Storage-Based Logic Built-In Self-Test with Variable-Length Test Data

Irith Pomeranz
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907, U.S.A.
E-mail: pomeranz@ecn.purdue.edu

***Abstract*—Storage-based logic built-in self-test ($LBIST$) approaches store deterministic test data on-chip, and use them for test application. The applied tests are closer to deterministic tests than the pseudo-random tests that are typically produced by $LBIST$. In earlier approaches, the stored test data consisted of scan vectors of equal length. This article describes a storage-based $LBIST$ approach where the stored test data have variable length. Instead of storing a scan vector directly, the approach described in this article stores a sequence that, when repeated, produces a scan vector. The use of variable-length sequences that are shorter than scan vectors reduces the storage requirements. The article describes a software procedure for computing a set of variable-length sequences for a set of target faults. Experimental results are presented for single stuck-at and single-cycle gate-exhaustive faults in benchmark circuits to demonstrate the discussion.**

I. Introduction

Logic built-in self-test ($LBIST$) is typically based on the use of pseudo-random test data that can be produced efficiently on-chip [1]-[17]. The use of $LBIST$ enhances security by avoiding the transfer of test data to and from a chip [5]. Another important advantage of $LBIST$ is that it allows in-field testing to be carried out [18]-[22].

However, pseudo-random test data are limited in the fault coverage they can achieve. This is typically addressed by the insertion of test-points. Storage-based $LBIST$ approaches address this issue by storing deterministic test data on-chip, and using them for the application of tests that are closer to deterministic tests [3], [16], [17]. Such tests provide better fault coverage, and require fewer or no test-points. In particular, the tests can be tailored to more complex defect behaviors and aging effects that need to be addressed by in-field testing.

With storage-based $LBIST$ it is important to limit the amount of deterministic test data stored on-chip. For this purpose, it is important to reuse the same test data for the application of different tests. The approach described in [3] stores two types of test data: (1) for every scan chain it stores a set of scan vectors from a deterministic test set, and (2) it stores combinations of indices of scan vectors to use for forming tests. A combination includes one index for every scan chain, and corresponds to a test. Without storing which combinations of indices to use, the number of applied tests will be excessive. Therefore, storage of indices is necessary under this approach.

The approach described in [16] stores scan vectors from a deterministic test set as a single set. It applies two types of tests: (1) tests referred to as random are formed by using scan vectors that are pseudo-randomly selected by $LFSR$s, and (2) tests referred to as deterministic are formed using stored indices of scan vectors, similar to [3]. The number of combinations of indices that need to be stored is reduced by the use of pseudo-random combinations that do not require storage of indices.

In [3], [16] and [17], the stored test data consist of scan vectors of equal length. This article describes a storage-based $LBIST$ approach where the stored test data have variable length. Instead of storing a scan vector directly, as in [3], [16] and [17], the approach described in this article stores a sequence that, when repeated, produces a scan vector. For example, the sequence 011 produces the scan vector 01101101 for a scan chain of length eight when it is repeated to produce eight bits.

The use of variable-length sequences as suggested in this article reduces the storage requirements since entire scan vectors do not need to be stored, and the stored sequences can be significantly shorter than scan vectors. This is illustrated by Figure 1, where the dashed box shows a memory for storing N scan vectors of length L. Inside the dashed box are N stored sequences that are shorter than scan vectors. Figure 1 is explained in more detail later.

The article describes a software procedure for computing a set of stored sequences for a set of target faults. The procedure accepts the following information produced by the procedure from [16]. (1) A set of scan vectors S. For a circuit where the length of the longest scan chain is L, all the scan vectors in S are L-bit vectors. (2) A number of tests R where a test is obtained by a pseudo-random combination of scan vectors from S. The corresponding set of tests is denoted by T_{rand}. (3) A set of tests T_{stor} where a test is described by a combination of indices of scan vectors from S. These combinations need to be stored on-chip in addition to S.

