IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 November 2023, accepted 5 December 2023, date of publication 8 December 2023, date of current version 14 December 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3341360

RESEARCH ARTICLE

Storage and Counter Based Logic Built-In Self-Test

IRITH POMERANZ, (Fellow, IEEE)
School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, USA
e-mail: pomeranz@purdue.edu

ABSTRACT Recent reports of silent data corruption because of hardware faults in large data centers bring to the forefront the importance of in-field testing. In-field testing, enabled by logic built-in self-test (*LBIST*), addresses defects that occur during the lifetime of a chip and ones that escaped manufacturing tests. A class of *LBIST* approaches for scan circuits store partitioned deterministic test data on-chip and produce tests by combining stored test data entries in one of two ways: 1) pseudo-random combinations are selected by linear-feedback shift-registers (*LFSR*s); or 2) deterministic combinations are stored on-chip as sets of indices of stored test data entries. This article introduces a third option where counters are used for forming combinations of stored test data entries. Counters do not require additional storage, and ensure complete fault coverage with a limited number of tests. Experimental results for benchmark circuits demonstrate the advantages of counters in the context where test data entries for on-chip storage are obtained by partitioning compressed deterministic tests, and the universally available on-chip decompression logic is used as part of the test application process.

INDEX TERMS Full scan design, 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 used for in-field testing during system startup or idle intervals [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]. *LBIST* is important for detecting defects that occur as chips deteriorate because of aging as well as defects that escaped manufacturing tests. Recent reports of silent data corruption because of hardware faults in large data centers [25], [26] stress the need for in-field testing enabled by *LBIST*. Silent data corruption implies that programs produce incorrect results without the errors being detected. This is possible when the errors affect the data in ways that do not cause exceptions to occur. For example, the incorrect data do not result in division by zero or access to restricted memory addresses that would have caused exceptions to occur. As a result, programs can continue to execute incorrectly for long periods of time.

The associate editor coordinating the review of this manuscript and approving it for publication was Xiao-Sheng Si.

The basis for most *LBIST* approaches is pseudo-random tests that can easily be produced on-chip using linear-feedback shift-registers (*LFSR*s) [1]. For increased fault coverage, pseudo-random tests are enhanced by one of several approaches: controlling the probabilities of 0s and 1s in the applied tests to obtain weighted pseudo-random tests [1], using bit-flipping or bit-fixing logic to assign specific values to certain inputs as in [2] or [11], or using multiple seeds for the *LFSR* [3]. Test points are also useful in increasing the fault coverage [1].

LBIST approaches that are based on storage of deterministic test data also exist. The stored test data in [7], [15], and [16] consist of compressed deterministic tests that are decompressed on-chip using the universally available on-chip decompression logic. To reduce the number of stored tests, each stored test is used in [15] and [16] for applying several different tests to the circuit. Different tests are obtained by complementing bits of stored or applied tests. The application of several tests from the same test data is effective not only for reducing the storage requirements but also for increasing the fault coverage [27], [28], [29], and it is used in this article

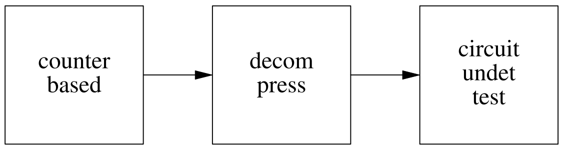

FIGURE 1. On-chip test generation logic.

as well. In addition to complementing bits of stored test data or applied tests, other operators are effective in increasing the fault coverage as discussed later.

The *LBIST* approaches described in [4], [18], [19], [23], and [24] store partitioned deterministic test data on-chip. Tests are formed by combining stored test data entries on-chip. The large number of options available for combining test data entries, and the fact that test data entries are derived from deterministic tests, ensure that high fault coverage can be achieved.

Partitioning of deterministic tests into scan vectors is used in [4], [18], [19], and [23]. In [4], all the possible combinations of the stored test data entries are used for forming tests. The more recent approaches control the number of applied tests by combining stored test data entries in one of two ways. In [18] and [19], pseudo-random combinations of stored test data entries are selected by *LFSR*s. In [18] and [23], deterministic combinations of stored test data entries are stored on-chip. Each combination is a set of indices of stored test data entries, indicating which test data entries should be combined to form a test. The two approaches for combining test data entries, pseudo-random and deterministic, may be used exclusively or together. At the cost of additional storage, deterministic combinations allow complete fault coverage to be achieved with significantly reduced numbers of tests compared with pseudo-random combinations.

This article introduces a third option where counters are used for forming combinations of stored test data entries. Counters do not require additional storage. However, they can ensure that deterministic tests are reproduced from the stored test data. Thus, complete fault coverage can be achieved with a limited number of tests.

Experimental results for benchmark circuits demonstrate the advantages of counters in the context where test data entries for on-chip storage are obtained by partitioning compressed deterministic tests. On-chip decompression logic is used as part of the test application process in this case as illustrated by Figure 1. The counter-based logic shown on the left in Figure 1 is the subject of this article. It produces compressed deterministic tests for the on-chip decompression logic that produces uncompressed tests for the circuit-under-test.

Partitioning of compressed deterministic tests was also used in [24]. However, only *LFSR*s are used in [24] for forming compressed tests using pseudo-random combinations of stored test data entries. As a result, a large number of tests needs to be applied, and the fault coverage is not always complete. By using counters as suggested in this article, complete fault coverage is guaranteed, and the number of tests is reduced. This is achieved in this article without the storage overhead of deterministic combinations. Both the fault coverage and the number of applied tests are important parameters of an *LBIST* approach. Achieving complete fault coverage is challenging with on-chip hardware, and it is common for *LBIST* approaches to tolerate a fault coverage loss. Large numbers of applied tests are tolerated to reduce the fault coverage loss. The counter-based approach does not require a loss of fault coverage and limits the number of applied tests. The reports of silent data corruption stress the importance of these characteristics.

The use of counters requires a new software procedure for optimizing the stored test data. Such a procedure is described in this article. The main differences from the software procedure described in [24] are related to the ability to achieve complete fault coverage. In [24], a loss of fault coverage is tolerated when optimizing the stored test data for pseudo-random combinations.

In summary, the key contribution of this article is to provide a fundamental improvement to a class of *LBIST* approaches where partitioned deterministic test data are stored on-chip, and tests are produced by combining stored test data entries on-chip. The number of options available for combining test data entries, and the fact that they are derived from deterministic tests, ensure that high fault coverage can be achieved not only for stuck-at faults, but also for single-cycle gate-exhaustive faults that are more difficult to detect. Earlier approaches combined test data entries in one of two ways: pseudo-randomly, in which case complete fault coverage was not guaranteed, or deterministically, in which case additional storage was required. By introducing a third option where counters are used for forming combinations of stored test data entries, the article ensures that additional storage is not required, and yet deterministic tests are reproduced from the stored test data. Thus, complete fault coverage can be achieved with a limited number of tests.

This contribution is important under the assumption that the hardware cost of an *LBIST* solution is justified by the need to perform in-field testing. In particular, it is assumed that storing partitioned deterministic test data entries on-chip is acceptable to achieve complete single stuck-at fault coverage as well as a high coverage of single-cycle gate-exhaustive faults.

