PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 37, NOS. 3-4, PP. 309-318 (1993)

EFFICIENCY TEST OF AUTOMATIC TEST PATTERN GENERATION METHODS

Zoltán HEGEDŰS

Department of Measurement and Instrument Engineering
Technical University of Budapest
H-1521 Hungary, Budapest

Received: December 5, 1992

Abstract

Automatic Test Pattern Generation (ATPG) is unavoidable for large combinational circuits. However, since ATPG is a known NP-complet problem, this is a very CPU-time consuming process. Therefore choosing the optimal ATPG algorithm for an industrial test generation system can be an important question. However, this question cannot be easily answered because of the implementational and evaluation differences of the published algorithms. This paper presents a software frame, where any ATPG method and their heuristic can be easily implemented allowing a correct comparison between different methods.

On the other hand the known ATPG methods cannot be ordered by quality, because their efficiency depends on the properties of the examined circuit. Therefore it seems to be reasonable to develop a hibrid strategy whose effectivity is independent of the circuit properties and near to the known strategies. The presented frame is an ideal environment for developing such a new method.

Experimental results are also presented on some implemented algorithms and heuristics using a variety of MSI components and ISCAS'85 benchmark circuits.

Keywords: automatic test pattern generation, development tool, heuristics.

Introduction

Generally deterministic test pattern generation for large combinational circuits is unavoidable for a certain percentage of the circuit faults. However, Automatic Test Pattern Generation (ATPG) techniques being NP-complet problems are [2] very CPU time consuming processes for larger circuits. Therefore the optimal choice of an ATPG method for a certain application can be a significant question. However the known ATPG algorithms like D-algorithm [3], PODEM [4], Composite Justification [5] have different approaches in finding an appropriate test pattern for a given fault. This fact implies that regarding their effectivity an absolute quality order cannot be established. Their effectivity depends on the properties of the examined circuit. Furthermore the implementation and evaluation differences of the published algorithms makes also difficult the right choice.

This paper describes a software frame (TESTFRAME) which allows the implementation of any kind of ATPG algorithms providing a possibility for a correct comparison. Furthermore the TESTFRAME provides an ideal environment for developing new ATPG methods. In Section 1 a simple formalism is introduced which is suitable to describe a variety of ATPG algorithms. Section 2 contains the description of TESTFRAME tool. In Section 3 some example can be found how to implement ATPG methods in TESTFRAME. Section 4 contains some examples of heuristics implemented in TESTFRAME. Section 5 summarizes some experimental results achieved on MSI elements and ISCAS'85 benchmark circuits [6].

1. General Description Formalism for ATPG Methods

To implement different ATPG methods in the same environment we need a formalism, that is appropriate to describe all of them. One such an approach can be if the ATPG algorithms are regarded as systematic searching in a weighted decision tree. In this case an ATPG algorithm can be given on the next way.

Let the examined circuit be given by a graph (*Fig. 1.*). In this graph let each gate output and primary input of the circuit be represented by a node in the graph. The nodes are connected to each other upon the circuit topology by directed edges. The direction of an edge corresponds to the direction of the signal flow in the circuit.

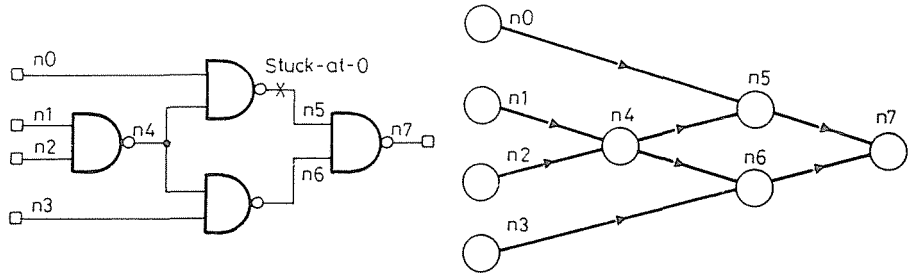

Fig. 1. Circuit diagram and the corresponding circuit graph

Now one vertex in the decision tree corresponds to a logic value combination of the nodes in the circuit graph (*Fig. 2*). The vertices of the decision tree are also connected to each other by directioned edges. Here

one edge corresponds to an assignment of a logic value to a certain node of the circuit graph. The root vertex of the decision tree represents the circuit state when every node has no definite value. The leafs of the decision tree will correspond to the valid test patterns and the inconsistent logic value combinations. One vertex can be reached from different states, since a combinational network is unsensible to the order of the logic value changes at its nodes. The searching in the decision tree should be systematic, since otherwise it cannot be ensured that if a test pattern exists it will be found. In the third section examples will be given, how an ATPG method can be corresponded to a certain weighting of the tree.