The software procedure described in this article replaces the

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

978-1-6654-5938-9/22/$31.00 ©2022 IEEE

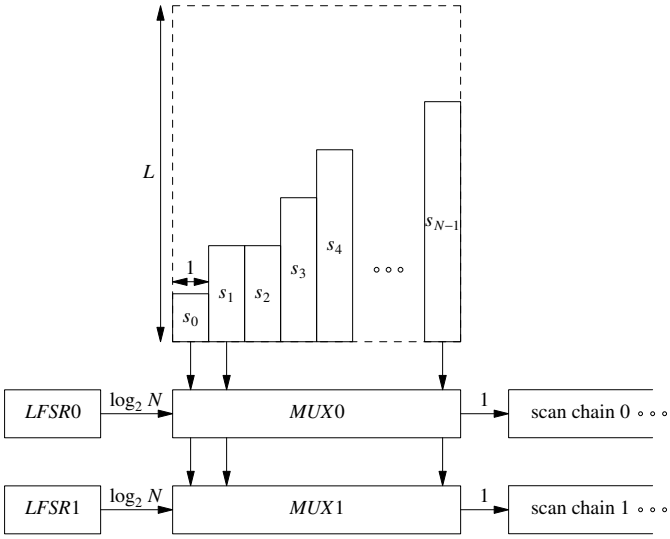

Fig. 1. On-chip test generation logic.

scan vectors in S with sequences whose lengths are bounded by L. The storage requirements are reduced in two ways. (1) When a scan vector of length L is replaced with a sequence of length $l < L$, the storage requirements are reduced by $L - l$ bits. (2) A by-product of the replacement of the scan vectors with shorter sequences is that some of the tests in T_{stor} become unnecessary. When fewer combinations of indices need to be stored, the storage requirements are reduced.

The software procedure described in this article limits the search space for the set S in several ways. (1) Except for sequences of lengths one, two and three, where all the possible sequences are considered, longer sequences are obtained by truncating scan vectors from S. Other options can be considered by the same procedure. (2) The procedure does not attempt to reduce the number of sequences in S relative to the number of scan vectors initially included in it. The reduction in storage requirements is achieved only by reducing the lengths of the sequences, and eliminating stored combinations of sequence indices that become unnecessary. (3) The procedure does not attempt to compute new combinations of sequences that yield effective tests to add to the ones in T_{stor}.

Even with these restrictions, the procedure reduces the storage requirements for benchmark circuits significantly relative to the fixed-length solution of [16].

The article is organized as follows. Section II describes the on-chip test generation logic and the set of tests it applies to the circuit. Section III describes the software procedure for computing the set S of stored sequences. Section IV presents experimental results for benchmark circuits.

II. On-Chip Test Generation Logic

The on-chip test generation logic for storage-based $LBIST$ with variable-length sequences is illustrated by Figure 1. For simplicity, Figure 1 shows only the logic needed for applying tests using pseudo-random combinations of sequences. This case is described first.

TABLE I
Finite-State Machine

	NS, Z			
PS	$a_0a_1 =$ 00	$a_0a_1 =$ 01	$a_0a_1 =$ 10	$a_0a_1 =$ 11
P_0	$P_0, 0$	$P_1, 0$	$P_1, 1$	$P_1, 1$
P_1	X, X	$P_0, 1$	$P_2, 0$	$P_2, 1$
P_2	X, X	X, X	$P_0, 1$	$P_3, 0$
P_3	X, X	X, X	X, X	$P_0, 1$

The circuit in Figure 1 has K scan chains, two of which are shown in the figure. The maximum length of a scan chain is denoted by L. For simplicity of discussion, all the scan chains are assumed to be padded such that their length is exactly L. Padding is not needed in the implementation of the circuit.

The on-chip test generation logic consists of a memory that holds a set $S = \{s_0, s_1, ..., s_{N-1}\}$ of variable-length sequences. The length of a sequence $s_i \in S$ is denoted by $l_i \leq L$. The dashed box in Figure 1 shows the memory needed if L-bit scan vectors are stored instead of shorter sequences as suggested in this article.

In addition, for scan chain $0 \leq k < K$, a linear-feedback shift-register referred to as $LFSRk$ provides the index of a sequence from S. A multiplexer referred to as $MUXk$ selects the sequence to be shifted into scan chain k.