The main limitations of the counter-based approach are the following. (1) Depending on the circuit, the requirement to achieve complete single stuck-at fault coverage may limit the reduction possible in the storage requirements relative to a compressed deterministic test set. (2) The software procedure requires repeated fault simulation to ensure that no loss of single stuck-at fault coverage occurs. This is a one-time computational cost that is required for reducing the stored test data.

The article is organized as follows. The partitioning of compressed tests into test data entries for on-chip storage is discussed in Section II. The on-chip test generation logic is

TABLE 1. **Partitioned seeds.**

i	$s_{i,0}$	$s_{i,1}$	$s_{i,2}$
0	00	01	10
1	11	10	11
2	10	01	00
3	01	11	01

TABLE 2. **Initial set of subvectors.**

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}
00	01	10	11	10	11	10	01	00	01	11	01

TABLE 3. **Counter-based combinations.**

c_0	c_1	c_2	$\sigma_{c_0,0}$	$\sigma_{c_0,1}$	$\sigma_{c_0,2}$	
0	1	2	00	01	10	s_0
1	2	3	01	10	11	
2	3	4	10	11	10	
3	4	5	11	10	11	s_1
4	5	6	10	11	10	
5	6	7	11	10	01	
6	7	8	10	01	00	s_2
7	8	9	01	00	01	
8	9	10	00	01	11	
9	10	11	01	11	01	s_3

described in Section III. A software procedure for computing the stored test data is described in Section IV. Experimental results for benchmark circuits are presented in Section V. Section VI concludes the article.

II. PARTITIONING OF COMPRESSED TESTS

This section describes how partitioning of compressed tests into test data entries for on-chip storage is carried out. The set of test data entries is optimized by the procedure described in Section IV.

Tests are assumed to be compressed into seeds for an *LFSR* that is part of the on-chip decompression logic. For simplicity of discussion, a test is compressed into a single seed. The discussion can be extended to the case where several seeds are used for producing a test. The length of the *LFSR* is denoted by L.

A compressed test set $S_{init} = \{s_0, s_1, \ldots, s_{m-1}\}$ consists of m seeds, from which the on-chip decompression logic produces m deterministic tests.

For on-chip storage, every seed in S_{init} is partitioned into p subvectors of equal length. The length of a subvector is denoted by l. The values of p and l are such that $p = \lceil L/l \rceil$, resulting in $p \cdot l \geq L$. When a seed $s_i \in S_{init}$ is partitioned into subvectors, the subvectors are denoted by $s_{i,0}$, $s_{i,1}$, …, $s_{i,p-1}$. If $p \cdot l > L$, the seeds are padded with unspecified values such that their length is equal to $p \cdot l$. When a seed is partitioned, unspecified values in its last subvector are specified randomly.

Table 1 shows an example of four 6-bit seeds for an *LFSR* of length $L = 6$. With $p = 3$ and $l = 2$, the subvectors of the partitioned seeds are shown in Table 1.

The subvectors from S_{init} are included in a set V that will be optimized and eventually used for on-chip storage. In [24], duplicated subvectors from S_{init} are avoided in the initial set V. In this article, all the subvectors from S_{init} are initially included in V such that $V = \{s_{0,0}, s_{0,1}, \ldots, s_{0,p-1}, s_{1,0}, s_{1,1}, \ldots, s_{1,p-1}, \ldots, s_{m-1,0}, s_{m-1,1}, \ldots, s_{m-1,p-1}\}$. This is important for achieving complete fault coverage. For ease of reference, the subvectors are renumbered such that $V = \{v_0, v_1, \ldots, v_{M-1}\}$, where initially, $M = m \cdot p$.

The initial set V for the example from Table 1 is shown in Table 2. With $p = 3$ subvectors required for forming a seed, three counters will be used in the example.

In general, p counters are required for forming seeds from the subvectors in V. The counters are denoted by c_0, c_1, …, c_{p-1}. The counters will go through $M - p + 1$ combinations such that $0 \leq c_0 \leq M - p$, and $c_{i+1} = c_i + 1$ for $0 \leq i < p - 1$. The value of c_0 is sufficient for identifying a combination because of the relation $c_{i+1} = c_i + 1$ for $0 \leq i < p - 1$. For $0 \leq c_0 \leq M - p$, the seed obtained for c_0 is denoted by σ_{c_0}, and $\sigma_{c_0} = \langle v_{c_0}, v_{c_1}, \ldots, v_{c_{p-1}} \rangle = \langle v_{c_0}, v_{c_0+1}, \ldots, v_{c_0+p-1} \rangle$.

The combinations of counter values for the example from Tables 1 and 2 are shown in Table 3. For every combination, the three subvectors that form the corresponding seed are also shown in Table 3. It can be seen from Table 3 that $\sigma_0 = \langle v_0, v_1, v_2 \rangle = s_0 \in S_{init}$, $\sigma_3 = \langle v_3, v_4, v_5 \rangle = s_1 \in S_{init}$, $\sigma_6 = \langle v_6, v_7, v_8 \rangle = s_2 \in S_{init}$, and $\sigma_9 = \langle v_9, v_{10}, v_{11} \rangle = s_3 \in S_{init}$.

In general, with the initial set V, $\sigma_{i \cdot p} = s_i \in S_{init}$ for $0 \leq i < m$. Therefore, the use of V with counter-based combinations guarantees complete fault coverage. It is important for this property to include in V all the subvectors obtained by partitioning the seeds from S_{init}.

III. ON-CHIP TEST GENERATION

The on-chip test generation logic is discussed in this section. The logic is similar to that used in [24] with the important difference that counters replace the *LFSR*s that produce pseudo-random combinations in [24].

A. ON-CHIP TEST GENERATION LOGIC

Figure 2 illustrates the on-chip test generation logic for the example from Tables 1, 2 and 3. The three counters c_0, c_1 and c_2 are shown on the left. The memory storing V is shown at the top of the figure. The on-chip decompression logic, with the *LFSR* it includes, are shown on the right. Three multiplexers receive control inputs from the counters. Each multiplexer produces one of the subvectors that is loaded into the *LFSR* in the decompression logic. The three subvectors together form a seed $\sigma_{c_0} = \langle v_{c_0}, v_{c_1}, v_{c_2} \rangle$. The decompression logic drives the scan chains of the circuit-under-test (not shown in Figure 2). The counter c_3 will be explained later.

In general, the parameters used for describing the on-chip test generation logic are the ones introduced in Section II, i.e., the number of subvectors in V is M, a seed for the *LFSR* in the decompression logic is formed from p subvectors each having l bits, and the length of the *LFSR* in the decompression logic is L. With this notation, the on-chip test generation logic includes the following components.

(1) V is stored in an $(l \cdot M)$-bit memory.

