A STATIC TEST COMPACTION TECHNIQUE FOR COMBINATIONAL CIRCUITS BASED ON INDEPENDENT FAULT CLUSTERING

Yahya E. Osais and Aiman H. El-Maleh

King Fahd University of Petroleum & Minerals, Computer Engineering Department,
Dhahran 31261, Saudi Arabia
{yosais,aimane}@ccse.kfupm.edu.sa

ABSTRACT

Testing system-on-chip involves applying huge amounts of test data, which is stored in the tester memory and then transferred to the circuit under test during test application. Therefore, practical techniques, such as test compression and compaction, are required to reduce the amount of test data in order to reduce both the total testing time and the memory requirements for the tester. In this paper, a new static compaction algorithm for combinational circuits is presented. The algorithm is referred to as *independent fault clustering*. It is based on a new concept called *test vector decomposition*. Experimental results for benchmark circuits demonstrate the effectiveness of the new static compaction algorithm.

1. INTRODUCTION

Advances in the semi-conductor process and design technology paved the way for System-on-Chip (SoC). Traditional IC design, in which every circuit is designed from scratch and reuse is limited only to standard cell libraries, is more and more replaced by the SoC design methodology. However, this new design methodology has its own challenges. A major challenge is how to reduce the increasing volume of test data. Basically, there are two approaches: *compression* and *compaction*. In the first approach, test data is kept compressed while it is stored in the tester memory and transferred to the SoC. Then, it is decompressed on the chip under test. This reduces the memory and transfer time requirements. In the second approach, however, the objective is to reduce the size of a test set while maintaining the same fault coverage.

Test compaction techniques are classified into two categories. The first category includes algorithms that can be integrated into the test generation process. Such algorithms are referred to as *dynamic* compaction algorithms, e.g. [1]. On the other hand, the second category includes algorithms that are applied after the test sets are generated. Such algorithms are referred to as *static* compaction algorithms. Static compaction has advantages over dynamic compaction. Generating smaller test sets using dynamic compaction is time consuming because many attempts to modify partially specified test vectors to detect additional faults often fail. In addition, dynamic compaction does not take advantage of random test pattern generation. Furthermore, static compaction is independent of ATPG.

Several static compaction algorithms based on different heuristics exist in the literature. One approach to static compaction is to drop from the test set redundant test vectors. A redundant test vector is a vector whose faults are all detectable by other test vectors. Redundant test vectors can be identified using set covering, e.g. [2], or test vector reordering with fault simulation, e.g. [3]. A second approach is to reduce the test set size by merging compatible test cubes. A test cube is a relaxed test vector. Test vectors can be relaxed using an ATPG or a stand-alone algorithm, such as [4]. Examples of this approach can be found in [4, 5].

Another approach to static compaction is to reduce the test set size by pruning the essential faults of some test vectors to make them redundant. A test vector becomes redundant if it detects no essential faults. A fault is essential if it is detected only by a single test vector. Static compaction algorithms based on essential fault pruning fall into two categories. In the first category, the essential faults of the test vector to be eliminated are pruned by modifying other test vectors in the test set in such away that they detect their already detected faults in addition to the pruned essential faults. Examples of such static compaction algorithms can be found in [5, 6]. On the other hand, in the second category, a set of N test vectors is replaced by a set of $M < N$ new test vectors. The basic idea is to determine the faults that are detected only by one or more test vectors among the N test vectors to be replaced and find $M < N$ test vectors that detect all these faults. Examples of such static compaction algorithms can be found in [7].

This paper is structured as follows. First, we introduce and motivate the new concept of *test vector decomposition*. Then, we describe the new static compaction algorithm based on independent fault clustering. Besides, we describe the iterative version of the algorithm. After that, we discuss the experimental results. Finally, we conclude by summarizing the results of the paper and their significance.

2. TEST VECTOR DECOMPOSITION

Test Vector Decomposition (TVD) 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 the assignments necessary for the detection of f. Besides, the child test vector may detect other faults in addition to f. For example, consider the test vector t_p = 010110 that detects the set of faults F_p = $\{f_1, f_2, f_3\}$. Using the relaxation algorithm in [4], t_p can be decomposed into three atomic components, which are (f_1,01xxxx), (f_2,0x01xx), and (f_3,x1xx10).

