

Scan-BIST Based on Cluster Analysis and the Encoding of Repeating Sequences

LEI LI

Vimicro Corp.

and

ZHANGLEI WANG and KRISHNENDU CHAKRABARTY

Duke University

We present a built-in self-test (BIST) approach for full-scan designs that extracts the most frequently occurring sequences from deterministic test patterns. The extracted sequences are stored on-chip, and are used during test application. Three sets of test patterns are applied to the circuit under test during a BIST test session; these include pseudorandom patterns, semirandom patterns, and deterministic patterns. The semirandom patterns are generated based on the stored sequences and they are more likely to detect hard-to-detect faults than pseudorandom patterns. The deterministic patterns are encoded using either the stored sequences or the LFSR reseeding technique to reduce test data volume. We use the cluster analysis technique for sequence extraction to reduce the amount of data to be stored. Experimental results for the ISCAS-89 benchmark circuits show that the proposed approach often requires less on-chip storage and test data volume than other recent BIST methods.

Categories and Subject Descriptors: B.7.3 [**Integrated Circuits**]: Reliability and Testing—*Built-in tests*

General Terms: Algorithms, Design, Reliability

Additional Key Words and Phrases: Built-in self-test (BIST), test compression, clustering test data volume

ACM Reference Format:

Li, L., Wang, Z., and Chakrabarty, K. 2007. Scan-BIST based on cluster analysis and the encoding of repeating sequences. *ACM Trans. Des. Automat. Elect. Syst.* 12, 1, Article 4 (January 2007), 21 pages. DOI = 10.1145/1188275.1188279 <http://doi.acm.org/10.1145/1188275.1188279>

This research was supported in part by the National Science Foundation under grants CCR-9875324 and CCR-0204077, and by a graduate fellowship from the Design Automation Conference.

A preliminary version of this article appeared in the *Proceedings of the 2005 Design, Automation and Test in Europe Conference*, pp. 1142–1147.

Authors' addresses: L. Li, #35 Xueyuan Rd, 6 Floor, Shining Tower, Haidian District, Beijing, China 100083; email: lei.li.11@gmail.com; Z. Wang and K. Chakrabarty, Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708; email: {zw8,krish}@ee.duke.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org. © 2007 ACM 1084-4309/2007/01-ART4 \$5.00 DOI 10.1145/1188275.1188279 <http://doi.acm.org/10.1145/1188275.1188279>

1. INTRODUCTION

High test data volume and limited tester channel bandwidth are two major problems encountered in the testing of today's system-on-chip integrated circuits. In order to mitigate these problems, a number of techniques based on test data compression, built-in self-test (BIST), and a combination of the two have been proposed in the literature.

In the test data compression approach, a deterministic test set is compressed and stored in tester memory. The compressed test set is transferred through tester channels to the circuit under test (CUT), where it is decompressed by decoding hardware. Test compression techniques based on on-chip pattern decompression were presented in Bayraktaroglu and Orailoglu [2001], Chandra and Chakrabarty [2003], Gonciari and Nicolici [2003], Jas and Touba [1998], Li et al. [2003], Rajski et al. [2002], Reda and Orailoglu [2002], Reddy et al. [2002], and Wurtenberger et al. [2004]. In BIST solutions, test patterns are generated by an on-chip pseudorandom pattern generator, usually a linear-feedback shift-register (LFSR). A number of BIST techniques based on test point insertion [Schotten and Meyr 1995], reseeding [Hellebrand et al. 2000; Liang et al. 2001; Rajski et al. 1998], bit-flipping [Wunderlich and Kiefer 1996], bit-fixing [Touba and McCluskey 1996], and weighted random pattern testing [Wang 2001] have been proposed. Deterministic test patterns are applied in BIST by either controlling the state of the pattern generator [Hellebrand et al. 2000; Krishna et al. 2001; Liang et al. 2001; Rajski et al. 1998] or by altering the output of the pattern generator [Touba and McCluskey 1996; Wang 2001; Wunderlich and Kiefer 1996].

Techniques based on the combination of data compression and BIST have also been developed recently [Jas et al. 2001; Krishna and Touba 2002]. The hybrid BIST scheme presented in Jas et al. [2001] applies weighted pseudorandom patterns to the circuit to achieve 100% fault coverage. The compressed weight set is stored on ATE and decompression is carried out using an on-chip lookup table. In Krishna and Touba [2002], the seeds for the LFSR were compressed using statistical coding.

In this article, we present a BIST approach for full-scan designs that extracts the most frequently occurring sequences from deterministic test patterns; these extracted sequences are stored on-chip. The test session consists of three stages. In the first stage, pseudorandom patterns generated by an LFSR are applied to the CUT to detect easy-to-detect faults. In the second stage, semirandom patterns are generated by randomly selecting some of the stored sequences and flipping some of their data bits. Since the semirandom patterns are generated based on the sequences extracted from deterministic patterns, they are more likely to detect the hard-to-detect faults than the pseudorandom patterns [Tsai et al. 1997]. Faults that are not detected by pseudorandom and semirandom patterns are detected by deterministic patterns in the third stage. The deterministic patterns are either encoded on the basis of the stored sequences, or encoded using LFSR reseeding, to reduce the test data volume, and are decoded/generated during the third stage of the test session. The overall approach is similar to that in Wurtenberger et al. [2004], where repeating scan slices were

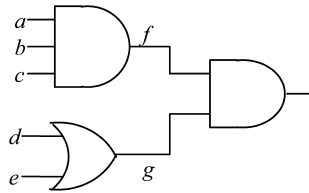


Fig. 1. A simple circuit used to motivate the clustering-based approach.

stored in a dictionary, and corrections (bit-flips) were applied to selected scan slices.

The rest of this article is organized as follows. Section 2 presents the proposed BIST approach and its associated synthesis procedure. The cluster analysis algorithms used in the synthesis procedure are described in Section 3. Section 4 describes the proposed BIST architecture. Section 5 presents experimental results for the ISCAS-89 benchmark circuits. Finally, Section 6 concludes the article.

2. PROPOSED APPROACH

The proposed BIST approach is based on the observation that identical or similar sequences often appear in many test patterns that are applied to a logic circuit. Consider the simple circuit shown in Figure 1. In order to detect the stuck-at faults $d/0$, $d/1$, $e/0$, $e/1$, $g/0$, and $g/1$, f needs to be set to 1 to propagate the faulty values to the output, which requires that the primary inputs “ abc ” be set to “111.” Thus the sequence “111” appears at the primary inputs “ abc ” in the six test patterns to detect the above stuck-at faults. Based on this observation, we extract a number of more frequently occurring sequences from the deterministic test patterns for a logic circuit and store them on the chip. By selecting sequences and flipping some of their data bits, we can either generate semirandom patterns or encode deterministic patterns, based on how we select the sequences and the bits to be flipped. In contrast to bit-flipping BIST [Wunderlich and Kiefer 1996], in which bits from a pseudorandom sequence are selectively flipped, here we carefully flip bits from a predetermined subsequence that is stored on-chip.

Figure 2 shows the design flow of the proposed BIST technique. The BIST procedure is divided into three stages, during which three sets of test vectors are applied to the CUT. These three test sets are shown in bold boxes. Software tools such as a fault simulator, a test generator (ATPG), and an LFSR simulator (LFSR) are shown in gray rounded boxes. Test vectors and intermediate data are shown in white rounded boxes. Fault sets (F_0 , F_1 , and F_2) are shown in three-dimensional (3D) boxes.