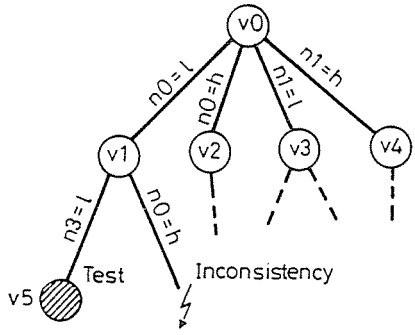

Fig. 2. Decision tree corresponding to ATPG

2. TESTFRAME Development Tool for ATPG Algorithms

Since TESTFRAME aims at comparing and developing ATPG methods it can rather be regarded as an experimental tool than an industrial software. Therefore some requirements have lower significance comparing to an industrial test system, however some properties get higher importance. For example we do not need fault simulation since we definitely want to examine the behaviour of an ATPG method for each assumed fault. Of similar reasons we do not need random or adaptive random test generation phase. However we do need precise evaluation, the possibility to easy install new ATPG strategies and the possible most general properties to support the realisation of every algorithm. For example the different methods use various logic value set. In [3] a 5-valued logic is described, [7] and [8] have 9- and 11-valued alphabet respectively. To support all of these alphabets in

TESTFRAME it was necessary to choose the possible widest logic value set which yields as the power set of $\{0/0, 1/1, 1/0, 0/1\}$[1] alphabet.

Using the $\{1, h, d, n\}$ notation for this basic values respectively, the 16-valued power set is:

$$\{-, l, n, ln, d, ld, dn, ldn, h, lh, hn, lhn, hd, lhd, hdn, lhdn\}^2$$

This choice[2] can make the computation slower but includes all possible logic sets and can ensure the least sever assignment to a node.

Some further properties of the software: Its fault model consists of the stuck-at-1 and stuck-at-0 faults. It is implemented in C language. Its size is only about 2000 source line. Only the C standard libraries were used to maintain the portability of the software. It has implemented versions under UNIX and DOS operating systems. The inputs of the software are the gate level description of the examined circuit and some parameters. The parameters can be given in a command file as well, since for larger networks the running time is expected to be rather long. By these parameters the computation time and the number of backtracks taken can be limited, furthermore the used ATPG methods and the required output files can be determined.

In the current state of the tool three ATPG algorithms can be chosen. These methods are: D-algorithm, Composite Justification, PODEM. As an output file enhanced circuit graph description, test pattern, statistics and trace information on the test computing process can be chosen. The philosophy of constructing the frame was to include every function in it which is needed by most of ATPG methods. Therefore in TESTFRAME the following functions are built-in:

- User interface/Command file treatment
- Circuit graph extractor and processor
- Systematic search method with decision maker and backtrack mechanism
- Forward and backward implication
- A checker function to detect if the current state of the circuit is a test for the given fault
- A procedure that is systematically searching for an appropriate input combination to a gate output value. This is used during back tracking.
- Fault effect propagation function

TESTFRAME uses two dinamic lists. The decision list contains the decision which can be made. The order of the elements is determined by their

[1] Here the two parts of one logic value separated by slash refer to the logic value of the node in the good and the faculty circuit, respectively

[2] Here '–' stands for the inconsistent state when no logic value can be assigned a node

weight.[3] The first element has the highest weight which represents the most fascinating decision. The allocation list contains the made decisions and implications in the order of insertion time. The first element represents the fault site. The elements of the two lists have the same structure. The most important slots of an element are: node index, logic value, weight.[4]

It has to be noticed, since TESTFRAME aims to be a generally usable tool, the fastest solution had to sacrificed in order to keep the generality. The recursive test computing process of the next main steps (*Fig. 3.*):

