An Efficient Test Relaxation Technique for Combinational & Full-Scan Sequential Circuits

Aiman El-Maleh and Ali Al-Suwaiyan
King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia
emails:{aimane, abosaleh}@ccse.kfupm.edu.sa

Abstract

Reducing test data size is one of the major challenges in testing systems-on-a-chip. This problem can be solved by test compaction and/or compression techniques. Having a partially specified or relaxed test set increases the effectiveness of test compaction and compression techniques. In this paper, we propose a novel and efficient test relaxation technique for combinational and full-scan sequential circuits. The proposed technique is faster than the brute-force test relaxation method by several orders of magnitude. The application of the technique in improving the effectiveness of test compaction and compression is illustrated.

1. Introduction

Recent VLSI technology enables us to implement very large systems containing millions of transistors on a single chip. The large volume of test data that has to be stored in the memory of the tester, and transferred between the tester and the chip is one of the major challenges in testing Systems-on-a-Chip (SOC) [13].

The problem of large test storage requirement has been solved by two techniques in the literature, namely test compaction and test compression. The goal of test compaction is to reduce (or compact) the number of test vectors into a smaller number that achieves the same fault coverage. Two types of test compaction exist, static compaction and dynamic compaction. In static compaction, the number of test vectors is reduced after they are generated, whereas in dynamic compaction, the number of test vectors is minimized during the automatic test pattern generation (ATPG) process. Examples of static compaction techniques include reverse order fault simulation [9], forced pair merging [7], and redundant vector elimination (RVE) [5]. Examples of dynamic compaction techniques include COMPACTEST [6], and bottleneck removal [11]. The objective of test set compression is to reduce the number of bits needed to represent the test set. For test data compression, it is essential that the compression is lossless. Several test compression techniques have been proposed [1] [2] [3].

Compaction and compression techniques can achieve better results if the test set is composed of test cubes, i.e., if the test set is partially specified or relaxed. In fact, most compression techniques in the literature assume a relaxed test set. Without the dynamic compaction option, ATPGs generally generate fully specified test sets. The problem of test relaxation has not been solved effectively in the literature. It can be defined as follows. *Given a combinational circuit and a fully specified test set, generate a partially specified test set that maintains the same fault coverage while maximizing the number of unspecified bits*. One obvious way to solve this problem is to use a brute-force technique, where every bit of the test vectors is tested whether changing it to an x reduces the fault coverage or not. This technique has a complexity of $O(nm)$ fault simulation runs, where n is the width of one vector, and m is the number of vectors. In each fault simulation run, only the detected faults by a particular vector are simulated. It should be noted that this technique does not produce an optimal solution as it follows a particular order. Obviously, this technique is impractical for large circuits.

A partially specified test set can also be obtained using dynamic ATPG compaction. In dynamic compaction, every partially specified vector is processed immediately after its generation by trying to specify primary inputs (PIs) that are unspecified so that it will detect additional new faults. Generally, in dynamic ATPG compaction, the remaining unspecified assignments on primary inputs are filled with random values. However, this feature can be disabled to obtain a compact and relaxed test set. This way of generating a relaxed test set slows down the test generation process. In addition, dynamic ATPG compaction cannot benefit from random test pattern generation, because this technique is fault oriented. Furthermore, this technique does not solve the problem of relaxing an already existing test set. Thus, effectively, the only existing solution to the problem of relaxing a given test set is the brute-force method.

In addition to improving the effectiveness of test com-

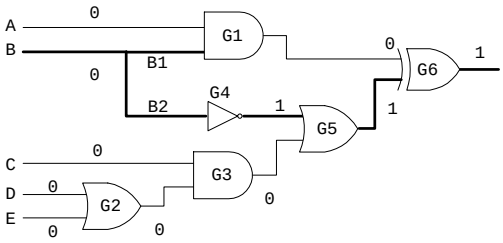

Figure 1. Circuit of Example 1.

paction and compression techniques, test relaxation can also help in test power reduction. The relaxed bits can be specified in a way that reduces the number of transitions during scan and hence reduce test power dissipation [10].

