A Partitioning and Storage Based Built-In Test Pattern Generation Method for Scan Circuits+

Irith Pomeranz
School of Electrical & Computer Eng.
Purdue University
W. Lafayette, IN 47907, USA

and

Sudhakar M. Reddy
Electrical & Computer Eng. Dept.
University of Iowa
Iowa City, IA 52242, USA

Abstract

We describe a built-in test pattern generation method for scan circuits. The method is based on partitioning and storage of test sets. Under this method, a precomputed test set is partitioned into several sets containing values of different primary inputs or state variables. The on-chip test set is obtained by implementing the Cartesian product of the various sets. The sets are reduced as much as possible before they are stored on-chip in order to reduce the storage requirements and the test application time.

1. Introduction

Storage-based methods for built-in test pattern generation store certain data on-chip. For example, the data may consist of a test set that was generated off-chip (or parts of such a test set). Storage-based methods use the stored information as part of an on-chip test pattern generator (*TPG*) in order to achieve complete fault coverage. The simplest form of a storage-based *TPG* consists of a memory that stores the complete test set, and a counter that can go through the memory addresses where the test set is stored. In this way, a precomputed test set can be applied to the circuit in full. To reduce the size of the on-chip memory required, one of the following two general approaches can be used.

Encoding techniques [1] allow the complete test set to be stored in a compressed way.

Alternatively, on-chip computations [2], [3] allow new test vectors to be obtained from existing ones. Under the method of [2], the next test vector to be applied to the circuit is obtained from the test vector currently applied by complementing single bits. The implementation of [2] stores in an on-chip memory the description of the operations to be applied to each test vector in order to obtain the next one. This method was applied to combinational circuits. The method of [3] was designed for synchronous sequential circuits. It is based on storage of short input sequences that are expanded on-chip into test sequences.

Recently [4], we proposed a new approach to storage-based test-pattern generation for synchronous sequential circuits. This method uses a simpler test application scheme than [3], reduces the overall amount of data that needs to be loaded to the chip (thus reducing the load time), and in several cases also reduces the memory requirements compared to [3]. The basic idea behind the method of [4] is demonstrated by the following example (although the method of [4] was proposed for synchronous sequential circuits, we demonstrate in the following example its application to a combinational circuit since it is closer to the application to scan circuits that we consider here). Consider a test set T = {00000, 00111, 01000, 01110, 10110, 10111} precomputed off-chip for a 5-input combinational circuit. To store this test set on-chip, we need a memory of $6 \cdot 5 = 30$ bits. Let us partition the test set into two subsets T_1 and T_2 such that T_1 contains the values of the first two inputs, and T_2 contains the values of the last three inputs. Thus, the pattern 00000 contributes 00 to T_1 and 000 to T_2, the pattern 00111 contributes 00 to T_1 and 111 to T_2, and so on. We obtain T_1 = {00, 01, 10} and T_2 = {000, 110, 111}. To store T_1 we need $3 \cdot 2 = 6$ bits and to store T_2 we need $3 \cdot 3 = 9$ bits, for a total of 15 bits. The memory requirements are thus reduced to half by partitioning the test set. To apply the original test set to the circuit, we need to apply certain pairs t_1t_2 simultaneously, where $t_1 \in T_1$ and $t_2 \in T_2$. For example, we need to apply $t_1 = 00$ and $t_2 = 000$ to obtain the first vector 00000 in T, while the pair $t_1 = 00$ and $t_2 = 110$ yields 00110 which is not in T. However, storing the pairs that need to be applied to the circuit may be a space consuming solution. Instead, we observe that T is contained in the Cartesian product $T_1 \times T_2$ of T_1 and T_2. The Cartesian product $T_1 \times T_2$ consists of every pair t_1t_2 such that $t_1 \in T_1$ and $t_2 \in T_2$. In the example, we obtain $T_1 \times T_2$ = {00000, 00110, 00111, 01000, 01110, 01111, 10000, 10110, 10111}, where the underlined vectors are in T. To implement the Cartesian product we only need two counters that will go through all the elements of T_1 and T_2. If the size of T is N, the size of T_1 and T_2 is at most N, and the size of the Cartesian product is at most N^2. Two effects help keep the number of tests in the Cartesian product significantly lower than N^2.

(1) The number of patterns in T_1 and T_2 is typically smaller than N. This is because some vectors in T contribute the same vector to T_1 or T_2. For example, both 00000 and 00111 contribute 00 to T_1 in the example above. Thus, instead of six vectors we obtained three vectors in each one of T_1 and T_2 above.