1. Getting the parameters from command file or console.
2. Circuit graph extracting. Constructing data structure upon the circuit topology and computing further information for later use of heuristics.[5]If required saving graph information file.
3. If there is no untested fault, making general statistics if required and exiting from the program.
4. Choosing the next fault and placing it as the only element of the decision list on the first recursion level.
5. If no more decision can be made (decision list is empty) backtracking. At this step level of recursion is decremented by one.
6. If the level of the recursion is 0 the test computing failed for the given fault. If required making test administration and local statistics.
7. If trace information required, making administration on the current state of the decision list. Taking the decision which has the highest weight from decision list, placing the decision to the allocation list. At this step the recursion gets one level deeper.
8. Forward and backward implication.
9. If the implication finds the current state of circuit inconsistent, redrawing last decision, jumping to 5.
10. Placing the implications to the allocation list. If trace information is required making administration on allocation list.
11. Input combination matching for the gate output which corresponds to the last decision made. This step is skipped if a primary input corresponds to the last decision. The matched inputs have weight 0 on the decision list.
12. If the current state of the network tests the assumed fault, making test administration, local statistics if required and jumping to 3.

---

[3]One new decision can be inserted to beginning or to the end of the group of those decisions which have the same weight. This makes possible easy realising of first depth and breadth searchings.

[4]The function of some more slot of the structure are not detailed here and in the description of the program.

[5]Detailed in Section 4.

13. Calling the weighting function that realises the chosen ATPG method. Occasionly ordering or placing new element to decision list.
14. Jump to 5.

3. Implementation of ATPG Methods in TESTFRAME

In TESTFRAME several ATPG methods and their different versions were implemented. For example here the weighting function for the D-algorithm and the Composite Justification are given. The D-algorithm's weighting process has two steps.

1. Making fault propagation and placing the propagated nodes to the decision list with weight 2.
2. If there are any element on the decision list with weight 0, changing their weight to 2 and reordering the list. Decisions can be generated weight 0 during gate input matching.

The weighting function of Composite Justification consists of two steps too. But the first is taken only once, at the beginning of the test computation for a fault.

1. If the last allocated node is the fault site placing all primary outputs to the decision tree with value 'dn' and weight 1.
2. If there are any element on the decision list with weight 0, changing their weight to 2 and reordering the list.

4. Implementation of Heuristics in TESTFRAME

Heuristics can be established in TESTFRAME using mainly the information gained during graph extracting.[6] Graph extracting generates information for one gate on; gate type, input number, fanout number, backward and foreward connected nodes, reachable primary inputs and outputs, node complexity, primary output complexity. The complexity of a node is computed similarly as in [9]. The primary output complexity at a node, shows that how many paths lead from the node to the given primary output. Since the primary output list of a node is ordered upon this complexity measure the Composite Justification will choose automatically for the first trial primary output with the lowest complexity.

In the current state of the frame two further heuristics are implemented. One is the intelligent backtrace algorithm published by Goel in

[6] Obviously the required information can be computed right when it is needed, but if there is an suitable, it is much less expensive to use which was gained during preprocessing since it was computed once for the whole test generation process.

