Storage-Based Logic Built-In Self-Test with Multicycle Tests

Irith Pomeranz

***Abstract*— Storing deterministic test data on-chip allows logic built-in self-test ($LBIST$) to produce a special type of random tests that consist of random combinations of deterministic test data. Such tests can achieve a higher fault coverage than random tests whose bits are determined randomly. A bottleneck of this approach is the volume of test data that need to be stored. This article observes that the use of multicycle tests can address this bottleneck by reducing the number of tests needed for detecting target faults, and thus the volume of test data needed for producing them. A software procedure is described to support this solution. Experimental results for benchmark circuits demonstrate the effectiveness of multicycle tests in this context.**

***Index Terms*— Linear-feedback shift-register ($LFSR$), logic built-in self-test ($LBIST$), multicycle tests, on-chip test generation.**

I. Introduction

The use of logic built-in self-test ($LBIST$) removes the need for a tester to store and apply tests [1]-[15]. An added benefit is enhanced security by avoiding the transfer of test data to and from the chip [4]. Storing deterministic test data on-chip allows $LBIST$ to produce a special type of tests that are formed by combinations of deterministic test data. Deterministic combinations of test data are used in [3]. Random as well as deterministic combinations are used in [15]. The use of random combinations of deterministic test data results in a new type of random tests that are more effective than the conventional random tests under which all the bits are assigned randomly. The deterministic test data preserve some of the ability of deterministic tests to activate and propagate faults, and carry this ability over to the random tests. The resulting random tests (random combinations of deterministic test data) can detect defects that cannot be detected without the use of deterministic test data. This is demonstrated in [15] by targeting gate-exhaustive faults.

A bottleneck of storage-based $LBIST$ is the amount of test data that needs to be stored on-chip when the goal is to achieve complete fault coverage. This article observes that multicycle tests are useful in addressing this bottleneck. A multicycle test has one or more clock cycles between scan operations [16]-[25]. With additional clock cycles between scan operations, the test can detect more faults. It is also possible for masking to occur with additional clock cycles. Procedures for computing multicycle tests avoid adding clock cycles that result in decreased fault coverage. The ability to detect more faults with each test makes multicycle tests effective for the compaction of deterministic test sets [16]-[17]. It can also potentially support the reduction of the volume of test data for storage-based $LBIST$. This is because fewer of the more effective multicycle tests are required, and less test data are needed for forming tests.

The on-chip test generation logic from [15] is adapted to the application of multicycle tests as follows.
(1) The stored test data are partitioned into a set S of scan vectors, and a set P of primary input vectors, which are stored separately. Such a partition is not needed in [15]. For a circuit with n scan chains, a test t_i is defined by n scan vectors, $s_{i,0}$, $s_{i,1}$, ..., $s_{i,n-1}$, a primary input vector, p_i, and a number of clock cycles, k_i. A test is denoted by $t_i = \langle s_{i,0}, s_{i,1}, ..., s_{i,n-1}, p_i, k_i \rangle$. All the clock cycles between the scan operations of a test are assumed to be functional capture cycles, and the primary input vector p_i is held constant for the duration of the test.
(2) Within the sets S and P, subsets S_C and P_C, respectively, are identified that are especially effective for forming tests. The vectors in these subsets are referred to as core vectors. The on-chip test generation logic applies random tests in two phases. In the first phase it uses core vectors from S_C and P_C to form random tests. In the second phase it uses vectors from the entire sets S and P. Focusing on S_C and P_C helps increase the fault coverage.

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.

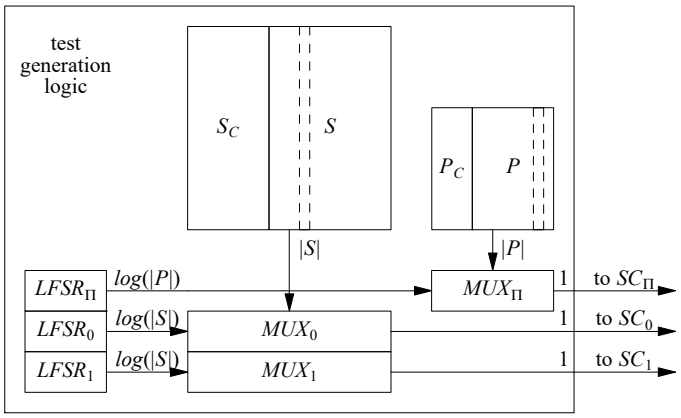

Fig. 1. On-chip test generation logic.

