Achieving High Encoding Efficiency With Partial Dynamic LFSR Reseeding

C. V. KRISHNA, ABHIJIT JAS, and NUR A. TOUBA
University of Texas, Austin

Previous forms of LFSR reseeding have been static (i.e., test application is stopped while each seed is loaded) and have required full reseeding (i.e., the length of the seed is equal to the length of the LFSR). A new form of LFSR reseeding is described here that is dynamic (i.e., the seed is incrementally modified while test application proceeds) and allows partial reseeding (i.e. length of the seed is less than that of the LFSR). In addition to providing better encoding efficiency, partial dynamic LFSR reseeding has a simpler hardware implementation than previous schemes based on multiple-polynomial LFSRs.

Categories and Subject Descriptors: B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance

General Terms: Design, Reliability

Additional Key Words and Phrases: Built-in self-test, compression, linear finite shift register, reseeding

1. INTRODUCTION

As integration density scales with technology, manufacturing test cost is contributing a larger share to the total cost of manufacturing a chip [ITRS 2001]. One of the key contributing factors to test cost is test time and test data volume. The limitations in the ability of external ATE (automatic test equipment) to scale for increasingly complex integrated circuit (IC) designs are well known. As more and more logic is placed on a single chip, both the test data storage requirements on the tester and the test data bandwidth requirements between the tester and chip are growing rapidly [Khoche and Rivoir 2000]. Buying new testers with more memory, channels, and higher speed of operation is not a good solution to this problem because such testers are prohibitively expensive. Test resource partitioning (TRP) provides a low-cost alternative solution to this problem. In TRP, some hardware is added on the chip to ease the burden on the

Authors' address: Computer Engineering Research Center, Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712-1084; email: {krishna,jas,touba}@ece.utexas.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2004 ACM 1084-4309/04/1000-0500 $5.00

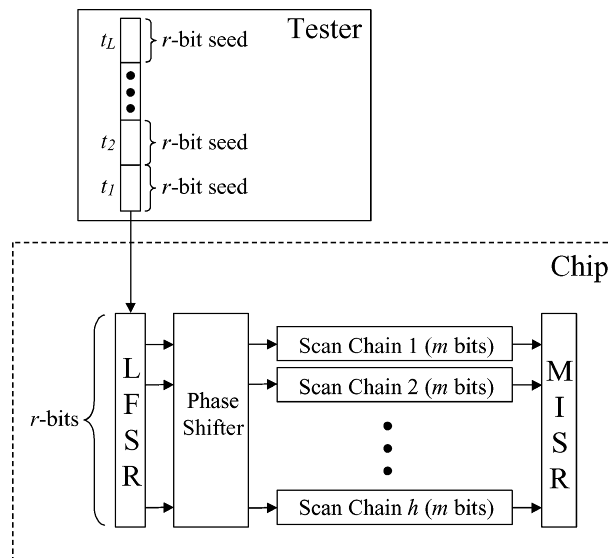

Fig. 1. Static LFSR reseeding (using an r-bit LFSR).

external tester. Such hardware often works in conjunction with the tester and helps reduce the test data and/or the test application time.

One attractive approach that has been used for compressing the amount of test data that needs to be stored on the tester and transferred to the chip is to use linear feedback shift register (LFSR) reseeding [Könemann 1991]. This approach is illustrated in Figure 1. An *LFSR seed* is the starting state of an LFSR when the LFSR is run in autonomous mode to fill a set of scan chains with a test vector (if there are m bits in each scan chain, then the LFSR is run for m cycles to fill the scan chains). Different LFSR seeds will produce different test vectors. Given a set of deterministic test cubes (test vectors in which bits unassigned by ATPG are left as don't cares and denoted by X's), the idea in LFSR reseeding is to compute a set of seeds that when expanded by the LFSR will produce the deterministic test cubes. A seed can be computed for each test cube by solving a system of linear equations based on the feedback polynomial of the LFSR [Könemann 1991]. So instead of storing each full test vector on the tester, a much smaller LFSR seed is stored instead (in Figure 1, a set of L test cubes are stored on the tester as a set of L seeds). The set of seeds stored on the tester are transferred to the LFSR one at a time and expanded into the corresponding full test vector in the scan chains. Since the seeds are much smaller than the full test vectors, the test data storage and bandwidth requirements for the external tester can be reduced by an order of magnitude or more. Another nice property of LFSR reseeding is that it can be seamlessly combined with pseudorandom built-in self-test (BIST) to form a mixed-mode testing approach. The LFSR can first be run in autonomous mode to generate some number of pseudorandom patterns to detect the random pattern testable faults, and then reseeding can be used to generate deterministic test cubes to detect the random pattern resistant faults. The encoding efficiency (μ) for a set

of test cubes is defined as the total number of specified bits in the test cubes (S_T) divided by the total number of bits required to encode it (S_E) [Hellebrand et al. 1995a].

Several techniques for improving the encoding efficiency of the basic LFSR reseeding methodology (originally described in [Könemann 1991]) have been proposed in Hellebrand et al. [1992, 1995a, 1995b], Venkataraman et al. [1993], Zacharia [1995], Zacharia et al. [1996], and Rajski et al. [1998a]. These techniques will be described in detail in Section 2 and compared with the new technique proposed in this article (preliminary results were published in Krishna et al. [2001]). This article describes a new form of LFSR reseeding that provides higher encoding efficiency and hence greater reduction in test data storage and bandwidth requirements. Previous forms of LFSR reseeding have been *static* (i.e., test application is stopped and the seed is loaded at one time) and have required *full reseeding* (i.e., $n = r$ bits are used for an r-bit LFSR). The new form of LFSR reseeding proposed here is *dynamic* (i.e., the seed is incrementally modified while test application proceeds) and allows *partial reseeding* (i.e., $n < r$ bits can be used). Full static forms of LFSR reseeding can be shown to be a special case of the new partial dynamic form of LFSR reseeding. In addition to providing better encoding efficiency, the partial dynamic form of LFSR reseeding proposed here has a simpler hardware implementation than existing methods based on multiple-polynomial LFSRs, and can generate each test vector in fewer clock cycles. A complete methodology based on partial LFSR reseeding is described in this article for compressing a set of deterministic test cubes.

