Storage-Based Logic Built-In Self-Test with Partitioned Deterministic Compressed Tests

Irith Pomeranz

***Abstract*— Logic built-in self-test ($LBIST$) is important for in-field testing. In a storage-based $LBIST$ approach, deterministic test data are stored on-chip and used for applying tests that are closer to deterministic tests than pseudo-random tests. Using the same stored test data for applying several different tests allows the volume of test data stored on-chip to be reduced, and the fault coverage to be increased. This observation was applied earlier in two ways: (1) by complementing bits of stored test data or applied tests to form additional tests, or (2) by forming different tests from different combinations of stored test data entries that are obtained by partitioning deterministic tests. Partitioning was applied earlier to uncompressed deterministic tests. In this article, partitioning is applied for the first time to compressed deterministic tests. Under the resulting $LBIST$ approach, tests are formed on-chip using pseudo-random combinations of partitioned compressed tests. A software procedure is described for deriving a reduced set of test data entries for on-chip storage. With compressed tests, the storage requirements are already reduced, and they are reduced further by the software procedure. Experimental results demonstrate the effectiveness of this $LBIST$ approach considering both single stuck-at and single-cycle gate-exhaustive faults in benchmark circuits.**

***Index Terms*— Full-scan, linear-feedback shift-register ($LFSR$), logic built-in self-test ($LBIST$), on-chip test generation, test data compression.**

I. Introduction

Logic built-in self-test ($LBIST$) is effective for in-field testing to address defects that escape manufacturing testing or occur during the lifetime of a chip [1]-[19]. In a storage-based $LBIST$ approach, deterministic test data are stored on-chip and used for test application [4], [14], [15], [17], [18]. The test data may be uncompressed [4], [17], [18] or compressed [14], [15]. With compressed test data the $LBIST$ approach uses the on-chip decompression logic for test application. In both cases, storage of deterministic test data results in applied tests that are closer to deterministic tests than the pseudo-random tests that are typically used for $LBIST$ [1]. With more effective tests, the fault coverage is increased, or the number of tests is reduced, allowing $LBIST$ to be used during system startup or idle intervals.

Using the same stored test data for applying several different tests allows the volume of test data stored on-chip to be reduced, and the fault coverage to be increased. This observation was used earlier in two ways, as discussed next.

The first approach was considered in [2], [3], [10], [14], [15], [20], [21], and consists of complementing one or more bits in the stored test data or applied tests. Complementation of bits allows a larger number of more effective tests to be formed. Complementation was shown to be effective for $LBIST$ as well as test data compression.

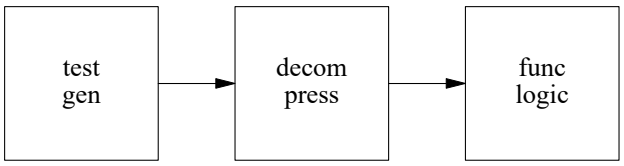

Fig. 1. On-chip test generation logic.

The second approach was applied to uncompressed deterministic tests, and described in [4], [17], [18]. Under this approach, uncompressed deterministic tests are partitioned, e.g., into scan vectors, and the partitioned test data entries are stored on-chip. Tests are formed on-chip using different combinations of the stored test data entries. Three options were considered for the combination of test data entries: using all the possible combinations, storing effective combinations on-chip, or using linear-feedback shift-registers ($LFSR$s) for forming pseudo-random combinations. With storage of partitioned deterministic test data entries, and combinations formed on-chip, the extent of the variations possible in applied tests is significant. This allows significant reductions in the volume of stored test data, and a significant increase in the fault coverage to be achieved. The use of pseudo-random combinations fits well with $LBIST$. Although a lower number of tests is needed if the combinations are stored, the storage requirements are also higher in this case. Adding deterministic combinations to pseudo-random combinations can help reduce the number of tests that need to be applied, and increase the fault coverage, at the cost of a limited increase in the storage requirements.

In this article, partitioning of test data, and the use of combinations for forming tests, are applied for the first time to compressed deterministic tests. The resulting $LBIST$ approach is illustrated by Figure 1. An on-chip test generation logic, shown on the left in Figure 1, produces compressed tests from partitioned test data. The compressed tests are used as input to the on-chip decompression logic. The on-chip decompression logic uses the tests to produce scan vectors that are shifted into the scan chains of the functional logic of the circuit, shown on the right in Figure 1. Partitioning as suggested in this article allows significant reductions in the stored test data to be achieved compared with an already compressed deterministic test set.

For the discussion in this article, the test data compression logic includes an $LFSR$, and tests are compressed into seeds for the $LFSR$. Considering a seed as a vector, the seeds are

Irith Pomeranz is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, U.S.A. (e-mail: pomeranz@ecn.purdue.edu).

The work was supported in part by NSF Grant No. CCF-2041649.

TABLE I
EXAMPLE SET OF SUBVECTORS

i	v_i
0	10110
1	10010
2	00000
3	10011
4	11010
5	00011
6	00001
7	11101
8	11111
9	10000
10	01010
11	01111

TABLE II
EXAMPLE SEEDS FOR APPLIED TESTS

i	i_0	i_1	i_2	i_3	$s_{i,0}$	$s_{i,1}$	$s_{i,2}$	$s_{i,3}$
0	8	10	9	6	11111	01010	10000	000[01]
1	3	1	4	5	10011	10010	11010	000[11]
2	11	9	7	6	01111	10000	11101	000[01]
3	6	2	1	2	00001	00000	10010	000[00]
4	11	11	2	5	01111	01111	00000	000[11]
5	0	8	5	11	10110	11111	00011	011[11]
6	10	1	1	2	01010	10010	10010	000[00]
7	7	9	2	11	11101	10000	00000	011[11]
8	11	5	9	6	01111	00011	10000	000[01]
9	2	2	8	1	00000	00000	11111	100[10]

partitioned into subvectors of equal length. Subvectors are stored and combined on-chip to form seeds for the decompression logic, and apply tests to the circuit. For simplicity the following assumptions are made. (1) A test is compressed into a single seed. (2) Subvectors are combined pseudo-randomly only. It should be noted that deterministic compressed tests are partitioned deterministically into subvectors. Only the combination of subvectors on-chip is performed pseudo-randomly.

A software procedure is described for deriving a reduced set of subvectors, V, from a compressed deterministic test set, S_{sa}, for single stuck-at faults. The software procedure is run a-priori to compute the stored test data entries. The test data compression logic is that for which S_{sa} was computed. Tests applied on-chip use pseudo-random combinations of subvectors from V to form seeds for the decompression logic. The tests target both single stuck-at faults and single-cycle gate-exhaustive faults. This demonstrates the ability to increase the fault coverage beyond that achieved by S_{sa}. Experimental results for benchmark circuits demonstrate significant reductions in the storage requirements of V compared with S_{sa}, and significant increases in the coverage of single-cycle gate-exhaustive faults. As with other $LBIST$ approaches, and to simplify the discussion in this article, it is assumed that a small loss of single stuck-at fault coverage is acceptable. This can be addressed in one of several ways discussed later. In addition it is assumed that it is acceptable to apply a large number of tests that are produced by the $LBIST$ logic through pseudo-random combinations of subvectors to cover both single stuck-at and single-cycle gate-exhaustive faults.

The article is organized as follows. Section II describes the on-chip storage of subvectors and the test application process. Section III describes the software procedure for computing the stored set of subvectors. Section IV presents experimental results for benchmark circuits. Section V concludes the article.

II. ON-CHIP STORAGE AND TEST APPLICATION

The on-chip storage of subvectors and test application process are illustrated in this section by considering benchmark circuit $s1423$.

