

ADVANCED ATPG BASED ON FAN, TESTABILITY MEASURES AND FAULT REDUCTION


International journal of VLSI design & Communication Systems (VLSICS), Vaishali Dhare

Cite this paper

Downloaded from [Academia.edu](#) 

[Get the citation in MLA, APA, or Chicago styles](#)

Related papers

[Download a PDF Pack](#) of the best related papers 



[New Techniques for Deterministic Test Pattern Generation](#)

Janak Patel

[ATPG for Heat Dissipation Minimization During Test Application](#)

Sandeep Gupta

[COMPACTEST: a method to generate compact test sets for combinational circuits](#)

Sudhakar Reddy

ADVANCED ATPG BASED ON FAN, TESTABILITY MEASURES AND FAULT REDUCTION

Vaishali Dhare and Dr. Usha Mehta

Institute of Technology, Nirma University, Ahmedabad, Gujarat, India

ABSTRACT

To generate test patterns for the complex VLSI designs, an efficient and simple technique is required. This paper presents the development of combinational ATPG based on FAN algorithm, testability measures and fault equivalence. The prime aspect of this work is to develop the ATPG algorithm with less number of faults and testability measures. The proposed ATPG algorithm is simple and can be used as an open source for academicians. Analysis on ISCAS 85 circuit along with some basic combinational circuits are presented for stuck-at faults.

KEYWORDS

ATPG, fault equivalence, testability measures, ISCAS, FAN.

1. INTRODUCTION

Automatic Test Pattern Generation (ATPG) is the process of generating patterns to test a circuit. A significant amount of research has been done in the area of gate-level combinational ATPG using the stuck-at-fault model. As the numbers of transistors are increasing on chip, it is becoming difficult to generate the test vectors to test all faults. The objective of testing is filtering defective chips from manufactured ones to reduce the fraction of defective parts those are erroneously sold to customers as defect-free parts [1]. The commercial ATPG tools are available in the market like FastScan by Mentor Graphics, TetraMAX by Synopsys. The cost of these ATPGs is high also it is not possible to analyze the algorithm of it. Our aim was to develop an open source advanced ATPG which would be freely available to generate the test patterns and to analyze the concept of controllability, observability and fault equivalence. The single stuck-at-fault model has been widely accepted as a standard target model to generate a set of test patterns to detect all the stuck faults in the circuit [2]. In this paper single stuck at fault model is considered. If we consider single stuck at fault (Stuck at 0 and Stuck at 1) then the number of faults and test vectors are $2n$, where n is number of nets. Size of test vector becomes large for large combinational circuits. The bandwidth requirement of ATE and test time may be more in this case [3]. No doubt that test vector compression methods are available but prior to it fault reduction is possible using fault equivalence method. The reduced fault set concept using fault equivalence is described and developed in [4]. ATPG (Automatic Test Pattern Generation) should be guided by the testability measures to choose a decision during justification and propagation. These measures serve as heuristics and represent the relative difficulty of justifying a gate value to a control input or propagating a fault effect to an observe point. The testability measures, controllability and observability is described and developed in [5].

The proposed ATPG algorithm is based on the concept of fault equivalence to reduce the number of faults and testability measure. The algorithm used for the development of this kind of ATPG is based on FAN: fanout oriented test generation algorithm. The proposed algorithm is developed using C++ language. The results for ISCAS (International Symposium on Circuits and Systems) 85 C17 benchmark circuit are given in result section.

2. FAN: FANOUT ORIENTED TEST GENERATION ALGORITHM

FAN algorithm is developed by H. Fujiwara and Shimono [6]. Terminology: When a line L is reachable from a fanout point, L is said to be bound. A signal line that is not bound is said to be free. When a line is adjacent to some bound line, it is said to be head line. D-frontier consists gates whose output values are X, but have D (or D') on their inputs.

Fan algorithm's strategies are discussed in brief. The proposed ATPG algorithm uses some of these concepts to make it less complex.

- In each step of the algorithm, determine as many signal values as possible which can be uniquely implied
- Assign a faulty signal value D or D' which is uniquely determined or implied by the fault under consideration
- When the D-frontier consists of a single gate, apply a unique sensitization
- Stop the backtrace at a head line, and postpone the line justification for the head line to the last
- Do multiple backtracing, that is, concurrent tracing along more than one path. This is more efficient than the backtrace along a single path.

