Hybrid BIST Based on Repeating Sequences and Cluster Analysis[1]

Lei Li[†] and Krishnendu Chakrabarty[‡]

[†]Wireless & Mobile Systems Group
Freescale Semiconductor, Inc.
Austin, TX 78735, USA
leili@freescale.com

[‡]Department of Electrical & Computer Engineering
Duke University
Durham, NC 27708, USA
krish@ee.duke.edu

Abstract

We present a hybrid BIST approach that extracts the most frequently occurring sequences from deterministic test patterns; these extracted sequences are stored on-chip. We use cluster analysis for sequence extraction, and encode deterministic patterns on the basis of the stored sequences. 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.

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 are presented in [1, 3, 9, 12, 14]. In BIST solutions, test patterns are generated by an on-chip pseudo-random pattern generator, usually a linear-feedback shift-register (LFSR). A number of BIST techniques based on test point insertion [13], reseeding [5, 10, 11], bit-flipping [16], and weighted random pattern testing [15] have been proposed. Deterministic test patterns are applied in BIST by either controlling the state of the pattern generator [5, 7, 10, 11] or by altering the output of the pattern generator [15, 16].

Techniques based on the combination of data compression and BIST have also been developed recently [6, 8]. The hybrid BIST scheme presented in [6] applies weighted pseudo-random 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 look-up table. In [8], the seeds for the LFSR are compressed using statistical coding.

In this paper, we present a hybrid BIST approach 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, pseudo-random patterns generated by an LFSR are applied to the CUT to detect easy-to-detect faults. In the second stage, semi-random patterns are generated by randomly selecting some of the stored sequences and flipping some of their data bits. Since the semi-random patterns are generated based on the sequences extracted from deterministic patterns, they are more likely to detect the hard-to-detect faults than the pseudo-random patterns. Faults that are not detected by pseudo-random and semi-random patterns are detected by deterministic patterns in the third stage. The deterministic patterns are encoded on the basis of the stored sequences to reduce the test data volume, and are decoded/generated during the third stage of the test session. The overall approach is similar to [14], where repeating scan slices are stored in a dictionary, and corrections (bit-flips) are applied to selected scan slices.

The rest of this paper is organized as follows. Section 2 presents the proposed hybrid 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 paper.

2 Proposed Approach

The proposed mixed-mode 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 semi-random patterns or encode deterministic patterns, based on how we select the sequences and the bits to be flipped.

Similar to other BIST approaches, we first apply a number of pseudo-random patterns to the CUT to detect easy-to-detect faults. Next we generate deterministic patterns for the remaining faults, 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 data bits in the test vectors are reorganized such that the positions

[1]This research was supported in part by the National Science Foundation under grants CCR-9875324 and CCR-0204077.

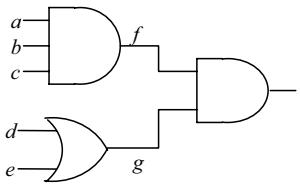

Figure 1. A simple circuit used to motivate the clustering-based approach.

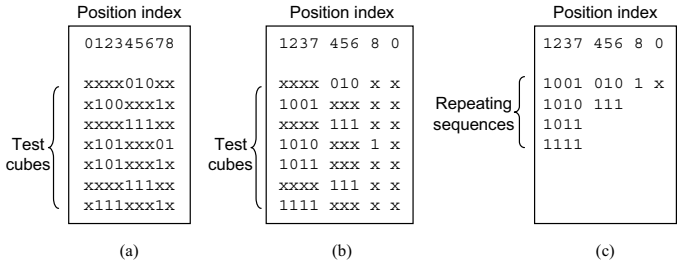

Figure 2. An example to illustrate the extraction of frequently occurring patterns.

that are specified in the same test vectors are grouped together. This is illustrated in Figure 2, 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 2, the position groups and the repeating sequences are easy to identify, and the number of groups and sequences are 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 semi-random 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 semi-random patterns shifted into the scan chain are obtained by doing an exclusive-or between the randomly selected sequences

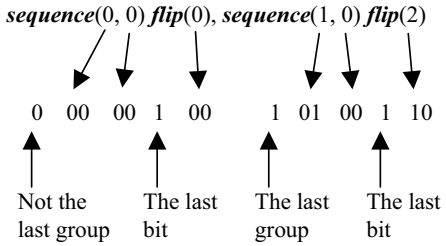

Figure 3. The encoded data for the deterministic test cube "x000011x1".

and the flip indication sequence.