The article describes a software procedure that initializes S and P based on a deterministic test set. It then selects the core vectors, and reduces the sets S and P to reduce the volume of stored test data. When S and P are reduced, only vectors from $S \setminus S_C$ and $P \setminus P_C$ are eliminated. The procedure selects the least effective vectors from S and P for removal. When the on-chip test generation logic focuses on the vectors that remain in S and P, the fault coverage is increased. Thus, in addition to reducing the volume of stored test data, removing vectors from S and P also helps the procedure increase the fault coverage. The use of S_C and P_C also reduces the computational effort of minimizing S and P, since only vectors in $S \setminus S_C$ and $P \setminus P_C$ are considered during this process.

The target faults in this article are single stuck-at faults. The initial test data for S and P are derived from a single-cycle deterministic test set T_{sa} for single stuck-at faults. With multicycle tests, delay faults are detected accidentally, and can be targeted directly by replacing T_{sa} with a two-cycle test set for delay faults.

Several other approaches exist for increasing the fault coverage achieved by $LBIST$ [1]-[2], [5], [7], [11]-[13]. These approaches include the insertion of test-points, observation of next-state variables, and reseeding of $LFSR$s. The use of multicycle tests as suggested in this article can be combined with these approaches to reduce the storage requirements of the on-chip test data.

The article is organized as follows. Section II describes the on-chip test generation logic. Section III describes the software procedure for selecting the core vectors. Section IV describes the software procedure for reducing the sets S and P. Experimental results for benchmark circuits are presented in Section V.

II. On-Chip Test Generation Logic

The on-chip test generation logic is illustrated by Figure 1. Figure 1 and Table I include the notation used in this article.

The circuit has m primary inputs, and n scan chains of equal length l. A scan chain is denoted by SC_i in Figure 1. Two of the n scan chains are shown in Figure 1, SC_0 and SC_1. For uniformity, the primary inputs are also included in a scan chain denoted by SC_Π.

TABLE I
NOTATION

symbol	meaning
m	number of primary inputs
n	number of scan chains
l	length of scan chain
SC_i	scan chain
S	set of scan vectors
S_C	subset of core scan vectors
P	set of primary input vectors
P_C	subset of core primary input vectors
$B(S,P)$	number of bits for storing S and P
k	number of functional capture cycles in a test
K	upper bound on k
N_R	number of random tests
$R_{K,1}$	set of random tests in the first phase
$R_{K,2}$	set of random tests in the second phase
T_K	set of tests that detect new faults
$n_a(s_j)$, $n_a(p_j)$	number of times $s_j \in S$, $p_j \in P$ appears in T_K
n_{S_C}	number of core scan vectors
n_{P_C}	number of core primary input vectors
Ω	set of options for (n_{S_C}, n_{P_C})
N_Ω	number of entries in Ω
T_C	subset of $R_{K,1}$ that detects new faults
F_C	subset of faults detected by T_C

Figure 1 shows a memory that contains the set of scan vectors S, and a memory that contains the set of primary input vectors P. The subsets of core vectors S_C and P_C are shown inside S and P, respectively. The dashed box inside S is an l-bit scan vector. The dashed box inside P is an m-bit primary input vector.

To select a primary input vector and a scan-in state for a test, a linear-feedback shift-register ($LFSR$) denoted by $LFSR_\Pi$ or $LFSR_i$ for $0 \leq i < n$ selects a vector from P or S randomly through a multiplexer. The number of bits for $LFSR_\Pi$ is $log_2(|P|)$, and the number of bits for $LFSR_i$ is $log_2(|S|)$. The selected vector is shifted one bit at a time into the corresponding scan chain. This requires $log_2(l)$ and $log_2(m)$ counters, not shown in Figure 1.

A multiplexer produces a single bit at a time to be shifted into the corresponding scan chain. This bit needs to be routed from the test generation logic to the scan chain. This is similar to the case where test data decompression is implemented on-chip, producing scan-in values that need to be loaded into the scan chains. Overall, in addition to the memory, the on-chip test generation logic includes $n+1$ $LFSR$s and $n+1$ multiplexers. The $LFSR$s can be replaced with a single larger $LFSR$ that produces different random values for all the scan chains. The benefit of using an on-chip memory to store deterministic test data is the effectiveness of the resulting random tests as discussed earlier.

For a parameter K, the test generation logic produces random tests with $k = K$, $K-1$, ..., 1 functional capture cycles. For a constant N_R, the number of k-cycle tests is N_R. A counter (not shown in Figure 1) controls the number of functional capture cycles of a test.

