Improving Linear Test Data Compression

Kedarnath J. Balakrishnan, *Member, IEEE*, and Nur A. Touba, *Senior Member, IEEE*

***Abstract*—The output space of a linear decompressor must be sufficiently large to contain all the test cubes in the test set. The ideas proposed in this paper transform the output space of a linear decompressor so as to reduce the number of inputs required thereby increasing compression while still keeping all the test cubes in the output space. Scan inversion is used to invert a subset of the scan cells while reconfiguration modifies the linear decompressor. Any existing method for designing a linear decompressor (either combinational or sequential) can be used first to obtain the best linear decompressor that it can. Using that linear decompressor as a starting point, the proposed methods improve the compression further. The key property of scan inversion is that it is a linear transformation of the output space and, thus, the output space remains a linear subspace spanned by a Boolean matrix. Using this property, a systematic procedure based on linear algebra is described for selecting the set of inverting scan cells to maximize compression. A symbolic Gaussian elimination method to solve a constrained Boolean matrix is proposed and utilized for reconfiguring the linear decompressor. The proposed schemes can be utilized in various design flow scenarios and require no or very little hardware overhead. Experiments indicate that significant improvements in compression can be achieved.**

***Index Terms*—Linear decompression, linear feedback shift register (LFSR) reseeding, on-chip decompression, test data compression, XOR network.**

I. Introduction

With integrated circuits, especially system-on-chip (SoC) designs, becoming increasingly complex with each generation, the amount of test data required to achieve acceptable test quality is becoming very large. Hence, the test data storage requirements on an external tester and the test data bandwidth requirements between the tester and chip are growing rapidly [1]. Test data compression techniques provide a means to reduce these requirements thereby allowing less expensive testers to be used as well as reducing the test time. Compressing the output response is relatively easy since lossy compression techniques can be employed, e.g., using a multiple-input signature register (MISR). 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 significant amount of research has been done on lossless compression techniques for test vectors.

Manuscript received October 11, 2005; revised May 19, 2006.

K. J. Balakrishnan is with NEC Laboratories America, Princeton, NJ 08540 USA (e-mail: bala@nec-labs.com).

N. A. Touba is with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712 USA (e-mail: touba@ece.utexas.edu).

Digital Object Identifier 10.1109/TVLSI.2006.886417

An important class of test vector compression schemes involve a decompressor which uses only linear operations to decompress the test vectors. This class, which will be henceforth referred to as linear compression schemes, uses a linear decompressor. A number of different techniques for designing linear decompressors have been proposed in the literature. These include techniques based on linear feedback shift register (LFSR) reseeding and combinational linear expansion circuits consisting of XOR gates. Linear compression exploits the unspecified (don't care) bit positions in test cubes (i.e., deterministic test vectors where the unassigned bit positions are left as don't cares) to achieve large amounts of compression.

The idea of using an LFSR as a linear decompressor and solving for test cubes using linear algebra was described in [2]. Several techniques for improving the compression of LFSR based linear compression schemes were described in [3]–[8].

Linear decompressors that can receive data from the tester in a continuous-flow (i.e., "streaming" data) are especially useful for test data compression. Continuous-flow linear decompressors can be directly connected to the tester and operate very efficiently since they simply receive the data as fast as the tester can transfer it. From a tool integration standpoint, this is very nice since it mimics the standard behavior of normal scan chains. There is no need for any special scheduling or synchronization. A number of techniques for designing both combinational and sequential continuous-flow linear decompressors have been proposed. Combinational continuous-flow linear decompressors are described in [9]–[13]. Sequential continuous-flow linear decompressors are described in [14]–[19]. Most of the commercial tools for compressing test vectors are based on linear decompressors.

If there are c scan cells, then the space of all possible scan vectors is 2^c. The output space of a linear decompressor is the set of scan vectors that can be generated by the linear decompressor. Each bit stored on the tester can be thought of as a "free-variable" that can be assigned any value (0 or 1). Consider the case where the linear decompressor receives an input sequence from the tester consisting of n-free-variables when generating a scan vector. Assuming the linear decompressor is always initialized to the same state before generating each scan vector (if it is a sequential circuit), then the size of the output space of the linear decompressor is less than or equal to 2^n (since that is the number of possible unique input sequences that could be applied to it). The output space will be equal to 2^n if every input sequence maps to a unique scan vector, and less than 2^n if some input sequences map to the same scan vector. In the degenerate case where the linear decompressor is just a set of wires directly connecting each scan chain to a tester channel, then $n = c$ and the content of every scan cell is equal to a unique free variable such that the output space of the linear decompressor contains all possible scan vectors. However, in order to get compression,

n needs to be less than c, and thus, in the general case, the output space of the linear decompressor will be a subset of all possible scan vectors.

In order to be able to compress a test set, the output space of any decompressor must contain all the test cubes in the test set. Linear decompressors have some advantages compared with nonlinear decompressors. They generally have a larger output space for the same n because of the use of XOR gates which tend to minimize the number of input sequences that map to the same scan vector. Another useful property is that the output space of a linear decompressor is a linear subspace spanned by a Boolean matrix $\boldsymbol{A}$. The advantage of having the output space be defined by $\boldsymbol{A}$ is that determining whether a particular test cube is contained in the output space and the corresponding input sequence to generate it can be done by solving a set of linear equations using Gaussian elimination.

When using an LFSR as a linear decompressor, it has been shown that if the number of free-variables used to generate a test cube is 20 more than the number of specified bits in a test cube, then the probability of the test cube not being in the output space is less than one in a million [2]. However, for a given test set, the number of free-variables can be reduced further provided the corresponding reduced output space still contains all the test cubes in the test set. Reducing the number of free-variables decreases the amount of storage required on the tester and, hence, increases the compression. As the number of free-variables are reduced, the output space becomes smaller and smaller until a point is reached where one or more test cubes are no longer in the output space. This point terminates the reduction in free-variables and limits the compression that can be achieved by the linear compression scheme for a particular test set.

This paper describes two techniques to alter and reshape the output space of a linear decompressor which will allow the number of free-variables to be reduced further while still keeping all the test cubes in the output space thereby increasing the compression. Any method for designing a linear decompressor can be used first to obtain the best linear decompressor that it can. Using that linear decompressor as a starting point, the proposed techniques reduce the number of free-variables further to improve the compression.

The first proposed technique is based on scan inversion that leads to a linear transformation of the output space. Preliminary results were presented in [20]. The advantages of this technique are that it can be implemented without any hardware overhead and since the transformation is linear, a systematic procedure exists to select the set of inverting scan cells based on linear algebra. The second methodology described in this paper enables the *reconfiguration* of any linear decompressor. Preliminary results were presented in [21]. Reconfiguration of a decompressor modifies the output space of the decompressor resulting in higher compression. This technique can be used in SoC designs that have intellectual property (IP) cores where the core internals are not visible. Further, in multicore SoCs, a single "reconfigurable" decompressor can be used for several cores instead of having several decompressors thereby providing a significant reduction in hardware overhead.

Note that the techniques described in this paper can be used in conjunction with any linear decompressor including all of the ones previously referenced to improve the compression further. They transform the output space of the linear decompressor in a way that allows a smaller number of free-variables from the tester to be used to encode the test set.

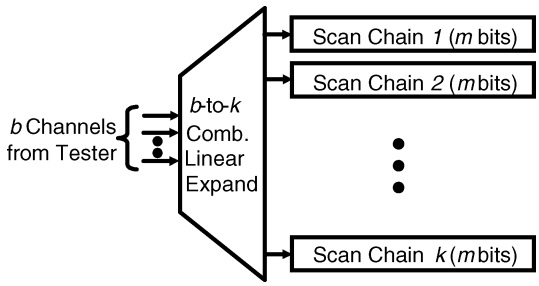

Fig. 1. Combinational linear decompressor.

