

Storage-Based Built-In Self-Test for Gate-Exhaustive Faults

Irith Pomeranz

Abstract—Built-in self-test (*BIST*) approaches are suitable for in-field testing since they do not require a tester for storage and application of test data. They also reduce the security vulnerabilities associated with loading and unloading of external test data into scan chains. As technologies evolve, in-field testing needs to address more complex defect and aging mechanisms that require specific deterministic tests. This can be addressed by *BIST* approaches that store test data on-chip, and use the data for on-chip generation of both random and deterministic tests. In this case, there is a tradeoff between the amount of stored test data and the comprehensiveness of the test set that can be applied. This paper explores this tradeoff in a specific context that has the following main features. (1) The initial stored test data is based on a stuck-at test set. (2) The target faults are single-cycle gate-exhaustive faults. (3) The stored test data is enhanced gradually by test data based on a gate-exhaustive test set to increase the coverage of gate-exhaustive faults.

Index Terms—Built-in self-test (*BIST*), gate-exhaustive faults, on-chip test generation, stuck-at faults.

I. INTRODUCTION

Built-in self-test (*BIST*) approaches are suitable for in-field testing since they do not require a tester for storage and application of test data [1]–[10]. They also reduce the security vulnerabilities associated with loading and unloading of external test data into scan chains [3].

As technologies evolve, in-field testing needs to address more complex defect and aging mechanisms that require specific tests [11]–[14]. This can be addressed by *BIST* approaches that store test data on-chip, and use the data for on-chip generation of both random and deterministic tests.

Different from hybrid approaches that combine test data compression with on-chip test generation [15]–[18], a *BIST* approach stores all the test data on-chip. This is the type of approach considered in this paper. In this approach, there is a tradeoff between the amount of stored test data and the comprehensiveness of the test set that can be applied. The paper explores this tradeoff in the following context.

The circuit under consideration has n scan chains of length l . One component of the stored test data is a set S_i of scan vectors. Using only scan vectors from S_i , an on-chip test generation logic applies a fixed number of tests referred to as random, where scan vectors are selected randomly from S_i , and a small number of tests referred to as deterministic that consist of specific scan vectors from S_i . The set of deterministic tests is denoted by $T_{i,dtrm}$, and it requires additional storage of test data (indices of scan vectors from S_i that form the tests in $T_{i,dtrm}$). The test set produced on-chip is denoted by T_i , and it changes with S_i and $T_{i,dtrm}$. The on-chip test generation logic is the same for every S_i and $T_{i,dtrm}$, and every circuit. Only its parameters (e.g., memory and multiplexer sizes) are circuit-dependent.

A deterministic test set T_{sa} for single stuck-at faults is used as an initial source for test data. The test data is a set S_0 of scan vectors based on T_{sa} , and a set $T_{0,dtrm}$ of deterministic tests to complement the random tests based on S_0 . With S_0 and $T_{0,dtrm}$, and the on-chip test generation logic, T_0 detects all the detectable single stuck-at faults. The set of single stuck-at faults is denoted by F_{sa} . The effectiveness of the random tests is enhanced by using stored scan vectors from T_{sa} .

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.

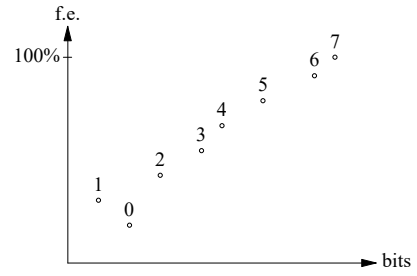


Fig. 1. Tradeoff between fault efficiency and stored test data.

In addition to stuck-at faults, the set of target faults includes single-cycle (static) gate-exhaustive faults [19]. These are used for representing the need to cover a large number of defects with specific behaviors that are more difficult to detect than stuck-at faults. Gate-exhaustive faults are suitable for this purpose since their number is large, and they require more values to be assigned by a test. The set of detectable gate-exhaustive faults is denoted by F_{geh} , and a deterministic test set for gate-exhaustive faults is denoted by T_{geh} . It is also possible to consider a test set for transition faults as an initial source for test data, and two-cycle (dynamic) gate-exhaustive faults as additional target faults.

When the pair $(S_0, T_{0,dtrm})$ is used for on-chip test generation, the coverage of gate-exhaustive faults may not be complete. An iterative software procedure produces a series of pairs $(S_0, T_{0,dtrm})$, $(S_1, T_{1,dtrm})$, ..., $(S_{m-1}, T_{m-1,dtrm})$. For $i > 0$, the procedure obtains S_i from S_{i-1} by adding at least one new scan vector based on T_{geh} . The scan vectors added to S_i allow the coverage of gate-exhaustive faults to be increased when the on-chip test generation logic is used for $(S_i, T_{i,dtrm})$. With $(S_{m-1}, T_{m-1,dtrm})$, complete coverage of gate-exhaustive faults is achieved. In many cases, the increased fault coverage is achieved by random tests that use scan vectors from S_i , and $T_{i,dtrm} = \emptyset$.

