IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 14 March 2024, accepted 31 March 2024, date of publication 5 April 2024, date of current version 12 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3385442

RESEARCH ARTICLE

Sharing of Topped-Off Compressed Test Sets Among Logic Blocks

IRITH POMERANZ, (Fellow, IEEE)
School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, USA
e-mail: pomeranz@purdue.edu

ABSTRACT Test data compression implies that a compressed test set is stored on a tester, and an on-chip decompression logic produces tests that can be applied to the circuit from compressed tests. Test data compression is used for reducing the test data volume and test application time. In a design that consists of several logic blocks, distributed test data compression refers to the case where each logic block has dedicated test data compression logic and compressed test set. Test generation procedures that are specific to this scenario attempt to share compressed tests among logic blocks to minimize the overall storage requirements. This article considers the problem of sharing a compressed test set among logic blocks when the goal is to detect faults from several different fault models. Considering a single logic block, a common practice is to consider the fault models one by one, and top off the test set as each additional fault model is considered. The sharing procedure described in this article takes advantage of the topped-off test sets of the logic blocks to consider the fault models one by one. For every fault model it considers the logic blocks one by one. This structure of the procedure provides opportunities to share compressed tests among all the logic blocks. Experimental results using benchmark circuits demonstrate the ability of the procedure to share tests for stuck-at, single-cycle gate-exhaustive and four-way bridging faults in groups of four logic blocks.

INDEX TERMS Bridging faults, linear-feedback shift-register (*LFSR*), single-cycle gate-exhaustive faults, test data compression, test generation.

I. INTRODUCTION

Test data compression is used universally to reduce the test data volume and test application time [1], [2], [3]. With test data compression, a compressed test set is stored on the tester. An on-chip decompression logic accepts compressed tests and produces tests that can be applied to the circuit.

In a design that consists of several logic blocks, such as an SoC or a processor with multiple cores, each logic block may have dedicated test data compression logic and compressed test set. This is referred to as distributed test data compression, and it is analogous to distributed logic built-in self-test [4]. Figure 1 illustrates this scenario considering two logic blocks, B_0 and B_1. The logic block marked M_0 is the decompression logic of B_0, and the logic block marked M_1 is the decompression logic of B_1. In general, a design may contain logic blocks B_0, B_1, ..., B_{n-1} with decompression logic represented by M_0, M_1, ..., M_{n-1}, respectively. For $0 \leq i < n$, the set of target faults for B_i is denoted by F_i, and its test set is denoted by T_i. The notation used in this article for the general case of distributed test data compression is shown in Table 1.

Test generation procedures that are specific to distributed test data compression attempt to share compressed tests among logic blocks to minimize the overall storage requirements [5], [6], [7]. This is illustrated in Figure 1 by the test set denoted by W. For the procedure from [5], M_i stands for the number of data channels used for B_i. For the procedures from [6] and [7], the decompression logic is based on a linear-feedback shift-register (*LFSR*), and tests are compressed into seeds (initial states) for the *LFSR*. For this case, M_i stands for the length of the *LFSR* used for logic block B_i. The two interpretations of M_i are analogous and serve a similar purpose of representing the width of the compressed test data.

The test generation procedure described in [5] considers groups of logic blocks such that all the logic blocks in the

The associate editor coordinating the review of this manuscript and approving it for publication was Poki Chen.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

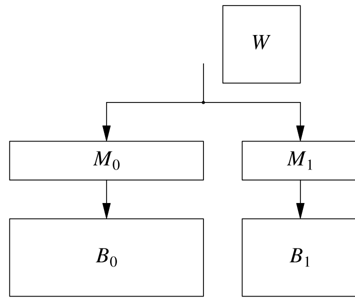

FIGURE 1. Test data compression logic.

TABLE 1. Notation.

symbol	meaning
B_i	logic block
M_i	decompression logic and length of *LFSR* for B_i
F_i	set of target faults for B_i
$F_{i,j}$	subset of F_i from fault model j
T_i	test set for B_i
$T_{i,j}$	subset of tests added to T_i to detect faults from $F_{i,j}$

same group have the same number of data channels. Thus, two logic blocks B_i and B_j are placed in the same group only if $M_i = M_j$. Moreover, the procedure from [5] considers all the logic blocks in a group together as one circuit. Under these restrictions, the procedure from [5] applies a conventional test generation procedure to obtain a compressed test set for the group of logic blocks. The procedures from [6] and [7] remove the restrictions imposed in [5] by using the following building blocks to perform test generation.

(BB1) The test generation procedures from [6] and [7] construct the shared test set W by considering logic blocks one by one. When they extend W to detect faults from F_i, for $0 \leq i < n$, they ensure that the fault coverages already achieved by W for other logic blocks are not affected.

(BB2) If t_a is an M_i-bit compressed test (seed) for B_i, and $M_j \leq M_i$, the procedures may use the first j bits of t_a as a seed for B_j. The resulting seed is denoted by $t_{a/j}$. For example, if $M_i = 8$, $M_j = 6$ and $t_a = 00001111$, the procedures may use $t_{a/j} = 000011$ as a seed for B_j. This building block was introduced in [6] for static test compaction, and used in [7] as well. It allows a seed to be shared with logic blocks that have smaller *LFSR* lengths.

(BB3) When the dynamic test compaction procedure from [7] targets a fault of a logic block B_i, it either adds a new seed to W, or extends an existing seed. If $t_b \in W$ is an M_j-bit seed for B_j, and $M_i > M_j$, the procedure may decide to extend t_b into an M_i-bit seed for B_i by adding $M_i - M_j$ bits. For example, if $M_j = 6$, $M_i = 8$ and $t_b = 000111$ is included in W, the procedure may replace t_b and use $t_b = 00011100$ instead. Because of BB2, the seed $t_{b/j} = 000111$ will be available in W for B_j using the first six bits of the extended seed t_b. This building block allows faults to be detected by adding fewer bits to the storage requirements of W than the number of bits required when adding a new seed.

These building blocks allow the procedures from [6] and [7] to consider any number of logic blocks with any *LFSR* lengths, and they are used in this article as well. To use both BB2 and BB3, the procedures from [6] and [7] are iterative. In a single iteration they use either BB2 or BB3.

This article considers the problem of sharing a compressed test set W among logic blocks when the goal is to detect faults from several different fault models. Test generation procedures consider several fault models to provide a comprehensive coverage of defects that may occur during fabrication or during the lifetime of a chip. A common practice when targeting a single logic block with several fault models is to consider the fault models one by one, and top off the test set as each additional fault model is considered [8], [9], [10], [11], [12], [13], [14], [15], [16], [17]. The procedures from [6] and [7] consider a single set of faults F_i. If a logic block B_i has m sets of target faults, $F_{i,0}, F_{i,1}, \ldots, F_{i,m-1}$, from m fault models, it is possible to define a set of faults $F_i = F_{i,0} \cup F_{i,1} \cup \ldots \cup F_{i,m-1}$, and apply the procedures from [6] and [7] with F_i. However, the computational effort will increase significantly if all the fault models are considered together. Therefore, the goal of this article is to allow both the logic blocks and the fault models to be considered one by one as in a top-off procedure.