Test generation proceeds in two phases. In the first phase the $LFSR$s select vectors from S_C and P_C to form a set of KN_R random tests denoted by $R_{K,1}$. In the second phase the $LFSR$s select vectors from the entire sets S and P to form a set of KN_R random tests denoted by $R_{K,2}$.

In software, the set of target faults F is simulated under $R_{K,1} \cup R_{K,2}$ with fault dropping. Tests that detect faults from F are added to a test set T_K. Forward-looking reverse order fault simulation removes unnecessary tests from T_K. A test set T_K will be used in the next sections to select the subsets of core vectors, and reduce S and P.

III. Selecting Subsets of Core Vectors

Initial sets of vectors S and P are obtained from a compact deterministic test set T_{sa} for single stuck-at faults. Every test $t_i \in T_{sa}$ is represented as $t_i = \langle s_{i,0}, s_{i,1}, ..., s_{i,n-1}, p_i, 1\rangle$. The vectors $s_{i,0}$, $s_{i,1}$, ..., $s_{i,n-1}$ are added to S, and the vector p_i is added to P. If the same vector appears more than once, it is stored only once in S or P. The number of bits for storing T_{sa} is denoted by $B(T_{sa}) \approx (nl+m)|T_{sa}|$. The number of bits for storing S and P is denoted by $B(S,P) = l|S| + m|P|$.

The computation of the core vectors is described for a number of functional capture cycles K. Initially, $S_C = \emptyset$ and $P_C = \emptyset$. Since $R_{K,1}$ consists of random combinations of vectors from S_C and P_C, the software procedure obtains $R_{K,1} = \emptyset$. Using $R_{K,2}$ it computes the set of effective tests T_K as described in Section II. A vector that appears more times in T_K is considered more important to include in the core subset. Accordingly, each vector $s_j \in S$ and $p_j \in P$ is associated with the number of times it appears in T_K. This number is denoted by $n_a(s_j)$ and $n_a(p_j)$, respectively.

The procedure orders S and P by decreasing order of $n_a(s_j)$ and $n_a(p_j)$, respectively. In this order, the top vectors in S and P are the ones most suitable for the core subsets S_C and P_C. To determine how many of the vectors will be included in the core subsets, the procedure considers options that are described by two parameters, n_{S_C} is the number of top vectors in S that are included in the core, and n_{P_C} is the number of top vectors in P that are included in the core. The selection of n_{S_C} and n_{P_C} is based on the following experimental observations.

An effective core should include a sufficient number of vectors to provide a significant fault coverage. At the same time, the number of vectors should be small enough to obtain tests that are focused on the most effective vectors, and require a limited number of storage bits. The two considerations are balanced by using a constant $0 < \omega < 1$ to ensure that S_C and P_C use approximately ω of the number of bits for T_{sa}. Thus, $B(S_C, P_C) = l|S_C| + m|P_C| \approx \omega B(T_{sa})$.

In addition, options where $B(S_C, P_C) < \omega B(T_{sa})$ may sometimes allow $R_{K,1}$ to detect more faults. The procedure considers a constant number N_Ω of options for n_{S_C} and n_{P_C} such that $B(S_C, P_C) \leq \omega B(T_{sa})$. The options are included in a set Ω by Procedure 1.

Procedure 1: Computing core options

1) Assign $\Omega = \emptyset$.
2) For n_{S_C} = 1, 2, ..., $|S|$ and for n_{P_C} = 1, 2, ..., $|P|$, if $ln_{S_C} + mn_{P_C} \leq \omega B(T_{sa})$:
   a) Add (n_{S_C}, n_{P_C}) to Ω such that Ω is sorted by increasing value of $ln_{S_C} + mn_{P_C}$.
   b) If $|\Omega| > N_\Omega$, remove the first entry from Ω.
3) For every $(n_{S_C}, n_{P_C}) \in \Omega$:
   a) Include in S_C the first n_{S_C} vectors from S.
   b) Include in P_C the first n_{P_C} vectors from P.
   c) Simulate F under $R_{K,1}$ with fault dropping.
4) Select the first pair $(n_{S_C}, n_{P_C}) \in \Omega$ for which the fault coverage is the highest.

For the pairs $(n_{S_C}, n_{P_C}) \in \Omega$, the procedure performs fault simulation with fault dropping of F under $R_{K,1}$ using the first n_{S_C} vectors from S to define S_C, and the first n_{P_C} vectors from P to define P_C. Of all the pairs in Ω, the procedure selects the one where $R_{K,1}$ yields the highest fault coverage.