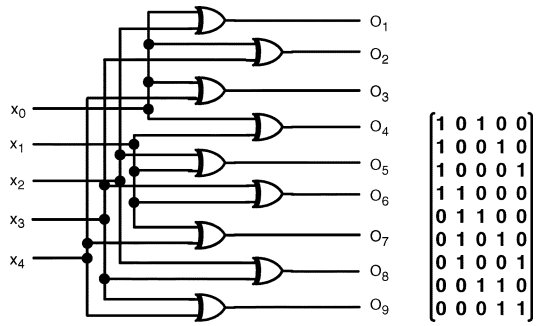

Fig. 2. XOR network and matrix for Fig. 1.

The rest of this paper is organized as follows. The next section describes the motivation behind using the proposed techniques to transform the output space of a linear decompressor. Section III describes the hardware implementation while Sections IV and V present systematic techniques for performing scan inversion and reconfiguration. Section VI discusses how the proposed techniques can be used to improve the compression of any linear compression scheme. Section VII details experimental results and conclusions are in Section VIII.

II. Transforming Output Space

This section describes the motivation behind the proposed techniques to transform the output space of a linear decompressor. As mentioned earlier, the output space of a linear decompressor can be described using a Boolean matrix $\boldsymbol{A}_{c\times n}$. Each row in $\boldsymbol{A}$ corresponds to a scan cell and each column corresponds to a free-variable in the input sequence.

Fig. 1 shows a combinational linear decompressor that receives b bits from the tester and expands it to k scan chains. The Boolean matrix for this decompressor can be constructed simply from the XOR network. Each row corresponding to a scan chain will have the inputs that are XORed together to get the scan chain value as 1 and all the others as 0. An example of a XOR network with $b = 5$ and $k = 9$ and the corresponding matrix is shown in Fig. 2. The columns of the matrix correspond to inputs with the first column representing x_0 and the last column representing x_4. Output O_1 is the XOR of inputs x_0 and x_2 and, hence, in the first row, the values corresponding to these two inputs are 1 while the rest are 0. Obtaining the Boolean matrix A

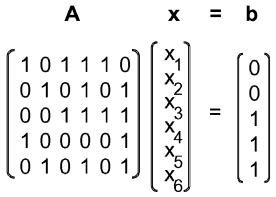

Fig. 3. System of equations for test cube t_1.

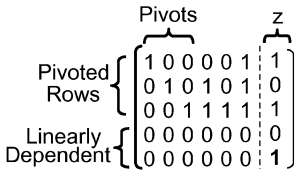

Fig. 4. After Gaussian–Jordan reduction.

for LFSR-based decompressors by symbolic simulation of the linear decompressor is described in detail in [8].

For a test cube to be compressible using a linear compression scheme, a solution to the system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$ must exist, where $\boldsymbol{x}$ is an assignment of values to the free-variables that are inputs to the linear decompressor when generating the test cube, and $\boldsymbol{b}$ is the value of each bit in the test cube. There is no need to solve the linear equations for the unspecified bits in the test cube, and hence, only the linear equations (rows) corresponding to the specified bits in $\boldsymbol{b}$ need to be considered. Gaussian elimination [22] can be used to solve the set of linear equations. Gaussian–Jordan reduction is used to perform row operations that transform a set of columns into an identity matrix. The elements that make the identity matrix are called pivots.

An example of a system of linear equations for a test cube t_1 is shown in Fig. 3 and the corresponding system after Gaussian–Jordan reduction is shown in Fig. 4. The rows after Gaussian–Jordan reduction can be classified as either pivoted rows or linearly dependent rows. The pivoted rows have pivots while the linearly dependent rows are all 0. For the example in Fig. 3, the first three rows are pivoted while the last two rows are linearly dependent. If all rows are pivoted, then a solution to the system of linear equations exists, and hence, the test cube can be decompressed using the linear decompressor. If some of the rows are linearly dependent, then a solution only exists if all of the corresponding values in $\boldsymbol{z}$ (the vector $\boldsymbol{b}$ after Gaussian elimination) are equal to 0 for the linearly dependent rows. If there is a linearly dependent row whose corresponding value in $\boldsymbol{z}$ is equal to 1, then no solution exists. In Fig. 4, the last row is linearly dependent but the corresponding value in $\boldsymbol{z}$ is 1, and thus, there is no solution. This is easy to see because in the original system of linear equations in Fig. 3, both rows 2 and 5 are identical in $\boldsymbol{A}$, however, the corresponding values in $\boldsymbol{b}$ have opposite values. Obviously, there is no assignment to $\boldsymbol{x}$ that will simultaneously solve the linear equations for both rows 2 and 5.

For the example in Fig. 3, let the specified values in $\boldsymbol{b}$ correspond to scan cells c_1 through c_5. If either scan cell c_2 or scan cell c_5 is inverted, then the system of linear equations becomes solvable. This is illustrated in Fig. 5 which shows the

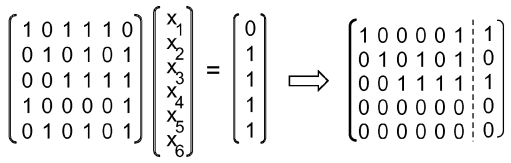

Fig. 5. System of equations for t_1 with c_2 inverted.

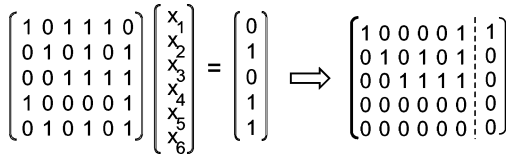

Fig. 6. Reconfigured system of equations for t_1.

new system of equations (with the value on the second row of the vector $\boldsymbol{b}$ changed to 1) and the result of Gaussian–Jordan reduction after scan cell c_2 is inverted. This is an example of how scan inversion can be used to make a test cube solvable. If the decompressor is reconfigured such that the equations corresponding to scan cell c_2 and scan cell c_3 were exchanged, the system of linear equations is again solvable. This is shown in Fig. 6. The last two rows are linearly dependent but the corresponding value in $\boldsymbol{z}$ is 0. This example illustrates how reconfiguration can be used to make a test cube solvable.

The proposed schemes have some relation to the idea of transforming the output space of a test pattern generator to encode test cubes which has been investigated in the past in the context of built-in self-test (BIST). Techniques for designing a nonlinear circuit to transform the output space of a pseudorandom generator to encode test cubes were described in [23]–[31]. Scan inversion has been used in other contexts, in BIST [32] and to reduce scan shift power [33].

A switch-based reconfiguration of the connection between external pins and the scan chains in the Illinois scan architecture [9] was proposed in [34]. In [13], control bits are used to determine the number of scan chains that are fed by the combinational network. Configuration multiplexers at the output of phase shifter network was proposed in [35].

The proposed methods differ significantly from these methods in that they use linear transformations, are based on linear algebra, are targeted towards lossless test vector compression, and can be implemented with very little overhead.

III. Implementation

A. Scan Inversion

Implementing inverted scan cells can be accomplished either by explicitly inserting inverters in the scan chains, or by simply using the $\bar{Q}$ output instead of the Q output when connecting the output of one scan cell to the next scan cell. An example of inverting the contents of a scan cell is shown in Fig. 8. Fig. 7 shows a normal scan chain with no inversion, while Fig. 8 shows the scan chain with the third scan cell inverted. This is accomplished by inverting before and after the third scan cell. If the same scan vector was shifted into both the normal scan chain in Fig. 7 and the scan chain in Fig. 8, the contents of the third scan cell in Fig. 8 would have the opposite value from what it

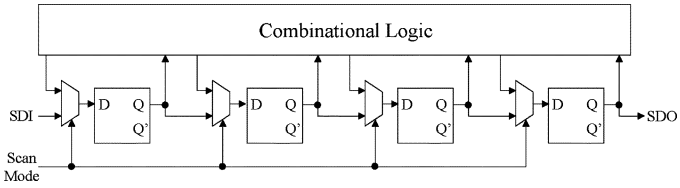

Fig. 7. Normal scan chain.

