Skewed-Load Tests for Transition and Stuck-at Faults

Irith Pomeranz

***Abstract*—Test generation procedures target a variety of fault models in order to produce test sets that are effective for defect detection. This paper considers the likely scenario where two-cycle skewed-load tests are generated to detect single transition faults, and the test set is complemented with tests for single stuck-at faults that are not detected by the transition fault test set. For this scenario, this paper makes several unique observations that can be utilized to produce a single compact test set that consists only of two-cycle skewed-load tests for both fault models. The first observation is that a single-cycle test for a stuck-at fault can be transformed into a skewed-load test that is guaranteed to detect the stuck-at fault without performing logic or fault simulation. The second observation is that skewed-load tests, which are transformed from single-cycle tests for stuck-at faults, sometimes detect more transition and stuck-at faults than tests that were generated for transition faults. The third observation is that a static test compaction procedure, which is based on the modification and removal of tests, is effective in this context because it allows tests for stuck-at faults to detect more transition faults and vice versa. This paper describes a test compaction procedure based on these observations and presents experimental results for benchmark circuits to demonstrate the effectiveness of the procedure.**

***Index Terms*—Skewed-load tests, stuck-at faults, test compaction, transition faults.**

I. Introduction

Test generation procedures target a variety of fault models in order to produce test sets that are effective for defect detection [1]–[6]. Fault models can be considered simultaneously by a test generation procedure that accommodates different fault models [1], or by a test generation procedure that uses a generalized fault model where faults from different models are described using the same format [2]. Fault models can also be considered consecutively, extending a test set to detect faults that it does not already detect accidentally as additional fault models are considered [4].

This paper considers the likely scenario where two-cycle skewed-load (launch-on-shift) tests [7] are generated to detect single transition faults, and the test set is complemented with tests for single stuck-at faults that are not detected by the transition fault test set. For this scenario, this paper makes the following unique observations and utilizes them to produce a single compact test set that consists only of two-cycle skewed-load tests for both fault models.

A. Observation 1

Test generation procedures for single stuck-at faults typically generate single-cycle tests. A single-cycle test for a single stuck-at fault can be transformed into a skewed-load test that is guaranteed to detect the fault without performing any logic or fault simulation. This property does not exist for broadside (launch-on-capture) tests [8]. It implies that it is possible to generate a skewed-load test set for both transition and stuck-at faults as illustrated in the first part of Fig. 1.

Manuscript received January 13, 2018; revised June 29, 2018 and August 11, 2018; accepted August 27, 2018. Date of publication October 1, 2018; date of current version September 18, 2019. This work was supported by NSF under Grant CCF-1714147. This paper was recommended by Associate Editor N. Nicolici.

The author is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: pomeranz@ecn.purdue.edu).

Digital Object Identifier 10.1109/TCAD.2018.2873233

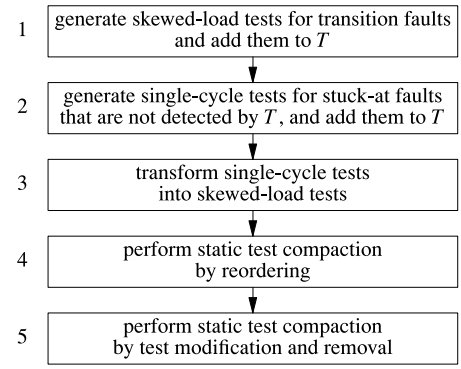

Fig. 1. Test compaction procedure.

Skewed-load tests for transition faults are generated first and added to a test set T in the box marked 1. Next, single-cycle tests are generated for single stuck-at faults and added to T in the box marked 2. The tests detect single stuck-at faults that are not already detected by T. The single-cycle tests are transformed into skewed-load tests in the box marked 3. The transformation of single-cycle tests into skewed-load tests is discussed in Section II.

B. Observation 2

Strictly viewed, a single-cycle test does not detect transition faults since transition faults require two cycles for fault activation. It is possible to use the last scan shift cycle of a single-cycle test as part of a two-cycle test [4]. However, such a test misses a scan-shift cycle that is required for bringing the circuit into a fully specified state. After a single-cycle test is transformed into a two-cycle skewed-load test, the skewed-load test detects transition faults. Moreover, experimental results show that such tests are sometimes more effective than tests for transition faults in detecting both types of faults (an example is given later). If such tests are placed earlier in the test set, fault simulation with fault dropping is sufficient for identifying tests that appear later and are not necessary for detecting any faults. Unnecessary tests can be removed from the test set. The unique feature of this observation is that it allows a test that was generated for a single stuck-at fault to remove a test that was generated for a transition fault. This is not possible without transforming a single-cycle test into a two-cycle skewed-load test. A static test compaction procedure that takes advantage of this observation to reorder the test set and reduce its number of tests is applied in the box marked 4 of Fig. 1. Section III discusses this static test compaction procedure. The importance of this procedure is mainly that it identifies effective targets for the next static test compaction procedure.

