Efficient Test Compaction for Combinational Circuits Based on Fault Detection Count-Directed Clustering

Aiman El-Maleh, Saqib Khurshid
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
aimane@ccse.kfupm.edu.sa, saqib@ccse.kfupm.edu.sa

Abstract—Test compaction is an effective technique for reducing test data volume and test application time. In this paper, we present a new static test compaction algorithm based on test vector decomposition and clustering. Test vectors are decomposed and clustered in an increasing order of faults detection count. This clustering order gives more degree of freedom and results in better compaction. Experimental results demonstrate the effectiveness of the proposed approach in achieving higher compaction in a much more efficient CPU time than previous clustering-based test compaction approaches.

I. INTRODUCTION

Recent advances in VLSI technology have enabled the fabrication of systems-on-a-chip with millions of transistors. This tremendous increase in transistor count has resulted in large increase in test data volume that often exceeds current testers’ memory capacity. In order to reduce the test data volume, two basic strategies have been investigated. The first strategy is based on reducing the required number of test vectors needed to achieve a given desired fault coverage, known as test compaction. The second strategy is based on representing the test data in a compressed form in the tester and using decompression circuitry on chip to decompress the test data before application, known as test compression. Both strategies are necessary to reduce test data volume and test application time.

Test compaction techniques can be classified as static or dynamic. In static test compaction, the number of test vectors is reduced after they are generated, whereas in dynamic test compaction, the number of test vectors is minimized during the automatic test pattern generation (ATPG) process. Static test compaction algorithms for combinational circuits can be divided into three broad categories [1]: (1) Redundant Vector Elimination, (2) Test Vector Modification, and (3) Test Vector Addition and Removal. In the first category, compaction is performed by dropping redundant test vectors. A redundant test vector is a vector whose faults are all detectable by other test vectors. Static compaction algorithms falling under this category are either based on set covering [4-6] or test vector reordering and fault simulation [7-11]. In the second category, compaction is performed by modifying test vectors. This is achieved by merging of compatible test vectors based on test relaxation or raising [12, 13-15], essential fault pruning [9, 15-17], or test vector decomposition and clustering [1-3]. Finally, the third category of static compaction algorithms consists of compaction algorithms that add new test vectors to a given test set in order to remove some of the already existing test vectors [10, 18].

Recently, two static compaction algorithms based on test vector decomposition and clustering have been proposed in [1-3]. The first technique, called Independent Fault Clustering (IFC) [1, 2], is based on clustering test vectors according to independent fault sets. The second technique, called Class-based Clustering (CBC) [1, 3], is based on classifying test vectors into classes and then eliminating vectors by moving their components to other vectors. In this work, we propose a new test compaction algorithm based on test vector decomposition and test vector clustering. Test vector clustering is performed based on fault detection count. This is in contrast to IFC which clusters test vectors based on independent fault sets. The effectiveness of the proposed approach is demonstrated by experimental results.

II. PROPOSED COMPACTION ALGORITHM

Test vector decomposition is the process of decomposing a test vector into its atomic components. An atomic component is a child test vector that is generated by relaxing its parent test vector for a single fault $f$. That is, the child test vector contains only the assignments necessary for the detection of $f$. Besides, the child test vector may detect other faults in addition to $f$.

In Independent Fault Clustering (IFC) [1, 2], Independent Fault Sets (IFSs) with respect to a given test set are first
derived. Two faults are independent if they are not detected by the same test vector. Then, test vector clustering is performed based on the derived independent fault sets. This is motivated by the fact that the size of the largest IFS gives an upper bound on the possible size of the final test set after compaction and that test vector components for faults belonging to different IFSs are potentially compatible. During test vector clustering, compatible components, corresponding to compatible faults, are mapped to the same compatibility set. Whenever a component is mapped to a compatibility set, it is merged with the partial test vector of that compatibility set. At the end of the clustering process, every compatibility set represents a single test vector.