(2) We apply a procedure where we omit as many patterns as possible from T_1 and T_2. This serves to reduce the memory requirements, but also reduces the number of tests applied to the circuit. Since the Cartesian product contains more tests than T, it is typically possible to omit patterns from T_1 and T_2 and still obtain complete fault coverage when applying $T_1 \times T_2$ to the circuit. In the example above, if we omit the pattern 10 from T_1, we obtain T_1 = {00, 01}, T_2 = {000, 110, 111} and $T_1 \times T_2$ = {00000, 00110, 00111, 01000, 01110, 01111}. If this test set detects all the circuit faults, it can replace the original test set. The memory requirements are reduced by two bits, and the number of tests applied to the circuit is reduced by three tests.

It is important to note that in most storage-based methods, including [2], [3], [4] and the method proposed here, encoding [1] can be used to further reduce the memory sizes required. It is also important to note that similar to [3] and [4], the number of

+ Research supported in part by NSF Grant No. CCR-0098091, and in part by SRC Grant No. 2001-TJ-949,950.

test vectors applied to the circuit under the method proposed here is larger than the number of vectors in the precomputed test set T. Consequently, improved defect coverages are likely to be obtained. We present experimental results to support this point by showing that the extra tests applied to the circuit are effective in detecting the circuit faults multiple times [5], [6].

Although the example above was given for a combinational circuit, we consider full-scan sequential circuits in this work. We assume that a test τ_i for a full-scan circuit has the following components. (1) A scan-in vector SI_i. The vector SI_i is applied through the scan chain at the beginning of the test. (2) A primary input sequence T_i consisting of one *or more* primary input vectors. The sequence T_i is applied to the primary inputs after SI_i is scanned in. While T_i is applied, the flip-flops are driven from the next-state variables of the circuit (without using the scan chain). At the end of a test, the final state is scanned out. We use the notation $\tau_i = (SI_i, T_i)$ for a scan-based test. For example, for a circuit with four state variables and two primary inputs, a possible test is (0000, (01, 10)), where 0000 is the initial state scanned in, and (01, 10) is the sequence applied to the primary inputs following the scan operation. The importance of applying input sequences T_i of length larger than one is that it contributes to at-speed testing of the circuit [7], [8].

A test set for a scan circuit consists of tests $\tau_1, \tau_2, \cdots, \tau_N$ where $\tau_i = (SI_i, T_i)$. In contrast, for a non-scan synchronous sequential circuit, a single test sequence T_i is considered in [4], and T_i is significantly longer than the sequences T_i included in scan-based tests. The difference in test set structure results in significant differences in the partitioning and test application schemes for the two types of circuits. For example, in [4], T_i is partitioned into equal length subsequences over which the Cartesian product is defined, whereas here, the subsequences T_i are short enough to be kept intact.

Earlier works on built-in test generation for scan circuits using tests of the form (SI_i, T_i) were based on random patterns [9]-[11]. The methods of [9] and [10] do not achieve complete fault coverage, and all three methods do not guarantee that complete fault coverage would be achieved. Although the method of [11] achieves complete fault coverage for all the benchmark circuits considered in [11], this is achieved at the cost of a more complex test application scheme. The method proposed here guarantees that the same fault coverage achieved by an off-chip test set would be achieved by the test set generated on-chip.

The paper is organized as follows. In Section 2 we describe the proposed procedure for partitioning a test set $T = \{\tau_1, \tau_2, \cdots, \tau_N\}$ of a scan circuit. We also describe the procedure for reducing the number of elements in the sets obtained after partitioning. This procedure reduces the storage requirements and the number of tests applied to the circuit while ensuring that the Cartesian product would detect all the faults detectable by T. In Section 3 we consider the test application process and the hardware required for implementing it in the case of scan circuits. In Section 4 we present experimental results. Section 5 concludes the paper.

2. Test set partitioning

Let $T = \{\tau_1, \tau_2, \cdots, \tau_N\}$ be a test set for a scan circuit, with $\tau_i = (SI_i, T_i)$ for $1 \le i \le N$. In this section, we first consider the partitioning of T. We then describe a procedure for reducing the sizes of the resulting sets. For illustration, we consider the test set for ISCAS-89 benchmark circuit $s27$ shown in Table 1. The circuit has three state variables and four primary inputs. Each test is shown on a separate row in the table. Storing this test set requires 36 bits (12 bits for storing 4 scan-in vectors and 24 bits for storing 6 primary input vectors contained in the input sequences T_i).

Table 1: Test set for $s27$

i	SI_i	T_i
1	011	(0000)
2	011	(1101)
3	000	(1010)
4	110	(0100, 0111, 1001)

The first level of partitioning we apply to the test set T results in a set Ψ that contains all the scan-in vectors SI_i, and a set Σ that contains all the primary input sequences T_i. For $s27$, we obtain $\Psi = \{000, 011, 110\}$ and $\Sigma = \{(0000), (1101), (1010), (0100, 0111, 1001)\}$. This requires storage of 33 bits. The Cartesian product $\Psi \times \Sigma$ defines the test set that will be applied to the circuit. We obtain $\Psi \times \Sigma = \{(000, (0000)), (000, (1101)), (000, (1010)), (000, (0100, 0111, 1001)), (011, (0000)), \cdots, (110, (0100, 0111, 1001))\}$ containing 12 tests.

