Reducing Test Data Volume Using LFSR Reseeding with Seed Compression

C.V. Krishna and Nur A. Touba

Computer Engineering Research Center
Department of Electrical and Computer Engineering
University of Texas, Austin, TX 78712-1084
E-mail: {krishna, touba}@ece.utexas.edu

Abstract

A new lossless test vector compression scheme is presented which combines linear feedback shift register (LFSR) reseeding and statistical coding in a powerful way. Test vectors can be encoded as LFSR seeds by solving a system of linear equations. The solution space of the linear equations can be quite large. The proposed method takes advantage of this large solution space to find seeds that can be efficiently encoded using a statistical code. Two architectures for implementing LFSR reseeding with seed compression are described. One configures the scan cells themselves to perform the LFSR functionality while the other uses a new idea of "scan windows" to allow the use of a small separate LFSR whose size is independent of the number of scan cells. The proposed scheme can be used either for applying a fully deterministic test set or for mixed-mode built-in self-test (BIST), and it can be used in conjunction with other variations of LFSR reseeding that have been previously proposed to further improve the encoding efficiency.

1. Introduction

Complex system-on-chip designs are creating serious challenges for external ATE (automatic test equipment). As the amount of logic placed on a single chip continues to increase, both the test data storage requirements on the tester and the test data bandwidth requirements between the tester and chip are growing rapidly [Khoche 00]. There is a need for test resource partitioning in which some hardware is added on the chip to ease the burden on the external tester.

Compressing the output response is relatively easy because lossy compression techniques can be employed (e.g., using a multiple input signature register). However, compressing test vectors is much more difficult because lossless compression techniques must be used. Recently, as reducing test vector volume has become such an important problem, a lot of research has been done on lossless compression techniques for test vectors. These techniques include using run-length codes [Jas 98], statistical codes [Jas 99], Golomb codes [Chandra 00], frequency directed codes [Chandra 01], parallel/serial scan chains (Illinois Scan Architecture) [Hamzaoglu 99], virtual scan chains [Jas 00], combinational linear decompressions [Bayraktaroglu 01], [Reddy 02], and mixing pseudo-random/ATPG bits [Khoche 02]. These techniques involve storing the test vectors in a compressed form on the tester, and then transferring them to the chip where they are decompressed using on-chip hardware. Another class of techniques involves reducing test vector volume by encoding test vectors in a longer BIST test sequence. These techniques include using hybrid patterns [Das 00], folding counters [Hellebrand 00], two-dimensional compression [Liang 01], and RESPIN [Dorsch 01].

In this paper, a new lossless test vector compression scheme is proposed which provides very high encoding efficiency. The proposed scheme combines linear feedback shift register (LFSR) reseeding and statistical coding in a powerful way. 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. Given a set of deterministic test cubes (test vectors in which bits unassigned by ATPG are left as don't cares), 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 91]. Since the seeds are much smaller than the full test vectors, the amount of test data can be reduced by an order of magnitude or more. However, the linear dependencies in the LFSR and the variance in the number of specified bits in each test cube limit the encoding efficiency of LFSR reseeding. Several techniques have been proposed for improving the encoding efficiency of LFSR reseeding [Hellebrand 92, 95a, 95b], [Venkataraman 93], [Zacharia 95, 96], [Rajski 98a], [Krishna 01]. In this paper, a new approach is proposed for improving the encoding efficiency of LFSR reseeding which involves compressing the seeds using a

0-7803-7542-4/02 $17.00

statistical code. The solution space for the system of linear equations corresponding to each LFSR seed is generally quite large, especially for test cubes with fewer specified bits. A procedure is described for exploiting the large linear solution space for each seed to find a set of seeds that can be efficiently encoded using a statistical code. The combination of test vector compression using LFSR reseeding and then further compression of the seeds using statistical encoding provides very high encoding efficiency. Compressing LFSR seeds provides more degrees of freedom for improving the encoding efficiency than compressing the test cubes directly because the large solution space for the linear equations allows a lot of flexibility to organize and align the bits in the seeds so that they can be efficiently encoded with compression codes.

Two architectures for implementing the proposed test vector compression scheme are presented. One configures part of the scan chains themselves to perform the LFSR functionality thus reducing hardware requirements (this is well suited for compression/decompression of deterministic test sets), and the other uses a new idea of "scan windows" to have a small separate LFSR whose size is independent of the number of scan cells. Having a separate LFSR requires more hardware area, but it allows mixed-mode built-in self-test (BIST) to be performed hence providing even greater reduction of test data storage requirements, and it is also useful for intellectual property cores and other designs where it is not possible or not desirable to configure the scan chains to perform the LFSR functionality.