Static compaction based on merging is a very simple and efficient technique. However, it has the following problems. First, for a highly incompatible test set, merging achieves little reduction. Secondly, raising test vectors to make them compatible is a very costly operation [5]. Thirdly, a test vector must be processed as a whole. Therefore, we propose that a test vector be decomposed into its atomic components before it is processed. In this way, a test vector that is originally incompatible with all other test vectors in a given test set can be eliminated if its components can be merged with other test vectors.

The problem of static compaction based on TVD can be modeled as a graph coloring problem, which is NP-hard [8]. Basically, given a test set T with single stuck-at fault coverage FC_T, the set of atomic components C_T is first obtained. Then, a graph G is built. In G, every node corresponds to a component and an edge exists between two nodes if their corresponding components are incompatible. Now, our objective is to partition C_T into k subsets such that k is as small as possible and no adjacent nodes belong to the same subset. The fault coverage of the new test set T^* whose size is k should be greater than or equal to FC_T.

3. INDEPENDENT FAULT CLUSTERING

Independent faults were defined in [9]. Basically, given a combinational circuit, let T_i be the set of all possible test vectors that detect f_i and T_j be the set of all possible test vectors that detect f_j. Then, two faults f_i and f_j are independent if and only if $T_i \cap T_j = \phi$. Independence among faults can also be defined with respect to a test set T. Let T_i' be the set of test vectors in T that detect f_i and T_j' be the set of test vectors in T that detect f_j. Then, two faults f_i and f_j are independent with respect to T if and only if $T_i' \cap T_j' = \phi$. In this paper, we use the term independent faults to mean independent faults with respect to a test set.

A fault set is called an Independent Fault Set (IFS) if all the faults in this set are pairwise independent. The problem of computing a maximum size IFS is NP-hard [10]. Therefore, only *maximal* IFSs can be practically computed.

In Independent Fault Clustering (IFC), IFSs are first derived. Then, a fault matching procedure is used to find sets of compatible faults, i.e. faults that can be detected by a single test vector. In the IFS derivation phase, independent faults are identified with respect to a test set. In the fault matching phase, 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, every compatibility set represents a single test vector.

Algorithm 1 IFC(T)

```
1. Fault simulate $T$ without fault dropping.
2. For every essential fault $f$ that is detected by a test vector $t$:
    2.1. Extract the atomic component $c_f$ from $t$.
    2.2. If the number of compatibility sets is zero, create a new compatibility set, map $c_f$ to it, and then go to Step 2.
    2.3. Map $c_f$ to an existing compatibility set, if possible, and then go Step 2.
    2.4. Create a new compatibility set and map $c_f$ to it.
3. Find sets of independent faults.
4. Sort sets of independent faults in decreasing order of their sizes.
5. For every fault in an IFS, sort the test vectors that detect the fault in decreasing order of the number of faults they detect.
6. For every fault $f$, where $f$ belongs to an IFS:
    6.1. For every test vector $t$ that detects $f$:
        6.1.1. Extract the atomic component $c_f$ from $t$.
        6.1.2. If the number of compatibility sets is zero, create a new compatibility set, map $c_f$ to it, and then go to Step 6.
        6.1.3. Map $c_f$ to an existing compatibility set, if possible, and then go to Step 6.
    6.2. Create a new compatibility set and map $c_f$ to it.
7. Return $T^*$.
```

The IFC algorithm is shown as Algorithm 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. Secondly, essential faults are matched. In this step, for every essential fault f detected by t, the atomic component c_f corresponding to f is extracted from t. Then, for every compatibility set CS_i, if c_f is compatible with the partial test vector in CS_i, c_f is mapped to CS_i. On the other hand, if the number of compatibility sets is zero or c_f is incompatible with all partial test vectors in the existing compatibility sets, a new compatibility set is created and c_f is mapped to it.

It should be observed that an essential fault has a single component while non-essential faults have more than one. Therefore, if a component of a non-essential fault f is incompatible with all the partial test vectors in the existing compatibility sets, the other components of f will be tried before creating a new compatibility set. On the other hand, if the component of an essential fault is incompatible with all the partial test vectors in the existing compatibility sets, a new compatibility set must be created. Hence, essential faults should be matched first. Another advantage of first matching essential faults is that the number of faults that will be considered when deriving IFSs is reduced.