The partitioning into Ψ and Σ as defined above is motivated by the fact that the number of state variables of a circuit is typically much larger than the number of primary inputs. In addition, the primary input sequences are typically short. Consequently, it is advantageous to deal with the state variables separately. When reducing the sizes of the sets Ψ and Σ, we will give a higher priority to reducing the size of Ψ since the vectors in Ψ tend to be larger than the input sequences in Σ when the number of state variables is large.

When the number of state variables is large, it is also advantageous to further partition the set Ψ. We achieve this by dividing the set of state variables of the circuit, denoted by Y, into two or more subsets $Y_1, Y_2, \cdots, Y_k$, and including the patterns obtained on each subset Y_j in a separate set Ψ_j. For simplicity and since the number of tests in the Cartesian product grows fast with k, we use $k = 2$. For a circuit with N_{SV} state variables, we include the first $N_{SV1} = N_{SV}/2$ state variables in the first subset Y_1, and the remaining $N_{SV2} = N_{SV} - N_{SV1}$ state variables in the second subset Y_2. The patterns obtained under T on the state variables in Y_1 are included in a set Ψ_1, and the patterns obtained under T on the state variables in Y_2 are included in a set Ψ_2. The test set to be applied to the circuit is obtained by computing the Cartesian product $\Psi_1 \times \Psi_2 \times \Sigma$. For $s27$, Y_1 contains the first state variable and Y_2 contains the last two state variables. We obtain $\Psi_1 = \{0, 1\}$, $\Psi_2 = \{00, 10, 11\}$ and $\Sigma = \{(0000), (1101), (1010), (0100, 0111, 1001)\}$. Storage of these sets requires 32 bits. The Cartesian product includes $2 \cdot 3 \cdot 4 = 24$ tests.

Once the sets Ψ and Σ (or Ψ_1, Ψ_2 and Σ) are defined, we attempt to reduce their sizes as much as possible without reducing the fault coverage achieved by $\Psi \times \Sigma$ (or $\Psi_1 \times \Psi_2 \times \Sigma$). We consider the case where Ψ_1, Ψ_2 and Σ are used. The case where Ψ is not partitioned into Ψ_1 and Ψ_2 can be accommodated by setting $\Psi_1 = \Psi$ and $\Psi_2 = \phi$. Procedure 1 below describes how the sets are reduced. Procedure 1 accepts the sets Ψ_1, Ψ_2 and Σ, and the set of faults F detected by the original test set T. The procedure attempts to omit elements of Ψ_1, Ψ_2 and Σ one at a time in an order that will be explained below. After every element is omitted, the reduced test set obtained by the Cartesian product $\Psi_1 \times \Psi_2 \times \Sigma$ is fault simulated. If, during the simulation process, it turns out that a fault $f \in F$ is not detected by $\Psi_1 \times \Psi_2 \times \Sigma$, fault simulation stops and the omitted element is restored. Only if all

the faults in F are detected, the omission is accepted. It is then made final, and the omitted element is never restored.

Procedure 1 first considers the patterns in Ψ_1. To determine the order by which the patterns will be considered, we associate with every pattern $\psi_{1i} \in \Psi_1$ the number of times it appears *in the original test set* T, i.e., the number of tests in T that contain ψ_{1i}. We denote this number by $n(\psi_{1i})$. For example, for $s27$, $\psi_{11} = 0$ appears three times in T (in three different tests), and $\psi_{12} = 1$ appears once in T (in a single test). Thus, $n(\psi_{11}) = 3$ and $n(\psi_{12}) = 1$. We attempt to omit patterns that appear small numbers of times before we attempt to omit patterns that appear large numbers of times in T. The motivation for this is as follows. If ψ_{1i} appears a large number of times in T, it is likely to contribute to a large number of tests in $\Psi_1 \times \Psi_2 \times \Sigma$ that detect new faults. If we omit it, there are likely to be many other patterns that it will not be possible to omit without reducing the fault coverage. Therefore, we prefer to keep ψ_{1i} in Ψ_1. In contrast, if ψ_{1i} appears a small number of times in T, it is likely that it will be possible to omit it, and that omitting it will have a small impact on the ability to omit other patterns. In the example of $s27$, we try to omit $\psi_{12} = 1$ first, and then we try to omit $\psi_{11} = 0$. In this example, none of them can be omitted.

We repeat the same process for Ψ_2. In the case of $s27$, we have $\psi_{21} = 00$ with $n(\psi_{21}) = 1$, $\psi_{22} = 10$ with $n(\psi_{22}) = 1$, and $\psi_{23} = 11$ with $n(\psi_{23}) = 2$. We attempt to omit ψ_{21}, then ψ_{22} and finally ψ_{23}. We find that $\psi_{21} = 00$ can be omitted.