Figure 1 illustrates the tradeoff explored by the software procedure that produces the pairs $(S_i, T_{i,dtrm})$ for $i \geq 0$. Figure 1 is based on the results obtained later for benchmark circuit *s1423*. A circle with i above it corresponds to a test set T_i . The horizontal axis in Figure 1 shows the number of storage bits for T_i . The vertical axis shows the fault efficiency, which is the percentage of detected faults out of the detectable faults, achieved for gate-exhaustive faults by T_i . The fault efficiency reaches 100% in iteration $i = 7$.

In Figure 1, when the pair $(S_1, T_{1,dtrm})$ is computed in iteration 1, it has a lower number of storage bits than $(S_0, T_{0,dtrm})$, and a higher fault efficiency. Therefore, the solution obtained in iteration 1 is preferred over the solution obtained in iteration 0. Overall, there are seven viable solutions in Figure 1, corresponding to $(S_i, T_{i,dtrm})$ for $i = 1, 2, 3, 4, 5, 6$ and 7.

The approach described in this paper is developed for the case where a circuit has a large number of scan chains, making the scan chain length small enough for storage of scan vectors to be feasible. A large number of scan chains is also used by test data compression approaches to control the test application time.

The paper is organized as follows. The on-chip test generation logic is described in Section II. A software procedure for computing the sets S_i and $T_{i,dtrm}$ is described in Section III. Experimental results for benchmark circuits are presented in Section IV.

II. ON-CHIP TEST GENERATION

The scan configuration assumed in this paper has n scan chains. For simplicity it is assumed that all the scan chains have the same length, l . This can be achieved by padding each scan chain until its

TABLE I
EXAMPLE TEST SET

(a) Scan Vectors							
p	0	1	2	3	4	5	
s_p	011	110	000	101	010	001	

(b) Test Set							
j	π_j	j_0	j_1	j_2	$t_{j,0}$	$t_{j,1}$	$t_{j,2}$
0	0	4	1	5	010	110	001
1	0	1	1	3	110	110	101
2	0	2	1	5	000	110	001
3	0	2	2	0	000	000	011
4	1	5	4	5	001	010	001
5	1	5	5	5	001	001	001

length reaches l . Padding is not needed for the physical circuit, only for the model used by the software procedure to compute the test data for the on-chip test generation logic.

One component of the test data stored on-chip is a set of scan vectors, $S_i = \{s_0, s_1, \dots, s_{v-1}\}$. For illustration, Table I(a) shows the set S_0 obtained for a circuit with $n = 3$ scan chains of length $l = 3$.

A test t_j is formed by selecting n scan vectors from S_i , one for every scan chain. With scan vectors $s_{j_0}, s_{j_1}, \dots, s_{j_{n-1}}$, we have that $t_j = \langle s_{j_0}, s_{j_1}, \dots, s_{j_{n-1}} \rangle$. For $0 \leq k < n$, the scan vector for scan chain k is denoted by $t_{j,k} = s_{j_k}$.

Table I(b) shows a test set T_0 obtained for the circuit from Table I(a). For every test t_j , Table I(b) shows the indices j_0, j_1, j_2 , and the subtests $t_{j,0}, t_{j,1}$ and $t_{j,2}$. The variable π_j is explained next.

The on-chip test generation logic applies two types of tests. A random test $t_j = \langle s_{j_0}, s_{j_1}, \dots, s_{j_{n-1}} \rangle$ is designated by $\pi_j = 0$. In this case, for $0 \leq k < n$, the scan vector s_{j_k} for the subtest $t_{j,k}$ is selected randomly.

A deterministic test $t_j = \langle s_{j_0}, s_{j_1}, \dots, s_{j_{n-1}} \rangle$ is designated by $\pi_j = 1$. In this case, the set of indices j_0, j_1, \dots, j_{n-1} is stored in an on-chip memory.

The on-chip test generation logic is illustrated by Figure 2. The lower part of Figure 2 shows the memory storing the set of scan vectors S_i . The size of the memory is $v \times l$. It also shows scan chain k of length l . A multiplexer called $MUX2k$ selects one of the scan vectors from S_i depending on the variable called $addrk$. The number of bits in $addrk$ is $\log_2(v)$. The selected scan vector is scanned into scan chain k . For a test t_j it defines the subtest $t_{j,k}$.

The value of $addrk$ is computed by the logic in the upper part of Figure 2. If only random tests are applied, the dashed part of Figure 2 is not needed, and an *LFSR* produces the values for $addrk$. In general, the deterministic part of the test set T_i , $T_{i,dtrm}$, is stored in a memory of size $d \times n \times \log_2(v)$, where d is the number of deterministic tests. The entries inside $T_{i,dtrm}$ in Figure 2 correspond to scan chain k . These are indices of scan vectors that need to be loaded into scan chain k under the deterministic tests t_0, t_1, \dots, t_{d-1} .