Note that, since the preliminary version of this work was published [Krishna et al. 2001], more recent work [Könemann et al. 2001; Rajski et al. 2002] has appeared which also utilizes a form of partial dynamic LFSR reseeding. However, the hardware schemes and methodology are different. The SmartBIST technique described in Könemann et al. [2001] uses a variable number of bits to encode each test cube. This is accomplished by having an extra channel from the tester that occasionally disables the scan clock. Encoding with a variable number of bits allows for greater encoding efficiency, but this is somewhat offset by the need for an extra tester channel and its associated test storage requirements. The proposed approach uses a fixed number of bits to encode each test cube thereby eliminating the need for the additional clock disable signal required in Könemann et al. [2001]. The Embedded Deterministic Test method described in Rajski et al. [2002] uses a ring generator which is an alternative linear finite state machine that offers some advantages over an LFSR. A major difference between the method in Rajski et al. [2002] and the proposed method is that in Rajski et al. [2002], the contents of the ring generator are reset between test cubes. This decouples the linear equations across different test cubes which reduces computation time, but this comes at the cost of less encoding efficiency. In the proposed method, partitioning strategies are described for trading off computation time with encoding efficiency as desired. This article also describes how "scan windows" can be used to reduce the size of the LFSR (note that this was not included in the preliminary version of this work in Krishna et al. [2001]).

The article is organized as follows: Section 2 discusses previous work done in this area. Section 3 gives an overview of the proposed method. Section 4 explains in details how the linear equations are formed and solved for dynamic partial reseeding. Section 5 discusses the concept of "scan windows" that can be used to reduce the hardware overhead as well as to allow the system of linear equations to be solved efficiently. Section 6 gives experimental results and Section 7 is a conclusion.

2. PREVIOUS WORK

The original idea of encoding scan patterns as LFSR seeds was proposed in Könemann [1991]. An encoding efficiency of 1 corresponds to the case where the set of test cubes was encoded with the same number of bits as the total number specified bits in all the test cubes. For the basic LFSR reseeding approach described in Könemann [1991], the encoding efficiency is limited by two factors:

(1) *Linear Dependencies in the LFSR*: The LFSR must be large enough to yield a solution to the linear equations for all the test cubes in the set. If s_{max} denotes the largest number of specified bits in any test cube in the set, then it has been estimated that the LFSR should have a length of $s_{max} + 20$ bits in order to reduce the probability of not finding a seed for a test cube to less than 10^{-6} [Chen 1986; Könemann 1991], due to linear dependencies in the LFSR.

(2) *Variance in the Number of Specified Bits in Test Cubes*: The number of specified bits in each test cube can vary considerably, however, the size of the LFSR is restricted by the test cube with the largest number of specified bits (s_{max}). So even though most test cubes may have many fewer than s_{max} specified bits, they still are encoded with LFSR seeds having $s_{max} + 20$ bits.

So the LFSR reseeding approach described in Könemann [1991] requires $s_{max} +$ 20 bits to encode each test cube regardless of the number of specified bits in the test cube.

Hellebrand, et al. [1992], proposed a method for improving the encoding efficiency of LFSR reseeding by using a *multiple-polynomial LFSR (MP-LFSR)*. An MP-LFSR has a programmable feedback function, and hence can implement different feedback polynomials. The linear dependency problem can be solved by having the ability to choose between different feedback polynomials when encoding a test cube. They showed that with 16 different polynomials, the probability of not being able to encode a test cube with s_{max} specified bits in an MP-LFSR with length s_{max} is less than 10^{-6}. This means that rather than using an LFSR with length $s_{max} + 20$ bits, an MP-LFSR with length s_{max} bits can be used instead. However, some means of identifying which polynomial to use for a particular seed is required. This can be accomplished implicitly by grouping together the seeds for specific polynomials and using a "next-bit" to indicate when the feedback polynomial needs to be changed [Venkataraman et al. 1993]. Thus, the number of bits required to encode each test cube can be reduced to $s_{max} + 1$bit. However, the encoding efficiency is still

limited by the fact that most test cubes may have many fewer than $s_{\max}$ specified bits.

Two approaches for addressing the problem related to the variance in the number of specified bits in the test cubes have been proposed. One involves concatenating test cubes [Hellebrand et al. 1995a], and the other involves using variable-length seeds [Zacharia 1995; Zacharia et al. 1996; Rajski et al. 1998a].

The idea of concatenating test cubes was proposed in Hellebrand et al. [1995a]. Instead of expanding each seed into a single test cube, it involves expanding each seed into some fixed number of test cubes, j. This is done by loading each seed into an MP-LFSR, and running the MP-LFSR in autonomous mode to generate the next j test vectors. So the set of test cubes is partitioned into groups where each group has no more than j test cubes in it and no more than a total of $s_{\max}$ specified bits. A bin-packing algorithm is used to partition the test cubes into as few groups as possible under these constraints. For the groups that have fewer than j test cubes, "dummy" test cubes are inserted. The linear equations for each test cube in a group are concatenated and solved all together to find a seed that will produce all the test cubes in the group. Each group of test cubes requires $s_{\max} + 1$ bits to encode. This approach allows test cubes with a small number of specified bits to be grouped together to achieve a better encoding efficiency. In this approach, the encoding efficiency is limited by how close the number of specified bits in each group is to $s_{\max}$. In Hellebrand et al. [1995b], a special ATPG procedure is described for improving the encoding efficiency of test cube concatenation.

An alternative to concatenating test cubes is to use variable-length seeds as proposed in Rajski et al. [1998a]. Here the idea is to configure part of the scan chains into variable size LFSRs. For test cubes with larger numbers of specified bits, larger LFSRs are used, and for test cubes with smaller numbers of specified bits, smaller LFSRs are used. Identifying the size of the LFSR for a seed can be accomplished implicitly by grouping together the seeds for specific LFSR sizes and using a "next-bit" to indicate when the LFSR size needs to be changed. When encoding a test cube with s specified bits, either an LFSR with $s + 20$ bits can be used, or an MP-LFSR with s bits plus a polynomial identifier having $\lceil log_2(number\ of\ polynomials) \rceil$ bits can be used. The encoding efficiency of this approach is limited by the extra bits required for identifying the size and polynomial of the MP-LFSR for each seed, and by the granularity in the variable size LFSRs (e.g., if the LFSR sizes go up by increments of Δ bits, then some seeds could be up to $\Delta - 1$ bits longer than necessary).