For the input sequences in Σ, the order is based on two parameters. The first parameter, $n(T_i)$, is the number of times T_i appears in T. This is similar to the parameter $n(\psi_{ij})$ used for the patterns in Ψ_1 and Ψ_2. The second parameter is the length of T_i, denoted by $L(T_i)$. We prefer to omit long sequences first, since their storage requirements are higher. We use the sequence length as the primary criterion, and the number of appearances in T to break ties. For $s27$, we have $T_1 = (0000)$, $T_2 = (1101)$, $T_3 = (1010)$ and $T_4 = (0100, 0111, 1001)$ with $n(T_i) = 1$ for $i = 1,2,3,4$, $L(T_i) = 1$ for $i = 1,2,3$ and $L(T_4) = 3$. We consider the sequences in the order $<T_4,T_1,T_2,T_3>$. We find that T_2 can be omitted.

The final result we obtain for $s27$ is $\Psi_1 = \{0, 1\}$, $\Psi_2 = \{10, 11\}$ and $\Sigma = \{(0000), (1010), (0100, 0111, 1001)\}$. Storage of these sets requires 26 bits. The number of tests applied to the circuit is $2 \cdot 2 \cdot 3 = 12$.

Procedure 1 that reduces the sets Ψ_1, Ψ_2 and Σ is given next.

Procedure 1: Reducing Ψ_1, Ψ_2 and Σ

(1) Let F be the set of faults detected by T. Mark all the patterns in Ψ_1 *unselected*.
(2) Select the *unselected* pattern $\psi_{1i} \in \Psi_1$ with the minimum value of $n(\psi_{1i})$. Mark ψ_{1i} *selected*.
(3) Omit ψ_{1i} from Ψ_1 and simulate $\Psi_1 \times \Psi_2 \times \Sigma$. If any fault in F remains undetected, restore ψ_{1i} into Ψ_1.
(4) If there are any *unselected* patterns in Ψ_1, go to Step 2.
(5) Mark all the patterns in Ψ_2 *unselected*.
(6) If there are no *unselected* patterns in Ψ_2, go to Step 10.
(7) Select the *unselected* pattern $\psi_{2i} \in \Psi_2$ with the minimum value of $n(\psi_{2i})$. Mark ψ_{2i} *selected*.
(8) Omit ψ_{2i} from Ψ_2 and simulate $\Psi_1 \times \Psi_2 \times \Sigma$. If any fault in F remains undetected, restore ψ_{2i} into Ψ_2.
(9) Go to Step 6.
(10) Mark all the input sequences in Σ *unselected*.
(11) Select the *unselected* sequence $T_i \in \Sigma$ that has the maximum length. If a choice exists, select T_i with the minimum value of $n(T_i)$. Mark T_i *selected*.
(12) Omit T_i from Σ and simulate $\Psi_1 \times \Psi_2 \times \Sigma$. If any fault in F remains undetected, restore T_i into Σ.
(13) If there are any *unselected* sequences in Σ, go to Step 11.

Note that the Cartesian product $\Psi_1 \times \Psi_2 \times \Sigma$ does not have to be maintained explicitly. Instead, the tests can be generated as needed. This prevents excessive storage requirements during fault simulation. Fault simulation time is reduced by stopping the simulation of $\Psi_1 \times \Psi_2 \times \Sigma$ as soon as an undetected fault which was detected by T is identified.

To further reduce the fault simulation time in Procedure 1, we observe that for every fault $f \in F$, it is possible to express the test $\tau \in T$ that detects it as a combination $\psi_{1i_1} \in \Psi_1$, $\psi_{2i_2} \in \Psi_2$ and $T_{i_3} \in \Sigma$, where Ψ_1, Ψ_2 and Σ are the original sets found based on T. If ψ_{1i_1}, ψ_{2i_2} and T_{i_3} are still in the corresponding sets, f is guaranteed to be detected by the Cartesian product and need not be simulated. Every time f has to be simulated again under the Cartesian product because ψ_{1i_1}, ψ_{2i_2} or T_{i_3} is removed from the corresponding set, we store the new combination that allows f to be detected. If the removal is accepted, we update ψ_{1i_1}, ψ_{2i_2} and T_{i_3} according to the new combination. In this way, we minimize the number of times a fault f has to be simulated after removing an element from one of the sets.

3. Hardware implementation

In this section, we compare the hardware required to implement the proposed partitioning-based method with the hardware required when the complete test set is stored.

One way to store the complete test set is by using the following memories (other options exist, but they have similar overheads). The memory referred to as SI stores the scan-in vectors. The ith memory entry, $SI[i]$, is the scan-in vector SI_i of τ_i. The memory referred to as T stores the test sequences as consecutive input vectors. To mark where a sequence starts and where it ends, we use a memory called B such that $B[i]$ marks the beginning of T_i, and a memory called L such that $L[i]$ is the length of T_i. The following procedure needs to be implemented in hardware to apply the test set to the circuit.