The main contribution of this article is the observation that the structure of the topped-off test sets allows all of BB1, BB2 and BB3 to be applied in a single iteration where both the logic blocks and the fault models are considered one by one. To achieve this goal, the sharing procedure described in this article considers the fault models one by one, and the logic blocks one by one for every fault model. This order ensures that the lengths of the seeds in W alternate, and seeds have both shorter and longer seeds ahead of them in the shared test set. The longer seeds allow BB2 to be applied, and the shorter seeds allow BB3 to be applied. This is achieved without the more computationally-intensive iterative parts of the procedures from [6] and [7].

The article reports on academic research. The problem formulation and the algorithm it develops are general and applicable to any design. However, several simplifying assumptions are made to allow the study of the problem and algorithm to be carried out in an academic environment using academic software tools. In particular, an academic version of a test data compression approach is used, where a test is compressed into a single *LFSR* seed. In addition, the sharing procedure is implemented using an academic fault simulation tool, and it is applied to benchmark circuits. With access to commercial tools that use a state-of-the-art compression architecture, fault models, fault simulation and test generation procedures it may be expected that the algorithm developed in this article can be implemented in an industrial environment and applied to industrial designs. The importance of addressing the problem can be seen from the discussion of distributed test data compression in [5], and the discussion of top-off procedures in [11], [12] and [14].

Experimental results for groups of four benchmark circuits demonstrate the ability of the sharing procedure suggested in this article to share compressed tests for single stuck-at,

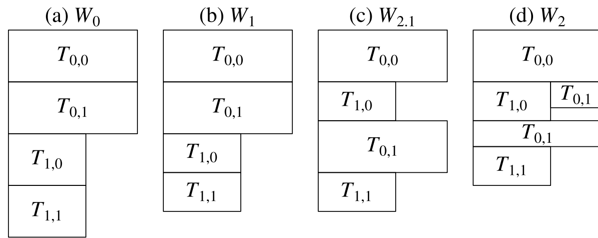

FIGURE 2. Test set sharing.

single-cycle gate-exhaustive [18] and four-way bridging [19] faults among logic blocks with different *LFSR* lengths.

The article is organized as follows. Section II describes the problem addressed and the solution suggested in this article. Section III describes baseline procedures that create shared test sets. Section IV describes the sharing procedure suggested in this article for topped-off test sets. Section V presents experimental results. Section VI analyzes the extent of sharing possible among logic blocks. Section VII concludes the article.

II. PRELIMINARIES

For simplicity of discussion, the sharing procedure is developed as a static test compaction procedure. Thus, it is assumed in this article that test generation was already performed for every one of the logic blocks individually using a top-off procedure. The article focuses on the structure of a sharing procedure whose goal is to share compressed tests among several logic blocks with several fault models in a single iteration while considering the logic blocks and the fault models one by one. Once the structure of the sharing procedure is established, it is possible to incorporate a test generation procedure, as well as perform several iterations to increase the level of sharing.

For illustration of the problem considered and the solution suggested in this article, Figure 2 considers the case where the number of logic blocks is $n = 2$ and the number of fault models is $m = 2$. The logic blocks are such that $M_0 > M_1$. For $i = 0$ and 1, logic block B_i has two sets of faults, $F_{i,0}$ and $F_{i,1}$. Accordingly, the test set $T_{i,0}$ for $F_{i,0}$ is topped off with a test set $T_{i,1}$ for $F_{i,1}$ to obtain the test set T_i for B_i. Without any sharing, the test set $W_0 = T_{0,0}T_{0,1}T_{1,0}T_{1,1}$ is shown in Figure 2(a).

Figure 2(b) illustrates the use of BB2 for reducing the number of seeds. In this case, the sets of faults are considered in the order $F_{0,0}$, $F_{0,1}$, $F_{1,0}$, $F_{1,1}$. When $F_{1,0}$ and then $F_{1,1}$ is considered, fault simulation of $F_{1,0}$ or $F_{1,1}$ under $T_{0,0}$ and $T_{0,1}$ makes some of the seeds in $T_{1,0}$ or $T_{1,1}$ unnecessary. The resulting test set is denoted by W_1, and it is smaller than W_0.

A higher level of sharing, implying a higher level of test compaction, requires BB3 to be used as well. To provide opportunities for both BB2 and BB3 to be applied, Figure 2(c) illustrates the case where the sets of faults are considered in the order $F_{0,0}$, $F_{1,0}$, $F_{0,1}$, $F_{1,1}$. In Figure 2(c), BB2 is used for reducing the number of tests added from $T_{1,0}$ and $T_{1,1}$ when $F_{1,0}$ and then $F_{1,1}$ are considered. This is possible since longer seeds from $T_{0,0}$ are available when $F_{1,0}$ is considered, and longer seeds from $T_{0,0}$ and $T_{0,1}$ are available when $F_{1,1}$ is considered. The resulting test set is denoted by $W_{2.1}$.

BB3 is applied in Figure 2(d). Figure 2(d) is similar to Figure 2(c), except that seeds from $T_{1,0}$ are extended to detect faults from $F_{0,1}$ when they are considered. This allows fewer seeds from $T_{0,1}$ to be added to the shared test set as new seeds. Thus, in Figure 2(d), BB2 is used for reducing the number of seeds added from $T_{1,0}$ and $T_{1,1}$, and BB3 is used for reducing the number of seeds added from $T_{0,1}$. The resulting test set is denoted by W_2, and it is smaller than W_0, W_1, and $W_{2.1}$. The sharing procedure described in this article produces the test set W_2 illustrated by Figure 2(d).

A baseline for comparison is established in this article by procedures that produce test sets as shown in Figure 2(a) and (b). The procedures from [6] and [7] are not applied since they are not geared toward the consideration of several fault models, and they require several iterations to apply both BB2 and BB3. If the basic test compaction procedure from [6] is extended to use several fault models in a single iteration, it will produce a test set similar to W_1 illustrated by Figure 2(b). Specifically, the basic test compaction procedure from [6] initially assigns $W = W_0$, and pads all the seeds in W randomly into M_0-bit seeds. For every seed $w_b \in W$, the procedure assigns $l(w_b) = 0$ initially to indicate that w_b is not used for detecting any faults. During the procedure, $l(w_b)$ will be changed to M_i if the first M_i bits of w_b are used for detecting faults from F_i. With $M_0 \geq M_1 \geq \ldots \geq M_{n-1}$, the procedure considers the logic blocks in the order B_0, B_1, ..., B_{n-1}. When B_i is considered, the procedure simulates F_i under the seeds in W with $l(w_b) = l$, for $l = M_0, \ldots, M_{i-1}, 0$. For every seed $w_b \in W$ in this order, it simulates F_i under the test produced by the first M_i bits of w_b with fault dropping. If w_b detects any faults, and $l(w_b) < M_i$, the procedure assigns $l(w_b) = M_i$. When the procedure terminates, seeds with $l(w_b) = 0$ can be removed from W, and the padding beyond $l(w_b)$ bits can be ignored to obtain the final test set W.