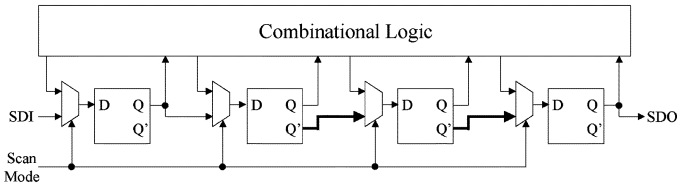

Fig. 8. Scan chain with third scan cell inverted.

would be in the normal scan chain while the contents of all other scan cells would be the same. Also, when the output response is shifted out, the bit corresponding to the third scan cell will have the opposite value from what it normally would have.

While the example in Fig. 8 only involves inverting one scan cell, any set of scan cells in the scan chain can be inverted by making appropriate connections from either Q or $\bar{Q}$ to the next scan cell. To invert the first scan cell in a scan chain, either an inverter can be placed on the scan_data_in (SDI) input or the XOR (XNOR) gate that is driving the SDI input can simply be changed to an XNOR (XOR) gate. If the inverted scan cells are implemented by simply changing some connections from Q to $\bar{Q}$ and changing some XOR gates to XNOR gates, then there is effectively no overhead for using scan inversion. Even if explicit inverters are used, the overhead would still be small. Note that this technique does not involve any special ordering of the scan cells. The scan chains can be ordered in any manner to optimize the layout.

B. Reconfiguring a Linear Decompressor

There are several possible ways to modify a linear decompressor such that it can be reconfigured. A simple modification would be to add multiplexers to each output of the network with the select input coming from *configuration bits*. The remaining data inputs could be some other output of the original decompressor. For example, consider the combinational decompressor using XOR gates shown in Fig. 2. Instead of directly sending the outputs to the scan chains, there is another stage with a multiplexer before each scan chain as illustrated in Fig. 9. These multiplexers will select which output is connected to which scan chain based on the configuration bits.

In general, the linear decompression network can be modified using the configuration bits. For each configuration, the output space of the linear decompressor will be different. Hence, this increases the chances of a test cube being generated using the decompressor. The configuration bits can either be stored on the tester and transferred to the decompressor with the test cubes or stored on-chip using a ROM. The number of different configurations required depends on the test set. In the case when each test cube in a test set requires a different configuration, these

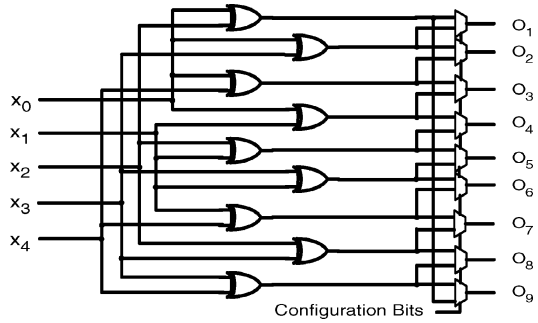

Fig. 9. Making a decompressor reconfigurable.

$$\mathbf{A}\quad\mathbf{x}\quad=\quad\mathbf{b}\qquad c_1\,c_2\,c_3\,c_4\,c_5$$

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

Fig. 10. Augmented system of equations for t_1.

need to be explicitly stored. In the case when all the test cubes can be generated using the same configuration, then the decompression network can be finalized for that configuration and no extra storage is required. This implies that the decompressor is simply *redesigned*.

IV. Selecting Set of Inverted Scan Cells

To implement scan inversion, a set of scan cells to be inverted need to be selected. If there are c scan cells, then there are 2^c different ways the scan cells can be inverted each of which results in a different transformation of the output space of the linear decompressor. Given a particular linear decompressor, the goal is to select the set of inverted scan chains so that the resulting transformed output space of the linear decompressor will contain all of the test cubes in the test set. This section describes a systematic procedure based on linear algebra for doing this.

Let $i = (i_1, i_2, \ldots, i_c)$ be a vector in which if $i_j = 1$ then scan cell j is inverted and if $i_j = 0$ then scan cell j is not inverted. For a given test cube, a set of constraints on i for which the test cube is solvable can be determined as follows. Consider the example system of linear equations in Fig. 3 and the corresponding Gaussian–Jordan reduced version in Fig. 4. For the rows in Fig. 4 that are pivoted, there are no constraints on i as it does not matter what the corresponding value of $\boldsymbol{z}$ is for those rows since they are always solvable. For the linearly dependent rows, the corresponding value of $\boldsymbol{z}$ must be 0, so whatever scan inversion takes place must ensure that the value for that row is 0 after Gauss-Jordan reduction. This places constraints on i. In the example in Fig. 4, the last two rows are linearly dependent, so constraints must be placed on i to ensure that the corresponding values in $\boldsymbol{z}$ for those rows are always 0. These constraints can be determined by augmenting the linear system with a dependency matrix as shown in Fig. 10. The dependency matrix is a square matrix with the number of rows and columns equal to the number of rows in $\boldsymbol{b}$ and initialized to an identity matrix

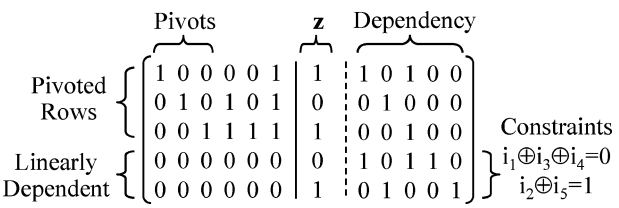

Fig. 11. Gaussian–Jordan reduction with augmented matrix for t_1.

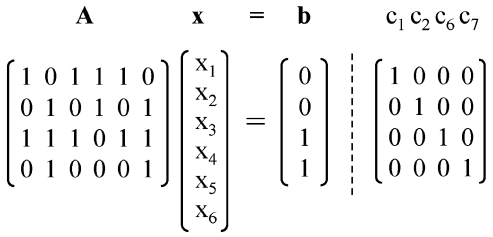

Fig. 12. Augmented system of equations for t_2.

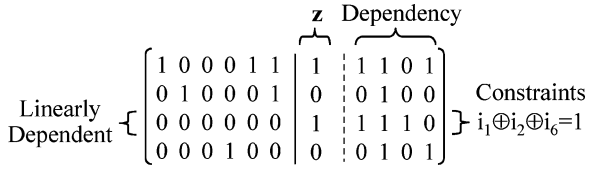

Fig. 13. Gaussian–Jordan reduction of augmented matrix for t_2.

since each value in $\boldsymbol{b}$ depends only on itself. The row operations that are performed during Gaussian–Jordan reduction are also applied to the dependency matrix. After Gaussian–Jordan reduction, the dependency matrix indicates which set of scan cells each value in $\boldsymbol{z}$ depends on. For each linearly dependent row, the constraints on i can be obtained by looking at the dependency for the value in $\boldsymbol{z}$ for that row. The value in $\boldsymbol{z}$ for each linearly dependent row must be 0, so a linear equation in terms of i can be written to ensure that the value in $\boldsymbol{z}$ will be 0. In the example in Fig. 11, the fourth row is linearly dependent and the corresponding value in $\boldsymbol{z}$ depends on scan cells c_1, c_3, and c_4. Since the corresponding value in $\boldsymbol{z}$ in the normal scan chain is 0, the constraint on scan inversion is that $i_1 \oplus i_3 \oplus i_4$ should be 0. In other words, either none of scan cells c_1, c_3, and c_4 should be inverted or two of them should be inverted (thus, canceling each other out). If one or all three of them are inverted, then the value in $\boldsymbol{z}$ for that row will become 1 and no solution for the test cube will exist. Similarly, a linear equation for the constraint due to the fifth row can be obtained. In this case, for the normal scan chain the value of $\boldsymbol{z}$ for this row is 1, so it is necessary that one of the scan cells that it depends on be inverted, either c_2 or c_5 in order to get a solution for the test cube.