For $i = 0$ to $N-1$ (where N is the number of tests in the test set):
  Scan in $SI[i]$.
  For $u = 0$ to $L[i]-1$:
    Apply to the primary inputs the vector $T[B[i]+u]$.

This implementation requires two counters, i and u.

The test set partitioned as proposed here can be stored using the following memories. The memory referred to as Ψ_1 stores the vectors contained in Ψ_1. The ith entry, $\Psi_1[i]$, is equal to the vector $\psi_{1i} \in \Psi_1$. The memory referred to as Ψ_2 stores the vectors contained in Ψ_2. The entry $\Psi_2[i]$ is equal to the vector $\psi_{2i} \in \Psi_2$. The memory referred to as Σ stores the test sequences in Σ as consecutive input vectors. A memory called B is arranged such that $B[i]$ marks the beginning of T_i, and a memory called L is arranged such that $L[i]$ is the length of T_i. The following procedure needs to be implemented in hardware to apply the test set to the circuit.

For $i_1 = 0$ to N_1–1 (where N_1 is the number of vectors in Ψ_1):
  For $i_2 = 0$ to N_2–1 (where N_2 is the number of vectors in Ψ_2):
    For $i_3 = 0$ to N_3–1 (where N_3 is the number of input sequences in Σ):
      Scan in the vector consisting of $\Psi_1[i_1]$ followed by $\Psi_2[i_2]$.
      For $u = 0$ to $L[i_3]$–1:
        Apply to the primary inputs the vector $T[B[i_3]+u]$.

This implementation requires four counters, i_1, i_2, i_3 and u. Except for the difference in the number of counters, the main difference in size between the two implementations results from the memory sizes. Experimental results presented in the following section show that the sizes are significantly reduced by the proposed method.

4. Experimental results

We applied the proposed built-in test generation method to ISCAS-89 and ITC-99 benchmark circuits. In two separate experiments, we used two precomputed test sets for every circuit. The first test set is derived from a combinational test set (a test set designed for the combinational logic of the circuit). For ISCAS-89 benchmark circuits, the combinational test set is the compacted test set generated by the procedure from [12]. For ITC-99 benchmark circuits, a combinational test set is obtained by applying 100,000 combinational test patterns and including in the test set only the patterns that detect new faults. A combinational test c_i is transformed into a scan-based test τ_i as follows. The values of the state variables obtained under c_i are assigned to the scan-in vector SI_i; and the primary input vector obtained under c_i is included in a test sequence T_i of length one. Starting from the test set obtained in this way, we apply the static compaction procedure of [13]. After static compaction, the number of tests is lower and the lengths of the sequences T_i is higher (the lower number of tests reduces the number of scan operations and the test application time; the total length of all the primary input sequences is kept the same or it is reduced by compaction). Since the test sets obtained in this way tend to contain very short input sequences, we also consider the test sets generated by the simulation-based test generation procedure of [14]. These test sets contain longer input sequences, and in most cases also a larger number of tests.

Information about the circuits we consider is shown in Table 2. Information about the test sets is shown later in Tables 3 and 4. In Table 2, after the circuit name we show the number of primary inputs and the number of state variables. We then show the number of faults and the number of faults detected by the test sets we consider. For ISCAS-89 benchmark circuits, all the detectable faults are detected by the test sets we use.

In Table 3, we show the following information for the test sets obtained by static compaction [13]. Under column *original*, we show the number of tests in T, the length of all the input sequences T_i in T, and the number of bits required to store T. Under column *partitioned*, we show the results obtained after partitioning T but before applying Procedure 1 to reduce the resulting set sizes. In all the cases, the test set is partitioned into three sets, Ψ_1, Ψ_2 and Σ. Under subcolumn *st*1 we show the number of vectors in Ψ_1, under subcolumn *st*2 we show the number of vectors in Ψ_2, under subcolumn *seq* we show the number of input sequences in Σ, and under subcolumn *len* we show the total length of all the sequences in Σ. Under column *final* we show the results obtained after applying Procedure 1. In addition to the sizes of Ψ_1, Ψ_2 and Σ shown under subcolumns *st*1, *st*2, *seq* and *len*, we show under subcolumn *tst* the number of tests applied to the circuit. This is the size of the Cartesian product $\Psi_1 \times \Psi_2 \times \Sigma$. Under subcolumn *stor* we show the storage requirements of the partitioned and reduced sets Ψ_1, Ψ_2 and Σ in bits. Under column *ratio* we show the storage requirements of the proposed method divided by the storage requirements for the original test set. In the last row of Table 3 we show the total storage requirements and the average ratio. In the last column of Table 3 we show the normalized run time of Procedure 1. The run time is normalized by dividing it by the time it takes to fault simulate the original test set. It is important to note that the original test set is small and fault simulation time for this test set is very short. The run time of Procedure 1 can be further reduced by incorporating techniques to speed-up the identification of elements that cannot be removed from the partitioned test set.