A test t_i consists of K scan vectors, $t_i = \langle t_{i,0}, t_{i,1}, ..., t_{i,K-1} \rangle$. The vector $t_{i,k}$ is an L-bit scan vector for scan chain k, where $0 \leq k < K$. During on-chip test generation, suppose that $LFSRk$ holds the index of $s_{i,k}$. In this case, the scan vector $t_{i,k}$ of the applied test t_i is determined by $s_{i,k}$. Specifically, a counter cycles through $s_{i,k}$ to produce L bits for scan chain k.

A finite-state machine can be used for implementing the memory from Figure 1, and the counters needed for test application. Such a finite-state machine is shown in Table I for the case where $N = 4$, $s_0 = 0$, $s_1 = 01$, $s_2 = 101$ and $s_3 = 1101$. In Table I, PS stands for the present-state of the finite-state machine, NS stands for its next-state, and Z stands for its primary output vector. An X is used for an entry (next-state or primary output vector) that can be specified arbitrarily. The inputs a_0 and a_1 in Table I represent the values assigned by the $LFSR$s to the multiplexer select inputs. With $L = 8$, the scan vectors produced for $a_0a_1 = 00$, 01, 10 and 11 are $v_0 = 00000000$, $v_1 = 01010101$, $v_2 = 10110110$ and $v_3 = 11011101$, respectively.

The number of tests formed by the on-chip test generation logic using pseudo-random combinations of sequences from S is denoted by R. The set of R tests formed based on S is denoted by T_{rand}. The set T_{rand} is not stored on-chip. The $LFSR$s produce pseudo-random indices of sequences, and the sequences are used for forming tests.

To apply a set T_{stor} of tests formed using specific indices of sequences, a memory for storing the indices needs to be added to Figure 1. An additional multiplexer for every scan chain needs to select between the $LFSR$ and the memory holding T_{stor}. In this case, application of R tests formed by pseudo-randomly selecting sequences is followed by application of

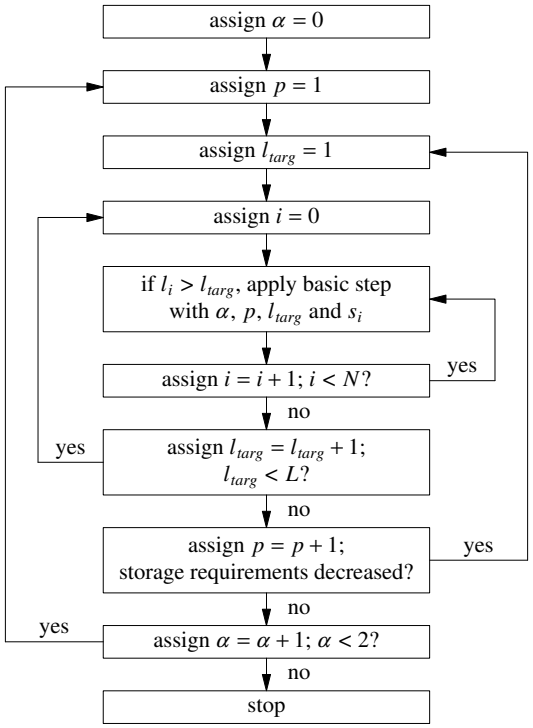

Fig. 2. Software procedure.

tests formed from indices stored in T_{stor}.

III. Software Procedure for Sequence Computation

The software procedure for computing the set S of variable-length sequences is described in this section. The overall structure of the procedure is illustrated by Figure 2. A basic step of the procedure is illustrated by Figure 3.

A. Overall Structure of the Procedure

The procedure uses the solution from [16] as an initial solution. The initial solution consists of a set of scan vectors $S^{init} = \{s_0, s_1, ..., s_{N-1}\}$, a set T^{init}_{rand} of R tests obtained from pseudo-random combinations of scan vectors, and a set T^{init}_{stor} of tests obtained from stored combinations of scan vectors. For $0 \leq i < N$, a scan vector $s_i \in S^{init}$ is considered as a sequence of length $l_i = L$.

In general, the software procedure results in $l_i \leq L$. When $l_i \leq L$, a scan vector is obtained from s_i by repeating s_i to produce L bits. The resulting scan vector is denoted by v_i. For the initial solution, $v_i = s_i$ for $0 \leq i < N$.

The set of target faults is denoted by F. Fault simulation with fault dropping of F under T^{init}_{rand} and T^{init}_{stor} yields a set of detected faults denoted by D.