Similar to other BIST approaches, we first apply a number of pseudorandom patterns to the CUT to detect easy-to-detect faults (stage 1). Next we generate deterministic patterns for the remaining faults (F_1 in Figure 2), and extract a number of frequently occurring sequences from these deterministic patterns. The extraction of repeating sequences is carried out in two steps. First, the

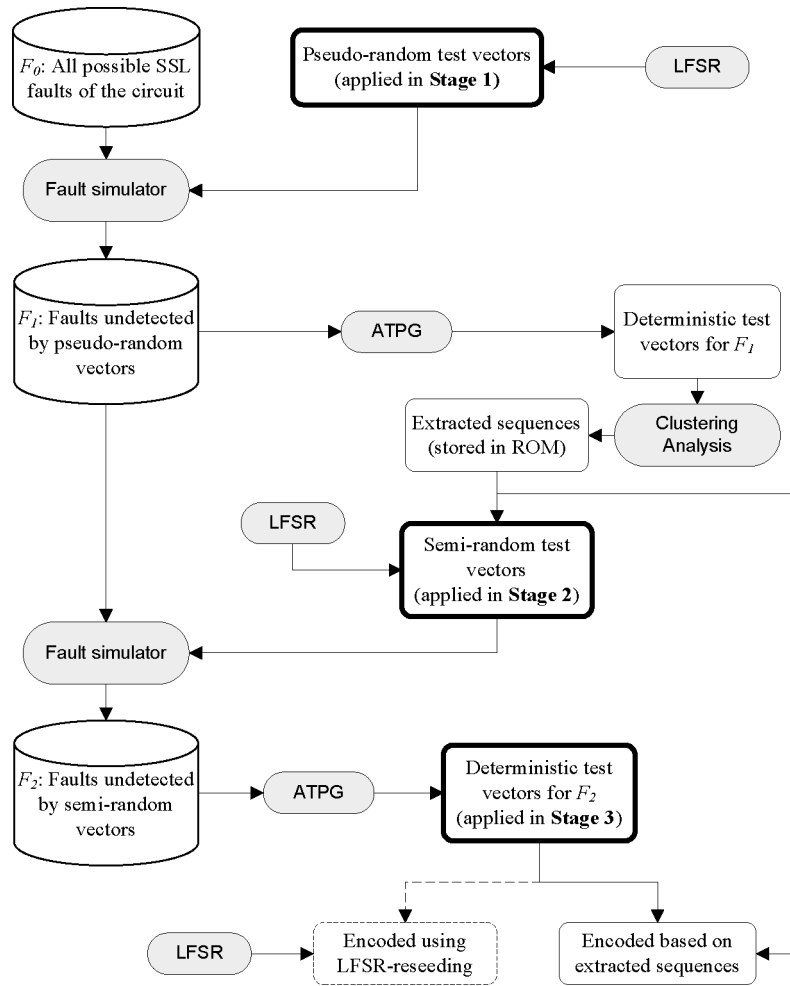


Fig. 2. Design flow of the proposed BIST technique.

data bits in the test vectors are reorganized such that the positions that are specified in the same test vectors are grouped together. This is illustrated in Figure 3, where positions 1, 2, 3, and 7 are grouped, and positions 4, 5, and 6 are grouped. Corresponding to the reorganization of the data bits in the test set, the scan cells in the scan chain also need to be reorganized. Repeating sequences are next identified for each group. As shown in the figure, four sequences are extracted for the first group. Finally, we need 23 bits to store the sequences (the x in the last group does not need to be stored). In other words, the test set can be viewed as a matrix in which each row is a test vector. First, columns are grouped/reorganized to bring together the specified bits in a row. Then the grouped columns are regarded as a submatrix and repeating sequences/rows are determined.

In the simple example of Figure 3, the position groups and the repeating sequences are easy to identify, and the number of groups and sequences are

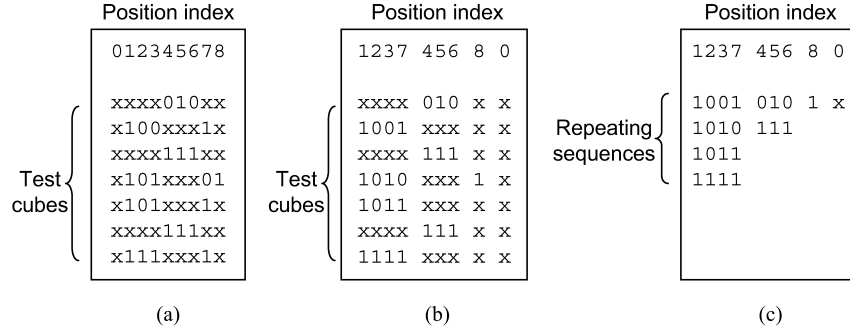


Fig. 3. An example to illustrate the extraction of frequently occurring patterns.

small. However, a test set for a real-life circuit can lead to a large number of groups and sequences. We need to reduce the number of groups such that the encoding of a deterministic pattern is beneficial, and we need to merge some of the sequences in each group to make the storage requirement manageable. We use cluster analysis for both position grouping and sequence merging; the details of these procedures are discussed in Section 3.

After grouping the positions in the scan chain and extracting the most occurring sequences in each group, we generate semirandom patterns based on the extracted sequences. Assuming that the positions in the scan chain are grouped into G groups, and R sequences are identified for each group, we use $\log_2 R$ outputs from the LFSR to generate a random number $r, 0 \leq r < R$, which selects a sequence from each group. Several outputs of the LFSR are then AND-ed to generate a flip indication sequence with a small proportion of 1s. The semirandom patterns shifted into the scan chain are obtained by doing an exclusive-or between the randomly selected sequences and the flip indication sequence.

After a number of semirandom test patterns are generated (Stage 2), we run fault simulation to determine the faults that are not detected by both the pseudorandom patterns and the semirandom patterns (F_2). Then we use an ATPG tool to generate deterministic test cubes for the remaining undetected faults (F_2). These deterministic cubes are then encoded by indicating the selection of sequence and bits to be flipped. For example, if we need to encode the cube “x000011x1” based on the sequence obtained in Figure 3(c), we first reorder the sequence to “000x,” “011,” “1,” and “x,” then encode it as “*sequence*(0, 0) *flip*(0), *sequence*(1, 0) *flip*(2),” where *sequence*(i, j) means selecting the j th sequence in the i th group, and *flip*(k) means flipping the k th bit. In this example, the number of groups is 4, the maximum number of sequence in a group is 4, and the maximum number of bits in a sequence is 4. So we need 2 bits each to encode the group index, sequence index, and the bit index. Since multiple groups might be needed for encoding a deterministic pattern, and multiple bits might need to be flipped in one sequence, a prefix is needed before the group index and the bit index to indicate if it is the last group for this pattern or if it is the last flipping bit in the current sequence. The encoded data for the deterministic pattern “x000011x1” is shown in Figure 4.

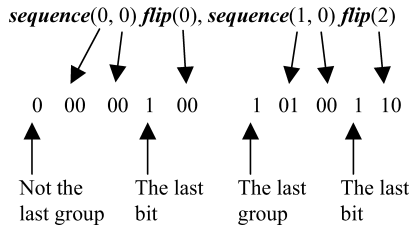


Fig. 4. The encoded data for the deterministic test cube “x000011x1.”

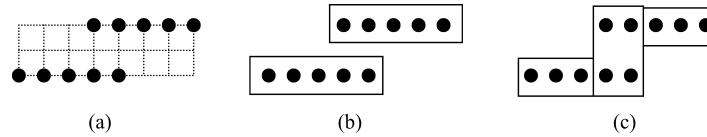


Fig. 5. An illustration of cluster analysis with different objectives.

