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

Aiman El-Maleh and Saqib Khursheed

Department of Computer Engineering

King Fahd University of Petroleum & Minerals

P.O. Box 1063, Dhahran 31261

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 technique based on test vector decomposition and clustering. Test vectors are decomposed and clustered for faults 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].

Static compaction techniques are preferred to dynamic compaction for several reasons. First, generating compact test sets using dynamic compaction is more time consuming as many attempts to modify partially specified test vectors to detect additional faults often fail [14]. In addition, dynamic compaction does not take advantage of random test generation which makes the ATPG process more efficient. Second, static compaction is ATPG independent allowing test sets to be generated using more efficient ATPG techniques. Finally, static compaction techniques could result in more compact test sets than dynamic compaction techniques as indicated by the results in [11, 17].

Recently, two static compaction techniques 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 test vectors by moving their components to other test vectors.

In this work, we propose a new test compaction technique based on test vector decomposition and clustering. Test vector decomposition and clustering is performed for faults based on the number of test vectors detecting each fault i.e., fault detection count. This is in contrast to IFC which clusters test vectors based on independent fault sets.

The rest of the paper is organized as follows. In Section II, the proposed test compaction technique is described. Experimental results are presented in Section III to demonstrate the effectiveness of the proposed technique. Finally, conclusions are given in Section IV.

II. PROPOSED TEST COMPACTION TECHNIQUE

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 test compaction technique, 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 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.

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.

Finally, the unmapped remaining non-essential faults are clustered. The algorithm creates an additional compatibility set for the remaining non-essential faults only if all components of a fault *f* are incompatible with all partial test vectors in the existing compatibility sets. It should be noted that at this stage compatibility is only checked with newly created clusters. 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_7 and f_8. The first component of f_3=x0xxxx11x1 will be attempted for clustering and it will be found incompatible with the existing sets. The second component of f_3=x0x1x1xxx1 is then successfully clustered into the second set. The first component of f_4=00xx1xx1xx is successfully clustered into the first set. However, none of the components of the faults f_5, f_7 and f_8 can be clustered in the existing sets and hence their clustering is delayed. Next, clustering is attempted for faults with detection count=3 i.e., f_6. While neither the first nor the second components of f_6 can be clustered into the existing sets, the third component of f_6=00xx11xx0x 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, f_7 and f_8. 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=1x1x1xx10x is mapped to a new set. Then, the first component of f_8= x1xx1xx001 is found incompatible with the third set and hence its second component is attempted. The second component of f_8=xxx11x0101 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, 1111100101}.

III. EXPERIMENTAL RESULTS

In order to demonstrate the effectiveness of the proposed FCC test compaction technique, 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], which achieve full coverage of all detectable faults in the circuits. HITEC test sets are used for comparison with the work in [1, 2, 12]. 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.

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 technique [1, 2] while columns seven and eight report the results of the proposed FCC technique. Test set sizes are given under the column headed #TV. The CPU time, in seconds, required by each of the techniques is given under the column headed Time. The FCC technique has shown better compaction quality on 12 out of 15 circuits, while 2 circuits resulted in a draw. In terms of overall savings, FFC 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. Furthermore, FFC consumes significantly lesser CPU time than IFC [1, 2]. 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 FFC, 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 FFC6+ 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.

In Table 4, a comparison is made for the largest circuits between the number of compacted test sets obtained by FCC6+ and those obtained by Mintest [17] using both dynamic and static compaction techniques. It should be observed that Mintest static compaction has reported the smallest known test sizes for several circuits. For five out of the six compared circuits, the test size of FCC6+ is smaller than Mintest dynamic test compaction. Comparison in terms of CPU time is not made as the CPU time taken by Mintest dynamic compaction is not available. However, it is known that running ATPG with dynamic test compaction is slower than regular ATPG mode. While for all the circuits, the number of test vectors obtained by Mintest static compaction is smaller, the CPU time is significantly higher than FCC6+, limiting the practicality of the technique for large industrial circuits.

It should be pointed out that any static compaction algorithm can be used after the proposed FFC algorithm. In fact, given a test set T, the FFC 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^*.

IV. CONCLUSIONS

In this work, we have proposed a new test compaction technique for combinational circuits based on test vector decomposition and clustering. Test vectors are decomposed and clustered for faults 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 test 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

[1] A. H. El-Maleh and Y. E. Osais, “Test vector decomposition based static compaction algorithms for combinational circuits,” ACM Transactions on Design Automation of Electronic Systems, vol. 8, no. 4, Oct. 2003, pp. 430–459.

