COM_TEST : A TEST PATTERN GENERATION SYSTEM

Levent Aksoy
e-mail:levent@ehb.itu.edu.tr

Ece Olcay Güneş
e-mail:olcay@ehb.itu.edu.tr

Istanbul Technical University, Faculty of Electrical & Electronics Engineering, Department of Electronics Engineering, 34496 Maslak, Istanbul, Turkey

Key words: Test pattern generation, OR and AND/OR decision tree, fault simulation, test set compaction

ABSTRACT

In this paper, a test pattern generation system for combinational circuits including test pattern generator, fault simulator, and test set compactor is introduced. The techniques that improve test pattern generation process are used in an implemented test pattern generation system called COM_TEST. The results of COM_TEST on ten circuits are given.

I. INTRODUCTION

As the complexity of VLSI circuits are increasing, the problem of test pattern generation is becoming more important. Since the scan-based design for testability techniques are increasingly used in VLSI circuits, test pattern generation for combinational circuits is getting even more important. The test pattern generation problem can be viewed as a finite space search problem of finding appropriate logic assignments to the primary inputs such that the given fault is detected [1]. Since the size of the search space is exponential in the number of primary inputs and the test generation problem is proven to be NP-complete [2], it is very important to find efficient methods to speed up test generation process. In order to find a solution in the search space, ATPG (Automatic Test Pattern Generation) algorithms usually build a decision tree and apply a backtracking search procedure. This decision tree is usually divided up into one solution area and into several non-solution areas and the aims of ATPG algorithms are to minimize the unidentified non-solution areas and avoid all non-solution areas during the search process. These aims can be achieved by special techniques that optimize the pruning of the search space, reduce the number of backtracks, recognize conflicts as early as possible [3]. In this paper the methods that reduce the size of search space and number of backtracks in the search are introduced and an ATPG system is presented. The rest of the paper is organized as follows. Section II gives the problem definition and ATPG system is introduced in Section III. In Section IV, V, and VI, test generation method, fault simulation methods and test set compaction methods are given respectively. Experimental results are given in Section VII and finally Section VIII presents conclusions.

II. PROBLEM REPRESENTATION

The problem of deterministically generating a test pattern for a given stuck at fault (s_a_v, vε{0,1}) is to find a combination of assignments of logic values (0 or 1) to the primary inputs which simulate the given fault (line justification) and monitor the given fault at at least one of the primary outputs (fault propagation) [3]. The task of a deterministic test pattern generator is to generate a test pattern or identify the fault as untestable (redundant) for every fault in the fault list and form a test set for testable faults. Since the search in finding a test pattern consumes unacceptable amounts of computation time, an aborting criterion is used in all test generators. If the search for a given fault could not be completed, the fault is identified as aborted [4].

III. TEST PATTERN GENERATION SYSTEM

A deterministic ATPG system called COM_TEST for single stuck at faults in combinational circuits that include primitive gates, is implemented as a computer program using MATLAB. COM_TEST is composed of test pattern generator, critical path tracing based fault simulator, redundancy fault simulator, and test set compactor.

The test pattern generator is based on structural methods. It uses the single stuck at fault model and the five-valued logic alphabet containing 0 (0/0), 1 (1/1), X (don't care, unidentified), D (1/0), $\bar{D}$ (0/1) values (a/b, "a" denotes the value in unfaulty circuit and "b" denotes the value in faulty circuit). It uses the observability and controllability values given in SCOAP [5] to guide the search process. Direct implication, forward and backward implications, is used as an implication method to find uniquely determined values, and to determine inconsistency. It uses a recursive unique sensitization method based on the method proposed in FAN [6] algorithm.

As a test pattern is generated for a given fault, critical path tracing based fault simulator [7] is used to find the faults in the fault list that are detected by this test pattern. When a fault is identified as an untestable fault, redundancy fault simulator is used to find other untestable faults in the fault

list with identified untestable fault. By doing so, speeding up the test pattern generation process is aimed.