A counter denoted by *cnt* determines which test is applied through a multiplexer denoted by $MUX1k$. A count value between 0 and $d-1$ corresponds to a deterministic test from $T_{i,dtrm}$. A count value of d corresponds to a random test. In this case, the *LFSR* provides $\log_2(v)$ bits selecting a scan vector randomly from S_i . The counter stays at d to apply R random tests, for a parameter R .

The memories S_i and $T_{i,dtrm}$, as well as the counter and *LFSR*, are common to all the scan chains. In the case of the *LFSR*, each scan chain uses a distinct subset of bits to obtain a different random number. In addition, each scan chain requires two multiplexers.

The overall storage requirements for the two memories are $v \cdot l + d \cdot n \cdot \log_2(v)$ bits. The memories dominate the size of the on-chip test generation logic.

The entire test generation logic, including both multiplexers for

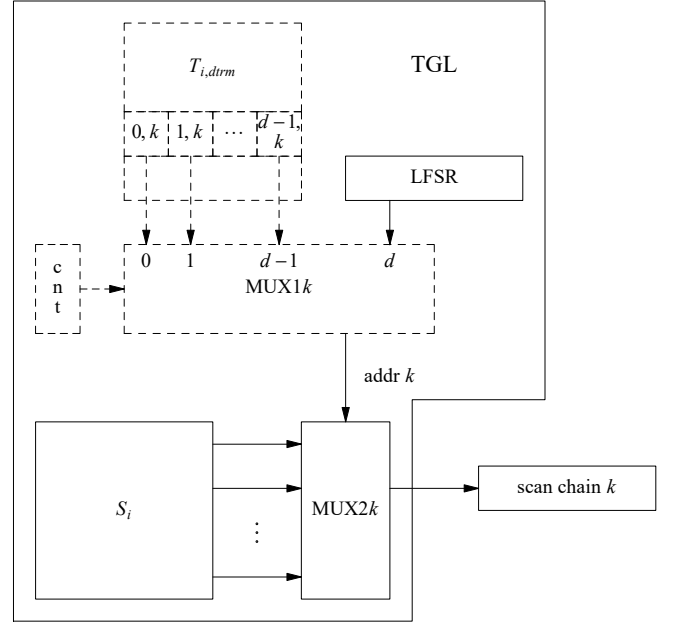


Fig. 2. On-chip test generation logic.

every scan chain, resides close to the memories, within the outline marked *TGL* in Figure 2. Each scan chain is driven by a single line, represented by the output of $MUX2k$ in Figure 2. The routing overhead is similar to that of test data decompression logic that drives all the scan chains.

For the output response it is assumed that sequential output compaction will be performed by output compaction logic such as a multiple-input shift-register (*MISR*) [1].

III. PROCEDURE FOR COMPUTING STORED TEST DATA

This section describes an iterative software procedure for computing sets of scan vectors S_i and deterministic tests $T_{i,dtrm}$ for on-chip test generation. For simplicity of discussion, S_i is associated with a test set T_i that consists of both random and deterministic tests. The non-random tests in T_i define $T_{i,dtrm}$.

A. Overview

The procedure accepts a test set T_{sa} for single stuck-at faults, and a test set T_{geah} for gate-exhaustive faults. It produces sets of scan vectors S_0, S_1, \dots, S_{m-1} , with test sets T_0, T_1, \dots, T_{m-1} .

At the beginning of iteration $i = 0$, the procedure initializes S_0 based on T_{sa} , as follows. All the distinct scan vectors of T_{sa} are included in S_0 , and $T_0 = T_{sa}$ initially. With this initialization, every test in T_0 can be expressed in terms of scan vectors from S_0 .

At the beginning of iteration $i > 0$, $S_i = S_{i-1}$ and $T_i = T_{i-1}$. At the end of iteration $0 \leq i \leq m-2$, the procedure adds at least one scan vector to S_i . It adds tests that use the new scan vectors to T_i . The procedure terminates when, at the end of iteration $m-1$, it does not add any scan vectors to S_{m-1} .

An arbitrary iteration i proceeds as described next. The procedure referred to as Procedure 0 is applied first to remove unnecessary scan vectors from S_i . The test set T_i is modified to ensure that it uses only scan vectors from S_i .

The procedure referred to as Procedure 1 first stores the current test set T_i in a test set denoted by T_{cand} , and initializes T_i to be empty. All the target single stuck-at faults are included in F_{sa} , and all the target gate-exhaustive faults are included in F_{geah} .

For a parameter R , Procedure 1 includes R random tests in T_i . For every test it performs fault simulation with fault dropping of F_{sa} and F_{gehx} . It then uses deterministic tests from T_{cand} to detect additional faults from F_{sa} and F_{gehx} using only scan vectors from S_i .