[2] Y. E. Osais and A. H. El-Maleh, “A static test compaction technique for combinational circuits based on independent fault clustering,” Proc. of 10th IEEE International Conference on Electronics, Circuits and Systems, vol. 3, Dec. 2003, pp. 1316-1319.

[3]A. H. El-Maleh and Y. E. Osais, “A class-based clustering static compaction technique for combinational circuits,” Proc. of 16th International Conference on Microelectronics, Dec. 2004, pp. 522 – 525.

[4] P. F. Flores, H. C. Neto, and J. P. Marques-Silva, “On applying set covering models to test set compaction,” Proc. of Ninth Great Lakes Symposium on VLSI, Mar. 1999, pp. 8–11.

[5] K. O. Boateng, H. Konishi, and T. Nakata, “A method of static compaction of test stimuli,” Proc. of Asian Test Symposium, Nov. 2001, pp. 137–142.

[6] D. S. Hochbaum, “An optimal test compression procedure for combinational circuits,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 15, no. 10, Oct. 1996, pp. 1294–1299.

[7] M. H. Schulz, E. Trischler, and T. M. Sarfert, “SOCRATES: A highly efficient automatic test pattern generation system,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 7, no. 1, Jan. 1988, pp. 126–137.

[8] I. Pomeranz and S. 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, Oct. 2001, pp. 1262–1265.

[9] I. Hamzaoglu and J. H. Patel, “Test set compaction algorithms for combinational circuits,” In Proc. of International Conference on Computer-Aided Design, Nov. 1998, pp. 283–289.

[10] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 14, no. 12, Dec. 1995, pp. 1496–1504.

[11] X. Lin, J. Rajski, I. Pomeranz, and S. M. Reddy, “On static test compaction and test pattern ordering for scan designs,” Proc. of International Test Conference, 2001, pp. 1088–1098.

[12] A. El-Maleh and A. Al-Suwaiyan, “An efficient test relaxation technique for combinational and full-scan sequential circuits,” Proc. of VLSI Test Symposium, 2002, pp. 53–59.

[13] B. Ayari and B. Kaminska, “A new dynamic test vector compaction for automatic test pattern generation,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 13, no. 3, March 1994, pp. 353-358.

[14] K. Miyase, S. Kajihara, and S. M. Reddy, “A method of static test compaction based on don’t care identification,” In Proc. of the First IEEE Int’l Workshop on Electronic Design, Test, and Application, Jan. 2002, pp. 392–395.

[15] J.-S. Chang and C.-S. Lin, “Test set compaction for combinational circuits,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 14, no. 11, Nov. 1995, pp. 1370–1378.

[16] L. N. Reddy, I. Pomeranz, and S. M. Reddy, “ROTCO: A reverse order test compaction technique,” In Proc. of the EURO-ASIC Conference, June 1992, pp. 189–194.

[17] I. Hamzaoglu and J. H. Patel, “Test set compaction algorithms for combinational circuits,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 19, no. 8, Aug. 2000, pp. 957–963.

[18] S. Kajihara, I. Pomranz, K. Kinoshita, and S. M. Reddy, “On compacting test sets by addition and removal of test vectors,” Proc. of VLSI Test Symposium, April 1994, pp. 25–28.

[19] T. M. Niermann and J. H. Patel, “HITEC: A test generation package for sequential circuits,” Proc. of European Conference Design Automation, 1991, pp. 214–218.

[20] H. K. Lee and D. S. 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, Sep. 1996, pp. 1048–1058.

```
Algorithm FCC(T)
1. Fault simulate $T$ without fault dropping.
   1.1. Record the number of test vectors detecting each fault.
2. Group the faults by their detection count.
   2.1. Sort the faults in ascending order of their detection count.
3. For every essential fault $f$ that is detected by a test vector $t$:
   3.1. Extract the atomic component $c_f$ from $t$.
   3.2. If the number of compatibility sets is zero, create a new compatibility set, map $c_f$ to it, and then go to Step 3.
   3.3. Map $c_f$ to an existing compatibility set, if possible, and then go to Step 3.
   3.4. Create a new compatibility set and map $c_f$ to it.
4. Fault simulate all the compatibility sets and drop all the remaining (Non-Essential) faults that are detected.
5. For the remaining non-essential (un-detected) fault(s) $f$ that is detected by a set of test vector $T'$:
   5.1. For every test vector $t'$, where $t'$ is a member of $T'$:
   5.2. Extract the atomic component $c_f$ from $t'$.
   5.3. If the number of compatibility sets is zero, create a new compatibility set, map $c_f$ to it, and then go to Step 5.
   5.4. Map $c_f$ to an existing compatibility set, if possible, and then go to Step 5, otherwise go to Step 5.1.
6. Random fill test vectors of all the compatibility sets.
7. Fault simulate all the compatibility sets and drop all the remaining (Non-Essential) faults that are detected.
8. For the remaining non-essential (un-detected) fault(s) $f$ that is detected by a set of test vector $T'$:
   8.1. For every test vector $t'$, where $t'$ is a member of $T'$:
   8.2. Extract the atomic component $c_f$ from $t'$.
   8.3. Map $c_f$ to an existing compatibility set, if possible, and then go to Step 8, otherwise go to Step 8.1.
   8.4. Create a new compatibility set and map $c_f$ to it.
9. Random fill all the vectors of $T^*$.
10. Return $T^*$.
```

