ATPG With Efficient Testability Measures and Partial Fault Simulation

Kamal Kumar Jain* James Jacob† Srinivas M K‡

Indian Institute of Science, Bangalore 5GO 012, India

Abstract: In this paper we propose an improved version of the test generation algorithm PODEM (Path Oriented DEcision Making) incorporating a different technique for back tracing and forward implication. We also propose a partial fault simulator which is integrated into the improved PODEM algorithm. The performance of this test generation package (when partial fault simulator is employed) is compared to that of a concurrent fault simulator using deterministically generated test patterns. It is shown that the runtime performance of our algorithm compares favourably with that of the concurrent fault simulator and is less memory intensive. We also present effective heuristics to determine some of the redundant faults and to derive *the test* vectors for some PI faults, by the use of implication relations. Experimental results on all tlie 10 ISCAS benchmark circuits [7]demonstrate that our algorithm is faster and more efficient than the PODEM algorithm for tlicse circuits.

1 Introduction

Many test generation algorithms have been proposed over the years. Recent algorithms such as PODEM [2], FAN [3], SOCRATES [4]etc. have been sucessful in generating test patterns with reasonable efficiency. We assume that the reader is familiar with tlie PODEM algorithm and we shall use some terminologies such as D, $\overline{D}$, PI, PO, backtracing, backtracking, forward implication, etc. without definitions. In this paper, we shall consider multi-input multi-output combinational circuits composed of AND, OR, NAND, NOR, NOT, BUFFER, XOR, and XNOR gates. The type of fault model assumed here is the standard single stuck fault, i.e., all faults can be modeled by lines which are stuck at logical 0 (s-a-0) or stuck at logical 1 (s-a-1). Only one line is assumed to be faulty at any given instant.

*Computer Science and Automation
†Electrical Communication Engg.
‡Computer Aided Design Lab

2 Improved PODEM

2.1 Modification 1

In the original paper on the PODEM algorithm XOR and XNOR gates were not considered anywhere in the flowcharts, although mentioned in an example given in tlie paper. In this algorithm we have included XOR and XNOR gates also.

2.2 Modification 2

If in tlie imply process, the faulty line is set to a logic level which is the same as stuck **at, level,** then backtracking should **be** done immediately, which will lead to faster execution. This point, which is missing in the **flowchart of tlie original** PODEM **description** [2], has been taken **care of in** our implimentation.

2.3 Modification 3

PODEM algorithm uses heuristics to guide the backtracing and implication processes that rely on estimates of the "ease of controlling" internal lines of a circuit to certain logic values, and on estimates of tlie "ease of observing" values on internal lines at primary outputs(POs) of the circuit. Since test for a stuck at fault on a line requires both controlling tlie line to a certain logic value and observing this value at a PO, controllability and observability are usally combined under the more geueral notion of testability. PODEM algorithm seems to lack in the careful consideration of the point of controllability. **An** example of such a case is given in figure **1.**

Consider the logic circuit of figure **1.** Let 0 be the objective logic value at the output of tlie gate M. In this example PODEM will choose tlie upper path in the process of backtracing [2]. But actually, the lower path is **mucli** more easier to control from tlie PIs, because to set a 0 at the output of the gate K either (I_1, I_2, I_3, I_4) or (I_3, I_4, I_5, I_6) should be set to (1,1,1,1), while a 0 at the output of gate L can be obtained by setting only either I_9 or I_{10} to 0. This problem arises due to the

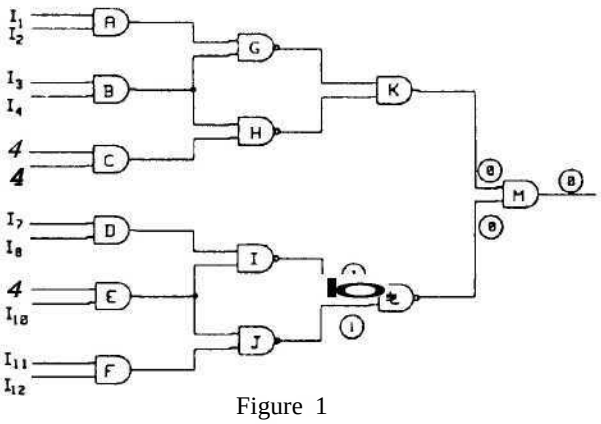

Figure 1

fact that PODEM looks only at one level lower (at the inputs only) in choosing a path, which may be a non-optimal decision, as shown in the last example.