In this paper, we propose a novel and efficient test relaxation technique for combinational and full-scan sequential circuits that maximizes the number of unspecified bits while maintaining the same fault coverage as the one obtained by the original test set. In brief words, the algorithm does the following for every test vector t of the test set. First, it fault-simulates the circuit under the test vector t and generates a list of newly detected faults. Then, for every newly detected fault f, it marks all the lines whose values are required for f to be detected (i.e., excited and propagated to a primary output). Obviously, the unmarked input lines are not required for fault detection, and thus they are relaxed. As compared to the brute-force method, our technique is faster by several orders of magnitude.

This paper is organized as follows. Section 2 illustrates our idea by an example. Section 3 formally describes our test relaxation algorithm. Experimental results are given in section 4. This section also shows the effectiveness of our technique in improving test compaction and compression techniques. Finally, the paper ends by a conclusion.

2. An Illustrative Example

In this section, we demonstrate our proposed test relaxation technique by an example. The following conventions are assumed. To indicate that a line l is stuck at value v, we use the notation l/v. The notation $A = v/\bar{v}$ is used to indicate that the fault-free value of line A is v, and the faulty value of line A is $\bar{v}$. When we say that line l is required, we mean that the value on line l is required.

Example 1: Consider the circuit shown in Figure 1. Suppose that we apply the test vector $ABCDE = 00000$. Under this test, the faults $G6/0$, $G5/0$, $G1/1$, $G4/0$, $B2/1$, and $B/1$ are detected. Assume that the newly detected fault is only $B/1$, i.e., other faults are either previously detected by an earlier test vector, or not part of the fault list. For the fault $B/1$ to be detected, it has to be activated (excited) and propagated to the primary output $G6$. The assignment $B = 0$ is required for exciting the fault. The assignments $G3 = 0$ and $G1 = 0$ are required for fault propagation. The assignment $G3 = 0$ can be satisfied by either one of the two assignments $C = 0$, or $DE = 00$. If we choose to satisfy $G3 = 0$ by the assignment $C = 0$, then the assignment $DE = 00$ is no longer necessary, and this implies that we can relax CDE to $0xx$. Similarly, if we choose to satisfy $G3 = 0$ by the assignment $DE = 00$, then CDE can be relaxed to $x00$. This shows that a test vector can be relaxed in more than one way, and some ways might have more relaxed bits than others.

The other requirement for fault propagation, which is $G1 = 0$, appears to be already satisfied because we already have marked the assignment $B = 0$ as required, and this assignment produces $G1 = 0$. This results in an incorrect relaxation of the input A. To show this, assume that stem B is faulty, i.e., $B = 0/1$. In this case, if line A is relaxed, the fault on the stem will not propagate to the output. It will be masked by the x value on line A, producing the value $1/x$ on the output $G6$. The problem occurs because we justified the requirement on line $G1$ from line $B1$, which is reachable from the fault on the stem B. Justifying a required value from a reachable line guarantees that the required value is satisfied in the fault-free machine but not in the faulty machine. This problem can be avoided by justifying the required value from an unreachable line. This guarantees that the value will be satisfied for both the fault-free and the faulty machines. For this example, the required value on line $G1$ has to be satisfied by marking line A as required, resulting in the test vector $ABCDE = 100xx$, or $ABCDE = 10x00$. This example shows that we need to identify lines that are reachable from faulty stems before justifying the required values. In Figure 1, reachable lines from stem B are $B1$, $B2$, $G4$, $G5$, and $G6$.

After this introductory example, we describe our technique in a formal way.

3. Proposed Technique

Before explaining the proposed technique, let's have the following definitions.

