Technical Note

A Method of Static Test Compaction Based on Don't Care Identification

Kohei Miyase,† Seiji Kajihara†,†† and Sudharkar M. Reddy†††

In this paper, we propose a procedure to compact a test set for a combinational circuit. Given a test set in which all input values are specified, the procedure first identifies don't care inputs of the test set, and next reassigns appropriate values to the don't cares to achieve test compaction, where an incompatibility graph is constructed and a vertex coloring problem solved. The procedure can be applied repeatedly, until further compaction cannot be derived. Experimental results show effectiveness of the proposed procedure for the ISCAS benchmark circuits.

1. Introduction

Generating compact test sets is important for reducing the cost of VLSI testing. Reserch on test compaction that can obtain a test set has been done for a long time. Test compaction methods proposed before were based on the following approaches:

(1) To generate a smaller initial test set which may contains redundant test vectors, e.g., dynamic compaction[1], maximal compaction[2], fault ordering based on independent fault sets[2),3)], rotating backtrace[2], fault selection with random pattern fault simulation[4].

(2) To remove redundant test vectors[1], e.g., static compaction, reverse order fault simulation[5], double detection[3], redundant vector elimination[6].

(3) To modefy/add some test vectors so that other test vectors become redundant, e.g., ROTCO[7], essential fault pruning[8], two_by_one[3], essential fault reduction[6].

Generating a smaller initial test set using dynamic compaction is very time consuming because many trials of test generation fail. Once all input values of test vectors were specified, it is difficult to modify the test vectors with cost effective manners. While the second approach in the above such as reverse order fault simulation can do that, the size of the obtained test sets is still large if the size of the given test set is large. The third appoach, such as the two_by_one algorithm and essential fault reduction, can bring close to minimum test sets, but take a large computation time, because the search space for modified test vectors of these methods is large.

In this paper, we present a new test compaction procedure based on identification and reassignment of don't care (X) input values of a given test set. Given an initial test set in which input values of test vectors have been specified to either 0 or 1, we identify X inputs of the test vectors as described in Ref. 9). The procedure we propose is one of the typical application in Ref. 9). Then, we reassign appropriate logic values to the X inputs by using static compaction techniques. In the static compaction part, we construct an incompatibility graph of the test vector with X values, and solve the coloring problem for the graph. A solution of the coloring problem implies how to merge the partially specified test vectors to achieve static compaction.

2. Definitions

In this work we consider a test set for single stuck-at faults of a combinational circuit or a full-scan sequential circuit. We define some terminology used in this paper below. Given a test set T, if any fault detected by test vector t_i in T is detected by at least one test vector in T-$\{t_i\}$, t_i is called a redundant test vector. When t_i is redundant, fault coverage of T-$\{t_i\}$ will be the same as that for T.

3. Overview of the Proposed Method

In **Fig. 1**, we explain the flow of the proposed test compaction procedure using the state of a test set during the compaction process. Figure 1 illustrates how a given test set is changed. Given a test set in which all input values has been specified (Fig. 1(a)), we first identify don't

† Department of Computer Sciences and Electronics, Kyushu Institute of Technology
†† Center for Microelectronics Systems, Kyushu Institute of Technology
††† Electrical and Computer Engineering Department, University of Iowa

Given tests (a)	Tests with Xs (b)	Static comp. (c)	Random fill (d)	Minimal tests (e)
t_1 11100	t_1 X11X0	t_1 X11X0	t_1 11100	t_1 11100
t_2 10111	t_2 10X1X	t_2 10110	t_2 10110	t_2 10110
t_3 10110	t_3 X0110	t_3 00X10	t_3 00110	t_3 00110
t_4 00010	t_4 00X10	t_4 1X0X1	t_4 10001	
t_5 11001	t_5 1X0X1			

Fig. 1 State of test set.

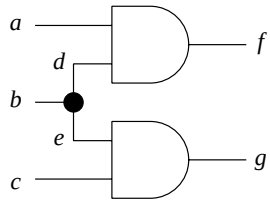

Fig. 2 Example circuit.

cares of the test set (Fig. 1(b)). Next we apply static compaction to the test set with Xs (Fig. 1(c)), where some compatible test vectors are merged into one test vector. Then we fill the remaining unspecified values randomly (Fig. 1(d)). Redundant test vectors are eliminated using double detection fault simulation[3] (Fig. 1(e)).

3.1 Don't Care Inputs of Test Vectors

Given a test set in which all inputs of test vectors were specified to either 0 or 1, some primary input values may be changed to opposite logic values without losing fault coverage. We can regard such input values as don't care (X). An example is shown using **Fig. 2**, **Table 1** and **Table 2**. For the circuit in Fig. 2, suppose that test set T in Table 1 is generated. Test set T' in Table 2 is obtained as a test set with Xs. Regardless of values of Xs in Table 2, T' keeps 100% fault coverage.