2. Related Work

The encoding efficiency for the basic LFSR reseeding approach described in [Könemann 91] 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 86], [Könemann 91], 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 basic LFSR reseeding approach described in [Könemann 91] requires $s_{max}+20$ bits to encode each test cube regardless of the number of specified bits in the test cube. Several techniques have been proposed to improve the encoding efficiency including using multiple-polynomial LFSRs [Hellebrand 92], [Venkataraman 93], concatenating test cubes [Hellebrand 95a], using variable-length seeds [Zacharia 95, 96], [Rajski 98a], using partial dynamic LFSR reseeding [Krishna 01], and using two-dimensional compression combining LFSR reseeding and folding counters [Liang 01]. The method proposed in this paper involves compressing the seeds that are used in LFSR reseeding. It can be used in conjunction with any of the LFSR reseeding schemes mentioned above to compress the seeds and hence provide even greater encoding efficiency. It is very effective even for the basic LFSR reseeding scheme where each test cube is encoded with $s_{max}+20$ bits because the large solution space for each seed enables it to be highly compressed using the proposed procedure. For sake of simplicity, the proposed seed compression methodology will be described here for the basic LFSR reseeding scheme, but note that it can also be used with any of the other more efficient LFSR reseeding schemes to provide even better results.

Statistical encoding can be applied directly on the test cubes themselves as was described in [Jas 99]. However, using statistical encoding on LFSR seeds instead of the test cubes provides two advantages. The first is that the number of blocks of data that need to be encoded is dramatically reduced because the LFSR seeds are much smaller than the test cubes (typically an order of magnitude smaller). The second is that the specified bits in the test cubes cannot be easily changed (it requires re-running the APTG with special constraints which is time consuming), whereas the specified bits in the LFSR seeds can be easily altered by simply choosing a different seed in the linear solution space. The degrees of freedom in choosing the LFSR seed allows a much more efficient encoding compared with encoding the test cubes directly. This is very apparent in the experimental results shown in Sec. 6.

3. Architectures for LFSR Reseeding with Seed Compression

Two architectures for implementing the proposed test vector compression scheme are presented here. One has the LFSR integrated as part of the scan chains while the other uses a separate small LFSR. Each architecture has certain advantages and is better suited for particular applications.

3.1 Integrated LFSR

LFSR reseeding involves loading one seed at a time into an LFSR and running the LFSR in autonomous mode to fill the scan chains. For encoding a fully deterministic test set that covers all the faults (i.e., no pseudo-random patterns are used), each test cube is generated by decompressing an LFSR seed. Since there is no need to save the state of the LFSR after it has decompressed a seed to fill the scan chains, its contents can be overwritten by the output response during the capture cycle. So in this case, the LFSR can be implemented by configuring part of the scan chains to perform the LFSR functionality, as illustrated in Fig. 1. By using the scan cells themselves to form the LFSR, there is no need for a separate LFSR and hence the hardware requirements are reduced.

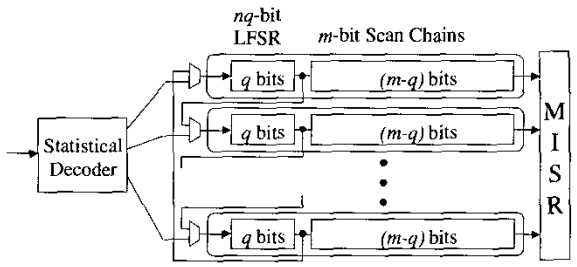

Figure 1. Architecture with Integrated LFSR

The same number of scan cells, q, from each scan chain are used to form the LFSR. If there are n scan chains, then the total length of the LFSR is nq. The value of q is selected so that nq is as close to s_{max}+20 as possible (equal to or slightly greater) where s_{max} is the maximum number of specified bits in any test cube. This is done so that it is possible to solve the linear equations for all the test cubes with very high probability [Könemann 91].

The scan architecture has 3 modes: scan mode, system mode, and decompression mode. The scan mode and system mode are the same as for all scan chains. In the decompression mode, the first q scan cells of each m-bit long scan chain are configured together as one long LFSR while the remaining m-q scan cells in each scan chain act as they normally do in scan mode (they are essentially fed from tap points in the LFSR). Note that there is no need for a phase shifter as the tap points are generally spaced apart sufficiently to eliminate correlations (assuming a modular LFSR structure [Bardell 87]). The LFSR is run in autonomous mode for m-q clock cycles to fill up the last m-q scan cells in each scan chain. The linear equations used to solve for the seed are formed so that the final contents of all the scan cells (including those that are part of the LFSR) will contain the test cube that needs to be applied to the circuit-under-test (CUT) at that point. The scan chain can then be put into system mode to apply the capture cycle so that the output response of the CUT is loaded into the scan chains.

The process of loading a seed into the LFSR is done as follows. The tester sends data to the statistical decoder which decodes it and shifts it into the first q bits of the scan chains (while the scan architecture is in scan mode). The decoding process will be described in more detail in Sec. 4. As the decoder shifts the first q bits into the scan chains, q bits of the output response get shifted out into the MISR. At this point, the seed has been loaded, and the scan architecture is put into decompression mode to run the LFSR for m-q cycles. During those cycles, the remainder of the output response gets shifted into the MISR as the next test vector is generated.