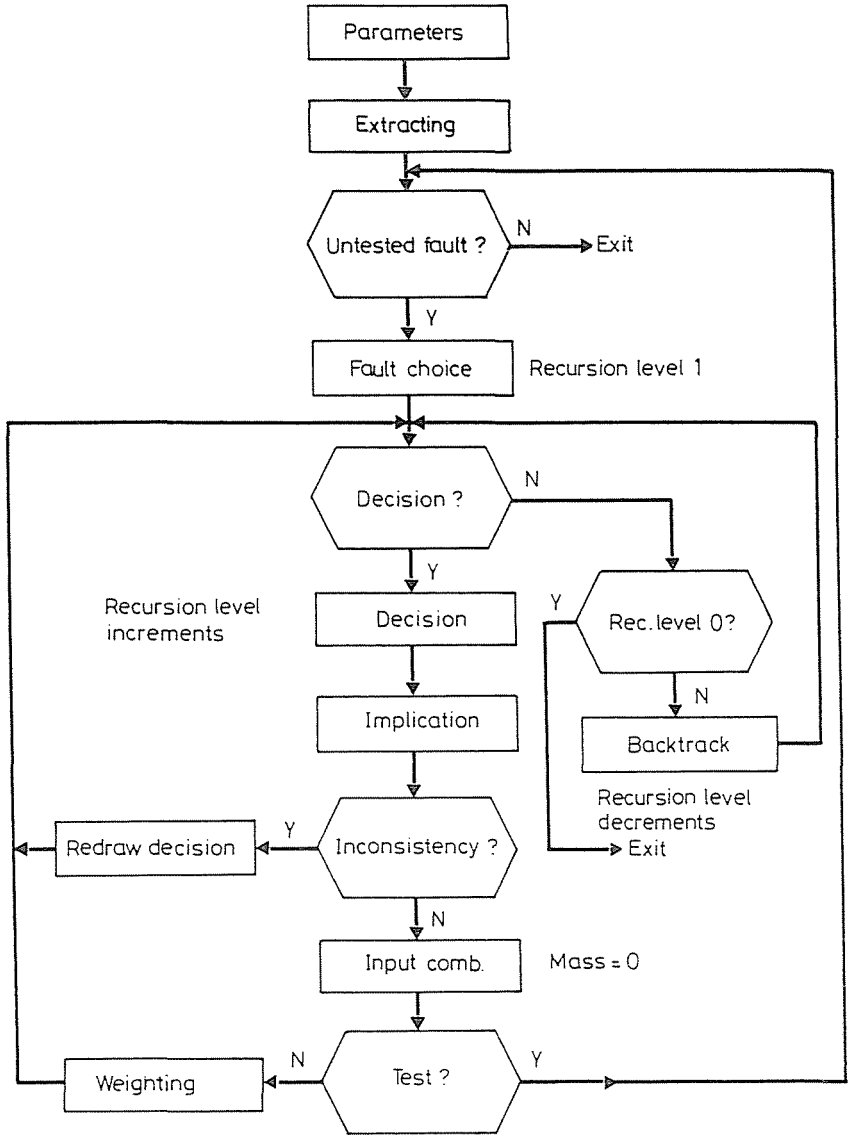

Fig. 3. Flow chart for TESTFRAME

[4]. This method chooses the easiest gate input if only one input has to be justified and the hardest if all has to be justified during backtracing. In TESTFRAME the inputs of each gate are ordered by their complexity index during preprocessing. Therefore during test computing, we only have to choose the right end of the current gate input list. The other heuristic helps to find the optimal path from the fault site to a primary output in D-algorithm. The list of foreward connected gates are also ordered during preprocessing upon the number of their inputs. Therefore the D-propagation will try first the gate which has the lowest number of inputs.

5. Experimental Results

Table I. shows the experimental results on the implemented ATPG methods and heuristics using TESTFRAME. Here the backtrack limit for the MSI elements was 1000 and for the benchmarks 20.

In some cases the circuits contain untestable faults[7] [8]. These faults are also taken into account in the summary, therefore the real test coverage is higher in some cases than it is marked.

In the case of c499 and c1355[8] the application of intelligent backtrace seems to reduce the fault coverage. However the D-algorithm for example is known to be poor at highly reconvergent exor networks [4]. Therefore the high fault coverage without intelligent backtrace now can rather be viewed as the result of a lucky description of the network.[9] To proof this several experiments were made with different descriptions of the same circuit and in each case the fault coverage fell below 70%. On the other hand replacing the applied node complexity measure by a static testability analysis which distinguish the controllability when logic 0 and 1 is required to a node may increase the performance of the implemented ATPG methods.

The applied intelligent D-propagation and intelligent primary output choice heuristic never reduced the performance in the examined cases. However since these are also statistic techniques certainly exists some description of the examined circuit when they do reduce the fault coverage.

It can be seen that the quality of the ATPG methods depends on the examined circuit, so it would be useful to develop a hibrid method combining the D-algorithm and the Composite Justification to reduce this dependence.

---

[7] For example c432 contains one according to [8], but unfortunately is some other publications different values can be found.

[8] The c499 and c1355 circuits realise the same function but the numerous exor gates in c499 are replaced by nand gates in c1355

[9] The description of a circuit is insensible to the order of inputs belonging to a gate.

Table 1
Experimental results using TESTFRAME. D-alg. stands for D-algorithm, Comp. for Composite Justification, iD-prop. for intelligent D-propagation, iBckTr. for intelligent backtrace, iPO-cho. for intelligent Primary Output choice, f.cov. for fault coverage, av.bck. for avarage number of backtracks and av.brc. for the average number of branches taken during test computing.