The circuit has a compressed deterministic test set S_{sa} for single stuck-at faults that consists of 55 seeds. The decompression logic for which S_{sa} is computed includes an $LFSR$ of length $L = 18$. The test set achieves complete coverage of single stuck-at faults, and a coverage of 90.381% for single-cycle gate-exhaustive faults. For this example, an 18-bit seed is partitioned into $p = 4$ subvectors of length $l = 5$. The last subvector obtained by partitioning a seed has three bits. It is padded as discussed later to obtain a 5-bit subvector, and only 5-bit subvectors are considered.

The software procedure described in Section III yields the set of subvectors $V = \{v_0, v_1, ..., v_{11}\}$ shown in Table I. With $l = 5$, the number of subvectors is bounded by $2^l = 32$. Only 12 subvectors need to be stored to form seeds that achieve complete single stuck-at fault coverage.

The set V is stored in an on-chip memory. To apply tests to the circuit, four indices of subvectors are selected pseudo-randomly, and the subvectors are used for forming seeds that initialize the $LFSR$ in the decompression logic. A total of 138 tests are needed for $s1423$ to detect all the detectable single stuck-at faults, and achieve a fault coverage of 99.214% for single-cycle gate-exhaustive faults. The seeds for the first ten tests are shown in Table II. A seed $s_i = \langle s_{i,0}, s_{i,1}, s_{i,2}, s_{i,3} \rangle$ is formed from the subvectors with indices i_0, i_1, i_2 and i_3. Only the first three bits of $s_{i,3}$ are used for test application. The remaining two bits, shown in square brackets in Table II, are discarded.

Figure 2 shows the on-chip test generation logic for $s1423$ on the left. The on-chip test generation logic drives the decompression logic on the right, which in turn drives the scan chains of the circuit (not shown in Figure 2).

The memory that stores the set V of subvectors is shown at the top of Figure 2. With $p = 4$, four multiplexers are used for selecting four subvectors that are loaded into the decompression logic to form a seed s_i. An $LSFR$, shown on the left in Figure 2, selects which subvectors will be used.

In general, let an $LFSR$ of length L be included in the decompression logic. Suppose that a seed is partitioned into p subvectors of length l. Using $p = \lceil L/l \rceil$ ensures that $l \cdot p \geq L$, and neither l nor p is larger than necessary. The on-chip test generation logic consists of the following components.
(1) A memory for $|V|$ l-bit subvectors.
(2) p l-bit multiplexers with $|V|$ data inputs and $\lceil log_2(|V|) \rceil$ select inputs.
(3) An $LFSR$ for p $\lceil log_2(|V|) \rceil$-bit random numbers.

In Figure 2, the memory consists of $|V| = 12$ l-bit subvectors with $l = 5$. There are $p = 4$ multiplexers with $|V| = 12$ data inputs and $\lceil log_2(|V|) \rceil = 4$ select inputs. Each data input has $l = 5$ bits. The $LFSR$ on the left in Figure 2 produces $p = 4$ numbers each with $\lceil log_2(|V|) \rceil = 4$ bits.

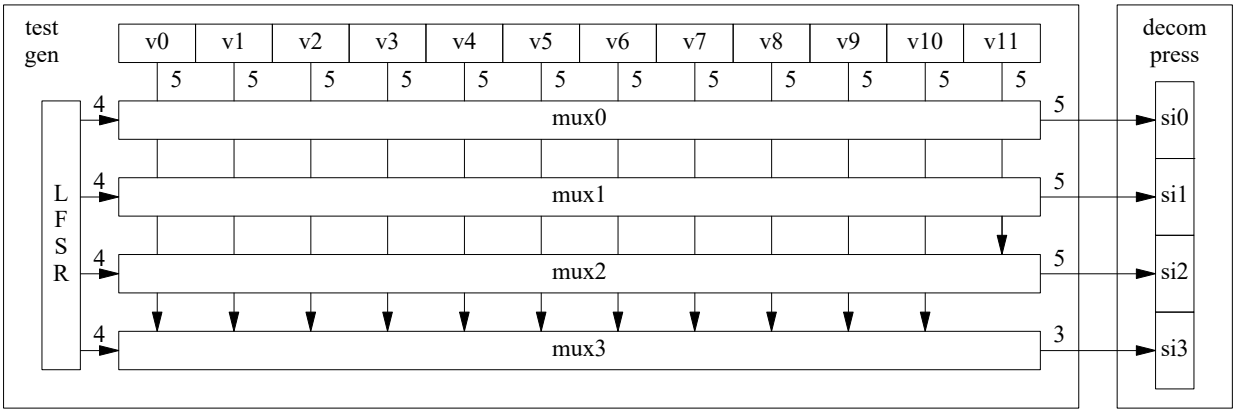

Fig. 2. On-chip test generation logic for $s1423$.

TABLE III
EXAMPLE OF INITIAL SET OF SUBVECTORS

i	$s_{i,0}$	$s_{i,1}$	$s_{i,2}$	$s_{i,3}$	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
0	01110	11100	00110	010xx	01110	11100	00110	010xx					
1	10001	10110	01001	110xx	01110	11100	00110	01001	10001	10110	110xx		
2	10010	11001	01101	111xx	01110	11100	00110	01001	10001	10110	11001	10010	01101

Given l and $p = \lceil L/l \rceil$, the software procedure described in Section III attempts to minimize the hardware overhead by minimizing $|V|$. The selection of l and p is discussed in Section IV.

If the on-chip decompression logic requires $m > 1$ seeds to form a test, the on-chip test generation logic can be used for forming m seeds for every test. For each seed, the on-chip test generation logic needs to select p l-bit subvectors as in Figure 2, and this needs to be repeated m times for every test. The software procedure can be extended to address this case as discussed at the end of Section III.

III. SOFTWARE PROCEDURE FOR COMPUTING STORED SET OF SUBVECTORS

This section describes the software procedure for computing the stored set of subvectors with given parameters l and $p = \lceil L/l \rceil$. The software procedure is run a-priori to compute the stored test data entries for on-chip test generation under the $LBIST$ approach.

The software procedure accepts a set of deterministic seeds S_{sa} computed for single stuck-at faults without considering the on-chip test generation logic. From S_{sa} the decompression logic alone, without the on-chip test generation logic, produces a set of tests denoted by T_{sa}. The test set T_{sa} targets single stuck-at faults, and achieves complete single stuck-at fault coverage. Let $S_{sa} = \{s_0, s_1, ..., s_{n-1}\}$. Based on S_{sa}, the software procedure computes an initial set V of subvectors. It uses V to compute the test set T that will be applied to the circuit by the $LBIST$ approach. The software procedure then reduces V without reducing the fault coverage of T. The various parts of the software procedure are described next. For the computation of T, the on-chip test generation logic is simulated to compute seeds, and the decompression logic is simulated to produce tests from the seeds.

A. Initial Set of Subvectors

To initialize the set V, the procedure partitions every seed $s_i \in S_{sa}$ into p l-bit subvectors. If $l \cdot p > L$, the last subvector of s_i is padded with unspecified values. After padding, let $s_i = \langle s_{i,0}, s_{i,1}, ..., s_{i,p-1} \rangle$. For $0 \leq j < p$, the procedure adds $s_{i,j}$ to V as follows.

The procedure checks if V already contains a subvector v_k that is compatible with $s_{i,j}$. If V does not contain any such subvector, the procedure adds $s_{i,j}$ to V. If it finds a subvector $v_k \in V$ that is compatible with $s_{i,j}$, it checks whether v_k has any unspecified values that are specified in $s_{i,j}$. The procedure copies such values from $s_{i,j}$ to v_k. In this case, $s_{i,j}$ is not added to V as a separate subvector.

