Increasing the Fault Coverage of a Truncated Test Set

IRITH POMERANZ, Purdue University

Defect-aware, cell-aware, and gate-exhaustive faults are described by input patterns of subcircuits or cells that are expected to activate defects. Even with single-cycle faults, an n-input subcircuit can have up to 2^n faults with unique fault detection conditions, resulting in a large test set. Such a test set may have to be truncated to fit in the tester memory or satisfy constraints on test application time. In this case, a loss of fault coverage is inevitable. This article considers the test set denoted by T_1 obtained after truncating a larger test set denoted by T_0. Suppose that the truncation reduces the set of detected faults from the set denoted by D_0 to the set denoted by D_1. The procedure described in this article modifies the tests in T_1 to gain the detection of faults from $D_0 \backslash D_1$, even at the cost of losing the detection of faults from D_1. The goal is to reduce the fault coverage loss by computing a test set denoted by T_2 that detects a set of faults denoted by D_2 such that $|T_2| = |T_1|$ and $|D_2| > |D_1|$. Experimental results for benchmark circuits demonstrate the ability of the procedure to increase the coverage of gate-exhaustive faults over several iterations.

CCS Concepts: • **Hardware → Test-pattern generation and fault simulation;**

Additional Key Words and Phrases: Gate-exhaustive faults, test compaction, test generation, stuck-at faults

ACM Reference format:
Irith Pomeranz. 2022. Increasing the Fault Coverage of a Truncated Test Set. *ACM Trans. Des. Autom. Electron. Syst.* 27, 6, Article 54 (June 2022), 16 pages.
https://doi.org/10.1145/3508459

1 INTRODUCTION

New technologies give rise to new types of defects [1–7] that can be modeled as defect-aware [8, 9], cell-aware [10–14] or gate-exhaustive [15–19] faults. These types of faults are described by input patterns of subcircuits or cells. The input patterns are such that they are expected to activate defects. In the case of defect-aware and cell-aware faults, the input patterns are selected by analyzing the layout of the subcircuit. In the case of gate-exhaustive faults, all of the input patterns are considered important. A fault associated with an input pattern p_i of a subcircuit G_i is detected by assigning p_i to the inputs of the gate and propagating a fault effect from the output of G_i to an observable output.

Even with single-cycle faults, an n-input subcircuit can have up to 2^n faults. The faults have unique detection conditions since they are associated with different input patterns of the subcircuit. As a result, a large test set may be obtained. In this case, the test set may have to be truncated to fit in the tester memory or satisfy constraints on the test application time. Before truncation,

This work was supported in part by SRC grant 2020-CT-2967.
Authors' address: I. Pomeranz, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA; email: pomeranz@ecn.purdue.edu.

1084-4309/2022/06-ART54 $15.00
https://doi.org/10.1145/3508459

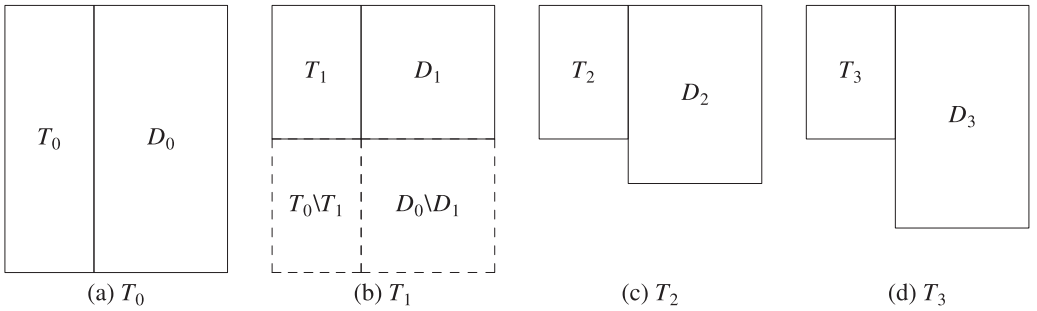

Fig. 1. Test set truncation.

a test generation procedure may achieve complete or close-to-complete fault coverage for the fault model(s) of interest. However, the number of tests in the test set may be excessive even if test compaction is used [20–23]. It is then necessary to truncate the test set. Truncation reduces the number of tests and, along with it, the fault coverage. As a result, the fault coverage after truncation is not complete. The procedures described in [24, 25] reorder a test set before it is truncated to steepen its fault coverage curve. With a steeper fault coverage curve, a higher fault coverage is obtained after truncation. However, a loss of fault coverage is inevitable when a test set is truncated.

This article considers the test set obtained after truncation. Suppose that a test set T_0 is truncated into a test set T_1. Suppose further that the set of detected faults is reduced from D_0 to D_1. This is illustrated by Figures 1(a) and 1(b). With the process from [24, 25], $T_1 \subset T_0$. Therefore, D_1 is limited by the tests included in T_0. To remove this limitation, the procedure described in this article modifies the tests in T_1 to gain the detection of faults from $D_0 \setminus D_1$, even at the cost of losing the detection of faults from D_1. The goal of the procedure is to obtain a test set T_2 that detects a set of faults D_2 such that $|T_2| = |T_1|$ and $|D_2| > |D_1|$. This is illustrated by Figure 1(c). Thus, the goal is to reduce the fault coverage loss that occurred when the test set was truncated by modifying the tests. In an iterative process, the procedure produces test sets T_k, for $k \geq 2$, such that $|T_k| = |T_{k-1}|$ and $|D_k| > |D_{k-1}|$, with reduced fault coverage loss. Figure 1(d) shows the test set T_3 and the set of detected faults D_3. Since the number of tests in T_0 is determined by a test-generation procedure, and the number of tests in T_k, for $k \geq 1$, is determined by constraints on the tester memory or test application time, the procedure described in this article does not attempt to change the number of tests in T_k.

The procedure is applied to a compact test set for single stuck-at faults and single-cycle gate-exhaustive faults. Truncation is performed after the test set is reordered using a procedure similar to the one from [25]. Truncation is done such that the stuck-at fault coverage is maintained. However, the single-cycle gate-exhaustive fault coverage decreases inevitably when the number of tests is reduced. The procedure described in this article is applied to increase the single-cycle gate-exhaustive fault coverage of the truncated test set.

Gate-exhaustive faults are interesting for this study since the number of faults is large, and many of them are undetectable. In addition, there are large variations among different circuits in the percentage of tests that need to be added to a stuck-at test set to detect gate-exhaustive faults. Since it is not known in advance how many tests will be needed, tests are generated for all of the faults. Truncation may then remove different percentages of tests for different circuits and result in a different loss of fault coverage for different circuits. This allows the procedure described in this article to be studied over a large range of truncated test sets. The same procedure is applicable with other fault models and when several different fault models are targeted [26–30]. To demonstrate the broader applicability of the procedure, it is applied to a randomly selected subset of all of the

gate-exhaustive faults. Cell-aware faults also constitute a subset of gate-exhaustive faults when the same cells or gates are used.

For further illustration, the procedure is applied to a compressed launch-on-shift test set that targets single stuck-at, transition, single-cycle, and two-cycle gate-exhaustive faults. In this case, truncation is performed such that it maintains the stuck-at and transition fault coverage. However, the coverage of gate-exhaustive faults of both types may be reduced. For the description of the algorithm developed in this article, single-cycle tests for stuck-at and single-cycle gate-exhaustive faults are considered as a concrete example. Compressed two-cycle tests for four fault models are considered later.

The article is organized as follows. Section 2 describes the problem addressed in this article. Section 3 provides an overview of the procedure used for solving it. Section 4 provides details of its implementation. Section 5 presents experimental results for benchmark circuits considering single-cycle tests. Section 6 discusses experimental results for compressed two-cycle tests. Section 7 presents concluding remarks.