The procedure described previously can be used to obtain a set of constraints on i for each test cube to make it solvable. Consider test cube t_2 whose system of linear equations is shown in Fig. 12. Note that this test cube has only four specified bits and, in this case, the specified bits are in scan cells c_1, c_2, c_6, and c_7. The system of linear equation after Gaussian–Jordan elimination is shown in Fig. 13. As can be shown, there is one

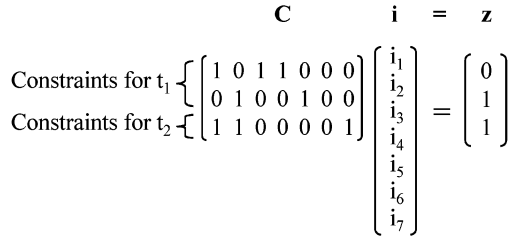

Fig. 14. Constraint matrix for test set $\{t_1, t_2\}$.

linearly dependent row in this case. The corresponding set of constraints on i can be obtained from the dependency matrix.

In order for the linear decompressor to be able to generate all the test cubes in the test set, there needs to be a solution for each test cube. Thus, i must be selected to allow all test cubes to be simultaneously solved. A solution for i is obtained by forming a constraint matrix $\boldsymbol{C}$ that incorporates all of the constraints for all the test cubes. Each constraint for each test cube is added as a row in $\boldsymbol{C}$. For example, suppose the test set consisted of test cube t_1 and t_2 whose constraints were obtained in Figs. 11 and 13. Then the first two rows in $\boldsymbol{C}$ would correspond to the two linear constraint equations from Fig. 11, and the last row in $\boldsymbol{C}$ would correspond to the linear constraint equation from Fig. 13. The resulting constraint matrix is shown in Fig. 14. Gaussian elimination is then used to find a solution to the system of linear equations $\boldsymbol{Ci} = \boldsymbol{z}$. The solution for i gives the set of scan cells that need to be inverted so that the test set can be generated by the linear decompressor. If no solution exists, then it is not possible for the linear decompressor to be used to generate the test set under any set of inverted scan cells.

The complexity of the Gaussian elimination method for solving a set of n linear equations with m variables is of the order of $O(nm^2)$. In the proposed scheme, as with any other linear test vector compression scheme, a set of linear equations needs to be solved for each test cube. Hence, if s is the number of specified bits in a cube and c is the number of compressed bits for that cube, then solving for that cube will require $O(sc^2)$ time. The only additional task involved in the proposed scheme is to solve for the constraint matrix. The time complexity for this additional step will depend on how many equations there are in the constraint matrix and the number of scan cells that are included in the inversion constraints. Note that there is no need to include the scan cells that are not present in any inversion constraint for any test cube.

For large designs, a partitioning approach can be used to reduce the time complexity of solving the constraint equations. For combinational decompressors, partitioning is simple. Each scan slice is treated as one partition and the constraints for all the scan flip-flops in this slice can be solved together for all the test vectors. This partitioning will not affect the improvement in compression obtained using scan inversion. For sequential decompressors, multiple scan slices can form one partition and the equations for each partition solved separately. However, this requires the sequential decompressor to be reset after each partition and, hence, increases the scan shift cycles. The execution time of scan inversion with and without partitioning are reported and discussed further for large scan architectures in the results section.

$$\begin{pmatrix} f_{11}(a,b,c) & f_{12}(a,b,c) & f_{13}(a,b,c) & \dots & f_{1m}(a,b,c) \\ f_{21}(a,b,c) & f_{22}(a,b,c) & f_{23}(a,b,c) & \dots & f_{2m}(a,b,c) \\ f_{31}(a,b,c) & f_{32}(a,b,c) & f_{33}(a,b,c) & \dots & f_{3m}(a,b,c) \\ \dots & \dots & \dots & \dots & \dots \\ f_{n1}(a,b,c) & f_{n2}(a,b,c) & f_{n3}(a,b,c) & \dots & f_{nm}(a,b,c) \end{pmatrix}$$

Fig. 15. Matrix with functions as elements.

V. Symbolic Gaussian Elimination

Section II described how reconfiguration of a linear decompressor can increase the chances of a test cube being solvable. In this section, we describe a systematic procedure to do reconfiguration using *symbolic Gaussian elimination*. The key idea is to form the matrix $\boldsymbol{A}$ in terms of the configuration bits (i.e., each entry in the matrix is a function of the configuration bits) and then find an assignment of the configuration bits that makes the system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$ solvable.

In traditional Gaussian elimination, elementary row operations are used to reduce the coefficient matrix (the matrix $\boldsymbol{A}$) to a set of pivoted and linearly dependent rows. This is done by going through each column of the matrix and choosing a nonzero element as the pivot. All other rows with a 1 in this column are XORed with this row as part of elementary row operation. The resultant matrix will either have pivoted rows or linearly dependent rows.

We extend this technique to coefficient matrices that have *functions* as elements. The functions are Boolean functions on a given set of variables. This is illustrated in Fig. 15, where each element of the matrix $\boldsymbol{F}$ is a function of the variables $\{a, b, c\}$. In this matrix, each f_{ij} is a function such that

$$f_{ij} : \{0,1\}^3 \rightarrow \{0,1\}.$$

In the trivial case when each function is a constant 1 or a constant 0, this matrix will degenerate to a Boolean matrix with entries 0 or 1. The variables $\{a, b, c\}$ correspond to the configuration bits of the linear decompressor. Solving for a system of linear equations such as $\boldsymbol{Fx} = \boldsymbol{y}$, with $\boldsymbol{F}$ as the matrix shown in Fig. 15 and $\boldsymbol{y}$ is a vector that implies we are looking for a solution for at least one combination of a, b, and c, or in other words, for at least one configuration.

Given such a system of linear equations, the algorithm for symbolic Gaussian elimination is given as follows. First, each element in the $\boldsymbol{y}$ vector is converted to a function in terms of the variables in $\boldsymbol{F}$. If the element in $\boldsymbol{y}$ is 1 then the function is identically equal to 1 and if the element is 0, the function is identically equal to 0. Then, the algorithm proceeds column wise choosing pivots in each column. Suppose f_{11} is the first pivot. Row operations are performed next for each row and this pivot. In the row operations, every element in a row is XORed with the result of the AND of its element in the pivot column and the corresponding element in the pivot row. This is illustrated in Fig. 16 that shows the matrix after row operations for pivot f_{11}. The idea is to take into account both cases when $f_{11} = 0$ and $f_{11} = 1$. This operation is repeated for every pivot.