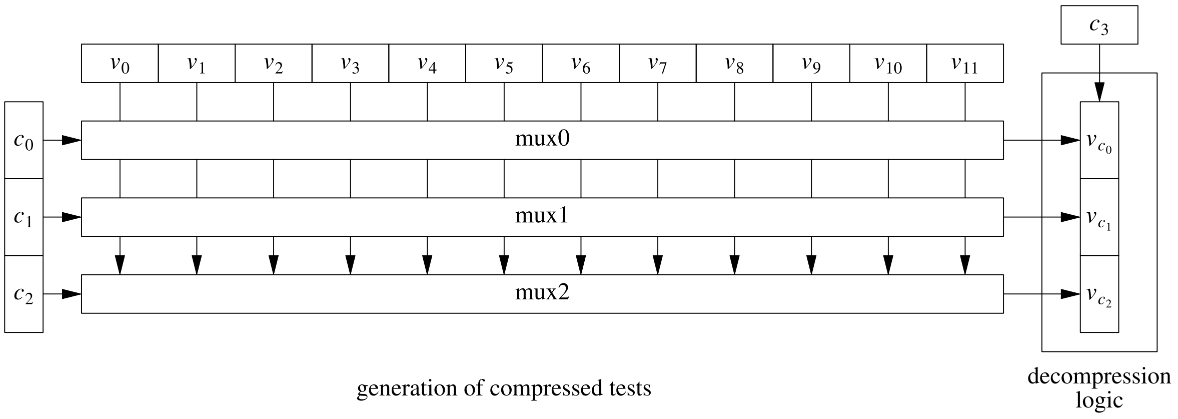
c_3
v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}
c_0
c_1
c_2
mux0
mux1
mux2
v_{c_0}
v_{c_1}
v_{c_2}
generation of compressed tests
decompression logic

FIGURE 2. **On-chip counter-based logic.**

(2) p $(log_2 M)$-bit counters, c_0, c_1, ..., c_{p-1}, hold a combination that forms a seed $\sigma_{c_0} = \langle v_{c_0}, v_{c_1}, \ldots, v_{c_{p-1}} \rangle$.

(3) Each one of p multiplexers has $(log_2 M)$-bit control inputs and M l-bit data inputs. The l-bit data output of a multiplexer is a subvector that is loaded into the *LFSR* in the decompression logic.

B. TEST APPLICATION

Test application proceeds as follows. An extension related to the additional counter c_3 in Figure 2 is described later.

For $0 \leq i < p$, the counter c_i is initialized to the value i. Thus, initially, the counters hold the combination $\langle 0, 1, \ldots, p-1 \rangle$. The counters are incremented $M - p$ times to form the combinations $\langle 1, 2, \ldots, p \rangle$, $\langle 2, 3, \ldots, p+1 \rangle$, ..., $\langle M-p, M-p+1, \ldots, M-1 \rangle$. These combinations are illustrated by Table 3.

For every one of the $M - p + 1$ combinations, i.e., for $0 \leq c_0 \leq M - p$, the counters are used for forming a seed $\sigma_{c_0} = \langle v_{c_0}, v_{c_1}, \ldots, v_{c_{p-1}} \rangle$. The decompression logic uses σ_{c_0} for applying a test to the circuit. The test is denoted by t_{c_0}.

C. COUNTER C_P

The role of the counter c_3 in Figure 2 is discussed next. In general, in addition to the counters c_0, c_1, ..., c_{p-1}, a counter denoted by c_p is used as follows.

The on-chip test generation logic is designed with parameters l and p such that $1 \leq l \leq L$ and $p = \lceil L/l \rceil$. With given values of l and p, the initial number of subvectors in V (before it is optimized by the procedure described in Section IV) is $M = |S_{init}| \cdot p \approx |S_{init}| \cdot L/l$. The number of tests applied to the circuit is $M - p$, which is also approximately $|S_{init}| \cdot L/l$. Thus, a higher value of l results in a lower number of applied tests. With a lower number of applied tests it is more difficult to optimize V, and the storage requirements of V are likely to remain higher.

The role of c_p is to equate the numbers of applied tests for different values of l and p such that a similar number of tests is applied based on the initial set V for every l and p. This is achieved as follows.

For a constant C_1, every seed σ_{c_0} produced by the on-chip test generation logic is used for applying $l \cdot C_1$ different tests. The counter c_p is initialized to $l \cdot C_1 - 1$ for every combination of $c_0, c_1, \ldots, c_{p-1}$, and counts down to zero. A different test is applied to the circuit based on σ_{c_0} for every value of c_p. This ensures that $l \cdot C_1$ tests are applied for every combination. Thus, with the inclusion of c_p, the number of applied tests for the initial set V is approximately $(|S_{init}| \cdot L/l) \cdot (l \cdot C_1) = |S_{init}| \cdot L \cdot C_1$, and this number is independent of l and p.

To obtain $l \cdot C_1$ different tests from a seed σ_{c_0}, for $0 \leq c_0 \leq M - p$, the approach from [28] is used. After a seed σ_{c_0} is loaded into the *LFSR*, the *LFSR* is clocked for c_p cycles. This takes the *LFSR* into a new state denoted by $\sigma_{c_0}^{c_p}$. The state $\sigma_{c_0}^{c_p}$ is used as a seed for applying a test to the circuit.

With $0 \leq c_0 \leq M - p$ and $0 \leq c_p < l \cdot C_1$, the set of tests applied to the circuit is denoted by T. The number of tests in T is $(M - p + 1) \cdot l \cdot C_1$. As V is reduced by the procedure described in Section IV, the number of tests in T is reduced without losing fault coverage.

IV. SOFTWARE PROCEDURE

The software procedure described in this section accepts a deterministic set of seeds S_{init}, and parameters l and p. It produces an optimized set of subvectors V for the counter-based *LBIST* approach. The software procedure is applied off-line to produce the set V for on-chip storage.

A. PRELIMINARIES

The set S_{init} is a compact set of seeds for single stuck-at faults. Using the seeds in S_{init}, without the *LBIST* logic, the decompression logic produces the test set T_{init}. The set of single stuck-at faults is denoted by F_0. The subset of F_0 detected by T_{init} is denoted by $D_0(T_{init})$. The *LBIST* approach targets a second set of faults, denoted by F_1, that consists of single-cycle gate exhaustive faults. The subset of F_1 detected by T_{init} is denoted by $D_1(T_{init})$. A second set of target faults is important for demonstrating the ability of the *LBIST* approach to detect faults with more complex behaviors than single stuck-at faults.

With the test set T produced by the *LBIST* logic, the subsets of detected faults are $D_0(T)$ and $D_1(T)$, respectively. As V is optimized and both V and T are reduced, the procedure requires that $D_0(T)$ would not lose any faults relative to $D_0(T_{init})$, and $D_1(T)$ would not be smaller than $D_1(T_{init})$. The asymmetry between the two requirements is based on the observation that T typically detects significantly more single-cycle gate-exhaustive faults than T_{init}, and maintaining the single stuck-at fault coverage is sufficient for ensuring that $D_1(T)$ would remain significantly larger than $D_1(T_{init})$.

TABLE 4. Subprocedure 1.

(a) Before Removal

	v_0 00	v_1 01	v_2 10	v_3 11	v_4 10	v_5 11	v_6 10	v_7 01	v_8 00	v_9 01	v_{10} 11	v_{11} 01
t_0	x	x	x									
t_1		x	x	x								
t_3				x	x	x						
t_7								x	x	x		
t_8									x	x	x	
e	1	2	2	2	1	1	0	1	2	2	1	0

(b) After Removal

	v_0 00	v_1 01	v_2 10	v_3 11	v_4 10	v_5 11	v_6 01	v_7 00	v_8 01	v_9 11
t_0	x	x	x							
t_1		x	x	x						
t_5						x	x	x		
t_6							x	x	x	
t_7								x	x	x
e	1	2	2	1	0	1	2	3	2	1

B. OVERVIEW

The software procedure initializes V as described in Section II. It then applies two subprocedures iteratively for removing subvectors from V. In every iteration, subprocedure 1 is applied first. If it does not remove any subvectors, subprocedure 2 attempts to remove subvectors from V one at a time. A new iteration starts as soon as a subvector is removed. The procedure terminates if subprocedure 2 considers all the subvectors in V for removal, and does not remove any one of them.