A method to identify X inputs of test vectors has been presented in 9). In the method, fault simulation and procedures similar to implication and justification of ATPG algorithms are employed with restrictions so that the obtained test set T' covers the original test set T. Static compaction is the most typical application in 9).

3.2 Incompatibility Graph

After identifying Xs of a given test set, we apply static compaction for the test set with partially specified test vectors. An example of static compaction is shown in Fig. 1(b) and Fig. 2(c). Since test vector t_2 and t_3 of Fig. 2(b) are compatible, they are merged into one test vector t_2 of Fig. 2(c). Note that two test vectors are called compatible when there exists a test vector covered by the two test vectors. In deciding how to merge, we model partially specified test vectors in an incompatibility graph and solve a coloring problem for the graph.

Table 1 Given test set T.

	a	b	c
t_1	1	1	0
t_2	1	0	1
t_3	0	1	0
t_4	0	1	1

Table 2 Obtained test set T'.

	a	b	c
t_1	1	1	X
t_2	1	0	1
t_3	0	1	0
t_4	X	1	1

t_1 X11X0
t_2 10X1X
t_3 X0110
t_4 00X10
t_5 1X0X1

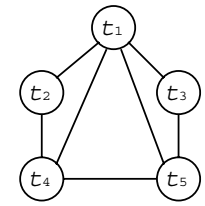

Fig. 3 Example incompatibility graph.

An incompatibility graph is a graph that represents relations of unspecified test vectors. A vertex of the graph represents a test vector. An edge between two vertices exists if and only if two test vectors corresponding to the vertices are incompatible. **Figure 3** shows some partially specified test vectors and the incompatibility graph for these test vectors. Test vectors t_1 and t_2 are incompatible because the second bits of the test vectors are different. Hence there is an edge between the vertices corresponding to t_1 and t_2. On the other hand, test vectors t_2 and t_3 are compatible, that is, it is possible to merge them into test vector 10110. Hence there is no edge between vertices corre-

Table 3 Experimental results.

circuits	#PIs	init		1st iteration		2nd iteration		final	
			#tests	#tests		#tests		#tests	#it
			DDT	COL	DDT	COL	DDT		
c432	36	54	48	46	40	40	40	40	2
c499	41	94	64	64	64	64	64	64	2
c880	60	78	54	50	44	43	41	37	5
c1355	41	129	95	95	95	95	95	95	2
c1908	33	150	124	124	123	123	123	123	2
c2670	233	142	101	101	98	97	94	94	3
c3540	50	207	153	153	147	146	142	131	8
c5315	178	186	117	98	92	88	87	84	4
c6288	32	38	29	29	28	28	28	28	2
c7552	207	290	209	184	176	173	171	161	7
s9234	247	480	377	203	200	193	189	158	7
s13207	700	586	481	283	279	273	273	271	4
s15850	611	500	435	181	180	174	173	142	11
s35932	1763	76	60	39	39	35	34	24	6
s38417	1664	1243	947	198	198	196	191	163	13
s38584	1464	854	627	239	239	227	226	193	9

Table 4 Runtime.

circuits		1st iteration			2nd iteration			final
	DDT	IDX	COL	DDT	IDX	COL	DDT	
c432	0.04	0.10	0.01	0.03	0.09	0.01	0.01	0.29
c499	0.05	0.12	0.01	0.01	0.12	0.01	0.01	0.35
c880	0.11	0.24	0.00	0.04	0.20	0.00	0.04	1.25
c1355	0.30	0.57	0.02	0.07	0.56	0.03	0.07	1.68
c1908	0.52	1.13	0.08	0.20	1.12	0.07	0.11	3.28
c2670	0.53	1.99	0.09	0.32	2.19	0.09	0.33	7.97
c3540	2.33	4.45	0.14	1.07	4.31	0.13	1.04	45.09
c5315	2.37	5.63	0.15	0.86	4.71	0.08	0.38	23.84
c6288	3.88	1.80	0.00	2.08	3.04	0.01	1.03	12.02
c7552	5.46	16.17	0.58	2.05	14.22	0.27	1.95	114.12
s9234	10.05	31.78	3.28	2.68	18.23	0.61	2.52	155.27
s13207	18.00	60.35	24.59	5.16	32.27	3.24	1.04	232.43
s15850	12.43	63.51	12.23	2.75	29.72	0.42	2.66	370.51
s35932	17.27	29.75	0.26	2.35	22.09	0.07	4.36	158.34
s38417	125.25	342.36	453.38	5.46	89.00	0.74	5.36	1965.05
s38584	85.19	240.71	102.09	5.13	103.88	1.78	9.52	1229.05

sponding to t_2 and t_3.