3. IMPLEMENTATION OF PROPOSED ADVANCED ATPG

The first task is to develop program to find the number of headlines, bound lines and free lines in the combinational circuit. Head line, bound lines and free lines are static i.e. they are fixed according to circuit topology. The status of all these line is stored in text file which will be act as an input to the further programs like multiple backtracing, forward implication, backward implication. The flow chart to find head line, bound line and free line is given in figure 1.

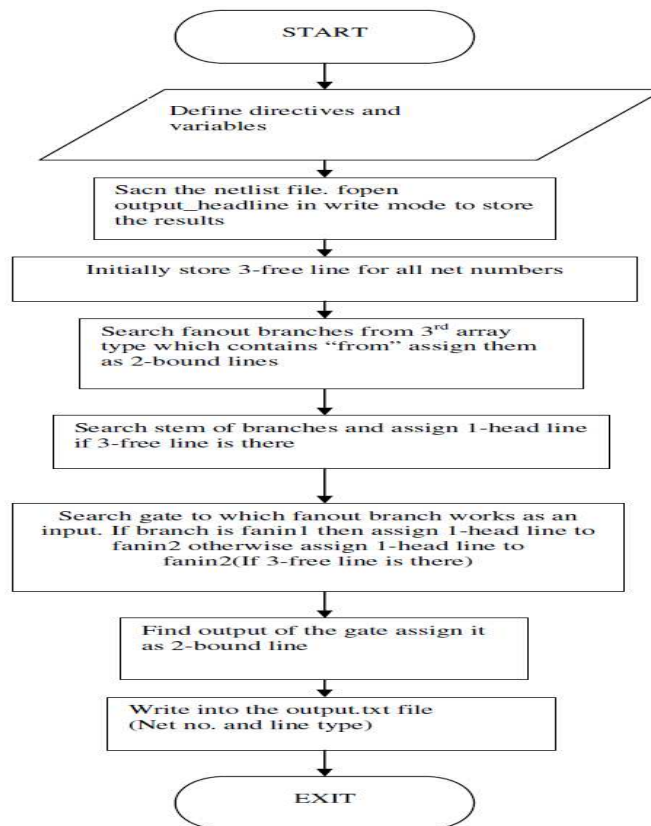


Figure 1. Flow chart to find head, bound and free lines

Logic to find headlines, bound lines and free lines

1. Define all the directives, data and variable type.
2. Read the netlist file using '*fopen*' in read mode for which head line, bound lines and free lines are to be decided. Netlist is in ISCAS format.
3. Open file in write mode "output headlines.txt" in which output is be stored.
4. Here 1 is used head line, 2 for bound line, 3 for free line. Integers are taken instead of character sothat variable type will not differ.
5. Initially assign number 3 which is considered as free line to all the nets of the circuit.
6. Search fanout branches net numbers (netno) from the netlist which is written in ISCAS format. Assign fanout branch as 2-bound line.
7. Search the 'stem' of fanout branches from netlist. Assign the stem's netno 1-head line.
8. assign that net as 1-head line.
9. Assign that net as 1-head line if 3-free line is present and no 2-bound line is assigned earlier to that net.
10. Search the gate to which the fanout branch works as an input, find second input to that gate.
11. Fanout branch may be first input (fanin1) or second input (fanin2), if it is fanin1 then assign 1-head line to fanin2 and if it is fanin2 assign 1-head line to fanin1, provided that there is 3-free line available and no 2-bound line is assigned.

The flow chart of proposed ATPG algorithm is given in figure 2. It is used for combinational circuit with 2-fanin-fanout gate like and, nand, or, nor, not and exor. Here single stuck at fault is

considered. The reduced fault list generated by fault equivalence method [4] is used as fault list for this algorithm. Now the number of faults are less. The different functions like backward implication, multiple backtrace, forward implication are developed separately (A, B,C respectively shown in figure 2).

