

# X-filter: Filtering unknowns from compacted test responses

Manish Sharma and Wu-Tung Cheng

Mentor Graphics Corporation

8005 SW Boeckman Road

Wilsonville, OR 97070

Email: {manish\_sharma,wu-tung\_cheng}@mentor.com

**Abstract**—Using off-the-shelf error correcting codes for compacting test response [1] is attractive because it provides multiple error detection and diagnosis support in the presence of multiple unknowns. However, this technique is not easily incorporated in a conventional scan testing environment because handling unknowns requires special ATE support or test response post-processing to determine if the test passed. In this paper we solve this problem using a novel technique, *X-filter*, which removes the effect of unknowns on compacted test response. *X-filter* achieves this by using only  $O(mx)$  additional hardware with  $mx$  inputs. ( $x$ =maximum number of unknowns that can be tolerated,  $m$ =number of bits in the compacted response). Unlike compaction methods that require creation of special codes [2], [3], *X-filter* is more flexible in terms of number of errors and unknowns handled, and it does not increase the number of output pins over those used by the error correcting code itself.

## I. INTRODUCTION

Full scan is one of the most widely used DFT technique, which enables high test qualities through the use of ATPG tools to generate test vectors. However, the cost of scan testing is becoming prohibitive because the number of scan flops in designs is increasing while the number of chip pins and tester channels that can be used for scanning in and scanning out are limited. This leads to longer scan chains, which implies larger test data volumes and longer chain load-unload times. To overcome this several test data compression techniques break long scan chains into a large number of short chains. To access these short chains using a relatively small number of pins, a test data decompression scheme is used on the input side [18]–[26] and on the output side a test response compactor [1]–[13] is used to reduce the number of test output bits as shown in Figure 1. The topic of discussion of this paper will be space compaction techniques which take the  $n$ -bit test response coming out of  $n$  scan chains every shift cycle and compact it to  $m$  ( $m < n$ ) bits before sending it out to the tester.

Several different space compaction methods have been recently proposed in the literature [1]–[3]. All of these use error correcting codes (ECC) in one way or another to achieve compaction while maintaining the capability of detecting and correcting errors even in the presence of unknowns in the test response [1], [14]. The use of ECC is very attractive because: (1) It is based on the well understood digital communication theory (2) provides good compaction ratios (3) enables the

detection and diagnosis of multiple errors in the presence of unknown output responses and (4) is design and test vector independent. However, in order to use off-the-shelf ECC in the presence of unknowns we need enhanced ATE support, e.g. for each scan out cycle we might need to compare the compacted response to several different possible correct responses. This is not easily done in a conventional scan test environment. Alternatively, it may require post-processing of test responses in order to determine if the test passed or failed. This is problematic in go/no-go testing, and also increases the test time since storing test responses on a tester is slower than just comparing responses on chip pins.

In order to overcome the limitations above we need to expend more on chip as well as tester data resources. One way to do this is to create specialized codes that ensure the segregation of the unknowns from the error information in the compacted response [2], [3], [15]. However creating these specialized codes is not easy, especially for multiple unknowns. In [2] special codes to handle only one unknown are presented. Creating codes for handling more than one unknown requires solving an NP-complete problem. Moreover, these codes require many more output pins over the off-the-shelf ECC which not only increases the output response data storage requirement but also the more expensive capture memory requirements on the ATE. Having additional output pins mainly for handling unknowns is an overkill because in real circuits only a fraction of the shift cycles may contain unknowns.

Another way to achieve the above objective is to mask the output response bits that are unknowns before compaction [16]. This method does not increase the number of output pins, but additional inputs are required to pass the information on location of the Xs in the test response to the masking

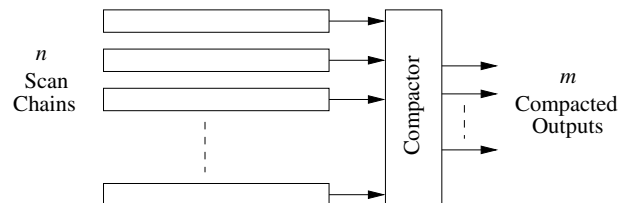


Fig. 1. Space compaction of output response

hardware. So, in order to mask up to  $x$  unknowns the masking logic will require  $x \log n$  inputs. Once the Xs are masked the now deterministic test response can be compacted using off-the-shelf error correcting codes. However, since masking is done before the test response is compacted, it requires  $O(nx)$  hardware, which can be very expensive.

In this paper we present a novel compaction method, *X-filter*, which removes the dependence of the compacted output response obtained using off-the-shelf ECC on the value of unknowns in the test response in a way that preserves the error information in the compacted output. So X-filter allows the use of any ECC for compaction in a standard scan test flow without requiring any special ATE support. At the same time it preserves all the advantages of using ECC for compaction listed above. In order to achieve these goals, X-filter requires the use of  $m \times x$  additional inputs ( $x$  is the maximum number of unknowns that need to be tolerated in one scan shift cycle) and  $O(m \times x)$  additional hardware. Now, for good ECC  $m$  is typically  $O(\log n)$ , so X-filter achieves the purpose of getting rid of the unknowns without using additional output pins, as in [2], [3], and without requiring too much extra hardware, as in masking. We make two comments about the requirement of  $m \times x$  inputs to the X-filter hardware: firstly, the input data required for the X-filter will have a lot of redundancy since not all shift cycles may have multiple unknowns or any unknowns at all, so this data can be stored in a compressed form on the ATE and decompressed using on chip hardware. Thus the actual number of external input pins required for X-filter can be much less than  $m \times x$ . Secondly, for some special cases, e.g. using the well known Hamming codes, the requirement of  $m \times x$  inputs to the X-filter hardware can be reduced even further.

The rest of this paper is organized as follows: Section II gives a brief background on the use of error correcting codes for test response compaction. The main idea behind X-filter is developed with the help of an example in Section III and Section IV describes the implementation of X-filter. In Section V we describe the special X-filter cases that lend themselves to more efficient implementation. The results are summarized in Section VI and we conclude the paper in Section VII.