Table 2: Circuit parameters

circuit	inp	s.v.	flts	det
s208	11	8	215	215
s298	3	14	308	308
s344	9	15	342	342
s382	3	21	399	399
s386	7	6	384	384
s400	3	21	421	415
s420	19	16	430	430
s510	19	6	564	564
s526	3	21	555	554
s641	35	19	467	467
s820	18	5	850	850
s953	16	29	1079	1079
s1423	17	74	1515	1501
s1488	8	6	1486	1486
b01	3	5	135	135
b02	2	4	70	70
b03	5	30	452	452
b04	12	66	1346	1344
b06	3	9	202	202
b09	2	28	420	420
b10	12	17	512	512
b11	8	30	1089	1078

Storage requirements are computed as follows. For the original test set T, let the number of tests in T be N and let the length of T_i be L_i. The total length of all the sequences T_i in T is $L = \sum_{i=1}^{N} L_i$. The number of bits required to store T is $NN_{SV}+LN_{PI}$, where N_{SV} is the number of state variables and N_{PI} is the number of primary inputs. The first component of the sum corresponds to storage of scan-in vectors, and the second component corresponds to storage of the input sequences. We ignore the memories required to store beginnings and lengths of sequences since they depend on the implementation, and they can only be reduced by the proposed method. For a partitioned test set with sets Ψ_1, Ψ_2 and Σ, let the size of Ψ_1 be N_1, let the size of Ψ_2 be N_2, and let the size of Σ be N_3. The number of bits required to store these sets is $N_1N_{SV1}+N_2N_{SV2}+LN_{PI}$, where N_{SV1} is the number of state variables whose vectors are stored in Ψ_1, N_{SV2} is the number of state variables whose vectors are stored in Ψ_2, and L is the length of all the input sequences in Σ, i.e., $L = \sum_{i=1}^{N_3} L_i$.

Table 3: Results for test sets obtained by static compaction

	original			partitioned			
circuit	tst	len	stor	st1	st2	seq	len
s208	23	27	481	10	14	23	27
s298	20	24	352	19	17	10	14
s344	11	15	300	10	8	5	8
s382	23	25	558	23	23	8	10
s386	42	70	742	8	8	36	64
s400	20	24	492	19	20	10	13
s420	40	43	1457	23	32	24	27
s510	23	54	1164	8	8	20	51
s526	44	50	1074	36	38	11	17
s641	15	22	1055	11	14	15	22
s820	42	94	1902	4	8	42	94
s953	19	76	1767	17	19	19	76
s1423	26	26	2366	26	26	26	26
s1488	38	101	1036	8	8	38	101
b01	5	24	97	4	4	5	24
b02	6	13	50	3	2	5	12
b03	22	34	830	22	22	17	29
b04	30	69	2808	30	30	30	69
b06	8	20	132	6	7	7	19
b09	22	36	688	22	22	10	18
b10	25	72	1289	23	25	25	72
b11	40	85	1880	40	39	40	85
total			22520				

	final							
circuit	tst	st1	st2	seq	len	stor	ratio	n.time
s208	343	7	7	7	9	155	0.32	131.27
s298	108	12	3	3	5	120	0.34	167.08
s344	75	5	5	3	5	120	0.40	23.79
s382	132	4	11	3	4	173	0.31	114.73
s386	420	6	7	10	20	179	0.24	19.41
s400	156	4	13	3	4	195	0.40	94.73
s420	5202	17	17	18	20	652	0.45	339.63
s510	420	5	7	12	33	663	0.57	26.75
s526	512	16	8	4	6	266	0.25	336.30
s641	330	5	6	11	18	735	0.70	37.39
s820	544	4	8	17	50	932	0.49	16.00
s953	132	12	1	11	63	1191	0.67	95.22
s1423	3672	17	24	9	9	1670	0.71	339.35
s1488	385	5	7	11	38	340	0.33	23.30
b01	16	2	2	4	21	73	0.75	4.00
b02	12	2	2	3	9	26	0.52	NA
b03	175	5	7	5	12	240	0.29	185.36
b04	594	6	9	11	37	939	0.33	394.95
b06	36	3	4	3	8	56	0.42	7.88
b09	210	7	10	3	4	246	0.36	100.28
b10	200	5	5	8	20	325	0.25	347.33
b11	456	8	19	3	7	461	0.25	758.57
total						9757	0.43	

The following points can be seen from Table 3. We first consider the results after partitioning T but before applying Procedure 1. In the worst case, if the number of tests in T is N, the numbers of vectors in Ψ_1 and Ψ_2 and the number of sequences in Σ are also N. In most cases, the numbers are smaller than N; however, they are not significantly smaller. This implies that partitioning alone does not reduce the storage requirements significantly. For example, for $s208$, the test set obtained by static compaction contains 23 tests. After partitioning, the numbers of vectors in Ψ_1 and Ψ_2 are 10 and 14, respectively, but the number of input sequences in Σ is 23. In this case, all the input sequences in T are different. Procedure 1 reduces the numbers of vectors in Ψ_1 and Ψ_2, and the number of input sequences in Σ substantially. As a result, it reduces the storage requirements, in most cases, to less than half of the original value. For $s208$, the numbers of vectors in Ψ_1 and Ψ_2 and the number of sequences in Σ are all 7. The storage requirements are reduced from 481 bits to 155 bits, or 0.32 of the original value.