During the software procedure, the set of sequences is S, and the sets of tests applied to the circuit are T_{rand} and T_{stor}. Initially, $S = S^{init}$, $T_{rand} = T^{init}_{rand}$, and $T_{stor} = T^{init}_{stor}$. The goal of the software procedure is to maintain the detection of the faults in D while reducing the lengths of the sequences in S to reduce the storage requirements. As S evolves, the set T_{rand} is recomputed to include R tests formed by pseudo-random combinations of sequences. The combinations are the same as in T^{init}_{rand}, since the same on-chip test generation logic is used for producing the indices. However, different tests are formed when S is modified relative to S^{init}, and different tests may be effective in detecting faults from D.

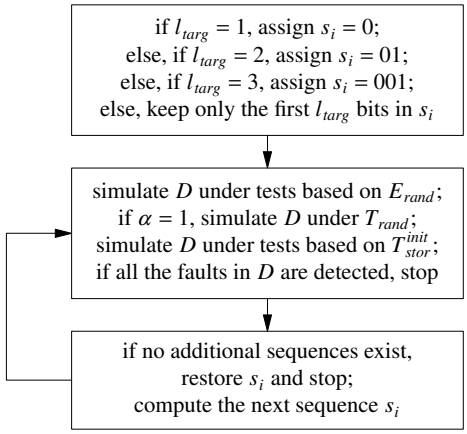

Fig. 3. Basic step.

The set T_{stor} also uses the same stored combinations of indices as T^{init}_{stor}, but the tests are different when S changes. Unnecessary tests are removed from T_{stor} to further reduce the storage requirements.

To reduce the lengths of the sequences in S, the procedure considers the sequences in S iteratively. An iteration is described by three parameters, $\alpha = 0$ or 1, $p = 1, 2, ...,$ and $l_{targ} = 1, 2, ... , L - 1$. With given values of α, p and l_{targ}, the procedure considers every sequence $s_i \in S$ such that $l_i > l_{targ}$. The procedure attempts to replace s_i with a sequence of length l_{targ}. By increasing l_{targ} gradually, the procedure gives a higher priority to shorter sequences.

As p is increased, the procedure iterates with $1 \leq l_{targ} < L$ as long as it can reduce the storage requirements.

The parameter α is introduced to control the computational effort as discussed later. Until then, the description corresponds to $\alpha = 1$, and α is not mentioned in the description of the procedure.

The storage requirements are determined by S and T_{stor}. The number of bits required for S is $\sum\{l_i : 0 \leq i < N\}$. A test in T_{stor} requires K indices of sequences to be stored, one for every scan chain. An index requires $log_2(N)$ bits. The total number of bits required for T_{stor} is $|T_{stor}|Klog_2(N)$. The total number of bits is $B = \sum\{l_i : 0 \leq i < N\} + |T_{stor}|Klog_2(N)$. For the initial solution with S^{init} and T^{init}_{stor}, the storage requirements are denoted by B^{init}.

B. Basic Step

A basic step of the procedure is applied to a sequence $s_i \in S$ in iteration $p \geq 1$ with a target length $l_{targ} < l_i$.

In general, there are $2^{l_{targ}}$ sequences of length l_{targ}, and any one of them can replace s_i. For small values of l_{targ} it is possible to consider all the sequences that yield different scan

TABLE II
REDUCED SEQUENCES

p	l_{targ}	s_i	v_i
0	0	01011101	01011101
1	1	0	00000000
1	2	01	01010101
1	3	010	01001001
1	4	0101	01010101
1	5	01011	01011010
1	6	010111	01011101
1	7	0101110	01011100

TABLE III
TRANSLATING TESTS WITH STORED INDICES

k	$s_{i,k}$	$t_{i,k}$
0	01011101	01011101
1	11110100	11110100
0	010	01001001
1	11110100	11110100

vectors. This option is discussed later. To allow large values of l_{targ} to be considered, the sequence s_i is modified into a sequence of length l_{targ} by keeping only its first l_{targ} bits. Only one option is considered in this case.

Table II shows an example for a circuit with $L = 8$. The initial scan vector s_i is shown in the row with $p = 0$ and $l_{targ} = 0$. With $p = 1$ and $l_{targ} = 1$, $s_i = 0$. The corresponding scan vector is $v_i = 00000000$.