Definition 1 *A gate input is said to be sensitive in a test vector t if complementing its value changes the value of the gate output from v to $\bar{v}$ where $v \in \{0, 1\}$ [8].*

Definition 2 *Let line l be an input of a gate G. The output of line l is defined to be the output of G.*

Definition 3 *Let line l be an input of a gate G. The side inputs of line l are defined to be the input lines of G other than l.*

Algorithm 1 Main Algorithm

```
for every test vector t do
    Fault simulate the circuit under the test t
    for every newly detected fault f do
        BuildRequirementList(f)
        for every line l in L do
            justify(l)
        end for
        Mark all lines as unreachable
    end for
    Output relaxed vector
    Mark all Lines as non-required
end for
```

Definition 4 *The requirement list L of a given fault is the set of lines whose values are required to detect (i.e., excite and propagate) that given fault.*

Definition 5 *A line l is said to be reachable from a stem s if the fault effect in stem s reaches line l.*

Algorithm 1 shows an outline of the proposed test relaxation technique. For every test vector t, the main algorithm proceeds as follows. It fault simulates the circuit to identify newly detected faults. Then, for every newly detected fault f, it builds the requirement list L of f, and justifies these requirements. In what follows, we explain each one of these tasks.

Algorithm 2 builds the requirement list L for the fault f. Assuming l is the faulty line, the algorithm works as follows. It adds l to the requirement list (fault activation). After the while loop is over, l is either an output or a fanout stem. In either case, the requirement for propagating f to l is stored in L. However, if l is an output, the algorithm terminates, because the requirements to detect f are completed. This situation occurs when the propagation path of f does not contain any fanout stem. Otherwise, it calls an event driven procedure for marking reachable lines from the fanout stem l, i.e., $MarkReachableLines$ shown in Algorithm 3. This procedure returns an output o in which f propagates to. The $BuildRequirementList$ procedure continues by tracing the propagation path of f starting from the output o. It adds all unreachable inputs in the propagation path of f to the requirement list L (to ensure fault propagation).

In the $MarkReachableLines$ algorithm, once an output is reached, the loop is over and that output will be returned. The function $Reachable(l, j)$ in the algorithm returns true if and only if the fault effect in stem j reaches the line l. The following lemmas provide the rules used by the function $Reachable(l, j)$.

Lemma 1 *Let l be the output of an AND, NAND, OR, or NOR gate. Then l is reachable from stem s iff one of the following conditions is satisfied:*

1. *Only sensitive inputs of l are reachable from stem s.*
2. *Only the non-sensitive inputs of l having controlling value are reachable from stem s, and none of the other gate inputs has an x value.*

Lemma 2 *Let l be the output of a 2-input XOR/XNOR gate. Then l is reachable from stem s iff only one input is reachable from stem s, and the other input does not have an x value.*

Algorithm 4 is the justification algorithm that is used in the main algorithm to justify the requirements. Assuming that line l is to be justified, the algorithm proceeds as follows. If l is a primary input (PI), the algorithm marks it as required and returns. If l is a single-input, XOR or XNOR gate, all the values on l's inputs have to be justified. Similarly, all the values on the inputs of l have to be justified if l has a non-controlling value (assuming 0-inversion). However, if l has a controlling value, then we need to check if it has an unreachable input with a controlling value. If it has, then it is sufficient to justify the value using that unreachable input. Otherwise, all the values on the inputs will be

Algorithm 2 BuildRequirementList(f)

```
Let l be the line that has the fault f
Add l to L
while l is not a fanout stem nor a primary output do
    Add side inputs of l to L
    l ← output of l
end while
if l is a primary output then
    Return
end if
Let s = l
o ← MarkReachableLines(l)
Let E ← ∅
Add o to E
while E ≠ ∅ do
    Let j be an element in E
    Remove j from E
    for every input i of j do
        if i ≠ s then
            if i is reachable then
                Add i to E
            else
                Add i to L
            end if
        end if
    end for
end while
```

Algorithm 3 MarkReachableLines(l)

```
Let E ← ∅
Let j ← l
Add l to E
while E ≠ ∅ do
  Let l ← element in E with minimal level
  remove l from E
  if Reachable(l, j) then
    if l is a primary output then
      Mark l as reachable from j
      Return l
    else if l is not a fanout stem then
      Mark l as reachable from j
      Add output of l to E
    else
      for every fanout branch b of l do
        Mark b as reachable from j
        Add output of b to E
      end for
    end if
  end if
end while
```

justified. This situation occurs when l can only be justified from a reachable line.