At the beginning of every iteration, the procedure uses the set V to compute the test set T. It then performs fault simulation with fault dropping of F_0 and F_1 under T. The procedure includes in a subset $T_{eff} \subset T$ the tests that are effective in detecting target faults. It then applies forward-looking reverse-order fault simulation to remove unnecessary tests from T_{eff}. This does not affect the on-chip test generation process of the *LBIST* approach. The set T_{eff} is used only by the software procedure to identify subvectors that are needed for detecting target faults.

A test $t_{c0}^{cp} \in T_{eff}$ is obtained from a seed σ_{c_0} that is formed from subvectors included in V, and using $0 \le c_p < l \cdot C_1$. The combination that defines σ_{c_0} is given by the counters $c_0, c_1, \ldots, c_{p-1}$. In terms of subvectors, $\sigma_{c_0} = \langle v_{c_0}, v_{c_1}, \ldots, v_{c_{p-1}} \rangle$. Based on T_{eff}, the procedure finds, for every subvector $v_i \in V$, the number of tests $t_{c0}^{cp} \in T_{eff}$ that v_i participates in. This number is denoted by $e(v_i)$. The value of $e(v_i)$ is incremented by one for every test $t_{c0}^{cp} \in T_{eff}$ with a seed σ_{c_0} such that $v_i \in \{v_{c_0}, v_{c_1}, \ldots, v_{c_{p-1}}\}$. Both subprocedures use the values of $e(v_i)$.

The two subprocedures are described next. Subprocedure 1 is unique to the use of counters as suggested in this article, and is not applicable to the pseudo-random combinations used in [24]. Subprocedure 2 is similar to the procedure used in [24], and it is described briefly.

C. SUBPROCEDURE 1

Subprocedure 1 removes from V subvectors for which $e(v_i) = 0$ is obtained based on T_{eff}. The removal of these subvectors does not affect the ability to produce the tests in T_{eff}.

Table 4 shows an example based on the example in Tables 1, 2 and 3. For simplicity, the counter c_p is not used in this example. For Table 4(a), $T_{eff} = \{t_0, t_1, t_3, t_7, t_8\}$. The subvectors for every test are marked with x's in Table 4(a). Thus, t_0 is obtained from $\sigma_0 = \langle v_0, v_1, v_2 \rangle = \langle 00, 01, 10 \rangle$, t_1 is obtained from $\sigma_1 = \langle v_1, v_2, v_3 \rangle = \langle 01, 10, 11 \rangle$, and t_8 is obtained from $\sigma_8 = \langle v_8, v_9, v_{10} \rangle = \langle 00, 01, 11 \rangle$.

The values of $e(v_i)$ are shown in the last row of Table 4(a). With these values, v_6 and v_{11} can be removed from V. The set V after the removal of these subvectors is shown in Table 4(b). The subvectors are renumbered $v_0, v_1, \ldots, v_9$. The tests that were included earlier in T_{eff} are available (with different subvector indices). For example, $\sigma_7 = \langle v_7, v_8, v_9 \rangle = \langle 00, 01, 11 \rangle$ was σ_8 earlier.

After removing subvectors from V, the procedure starts a new iteration where it recomputes the test sets T and T_{eff}. Table 4(b) shows the tests in T_{eff} based on the updated set V. An additional subvector can be removed from V, v_4.

In general, at the beginning of every iteration, the procedure computes T and T_{eff} based on V. Based on T_{eff} it finds the value of $e(v_i)$ for every $v_i \in V$. Subprocedure 1 then removes every subvector $v_i \in V$ for which $e(v_i) = 0$. If any subvector is removed from V, the iteration ends, and a new iteration starts.

D. SUBPROCEDURE 2

Subprocedure 2 is called only if subprocedure 1 cannot remove any subvector from V. Subprocedure 2 is based on the observation that it may be possible to remove a subvector $v_i \in V$ even if $e(v_i) > 0$. The removal of v_i allows the procedure to form new tests based on V. These tests may detect all the target faults. In this case, the removal of v_i is acceptable.

Subprocedure 2 considers the subvectors from V one by one until it finds a subvector that can be removed. The order by which subprocedure 2 considers the subvectors is the ascending value of $e(v_i)$. This is based on the assumption that a subvector is more likely to be removed if $e(v_i)$ is lower.

After removing a subvector $v_i \in V$, subprocedure 2 computes the test sets T and T_{eff}. If $|D_0(T)|$ does not decrease relative to its value before the removal of v_i, and $|D_1(T)| \ge D_1(T_{init})$, the removal of v_i is accepted, and a new iteration starts. Otherwise, v_i is reintroduced into V, and additional subvectors are considered for removal by subprocedure 2.

TABLE 5. Values of l and p for $s1423$.

l	p	subp1	entire
3	6	0.400	0.173
9	2	0.436	0.155
2	9	0.444	0.103
7	3	0.445	0.184
11	2	0.489	0.167
6	3	0.491	0.176
10	2	0.495	0.192
1	18	0.504	0.130
17	2	0.515	0.155
4	5	0.521	0.158
5	4	0.525	0.182
8	3	0.558	0.226
13	2	0.578	0.236
12	2	0.594	0.218
15	2	0.652	0.258
14	2	0.665	0.226
16	2	0.679	0.210

TABLE 6. Values of l and p for $b04$.

l	p	subp1	entire
7	4	0.838	0.228
1	28	0.860	0.505
21	2	0.860	0.287
4	7	0.866	0.176
2	14	0.872	0.258
3	10	0.873	0.262
5	6	0.882	0.247
14	2	0.882	0.309
15	2	0.914	0.252
6	5	0.945	0.284
22	2	0.971	0.324
16	2	0.975	0.252
10	3	0.977	0.284
8	4	0.992	0.269
17	2	1.000	0.286
11	3	1.075	0.289
12	3	1.084	0.277
9	4	1.116	0.217
18	2	1.153	0.284
19	2	1.158	0.339
24	2	1.185	-
13	3	1.188	-
20	2	1.239	-
23	2	1.377	-
26	2	1.393	-
25	2	1.418	-
27	2	1.446	-

V. EXPERIMENTAL RESULTS

The software procedure was applied to benchmark circuits considering single stuck-at and single-cycle gate-exhaustive faults.

A. SETUP AND COMPARISON

The implementation of the software procedure was performed in an academic environment using academic software tools. The procedure was applied to benchmark circuits that are available for academic research, and to which the academic software tools are applicable.

The counter-based approach is compared with two other approaches, for which a direct comparison is possible: the test set S_{init} represents the case where test data compression is used without partitioning of compressed tests as in [7], [15], and [16]; and the results from [24] represent the case where partitioned compressed tests are combined pseudo-randomly. *LBIST* approaches typically do not consider single-cycle gate-exhaustive faults, and many of them do not achieve complete single stuck-at fault coverage. A comparison with an approach that results in lower fault coverages is not meaningful since the cost of achieving fault coverage can be high for an *LBIST* approach. In addition, approaches that store test data that is derived from uncompressed tests have significantly higher storage requirements.

To allow a direct comparison with [24], the set of seeds S_{init} is the compact deterministic set of seeds for single stuck-at faults used in [24]. The set of single-cycle gate-exhaustive faults F_1 is also the one used in [24].