The matrix after doing all the row operations for each pivot will look like Fig. 17 where f'_{ij} are again functions of $\{a, b, c\}$. For ease of explanation, assume that the pivot for each column

$$\begin{pmatrix} f_{11} & f_{12} & \dots & f_{1m} \\ f_{21} \oplus f_{21} \bullet f_{11} & f_{22} \oplus f_{21} \bullet f_{12} & \dots & f_{2m} \oplus f_{21} \bullet f_{1m} \\ f_{31} \oplus f_{31} \bullet f_{11} & f_{32} \oplus f_{31} \bullet f_{12} & \dots & f_{3m} \oplus f_{31} \bullet f_{1m} \\ \dots & \dots & \dots & \dots \\ f_{n1} \oplus f_{n1} \bullet f_{11} & f_{n2} \oplus f_{n1} \bullet f_{12} & \dots & f_{nm} \oplus f_{n1} \bullet f_{1m} \end{pmatrix}$$

Fig. 16. Matrix after row operations for pivot f_{11}.

$$\begin{pmatrix} f'_{11} & f'_{12} & f'_{13} & \dots & f'_{1m} \\ f'_{21} & f'_{22} & f'_{23} & \dots & f'_{2m} \\ f'_{31} & f'_{32} & f'_{33} & \dots & f'_{3m} \\ \dots & \dots & \dots & \dots & \dots \\ f'_{n1} & f'_{n2} & f'_{n3} & \dots & f'_{nm} \end{pmatrix}$$

Fig. 17. Matrix after all row operations.

i is given by element f'_{ii}. The next step would be to ensure that each pivot has at least one minterm for which the function equates to one. If the pivot function does not have a single minterm that equates to one, then that row is equivalent to a linearly dependent row in the normal Gaussian eliminated matrix. Hence, the corresponding element in the $\boldsymbol{y}$ matrix should be zero. This condition for a pivot f_{ii} can be written in mathematical form as

$$f'_{ii} + \bar{y_i} = 1.$$

Since this condition must be valid for each pivot, the overall condition can be written in a product of sum form as

$$(f'_{11} + \bar{y_1})(f'_{22} + \bar{y_2}) \cdots (f'_{nn} + \bar{y_n}) = 1.$$

If the previous condition is satisfied, there exists a solution to the system of equations. The number of minterms of $\{a, b, c\}$ for which the previous condition is satisfied will indicate the number of different configurations possible.

Note that there may be more than one possible pivot for each column and, hence, the pivot is selected using a heuristic. For the first column, the element that has the maximum number of minterms is chosen as the pivot since this retains the best chances of solving the system of equations. For the next columns, the element that has the most number of minterms in common with the current pivots is chosen. This ensures that the algorithm proceeds in such a way that locally maximal number of solutions (configurations) are possible after each step.

The algorithm described before can be implemented with very little overhead with respect to the basic Gaussian elimination method. Each function is stored in terms of its minterms. For example, if three configuration bits are used, then there are eight possible minterms. Each element in the matrix consists of eight values, one corresponding to each minterm. The row operations are performed on the corresponding minterms. The only additional step in this procedure is evaluating the final product of sums condition. This can be achieved by simply performing bitwise operations on the entries so that the corresponding minterms are evaluated together. The number of

entries in the product of sums condition depends on the number of pivots and, hence, on the size of the matrix. The complexity of the evaluation step increases linearly with the size, since the bitwise operations need to be performed for each additional pivot. The number of minterms for each function depend on the number of configuration bits. The number of configuration bits is a design parameter that can be determined based on the compression required and the maximum allowed running time of the algorithm.

VI. Increasing Test Compression

Given a linear decompressor, the previous sections described how to reconfigure it or select a set of scan cells to invert so that all the test cubes in the test set will be in the output space of the decompressor. It is obvious that scan inversion has to be fixed for all the test cubes in the test set, i.e., the set of scan cells that are inverted will be the same for all the test cubes. However, reconfiguration can be done in two ways. The first is to search for a single configuration by which all the cubes in the test set can be compressed. In this case, the configuration can be hardwired into the decompressor, i.e., the decompressor is *redesigned* and no explicit configuration bits are required. The other method would be to have one configuration for each test cube which is loaded into the decompressor every time a new test cube is loaded. The configuration bits for each test cube need to be stored explicitly. Since the proposed schemes work on a specific set of scan patterns, they are not guaranteed to be able to encode top-off patterns that were not accounted for during the design flow. However, if the test infrastructure has a bypass serial mode for diagnosis, any unencodable top-off patterns could be applied that way if necessary.

Compression obtained using a linear decompression scheme can be improved using the proposed techniques in several ways. One method would be to reduce the number of free-variables that the decompressor receives per test cube from the tester as much as possible while still keeping the test set compressible through scan inversion or reconfiguration. Any method can be used to design the initial decompressor. Then the number of free-variables that are input to the decompressor per test cube can be incrementally reduced and the techniques described in Sections IV and V can be used to check whether it is possible to still solve for all the test cubes using the proposed schemes. If so, then this process of incrementally reducing the number of free-variables and checking for a solution is repeated until a point is reached when no further reduction in the number of free-variables per test cube is possible while still being able to solve for all test cubes. The end result will be a linear decompressor that generates the exact same test set, but uses fewer tester channels thereby reducing tester storage and bandwidth requirements.

If the number of tester channels that are allocated for feeding the linear decompressor is fixed, then another way the proposed techniques can be used is to allow more specified bits per test cube. The idea is to keep the number of free-variables that the decompressor receives per test cube constant, but use the proposed techniques to relax the constraints on automatic test pattern generation (ATPG) such that more specified bits per test cube can be generated. This will allow more static and dynamic compaction while still being able to solve for the test cubes. Some test compression methodologies (e.g., [10], [11], [16]) involve fixing the decompressor design and then constraining the ATPG so that the resulting test cubes will be in the output space of the decompressor. The constraints on the ATPG reduce the amount of static and dynamic compaction that are performed and, therefore, can result in more test cubes and, hence, more test time. Reconfiguration of the linear decompressor or scan inversion can be used to allow more specified bits per test cube while still being able to solve for the test cube. This can be used to relax the constraints on the ATPG and thereby allow more static and dynamic compaction which will in turn reduce the total number of test cubes and, hence, result in a reduction of both test time and tester storage requirements.

VII. Experimental Results

Two sets of experiments were performed to evaluate the effectiveness of the proposed methods. The first set of experiments consisting of those described in the previous section was done on randomly generated test cubes for large industrial-size scan architectures. The other set of experiments were performed on 100% stuck-at fault coverage test sets for the largest ISCAS'89 [36] benchmark circuits. The experiments were performed on both types of linear decompressors—combinational and sequential. The results for each are discussed in detail as follows.

A. Combinational Decompressor

The combinational linear decompressor consists of XOR gates as shown in Fig. 1. Circuits are assumed to have either 512 or 1024 scan chains and the initial number of channels from the tester is assumed to be 32.

The results for scan inversion are shown in Table I. For each randomly generated test cube, the number of specified bits was incrementally increased until it could no longer be solved (i.e., it was no longer in the output space). The average percentage of specified bits per test cube that could be solved for is shown along with the corresponding encoding efficiency. This measures the best encoding efficiency that can be achieved for normal scan chains without scan inversion. Recall that the encoding efficiency is defined as the ratio of the number of specified bits in the test set to the number of bits stored on the tester. Note that since each test cube is encoded independently, there is no dependence on the number of test cubes. The running time of the program to find the tester variables by solving the linear equations for all the test cubes is shown in column 6. All reported times in this paper are on a 3.2-GHz Pentium-4 machine with 2-GB memory running linux. Results are then shown for scan inversion using it in the two ways described in the previous section. The first is using scan inversion to reduce the number of tester channels while still encoding the same set of test cubes as before. The number of reduced tester channels along with the resulting encoding efficiency that is achieved are shown in columns 7 and 8, respectively. The time taken for the two phases of the algorithm (solving equations for test patterns and solving constraint equations) are shown in columns 9 and 10, respectively. The second way that scan inversion is used is to keep the tester channels at 32 and increase the percentage