After considering all the seeds in S_{sa}, if V contains any unspecified values, the procedure specifies them randomly.

In the example of $s1423$, with $L = 18$, $l = 5$ and $p = 4$, the first seeds in S_{sa} are shown in Table III. The seed s_0 contributes to V the subvectors $v_0 = s_{0,0}$, $v_1 = s_{0,1}$, $v_2 = s_{0,2}$, and $v_3 = s_{0,3}$.

The seed s_1 contributes to V the subvectors $v_4 = s_{1,0}$, $v_5 = s_{1,1}$, and $v_6 = s_{1,3}$. The subvector $s_{1,2}$ is compatible with v_3, causing the unspecified values of v_3 to be specified based on $s_{1,2}$.

The seed s_2 contributes to V the subvectors $v_7 = s_{2,0}$ and $v_8 = s_{2,2}$. The subvector $s_{2,1}$ is compatible with v_6, causing the unspecified values of v_6 to be specified based on $s_{2,1}$. The subvector $s_{2,3}$ is compatible with v_1, which is already fully specified.

Additional subvectors are added to the initial set V in a similar manner.

B. Set of Applied Tests

With a set V of subvectors and parameters l and p, a test set T is constructed by the $LBIST$ approach as follows.

The number of tests in T is a constant denoted by N_T. For $i = 0$, 1, ..., $N_T - 1$, a seed s_i is formed by selecting p subvectors from V pseudo-randomly. With subvector indices $\langle i_0, i_1, ..., i_{p-1} \rangle$, $s_i = \langle s_{i,0}, s_{i,1}, ..., s_{i,p-1} \rangle$ such that $s_{i,0} = v_{i_0}$, ..., $s_{i,p-2} = v_{i_{p-2}}$, and $s_{i,p-1} = v_{i_{p-1}}$ is truncated to form an L-bit seed.

The test produced by the decompression logic from s_i is denoted by t_i. The test t_i is included in T together with the indices $\langle i_0, i_1, ..., i_{p-1} \rangle$.

Two sets of target faults are considered in this article. The set F_0 is the set of single stuck-at faults that are also targeted by the set of seeds S_{sa}. The set F_1 consists of single-cycle gate-exhaustive faults. The set F_1 is used for demonstrating that extra coverage is achieved when T is applied to the circuit instead of T_{sa}.

Fault simulation is carried out for F_0 and F_1 under T. Tests that are effective in detecting target faults are included in a subset of tests denoted by $T_{eff} \subseteq T$. Forward-looking reverse order fault simulation is applied to T_{eff} to remove unnecessary tests. The test set T_{eff} is used by the software procedure to identify subvectors that contribute to the fault coverage. The test set used for on-chip test generation is T, and not T_{eff}.

For T and T_{eff}, the sets of detected faults are denoted by $D_0(T) \subseteq F_0$ and $D_1(T) \subseteq F_1$. In addition, fault simulation of F_0 and F_1 under the test set T_{sa} produced by the seeds in S_{sa} yields subsets of detected faults $D_0(T_{sa}) \subseteq F_0$ and $D_1(T_{sa}) \subseteq F_1$.

C. Reducing the Set of Subvectors

The procedure described in this section removes subvectors from V without losing fault coverage.

The procedure considers the subvectors from V one by one. When a subvector $v_{rem} \in V$ is considered, it is removed from V temporarily. With the reduced set V the procedure recomputes the test set T, and checks the effect on the fault coverage. To accept the removal of v_{rem}, the procedure requires the following two conditions to be satisfied.

(1) $|D_0(T)|$ should not decrease below its value before the removal of v_{rem}. This ensures that the stuck-at fault coverage would not decrease when a subvector is removed from V.

(2) $|D_1(T)|$ should not decrease below $|D_1(T_{sa})|$. Experimental results indicate that the single-cycle gate-exhaustive fault coverage of T is significantly higher than that of T_{sa}. Moreover, the removal of subvectors that satisfy the first condition has a small effect on $D_1(T)$. A weak condition on $D_1(T)$ is thus sufficient, and it allows more subvectors to be removed. The condition used ensures that T continues to detect at least as many single-cycle gate-exhaustive faults as T_{sa}.

If these conditions are satisfied, v_{rem} is removed from V permanently. Otherwise, the procedure restores v_{rem} into V.

The order by which the subvectors from V are considered for removal is discussed next. The order attempts to ensure that subvectors, which are more likely to be removed, will be considered earlier. This reduces the computational effort of the software procedure.

The likelihood that a subvector will be removed is measured based on the test set T_{eff} obtained for V before any subvectors are removed from it. Using T_{eff}, the procedure associates with every subvector $v_i \in V$ a variable denoted by $used(v_i)$. The value of $used(v_i)$ is equal to the number of times v_i appears in a seed s_j whose test t_j is included in T_{eff}.

TABLE IV
EXAMPLE 1 OF REMOVAL OF SUBVECTORS

iter	$used(v_i)$ for $v_i \in V$	v_{rem}
1	18 20 10 23 25 16 17 17 17 17 25 13 22 17 18 19 27 14 13 15 16 15 13 21 19 15 11 22 16 17 19 17	$v_2(10)$
2	12 22 16 10 15 16 18 17 15 17 22 16 16 15 16 22 28 20 24 24 22 20 17 17 10 19 16 22 20 18 10	$v_3(10)$
3	25 27 17 15 20 18 11 13 17 20 17 18 16 10 13 15 19 26 23 13 21 12 16 15 21 16 20 15 29 22	$v_{13}(10)$, $v_6(11)$
4	16 17 18 11 32 19 21 17 25 20 20 17 16 17 18 15 18 26 17 27 22 21 21 22 18 27 16 22 20	$v_3(11)$, $v_{15}(15)$
5	14 23 18 16 26 25 19 22 21 19 18 17 11 20 22 27 16 15 16 25 17 25 15 18 18 32 21 20	$v_{12}(11)$
6	22 14 16 24 16 18 24 17 20 24 16 17 30 17 20 22 15 24 22 16 22 21 15 21 29 24 22	$v_1(14)$
7	23 20 12 23 21 13 32 28 14 19 23 19 24 13 17 26 19 23 23 21 24 26 29 19 12 25	$v_2(12)$, $v_{24}(12)$, $v_5(13)$
8	19 20 18 21 21 28 26 19 24 19 24 23 22 16 22 25 17 18 17 26 20 32 16 21 14	$v_{24}(14)$, $v_{13}(16)$
9	27 19 21 29 21 21 32 25 26 21 20 21 26 22 17 15 19 31 27 18 28 20 14 24	$v_{22}(14)$
10	25 23 29 24 32 21 29 21 30 26 14 23 26 24 29 14 22 23 29 18 26 22 22	$v_{10}(14)$
.	.	.
19	34 44 33 38 35 31 37 48 39 43 43 49 51 43	$v_5(31)$, $v_2(33)$
20	38 42 41 33 34 44 42 45 54 40 41 46 36	$v_3(33)$
21	50 42 42 40 39 49 44 59 36 47 56 48	

Since T_{eff} changes every time a subvector is removed from V, the procedure is iterative. At the beginning of an iteration, V is used for computing T_{eff}, and the variables $used(v_i)$ are updated. The procedure considers the subvectors in V from low to high value of $used(v_{rem})$. An iteration terminates if a subvector is removed from V permanently. In this case, a new iteration starts. The procedure terminates if an iteration ends without removing any subvectors from V.

In the example of $s1423$, iteration 1 starts with an initial set V consisting of 32 subvectors. After computing T_{eff}, the values of $used(v_i)$ are as shown in Table IV in the rows with a 1 under column $iter$. The values are shown in the order $used(v_0)$, $used(v_1)$, ... Based on the values of $used(v_i)$, the procedure considers $v_{rem} = v_2$ with $used(v_{rem}) = 10$ first. It finds that the subvector can be removed without reducing the fault coverage. This is shown under column v_{rem} of Table IV. The number in parentheses is the value of $used(v_{rem})$.