All the detectable stuck-at faults from F_{sa} are guaranteed to be detected by T_i after Procedure 1 is applied. The procedure terminates if all the detectable gate-exhaustive faults from F_{gehx} are detected by T_i as well. Otherwise, Procedure 2 adds to T_i a limited number of additional deterministic tests based on T_{gehx} to detect additional gate-exhaustive faults. As much as possible, Procedure 2 uses only scan vectors that are already included in S_i . When this is not possible, Procedure 2 adds to T_i tests that require new scan vectors, which are added to S_i . The procedure prefers tests that require the smallest possible numbers of new scan vectors.

As T_i is modified, the procedure maintains fault detection information for T_i . Fault detection information is obtained by fault simulation with fault dropping of $F_{sa} \cup F_{gehx}$ under T_i . For a fault $f \in F_{sa} \cup F_{gehx}$, the first test in T_i that detects it is denoted by $t_{det(f)}$. For an undetected fault, $det(f) = -1$. Fault simulation with fault dropping is also applied at the end of every procedure.

B. Procedure 0

For Procedure 0 all the tests in T_i are considered as deterministic. Procedure 0 associates with every scan vector $s_p \in S_i$ the number of times it is used by a test in T_i . This number is denoted by $a(s_p)$. The procedure considers the scan vectors by increasing order of $a(s_p)$. This is based on the expectation that it will be easier to remove from S_i a scan vector that is used fewer times by tests from T_i . The procedure maintains the variables $a(s_p)$ up-to-date as it modifies the test set. A scan vector s_p with $a(s_p) = 0$ is removed from S_i .

When the procedure considers s_p for removal, let the subset of tests where s_p appears be $T(s_p)$. Let the subset of faults that the tests in $T(s_p)$ detect be $F(s_p)$. To facilitate the modification of the tests in $T(s_p)$, the procedure simulates $F(s_p)$ under $T_i \setminus T(s_p)$. If a fault $f \in F(s_p)$ is detected by a test $t_j \in T_i \setminus T(s_p)$, the procedure assigns $det(f) = j$, and removes the fault from $F(s_p)$.

The procedure considers every subtest $t_{j,k}$ such that $t_{j,k} = s_p$ separately. For $t_{j,k}$, it considers as an alternative every scan vector $s_q \in S_i$ such that $a(s_q) \neq 0$ and $q \neq p$. To check whether s_q is an acceptable alternative, the procedure assigns $t_{j,k} = s_q$. It then simulates every fault $f \in F_{sa} \cup F_{gehx}$ such that $det(f) = j$ under the modified t_j . The modification of t_j is accepted if all the faults with $det(f) = j$ are detected. Otherwise, the procedure reassigns $t_{j,k} = s_p$. If none of the alternatives to s_p is acceptable, the procedure concludes that s_p cannot be removed, and it does not consider other subtests where s_p appears.

The procedure considers the tests in $T(s_p)$ by decreasing order of the number of faults they detect. This is based on the expectation that tests with larger numbers of detected faults are more difficult to modify. If such a test cannot be modified, the procedure will not consider other tests in $T(s_p)$.

The procedure considers the replacement scan vectors s_q by increasing order of the number of appearances in T_i , $a(s_q)$. This is based on the expectation that a more uniform use of scan vectors allows faults to be detected more uniformly, and makes it easier to modify the test set.

After considering all the scan vectors in S_i for removal, if any scan vector was removed from S_i , the procedure performs another pass over the scan vectors in S_i , in case additional scan vectors can be removed after modifying the test set. Procedure 0 terminates after a pass that does not reduce the number of scan vectors in S_i .

C. Procedure 1

Procedure 1 initially assigns $T_{cand} = T_i$, $T_i = \emptyset$, and includes all the target faults in F_{sa} and F_{gehx} .

Procedure 1 includes R random tests in T_i , for a parameter R . A random test t_j is constructed by selecting $t_{j,k}$ randomly from S_i (or using the *LFSR* in Figure 2), for $0 \leq k < n$. The procedure performs fault simulation with fault dropping of t_j under $F_{sa} \cup F_{gehx}$, for every random test t_j .

Next, the procedure uses tests from T_{cand} as deterministic tests to detect additional faults from $F_{sa} \cup F_{gehx}$. The goal is to ensure that both the stuck-at and gate-exhaustive fault coverages of T_i are not lower than those of T_{cand} . This ensures that the stuck-at fault coverage is maintained, and the gate-exhaustive fault coverage increases monotonically with every iteration until all the detectable gate-exhaustive faults are detected in the last iteration. For every test $t_j \in T_{cand}$, the procedure applies the following steps.

The procedure simulates $F_{sa} \cup F_{gehx}$ under t_j . If no faults are detected, the procedure does not consider t_j further. Otherwise, it attempts to modify t_j to increase the number of faults it detects out of $F_{sa} \cup F_{gehx}$. For this purpose, the procedure includes in $F(t_j)$ all the faults that t_j detects. It then considers every scan chain $0 \leq k < n$, and every scan vector $s_p \in S_i$. If $t_{j,k} \neq s_p$, the procedure defines a test t_j^{mod} that is equal to t_j , except that $t_{j,k}^{mod} = s_p$. It simulates the faults in $F(t_j)$ under t_j^{mod} . If all the faults are detected, it also simulates the faults in $F_{sa} \cup F_{gehx}$ under t_j . Let the subset of detected faults be $F(t_j^{mod})$. The procedure accepts the modification of t_j if $F(t_j) \subseteq F(t_j^{mod})$. In this case, it assigns $t_j = t_j^{mod}$ and $F(t_j) = F(t_j^{mod})$. Otherwise, it discards t_j^{mod} .