Note that in justifying a required controlling value, there could be several unreachable inputs with controlling value. In this case, priority is given to an input that is already marked as required. Otherwise, cost functions are used to guide the selection. Our objective from cost functions is to justify the required values by the smallest number of assignments on the primary inputs. This will result in increasing the number of x's extracted from relaxing a test vector. Cost functions are used to provide a relative measure on the selection that reduces the number of required assignments on the PIs.

The well-known recursive controllability cost functions [8] can be used for this purpose as they give a relative measure of the number of PI assignments required to justify a required value. For every line l, we compute two cost functions $C_0(l)$ and $C_1(l)$. For example, for an AND gate whose output is l and that has i inputs, the cost functions are computed as:

$$C_0(l) = \min_i C_0(i)$$

$$C_1(l) = \sum_i C_1(i)$$

The cost is computed for other gates in a similar way. Initially, $C_0(l)$ and $C_1(l)$ are assigned a value of 1 for PIs. In our work, we call these cost functions the regular cost functions. Regular cost functions are accurate for fanout-free

Algorithm 4 justify(l)

```
if if l is a PI then
  mark l as required
else if l is an output of a one input, XOR, or XNOR gate,
or l has a non-controlling value then
  for every input j of l do
    justify(j)
  end for
else if there is an unreachable input line j of l with controlling value then
  justify(j)
else
  for every input j of l do
    justify(j)
  end for
end if
```

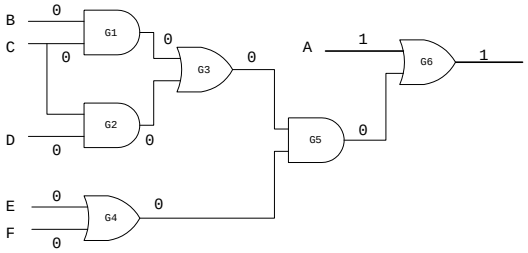

Figure 2. Illustration of selection criteria.

circuits. However, when fanouts exist, regular cost functions do not take advantage of the fact that a stem can justify several required values.

To take advantage of that, we propose new cost functions, called the fanout-based cost functions. Note that these cost functions are different from the fanout-based cost functions given in [8]. These functions are computed for an AND gate as follows. Let l be the output of an AND gate with i inputs. Let $F(l)$ denote the fanout (i.e., the number of fanout branches) of line l. Then, the fanout-based cost functions are computed as:

$$C_0(l) = \frac{\min_i C_0(i)}{F(l)}$$

$$C_1(l) = \frac{\sum_i C_1(i)}{F(l)}$$

The advantage of the proposed fanout-based cost functions over the regular cost functions given in [8] is illustrated by the following example.

Example 2: Consider the circuit shown in Figure 2. The detected faults under the shown test vector are $G6/0$ and $A/0$. The value 0 on $G5$ is required. Note that this value can be justified either through $G3$ or $G4$. If the regular cost functions, given in [8], are used then $C_0(G3) = 2$ and

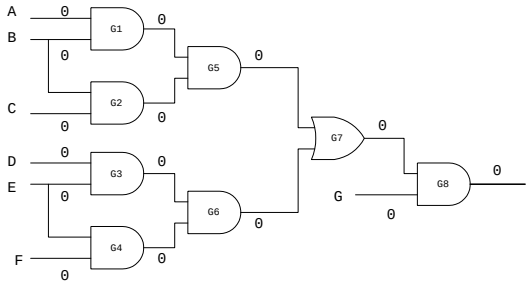

Figure 3. Another illustration of selection criteria.

$C_0(G4) = 2$. If $G4$ is selected this will result in the two required values $E = 0$ and $F = 0$ and the test vector will be relaxed to $ABCDEF = 1xxx00$. If $G3$ is selected, then this may result in either of the following assignments $\{B = 0, C = 0\}$, $\{B = 0, D = 0\}$, $\{C = 0, D = 0\}$, or $\{C = 0\}$. According to this selection criterion, any of these choices is possible since they have the same cost. However, if the fanout-based criterion is used, then $C_0(G3) = 1$ and $C_0(G4) = 2$, which will select $G3$ to justify the value. Now in justifying $G1 = 0$, the assignment $C = 0$ will be selected since $C_0(C) = 1/2$ while $C_0(B) = 1$. Similarly, in justifying the requirement $G2 = 0$, the assignment $C = 0$ will be selected for the same reason. Thus, the test in this case will be relaxed to $1x0xxx$.