Note that the previous approaches for LFSR reseeding have involved *static reseeding*. Static reseeding is defined here as stopping test application and loading a new seed before resuming test appplication. All the approaches except for test cube concatenation [Hellebrand et al. 1995a] stop the test application after each test vector to load a new seed. The test cube concatenation approach applies a small fixed number of test cubes (e.g., $j = 8$) before loading a new seed, but when it does load a new seed, it stops the test application to do so. The reseeding approach proposed in this article is a *dynamic reseeding* method in which the seed is modified incrementally while the test application proceeds. In addition to reducing the number of cycles required to apply each test vector,

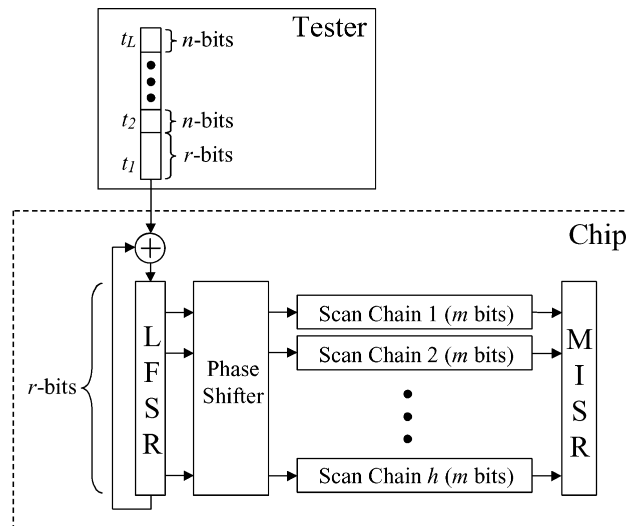

Fig. 2. Proposed partial reseeding scheme. ($n < r$ for an r-bit LFSR).

dynamic reseeding has some other nice properties that result in better encoding efficiency as will be described in Section 3.

The previous approaches for LFSR reseeding have also required *full reseeding*. If n is the number of bits that are used for reseeding, then full reseeding is defined here as the case where n equals r for an r-bit LFSR. The variable-length seed method [Rajski et al. 1998a] allows for shorter seeds, but the size of the seed is still equal to the size of the variable length LFSR. The reseeding approach proposed in this article allows for partial reseeding, where n is less than r for an r-bit LFSR.

3. OVERVIEW OF PROPOSED PARTIAL RESEEDING METHOD

The proposed partial reseeding approach is illustrated in Figure 2. Note that an extra XOR gate is included in the feedback of the LFSR. This is similar to the architecture described in Kay and Maurad [2000], although it is used in a different way here. The LFSR length, r, is at least $s_{\max}+20$ where $s_{\max}$ is the maximum number of specified bits in any test cube. The r-bit LFSR is initialized with a starting r-bit seed. This initial seed is used to generate the first test cube by running the LFSR for m clock cycles (where m is the scan length) to fill the scan chains. For the second test cube, the LFSR is run for another m clock cycles to generate the next test cube. However, during each of the first n clock cycles, a bit is shifted in from the tester and XORed with the feedback of the LFSR. These n bits coming in from the tester alter the state of the LFSR and in effect "dynamically reseed" the LFSR. For an r-bit LFSR, n is significantly smaller than r, so it is a "partial reseeding."

After the first n clock cycles, the tester stops shifting in data and the LFSR simply cycles through its normal sequence of states until the scan chains are full. This partial dynamic reseeding process is repeated for each of the subsequent test cubes that are generated by the LFSR. For each test cube, a bit

is shifted in from the tester during each of the first n clock cycles as the scan chains are filled. The total number of bits required to encode a set of L test cubes using the proposed approach with an r-bit LFSR is $n \cdot (L-1) + r$. Notice that the number of bits required for encoding is not proportional to $s_{\max}$. This is a nice property. For all the previously proposed approaches for LFSR reseeding, with the exception of the variable-length seed method [Rajski et al. 1998a], the number of bits required for encoding is proportional to $s_{\max}$. The variable-length seed method avoids dependence on $s_{\max}$ but at the cost of the extra complexity needed to implement variable-size LFSRs.

The additional hardware required for the proposed partial reseeding method beyond what is needed for the standard STUMPS architecture [Bardell and McAnney 1982] is just an additional XOR gate in the feedback of the LFSR which is controlled from the tester. There is no need for a multiple-polynomial LFSR or any added complexity. The simplicity of partial reseeding is another nice property that it has.

In dynamic reseeding, the state of the LFSR after the application of test cube t_i carries forward to the generation of the test cube t_{i+1}. This is very beneficial in the following way. Depending on the number of specified bits in test cube t_i, the solution space for the system of linear equations that is solved to find a seed that produces t_i can be very large. Generally, the fewer specified bits in test cube t_i, the larger the solution space is for seeds that generate t_i. With dynamic reseeding, the degrees of freedom in the solution space for t_i can be used to ease the problem of finding a solution for t_{i+1}. By using the degrees of freedom in the solution space for t_i, fewer additional bits need to be shifted in from the tester to find a solution for t_{i+1}. This allows n (the number of bits coming from the tester) to be smaller than r (the size of the LFSR) which results in partial reseeding. In static reseeding methods, the state of the LFSR after applying test cube t_i is completely overwritten when a new seed is loaded for test cube t_{i+1}. Hence, the degrees of freedom in the solution space for test cube t_i are completely wasted. With dynamic reseeding, the degrees of freedom in the solution space for test cube t_i are preserved and can be used for solving the linear equations for subsequent test cubes. This results in a much better encoding efficiency.

For partial dynamic reseeding, to maximally exploit the ability to use the degrees of freedom in the solution space of the previous test cube when solving the linear equations for next test cube, the test cubes should be ordered in the following way. The test cubes with the most number of specified bits should be interleaved with the test cubes with the fewest number of specified bits (e.g., have the least specified test cube followed by most specified test cube, followed by second least specified test cube, followed by second most specified test cube, etc.). This eases the burden on solving the linear equations by matching the larger solution spaces for a preceding test cube with the most specified (i.e., hardest to solve test cubes), and the smaller solution spaces for a preceding test cube with the least specified (i.e., easiest to solve) test cubes. By so doing, the value of n can be minimized.