The case where $l = L$ and $p = 1$ corresponds to storing the entire set S_{init}, and using all the seeds from S_{init} for test application. This case is designated by an I (for initial). The storage requirements for this case are computed as $|S_{init}| \cdot L$, corresponding to storage of $|S_{init}|$ L-bit seeds.

For comparison with [24], up to two cases from [24] are reported. These cases are designated by an R (for pseudo-random combinations), and differ in the values of l and p. The first case has $l = 1$ and $p = L$. In this case, the procedure from [24] produces 1,000,000 pseudo-random seeds. When this number of pseudo-random seeds does not achieve complete single stuck-at fault coverage, the procedure from [24] selects to use $l > 1$ and $p = \lceil l/L \rceil$ for increased fault coverage. The value of $l > 1$ selected in [24] is the second case reported. In this case, the procedure uses 1,000,000 pseudo-random combinations of subvectors to form seeds.

The software procedure for the counter-based approach was applied with $1 \le l \le L - 1$, $p = \lceil L/l \rceil$ and $C_1 = 16$. To select values for l and p without applying the entire software procedure to every pair of values, a single application of subprocedure 1 was used for obtaining a set of subvectors denoted by $V_{l,p}$ for every l and p. The number of bits required for storing $V_{l,p}$ is computed as $l \cdot |V_{l,p}|$. The entire software procedure was applied with the 20 values of l and p that have the smallest values of $l \cdot |V_{l,p}|$. The results are reported for the values of l and p with the smallest final storage requirements.

Tables 5 and 6 illustrate this selection process considering benchmarks $s1423$ and $b04$ with $L = 18$ and $L = 28$, respectively. The fraction of storage requirements for $V_{l,p}$ relative to S_{init} is computed as $(l \cdot |V_{l,p}|)/(L \cdot |S_{init}|)$, and reported for every pair of l and p values under column *subp*1. The values of l and p are given in the ascending order of this fraction. For the first 20 pairs of values, the fraction of storage requirements is also reported after the entire software procedure is applied under column *entire*. The best storage requirements are obtained for the third pair in Table 5, and for the fourth pair in Table 6. These pairs are selected for the circuits.

For both pseudo-random and counter-based combinations, the number of applied tests is the number of tests from T that need to be applied until the final fault coverages are obtained for F_0 and F_1. Thus, although 1,000,000 tests are considered in [24], the number of applied tests may be lower if the fault coverages reach their final values before applying 1,000,000

TABLE 7. Experimental results group 1.

circuit	typ	inp	L	l	p	iter	subv	bits	frac	tests	eff	s.a.	diff	g.exh	diff	ntime
s35932	I	1763	13	13	1	0	57	741	1.000	57	57	89.809	0.000	98.509	0.000	1.00
s35932	R	1763	13	1	13	0	2	2	0.003	232	81	89.809	0.000	99.996	1.488	3.71
s35932	C	1763	13	7	2	3	3	21	0.028	224	81	89.809	0.000	99.989	1.480	29.29
sasc	I	132	13	13	1	0	41	533	1.000	41	41	100.000	0.000	67.374	0.000	1.00
sasc	R	132	13	1	13	0	2	2	0.004	416	167	100.000	0.000	93.923	26.550	4.83
sasc	C	132	13	9	2	3	3	27	0.051	262	151	100.000	0.000	92.007	24.633	29.67
des_area	I	367	14	14	1	0	158	2212	1.000	158	158	100.000	0.000	72.424	0.000	1.00
des_area	R	367	14	1	14	0	2	2	0.001	739	538	100.000	0.000	93.432	21.008	4.06
des_area	C	367	14	8	2	4	5	40	0.018	503	416	100.000	0.000	90.316	17.892	30.53
systemcdes	I	320	14	14	1	0	102	1428	1.000	102	102	100.000	0.000	93.391	0.000	1.00
systemcdes	R	320	14	1	14	0	2	2	0.001	1297	184	100.000	0.000	99.646	6.255	4.48
systemcdes	C	320	14	3	5	7	12	36	0.025	381	171	100.000	0.000	98.428	5.037	105.82
s1423	I	91	18	18	1	0	55	990	1.000	55	55	99.076	0.000	90.381	0.000	1.00
s1423	R	91	18	1	18	0	2	2	0.002	25579	137	99.076	0.000	99.758	9.377	54.11
s1423	C	91	18	2	9	95	51	102	0.103	1352	115	99.076	0.000	97.883	7.502	19123.00
usb_phy	I	112	18	18	1	0	36	648	1.000	36	36	100.000	0.000	82.536	0.000	1.00
usb_phy	R	112	18	1	18	0	2	2	0.003	2913	114	100.000	0.000	99.677	17.141	18.00
usb_phy	C	112	18	3	6	21	16	48	0.074	525	118	100.000	0.000	99.224	16.688	837.00
b04	I	78	28	28	1	0	34	952	1.000	34	34	99.851	0.000	79.263	0.000	1.00
b04	R	78	28	1	28	0	2	2	0.002	46318	239	99.851	0.000	98.560	19.297	136.00
b04	C	78	28	4	7	161	42	168	0.176	2241	125	99.851	0.000	91.014	11.751	64278.00
aes_core	I	788	28	28	1	0	380	10640	1.000	380	380	100.000	0.000	98.259	0.000	1.00
aes_core	R	788	28	1	28	0	2	2	0.000	3548	695	100.000	0.000	99.997	1.738	3.76
aes_core	C	788	28	21	2	6	8	168	0.016	2250	707	100.000	0.000	99.979	1.720	55.09
systemcaes	I	928	29	29	1	0	109	3161	1.000	109	109	99.995	0.000	79.613	0.000	1.00
systemcaes	R	928	29	1	29	0	2	2	0.001	39418	1047	99.995	0.000	99.989	20.376	14.78
systemcaes	C	928	29	5	6	139	37	185	0.059	2491	902	99.995	0.000	97.080	17.467	8435.29
s5378	I	214	36	36	1	0	133	4788	1.000	133	133	99.131	0.000	67.605	0.000	1.00
s5378	R	214	36	1	36	0	2	2	0.000	53432	898	99.131	0.000	99.533	31.928	43.62
s5378	C	214	36	7	6	201	108	756	0.158	11476	865	99.131	0.000	97.853	30.247	99321.78
b07	I	53	36	36	1	0	41	1476	1.000	41	41	99.915	0.000	59.459	0.000	1.00
b07	R	53	36	1	36	0	2	2	0.001	37994	388	97.041	-2.874	96.674	37.214	3168.50
b07	R	53	36	5	8	10	22	110	0.075	963258	404	99.915	0.000	99.636	40.177	81965.01
b07	C	53	36	6	6	126	59	354	0.240	5089	320	99.915	0.000	92.256	32.796	85992.00
simple_spi	I	146	38	38	1	0	46	1748	1.000	46	46	100.000	0.000	55.066	0.000	1.00
simple_spi	R	146	38	1	38	0	2	2	0.001	129112	453	98.620	-1.380	97.568	42.503	2078.71
simple_spi	R	146	38	2	19	1	3	6	0.003	620445	511	100.000	0.000	99.873	44.807	2712.29
simple_spi	C	146	38	3	13	154	108	324	0.185	4561	454	100.000	0.000	98.176	43.110	65494.50
i2c	I	145	43	43	1	0	58	2494	1.000	58	58	100.000	0.000	73.974	0.000	1.00
i2c	R	145	43	1	43	0	2	2	0.001	261195	278	96.577	-3.423	93.748	19.774	2086.92
i2c	R	145	43	15	3	138	35	525	0.211	158072	332	100.000	0.000	97.797	23.823	142832.39
i2c	C	145	43	2	22	65	771	1542	0.618	23969	291	100.000	0.000	97.301	23.327	100125.00
spi	I	274	44	44	1	0	360	15840	1.000	360	360	99.985	0.000	67.073	0.000	1.00
spi	R	274	44	1	44	0	2	2	0.000	65510	2432	99.354	-0.631	97.956	30.883	162.45
spi	R	274	44	6	8	13	51	306	0.019	683691	2536	99.985	0.000	99.931	32.858	1213.15
spi	C	274	44	12	4	610	69	828	0.052	12593	2243	99.985	0.000	94.900	27.827	81388.95
s13207	I	700	47	47	1	0	251	11797	1.000	251	251	98.462	0.000	62.259	0.000	1.00
s13207	R	700	47	1	47	0	2	2	0.000	190271	1905	98.462	0.000	97.623	35.364	181.57
s13207	C	700	47	27	2	203	83	2241	0.190	35243	1603	98.462	0.000	92.959	30.700	100089.23