While the above encoding method is guaranteed to compress any arbitrary test pattern, its implementation can lead to a complex on-chip decoder. We can also use a standard LFSR reseeding method to encode these deterministic test patterns, as long as the maximum number of specified bits in these patterns is less than the order of the on-chip LFSR [Koenemann 1991]. We show in Section 5 that, for most cases involving benchmark circuits, the deterministic test patterns can be encoded using reseeding with a 20-stage LFSR. These results demonstrate the effectiveness of the semirandom test patterns. The use of LFSR reseeding simplifies the BIST architecture, and results in less area overhead since we can reuse the LFSR hardware. However, reseeding with a small 20-stage LFSR does not guarantee that any arbitrary pattern can be encoded. In such cases, either a larger LFSR can be used or we can revert to the encoding procedure based on the extracted sequences. In Figure 2, these two alternative encoding methods are shown in different styles, one with solid lines and the other with dashed lines.

3. CLUSTER ANALYSIS

Cluster analysis is used for numerical classification extensively in many fields such as biology, geography, and marketing [Everitt et al. 2001]. The objective of cluster analysis varies depending on the target application. Consider the nodes in Figure 5(a). In a cluster, each node can be reached from another node through a series of hops to the neighbor nodes. If the objective is to minimize the maximum hopping span, then we can get the clustering results shown in Figure 5(b). With a different objective, for example, minimizing the maximum distance between any pair of nodes in a cluster, we get the clustering results shown in Figure 5(c).

In our approach, we use an agglomerative clustering algorithm for both position grouping and sequence merging [Everitt et al. 2001]. Figure 6 provides an overview of a general agglomerative algorithm. The input to the procedure is a system S containing a given number of clusters. The distance between each pair of clusters is calculated and stored in the distance matrix D . In each loop

Procedure *Agglomerate* (S , $maxClusterNum$, $minMergeDistance$)

```

1  /* Initialize the distance matrix. */
2   $D = initialize(S)$ ;
3  while ( $size(S) > maxClusterNum$ )
4      ( $minDistance, i, j$ ) =  $getMinimum(D)$ ;
5      if ( $minDistance > minMergeDistance$ )
6          break; /* Jump out of the while loop. */
7      else
8           $merge(i, j)$ ;  $deleteNode(S, j)$ ;
9           $deleteRowColumn(D, j)$ ;  $updateDistance(D, i)$ ;
10     end if;
11 end while;
```

Fig. 6. The agglomerative algorithm.

of the procedure, the minimum distance and its associated cluster indices i and j are found. If the minimum distance is not larger than the preset threshold $minMergeDistance$, the clusters i and j are merged and j is deleted from the graph and the distance matrix. All the elements in the distance matrix related to i are also updated. The procedure terminates when either $maxClusterNum$ or $minMergeDistance$ is reached. The definitions of cluster and distance are different for position grouping and sequence merging. The following subsections explain the two problems in detail.

3.1 Clustering for Position Grouping

For position grouping, a cluster contains one or more positions (or scan cells) in the scan chain. A position is referred to as an *X-position* if and only if it contains no specified bits in any test pattern in the given test set; otherwise it is a *non-X-position*. If there are L_{nx} *non-X-positions*, then the initial system S contains L_{nx} clusters with each cluster containing one *non-X-position* ($L_{nx} \leq L$, L is the length of the scan chain).

Each position j has an associated index set I_j that records all the test vectors for which this position is specified. The distance between two positions i and j is calculated as the number of test vectors in which one position is specified while the other is unspecified, that is, it simply equals $|(I_i - I_j) \cup (I_j - I_i)|$. In order to calculate the distance between two clusters, with each cluster containing one or more positions, we define the centroid of a cluster as follows. The *centroid* of a cluster is a dummy position whose associated index set is the union of the associated index sets of all the positions contained in this cluster. The distance between two clusters is calculated as the distance between their centroids of the two clusters. We adopt this definition of a centroid because we are trying to group the positions that are specified in the same test vectors. After several positions are merged into a group, the corresponding group appears in a test vector as long as one of its position contains a specified-bit.

Figure 7 illustrates an example of position grouping. We use the same test data as in Figure 3. However, we set the threshold $maxClusterNum$ to 2, excluding a group that only contains *X-positions*. In this example, position 0 is unspecified in all the seven test vectors; therefore it is an *X-position*. As shown

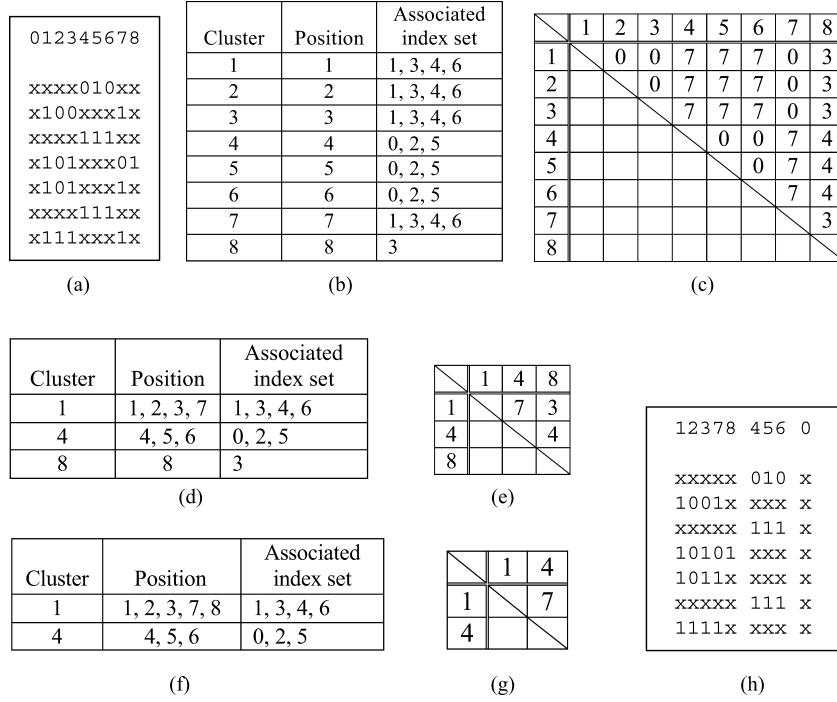


Fig. 7. Clustering for position grouping.

in Figure 7(b), the initial system consists of eight clusters, each of which contains one position. The test vector index set associated with the centroid of each cluster is also listed. The distances between each pair of clusters are listed in Figure 7(c). If we go through the procedure in Figure 6, at the first iteration, line 4 returns ($mindistance = 0, i = 1, j = 2$). Therefore clusters 1 and 2 are merged and cluster 2 is deleted at line 8. The distance matrix D is updated accordingly at line 9. After the first five iterations, all pairs of clusters with mutual distance 0 are merged; the remaining clusters and their pairwise distances are shown in Figures 7(d) and 7(e). Next, clusters 1 and 8 are merged because the distance between them is the minimum entry in the distance matrix. After the merging of clusters 1 and 8, the $maxClusterNum$ threshold is reached at line 3. Figures 7(f) and 7(g) show the eventual outcome of cluster analysis. The positions are then grouped as shown in Figure 7(h).