If the number of faults detected by t_j was increased, the procedure performs another pass over all the scan chains and all the scan vectors in S_i . The final test t_j is added to T_i , and the faults it detects are removed from F_{sa} and F_{gehx} .

After considering every test $t_j \in T_{cand}$, T_i detects all the faults from $F_{sa} \cup F_{gehx}$ that T_{cand} detects, and possibly additional faults from F_{gehx} .

D. Procedure 2

Procedure 2 adds to T_i a limited number of tests based on T_{gehx} . Its goal in selecting which tests will be added is to detect as many additional gate-exhaustive faults as possible using only scan vectors that are already included in S_i , or require the addition of as few new scan vectors to S_i as possible. It stops after a limited increase in the gate-exhaustive fault efficiency is achieved to avoid a large increase in the storage requirements in a single iteration. The number of tests from T_{gehx} that the procedure uses depends on the circuit and the iteration. The tests are modified as described below to ensure that as few new scan vectors as possible are added to S_i .

Procedure 2 is applied iteratively until it achieves its goal. In each pass of Procedure 2, it considers every test $t_j \in T_{gehx}$. When it considers t_j , it first performs fault simulation of F_{gehx} under t_j . If any faults are detected, the procedure continues with t_j as follows.

It assigns $t_j^{new} = t_j$, and includes the faults detected by t_j in a set $F(t_j^{new})$. It also assigns $j_k^{new} = -1$ for $0 \leq k < n$ to indicate that the scan vectors of t_j^{new} may not be included in S_i . It then considers every scan chain $0 \leq k < n$, and every scan vector $s_p \in S_i$. The procedure defines a test t_j^{mod} that is equal to t_j^{new} , except that $t_{j,k}^{mod} = s_p$ and $j_k^{mod} = p$. It simulates the faults in $F(t_j)$ under t_j^{mod} . If all the faults are detected, it also simulates the faults in F_{gehx} under t_j . Let the subset of detected faults be $F(t_j^{mod})$. The procedure accepts the modification of t_j^{new} if $F(t_j^{new}) \subseteq F(t_j^{mod})$. In this case, it assigns $t_j^{new} = t_j^{mod}$ and $F(t_j^{new}) = F(t_j^{mod})$. Otherwise, it discards t_j^{mod} .

If the procedure modified $j_k^{new} = -1$ into $j_k^{new} \geq 0$ for at least one scan chain k , it performs another pass over all the scan chains with $j_k^{new} = -1$. When modification of t_j^{new} ends, let $n(t_j^{new})$ be the number of detected gate-exhaustive faults, and $m(t_j^{new})$ the number of scan chains k for which $j_k^{new} = -1$.

If $m(t_j^{new}) = 0$, t_j^{new} uses only scan vectors from S_i , and detects $n(t_j^{new})$ gate-exhaustive faults. In this case, the procedure adds t_j^{new} to T_i , and removes the faults it detects from F_{gexh} . Otherwise, the procedure defers the decision on t_j^{new} to the end of the pass.

At the end of the pass, if no test with $m(t_j^{new}) = 0$ was added to T_i , the procedure selects the test t_j^{new} with the smallest value of $m(t_j^{new})$, and the largest value of $n(t_j^{new})$. It adds to S_i all the new scan vectors used by t_j^{new} , i.e., $S_i = S_i \cup \{t_{j,k}^{new} : 0 \leq k < n, j_k = -1\}$. It also adds t_j^{new} to T_i .

After adding at least one new scan vector to S_i , Procedure 2 terminates if the gate-exhaustive fault efficiency is increased by a parameter denoted by Δ_{gexh} . This parameter is needed to ensure that the gate-exhaustive fault efficiency increases substantially with every iteration. Procedure 2 also terminates when all the faults in F_{gexh} are detected. Before using T_i in iteration $i + 1$, the procedure performs forward-looking reverse order fault simulation to remove unnecessary tests from T_i .

IV. EXPERIMENTAL RESULTS

The software procedure for computing the sets S_i and T_i was applied to benchmark circuits.

The following parameter values were used. For a circuit with K flip-flops, the flip-flops were partitioned into n scan chains such that $n^2 \geq K$. If necessary, $n^2 - K$ flip-flops were added for padding. The length of a scan chain was $l = n$. This yields a large number of short scan chains. Other configurations with the same property can be used instead.

