRs are not loaded. Let the length of each scan chain be L. Then for the next L clock cycles, the LFSR generates the test data. For each L/B clock cycles, if the corresponding hold flag for a scan chain is 0, then the scan chain is loaded from the LFSR. If the corresponding hold flag is a 1, then the last value shifted into the scan chain is repeatedly shifted into the scan chain and the data from the LFSR is ignored. After each L/B clock cycles, the hold flag shift register is shifted so that the next hold flag becomes active for its corresponding block and is used as the control signal to a MUX (as shown in Fig. 4). After the scan chains have been filled, then the scan vector is applied to the circuit-under-test and the response is loaded back into the scan chain. The process is then repeated to generate the next scan vector.

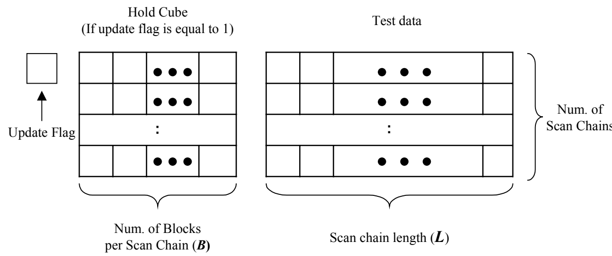

Figure 5. Data format

The hardware overhead consists of one 2-to-1 MUX and a HF-SR per scan chain, one 1-bit update flag flip-flop and the small FSM controller. The FSM controller consists of a bit counter (which is present for LFSR reseeding anyway) and some small combinational logic. The size of the HF-SR dominantly determines the hardware overhead in this scheme. It depends on the number of scan chains and the total number of blocks.

5. Experimental results

Experimental results for the proposed scheme for the largest ISCAS 89 benchmark circuits are shown in Table 1. Results are shown for dividing each test cube into different numbers of blocks (note that there is one hold flag for each block). The test cubes were partitioned into hold cube compatible sets, and the number of such sets is shown in each case. The total number of specified bits required for the proposed encoding scheme is shown (including update flags, hold flags, and data bits). The total number of specified bits and total number of transitions (computed as described in [Sankaralingam 00]) for the proposed encoding scheme is compared with that for the original test cubes. The number of transitions includes the number of transitions in HF-SRs. In most cases, the total number of specified bits is reduced (which will result in less test data storage) while a substantial reduction in the total number of transitions is obtained (which results in less test power). Note that there is a tradeoff between the numbers of specified bits in the test data and hold flags. The more blocks that are used, the less specified bits in the test data, but the more specified hold flags. Moreover, the reduction in the number of transitions increases as the number of blocks in a test pattern increases. The hardware overhead also increases in this case as the size of the hold flag shift registers becomes larger. The hardware overhead depends the number of scan chains, which is chosen depending on the circuit size. The third to last column indicates

Table 1. Results for proposed encoding scheme

Circuit Information				Test Storage				Test Power		Overhead		Test time
Circuit Name	Num. Test Pattern	Num. Blocks	Num. Compatible Sets	Num. Specified Data Bits	Num. Specified Hold lags	Total Specified Bits	Change (%)	Num. Transitions	Transition Reduction (%)	Num Mux	Size HF-SR	Change (%)
s5378	196	31	143	4162	1966	6128	+15	40233	52	11	3	+11
		22	126	4664	986	5650	+4	148981	40		2	+7
s9234	205	31	191	6743	3154	9897	-4	51426	52	11	3	+12
		11	133	8495	1275	9767	-5	204919	38		1	+3
s13207	266	100	114	7006	2391	9397	-1	169626	53	21	5	+6
		20	93	8183	1152	9335	-1	968199	41		1	+1
s15850	269	51	125	8952	2157	11109	0	271198	48	21	3	+4
		31	118	9301	1627	10928	-1	482798	43		2	+2
s38417	376	185	141	20084	8777	28861	-5	917155	50	31	6	+4
		152	146	20933	7399	28332	-7	735408	52		5	+3
s38584	296	209	255	21233	12348	33581	+25	398761	53	31	7	+12
		21	205	24964	3630	28594	+8	6037549	27		1	+1

Table 2. Results comparing partial reseeding alone with using partial reseeding in conjunction with the proposed scheme

Circuit	Partial Reseeding Algorithm [Krishna 01]		Partial Reseeding with Proposed Encoding Scheme				
	Num. Specified Bits	Test Storage	Num. Specified Bits	Test Storage	Reduction of Specified Bits (%)	Change in Test Storage (%)	Transition Reduction (%)
s5378	508	533	501	514	1	-3.6	54
s9234	5198	5537	4031	4556	22	-17.7	57
s13207	2824	3008	2779	3021	1	+0.4	56
s15850	5092	5204	4166	4896	18	-5.9	57
s38417	23984	24513	20688	23166	13	-5.5	52
s38584	2848	2942	2263	2827	7	-3.9	45