2 PROBLEM FORMULATION

The procedure described in this article accepts a test set T_0 that detects a set of faults D_0. The set D_0 consists of single-cycle gate-exhaustive faults as well as single stuck-at faults. The procedure truncates T_0 to obtain the test set T_1 and then modifies T_1 to obtain the test set T_2. Truncation of T_1 and modification of T_2 are performed such that T_1, and then T_2, detect all of the single stuck-at faults in D_0. This is justified by the fact that complete coverage of single stuck-at faults is a minimum requirement for testing a circuit.

Fault simulation with fault dropping of D_0 under T_0 yields, for every test $t_i \in T_0$, a set of detected stuck-at faults denoted by $D_{sa}(t_i)$, and a set of detected gate-exhaustive faults denoted by $D_{ge}(t_i)$. Before truncating T_0, its tests are ordered from high to low size of $D_{sa}(t_i)$. For the same size of $D_{sa}(t_i)$, the tests are ordered from high to low size of $D_{ge}(t_i)$. With this order, the first tests in T_0 detect all of the single stuck-at faults. The remaining tests detect only gate-exhaustive faults. In both cases, the tests are ordered such that tests appearing earlier detect more faults.

After reordering the tests, fault simulation is carried out again to update the sets $D_{sa}(t_i)$ and $D_{ge}(t_i)$. Reordering and fault simulation are repeated until no further changes to the order of the tests are obtained.

Considering the ordered test set T_0, and a parameter $0 < \theta < |T_0|$, a truncated test set T_1 includes the first θ tests from T_0. Fault simulation with fault dropping of D_0 under T_1 yields a set $D_1 \subset D_0$ of detected faults. The value of θ is selected such that D_1 includes all the single stuck-at faults from D_0.

Figures 1(a) and (b) demonstrate the truncation of T_0 into T_1. The test set T_1 excludes the tests in $T_0 \backslash T_1$ to reduce the number of tests. As a result, it loses the detection of the faults in $D_0 \backslash D_1$. Reordering of T_0 ensures that T_1 loses the detection of as few faults as possible. However, a loss in fault coverage is inevitable when the test set is truncated. In addition, the selection of T_1 is performed without modifying the tests in T_0. Even though T_0 is optimized for the detection of D_0, the truncated test set T_1 may not be optimized for the detection of D_1. Thus, with the same number of tests, it may be possible to detect more faults if different tests are used.

The possibility of detecting more faults is illustrated by Figure 1(c). The test set T_2 has the same size as T_1. It detects a set of faults $D_2 \subset D_0$ that is larger than D_1. The computation of T_2 is the goal of the procedure described in this article. The same procedure is applied iteratively to produce test sets T_3, T_4, ... with increased sets of detected faults. Figure 1(d) illustrates the case of T_3.

Problem Formulation: Given a test set T_0 that detects a set of faults D_0, and a truncated test set T_1 that detects a set of faults $D_1 \subset D_0$, compute a test set T_2 with a set of detected faults $D_2 \subseteq D_0$ such that $|T_2| = |T_1|$ and $|D_2| > |D_1|$.

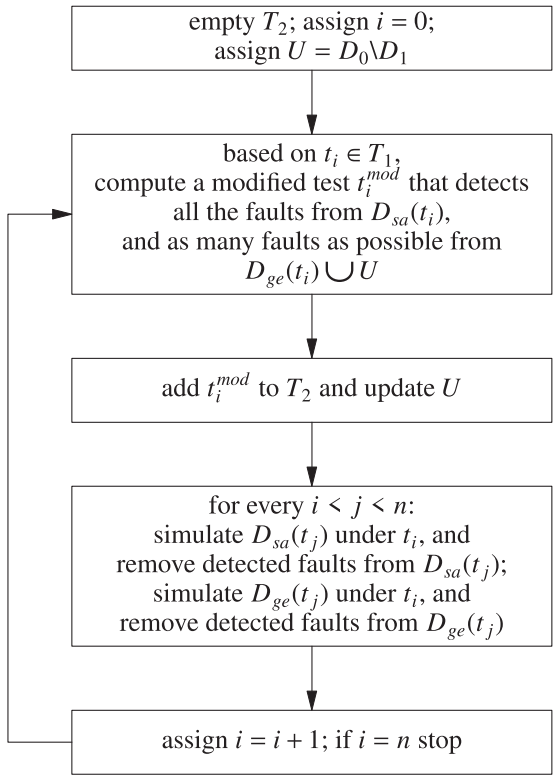

Fig. 2. Procedure for increasing the fault coverage after truncation.

3 PROCEDURE OVERVIEW

The procedure described in this article uses the gate-exhaustive faults in $D_0 \backslash D_1$ as targets for the modification of the tests in T_1. The goal is to increase the numbers of gate-exhaustive faults the tests in T_1 detect. In this process, the procedure accepts losing the detection of gate-exhaustive faults from D_1 if it can detect gate-exhaustive faults from $D_0 \backslash D_1$ in their place such that the number of detected gate-exhaustive faults is increased. This proceeds as illustrated by Figure 2 and described next.

3.1 Procedure

Let $T_1 = \{t_0, t_1, \ldots, t_{n-1}\}$. Fault simulation with fault dropping yields sets of detected faults $D_{sa}(t_i)$ and $D_{ge}(t_i)$ for every test $t_i \in T_1$. The procedure constructs a test set T_2 by considering the tests from T_1 one by one. Every test is modified before it is added to T_2, as described next.

Initially, $T_2 = \emptyset$, and $U = D_0 \backslash D_1$ includes the gate-exhaustive faults that are detected by T_0 but not by T_1. When a test $t_i \in T_1$ is considered, the goal of the procedure is to compute a new test t_i^{mod} that satisfies the following conditions.

(1) t_i^{mod} detects all of the faults from $D_{sa}(t_i)$. This is important for ensuring that all of the single stuck-at faults detected by T_0 (and T_1) are also detected by T_2.

(2) t_i^{mod} detects as many faults as possible from the set $D_{ge}(t_i) \cup U$. This condition allows the procedure to substitute gate-exhaustive faults in D_1 with gate-exhaustive faults in U if this results in an increase in the gate-exhaustive fault coverage of T_2. Without modification, $t_i^{mod} = t_i$ detects

the faults in $D_{ge}(t_i)$. Therefore, the minimum number of gate-exhaustive faults that t_i^{mod} will detect is $|D_{ge}(t_i)|$. By optimizing t_i^{mod}, the procedure may increase the number of detected gate-exhaustive faults, thus increasing the gate-exhaustive fault coverage of T_2 relative to T_1.

The computation of t_i^{mod} can be done by a test generation procedure targeting the faults in $D_{sa}(t_i) \cup D_{ge}(t_i) \cup U$. A different implementation is described in Section 4.

After computing t_i^{mod}, the procedure includes it in T_2. It updates U to include gate-exhaustive faults from $D_{ge}(t_i)$ whose detection was lost, and exclude new gate-exhaustive faults that t_i^{mod} detects. The updated set is $(U \cup D_{ge}(t_i)) \setminus D_{ge}(t_i^{mod})$.

Next, for $i < j < n$, the procedure simulates the faults in $D_{sa}(t_j) \cup D_{ge}(t_j)$ under t_i^{mod}. If a fault $f \in D_{sa}(t_j)$ is detected by t_i^{mod}, the procedure removes the fault from $D_{sa}(t_j)$. Similarly, if a fault $f \in D_{ge}(t_j)$ is detected by t_i^{mod}, the procedure removes the fault from $D_{ge}(t_j)$. This is important since having fewer faults in $D_{sa}(t_j)$ and $D_{ge}(t_j)$ ensures that t_j is easier to modify later.