So, it will be better if all the paths from the objective line to tlie PIs can be considered arid tlie appropriate path chosen.

2.3.1 Algorithms For Testability Measures

There are a number of algorithms available for determining the controllability and observability for every line of the circuit. Some of the algorithms are CAMELOT, COMET, TMEAS, SCOAP, and VICTOR [1]. Tlie most popular among them is the "SCOAP" algorithm [5]. In this paper, Testability Measures have been used in order to speed up the test generation algorithm. Controllabilities and observabilities are calculated before the start of tlie test generation procedure. Controllability measures have been used in "easiest" and "hardest" functions [2] of PODEM and observability measures have been used in order to determine the X - path after the imply process [2]. We have introduced tlie SCOAP controllability measures to guide the backtracing process of the test generation algorithm. Since SCOAP is fairly well known; we do not discuss the details of how to obtain tlie controllability measures.

In the example of figure 1, 0-controllabilities of nodes K and L are calculated as S and 7 respectively, so now our algorithm will choose tlie lower path.

It was found experimentally that introduction of observability measures did not yield any definite advantage and hence it was abandoned. However a different heuristic was developed to guide the forward implication process and is explained below.

2.4 Modification 4

A preprocessing was done before tlie begining of tlie test generation procedure, which stores the information at each line about the POs which are likely to be affected by this line. So, when an X-path from a line is to be found, first, tlie logic level of POs which arc affected by this line are checked. If no PO is at X then this implies that there exists no X-path from that line to any PO of the circuit, and immediately backtracking is performed, wliicli will lead to considerable savings in computation time. The results obtained with this modification (preprocessing for X-path) were very effective. So, this modification was retained and the modifications using obscrvnbility measures were discarded.

Tlie results obtained after combining preprocessing for the X-path and modifications for the controllabilty measures are shown in table 1 and table 2[9]. The test vectors were minimised by a simple strategy and their number is also reported in table 1.

3 Fault Simulator

Tlie fault coverage of a set of test vectors is measured through fault simulation. A combination of test generation and fault simulation is effective in speeding up the test generation process. Hence a partial fault simulator which will run along with the test generation algorithm was developed and integrated into our algorithm. The basic strategy is to employ a partial fault-simulation after each test vector (for a fault) is generated, so as to eliminate the covered faults from tlie fault list. Usually faults detected by a given test are dropped from the fault list. This is known as fault dropping. Note that a test vector can be a test for more than one fault.

Parallel, Concurrent, and Deductive Fault simulators[1] require excessive memory and CPU time as circuit size increases. The partial siinulator developed and implemented by us is iiot based on any of tlie above three. This partial fault simulation draws upon and extends the ideas of TEST DETECT [S] developed by tlie inventors of tlie D - algorithm.

An extended version of the partial fault simulator was also developed in course of our work. A comparison was made between this one and tlie earlier partial fault simulator. It was obsrrvctl tlint although tlie partial one had to generate more number of test vectors than the extended one , it was much more efficient in terms of CPU time than the latter. However, the extended version is also supported by our package as a user option and can be used if minimality of test set is crucial.

Table 1: Times and Test set size on SUN 3/60

Circuit	Total Faults	Gates	PI	PO	CPU time in Sec. Methods				Minimal Test Size Methods		
					A	B	C	D	B	C	D
74181	182	86	14	S	4	4	3	5.7	37	31	31
C432	524	160	32	7	98	92	52	67	122	109	99
C499	758	202	41	36	197	183	74	186	358	149	126
C880	942	383	60	26	76	62	43	82	101	90	90
C1355	1574	546	41	32	662	590	274	1785	315	270	176
C1908	1879	880	33	25	640	620	374	1854	435	359	271
C2670	2747	1193	233	140	1091	1020	886	3238	206	184	146
C3540	3428	1669	50	22	2105	1668	1276	4796	439	392	307
C531S	5350	2307	178	123	3068	2354	1425	4742	400	345	261
C62S8	7744	2406	32	32	8726	8081	2515	>15 Hrs	1661	1074	-
C7552	7550	3512	207	108	11148	8277	4985	20374	724	558	353