Fig. 1 Fault-detection count-based clustering (FCC).

TABLE 1 Example Test Vectors and their components.

Test Vector		**Fault Detected**	**Fault Component**
v_1	0000111110	f_1	0x0xx1xxx0
		f_4	00xx1xx1xx
v_2	1101101001	f_2	1xx1xx10x1
		f_5	xxxx10x0xx
		f_6	1xx1x0x0x1
		f_8	x1xx1xx001
v_3	1010111101	f_3	x0xxxx11x1
		f_5	xxxx1xx101
		f_6	1xxx11x10x
		f_7	1x1x1xx10x
v_4	0011110101	f_3	x0x1x1xxx1
		f_4	00xxx1x1x1
		f_6	00xx11xx0x
		f_7	xx1x1x0x0x
		f_8	xxx11x0101

TABLE 2 Illustration of steps of FCC algorithm on the given example.

	After mapping faults with detection count=1		**After mapping faults with detection count=2**		**After mapping faults with detection count=3**		**After Merging Components**	**After Random Filling**
Cluster	**Fault**	**Fault Component**	**Fault**	**Fault Component**	**Fault**	**Fault Component**	**Test Vector**	**Test Vector**
1	f_1	0x0xx1xxx0	f_1	0x0xx1xxx0	f_1	0x0xx1xxx0	000x11x100	0001110100
			f_4	00xx1xx1xx	f_4	00xx1xx1xx		
					f_6	00xx11xx0x		
2	f_2	1xx1xx10x1	f_2	1xx1xx10x1	f_2	1xx1xx10x1	10x1x110x1	1011011001
			f_3	x0x1x1xxx1	f_3	x0x1x1xxx1		

TABLE 3 Comparison of compaction results.

Circuit	**Orig. #TV**	**ROF**	**RM[12]**	**IFC[1, 2]**		**FCC**		**ROF+IFC-ITR[1, 2]**		**FCC6+**	
		#TV	**#TV**	**#TV**	**Time(s)**	**#TV**	**Time(s)**	**#TV**	**Time(s)**	**#TV**	**Time(s)**
c2670	154	106	100	98	0.993	98	0.04	85	42.07	82	3.93
c3540	350	83	80	99	2.01	75	1.01	75	26.95	63	5.05
c5315	193	119	106	107	3.97	80	1.96	86	88.04	61	10.94
s13207.1f	633	476	252	244	34.06	238	10.02	238	473.12	234	69.02
s15850.1f	657	456	181	142	50.97	144	15.97	129	374.95	118	1365.98
s208.1f	78	33	33	34	0.001	32	0.001	32	0.01	32	0.01
s3271f	256	115	76	60	1.95	59	1.93	60	18.98	55	3.95
s3330f	704	277	248	238	3.05	230	0.99	196	30.02	192	4.2
s3384f	240	82	75	72	1.98	72	0.96	72	7.07	72	2.98
s38417f	1472	822	187	150	838	130	225.95	120	3775.06	108	2337
s38584f	1174	819	232	148	4718	138	154.02	124	8217.08	114	1735.17
s4863f	132	65	59	50	3.02	47	3.95	42	70.88	38	6.96
s5378f	359	252	145	120	3.05	119	1	117	109	107	13.99
s6669f	138	52	42	40	7.91	36	5.02	30	175.01	28	12.02
s9234.1f	620	375	202	182	11.06	170	3.04	155	200.93	139	27.04

TABLE 4 Comparison with Mintest [17] dynamic and static compaction test sets.

Circuit	**FCC6+**		**Mintest Dynamic [17]**	**Mintest Static [17]**	
	#TV	**CPU Time**	**#TV**	**#TV**	**CPU Time**
s5378f	107	13.99	111	97	131.5
s9234.1f	139	27.04	159	105	3157.1
s13207.1f	234	69.02	236	233	1178.4
s15850.1f	118	1365.98	126	94	9252.1
s38417f	108	2337	99	68	28955.8
s38584f	114	1735.17	136	110	38538.8