In iteration 2, V consists of 31 subvectors. Based on T_{eff}, the values of $used(v_i)$ are as shown in Table IV in the rows with a 2 under column $iter$. With these values of $used(v_i)$, the procedure considers $v_{rem} = v_3$ that has $used(v_{rem}) = 10$ first. It finds that the subvector can be removed without reducing the fault coverage.

In iteration 3, the procedure considers $v_{rem} = v_{13}$ with $used(v_{rem}) = 10$ first. It finds that the subvector cannot be removed. It considers $v_{rem} = v_6$ with $used(v_{rem}) = 11$ second. This subvector can be removed without reducing the fault coverage.

In iteration 4, the subvectors considered for removal are $v_{rem} = v_3$ with $used(v_{rem}) = 11$, followed by $v_{rem} = v_{15}$ with $used(v_{rem}) = 15$. The second subvector is removed.

TABLE V
EXAMPLE 2 OF REMOVAL OF SUBVECTORS

iter	subv	s.a.	g.exh
0	32	99.155	98.805
1	31	99.324	98.753
2	30	99.324	99.116
3	29	99.831	99.584
4	28	99.915	99.532
5	27	99.915	99.844
6	26	99.915	100.000
7	25	99.915	99.896
8	24	99.915	99.792
9	23	99.915	99.792
10	22	99.915	99.636

The procedure continues in the same manner. Even in later iterations, it finds a subvector it can remove after trying a small number of subvectors with the lowest values of $used(v_i)$. It terminates in iteration 21 with 12 subvectors in V and the values of $used(v_i)$ shown in the last row of Table IV.

Although the main purpose of the procedure is to reduce the number of subvectors in V, it also increases the single stuck-at fault coverage when the test set T based on the initial set V does not detect all the single stuck-at faults detected by T_{sa}. The single-cycle gate-exhaustive fault coverage is typically significantly higher than that of T_{sa}. To maximize the increase in the single stuck-at fault coverage, it is important to remove subvectors from V gradually. This is achieved by removing subvectors one by one.

An example of the increase in the single stuck-at fault coverage is given in Table V considering benchmark circuit $b07$. The circuit has a compressed deterministic test set S_{sa} for single stuck-at faults that consists of 41 seeds for an $LFSR$ of length $L = 36$. The test set T_{sa} achieves complete single stuck-at fault coverage equal to 99.915%. The single-cycle gate-exhaustive fault coverage of T_{sa} is 59.459%. A seed is partitioned into $p = 8$ subvectors of length $l = 5$. The initial set V contains 32 subvectors. The procedure removes subvectors from V one by one as shown in Table V. For every iteration, Table V shows the index of the iteration, the number of subvectors in V, the single stuck-at fault coverage, and the single-cycle gate-exhaustive fault coverage. Table V demonstrates the following points. (1) The single stuck-at fault coverage increases as the procedure removes subvectors from V, and reaches complete fault coverage in iteration 4. (2) The single-cycle gate-exhaustive fault coverage increases significantly when T is used instead of T_{sa}. As subvectors are removed from V, the single-cycle gate-exhaustive fault coverage changes such that it remains significantly higher than that of T_{sa}.

If the on-chip decompression logic requires $m > 1$ seeds to form a test, the software procedure needs to be extended as follows. (1) When constructing an initial set V of subvectors, the procedure needs to partition every seed of every compressed test into l-bit subvectors, obtaining $m \cdot p$ subvectors per test. (2) When constructing a set of applied tests T, the procedure needs to consider m consecutive seeds for every test.

IV. EXPERIMENTAL RESULTS

The results of the software procedure for single stuck-at faults and single-cycle gate-exhaustive faults in benchmark circuits are presented in this section.

A. setup

The software procedure was applied with the following parameter values. The set S_{sa} is a compact deterministic set of seeds for single stuck-at faults. The $LFSR$ used for S_{sa} is the smallest primitive $LFSR$ from [1] that allows complete or almost complete fault coverage to be achieved for single stuck-at faults. The set F_1 of single-cycle gate-exhaustive faults is the set of detectable faults from [22].

The number of tests in T is $N_T = 1,000,000$. With an $LFSR$ of length L, this implies that all the seeds can be considered when $2^L \leq 1,000,000$, or $L < 20$, and most of the seeds can be considered when $L = 20$. The results are interesting for circuits with $L > 20$, where $2^L \gg 1,000,000$. The number of tests applied until the fault coverages reach their final values is denoted by n_{eff}.

The software procedure is applied with $1 \leq l \leq L - 1$. For every value of l, $p = \lceil L/l \rceil$. With $l = 1$ and $p = L$, $V = \{0, 1\}$, and the on-chip test generation logic produces pseudo-random seeds by selecting bits pseudo-randomly from V. This case does not require any storage. In general, as l is increased and p is decreased, the number of subvectors in V increases, causing the storage requirements to increase. At the same time, the longer subvectors allow the single stuck-at fault coverage to increase, becoming closer to that of T_{sa}. The fault coverage of single-cycle gate-exhaustive faults is high for all the values of l and p. Based on this discussion, the results are reported for the lowest value of l for which the highest single stuck-at fault coverage is achieved. This value of l is denoted by l_{sel}. In addition, the case of $l = 1$ is reported for all the circuits to allow a comparison with the case where N_T pseudo-random seeds are used.

Typically with l_{sel}, the storage requirements for V are significantly lower than for S_{sa}. Results for values of l that are lower than l_{sel} are reported only when the storage requirements for V with $l = l_{sel}$ are close to the storage requirements for S_{sa}.

When the single stuck-at fault coverage for l_{sel} is lower than that of T_{sa}, several options exist for increasing the fault coverage: (1) increasing the number of tests, N_T, (2) adding test points, (3) complementing bits of seeds or applied tests as in earlier approaches, (4) storing combinations of subvectors that form compressed tests for the undetected faults, or (5) storing unpartitioned seeds for topping off the seeds produced by the on-chip test generation logic through pseudo-random combinations of subvectors. The use of unpartitioned seeds is demonstrated at the end of this section. As with other $LBIST$ approaches, it is assumed that a small loss of fault coverage is acceptable, and that it is acceptable to apply a large number of tests produced by the $LBIST$ logic through pseudo-random combinations to cover both single stuck-at and single-cycle gate-exhaustive faults.

TABLE VI

EXPERIMENTAL RESULTS (COMPLETE SINGLE STUCK-AT FAULT COVERAGE WITH $l_{sel} = 1$)