C. Observation 3

Further compaction can be achieved by modifying a test $t_{\rm mod} \in T$ to detect (transition and stuck-at) faults that are detected by a test $t_{\rm rem} \in T$. If the modification is successful, and $t_{\rm mod}$ detects all the faults that $t_{\rm rem}$ detects, $t_{\rm rem}$ can be removed from T. This approach to test compaction was used earlier in [9]. Its unique feature in the context considered in this paper is that $t_{\rm mod}$ and $t_{\rm rem}$ may have been

generated for different fault models. For example, t_{rem} may have been generated for transition faults, while t_{mod} may have been generated as a single-cycle test for stuck-at faults. After t_{mod} is transformed into a skewed-load test and modified, the number of transition faults that it detects is increased, allowing t_{rem} to be removed. The corresponding static test compaction procedure is applied in the box marked 5 in Fig. 1. Section IV discusses this static test compaction procedure.

It is possible to apply the two static test compaction procedures iteratively. After the test modification and removal procedure modifies and removes some tests, the numbers of faults that the remaining tests detect change. Applying the reordering procedure again places the tests in different positions, allowing the test modification and removal procedure to continue modifying and removing tests.

As an alternative to the procedure described in this paper, it is possible to generate skewed-load tests for transition and stuck-at faults together without first generating single-cycle tests for stuck-at faults. However, existing test generation procedures typically generate single-cycle tests, and not skewed-load tests, for stuck-at faults. In addition, mixing the faults may cause the test generation procedure to generate more tests than necessary if stuck-at faults that are targeted earlier are detected again later by tests for transition faults. The procedure described in this paper allows transition faults to be targeted first, followed by stuck-at faults, as in [4]. It identifies opportunities to reduce the number of tests and thus provide test compaction for the two fault models together.

It should also be noted that it is possible to reverse the order by which transition and stuck-at faults are targeted in Fig. 1, and target stuck-at faults first. Experimental results indicate that smaller test sets are obtained if transition faults are targeted first.

Finally, it should be noted that, under skewed-load tests, the first cycle of the test is applied in scan mode, and the second cycle is applied in functional mode under a fast clock. Therefore, the scan enable signal needs to change at the circuit speed between the first and second cycle. The generation of fast scan enable signals locally from the global scan enable signal was considered in [10] and [11].

II. Transforming Single-Cycle Stuck-at Tests

This section discusses the transformation of a single-cycle test for a single stuck-at fault into a skewed-load test.

A single-cycle test is denoted by $\langle s_i, v_i \rangle$, where s_i is the scan-in state, and v_i is the primary input vector that is applied during the functional capture cycle of the test.

A skewed-load test is denoted by $\langle s_{i,0}, s_{i,1}, v_i \rangle$, where $s_{i,0}$ is the scan-in state. After $s_{i,0}$ is scanned in, the first cycle of the test is applied in shift mode, bringing the circuit into state $s_{i,1}$. For example, suppose that the circuit has five flip-flops in a single scan chain that is shifted to the right. With $s_{i,0} = 00111$, it is possible to obtain $s_{i,1} = 00011$ or $s_{i,1} = 10011$, depending on the scan-in vector during the first cycle of the test. In the description of the test, the scan-in vector is inferred from $s_{i,1}$. For example, in the test $\langle 00111, 00011, 00 \rangle$, the scan-in vector is 0.

The second cycle of the test is a functional capture cycle. During both cycles of the test, the primary input vector v_i is applied to the primary inputs. A constant primary input vector is used for addressing tester limitations that prevent the primary input vector from being changed at-speed during the test.

The transformation of a single-cycle test $\langle s_i, v_i \rangle$ into a skewed-load test is illustrated by Fig. 2. Fig. 2 shows two copies of the combinational logic of the circuit that correspond to the two cycles of the skewed-load test. The transformation proceeds as follows. The primary input vector v_i is used for the skewed-load test. The state $s_{i,1}$ is selected to be equal to s_i. This determines most of the bits of $s_{i,0}$. For example, suppose that $\langle s_i, v_i \rangle = \langle 00011, 00 \rangle$. With $s_{i,1} = s_i = 00011$, it is possible to use $s_{i,0} = 00110$ or $s_{i,0} = 00111$. In both cases, a right shift of $s_{i,0}$ with a scan-in vector of 0 yields $s_{i,1}$. The bits of $s_{i,0}$ that are not determined by $s_{i,1}$ are determined arbitrarily. The resulting skewed-load test is $\langle s_{i,0}, s_i, v_i \rangle$.