To use both BB2 and BB3, the iterative process from [6] introduces a new padding for the seed at the end of the shared test set, rotates the test set, and then applies the basic test compaction procedure again. For the new seed at the top of the test set, padding has a similar effect to BB3 since a seed that originally had fewer than M_0 bits now has M_0 bits and can be used for any logic block.

For a shared test set W, the number of iterations in [6] is on the order of $|W|$.

The dynamic test compaction procedure from [7] has two forms, one with a lower and one with a higher computational effort. The former is described next. The procedure starts from a shared test set $W = T_{n-1}$. It considers the logic blocks in the order M_{n-2}, M_{n-3}, ..., M_0. When M_i is considered, the procedure performs the following steps. It first selects the test from T_i that detects the largest number of faults from F_i. Let the selected test be t_a. The procedure considers two options for t_a. The first option is to add t_a to W as a new seed. The second option is to pad one of the seeds from W using the

extra bits from t_a. Of all the seeds in W, the procedure selects the one that yields the largest number of detected faults. Let the selected seed be $w_b \in W$. Between adding t_a as a new seed and padding w_b the procedure selects the option that detects more faults. It finalizes this option, and continues to the next test from T_i until all the faults from F_i are detected.

The dynamic test compaction procedure from [7] performs on the order of $|T_i|$ steps for every logic block. In every step it requires fault simulation of all the tests from T_i, and all the tests from W.

These more computationally intensive procedures from [6] and [7] are not considered in this article.

III. BASELINE PROCEDURES

This section describes baseline procedures for sharing test sets among logic blocks.

Using the notation from Sections I and II, for $0 \leq i < n$, B_i is a logic block with a set of faults F_i, and a compressed test set T_i that consists of seeds for an *LFSR* of length M_i. The logic blocks are ordered such that $M_0 \geq M_1 \geq \ldots \geq M_{n-1}$.

With m fault models, F_i consists of subsets $F_{i,j}$ for $0 \leq j < m$. The test set T_i is partitioned into subsets $T_{i,j}$ for $0 \leq j < m$, where $T_{i,j}$ is added to T_i to detect faults from $F_{i,j}$ that are not detected by $T_{i,0}, \ldots, T_{i,j-1}$.

The test set W_0 is illustrated by Figure 2(a) for $n = 2$ and $m = 2$. It is obtained by concatenating $T_{i,j}$ for $i = 0, 1, \ldots, n-1$ and $j = 0, 1, \ldots, m-1$. No sharing of seeds occurs in this case.

The test set W_1 is illustrated by Figure 2(b) for $n = 2$ and $m = 2$, and produced by Procedure 1. Sharing of seeds occurs in W_1 by using BB2. Specifically, before adding seeds from $T_{i,j}$ to detect faults from $F_{i,j}$, Procedure 1 simulates $F_{i,j}$ under seeds that already exist in W_1. Because of the order of the logic blocks from high to low *LFSR* length, a seed $w_b \in W_1$ has at least M_i bits when B_i is considered. Therefore, the first M_i bits of the seed can be used for B_i.

Initially in Procedure 1, $W_1 = \emptyset$. For $i = 0, 1, \ldots, n-1$ and $j = 0, 1, \ldots, m-1$, fault simulation of $F_{i,j}$ under W_1 is carried out first by calling Procedure *fsim*(). For later use, the length of a seed w_b is denoted by $l(w_b)$, and Procedure *fsim*() simulates a seed w_b only if $l(w_b) \geq M_i$. This added condition is not needed for W_1, but it will be useful later.

Next in Procedure 1, tests from $T_{i,j}$ that detect faults from $F_{i,j}$ are added to W_1 to obtain complete fault coverage for $F_{i,j}$. Procedure *addtests*() is used for this purpose.

Procedure 1: Shared test set W_1

1) Assign $W_1 = \emptyset$.
2) For $i = 0, 1, \ldots, n-1$:
   a) For $j = 0, 1, \ldots, m-1$:
      i) Call Procedure *fsim*$(1, i, j)$.
      ii) Call Procedure *addtests*$(1, i, j)$.

Procedure *fsim*(s, i, j)**:**

1) For every seed $w_b \in W_s$, if $l(w_b) \geq M_i$:
   a) Let $w_{b/i}$ consist of the first M_i bits of w_b.
   b) Simulate $F_{i,j}$ under the test produced by $w_{b/i}$ with fault dropping.

Procedure *addtests*(s, i, j)**:**

1) For every seed $t_a \in T_{i,j}$:
   a) Simulate $F_{i,j}$ under the test produced by t_a with fault dropping.
   b) If t_a detected any faults, add it to W_s.

IV. SHARING PROCEDURE FOR TOPPED-OFF TEST SETS

This section describes the procedure suggested in this article for sharing of compressed topped-off tests among logic blocks. The procedure is referred to as Procedure 2. The shared test set it produces is denoted by W_2.

A. PROCEDURE OVERVIEW

Procedure 2 considers one fault model at a time. For every fault model it considers the logic blocks one by one. Using only Procedures *fsim*() and *addtests*(), this would result in a shared test set denoted by $W_{2.1}$, and illustrated by Figure 2(c) for $n = 2$ and $m = 2$.

Procedure 2 includes two additional procedures that apply BB3 to the seeds in W_2, Procedure *extend*() and Procedure *unextend*(). The two procedures are described considering a logic block B_i and a set of faults $F_{i,j}$.

Procedure 2: Shared test set W_2

1) Assign $W_2 = \emptyset$.
2) For $j = 0, 1, \ldots, m-1$:
   a) For $i = 0, 1, \ldots, n-1$:
      i) Call Procedure *fsim*$(2, i, j)$.
      ii) Call Procedure *extend*$(2, i, j)$.
      iii) Call Procedure *addtests*$(2, i, j)$.
      iv) Call Procedure *unextend*$(2, i, j)$.

Procedure *extend*(s, i, j):

1) For every seed $w_b \in W_s$ assign $l_{prev}(w_b) = l(w_b)$.
2) For every seed $w_b \in W_s$ such that $l(w_b) < M_i$:
   a) For every seed $t_a \in T_{i,j}$:
      i) Find the extended seed $w_{b,a}$.
      ii) Simulate $F_{i,j}$ under $w_{b,a}$ and find the number of detected faults, $d(w_{b,a})$.
3) Select the extended seed $w_{b,a}$ with the largest value of $d(w_{b,a})$, and the smallest value of $M_i - l(w_b)$.
4) If $d(w_{b,a}) = 0$, stop.
5) Replace w_b with $w_{b,a}$ in W_s. Perform fault simulation with fault dropping of $F_{i,j}$ under w_b. Go to Step 2.

Procedure *unextend*(s, i, j)**:**

1) For $l_{prev} = M_{n-1}, M_{n-2}, \ldots, M_{i-1}$:
   a) For every seed $w_b \in W_s$ such that $l(w_b) > l_{prev}(w_b)$ and $l_{prev}(w_b) = l_{prev}$:
      i) Let F_{targ} consist of all the faults from $F_{i,j}$ that are detected by w_b.
      ii) Simulate F_{targ} under $W_s \setminus \{w_b\}$ and mark detected faults.
      iii) If all the faults in F_{targ} are marked detected, assign $l(w_b) = l_{prev}(w_b)$.