Experimental results indicate that a test t_i close to the beginning of T_1 cannot be modified to increase $D_{ge}(t_i)$ significantly. The reason is that the sets $D_{sa}(t_i)$ and $D_{ge}(t_i)$ are large for tests that appear at the beginning of the test set, and the flexibility to modify them is low. To avoid the computational effort of attempting to modify tests that cannot be modified, the procedure attempts to modify tests starting from $t_{m/4}$, where m is the number of tests for stuck-at faults in T_1. The procedure copies $t_0, \ldots, t_{m/4-1}$ from T_1 to T_2 unmodified. It attempts to modify the tests $t_{m/4}, \ldots, t_{n-1}$.

Using the same process for $k > 2$, the procedure uses T_{k-1} to compute a modified test set T_k with an increased gate-exhaustive fault coverage. This continues until the gate-exhaustive fault coverage of T_k is equal to that of T_{k-1}.

3.2 Example

To illustrate the procedure, benchmark circuit $s1423$ is considered in Table 1. The test set T_0 contains 118 tests, including 38 tests for stuck-at faults. The test set detects 1653 gate-exhaustive faults. It is truncated into a test set that contains 76 tests and detects 1599 gate-exhaustive faults. The procedure considers the tests $t_9, t_{10}, \ldots, t_{75}$. Some of these tests are shown in Table 1. The first column of Table 1 shows the index of a test t_i. This is followed by the number of gate-exhaustive faults detected by t_i. Column *init* shows the number of faults detected by t_i initially, when t_i is first considered. Column $modify$ shows the number of faults detected by t_i after it is modified. Column *move* shows the number of faults detected by t_i after the procedure simulates under t_i faults that are detected by tests appearing later than t_i. This moves detected faults from later tests to t_i. Column $from$ shows the indices of the tests from which faults are moved to t_i.

When t_9 is considered, the procedure finds that it detects 38 gate-exhaustive faults. Modifying t_9 does not result in the detection of new faults, and no faults are moved from later tests.

When t_{10} is considered, the procedure finds that it detects 43 gate-exhaustive faults. Modifying t_{10} does not result in the detection of new faults. One fault is moved from t_{18} to t_{10}. Later, when t_{18} is considered, this allows the procedure to modify it such that it detects one new fault.

Five faults are moved from t_{21} to tests that appear earlier, one to each of t_{11}, t_{12}, and t_{17}, and two to t_{13}. When t_{21} is considered, the procedure finds that it detects four gate-exhaustive faults. Modifying t_{21} results in the detection of one new fault, and four faults are moved from later tests to t_{21}.

Additional examples are shown in Table 1. After the construction of T_2 is complete, the test set detects 1629 gate-exhaustive faults.

It is also interesting to consider the subsets of gate-exhaustive faults detected by a test t_i before and after it is modified. The test t_{65} of $s1423$ is considered in Table 2. Before the test is modified,

Table 1. Example of Computing T_2

i	init	modify	move	from
9	38	38	38	
10	43	43	44	(18)
11	31	31	32	(21)
12	42	42	43	(21)
13	24	26	28	(21)
14	35	35	35	
15	21	21	21	
16	21	21	21	
17	19	19	22	(21, 27, 41)
18	17	18	18	
19	15	17	17	
20	12	12	12	
21	4	5	9	(22, 42, 50, 66)
22	10	12	13	(34)
29	4	7	8	(46)
30	3	4	4	
33	5	7	7	
35	3	4	4	
36	2	3	3	
37	1	3	3	
42	7	8	8	
49	4	5	6	(57)
50	2	3	3	

Table 2. Example of Detected Faults

i	step	detected			
65	init	(689,10011011)	(706,10001)	(986,00011)	
	modify	(689,10011011)	(706,10001)	(986,00101)	(1421,10011)

it detects three faults. After it is modified, it detects four faults. The faults are shown in Table 2 in the format (G_a, p_b), where G_a is the index of a gate, and p_b is the input pattern assigned to the gate inputs to activate the fault.

In the case of t_{65}, to allow a new fault, (1421,10011), to be detected, the fault (986,00011) is replaced with the fault (986,00101).

4 TEST MODIFICATION

This section describes the computation of a modified test t_i^{mod} based on a test t_i with subsets of detected faults $D_{sa}(t_i)$ and $D_{ge}(t_i)$. As before, the set of gate-exhaustive faults whose detection was lost by the truncation of the test set and not recovered is denoted by U. The advantage of the procedure described in this section over test generation that targets the faults in $D_{sa}(t_i) \cup D_{ge}(t_i) \cup U$ is that it keeps t_i^{mod} as close as possible to t_i. In particular, it does not lose the detection of any fault from $D_{sa}(t_i)$ and, thus, does not need to target these faults again. In addition, it does not lose the detection of a fault from $D_{ge}(t_i)$ unless it can detect a different gate-exhaustive fault in its place. Consequently, t_i^{mod} detects as many faults from $D_{ge}(t_i)$ as possible.

The procedure starts with $t_i^{mod} = t_i$. It modifies t_i^{mod} by complementing its bits one by one. After bit b is complemented, let the test $t_{i,b}^{mod}$ be obtained. To decide whether the complementation of bit b is acceptable, the procedure performs fault simulation as follows.

(1) The procedure simulates $D_{sa}(t_i^{mod})$ under $t_{i,b}^{mod}$. If any fault is not detected, the complementation of bit b is not acceptable. This condition ensures that all of the single stuck-at faults detected by T_0 (and T_1) are also detected by T_2.

(2) The procedure simulates $D_{ge}(t_i^{mod}) \cup U$ under $t_{i,b}^{mod}$. Let the subset of detected faults be $D_{ge}(t_{i,b}^{mod})$. If $|D_{ge}(t_{i,b}^{mod})| < |D_{ge}(t_i^{mod})|$, the complementation of bit b is not acceptable. This condition ensures that the number of detected gate-exhaustive faults is not decreased by the modification.

If both checks indicate that $t_{i,b}^{mod}$ is acceptable, the procedure assigns $t_i^{mod} = t_{i,b}^{mod}$. Faults in $D_{ge}(t_{i,b}^{mod}) \cap U$ are removed from U, and faults in $D_{ge}(t_i^{mod}) \setminus D_{ge}(t_{i,b}^{mod})$ are added to U to keep the set U up-to-date. The set U is updated by assigning $U = (U \cup D_{ge}(t_i^{mod})) \setminus D_{ge}(t_{i,b}^{mod})$.

All of the bits of t_i^{mod} are considered a constant number of times. The constant four is used for the experiments. This is based on the experimental observation that fewer than four passes over the bits of a test are typically needed before no additional improvements in the subsets of detected faults can be achieved.

5 EXPERIMENTAL RESULTS FOR SINGLE-CYCLE TESTS

This section presents the results of applying the procedure for increasing the fault coverage of a truncated test set to benchmark circuits using single-cycle tests for stuck-at and single-cycle gate-exhaustive faults.

The set of gate-exhaustive faults is the one computed in [19]. In addition, in a separate experiment, a quarter of the gate-exhaustive faults from [19] are selected randomly and used as target faults. This demonstrates a case that is closer to the case in which cell-aware faults are used. In the case of cell-aware faults, assuming the same cells or gates, a subset of gate-exhaustive faults are identified as important for every gate; only these faults are included in the set of target faults.

The test set T_0 consists of a compact test set for single stuck-at faults, topped off with tests for gate-exhaustive faults from [19]. Only tests that detect additional gate-exhaustive faults are included in T_0. The set D_0 includes all of the stuck-at and gate-exhaustive faults detected by T_0.

The set of detectable gate-exhaustive faults in D_0 is taken as the universe of gate-exhaustive faults for the computation of a percentage of detected faults, which is referred to as the gate-exhaustive fault efficiency. The test set is reordered and then truncated as described in Section 2 using a parameter $0 < \theta < |T_0|$. Values of θ are selected as follows.