So far, in this article, dynamic reseeding has been described with the data being shifted in from the tester at the same time as data is being shifted from

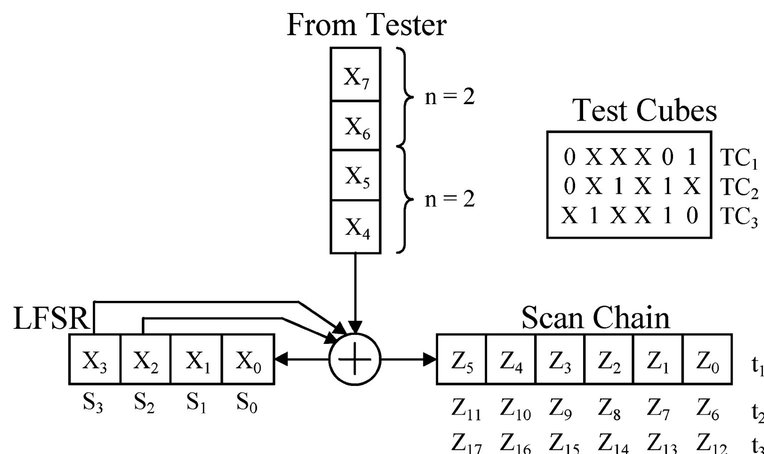

Fig. 3. Example of forming equations for partial reseeding.

the LFSR into the scan chains (*"dynamic reseeding concurrent with scan chain loading"*). However, for some applications, it may be desirable to load the scan chains at a faster clock rate than the tester clock rate. In that case, dynamic reseeding can also be implemented by shifting in the n bits of data from the tester and cycling the LFSR without loading the scan chains (*"dynamic reseeding before scan chain loading"*). After all n bits have come in from the tester and dynamically reseeded the LFSR, then the scan chains can be loaded from the LFSR at a clock rate that is faster than the tester clock rate since no more interaction with the tester is required for that test vector. Dynamic reseeding before scan chain loading retains all the same properties as dynamic reseeding concurrent with scan chain loading (the only difference is the phase of the linear equations). Note that dynamic reseeding before scan chain loading with n equal to r is equivalent to conventional full static reseeding. If n equals r, then the state of the r-bit LFSR can be completely controlled by the $n = r$ bits coming in from the tester. The LFSR can be forced into any state by the proper selection of the $n = r$ bits coming from the tester. Therefore, full static reseeding can be considered a special case of partial dynamic reseeding.

4. FORMING AND SOLVING LINEAR EQUATIONS FOR PARTIAL RESEEDING

Now that partial reseeding has been described, the next issues are how to form and solve the linear equations for the $n \cdot (L-1) + r$ bits that are stored on the tester in order to generate a set of L test cubes, and how to choose the minimum value of n that will result in a solution. Forming the linear equations is done by representing the $n \cdot (L-1) + r$ bits stored on the tester with symbols and symbolically simulating the LFSR operation to generate the linear equations for each specified bit in the test cubes. A small example is shown in Figure 3. A 4-bit LFSR is used to generate three test cubes (TC_1, TC_2, and TC_3) with $n = 2$. In this case, $n \cdot (L-1) + r = 2(3-1) + 4 = 8$. So the test cubes are encoded with 8 bits of data that are symbolically represented by X_0 through X_7. The scan chain is 6 bits long, so there is a total of 18 bits in the 3 test cubes. The equations for

$$\mathbf{T} = \begin{pmatrix} 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \\ 1\,1\,0\,0 \end{pmatrix} \qquad \mathbf{S}(t{=}0) = \begin{pmatrix} X_3 \\ X_2 \\ X_1 \\ X_0 \end{pmatrix}$$

Fig. 4. State transition matrix **T** and starting state **S**(t = 0) for LFSR in Fig. 3.

$$S(1) = T * S(0) = \begin{pmatrix} X_2 \\ X_1 \\ X_0 \\ X_3 + X_2 \end{pmatrix} \qquad S(7) = T * S(6) + \begin{pmatrix} 0 \\ 0 \\ 0 \\ X_4 \end{pmatrix} = \begin{pmatrix} X_2 \\ X_1 \\ X_0 \\ X_3 + X_4 \end{pmatrix}$$

$$S(2) = T * S(1) = \begin{pmatrix} X_1 \\ X_0 \\ X_3 + X_2 \\ X_2 + X_1 \end{pmatrix} \qquad S(8) = T * S(7) + \begin{pmatrix} 0 \\ 0 \\ 0 \\ X_5 \end{pmatrix} = \begin{pmatrix} X_1 \\ X_0 \\ X_3 + X_2 \\ X_2 + X_5 \end{pmatrix}$$

$$\vdots \qquad\qquad\qquad\qquad \vdots$$

$$S(6) = T * S(5) = \begin{pmatrix} X_1 \\ X_0 \\ X_3 + X_2 \\ X_2 + X_1 \end{pmatrix} \qquad S(12) = T * S(11) = \begin{pmatrix} X_1 \\ X_0 \\ X_3 + X_2 \\ X_2 + X_1 \end{pmatrix}$$

Fig. 5. LFSR states during simulation of LFSR in Fig. 3.

these 18 bits are represented by Z_0 through Z_{17}. The equations for Z_0 through Z_{17} are determined based on the *state transition matrix*($\boldsymbol{T}$) of the LFSR and the variables X_0 through X_7. The state transition matrix for the LFSR in Figure 3 is shown in Figure 4. The state of the LFSR can be represented by the bits S_3, S_2, S_1, and S_0, as shown in Figure 3. Given a particular state $\boldsymbol{S}(t)$ of the LFSR at time t, the state $\boldsymbol{S}(t+1)$ of the LFSR after one cycle of simulation of the LFSR is given by the matrix product of the state transition matrix and the current state of the LFSR, that is, $\boldsymbol{S}(t+1) = \boldsymbol{T} * \boldsymbol{S}(t)$. Proceeding in an iterative manner, it can be shown that the state of the LFSR after m cycles is given by $\boldsymbol{S}(t+m) = \boldsymbol{T}^m * \boldsymbol{S}(t)$.