When test pattern generation is completed, test set compactor that is based on the method proposed in COM_TEST, is used to find a test set that contains a minimum number of test pattern. So, easing of testing VLSI circuits with small amount of test pattern and storage requirement is achieved.

IV. TEST PATTERN GENERATION

Test pattern generation is cone-oriented [8] where a cone contains all the paths from the fault to a primary output. Test pattern generation for a stuck at fault is composed of fault injection, fault propagation, and line justification phases. During test pattern generation, necessary and optional assignments are made and all line values must be consistent. Consistency is controlled during direct implication. In fault propagation phase, another control level is X-path check. A cone passes X-path check control, if there is at least one path starting from the faulty line to the end of the cone (a primary output) where all lines have X value. Whenever one of these control levels fail, there exists a conflict. Before starting test generation, the levels of lines in the circuit, 0 and 1 controllability values of the lines, and paths of the lines to the primary outputs (cones) with observability values are obtained.

With given f s_a_v fault, initially fault injection phase is applied. In this phase, f = $\bar{v}$ assignment is made, direct implication is applied and fault value is inserted to the circuit. In fault propagation phase, a cone is selected among the cone list according to minimum observability values. By doing so, completing this phase with a minimum number of assignment is aimed. Unique sensitization method is applied to find necessary assignments for fault propagation by finding dominators in the selected cone. Since a dominator is a bottleneck in the circuit that the fault effect must be passed through, a gate which has a dominator as an output, must be sensitized according to the fault [6]. In application of the unique sensitization method, X-path check control is made and the cone is reduced to a structure containing the paths that pass X-check control recursively. If there is a conflict, selected cone is rejected and another cone is selected if there is any. Assume a circuit has a g s_a_0 fault and fault injection phase is applied as given in Figure 1.

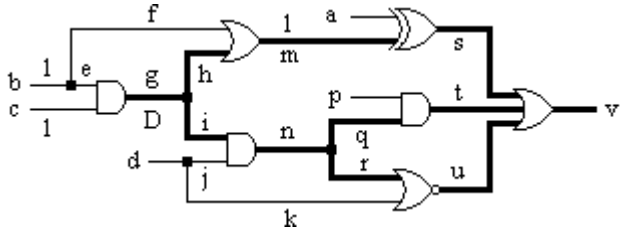

Figure 1. A combinational circuit with g s_a_0 fault

Line g has three paths, given in bold, g-h-m-s-v, g-i-n-q-t-v, g-i-n-r-u-v, and a dominator, v. In this condition, there is not any necessary assignment to be made. Since the first path of the cone fails to pass X-path check control, this path is removed from the cone and the new cone is formed with g-i-n-q-t-v, g-i-n-r-u-v as paths, and n, v as dominators. By making necessary assignments, j=1 and s=0, the second path of the cone fails to pass X-check control. This path is also removed from the cone, and the cone has g-i-n-q-t-v path and n, t, and v dominators. In this condition, p=1 assignment is made. Additionally, fault effect is reached to the primary output with consistency as shown in Figure 2. Since there is no unjustified gates, abcdp=11111 is determined as a test pattern for g s_a_0 fault.