While the fanout-based cost functions provide better selection criteria than the regular cost functions in general, there are some cases where this is not true as illustrated by the following example.

Example 3: Consider the circuit shown in Figure 3. The detected fault under the shown test vector is $G8/1$. To justify the required value on $G8$, we could either select $G7 = 0$ or $G = 0$. Using the fanout-based cost functions, $C_0(G7) = 1$ and $C_0(G) = 1$. If $G7 = 0$ is selected, then this will result in two primary input assignments, namely $B = 0$ and $E = 0$. However, using the regular cost functions $C_0(G7) = 2$ and $C_0(G) = 1$. Thus, the assignment $G = 0$ will be selected resulting in a more relaxed vector. Thus, in this example using the regular cost functions leads to a better solution.

To take advantage of both cost functions, we propose a weighted sum cost function of the two cost functions. Let $C_{01}(l)$ $(C_{11}(l))$ denote $C_0(l)$ $(C_1(l))$ based on the regular cost functions, while $C_{02}(l)$ $(C_{12}(l))$ denote $C_0(l)$ $(C_1(l))$ based on the fanout-based cost functions. Then, the proposed cost functions are as follows:

$$C_0(l) = A \cdot C_{01}(l) + B \cdot C_{02}(l)$$

$$C_1(l) = A \cdot C_{11}(l) + B \cdot C_{12}(l)$$

As will be shown in the next section, a weight of $A = 1$ and $B = 6$ seems to be a good heuristic.

4. Experimental Results

In order to demonstrate the effectiveness of our proposed test relaxation technique, we have performed experiments on a number of the largest ISCAS85 and full-scanned versions of ISCAS89 benchmark circuits. The experiments were run on a SUN Ultra60 (UltraSparc II-450 MHZ) with a RAM of 512 MB. We have used the test sets generated by MinTest [5], which are highly compacted test sets that achieve 100% fault coverage of the detectable faults in each circuit.

In Table 1, we compare the proposed test set relaxation technique with the brute-force relaxation method. The first, second, and third columns in the table indicate the circuit name, the number of primary inputs, and the number of test vectors in each circuit, respectively. We compare the two techniques in terms of the percentage of x's extracted, and the CPU time taken for relaxation. We have used the fault simulator HOPE [4] for fault simulation purposes. It is important to point out here that the fault coverage of the relaxed test sets based on the brute-force method and the proposed technique is the same as the fault coverage of the original test set, i.e. exact test set relaxation and no drop in the fault coverage.

It is very interesting to observe that, for all the circuits, the CPU time taken by our proposed technique is less than the brute-force method by several orders of magnitude. The brute-force method requires astronomical CPU times for large circuits and hence is impractical.

The percentage of x's obtained by our technique is also close to the percentage of x's obtained by the brute-force method for most of the circuits. The difference in the percentage of x's obtained ranges between 1% and 8%. The average difference is about 3%. For nine of the ten circuits, the difference is less than 4%. It should be observed that the brute-force method implicitly chooses the output for detecting a fault that maximizes the number of x's according to the order used. However, our technique does not do any optimization in selecting the best output for detecting a fault. This can be investigated in future work.

In Table 2, we show the effect of the cost functions used in justifying required values on the percentage of x's extracted. Results are shown for several combinations of the two cost functions, namely the regular and the fanout-based cost functions, used in our work by varying the weights given to each. Note that weight A is for the regular cost function and weight B is for the fanout-based cost function. As can be seen from the table, for all the circuits the use of cost functions results in higher percentage of x's extracted than without using the cost functions. A difference of up

Table 1. Test relaxation comparison between the proposed technique and the brute-force method.