The above example assumes that the scan chain can be arbitrarily reordered. For practical circuits with layout constraints, however, this assumption is unlikely to be valid. A flip-flop needs to be placed close to its associated combinational I/Os, and hence it can only be relocated in a restricted manner. Arbitrary reordering of scan cells can lead to excessive wiring overhead and performance degradation. Therefore, during the position grouping procedure, we introduce another threshold, $maxOffset$, to control how positions (scan cells) can be reordered. If the k th scan cell along the scan chain becomes the k' th scan cell after reordering, then the condition $|k - k'| \leq maxOffset$ must hold. This criterion

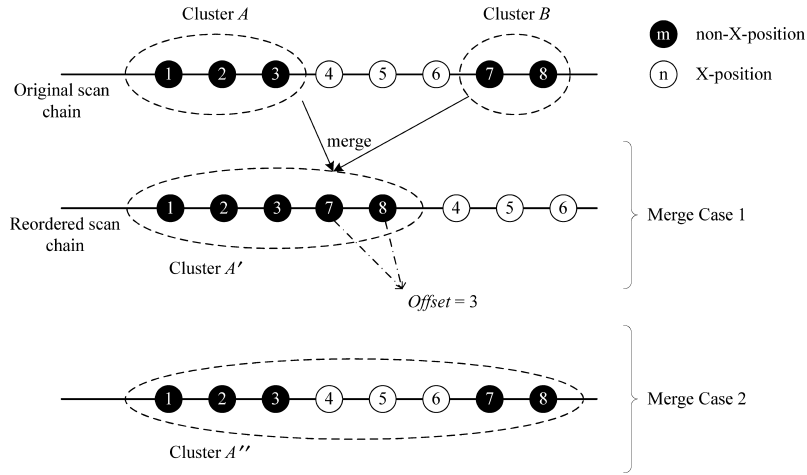


Fig. 8. Cluster merging and scan cell reordering.

ensures that a scan cell can only be moved within its proximity; therefore, reordering and the resulting wiring changes only occur in a small area. When $maxOffset$ equals the length of the scan chain L , arbitrary reordering is enabled. This constrained scan cell reordering method is referred to as *partial reordering*.

Figure 8 shows an example of cluster merging and scan cell reordering. Before merging, there are two clusters, $A = \{1, 2, 3\}$, and $B = \{7, 8\}$. Positions 4–6 are *X-positions*. After merging (as in Merge Case 1), the new cluster $A' = \{1, 2, 3, 7, 8\}$. Positions 7 and 8 are moved by an offset of 3. In the reordered scan chain, positions 7 and 8 become positions 4 and 5, respectively. If $maxOffset \geq 3$, then clusters A and B can be merged as in Merge Case 1; otherwise A and B can only be merged as in Merge Case 2, where *X-positions* are inserted into the new cluster A'' such that no offset violation occurs. Adding *X-positions* to a cluster results in larger storage size and less efficient extracted sequences.

When applying the agglomerative algorithm shown in Figure 6 for position grouping with partial reordering, the *getMinimum* procedure at line 4 first only considers cluster pairs that can be merged as in Merge Case 1 in Figure 8. The distance between a pair of clusters (i, j) is considered to be ∞ if $|k - k'| \leq maxOffset$ is violated after merging. If no more clusters can be merged using Merge Case 1 and the number of clusters in the current system is larger than $numMaxCluster$, *getMinimum* switches to Merge Case 2 such that clusters can be merged by inserting *X-positions*.

The time complexity of clustering for position grouping is $O(L_{nx}^3)$. As can be seen from Figure 6, initially the system \mathcal{S} contains L_{nx} clusters. In each iteration (lines 3–10), one cluster is eliminated until the remaining number of clusters is less than or equal to $maxClusterNum$, or no clusters can be merged. Therefore, in the worst case $(L_{nx} - maxClusterNum) \simeq L_{nx}$ iterations are needed. At line 4, the distance matrix D is searched for the minimum distance. The time complexity of the search operation is $O(|\mathcal{S}|^2/2)$. The time complexities of *deleteNode*

and *deleteRowColumn* are $O(1)$ and $O(|S|)$, respectively. The time complexity of *merge*(i, j) is proportional to the total size of the two clusters, which is $O(L_{nx})$. The procedure *updateDistance* computes the distance between the newly generated cluster and other clusters, and has a time complexity of $O(|S|L_{nx})$, since the total size of any pair of clusters is always less than L . Therefore, the overall time complexity is $O(L_{nx}^3)$.

3.2 Clustering for Sequence Merging

After position grouping, we reorder the columns of the test data matrix to bring the positions/columns in a group together. The basic idea was illustrated in Figure 3. For each group, that is, a submatrix, an initial cluster system is generated and the agglomerative algorithm is used for sequence merging. In this cluster system, a cluster contains one or more *nodes*, with each node representing a test vector fragment formed by the positions (scan cells) belonging to the group. A node contains at least one specified bit at those positions. The number of nodes in a group is equal to the number of test vectors in which the group appears, that is, the number of rows in the submatrix that contain at least one specified bit. A three-valued sequence (with elements from $\{0, 1, x\}$) is associated with each node. The distance between a pair of nodes is the number of bits that are in conflict for the two sequences that are associated with them. Two bits are in conflict with each other if both of them are specified and they are set to different values. In the initial cluster system, each cluster contains one node.

We define the centroid of a cluster as a dummy node associated with a three-valued sequence, which is obtained as follows. The length of the sequence is the same as the sequences associated with the nodes in this cluster. For each bit in the string, we count the number of 0s (*zeroCounts*) and the number of 1s (*oneCounts*) in this bit position for all the sequences associated with the nodes in this cluster. If both numbers are equal to 0, this bit is set to 'x.' Otherwise this bit is set to 0 if $zeroCounts \geq oneCounts$, and 1 if $zeroCounts < oneCounts$. The distance between a pair of clusters is then calculated as the product of the distance between the centroids of the pair of clusters and the total number of nodes in these two clusters. The above method for determining the centroid of a cluster and the distance between a pair of clusters has been developed to ensure that only a small number of bits need to be flipped to generate all the sequences in a cluster from the centroid sequence of the cluster.

Figure 9 illustrates cluster analysis for sequence merging. The input data in this example corresponds to the first group of sequences in Figure 7(h). We set the threshold *maxClusterNum* to 2. Figure 9(b) shows the initial clusters and the sequences associated with their centroids. The distance matrix is shown in Figure 9(c). Note that the distance between a pair of clusters is the product of the number of conflict bits with the number of nodes in this pair of clusters. Thus although there are 2 conflict bits between clusters 1 and 2, their distance is 4, as shown in the distance matrix. First, clusters 1 and 3 are merged because the distance between them (2) is the smallest entry in the distance matrix. After merging clusters 1 and 3, the distance matrix is updated. Then clusters

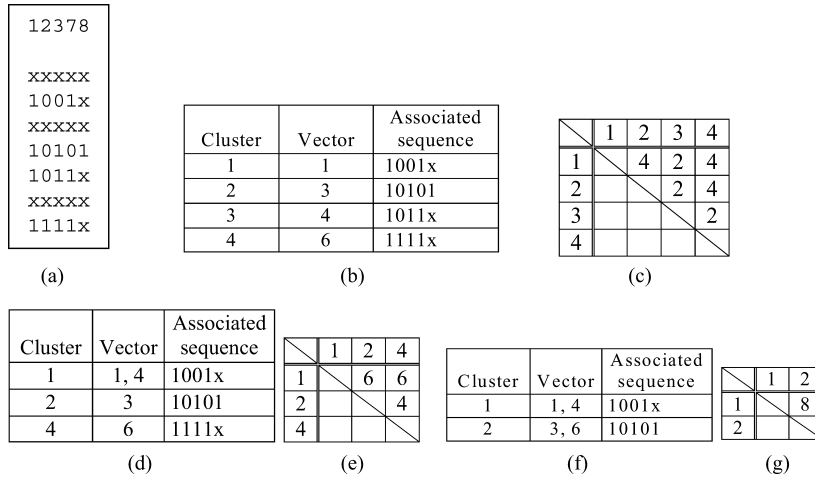


Fig. 9. Clustering for sequence merging.