Let θ_0 be the number of tests in a compact test set for single stuck-at faults. After reordering, these tests appear at the beginning of T_0. For $\mu = 2$ and 3, the procedure is applied with $\theta = \mu\theta_0$. The rationale for this selection is as follows.

With $\theta = 2\theta_0$, the number of additional tests available for detecting gate-exhaustive faults is the same as the number required for detecting single stuck-at faults. The case in which $\theta = 3\theta_0$ illustrates the situation in which a larger test set can be accommodated.

For some benchmark circuits, $\theta = 2\theta_0$ results in the removal of a large percentage of tests from T_0. For other circuits, the percentage of tests removed is smaller. The use of $\theta = 3\theta_0$ results in lower percentages of tests being removed from T_0. The percentage of tests removed determines the fault coverage loss for gate-exhaustive faults. Overall, the use of $\theta = 2\theta_0$ and $3\theta_0$, and the selection of gate-exhaustive faults, result in a large range of different truncated test sets, with different fault coverages, that bring out the ability of the procedure described in this article to increase the fault coverage in different situations.

Table 3. Experimental Results $|T_1|/|T_0| < 35\%$

circuit	inp	μ	k	tests	%tests	s.a.	g.exh	gap	ntime
s38417	1664	2	1	206	12.61	99.680	80.784	0.000	12.58
s38417	1664	2	2	206	12.61	99.680	85.846	26.343	811.73
s38417	1664	2	50	206	12.61	99.680	88.409	39.682	43091.31
s38417	1664	s2	1	206	19.25	99.680	81.811	0.000	11.52
s38417	1664	s2	2	206	19.25	99.680	91.297	52.153	603.25
s38417	1664	s2	28	206	19.25	99.680	94.916	72.050	19760.49
b20	527	2	1	852	20.86	95.586	90.162	0.000	17.94
b20	527	2	2	852	20.86	95.766	90.871	7.200	270.33
b20	527	2	23	852	20.86	95.807	91.341	11.983	5702.19
s15850	611	2	1	236	21.55	97.511	83.332	0.000	9.20
s15850	611	2	2	236	21.55	97.511	85.618	13.717	339.24
s15850	611	2	32	236	21.55	97.511	88.705	32.237	10060.41
simple_spi	146	2	1	72	23.38	100.000	67.806	0.000	5.19
simple_spi	146	2	2	72	23.38	100.000	74.063	19.434	76.12
simple_spi	146	2	7	72	23.38	100.000	74.747	21.558	432.94
b14	280	2	1	706	25.35	95.966	86.763	0.000	10.65
b14	280	2	2	706	25.35	96.193	86.955	1.455	205.40
b14	280	2	6	706	25.35	96.205	87.128	2.764	984.46
s9234	247	2	1	286	28.66	93.946	85.674	0.000	13.72
s9234	247	2	2	286	28.66	93.946	87.703	14.167	192.38
s9234	247	2	14	286	28.66	93.946	88.819	21.955	2477.91
b20	527	3	1	1278	31.29	95.586	92.650	0.000	16.97
b20	527	3	2	1278	31.29	95.773	92.905	3.468	388.78
b20	527	3	13	1278	31.29	95.812	93.123	6.435	5310.94
s15850	611	3	1	354	32.33	97.511	90.083	0.000	9.32
s15850	611	3	2	354	32.33	97.511	91.042	9.673	528.24
s15850	611	3	53	354	32.33	97.511	93.833	37.818	26999.10
simple_spi	146	s2	1	72	32.43	100.000	72.138	0.000	9.86
simple_spi	146	s2	2	72	32.43	100.000	90.679	66.545	148.57
simple_spi	146	s2	6	72	32.43	100.000	93.820	77.818	818.14
i2c	145	2	1	90	33.46	100.000	88.819	0.000	5.94
i2c	145	2	2	90	33.46	100.000	91.986	28.325	71.56
i2c	145	2	12	90	33.46	100.000	92.812	35.714	822.56
sasc	132	2	1	44	34.65	100.000	74.347	0.000	5.17
sasc	132	2	2	44	34.65	100.000	79.323	19.396	135.33
sasc	132	2	6	44	34.65	100.000	79.853	21.463	720.33

The results are shown in Tables 3 to 7 as follows. The circuits are arranged from low to high value of $|T_1|/|T_0| \cdot 100$, which is the percentage of tests from T_0 included in T_1 after truncation. Each table has a different range of the percentage $|T_1|/|T_0| \cdot 100$.

There are several rows for every circuit, corresponding to T_1, T_2, and the final test set obtained by the procedure from Figure 2.

For every test set, after the circuit name, column *inp* shows the number of inputs. Column μ shows the value of this parameter. When only a quarter of the gate-exhaustive faults are considered, the value of μ is preceded by an "s." Column k shows the iteration of the procedure.

Table 4. Experimental Results $35\% \leq |T_1|/|T_0| < 45\%$

circuit	inp	μ	k	tests	%tests	s.a.	g.exh	gap	ntime
simple_spi	146	3	1	108	35.06	100.000	80.598	0.000	7.63
simple_spi	146	3	2	108	35.06	100.000	85.436	24.935	163.88
simple_spi	146	3	7	108	35.06	100.000	86.348	29.634	978.63
spi	274	2	1	804	35.34	99.992	83.994	0.000	14.72
spi	274	2	2	804	35.34	99.992	86.461	15.411	236.57
spi	274	2	23	804	35.34	99.992	88.720	29.524	4549.54
wb_dma	738	2	1	130	36.93	100.000	91.571	0.000	15.83
wb_dma	738	2	2	130	36.93	100.000	95.179	42.814	765.61
wb_dma	738	2	12	130	36.93	100.000	96.282	55.894	9745.93
b14	280	3	1	1059	38.03	95.966	89.029	0.000	10.74
b14	280	3	2	1059	38.03	96.211	89.241	1.931	299.23
b14	280	3	9	1059	38.03	96.228	89.324	2.692	2340.71
s15850	611	s2	1	236	38.94	97.511	87.190	0.000	9.51
s15850	611	s2	2	236	38.94	97.511	92.066	38.063	556.88
s15850	611	s2	13	236	38.94	97.511	96.076	69.369	6684.05
systemcaes	928	2	1	240	39.41	99.997	91.324	0.000	11.54
systemcaes	928	2	2	240	39.41	99.997	92.465	13.147	445.55
systemcaes	928	2	7	240	39.41	99.997	92.798	16.990	3251.41
s38584	1464	2	1	284	40.51	95.567	94.141	0.000	9.41
s38584	1464	2	2	284	40.51	95.567	96.558	41.261	1329.47
s38584	1464	2	20	284	40.51	95.567	97.379	55.266	31337.49
b15	483	2	1	850	41.04	98.938	88.247	0.000	14.82
b15	483	2	2	850	41.04	98.938	91.518	27.831	344.13
b15	483	2	24	850	41.04	98.938	92.840	39.083	7524.47
sasc	132	s2	1	44	42.72	100.000	79.608	0.000	6.75
sasc	132	s2	2	44	42.72	100.000	98.042	90.400	125.00
sasc	132	s2	4	44	42.72	100.000	98.858	94.400	415.00
s9234	247	3	1	429	42.99	93.946	90.849	0.000	13.82
s9234	247	3	2	429	42.99	93.946	91.988	12.451	314.57
s9234	247	3	12	429	42.99	93.946	92.463	17.639	2815.32
systemcaes	928	s2	1	240	43.56	99.997	92.152	0.000	13.49
systemcaes	928	s2	2	240	43.56	99.997	98.498	80.859	255.73
systemcaes	928	s2	8	240	43.56	99.997	99.495	93.560	2781.03

Column *tests* shows the number of tests in T_k. This number does not change with k. Column *%tests* shows the number of tests in T_k as a percentage of the number of tests in T_0, $|T_k|/|T_0| \cdot 100$.