3.3 Vertex Coloring Problem

For the incompatibility graph obtained, we assign a color to each vertex of the graph such that any pair of adjacent vertices have different colors. From a solution of the coloring problem, we can find that test vector t_i represented by vertex v_i can be merged with other test vectors whose vertex color is the same as of v_i. For example in Fig. 3, color c_1 is assigned to t_1, color c_2 is assigned to t_2 and t_5, and color c_3 is assigned to t_3 and t_4. As a result, the given five test vectors can be reduced to three test vectors. There are some assignments of colors in general. While the vertex coloring problem is to find an assignment of the minimum number of colors, it is very time-consuming to find the minimum number of colors. In the proposed method, we assign colors to each vertex in order of the number of edges incident on it.

4. Procedures

Given a test set, the proposed test compaction procedure is as follows.

Step 1: Apply double detection fault simulation[3] to the given test set so that the test set becomes minimal.

Step 2: Identify Xs of test vectors.

Step 3: Apply static compaction, which is reduced to the coloring problem, for an incompatibility graph of the test set.

Step 4: Fill the remaining Xs with binary values randomly.

Step 5: Apply double detection fault simulation to the resulting test set so that the test set becomes minimal.

Step 6: If the test set was compacted at Step 3, then return to Step 2. Otherwise stop.

5. Experimental Results

We implemented the proposed method on SUN Ultra 5 workstation using C programming language and applied it to ISCAS'85 and full-scan version of ISCAS'89 benchmark circuits. **Table 3** and **Table 4** show the results obtained. Note that initial test sets used in this experiment are the ones generated by a SOCRATES-based test generator without any test compaction techniques.

The first three columns of Table 3 give the circuit name, the number of primary inputs and test vectors of the initial test set. The column headed "1st iteration" and "2nd iteration" show results when the proposed procedure was applied once and twice, respectively. The column headed "final" shows results by iteratively applying the procedure until no more compaction is obtained. The columns headed "DDT" and "COL" show the the number of test vectors after double detection and static compaction based on soloving the coloring problem, respectively. The column headed "it" gives the number of iterations performed. Table 4 gives CPU time of each process, where the column headed "IDX" means identification of X inputs, and the last column headed "final" shows runtime included iteration processes in Table 3.

The proposed method could reduce the number of test vectors for every circuit. Especially, large number of test vectors were removed by the proposed static compaction method from test sets for larger circuits such as s38417 and s38584. We could apply the procedure iteratively until further reduction becomes impossible. Table 4 shows many iterations of the proposed method is time-consuming. Hence we may stop iterations when we want to do.

6. Conclusions

We proposed a new method of test compaction by don't care identification and reassignment. The reassignment was done using static compaction technique that solves a coloring problem for an incompatibility graph. We iteratively applied the compaction procedure until further compaction becomes impossible.

References

1) Goel, P. and Rosales, B.C.: Test Generation and Dynamic Compaction of Tests, *Digest of Papers 1979 Test Conf.*, pp.189-192 (1979)
2) Pomeranz, I., Reddy, L. and Reddy, S.M.: COMPACTEST: A Method To Generate Compact Test Sets for Combinational Circuits, *Proc. Int. Test Conf.*, pp.194-203 (1991).
3) Kajihara, S., Pomeranz, I., Kinoshita, K. and Reddy S.M.: Cost Effective Generation of Minimal Test Sets for Stuck at Faults in Combinational Logic Circuits, *IEEE Trans. on Computer-Aided Design*, pp.1496-1504 (1995).
4) Kajihara, S. and Saluja, K.K.: On Test Pattern Compaction Using Random Pattern Fault Simulation, *11th IEEE International Conference on VLSI Design*, pp.464-469 (1998).
5) Schulz, M., Trischler, E. and Sarfert, T.: SOCRATES: A Highly Efficient Automatic Test Pattern Generation System, *IEEE Trans. on CAD.*, pp.126-137 (1988).
6) Hamzaoglu, I. and Patel, J.H.: Test Set Compaction Algorithms for Combinational Circuits, *ICCAD*, pp.283-288 (1998).
7) Reddy, L.N., Pomeranz, I. and Reddy, S.M.: ROTCO: A Reverse Order Test Compaction Technique, *1992 IEEE EURO-ASIC Conference*, pp.189-194 (1992).
8) Chang, J. -S. and Lin, C. -S.: Test Set Compaction for Combinational Circuits, *IEEE Trans. on Computer-Aided Design*, pp.1370-1378 (1995).
9) Kajihara, S. and Miyase, K.: On Identifying Don't Care Inputs of Test Patterns for Combinational Circuits, *Int'l Conf. on Computer Aided Design 2001*, pp.364-369 (2001).

(Received September 17, 2001)
(Accepted January 16, 2002)