IV. Reducing Sets of Vectors

This section describes a software procedure that reduces the sets of vectors S and P. The procedure accepts the sets S and P as well as the cores S_C and P_C. It also accepts a number of functional capture cycles K. The procedure removes vectors from $S \setminus S_C$ and $P \setminus P_C$ to reduce the number of bits for on-chip storage as well as increase the fault coverage. To ensure that the fault coverage is increased, the

procedure selects the least effective vectors from $S \setminus S_C$ and $P \setminus P_C$ for removal. This allows it to focus on more effective vectors, and thus increase the fault coverage. It reintroduces the vectors into S and P if the fault coverage is reduced.

The core vectors in S_C and P_C may be computed for a number of functional capture cycles that is different from K. The procedure first finds the tests based on S_C and P_C using tests with $1 \leq k \leq K$ functional capture cycles. For this purpose, the procedure simulates F under $R_{K,1}$ with fault dropping. It finds a test set T_C, and a set of faults F_C that are not detected by the core. Since the core will remain intact as S and P are reduced, only the faults in F_C are considered when attempting to remove vectors from S and P.

Using the given sets S and P, the procedure performs fault simulation with fault dropping of F_C under $R_{K,2}$, and computes a test set T_K that includes T_C. The procedure stores the number of detected faults in a variable denoted by n_{det}. The number of detected faults will not decrease as S and P are reduced.

An iteration of the procedure consists of the following steps. The procedure stores S and P in sets S_{prev} and P_{prev} to ensure that it can recover them. It then permutes $S \setminus S_C$ and $P \setminus P_C$ randomly. The permutation does not affect $R_{K,1}$. The goal is to obtain a different set of tests $R_{K,2}$ in every iteration.

Using the permuted sets S and P, the procedure performs fault simulation with fault dropping of F_C under $R_{K,2}$, and computes a test set T_K that includes T_C. Based on T_K, it updates the number of times each vector appears in T_K. As before, this number is denoted by $n_a(s_j)$ or $n_a(p_j)$.

For a constant $0 < \rho < 1$, the procedure removes from $S \setminus S_C$ up to $\rho|S|$ scan vectors for which $n_a(s_j) = 0$. In addition, it removes from $P \setminus P_C$ up to $\rho|P|$ scan vectors for which $n_a(p_j) = 0$. The removed vectors are the least effective ones in S and P. Removing them is likely to help the procedure focus on the more effective vectors, and maintain or increase the fault coverage.

To check the effect on the fault coverage, the procedure performs fault simulation with fault dropping of F_C under $R_{K,2}$. Let the number of detected faults be m_{det}. If $m_{det} \geq n_{det}$, the removal of the vectors is accepted. The procedure assigns $n_{det} = m_{det}$. If $m_{det} < n_{det}$, the procedure restores S and P by assigning $S = S_{prev}$ and $P = P_{prev}$.

The procedure performs a constant number, N_{ITER}, of iterations. The procedure is summarized as Procedure 2.

Procedure 2: Reducing the sets S and P

1) Perform fault simulation with fault dropping of F under $R_{K,1}$. Compute a test set T_K, and assign $T_C = T_K$. Assign $F_C = F$.
2) Perform fault simulation with fault dropping of F_C under $R_{K,2}$. Store the number of detected faults in n_{det}. Assign $n_{iter} = 1$.
3) Assign $S_{prev} = S$ and $P_{prev} = P$.
4) Permute $S \setminus S_C$ and $P \setminus P_C$ randomly.
5) Perform fault simulation with fault dropping of F_C under $R_{K,2}$ using the permuted sets. Compute a test set T_K that includes T_C. Update the number of times each vector $s_j \in S$ and $p_j \in P$ appears in T_K.
6) Assign $n_r = 0$. For $j = |S_C|, |S_C| + 1, ..., |S| - 1$, as long as $n_r < \rho|S_{prev}|$, if $n_a(s_j) = 0$, remove s_j from S and increment n_r.
7) Assign $n_r = 0$. For $j = |P_C|, |P_C| + 1, ..., |P| - 1$, as long as $n_r < \rho|P_{prev}|$, if $n_a(p_j) = 0$, remove p_j from P and increment n_r.
8) Perform fault simulation with fault dropping of F_C under $R_{K,2}$. Store the number of detected faults in m_{det}.
9) If $m_{det} \geq n_{det}$, assign $n_{det} = m_{det}$. Else, assign $S = S_{prev}$, $P = P_{prev}$.
10) Assign $n_{iter} = n_{iter} + 1$. If $n_{iter} \leq N_{ITER}$, go to Step 3.