## II. THE USE OF ERROR CORRECTING CODES FOR TEST RESPONSE COMPACTION

### A. Basic theory

In this section we briefly describe the concept behind using ECC for compaction. The reader is also referred to the excellent description in the original paper [1]. Here, we give a slightly different description as opposed to the one given in [1]. In what follows we use the following convention for the use of symbols: uppercase bold face symbols denote matrices and sets, lowercase bold face symbols denote vectors and non-bold face symbols are used to denote scalar variables.

A space compactor of test response takes an  $n$ -bit test response, denoted by an  $n \times 1$  vector  $\mathbf{t}$ , out of  $n$ -scan chains at the end of each shift-out cycle, and compacts it into an  $m$ -bit output vector, denoted by a  $m \times 1$  vector  $\mathbf{s}$  (Figure 1).

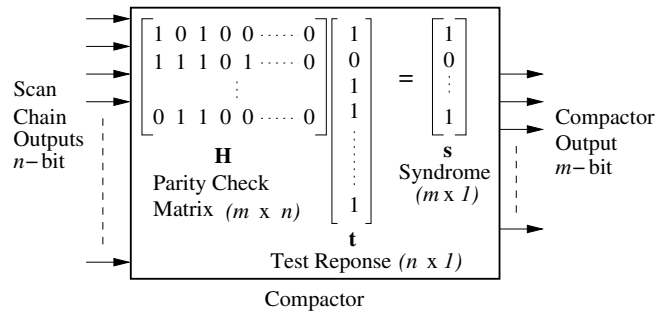


Fig. 2. Using error correcting codes for compaction

In other words the compactor exhaustively partitions the set of all possible  $n$ -bit binary vectors into  $2^m$  sets, denoted by  $\mathbf{T}_i$  ( $0 \leq i < 2^m$ ). A set  $\mathbf{T}_i$  contains all those and only those test responses which are mapped to the same  $m$ -bit compacted output vector. Now, if the compaction scheme is to allow the detection of up to  $e$  errors in the presence of up to  $x$  unknowns in a scanout shift cycle, every pair of test response vectors in the same set must be at least distance  $d$  apart where  $d > e + x$ . In other words each of the sets  $\mathbf{T}_i$  is a distance  $d$ , binary error correcting code.

Based on the above, we can use a class of error correcting codes called binary linear block codes to construct a compaction scheme as described below. In this context, an  $(n, k, d)$  binary linear code is defined as a set of  $2^k$   $n$ -bit vectors (called codewords), such that the minimum distance between any two codewords is  $d$ . The first set,  $\mathbf{T}_0$ , of test response vectors is chosen to be an appropriate  $(n, k, d)$  code (where  $n - k = m$ ) and the cosets of this code form the remaining  $2^m - 1$  sets. A coset (or a translate) of an  $(n, k, d)$  linear code is defined as the set of vectors obtained by adding a fixed  $n$ -bit vector  $\mathbf{a}$  to all the codewords [17]. Note, that a coset is also a set of  $2^k$   $n$ -bit vectors with minimum distance  $d$ . One way to define a code and its cosets is through its parity check matrix,  $\mathbf{H}$ . A parity check matrix of an  $(n, k, d)$  code is defined as an  $(n - k) \times n$  matrix with rank  $(n - k)$  such that an  $n$ -bit vector  $\mathbf{c}$  is a codeword iff  $\mathbf{H} \cdot \mathbf{c} = 0$ . For an  $n$ -bit vector  $\mathbf{a}$ ,  $\mathbf{H} \cdot \mathbf{a}$  is also called the syndrome of  $\mathbf{a}$ , denoted by  $S(\mathbf{a})$ .

*Theorem 2.1:* Two  $n$ -bit vectors,  $\mathbf{a}$  and  $\mathbf{b}$  belong to the same coset iff their syndromes are equal.

*Proof:* See any standard coding theory text, e.g. [17] •

Based on this it is easy to see that the syndrome of a test response vector can be used as the compacted output. This summarizes the basic theory behind using ECC for compaction (Figure 2). The following theorem is proved about using ECC for compaction in [1]:

*Theorem 2.2:* If a distance  $d$  code is used to implement I-compaction, then:

- 1) Upto  $d - 1$  single bit errors in a single shift cycle can be detected.
- 2) Upto  $e$  single bit errors can be detected in the presence of  $x$  unknowns, where  $e + x < d$ .
- 3) Upto  $t$  errors can be corrected in the presence of  $x$  unknowns, where  $2t + x < d$ .

### B. Main limitation of using off-the-shelf error correcting codes for test response compaction

Even though off-the-shelf ECC achieve good compaction ratios and allows error detection and correction in the presence of multiple unknowns, incorporating them in a conventional test environment is hard because it requires special ATE support. The reason for this is explained through the following example. Assume that we are using the (8,4,4) extended hamming code for compaction. This code will compact the output of 8 scan chains into 4 bits and it will allow the detection of upto 3 –  $x$  errors in the presence of  $x$  unknowns and correction of one error in the presence of one unknown. A parity check matrix for this code is:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (1)$$

Now consider a test response vector of<sup>1</sup>:

$$\mathbf{t}' = [ 0 \ 0 \ 0 \ X_0 \ X_1 \ 0 \ 0 \ 0 ] \quad (2)$$

where  $X_0$  and  $X_1$  are unknown values. For this test response, the four bit compacted output vector,  $\mathbf{s}' = [s_0 s_1 s_2 s_3]$  will be:

$$\begin{aligned} s_0 &= X_1 \\ s_1 &= X_0 \\ s_2 &= X_0 \\ s_3 &= X_0 + X_1 \end{aligned} \quad (3)$$

This means that, depending on the actual values of the unknowns on silicon, the output compacted vector can have four different correct values: 0000, 0111, 1001 and 1110. In other words, on an ATE we will have to compare the compacted output to four different expected values in order to ascertain that the test passed. This is hard to implement on a conventional tester. As another implementation strategy, the compacted output from a chip under test can be stored on the tester and then this data can be post-processed after the test is complete to determine if it passed or not. Such a strategy also has several disadvantages: this does not fit into the go/no-go kind of production test environment, storing data on the tester during test slows down the overall test and only a limited amount of data can be stored on the tester. This is further complicated during wafer sort where the test output is not offloaded from the tester until all the dies have been tested, which means that a lot of output data will have to be stored on the tester. Finally, both of the above implementation strategies are not suitable for the case where multiple chips are being tested in parallel using one tester.