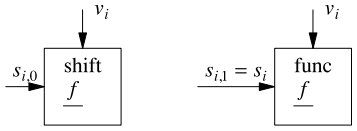

Fig. 2. Transforming a single-cycle test into a skewed-load test.

Suppose that f is a single stuck-at fault that is detected by the single-cycle test $\langle s_i, v_i \rangle$. It is guaranteed that f is detected by the skewed-load test $\langle s_{i,0}, s_i, v_i \rangle$. This can be seen as follows.

In Fig. 2, the fault f is shown in both clock cycles of the test. During the first cycle of the test, the circuit is in state $s_{i,0}$ and the primary input vector is v_i. Even if f is activated and propagated by this pattern, the flip-flops do not capture fault effects from the combinational logic of the circuit during a scan shift cycle. Therefore, during the second cycle, the circuit is in state $s_{i,1} = s_i$, and the primary input vector is v_i. This results in the application of the single-cycle test $\langle s_i, v_i \rangle$ to the circuit. The second clock cycle is a functional capture cycle, and fault effects from the combinational logic are captured in the flip-flops. Therefore, the scan-out operation at the end of the skewed-load test detects the fault.

A similar transformation of a single-cycle test into a broadside test requires $s_{i,0}$ to be such that a state-transition from $s_{i,0}$ to s_i exists. This has the computational complexity of state justification, and $s_{i,0}$ does not always exist. Moreover, the broadside test is not guaranteed to detect the single stuck-at fault f even if it exists since the faulty circuit may not enter state s_i. The simplicity of transforming a single-cycle test into a skewed-load test, without any logic or fault simulation, is the motivation for considering skewed-load tests for the detection of transition and stuck-at faults.

For simplicity, transition and stuck-at faults are represented using the same format, as follows. The transition fault $g : a \rightarrow b$, with $a \in \{0, 1\}$, $b \in \{0, 1\}$, and $a = b'$, delays the $a \rightarrow b$ transition on line g. Detection of the fault by a skewed-load test requires the assignment $g = a$ in the first cycle of the test, and $g = b$ in the second cycle. In addition, the fault g stuck-at a should be detected in the second cycle.

With $a = x$, where x is an unspecified value, the fault $g : x \rightarrow b$ is a stuck-at fault. The requirement $g = x$ in the first cycle does not put any constraints on g in this cycle. The assignment $g = b$ is necessary for detecting the stuck-at fault in the second cycle.

The set of all the target faults is denoted by F, and consists of single transition and stuck-at faults.

Referring to Fig. 1, test generation for F starts by generating skewed-load tests for the transition faults in F, and adding the tests to a test set T. Next, fault simulation of stuck-at faults in F is carried out under T with fault dropping. Test generation for the stuck-at faults that remain in F is carried out next, and the tests are added to T. Single-cycle tests are generated for stuck-at faults. The tests are then transformed into skewed-load tests as described in this section. A test t_i has $\pi_i = \text{TR}$ if it was generated for a transition fault, and $\pi_i = \text{SA}$ if it was generated for a stuck-at fault. The resulting test set is referred to as the initial test set.

III. Test Reordering

This section describes the static test compaction procedure that is based on reordering of the test set. The procedure is motivated by observation 2 from Section I.

After creating the initial test set T, the test set is ordered such that tests, which were generated for transition faults ($\pi_i = \text{TR}$), appear first, followed by tests that were generated for single stuck-at faults ($\pi_i = \text{SA}$). The reordering procedure reverses this order, and places the tests with $\pi_i = \text{SA}$ ahead of the tests with $\pi_i = \text{TR}$. It performs

fault simulation with fault dropping of T in its new order to identify tests that do not detect any faults. Such tests are removed from T.

By placing the tests with $\pi_i = \text{SA}$ ahead of the tests with $\pi_i = \text{TR}$, the procedure allows tests with $\pi_i = \text{SA}$ that are especially effective in detecting transition faults to remove tests with $\pi_i = \text{TR}$.

In an iterative application of the static test compaction procedures, this part of the test reordering procedure is applied only once, at the beginning. The following part of the test reordering procedure is applied in every iteration.