TABLE II
INCREASED FAULT COVERAGE FOR $s5378$

K	$\|S\|$	$\|P\|$	s.a.
1	1625	122	99.671
1	1298	94	99.803
2	1298	94	99.912
2	971	67	99.978
3	971	67	99.978
3	598	46	100.000

Table II demonstrates the increase in the fault coverage that occurs as less effective vectors are removed from S and P. Table II is based on benchmark circuit $s5378$ with the setup described in Section V. For $K = 1$, 2 and 3, the first (second) row for K shows the numbers of vectors in S and P, and the stuck-at fault efficiency, before (after) S and P are reduced. Table II demonstrates that, for a value of K where the stuck-at fault efficiency is initially lower than 100%, the removal of less effective vectors from S and P increases the fault efficiency. Table II also demonstrates the increase in the fault efficiency that occurs as K is increased.

V. EXPERIMENTAL RESULTS

The software procedure was applied to single stuck-at faults in benchmark circuits.

The test set T_{sa} is a compact test set for single stuck-at faults. The set of target faults F consists of all the single stuck-at faults that are detected by T_{sa}. The percentage of detected faults is referred to as a fault efficiency (and not a fault coverage) since undetectable faults are eliminated from F. The software procedure is applied with the following parameter values.

For a circuit with N_{SV} state variables, the number of scan chains n and the length of a scan chain l are the smallest values such that $n = l$ and $nl \geq N_{SV}$. A large number of short scan chains is also used by test data compression methods. In the context of storage-based $LBIST$, it keeps the storage requirements of S manageable.

Core vectors are computed using $K = 1$. The procedure for reducing the sets of vectors is applied using $K = 1$, 2, ..., 8. With $K = 1$, the sets S, P, S_C and P_C are optimized for the application of single-cycle tests. After removing vectors using $K > 1$, the procedure benefits from the use of multicycle tests. These cases are compared to demonstrate the importance of using multicycle tests.

The number of random tests for every value of k is $N_R = 8192$. When computing core vectors, the procedure considers $N_\Omega = 64$ options with a number of storage bits that does not exceed $\omega B(T_{sa})$, for $\omega = 0.0625$. The value $\omega = 0.0625$ is small enough so as not to limit the reduction of S and P, yet large enough to achieve a high fault coverage. The best core is found within fewer than the last $N_\Omega = 64$ options in Ω, and a higher value of N_Ω is not needed.

In every iteration of Procedure 2, the procedure removes up to $\rho = .03125$ of the vectors from S and P. This prevents the removal of excessive numbers of vectors when the fault coverage is incomplete, and allows the procedure to increase the fault coverage as it performs additional iterations. The number of iteration is $N_{ITER} = 128$. With these values of ρ and N_{ITER}, each vector is considered for removal approximately four times. Increasing N_{ITER} reduces the storage requirements of S and P, but also increases the computational effort of the software procedure.

The use of multicycle tests has an advantage when considering the detection of delay faults. To demonstrate this advantage, transition faults are simulated under the tests in $R_{K,1} \cup R_{K,2}$.

To demonstrate the possibility of combining the use of multicycle tests with other approaches for increasing the fault coverage, several