The required increase in the gate-exhaustive fault efficiency for every iteration, Δ_{gexh} , was set as follows. When the gate-exhaustive fault efficiency after the first application of Procedure 1 is at least 95%, $\Delta_{gexh} = 0.2\%$. Otherwise, $\Delta_{gexh} = 1\%$. The circuit name is followed by ".1" in this case. These values prevent the procedure from performing an excessive number of iterations.

When the fault efficiency after the first application of Procedure 1 is lower than 95%, Procedure 2 selects a deterministic test after considering ten tests from T_{gexh} that detect new faults. This is important for limiting the computational effort of the procedure.

To define gate-exhaustive faults, the circuit was partitioned into two-level subcircuits with at most 10 inputs. Each subcircuit was used as a gate, and all its gate-exhaustive faults were added to F_{gexh} . Test generation was carried out to produce the test set T_{gexh} . Undetectable faults were eliminated from F_{gexh} .

The number of random tests was $R = 4000|T_{sa}|$. This number was selected based on experimental results indicating that a small number of deterministic tests is typically needed for complementing this number of random tests.

The results for iteration i of the procedure are reported before Procedure 2 adds new scan vectors to S_i . Up to this point, the procedure utilizes the scan vectors in S_i for both random and deterministic tests (new scan vectors that are added to S_i are utilized for random tests only in iteration $i + 1$).

Referring to Figure 1, not every iteration yields improved results compared with later iterations. The results are reported only for iterations with improved results.

The results are reported in Table II as follows. For most of the circuits in Table II, there is a row for every iteration that yields an improved solution until 100% fault efficiency is reached for gate-exhaustive faults. These circuits are arranged by increasing number

TABLE II
EXPERIMENTAL RESULTS

circuit	K	n	i	dtrm	vect	tot	bits tot/gexh	gexh f.e.	ntime
s35932	1764	42	0	0	444	18648	0.392	100.000	9505.74
b04	81	9	1	1	30	315	0.017	100.000	1394.00
des_area	400	20	0	0	431	8620	0.057	100.000	1659.84
sasc	144	12	0	0	112	1344	0.089	100.000	1023.12
simple_spi	169	13	1	0	47	611	0.015	100.000	8612.06
spi	289	17	4	1	241	4233	0.008	100.000	63387.09
systemcdes	324	18	0	0	224	4032	0.125	100.000	1753.96
usb_phy	121	11	0	0	43	473	0.049	100.000	484.20
i2c	169	13	1	0	60	780	0.022	99.339	3217.24
i2c	169	13	3	2	57	897	0.025	99.807	7068.20
i2c	169	13	4	5	60	1170	0.032	100.000	8998.16
tv80	400	20	1	128	260	28240	0.049	99.574	12073.79
tv80	400	20	2	129	262	28460	0.050	99.778	24388.65
tv80	400	20	4	134	287	29860	0.052	100.000	33120.68
b15	484	22	1	1503	221	269390	0.261	99.539	52786.41
b15	484	22	2	1626	214	290884	0.282	99.705	76645.83
b15	484	22	3	1768	212	315832	0.306	99.932	115207.67
b15	484	22	4	1802	206	321684	0.312	100.000	146831.08
wb_dma	784	28	1	14	209	8988	0.039	99.455	11906.35
wb_dma	784	28	2	17	229	10220	0.044	99.660	19847.86
wb_dma	784	28	3	17	257	11480	0.049	99.865	25788.47
wb_dma	784	28	4	22	266	12992	0.056	100.000	30586.44
s5378	225	15	1	0	159	2385	0.028	98.833	3729.10
s5378	225	15	3	0	171	2565	0.030	99.238	9174.20
s5378	225	15	4	0	179	2685	0.031	99.440	10663.98
s5378	225	15	5	0	183	2745	0.032	99.642	13204.55
s5378	225	15	7	0	193	2895	0.034	100.000	16628.05
s9234	256	16	0	145	303	25728	0.117	98.493	2576.53
s9234	256	16	1	213	232	30976	0.141	99.181	8841.67
s9234	256	16	2	234	222	33504	0.152	99.632	14736.02
s9234	256	16	3	262	224	37120	0.169	99.941	20309.77
s9234	256	16	4	264	221	37328	0.170	100.000	25101.02
s1423	100	10	1	0	53	530	0.067	98.609	1241.20
s1423	100	10	2	1	53	590	0.075	98.851	1936.00
s1423	100	10	3	1	57	630	0.080	99.093	2473.00
s1423	100	10	4	1	59	650	0.082	99.335	3153.60
s1423	100	10	5	1	63	690	0.087	99.577	3681.20
s1423	100	10	6	1	67	740	0.093	99.819	4205.00
s1423	100	10	7	1	69	760	0.096	100.000	4861.20
s13207.1	729	27	1	0	484	13068	0.026	90.974	26106.04
s13207.1	729	27	2	0	498	13446	0.027	91.992	40675.93
s13207.1	729	27	3	0	522	14094	0.028	93.148	49713.95
s13207.1	729	27	4	0	548	14796	0.030	94.176	62855.36
s13207.1	729	27	5	0	586	15822	0.032	95.232	73768.11
s13207.1	729	27	6	0	611	16497	0.033	96.345	87076.30
s13207.1	729	27	7	0	633	17091	0.034	97.420	96723.46
s13207.1	729	27	8	0	648	17496	0.035	98.444	103565.09
s13207.1	729	27	9	0	714	19278	0.039	99.472	112481.73
s13207.1	729	27	10	0	737	19899	0.040	100.000	122845.27
s15850.1	625	25	0	3	288	7875	0.013	86.801	1509.86
s15850.1	625	25	1	32	349	15925	0.025	88.121	5625.45
s15850.1	625	25	2	60	433	24325	0.039	89.297	10849.58
s15850.1	625	25	3	46	521	24525	0.039	90.465	19150.70
s15850.1	625	25	4	59	607	29925	0.048	91.554	25021.39
s15850.1	625	25	5	75	699	36225	0.058	92.622	31400.64
s15850.1	625	25	7	72	908	40700	0.065	94.663	46743.86
s15850.1	625	25	8	66	1002	41550	0.066	95.694	59461.09
s15850.1	625	25	10	66	1233	48975	0.078	97.706	83019.74
s15850.1	625	25	11	81	1356	56175	0.089	98.709	96543.31
s15850.1	625	25	12	88	1420	59700	0.095	99.733	114753.56
s15850.1	625	25	13	95	1448	62325	0.099	100.000	127918.80
b14.1	289	17	0	216	250	33626	0.045	85.838	7310.89
b14.1	289	17	1	287	218	42738	0.057	86.705	16520.68
b14.1	289	17	2	377	225	55097	0.074	87.892	25899.69
b14.1	289	17	3	477	243	69003	0.093	88.984	36228.38
b14.1	289	17	4	662	278	106012	0.142	90.216	49101.41
b14.1	289	17	5	804	281	127789	0.172	91.122	64176.50
b14.1	289	17	6	991	284	156451	0.210	92.200	77742.39
b14.1	289	17	7	1148	309	180897	0.243	93.202	92378.23
b14.1	289	17	8	1287	341	202708	0.272	94.152	109924.09
b14.1	289	17	9	1453	346	228191	0.306	95.172	126674.31
b14.1	289	17	10	1602	349	251039	0.337	96.212	148285.70
b14.1	289	17	11	1793	366	280551	0.377	97.381	170142.53
b14.1	289	17	12	1928	377	301393	0.405	98.318	190629.62
b14.1	289	17	13	2057	398	321487	0.432	99.242	219983.75
b14.1	289	17	14	2181	414	340731	0.457	100.000	240957.95
s38417	1681	41	0	53	1393	81016	0.029	96.530	20061.56
s38584	1521	39	0	36	577	36543	0.043	99.509	5836.27