X-filter, the new technique described in this paper overcomes this limitation at the expense of  $O(mx)$  additional hardware which requires  $mx$  inputs. The way this is done is the topic of the next section.

<sup>1</sup>The ' indicates transpose

## III. X-FILTER THEORY

### A. Basic idea

The main limitation with using off-the-shelf ECC for compaction is that the unknowns in the test output response cause the compacted output to take on more than one "correct" value, which is hard to handle on conventional testers. In order to overcome this, in X-filter the compacted output is processed in a way that its dependence on value of the unknowns is removed. We illustrate how this is done by continuing on the example presented in Section II-B. Any test response vector  $\mathbf{t}$  can be broken into three components:

$$\mathbf{t} = \mathbf{t}_k + \mathbf{t}_x + \mathbf{t}_e$$

The vector  $\mathbf{t}_k$  contains the known bits in  $\mathbf{t}$  and zeros in locations corresponding to the unknown bits. The vector  $\mathbf{t}_x$  contains the unknown bits in  $\mathbf{t}$  and zero in all other locations. The last component  $\mathbf{t}_e$  contains a 1 in all those locations in which there is an error and zero everywhere else. Now, as in the example in Section II-B, consider a case where the fourth and fifth bits in the test response are unknown and there is an error in the seventh bit location. The components of this test response can be written as:

$$\begin{aligned} \mathbf{t}_k' &= [ d_0 \ d_1 \ d_2 \ 0 \ 0 \ d_5 \ d_6 \ d_7 ] \\ \mathbf{t}_x' &= [ 0 \ 0 \ 0 \ X_0 \ X_1 \ 0 \ 0 \ 0 ] \\ \mathbf{t}_e' &= [ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ e_0 \ 0 ] \end{aligned} \quad (4)$$

Based on the above and using the parity check matrix from Equation(1), the output compacted bits can be written as:

$$\begin{aligned} s_0 &= d_0 + d_1 + d_2 && +X_1 \\ s_1 &= d_0 + d_1 + d_5 && +X_0 \\ s_2 &= d_0 + d_2 + d_6 && +e_0 \ +X_0 \\ s_3 &= d_0 + d_1 + d_2 + d_5 + d_6 + d_7 && +e_0 \ +X_0 + X_1 \\ \mathbf{s} &= \mathbf{s}_k && +\mathbf{s}_e \ +\mathbf{s}_x \end{aligned}$$

As suggested above, analogous to the test response vector, we can break the compacted vector into three components: due to the known bits, unknown bits and the error. Now, if we just focus on the dependence of the compacted bits on the unknowns ( $\mathbf{s}_x$ ) then we get four linear expressions in  $X_0$  and  $X_1$ . It is easy to see that the first two of these expressions are linearly independent and the last two are linearly dependent on the first two. In other words, the last two expressions can be expressed as linear combinations of the first two expressions. We will use these relationships to remove the dependence of  $\mathbf{s}$  on the unknowns in the following way:

- 1) Ignore those bits in  $\mathbf{s}$  that correspond to the *linearly independent* expressions in  $\mathbf{s}_x$ .
- 2) To the bits in  $\mathbf{s}$  that correspond to the *linearly dependent* expressions in  $\mathbf{s}_x$ , add appropriate linear combinations of the bits that correspond to linearly independent expressions so as to get rid of the unknowns.

The above two steps define the main operations in X-filter. Hence, for our example we would obtain the X-filtered com-

packed output,  $\mathbf{s}_t$ , as follows:

$$\mathbf{s}_t = \begin{bmatrix} 0 \\ 0 \\ s_2 + s_1 \\ s_3 + s_0 + s_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ d_1 + d_2 + d_5 + d_6 + e_0 \\ d_0 + d_1 + d_6 + d_7 + e_0 \end{bmatrix}$$

The following observations can be made from the above equations:

- 1) The bits in  $\mathbf{s}_t$  do not depend on the value of the unknowns.
- 2) The error bit is preserved in the last two bits of  $\mathbf{s}_t$ , i.e. the error will be detected in  $\mathbf{s}_t$ . In fact the reader can easily see that any single bit error in the known bits of the test response can be detected in  $\mathbf{s}_t$ .

As another example consider a test response with one unknown in the third bit-location:

$$\begin{aligned} \mathbf{t}_k' &= [ d_0 \ d_1 \ 0 \ d_3 \ d_4 \ d_5 \ d_6 \ d_7 ] \\ \mathbf{t}_x' &= [ 0 \ 0 \ X_0 \ 0 \ 0 \ 0 \ 0 \ 0 ] \end{aligned} \quad (5)$$

In this case the compacted bits out of the ECC will be:

$$\begin{aligned} s_0 &= d_0 + d_1 + d_4 && + X_0 \\ s_1 &= d_0 + d_1 + d_3 + d_5 \\ s_2 &= d_0 + d_3 + d_6 && + X_0 \\ s_3 &= d_0 + d_1 + d_3 + d_4 + d_5 + d_6 + d_7 && + X_0 \end{aligned}$$

So, for this case the first expression in  $\mathbf{s}_x$  is linearly independent while the remaining three are linearly dependent on the first. Therefore, X-filtering on  $\mathbf{s}$  results in:

$$\mathbf{s}_t = \begin{bmatrix} 0 \\ s_1 \\ s_2 + s_0 \\ s_3 + s_0 \end{bmatrix} = \begin{bmatrix} 0 \\ d_0 + d_1 + d_3 + d_5 \\ d_1 + d_3 + d_4 + d_6 \\ d_3 + d_5 + d_6 + d_7 \end{bmatrix}$$

Now, let us say that the two bits  $d_0$  and  $d_1$  are in error then this error will be detected in the third bit of  $\mathbf{s}_t$ . As another example, let us say that the fourth bit of  $\mathbf{s}_t$  shows an error, then we can uniquely attribute it to  $d_7$ , assuming a single bit error. In fact the reader can easily reason that any two bit errors can be detected and any single bit error can be corrected. Finally, note that all the observations in both of these examples are in accordance with Theorem 2.2.