3.2 Separate LFSR

In some cases it may be preferable to have a separate LFSR. For some designs (e.g., intellectual property cores, hard macrocells, legacy designs, etc.), it may not be possible or desirable to make any changes to the existing scan architecture. Another case where a separate LFSR may be preferable is if pseudo-random patterns are going to be used. An integrated LFSR has its state overwritten during the capture cycle, so it cannot be used as a proper pseudo-random pattern source (unless a circular BIST [Krasniewski 87], [Stroud 88], methodology is adopted). By having a separate LFSR, a mixed-mode BIST strategy can be employed in which pseudo-random patterns are first applied to detect the random pattern testable faults, and then LFSR reseeding is used to generate deterministic patterns to detect the random pattern resistant faults. Mixed-mode BIST can provide an even greater reduction in tester storage requirements.

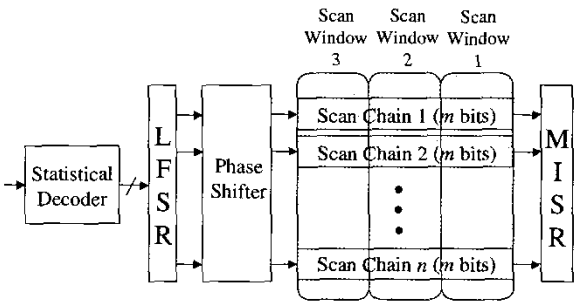

Figure 2. Architecture with Separate LFSR

One of the problems for all LFSR reseeding approaches where a separate LFSR is used is that the size of the LFSR scales with s_{max} and thus can become problematic for large industrial circuits. Here we propose a solution to this problem (a solution that can be applied

for any LFSR reseeding scheme) which involves the use of what we will term "scan windows." 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 Fig. 2. In Fig. 2, the scan chains are divided into 3 scan windows where each scan window will have *n(m/3)* scan cells in it. So instead of generating an entire test cube with one LFSR seed, multiple seeds are used to generate a single test cube. By so doing, 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 of 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 any size provided an appropriate linear phase shifter is used [Rajski 98b].

LFSR reseeding with *w*-bit wide scan windows is performed by simply loading each seed into the LFSR and running it in autonomous mode for *w* cycles. After *m/w* scan windows have been loaded, then the scan chains contain a complete test cube. The system clock is then applied and the process continues. Note that if the length of the scan chains do not divide evenly into 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, the extra bits corresponding to the X's will simply be shifted off the end of the scan chain).

It is interesting to note that in the degenerate case where the scan window size is equal to 1, then the seed is essentially decompressed through a linear combinational network and loaded immediately into one bit-slice of the scan chains. This is effectively equivalent to what is proposed in [Bayraktaroglu 01] where they design a linear combinational network to expand a smaller input vector (which is essentially equivalent to a "seed") to fill one bit-slice of a larger number of scan chains. 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). Moreover, in the method proposed here, we are providing even greater compression than simply doing a linear expansion of a seed because we are also statistically encoding the seed as well in a clever way. For a particular scan window, the number of specified bits may be much less than s_{max} and the method proposed here is able to take advantage of that to greatly improve the amount of compression. This is done by using the resulting degrees of freedom in the solution space of the linear equations to find seeds with short statistical encodings as will be described in Sec. 5.

4. Statistical Coding

In statistical coding, variable length codewords are used to represent fixed-length blocks of data. For example, if a data set is divided into 4-bit blocks, then there are 2^4 or 16 unique 4-bit blocks. Each of the 16 possible 4-bit blocks can be represented by a binary *codeword*. The size of each codeword is variable (it need not be 4 bits). The idea is to make the codewords that occur most frequently have a smaller number of bits, and those that occur least frequently to have a larger number of bits. This minimizes the average length of a codeword. The goal is to obtain a coded representation of the original data set that has the smallest number of bits.

A Huffman code [Huffman 52] is an optimal statistical code that is proven to provide the shortest average codeword length among all uniquely decodable variable length codes. An important property of Huffman codes is that they are prefix-free. No codeword is a prefix of another codeword. This greatly simplifies the decoding process.

The amount of compression that can be achieved with statistical coding depends on how skewed the frequency of occurrence is for the different codewords. If all of the codewords occur with equal frequency, then no compression can be achieved.

The idea proposed here is to encode the LFSR seeds using a statistical code. For an *r*-bit LFSR, the *r*-bit seed is partitioned into equal size blocks (if *r* does not divide out evenly, the length of the seed can be slightly extended by appending don't cares to make all the blocks equal size). The frequency of occurrence of the codewords can be skewed by intelligent selection of the seeds from the solution space of the linear equations as will be described in Sec. 5. Given the frequency of each codeword, an optimal Huffman code can be constructed [Huffman 52]. A decoder for the Huffman code can be implemented with a finite state machine (FSM) having 2^b-*1* states for a *b*-bit block size. If the block size is chosen to be small (e.g. 4 or 5 bits blocks), then the Huffman decoder requires very little overhead. Alternatively, larger block sizes can be used with a selective coding approach to simplify the decoder (see [Jas 99] for details).