2 and 4 are merged. This concludes the cluster analysis because the threshold $maxClusterNum$ is reached. The final sequences extracted for this group are “1001x” and “10101.”

The time complexity for pattern merging can be obtained in a similar way as that for position grouping. If Group i contains P_i nodes (a node is a test vector that contains at least 1 specified bit at the positions of this group), then the time complexity of pattern merging for this group is $O(P_i^3)$. Since the maximum number of groups is a user-defined parameter ($maxClusterNum$ in Figure 6, denoted here as N for simplicity), which is a constant, the overall time complexity of the pattern merging procedure can be derived as follows. Since $P_i \leq P$, where P is the total number of patterns for the remaining faults (F_1 in Figure 2), we have $\sum_{i=1}^N P_i^3 \leq NP^3$. Therefore, the overall time complexity is $O(P^3)$.

4. PROPOSED BIST ARCHITECTURE

Figure 10 shows the proposed BIST architecture. We first describe the architecture for a CUT with a single scan chain. We then discuss the extension for multiple scan chains. For a single scan chain, the signals connected to the two multiplexers and the exclusive-or gate are all 1-bit wide.

4.1 Single Scan Chain

In the first stage of the test session, the signal *Select_random* is set to 0. Hence the pseudorandom patterns generated by the LFSR are shifted into the scan chain through MUX I and applied to the CUT. In this stage, Bit counter A is also used to indicate the end of each test pattern. The path taken by the test data in this stage is shown in Figure 11.

In the second stage, the signal *Select_random* is set to 1, and *Select_flip* is set to 0. The path taken by the test data in this stage is shown in Figure 12. The test data is obtained by doing an the exclusive-or between the *Flip_indication_R* signal and the ROM output. The signal *Flip_indication_R* is obtained by

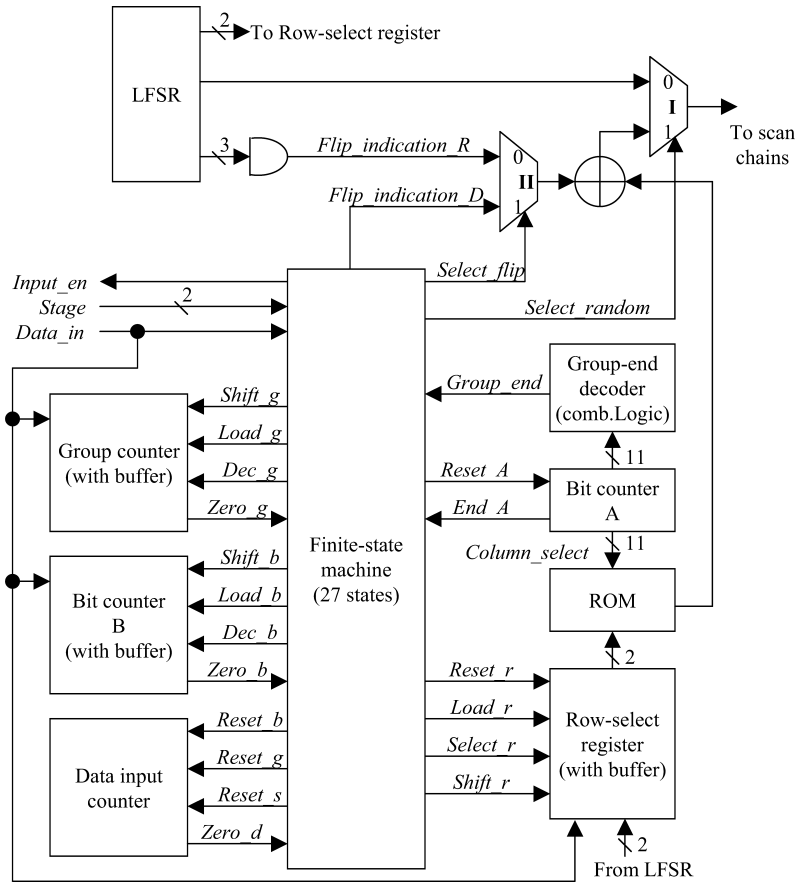


Fig. 10. Proposed BIST architecture.

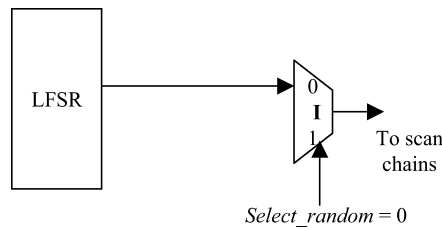


Fig. 11. Test data path in the first stage of the test session.

AND-ing several bits from the LFSR. Thus it contains much more 0s than 1s, a feature that is used to flip some of the bits in the data sequence obtained from the ROM. The ROM contains $R \times C$ bits organized as a matrix $[M]_{R \times C}$ with R rows and C columns. Let the content of the ROM for the location addressed by the pair (i, j) , $0 \leq i < R$ and $0 \leq j < C$, be $M_{i,j}$. The number of columns C in the ROM is often less than the scan chain length L . Note that the *Column_select* signal comes from Bit counter A and it can be larger than C . In this case, the

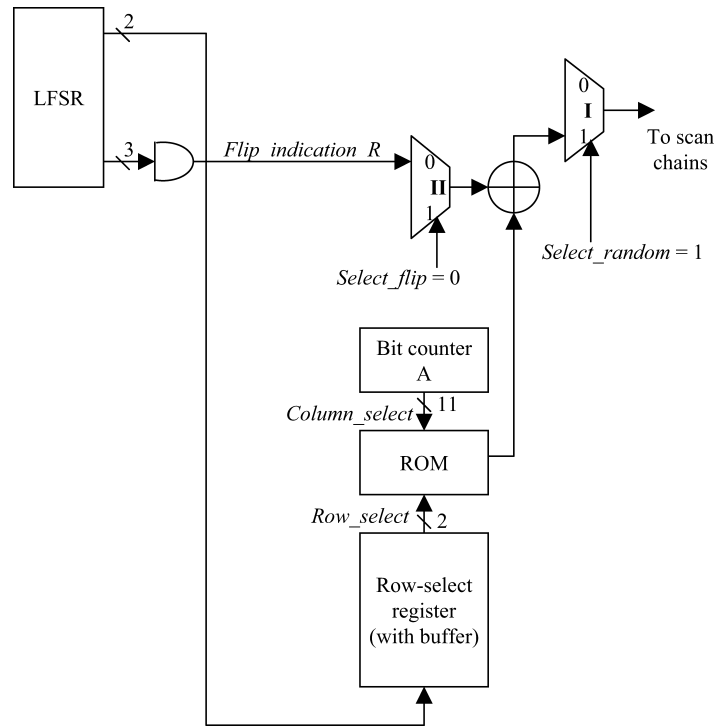


Fig. 12. Test data path in the second stage of the test session.

ROM simply outputs the value 0. The data in the ROM is divided horizontally into G groups, where each row in a group corresponds to an extracted sequence. The Group-end decoder always outputs 0, except at the last bit of each group, when it outputs 1 to indicate the end of the current group. The signals $Load_r$ and $Select_r$ are then set to 1 to load the random number from the LFSR into the Row-select register such that a random sequence is selected from the next group.