After a number of semi-random test patterns are generated, we run fault simulation to determine the faults that are not detected by both the pseudo-random patterns and the semi-random patterns. Then we use an ATPG tool to generate deterministic test cubes for the remaining undetected faults. 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 2(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 jth sequence in the ith group, and *flip*(k) means flipping the kth 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 3.

3 Cluster Analysis

Cluster analysis is used for numerical classification in many fields [2]. We use an agglomerative clustering algorithm for both position grouping and sequence merging [2]. The test set is first represented as a graph, to which the agglomerative algorithm is applied. The basic idea is to first view each node in the graph as a cluster, then continuously merge pairs of closest clusters until a predefined threshold is reached. This threshold can be either the number of clusters in the system, referred to as $maxClusterNum$, or the minimum distance between a pair of clusters that can be merged, referred to as $minMergeDistance$.

The details of the agglomerative algorithm are described in Figure 4. The input to the procedure is the system $\mathcal{S}$ containing a given number of clusters, where each cluster corresponds to a node in the graph. The distance between each pair of clusters is calculated and stored in the distance matrix D. In each loop of the procedure, the minimum distance and its associated cluster indices i and j are found. If the minimum distance is not

Procedure *Agglomerate* ($\mathcal{S}$, $maxClusterNum$, $minMergeDistance$)

```
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, |);
9       deleteRomColumn(D, j); updateDistance(D, j);
10    end if; end while;
```

Figure 4. The agglomerative algorithm.

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.

Although the agglomerative algorithm is used for both position grouping and sequence merging, the calculation of the distance between a pair of clusters is different for the two problems. In position grouping, there are L nodes in the graph, where L is the length of the scan chain, and each node represents a position in the scan chain. Each node j has an associated index set I_j that records all the test vectors for which this position is specified. The distance between two nodes i and j is calculated as the number of test vectors in which one node/position is specified while the other is unspecified, i.e., it simply equals $|(I_i - I_j) \bigcup (I_j - I_i)|$. As described above, the initial cluster system contains L clusters, each of which consists of one node. In order to calculate the distance between two clusters, we define the centroid of a cluster as follows. The *centroid* of a cluster is a dummy node whose associated index set is the union of the associated index sets of all the nodes 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 is specified.

Figure 5 illustrates position grouping. We use the same test data as in Figure 2. However, we set the threshold $maxClusterNum$ to 2, excluding a group that contains all the positions that are unspecified in all the test vectors. In this example, position 0 is unspecified in all the 7 test vectors; therefore it is not considered for cluster analysis. As shown in Figure 5(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 5(c). First, all pairs of clusters with mutual distance 0 are merged; the remaining clusters and their pairwise distances are shown in Figures 5(d) and 5(e). Next, the Clusters 1 and 8 are merged because the distance between them is the minimum entry in

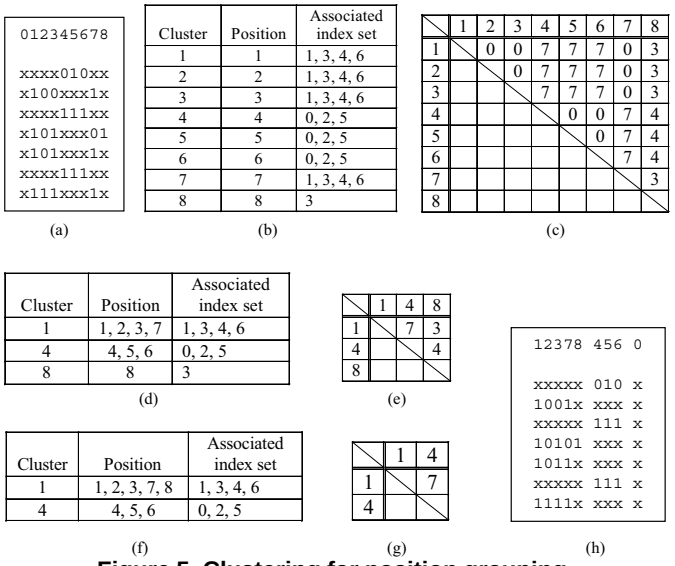

Figure 5. Clustering for position grouping.

the distance matrix. After the merging of clusters 1 and 8, the $maxClusterNum$ threshold is reached. Figures 5(f) and 5(g) show the eventual outcome of cluster analysis. The positions are then grouped as shown in Figure 5(h).