TABLE 2. Extending seeds.

ind	seed	det
51	100000101001110101010100100	
0	0100001100101000010100110 10	
51,0	1000001010011101010101001 00 10	1
51	100000101001110101010100100	
1	0000100100101100100111001 01	
51,1	100000101001110101010100100 01	2
95	00110010000000	
0	01000011001010 0001010011010	
95,0	00110010000000 0001010011010	1
95	00110010000000	
1	00001001001011 0010011100101	
95,1	00110010000000 0010011100101	3
107	01001000111011	
37	01010011101011 1100100110010	
107,37	01001000111011 1100100110010	16
157	011101101011	
0	010000110010 100001010011010	
157,0	011101101011 100001010011010	6
157	011101101011	
1	000010010010 110010011100101	
157,1	011101101011 110010011100101	3

B. PROCEDURE EXTEND()

Procedure 2 calls procedure *extend*() after simulating $F_{i,j}$ under W_2. Fault simulation removes faults that are already detected by W_2. Procedure *extend*() applies BB3 to the seeds in W_2. Thus, the procedure extends some of the seeds in W_2, whose length is lower than M_i, into M_i-bit seeds. The goal of the extension is to detect additional faults from $F_{i,j}$. This reduces the number of M_i-bit seeds that will be added when Procedure *addtests*() is applied. Thus, instead of M_i bits for every new seed, smaller numbers of bits will be added for detecting faults from $F_{i,j}$

Procedure *extend*() uses the seeds in $T_{i,j}$ as a source of extensions for the seeds in W_2. Considering a seed $w_b \in W_2$ such that $l(w_b) < M_i$, and a seed $t_a \in T_{i,j}$, the procedure obtains an extended seed $w_{b,a}$ whose first $l(w_b)$ bits are copied from w_b, and last $M_i - l(w_b)$ bits are copied from t_a.

Table 2 shows several examples based on a group of logic blocks referred to later as G_0. The group consists of $n = 4$ logic blocks with $M_0 = 27$, $M_1 = 25$, $M_2 = 14$, and $M_3 = 12$. The number of fault models considered is $m = 3$. In the example, after considering $F_{i,0}$ for $i = 0, 1, 2$ and 3, the set W_2 consists of 51 seeds of length 27, 44 seeds of length 25, 62 seeds of length 14, and 10 seeds of length 12. When $F_{0,1}$ is considered next, fault simulation of $F_{0,1}$ under W_2 shows that 88.536% of the faults in $F_{0,1}$ are detected. Table 2 shows several of the extensions considered for detecting additional faults from $F_{0,1}$. Every triple of rows in Table 2 shows a seed $w_b \in W_2$ on the first row, a seed $t_a \in T_{0,1}$ on the second row, and the extended seed $w_{b,a}$ on the third row. The third row also shows the number of faults from $F_{0,1}$ that will be detected if $w_{b,a}$ replaces w_b in W_2. The best option is $w_{107,37}$ with 16 detected faults.

In general, the procedure iterates through a process where, in every iteration, it considers every seed $w_b \in W_2$ such that $l(w_b) < M_i$, and every seed $t_a \in T_{i,j}$. It obtains the extended seed $w_{b,a}$. It then performs fault simulation of $F_{i,j}$ under $w_{b,a}$ to find the number of detected faults. Of all the options for $w_{b,a}$, the procedure selects the one that detects the largest number of faults. If a choice exists, the procedure prefers the option for which $l(w_b)$ is the largest since this will result in the smallest number of additional bits for W_2.

The procedure repeats the selection of an extended seed as long as it can detect additional faults from $F_{i,j}$.

Several observations are used for speeding up the procedure.

(1) For a seed $w_b \in W_2$, two different seeds $t_{a_0} \in T_{i,j}$ and $t_{a_1} \in T_{i,j}$ may result in the same extended seed if the last $M_i - l(w_b)$ bits of t_{a_0} and t_{a_1} are the same. Procedure *extend*() considers only t_{a_0} for extending w_b in this case. For example, with $M_i - l(w_b) = 2$, at most four seeds from $T_{i,j}$ will be considered, with the last two bits being 00, 01, 10 or 11.

(2) After computing the number of detected faults $d(w_{b,a})$ for $w_b \in W_2$ and $t_a \in T_{i,j}$ in an arbitrary iteration, the value of $d(w_{b,a})$ obtained in the next iteration cannot increase. This is because faults from $F_{i,j}$ are removed from consideration at the end of an iteration. Procedure *extend*() stores the previous value obtained for $d(w_{b,a})$ in a variable denoted by $d_{prev}(w_{b,a})$. It considers the pairs $w_b \in W_2$ and $t_a \in T_{i,j}$ from high to low value of $d_{prev}(w_{b,a})$, and from low to high value of $M_i - l(w_b)$. As it considers pairs of seeds, it stores the best number of detected faults for the iteration in a variable denoted by d_{best}. It does not consider a pair $w_b \in W_2$ and $t_a \in T_{i,j}$ if $d_{prev}(w_{b,a}) < d_{best}$ or $d_{prev}(w_{b,a}) = 0$ since such a pair will not be selected.

C. PROCEDURE UNEXTEND()

Procedure *addtests*() is applied after Procedure *extend*() to add seeds from $T_{i,j}$ to W_2 for faults from $F_{i,j}$ that are not detected by W_2. Some of the seeds added by Procedure *addtests*() may make some of the extensions made by Procedure *extend*() unnecessary. The goal of Procedure *unextend*() is to identify such extensions and eliminate them to reduce the storage requirements of W_2.

To allow Procedure *unextend*() to identify the extensions made by Procedure *extend*(), Procedure *extend*() stores the previous length of every seed $w_b \in W_2$ in a variable denoted by $l_{prev}(w_b)$. An extension was made if $l(w_b) > l_{prev}(w_b)$.

Procedure *unextend*() considers the seeds in W_2 from low to high value of $l_{prev}(w_b)$. This gives a preference to recovering shorter seeds, with a smaller contribution to the storage requirements of W_2.

For a seed $w_b \in W_2$ with $l(w_b) > l_{prev}(w_b)$, to be able to undo the extension, it is necessary to consider all the faults from $F_{i,j}$ that are detected by w_b. This subset of faults is denoted by F_{targ}. The procedure simulates F_{targ} under the other seeds in W_2. If all the faults in F_{targ} are detected, the procedure assigns $l(w_b) = l_{prev}(w_b)$ to undo the extension of w_b.

V. EXPERIMENTAL RESULTS

This section presents experimental results for groups of benchmark circuits that are considered as logic blocks in a design.