In the third stage, both $Select_random$ and $Select_flip$ are set to 1. The path taken by the test data in this stage is shown in Figure 13. Thus the test data shifted into the scan chain is obtained via an exclusive-or operation between $Flip_indication_D$ signal and the ROM output. $Flip_indication_D$ is generated by the finite-state machine based on the encoded data from $Data_in$. The Group counter contains an extra buffer besides the standard counter in it. The extra buffer is used to store the data that is shifted in from $Data_in$ while the standard counter is operating. With the signal $Load_g$ set to 1, the data stored in the buffer is loaded into the standard counter, which is then ready for counting down. The same pipeline structure is used for Bit counter B and Row-select register. The Group counter is used to indicate whether the selected group is reached. After the selected group is reached, the Row-select register is loaded with the selected sequence index which is stored in its buffer. The Bit counter B is also loaded with the flipping bits index stored in its buffer and it then starts to count down. When the Bit counter B decrements to 0, $Flip_indication_D$ is set

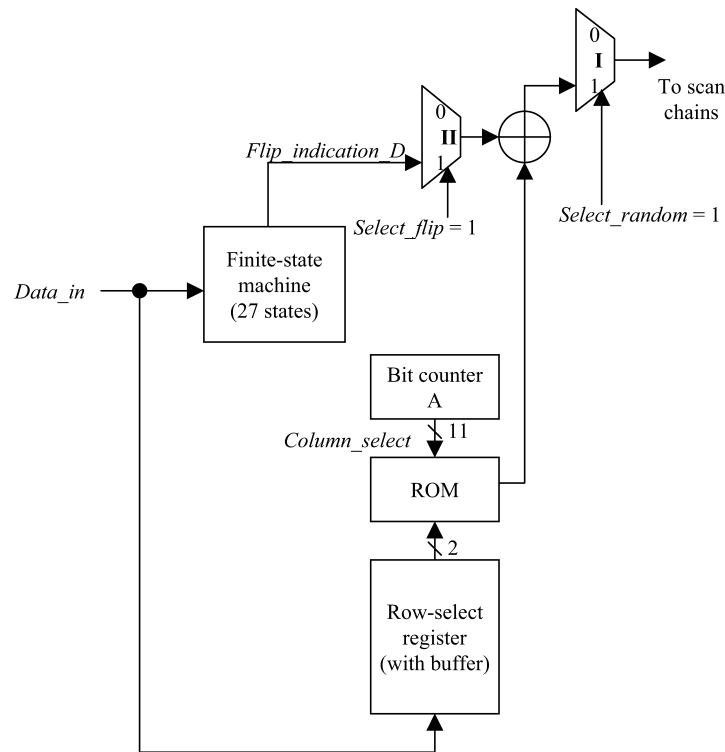


Fig. 13. Test data path in the third stage of the test session.

to 1 to flip the data bit from the ROM. The Data input counter is used to count the number of bits shifted from *Data_in* and indicate whether the data for the group index, sequence index, or bit index has been completely shifted into its buffer.

If LFSR reseeding is used in the third stage, the BIST architecture is considerably simplified. In Figure 10, blocks specific to the stage-3-decoding, including Mux II, Group counter, Bit counter B, Data input counter, and their associated signals are no longer used. The signals *Data_in*, *Input_en*, *Select_r*, and *Shift_r* are also removed. Only one new signal is added from the FSM to the LFSR to control the load of LFSR seeds. The FSM contains only three states since no complex decoding procedure is used. The test data path is the same as shown in Figure 11.

4.2 Multiple Scan Chains

For multiple scan chains, the ROM is reorganized as shown in Figure 14, which corresponds to the case of four scan chains for the CUT. The number of banks in the ROM equals the number of scan chains, that is, four in this example. In each scan cycle, 1 bit from each bank is shifted into the scan chain. All of the 4 bits are from the same position of the banks, which is indicated by the same column-select and row-select signals. The organization of the sequences

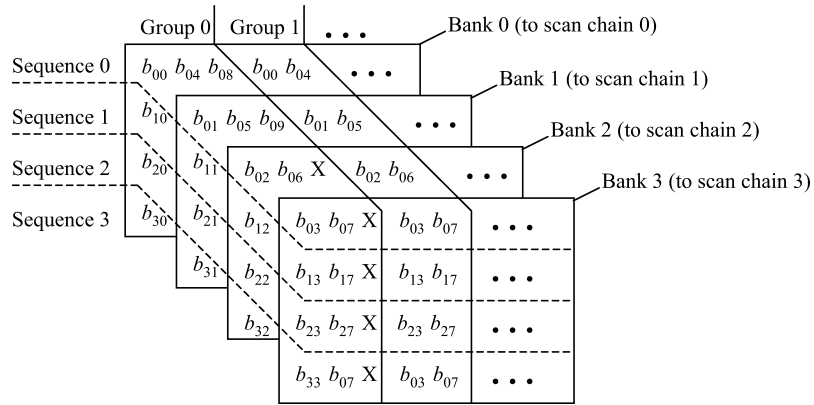


Fig. 14. Organization of the stored sequences.

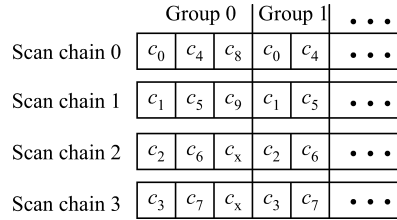


Fig. 15. Reorganization of the scan cells.

is also shown in the figure, where $b_{i,j}$ denotes the j th bit of the i th sequence in a group, the groups of the sequences are divided by the solid line, and the sequences in each group are divided by the dashed line. As shown in the figure, the data bits of the i th sequence are placed in the i th row of the ROM banks, and the current columns of all the banks are filled before proceeding to the next columns. The number of bits in a group may not always divide the number of ROM banks (the number of scan chains) exactly; thus some cells in the last column of this group need to be filled randomly.

In order to exploit position grouping, the cells in the scan chains also need to be reorganized, that is, the cells belonging to the first group are first placed at the first positions of the scan chains, then at the second positions of the scan chains, and so on. Figure 15 shows the reorganization of the scan cells corresponding to the data organization in the ROM shown in Figure 14. In Figure 15, Group 0 contains 10 scan cells. The first four cells are placed at the first position of each scan chain, the next four cells are placed at the second position of each scan chain, and the remaining two cells are placed at the third position of scan chain 0 and scan chain 1. Scan cells that are unspecified in all the deterministic patterns are used to fill the third position of scan chain 2 and scan chain 3.

For a CUT with m scan chains ($m > 1$) and the above organization of the ROM, the decoding architecture operates in nearly the same fashion as a single scan chain. The differences are as follows. First, the output from the ROM, the signals *Flip_indication_R* and *Flip_indication_D*, and the input signals to MUX

I are all m -bit wide. Second, in the third stage of the test session, recall that, for a single scan chain, the bit index is completely shifted into the Bit counter B to determine the scan cycle in which the signal *Flip_indication_D* is set to 1. The bit index is divided into two parts. The lower-order $\log_2 m$ bits are used to determine which of the m bits in the signal *Flip_indication_D* are set to 1; the remaining bits are shifted into the Bit counter B to determine in which scan cycle to set the selected bit of *Flip_indication_D* to 1.