Column *s.a.* shows the stuck-at fault coverage. Column *g.exh* shows the gate-exhaustive fault efficiency. Column *gap* measures the improvement in the gate-exhaustive fault efficiency as $\frac{|D_k|-|D_1|}{|D_0|-|D_1|} \cdot 100\%$. The denominator is the gap between D_1 and D_0. The numerator is the extent to which the gap is covered by T_k. The ratio is the percentage improvement in the gate-exhaustive fault efficiency gap. This metric has a range of 0% to 100% even when the fault efficiency achieved by D_1 is high, and the maximum improvement possible in the fault efficiency is low. For example, when the fault efficiency is increased by 1% from 50% to 51%, the improvement in the gap is $(51-50)/(100-50) \cdot 100 = 2\%$. When the fault efficiency is increased by 1% from 90% to 91%, the improvement in the gap is $(91-90)/(100-90) \cdot 100 = 10\%$.

Table 5. Experimental Results $45\% \leq |T_1|/|T_0| < 55\%$

circuit	inp	μ	k	tests	%tests	s.a.	g.exh	gap	ntime
des_area	367	2	1	232	45.05	100.000	84.062	0.000	25.05
des_area	367	2	2	232	45.05	100.000	89.429	33.677	316.13
des_area	367	2	28	232	45.05	100.000	92.541	53.202	9420.62
s5378	214	2	1	222	45.12	98.867	86.650	0.000	10.63
s5378	214	2	2	222	45.12	98.867	91.894	39.277	178.22
s5378	214	2	21	222	45.12	98.867	93.496	51.282	3543.05
b20	527	s2	1	852	47.05	95.586	90.961	0.000	17.85
b20	527	s2	2	852	47.05	95.963	92.618	18.330	517.56
b20	527	s2	21	852	47.05	96.049	94.029	33.948	9696.21
des_area	367	s2	1	234	47.46	100.000	87.078	0.000	9.56
des_area	367	s2	2	234	47.46	100.000	98.685	89.822	162.72
des_area	367	s2	9	234	47.46	100.000	99.954	99.645	1465.40
wb_dma	738	s2	1	132	47.65	100.000	92.385	0.000	8.28
wb_dma	738	s2	2	132	47.65	100.000	99.077	87.879	442.68
wb_dma	738	s2	6	132	47.65	100.000	99.667	95.623	2931.31
s38584	1464	s2	1	284	48.71	95.567	94.921	0.000	9.50
s38584	1464	s2	2	284	48.71	95.567	99.434	88.850	1223.90
s38584	1464	s2	8	284	48.71	95.567	99.832	96.690	8719.52
i2c	145	3	1	135	50.19	100.000	94.822	0.000	8.14
i2c	145	3	2	135	50.19	100.000	96.062	23.936	116.21
i2c	145	3	5	135	50.19	100.000	96.613	34.574	457.21
s13207	700	2	1	474	51.02	98.869	90.880	0.000	9.44
s13207	700	2	2	474	51.02	98.869	94.591	40.693	750.51
s13207	700	2	9	474	51.02	98.869	95.298	48.449	7042.33
i2c	145	s2	1	90	51.72	100.000	90.077	0.000	8.00
i2c	145	s2	2	90	51.72	100.000	96.031	60.000	119.09
i2c	145	s2	6	90	51.72	100.000	97.354	73.333	665.82
sasc	132	3	1	66	51.97	100.000	86.746	0.000	5.17
sasc	132	3	2	66	51.97	100.000	92.659	44.615	161.83
sasc	132	3	4	66	51.97	100.000	93.026	47.385	502.33
spi	274	3	1	1206	53.01	99.992	90.798	0.000	15.03
spi	274	3	2	1206	53.01	99.992	92.946	23.343	276.91
spi	274	3	25	1206	53.01	99.992	94.644	41.792	5770.38
b15	483	s2	1	850	53.26	98.938	90.626	0.000	12.14
b15	483	s2	2	850	53.26	98.938	96.943	67.391	295.02
b15	483	s2	20	850	53.26	98.941	99.049	89.855	5475.68

Column *ntime* shows the normalized runtime for the computation of $T_1, T_2, \ldots, T_k$. The normalized runtime is the runtime divided by the runtime for fault simulation of T_0. The runtime for T_1 includes the reordering of T_0 before it is truncated.

The following points can be seen from Tables 3 to 7. With both values of μ, and considering all or a subset of the gate-exhaustive faults, the size of T_1 as a percentage of T_0 varies significantly with the circuit. The gate-exhaustive fault efficiency of the truncated test set also varies with the circuit. Overall, Tables 3 to 7 show a wide range of truncated test sets. In several cases at the end

Table 6. Experimental Results $55\% \leq |T_1|/|T_0| < 70\%$

circuit	inp	μ	k	tests	%tests	s.a.	g.exh	gap	ntime
s9234	247	s2	1	286	55.00	93.946	89.364	0.000	14.37
s9234	247	s2	2	286	55.00	93.946	94.349	46.875	313.39
s9234	247	s2	8	286	55.00	93.946	95.394	56.696	2089.03
wb_dma	738	3	1	195	55.40	100.000	96.942	0.000	15.91
wb_dma	738	3	2	195	55.40	100.000	97.968	33.543	822.42
wb_dma	738	3	11	195	55.40	100.000	98.372	46.751	9747.42
tv80	372	2	1	1016	55.46	99.688	95.320	0.000	13.89
tv80	372	2	2	1016	55.46	99.692	96.958	34.992	196.87
tv80	372	2	35	1016	55.46	99.709	98.210	61.742	6512.72
systemcaes	928	3	1	360	59.11	99.997	94.908	0.000	11.62
systemcaes	928	3	2	360	59.11	99.997	95.683	15.231	511.46
systemcaes	928	3	11	360	59.11	99.997	96.248	26.327	7023.50
usb_phy	112	2	1	64	60.95	100.000	95.860	0.000	6.00
usb_phy	112	2	2	64	60.95	100.000	98.318	59.375	122.00
usb_phy	112	2	4	64	60.95	100.000	98.512	64.062	406.25
s5378	214	s2	1	222	60.99	98.867	90.660	0.000	9.66
s5378	214	s2	2	222	60.99	98.867	98.319	82.000	189.65
s5378	214	s2	7	222	60.99	98.867	99.626	96.000	1143.46
s13207	700	s2	1	474	61.24	98.869	92.832	0.000	10.09
s13207	700	s2	2	474	61.24	98.869	98.755	82.632	904.42
s13207	700	s2	7	474	61.24	98.869	99.774	96.842	5691.81
b15	483	3	1	1275	61.56	98.938	96.101	0.000	14.88
b15	483	3	2	1275	61.56	98.941	97.529	36.636	433.02
b15	483	3	30	1275	61.56	98.941	98.262	55.416	12260.40
b14	280	s2	1	708	63.38	95.966	90.087	0.000	10.85
b14	280	s2	2	708	63.38	96.147	91.551	14.767	360.93
b14	280	s2	12	708	63.38	96.246	92.681	26.166	3860.31
spi	274	s2	1	806	64.64	99.992	88.636	0.000	12.96
spi	274	s2	2	806	64.64	99.992	97.589	78.780	87.33
spi	274	s2	7	806	64.64	99.992	99.889	99.024	452.95
des_area	367	3	1	348	67.57	100.000	93.868	0.000	20.55
des_area	367	3	2	348	67.57	100.000	98.360	73.254	420.62
des_area	367	3	21	348	67.57	100.000	99.801	96.758	9237.82
s5378	214	3	1	333	67.68	98.867	94.819	0.000	10.72
s5378	214	3	2	333	67.68	98.867	97.511	51.952	222.10
s5378	214	3	12	333	67.68	98.867	98.849	77.778	2482.85

of Table 7, no truncation is needed with the limit on the number of tests. These cases are included for completeness.