Figure 3 summarizes the block level functionality of X-filter. The detailed implementation of the X-filter block is described in Section IV, however before that, in the next sub-section we develop the material presented above more formally and prove that it works.

### B. Formal development of X-filter theory

In this sub-section we do a formal presentation of the X-filter process presented in the previous sub-section and prove some key results related to it. First, we define a  $(m \times x)$  matrix  $\mathbf{H}_x$  which contains all the columns of the parity check matrix  $\mathbf{H}$  corresponding to the locations of the unknowns in the test response. Second, we define a  $(m \times m)$  matrix called the X-filter matrix, denoted by  $\mathbf{DX}(\mathbf{H}_x)$ . This matrix is a function of the unknown locations in the test response. The  $i$ -th column of

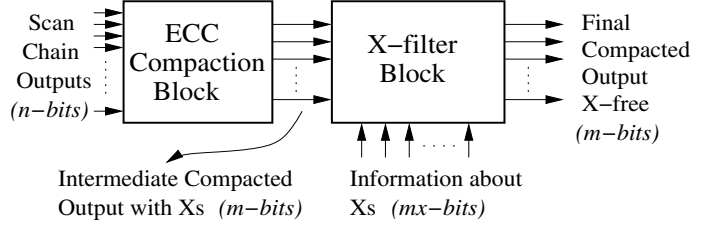


Fig. 3. Working of compaction with X-filter

$\mathbf{DX}(\mathbf{H}_x)$  corresponds to the  $i$ -th bit of the compacted vector,  $s_i$  and row  $i$  of  $\mathbf{DX}(\mathbf{H}_x)$  describes the X-filter step for  $s_i$ . The way this matrix is constructed is as follows: For a given locations of the unknowns, determine which rows in  $\mathbf{H}_x$  are linearly independent. The rows of  $\mathbf{DX}(\mathbf{H}_x)$  corresponding to the linearly independent rows are all zeros. The rows of  $\mathbf{DX}(\mathbf{H}_x)$  corresponding to the remaining linearly dependent rows in  $\mathbf{H}_x$  define the linear dependence relation for those rows. As an example consider the test response given in Equation(4). The X-filter matrix for these locations of the unknowns will be:

$$\mathbf{H}_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}; \mathbf{DX}(\mathbf{H}_x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

As another example the X-filter matrix for the test response in Equation(5):

$$\mathbf{H}_x = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}; \mathbf{DX}(\mathbf{H}_x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

The construction of the X-filter matrix, and the X-filter step itself, depends on the existence of  $x$  linearly independent expressions in  $\mathbf{H}_x$ , where  $x$  is the number of unknowns in the corresponding test response. The structure of the code guarantees that this will always be true for all  $x < d$  because if it were not true for some location of the unknowns, we would have a situation where a distance- $d$  code cannot correct  $x < d$  erasures in a codeword, which is a contradiction.

Using the X-filter matrix, the X-filtering process on a compacted output vector  $\mathbf{s}$  can be written as:

$$\mathbf{s}_t = \mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{s}$$

Now we know that:

$$\mathbf{s} = \mathbf{H} \cdot \mathbf{t} = \mathbf{H} \cdot \mathbf{t}_k + \mathbf{H} \cdot \mathbf{t}_e + \mathbf{H} \cdot \mathbf{t}_x$$

so, we get:

$$\mathbf{s}_t = \mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot \mathbf{t}_k + \mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot \mathbf{t}_e + \mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot \mathbf{t}_x$$

Since, by construction,

$$\mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot \mathbf{t}_x = \mathbf{0}$$

we have:

$$\mathbf{s}_t = \mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot \mathbf{t}_k + \mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot \mathbf{t}_e$$

So, at this point we have established that when the number of unknowns is less than  $d$ , a X-filter matrix can be constructed for any unknown locations. This X-filter matrix can be used to produce a compacted output response,  $s_t$ , which is independent of the unknown values. Next, using the above setup we prove that  $s_t$  allows the same error correction and error detection capabilities as described in Theorem 2.2.

*Lemma 3.1:* For a given location of unknowns in a test response with the number of unknowns  $x < d$ , if  $\mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{a} = 0$  for a  $m$ -bit vector, then there exists a value assignment,  $\mathbf{t}_x$ , to the unknowns such that  $\mathbf{H} \cdot \mathbf{t}_x = \mathbf{a}$ .

*Proof:* We prove this by constructing a vector  $\mathbf{t}_x$  which satisfies the statement. With the unknowns as variables in  $\mathbf{t}_x$ ,  $\mathbf{H} \cdot \mathbf{t}_x$  represents a set of  $m$  linear equations in  $x$  unknowns. As argued previously out of these  $m$  equations there will be  $x$  linearly independent equations with the remaining  $m - x$  equations linearly dependent on these  $x$  equations. So, we can solve the  $x$  linearly independent equations for values of the unknowns, to obtain the corresponding  $x$  bits in  $\mathbf{a}$ . Since  $\mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{a} = 0$ , keeping the construction of  $\mathbf{DX}(\mathbf{H}_x)$  in mind, if we plug these values of the unknowns in the remaining  $m - x$  linearly dependent equations we will get the remaining  $m - x$  bits of  $\mathbf{a}$ . This completes the proof. •

*Lemma 3.2:* For a test response with  $x$  unknowns and  $e$  errors, where  $e + x < d$ ,  $\mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot \mathbf{t}_e$  is non-zero.