During fault simulation with fault dropping of the test set T, the procedure associates with every test $t_i \in T$ the number of faults from F that it detects. This number is denoted by d_i. Since fault simulation is carried out with fault dropping, it is typically the case that $d_i > d_j$ for $i < j$, i.e., a test that appears earlier in the test set detects more faults. However, occasionally, it is possible to obtain $d_i < d_j$ for $i < j$, i.e., t_j detects more faults than t_i even though it is simulated later. In this case, t_j is more effective than t_i for test compaction. If the order of the tests is reversed, such that t_j appears before t_i, and fault simulation with fault dropping is repeated, it is likely that d_j will increase further, and d_i will decrease. If $d_i = 0$ is obtained for t_i in its new position, t_i can be removed from T.

Based on this discussion, the static test compaction procedure repeats the following steps iteratively. It reorders the test set such that the tests appear by descending number of detected faults. For tests with the same number of detected faults, the procedure places tests with $\pi_i = \text{SA}$ earlier than tests with $\pi_i = \text{TR}$. This, again, allows tests with $\pi_i = \text{SA}$ that are especially effective in detecting transition and stuck-at faults to remove tests with $\pi_i = \text{TR}$.

Using the new order, the procedure performs fault simulation with fault dropping, and recalculates the value of d_i for every $t_i \in T$. Finally, it removes tests with $d_i = 0$ from the end of the test set.

The procedure terminates when reordering does not change the order of the tests. The procedure may perform several iterations before this condition is reached. The procedure is guaranteed to terminate. This can be seen by considering the cases where the order is changed, as follows. Suppose that the test t_{i_0} detects the largest number of faults, with $i_0 > 0$ and $d_{i_0} > d_0$. Reordering brings t_{i_0} to the top of the test set. Fault simulation with fault dropping does not decrease its number of detected faults when it is simulated first. Therefore, the number of faults for the test at the top of the test set increases from d_0 to at least d_{i_0}. Let us renumber the test at the top of the test set as t_0, and its number of detected faults as d_0. Because other tests have changed their position, it is possible that a different test, t_{i_1}, would detect the largest number of faults, with $i_1 > 0$ and $d_{i_1} > d_0$. Reordering brings t_{i_1} to the top of the test set. Fault simulation with fault dropping does not decrease its number of detected faults. Therefore, the number of faults for the test at the top of the test set increases further. A test reaches the top of the test set at most once. If it is moved to a different position, its number of detected faults cannot increase above that of the test at the top of the test set. Therefore, after a number of iterations that is bounded by the number of tests, the test at the top of the test set will not change any further. It is now possible to consider the second test in the same manner and show that after a number of iterations that is bounded by the number of tests, it will not change any further. The same arguments can be applied to the third, fourth, ... test to show that the procedure eventually terminates.

To demonstrate that tests with $\pi_i = \text{SA}$ sometimes detect more transition and stuck-at faults than tests with $\pi_i = \text{TR}$, the next example considers *s*5378. After adding stuck-at tests to a transition fault test set, and reversing the order of the tests, the number of tests with $\pi_i = \text{TR}$ is 183, and the number of tests with $\pi_i = \text{SA}$ is 48.

The reordering procedure reduces the number of tests to 230. Tests with $\pi_i = \text{SA}$ appear in the final reordered test set as shown in Table I. Several tests with $\pi_i = \text{SA}$ remain ahead of tests with $\pi_i = \text{TR}$ because they detect more faults.

TABLE I
EXAMPLE OF REORDERING

i	π_i	d_i	D_i	i	π_i	d_i	D_i
0	SA	1799	1799	14	TR	143	2113
1	SA	1562	2037	15	SA	124	2010
2	TR	1469	2818	16	TR	107	2145
.	.	.	.	.	.		
26	TR	46	1825	37	TR	30	1873
27	SA	45	2088	38	SA	29	2135
28	TR	44	1950	39	TR	29	1835
.	.	.	.	.	.		

Column D_i of Table I shows the number of stuck-at and transition faults that are detected by each test when fault simulation is carried out without fault dropping. Here, again, there are tests with $\pi_i = \text{SA}$ that detect more faults than tests with $\pi_i = \text{TR}$.

Table I demonstrates that reordering based on the numbers of detected faults can change the position of a test significantly. Even if it does not reduce the number of tests significantly, because the test set has already been compacted, the placement of the tests is beneficial for the procedure described in the next section.

IV. TEST MODIFICATION AND REMOVAL

The static test compaction procedure described in this section attempts to remove tests from T by ensuring that the faults they detect are detected by other tests. The overall structure of the procedure is given in Procedure 1.

Procedure 1: Test modification and removal.