If this sequence is not accepted, s_i will be considered with $l_{targ} = 2$. In this case, $s_i = 01$, yielding the scan vector $v_i = 01010101$.

For $3 \leq l_{targ} \leq 7$, s_i is truncated as shown in Table II to obtain a sequence of length l_{targ} with the corresponding scan vector v_i.

It is interesting to note that $s_i = 0101$ obtained for $l_{targ} = 4$ yields the same scan vector as $s_i = 01$ obtained for $l_{targ} = 2$. It is possible to check for this occurrence and avoid considering $l_{targ} = 4$ in this case. It is also interesting to note that $l_{targ} = 6$ yields the initial scan vector, and larger values of l_{targ} do not need to be considered.

Every option for s_i obtained with p and l_{targ} is evaluated as follows. The procedure simulates the faults in D under the tests in T_{rand} and T_{stor}. The set T_{rand} is formed by pseudo-random combinations of sequences from S, using the modified set S. As mentioned earlier, T_{rand} uses the same combinations of indices as T^{init}_{rand}, but the tests are different since the sequences in S are different.

To construct T_{stor}, every test $t_i \in T^{init}_{stor}$ is translated into a test that uses the same stored indices, but with the modified set S.

An example is shown in Table III. The circuit has $K = 2$ scan chains of length $L = 8$. The test $t_i = \langle t_{i,0}, t_{i,1} \rangle \in T^{init}_{stor}$ is shown in the first two rows of Table III. The sequences $s_{i,0}$ and $s_{i,1}$ in the first two rows are the 8-bit scan vectors from S^{init}. Suppose that $s_{i,0}$ is replaced with 010, replacing the scan vector $v_{i,0}$ with 01001001. The resulting test is the one shown in the next two rows of Table III.

If the modified test t_i does not detect any faults from D, it is not added to T_{stor}. This allows the number of tests in T_{stor} to be lower than that in T^{init}_{stor}.

The procedure accepts the replacement of s_i if T_{rand} and T_{stor} together detect all the faults in D, and the number of bits B is reduced. Otherwise, s_i is restored to its earlier value.

C. Computational Effort

To measure the computational effort of the procedure, the runtime required for simulating the initial solution is denoted by ρ_0. This includes the simulation of R tests from T^{init}_{rand} that are formed using pseudo-randomly selected indices of scan vectors, followed by tests from T^{init}_{stor} that are formed using stored indices of scan vectors.

A basic step requires fault simulation of T_{rand} and T_{stor}, which has a runtime that is similar to ρ_0. For $1 \leq l_{targ} < L$, the procedure performs at most one basic step for every one of N sequences in S. Considering all the values of l_{targ} and all the sequences in S, the procedure performs at most $N(L-1)$ basic steps. With p iterations, the number of basic steps is $pN(L-1)$. Accordingly, the runtime is expected to be $pN(L-1)\rho_0$. The normalized runtime is defined as the runtime divided by ρ_0, or $pN(L-1)$.

To speed up the procedure, two experimental observations are used as discussed next.

The first observation is that there is a small subset of faults that typically remain undetected in basic steps where not all the faults from D are detected. Thus, a small subset of faults prevents the replacement of sequences. If these faults are simulated earlier, and remain undetected, other faults do not need to be simulated in the same basic step.

The second observation used for reducing the computational effort is that the same pseudo-random combinations of sequences are typically effective in detecting target faults, even when the sequences in S change. To take advantage of this observation, pseudo-random combinations from T^{init}_{rand} that contribute to fault detection are stored in a set denoted by E_{rand}. The set E_{rand} is computed in a preprocessing step by performing eight-detection fault simulation of T^{init}_{rand}, and storing in E_{rand} the pseudo-random combinations that contribute fault detections. During the procedure, before simulating all the tests in T_{rand}, the combinations stored in E_{rand} are used for forming tests, and these tests are simulated first.

The procedure uses E_{rand} in two ways. When $\alpha = 0$, only tests based on E_{rand} are simulated. When $\alpha = 1$, tests based on E_{rand} are simulated first, followed by all the tests in T_{rand}. The procedure is run first with $\alpha = 0$, and then with $\alpha = 1$. With every value of α, the procedure iterates using $p = 1$, 2, ..., and $1 \leq l_{targ} < L$.