4.3 Hardware Overhead

We first implemented the BIST architectures for a single scan chain using the *lsi_10k* library of Synopsys Design Compiler. Using the wire load model for the *lsi_10k* library, we designated the normalized area for a unit-length of wire to be 0.2 (assuming that the area of an inverter is 1 unit) to take into account the additional area due to interconnects. The area overhead for the FSM, measured in Synopsys gate equivalents, is 187.97; this number accounts for 10 flip-flops and 61 logic gates. The maximum area overhead for the Group-end decoder is 120.53, which includes 65 logic gates (for circuit *s38417*). The hardware overhead for the architecture shown in Figure 10, excluding the LFSR and the ROM, is 834.07 for the circuit *s38417*, including 47 flip-flops and 158 gates. This amounts to only 2.04% of the original circuit. If LFSR reseeding is used instead of the encoding procedure based on extracted sequences, then the area overhead for the decode architecture, excluding the LFSR and the ROM, is 316.42, including 14 flip-flops and 106 gates; this amounts to only 0.7% of the original circuit.

To extend the BIST architecture to multiple scan chains, the FSM needs to be modified to generate m data bits in each scan clock cycle, where m is the number of scan chains. The data in the ROM needs to be reorganized as discussed above, but without any change in the storage requirements. The other components are the same as for the single scan chain architecture. The hardware overhead for the extended architecture for four scan chains, excluding the LFSR and the ROM, is 870.95 for the circuit *s38417*, including 49 flip-flops and only 171 gates. When LFSR reseeding is used, the area overhead for multiple scan chains is the same as the case of single scan chain, excluding the LFSR, the phase-shifter, and the ROM.

5. EXPERIMENTAL RESULTS

In this section, we present experimental results for the seven largest ISCAS-89 benchmark circuits. Table I presents the first set of results. We first apply 10,000 pseudorandom patterns to the CUT. Next we use the ATPG tool *Atalanta* to generate a set of deterministic test patterns for the remaining faults. The number of extracted sequences is four for all the circuits. After sequence extraction, 10,000 semirandom patterns are generated and applied to the CUT. Finally, we use *Atalanta* to generate deterministic patterns for the remaining faults and we encode the deterministic patterns based on the extracted sequences.

The number of bits needed for storing the extracted sequences on-chip is listed in the sixth column (ROM). For the circuits *s5378* and *s38584*, the amount

Table I. Results for ISCAS-89 Benchmarks, Obtained with Arbitrary Scan Cell Reordering

Circuit	P_0	F_1	P_1	Grp	ROM (bits)	F_2	P_2	$FC_2(\%)$	T_E (bits)	T_{seed} (bits)
s5378	4563	51	38	8	136	3	3	99.93	85	60
s9234	6475	735	317	8	719	81	38	98.75	1447	—
s13207	9664	624	366	16	1021	43	30	99.56	610	600
s15850	11336	667	248	16	962	2	2	99.98	36	40
s35932	35110	0	—	—	—	—	—	—	—	—
s38417	31015	2245	1006	32	1899	105	71	99.66	2301	—
s38584	34797	448	271	32	1007	34	28	99.90	668	560

P_0 : Number of total patterns.

F_1 : Number of faults left undetected by 10000 pseudorandom patterns.

P_1 : Number of test patterns generated by Atalanta for the undetected faults.

Grp : Number of groups for the sequence extraction procedure.

ROM : Number of bits used to store the extracted sequences.

F_2 : Number of remaining faults after application of 10000 semirandom patterns.

P_2 : Number of test vectors that need to be encoded.

FC_2 : Fault coverage obtained using only the first two phases.

T_E : Number of bits needed for encoding the deterministic test patterns.

T_{seed} : Number of bits needed to store the seeds for LFSR reseeding.

of stored data is less than the test data volume corresponding to just one test pattern. For the two worst cases, that is, for s9234 and s38417, the on-chip storage requirements are less than the test data volume for three test patterns. For circuits s5378 and s15850, only three and two deterministic patterns need to be encoded, respectively. The CPU times for the computation range from 7 min to 19 min on a 1.4-GHz Pentium 4 PC with 512 MB of memory.

Only a small number of test patterns need to be encoded for deterministic test, and only these patterns are required to be fed through the tester channel (column T_E in Table I). The pseudorandom test and semirandom test can be run at higher speed since no data is required from the tester in these stages.

If LFSR reseeding is used, the storage requirement for the LFSR seeds (T_{seed}) is listed in the last column. For circuits s5378, s13207, s15850, and s38584, the deterministic patterns contain very few specified bits and they can be encoded using LFSR reseeding with a 20-stage LFSR, resulting in even less test data volume. For s9234 and s38417, the maximum numbers of specified bits are 51 and 80, respectively. Hence the proposed encoding method based on the extracted sequences should be used for these circuits.

Table II compares the storage requirements of the proposed approach with test vector encoding using partial LFSR reseeding [Krishna et al. 2001], BIST based on reseeding of folding counter [Hellebrand et al. 2000], and two-dimensional test data compression [Liang et al. 2001]. The storage requirements reported for the proposed approach include the data for storing the extracted sequences and the data for encoding the deterministic patterns. The results presented in the literature for these methods also rely on 10,000 initial pseudorandom patterns to eliminate the easy-to-detect faults. The proposed approach requires less storage than the partial LFSR reseeding method [Krishna et al. 2001]. Compared to BIST based on reseeding of folding counter,

Table II. Comparison of Storage (in Bits) Required for Various BIST Methods

Circuit	Partial Reseeding [Krishna et al. 2001]	Reseeding of Folding Counter [Hellebrand et al. 2000]	2-D Compression [Liang et al. 2001]	Proposed Approach (ROM+ T_E)	Proposed Approach (ROM+ T_{seed})
s5378	502	132	196	221	196
s9234	5013	2310	3800	2166	—
s13207	3008	247	1044	1631	1621
s15850	5204	2403	3360	998	1002
s38417	24513	6802	11214	4200	—
s38584	2942	660	2891	1675	1567

Table III. Comparison with Jas et al. [2001]

Circuit	Hybrid BIST [Jas et al. 2001]			Proposed Approach		
	On-Chip Storage Requirement (bits)	Encoded Data Volume (bits)	Total Test Data Volume (bits)	On-Chip Storage Requirement (bits)	Encoded Data Volume T_E (bits)	Total Test Data Volume (bits)
s5378	N/A	N/A	N/A	88	13	101
s9234	452	865	1317	620	1178	1798
s13207	168	263	431	263	12	275
s15850	436	1070	1506	456	108	564
s38417	2336	4680	7016	1797	2053	3850
s38584	712	961	1673	417	826	1243

the proposed method provides better results in three out of six cases. Note, however, that width compression is used in Hellebrand et al. [2000] to reduce test data volume. While width compression can indeed reduce test data volume, it requires a special scan out procedure to shift out the test responses. Compared to two-dimensional test data compression, the proposed approach provides better or equal results in five out of six cases. We only compare the storage requirements of these techniques. Since the proposed BIST approach requires an on-chip ROM and other control logics, the area overhead may be higher than that for other techniques, especially when compared with non-BIST techniques.