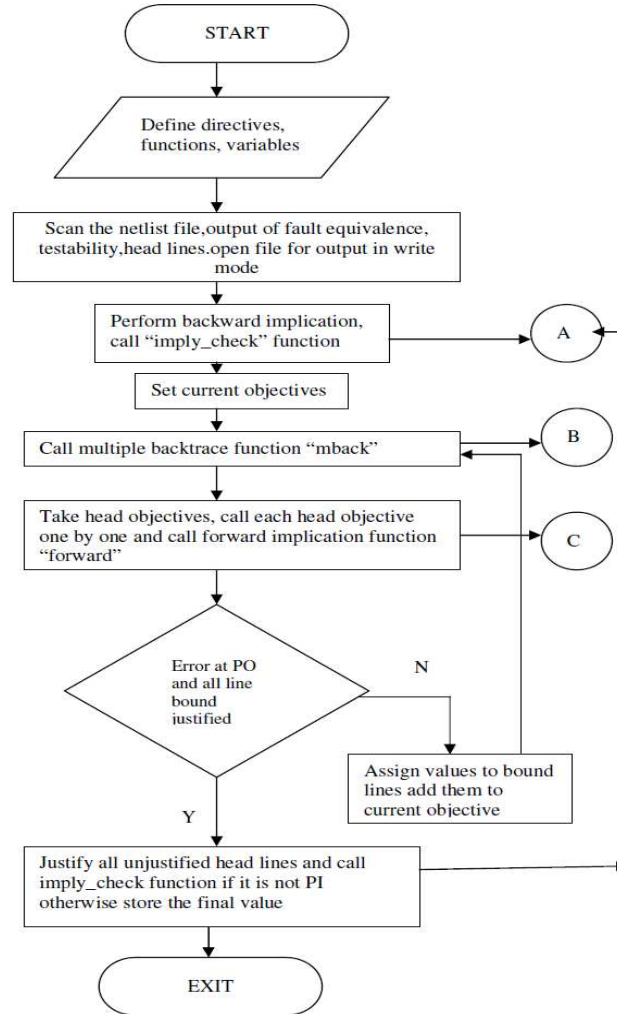


Figure 2 .Proposed ATPG algorithm

FAN algorithm applies unique implication in both forward and backward directions, in the proposed ATPG algorithm first backward implication is performed.

Take justification value v' where v (value-logic 0 or 1) is stuck at fault present on the net. Now perform the implication in backward direction. The flow chart of backward implication is given in figure 3. With justification value start backward implication until primary input reaches. In this strategy use testability measures controllability 0 and controllability 1 .When only one fanin is to be justify then use fanin with minimum controllability value. We got the values of combinational controllability 0 and 1 (CC0, CC1) from [5]. Here '8' is used as don't care bit 'X'.