TABLE I
RESULTS FOR SCAN INVERSION ON COMBINATIONAL LINEAR DECOMPRESSOR

Num. Scan Chains	Num. Test Cubes	Without Scan Inversion				With Scan Inversion					
		Tester Channels	Percentage Specified	Encoding Efficiency	Running Time (sec)	Reduced Channels	Encoding Efficiency	Time (sec)		Increased % Specified	Encoding Efficiency
								Phase-I	Phase-II		
512	200	32	2.7 %	0.43	0.96	11	1.26	1.11	0.01	6.2 %	0.99
	400	32	2.7 %	0.43	1.92	13	1.06	2.02	0.02	5.7 %	0.91
	600	32	2.7 %	0.43	3.02	15	0.92	3.36	0.02	5.5 %	0.88
	1000	32	2.7 %	0.43	4.89	16	0.86	5.44	0.03	5.2 %	0.83
1024	200	32	1.3 %	0.42	1.26	11	1.21	1.50	0.04	3.5 %	1.12
	400	32	1.3 %	0.42	2.61	11	1.21	3.07	0.08	3.1 %	0.99
	600	32	1.3 %	0.42	3.92	13	1.02	4.62	0.13	2.9 %	0.93
	1000	32	1.3 %	0.42	6.47	14	0.95	7.75	0.46	2.8 %	0.90

TABLE II
RESULTS FOR RECONFIGURATION ON COMBINATIONAL LINEAR DECOMPRESSOR

Num. Scan Chains	Scan Chain Length	Without Reconfiguration				Reducing Channels			Increasing Spec. Bits		
		Tester Chan.	Percent. Spec.	Encod. Effic.	Compr. Ratio	Red. Chan.	Compr. Ratio	% Impr.	Incr. % Spec.	Encod. Effic.	% Impr.
512	24	32	3.61 %	0.578	16.0	23	21.4	33.8 %	4.69 %	0.727	25.8%
	32	32	3.42 %	0.547	16.0	23	21.6	35.0 %	4.30 %	0.667	21.9%
	64	32	2.83 %	0.453	16.0	22	22.9	43.1 %	4.00 %	0.621	37.1%
	128	32	2.67 %	0.427	16.0	24	21.2	32.5 %	2.91 %	0.661	54.8 %
1024	24	32	1.66 %	0.531	32.0	24	41.0	28.1 %	2.34 %	0.727	36.9%
	32	32	1.66 %	0.531	32.0	24	41.4	29.4 %	2.25 %	0.697	31.3%
	64	32	1.49 %	0.476	32.0	22	45.8	43.1%	2.18 %	0.677	42.2%
	128	32	1.50 %	0.480	32.0	22	46.2	44.4%	2.20 %	0.680	41.7%

of specified bits per test cube as much as possible until it is no longer possible to solve for all the test cubes. The maximum percentage of specified bits and the resulting encoding efficiency are shown in the last two columns of Table I. Note that when using scan inversion, the effectiveness reduces as the number of test cubes increases since it is harder to keep all of the test cubes in the output space. Results are shown for several different numbers of test cubes.

A number of interesting observations can be made from Table I. Scan inversion is remarkably effective at improving the encoding efficiency for combinational decompressors (it is more than doubled in most cases). Typically the encoding efficiency for combinational decompressors is fairly low because of the fact that they must receive enough free-variables every clock cycle to be able to encode each bit-slice of the scan chains. The worst case most heavily specified bit-slices typically limit the encoding efficiency. As can be shown in Table I, the encoding efficiency is less than 0.5 without scan inversion. However, with scan inversion, the most heavily specified bit slices can be solved with fewer free-variables per clock cycle thereby allowing the number of tester channels to be reduced substantially. The fact that the encoding efficiency can be increased to around 0.9 with a combinational decompressor is a surprising result. This level of encoding efficiency is of the order of what is typically achieved with sequential decompressors, however, in this case no LFSR is needed thereby reducing the hardware overhead. Combinational decompressors are attractive because they are very simple requiring low hardware overhead. The only drawback to using them has been the substantially reduced encoding efficiency compared with sequential decompressors, however, these results indicate that with scan inversion the gap in encoding efficiency between combinational and sequential decompressors can be significantly reduced. The results for keeping the number of channels at 32 and increasing the percentage of specified bits per test cube did not provide as high encoding efficiency as reducing the tester channels did.

The execution time of scan inversion algorithm as reported in Table I indicate that the increase with respect to no inversion is around 10% for 512 scan chains and 20% for 1024 scan chains. Further, the time required for solving constraint equations (reported as "Phase-II") is negligible. This is because the constraint matrix is constructed and solved separately for the flip-flops in each scan slice. This type of partitioning does not affect the compression for combinational decompressors. The maximum memory used by the scan inversion program (1024 scan chains and 1000 test cubes) was 11 MB.

For reconfiguration, a four-input multiplexer was added to the output of the combinational decompressor with the select bits coming from the configuration bits. All the experiments were performed with a total of eight configuration bits. The results are presented in Table II. The first two columns show the number of scan chains and length of each scan chain. The length of the scan chains varied from 24 to 128. The columns under "Without Reconfiguration" show the maximum compression that can be obtained using the given decompressor. For each randomly generated test cube, the number of specified bits was incrementally increased until it could no longer be solved using the given decompressor. The maximum percentage of specified bits per test cube that could be solved by the given decompressor is shown in column 4. The corresponding encoding efficiency and compression ratios are given in columns 5 and 6. Compression ratio is the ratio of the original bits in the test set to the compressed bits.

TABLE III
RESULTS FOR ISCAS'89 BENCHMARKS WITH COMBINATIONAL DECOMPRESSOR

Circuit	Illinois [9]	XOR Net. [10]	Adj. Width [13]	Scan Inversion	Reconfiguration
s13207	109,772	25,344	14,307	13,298	14,098
s15850	32,758	22,784	15,067	15,590	18,080
s38417	96,269	89,856	49,001	43,256	54,020
s38584	96,056	38,976	28,994	29,558	31,436

The columns under "Reducing Channels" show the results for reduction in the number of tester channels using reconfiguration. The reduced number of tester channels along with the corresponding compression ratio are shown under "Red. Chan." and "Compr. Ratio." The configuration bits are assumed to be shifted in at the first cycle so there is no need to have separate channels for the configuration bits. Note that each test pattern requires one additional shift cycle which is taken into account when calculating the compression ratio. The area overhead of the reconfiguration part is one multiplexer for each scan chain and one flip-flop for each configuration bit. The column "% Impr." shows the improvement in compression ratio due to the reconfiguration. The average percentage improvement for all the different scan sizes is around 36.2% which is a significant improvement for the extra hardware overhead. The columns under "Increasing Spec. Bits" show the results for increasing the percentage of specified bits that can be handled by a given decompressor using reconfiguration. The tester channels are kept constant at 32 and the number of specified bits are increased as much as possible until it is no longer possible to solve for all the test cubes. The new percentage of specified bits and encoding efficiency are shown as well as the percentage improvement in the encoding efficiency. The experiments assume a single configuration for each test cube and the configuration bits are taken into account while calculating the compression results. In this experiment, the average percentage improvement for all the different scan sizes is 36.5%.

To compare the improvement in the compression obtained using a combinational linear decompressor by the proposed techniques, another set of experiments were performed. These were done on 100% stuck-at fault coverage test sets for the largest ISCAS'89 [36] benchmark circuits. Table III compares the compression results obtained using reconfiguration and scan inversion with some of the combinational decompressor techniques proposed earlier. The amount of test data that need to be stored on the tester for the Illinois scan architecture [9], XOR network [10] and the adjustable width technique [13] are compared with those of the proposed schemes.