With the reduced computational effort obtained for $\alpha = 0$, it is possible to consider more than one option for reducing the length of a sequence. The procedure takes advantage of this possibility to consider all the possible sequences for $1 \leq l_{targ} \leq 3$. Of all the options, the procedure selects the first one that allows all the faults in D to be detected.

IV. EXPERIMENTAL RESULTS

The results of the software procedure are presented in this section. The setup is as in [16] to allow the solution from [16],

TABLE IV
EXPERIMENTAL RESULTS

circuit	inp	K	α	p	S	ave	stor	bits	ratio	s.a.	g.exh	ntime
b14	289	17	0	0	414	17.00	2181	340731	1.000	95.421	100.000	1.00
b14	289	17	0	2	414	14.95	2175	338964	0.995	95.421	100.000	13.45
b14	289	17	1	1	414	14.92	2167	337727	0.991	95.421	100.000	536.80
s5378	225	15	0	0	193	15.00	0	2895	1.000	99.131	100.000	1.00
s5378	225	15	0	3	193	11.25	0	2171	0.750	99.131	100.000	302.35
s5378	225	15	1	2	193	10.85	0	2095	0.724	99.131	100.000	3600.49
s15850	625	25	0	0	1448	25.00	95	62325	1.000	96.682	100.000	1.00
s15850	625	25	0	3	1448	19.41	87	52030	0.835	96.682	100.000	6294.34
s15850	625	25	1	2	1448	19.08	57	43304	0.695	96.682	100.000	26133.68
s38584	1521	39	0	0	453	39.00	55	36972	1.000	95.852	100.000	1.00
s38584	1521	39	0	4	453	21.52	47	26245	0.710	95.852	100.000	2353.43
s38584	1521	39	1	2	453	20.04	36	21716	0.587	95.852	100.000	11262.38
tv80	400	20	0	0	287	20.00	134	29860	1.000	99.751	100.000	1.00
tv80	400	20	0	3	287	14.29	112	24262	0.813	99.763	100.000	218.58
tv80	400	20	1	3	287	12.89	69	16120	0.540	99.776	100.000	4520.03
s1423	100	10	0	0	70	10.00	1	770	1.000	99.076	100.000	1.00
s1423	100	10	0	2	70	6.43	1	520	0.675	99.076	100.000	31.54
s1423	100	10	1	1	70	6.06	0	424	0.551	99.076	100.000	420.30
s13207	729	27	0	0	737	27.00	0	19899	1.000	98.462	100.000	1.00
s13207	729	27	0	4	737	15.82	0	11659	0.586	98.462	100.000	5080.82
s13207	729	27	1	2	737	14.86	0	10953	0.550	98.462	100.000	26276.92
simple_spi	169	13	0	0	47	13.00	0	611	1.000	100.000	100.000	1.00
simple_spi	169	13	0	2	47	8.02	0	377	0.617	100.000	100.000	34.43
simple_spi	169	13	1	2	47	5.85	0	275	0.450	100.000	100.000	507.99
b07	64	8	0	0	30	8.00	3	360	1.000	99.915	100.000	1.00
b07	64	8	0	2	30	5.07	2	232	0.644	99.915	100.000	13.10
b07	64	8	1	3	30	3.97	1	159	0.442	99.915	100.000	253.08
wb_dma	784	28	0	0	266	28.00	22	12992	1.000	100.000	100.000	1.00
wb_dma	784	28	0	3	266	18.30	17	9152	0.704	100.000	100.000	491.09
wb_dma	784	28	1	2	266	16.65	5	5689	0.438	100.000	100.000	5087.86
i2c	169	13	0	0	60	13.00	5	1170	1.000	100.000	100.000	1.00
i2c	169	13	0	3	60	8.50	1	588	0.503	100.000	100.000	79.36
i2c	169	13	1	1	60	8.10	0	486	0.415	100.000	100.000	492.37
b04	81	9	0	0	30	9.00	1	315	1.000	99.851	100.000	1.00
b04	81	9	0	2	30	5.60	0	168	0.533	99.851	100.000	8.83
b04	81	9	1	2	30	3.57	0	107	0.340	99.851	100.000	163.58
spi	289	17	0	0	241	17.00	1	4233	1.000	99.985	100.000	1.00
spi	289	17	0	4	241	9.05	0	2181	0.515	99.985	100.000	229.99
spi	289	17	1	3	241	4.74	0	1143	0.270	99.985	100.000	2493.95
usb_phy	121	11	0	0	43	11.00	0	473	1.000	100.000	100.000	1.00
usb_phy	121	11	0	2	43	2.91	0	125	0.264	100.000	100.000	10.98
usb_phy	121	11	1	1	43	1.95	0	84	0.178	100.000	100.000	96.97
sasc	144	12	0	0	112	12.00	0	1344	1.000	100.000	100.000	1.00
sasc	144	12	0	3	112	3.44	0	385	0.286	100.000	100.000	42.58
sasc	144	12	1	2	112	1.89	0	212	0.158	100.000	100.000	283.65
des_area	400	20	0	0	431	20.00	0	8620	1.000	100.000	100.000	1.00
des_area	400	20	0	4	431	6.42	0	2765	0.321	100.000	100.000	175.27
des_area	400	20	1	4	431	2.09	0	902	0.105	100.000	100.000	2943.86
s641	64	8	0	0	27	8.00	24	1176	1.000	100.000	100.000	1.00
s641	64	8	0	2	27	5.41	14	706	0.600	100.000	100.000	11.86
s641	64	8	1	5	27	3.67	0	99	0.084	100.000	100.000	380.28
systemcdes	324	18	0	0	224	18.00	0	4032	1.000	100.000	100.000	1.00
systemcdes	324	18	0	4	224	2.58	0	579	0.144	100.000	100.000	21.43
systemcdes	324	18	1	1	224	1.44	0	323	0.080	100.000	100.000	124.34
s9234	256	16	0	0	221	16.00	264	37328	1.000	93.475	100.000	1.00
s9234	256	16	0	2	221	13.47	241	33824	0.906	93.475	100.000	69.71
s9234	256	16	1	18	221	5.76	0	1272	0.034	93.475	100.000	9512.94
s35932	1764	42	0	0	444	42.00	0	18648	1.000	89.809	100.000	1.00
s35932	1764	42	0	4	444	1.97	0	876	0.047	89.809	100.000	733.79
s35932	1764	42	1	2	444	1.25	0	555	0.030	89.809	100.000	7917.07