Table 3. Results comparing proposed scheme with previous schemes

Circuit	Dual-LFSR reseeding [Rosinger 02]			Alternating run-length code [Chandra 02]			Proposed scheme		
	Test Storage	Comp. (%)	Power Reduction (%)	Test Storage	Comp. (%)	Power Reduction (%)	Test Storage	Comp. (%)	Power Reduction (%)
s9234	19440	68	24	21612	44	76	10302	79	53
s13207	11803	94	25	32648	80	93	10484	94	53
s15850	14518	90	25	26306	65	85	11411	93	52
s38417	66234	92	25	64976	60	81	32152	95	52
s38584	23835	94	25	77372	61	83	31152	93	40

the number of 2-to-1 MUXs required which is equal to the number of scan chains because one MUX is located on the entrance of each scan chain. The size of HF-SR is indicated in the second to last column. It depends on the number of blocks. Test time is increased because it takes several clock cycles to insert hold flags into HF-SRs. As shown in the last column, the test time increase due to the proposed scheme is very small.

The proposed encoding scheme can be used in conjunction with any LFSR reseeding scheme. Experiments were performed for using the proposed encoding scheme in conjunction with the partial LFSR reseeding scheme described in [Krishna 01]. Results are shown in Table 2. The exact same set of test cubes that were used for generating the results published in [Krishna 01] were encoded using the proposed encoding scheme in conjunction with the scheme in [Krishna 01]. As can be seen, in most cases both the test storage and test power are reduced using the proposed scheme.

Table 3 shows a comparison of the experimental results in [Rosinger 02] and [Chandra 02] with the proposed encoding scheme (used in conjunction with partial LFSR reseeding as described in [Krishna 01]). As can be seen, the proposed scheme reduces the test storage requirements much more than the other schemes. In terms of reducing test power, the proposed scheme is much more effective than the scheme in [Rosinger 02] which is also applicable for LFSR reseeding. Moreover, the compression ratio in the proposed scheme is similar or even higher than that in [Rosinger 02] even though 1,000 pseudo-random patterns are applied first in [Rosinger 02]. Note that the results for both [Chandra 02] and the proposed scheme are for encoding the entire deterministic test set. While the test power for the proposed scheme is not reduced as much as for the scheme in [Chandra 02] which is based on run-length encoding, much more compression is achieved. The key advantage of the proposed scheme compared with [Chandra 02] is that it is compatible with LFSR reseeding which is used in commercial tools due to its superior encoding efficiency.

6. Conclusion

LFSR reseeding is a powerful approach for reducing test storage. The proposed encoding scheme provides a way to reduce test power for LFSR reseeding. It acts as a second stage of compression after LFSR reseeding. By employing hold flags, not only is test power reduced, but also test storage can be reduced.

Acknowledgements

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

References

[Chandra 01] Chandra, A., and K. Chakrabarty, “Combining low-power scan testing and test data compr---ession for system-on-a-chip,” *Proc. of Design Automation Conf.*, pp. 166-169, 2001.

[Chandra 02] Chandra, A., and K. Chakrabarty, “Reduction of SOC Test Data Volume, Scan Power and Testing Time Using Alternating Run-length Codes,” *Proc. of Design Automation Conf.*, pp. 673-678, 2002.

[Girard 02] Girard, P., “Survey of Low-Power Testing of VLSI Circuits,” *IEEE Design & Test of Computers*, pp. 82-92, 2002.

[Hellebrand 92] Hellebrand, S., S. Tarnick, J. Rajski, and B. Courtois, “Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Register,” *Proc. of International Test Conference*, pp. 120-129, 1992.

[Hellebrand 95] Hellebrand, S., 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. on Computers*, Vol. 44, No. 2, pp. 223-233, Feb. 1995

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

[Koenemann 01] Koenemann, B., C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, “A SmartBIST variant with guaranteed encoding,” *Proc. of VLSI Test Symposium*, pp. 325-330, 2001

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

[Rajski 02] Rajski, J., J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, T. Kun-Han, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eider, and Q. Jun, “Embedded deterministic test for low cost manufacturing test,” *Proc. of International Test Conference*, pp. 301-310, 2002.

[Rosinger 02] Rosinger, P. M., B.M. Al-Hashimi, and N. Nicolici, “Low Power Mixed-Mode BIST Based on Mask Pattern Generation Using Dual LFSR Re-seeding,” *Proc. of Int. Conf. on Comp. De.*, pp. 474-479, 2002.

[Samaranayake 02] Samaranayake, S., N. Sitchinava, R. Kapur, M.B. Amin, and T.W. Williams, “Dynamic Scan: Driving Down the Cost of Test,” *Computer*, Vol. 35, Issue 10, pp. 63 – 68, 2002

[Sankaralingam 00] Sankaralingam, R., R.R. Oruganti, and N.A. Touba, “Static compaction techniques to control scan vector power dissipation,” *Proc. of VLSI Test Symp.*, pp. 35-40, 2000.

[Zacharia 95] Zacharia, N., J. Rasjski, and J. Tyszer, “Decompression of Test Data Using Variable-Length Seed LFSRs,” *Proc. of VLSI Test Symposium*, pp. 426-433, 1995.