Figure 5 shows how the equations for the variables Z_0 through Z_{17} can be obtained. Since the LFSR is initialized with a 4-bit seed, the starting state of the LFSR is as shown in Figure 4. Since no data is brought from the tester for the generation of the first test cube, the states of the LFSR are updated in a straightforward manner for 6 clock cycles by multiplying the current state with **T**, as shown in the left-hand column in Figure 5. But for the second test cube, during the first two cycles, two new variables are introduced into the LFSR. States ***S(7)*** and ***S(8)*** in Figure 5 show the manner in which the LFSR state is updated during these two clock cycles. For the remaining 4 clock cycles that are required to generate the second test cube, no new variables are introduced from

Test Cube TC_1	Test Cube TC_2	Test Cube TC_3
$Z_0 = X_2 + X_3$	$Z_6 = X_1 + X_2 + X_3 + X_4$	$Z_{12} = X_2 + X_4 + X_6$
$Z_1 = X_1 + X_2$	$Z_7 = X_0 + X_1 + X_2 + X_5$	$Z_{13} = X_1 + X_5 + X_7$
$Z_2 = X_0 + X_1$	$Z_8 = X_0 + X_1 + X_2 + X_3$	$Z_{14} = X_0 + X_4$
$Z_3 = X_0 + X_2 + X_3$	$Z_9 = X_0 + X_1 + X_3 + X_4$	$Z_{15} = X_2 + X_3 + X_4 + X_5 + X_6$
$Z_4 = X_1 + X_3$	$Z_{10} = X_0 + X_3 + X_4 + X_5$	$Z_{16} = X_1 + X_2 + X_4 + X_5 + X_6 + X_7$
$Z_5 = X_0 + X_2$	$Z_{11} = X_3 + X_5$	$Z_{17} = X_0 + X_1 + X_4 + X_5 + X_7$

Fig. 6. Equations for example in Fig. 3.

the tester, and the LFSR state is again updated in a straightforward manner as was done for the first test cube. Note that the equation that corresponds to bit S_0 of the LFSR gets shifted into the scan chain during each clock cycle. Thus bit S_0 of state $\boldsymbol{S}(1)$ corresponds to Z_0, bit S_0 of state $\boldsymbol{S}(2)$ corresponds to Z_1, and so on. The resulting equations for Z_0 through Z_{17} are shown in Figure 6.

Note that for simplicity, the example shows an LFSR feeding a single scan chain, however without loss of generality, the same procedure would apply for an LFSR feeding multiple scan chains. Once the linear equations have been formed, they can be efficiently solved using Gauss–Jordan elimination. For the example in Figure 3 where test cubes t_1, t_2, and t_3, are $0XXX01$, $0X1X1X$, and $X1XX10$, respectively, a solution can be obtained by solving the equations for the bits with specified values. One solution would be $X_0 = 1, X_1 = 1, X_2 = 1, X_3 = 0, X_4 = 1, X_5 = 0, X_6 = 0, X_7 = 0$.

The larger the value of n, the more likely there is to be a solution to the linear equations. If the value of n is too small, a solution may not exist. If the value of n is greater than or equal to $s_{\max} + 20$, then there is an extremely high probability of finding a solution. The minimum value of n for which a solution might reasonably be found would be s_{avg} which is the average number of specified bits per test cube. One strategy for quickly finding a small value of n that gives a solution is to do a binary search between s_{avg} and $s_{\max} + 20$. This would require $\lceil log_2(s_{\max} - s_{avg} + 20) \rceil$ iterations. Each iteration involves forming and trying to solve the system of linear equations.

One disadvantage of partial dynamic reseeding compared with full static reseeding is that the computation time for solving the linear equations is longer. In full static reseeding, the linear equations for each test cube can be solved independently. In partial dynamic reseeding, the linear equations for the test cubes need to be solved all together. Although this results in a more efficient solution, it may not scale well for large test sets. However, the solution for this problem is very simple. For large test sets, the test cubes can be partitioned into smaller subsets of k test cubes each. Partial dynamic reseeding can then be done for each subset of k test cubes. After each subset of k test cubes is generated by the LFSR with partial dynamic reseeding, the seed of the LFSR is re-initialized before the next subset of k test cubes are generated by the LFSR with partial dynamic reseeding. Thus the linear equations for each subset of k test cubes can be formed and solved independently. The value of k can be chosen based

on the amount of computation time that is acceptable. This provides a very easy tradeoff between computation time and the optimality of the result. Note that in the degenerate case where k is equal to 1, partial dynamic reseeding reduces to full static reseeding. It should be noted that generating and solving the system of linear equations for partial dynamic reseeding can be done in polynomial time (it is not exponential), and the procedures are very efficient and fast. Our experiments indicate that values of k in the order of hundreds can be processed in a few hours. So in many cases, if the number of test cubes is in the order of hundreds, it may not be necessary to partition the problem.

Partitioning in some cases can actually be used to obtain a better solution if there is a large variance in the number of specified bits in the test cubes. One strategy for partitioning would be to group the test cubes with the largest number of specified bits in one partition, and the test cubes with a smallest number of specified bits in another partition. The partition with the larger number of specified bits would end up with a larger value of n, and the partition with the smaller number of specified bits would have a smaller value of n. The encoding efficiency for the two separate partitions may be higher than the encoding efficiency of processing all of the test cubes together in one partition. Experimental results for partitioning are discussed in Section 6. Note that the tester program required for handling multiple values of n would be more complex than for a single value of n.

5. SCAN WINDOWS

One of the problems for all LFSR reseeding approaches where a separate LFSR is used (unlike schemes where the LFSRs are configured from the scan chains themselves [Rajski et al. 1998a]) is that the size of the LFSR scales with s_{max} and thus can become problematic for large industrial circuits. This problem can be solved by using "scan windows" [Krishna and Touba 2002]. Note that the technique based on scan windows is a generic technique and can be used for any LFSR reseeding based scheme. The idea is to conceptually (not physically) partition the scan chains into scan windows, and use LFSR reseeding to fill each scan window one at a time, as illustrated in Figure 7. In Figure 7, the scan chains are divided into 3 scan windows where each scan window will have $h(m/3)$ scan cells in it. So instead of generating an entire test cube with one partial seed from the tester, multiple partial seeds are used to generate a single test cube. By doing so the number of specified bits that needs to be generated by each seed is reduced (i.e., the s_{max} for the scan windows is less than the s_{max} for the complete test cubes). The size of the scan windows can be chosen based on how large an LFSR is to be used. If an r-bit LFSR is to be used, then the size of the scan windows can be chosen so that the maximum number of specified bits for any scan window (i.e., the s_{max} of the scan windows) does not exceed r-20. Note that the LFSR can be of any size (> 20) provided an appropriate linear phase shifter is used [Rajski et al. 1998b].