with fixed-length scan vectors, to be used as an initial solution for the procedure described in this article.

Specifically, a circuit has K scan chains of length L, with $K = L$, to ensure a large number of short scan chains, for which storage of scan vectors is feasible.

The set of target faults consists of stuck-at faults and detectable single-cycle gate-exhaustive faults. Accordingly, the coverage metric for gate-exhaustive faults is referred to as a fault efficiency.

The $LBIST$ logic follows the implementation illustrated by Figure 1. Its size is determined by the number of storage bits required for S and T_{stor}.

The results are shown in Table IV. The first row for every circuit describes the initial solution from [16]. The second (third) row describes the final solution obtained by the procedure described in this article with $\alpha = 0$ ($\alpha = 1$).

After the circuit name, column inp shows the number of inputs. Column K shows the number of scan chains. Column α shows the value of α. Column p shows the iteration. Column S shows the number of sequences in S. Column ave shows

the average length of a sequence in S. Column $stor$ shows the number of tests in T_{stor}. Column $bits$ shows the number of storage bits required for S and T_{stor}, which is B^{init} for the initial solution from [16], and B for the solution obtained by the procedure described in this article. Column $ratio$ shows the ratio B/B^{init}. The circuits are arranged from high to low value of this ratio. Column $s.a.$ shows the single stuck-at fault coverage. Column $g.exh$ shows the single-cycle gate-exhaustive fault efficiency. Column $ntime$ shows the normalized runtime.

The following points can be seen from Table IV. By using variable-length sequences, the procedure described in this article reduces the average length of a sequence in S such that it is significantly lower than L. As a by-product, it also reduces the number of tests in T_{stor} that require storage of sequence indices. Overall, this reduces the number of bits required for on-chip test generation.