In most of the cases, the procedure described in this article is able to increase the gate-exhaustive fault efficiency of T_1 significantly. Even when the fault efficiency achieved by T_1 is high, the increase captured by the gap coverage is large.

The normalized runtime increases with the number of iterations. Per iteration, it is similar for circuits of different sizes with different gate-exhaustive fault efficiencies. This indicates that the procedure scales similar to a fault simulation procedure.

Table 7. Experimental Results $|T_1|/|T_0| \geq 70\%$

circuit	inp	μ	k	tests	%tests	s.a.	g.exh	gap	ntime
s13207	700	3	1	711	76.53	98.869	97.840	0.000	9.21
s13207	700	3	2	711	76.53	98.869	98.934	50.655	982.02
s13207	700	3	13	711	76.53	98.869	99.528	78.166	13141.73
tv80	372	s2	1	1016	80.70	99.688	97.234	0.000	13.85
tv80	372	s2	2	1016	80.70	99.702	99.112	67.895	162.86
tv80	372	s2	11	1016	80.70	99.719	99.782	92.105	1503.57
tv80	372	3	1	1524	83.19	99.688	98.996	0.000	13.93
tv80	372	3	2	1524	83.19	99.705	99.316	31.884	266.10
tv80	372	3	12	1524	83.19	99.709	99.589	59.058	2907.29
usb_phy	112	s2	1	64	85.33	100.000	98.705	0.000	6.33
usb_phy	112	s2	2	64	85.33	100.000	100.000	100.000	103.33
systemcdes	320	2	1	156	89.14	100.000	99.646	0.000	18.80
systemcdes	320	2	2	156	89.14	100.000	100.000	100.000	280.84
aes_core	788	2	1	416	89.46	100.000	99.741	0.000	28.41
aes_core	788	2	2	416	89.46	100.000	100.000	100.000	293.94
usb_phy	112	3	1	96	91.43	100.000	99.612	0.000	8.00
usb_phy	112	3	2	96	91.43	100.000	99.935	83.333	199.67
aes_core	788	s2	1	416	95.41	100.000	99.928	0.000	19.65
aes_core	788	s2	2	416	95.41	100.000	100.000	100.000	183.65
s35932	1763	2	1	40	100.00	89.781	100.000	-	5.69
s35932	1763	3	1	40	100.00	89.781	100.000	-	5.77
s35932	1763	s2	1	39	100.00	89.781	100.000	-	6.81
systemcdes	320	3	1	175	100.00	100.000	100.000	-	15.93
systemcdes	320	s2	1	150	100.00	100.000	100.000	-	16.16
aes_core	788	3	1	465	100.00	100.000	100.000	-	28.33

When the number of iterations is large, it is typically possible to terminate the procedure after a smaller number of iterations with a significantly reduced runtime and a small loss in fault efficiency. This can be seen from the increase in the fault efficiency achieved in iteration $k = 2$ compared with the final iteration. To demonstrate this point more clearly, the results of all of the iterations for $s38417$ with $\mu = 2$ are shown in Figure 3. There is a circle in Figure 3 for every iteration, showing the normalized runtime divided by 1000 on the horizontal axis, and the gate-exhaustive fault efficiency on the vertical axis. Figure 3 demonstrates that an earlier termination condition is possible with a small loss in fault efficiency.

6 EXPERIMENTAL RESULTS FOR COMPRESSED LAUNCH-ON-SHIFT TESTS

This section presents the results of applying the procedure for increasing the fault coverage of a truncated test set to benchmark circuits using compressed launch-on-shift tests for stuck-at, transition, single-cycle and two-cycle gate-exhaustive faults.

The set of single-cycle gate-exhaustive faults is the same as the one used in Section 5. To define two-cycle gate-exhaustive faults, the gate and input pattern from every single-cycle gate-exhaustive fault is used for the second pattern of one or more faults. Every input pattern of the same gate that differs from the second pattern in the value of a single input is used as a first pattern. The resulting two-cycle gate-exhaustive fault is considered if the two input patterns yield different values on the output of the gate.

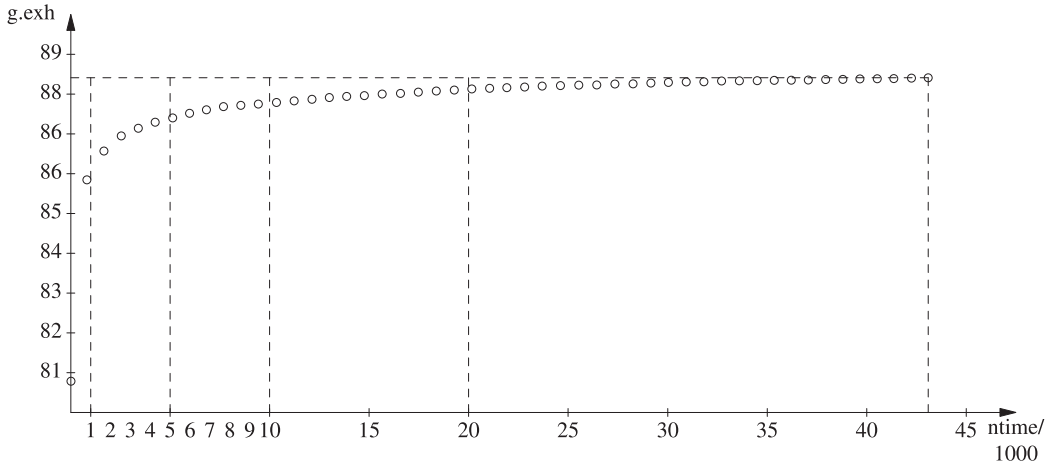

Fig. 3. Results for $s38417$ with $\mu = 2$.

The test set T_0 targets the four fault models. The detected faults are included in D_0. The remaining single-cycle and two-cycle gate-exhaustive faults are removed from consideration.

When T_0 is truncated, the test set T_1 detects all of the stuck-at and transition faults detected by T_0. A loss of fault coverage occurs for single-cycle and two-cycle gate-exhaustive faults. The number of tests in T_1 is determined using $\mu = 2$, with θ_0 being the number of tests in a compact test set for single stuck-at and transition faults.

When T_1 is modified to increase the set D_1 of faults that it detects, the procedure considers single-cycle and two-cycle gate-exhaustive faults together. Thus, it accepts a modification that detects more faults of one type and fewer of the other type if the overall number of detected faults is increased. To capture the overall effect, a combined fault efficiency is computed for gate-exhaustive faults of both types.

The tests in T_0 are launch-on-shift tests that are compressed into seeds for an *LFSR*. To modify tests and increase their sets of detected faults, the procedure complements bits of the seeds that produce them. Thus, the number of bits that need to be considered is significantly smaller than the number of inputs and equal to the number of *LFSR* bits. Complementing bits of the compressed tests, instead of complementing bits of the applied tests, ensures that the tests remain applicable under the same test data compression approach.

The results are presented in Table 8. The format is similar to that of Tables 3 to 7. Column *LFSR* shows the number of *LFSR* bits used for compressed tests. The fault coverage metrics reported in Table 8 are the following. Column $s.a.$ shows the stuck-at fault coverage. Column $trans$ shows the transition fault coverage. Column $g.exh1$ shows the single-cycle gate-exhaustive fault efficiency. Column $g.exh2$ shows the two-cycle gate-exhaustive fault efficiency. Column $g.exh12$ shows the combined single-cycle and two-cycle gate-exhaustive fault efficiency. Column gap measures the improvement in the gate-exhaustive fault efficiency considering both types of faults.