Note that in [9] and [10] the decompressor design is integrated into the test pattern generation, while in [13] compression is performed on already generated test patterns. However, the scheme proposed in [13] requires that each group of scan chains be controlled by separate clocks [so that one or more group(s) can be shifted simultaneously]. The test patterns used in our experiments were the same as in [13]. Both reconfiguration and scan inversion perform better than Illinois scan [9] and XOR network [10] for all circuits. This is expected since the main aim of these techniques is to improve compression of such schemes. The compression obtained by the proposed schemes are similar to [13], while controlling clocks during scan shifting is not needed.

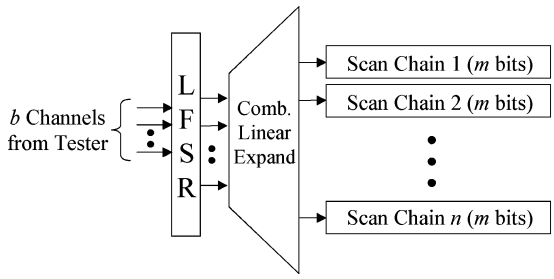

Fig. 18. Sequential linear decompressor.

B. Sequential Decompressor

Experiments were performed to study the impact of scan inversion on compression of a continuous flow sequential linear decompressor as shown in Fig. 18. The results for randomly generated test cubes are shown in Table IV. In this case, the number of channels from the tester for the normal scan chain without inversion was 16 and a 64-bit LFSR was used. Because the original sequential linear decompressor is very efficient already, the number of channels that can be reduced using scan inversion is not as spectacular as for combinational decompressors. Nonetheless, a significant increase in the encoding efficiency (above 1 in many cases) can be achieved especially for 1024 scan chains where the percentage of specified bits per test cube is less. Note also that the results for keeping the number of channels at 16 and increasing the percentage of specified bits per test cube were actually better than reducing the tester channels. This is the opposite of what happened for combinational decompressors where reducing channels achieved higher encoding efficiency.

The execution time of scan inversion for reducing channels of a sequential decompressor is shown in Table V. For each scan architecture and number of test cubes reported in Table IV, the corresponding running times are shown for basic compression (row "Without Inversion"), with inversion (row "Inversion") and using partitioning with inversion (row "Partitioning"). To get the results for partitioning, the scan flip-flops were divided into two partitions and the equations for each partition were solved separately. Though it is possible that partitioning may reduce the optimality of the results, we did not see any change in the number of reduced channels in our experiments when using partitioning. The time taken by both phases of scan inversion; solving test pattern equations (labeled "Phase-I") and solving constraint equations (labeled "Phase-II") are shown separately. From Table IV, it is clear that the time taken for the first phase increases steadily with the number of test cubes as expected while the time for the second phase varies depending on the number of constraint equations to be solved. By partitioning into two partitions, the running time of scan inversion is reduced by a factor of 3.2 on average. The maximum memory used by the scan inversion program was 1.1 GB (for 1024 scan chains and 1000 test cubes) while partitioning reduced the memory usage to 800 MB for the same scan architecture.

Experiments were also performed on 100% stuck-at fault coverage test sets for the largest ISCAS'89 benchmark circuits to

TABLE IV
RESULTS FOR SCAN INVERSION ON SEQUENTIAL LINEAR DECOMPRESSOR

Num. Scan Chains	Num. Test Cubes	Without Scan Inversion			With Scan Inversion			
		Tester Channels	Percentage Specified	Encoding Efficiency	Reduced Channels	Encoding Efficiency	Increased % Specified	Encoding Efficiency
512	200	16	3.0 %	0.94	13	1.13	3.6 %	1.13
	400	16	3.0 %	0.94	14	1.07	3.3 %	1.04
	600	16	3.0 %	0.94	15	1.00	3.2 %	1.01
	1000	16	3.0 %	0.94	16	0.94	3.1 %	0.98
1024	200	16	1.5 %	0.94	11	1.37	2.1 %	1.32
	400	16	1.5 %	0.94	13	1.16	1.9 %	1.20
	600	16	1.5 %	0.94	14	1.07	1.8 %	1.13
	1000	16	1.5 %	0.94	15	1.00	1.7 %	1.07

TABLE V
EXECUTION TIME FOR SCAN INVERSION ON SEQUENTIAL LINEAR DECOMPRESSOR

Num Scan Chains		512				1024			
Num Test Cubes		200	400	600	1000	200	400	600	1000
Without Inversion		63.8s	149.1s	199.3s	366.8s	80.0s	161.5s	202.2s	370.5s
Inversion	Phase-I	162.5s	387.5s	788.6s	2170.4s	265.9s	651.5s	1153.2s	2486.1s
	Phase-II	673.8s	631.7s	10.9s	37.8s	1581.6s	1670.4s	1300.2s	845.0s
Partitioning	Phase-I	78.4s	177.7s	305.5s	510.6s	49.1s	238.8s	400.9s	777.4s
	Phase-II	98.3s	52.5s	1.8s	7.6s	680.1s	620.2s	657.0s	169.7s

TABLE VI
COMPARISON OF TEST DATA FOR DIFFERENT ENCODING SCHEMES

Circuit	Mintest [37]		FDR Codes [38]		Mutation Enc. [39]		Seed Over-lapping [18]		Sequential Decompressor		Seq. Decomp. With Inversion	
	Num. Vect.	Total Bits	Num. Vect.	Total Bits	Num. Vect.	Total Bits	Num. Vect.	Total Bits	Num. Vect.	Total Bits	Num. Vect.	Total Bits
s13207	233	163,100	236	30,880	274	16,913	272	17,970	266	15,020	266	9,708
s15850	96	58,656	126	26,000	185	14,676	174	15,774	226	16,153	226	10,726
s38417	68	113,152	99	93,466	231	55,848	288	60,684	105	50,365	105	36,864
s38584	110	161,040	136	77,812	220	47,886	215	31,061	192	38,192	192	27,555

compare the results with previously published techniques. In this case, a scan architecture with 64 scan chains was assumed and an LFSR of size 64 was used. Table VI shows the results obtained by the proposed method along with other test data compression schemes. The first column shows the circuit name and the next two columns are the number of vectors and the total test size of the dynamically compacted test cubes generated by MINTEST [37]. The next six columns show the number of test vectors and the compressed test size for frequency directed run-length codes [38], mutation encoding scheme [39], and seed overlapping scheme [18]. The next four columns show results for the sequential decompressor of Fig. 18 without and with scan inversion, respectively. As can be seen from the results, scan inversion substantially reduces the tester storage requirements compared to using the sequential decompressor without scan inversion.

VIII. CONCLUSION

Two methods for improving the compression of linear compression schemes, scan inversion, and reconfiguration of the decompressor, have been proposed in this paper. A systematic procedure based on linear algebra was described for selecting the set of inverted scan cells. Experimental results show that scan inversion can dramatically improve the encoding efficiency of combinational linear decompressors bringing it close to that of sequential decompressors. Scan inversion can also significantly improve the encoding efficiency for sequential linear decompressors. Scan inversion can be implemented with no hardware overhead.

The reconfiguration of a linear decompressor is represented as a constrained Boolean matrix and a symbolic Gaussian elimination method is proposed to solve it. Reconfiguration requires very little hardware. Experimental results show that compression obtained using a linear decompressor can be significantly improved using reconfiguration.

REFERENCES

[1] A. Khoche and J. Rivoir, "I/O bandwidth bottleneck for test: Is it real," in *Proc. Int. Workshop Test Resource Partitioning*, 2000, pp. 2.3-1–2.3-6.
[2] B. Könemann, "LFSR-coded test patterns for scan designs," in *Proc. Eur. Test Conf.*, 1991, pp. 237–242.
[3] S. Hellebrand, S. Tarnick, J. Rajski, and B. Courtois, "Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers," in *Proc. Int. Test Conf.*, 1992, pp. 120–129.
[4] N. Zacharia, J. Rajski, and J. Tyszer, "Decompression of test data using variable-length seed LFSRs," in *Proc. VLSI Test Symp.*, 1995, pp. 426–433.
[5] N. Zacharia, J. Rajski, J. Tyszer, and J. Waicukauski, "Two dimensional test data decompressor for multiple scan designs," in *Proc. Int. Test Conf.*, 1996, pp. 186–194.