tests. For the counter-based approach the number of tests is $(M-p+1)\cdot l\cdot C_1$, but the final fault coverages may be reached for a lower number of tests.

The test application time is determined by the number of applied tests as follows. Let the number of applied tests be N_A. Let the number of flip-flops in the longest scan chain be K. Each test requires the *LFSR* to be loaded with a seed. With the counter c_p under the counter-based approach, the *LFSR* is clocked for an average of $l \cdot C_1/2 = 8\ l$ clock cycles before a seed is obtained. A test consists of a scan-in operation of K scan shift cycles, a functional capture cycle, and a scan-out operation of K scan shift cycles. Scan-in and scan-out operations of consecutive tests are overlapped. Therefore, the test set requires $8lN_A$ clock cycles for the *LFSR* if c_p is used, $(N_A+1)K$ scan shift cycles, and N_A functional capture cycles. Scan shifting is typically carried out using a slower clock than the functional clock or the *LFSR* clock. With $K \gg 1$, $K \gg L \geq l$ and $N_A \gg 1$, the test application time is approximately that of $N_A \cdot K$ scan shift cycles. Thus, the number of applied tests also determines the test application time, and a comparison of the number of applied tests between different cases provides a comparison of the test application time between the same cases. A more detailed comparison is discussed later.

B. RESULTS

The results are reported in Tables 7, 8 and 9. For the circuits in Table 7, the procedure from [24] achieves complete single stuck-at fault coverage. For the circuits in Table 8, the procedure from [24] achieves less than complete single stuck-at fault coverage. With counter-based combinations as suggested in this article the single stuck-at fault coverage is always complete, equal to that of S_{init}. For the circuits in Table 9, only the procedure for the counter-based approach is applied to demonstrate its applicability to these circuits. The first subprocedure is used for selecting values of l and p for these circuits.

TABLE 8. Experimental results group 2.

circuit	typ	inp	L	l	p	iter	subv	bits	frac	tests	eff	s.a.	diff	g.exh	diff	ntime
wb_dma	I	738	47	47	1	0	84	3948	1.000	84	84	100.000	0.000	73.840	0.000	1.00
wb_dma	R	738	47	1	47	0	2	2	0.001	252167	771	99.615	-0.385	99.077	25.237	577.39
wb_dma	R	738	47	5	10	4	28	140	0.035	838936	767	99.978	-0.022	99.532	25.692	6023.19
wb_dma	R	738	47	10	5	0	277	2770	0.702	986814	792	99.989	-0.011	99.577	25.737	472.61
wb_dma	C	738	47	16	3	126	84	1344	0.340	20942	695	100.000	0.000	95.885	22.045	100059.36
s15850	I	611	57	57	1	0	197	11229	1.000	197	197	96.682	0.000	73.285	0.000	1.00
s15850	R	611	57	1	57	0	2	2	0.000	261955	1788	94.559	-2.124	87.761	14.475	2007.22
s15850	R	611	57	4	15	1	15	60	0.005	999797	2267	96.119	-0.563	92.853	19.567	3443.86
s15850	C	611	57	29	2	34	350	10150	0.904	161473	1849	96.682	0.000	90.213	16.928	100362.86
s9234	I	247	75	75	1	0	134	10050	1.000	134	134	93.475	0.000	70.825	0.000	1.00
s9234	R	247	75	1	75	0	2	2	0.000	261492	921	90.761	-2.714	91.300	20.475	2836.36
s9234	R	247	75	3	25	0	8	24	0.002	996368	1093	92.796	-0.679	96.131	25.306	2512.70
s9234	C	247	75	25	3	63	321	8025	0.799	127201	889	93.475	0.000	93.116	22.291	100497.60
s38584	I	1464	98	98	1	0	218	21364	1.000	218	218	95.852	0.000	88.313	0.000	1.00
s38584	R	1464	98	1	98	0	2	2	0.000	130393	1687	95.648	-0.204	99.513	11.201	1061.35
s38584	R	1464	98	5	20	4	28	140	0.007	995504	1724	95.796	-0.055	99.766	11.453	62243.02
s38584	C	1464	98	14	7	127	492	6888	0.322	108641	1664	95.852	0.000	99.750	11.437	100141.85
tv80	I	372	109	109	1	0	370	40330	1.000	370	370	99.527	0.000	77.264	0.000	1.00
tv80	R	372	109	1	109	0	2	2	0.000	261397	2106	97.947	-1.579	94.625	17.361	421.39
tv80	R	372	109	7	16	4	124	868	0.022	998072	2320	98.919	-0.607	97.424	20.159	33913.94
tv80	C	372	109	37	3	94	811	30007	0.744	478337	2262	99.527	0.000	97.533	20.269	100253.98
b15	I	483	113	113	1	0	266	30058	1.000	266	266	98.580	0.000	59.444	0.000	1.00
b15	R	483	113	1	113	0	2	2	0.000	261591	1387	95.186	-3.393	71.993	12.549	1297.76
b15	R	483	113	5	23	0	32	160	0.005	999779	1635	97.708	-0.872	83.513	24.069	1104.02
b15	C	483	113	19	6	49	1216	23104	0.769	367912	1513	98.610	0.030	81.348	21.904	100515.74
b20	I	527	119	119	1	0	238	28322	1.000	238	238	93.304	0.000	64.250	0.000	1.00
b20	R	527	119	1	119	0	2	2	0.000	260752	1355	87.714	-5.589	84.857	20.606	2647.18
b20	R	527	119	5	24	3	29	145	0.005	997430	1525	88.861	-4.442	86.057	21.807	13758.22
b20	C	527	119	1	119	10	23102	23102	0.816	367729	1484	93.308	0.004	87.756	23.506	100975.23
b14	I	280	128	128	1	0	290	37120	1.000	290	290	94.960	0.000	72.068	0.000	1.00
b14	R	280	128	1	128	0	2	2	0.000	2040	278	79.020	-15.940	67.510	-4.558	4204.46
b14	R	280	128	3	43	3	5	15	0.000	973582	790	85.883	-9.077	78.680	6.612	36678.39
b14	C	280	128	32	4	47	254	8128	0.219	128001	625	90.221	-4.739	79.983	7.916	100601.54
b14	C	280	128	32	4	23	910	29120	0.784	463873	810	94.960	0.000	84.670	12.602	101007.50