The Huffman decoder generates data one block at a time. For a separate LFSR, the flip-flops in the LFSR can have a parallel load capability and be loaded one block at a time (as illustrated in Fig. 3). The assignment of LFSR flip-flops to blocks can be done in any manner (the blocks need not be contiguous). This degree of freedom can be used to improve the encoding efficiency as will be

described in Sec. 5. For an integrated LFSR, the scan cells that make up the LFSR must be loaded by shifting the scan chains (they cannot have parallel load since that would overwrite the output response that they contain). For b-bit blocks, each block is loaded by shifting data into b scan chains in parallel. This is illustrated in Fig. 4 where there are 4-bit blocks which are loaded by shifting into 4 scan chains simultaneously. Note that if there are n scan chains, n/b blocks must first be decoded by the decoder, then all n/b blocks are shifted into one bit-slice of the scan chains during one scan cycle. So for the example in Fig. 4, the decoder would first decode two 4-bit blocks, and then shift them into one bit-slice of the 8 scan chains during one scan cycle. For the integrated LFSR, the assignment of LFSR flip-flops to blocks is much more constrained than for the separate LFSR. This must be taken into account when selecting seeds to skew the codeword frequency as will be described in Sec. 5.

For more details on implementing the statistical decoder and its interaction with the tester, the interested reader can see [Jas 99].

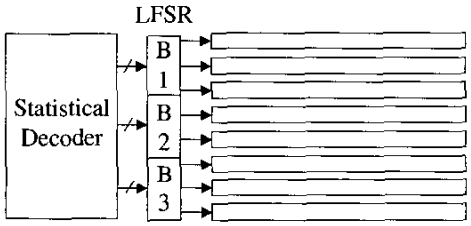

Figure 3. Division of Separate LFSR into Blocks (*B1-B3*)

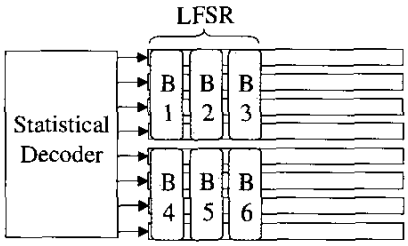

Figure 4. Division of Integrated LFSR into Blocks (*B1-B6*)

5. Using Linear Solution Space to Statistically Encode Seeds

Given the scan architecture, LFSR feedback polynomial, and phase shifter (if any), the LFSR can be symbolically simulated to determine the set of linear equations for each scan cell. This is done by having each bit in the seed of the LFSR be represented by a variable, and then the operation of the LFSR and phase shifter are symbolically simulated. Each time two bits are exclusive-ORed together, they form a linear combination of the corresponding variables. When the scan chains have been fully loaded, the final linear combination of variables that exists for each scan cell forms the linear equation for that scan cell. The symbolic simulation need only be done once.

$$\begin{bmatrix} 1&1&0&0&1&0&1 \\ 1&0&1&1&1&1&0 \\ 0&0&1&0&0&1&1 \\ 0&1&0&1&1&1&1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Figure 5. Example of System of Linear Equations

$$\begin{array}{c} \text{Pivots} \quad \text{Free Variables} \\ \left[\begin{array}{ccccccc|c} 1&0&0&0&1&1&1&1 \\ 0&1&0&0&0&1&0&0 \\ 0&0&1&0&0&1&1&1 \\ 0&0&0&1&0&1&0&0 \end{array}\right] \end{array}$$

Figure 6. Gauss-Jordan Reduction of Example in Fig. 5

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = x_5 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + x_6 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + x_7 \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Figure 7. Solution Space for Example in Fig. 5

For each test cube in the test set, a system of linear equations is formed in which there is one linear equation corresponding to each specified bit in the test cube. The resulting system of linear equations has the form $Ax=y$ where y is a column vector of the specified bits in the test cube (which can have up to s_{max} rows), and the solution x is the seed (which will have at least s_{max}+20 rows). In general there will be many solutions for x (i.e., many seeds that will produce the test cube). An example of a system of linear equations for a test cube is shown in Fig. 5, there is one row (i.e., equation) for each specified bit in the test cube. Gauss-Jordan Elimination [Cullen 97] can be used to transform a set of columns into an identity matrix (these columns are called the *pivots*) while the remaining columns are *free-variables*. Figure 6 shows the resulting matrix after Gauss-Jordan Elimination has been performed on the example. Without loss of generality, suppose the first p columns are pivots and the remaining f columns are free-variables, then the set of solutions for x can be represented as any combination of values for the free-variables (x_{p+1} to x_{p+f}), and then the pivots (x_1 to x_p)

are calculated as the linear sum of the column vectors for the free-variables with value 1. The set of solutions for the example is shown in Fig. 7.

Our goal is to choose a seed solution for each test cube such that the complete set of seeds for all the test cubes can be efficiently encoded using a statistical code. To increase the encoding efficiency, the frequency of occurrence for the codewords should be as skewed as possible. To accomplish this, we use a two phase process first doing global processing across all seeds to select the statistical encoding, and then optionally doing local processing one seed at a time to fine tune the result.