*Proof:* Assume that  $\mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot \mathbf{t}_e = 0$ . Then, from Lemma 3.1, there exists a value assignment to the unknowns,  $\mathbf{t}_x$  such that  $\mathbf{H} \cdot \mathbf{t}_x = \mathbf{H} \cdot \mathbf{t}_e$ . From Theorem 2.1 this implies that  $\mathbf{t}_x$  and  $\mathbf{t}_e$  belong to the same coset which is a contradiction since the distance between  $\mathbf{t}_x$  and  $\mathbf{t}_e$  is less than  $d$ . •

*Lemma 3.3:* For a test response with  $x$  unknowns and  $t$  errors, where  $2t + x < d$ , and for two different error patterns,  $\mathbf{t}_{e1}$  and  $\mathbf{t}_{e2}$ ,  $\mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot \mathbf{t}_{e1} \neq \mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot \mathbf{t}_{e2}$

*Proof:* Assume that  $\mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot \mathbf{t}_{e1} = \mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot \mathbf{t}_{e2}$ . This implies:  $\mathbf{DX}(\mathbf{H}_x) \cdot \mathbf{H} \cdot (\mathbf{t}_{e1} + \mathbf{t}_{e2}) = 0$ . So, from Lemma 3.1, there exists a value assignment to the unknowns,  $\mathbf{t}_x$  such that  $\mathbf{H} \cdot \mathbf{t}_x = \mathbf{H} \cdot (\mathbf{t}_{e1} + \mathbf{t}_{e2})$ , which means that  $\mathbf{t}_x$  and  $\mathbf{t}_{e1} + \mathbf{t}_{e2}$  belong to the same coset (From Theorem 2.1). However, since  $2t + x < d$ , the distance between  $\mathbf{t}_x$  and  $\mathbf{t}_{e1} + \mathbf{t}_{e2}$  is less than  $d$  and therefore we have a contradiction. •

Hence, to summarize, the X-filter step gets rid of the dependence of the compacted output on the value of the unknowns in the test response at the same time preserving the error correction and detection capabilities of ECC. In the next section we describe how this scheme can be implemented in hardware.

#### IV. IMPLEMENTATION OF THE X-FILTER

In order to execute X-filtering we need two sets of information for each scanout shift cycle:

- 1) Which rows of  $\mathbf{H}_x$  are linearly independent and which are linearly dependent?
- 2) What are the dependence relations of the linearly dependent rows on the linearly independent rows?

This information can be pre-computed and represented using  $m \times x$  bits as follows: For each row of  $\mathbf{H}_x$  an  $x$  bit vector is

determined which indicates whether the corresponding row is linearly independent or how it is dependent on the linearly independent rows. These  $x$ -bit vectors are determined using the following algorithm:

```

li_count = 0 /* LINEARLY INDEPENDENT ROWS
              FOUND COUNTER */
LI = {}      /* ARRAY TO STORE LINEARLY
              INDEPENDENT ROWS */
V = {}      /* ARRAY TO STORE m, x-BIT
              VECTORS THAT GO ON TESTER */
Ix = x x Identity Matrix
for ( row = 0 to m-1 )
  if (Hx[row] is Linearly Independent
      of rows in LI)
    LI[li_count] = Hx[row]
    V[row] = Ix[li_count]
    li_count++
  else
    V[row] = Array defining relation of
              Hx[row] to the Linearly
              Independent rows in LI

```

As an example a hypothetical  $\mathbf{H}_x$  for three unknowns and the corresponding vectors are given below:

$$\mathbf{H}_x = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}; \mathbf{V} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

So, the X-filter block of Figure 3 will use these  $m$ ,  $x$ -bit vectors to perform X-filtering on the output of the ECC based compactor. The X-filter block could be implemented in several ways, however we choose a iterative logic array based design which is very simple to understand and can easily be extended for any values of  $m$  and  $x$  by just using more cells. Iterative logic array based designs are based on relatively simple logic cells, which are stacked together to achieve the desired functionality. For our case we will use  $m$  identical