Table III compares the proposed approach with hybrid BIST based on weighted pseudorandom patterns Jas et al. [2001]. For both methods, the data volume is divided into two parts, on-chip storage and encoded test data. The results of Jas et al. [2001] relied on 32,000 initial pseudorandom patterns to eliminate easy to detect faults. Thus we also applied 32,000 pseudorandom patterns to the CUT in the first stage of the test session. The number of semi-random patterns applied to the CUT is kept at 10,000. Compared to hybrid BIST based on weighted pseudorandom patterns, the proposed approach requires less on-chip storage for the larger circuits although it needs slightly more on-chip storage for the smaller circuits. The encoded data volume of the proposed approach is less than that of Jas et al. [2001] for all but one circuit. The total test data volume, which includes the amount of on-chip stored data and the amount of encoded data transferred from the tester, is also shown in

Table IV. Results for ISCAS-89 Benchmarks, Obtained with Constrained Scan Cell Reordering

Circuit	P_0	F_1	P_1	$maxOffset$	Grp	ROM (bits)	F_2	P_2	T_E (bits)	T_{seed} (bits)
s5378	4563	51	38	0	8	257	11	9	208	180
				10	8	217	13	11	252	220
				20	8	212	12	10	237	200
s9234	6475	735	317	0	8	869	114	59	2755	—
				10	8	825	148	67	2776	—
				20	8	834	130	71	3089	—
s13207	9664	624	366	0	16	1918	27	13	569	260
				10	16	1407	15	9	414	300
				20	16	1280	6	6	354	120
s15850	11336	667	248	0	16	2019	57	18	629	540
				10	16	1778	40	25	947	1000
				20	16	1699	63	23	917	690
s38417	31015	2245	1006	0	32	4879	83	59	1952	—
				10	32	4442	74	61	2302	—
				20	32	4099	66	49	1614	—
s38584	34797	448	271	0	32	3804	41	34	2236	1836
				10	32	2902	63	38	2046	2052
				20	32	2203	40	27	1626	1458

Table III for both methods. The proposed method leads to lower total test data volume compared to Jas et al. [2001] in four out of five cases.

We have implemented the BIST logic using Synopsys tools and reported the results in Section 5.3. The area overhead appears to be reasonable and, and based on limited published data, of the same magnitude as the other four methods.

Finally, Table IV presents results obtained with constrained scan cell reordering for the ISCAS-89 benchmark circuits. During position grouping, constraints are introduced as described in Section 3. When $maxOffset$ is set to 0, scan cell reordering is disabled. As expected, in this case, more storage is required for the extracted sequences and the encoded data. Compared to the results for Hellebrand et al. [2000] and Liang et al. [2001] listed in Table II, the storage requirement is generally higher, but scan cell reorganization and a special response unloading sequence are required in Hellebrand et al. [2000]. The approach in Liang et al. [2001] requires iterative fault simulation and ATPG to reduce the number of seeds and thereby the test data volume; this can be computationally expensive for large circuits. When constrained scan cell reordering is enabled, that is, $maxOffset > 0$, less storage requirement is achieved. Compared with Hellebrand et al. [2000] in Table II, for s15850 and s38417 the proposed method can achieve smaller data volume. Compared with Liang et al. [2001], in three out of six cases, the proposed approach provides better results.

Table IV demonstrates that there is a tradeoff between storage size and the number of groups. Larger values of $maxOffset$ provide more flexible scan cell reordering and hence result in better performance in most cases. Especially for larger circuits such as s38417 and s38584, larger $maxOffset$ leads to significant reduction in data volume.

6. CONCLUSION

We have presented a new BIST approach for full-scan designs that extracts the most frequently occurring sequences from deterministic test patterns. The extracted sequences are stored on-chip. The test session consists of three stages: pseudorandom test, semirandom test, and deterministic test. The semirandom patterns are generated based on the sequences extracted from deterministic patterns and they are more likely to cover the hard-to-detect faults. The deterministic wrap-up patterns are encoded either based on the stored sequences, or using LFSR reseeding, to reduce the test data volume. Experimental results for the ISCAS-89 benchmark circuits demonstrate that the proposed approach often requires less on-chip storage and test data volume than recently proposed methods.

REFERENCES

- BAYRAKTAROGU, I. AND ORAILOGLU, A. 2001. Test volume and application time reduction through scan chain concealment. In *Proceedings of the ACM/IEEE Design Automation Conference*. 151–155.
- CHANDRA, A. AND CHAKRABARTY, K. 2003. Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes. *IEEE Trans. Comput.* 52, 1076–1088.
- EVERITT, B. S., LANDAU, S., AND LEESE, M. 2001. *Cluster Analysis*. Oxford University Press Inc., New York, NY.
- GONCIARI, P. T. AND NICOLICI, B. A.-H. 2003. Variable-length input Huffman coding for system-on-a-chip test. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 22, 783–796.
- HELLEBRAND, S., LIANG, H.-G., AND WUNDERLICH, H.-J. 2000. A mixed-mode BIST scheme based on reseeding of folding counters. In *Proceedings of the International Test Conference*. 778–784.
- JAS, A., KRISHNA, C. V., AND TOUBA, N. A. 2001. Hybrid BIST based on weighted pseudo-random testing: A new test resource partitioning scheme. In *Proceedings of the VLSI Test Symposium*. 2–8.
- JAS, A. AND TOUBA, N. A. 1998. Test vector decompression via cyclical scan chains and its application to testing core-based design. In *Proceedings of the International Test Conference*. 458–464.
- KOENEMANN, B. 1991. LFSR-Coded Test Patterns for Scan Designs. In *Proceedings of the European Test Conference*. 237–242.
- KRISHNA, C. V., JAS, A., AND TOUBA, N. A. 2001. Test vector encoding using partial LFSR reseeding. In *Proceedings of the International Test Conference*. 885–893.
- KRISHNA, C. V. AND TOUBA, N. A. 2002. Reducing test data volume using LFSR reseeding withseed compression. In *Proceedings of the International Test Conference*. 321–330.
- LI, L., CHAKRABARTY, K., AND TOUBA, N. A. 2003. Test data compression using dictionaries with selective entries and fixed-length indices. *ACM Trans. Des. Automat. Electron. Syst.* 8, 470–490.
- LIANG, H.-G., HELLEBRAND, S., AND WUNDERLICH, H.-J. 2001. Two-dimensional test data compression for scan-based deterministic BIST. In *Proceedings of the International Test Conference*. 894–902.
- RAJSKI, J., KASSAB, M., MUKHERJEE, N., TAMARAPALLI, N., TYSZER, J., AND QIAN, J. 2002. Embedded deterministic test for low-cost manufacturing test. In *Proceedings of the International Test Conference*. 301–310.
- RAJSKI, J., TYSZER, J., AND ZACHARIA, N. 1998. Test data decompression for multiple scan designs with boundary scan. *IEEE Trans. Comput.* 47, 1188–1200.
- REDA, S. AND ORAILOGLU, A. 2002. Reducing test application time through test data mutation encoding. In *Proceedings of the Design Automation and Test in Europe Conference*. 387–393.
- REDDY, S. M., MIYASE, K., KAJIHARA, S., AND POMERANZ, I. 2002. On test data volume reduction for multiple scan chain design. In *Proceedings of the VLSI Test Symposium*. 103–108.

- SCHOTTEN, C. AND MEYR, H. 1995. Test point insertion for an area efficient BIST. In *Proceedings of the International Test Conference*. 515–523.
- TOUBA, N. A. AND McCLUSKEY, E. J. 1996. Altering a pseudo-random bit sequence for scan based BIST. In *Proceedings of the International Test Conference*. 167–175.
- TSAI, K.-H., HELLEBRAND, S., RAJSKI, J., AND MAREK-SADOWSKA, M. 1997. STARBIST: Scan Auto-correlated Random Pattern Generation. In *Proceedings of the Design Automation Conference*. 472–477.
- WANG, S. 2001. Low hardware overhead scan based 3-weight weightedrandom BIST. In *Proceedings of the International Test Conference*. 868–877.
- WUNDERLICH, H.-J. AND KIEFER, G. 1996. Bit-flipping BIST. In *Proceedings of the International Conference on Computer-Aided Design*. 337–343.
- WURTENBERGER, A., TAUTERMANN, C. S., AND HELLEBRAND, S. 2004. Data compression for multiple scan chains using dictionaries with corrections. In *Proceedings of the International Test Conference*. 926–935.

Received July 2005; revised April 2006, July 2006; accepted July 2006