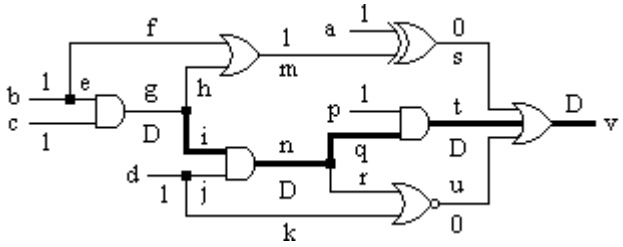

Figure 2. Application of unique sensitization method

If the fault effect is not reached to a primary output, OR decision tree [1] is constituted to complete fault propagation. OR decision tree's node is the line that is a input of a gate contains a unidentified valued output and at least one faulty valued input, and the line that is in a path which passes X-path check control. The edge of this node is a decision that is made to approach the faulty value one level higher to a primary output. In OR decision tree, at any time only one node and its decision are considered. Order in considering nodes is determined by the level of lines in the circuit and order in making decisions is determined by controllability value of the lines, if there are more than one node and decision respectively. When a value is assigned to a line, direct implication is applied to find uniquely determined values. After applying direct implication, X-path check control is made. If a conflict occurs, the decision that is made for the current node is rejected and the other decision is made if there is any. If there is not, current node is removed from the OR decision tree. This progress continues until a fault effect is reached to a primary output or decision tree is exhausted. When the decision tree is exhausted, another cone in the cone list is selected for fault propagation if there is any. When a fault effect is reached to a primary output, line justification phase is applied. Suppose a circuit has a e s_a_1 fault and fault injection and unique sensitization phases are applied as given in Figure 3.

Fault propagation using OR decision tree is based on multiple path fault propagation method. In constituting OR decision tree for e s_a_1 fault, initially i and j lines are determined as to be a node. Line j is located in OR decision tree as a root node and its decision is considered.

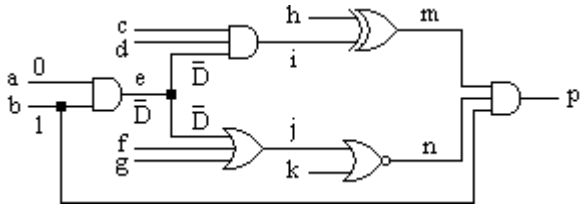

Figure 3. A combinational circuit with e s_a_1 fault

Decisions that are considered, are categorized as the ones that carry and do not carry the fault effect from input to output of the gate. The decision that carries the fault effect to output for node j is f=0 and g=0. The decisions that do not carry the fault effect to output for node j are f=1 and g=X, and f=X and g=1. So, searching in OR decision tree with smaller number of decisions is achieved. Initial decisions for a node are the ones that carry the fault effect from input to output of the gate. OR decision tree that is constituted for e s_a_1 fault is given in Figure 4 and since there exits no unjustified gate, abcdfghk=01110010 is identified as a test pattern for e s_a_1 fault.

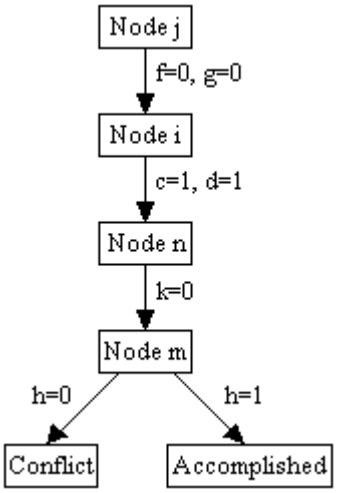

Figure 4. OR decision tree constituted for e s_a_1 fault in Figure 3

In line justification phase, if there is not any unjustified gate, current primary input values are taken as a test pattern for a given fault. An unjustified gate is a gate with a determined output value that, this value is not determined by the gate input(s)' value(s). If an unjustified gate exists, AND/OR decision tree [4] is constituted to justify the gate. Initially all faulty line values are converted to their unfaulty values according to 5-valued logic alphabet. There are two kind of levels; AND, and OR. In an AND level, AND nodes are the outputs of the unjustified gates and in an OR level, a OR node of an AND node is a combination of decisions that include gate's input(s) and value(s) that justify the gate. In AND level, order of locating AND nodes is done by maximum level of output line in the circuit. In OR level, order of locating decisions in an OR node is done by minimum controllability values of inputs of the gate. In line justification phase, consistency control is made after every assignment. If there is a conflict, current decision is rejected and another decision is made if there is any. This progress continues until there exists no unjustified gate with consistency, decision tree is exhausted, or a backtrack limit, determined as 16, is exceeded. If decision tree is exhausted, or the backtrack limit is exceeded, the last taken decision in fault propagation is rejected and another decision is made if there is any. Suppose a circuit has a t s_a_1 fault, fault propagation phase is applied and all faulty line values are converted to their unfaulty values as given in Figure 5.