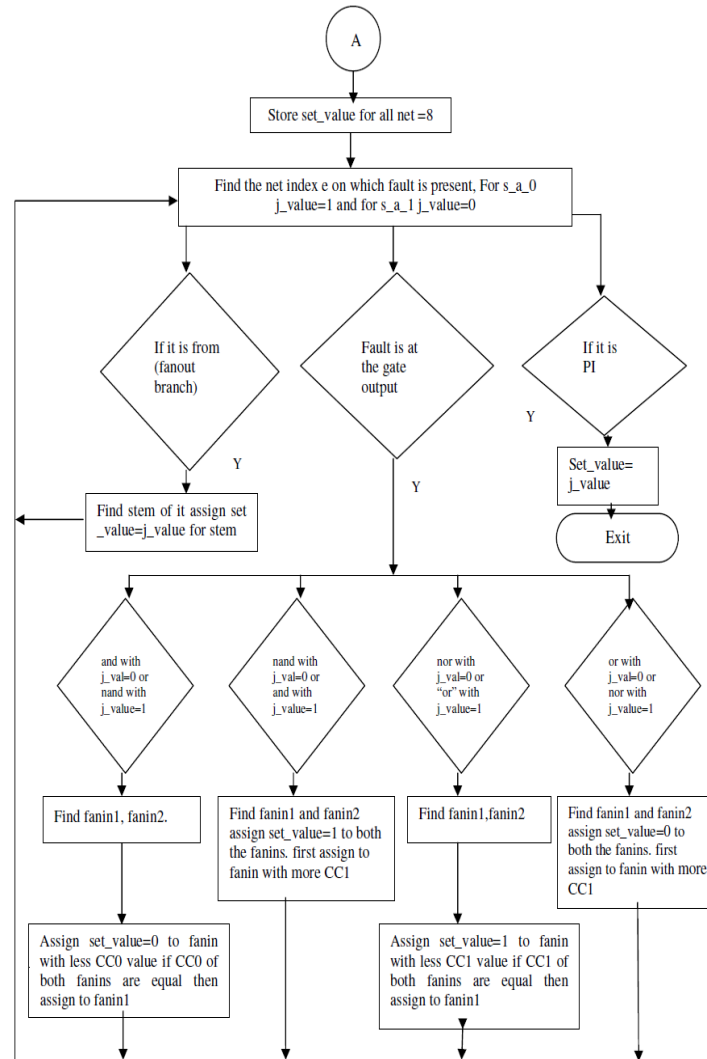


Figure 3. Flow chart of backward implication

Next step is to decide the current objective for multiple backtracing for each fault. If fault is present on PI (primary Input) and it is stem then find branches of it, assign justification values to branches. The branches may be fanin to gate, take gate type and non-controlling value of the gate. This is now 'current objective' value and that net number is current objective netno. If fault is on PI and it is fanin to gate then again repeat the process mentioned as above. In case of xor gate where controlling value is not exists, according to the justification value, both the fanins will be added to current objective netno with corresponding values.

After getting the final current objectives start multiple backtrace with current objective netno and value. We used multiple backtrace as used by FAN algorithm. Do backtracing until the current objective reaches null. If stem comes in between then add it to 'stem objective'. Here an array is formed for 'stem objective' with stem netno, n_0 (netno), n_1 (netno). Where n_0 (netno) shows how many times the logic value 0 comes at the stem netno and n_1 (netno) shows the how many times the logic 1 value comes at stem netno. If one of them is zero then contradiction will not occurs. If $n_0 > n_1$ then take logic value 0 for further backtracing. If $n_1 > n_0$ then take logic value 1 for further

backtracing. Here again testability measures controllability 0 and controllability 1 is used while deciding the value of one fanin or both the fanins. Take stem objective one by one as current objective and do multiple backtracing until head objective reaches. Store the head objective. The flow chart of multiple backtracing is shown in figure 4. The outcome of multiple backtrace is head objective's netno and its logical value.