Our proposed compaction algorithm, Fault-detection Count-based Clustering (FCC), is based on clustering test vector components based on fault-detection count. Components derived from faults with the smallest detection count are clustered first followed by faults with increasing detection count. This is motivated by the fact that faults with N detection count have N test vector components and have a higher chance of being compatible with existing clusters. If a test vector component is not compatible with all the existing clusters, other test vector components are attempted. A new cluster is created only when all the N test vector components are not compatible with all the existing clusters.

The FCC algorithm is shown in Fig. 1 and proceeds as follows. First, the given test set \( T \) is fault simulated without fault dropping. This step is performed to find the number and set of test vectors that detect every fault. Second, all the faults are sorted using their detection count in ascending order. Next, test vector components for essential faults (i.e. detection count=1) are clustered. In this step, for every essential fault \( f \) detected by \( t \), the atomic component \( cf \) corresponding to \( f \) is extracted from \( t \). Then, for every compatibility set \( CS \), if \( cf \) is compatible with the partial test vector in \( CS \), \( cf \) is mapped to \( CS \). On the other hand, if the number of compatibility sets is zero or \( cf \) is incompatible with all partial test vectors in the existing compatibility sets, a new compatibility set is created and \( cf \) is mapped to it. Next, the algorithm fault simulates the existing compatibility sets and drops all detected faults. This step saves the computation time which is otherwise spent on extracting atomic components of yet unmapped, non-essential faults and then either mapping them to existing compatibility sets or creating a new compatibility set for such faults. In addition, it could result in higher compaction.

The algorithm then focuses on remaining unmapped, non-essential faults. This step exhaustively checks every component of a non-essential fault and attempts to minimize creating a new compatibility set. For every fault, an atomic component of a fault \( f \) is extracted, if it is incompatible with all partial test vectors in the existing compatibility sets, a new component is tried. In this step, a new compatibility set is created only if the number of compatibility sets is zero, which is possible only when there are no essential faults. At this point, only those non-essential faults remain which require a new compatibility set and none of their atomic component could be mapped to any of the partially filled existing compatibility sets.

The algorithm then randomly fills the partially filled test vectors of existing compatibility sets and fault simulates all the compatibility sets. This is done to maximize the chances of detecting yet unmapped, non-essential faults and therefore save an extra compatibility set. It should be noted that random filling in step 6 does not affect compaction, since it is guaranteed that none of the remaining test vector components could map to any of the existing partially filled test vectors.

Algorithm FCC(T)
1. Fault simulate \( T \) without fault dropping.
2. Group the faults by their detection count.
3. For every essential fault \( f \) that is detected by a test vector \( t \):
   1. Extract the atomic component \( cf \) from \( t \).
   2. If the number of compatibility sets is zero, create a new compatibility set, map \( cf \) to it, and then go to Step 3.
   3. Map \( cf \) to an existing compatibility set, if possible, and then go to Step 3.
4. Fault simulate all the compatibility sets and drop all the remaining (NonEssential) faults that are detected.
5. For the remaining non-essential (un-detected) fault(s) \( f \) that is detected by a set of test vector \( T' \):
   1. For every test vector \( t' \), where \( t' \) is a member of \( T' \):
      1. Extract the atomic component \( cf \) from \( t' \).
   2. Map \( cf \) to an existing compatibility set, if possible, and then go to Step 5.
   3. Create a new compatibility set and map \( cf \) to it.
6. Random fill test vectors of all the compatibility sets.
7. Fault simulate all the compatibility sets and drop all the remaining (NonEssential) faults that are detected.
8. For the remaining non-essential (un-detected) fault(s) \( f \) that is detected by a set of test vector \( T' \):
   1. For every test vector \( t' \), where \( t' \) is a member of \( T' \):
      1. Create a new compatibility set and map \( cf \) to it.
   2. Random fill all the vectors of \( T' \).
10. Return \( T' \).

Fig. 1 Fault-detection count-based clustering (FCC).
Finally, the algorithm creates an additional compatibility set for the remaining unmapped, non-essential faults, only if all components of a fault $f$ are incompatible with all partial test vectors in the existing compatibility sets. At the end, the algorithm randomly fills the remaining partially filled test vectors and returns the compatibility sets as the compacted test set.

It is worth mentioning that for large circuits with large number of faults, fault simulation without dropping can be restricted to a $k$ number of fault detects. The value of $k$ chosen provides a tradeoff in memory and CPU time requirement and the achieved level of compaction.