TABLE 3. Logic blocks.

i	block	inp	lfsr	seeds	bits
0	sasc	132	12	154	1848
1	usb_phy	112	27	128	3456
2	systemcdes	320	14	251	3514
3	s1423	91	25	144	3600
4	b07	53	18	286	5148
5	s35932	1763	40	145	5800
6	pci_spoci_ctrl	83	25	340	8500
7	b04	78	39	245	9555
8	des_area	367	13	736	9568
9	i2c	145	52	306	15912
10	simple_spi	146	59	339	20001
11	b14	280	39	747	29133
12	s5378	214	57	600	34200
13	wb_dma	738	62	619	38378
14	s9234	247	48	816	39168
15	systemcaes	928	71	716	50836
16	b15	483	53	1057	56021
17	b20	527	61	1208	73688
18	tv80	372	56	1576	88256
19	s13207	700	79	1189	93931
20	aes_core	788	51	2127	108477
21	s15850	611	76	1608	122208
22	s38584	1464	97	1376	133472
23	spi	274	76	2113	160588
24	wb_conmax	1900	54	3146	169884
25	s38417	1664	86	2902	249572
26	b17	1444	94	3840	360960

Three sets of target faults are considered for every logic block, stuck-at, single-cycle gate-exhaustive [18], and four-way bridging [19] faults. A test set T_i is produced for every logic block B_i by topping off a stuck-at test set considering the fault models in this order. The number of bits required for storing T_i is equal to $M_i|T_i|$.

Table 3 shows information about the logic blocks considered. The logic blocks are ordered from low to high value of $M_i|T_i|$. In Table 3, after the index i of a logic block, column *block* shows the name of B_i, column *inp* shows the number of inputs of B_i, column *lfsr* shows the length M_i of the *LFSR*, column *seeds* shows the number of seeds in T_i, and column *bits* shows the number of bits $M_i|T_i|$.

Groups of four benchmark circuits are considered as follows. In the order of the logic blocks given in Table 3, every four consecutive logic blocks are considered as a group. For $0 \leq i \leq 23$, the group G_i consists of the four logic blocks starting from B_i, i.e., B_i, B_{i+1}, B_{i+2} and B_{i+3}. Using logic blocks with similar storage requirements ensures, to the extent possible, that no single logic block dominates the results.

The goal of considering every four consecutive logic blocks as a group is to produce a large number of datapoints for studying the results of Procedures 1 and 2. It is not expected that all the groups will be used for the same design. Nonoverlapping groups are considered in Section VI.

Three shared test sets are considered in this section, W_0 for which no sharing is attempted, W_1 obtained by Procedure 1, and W_2 obtained by Procedure 2. As discussed earlier, W_1 represents a test set that would be produced by the non-iterative part of the procedure from [6] if it is extended to consider several fault models, and the more computationally intensive procedures from [6] and [7] are not used in this article.

TABLE 4. Shared test sets W_0, W_1 and W_2 for groups of four logic blocks.

group	lfsr	s	tests	bits	frac	ntime	M0	M1	M2	M3
G0	27	0	677	12418	1.000	1.00	128	144	251	154
G0	27	1	509	10166	0.819	1.62	128	136	185	60
G0	27	2	418	8832	0.711	7.70	130	126	114	48
G1	27	0	809	15718	1.000	1.00	128	144	286	251
G1	27	1	627	12938	0.823	1.75	128	136	250	113
G1	27	2	542	11445	0.728	7.97	133	128	180	101
G2	40	0	826	18062	1.000	1.00	145	144	286	251
G2	40	1	652	15459	0.856	1.10	145	139	258	110
G2	40	2	539	12903	0.714	51.05	125	121	194	99
G3	40	0	915	23048	1.000	1.00	145	484	484	286
G3	40	1	827	21219	0.921	1.04	145	449	449	233
G3	40	2	714	18506	0.803	75.67	124	418	418	172
G4	40	0	1016	29003	1.000	1.00	145	245	340	286
G4	40	1	917	26584	0.917	0.96	145	219	327	226
G4	40	2	794	23433	0.808	68.32	125	217	262	190
G5	40	0	1466	33423	1.000	1.00	145	245	340	736
G5	40	1	1285	30238	0.905	1.17	145	219	327	594
G5	40	2	1139	27053	0.809	98.50	122	216	278	523
G6	52	0	1627	43535	1.000	1.00	306	245	340	736
G6	52	1	1375	39157	0.899	1.41	306	216	311	542
G6	52	2	1248	37461	0.860	11.55	331	204	252	461
G7	59	0	1626	55036	1.000	1.00	339	306	245	736
G7	59	1	1370	50200	0.912	1.65	339	300	196	535
G7	59	2	1250	48798	0.887	12.11	421	234	156	439
G8	59	0	2128	74614	1.000	1.00	339	306	747	736
G8	59	1	1668	67048	0.899	3.73	339	300	695	334
G8	59	2	1449	58506	0.784	21.61	389	227	497	336
G9	59	0	1992	99246	1.000	1.00	339	600	306	747
G9	59	1	1887	94809	0.955	8.53	339	594	288	666
G9	59	2	1572	82387	0.830	33.82	391	577	221	383
G10	62	0	2305	121712	1.000	1.00	619	339	600	747
G10	62	1	2078	110259	0.906	6.02	619	236	570	653
G10	62	2	1637	91476	0.752	38.97	625	221	491	300
G11	62	0	2782	140879	1.000	1.00	619	600	816	747
G11	62	1	2530	129584	0.920	7.41	619	579	695	637
G11	62	2	1850	100202	0.711	46.29	574	626	398	252
G12	71	0	2751	162582	1.000	1.00	716	619	600	816
G12	71	1	2490	148734	0.915	4.43	716	553	556	665
G12	71	2	2080	131497	0.809	114.17	997	379	380	324
G13	71	0	3208	184403	1.000	1.00	716	619	1057	816
G13	71	1	2861	166228	0.901	5.39	716	553	938	654
G13	71	2	2086	130035	0.705	81.35	961	411	410	304
G14	71	0	3797	219713	1.000	1.00	716	1208	1057	816
G14	71	1	3406	199538	0.908	6.84	716	1159	903	628
G14	71	2	2582	159942	0.728	78.59	929	963	424	266
G15	71	0	4557	268801	1.000	1.00	716	1208	1576	1057
G15	71	1	4124	245124	0.912	8.69	716	1159	1464	785
G15	71	2	3309	202838	0.755	98.18	910	1003	1019	377
G16	79	0	5030	311896	1.000	1.00	1189	1208	1576	1057
G16	79	1	4534	284652	0.913	11.08	1189	1141	1436	768
G16	79	2	3553	238109	0.763	106.05	1592	718	888	355
G17	79	0	6100	364352	1.000	1.00	1189	1208	1576	2127
G17	79	1	5341	324273	0.890	2.99	1189	1141	1436	1575
G17	79	2	4825	302503	0.830	32.70	1631	665	822	1707
G18	79	0	6500	412872	1.000	1.00	1189	1608	1576	2127
G18	79	1	5695	370657	0.898	2.91	1189	1598	1394	1514
G18	79	2	5291	355334	0.861	28.04	1566	1551	574	1600
G19	97	0	6300	458088	1.000	1.00	1376	1189	1608	2127
G19	97	1	5609	421569	0.920	2.01	1376	1163	1586	1484
G19	97	2	5088	391185	0.854	212.41	1981	532	1027	1548
G20	97	0	7224	524745	1.000	1.00	1376	3721	3721	2127
G20	97	1	6048	455119	0.867	1.85	1376	3335	3335	1337
G20	97	2	5567	428867	0.817	298.62	1950	2210	2210	1407
G21	97	0	8243	586152	1.000	1.00	1376	3721	3721	3146
G21	97	1	6729	495904	0.846	2.56	1376	3335	3335	2018
G21	97	2	5938	454368	0.775	285.79	1870	2423	2423	1645
G22	97	0	9537	713516	1.000	1.00	1376	2902	2113	3146
G22	97	1	7325	575340	0.806	2.92	1376	2875	1301	1773
G22	97	2	6523	526586	0.738	208.51	1932	2188	966	1437
G23	94	0	12001	941004	1.000	1.00	3840	2902	2113	3146
G23	94	1	9028	753834	0.801	3.12	3840	2811	1035	1342
G23	94	2	8160	693000	0.736	97.38	3997	2417	688	1058