A: Basic PODEM algorithm Timeout value per fault = 2sec

B: New PODEM = PODEM + controlnbility + preprocessing

C: New PODEM + partial fault simulator

D: New PODEM + extended fault simulator

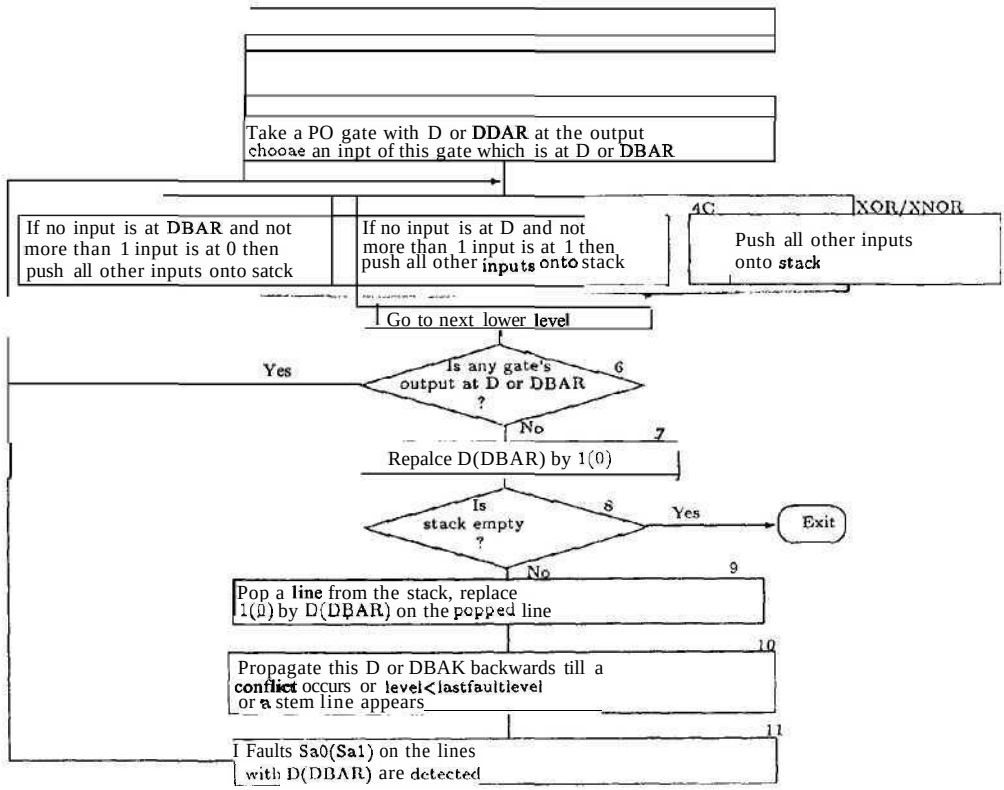

Figure 2: Flow chart of Partial Fault Sin-iulator

3.1 Partial Fault Simulator

Tlie flowchart of the algorithm, developed for tlie partial fault simulator is shown in figure 2. Tlie explanation of every box is given below:

Box 1: A test vector is chosen, wliicli isjust generated by tlie PODEM algorithm.

Box 2: The test vector will be able to detect all tlie saO(sal) faults on the lines, along asensitized path to a PO and having a $D(\overline{D})$ logic value. This is easily understood from the basic concept of path sensitization.

Box *3:* Now a PO gate is taken which is having a D or $\overline{D}$ on its output. Beacuse the output is at D or $\overline{D}$ at least one input will be at D or $\overline{D}$. Choose such an input.

Box 4: Depending upon the function of the gate, other inputs of tlie gate are pushed onto the stack for the purpose of backward implication of D *or* $\overline{D}$. But, for example, in tlie case of AND/NAND gate (box 4A), if any input is at $\overline{D}$ then any other input, cannot be pushed onto tlie stack, because $\overline{D}$ is being replaced by a.O in tlie backward implication and in that case tlie output value of tlie gate will not bo at D or $\overline{D}$.