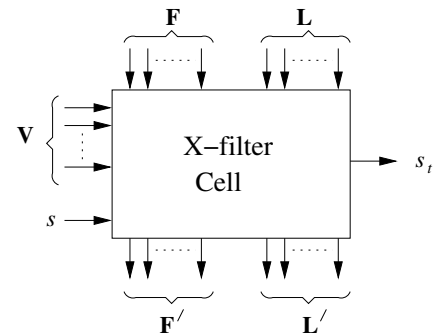


Fig. 4. IO of an X-filter Cell

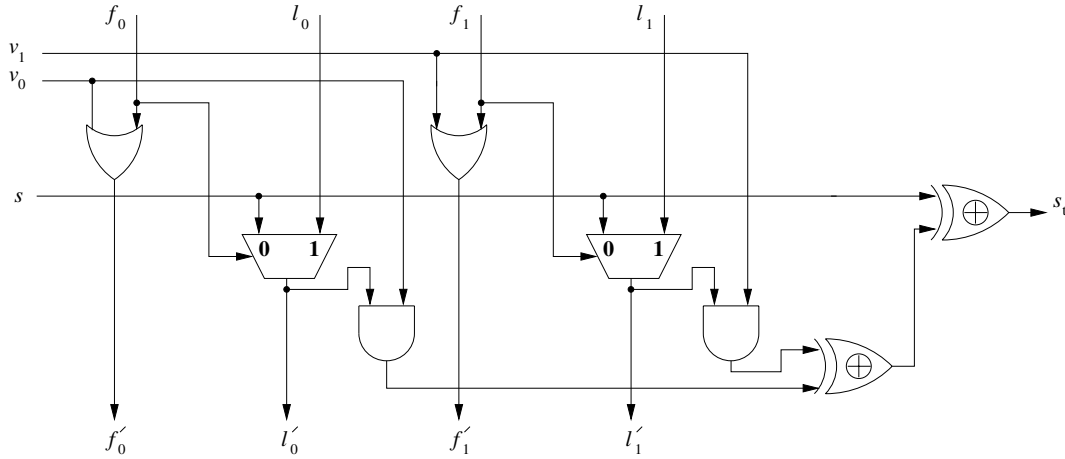


Fig. 6. Implementation of a cell in the X-filter block

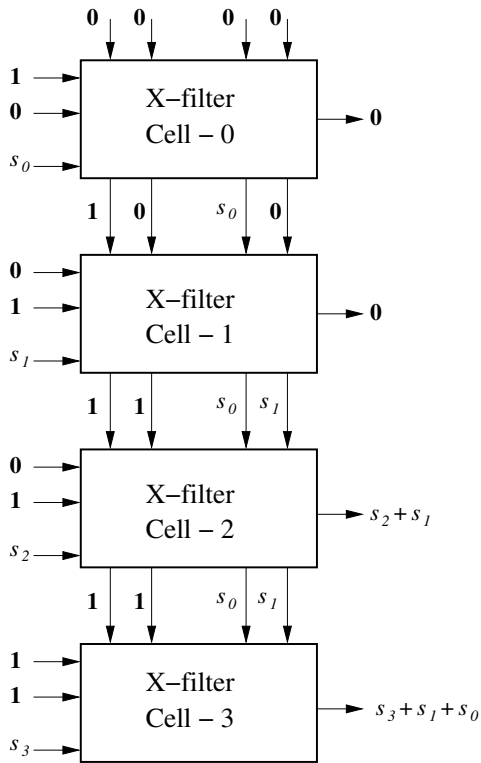


Fig. 5. Functioning of the X-filter block

cells, each of whose complexity is a function of  $x$ . As shown in Figure 4, each cell has two sets of  $x$ -bit inputs ( $\mathbf{F}$  and  $\mathbf{L}$ ), and, two sets of  $x$  bit outputs ( $\mathbf{F}'$  and  $\mathbf{L}'$ ) which feed into the next cell. In addition a cell also takes in as input one bit of the compacted output from the ECC Compactor block ( $s$ ) and the corresponding  $x$ -bit vector ( $\mathbf{V}$ ) and produces the X-filtered bit ( $s_t$ ). In order to understand how the cell functions we first describe the meaning of each of the sets of inputs and outputs. The input set  $\mathbf{F}$  are a set of flags which indicate uptill this cell what linearly independent rows have been encountered. So, the  $\mathbf{F}$  input to the very first cell is all zeros. When the very

first linearly independent row is encountered at a cell the bit (which is indicated by a  $100 \dots 0$  on the  $\mathbf{V}$  input) the  $f_0$  bit in  $\mathbf{F}$  goes from zero to one and remains at one for the subsequent cells. Similarly,  $f_1$  goes high when we encounter the second linearly independent row, and so on. The  $\mathbf{L}$  input set carries the values of the linearly independent equations discovered so far. So, again, the value of this input set for the very first cell is all zeros, and again, when the first linearly independent row is encountered the  $l_0$  bit in the  $\mathbf{L}$  takes on the value of the  $s$  input and so on. This functionality is shown in Figure 5 for the example of the test response given in Equation(4). For this test response,  $\mathbf{H}_x$  and  $\mathbf{V}$  are:

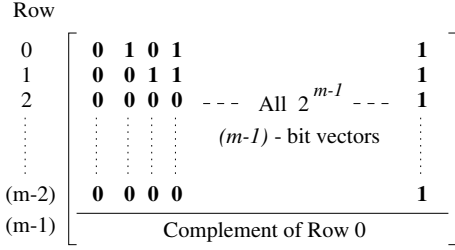
$$\mathbf{H}_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}; \mathbf{V} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Finally, based on the functionality description above, the logic inside the cell can be easily derived and is shown in Figure 6 for the case of two unknowns. The reader can easily verify that the logic performs the required functionality. Hence, the hardware requirements for the X-filter are:  $m \times 2$ -input OR gates,  $m \times 2$ -input AND gates,  $m \times 2$ -input XOR gates and  $m \times 2$ -input MUXes. Several optimizations to the above simple implementation of the X-filter cell are possible but in the general the hardware requirements are  $O(mx)$ .

## V. IMPLEMENTATIONS OF X-FILTER FOR SPECIAL CASES

In this section we describe two special cases which have more efficient implementations than the general case. In both these cases, the optimization results from some special properties that the codes used have. As the first example consider the construction of an X-filter for an extended Hamming code based compactor that can tolerate at most one unknown ( $x = 1$ ). One possible construction for an  $(2^{m-1}, m, 4)$  extended Hamming code is given below:

Since, the last row is the complement of the first row we know that if there is an unknown  $X_0$  in the test response,



$s_0 + s_{m-1}$  will always be dependent on  $X_0$ . Therefore, we just need  $m - 2$  additional inputs to the X-filter to tell it what other compacted bits are dependent on  $X_0$  so that it can be filtered out. Hence, for this special case we need two less inputs as compared to the general case.