circuit	inp	L	l	p	iter	subv	bits	frac	tests	eff	s.a.	diff	g.exh	diff	ntime
s35932	1763	13	13	1	0	57	741	1.000	57	57	89.809	0.000	98.509	0.000	1.00
s35932	1763	13	1	13	0	2	2	0.003	232	81	89.809	0.000	99.996	1.488	3.71
sasc	132	13	13	1	0	41	533	1.000	41	41	100.000	0.000	67.374	0.000	1.00
sasc	132	13	1	13	0	2	2	0.004	416	167	100.000	0.000	93.923	26.550	4.83
des_area	367	14	14	1	0	158	2212	1.000	158	158	100.000	0.000	72.424	0.000	1.00
des_area	367	14	1	14	0	2	2	0.001	739	538	100.000	0.000	93.432	21.008	4.06
systemcdes	320	14	14	1	0	102	1428	1.000	102	102	100.000	0.000	93.391	0.000	1.00
systemcdes	320	14	1	14	0	2	2	0.001	1297	184	100.000	0.000	99.646	6.255	4.48
s1423	91	18	18	1	0	55	990	1.000	55	55	99.076	0.000	90.381	0.000	1.00
s1423	91	18	1	18	0	2	2	0.002	25579	137	99.076	0.000	99.758	9.377	54.11
usb_phy	112	18	18	1	0	36	648	1.000	36	36	100.000	0.000	82.536	0.000	1.00
usb_phy	112	18	1	18	0	2	2	0.003	2913	114	100.000	0.000	99.677	17.141	18.00
b04	78	28	28	1	0	34	952	1.000	34	34	99.851	0.000	79.263	0.000	1.00
b04	78	28	1	28	0	2	2	0.002	46318	239	99.851	0.000	98.560	19.297	136.00
aes_core	788	28	28	1	0	380	10640	1.000	380	380	100.000	0.000	98.259	0.000	1.00
aes_core	788	28	1	28	0	2	2	0.000	3548	695	100.000	0.000	99.997	1.738	3.76
systemcaes	928	29	29	1	0	109	3161	1.000	109	109	99.995	0.000	79.613	0.000	1.00
systemcaes	928	29	1	29	0	2	2	0.001	39418	1047	99.995	0.000	99.989	20.376	14.78
s5378	214	36	36	1	0	133	4788	1.000	133	133	99.131	0.000	67.605	0.000	1.00
s5378	214	36	1	36	0	2	2	0.000	53432	898	99.131	0.000	99.533	31.928	43.62
s13207	700	47	47	1	0	251	11797	1.000	251	251	98.462	0.000	62.259	0.000	1.00
s13207	700	47	1	47	0	2	2	0.000	190271	1905	98.462	0.000	97.623	35.364	181.57

TABLE VII

EXPERIMENTAL RESULTS (COMPLETE SINGLE STUCK-AT FAULT COVERAGE WITH $l_{sel} > 1$)

circuit	inp	L	l	p	iter	subv	bits	frac	tests	eff	s.a.	diff	g.exh	diff	ntime
b07	53	36	36	1	0	41	1476	1.000	41	41	99.915	0.000	59.459	0.000	1.00
b07	53	36	1	36	0	2	2	0.001	37994	388	97.041	-2.874	96.674	37.214	3168.50
b07	53	36	5	8	10	22	110	0.075	963258	404	99.915	0.000	99.636	40.177	81965.01
simple_spi	146	38	38	1	0	46	1748	1.000	46	46	100.000	0.000	55.066	0.000	1.00
simple_spi	146	38	1	38	0	2	2	0.001	129112	453	98.620	-1.380	97.568	42.503	2078.71
simple_spi	146	38	2	19	1	3	6	0.003	620445	511	100.000	0.000	99.873	44.807	2712.29
i2c	145	43	43	1	0	58	2494	1.000	58	58	100.000	0.000	73.974	0.000	1.00
i2c	145	43	1	43	0	2	2	0.001	261195	278	96.577	-3.423	93.748	19.774	2086.92
i2c	145	43	15	3	138	35	525	0.211	158072	332	100.000	0.000	97.797	23.823	142832.39
spi	274	44	44	1	0	360	15840	1.000	360	360	99.985	0.000	67.073	0.000	1.00
spi	274	44	1	44	0	2	2	0.000	65510	2432	99.354	-0.631	97.956	30.883	162.45
spi	274	44	6	8	13	51	306	0.019	683691	2536	99.985	0.000	99.931	32.858	1213.15

B. Results

The results of the software procedure are presented in Tables VI, VII and VIII. For the circuits in Table VI, $l_{sel} = 1$. For the circuits in Tables VII and VIII, $l_{sel} > 1$. For the circuits in Tables VI and VII, complete single stuck-at fault coverage is achieved with l_{sel}. For the circuits in Table VIII, the single stuck-at fault coverage is incomplete.

The circuits in each table are arranged from low to high value of L. The first row for every circuit provides a baseline where the compressed compact deterministic test set S_{sa} is stored on-chip, and the test set T_{sa} is applied. This is represented by $l = L$, $p = 1$ and $V = S_{sa}$. Additional rows show the final results of the software procedure with $1 \leq l \leq l_{sel}$.

In each row, after the circuit name, column inp shows the number of inputs, and column L shows the length of the $LFSR$. Columns l and p show the values of the corresponding parameters. Column $iter$ shows the iteration of the software procedure, where iteration 0 corresponds to the initial set V, and every additional iteration reduces the number of subvectors in V by one. Column $subv$ shows the number of subvectors in V. Column $bits$ shows the number of bits required for V, equal to $l \cdot |V|$. Column $frac$ shows the number of bits required for V divided by the number of bits required for S_{sa}, $(l \cdot |V|)/(L \cdot |S_{sa}|)$. Column $tests$ shows the number of tests from T that need to be applied until the last test that is effective in detecting target faults. This number of tests can be used instead of N_T, and was denoted earlier by n_{eff}. Column eff shows the number of tests in T_{eff}. These are the tests that increase the fault coverages of target faults. Column $s.a.$ shows the single stuck-at fault coverage. Column $diff$ that follows is the increase or reduction in the single stuck-at fault coverage relative to S_{sa}. Column $g.exh$ shows the single-cycle gate-exhaustive fault coverage. Column $diff$ that follows is the increase or reduction in the single-cycle gate-exhaustive fault coverage relative to S_{sa}. Column $ntime$ shows the runtime of the software procedure divided by the runtime required for fault simulation of T_{sa}. This is referred to as the normalized runtime.

C. Discussion

The following points can be seen from Tables VI-VIII. The set S_{sa} represents the conventional approach to test data compression where deterministic tests are compressed into seeds for an $LFSR$. Considering columns $bits$ and $frac$, the storage requirements of V produced by the software procedure

TABLE VIII
EXPERIMENTAL RESULTS (INCOMPLETE SINGLE STUCK-AT FAULT COVERAGE)

circuit	inp	L	l	p	iter	subv	bits	frac	tests	eff	s.a.	diff	g.exh	diff	ntime
wb_dma	738	47	47	1	0	84	3948	1.000	84	84	100.000	0.000	73.840	0.000	1.00
wb_dma	738	47	1	47	0	2	2	0.001	252167	771	99.615	-0.385	99.077	25.237	577.39
wb_dma	738	47	5	10	4	28	140	0.035	838936	767	99.978	-0.022	99.532	25.692	6023.19
wb_dma	738	47	10	5	0	277	2770	0.702	986814	792	99.989	-0.011	99.577	25.737	472.61
s15850	611	57	57	1	0	197	11229	1.000	197	197	96.682	0.000	73.285	0.000	1.00
s15850	611	57	1	57	0	2	2	0.000	261955	1788	94.559	-2.124	87.761	14.475	2007.22
s15850	611	57	4	15	1	15	60	0.005	999797	2267	96.119	-0.563	92.853	19.567	3443.86
s9234	247	75	75	1	0	134	10050	1.000	134	134	93.475	0.000	70.825	0.000	1.00
s9234	247	75	1	75	0	2	2	0.000	261492	921	90.761	-2.714	91.300	20.475	2836.36
s9234	247	75	3	25	0	8	24	0.002	996368	1093	92.796	-0.679	96.131	25.306	2512.70
s38584	1464	98	98	1	0	218	21364	1.000	218	218	95.852	0.000	88.313	0.000	1.00
s38584	1464	98	1	98	0	2	2	0.000	130393	1687	95.648	-0.204	99.513	11.201	1061.35
s38584	1464	98	5	20	4	28	140	0.007	995504	1724	95.796	-0.055	99.766	11.453	62243.02
tv80	372	109	109	1	0	370	40330	1.000	370	370	99.527	0.000	77.264	0.000	1.00
tv80	372	109	1	109	0	2	2	0.000	261397	2106	97.947	-1.579	94.625	17.361	421.39
tv80	372	109	7	16	4	124	868	0.022	998072	2320	98.919	-0.607	97.424	20.159	33913.94
s38417	1664	111	111	1	0	283	31413	1.000	283	283	99.471	0.000	70.723	0.000	1.00
s38417	1664	111	1	111	0	2	2	0.000	261979	4395	98.406	-1.065	89.715	18.993	760.24
s38417	1664	111	4	28	2	14	56	0.002	998535	5410	99.343	-0.128	95.892	25.169	1973.63
b15	483	113	113	1	0	266	30058	1.000	266	266	98.580	0.000	59.444	0.000	1.00
b15	483	113	1	113	0	2	2	0.000	261591	1387	95.186	-3.393	71.993	12.549	1297.76
b15	483	113	5	23	0	32	160	0.005	999779	1635	97.708	-0.872	83.513	24.069	1104.02
b20	527	119	119	1	0	238	28322	1.000	238	238	93.304	0.000	64.250	0.000	1.00
b20	527	119	1	119	0	2	2	0.000	260752	1355	87.714	-5.589	84.857	20.606	2647.18
b20	527	119	5	24	3	29	145	0.005	997430	1525	88.861	-4.442	86.057	21.807	13758.22
b14	280	128	128	1	0	290	37120	1.000	290	290	94.960	0.000	72.068	0.000	1.00
b14	280	128	1	128	0	2	2	0.000	2040	278	79.020	-15.940	67.510	-4.558	4204.46
b14	280	128	3	43	3	5	15	0.000	973582	790	85.883	-9.077	78.680	6.612	36678.39