For a shared test set W_s, where $s = 0$, 1 or 2, the number of bits required for storing it is denoted by $S(W_s)$. The fraction $\sigma(W_s) = S(W_s)/S(W_0)$ shows the reduction in the storage requirements when W_s is used instead of W_0.

Table 4 compares the test sets W_0, W_1 and W_2. For every test set W_s, where $s = 0$, 1 or 2, column *group* shows the name of the group it is computed for. Column *lfsr* shows the

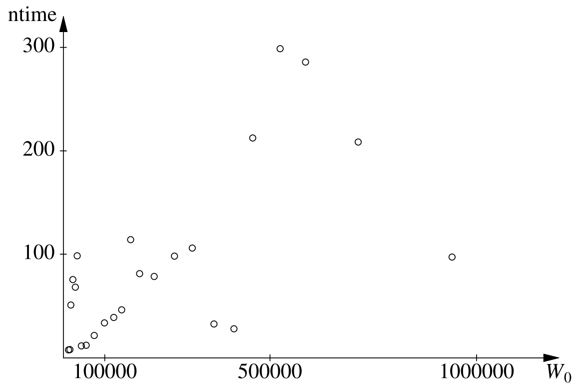

FIGURE 3. Normalized runtime.

maximum length of an *LFSR* for a logic block in the group. Column s shows the index of the shared test set W_s. Column *tests* shows the total number of tests in W_s. Column *bits* shows the total number of bits required for storing W_s, denoted by $S(W_s)$. Column *frac* shows the number of bits required for W_s divided by the number of bits required for W_0, denoted by $\sigma(W_s) = S(W_s)/S(W_0)$. This fraction provides the reduction in the number of bits achieved by Procedures 1 and 2. Column *ntime* shows the runtime for obtaining W_s divided by the runtime for fault simulation of W_0. This is referred to as the normalized runtime, and measures the computational effort of Procedures 1 and 2. Column *Mi*, for $0 \leq i \leq 3$, shows the number of seeds of length M_i in W_s. In the row for W_0, this is typically the number of seeds in T_i (exceptions occur if two logic blocks have the same *LFSR* length, and their numbers of tests are added; this occurs for G_3, G_{20} and G_{21}).

The following points can be seen from Table 4.

(1) By applying both BB2 and BB3, Procedure 2 is able to reduce the storage requirements of the shared test set significantly compared with W_0 and W_1. The reductions are similar for different groups of logic blocks of different sizes.

(2) The normalized runtime is also similar for different groups of logic blocks, and groups with larger logic blocks sometimes have lower normalized runtimes. This is a result of the fact that Procedure 2 is based on fault simulation, and thus, scales similar to a fault simulation procedure. To demonstrate this point, Figure 3 plots the normalized runtime for the computation of W_2 as a function of the size of W_0 that measures the size of the group. Figure 3 demonstrates that the normalized runtime does not increase with the size of W_0. This again supports the conclusion that the procedure scales similar to a fault simulation procedure.

(3) It is possible to extract from W_2 individual test sets for the logic blocks. The numbers of tests under columns *Mi*, for $0 \leq i \leq 3$, provide an indication of the numbers of tests that would be obtained. Based on these columns, sharing of seeds typically results in an increase in the number of seeds required for B_0, and a reduction in the numbers of seeds required for the other logic blocks. Overall, the number of seeds in W_2 is lower than the number of seeds in W_0 that does not share seeds among the logic blocks.

VI. ANALYSIS OF SHARING

As noted in [6], the ability to share compressed tests among logic blocks in a group varies with the group. The extent of sharing for a group can be assessed by considering pairs of logic blocks in the group. Moreover, it is possible to select groups of logic blocks for which the extent of sharing is expected to be high by using information about the sharing possible for pairs. This section computes the extent of sharing for pairs of logic blocks, and demonstrates that it is possible to construct groups for which significant sharing will be obtained when several fault models are targeted by a top-off procedure.

Table 5 shows some of the results obtained when Procedures 1 and 2 are applied to all the pairs of logic blocks from Table 3. A pair that consists of logic blocks B_{i_0} and B_{i_1} is denoted by $P_{i_0.i_1}$, where $0 \leq i_0 < i_1 \leq 26$. Following the name of the pair, Table 5 shows the fractions of storage requirements, $\sigma(W_1) = S(W_1)/S(W_0)$ and $\sigma(W_2) = S(W_2)/S(W_0)$, obtained by Procedures 1 and 2, respectively. The pairs in each part of Table 5 are ordered from low to high value of $\sigma(W_2)$. In the first part of Table 5, all the pairs for which $\sigma(W_2) < 0.820$ are shown. Additional pairs are shown in the other parts of Table 5 as discussed later.

From Table 5 it can be seen that, similar to the case where groups of four are considered, $\sigma(W_2) < \sigma(W_1)$ is obtained for pairs as well. Moreover, a lower value of $\sigma(W_1)$ does not predict a lower value of $\sigma(W_2)$, and the two fractions vary independently.

Another important observation from the first part of Table 5 is that the lowest values of $\sigma(W_2)$ are obtained for logic blocks B_{i_0} and B_{i_1} such that $i_1 - i_0$ is small. For example, for $\sigma(W_2) < 0.75$ it is the case that $i_1 - i_0 \leq 4$. This is to be expected when the logic blocks are ordered based on their storage requirements, and sharing is more effective when none of the logic blocks dominates the storage requirements.

The first part of Table 5 identifies pairs of logic blocks for which the extent of sharing is the highest. These pairs can be used for forming larger groups for which the extent of sharing is expected to be high. For example, using the first two pairs of logic blocks in Table 5, $P_{1.5}$ and $P_{13.14}$, the group that consists of B_1, B_5, B_{13} and B_{14} would be formed.