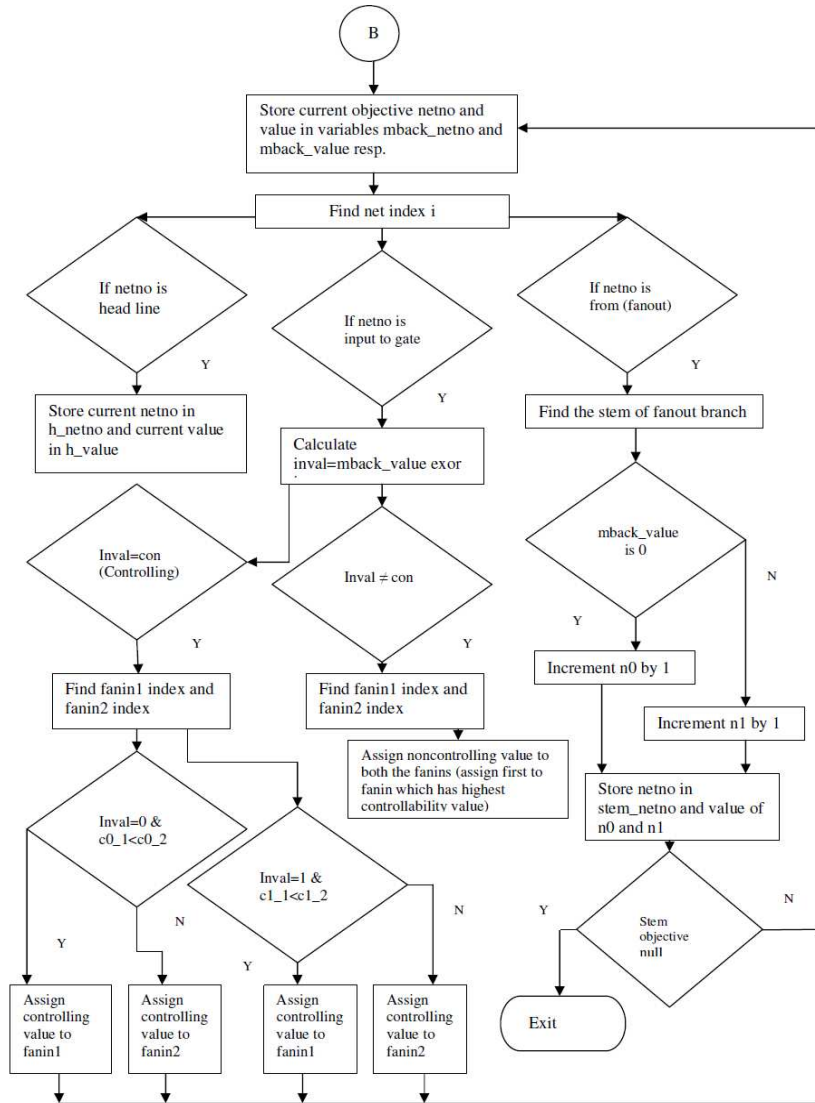


Figure 4. Flow chart of multiple backtrace

In next step take head objective one by one and perform forward implication. This process starts from head line upto PO. If PO reaches then stop, making PO flag ON. In each step of forward implication, check the value on net assigned during the backward implication. If both the values, assigned during backward implication and value to be assign during forward implication are matches then only assign otherwise take the reverse decision, make complimentary value of head objective and do forward implication. Even if there is a mismatch or contradiction then return to the end of program. Forward implication logic is described by the flow chart given in figure 5.

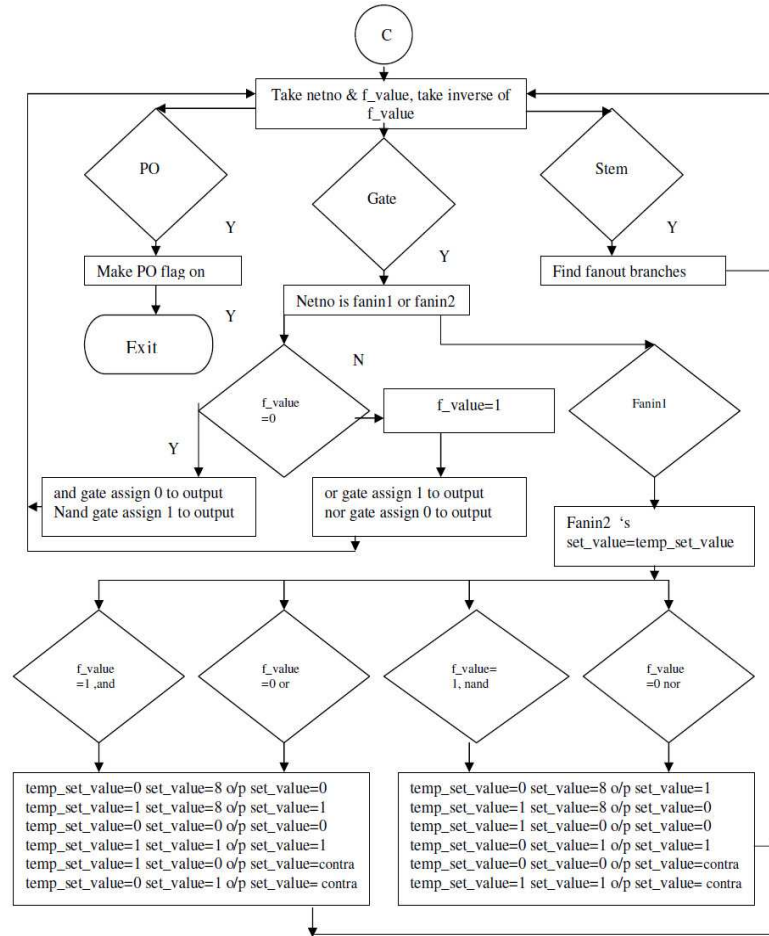


Figure 5 .Flow chart of forward implication

Check whether error reached at PO or not. Also check the D-frontier. It checks the D-frontier at one level. Check whether all bound lines are justified or not. If some bound lines are remain unjustified then according to the gate type, other fanin and output value decide value of unjustified bound line and take them as current objective. Call multiple backtrace function for these lines.