TABLE 9. Experimental results group 3.

circuit	typ	inp	L	l	p	iter	subv	bits	frac	tests	eff	s.a.	diff	g.exh	diff	ntime
wb_conmax	I	1900	54	54	1	0	911	49194	1.000	911	911	99.353	0.000	86.218	0.000	1.00
wb_conmax	C	1900	54	23	3	1	31	713	0.014	10336	3833	99.370	0.017	92.382	6.164	14.13
b17	I	1444	94	94	1	0	1307	122858	1.000	1307	1307	78.544	0.000	45.782	0.000	1.00
b17	C	1444	94	1	94	1	1168	1168	0.010	17191	2769	78.802	0.257	49.064	3.282	29.14

The first row for every circuit, with I under column *typ*, considers S_{init}. The next one or two rows for every circuit, with R under column *typ*, show the results from [24] with $l = 1$, and $l > 1$ if it increases the single stuck-at fault coverage. The last row shows the results of the counter-based procedure suggested in this article.

For $b14$, an additional row is shown for the counter-based procedure (the row before last) where a loss of fault coverage is allowed. The best result with the approach from [24] has a 9% fault coverage loss. The counter-based approach was allowed a 5% fault coverage loss to demonstrate that this reduces significantly its storage requirements. The article does not advocate a fault coverage loss, although this is common under *LBIST*. The additional row is included only to demonstrate the tradeoff between the storage requirements and the fault coverage.

In each row, after the circuit name, and the type of the procedure used, column *inp* shows the number of inputs. 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. Column *subv* shows the number of subvectors in V. Column *bits* shows the number of bits required for V, which equals $l \cdot |V|$. Column *frac* shows the number of bits required for V divided by the number of bits required for S_{init}. The fraction is computed as $(l \cdot |V|)/(L \cdot |S_{init}|)$. Column *tests* shows the number of applied tests until the final fault coverages are obtained. Column *eff* shows the number of tests in T_{eff}. The tests in T_{eff} increase the fault coverages when they are applied. Column *s.a.* shows the single stuck-at fault coverage. Column *g.exh* shows the single-cycle gate-exhaustive fault coverage. For both fault coverages, column *diff* shows the increase in the fault coverage relative to S_{init} (a negative number implies a reduction relative to S_{init}). Column *ntime* shows the runtime of the software procedure divided by the runtime required for fault simulation of S_{init}. This is referred to as the normalized runtime. The normalized runtime measures the computational effort in terms of the fault simulation time of the basic test set S_{init}.

C. DISCUSSION

The following points can be seen from Tables 7, 8 and 9. The set S_{init}, in the first row for every circuit, represents the conventional approach to test data compression. Under this approach, deterministic tests are compressed into seeds for an *LFSR*. For both [24] and the counter-based approach in this article, and most of the circuits considered, the storage

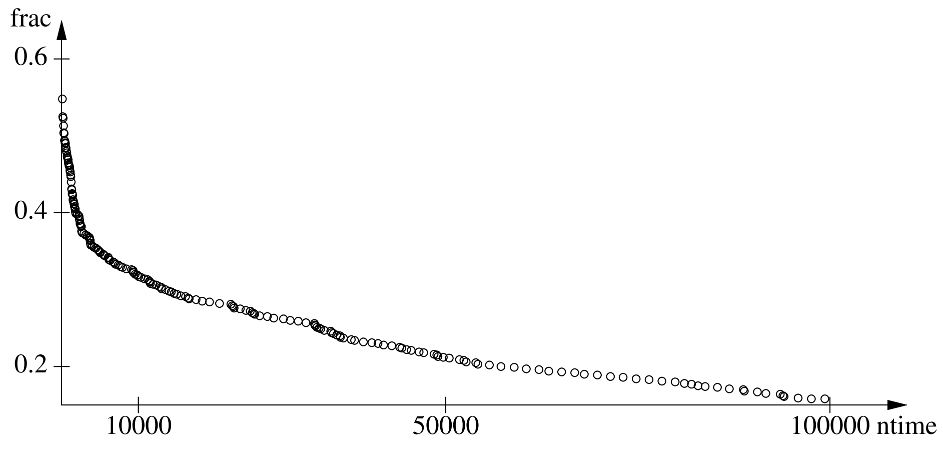

FIGURE 3. Storage reduction vs. normalized runtime for *s*5378.

requirements of V are significantly lower than the storage requirements of S_{init}. The reduction is made possible by the partitioning of S_{init} into subvectors, and the reduction of V that is possible when combinations of subvectors use each subvector multiple times for test application.

In Table 7, both the procedure from [24] and the counter-based procedure achieve complete single stuck-at fault coverage. With up to 1,000,000 pseudo-random combinations, the procedure from [24] has lower storage requirements for V. However, it typically needs to apply significantly more tests than the counter-based procedure described in this article.

The increased number of tests translates into a higher number of clock cycles required for test application. The test application time was discussed in Section V-A. Taking all the clock cycles into account, and assuming that the same clock is used for scan shifting, functional capture cycles, and *LFSR* clocking, *s*13207 requires 133M clock cycles for the tests from [24] (without any *LFSR* clocking), and 32M clock cycles for the counter-based approach (with *LFSR* clocking included). This is a 4-fold reduction in the number of clock cycles using the counter-based approach. A similar calculation for *spi* shows a 40-fold reduction.

The single-cycle gate-exhaustive fault coverage is higher than that of S_{init} for both *LBIST* approaches. With more applied tests, the fault coverage is typically higher with the approach from [24].

In Table 8, the procedure from [24] does not achieve complete single stuck-at fault coverage, whereas the counter-based approach does. Even with the higher single stuck-at fault coverage, the counter-based approach uses fewer tests than the approach from [24] that uses pseudo-random combinations.

The normalized runtime does not increase with the size of the circuit. Thus, the software procedure scales similar to a fault simulation procedure for the basic test set S_{init}. Fault simulation time is manageable for circuits of any size.

It is possible to reduce the runtime of the software procedure by noting that the procedure saturates, causing later iterations to take longer while achieving smaller reductions in the storage requirements. To demonstrate this point, Figure 3 shows the storage reduction as a function of the normalized runtime for benchmark circuit *s*5378. Based on Figure 3, a significant reduction in storage requirements is achieved even if the normalized runtime is limited.

Overall, the counter-based approach achieves complete single stuck-at fault coverage with reduced storage requirements compared with S_{init}, and a reduced number of tests even with a higher single stuck-at fault coverage compared with [24]. The single-cycle gate-exhaustive fault coverage is significantly higher than that of S_{init}.

VI. CONCLUDING REMARKS

This article introduced a new option for the formation of tests under a class of logic built-in self-test (*LBIST*) approaches that store partitioned deterministic test data on-chip, and produce tests by combining stored test data entries. The two options considered earlier used pseudo-random combinations, or deterministic combinations that were also stored on-chip. Under the counter-based option introduced in this article, combinations of stored test data entries that form tests are created using counters. Counters do not require additional storage. However, they can ensure that deterministic tests are reproduced from the stored test data. Thus, complete fault coverage can be achieved with a limited number of tests. A software procedure optimized the stored test data for counter-based test application, and reduced the storage requirements, without losing fault coverage. Experimental results for benchmark circuits demonstrated the advantages of the counter-based approach when test data entries are obtained by partitioning compressed deterministic tests. In this case, the universally available on-chip decompression logic is part of the on-chip test generation logic.