of iterations that produce improved solutions. Additional circuits are considered with a single iteration to demonstrate the results achievable for them.

After the circuit name, column K shows the number of flip-flops. Column n shows the number of scan chains. This is also the number of flip-flops in a scan chain.

Column i shows the iteration. Column $dtrm$ shows the number of deterministic tests in $T_{i,dtrm}$. Column $vect$ shows the number of scan vectors in S_i .

Column $bits$ shows the number of storage bits required for S_i and $T_{i,dtrm}$ (subcolumn tot), and the number of storage bits divided by the number of bits required for storing T_{gexh} (subcolumn $tot/gexh$).

The stuck-at fault coverage of T_i is always equal to the fault coverage of T_{sa} , which detects all the detectable stuck-at faults. Column $gexh\ f.e.$ shows the gate-exhaustive fault efficiency achieved by T_i . Column $ntime$ shows the normalized runtime of the software procedure, which is the runtime divided by the runtime for fault simulation with fault dropping of F_{sa} under T_{sa} , and F_{gexh} under T_{gexh} . Since the software procedure is based on repeated fault simulation, normalizing its runtime to fault simulation time provides an indication of its computational effort.

The following points can be seen from Table II. The on-chip test generation logic is able to achieve 100% gate-exhaustive fault efficiency. This is a benefit of storing test data and allowing deterministic tests to be applied.

The number of on-chip stored bits is a small fraction of the number of bits in a deterministic gate-exhaustive test set. This is made possible by the on-chip generation of random tests.

Figure 1 is based on $s1423$ and demonstrates the tradeoff between the number of storage bits and the fault efficiency obtained for gate-exhaustive faults. A similar tradeoff exists for other circuits in Table II. There is only a small number of circuits for which the best solution is obtained in iteration 0, and additional iterations are not required.

The normalized runtime is that of the software procedure. The procedure is based on fault simulation of large numbers of tests to select the best stored data for the on-chip test generation logic. The normalized runtime is similar for circuits of different sizes, indicating that the procedure scales similar to a fault simulation procedure.