It might be possible that the head line may not be primary input. If it is not primary input then call backward implication function for those lines.

4. EXAMPLE OF PROPOSED ATPG ALGORITHM

ISCAS 85 C17 benchmark circuit is taken to explain the working of proposed ATPG algorithm. Consider the stuck at 0 (s_{a_0}) fault on net no. 23. Now the justification value j value (v') is 1.

1. Initially consider don't care bit X on each net.
2. First apply backward implication: start from net under fault up to the primary input with j value =1. Implied values are shown in figure 6. In this case when only one input is to be justify then compare the controllability values of both the fanins of gate and assign value

to the fanin with less controllability value (cc0 ,cc1 depending upon the value to be justified).

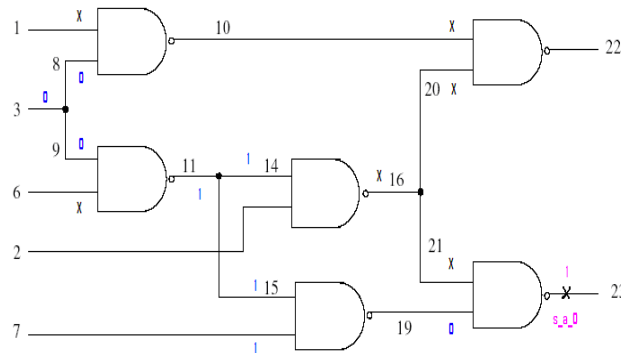


Figure 6. C17 circuit with backward implication values

3. Fault is at PO sothat D-frontier of it becomes gate ahead to net number 23. Initial objective is (23, 1). Select one gate from D-frontier, assign controlling value to the input of that gate which has X value. In this case assign controlling value of nand gate 0 to net no. 21 which has X value. So (21, 1) is current objective.
4. Start multiple backtrace with current objectives.
5. Remove one current objective, which is process entry for multiple backtrace.
6. (21, 1) is fanout branch, assign value to stem 16. (16, 1) is added to stem objective.
7. Now current objective is null but stem objective is present sothat take (16, 1) as current objective.
8. (16, 1) is output of gate. Calculate $inval = V_k \oplus i$, Where V_k is multiple backtrace value and i is inversion value of that gate. Here $inval=0$. So $inval$ is equal to controlling value of nand gate ($0=0$). Assign controlling value C to one of the input i.e. 0. Again it uses controllability to assign value to the input. Here value 0 is assign to input netno 2. Now a current objective is (2, 0).
9. Take one current objective (2, 0). It is head line so stop the backtrace here for this objective. Store net no. and value in head objective (2, 0).The various objective status is shown in table 1.

Current Objective	Process Entry	Stem Objective	Head Objective
(21,1)	(21,1)		
(16,1)	(16,1)	(16,1)	
(2,1)	(2,1)		(2,1)

Table 1. Status of various objectives

10. At this stage stem and current objectives are null. Take head objective one by one and apply forward implication. Forward implication values are shown in figure 7. Here only one head objective is present (2, 0).

11. Here no contradiction has occurred while doing forward implication.

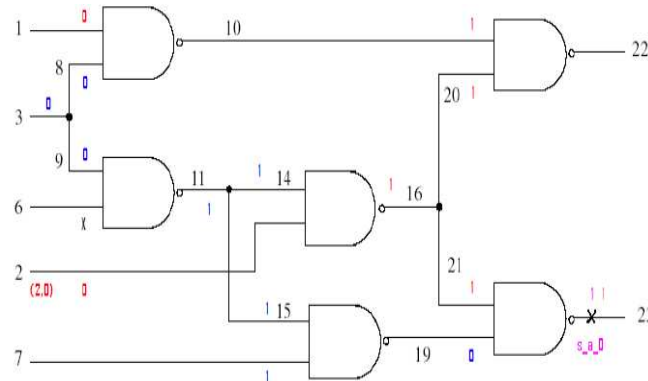


Figure 7. C17 circuit with forward implication values

12. Now check whether there is any unjustified bound line present or not. Netno10 and 22 are bound lines and these are unjustified. Start from PO, unjustified line closest to PO is 22, both of these inputs are not set to value. Take another unjustified bound line. 10 is input to nand gate, output of nand gate is undefined and other input is 1. So set value of 10 as non-controlling value of gate i.e. 1.