are significantly smaller than the storage requirements of S_{sa} for most of the circuits considered. Thus, the use of the on-chip test generation logic reduces the storage requirements compared with the conventional approach to test data compression represented by S_{sa}.

For the circuits with the lower values of L, $l_{sel} = 1$ is typically sufficient for achieving complete fault coverage of single stuck-at faults. The circuit with the largest value of L for which $l_{sel} = 1$ is $s13207$, with $L = 47$. In this case, $2^L \gg N_T$.

For the circuits with the higher values of L, $l_{sel} > 1$ is typically needed, and using $l = 1$ that produces pseudo-random seeds results in a lower single stuck-at fault coverage than l_{sel}. Storage of V is important in this case for increasing the single stuck-at fault coverage. Moreover, the software procedure reduces V to reduce the storage requirements and increase the single stuck-at fault coverage. The software procedure does not allow the single stuck-at fault coverage to decrease as it removes subvectors from V, and it keeps track of cases where the fault coverage is increased. This allows it to identify cases where removing subvectors increases the single stuck-at fault coverage.

The improvement in the fault coverage of single-cycle gate-exhaustive faults is significant in all the cases when V is used for on-chip test generation instead of applying T_{sa}. The single-cycle gate-exhaustive fault coverage does not increase monotonically with the iterations of the software procedure since the procedure only requires that the fault coverage would be at least as high as that of T_{sa}. For the circuits with $l_{sel} > 1$, if $l = 1$ is used for producing pseudo-random seeds, the fault coverage for single-cycle gate-exhaustive faults is significantly lower than that achieved with l_{sel}.

The increase in the fault coverage of single-cycle gate-exhaustive faults requires an increase in the number of applied tests. Columns $tests$ and eff describe this increase. Even if only tests that increase the fault coverage are considered (column eff), the number of tests required for detecting single-cycle gate-exhaustive faults is significantly larger than the number of tests in T_{sa} that are required for detecting single stuck-at faults. With on-chip test generation that combines subvectors pseudo-randomly, not every applied test is effective in increasing the fault coverage, and the number of applied tests (shown under column $tests$) must be allowed to be significantly higher than the number of tests in T_{sa}.

The normalized runtime has a strong dependence on the number of subvectors in V, and the number of subvectors increases with l_{sel}. The normalized runtime does not increase with the size of the circuit, implying that the software procedure scales similar to a fault simulation procedure. The part of the procedure that reduces the number of subvectors in V can be terminated earlier to reduce the normalized runtime for the larger values of l_{sel}.

Other parameters also do not deteriorate for larger circuits. Considering the loss in single stuck-at fault coverage, a larger circuit may have a lower fault coverage loss than a smaller circuit. For example, the loss in single stuck-at fault coverage is 0.872 for $b15$, and 0.128 for $s38417$ that is a larger circuit. Comparing the number of applied tests to the number of tests in T_{sa}, the increase for a larger circuit may be smaller than the increase for a larger circuit. For example, for $b07$ the increase is 963258/41=23494.1, and for $b15$ that is larger the increase is 999779/266=3758.6. Overall, the results do not deteriorate with the size of the circuit.

TABLE IX
EXPERIMENTAL RESULTS (WITH UNPARTITIONED SEEDS)