Results using the test sets from [14] are shown in Table 4. The original test sets in this case are larger than the test sets of Table 3, which were obtained by static compaction. Consequently, their storage requirements are higher.

Table 4: Results for test sets from [14]

	original			partitioned			
circuit	tst	len	stor	st1	st2	seq	len
s208	29	50	782	10	16	29	50
s298	16	124	596	16	15	15	122
s344	11	92	993	10	11	11	92
s382	26	194	1128	26	26	21	173
s420	58	75	2353	37	51	58	75
s526	43	378	2037	41	43	36	320
s641	34	332	12266	32	34	34	332
s820	91	221	4433	4	8	91	221
s1423	49	770	16716	49	49	49	770
s1488	78	218	2212	8	8	77	216
b01	11	32	151	4	6	11	32
b02	8	19	70	4	2	7	18
b03	22	122	1270	19	19	22	122
b04	32	889	12780	27	27	32	889
b06	13	28	201	7	11	9	21
b09	30	143	1126	29	29	16	96
b10	33	176	2673	29	29	33	176
b11	35	515	5170	34	34	35	515
total			66957				

	final						
circuit	tst	st1	st2	seq	len	stor	ratio
s208	288	6	8	6	15	221	0.28
s298	24	4	2	3	30	132	0.22
s344	48	3	4	4	43	440	0.44
s382	112	4	7	4	37	228	0.20
s420	1287	11	13	9	14	458	0.19
s526	192	6	8	4	25	223	0.11
s641	120	3	2	20	208	7327	0.60
s820	608	4	8	19	30	572	0.13
s1423	630	5	9	14	239	4581	0.27
s1488	576	6	6	16	39	348	0.16
b01	18	1	3	6	22	77	0.51
b02	12	2	2	3	6	20	0.29
b03	168	4	6	7	32	310	0.24
b04	228	3	4	19	664	8199	0.64
b06	60	3	4	5	8	56	0.28
b09	165	5	11	3	19	262	0.23
b10	169	4	4	10	50	668	0.25
b11	420	4	7	15	245	2125	0.41
total						26247	0.30

From Tables 3 and 4 it can be seen that the number of tests applied by the proposed method (the size of the Cartesian product) is larger than the number of tests in the original test set T. Next, we provide evidence to show that the extra tests applied under the proposed method are not arbitrary tests with respect to the circuit-under-test, but rather high-quality tests that detect large numbers of faults. It was established in earlier works ([5], [6]) that n-detection test sets for stuck-at faults (i.e., test sets that detect each stuck-at fault n times, where $n > 1$) are effective in detecting defects. Using this result, showing that the

tests applied by the proposed method achieve large numbers of detections of stuck-at faults will support the argument that the proposed method, by applying a number of tests which is larger than necessary, improves the defect coverage.

Results regarding the numbers of detections achieved by the proposed method are given in Table 5. The original test set in this case is the one obtained by static compaction [13]. Table 5 is organized as follows. After the circuit name, we show the minimum and the maximum number of times a stuck-at fault is detected by the tests generated by the proposed method. We then show the average number of times a stuck-at fault is detected. For comparison, we show the same information for the original test set. The following points can be seen from Table 5.

Table 5: Numbers of detections

	proposed			original		
circuit	min	max	ave	min	max	ave
s208	1	337	71.51	1	22	5.58
s298	1	108	29.68	1	17	5.02
s344	1	75	24.61	1	11	3.44
s382	1	132	30.15	1	22	5.32
s386	1	420	55.48	1	42	5.25
s400	1	156	35.75	1	19	4.84
s420	1	4986	934.16	1	38	9.13
s510	1	420	123.93	1	23	5.80
s526	1	444	97.17	1	40	8.60
s641	1	328	111.75	1	15	5.03
s820	1	543	61.80	1	42	5.05
s953	1	132	43.65	1	19	6.16
s1423	1	3264	868.19	1	23	6.42
s1488	1	385	71.49	1	38	6.04
b01	1	16	10.26	1	5	3.12
b02	1	12	5.86	1	6	2.31
b03	2	165	64.28	1	21	6.38
b04	1	574	184.38	1	29	8.52
b06	1	36	14.25	1	8	3.22
b09	1	201	53.21	1	21	6.10
b10	1	200	50.28	1	25	6.78
b11	1	428	92.62	1	39	9.04

The minimum number of times a fault is detected by the Cartesian product is in most cases one. This is a result of the fact that the subsets Ψ_1, Ψ_2 and Σ are minimized as much as possible. Specifically, the omission of any additional entry from either one of these subsets will leave a fault undetected. The faults that prevent the omission of another entry are the ones detected only once. A similar situation can be seen for the original test set which is a compacted test set.