We next illustrate the steps of the proposed FCC algorithm through an example. Table 1 shows a set of four test vectors along with their detected faults and the components generated for each fault. Faults $f_1$ and $f_2$ are essential faults and will be clustered first resulting in two sets as shown in Table 2. We assume in this example that fault simulating the resulting compatibility sets will not detect additional faults. Then, faults with detection count=2 will be clustered next i.e., faults $f_3, f_4, f_5, f_6$ and $f_7$. The first component of $f_3=x0x0xx11x11x0$ will be attempted for clustering and it will be found incompatible with the existing sets. The second component of $f_3=x0x0xx11x1xx1$ is then successfully clustered into the second set. The first component of $f_4=x0x0xx1xx1xxx1$ is successfully clustered into the first set. However, none of the components of the faults $f_5, f_6$ and $f_7$ can be clustered in the existing sets and hence their clustering is delayed. Next, clustering is attempted for faults of detection count=3 i.e., $f_8$. While neither the first nor the second components of $f_8$ can be clustered into the existing sets, the third component of $f_8=x0x0xx1xx1xx0xx1$ is successfully clustered into the first set. Next, the algorithm will randomly fill the merged test vectors of the compatibility sets and will fault simulate the remaining undetected faults i.e., $f_5$ and $f_7$. We assume in this example that fault $f_5$ will be detected by the randomly filled test vectors. Finally, $f_7$ and $f_8$ will be clustered next. The first component of $f_7=x0x0xx1xx1x1xx1x0$ is mapped to a new set. Then, the first component of $f_7=x0x0xx1xx1xx1x0$ is found incompatible with the third set and hence its second component is attempted. The second component of $f_7=x0x0xx1xx1xx1x0$ is then found compatible with third set and is clustered with it creating the merged test vector $1x111x0101$, which is randomly filled to create a fully specified test set. Thus, the test set is compacted into the following three test vectors: \{0001110100, 1011011001, 1110100101\}.

### III. EXPERIMENTAL RESULTS

In order to demonstrate the effectiveness of the proposed FCC test compaction algorithm, we have performed experiments on a number of the ISCAS85 and full-scanned versions of ISCAS89 benchmark circuits. The experiments were run on a Pentium Mobile, with 2.0 GHz processor and 1GB DDR2 RAM. We have used test sets generated by HITEC [19]. In addition, we have used the fault simulator HOPE [20] for fault simulation purposes and the test relaxation algorithm in [12] for test vector component generation.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Fault</th>
<th>Fault Component</th>
<th>Fault</th>
<th>Fault Component</th>
<th>Fault</th>
<th>Fault Component</th>
<th>Test Vector</th>
<th>Test Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$f_1$</td>
<td>0x0xx1xx0</td>
<td>$f_1$</td>
<td>0x0xx1xx0</td>
<td>$f_1$</td>
<td>0x0xx1xx0</td>
<td>0001110100</td>
<td>0001110100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$f_2$</td>
<td>00xx1xx1xx</td>
<td>$f_2$</td>
<td>00xx1xx1xx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$f_6$</td>
<td>00xx1xx0xx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$f_2$</td>
<td>1xx1xx10x1</td>
<td>$f_2$</td>
<td>1xx1xx10x1</td>
<td>$f_2$</td>
<td>1xx1xx10x1</td>
<td>1011011001</td>
<td>1011011001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$f_5$</td>
<td>x01xx1xx0xx</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 1 Example Test Vectors and their components.**