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 2. For each group, i.e., a submatrix, an initial cluster system is generated and the agglomerative algorithm is used for sequence merging. The number of nodes for a group is equal to the number of test vectors in which the group appears, i.e., the number of rows in the submatrix that contain at least one specified bit. A 3-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 exactly one node.

We define the centroid of a cluster as a dummy node associated with a three-value 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 6 illustrates cluster analysis for sequence merg-

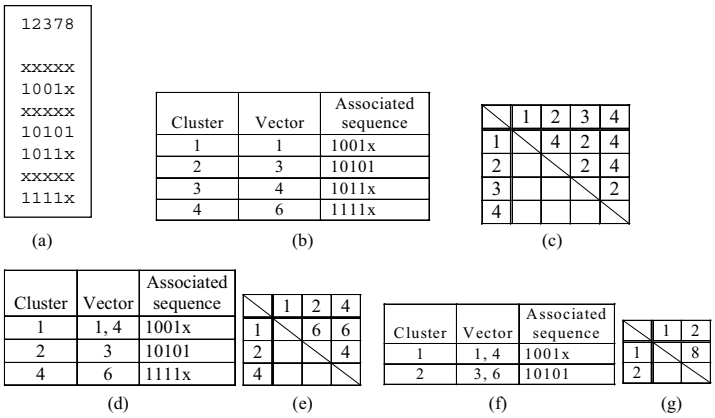

Figure 6. Clustering for sequence merging.

ing. The input data in this example corresponds to the first group of sequences in Figure 5(h). We set the threshold $maxClusterNum$ to 2. Figure 6(b) shows the initial clusters and the sequences associated with their centroids. The distance matrix is shown in Figure 6(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 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".

4 Proposed BIST Architecture

Figure 7 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.

In the first stage of the test session, the signal $Select\_random$ is set to 0. Hence the pseudo-random 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.

In the second stage, the signal $Select\_random$ is set to 1, and $Select\_flip$ is set to 0. 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 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

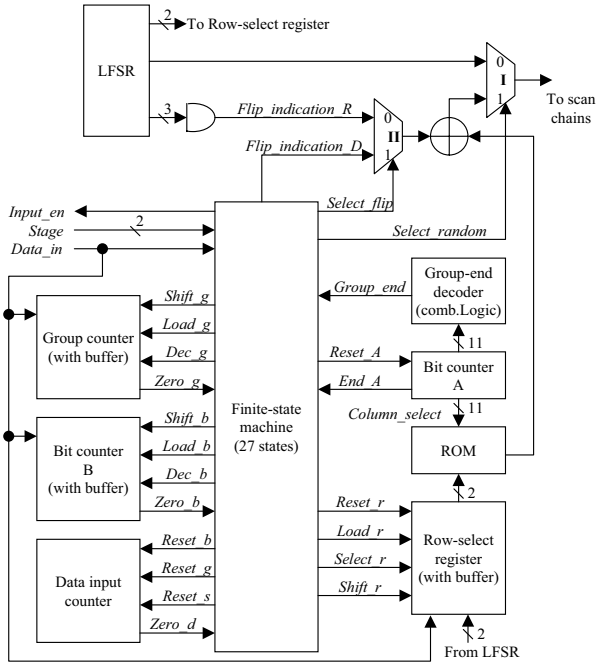

Figure 7. Proposed BIST architecture.

counter A and it can be larger than C. In this case, the 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. 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 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.