Boxes 5 and G: Tlie loop of the boxes 5,6, and 4 is esecuted till all the posible lines for which backward implication can be performed are over.

Boxes 7,8,9, and 10: A line is popped and a 1(0) is replaced by $D(\overline{D})$ on the popped line and this D or D is propagated backward till a conflict occurs or the current circuit level becomes less than tlie level of the faulty line for which test vector was generated or a fanout stein line appears. Stopping at fanout stem line is one of tlie reasons, why this fault simulation is partial. The backward implication is stopped at tlie level less than the faulty line level because of tlie definite order of the faults in tlie fault list.

Tlie order of the faults in tlie fault list is giveii below:

1. All the faults of the PIs.
2. The output faults of the next higher level.
3. The input faults of this level.
4. The output faults of the nest higher level and so on.

So, when current level has come down to less tlian tlie faulty line level for which tlie test vector was generated, then there is no point in going further back because tlie faults of the nest lower level would have been considered earlier. This saves a lot of time.

Box 11: this box is similar to box 2.

Example: Tlie values at each line of the circuit after generating the test vector (0x10) for the fault, 1 s-a-1 by PODEM is shown in figure 3. From box 2 of the flowchart it is clear that 15 s-a-0, 14 s-a-0, G s-a-1, and 1 s-a-1 are testable by this test vector. According to box 4A, push line 12 and 13 onto the stack. Line 14 is at D; so according to box 4B push line 8 on to the stack. Now replace all $D(\overline{D})$ by 1(0) in the circuit. A line is popped from the stack. This is line number 8 and having a logic: value *of* 0. Replace this 0 by a $\overline{D}$. Line 4 is a fanout stem line so according to box 10 this D can not be propagated to line 4. 8 s-a-1 is detected by this test vector (box 11). Similar procedures result. in delecting 12 s-a-0, 10 S-a-0, 13 s- a-0, aiid 11 s-a-0. Thus tlie test vector (0x10) could detect 9 faults.

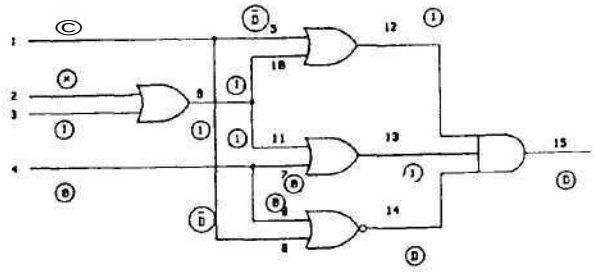

Figure *3:*

3.2 Other Features Of The Package

3.2.1 Implication For PI Faults

If for a PI both stuck-at-0 and stuck-at-1 faults are included in tlie fault list, then tlie test vector lor only one type of fault needs to be generated. Test vector for tlie other type of fault (opposite type) can be determined just by complementing tlie bit (PI) for wliicli the fault, is being considered. For example, in the last example circuit, tlie test vector (1x10) for line 1 s-a-0 can be generated just. by implication, if test vector (0x10) for 1 s-a-1 has already been generated.

3.2.2 Implications For Untestable Faults

Three observations:

1. If a saO(sal) fault on a PI is untestable then sal(sa0) fault on that PI is also untestable.
2. If a fault / is untestable, all faults in its equivalence class are also uiitcstable.
3. If a dominating fault is untestable, all faults dominated by this untestable fault are also untestable.

The truth of these observations is obvious and hence no formal proof is given.