circuit	inp	L	l	p	iter	subv	unpart	bits	frac	tests	eff	s.a.	diff	g.exh	diff	ntime
b07	53	36	36	1	0	41	0	1476	1.000	41	41	99.915	0.000	59.459	0.000	1.00
b07	53	36	3	12	2	6	0	18	0.012	9884	280	97.295	-2.620	85.603	26.143	277.00
b07	53	36	3	12	4	6	0	18	0.012	97081	303	98.732	-1.183	88.202	28.742	1040.50
b07	53	36	3	12	4	6	0	18	0.012	921022	304	99.408	-0.507	89.449	29.990	3482.50
b07	53	36	3	12	4	6	3	126	0.085	921025	307	99.915	0.000	89.813	30.353	3483.00
i2c	145	43	43	1	0	58	0	2494	1.000	58	58	100.000	0.000	73.974	0.000	1.00
i2c	145	43	4	11	8	8	0	32	0.013	9659	238	94.737	-5.263	89.975	16.001	1440.78
i2c	145	43	4	11	10	8	0	32	0.013	99783	295	97.261	-2.739	94.492	20.518	2522.89
i2c	145	43	4	11	10	8	0	32	0.013	941774	347	99.529	-0.471	98.568	24.594	4495.89
i2c	145	43	4	11	10	8	6	290	0.116	941780	353	99.914	-0.086	98.981	25.007	4496.44
wb_dma	738	47	47	1	0	84	0	3948	1.000	84	84	100.000	0.000	73.840	0.000	1.00
wb_dma	738	47	4	12	5	11	0	44	0.011	9941	643	98.769	-1.231	92.974	19.135	267.00
wb_dma	738	47	4	12	7	11	0	44	0.011	99504	763	99.626	-0.374	98.962	25.122	623.56
wb_dma	738	47	4	12	7	11	0	44	0.011	911910	789	99.967	-0.033	99.526	25.686	1058.65
wb_dma	738	47	4	12	7	11	3	185	0.047	911913	792	100.000	0.000	99.545	25.705	1058.79
s15850	611	57	57	1	0	197	0	11229	1.000	197	197	96.682	0.000	73.285	0.000	1.00
s15850	611	57	4	15	4	12	0	48	0.004	9992	802	91.761	-4.921	76.307	3.022	301.97
s15850	611	57	4	15	6	12	0	48	0.004	99928	1496	94.064	-2.618	84.803	11.518	1006.07
s15850	611	57	4	15	6	12	0	48	0.004	999247	2244	95.787	-0.896	92.434	19.149	2745.35
s15850	611	57	4	15	6	12	19	1131	0.101	999266	2263	96.324	-0.358	92.816	19.531	2749.48
s9234	247	75	75	1	0	134	0	10050	1.000	134	134	93.475	0.000	70.825	0.000	1.00
s9234	247	75	4	19	0	16	0	64	0.006	9967	640	84.900	-8.575	74.861	4.036	72.44
s9234	247	75	4	19	2	16	0	64	0.006	99554	826	89.476	-3.999	87.228	16.404	1406.86
s9234	247	75	4	19	2	16	0	64	0.006	997175	1109	92.031	-1.444	95.869	25.045	3959.54
s9234	247	75	4	19	2	16	13	1039	0.103	997188	1122	93.316	-0.159	96.499	25.674	3962.95
s38584	1464	98	98	1	0	218	0	21364	1.000	218	218	95.852	0.000	88.313	0.000	1.00
s38584	1464	98	3	33	0	8	0	24	0.001	9993	1388	94.485	-1.366	96.348	8.036	29.34
s38584	1464	98	3	33	2	8	0	24	0.001	99110	1655	95.681	-0.171	99.456	11.143	346.18
s38584	1464	98	3	33	2	8	0	24	0.001	991660	1710	95.733	-0.118	99.770	11.457	1403.72
s38584	1464	98	3	33	2	8	14	1396	0.065	991674	1724	95.852	0.000	99.852	11.539	1406.58
tv80	372	109	109	1	0	370	0	40330	1.000	370	370	99.527	0.000	77.264	0.000	1.00
tv80	372	109	6	19	1	63	0	378	0.009	9990	1241	91.041	-8.486	78.596	1.332	108.97
tv80	372	109	6	19	3	63	0	378	0.009	99986	1882	96.911	-2.615	90.892	13.628	2163.75
tv80	372	109	6	19	3	63	0	378	0.009	999575	2299	98.747	-0.780	97.158	19.894	2554.36
tv80	372	109	6	19	3	63	34	4084	0.101	999609	2333	99.437	-0.090	97.606	20.341	2556.29
s38417	1664	111	111	1	0	283	0	31413	1.000	283	283	99.471	0.000	70.723	0.000	1.00
s38417	1664	111	3	37	3	5	0	15	0.000	9996	2027	93.679	-5.792	75.511	4.788	367.47
s38417	1664	111	3	37	5	5	0	15	0.000	99955	3768	97.742	-1.729	87.013	16.290	692.07
s38417	1664	111	3	37	5	5	0	15	0.000	994091	5552	99.304	-0.167	96.303	25.580	1155.05
s38417	1664	111	3	37	5	5	29	3234	0.103	994120	5581	99.452	-0.019	96.467	25.745	1156.87
b15	483	113	113	1	0	266	0	30058	1.000	266	266	98.580	0.000	59.444	0.000	1.00
b15	483	113	5	23	0	32	0	160	0.005	9975	1179	77.546	-21.033	51.187	-8.257	93.75
b15	483	113	5	23	2	32	0	160	0.005	99982	1367	91.300	-7.280	64.795	5.351	2673.95
b15	483	113	5	23	2	32	0	160	0.005	999779	1635	97.708	-0.872	83.513	24.069	3760.90
b15	483	113	5	23	2	32	26	3098	0.103	999805	1661	98.430	-0.149	84.341	24.897	3763.23
b20	527	119	119	1	0	238	0	28322	1.000	238	238	93.304	0.000	64.250	0.000	1.00
b20	527	119	3	40	3	5	0	15	0.001	9991	877	86.372	-6.931	77.874	13.624	384.14
b20	527	119	3	40	5	5	0	15	0.001	99695	1275	87.369	-5.935	83.874	19.624	864.00
b20	527	119	3	40	5	5	0	15	0.001	993897	1506	88.618	-4.686	85.832	21.582	3490.66
b20	527	119	3	40	5	5	24	2871	0.101	993921	1530	91.944	-1.360	87.038	22.788	3500.70
b14	280	128	128	1	0	290	0	37120	1.000	290	290	94.960	0.000	72.068	0.000	1.00
b14	280	128	4	32	4	12	0	48	0.001	9811	455	82.647	-12.313	74.430	2.362	489.94
b14	280	128	4	32	6	12	0	48	0.001	98718	561	83.829	-11.131	76.485	4.417	1288.84
b14	280	128	4	32	6	12	0	48	0.001	944346	708	85.462	-9.498	78.866	6.798	4241.08
b14	280	128	4	32	6	12	29	3760	0.101	944375	737	92.696	-2.264	82.404	10.336	4253.62

D. Using Unpartitioned Seeds and Lower Values of N_T

This section explores two options. The first option is to increase the fault coverage by storing a small number of unpartitioned seeds from S_{sa}. Unpartitioned seeds are used for topping off the tests produced by the on-chip test generation logic through pseudo-random combinations of subvectors from V. By storing a sufficient number of unpartitioned seeds from S_{sa} it is possible to achieve complete single stuck-at fault coverage for every value of l. However, the storage requirements may be high. For the experiment in this section, the storage requirements for V and the unpartitioned seeds is limited to approximately a tenth of the storage requirements of S_{sa}.

As l is increased, the single stuck-at fault coverage achieved based on the subvectors in V increases, and fewer unpartitioned seeds need to be stored. As before, the results are reported for the lowest value of l for which the highest single stuck-at fault coverage is achieved. This value of l is denoted by l_{sel}.

The second option explored in this section is to reduce the number of tests N_T produced from pseudo-random combinations of stored subvectors. Earlier, $N_T = 1,000,000$ was used. Experimental results with $N_T = 10,000$ show that reducing N_T reduces the runtime of the software procedure.

However, it also reduces the single stuck-at and single-cycle gate-exhaustive fault coverages. Whereas the single stuck-at fault coverage can be recovered by adding unpartitioned seeds, the single-cycle gate-exhaustive fault coverage remains significantly lower even after adding unpartitioned seeds. Therefore, after applying the software procedure using $N_T = 10,000$, tests are computed using $N_T = 100,000$, and then using $N_T = 1,000,000$ as before, to demonstrate the effects on the fault coverages. Unpartitioned seeds are added to the tests produced with $N_T = 1,000,000$.

The results in Tables VI, VII and VIII demonstrate that $l_{sel} \leq 5$ is typically obtained. The value of l is limited to eight in this section.

The results for several circuits from Tables VII and VIII are presented in Table IX. The format is similar to Tables VII and VIII, with the addition of the number of unpartitioned seeds under column $unpart$. The storage requirements and number of applied tests take into consideration the unpartitioned seeds.

The first row for every circuit in Table IX describes S_{sa}. The second row shows the results of the software procedure using $N_T = 10,000$. The third and fourth rows show the results obtained when the set of subvectors produced by the software procedure is used for producing $N_T = 100,000$ and $N_T = 1,000,000$ tests, respectively. The fifth row shows the results obtained by adding unpartitioned seeds with a limit on the storage requirements as discussed above.

The following points can be seen from Table IX. The normalized runtime of the software procedure is reduced significantly when N_T is reduced from $N_T = 1,000,000$ to $N_T = 10,000$.

Without the topoff seeds the storage requirements are similar for different values of N_T, and they are significantly lower than the storage requirements for S_{sa}, which represents the conventional approach to test data compression. With the topoff seeds the limit on the storage requirements ensures that they are still significantly lower than those of S_{sa}.

The single stuck-at fault coverage as well as the single-cycle gate-exhaustive fault coverage are significantly lower when N_T is lower. This makes it important to use a higher value of N_T for test application. In particular, the higher value of N_T ensures that the single stuck-at fault coverage is higher, requiring fewer topoff seeds. It is also important for the single-cycle gate-exhaustive fault coverage, which cannot be recovered by topoff seeds targeting single stuck-at faults.