13. Now the current objective is (1, 0).The value of netno 1 is set to 0.Thus the test vector generated is 000X1.

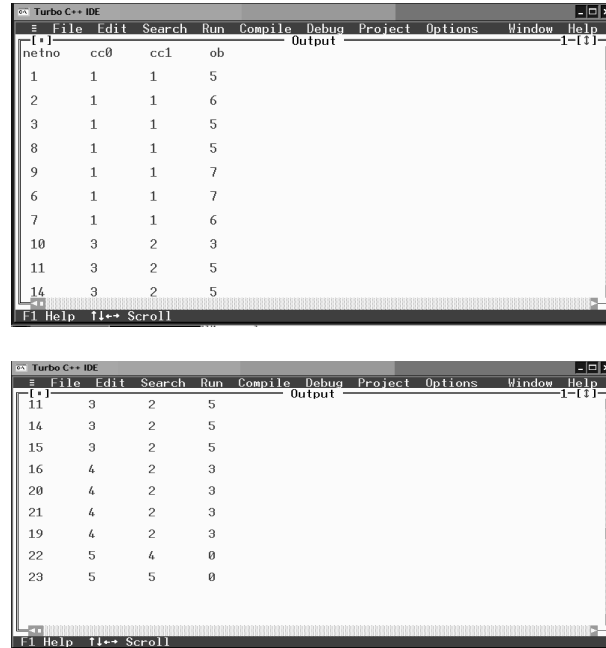
5. RESULTS

The programs for all above concepts are developed in C++.The snap shots of output window are shown in this section. The proposed ATPG algorithm is generic. Figure 8 shows the reduced fault list for ISCAS85 C17 benchmark circuit. The first column of the output window shows the net number and second column shows the fault present on that corresponding net number. The fault present is either stuck at 0 or stuck at 1.



Figure 8. Snap shot of output of ISCAS 85 C17 benchmark circuit for reduced fault set

Figure 9 shows the output of testability measures controllability and observability. Here first column of output window shows the net number, second column is for controllability 0- CC0, third is controllability 1-CC1 and forth column shows the observability-ob for corresponding net number.



The figure consists of two screenshots of the Turbo C++ IDE. The top screenshot shows the output for controllability, and the bottom screenshot shows the output for observability. Both outputs are presented as tables with columns for net number, stuck-at fault (cc0, cc1), and the resulting value (ob).

Controllability Output:

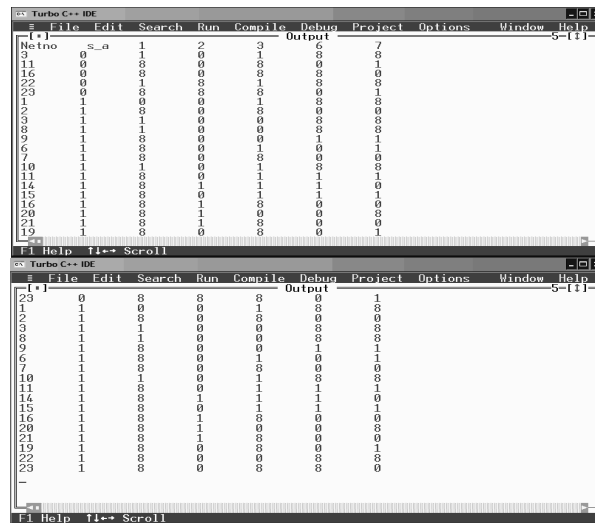
netno	cc0	cc1	ob
1	1	1	5
2	1	1	6
3	1	1	5
8	1	1	5
9	1	1	7
6	1	1	7
7	1	1	6
10	3	2	3
11	3	2	5
14	3	2	5

Observability Output:

netno	cc0	cc1	ob
11	3	2	5
14	3	2	5
15	3	2	5
16	4	2	3
20	4	2	3
21	4	2	3
19	4	2	3
22	5	4	0
23	5	5	0

Figure 9. Snap shot of output of ISCAS 85 C17 benchmark circuit for controllability and observability

Figure 10 shows the test vector generated by proposed ATPG algorithm. The size of test vector is same as the number of PIs. The first column is net number, second column is stuck at fault present on that net either stuck at 0 or stuck at 1. All other columns represents test vector for corresponding fault on net. Here “8” is stored instead of don’t care (x).