<table>
<thead>
<tr>
<th>Test Vector</th>
<th>Fault Detected</th>
<th>Fault Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>$f_1$</td>
<td>0x0xx1xx0</td>
</tr>
<tr>
<td></td>
<td>$f_4$</td>
<td>00xx1xx1xx</td>
</tr>
<tr>
<td>v2</td>
<td>$f_2$</td>
<td>1xx1xx10x1</td>
</tr>
<tr>
<td></td>
<td>$f_3$</td>
<td>xxxxx1xx0xx</td>
</tr>
<tr>
<td></td>
<td>$f_6$</td>
<td>1xx1xx1xx0xx</td>
</tr>
<tr>
<td></td>
<td>$f_5$</td>
<td>1xx1xx10x1</td>
</tr>
<tr>
<td>v3</td>
<td>$f_1$</td>
<td>x0xxxx11x1</td>
</tr>
<tr>
<td></td>
<td>$f_3$</td>
<td>xxxxx1xx10x1</td>
</tr>
<tr>
<td></td>
<td>$f_6$</td>
<td>1xx1xx1xx0xx</td>
</tr>
<tr>
<td></td>
<td>$f_7$</td>
<td>1xx1xx10x1</td>
</tr>
<tr>
<td>v4</td>
<td>$f_1$</td>
<td>x0xx1xx0xx</td>
</tr>
<tr>
<td></td>
<td>$f_3$</td>
<td>x0xx1xx0xx</td>
</tr>
<tr>
<td></td>
<td>$f_5$</td>
<td>x0xx1xx0xx</td>
</tr>
<tr>
<td></td>
<td>$f_7$</td>
<td>xx1xx1xx0xx</td>
</tr>
<tr>
<td></td>
<td>$f_8$</td>
<td>xxxxx1xx0xx</td>
</tr>
</tbody>
</table>

**TABLE 2 Illustration of steps of FCC on the given example.**
In Table 3, we compare the test compaction results of IFC [1, 2] and FCC algorithms when applied on the original test set. The first column gives the circuit name. The second column specifies the number of test vectors in the original test set before applying any compaction. The third and fourth columns give test set sizes after applying reverse-order fault simulation (ROF) and random merging (RM) [12], respectively. ROF is based on applying reverse-order and random order fault simulation for 20 iterations. RM is based on relaxing the test vectors generated by ROF and merging compatible vectors. Columns five and six give the results of the IFC algorithm [1, 2] while Columns seven and eight report the results of the proposed FCC algorithm. Test set sizes are given under the column headed #TV. The CPU time required by each of the algorithms is given under the column headed Time. The FCC algorithm has shown better compaction quality on 12 out of 15 circuits, while 2 circuits resulted in a draw. In terms of overall savings, FCC has saved more than 120 test vectors than IFC [1, 2] (with an average compaction improvement of 7%). For example, for the circuits c3540 and c5315, FCC achieved 24% and 25% higher compaction than IFC, respectively. It should also be noted that FCC consumes significantly lesser CPU time. It has shown 13.37 times overall improvement than IFC [1, 2]. In order to increase the level of compaction, FCC can be applied in an iterative manner until no compaction improvement is possible. We have experimented with an iterative version of FCC, called FCC6+, by applying FCC iteratively until the length of the test set cannot be reduced in the last six iterations. Unspecified bits in the test set T are assigned random values before every call to the FCC algorithm. Columns nine and ten in Table 3 report the results of an iterative version of IFC applied on the test set generated by ROF, called ROF+ITER_IFC [1, 2]. Columns eleven and twelve report the results of FCC6+. It can be seen that FCC6+ has achieved higher test compaction than ROF+ITER_IFC on 12 out of 15 circuits, while 2 resulted in a draw. For example, for the circuit c5315, FCC6+ has achieved 29% more compaction than ROF+ITER_IFC. Furthermore, it has shown higher overall savings (with an average compaction improvement of 8%) in a much more efficient CPU time (ranging from 1 to 14 times less CPU time). It should be observed that FCC6+ consumes more time on s15850 at the expense of more compaction as the algorithm continued on iterating due to more compaction improvements achieved.

IV. CONCLUSIONS

In this work, we have proposed a new test compaction technique for combinational circuits based on test vector clustering. Test vectors are decomposed and clustered in an increasing order of fault detection count. Experimental results have demonstrated the effectiveness of the proposed technique in achieving higher level of compaction in a much more efficient CPU time than previously proposed clustering-based compaction techniques. An iterative application of the proposed technique has also shown significant increase in the achieved level of test compaction.
ACKNOWLEDGMENT

This work is supported by King Fahd University of Petroleum & Minerals under project#FT-2004/07.

REFERENCES