			Percentage of x's		CPU Time (seconds)	
Circuit	No. Inp.	No. Vec.	Brute-Force Technique	Proposed Technique	Brute-Force Technique	Proposed Technique
c5315	178	37	54.37	52.080	1192	1.1
c7552	207	73	55.45	52.075	6645	5.9
c2670	233	44	69.63	68.767	2757	1.0
s5378	214	97	74.14	70.753	7451	1.0
s9234.1	247	105	70.29	66.408	19837	3.0
s15850.1	611	94	80.96	78.830	87120	6.0
s13207.1	700	233	93.36	92.928	629100	5.0
s38584.1	1464	110	80.72	77.951	715000	14.0
s38417	1664	68	67.36	66.171	374000	10.1
s35932	1763	12	36.68	28.238	20358	10.0

Table 2. Cost function effect on the extracted percentage of x's.

Circuit	A=0 B=0	A=0 B=1	A=1 B=0	A=1 B=1	A=1 B=2	A=1 B=3	A=1 B=4	A=1 B=5	A=1 B=6
c5315	48.527	52.065	50.076	**52.080**	**52.080**	**52.080**	**52.080**	**52.080**	**52.080**
c7552	48.091	52.068	48.329	**52.075**	**52.075**	**52.075**	**52.075**	**52.075**	**52.075**
c2670	65.899	68.377	66.407	68.748	68.748	68.757	**68.767**	**68.767**	**68.767**
s5378	68.422	**71.057**	69.891	70.484	70.484	70.647	70.652	70.753	70.753
s9234.1	63.621	65.949	64.372	66.046	66.046	66.030	66.189	66.377	**66.408**
s15850.1	77.693	**78.971**	77.855	78.391	78.448	78.791	78.823	78.830	78.830
s13207.1	92.457	92.920	92.485	92.920	92.925	92.928	92.928	**92.929**	92.928
s38584.1	75.328	**78.072**	75.839	77.794	77.795	77.795	77.827	77.933	77.951
s38417	65.518	**66.467**	65.865	66.162	66.164	66.163	66.171	66.171	66.171
s35932	22.986	27.415	28.120	**28.238**	**28.238**	**28.238**	**28.238**	**28.238**	**28.238**
Average	62.854	65.336	63.924	65.294	65.300	65.350	65.375	65.415	**65.420**

to 5% is observed. Furthermore, it is clearly indicated from the table that the proposed fanout-based cost functions produce better results than the regular cost functions. However, a combined cost function with a weight of 1 for the regular cost function and a weight of 6 for the fanout- based cost function seems to be a good cost measure as it provides the highest percentage of extracted x's on average.

To illustrate the application of test relaxation in improving the effectiveness of test compression, we have applied the Frequency-Directed Run-Length (FDR) compression technique in [1] on the used test sets. Table 3 shows the test compression results. The first column shows the circuit name, and the last three columns show the compression ratio for the original test set without relaxation, the relaxed test set based on the brute-force method, and the relaxed test set based on the proposed technique, respectively. As is shown in the table, for all the circuits, the FDR technique achieves negative compression ratios. (i.e., compressed test set is larger than original test set). However, significant compression is achieved based on the relaxed test sets. Both the brute-force relaxation and our proposed relaxation technique achieve comparable compression ratio for most of the circuits. In general, the higher the percentage of x's extracted the higher the compression ratio. However, the location of x's and their distribution certainly have an impact on the results. From those results, it is clear that in order to have effective test compression, it is crucial to have a relaxed test set and an efficient test relaxation technique.

In order to demonstrate the impact of test relaxation on test compaction, we have used HITEC [12] to generate test sets that detect all the detectable faults on some of the benchmark circuits used. The second column in Table 4 shows the number of test vectors obtained by HITEC. Then, we compacted the test vectors based on applying the reverse-order and random order fault simulation for 20 iterations. The compacted test sets are shown in the third

Table 3. Effect of relaxation on test compression ratio.

Circuit	Compression Ratio		
	Fully Specified Tests	Relaxed Tests	
		Brute-Force	Proposed
c5315	-25.63	20.47	20.06
c7552	-19.48	37.13	34.18
c2670	-24.93	43.84	43.46
s5378	-27.86	46.85	43.67
s9234.1	-25.52	34.80	31.99
s15850.1	-23.41	56.49	56.77
s13207.1	-25.97	78.78	79.51
s38417	-29.08	37.66	35.92

Table 4. Effect of relaxation on test compaction.