Partial dynamic LFSR reseeding can easily be applied with scan windows. The implementation is very similar to partial reseeding of the entire test vector. Partial reseeding for each w-bit wide scan window is performed by simply

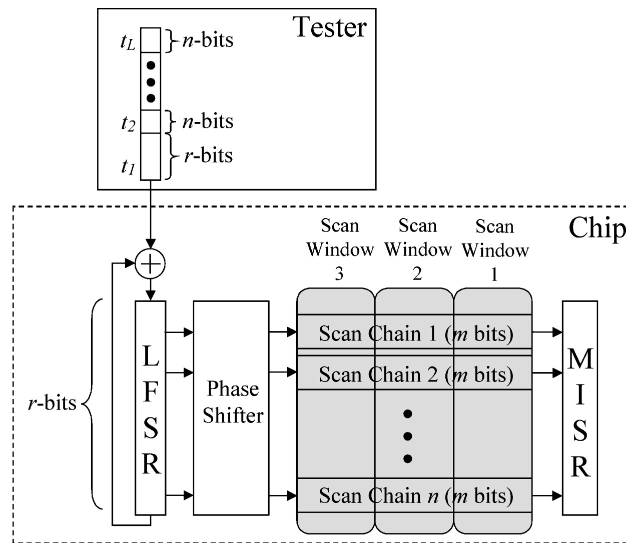

Fig. 7. Architecture with scan windows.

shifting in a bit from the tester and XORing it with the feedback of the LFSR during each of the first $n(< w)$ clock cycles. The LFSR is then run in an autonomous mode for the remaining $w - n$ cycles to fill up the scan window. After all the scan windows have been loaded, the scan chains contain a complete test cube. The system clock is then applied and the partial dynamic reseeding process is repeated for the next test cube. Note that if the length of the scan chains do not evenly divide into the scan windows, the scan vectors can be augmented with a sufficient number of X's so that the length of each scan vector is a multiple of the scan window size (during test application time the extra bits corresponding to the X's will simply be shifted off the end of the scan chain). The advantage of having a larger scan window size is that the ratio of the total number of scan cells in the scan window versus the $s_{\max}$ of the scan window is generally much larger (it cannot be smaller), thus permitting a greater compression ratio (i.e., the number of scan bits that can be generated from the same size seed is greater).

6. EXPERIMENTAL RESULTS

Experiments were performed on the largest ISCAS 89 benchmark circuits [Brglez et al. 1989]. For each circuit, 10,000 pseudo-random patterns were applied using the LFSR to detect the easy faults. ATPG was performed to generate test cubes for the remaining faults. Partial reseeding was then used to encode the set of test cubes. The results are shown in Table I. The number of scan elements is shown for each circuit followed by the number of test cubes after merging. Compatible test cubes were merged (using static compaction), as described in Hellebrand et al. [1995a], in a way that did not increase $s_{\max}$. The total number of specified bits for the set of test cubes is shown in Table I, followed by the different scan window sizes used for each circuit

Table I. Results for Partial Reseeding After Pseudo-Random Sequence of 10,000 Patterns

Circuit		Num	Num	Scan				Test Data		
Name	Scan Elements	Test Cubes	Specified Bits	Window Size	S_{max}	LFSR size	Bits per Vector (n)	Test Storage	Encoding Efficiency	Compression Ratio
s5378	214	30	493	214	18	38	16	502	.982	12.8
				107	10	30	8	502	.982	12.8
				54	6	26	4	502	.982	12.8
				27	4	24	2	502	.982	12.8
s9234	247	138	4674	247	61	81	36	5013	.932	6.8
				124	31	51	18	5001	.935	6.8
				62	16	36	9	4995	.936	6.8
				31	9	29	5	5544	.843	6.2
s13207	700	157	2824	700	24	44	19	3008	.938	36.5
				350	17	37	10	3167	.892	34.7
				175	11	31	5	3166	.892	34.7
				88	6	26	3	3791	.745	28.9
s15850	611	167	5092	611	38	58	31	5204	.978	19.6
				306	22	42	16	5370	.948	19.0
				153	13	33	8	5369	.948	19.0
				77	9	29	4	5369	.948	19.0
s38417	1664	340	23984	1664	85	105	72	24513	.978	23.1
				832	51	71	36	24515	.978	23.1
				416	27	47	18	24509	.979	23.1
				208	15	35	9	24506	.979	23.1
s38584	1464	62	2848	1464	55	75	47	2942	.968	30.9
				732	30	50	24	3002	.949	30.2
				366	17	37	12	3001	.949	30.2
				183	9	29	6	2999	.950	30.3

(the first row for each circuit corresponds to having a single scan window for the entire test cube). For each of these different scan window sizes, s_{max} (the maximum number of specified bits in a scan window) is shown, followed by the size of the LFSR that is used. The LFSR size was chosen to be $s_{max} + 20$. A binary search (as described in Section 4) was used to find the lowest value for n (the number of bits used per test vector) for which a solution to the linear equations could be found. The test storage requirements (i.e., the number of encoded bits that would have to be stored on the tester) is shown followed by the encoding efficiency. The encoding efficiency is the ratio of the number of specified bits in the set of test cubes to the test storage requirements. As can be seen, the encoding efficiency for partial reseeding is very high. The last column shows the compression ratio that is achieved with partial reseeding which is the ratio of the test storage requirements for the unencoded test vectors (i.e., simply storing the test vectors themselves on the tester) compared with the test storage requirements using partial reseeding. As can be seen, partial reseeding generally provides an order of magnitude reduction in test storage requirements.

Figure 8 shows how the number of bits used per test vector varies with the size of the LFSR that is used. It can be seen that even if the size of the LFSR is slightly less than $s_{max} + 20$, it is possible to find a solution to the linear equations. Based on the graph, the smallest LFSR size corresponding to the lowest value of n can be used to minimize the hardware overhead as well as the tester storage requirements.