Example: If stuck-at-0 fault, at the input of an AND

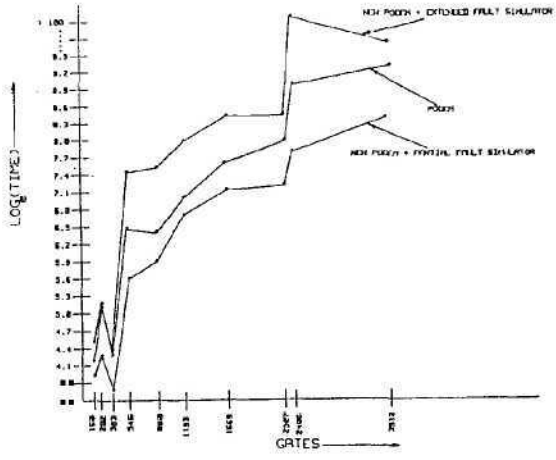

Figure **4:** Performance of different versions

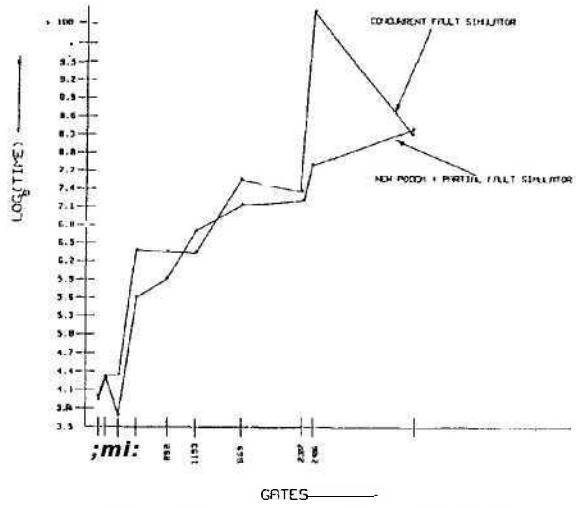

Figure 5: Time comparison with CFS

gate is found untestable then the stuck-at-0 faults on all other inputs and the output of that gate will be untestable. These faults are declared untestable by the property of fault equivalence.

This property can be propagated further forward and backward until a fanout stem line appears. For example, in tlie last example circuit, faults 10 s-a-1 and 5 s-a-1 can be declared untestable by the implication of 12 s-a-1 being untestable.

This particular feature of the pacltage is very important since in general tlie untestable faults are hard to prove redundant and PODEM will take a lot of time to deal with these faults. So, without going into the test generation program, many of these type of faults are declared untestable just by implication.

3.2.3 Extended Version Of Partial Fault Simulation

Tlie difference in the partial fault simulation and tlie extended partial fault simulation appears only in the box 10 of the flowchart shown in figure 2. In partial fault simulation the backward implication of D or $\overline{D}$ stops at a fanout stem line, while in tlie extended partial fault simulation, after setting the fanout stem line to D or $\overline{D}$ (as may be the case), a forward implication is performed and if the logic value of the PO which was coiisiderd in tlie begining of the simulation remains unchanged at D or $\overline{D}$ then tlie backward implication is continued from the fanout stem. The forward implication consumes a large amount of time making tlie extended partial fault simulation inefficient. The results for both the versions are shown in table 1 and table 2.

4 Results

Tlie improved PODEM algorithm along with the integrated fault simulator was coded in C and implemented on a SUN 3/60 workstation. It, consists of about 2000 lines of code. Tlie efficiency of our strategy of integrating a partial fault simulator into an improved version of PODEM algorithm is demonstrated by comparing its performance to that of a Concurrent fault simulator for all tlie ISCAS benchmark circuits. Fault simulation is generally known to be less expensive compared to computation intensive test generation process, as we do not have baclitraclting in fault simulation. Tlie test vectors derived by PODEM (employing Partial Fault Simulation) were given to CFS, a concurrent fault simulator available in the CAD Lab. at IISc. The simulator evaluated the coverage of the deterministic test vectors generated by PODEM for all the 10 ISCAS benchmark circuits. Contrary to our expectations it was found that in many cases the time taken by CFS to evaluate the coverage of tlie test. vectors was greater than tlie time taken by our package to generate the same vectors. The results are shown in table 3. CFS is memory intensive and hence it ran out of virtual meniory for C6288 on a SUN 3/60 workstation with 4MB main memory, while our PODERI implimentation could handle all tlie benchmark circuits. Tlie results clearly demonstrate that a combination of test generation and fault simulation can compare favourably in performance to that of a stand alone fault simulator, even though deterministic test generation is more compute intensive as it employs backtracking. The concurrent fault simulation naturally reported a higher fault coverage than that determined