5.1 Global Processing

To make the frequency of occurrence for the codewords as skewed as possible, the same bit positions in the seeds should have the same value as often as possible. Notice that in using Gauss-Jordan Elimination to find a solution for a particular seed, the free-variables can be assigned any value independently of each other (whereas the pivots are dependent on the values assigned to the free-variables). The key idea in our strategy is to do Gauss-Jordan Elimination when solving for each seed and to process the columns in the same order every time (e.g., first try to make x_1 a pivot, then try to make x_2 a pivot, etc.). Each seed has its own system of linear equations, if we order the columns in the A matrix for each seed in the same way, and then process the columns from left to right when performing the steps of Gauss-Jordan Elimination, then we have observed that the probability of a column ending up as a pivot decreases significantly from left to right. Some columns in A may be linearly dependent and become all 0's during Gauss-Jordan Elimination thus moving the final pivot toward the right, however, if the LFSR has a primitive feedback polynomial and an appropriate phase shifter is used, then probability of linear dependence in the columns is fairly low. Thus for a test cube with s specified bits, the final pivot will tend to be fairly close to the s-th column. Since we can set the value of the free-variables to anything, we can begin by setting them all to value 0 (the value of the pivots are dependent on the value of the free-variables, so they will take on a particular set of values based on this assignment to the free-variables). Since the columns to the right are skewed towards free-variables which are all set to 0, the corresponding bit positions in the seeds will be skewed to 0 which is good for statistical coding.

An example of the tendency for the pivots to be towards the left is shown in Fig. 8. The graph shows the percentage of seeds that have a pivot in each column from left to right when solving the system of linear equations for the test cubes for circuit *s9234*. As can be seen, the pivots are very much skewed to the left, and thus the bits towards the right of the seed will be very skewed towards 0.

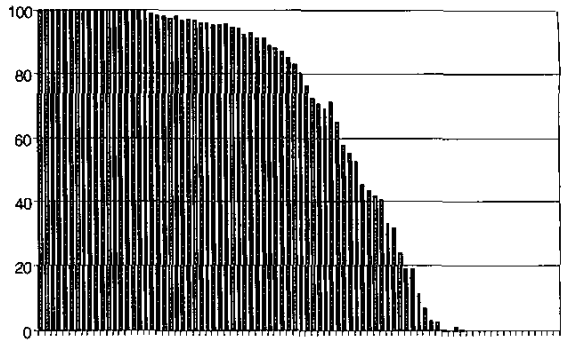

Figure 8. Percentage of Seeds Having Pivot in Each Column

For a separate LFSR, the assignment of LFSR flip-flops to blocks for statistical encoding can be done in any manner. Thus, we can compute the percentage of 0's in each bit position of the set of seeds and sort them. If there are k blocks in each seed, then the k bit positions in the set of seeds that are most skewed towards 0 can be assigned to the most significant bit position in each of the k blocks. The next k bit positions that are most skewed towards 0 can be assigned to the second most significant bit position in each of the k blocks, and so forth. Lastly, the k bit positions that are least skewed toward 0 (and therefore most skewed towards 1) are assigned to the least significant bit positions of the k blocks. At this point, each seed has been mapped to k blocks where the number of 0 and 1's in each bit position of the blocks have been skewed. This causes the frequency of occurrence of the codewords to be skewed, and thus allows the statistical coding to compress the data.

For an integrated LFSR, the assignment of LFSR flip-flops to blocks is significantly constrained (as was described in Sec. 4). However, we can compensate for this by using the degree of freedom in setting the ordering of the Gauss Jordan Elimination to still get nearly the same effect. In this case, we make the assignment of bit positions in the seeds to blocks before solving the linear equations. We can choose this assignment to simplify the process of loading the blocks from the decoder to the scan chains. Then based on where each bit position in the seeds occurs in the blocks, we can set the order in which the Gauss Jordan Elimination is done for all the seeds. If there are k blocks in each seed, then the k bit positions in the seeds that correspond to the most significant bit position in the blocks can be ordered on the far right side of the A matrix. Those corresponding to the second most significant bit position can be ordered just to the left of that, and so forth. Lastly, those corresponding to the least significant bit position are ordered on the far left side of

the A matrix. The reason for doing this is that when the Gauss Jordan Elimination proceeds from the left to the right, the columns towards the right will tend to have the most free-variables and thus tend to be most skewed. The most skewed columns will be aligned in the blocks to again cause the frequency of occurrence of the codewords to be skewed.

So far we have only discussed the case where all the free-variables are set to 0. It is also possible to consider solutions where some or all of the free-variables are set to 1 as well. Free-variables that correspond to the same bit position in the blocks should all be set to the same value (either 0 or 1) to ensure the best frequency skew. Within this constraint, however, other assignments to the free-variables can be explored for different bit positions in the blocks. Changing the assignment of the free-variables will change the values of the pivots, and may give a slightly better frequency skew. An optional step would be to try several different assignments of the free-variables for different bit positions of the blocks and keep the one that gives the best frequency skew in the blocks.

Once the final assignment of free-variables has been selected and the bit positions in the seeds have been assigned to blocks, then the frequency of each codeword can be calculated. Based on this frequency distribution, the optimal Huffman code can be constructed by forming a Huffman Tree [Huffman 52], or a selective statistical code can be formed [Jas 99].