As a second example we will use a special code to construct a compactor which can tolerate one, two or any odd number of errors when no unknowns are present and one error in the presence of one unknown. This example brings out a more generalized view of X-filtering outside of the framework presented above. The parity check matrix for the special code consists  $m$  rows where  $m$  is a multiple of 4, and two groups of columns:  $\mathbf{H}_o$  and  $\mathbf{H}_e$ . The group  $\mathbf{H}_e$  contains all  $m$ -bit vectors with  $m/2$  ones and the group  $\mathbf{H}_o$  contains all  $m$ -bit vectors with  $m/2 - 1$  ones. Therefore, the total number of columns for this code will be:

$$n = \binom{m}{m/2} + \binom{m}{m/2 + 1}$$

An example for  $m = 4$  is given below:

$$\mathbf{H} = [\mathbf{H}_o | \mathbf{H}_e] = \left[ \begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{array} \right]$$

Note that all the columns in  $\mathbf{H}_e$  have even number of ones and all the columns in  $\mathbf{H}_o$  have odd number of ones. It is a simple exercise to check that this code will detect one, two or any odd number of errors in the absence of unknowns. When one unknown is present there are two possible scenarios:

**Case(I):** The location of the unknown corresponds to a column in  $\mathbf{H}_o$ . In this case since the number of ones in the columns in  $\mathbf{H}_o$  is always one less than the number of ones in the columns in  $\mathbf{H}_e$  a single error in any location can be detected without any X-filtering.

**Case(II):** The location of the unknown corresponds to a column in  $\mathbf{H}_e$ . In this case errors in certain locations corresponding to columns in  $\mathbf{H}_o$  may get hidden behind the unknown and will not be detected at the output pins. As an example for the above parity matrix if the unknown location corresponds to the fifth column then an error in location corresponding to the first column or the second column will not be detected in the compacted output bits.

In order to remedy the situation in **Case(II)**, we construct an X-filter which takes the sum of all compacted output bits ( $s_{sum}$ ) and adds it to all the compacted output bits as shown in Figure 7 for the case of  $m = 4$ : The control input  $c$  is 1 when

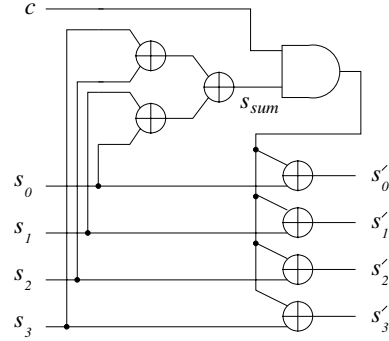


Fig. 7. X-filter for the special code for detecting one error in the presence of one unknown

there is an unknown with location corresponding to a column in  $\mathbf{H}_e$  and zero otherwise. This is the only input required for the X-filter. Now, when the unknown location corresponds to a column in  $\mathbf{H}_e$  the sum  $s_{sum}$  will be independent of the value of the unknown but dependent on an error in any location corresponding to a column in  $\mathbf{H}_o$ . Moreover, if the error location corresponds to a column in  $\mathbf{H}_o$ , there must exist at least one output bit which is independent of both the error and the unknown. So, an error which might have hidden in behind unknowns will now appear on this output bit. Hence, in this example any one error can be detected in the presence of one unknown with an X-filter which takes only one input.

## VI. RESULTS

In this section we tabulate several parameters associated with some example compaction scheme based on two well known error correcting codes: the distance 4 extended hamming codes and the distance 5 BCH codes as well as the special techniques described in Section V. The results are given in Table I and Table II. In Table I we give compaction schemes comparable to those given in [2], so that we can compare our results against X-compact. In Table II we present some examples of more powerful compaction schemes possible with X-filter. The first four columns in both the tables list the parameters of the compaction scheme, the maximum number of scan chains supported, the maximum number of unknowns tolerated, maximum number of errors that are guaranteed detection and maximum number of errors guaranteed correction, respectively. The next three columns list the parameters of a X-filter based compactor with the above parameters: the error correcting code used for compaction, the number of output pins and the number of inputs to the X-filter. The last column in Table I, gives the number of output pins required for an equivalent X-compact scheme.

It can be seen from Table I that our methodology requires much less output pins as compared to X-compact in most cases and in all cases the (number of output pins) + (number of inputs) is equal to the number of output pins required in X-compact. Using more inputs rather than output pins is a big advantage since the input data that has to be fed into the X-filter can be compressed further using techniques similar

TABLE I  
EXAMPLES OF COMPACTION SCHEMES ACHIEVABLE WITH X-FILTER COMPARED WITH X-COMPACT

Compaction Scheme Parameters				X-filter Based Compaction			X-compact
# Scan Chains ( $n$ )	Max. # Unknowns Tolerated ( $x$ )	Guaranteed Error Detection	Guaranteed Error Correction	ECC Code	# Output Pins ( $m$ )	# Inputs	# Output Pins
126	1	$2-x$	0	Code in Section V	8	1	9
1716	1	$2-x$	0	Code in Section V	12	1	13
512	1	$3-x$	1	Extended Hamming	10	8	18
2048	1	$3-x$	1	Extended Hamming	12	10	22

TABLE II  
EXAMPLES OF MORE POWERFUL COMPACTION SCHEMES ACHIEVABLE WITH X-FILTER

Compaction Scheme Parameters				X-filter Based Compaction		
# Scan Chains ( $n$ )	Max. # Unknowns Tolerated ( $x$ )	Guaranteed Error Detection	Guaranteed Error Correction	ECC Code	# Output Pins ( $m$ )	# Inputs
512	2	$3-x$	1 if $x \leq 1$	Extended Hamming	10	20
511	1	$4-x$	2	BCH	18	18
511	2	$4-x$	2 if $x \leq 1$ 1 if $x \leq 2$	BCH	18	36

to those used for test data compression. The reader is also reminded that the codes in the above examples are capable of detecting more errors in special cases than are listed in the table. For example any odd number of errors are also detected.

Table II highlights the flexibility of using any off-the-shelf ECC in our scheme. If desired, higher X-tolerance and error detection capability can be easily achieved by using more powerful codes like BCH codes as presented in Table II.

## VII. CONCLUSIONS

In this paper we present X-filter a novel test response compaction scheme. X-filter enables the use of off-the-shelf error correcting codes to be used for compaction without requiring any special ATE support or data post processing, at the same time preserving their error correction and detection capabilities. The flexibility of using off-the-shelf ECC allows us to construct schemes with multiple X tolerance and multiple error detection and correction capabilities. These goals are achieved at the expense of only  $O(mx)$  additional on chip hardware which requires  $mx$  inputs and is independent of the design, test vector set and the ECC used for compaction.