Table 8 shows improvements in the gap coverage that are similar to those in Tables 3 to 7.

To increase the gate-exhaustive fault efficiency considering both types of gate-exhaustive faults together, the procedure may allow the fault efficiency for single-cycle faults to decrease while increasing the fault efficiency for two-cycle faults. To avoid this effect, it is possible to consider the two fault types separately and require that the number of detected faults would not decrease for either one of them as compressed tests are modified.

Table 8. Experimental Results for Four Fault Models and Compressed Tests

circuit	inp	LFSR	μ	k	tests	%tests	s.a.	trans	g.exh1	g.exh2	g.exh12	gap	ntime
b14	280	128	2	1	978	42.65	96.263	95.390	92.381	88.240	90.486	0.000	7.24
b14	280	128	2	2	978	42.65	96.269	95.390	91.795	95.534	93.506	31.745	74.22
b14	280	128	2	25	978	42.65	96.269	95.390	91.828	99.289	95.243	50.000	1550.84
s15850	611	57	2	1	852	48.44	97.511	93.052	91.308	97.251	93.686	0.000	8.48
s15850	611	57	2	2	852	48.44	97.511	93.052	90.848	98.999	94.109	6.706	14.72
s15850	611	57	2	17	852	48.44	97.511	93.052	92.213	99.609	95.172	23.542	118.13
spi	274	44	2	1	1468	50.73	99.992	98.112	92.154	77.677	87.450	0.000	15.06
spi	274	44	2	2	1468	50.73	99.992	98.112	90.878	87.866	89.899	19.515	24.59
spi	274	44	2	27	1468	50.73	99.992	98.112	92.307	93.155	92.582	40.896	257.26
s5378	214	36	2	1	630	53.16	98.867	93.022	98.520	74.765	91.173	0.000	11.41
s5378	214	36	2	2	630	53.16	98.867	93.022	98.208	77.515	91.808	7.195	20.06
s5378	214	36	2	22	630	53.16	98.867	93.022	98.894	84.267	94.370	36.220	192.40
s13207	700	47	2	1	1050	54.63	98.869	90.599	96.796	88.042	94.976	0.000	10.22
s13207	700	47	2	2	1050	54.63	98.869	90.599	96.863	90.364	95.512	10.666	17.04
s13207	700	47	2	21	1050	54.63	98.869	90.599	98.273	93.308	97.241	45.083	146.64
tv80	372	109	2	1	1702	57.81	99.723	95.612	98.849	84.954	93.156	0.000	11.78
tv80	372	109	2	2	1702	57.81	99.723	95.612	98.470	88.008	94.184	15.017	34.19
tv80	372	109	2	76	1702	57.81	99.730	95.633	98.820	94.805	97.175	58.718	1925.82
usb_phy	112	18	2	1	182	58.90	100.000	97.395	100.000	86.567	94.291	0.000	8.20
usb_phy	112	18	2	2	182	58.90	100.000	97.395	99.805	89.289	95.336	18.301	24.50
usb_phy	112	18	2	11	182	58.90	100.000	97.395	99.870	94.118	97.425	54.902	159.20
i2c	145	43	2	1	304	62.94	100.000	96.247	98.923	84.002	93.292	0.000	10.28
i2c	145	43	2	2	304	62.94	100.000	96.247	97.790	90.474	95.029	25.897	41.02
i2c	145	43	2	13	304	62.94	100.000	96.270	98.011	95.807	97.179	57.949	364.21
s9234	247	75	2	1	710	65.80	93.946	87.736	95.770	99.312	96.936	0.000	15.25
s9234	247	75	2	2	710	65.80	93.946	87.736	95.685	99.656	96.992	1.847	41.63
s9234	247	75	2	6	710	65.80	93.946	87.752	95.721	99.853	97.081	4.749	152.35
sasc	132	13	2	1	182	65.94	100.000	99.543	98.775	88.526	95.911	0.000	7.85
sasc	132	13	2	2	182	65.94	100.000	99.543	98.530	91.053	96.440	12.950	18.54
sasc	132	13	2	11	182	65.94	100.000	99.543	98.857	93.263	97.293	33.813	108.69
simple_spi	146	38	2	1	248	68.32	100.000	97.906	94.135	94.602	94.245	0.000	7.59
simple_spi	146	38	2	2	248	68.32	100.000	97.906	94.557	95.458	94.770	9.123	36.85
simple_spi	146	38	2	15	248	68.32	100.000	97.906	95.535	97.258	95.941	29.474	373.48
b20	527	119	2	1	1328	71.09	95.675	94.194	98.591	99.336	98.877	0.000	15.10
b20	527	119	2	2	1328	71.09	95.677	94.196	98.532	99.711	98.985	9.607	47.82
b20	527	119	2	13	1328	71.09	95.693	94.215	98.609	99.894	99.103	20.087	388.14
systemcdes	320	14	2	1	438	73.74	100.000	99.867	100.000	96.292	98.479	0.000	9.29
systemcdes	320	14	2	2	438	73.74	100.000	99.867	99.978	96.387	98.505	1.717	12.34
systemcdes	320	14	2	10	438	73.74	100.000	99.867	99.978	96.785	98.668	12.446	48.52
wb_dma	738	47	2	1	536	75.49	100.000	98.496	99.434	91.216	96.720	0.000	9.65
wb_dma	738	47	2	2	536	75.49	100.000	98.496	98.996	93.801	97.280	17.083	24.60
wb_dma	738	47	2	28	536	75.49	100.000	98.526	99.241	96.450	98.319	48.752	377.57
b15	483	113	2	1	1564	75.78	98.915	96.289	98.976	98.866	98.941	0.000	12.94
b15	483	113	2	2	1564	75.78	98.915	96.297	98.899	99.351	99.044	9.718	46.39
b15	483	113	2	21	1564	75.78	98.918	96.311	98.956	99.832	99.236	27.887	858.86
aes_core	788	28	2	1	2002	78.54	100.000	99.251	100.000	97.541	98.927	0.000	17.73
aes_core	788	28	2	2	2002	78.54	100.000	99.251	100.000	97.646	98.973	4.269	20.15
aes_core	788	28	2	38	2002	78.54	100.000	99.251	100.000	98.630	99.402	44.269	152.32
s38584	1464	98	2	1	1360	81.49	95.567	90.417	100.000	96.870	98.720	0.000	17.80
s38584	1464	98	2	2	1360	81.49	95.567	90.417	99.829	97.984	99.075	27.710	26.41
s38584	1464	98	2	43	1360	81.49	95.567	90.417	99.949	99.536	99.780	82.822	400.18
s38417	1664	111	2	1	2088	85.36	99.680	98.200	99.283	99.560	99.378	0.000	23.07
s38417	1664	111	2	2	2088	85.36	99.680	98.200	99.201	99.848	99.424	7.301	32.14
s38417	1664	111	2	25	2088	85.36	99.680	98.200	99.465	99.972	99.640	42.035	298.86
systemcaes	928	29	2	1	750	91.35	99.997	99.519	99.771	98.996	99.501	0.000	12.97
systemcaes	928	29	2	2	750	91.35	99.997	99.519	99.728	99.317	99.585	16.822	22.24
systemcaes	928	29	2	17	750	91.35	99.997	99.519	99.875	99.525	99.753	50.467	179.52
des_area	367	14	2	1	736	96.08	100.000	99.765	99.891	99.795	99.866	0.000	10.89
s35932	1763	13	2	1	240	97.56	89.781	88.666	100.000	99.980	99.990	0.000	14.07
s35932	1763	13	2	2	240	97.56	89.781	88.666	100.000	99.987	99.993	33.333	15.31

7 CONCLUDING REMARKS

This article considered a test set T_1 obtained after truncating a larger test set T_0 to fit tester memory limits or satisfy constraints on test application time. Truncation causes the set of detected faults to decrease from D_0 to D_1. Without modifying the tests in T_0, the set D_1 is limited by the tests included in T_0. The procedure described in this article modifies the tests in T_1 to gain the detection of faults from $D_0 \backslash D_1$, even at the cost of losing the detection of faults from D_1. The goal is to obtain a test set T_2 that detects a set of faults D_2 such that $|T_2| = |T_1|$ and $|D_2| > |D_1|$. In an iterative process, the procedure produces test sets T_k, for $k \geq 2$, such that $|T_k| = |T_{k-1}|$ and $|D_k| > |D_{k-1}|$. Experimental results were presented for gate-exhaustive faults in benchmark circuits to demonstrate the ability of the procedure to increase the fault coverage of a truncated test set.