5.2 Local Processing (Optional)

Given the statistical code that is to be used, the number of bits required to encode each block is known. An optional step is to do some local processing on one seed at a time to fine tune the result. The system of linear equations for each test cube is independent of all the other test cubes, so changing the assignment of the free-variable for one seed has only a local effect on the pivots of that seed. A different assignment of the free-variables for a seed (different from the global assignment which was done across all seeds) may result in a shorter encoding for the seed. For each seed, a branch and bound procedure can be used to try to find a better assignment of the free-variables. Changing the value of one free-variable will change the codeword for the block in which the free-variable resides (likely making it worse, i.e., making the encoding longer), however it may also change the value of several pivots thereby changing the encoding of the blocks in which they reside (possibly making them better, i.e., making the encoding shorter). The net effect may be to reduce the number of bits needed to encode the seed as a whole. A branch and bound procedure can be used to search for a better assignment of the free-variables. The bound is formed by the best encoding that has been seen so far. The search space is exponential in the number of free-variables in the worst-case, so it may be necessary to place some limit on the maximum number of iterations used for each seed. When all seeds have been processed, then the new frequency of each codeword is calculated and the final statistical code is computed.

6. Experimental Results

Experiments were performed on the largest ISCAS 89 benchmark circuits [Brglez 89]. For each circuit, ATPG was performed to generate test cubes for the non-redundant faults. The system of linear equations for each test cube was then processed with Gauss-Jordan Elimination always following the same ordering as described in Sec. 5. The two phase process of global and local processing were then done to obtain a set of seeds for all the test cubes that could be efficiently encoded using a statistical code. Once the seeds were obtained, they were encoded using a Huffman code, and the total test storage requirement was then computed. The results are presented in Table 1. The number of scan elements is shown for each circuit (it is assumed that their primary inputs are controlled by a scan chain) followed by the number of test cubes. Results are then shown for using an integrated LFSR architecture. The first column shows the size of the LFSR used for each circuit. The test storage requirement (i.e., the number of encoded bits that would have to be stored on the tester) is shown followed by the percentage compression obtained using this technique. The percentage compression shows the compression achieved by storing the encoded seeds on the tester as compared with the test storage requirements for the unencoded test vectors (i.e., simply storing the test vectors themselves on the tester). As can be seen, the compression obtained by using the integrated LFSR technique is very high. Results are then shown for the separate LFSR architecture for different scan window sizes followed by the size of the LFSR used in each case (the LFSR size varies because the maximum number of specified bits, s_{max}, for different scan window sizes varies. The test storage requirements for this technique are then shown, followed by the compression achieved. In comparing the results for using an integrated LFSR versus a separate LSFR, it can be seen that the integrated LFSR provides better results as is expected since it is using a larger LFSR and is generating the entire test cube with each seed. For the separate LFSR, as the scan window size is reduced, the size of the LFSR is also reduced since the s_{max} for the scan windows becomes smaller. However, as the scan window size is reduced, the encoding efficiency drops, so the amount of compression becomes less.

Table 1. Results for Integrated and Separate LFSR for Block Size of 4

Circuit			Integrated LFSR			Separate LFSR			
Name	Scan Elements	Num. Test Cubes	LFSR Size	Test Storage (bits)	Percent Compress	Scan Window Size	LFSR Size	Test Storage (bits)	Percent Compress
s5378	214	196	40	6,180	85	107	28	6,667	84
						54	20	7,306	83
						27	12	8,028	81
s9234	247	205	68	12,112	76	124	38	12,445	75
						62	24	13,059	74
						31	24	12,955	74
s13207	700	266	48	11,285	94	350	36	11,859	94
						175	20	12,079	94
						88	16	14,079	92
s15850	611	269	54	12,438	92	306	32	12,663	92
						153	24	14,439	91
						77	20	17,033	90
s38417	1664	376	104	34,767	94	832	60	36,430	94
						416	38	37,773	94
						208	28	42,360	93
s38584	1464	296	120	29,397	93	732	61	30,355	93
						366	38	31,971	93
						183	28	35,427	92

Table 2. Results for Integrated LFSR with Different Block Sizes

Circuit Name	Scan Elements	Num. Test Cubes	LFSR Size	Block Size	Decoder States	Test Storage (bits)	Percentage Compression
s5378	214	196	40	4	15	6,180	85
			40	6	63	6,030	86
			40	8	255	5,882	86
s9234	247	205	68	4	15	12,112	76
			68	6	63	11,868	77
			68	8	255	11,666	77
s13207	700	266	48	4	15	11,285	94
			48	6	63	11,018	94
			48	8	255	10,793	94
s15850	611	269	54	4	15	12,438	92
			54	6	63	12,105	93
			54	8	255	11,986	93
s38417	1664	376	104	4	15	34,767	94
			104	6	63	34,298	95
			104	8	255	33,850	95
s38584	1464	296	120	4	15	29,397	93
			120	6	63	28,833	93
			120	8	255	28,659	93