There are large variations in the results for different circuits depending on the ability to use sequences shorter than L for producing scan vectors.

Much of the reduction in the storage requirements is achieved using $\alpha = 0$, with significantly reduced computational effort compared to $\alpha = 1$. In addition, after the sequence lengths are reduced with $\alpha = 0$, fewer options remain to be considered with $\alpha = 1$, and the computational effort for $\alpha = 1$ is reduced. The use of $\alpha = 1$ is important for achieving additional reductions in the storage requirements.

The normalized runtime of the software procedure is significantly smaller than the bound $pN(L-1)$ since complete fault simulation is avoided when $\alpha = 0$, and in many cases when a sequence is unacceptable with both values of α.

In addition, the normalized runtime of the software procedure is similar for circuits of different sizes. This indicates that the procedure scales similar to a fault simulation procedure.

V. Concluding Remarks

This article considered a storage-based logic built-in self-test approach. Earlier approaches stored scan vectors from a deterministic test set, and used combinations of stored scan vectors to form tests on-chip. The main contribution of the article is to suggest that, instead of storing equal-length scan vectors, it is possible to store variable-length sequences from which scan vectors can be obtained. The article described a software procedure for computing a set S of stored sequences. The procedure starts from a solution with fixed-length sequences that are equal to scan vectors. It reduces the lengths of the sequences in an iterative process. Experimental results were presented for benchmark circuits to demonstrate the ability of the procedure to reduce the storage requirements compared with a fixed-length solution considering stuck-at and single-cycle gate-exhaustive faults.

References

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

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

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

[4] H. Takahashi, Y. Tsugaoka, H. Ayano and Y. Takamatsu, "BIST Based Fault Diagnosis Using Ambiguous Test Set", in Proc. Symp. on Defect and Fault Tolerance in VLSI Systems, 2003, pp. 89-96.

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

[6] M. Abramovici, C. E. Stroud and J. M. Emmert, "Online BIST and BIST-based Diagnosis of FPGA Logic Blocks", in IEEE Trans. on VLSI Systems, Dec. 2004, Vol. 12, No. 12, pp. 1284-1294.

[7] A. B. Kahng and S. Reda, "New and Improved BIST Diagnosis Methods from Combinatorial Group Testing Theory", in IEEE Trans. on Computer-Aided Design, March 2006, Vol. 25, No. 3, pp. 533-543.

[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", in IEEE Trans. on Computer-Aided Design, Feb. 2010, Vol. 29, No. 2, pp. 299-312.

[9] 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.

[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., 2011, pp. 54-59.

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

[12] 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.

[13] R. Wang, K. Chakrabarty and S. Bhawmik, "Built-In Self-Test and Test Scheduling for Interposer-Based 2.5D IC", in ACM Trans. on Design Automation, Vol. 20, No. 4, Sep. 2015, Art. 58.

[14] M. Agrawal, K. Chakrabarty and B. Eklow, "A Distributed, Reconfigurable, and Reusable BIST Infrastructure for Test and Diagnosis of 3-D-Stacked ICs", in IEEE Trans. on Computer-Aided Design, Feb. 2016, Vol. 35, No. 2, pp. 309-322.

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

[16] I. Pomeranz, "Storage-Based Built-In Self-Test for Gate-Exhaustive Faults", in IEEE Trans. on Computer-Aided Design, 2021.

[17] I. Pomeranz, "Storage-Based Logic Built-In Self-Test with Multicycle Tests", in IEEE Trans. on Computer-Aided Design, 2021.

[18] A. H. Baba and S. Mitra, "Testing for Transistor Aging", in Proc. VLSI Test Symp., 2009, pp. 215-220.

[19] 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.

[20] S. Jin, Y. Han, H. Li and X. Li, "Unified Capture Scheme for Small Delay Defect Detection and Aging Prediction", in IEEE Trans. on VLSI Systems, May 2013, Vol. 21, No. 5, pp. 821-833.

[21] 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.

[22] S. Wang, T. Aono, Y. Higami, H. Takahashi, H. Iwata, Y. Maeda and J. Matsushima, "Capture-Pattern-Control to Address the Fault Detection Degradation Problem of Multi-cycle Test in Logic BIST", in Proc. Asian Test Symp., 2018, pp. 155-160.