The maximum and average numbers of detections are significantly higher for the Cartesian product than for the original test set. This supports our argument that the Cartesian product includes tests of high quality that detect large numbers of faults.

5. Concluding remarks

We described a partitioning and storage based built-in test pattern generation method for full-scan circuits. Under the proposed method, a precomputed test set is partitioned into several sets. One or more of the sets contain values of state variables. The remaining set contains input sequences. The on-chip test set is obtained by implementing the Cartesian product of the various sets. The sets were reduced as much as possible by omitting vectors or input sequences in order to reduce the storage requirements and the test application time.

In the application to full-scan circuits, we used the fact that in most cases, the number of primary inputs is smaller than the number of state variables, and the primary input sequences are relatively short. Consequently, we partitioned the scan-in vectors, but we did not partition the input sequences. If necessary, it is possible to partition the input sequences by partitioning the set of primary inputs (if it is large), or by partitioning the input sequences into subsequences of limited lengths (if the input sequences are long).

To further reduce the storage requirements, it is possible to use the proposed method in conjunction with a random pattern generator, and apply it only to faults that remain undetected after random pattern generation. This would reduce the sizes of the sets that need to be stored on-chip.

References

[1] V. Iyengar, K. Chakrabarty, and B. T. Murray "Built-in Self Testing of Sequential Circuits Using Precomputed Test Sets," in Proc. VLSI Test Symp., April 1998, pp. 418-422.

[2] R. Dandapani, J. H. Patel and J. A. Abraham, "Design of Test Pattern Generation for Built-In Test", in Proc. Intl. Test Conf., 1984, pp. 315-319.

[3] I. Pomeranz and S. M. Reddy, "Built-In Test Sequence Generation for Synchronous Sequential Circuits Based on Loading and Expansion of Test Subsequences", in Proc. 36th Design Autom. Conf., June 1999, pp. 754-759.

[4] I. Pomeranz and S. M. Reddy, "A Partitioning and Storage Based Built-In Test Pattern Generation Method for Synchronous Sequential Circuits", in Proc. Intl. Conf. on Computer-Design, Sept. 2001.

[5] S. C. Ma, P. Franco and E. J. McCluskey, "An Experimental Chip to Evaluate Test Techniques Experiment Results", in Proc. 1995 Intl. Test Conf., Oct. 1995, pp. 663-672.

[6] J. T.-Y. Chang, C.-W. Tseng, C.-M. J. Li, M. Purtell and E. J. McCluskey, "Analysis of Pattern-Dependent and Timing-Dependent Failures in an Experimental Test Chip", in Proc. 1998 Intl. Test Conf., Oct. 1998, pp. 184-193.

[7] P. C. Maxwell, R. C. Aitken, K. R. Kollitz and A. C. Brown, "IDDQ and AC Scan: The War Against Unmodelled Defects", in Proc. 1996 Intl. Test Conf., Oct. 1996, pp. 250-258.

[8] "Best Methods for At-Speed Testing?", Panel 3, 16th VLSI Test Symp., April 1998, p. 460.

[9] H.-C. Tsai, K.-T. Cheng and S. Bhawmik, "Improving the Test Quality for Scan-based BIST Using General Test Application Scheme", in Proc. Design Autom. Conf., June 1999, pp. 748-753.

[10] Y. Huang, I. Pomeranz, S. M. Reddy and J. Rajski, "Improving the Proportion of At-Speed Tests in Scan BIST", in Proc. Intl. Conf. on Computer-Aided Design, Nov. 2000.

[11] I. Pomeranz, "Random Limited-Scan to Improve Random Pattern Testing of Scan Circuits", in Proc. 38th Design Autom. Conf., June 2001, pp. 145-150.

[12] S. Kajihara, I. Pomeranz, K. Kinoshita and S. M. Reddy, "Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits", IEEE Trans. on Computer-Aided Design, Dec. 1995, pp. 1496-1504.

[13] I. Pomeranz and S. M. Reddy, "Static Test Compaction for Scan-Based Designs to Reduce Test Application Time", in Proc. 7th Asian Test Symp., Dec. 1998, pp. 198-203.

[14] I. Pomeranz and S. M. Reddy, "Simulation Based Test Generation for Scan Designs", in Proc. Intl. Conf. on Computer-Aided Design, Nov. 2000, pp. 544-549.