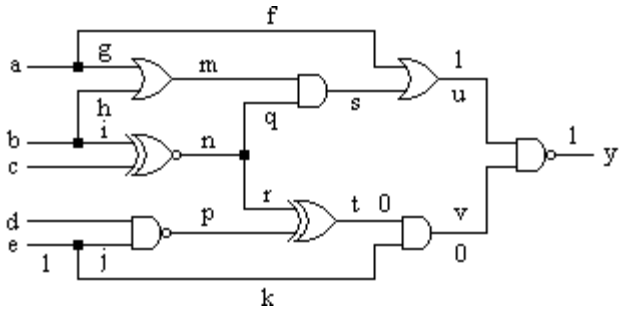

Figure 5. A combinational circuit with t s_a_1 fault

Line justification method is based on traversing along multiple paths. Initially for the first level, unjustified gates are found as the ones whose outputs are u and t lines. Then u line is located in the first AND level with its value and an OR node is formed to justify u line as decisions that can be made. There are two decisions that can be considered, f=1, s=X and f=X, s=1. The decisions are taken in order of minimum controllability values of the lines. By doing so, completing line justification phase with a minimum number of assignment is aimed. After making f=1 assignment, direct (backward) implication is applied to find uniquely determined line values. After direct implication, since t line is still unjustified, t=0 is located in the first AND level as an AND node and its OR node is constituted. With assignments made in the first OR level, a new unjustified gate is introduced and so, second level is formed in the decision tree. Constituted AND/OR decision tree for t s_a_1 fault is given in Figure 6. Additionally, abcde=10001 is found as a test pattern for t s_a_1 fault.

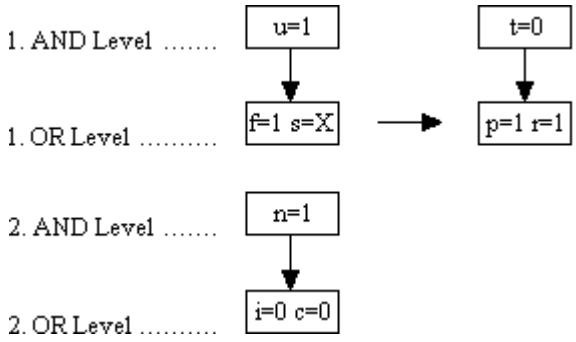

Figure 6. AND/OR decision tree constituted for t s_a_1 fault in Figure 5

V. FAULT SIMULATION

In COM_TEST, critical path tracing based fault simulator is implemented. Fault simulation is cone-oriented and starts from a primary output and traverse along the critical paths through primary inputs. Initially unfaulty circuit is obtained by applying test pattern for a given fault to primary inputs and determining primary outputs as critical. Determining a line as critical denotes that a stuck at fault containing the critical line and its complement value in the unfaulty circuit, can be tested by the applied test pattern. Critical lines are determined by finding sensitive inputs of a gate along the path. For a primary output, a critical path tracing continues until there is no critical path, or the critical path is reached to a primary input. When critical path tracing is completed for a primary output, the other primary output is considered and when encounter a line that was determined as critical in another cone, tracing along this line is halted.

A fault is identified as untestable only when the test generator fails to generate a test pattern to detect it after exhausting the search space. ATPG algorithm spends a large portion of its time dealing with untestable faults. In COM_TEST, when a fault is determined untestable, redundancy simulator is used to find other untestable faults in the fault list by this information. Identifying other faults as untestable is done by determining the assignments that are necessary for identified untestable fault, finding the faults that require these assignments to be detected.