TABLE III
EXPERIMENTAL RESULTS

circuit	sv	pi	K	tests	func	s.a.	trans	$\|S\|$	$\|P\|$	bits	ratio	ntime
s35932	1728	35	1	70	1.00	100.000	0.000	33	20	2086	0.059	3296.98
s35932	1728	35	2	44	1.75	100.000	71.800	33	20	2086	0.059	4645.66
aes_core	530	258	1	586	1.00	100.000	0.000	179	26	11004	0.067	374.22
aes_core	530	258	2	361	2.00	100.000	95.491	179	26	11004	0.067	568.72
systemcdes	190	130	1	106	1.00	100.000	0.000	26	17	2574	0.102	819.13
systemcdes	190	130	2	75	2.00	100.000	95.228	25	17	2560	0.101	1113.30
systemcdes.obs	190	130	5	39	5.00	100.000	95.770	14	16	2276	0.090	4199.81
systemcaes	670	258	1	190	1.00	100.000	0.000	372	15	13542	0.121	444.17
systemcaes	670	258	2	153	2.00	100.000	86.735	121	15	7016	0.062	745.40
simple_spi	131	15	1	67	1.00	99.952	0.000	308	26	4086	0.777	588.71
simple_spi	131	15	2	57	1.96	100.000	68.848	244	21	3243	0.617	994.81
simple_spi	131	15	3	59	2.90	100.000	73.351	47	19	849	0.162	1468.12
simple_spi	131	15	7	49	6.55	100.000	74.188	34	19	693	0.132	6626.11
simple_spi.obs	131	15	8	47	7.96	100.000	78.110	29	19	633	0.120	10981.14
i2c	128	17	1	86	1.00	99.358	0.000	397	35	5359	0.821	277.23
i2c	128	17	2	78	1.94	100.000	63.380	337	28	4520	0.693	502.85
i2c	128	17	3	74	2.92	100.000	67.669	64	25	1193	0.183	800.27
i2c	128	17	7	72	6.61	100.000	68.415	38	25	881	0.135	3286.18
i2c.obs	128	17	7	60	7.00	100.000	71.865	35	26	862	0.132	3179.64
s5378	179	35	1	220	1.00	99.803	0.000	1298	94	21462	0.822	368.71
s5378	179	35	2	226	1.86	99.978	70.283	971	67	15939	0.611	774.79
s5378	179	35	3	229	2.66	100.000	71.180	598	46	9982	0.382	1534.58
s5378	179	35	4	229	3.53	100.000	72.229	342	46	6398	0.245	2700.51
s5378	179	35	5	239	4.38	100.000	71.426	179	46	4116	0.158	4260.12
s5378	179	35	8	236	7.02	100.000	71.237	129	46	3416	0.131	11466.29
s5378.obs	179	35	7	198	6.40	100.000	75.008	99	37	2681	0.103	8906.49
s13207	669	31	1	417	1.00	98.986	0.000	5433	207	147675	0.861	160.38
s13207	669	31	2	376	1.89	99.897	73.769	5433	207	147675	0.861	325.66
s13207	669	31	3	363	2.79	99.969	76.159	4938	186	134154	0.782	584.23
s13207	669	31	4	358	3.75	100.000	76.846	4633	174	125852	0.734	961.37
s13207	669	31	5	344	4.62	100.000	77.214	3707	136	100598	0.587	1471.14
s13207	669	31	6	351	5.53	100.000	77.548	1667	54	45016	0.262	2139.41
s13207	669	31	7	355	6.51	100.000	77.756	538	15	14453	0.084	2864.87
s13207	669	31	8	360	7.44	100.000	78.102	475	13	12753	0.074	3629.56
spi	229	45	1	458	1.00	99.431	0.000	5791	380	109756	0.987	276.10
spi	229	45	2	373	1.96	99.738	62.166	5099	333	96569	0.868	632.12
spi	229	45	3	368	2.90	99.861	67.505	4489	291	84919	0.763	1194.78
spi	229	45	4	378	3.88	99.908	67.330	4348	281	82213	0.739	2016.47
spi	229	45	5	377	4.84	99.969	68.015	4348	281	82213	0.739	3089.11
spi	229	45	6	377	5.79	99.985	69.644	4348	281	82213	0.739	4446.67
spi	229	45	8	368	7.73	100.000	72.268	4080	263	77115	0.693	8085.39
spi.obs	229	45	8	347	7.86	100.000	81.697	387	116	11412	0.103	7712.57
b14	247	33	1	172	1.00	88.045	0.000	4224	266	76362	0.821	296.50
b14	247	33	2	181	1.88	89.038	61.478	3603	224	65040	0.700	965.22
b14	247	33	3	157	2.73	93.759	67.747	3274	203	59083	0.636	1862.15
s15850	597	14	1	320	1.00	97.803	0.000	2726	106	69634	0.863	273.35
s15850	597	14	2	284	1.91	98.500	60.185	2399	91	61249	0.759	727.94
s15850	597	14	3	301	2.83	98.924	60.948	1981	73	50547	0.627	1510.50
b20	494	33	1	248	1.00	92.664	0.000	5735	238	139759	0.878	275.77
b20	494	33	2	254	1.85	93.064	73.921	5212	215	126971	0.798	874.42
b20	494	33	3	237	2.81	96.605	76.778	4445	210	109165	0.686	1624.78
s38417	1636	28	1	618	1.00	97.530	0.000	4403	106	183491	0.904	190.54
s38417	1636	28	2	653	1.83	98.981	91.339	4265	102	177721	0.875	415.70
s38417	1636	28	3	675	2.68	99.507	93.936	3753	88	156337	0.770	674.13
wb_dma	523	215	1	186	1.00	99.362	0.000	1379	59	44402	0.912	364.04
wb_dma	523	215	2	187	1.89	99.582	60.069	1293	55	41564	0.853	827.78
wb_dma	523	215	3	191	2.80	99.714	61.235	1212	51	38841	0.797	1688.73
wb_dma.obs	523	215	3	191	2.85	99.857	63.771	1174	49	37537	0.771	1504.88
wb_dma.obs	523	215	5	174	4.67	99.901	64.289	1174	49	37537	0.771	4395.95
wb_dma.obs	523	215	8	183	7.61	99.901	64.594	1032	41	32551	0.668	12978.67
s9234	228	19	1	300	1.00	97.730	0.000	2143	134	36834	0.932	207.52
s9234	228	19	2	288	1.72	98.239	66.017	2076	129	35667	0.903	513.12
s9234	228	19	3	282	2.43	98.903	68.935	2011	124	34532	0.874	985.63
s9234.obs	228	19	3	213	2.89	99.925	79.209	1770	108	30372	0.769	708.56
s9234.obs	228	19	4	208	3.85	99.985	80.321	1460	87	25013	0.633	1162.00
s9234.obs	228	19	5	194	4.79	100.000	80.833	1369	81	23443	0.593	1756.14
s9234.obs	228	19	6	196	5.81	100.000	81.351	765	39	12981	0.328	2475.76
s9234.obs	228	19	7	185	6.79	100.000	81.248	348	39	6309	0.160	3346.90
s9234.obs	228	19	8	193	7.77	100.000	80.780	122	39	2693	0.068	4373.12