Circuit	Number of Test Vectors			
	Original	ROF	Brute-Force	Proposed
c5315	193	119	100	103
c2670	154	106	98	101
s5378	359	252	136	140
s9234.1	620	376	200	214
s13207.1	633	478	252	257
s35932	80	63	37	33

column of the table. Next, we relaxed the compacted test sets based on both the brute-force test relaxation technique and our proposed technique. To achieve further compaction on the relaxed test sets, we merged compatible test vectors. Two test vectors are considered compatible if their corresponding bits are either equal or one of them is an x. The fourth and fifth columns show the number of merged test vectors based on the brute-force relaxed test set and the relaxed test set based on our technique, respectively. As can be seen from the results, over 40% test compaction is achieved for most of the circuits based on merging compatible vectors. Due to the larger percentage of x's achieved by the brute-force method, more compacted test sets are obtained except for circuit s35932. As demonstrated by the results, starting with a compacted test set, significantly higher compaction can be achieved by relaxing the test set and merging compatible vectors.

5. Conclusion

In this paper, we have presented a novel and efficient test relaxation technique for combinational and full-scan sequential circuits. While achieving slightly less test relaxation quality than brute-force test relaxation, the technique is faster by several orders of magnitude. Based on cost functions, the technique maximizes the number of x's extracted without any drop in the fault coverage. The application of the proposed test relaxation technique in achieving more effective test compaction and compression has been demonstrated.

Acknowledgment

The authors would like to thank King Fahd University of Petroleum and Minerals for support.

References

[1] A. Chandra and K. Chakrabarty. Frequency-directed run-length (FDR) codes with application to system-on-a-chip test data compression . In *19th IEEE Proceedings on. VTS*, pages 42–47, 2001.

[2] A. El-Maleh, S. Zahir, and E. Khan. A Geometric-Primitive-Based Compression Scheme for Testing Systems-on-a-Chip. In *Proc. IEEE VLSI Test Symposium*, pages 54–59, Apr. 2001.

[3] A. Jas and N. Touba. Using an Embedded Processor for Efficient Deterministic Testing of System-on-a-chip. In *International Conference on Computer Design*, 1999.

[4] H. K. Lee and D. S. Ha. HOPE: An Effecient Parallel Fault Simulator for Synchronous Sequential Circuits. *IEEE Trans. on Computer Aided Design*, 15(9):1048–1058, Sep. 1996.

[5] I. Hamzaoglu and J. Patel. Test Set Compaction Algorithms for Combinational Circuits. In *International Conference on Computer-Aided Design*, Nov. 1998.

[6] I. Pomeranz, L. Reddy, and S. Reddy. COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits. In *Proc. Intenational Test Conference*, pages 194–203, 1991.

[7] J. Chang and C. Lin. Test Set Compaction for Combinational Circuits. *IEEE Trans. on Computer Aided Design*, pages 1370–1378, Nov. 1995.

[8] M. Abramovici, M. Breuer and A. Friedman. *Digital System Testing and Testable Design*. IEEE Press, 1990.

[9] M. Schulz, E. Trischhler, and T. Sarfert. SOCRATES: A Highly Efficient Automatic Test Pattern Generation System. *IEEE Trans. on Computer-Aided Design*, pages 126–137, Jan. 1988.

[10] R. Sankaralingam, R.R. Oruganti, and N.A. Touba. Static Compaction Techniques to Control Scan Vector Power Dissipation. In *18th IEEE Proceedings on. VTS*, pages 35–40, 2000.

[11] S. Chakradhar and A. Raghunathan. Bottleneck Removal Algorithm for Dynamic Compaction in Sequencial Circuits. *IEEE Trans. on Computer-Aided Design*, 1997.

[12] Thomas M. Niermann and Janak H. Patel. HITEC: A test generation package for sequential circuits. In *Proc. of the European Conference on Design Automation (EDAC)*, pages 214–218, 1991.

[13] Y. Zorian, E. J. Marinissen and S. Dey. Testing Embedded-Core Based System Chips. In *Proc. International Test Conference*, pages 130–143.