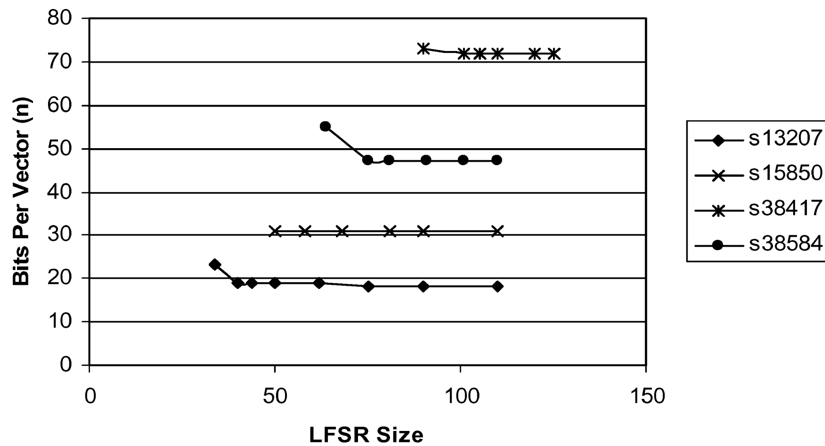

Fig. 8. Variation of bits per vector (n) with LFSR size.

Table II. Comparison of Reseeding Schemes for Same Test Set

	Standard LFSR Reseeding [Könemann 1991]			Test Cube Concatenation [Hellebrand et al. 1995a]			Variable-Length Seeds [Rajski et al. 1998a]			Proposed Partial Reseeding		
Circuit Name	LFSR size	Total Bits	Eff.	LFSR size	Total Bits	Eff.	LFSR size	Total Bits	Eff.	LFSR size	Total Bits	Eff.
s5378	38	1140	.432	18	570	.865	18	658	.749	38	502	.982
s9234	81	11178	.418	61	5332	.877	61	5364	.871	81	5013	.932
s13207	44	6908	.409	24	3925	.719	24	3609	.782	44	3008	.938
s15850	58	9686	.526	38	6513	.782	38	5927	.859	58	5204	.978
s38417	105	35700	.672	85	29240	.820	85	25684	.934	105	24513	.978
s38584	75	4650	.612	55	3472	.820	55	3158	.902	75	2942	.968

Table II shows a comparison of partial reseeding with previous reseeding schemes (all of which are based on full static reseeding). Each of the reseeding schemes was used to encode the set of test cubes in Table I. The exact same set of test cubes is encoded in each case to provide an "apples to apples" comparison. For each reseeding scheme, three things are shown: the size of the LFSR, the total number of encoded bits (i.e., the test storage requirement), and the encoding efficiency. For the test cube concatenation [Hellebrand et al. 1995a] and the variable-length seeds [Rajski et al. 1998a] schemes, an MP-LFSR with 16 polynomials was used, so the LFSR size is reduced. As can be seen, partial reseeding clearly provides the highest encoding efficiency. In terms of hardware overhead and control complexity, standard LFSR reseeding is the simplest scheme. Partial reseeding requires an XOR gate in the feedback tap that is controlled by the tester. The test cube concatenation scheme [Hellebrand et al. 1995a] requires an MP-LFSR with a control mechanism to change the feedback polynomial based on the "next-bit" (however, the LFSR length is reduced). The variable-length seeds scheme [Rajski et al. 1998a] requires a control mechanism to change the size of the LFSR based on the "next-bit" (some of the stages of the LFSR can be configured from the scan chains themselves). It should be

Table III. Comparison of Best Published Results for Reseeding Schemes (Test Sets are Different)

Circuit Name	[Hellebrand et al. 1995a] Test Groups	LFSR size	Total bits	[Rajski et al. 1998a] Num. Vectors	LFSR size	Total bits	Proposed Partial Reseeding Num. Vectors	LFSR size	Bits/Vect (n)	Total bits
s5378	24	27	726	NA	NA	NA	30	38	16	502
s9234	103	61	6923	104	96	4720	138	81	36	5013
s13207	138	24	3570	176	192	5784	157	44	19	3008
s15850	134	46	6528	56	256	6269	167	58	31	5204
s38417	259	91	24283	78	480	16797	340	105	72	24513
s38584	46	70	3406	52	224	3901	62	75	47	2942

noted that the encoding efficiency of the test cube concatenation scheme could be improved by using the special ATPG procedure described in Hellebrand et al. [1995b] to better select the test cubes, and the encoding efficiency of the variable-length seeds scheme could be improved by using a larger LFSR (constructed from the scan chains themselves) which would enable more test cube merging.

In Table III, the results for partial LFSR reseeding are compared with the best previously published results. In this case, the test sets are all different, so the optimality of the ATPG and compaction procedures used to obtain the test sets strongly affects the results. In Hellebrand et al. [1995b], a special ATPG procedure was used to find a set of test cubes that optimizes the effectiveness of test cube concatenation. In Rajski et al. [1998a], the scan chains were configured into extra large LFSRs to enable more test cube merging to be used. For each reseeding scheme, three things are shown: the number of vectors (or test groups in the case of Hellebrand et al. [1995b]), size of the LFSR, and the total number of encoded bits (i.e., the test storage requirement).

As can be seen, even though the set of test cubes for partial reseeding are not optimized, the results still compare favorably. It is very likely that if a better ATPG procedure was used to obtain the test cubes for partial reseeding, the results could be improved considerably. For the circuit, *s*38417, the variable-length seeds scheme [Rajski et al. 1998a] clearly outperformed the other schemes because of its inherent ability to configure very large LFSRs out of the scan chains in the CUT. If it were possible to configure a larger LFSR for partial seeding (perhaps from some idle scan chains that are not part of the CUT), then the results for partial reseeding could also be improved.

In Table IV, results are shown for partitioning the set of test cubes. For each circuit, the set of test cubes was partitioned into two equal subsets. The least specified test cubes were placed in partition 1, and the most specified test cubes were placed in partition 2. The system of linear equations for each partition was solved independently. The value of n and the test storage requirements are shown for each partition followed by the total test storage requirements for the two partitions combined. As can be seen, in some cases the total test storage requirements were slightly better with partitioning, and in some cases they are slightly worse. Overall, partitioning did not make too much difference in the results.