REFERENCES

[1] A. Sreedhar, A. Sanyal, and S. Kundu. 2008. On modeling and testing of lithography related open faults in Nano-CMOS circuits. In *Proc. Design, Autom. & Test in Europe Conf.* ACM, 616–621.

[2] M. O. Simsir, A. Bhoj, and N. K. Jha. 2010. Fault modeling for FinFET circuits. In *Proc. Intl. Symp. on Nanoscale Architectures.* 41–46.

[3] J. Zha, X. Cui, and C. L. Lee. 2012. Modeling and testing of interference faults in the nano NAND flash memory. In *Proc. Design, Automation & Test in Europe Conf.* IEEE, 527–531.

[4] S.-Y. Huang, Y.-H. Lin, K.-H. Tsai, W.-T. Cheng, S. Sunter, Y.-F Chou, and D.-M. Kwai. 2012. Small delay testing for TSVs in 3-D ICs. In *Proc. Design Automation Conf.* 1031–1036.

[5] S. Di Carlo, G. Gambardella, P. Prinetto, D. Rolfo, and P. Trotta. 2015. SATTA: A self-adaptive temperature-based TDF awareness methodology for dynamically reconfigurable FPGAs. In *ACM Trans. on Reconfigurable Technology and Systems* 8, 1 (2015), Article 1, 1–22.

[6] H. G. Mohammadi, P.-E. Gaillardon, and G. De Micheli. 2015. Fault modeling in controllable polarity silicon nanowire circuits. In *Proc. Design, Automation & Test in Europe Conf.* IEEE, 453–458.

[7] C.-Y. Hsieh, C.-H. Wu, C.-H. Huang, H.-S. Goan, and J. C. M. Li. 2020. Realistic fault models and fault simulation for quantum dot quantum circuits. In *Proc. Design Automation Conf.* IEEE, 46, 1–6.

[8] D. Kim, M. E. Amyeen, S. Venkataraman, I. Pomeranz, S. Basumallick, and B. Landau. 2007. Testing for systematic defects based on DFM guidelines. In *Proc. Intl. Test Conf.* IEEE, 1–10.

[9] A. Sinha, S. Pandey, A. Singhal, A. Sanyal, and A. Schmaltz. 2017. DFM-aware fault model and ATPG for intra-cell and inter-cell defects. In *Proc. Intl. Test Conf.* IEEE, 1–10.

[10] F. Hapke and J. Schloeffel. 2012. Introduction to the defect-oriented cell-aware test methodology for significant reduction of DPPM rates. In *Proc. European Test Symp.* 1–6.

[11] F. Yang, S. Chakravarty, A. Gunda, N. Wu, and J. Ning. 2014. Silicon evaluation of cell-aware ATPG tests and small delay tests. In *Proc. Asian Test Symp.* IEEE, 101–106.

[12] A. D. Singh. 2016. Cell aware and stuck-open tests. In *Proc. European Test Symp.* IEEE, 1–6.

[13] Y.-H. Huang, C.-H. Lu, T.-W. Wu, Y.-T. Nien, Y.-Y. Chen, M. Wu, Ji.-N. Lee, and M. C.-T. Chao. 2017. Methodology of generating dual-cell-aware tests. In *Proc. VLSI Test Symp.* IEEE, 1–6.

[14] S. P. Dixit, D. D. Vora, and K. Peng. 2018. Challenges in cell-aware test. In *Proc. European Test Symp.* 1–6.

[15] E. J. McCluskey. 1993. Quality and single-stuck faults. In *Proc. Intl. Test Conf.* IEEE, 597.

[16] R. Guo, S. Mitra, E. Amyeen, J. Lee, S. Sivaraj, and S. Venkataraman. 2006. Evaluation of test metrics: Stuck-at, bridge coverage estimate and gate exhaustive. In *Proc. VLSI Test Symp.* IEEE, 66–71.

[17] A. Jas, S. Natarajan, and S. Patil. 2007. The region-exhaustive fault model. In *Proc. Asian Test Symp.* 13–18.

[18] I. Pomeranz. 2019. Iterative test generation for gate-exhaustive faults to cover the sites of undetectable target faults. In *Proc. Intl. Test Conf.* Paper 1.3, IEEE, 1–7.

[19] I. Pomeranz. 2021. Maximal independent fault set for gate-exhaustive faults. *IEEE Trans. on Computer-Aided Design* 40, 3 (2021), 598–602.

[20] P. Goel and B. C. Rosales. 1979. Test generation and dynamic compaction of tests. In *Proc. Test Conf.* IEEE, 189–192.

[21] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy. 1995. Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits. In *IEEE Trans. on Computer-Aided Design.* IEEE, 1496–1504.

[22] D. Xiang, J. Li, K. Chakrabarty, and X. Lin. 2013. Test compaction for small-delay defects using an effective path selection scheme. *ACM Trans. on Design Automation* 18, 3 (2013), Article 44, 1–23.

[23] D. Xiang, K. Shen, B. B. Bhattacharya, X. Wen, and X. Lin. 2016. Thermal-aware small-delay defect testing in integrated circuits for mitigating overkill. *IEEE Trans. on Computer-Aided Design* 35, 3 (2016), 499–512.

[24] W. Jiang and B. Vinnakota. 1999. Defect-oriented test scheduling. In *Proc. VLSI Test Symp.* IEEE, 433–438.
[25] X. Lin, J. Rajski, I. Pomeranz, and S. M. Reddy. 2001. On static test compaction and test pattern ordering for scan designs. In *Proc. Intl. Test Conf.* IEEE, 1088–1097.
[26] L. N. Reddy, I. Pomeranz, and S. M. Reddy. 1992. COMPACTEST-II: A method to generate compact two-pattern test sets for combinational logic circuits. In *Proc. Intl. Conf. on Computer-Aided Design.* IEEE, 568–574.
[27] R. Desineni, K. N. Dwarkanath, and R. D. Blanton. 2000. Universal test generation using fault tuples. In *Proc. Intl. Test Conf.* IEEE, 812–819.
[28] S. M. Reddy, G. Chen, J. Rajski, I. Pomeranz, P. Engelke, and B. Becker. 2005. A unified fault model and test generation procedure for interconnect opens and bridges. In *Proc. Europ. Test Symp.* IEEE, 22–27.
[29] S. Alampally, R. T. Venkatesh, P. Shanmugasundaram, R. A. Parekhji, and V. D. Agrawal. 2011. An efficient test data reduction technique through dynamic pattern mixing across multiple fault models. In *Proc. VLSI Test Symp.* IEEE, 285–290.
[30] C.-H. Wu and K.-J. Lee. 2016. Transformation of multiple fault models to a unified model for ATPG efficiency enhancement. In *Proc. Intl. Test Conf.* IEEE, 1–10.

Received July 2021; revised October 2021; accepted December 2021