After essential faults are matched, IFSs are derived. Faults in an IFS are pairwise independent. Therefore, a fault f_i can be added to an IFS S if and only if for every fault f_j in S, the intersection of the sets of test vectors that detect f_i and f_j is empty. Next, IFSs are sorted in decreasing order of their sizes and for every fault in an IFS, the set of test vectors that detect the fault is sorted in decreasing order of the number of faults they detect. This is because a component that is extracted from a test vector that detects a large number of faults has high compatibility since it is compatible with all the components of the faults detected by that test vector.

Now, for every fault f in an IFS, its atomic component is extracted and then mapped to an appropriate compatibility set. For every component of a fault f, if it is incompatible with all partial test vectors in the existing compatibility sets, a new component will be tried. A new compatibility set is created if the number of compatibility sets is zero or all components of a fault f are incompatible with all partial test vectors in the existing compatibility sets. At the end, the algorithm returns the number of compatibility sets as the size of the new test set.

It should be pointed out that any static compaction algorithm can be used after our IFC algorithm. In fact, given a test set T, the IFC algorithm will generate a new test set T^* whose characteristics are different from the characteristics of T. Thus, a static compaction algorithm that cannot compact T may manage to compact T^*.

The procedure $IFC()$ can be called on a test set many times. Basically, the new test set generated by the procedure $IFC()$ is treated as the test set to be compacted. Therefore, IFC is carried out iteratively until the length of the test set cannot be reduced any more. This process is called $iterative$ IFC and is shown as Algorithm 2. Unspecified bits in the test set T are assigned random values before every call to the procedure $IFC()$.

Algorithm 2 Iter_IFC(T)

```
1. Randomly fill the unspecified bits in $T$.
2. $T^*$ = IFC(T)
3. If $|T^*| < |T|$, copy $T^*$ to $T$ and go to Step 1.
   Else If $|T^*| == |T|$, return $T^*$.
   Else return $T$.
```

4. EXPERIMENTAL RESULTS

In order to demonstrate the effectiveness of the IFC and Iter_IFC algorithms, we have performed experiments on a number of the 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 test sets generated by HITEC [11]. In addition, we have used the fault simulator HOPE [12] for fault simulation purposes and the test relaxation algorithm in [4] for component generation.

In Table 1, we report the results of applying the Random Merging (RM), Graph Coloring (GC), IFC, and Iter_IFC algorithms on the test sets after they are compacted using Reverse-Order Fault simulation (ROF). The GC algorithm is called the Brelaz Color-Degree algorithm and is explained in [13]. In the table, the first, second, and third columns indicate the circuit name, test set size, and fault coverage, respectively. The fourth and fifth columns give test set sizes after applying ROF and RM, respectively. Columns six through eight give the results of the GC algorithm. The number of components obtained after dropping redundant ones is given under the column headed $\#Comp$. Test set sizes are given under the column headed $\#TVs$. The total time required by the GC algorithm is given under the column headed $Time$. Columns nine and ten give the results of the IFC algorithm. Test set sizes are given under the column headed $\#TVs$ and the total time required by the IFC algorithm is given under the column headed $Time$. Finally, the last three columns give the results of the iterative IFC algorithm. Column eleven gives the test set sizes after applying IFC iteratively until no improvement is noticed. Columns twelve and thirteen give the number of iterations that were run and the time taken by the iterative IFC algorithm, respectively.

As can be seen from Table 1, for most of the circuits, the GC algorithm is able to compute test sets whose sizes are smaller than the sizes of the test sets obtained by RM. This observation reveals the potential of the TVD technique. Test sets computed by the GC algorithm are as much as 11.9% smaller than those computed by RM, e.g. 1% smaller for c2670, 9.5% smaller for s38584f, and 11.9% smaller for s4863f.

It can be seen that the results obtained by the IFC algorithm are better than those obtained by the RM and GC algorithms. The percentage improvement over the RM algorithm varies between 3.6% for s13207.1f and 37.5% for s38584f. On the other hand, the percentage improvement over the GC algorithm varies between 1.4% for s3384f and 31% for s38584f. The runtime of the IFC algorithm is better than that of the GC algorithm.

It can be seen that Iter_IFC improves over both RM and IFC. The percentage improvement over RM varies from 4% to 46.6%, e.g. 4% for s3384f, 35.8% for s38417f, and 46.6% for s38584f. On the other hand, the percentage improvement over IFC varies from 1.6% to 17.2%, e.g. 1.6% for s3271f, 14.5% for s38584f, and 17.2% for s38417f.