VI. TEST SET COMPACTION

In COM_TEST, test pattern generation can be applied with fault simulators and without fault simulators. In test pattern generation with fault simulators, test patterns in the test set are identified, i.e. include no X value, and a table is constituted including test patterns and faults detected by the test patterns. Compaction of the test set is implemented by associating with the methods that find minimal sum of product form of a Boolean function with given minterms. Since only one test set that contains minimum number of test pattern is required, techniques that are used to speed up the method are used.

In test pattern generation without fault simulators, test patterns in the test set may be unidentified, i.e. include X value. Test pattern compaction definition can be made as, two test patterns can be compacted if both have the same value, or different values if one of them has a X value, for all values in the test pattern array. For example A : X01X0X and B : X0X10X test patterns can be compacted as C : X0110X. With given test set, the aim is to find a test set that contains minimum number of test pattern. This aim is realized by associating with the methods that find minimal covered compatible classes in state reduction of incompletely specified sequential circuits [9]. In this phase, this aim is realized by finding compactable pairs, forming compactable class, constituting a table of compactable class elements and compactable classes, finding minimal covered class and compacting these classes. In addition to this method, test set compaction is realized by an optimal and faster method because of computational complexity of the minimal method.

VII. EXPERIMENTAL RESULTS

COM_TEST is tested on ten circuits which are listed in Table 1 according to specific properties with a computer containing 128MB RAM and 880Mhz processor. In Table 1, i, o, c, and l denote the number of inputs, outputs, components, and lines of the test circuits respectively.

Circuit	i	o	c	l
c17	5	2	9	17
ecat	6	1	13	23
fig211	13	2	23	44
comp	6	4	35	49
karsi	8	3	40	62
74283	9	5	57	108
7485	11	3	105	147
74181	14	8	100	198
c432	36	7	249	432
c880	60	26	508	880

Table 1. Test circuits

COM_TEST is ran on two modes, with fault simulation and without fault simulation. And test generation without fault simulation is ran on two modes, minimal and optimal test set compaction. Table 2 and 3 give the results of test generation with fault simulators and without fault simulators respectively. In Table 2 and 3, tested, red, and aborted denote the number of faults identified as testable, redundant, and aborted respectively. BT_or and BT_aor stand for the number of backtracks made in OR and AND/OR tree respectively. ttpg and tpf show the test pattern generation time and elapsed time per fault in test pattern generation respectively. ntp, ttsc, and CPU indicate the number of test pattern after test set compaction, elapsed time during test set compaction and total cputime elapsed in test generation respectively. OOM denotes out of memory and TSC denotes test set compaction.

In test generation progress of total 3940 faults, there are 6 aborted faults and 102 of backtrack number (BT-aor) in c432 occur in these aborted faults. 64 redundant faults are identified with identifying 24 fault as untestable. In c880, test pattern generation with fault simulation consumes much time than without fault simulation because of selection 1 logic value instead of X value in test patterns and owning primary outputs that are present in different levels of the circuit. There are memory requirement problem in compaction of test sets whose compactable classes are too large to reduce. Therefore, optimal test set compaction method is applied.

Circuit	tested	red	aborted	BT-or	BT-aor	ttpg	tpf	ntf	ttsc	CPU(s)
c17	34	0	0	0	0	0,33	0,01	6	0	1,1
ecat	38	8	0	56	20	1,71	0,04	7	0	3,07
fig211	38	50	0	4	5	1,21	0,01	6	0	3,4
comp	118	0	0	0	0	1,49	0,01	8	0,11	4,45
karsi	124	0	0	0	0	3,02	0,02	12	0,22	6,26
74283	216	0	0	0	0	10,72	0,05	20	0,82	16,97
7485	294	0	0	0	0	11,53	0,04	16	0,6	19,88
74181	394	2	0	0	0	39,05	0,01	24	2,03	52,34
c432	854	4	6	0	127	455,94	0,53	94	17,03	501,19
c880	1760	0	0	172	0	3025,13	1,72	146	81,24	3170,47