Table 3. Comparison of Test Data for Different Encoding Schemes

Circuit Name	MinTest [Hamzaoglu 98]		Illinois Scan Architecture [Hamzaoglu 99]		FDR Codes [Chandra 01]		Linear Decompressors [Bayraktaroglu 01]		Statistical Coding of Test Cubes [Jas 99]		Proposed Approach	
	Num. Vect.	Total Bits	Num. Vect.	Total Bits	Num. Vect.	Total Bits	Num. Vect.	Total Storage	Num. Vect.	Total Bits	Num. Vect.	Total Bits
s5378	97	20,758	160	14,572	111	12,346	NA	NA	196	15,417	196	6,180
s9234	106	26,182	238	27,111	159	22,152	NA	NA	205	19,912	205	12,112
s13207	233	163,100	273	109,772	236	30,880	251	24,096	266	52,741	266	11,285
s15850	96	58,656	178	32,758	126	26,000	170	16,320	269	49,163	269	12,438
s38417	68	113,152	337	96,269	99	93,466	296	63,936	376	172,216	376	34,767
s38584	110	161,040	239	96,056	136	77,812	182	34,944	296	128,046	296	29,397

Table 2 shows the results for the integrated LFSR technique using different block sizes for the Huffman code. Results are shown for block sizes of 4, 6, and 8. The number of decoder states for the different block sizes are shown, followed by the test storage requirements and the compression achieved in each case. While larger block sizes give relatively better compression, they also require a larger hardware overhead. So there is a tradeoff between the block size used and the amount of hardware overhead that is appropriate.

Table 3 shows a comparison between the proposed method using an integrated LFSR with a block size of 4 compared with previously published test vector compression schemes as well as with MINTEST [Hamzaoglu 98] which is an ATPG procedure that uses dynamic compaction. In this case, the test sets for the different schemes are not the same, so the optimality of the ATPG and compaction procedures used to obtain the test sets do impact the results. For the results shown for statistical coding of the test cubes, we used the methodology described in the [Jas 99] on the same set of test cubes that were used for the proposed method (i.e., the numbers are not being taken directly out of the [Jas 99] paper itself). This shows a direct comparison between encoding the test cubes themselves with a statistical code versus encoding LFSR seeds that produce the test cubes as is proposed here. As can be seen, much greater encoding efficiency can be obtain by encoding the seeds.

Note that better results could be obtained for the proposed procedure by using multiple-polynomial LFSRs [Hellebrand 95a] to reduce the length of the seeds. Other variations of LFSR reseeding could similarly be combined with the proposed approach for seed compression. Another way to reduce the tester storage requirements when using a separate LFSR would be to first apply pseudo-random patterns to detect the random pattern testable faults, and then use reseeding with seed compression for detecting the remaining faults (i.e., use a mixed-mode BIST approach).

7. Conclusions

Compressing LFSR seeds provides more degrees of freedom for improving the encoding efficiency than compressing the test cubes directly. Generally the specified bits in the test cubes and the order of the scan cells in the scan chains are difficult to change. However, for LFSR seeds, the large solution space for the linear equations allows a lot of flexibility to organize and align the bits in the seeds so that they can be efficiently encoded with compression codes.

The seed compression scheme described in this paper allows the test vector volume for the circuit-under-test to be greatly compressed. The additional hardware required for implementing this scheme depends on whether an integrated LFSR or separate LFSR is used. For an integrated LFSR, additional hardware is needed to reconfigure some of the scan cells to form an LFSR. For a separate LFSR, the size of the LFSR depends on how large of scan windows are used. If a mixed-mode BIST scheme is to be used, then the LFSR will exist already for pseudo-random pattern testing. The other source of additional hardware is the statistical decoder. The size of the decoder depends on what block size is used. If the block size is kept small, then the decoder will be just a small finite state machine.

The scheme described in this paper uses statistical codes for compressing the LSFR seeds. One area for further research would be to investigate the use of other codes that can efficiently exploit the large solution space for the linear equations.

Acknowledgements

This material is based on work supported in part by the National Science Foundation under Grant No. MIP-9702236 and in part by the Texas Advanced Technology Program under Grant No. 003658-0644-1999.

References

[Bardell 87] Bardell, P.H., and W.H. McAnney, and J. Savir, *Built-in Test for VLSI: Pseudorandom Techniques*, John Wiley & Sons, 1987.

[Bayraktaroglu 01] Bayraktaroglu, I., and A. Ogailoglu, "Test Volume and Application Time Reduction Through Scan Chain Concealment," *Proc. of Design Automation Conference*, pp. 151-155, 2001.

[Brglez 89] Brglez, F., D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. of International Symposium on Circuits and Systems*, pp. 1929-1934, 1989.

[Chandra 00] Chandra, A., and K. Chakrabarty, "Test Data Compression for System-on-a-Chip Using Golomb Codes," *Proc. of VLSI Test Symposium*, pp. 113-120, 2000.

[Chandra 01] Chandra, A., and K. Chakrabarty, "Frequency-Directed Run Length (FDR) Codes with Application to System-on-a-Chip Test Data Compression," *Proc. of VLSI Test Symposium*, pp. 42-47, 2001.

[Chen 86] Chen, C.L., "Linear Dependencies in Linear Feedback Shift Registers", *IEEE Transactions on Computers*, Vol. C-35, No. 12, pp. 1086-1088, Dec. 1986.