1) For $t_{\text{rem}} = t_{n-1}, t_{n-2}, \ldots, t_0$, if $d_{\text{rem}} = 1$
   a) For $t_{\text{mod}} = t_{\text{rem}-1}, t_{\text{rem}-2}, \ldots, t_0$, if $d_{\text{mod}} > 1$
      i) Attempt to modify t_{mod} such that it would detect all the faults in $F_{\text{mod}} \cup \{f_{\text{rem}}\}$.
      ii) If the modification of t_{mod} is successful
         A) Perform fault simulation with fault dropping of T.
         B) Remove from T every test t_i with $d_i = 0$.

Suppose that a test $t_{\text{rem}} \in T$ detects a subset of faults that is denoted by F_{rem}, with $d_{\text{rem}} = |F_{\text{rem}}|$. Suppose that a test $t_{\text{mod}} \in T$ detects a subset of faults that is denoted by F_{mod}, with $d_{\text{mod}} = |F_{\text{mod}}|$. If t_{mod} can be modified to detect the faults in $F_{\text{mod}} \cup F_{\text{rem}}$, then t_{rem} can be removed from the test set. The likelihood of modifying t_{mod} successfully is higher when d_{rem} is smaller, as well as when d_{mod} is smaller. In this section, t_{rem} is considered for removal only if $d_{\text{rem}} = 1$. The one fault that is included in F_{rem} is denoted by f_{rem}.

For a test set T with n tests, the procedure considers tests for removal in the order $t_{\text{rem}} = t_{n-1}, t_{n-2}, \ldots, t_0$. A test t_{rem} is considered for removal only if $d_{\text{rem}} = 1$.

For t_{rem}, the procedure considers tests for modification in the order $t_{\text{mod}} = t_{\text{rem}-1}, t_{\text{rem}-2}, \ldots, t_0$. A test t_{mod} is considered for modification only if $d_{\text{mod}} > 1$. In this way, the procedure avoids modifying a test that detects one fault, and can potentially be removed later if it is not modified.

For t_{rem} and t_{mod}, the procedure attempts to modify t_{mod} such that it would detect $F_{\text{mod}} \cup \{f_{\text{rem}}\}$. If the modification is successful, with t_{mod} appearing before t_{rem} in T, fault simulation with fault dropping of T will result in $d_{\text{rem}} = 0$. There may be other tests in T that can be removed after t_{mod} is modified. This can occur directly because of the modification of t_{mod}, or because of the modification of tests that were considered earlier. The procedure removes from T every test t_i with $d_i = 0$.

The modification of t_{mod} can be implemented by generating a new test that will replace t_{mod}. The implementation used in this paper is based on complementing bits of t_{mod} one by one. The complementation of a bit is accepted if t_{mod} continues to detect all the faults in F_{mod}. Otherwise, the bit is complemented back to its original value. The modification terminates if t_{mod} also detects f_{rem}. Otherwise, it