[6] J. Rajski, J. Tyszer, and N. Zacharia, "Test data decompression for multiple scan designs with boundary scan," *IEEE Trans. Comput.*, vol. 47, no. 11, pp. 1188–1200, Nov. 1998.
[7] C. V. Krishna, A. Jas, and N. A. Touba, "Test vector encoding using partial LFSR reseeding," in *Proc. Int. Test Conf.*, 2001, pp. 885–893.
[8] C. V. Krishna and N. A. Touba, "Reducing test data volume using LFSR reseeding with seed compression," in *Proc. Int. Test Conf.*, 2002, pp. 321–330.
[9] I. Hamzaoglu and J. H. Patel, "Reducing test application time for full scan embedded cores," in *Proc. Int. Symp. Fault Tolerant Comput.*, 1999, pp. 260–267.
[10] I. Bayraktaroglu and A. Orailoglu, "Test volume and application time reduction through scan chain concealment," in *Proc. Des. Autom. Conf.*, 2001, pp. 151–155.
[11] ——, "Decompression hardware determination for test volume and time reduction through unified test pattern compaction and compression," in *Proc. VLSI Test Symp.*, 2003, pp. 113–118.
[12] S. Mitra and K. Kim, "XMAX: X-tolerant architectures for maximal test compression," in *Proc. Int. Conf. Comput. Des.*, 2003, pp. 326–330.
[13] C. Krishna and N. Touba, "Adjustable width linear combinational scan vector decompression," in *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, 2003, pp. 863–866.
[14] A. Jas, B. Pouya, and N. Touba, "Virtual scan chains: A means for reducing scan length in cores," in *Proc. VLSI Test Symp.*, 2000, pp. 73–78.
[15] B. Könemann, "A SmartBIST variant with guaranteed encoding," in *Proc. Asian Test Symp.*, 2001, pp. 325–330.
[16] J. Rajski *et al.*, "Embedded deterministic test," *IEEE Trans. Comput.-Aided Des. Integr. Circuit Syst.*, vol. 23, no. 5, pp. 776–792, May 2004.
[17] E. Volkerink and S. Mitra, "Efficient seed utilization for reseeding based compression," in *Proc. VLSI Test Symp.*, 2003, pp. 232–237.
[18] W. Rao, I. Bayraktaroglu, and A. Orailoglu, "Test application time and volume compression through seed overlapping," in *Proc. Des. Autom. Conf.*, 2003, pp. 732–737.
[19] C. Krishna and N. Touba, "3-stage variable length continuous-flow scan vector decompression scheme," in *Proc. VLSI Test Symp.*, 2004, pp. 79–86.
[20] K. J. Balakrishnan and N. Touba, "Improving encoding efficiency for linear decompressors using scan inversion," in *Proc. IEEE Int. Test Conf.*, 2004, pp. 936–943.
[21] ——, "Reconfigurable linear decompressors using symbolic Gaussian elimination," in *Proc. Des. Autom. Test Eur.*, 2005, pp. 1130–1135.
[22] C. Cullen, *Linear Algebra With Applications*. Reading, MA: Addison-Wesley, 1997.
[23] N. Touba and E. McCluskey, "Transformed pseudo-random patterns for BIST," in *Proc. IEEE VLSI Test Symp.*, 1995, pp. 410–416.
[24] ——, "Synthesis of mapping logic for generating transformed pseudo-random patterns for BIST," in *Proc. IEEE Int. Test Conf.*, 1995, pp. 674–682.
[25] ——, "Bit-fixing in pseudo-random sequences for scan BIST," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 20, no. 4, pp. 545–555, Apr. 2001.
[26] M. Chatterjee and D. Pradhan, "A new pattern biasing technique for BIST," in *Proc. VLSI Test Symp.*, 1995, pp. 417–425.
[27] H.-J. Wunderlich and G. Kiefer, "Bit-flipping BIST," in *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, 1996, pp. 337–343.
[28] G. Kiefer and H.-J. Wunderlich, "Using BIST control for pattern generation," in *Proc. Int. Test Conf.*, 1997, pp. 347–355.
[29] ——, "Deterministic BIST with multiple scan chains," in *Proc. Int. Test Conf.*, 1998, pp. 1057–1064.
[30] L. Li and K. Chakrabarty, "Deterministic BIST based on a reconfigurable interconnection network," in *Proc. Int. Test Conf.*, 2003, pp. 460–469.
[31] A. Al-Yamani and E. J. McCluskey, "Built-in reseeding for serial BIST," in *Proc. VLSI Testing Symp.*, 2003, pp. 63–68.
[32] K.-H. Tsai, J. Rajski, and M. Marek-Sadowska, "Scan encoded test pattern generation for BIST," in *Proc. IEEE Int. Test Conf.*, 1997, pp. 548–556.
[33] I. Bayraktaroglu and A. Orailoglu, "Test power reduction through minimization of scan chain transitions," in *Proc. VLSI Testing Symp.*, 2002, pp. 166–171.
[34] H. Tang, S. Reddy, and I. Pomeranz, "On reducing test data volume and test application time for multiple scan chain designs," in *Proc. Int. Test Conf.*, 2003, pp. 1079–1088.
[35] K. A. B. , J. Hewitt, and N. Nicolici, "Embedded compact deterministic test for IP-protected cores," in *Proc. Int. Symp. Defect Fault Tolerance*, 2003, pp. 519–526.
[36] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. Int. Symp. Circuits Syst.*, 1989, pp. 1929–1934.
[37] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits," in *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, 1998, pp. 283–289.
[38] A. Chandra and K. Chakrabarty, "Frequency-Directed Run length (FDR) codes with application to system-on-a-chip test data compression," in *Proc. VLSI Test Symp.*, 2001, pp. 42–47.
[39] S. Reda and A. Orailoglu, "Reducing test application time through test data mutation encoding," in *Proc. Des., Autom. Test Eur.*, 2002, pp. 387–393.
[40] K. J. Balakrishnan, "New approaches and limits to test data compression for systems-on-chip," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Texas at Austin, Austin, 2004.

Kedarnath J. Balakrishnan (M'00) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Bombay, India, in 2000, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Texas at Austin, Austin, in 2002 and 2004, respectively.

He is currently a Research Staff Member with NEC Laboratories America, Princeton, NJ, where he is leading System LSI Test projects in both gate level and RTL designs. His research has focused on test volume reduction techniques using test compression and built-in self-test (BIST), system-on-a-chip (SoC) testing issues, and design for test techniques for delay testing.

Nur A. Touba (M'92) received the B. S. degree from the University of Minnesota, Minneapolis, in 1990, and the M.S. and Ph.D. degrees from Stanford University, Stanford, CA, in 1991 and 1996, respectively, all in electrical engineering.

He is currently an Associate Professor in the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin.

Dr. Touba was a recipient of the National Science Foundation (NSF) Early Faulty CAREER Award in 1997 and the Best Paper Award at the 2001 VLSI Test Symposium. He is on the program committee for the International Test Conference, International Conference on Computer Design, Design Automation, and Test in Europe Conference, International On-Line Test Symposium, European Test Symposium, Asian Test Symposium, Defect and Fault Tolerance Symposium, Microprocessor Test and Verification Workshop, International Workshop on Open Source Test Technology Tools, and International Test Synthesis Workshop.