5. CONCLUSIONS

In this paper, we have proposed a new static compaction algorithm, referred to as Independent Fault Clustering (IFC). In IFC, independent faults are found and then compatible faults are matched together. Two independent faults can be mapped to the same compatibility set if their components are compatible. We have also considered the iterative version of IFC to further reduce the length of the com-

Cct	# TVs	FC	ROF	RM	GC			IFC		Iter_IFC		
					# Comp	# TVs	Time (sec.)	# TVs	Time (sec.)	# TVs	# Iterations	Time (sec.)
c2670	154	95.74	106	100	761	99	8.03	96	6.95	85	6	42
c5315	193	98.90	119	106	1491	117	34.95	103	31	86	4	88
s13207.1f	633	98.46	476	252	3516	248	339.93	243	169	238	2	473
s15850.1f	657	96.67	456	181	4135	169	463.95	144	249	129	1	375
s3271f	256	100	115	76	1212	69	14.97	61	7	60	2	19
s3330f	704	100	277	248	1263	233	11	208	9	196	3	30
s3384f	240	100	82	75	1048	73	15	72	7.97	72	1	7
s38417f	1472	99.44	822	187	12215	173	5327	145	2072	120	2	3775
s38584f	1174	95.85	819	232	16086	210	9250	145	2590	124	3	8217
s4863f	132	100	65	59	607	52	24	49	25.96	42	3	71
s5378f	359	99.13	252	145	1460	130	34.95	123	23	117	6	109
s6669f	138	100	52	42	1286	40	60	35	37.91	30	4	175
s9234.1f	620	93.48	375	202	2093	185	104	172	68	155	4	201

Table 1. Results by the RM, GC, IFC, and Iter_IFC algorithms.

pacted test set. Experimental results have been reported to demonstrate the effectiveness of the two algorithms. In general, the IFC algorithm has achieved an improvement of as much as 37.5% over random merging. Besides, the Iter_IFC algorithm has achieved an improvement of as much as 46.6% over random merging and 17.2% over IFC.

ACKNOWLEDGMENT

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

REFERENCES

[1] P. Goel and B. C. Rosales, "Test Generation and Dynamic Compaction of Tests," in *Dig. Papers Test Conf.*, Oct. 1979, pp. 189–192.

[2] Kwame Osei Boateng, Hideaki Konishi, and Tsuneo Nakata, "A Method of Static Compaction of Test Stimuli," in *Proc. of the Asian Test Symposium*, Nov. 2001, pp. 137–142.

[3] Irith Pomeranz and Sudhakar M. Reddy, "Forward-Looking Fault Simulation for Improved Static Compaction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 10, pp. 1262–1265, Oct. 2001.

[4] Aiman El-Maleh and Ali Al-Suwaiyan, "An Efficient Test Relaxation Technique for Combinational and Full-Scan Sequential Circuits," in *Proc. of the VLSI Test Symposium*, 2002, pp. 53–59.

[5] Jau-Shien Chang and Chen-Shang Lin, "Test Set Compaction for Combinational Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 11, pp. 1370–1378, Nov. 1995.

[6] Lakshmi N. Reddy, Irith Pomeranz, and Sudhakar M. Reddy, "ROTCO: A Reverse Order Test Compaction Technique," in *Proc. of the EURO-ASIC Conference*, June 1992, pp. 189–194.

[7] Seiji Kajihara, Irith Pomranz, Kozo Kinoshita, and Sudhakar M. Reddy, "On Compacting Test Sets by Addition and Removal of Test Vectors," in *VLSI Test Symposium*, April 1994, pp. 25–28.

[8] Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guid to the Theory of NP-Completeness*, W.H. Freedman, 1979.

[9] Sheldon B. Akers and Balakrishnan Krishnamurthy, "Test Counting: A Tool for VLSI Testing," *IEEE Design and Test of Computers*, vol. 6, no. 5, pp. 58–73, Oct. 1989.

[10] B. Krishnamurthy and S. B. Akers, "On the Complexity of Estimating the Size of a Test Set," *IEEE Transactions on Computers*, vol. C-33, no. 8, pp. 750–753, Aug. 1984.

[11] T. M. Niermann and J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," in *Proc. of the European Conference on Design Automation*, Feb. 1991, pp. 214–218.

[12] Hyung Ki Lee and Dong Sam Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1048–1058, Sept. 1996.

[13] James Mchugh, *Algorithmic Graph Theory*, Prentice Hall, 1990.