TABLE II
EXPERIMENTAL RESULTS

				tests				rem,mod				f.c.			
circuit	sv	iter	proc	tot	frac	tr	sa	tr,tr	tr,sa	sa,tr	sa,sa	tr	sa	ntime	fstime
s5378	179	0	initial	244	1.000	196	48	0	0	0	0	79.594	99.131	1.00	2.99
s5378	179	0	reorder	231	0.947	183	48	0	0	0	0	79.594	99.131	1.77	2.99
s5378	179	1	mod+rem	219	0.898	171	48	4	4	0	0	79.594	99.131	173.98	2.99
s5378	179	3	mod+rem	195	0.799	152	43	19	6	5	0	79.594	99.131	705.02	2.99
s5378	179	4	mod+rem	186	0.762	147	39	21	7	7	1	79.594	99.131	929.87	2.99
s9234	228	0	initial	328	1.000	239	89	0	0	0	0	80.507	93.475	1.00	22.56
s9234	228	0	reorder	311	0.948	222	89	0	0	0	0	80.507	93.475	1.69	22.56
s9234	228	1	mod+rem	295	0.899	209	86	4	2	0	0	80.534	93.475	49.72	22.56
s9234	228	4	mod+rem	262	0.799	181	81	18	9	3	1	80.691	93.475	392.51	22.56
s9234	228	4	mod+rem	260	0.793	179	81	20	9	3	1	80.691	93.475	437.52	22.56
s13207	669	0	initial	408	1.000	396	12	0	0	0	0	89.024	98.462	1.00	96.42
s13207	669	0	reorder	405	0.993	393	12	0	0	0	0	89.024	98.462	1.67	96.42
s13207	669	2	mod+rem	366	0.897	355	11	36	0	1	0	89.024	98.462	922.59	96.42
s13207	669	6	mod+rem	353	0.865	346	7	44	1	5	0	89.024	98.462	1803.34	96.42
s15850	597	0	initial	323	1.000	243	80	0	0	0	0	88.840	96.682	1.00	74.09
s15850	597	0	reorder	282	0.873	202	80	0	0	0	0	88.840	96.682	1.67	74.09
s15850	597	1	mod+rem	258	0.799	179	79	14	5	0	0	88.850	96.682	231.02	74.09
s15850	597	3	mod+rem	236	0.731	158	78	26	11	1	0	88.856	96.682	731.55	74.09
s35932	1728	0	initial	46	1.000	29	17	0	0	0	0	73.492	89.809	1.00	68.71
s35932	1728	0	reorder	46	1.000	29	17	0	0	0	0	73.492	89.809	1.85	68.71
s35932	1728	1	mod+rem	45	0.978	28	17	1	0	0	0	73.492	89.809	11.04	68.71
s38417	1636	0	initial	716	1.000	642	74	0	0	0	0	96.835	99.471	1.00	395.97
s38417	1636	0	reorder	668	0.933	594	74	0	0	0	0	96.835	99.471	1.77	395.97
s38417	1636	1	mod+rem	644	0.899	570	74	5	0	0	0	96.854	99.471	1281.68	395.97
s38417	1636	1	mod+rem	572	0.799	498	74	55	5	0	0	96.877	99.471	8300.45	395.97
s38417	1636	2	mod+rem	556	0.777	483	73	66	5	1	0	96.883	99.471	9152.70	395.97
s38584	1452	0	initial	401	1.000	344	57	0	0	0	0	79.241	95.852	1.00	195.74
s38584	1452	0	reorder	381	0.950	324	57	0	0	0	0	79.241	95.852	1.80	195.74
s38584	1452	1	mod+rem	359	0.895	303	56	14	0	0	0	79.244	95.852	338.19	195.74
s38584	1452	3	mod+rem	320	0.798	268	52	39	4	4	0	79.246	95.852	1783.89	195.74
b14	247	0	initial	384	1.000	214	170	0	0	0	0	88.044	94.970	1.00	48.06
b14	247	0	reorder	380	0.990	210	170	0	0	0	0	88.044	94.970	1.72	48.06
b14	247	1	mod+rem	357	0.930	192	165	7	3	1	0	88.545	95.061	381.14	48.06
b15	447	0	initial	559	1.000	401	158	0	0	0	0	93.625	98.620	1.00	80.55
b15	447	0	reorder	550	0.984	392	158	0	0	0	0	93.625	98.620	1.74	80.55
b15	447	1	mod+rem	501	0.896	356	145	8	14	3	3	93.718	98.625	410.49	80.55
b15	447	3	mod+rem	483	0.864	341	142	16	17	3	6	93.746	98.625	1104.12	80.55
b20	494	0	initial	420	1.000	207	213	0	0	0	0	89.292	94.167	1.00	125.35
b20	494	0	reorder	403	0.960	190	213	0	0	0	0	89.292	94.167	1.74	125.35
b20	494	1	mod+rem	377	0.898	165	212	8	8	0	0	89.635	94.198	59.96	125.35
b20	494	2	mod+rem	358	0.852	148	210	11	20	1	1	89.861	94.251	411.63	125.35
spi	229	0	initial	512	1.000	500	12	0	0	0	0	90.491	99.985	1.00	10.20
spi	229	0	reorder	508	0.992	496	12	0	0	0	0	90.491	99.985	1.92	10.20
spi	229	8	mod+rem	469	0.916	458	11	26	2	0	0	90.491	99.985	6491.50	10.20
tv80	359	0	initial	729	1.000	626	103	0	0	0	0	94.126	99.540	1.00	49.91
tv80	359	0	reorder	720	0.988	617	103	0	0	0	0	94.126	99.540	1.97	49.91
tv80	359	2	mod+rem	655	0.898	560	95	28	8	0	3	94.164	99.546	1889.21	49.91
tv80	359	5	mod+rem	642	0.881	553	89	34	9	6	3	94.164	99.546	4918.00	49.91

terminates after considering all the bits of t_{mod} a constant number of times. The constant four is used for the implementation in this paper.

In the worst case, with n tests in T, the procedure considers $O(n^2)$ pairs of tests. For every pair it performs the equivalent of test generation for the faults that the pair detects. Several effects reduce the computational effort relative to the worst case.

1) Only a subset of the tests have $d_i = 1$. Only these tests are used as t_{rem} and only the remaining tests are used as t_{mod}.
2) The number of tests is reduced as tests are removed.
3) Test generation is avoided in the implementation described in this paper by using bitwise complementation to modify a test.