For multiple scan chains, the ROM is reorganized as shown

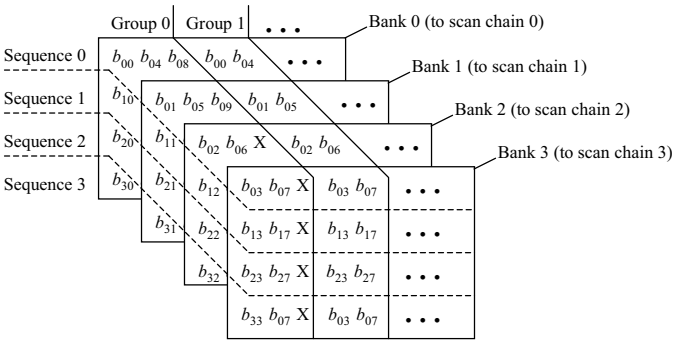

Figure 8. Organization of the stored sequences.

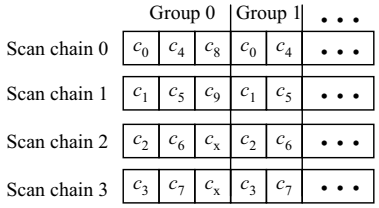

Figure 9. Reorganization of the scan cells.

in Figure 8, which corresponds to the case of 4 scan chains for the CUT. The number of banks in the ROM equals the number of scan chains, i.e., four in this example. In each scan cycle, one bit from each bank is shifted into the scan chain. All of the four 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 is also shown in the figure, where $b_{i,j}$ denotes the jth bit of the ith 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 ith sequence are placed in the ith 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.

The cells in the scan chains also need to be reorganized, i.e., 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 9 shows the reorganization of the scan cells corresponding to the data organization in the ROM shown in Figure 8. In Figure 9, Group 0 contains 10 scan cells. The first 4 cells are placed at the first position of each scan chain, the next 4 cells are place at the second position of each scan chain, and the remaining 2 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.

Table 1. Results for ISCAS-89 benchmarks.

Circuits	No. of total patterns	N_1	N_2	No. of groups for the sequence extraction procedure	No. of bits used to store the extracted sequences	No. of remaining faults after application of 10000 semi-random patterns	No. of test vectors need to be encoded	No. of bits needed for encoding test patterns
s5378	4563	51	38	8	136	3	3	85
s9234	6475	735	317	8	719	81	38	1447
s13207	9664	624	366	16	1021	43	30	610
s15850	11336	667	248	16	962	2	2	36
s35932	35110	0	—	—	—	—	—	—
s38417	31015	2245	1006	32	1899	105	71	2301
s38584	34797	448	271	32	1007	34	28	668

N_1: No. of faults left undetected by 10000 pseudo-random patterns
N_2: No. of test patterns generated by Atalanta for the undetected faults

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, which includes 10 flip-flips 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 7, 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% overhead.

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.

5 Experimental Results

In this section, we present experimental results for the seven largest ISCAS-89 benchmark circuits. Table 1 presents the first set of results. We first apply 10000 pseudo-random 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, 10000 semi-random 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 seventh column. For the circuits

s5378 and s38584, the amount of stored data is less than the test data volume corresponding to just one test pattern. For the two worst cases, i.e., for s9234 and s38417, the on-chip storage requirements are less than the test data volume for three test patterns. For the circuits s5378 and s15850, only three and two deterministic patterns need to be encoded, respectively. The CPU times for the computation range from 7 minutes to 19 minutes on a 1.4 GHz Pentium 4 PC with 512 MB of memory.

Only a small number of deterministic test patterns need to be encoded, and only these patterns are required to be fed through the tester channel. The pseudo-random test and semi-random test can be run at higher speed since no data is required from the tester in these stages.

Table 2 compares the storage requirements of the proposed approach with test vector encoding using partial LFSR reseeding [7], BIST based on reseeding of folding counter [5], and two-dimensional test data compression [10]. The storage requirement 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 10000 initial pseudo-random patterns to eliminate the easy-to-detect faults. The proposed approach requires less storage than partial LFSR reseeding method [7]. Compared to BIST based on reseeding of folding counter, the proposed method provides better results in three out of six cases. Note however that width compression is used in [5] to reduce test data volume. While width compression can indeed reduce test data volume, it requires a special scan out procedure is needed to shift out the test responses. Compared to two-dimensional test data compression, the proposed approach provides better results in four out of six cases.