[Cullen 97] Cullen, C.G., *Linear Algebra with Applications*, Addison-Wesley, ISBN 0-673-99386-8, 1997.

[Das 00] Das, D., and N.A. Touba, "Reducing Test Data Volume Using External/LBIST Hybrid Test Patterns," *Proc. of International Test Conference*, pp. 115-122, 2000.

[Dorsch 01] Dorsch, R., and H.-J. Wunderlich, "Tailoring ATPG for Embedded Testing," *Proc. of International Test Conference*, pp. 530-537, 2001.

[Hamzaoglu 98] Hamzaoglu, I., and J. Patel, "Test Set Compaction Algorithms for Combinational Circuits," *Proc. of International Conference on Computer-Aided Design (ICCAD)*, pp. 283-289, 1998.

[Hamzaoglu 99] Hamzaoglu, I., and J.H. Patel, "Reducing Test Application Time for Full Scan Embedded Cores," *Proc. of Int. Symp. on Fault Tolerant Computing*, pp. 260-267, 1999.

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

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

[Hellebrand 95b] Hellebrand, S., B. Reeb, S. Tarnick, and H.-J. Wunderlich, "Pattern Generation for a Deterministic BIST Scheme," *Proc. of International Conference on Computer-Aided Design (ICCAD)*, pp. 88-94, 1995.

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

[Huffman 52] Huffman, D.A., "A Method for the Construction of Minimum Redundancy Codes," *Proc. of IRE*, Vol. 40, No. 9, pp. 1098-1101, Sep. 1952.

[Jas 98] Jas, A., and N.A. Touba, "Test Vector Decompression Via Cyclical Scan Chains and Its Application to Testing Core-Based Designs", *Proc. of IEEE International Test Conference*, pp. 458-464, 1998.

[Jas 99] Jas, A., J. Ghosh-Dastidar, and N.A. Touba, "Scan Vector Compression/Decompression Using Statistical Coding", *Proc. of IEEE VLSI Test Symposium*, pp. 114-120, 1999.

[Jas 00] Jas, A., and N.A. Touba, "Virtual Scan Chains: A Means for reducing Scan Length in Cores", *Proc. of IEEE VLSI Test Symposium*, pp. 73-78, 2000.

[Khoche 00] Khoche, A., and J. Rivoir, "I/O Bandwidth Bottleneck for Test: Is it Real?," *Proc. of International Workshop on Test Resource Partitioning*, 2000.

[Khoche 02] Khoche, A., E. Volkerink, J. Rivoir, and S. Mitra, "Test Vector Compression Using EDA-ATE Synergies," *Proc. of IEEE VLSI Test Symposium*, pp. 97-102, 2002.

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

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

[Könemann 00] Könemann, B., "Logic DFT and Test Resource Partitioning for 100M Gate ASICs," *Proc. of International Workshop on Test Resource Partitioning*, 2000.

[Krasniewski 89] Krasniewski, A., and S. Pilarski, "Circular Self-Test Path: A Low-Cost BIST Technique for VLSI Circuits," *IEEE Trans. on Computer-Aided Design*, Vol. 8, No. 1, pp. 46-55, Jan. 1989.

[Liang 91] Liang, H.-G., S. Hellebrand, and H.-J. Wunderlich, "Two-Dimensional Test Data Compression for Scan-Based Deterministic BIST," *Proc. of International Test Conference*, pp. 894-902, 2001.

[Rajski 98a] Rajski, J., J. Tyszer, and N. Zacharia, "Test Data Decompression for Multiple Scan Designs with Boundary Scan", *IEEE Transactions on Computers*, Vol. 47, No. 11, pp. 1188-1200, Nov. 1998.

[Rajski 98b] Rajski, J., N. Tamarapalli, and J. Tyszer, "Automated Synthesis of Large Phase Shifters for Built-In Self-Test", *Proc. of Int. Test Conf.*, pp. 1047-1056, 1998.

[Reddy 02] Reddy, S.M., K. Miyase, S. Kalihara, and I. Pomeranz, "On Test Data Volume Reduction for Multiple Scan Chain Design," *Proc. of IEEE VLSI Test Symposium*, pp. 103-108, 2002.

[Stroud 88] Stroud, C.E., "Automated BIST for Sequential Logic Synthesis," *IEEE Design & Test*, pp. 22-32, Dec. 1988.

[Venkataraman 93] Venkataramann, S., J. Rajski, S. Hellebrand, and S. Tarnick, "An Efficient BIST Scheme Based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers," *Proc. of International Conference on Computer-Aided Design (ICCAD)*, pp. 572-577, 1993.

[Zacharia 95] Zacharia, N., J. Rajski, and J. Tyszer, "Decompression of Test Data Using Variable-Length Seed LFSRs," *Proc. of VLSI Test Symp.*, pp. 426-433, 1995.

[Zacharia 96] Zacharia, N., J. Rajski, J. Tyszer, and J. Waicukauski "Two Dimensional Test Data Decompressor for Multiple Scan Designs," *Proc. of International Test Conference*, pp. 186-194, 1996.