The computational effort is reduced further by noting that tests at the beginning of the test set typically detect large numbers of faults and cannot be modified to detect additional faults. Thus, instead of considering $t_{\text{mod}} = t_{\text{rem}-1}, \ldots, 0$, the procedure considers $t_{\text{mod}} = t_{\text{rem}-1}, \ldots, t_{\text{HI}}$, where d_{HI} makes it unlikely that an earlier test will be removed or modified.

V. EXPERIMENTAL RESULTS

The test compaction procedure is applied to benchmark circuits as described in this section.

A skewed-load test set for transition faults is generated using the test compaction procedure from [12]. A single-cycle test set for stuck-at faults is generated using the test compaction procedure from [13]. Compacting the combined test set when the individual test sets are already compact is expected to require a nontrivial run time.

The test reordering procedure, and the test modification and removal procedure, are applied iteratively in this order until no further reductions in the numbers of tests are obtained.

The test modification and removal procedure is applied with $t_{\text{HI}} = n/4$ based on the experimental observation that tests in the top quarter of the test set typically cannot be modified to detect additional faults.

The results are shown in Table II as follows. Column *sv* shows the number of state variables. Column *iter* shows the iteration of the procedure. The value 0 is reserved for the following test sets.

1) The initial test set, after single-cycle tests are added to the transition fault test set, and transformed into skewed-load tests.
2) The reordered test set where tests for stuck-at faults are placed before tests for transition faults.

Values that are larger than 0 under column *iter* correspond to iterations where the test reordering procedure, and the test modification and removal procedure, are applied.

The name of the procedure applied to T is shown under column *proc*, where *initial* stands for the initial test set, *reorder* for test reordering, and *mod* + *rem* for test modification and removal.

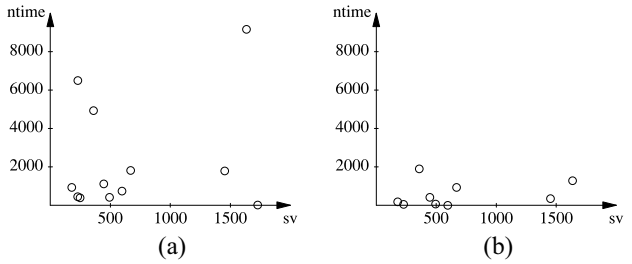

Fig. 3. Normalized run time. (a) Termination. (b) Reduction to 0.9.

Table II reports on: 1) iteration 0; 2) the cases where the number of tests in T is reduced to approximately $0.9, 0.8, \ldots$ of its initial value (this may occur during an iteration); and 3) the final test set.

For every test set, column *tests* subcolumn *tot* shows the total number of tests in T. Subcolumn *frac* shows the number of tests as a fraction of the number of tests in the initial test set. Subcolumn *tr* shows the number of tests in T that were generated for transition faults. Subcolumn *sa* shows the number of tests in T that were generated for stuck-at faults.

Column *rem*, *mod* shows the numbers of pairs $t_{\rm rem}, t_{\rm mod}$ such that the modification of $t_{\rm mod}$ allowed $t_{\rm rem}$ (and possibly additional tests) to be removed by the test modification and removal procedure. The cases are partitioned into cases where $\pi_{\rm rem}, \pi_{\rm mod} =$ TR, TR; TR, SA; SA, TR; and SA, SA.

Column $f.c.$ shows the transition fault coverage of the test set followed by its stuck-at fault coverage.

Column *ntime* shows the cumulative normalized run time of the procedure. For normalization, the run time is divided by the fault simulation time of the initial test set with fault dropping. Fault simulation time is used for normalization since the test compaction procedure is implemented using only fault simulation with fault dropping. The fault simulation procedure that is used for the implementation of the test compaction procedure is not efficient and does not include any of the commonly used parallel simulation techniques. The fault simulation time in seconds on a Linux machine with a 3 GHz processor is shown in column *fstime*.

The following points can be seen from Table II. The test compaction procedure typically reduces the number of tests significantly compared with the initial test set. The reduction is typically higher when more tests for stuck-at faults are included in the initial test set. This is because each one of the test sets separately is compact. If fewer stuck-at tests are added to the transition fault test set, there are also fewer opportunities for further compaction.

Most of the reduction in the number of tests is achieved by the test modification and removal procedure. Nevertheless, the test reordering procedure is important for placing the tests in positions where they can be modified and removed more easily.

In iteration 0, the reordering procedure removes tests that were generated for transition faults because they become unnecessary when tests that were generated for stuck-at faults are placed ahead of them in the test set. This reduction is made possible by replacing single-cycle tests for stuck-at faults with skewed-load tests.