Table 2: Fault coverage with different versions on SUN 3/60

Circuit	Total Faults	Faults Aborted Methods				Faults Redundant Methods				Fault Coverage (%) Methods			
		A	B	C	D	A	B	C	D	A	B	C	D
74181	182	0	0	0	0	1	1	1	1	99.50	99.50	99.50	99.50
C432	524	26	26	13	13	0	0	0	0	95.04	95.04	97.52	97.52
C499	758	2G	2G	1G	20	0	0	0	0	96.57	96.57	97.9	97.36
C880	942	0	0	0	0	0	0	0	0	100.00	100.00	100.00	100.00
C1355	1574	8	16	8	8	0	0	0	0	99.49	98.98	99.49	99.49
C1908	1879	28	57	41	37	4	5	5	5	98.30	93.60	97.60	97.80
C2S70	2747	118	147	138	103	27	29	29	29	94.70	93.60	93.90	95.20
C3540	3428	384	301	229	181	15	31	31	31	88.4	90.3	92.40	93.80
C5315	5350	200	127	98	97	36	36	38	36	95.60	97.00	97.50	97.50
C6288	7744	111	111	12	-	32	32	32	-	98.20	98.80	99.40	-
C75S2	7550	2475	1162	949	771	48	52	52	52	66.60	83.90	86.70	89.JO

A: Basic PODEM algorithm D: New PODEM = PODEM + controlability + preprocessing

C: New PODEM + partial fault simulator D: New PODEM + extended fault simulator Timeout value per fault = 2sec

by PODEM and this is easily explained by the fact that the simulator we have integrated into PODEM is a partial one.

5 Conclusions

The results indicate that our algorithm is better than the basic PODEM algorithm. Testabillity measures are very effective. A partial fault simualtor is more effective than a complete fault simulator. Some more modifications can make this algorithm more efficient. Some of these modifications are multiple path backtracing[3], unique sensitization [3], identification techniques for the redundant faults [6] and a better fault simulator.

Acknowledgement

The authors would like to tha.nk Prof. Sharad C. Seth of University of Nebraska, Lincoln for providing *a Pascal* implementation of basic PODEM algorithm and a concurrent fault simulation package called CFS.

References

[i] V.D. Agarwal and Sharad C. Seth, "Test Generation for VLSI Chips," Tutorial, IEEE Computer Society Press, Computer Society Order Number 7S6, 1988.

[2] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests For Combinational Logic Circuits," IEEE Trans, Computers, pp. 215-222, March,1981.

[3] H. Fujiwara and T. Shimono, "On The Acceleration of Test Generation Algorithm," IEEE Trans. Computers, pp 1137-1155,Dec. 1983.

Table 3: CPU time comparison on SUN 3/60

Circuit	New PODEM + Partial Fault Simulator	Concurrent Fault Simulator
C432	52 Sec.	57 Sec.
C499	74 Sec.	75 Sec.
C880	43 Sec.	1 . Sec.
C1355	274 Sec.	580 Sec.
C1908	374 Sec.	570 Sec.
C2670	886 Sec.	580 Sec.
C3540	1276 Sec.	1817 Sec.
C5315	1425 Sec.	1566 Sec.
C6288	2515 Sec.	Not. Completed (lime > 20 Hrs.)
C75S2	4985 Sec.	3429 Sec.

[I] Schulz, M. H., E. Trischler, and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation Systems," IEEE Trans. on CAD, pp. 126-137, Jan. 1988.

[5] L. H. Goldstein and E. L. Thigpen,"SCOAP: Sandia Controllability/Observability Analysis Program," Proc. J7th DCS. Auto. Conf., Minneapolis, MN, pp. 190-196, June 1980.

[G] M. 11. Schulz and E. Auth, "Advanced Automatic Test Pattern Generation and Redundancy Identification Techniques," IEEE FTCS 1988.

[7] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinatioanl Benchmark Circuits and a Target Translator in Fortran," Proc. IEEE Int. Symposium on Circuits and Systems; Special Session on ATPG and Fault Simulation, June 1985.

[8] J.P. Roth, "Programmed Algorithms to Compute Tests to Detect and Distinguish between Failures in Logic Circuits," IEEE Trans. Computer, vol. EC-16, no.5, Oct. 19G7, pp. 567-580.

[9] Kamal Kumar Jain, "Development of an efficient software package for Test Generation of Combinational Circuits," M.E Thesis. Dept. of Electrical Communication Engineering, Indian Institute of Science, Bangalore, June 1990.