Table 2. Test pattern generation results with fault simulation

								TSC Minimal			TSC Optimal		
Circuit	tested	red	aborted	BT-or	BT-aor	ttpg	tpf	ntp	ttsc	CPU(s)	ntp	ttsc	CPU(s)
c17	34	0	0	0	0	0,83	0,02	7	0,05	2,25	7	0	1,81
ecat	38	8	0	112	76	4,33	0,09	11	0,05	5,71	11	0	5,66
fig211	38	50	0	12	13	2,75	0,03	8	0,11	5,38	9	0,06	5
comp	118	0	0	0	0	6,15	0,05	13	0,33	9,5	13	0,05	9
karsi	124	0	0	0	0	7,75	0,06	17	0,28	11,37	17	0,11	10,93
74283	216	0	0	0	0	19,88	0,09	16	38,23	63,99	18	0,17	25,32
7485	294	0	0	0	0	36,58	0,12	26	1,32	47,12	27	0,22	44,54
74181	394	2	0	0	5	57,73	0,15	OOM			61	0,8	69,7
c432	854	4	6	0	228	408,04	0,47	OOM			82	4,83	441,65
c880	1760	0	0	404	0	708,98	0,4	OOM			114	29,82	805,04

Table 3. Test pattern generation results without fault simulation

VIII. CONCLUSION

COM_TEST is implemented as test pattern generation system for single stuck at faults in combinational circuits. It searches a solution not in the entire of the circuit instead in the cones of the circuit. It identifies as many necessary assignments in unique sensitization method. It finds as many possible uniquely determined values and so avoids from unnecessary operations and unidentified solution areas during the search. It uses testability measures to speed up the search in decision trees and traces all the necessary paths to find a solution. After a test pattern is generated for a given fault or the fault is identified as untestable, it uses fault simulators to find the faults detected by the test pattern or identify the faults as untestable with obtained untestable fault respectively. After test pattern generation, it uses test set compaction methods to find a test set containing a minimum number of test patterns in convenient size of memory.

REFERENCES

1. P. Goel, An Implicit Enumeration to Generate Tests for Combinational Logic Circuits, IEEE Transactions on Computers, Vol. 30, pp.215-222, 1981.
2. H. Fujiwara and S. Toida, The Complexity of Fault Detection Problems for Combinational Logic Circuits, IEEE Transactions on Computers, Vol. 31, pp. 555-560, 1982.
3. M. Schulz, E. Trischler, and T. Sarfert, SOCRATES: A Highly Efficient Automatic Test Pattern Generation System, IEEE Transactions on Computer-Aided Design, Vol. 7, pp. 126-136, 1988.
4. W. Kunz, and D. Stoffel, Reasoning in Boolean Networks: Logic Synthesis and Verification Using Testing Techniques, Kluwer Academic Publishers, London, 1997.
5. L. Goldstein, Controllability and Observability Analysis of Digital Circuits, IEEE Transactions on Computers, Vol. 30, pp. 215-222, 1979.
6. H. Fujiwara, and T. Shimono, On the Acceleration of Test Generation Algorithms, IEEE Transactions on Computers, Vol. 32, pp. 1137-1144, 1983.
7. M. Abramovici, J. Kulikowski, P. Menon, and D. Miller, Critical Path Tracing: An Alternative to Fault Simulation, IEEE Design & Test of Computers, Vol.1, pp. 83-93, 1984.
8. U.Mahlstedt, T. Gruning, T. Ozcan, and W. Daehn, CONTEST: A Fast ATPG Tool for Very Large Combinational Circuits, Proceedings of International Conference on Computer-Aided Design, pp. 222-225, 1990.
9. O. Ucar, and A. Dervisoglu, State Reduction of Incompletely Specified Finite Sequential Machines by the Use of Closed Compatible Pairs, Proceedings of European Conference on Circuit Theory and Design, 1999.