Table 3 compares the proposed approach with hybrid BIST based on weighted pseudo-random patterns [6]. For both methods, the data volume is divided into two parts, on-chip storage and encoded test data. The results of [6] relies on 32000 initial pseudo-random patterns to eliminate easy to detect faults. Thus we also applied 32000 pseudo-random 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 10000. Compared to hybrid BIST based on weighted pseudo-random 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 [6] for all but one circuit.

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.

6 Conclusion

We have presented a new hybrid BIST approach that uses cluster analysis to extract the most frequently occurring sequences deterministic test patterns. 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.

Table 2. Comparison of storage (in bits) required for various BIST methods.

Circuit	Partial reseeding [7]	Reseeding of folding counter [5]	2-D compression [10]	Proposed approach
s5378	502	132	196	221
s9234	5013	2310	3800	2166
s13207	3008	247	1044	1631
s15850	5204	2403	3360	998
s38417	24513	6802	11214	4200
s38584	2942	660	2891	1675

Table 3. Comparison with [6].

	Hybrid BIST [6]		Proposed approach	
Circuit	On-chip storage requirement (bits)	Encoded data volume (bits)	On-chip storage requirement (bits)	Encoded data volume (bits)
s5378	N/A	N/A	88	13
s9234	452	865	620	1178
s13207	168	263	263	12
s15850	436	1070	456	108
s38417	2336	4680	1797	2053
s38584	712	961	417	826

References

[1] A. Chandra and K. Chakrabarty, "Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes", *IEEE Trans. Computers*, vol. 52, pp. 1076–1088, August 2003.

[2] B. S. Everitt, S. Landau, and M. Leese, *Cluster Analysis*, Oxford University Press Inc., New York, NY, 2001.

[3] P. T. Gonciari, B. Al-Hashimi and N. Nicolici, "Improving compression ratio, area overhead, and test application time for system-on-a-chip test data compression/decompression," *Proc. DATE Conf.*, pp. 604–611, 2002.

[4] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits," *Proc. ICCAD*, pp. 283-289, 1998.

[5] S. Hellebrand, H.-G. Liang and H.-J. Wunderlich, "A mixed-mode BIST scheme based on reseeding of folding counters," *Proc. ITC*, pp. 778–784, 2000.

[6] A. Jas, C. V. Krishna and N. A. Touba, "Hybrid BIST based on weighted pseudo-random testing: a new test resource partitioning scheme," *Proc. VTS*, pp. 2–8, 2001.

[7] C. V. Krishna, A. Jas and N. A. Touba, "Test vector encoding using partial LFSR reseeding," *Proc. ITC*, pp. 885–893, 2001.

[8] C. V. Krishna and N. A. Touba, "Reducing test data volume using LFSR reseeding with seed compression," *Proc. ITC*, pp. 321–330, 2002.

[9] L. Li and K. Chakrabarty, "Test data compression using dictionaries with fixed-length indices," *Proc. VTS*, pp. 219–224, 2003.

[10] H.-G. Liang, S. Hellebrand and H.-J. Wunderlich, "Two-dimensional test data compression for scan-based deterministic BIST," *Proc. ITC*, pp. 894–902, 2001.

[11] J. Rajski, J. Tyszer and N. Zacharia, "Test data decompression for multiple scan designs with boundary scan," *IEEE Trans. Computers*, vol. 47, pp. 1188–1200, November 1998.

[12] J. Rajski et al., "Embedded deterministic test for low-cost manufacturing test," *Proc. ITC*, pp. 301–310, 2002.

[13] C. Schotten and H. Meyr, "Test point insertion for an area efficient BIST," *Proc. ITC*, pp. 515–523, 1995.

[14] A. Wuertenberger, C. S. Tautermann and S. Hellebrand, "Data compression for multiple scan chains using dictionaries with corrections", *Proc. ITC*, pp. 926–934, 2004.

[15] S. Wang, "Low hardware overhead scan based 3-weight weighted random BIST," *Proc. ITC*, pp. 868–877, 2001.

[16] H.-J. Wunderlich and G. Kiefer, "Bit-flipping BIST," *Proc. ICCAD*, pp. 337-343, 1996.