However, the selection of nonoverlapping pairs, such as $P_{1.5}$ and $P_{13.14}$, may include in the same group pairs for which the extent of sharing is low. For example, the use of $P_{1.5}$ and $P_{13.14}$ to form a group includes in the group the pairs $P_{1.13}$, $P_{1.14}$, $P_{5.13}$ and $P_{5.14}$, for which the reductions in storage requirements are 0.946, 0.945, 0.871 and 0.871, respectively.

The data in the first part of Table 5 is considered again, this time with overlapping pairs. After using $P_{1.5}$ to initialize a group, the pairs $P_{3.5}$ and $P_{5.7}$, that overlap with the logic blocks in the group, may be used for forming a group that consists of B_1, B_5, B_3 and B_7. This group is denoted by Q_0.

TABLE 5. Pairs of logic blocks.

Q_0

pair	proc1	proc2
P1.5	0.939	0.719
P13.14	0.957	0.745
P3.5	0.987	0.747
P5.7	0.934	0.766
P13.16	0.965	0.777
P10.14	0.961	0.792
P0.5	0.969	0.799
P4.5	0.972	0.800
P5.9	0.805	0.814
P2.5	0.982	0.816
.	.	.

Q_1

pair	proc1	proc2
P13.14	0.957	0.745
P13.16	0.965	0.777
P10.14	0.961	0.792
P23.26	0.882	0.831
P16.17	0.955	0.838
.	.	.

Q_2

pair	proc1	proc2
P23.26	0.882	0.831
P0.2	0.875	0.853
P0.4	0.887	0.857
P2.4	0.877	0.859
P0.8	0.859	0.861
P23.25	0.901	0.863
P22.23	0.908	0.871
P24.26	0.915	0.874
P11.18	0.972	0.874
.	.	.

Q_3

pair	proc1	proc2
P0.2	0.875	0.853
P0.4	0.887	0.857
P2.4	0.877	0.859
P0.8	0.859	0.861
P11.18	0.972	0.874
P2.8	0.947	0.878
P4.8	0.933	0.896
P15.18	0.979	0.900
P2.6	0.918	0.903
P17.18	0.976	0.905
.	.	.

Q_4

pair	proc1	proc2
P11.18	0.972	0.874
P15.18	0.979	0.900
P17.18	0.976	0.905
P17.24	0.969	0.912
P18.21	0.968	0.913
P11.17	0.971	0.914
.	.	.

Q_5

pair	proc1	proc2
P21.24	0.963	0.924
P12.21	0.984	0.931
P20.24	0.938	0.932
P19.24	0.971	0.946
P9.21	0.990	0.946
P12.19	0.986	0.952
P20.21	0.985	0.959
.	.	.

Q_6

pair	proc1	proc2
P9.19	0.986	0.965
P6.9	0.975	0.972
P6.19	0.989	0.995

Let the set of all the pairs of logic blocks be Ψ. After selecting the group Q_0 based on Ψ, it is possible to remove from Ψ every pair $P_{i_0.i_1}$ such that either B_{i_0} or B_{i_1} is already included in Q_0. The selection process can then be repeated to select a group Q_1 that does not overlap with Q_0. This can be repeated for $j \geq 0$ to select a group Q_j that does not overlap with $Q_0, \ldots, Q_{j-1}$.

TABLE 6. Shared test sets W_0, W_1 and W_2 for groups selected by procedure 3.

group	lfsr	s	tests	bits	frac	ntime	M0	M1	M2	M3
Q0	40	0	662	22411	1.000	1.00	145	245	128	144
Q0	40	1	554	19221	0.858	0.93	145	219	65	125
Q0	40	2	448	16164	0.721	15.61	132	208	36	72
Q1	62	0	2831	153568	1.000	1.00	619	339	1057	816
Q1	62	1	2506	136425	0.888	6.28	619	236	975	676
Q1	62	2	1860	103262	0.672	16.57	596	183	725	356
Q2	97	0	10231	904592	1.000	1.00	1376	3840	2902	2113
Q2	97	1	8859	797990	0.882	5.92	1376	3785	2768	930
Q2	97	2	7722	712774	0.788	88.63	2090	3569	1777	286
Q3	18	0	1427	20078	1.000	1.00	286	251	736	154
Q3	18	1	1080	15636	0.779	1.34	286	175	610	9
Q3	18	2	996	14564	0.725	5.01	296	148	540	12
Q4	71	0	4247	241913	1.000	1.00	716	1208	1576	747
Q4	71	1	3942	227036	0.939	6.24	716	1159	1464	603
Q4	71	2	3021	184747	0.764	122.06	908	958	988	167
Q5	76	0	7481	434769	1.000	1.00	1608	600	3146	2127
Q5	76	1	6326	374634	0.862	1.82	1608	556	2824	1338
Q5	76	2	6045	362295	0.833	24.06	1749	464	2497	1335
Q6	79	0	1835	118343	1.000	1.00	1189	306	340	-
Q6	79	1	1744	115258	0.974	1.36	1189	276	279	-
Q6	79	2	1594	113128	0.956	105.34	1277	160	157	-

Procedure 3 summarizes this process when the goal is to create nonoverlapping groups of size Γ. Step 4 of Procedure 3 considers all the pairs from Ψ in every iteration for the following reason. It is possible that a pair $P_{i_0.i_1}$ will not be added to Q_j in one iteration because neither B_{i_0} nor B_{i_1} is included in Q_j. If one of the two logic blocks is added to Q_j in a later iteration, it is important to consider $P_{i_0.i_1}$ again.

Procedure 3: Forming groups $Q_0, Q_1, \ldots$

1) Compute Ψ. Sort the pairs in Ψ from low to high value of $\sigma(W_2)$.
2) Assign $j = 0$.
3) Select the first pair $P_{i_0.i_1} \in \Psi$ and assign $Q_j = \{B_{i_0}, B_{i_1}\}$.
4) For every pair $P_{i_0.i_1} \in \Psi$, in the order of the sorted set, if exactly one of B_{i_0} and B_{i_1} is included in Q_j:
   a) If only B_{i_0} is included in Q_j, add B_{i_1} to Q_j.
   b) If only B_{i_1} is included in Q_j, add B_{i_0} to Q_j.
   c) Go to Step 5.
5) If $|Q_j| < \Gamma$, go to Step 4.
6) Remove from Ψ every pair $P_{i_0.i_1}$ such that either $B_{i_0} \in Q_j$ or $B_{i_1} \in Q_j$.
7) If $|\Psi| \geq \Gamma$, assign $j = j + 1$ and go to Step 3.

Based on Table 5, the first group selected by Procedure 3 with $\Gamma = 4$ is $Q_0 = \{B_1, B_5, B_3, B_7\}$. After removing the pairs that include the logic blocks of Q_0, the pairs shown in the second part of Table 5 remain. Based on these pairs, Procedure 3 constructs $Q_1 = \{B_{13}, B_{14}, B_{16}, B_{10}\}$. Table 5 also shows the pairs that remain for constructing $Q_2 = \{B_{23}, B_{26}, B_{25}, B_{22}\}$, $Q_3 = \{B_0, B_2, B_4, B_8\}$, $Q_4 = \{B_{11}, B_{18}, B_{15}, B_{17}\}$, $Q_5 = \{B_{21}, B_{24}, B_{12}, B_{20}\}$, and $Q_6 = \{B_9, B_{19}, B_6\}$. It should be noted that 27 logic blocks from Table 3 were divided into groups of four, leaving only three logic blocks for Q_6.