Future work can consider the following directions to obtain an industry-strength implementation of the counter-based approach. (1) Instead of the academic implementation reported in this article, it is possible to implement the counter-based approach using commercial software tools. (2) Verilog code can be written for automating the insertion of the *LBIST* logic. (3) A functional memory (one that already exists on-chip) can be used for storing test data entries, instead of using a dedicated memory for *LBIST*.

REFERENCES

[1] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-in Test for VLSI Pseudorandom Techniques*. Hoboken, NJ, USA: Wiley, 1987.

[2] N. A. Touba and E. J. McCluskey, ''Bit-fixing in pseudorandom sequences for scan BIST,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 4, pp. 545–555, Apr. 2001.

[3] S. Hellebrand, H.-G. Liang, and H.-J. Wunderlich, ''A mixed mode BIST scheme based on reseeding of folding counters,'' *J. Electron. Test.*, vol. 17, pp. 341–349, Jun. 2001.

[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,'' *IEEE Trans. Comput.*, vol. 51, no. 1, pp. 1282–1293, Nov. 2002.

[5] S. Pateras, ''Security vs. test quality: Fully embedded test approaches are the key to having both,'' in *Proc. Int. Test Conf.*, 2004, p. 1413.

[6] D. Xiang, M. Chen, and H. Fujiwara, ''Using weighted scan enable signals to improve test effectiveness of scan-based BIST,'' *IEEE Trans. Comput.*, vol. 56, no. 12, pp. 1619–1628, Dec. 2007.

[7] D. J. C. Alves and E. Barros, ''A logic built-in self-test architecture that reuses manufacturing compressed scan test patterns,'' in *Proc. 22nd Annu. Symp. Integr. Circuits Syst. Design, Chip Dunes*, Aug. 2009, pp. 1–6.

[8] 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,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 2, pp. 299–312, Feb. 2010.

[9] R. S. Oliveira, J. Semião, 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. 12th Latin Amer. Test Workshop (LATW)*, Mar. 2011, pp. 1–6.

[10] 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.*, Nov. 2011, pp. 54–59.

[11] M. E. Imhof and H.-J. Wunderlich, ''Bit-flipping scan—A unified architecture for fault tolerance and offline test,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2014, pp. 1–6.

[12] F. Reimann, M. Glaß, J. Teich, A. Cook, L. R. Gómez, D. Ull, H.-J. Wunderlich, U. Abelein, and P. Engelke, ''Advanced diagnosis: SBST and BIST integration in automotive E/E architectures,'' in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–6.

[13] G. Contreras, N. Ahmed, L. Winemberg, and M. Tehranipoor, ''Predictive LBIST model and partial ATPG for seed extraction,'' in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFTS)*, Oct. 2015, pp. 139–146.

[14] C.-M. Shiao, W.-C. Lien, and K.-J. Lee, ''A test-per-cycle BIST architecture with low area overhead and no storage requirement,'' in *Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT)*, Apr. 2016, pp. 1–4.

[15] Y. Liu, N. Mukherjee, J. Rajski, S. M. Reddy, and J. Tyszer, ''Deterministic stellar BIST for in-system automotive test,'' in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2018, pp. 1–9.

[16] B. Kaczmarek, G. Mrugalski, N. Mukherjee, J. Rajski, L. Rybak, and J. Tyszer, ''Test sequence-optimized BIST for automotive applications,'' in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2020, pp. 1–6.

[17] A. Koneru and K. Chakrabarty, ''An interlayer interconnect BIST and diagnosis solution for monolithic 3-D ICs,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 3056–3066, Oct. 2020.

[18] I. Pomeranz, ''Storage-based built-in self-test for gate-exhaustive faults,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 10, pp. 2189–2193, Oct. 2021.

[19] I. Pomeranz, ''Zoom-in feature for storage-based logic built-in self-test,'' in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2021, pp. 1–6.

[20] D. K. Maity, S. K. Roy, and C. Giri, ''A cost-effective built-in self-test mechanism for post-manufacturing TSV defects in 3D ICs,'' *ACM J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 4, pp. 1–23, Oct. 2022.

[21] M. B. G. Remadevi and R. Bakthavatchalu, ''Design of a programmable low power linear feedback shift register for BIST applications,'' in *Proc. IEEE Int. Test Conf. India (ITC India)*, Jul. 2022, pp. 1–4.

[22] S. Wang, X. Zhou, Y. Higami, H. Takahashi, H. Iwata, Y. Maeda, and J. Matsushima, ''Test point insertion for multi-cycle power-on self-test,'' *ACM Trans. Design Autom. Electron. Syst.*, vol. 28, no. 3, pp. 1–21, May 2023.

[23] S. Gopalsamy and I. Pomeranz, ''Fully deterministic storage based logic built-in self-test,'' in *Proc. IEEE 41st VLSI Test Symp. (VTS)*, Apr. 2023, pp. 1–7.

[24] I. Pomeranz, ''Storage-based logic built-in self-test with partitioned deterministic compressed tests,'' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 31, no. 9, pp. 1259–1268, Sep. 2023.

[25] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, ''Silent data corruptions at scale,'' 2021, *arXiv:2102.11245*.

[26] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, ''Cores that don't count,'' in *Proc. Workshop Hot Topics Operating Syst.*, Jun. 2021, pp. 9–16.

[27] I. Pomeranz and S. M. Reddy, ''Static test data volume reduction using complementation or modulo-*M* addition,'' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 6, pp. 1108–1112, Jun. 2011.

[28] I. Pomeranz, ''Extra clocking of LFSR seeds for improved path delay fault coverage,'' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 544–552, Feb. 2020.

[29] I. Pomeranz, ''Input test data volume reduction using seed complementation and multiple LFSRs,'' in *Proc. IEEE 38th VLSI Test Symp. (VTS)*, Apr. 2020, pp. 1–6.

IRITH POMERANZ (Fellow, IEEE) received the B.Sc. degree (summa cum laude) in computer engineering and the D.Sc. degree from the Department of Electrical Engineering, Technion—Israel Institute of Technology, Haifa, Israel, in 1985 and 1989, respectively.

From 1989 to 1990, she was a Lecturer with the Department of Computer Science, Technion—Israel Institute of Technology. From 1990 to 2000, she was a Faculty Member with the Department of Electrical and Computer Engineering, The University of Iowa, Iowa City, IA, USA. In 2000, she joined Purdue University, West Lafayette, IN, USA, where she is currently the Cadence Professor with in the Elmore Family School of Electrical and Computer Engineering.

Dr. Pomeranz is a Golden Core Member of the IEEE Computer Society. She was a recipient of the NSF Young Investigator Award, in 1993, and 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 coauthored 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 in February 2020. She served as the Program Co-Chair for the 1999 Fault-Tolerant Computing Symposium. She served as the Program Chair for the 2004 and 2005 VLSI Test Symposiums and the General Chair for the 2006 VLSI Test Symposium. She served as an Associate Editor for *ACM Transactions on Design Automation*, IEEE TRANSACTIONS ON COMPUTERS, and IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS. She served as a Guest Editor for IEEE TRANSACTIONS ON COMPUTERS Special Issue on ''Dependability of Computing Systems,'' in January 1998.