The number of deterministic tests in Table II is typically small, implying that the number of storage bits for deterministic tests is kept low. When it is zero, the logic in the upper part of Figure 2 consists only of an $LFSR$. The number of deterministic tests can be reduced further by increasing the number of random tests applied based on S_i .

The random tests produced by the on-chip test generation logic are different from the conventional random tests in that they are formed by random combinations of stored scan vectors, which are derived from a deterministic test set. The more effective the random tests are for a circuit, the lower the sizes of S_i and $T_{i,dtrm}$. This is independent of other parameters such as the size of the circuit.

V. CONCLUDING REMARKS

This paper described a *BIST* approach that stores test data on-chip, and uses the stored test data to generate both random and deterministic tests on-chip. This approach offers a tradeoff between the amount of stored test data and the comprehensiveness of the test set that can be applied. The paper explored this tradeoff in a specific context where the circuit under consideration has a large number of short scan chains, allowing storage of scan vectors. The initial stored test data is based on a stuck-at test set, but the set of target faults includes single-cycle gate-exhaustive faults. Under the approach described in the paper, the stored test data is enhanced gradually by test data from a gate-exhaustive test set, and the coverage

of gate-exhaustive faults is increased gradually. Experimental results demonstrated this tradeoff for benchmark circuits.

REFERENCES

- [1] P. H. Bardell, W. H. McAnney and J. Savir, *Built-In Test for VLSI Pseudorandom Techniques*, Wiley Interscience, 1987.
- [2] 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. on Computers*, Nov. 2002, pp. 1282-1993.
- [3] S. Pateras, "Security vs. Test Quality: Fully Embedded Test Approaches are the Key to Having Both", *Intl. Test Conf.*, 2004, Panel P2.2, p. 1413.
- [4] 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.
- [5] 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.
- [6] F. Reimann, M. Glas, J. Teich, A. Cook, L. Rodriguez Gmez, 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.
- [7] S. U. Hussain, S. Yellapantula, M. Majzoobi and F. Koushanfar, "BIST-PUF: Online, Hardware-Based Evaluation of Physically Unclonable Circuit Identifiers", in *Proc. Intl. Conf. on Computer-Aided Design*, 2014, pp. 162-169.
- [8] R. Wang, K. Chakrabarty and S. Bhawmik, "Built-In Self-Test and Test Scheduling for Interposer-Based 2.5D IC", *ACM Trans. on Design Automation*, Vol. 20, No. 4, Sep. 2015, Art. 58.
- [9] D. Xiang, X. Wen and L. Wang, "Low-Power Scan-Based Built-In Self-Test Based on Weighted Pseudorandom Test Pattern Generation and Reseeding", in *IEEE Trans. on VLSI Systems*, March 2017, Vol. 25, No. 3, pp. 942-953.
- [10] O. E. Erol and S. Ozev, "Knowledge- and Simulation-Based Synthesis of Area-Efficient Passive Loop Filter Incremental Zoom-ADC for Built-In Self-Test Applications", *ACM Trans. on Design Automation*, Vol. 24, No. 1, Jan. 2019, Art. 3.
- [11] A. H. Baba and S. Mitra, "Testing for Transistor Aging", in *Proc. VLSI Test Symposium*, 2009, pp. 215-220.
- [12] Y. Sato, "Circuit Failure Prediction by Field Test - A New Task of Testing", in *Proc. Intl. Symp. on Defect and Fault Tolerance in VLSI Systems*, 2010, pp. 69-70.
- [13] S. Jin, Y. Han, H. Li and X. Li, "Unified Capture Scheme for Small Delay Defect Detection and Aging Prediction", *IEEE Trans. on VLSI Systems*, May 2013, Vol. 21, No. 5, pp. 821-833.
- [14] A. Sivadasan, R. J. Shah, V. Huard, F. Cacho and L. Anghel, "NBTI Aged Cell Rejuvenation with Back Biasing and Resulting Critical Path Reordering for Digital Circuits in 28nm FDSOI", in *Proc. Design, Automation & Test in Europe Conf.*, 2018, pp. 997-998.
- [15] A. Jas, C. V. Krishna and N. A. Touba, "Weighted Pseudorandom Hybrid BIST", *IEEE Trans. on VLSI Systems*, Dec. 2004, Vol. 12, No. 12, pp. 1277-1283.
- [16] S. Wang, K. J. Balakrishnan and S. T. Chakradhar, "XWRC: Externally-Loaded Weighted Random Pattern Testing for Input Test Data Compression", *Intl. Test Conf.*, 2005, Paper 24.2, pp. 1-10.
- [17] M. Filipek, G. Mrugalski, N. Mukherjee, B. Nadeau-Dostie, J. Rajski, J. Solecki and J. Tyszer, "Low-Power Programmable PRPG With Test Compression Capabilities", *IEEE Trans. on VLSI Systems*, June 2015, Vol. 23, No. 6, pp. 1063-1076.
- [18] 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.
- [19] E. J. McCluskey, "Quality and Single-stuck Faults", in *Proc. Intl. Test Conf.*, 1993, p. 597.