Table IV. Results for Partial Reseeding with Partitioning of the Test Set

Circuit		No Partitioning			Partitioning				
				Total	Partition 1		Partition 2		Total
	Num	LFSR	Bits/Vect.	Test	Bits/Vect.	Test	Bits/Vect.	Test	Test
Name	Vectors	Size	(n)	Storage	(n)	Storage	(n)	Storage	Storage
s5378	30	38	16	502	15	248	16	262	510
s9234	138	81	36	5013	23	1645	45	3141	4786
s13207	157	44	19	3008	15	1199	22	1760	2959
s15850	167	58	31	5204	27	2272	35	2963	5235
s38417	340	105	72	24513	65	11090	78	13287	24377
s38584	62	75	47	2942	42	1335	52	1635	2970

7. CONCLUSIONS

Partial dynamic LFSR reseeding is an attractive approach for compressing test data. It offers the following features:

- Better encoding efficiency than full static reseeding;
- Encoding efficiency not proportional to $s_{\max}$;
- Encoding efficiency improves as the size of the LFSR is increased;
- Very simple hardware implementation and control complexity;
- Very little additional hardware required beyond what is needed for STUMPS [Bardell and McAnney 1982];
- Fewer clock cycles are required to generate each test vector than with static LFSR reseeding;
- Easy to tradeoff computation time and optimality of encoding;

The drawback of partial dynamic LFSR reseeding compared with full static LFSR reseeding is that the computation time for solving the linear equations is longer. However, the computation time can be kept manageable by partitioning the set of test cubes. Full static LFSR reseeding is actually a special case of partial dynamic LFSR reseeding where the partition size is a single test cube.

Partial dynamic LFSR reseeding can be used in conjunction with pseudorandom BIST to form a mixed-mode testing approach. Pseudorandom patterns can be used to detect the random pattern testable faults, and partial LFSR reseeding can be used to generate test cubes that detect the random pattern resistant faults. This approach avoids the need for test points. The encoded test data for the partial LFSR reseeding can be either stored on the tester or stored on the chip in a ROM.

Partial dynamic LFSR reseeding can also be used in conjunction with external testing to reduce the test data storage and bandwidth requirements for the tester. The test data on the tester can be stored in compressed form and then decompressed using partial LFSR reseeding. Partial LFSR reseeding can reduce tester storage requirements by an order of magnitude or more.

REFERENCES

Bardell, P. H. and McAnney, W. H.. 1982. Self-testing of multichip logic modules. *Proceedings of International Test Conference*, pp. 200–204.

Barnhart, C., Brunkhorst, V., Distler, F., Farnsworth, O., Keller, B., and Koenemann, B. 2001. OPMISR: The foundation for compressed ATPG vectors. In *Proceedings of the International Test Conference*, pp. 748–757.

Bayraktaroglu, I. and Ogailoglu, A. 2001. Test volume and application time reduction through scan chain concealment. In *Proceedings of the Design Automation Conference*, pp. 151–155.

Brglez, F., Bryan, D., and Kozminski, K. 1989. Combinational profiles of sequential benchmark circuits. In *Proceedings of the International Symposium on Circuits and Systems*, pp. 1929–1934.

Chen, C. L. 1986. Linear dependencies in linear feedback shift registers. *IEEE Trans. Comput. C-35*, 12 (Dec). pp. 1086–1088.

Hellebrand, S., Tarnick, S., Rajski, J., and Courtois, B. 1992. Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers. In *Proceedings of the International Test Conference*, pp. 120–129.

Hellebrand, S., Rajski, J., Tarnick, S., Venkataraman S., and Courtois, B. 1995a. Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers. *IEEE Trans. Comput.*, *44*, 2 (Feb.), pp. 223–233.

Hellebrand, S., Reeb, B., Tarnick, S., and Wunderlich, H.-J. 1995b. Pattern generation for a deterministic BIST scheme. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. pp. 88–94.

International Technology Roadmap for Semiconductors (ITRS), 2001 Edition, Test and Test Equipment Section.

Kay, D. and Mourad, S. 2000. Controllable LFSR for BIST. In *Proceedings of the Instrumentation and Measurement Technology Conference*, Vol. 1, pp. 223–228.

Khoche, A. and Rivoir, J. 2000. I/O bandwidth bottleneck for test: Is it real?. In *Proceedings of the International Workshop on Test Resource Partitioning*.

Könemann, B. 1991. LFSR-coded test patterns for scan designs. In *Proceedings of the European Test Conference*, pp. 237–242.

Könemann, B. 2000. Logic DFT and test resource partitioning for 100M gate ASICs. In *Proceedings of the International Workshop on Test Resource Partitioning*.

Koenemann, B., Barnhart, C., Keller, B., Snethen, T., Farnsworth, O., and Wheater, D. 2001. A SmartBIST variant with guaranteed encoding. In *Proceedings of the Asian Test Symposium*, pp. 325–330.

Krishna, C. V., Jas, A., and Touba, N. A. 2001. Test vector encoding using partial LFSR reseeding. In *Proceedings of the International Test Conference*, pp. 885–893.

Krishna, C. V. and Touba, N. A. 2002. Reducing test data volume using LFSR reseeding with seed compression. In *Proceedings of the International Test Conference*, pp. 321–330.

Rajski, J., Tyszer, J., and Zacharia, N. 1998a. Test data decompression for multiple scan designs with boundary scan. *IEEE Trans. Comput.*, *47*, 11 (Nov.), pp. 1188–1200.

Rajski, J., Tamarapalli, N., and Tyszer, J. 1998b. Automated synthesis of large phase shifters for built-in self-test. In *Proceedings of the International Test Conferenc*, pp. 1047–1056.

Rajski, J., Tyszer, J., Kassab, M., Mukherjee, N., Thompson, R., Tsai, K.-H., Hertwig, A., Tamarapalli, N., Mrugalski, G., Eider, G., and Qian, J. 2002. Embedded deterministic test for low-cost manufacturing test. In *Proceedings of the International Test Conference*, pp. 301–310.

Venkataramann, S., Rajski, J., Hellebrand, S., and Tarnick, S. 1993. An efficient BIST scheme based on reseeding of multiple polynomial linear feedback shift registers. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 572–577.

Zacharia, N., Rajski, J., and Tyszer, J. 1995. Decompression of test data using variable-length seed LFSRs. In *Proceedings of the VLSI Test Symposium*, pp. 426–433.

Zacharia, N., Rajski, J., Tyszer, J., and Waicukauski, J. 1996. Two dimensional test data decompressor for multiple scan designs. In *Proceedings of the International Test Conference*, pp. 186–194.

Received March 2003; revised September 2003 and June 2004; accepted June 2004