Table 6 shows the results obtained when W_0, W_1 and W_2 are computed for Q_0, Q_1, Q_2, Q_3, Q_4, Q_5 and Q_6. The following points can be seen from Table 6.

(1) The results in Table 6 support the expectation that significant sharing will be obtained for the groups produced by Procedure 3.
(2) As in Table 4, Procedure 2 is able to reduce the storage requirements of the shared test set significantly compared with W_0 and W_1.
(3) The reductions are overall larger for the logic blocks selected earlier, for which more sharing is expected.

VII. CONCLUDING REMARKS

Under distributed test data compression, each logic block in a design has dedicated on-chip decompression logic and compressed test set. When generating a test set that targets several fault models for a logic block, a common practice is to top off the test set by considering the fault models one by one. In this scenario, this article considered the problem of sharing a compressed test set among the logic blocks of a design. Earlier procedures that allow unrestricted sharing of compressed tests among logic blocks consider a single set of faults for every logic block, and the logic blocks one by one. They use several iterations to ensure that every logic block can benefit from sharing of compressed tests. The sharing procedure suggested in this article considers the fault models one by one. For every fault model it considers the logic blocks one by one. This structure of the procedure provides the flexibility for the procedure to share compressed tests among all the logic blocks in a single iteration, avoiding the computationally-intensive iterative parts of the earlier procedures. Experimental results for groups of four benchmark circuits demonstrated the ability of the procedure to share tests for stuck-at, single-cycle gate-exhaustive and four-way bridging faults among logic blocks.

REFERENCES

[1] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann, "OPMISR: The foundation for compressed ATPG vectors," in *Proc. Int. Test Conf.*, Nov. 2001, pp. 748–757.

[2] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, K.-H. Tsai, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eide, and J. Qian, "Embedded deterministic test for low cost manufacturing test," in *Proc. Int. Test Conf.*, Oct. 2002, pp. 301–310.

[3] N. A. Touba, "Survey of test vector compression techniques," *IEEE Design Test Comput.*, vol. 23, no. 4, pp. 294–303, Apr. 2006.

[4] P. H. Bardell, W. H. McAnney, and J. Savir, *Built—In Test for VLSI Pseudorandom Techniques*. Hoboken, NJ, USA: Wiley, 1987.

[5] Y. Huang, M. Kassab, J. Jahangiri, J. Rajski, W.-T. Cheng, D. Han, J. Kim, and K. Y. Chung, "Test compression improvement with EDT channel sharing in SoC designs," in *Proc. IEEE 23rd North Atlantic Test Workshop*, May 2014, pp. 22–31.

[6] I. Pomeranz, "Sharing of compressed tests among logic blocks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 31, no. 4, pp. 421–430, Apr. 2023.

[7] I. Pomeranz, "Dynamic test compaction of a compressed test set shared among logic blocks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 1, pp. 394–402, Jan. 2024.

[8] 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. IEEE/ACM Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 568–574.

[9] R. Desineni, K. N. Dwarkanath, and R. D. Blanton, "Universal test generation using fault tuples," in *Proc. Int. Test Conf.*, Oct. 2000, pp. 812–819.

[10] G. Chen, S. Reddy, I. Pomeranz, J. Rajski, P. Engelke, and B. Becker, "A unified fault model and test generation procedure for interconnect opens and bridges," in *Proc. Eur. Test Symp. (ETS)*, May 2005, pp. 22–27.

[11] 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. 14th Asian Test Symp. (ATS)*, Dec. 2005, pp. 330–336.

[12] D. Kim, M. E. Amyeen, S. Venkataraman, I. Pomeranz, S. Basumallick, and B. Landau, "Testing for systematic defects based on DFM guidelines," in *Proc. IEEE Int. Test Conf.*, Oct. 2007, pp. 1–10.

[13] 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. 29th VLSI Test Symp.*, May 2011, pp. 285–290.

[14] F. Hapke, W. Redemund, A. Glowatz, J. Rajski, M. Reese, M. Hustava, M. Keim, J. Schloeffel, and A. Fast, "Cell-aware test," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 9, pp. 1396–1409, Sep. 2014.

[15] C.-H. Wu and K.-J. Lee, "Transformation of multiple fault models to a unified model for ATPG efficiency enhancement," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2016, pp. 1–10.

[16] Y.-C. Kung, K.-J. Lee, and S. M. Reddy, "Generating single- and double-pattern tests for multiple CMOS fault models in one ATPG run," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 6, pp. 1340–1345, Jun. 2020.

[17] R. Asami, T. Hosokawa, M. Yoshimura, and M. Arai, "A multiple target test generation method for gate-exhaustive faults to reduce the number of test patterns using partial MaxSAT," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2020, pp. 1–6.

[18] E. J. McCluskey, "Quality and single-stuck faults," in *Proc. IEEE Int. Test Conf. - (ITC)*, Oct. 1993, p. 597.

[19] S. Sengupta, S. Kundu, S. Chakravarty, P. Parvathala, R. Galivanche, G. Kosonocky, M. Rodgers, and T. M. Mak, "Defect-based test: A key enabler for successful migration to structural test," *Intel Technol. J.*, vol. 1, pp. 1–4, Jul. 1999.

IRITH POMERANZ (Fellow, IEEE) received the B.Sc. (summa cum laude) and D.Sc. degrees from the Department of Electrical Engineering, Technion—Israel Institute of Technology, in 1985 and 1989, respectively.

From 1989 to 1990, she was a Lecturer with the Department of Computer Science, Technion—Israel Institute of Technology. From 1990 to 2000, she was a Faculty Member of the Department of Electrical and Computer Engineering, The University of Iowa. In 2000, she joined Purdue University, West Lafayette, IN, USA, where she is currently the Cadence Professor in the Elmore Family School of Electrical and Computer Engineering.

Prof. Pomeranz is a Golden Core Member of the IEEE Computer Society. She was a recipient of the NSF Young Investigator Award, in 1993, and The University of Iowa Faculty Scholar Award, in 1997. Three of her conference papers won best paper awards, and four other papers were nominated for best paper awards. One of the papers she coauthored was selected by the 2016 International Test Conference as the most significant paper published ten years before. She delivered a keynote speech at the 2006 Asian Test Symposium. She was one of the very first three featured authors on IEEE Xplore, posted in February 2020. She served as Associate Editor for *ACM Transactions on Design Automation*, IEEE Transactions on Computers, and IEEE Transactions on Very Large Scale Integration (VLSI) Systems. She served as a Guest Editor for IEEE Transactions on Computers, in January 1998, Special Issue on Dependability of Computing Systems, and the Program Co-Chair for the 1999 Fault-Tolerant Computing Symposium. She served as the Program Chair for the 2004 and 2005 VLSI Test Symposium and the General Chair for the 2006 VLSI Test Symposium.