benchmark circuits are considered with observation points inserted into them. In this case, the name of the circuit is followed by *.obs*.

The results are shown in Table III. Most of the circuits are considered with $1 \leq K \leq 8$. There is a row in Table III for every value of K where the single stuck-at fault efficiency is increased, or the storage requirements are reduced, satisfying the following condition. As before, $B(S,P) = l|S| + m|P|$ is the number of bits for storing S and P, and $B(T_{sa})$ is the number of bits for storing the stuck-at test set T_{sa}. A value of K is reported if $B(S,P)/B(T_{sa})$ decreases below ..., 0.3, 0.2, 0.1. In addition, the last value of K where the storage requirements are reduced is reported. For a circuit with observation points only the results with the final value of K are shown.

The circuits in Table III are arranged according to the value of K

where the fault efficiency reaches 100%, from low to high. This value of K is denoted by K_{sa}. For the same value of K_{sa}, the circuits are arranged according to the ratio $B(S,P)/B(T_{sa})$, from low to high. For the circuits at the end of Table III, only $1 \leq K \leq 3$ is reported, and the fault efficiency does not reach 100% for these values of K. Thus, the circuits that appear later in Table III have more faults that are difficult to detect. The circuits at the end of Table III are the ones with the most difficult to detect faults, and $1 \leq K \leq 3$ is used for demonstrating the importance of multicycle tests. Observation points are considered as well for several of these circuits.

In Table III, after the circuit name, column sv shows the number of state variables and column pi shows the number of primary inputs. Column K shows the value of K. Column $tests$ shows the number of tests in the test set T_K. Column $func$ shows the average number of functional capture cycles for a test in T_K. Column $s.a.$ shows the fault efficiency for stuck-at faults. Column $trans$ shows the fault coverage for transition faults. Column $|S|$ shows the number of scan vectors in S. Column $|P|$ shows the number of primary input vectors in P. Column $bits$ shows the number of bits for storing S and P, $B(S,P) = l|S| + m|P|$. Column $ratio$ shows the number of bits for storing S and P as a fraction of the number of bits for storing T_{sa}. Column $ntime$ shows the normalized runtime, where the cumulative runtime of the software procedure is divided by the runtime for fault simulation with fault dropping of $R_{1,2}$ using the initial sets S and P, with $S_C = \emptyset$ and $P_C = \emptyset$. Under $R_{1,2}$, fault simulation is carried out for $N_R = 8192$ random single-cycle tests.

The following points can be seen from Table III. The fault efficiency for single stuck-at faults reaches 100% for most of the circuits. It is close to 100% for the circuits at the end of Table III, with the most difficult to detect faults.

For many of the circuits, the final ratio of storage requirements is lower than 0.1. For the circuits that appear later in Table III, the faults are more difficult to detect, and the increase in the fault efficiency requires more of the vectors to be retained in S and P. Nevertheless, reductions in storage requirements occur for these circuits as well. These effects demonstrate the importance of using multicycle tests for reducing the storage requirements.

The accidental transition fault coverage that occurs for multicycle tests is another reason to use multicycle tests. The transition fault coverage is typically higher for higher values of K.