		7485	74181	c432	c499	c880	c1355
D-alg	f. cov. [%]	100.0	100.0	90.31	100.0	100.0	100.0
	av. bck.	7.440	1.052	2.105	0.000	0.002	0.131
	av. brc	37.49	20.55	91.08	119.4	74.94	175.3
D-alg. + iD-prop.	f. cov. [%]	100.0	100.0	92.60	100.0	100.0	100.0
	av. bck.	3.881	1.052	1.584	0.000	0.002	0.131
	av. brc.	31.64	20.52	84.74	119.4	74.85	174.3
D-alg. + iBckTr.	f. cov. [%]	100.0	100.0	95.92	89.71	100.0	98.47
	av. bck.	7.179	0.922	0.911	2.370	0.028	2.050
	av. brc.	38.66	19.85	77.59	123.5	75.75	177.2
D-alg. + iD-prop. + iBckTr.	f. cov. [%]	100.0	100.0	98.21	89.71	100.0	98.47
	av. bck.	3.738	0.922	0.452	2.370	0.012	2.050
	av. brc.	33.06	19.83	68.42	123.5	75.64	177.2
Comp. J.	f. cov. [%]	95.24	100.0	74.23	100.0	99.89	98.64
	av. bck.	51.55	2.188	5.681	0.335	0.035	0.982
	av. brc.	75.11	25.18	89.20	123.7	79.81	189.2
Comp. J. + iPO-cho	f. cov. [%]	100.0	100.0	91.84	100.0	100.0	98.64
	av. bck.	0.250	2.597	2.168	0.335	0.002	0.983
	av. brc.	24.45	24.81	82.29	123.7	77.80	189.0
Comp. J. + iBckTr.	f. cov. [%]	90.48	97.40	68.37	97.94	99.77	93.27
	av. bck.	95.58	27.75	6.449	0.593	0.105	2.855
	av. brc.	120.5	49.82	72.88	130.3	80.40	181.1
Comp. J. + iPO-cho. + iBckTr.	f. cov. [%]	100.0	98.70	96.68	97.94	99.77	93.27
	av. bck.	0.000	15.22	0.737	0.593	0.087	2.849
	av. brc.	25.36	37.23	65.88	130.3	78.05	181.1

Conclusions

In this paper a software frame TESTFRAME and a possible general description formalism for ATPG methods were presented. ATPG algorithms implemented in TESTFRAME were also presented. Upon experimental results the implemented methods working under TESTFRAME have acceptable performance on fault coverage, number of backtracks and number of taken branches. The frame was proved to be an efficient tool for a very simple implementation of different ATPG algorithms. Some possibilities of

implementing heuristics in TESTFRAME were also given. Since the performance of the implemented ATPG algorithms depends on the properties of the examines network it seems to be reasonable to develop a hibrid, circuit independent method using the advantages of TESTFRAME.

References

1. FUJIWARA, H.: Logic testing and design for testability. MIT PRESS 1985.
2. FUJIWARA, H., TOIDA, S.: The complexity of fault detection: An approach to design for testability. *Proc. FTCS-12,* 1982. pp. 101–108.
3. ROTH, J. P.: Diagnosis of Automata Failures: A Calculus and a Method. *IBM Journal of Research and Development,* Vol. 10, pp. 278–291. July. 1966.
4. GOEL, P.: PODEM-X: An automatic test generation system for VLSI logic structures. *Proc. 18th Design Automation Conf.,* 1981, pp. 160–268.
5. SZIRAY, J.: Test calculation for logic networks by composit justification. *Digital Process,* Vol. 2, 1979, pp. 3–15.
6. BRGLEZ, F., POWNALL, P., HUM, R.: Accelerated ATPG and fault grading via testability analysis. *Proc. ISCAS'85,* June 1985, pp. 695–698.
7. MUTH, P.: A Nine-Valued Circuit Model for Test Generation. *IEEE Trans. on Computers,* Vol. C–25, no. 6, June 1976, pp. 630–636.
8. CHENG, W. T.: Split Circuit Model for Test Generation. *Proc. 25th Design Automation Conf.,* Anaheim, CA, June 1988, pp. 96–101.
9. SIPOS, T., PATARICZA, A., SZIRAY, J.: An artificial intelligence application in fault-oriented algorithms. *Proc. UP'89, 6th Symposium on Microcomputer and Microprocessor Application,* Budapest, Oct. 1989.