The test modification and removal procedure is able to remove additional tests that were generated for transition faults. There are many cases where tests for stuck-at faults are modified to detect transition faults. Again, this reduction is made possible by replacing single-cycle tests for stuck-at faults with skewed-load tests.

Fig. 3 shows the normalized run time of the procedure as a function of the number of state variables, which is used for measuring the size of a circuit. Every circuit from Table II has a circle in Fig. 3(a) that shows the number of state variables on the horizontal axis, and the normalized run time at termination on the vertical axis. Fig. 3(b) shows the normalized run time when the number of tests decreases to approximately 0.9 of its initial value, if such a point exists. Overall, Fig. 3 lacks a clear trend of how the normalized run time changes with the size of the circuit. In particular, the normalized run time does not increase with the size of the circuit. This implies that the procedure scales similar to a fault simulation procedure. Furthermore, based on Table II and Fig. 3, it is typically possible to terminate the procedure earlier with a small loss in the level of test compaction, and a significantly lower normalized run time.

VI. Conclusion

This paper made several observations that can be utilized to produce a single compact test set that consists only of skewed-load tests for both transition and stuck-at faults.

1) A single-cycle test for a stuck-at fault can be transformed into a two-cycle skewed-load test that is guaranteed to detect the stuck-at fault without performing logic or fault simulation.
2) Skewed-load tests, which are transformed from single-cycle tests for stuck-at faults, sometimes detect more transition and stuck-at faults than tests that were generated for transition faults. This was used for reordering the test set, contributing to test compaction as well as the effectiveness of the third observation.
3) A static test compaction procedure, which is based on the modification and removal of tests, is effective in this context because it allows tests for stuck-at faults to detect more transition faults, and tests for transition faults to detect more stuck-at faults.

This paper described a test compaction procedure based on these observations. Experimental results were presented for benchmark circuits to demonstrate the effectiveness of the procedure.

References

[1] L. N. Reddy, I. Pomeranz, and S. M. Reddy, "COMPACTEST-II: A method to generate compact two-pattern test sets for combinational logic circuits," in *Proc. Int. Conf. Comput.-Aided Design*, 1992, pp. 568–574.
[2] R. Desineni, K. N. Dwarkanath, and R. D. Blanton, "Universal test generation using fault tuples," in *Proc. Int. Test Conf.*, 2000, pp. 812–819.
[3] S. M. Reddy *et al.*, "A unified fault model and test generation procedure for interconnect opens and bridges," in *Proc. Eur. Test Symp.*, 2005, pp. 22–27.
[4] S. Goel and R. A. Parekhji, "Choosing the right mix of at-speed structural test patterns: Comparisons in pattern volume reduction and fault detection efficiency," in *Proc. Asian Test Symp.*, 2005, pp. 330–336.
[5] S. Alampally, R. T. Venkatesh, P. Shanmugasundaram, R. A. Parekhji, and V. D. Agrawal, "An efficient test data reduction technique through dynamic pattern mixing across multiple fault models," in *Proc. VLSI Test Symp.*, 2011, pp. 285–290.
[6] C.-H. Wu and K.-J. Lee, "Transformation of multiple fault models to a unified model for ATPG efficiency enhancement," in *Proc. Int. Test Conf.*, 2016, pp. 1–10.
[7] J. Savir and S. Patil, "Scan-based transition test," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 12, no. 8, pp. 1232–1241, Aug. 1993.
[8] J. Savir and S. Patil, "Broad-side delay test," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 8, pp. 1057–1064, Aug. 1994.
[9] I. Pomeranz, "Static test compaction for scan circuits by using restoration to modify and remove tests," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 12, pp. 1955–1964, Dec. 2014.
[10] N. Ahmed, M. Tehranipoor, C. P. Ravikumar, and K. M. Butler, "Local at-speed scan enable generation for transition fault testing using low-cost testers," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 5, pp. 896–905, May 2007.
[11] G. Xu and A. D. Singh, "Scan cell design for launch-on-shift delay tests with slow scan enable," *IET Comput. Digit. Techn.*, vol. 1, no. 3, pp. 213–219, May 2007.
[12] I. Pomeranz, "Static test compaction for delay fault test sets consisting of broadside and skewed-load tests," in *Proc. VLSI Test Symp.*, 2011, pp. 84–89.
[13] I. Pomeranz and S. M. Reddy, "Forming N-detection test sets from one-detection test sets without test generation," in *Proc. Int. Test Conf.*, Nov. 2005, pp. 527–535.