The number of applied tests in $R_{K,1} \cup R_{K,2}$ is $2N_RK = 16384K$, and it increases with K. The number of tests in T_K that are effective in detecting faults decreases as K is increased. This is consistent with the effectiveness of multicycle tests for test compaction.

These effects are independent of the size of the circuit, and circuits of different sizes show similar trends of reduced storage requirements and increased fault coverages. The normalized runtime of the software procedure does not increase with the size of the circuit, indicating that the procedure scales similar to a fault simulation procedure.

VI. Concluding Remarks

This article suggested the use of multicycle tests as part of storage-based logic built-in self test ($LBIST$). Multicycle tests are effective for reducing the volume of test data that need to be stored since each test detects more faults. As a result, fewer tests are needed, requiring smaller numbers of scan and primary input vectors. A software procedure was described to support this solution. Experimental results for benchmark circuits demonstrated that, as the procedure reduces the number of vectors that need to be stored, and the number of clock cycles is increased, the fault coverage increases, and the storage requirements decrease.

References

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

[2] S. Hellebrand, S. Tarnick, J. Rajski and B. Courtois, "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Register", in Proc. Intl. Test Conf., 1992, pp. 120-129.

[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", IEEE Trans. on Computers, Nov. 2002, pp. 1282-1993.

[4] 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.

[5] S. Udar and D. Kagaris, "LFSR Reseeding with Irreducible Polynomials", in Proc. Intl. On-Line Testing Symp., 2007, pp. 293-298.

[6] 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.

[7] O. Acevedo and D. Kagaris, "Using the Berlekamp-Massey Algorithm to Obtain LFSR Characteristic Polynomials for TPG", in Proc. Intl. Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2012, pp. 233-238.

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

[9] 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.

[10] 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.

[11] F. Zhang, D. Hwong, Y. Sun, A. Garcia, S. Alhelaly, G. Shofner, L. Winemberg and J. Dworak, "Putting Wasted Clock Cycles to Use: Enhancing Fortuitous Cell-aware Fault Detection with Scan Shift Capture", in Proc. Intl. Test Conf., 2016 pp. 1-10.

[12] G. Mrugalski, J. Rajski, J. Solecki, J. Tyszer and C. Wang, "Trimodal Scan-Based Test Paradigm", IEEE Trans. on VLSI Systems, March 2017, Vol. 25, No. 3, pp. 1112-1125.

[13] Y. Liu, J. Rajski, S. M. Reddy, J. Solecki and J. Tyszer, "Staggered ATPG with Capture-per-cycle Observation Test Points", in Proc. VLSI Test Symp., 2018, pp. 1-6.

[14] 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.

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

[16] S. Y. Lee and K. K. Saluja, "Test Application Time Reduction for Sequential Circuits with Scan", IEEE Trans. on Computer-Aided Design, Sept. 1995, pp. 1128-1140.

[17] I. Pomeranz and S. M. Reddy, "Static Test Compaction for Scan-Based Designs to Reduce Test Application Time", in Proc. Asian Test Symp., 1998, pp. 198-203.

[18] X. Lin and R. Thompson, "Test Generation for Designs with Multiple Clocks", in Proc. Design Autom. Conf., 2003, pp. 662-667.

[19] G. Bhargava, D. Meehl and J. Sage, "Achieving Serendipitous N-Detect Mark-Offs in Multi-Capture-Clock Scan Patterns", in Proc. Intl. Test Conf, 2007, Paper 30.2.

[20] I. Park and E. J. McCluskey, "Launch-on-Shift-Capture Transition Tests", in Proc. Intl. Test Conf., 2008, pp. 1-9.

[21] E. K. Moghaddam, J. Rajski, S. M. Reddy and M. Kassab, "At-Speed Scan Test with Low Switching Activity", in Proc. VLSI Test Symp., 2010, pp. 177-182.

[22] D. Erb, K. Scheibler, M. Sauer, S. M. Reddy and B. Becker, "Multi-cycle Circuit Parameter Independent ATPG for Interconnect Open Defects", in Proc. VLSI Test Symp., 2015, pp. 1-6.

[23] I. Pomeranz, "A Multi-Cycle Test Set Based on a Two-Cycle Test Set with Constant Primary Input Vectors", IEEE Trans. on Computer-Aided Design, July 2015, pp. 1124-1132.

[24] T. McLaurin and I. P. Lawrence, "Improving Power, Performance and Area with Test: A Case Study", in Proc. Intl. Test Conf., 2018, pp. 1-10.

[25] I. Pomeranz, "Test Compaction Under Bounded Transparent-Scan", in Proc. VLSI Test Symp., 2019.