X-filter does not increase the number of output pins over that required by the underlying error correcting code itself. Achieving output compaction by using more inputs rather than output pins is another major advantage of our scheme because the input data to the X-filter can be stored in compressed form on the ATE and then decompressed using on chip decompressor. So, the number of actual input pins required to feed the X-filter data into the chip can be much less than  $mx$ . The redundancy in the X-filter input data comes from the fact that typically only a few shift cycles contain unknowns and X-filter is required in only a few shift cycles. Using more

inputs rather than output pins is also attractive from the point of view of testing several devices in parallel.

Finally, since error correcting codes allow powerful error correction capabilities, our technique extends powerful diagnosis support during production test, and does not necessarily require a bypass mode for diagnosis. This is extremely critical in yield learning environment where the fail data collected during production testing is used in a continuous yield learning exercise, which is becoming very important with 90nm and later technologies.

## REFERENCES

- [1] J.H. Patel and S. Lumetta and S.M. Reddy, *Application of Saluja-Karpovsky Compactors to Test Responses with Many Unknowns*, Proc. IEEE VLSI Test Symp., pp. 107-112, 2003.
- [2] S. Mitra and K.S. Kim, *X-Compact: An Efficient Response Compaction Technique*, IEEE Trans. CAD, Vol. 23, Issue 3, pp. 421-432, March 2004.
- [3] P. Wohl and L. Huisman, *Analysis and Design of Optimal Combinational Compactors*, Proc. IEEE VLSI Test Symp., pp. 101-106, 2003.
- [4] J. Rajski, J. Tyszer, C. Wang and S. Reddy, *Convolutional Compaction of Test Responses*, Proc. IEEE International Test Conference, pp. 745-754, 2003.
- [5] Chen Wang, S.M. Reddy, I. Pomeranz, J. Rajski and J. Tyszer, *On compacting test response data containing unknown values*, Proc. International Conference on Computer Aided Design, pp. 855-862, 2003.
- [6] O. Sinanoglu, and A. Orailoglu, *Parity-based output compaction for core-based SOCs*, Proc. European Test Workshop, pp. 15-20, 2003.
- [7] B. Konemann, J. Mucha and G. Zwiehoff, *Built-in logic block observation technique*, Proc. IEEE International Test Conference, pp. 37-41, 1979.
- [8] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann, *OPMISR: The foundation for compressed ATPG vectors*, Proc. IEEE International Test Conference, pp. 748-757, 2001.
- [9] P. Wohl, J.A. Waicukauski and S. Patel, *Scalable selector architecture for X-tolerant deterministic BIST*, Proc. Design Automation Conference, pp. 934-939, 2004.
- [10] K. Chakrabarty, *Zero-aliasing space compaction using linear compactors with bounded overhead* IEEE Trans. Computer-Aided Design, vol. 17, pp. 452-457, May 1998.

- [11] K. Chakrabarty and J.P. Hayes, *Test Response Compaction Using Multiplexed Parity Trees*, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 15, no. 11, pp. 1399-1408, Nov. 1996.
- [12] B. Pouya and N. Touba, *Synthesis of Zero-Aliasing Space Elementary-Tree Space Compactors*, Proc. IEEE VLSI Test Symp., pp. 70-77, 1998.
- [13] O. Sinanoglu, and A. Orailoglu *Efficient construction of aliasing-free compaction circuitry* IEEE Micro, Volume: 22, Issue: 5, pp. 82-92, Sept.-Oct. 2002
- [14] K. K. Saluja and M. Karpovsky, *Testing computer hardware through data compression in space and time*, Proc. IEEE International Test Conference, pp. 83-89, 1983.
- [15] S. Lumetta and S. Mitra, *X-codes: Error control with unknowable inputs*, Proc. Int. Symp. Inform. Theory, p. 102, 2003.
- [16] V. Chickermane, B. Foutz and B. Keller, *Channel Masking Synthesis for Efficient On-Chip Test Compression*, Proc. IEEE International Test Conference, pp. 452-461, 2004.
- [17] R. Blahut, *Algebraic Codes for Data Transmission*, Cambridge University Press.
- [18] J. Rajska et al., *Embedded deterministic test for low cost manufacturing test*, Proc. IEEE International Test Conference, pp. 301-310, 2002.
- [19] I. Hamzaoglu and J. H. Patel, *Reducing test application time for full scan embedded cores*, Proc. Int. Symp. Fault-Toler. Comput., pp. 260-267, 1999.
- [20] E. H. Volkerink, A. Khoche, and S. Mitra, *Packet-based test data compression techniques*, Proc. IEEE International Test Conference, pp. 154-163, 2002.
- [21] P. Wohl, J. A. Waicukauski, S. Patel, and M. B. Amin, *Efficient compression and application of deterministic patterns in a logic BIST architecture*, Proc. Design Automation Conf., pp. 566-569, 2003.
- [22] A. Jas, B. Pouya and N. A. Touba, *Virtual Scan Chains: A Means for Reducing Scan Length in Cores*, Proc. VLSI Test Symposium, pp. 73-78, 2000.
- [23] I. Bayraktaroglu and A. Orailoglu *Test volume and application time reduction through scan chain concealment*. Proceedings IEEE Design Automation Conference, pp. 151-155, 2001.
- [24] A. Chandra and K. Chakrabarty, *Frequency-Directed Run Length (FDR) Codes with Application to System-on-a-Chip Test Data Compression* Proc. IEEE VLSI Test Symp., pp. 42-47, 2001.
- [25] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheeler, *A SmartBIST variant with guaranteed encoding*, Proc. IEEE Asian Test Symp., pp. 325-330, 2001.
- [26] C.V. Krishna and N.A. Touba, *3-Stage Variable Length Continuous-Flow Scan Vector Decompression Scheme*, Proc. IEEE VLSI Test Symp., pp. 79-86, 2004.