The figure consists of two screenshots of the Turbo C++ IDE. The top screenshot shows the output for stuck-at-0 faults, and the bottom screenshot shows the output for stuck-at-1 faults. Both outputs are presented as tables with columns for net number, stuck-at fault (s_a), and the resulting test vector (cc0, cc1, ob).

Stuck-at-0 Output:

Netno	s_a	cc0	cc1	ob
3	0	1	2	3
11	0	8	0	8
16	0	8	0	8
22	0	8	0	8
23	0	8	0	8
1	1	8	0	8
2	1	8	0	8
3	1	8	0	8
8	1	8	0	8
9	1	8	0	8
6	1	8	0	8
7	1	8	0	8
10	1	8	0	8
11	1	8	0	8
14	1	8	0	8
15	1	8	0	8
16	1	8	0	8
20	1	8	0	8
21	1	8	0	8
19	1	8	0	8

Stuck-at-1 Output:

Netno	s_a	cc0	cc1	ob
23	0	8	8	8
1	1	8	0	8
2	1	8	0	8
3	1	8	0	8
8	1	8	0	8
9	1	8	0	8
6	1	8	0	8
7	1	8	0	8
10	1	8	0	8
11	1	8	0	8
14	1	8	0	8
15	1	8	0	8
16	1	8	0	8
20	1	8	0	8
21	1	8	0	8
19	1	8	0	8
22	1	8	0	8
23	1	8	0	8

Figure 10. Snap shot of output of proposed ATPG for ISCAS 85 C17 benchmark circuit

The test patterns are generated for some combinational circuits using proposed ATPG. The number of reduced bits and computing time for these circuits are mentioned in table 2.

Circuit	# net	# bits		Computing time (sec)
		Before fault reduction	After fault reduction	
C17	17	170	110	0.0467000
Circuit 1	29	348	312	0.0560000
Circuit 2	12	72	63	0.0260000

Table 2. Comparisons of bits and computing time

6. CONCLUSION

The proposed combinational ATPG algorithm is less complex. Initial fault reduction is done using fault equivalence method. The algorithm is also follows the testability measures for better assignment of justification values. This can be keep as freeware for test generation and analyses purpose to academicians.

REFERENCES

- [1] Sooryong Lee, Brad Cobb, Jennifer Dworak, Michael R. Grimala and M. Ray Mercer, "A New ATPG Algorithm to Limit Test Set Size and Achieve Multiple Detections of all Faults", in the proceedings of the Design Automation and Test in Europe Conference, 2002.
- [2] Abramovici, M.A. Breuer and A.D. Friedman, "Digital Systems Testing and Testable Design", IEEE Press, NJ 1990
- [3] Nur A. Touba, "Survey of Test Vector Compression Techniques,"IEEE Design & Test of Computers, vol. 23, no. 4, pp. 294-303, July/August, 2006.
- [4] Vaishali Dhare, Usha Mehta, "Implementation of Compaction Algorithm for ATPG Generated Partially Specified Test Data", International Journal of VLSI design & Communication Systems, Pg.no.93-103, Vol.4, No.1, February 2013.
- [5] Vaishali Dhare, Usha Mehta, "Development of Controllability Observability Aided Combinational ATPG with Fault Reduction", Lecture Notes in Computer Science, Recent Trends in Networks and Communications, Springer, Pg. No 682-692, July 2010.
- [6] Hideo Fujiwara., Takeshi Shimono, "On the Acceleration of Test Generation Algorithm", IEEE transaction on computers, Vol C-32, No 12, December 1983, pp.1137-1144.
- [7] S.C. Chang, W.B. Jone and SS. Chang, "TAIR: Testability Analysis by Implication Reasoning", IEEE Trans on CAD, Vol. 19, No. 1, January 2000, pp. 152-160.
- [8] David Bryan, ISCAS'85 benchmark circuits and netlist format, North Carolina State University.
- [9] Micheal Bushnell and Vishwani Agrawal, "Essentials of Electronic Testing". Springer Publication.