V. Concluding Remarks

A class of storage-based logic built-in self-test ($LBIST$) approaches is based on storage of test data entries obtained by partitioning deterministic tests, and forming tests from different combinations of stored test data entries. In this article, partitioning was applied for the first time to compressed deterministic tests. Tests were formed on-chip by using pseudo-random combinations of subvectors of compressed deterministic tests to form seeds for the on-chip decompression logic, which produces tests for the circuit. A software procedure was described for deriving and reducing the stored test data. Experimental results demonstrated the effectiveness of this approach in reducing the volume of stored test data relative to compressed deterministic test sets for benchmark circuits. Both single stuck-at and single-cycle gate-exhaustive faults were considered to demonstrate the ability to achieve comprehensive fault coverage.

References

[1] P. H. Bardell, W. H. McAnney and J. Savir, $Built-In\ Test\ for\ VLSI\ Pseudorandom\ Techniques$, Wiley Interscience, 1987.

[2] N. A. Touba and E. J. McCluskey, "Bit-fixing in Pseudorandom Sequences for Scan BIST", in IEEE Trans. on Computer-Aided Design, April 2001, Vol. 20, No. 4, pp. 545-555.

[3] S. Hellebrand, H.-G. Liang and H.-J. Wunderlich, "A Mixed Mode BIST Scheme Based on Reseeding of Folding Counters", Journal of Electronic Testing, 2001, Vol. 17, pp. 341–349.

[4] I. Pomeranz and S. M. Reddy, "A Storage Based Built-In Test Pattern Generation Method for Scan Circuits Based on Partitioning and Reduction of a Precomputed Test Set", in IEEE Trans. on Computers, Nov. 2002, pp. 1282-1993.

[5] S. Pateras, "Security vs. Test Quality: Fully Embedded Test Approaches are the Key to Having Both", in Proc. Intl. Test Conf., 2004, Panel P2.2, p. 1413.

[6] D. Xiang, M. Chen and H. Fujiwara, "Using Weighted Scan Enable Signals to Improve Test Effectiveness of Scan-Based BIST", in IEEE Trans. on Computers, Dec. 2007, vol. 56, no. 12, pp. 1619-1628.

[7] L.-T. Wang, X. Wen, S. Wu, H. Furukawa, H.-J. Chao, B. Sheu, J. Guo and W.-B. Jone, "Using Launch-on-Capture for Testing BIST Designs Containing Synchronous and Asynchronous Clock Domains", in IEEE Trans. on Computer-Aided Design, Feb. 2010, Vol. 29, No. 2, pp. 299-312.

[8] R. S. Oliveira, J. Semiao, I. C. Teixeira, M. B. Santos and J. P. Teixeira, "On-line BIST for Performance Failure Prediction under Aging Effects in Automotive Safety-critical Applications", in Proc. Latin American Test Workshop, 2011, pp. 1-6.

[9] Y. Sato, H. Yamaguchi, M. Matsuzono and S. Kajihara, "Multi-Cycle Test with Partial Observation on Scan-Based BIST Structure", in Proc. Asian Test Symp., 2011, pp. 54-59.

[10] M. E. Imhof and H. Wunderlich, "Bit-Flipping Scan - A Unified Architecture for Fault Tolerance and Offline Test", in Proc. Design, Automation & Test in Europe Conf., 2014, pp. 1-6.

[11] F. Reimann, M. Glas, J. Teich, A. Cook, L. Rodríguez Gomez, D. Ull, H.-J. Wunderlich, P. Engelke and U. Abelein, "Advanced Diagnosis: SBST and BIST Integration in Automotive E/E Architectures", in Proc. Design Autom. Conf., 2014, pp. 1-9.

[12] G. Contreras, N. Ahmed, L. Winemberg and M. Tehranipoor, "Predictive LBIST Model and Partial ATPG for Seed Extraction", in Proc. Intl. Symp. on Defect and Fault Tolerance, 2015, pp. 139-146.

[13] C. Shiao, W. Lien and K. Lee, "A Test-per-cycle BIST Architecture with Low Area Overhead and no Storage Requirement", in Proc. Intl. Symp. on VLSI Design, Automation and Test, 2016, pp. 1-4.

[14] Y. Liu, N. Mukherjee, J. Rajski, S. M. Reddy and J. Tyszer, "Deterministic Stellar BIST for In-System Automotive Test", in Proc. Intl. Test Conf., 2018, pp. 1-9.

[15] B. Kaczmarek, G. Mrugalski, N. Mukherjee, J. Rajski, Ł. Rybak and J. Tyszer, "Test Sequence-Optimized BIST for Automotive Applications", in Proc. European Test Symp., 2020, pp. 1-6.

[16] A. Koneru and K. Chakrabarty, "An Interlayer Interconnect BIST and Diagnosis Solution for Monolithic 3-D ICs", in IEEE Trans. on Computer-Aided Design, Oct. 2020, Vol. 39, No. 10, pp. 3056-3066.

[17] I. Pomeranz, "Storage-Based Built-In Self-Test for Gate-Exhaustive Faults", in IEEE Trans. on Computer-Aided Design, Oct. 2021, Vol. 40, No. 10, pp. 2189-2193.

[18] I. Pomeranz, "Zoom-In Feature for Storage-Based Logic Built-In Self-Test", in Proc. Intl. Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2021.

[19] M. B. G. Remadevi and R. Bakthavatchalu, "Design of a Programmable Low Power Linear Feedback Shift Register for BIST Applications", in Proc. Intl. Test Conf. India, 2022, pp. 1-4

[20] I. Pomeranz and S. M. Reddy, "Static Test Data Volume Reduction Using Complementation or Modulo-M Addition", in IEEE Trans. on VLSI Systems, June 2011, pp. 1108-1112.

[21] I. Pomeranz, "Input Test Data Volume Reduction Using Seed Complementation and Multiple $LFSR$s", in Proc. VLSI Test Symp., 2020, pp. 1-6.

[22] I. Pomeranz, "Maximal Independent Fault Set for Gate-Exhaustive Faults", in IEEE Trans. on Computer-Aided Design, March 2021, Vol. 40, No. 3, pp. 598-602.

PLACE PHOTO HERE

Irith Pomeranz (M'89-SM'96-F'99) received the B.Sc degree (Summa cum Laude) in Computer Engineering and the D.Sc degree from the Department of Electrical Engineering at the Technion - Israel Institute of Technology in 1985 and 1989, respectively. From 1989 to 1990 she was a Lecturer in the Department of Computer Science at the Technion. From 1990 to 2000 she was a faculty member in the Department of Electrical and Computer Engineering at the University of Iowa. In 2000 she joined Purdue University in West Lafayette IN where she is the Cadence Professor in the Elmore Family School of Electrical and Computer Engineering. Her research interests include testing of VLSI circuits, design for testability, and defect diagnosis. Dr. Pomeranz is a recipient of the NSF Young Investigator Award in 1993, and of the University of Iowa Faculty Scholar Award in 1997. Three of her conference papers won best paper awards, and four other papers were nominated for best paper awards. One of the papers she co-authored was selected by the 2016 International Test Conference as the most significant paper published ten years before. She delivered a keynote speech at the 2006 Asian Test Symposium. She was one of the very first three featured authors on IEEE Xplore, posted February 11-25, 2020. She served as associate editor of the ACM Transactions on Design Automation, the IEEE Transactions on Computers, and the IEEE Transactions on VLSI Systems. She served as guest editor of the IEEE Transactions on Computers January 1998 special issue on Dependability of Computing Systems, and as program co-chair of the 1999 Fault-Tolerant Computing Symposium. She served as program chair of the 2004 and 2005 VLSI Test Symposium, and as general chair of the 2006 VLSI Test Symposium. She is a Golden Core Member of the IEEE Computer Society.