Proceedings of the IEEE VLSI Test Symposium, pp. 446-452, April 1998

New Techniques for Deterministic Test Pattern Generation *

Ilker Hamzaoglu and Janak H. Patel
Center for Reliable & High-Performance Computing
University of Illinois, Urbana, IL 61801

Abstract

This paper presents new techniques for speeding up deterministic test pattern generation for VLSI circuits. These techniques improve the PODEM algorithm by reducing number of backtracks with a low computational cost. This is achieved by finding more necessary signal line assignments, by detecting conflicts earlier, and by avoiding unnecessary work during test generation. We have incorporated these techniques into an ATPG system for combinational circuits, called ATOM. The performance results for the ISCAS85 and full scan version of the ISCAS89 benchmark circuits demonstrated the effectiveness of these techniques on the test generation performance.

1 Introduction

As the complexity of VLSI circuits and their quality requirements are increasing, the problem of test generation is becoming more important. Since the scan-based design for testability techniques are increasingly used in very large circuits, test generation for combinational circuits is getting even more important for these circuits.

The test 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 [6]. 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 [4], it is very important to find efficient techniques to speed up the test generation process. After the branch and bound search algorithm introduced in PODEM [6], a number of techniques have been proposed in the literature to improve its performance [1, 5, 7, 9, 12, 13, 14, 17].

The test generation problem can also be viewed as a boolean satisfiability (SAT) problem. Because it is possible to reduce it to an equivalent SAT problem using a polynomial time reduction. Recently a number of test generation systems are introduced for solving the test generation problem using fast SAT solvers [10, 15, 16].

In this paper, we have introduced new structure based techniques for speeding up test pattern generation for VLSI circuits. These techniques are improved unique sensitization, improved backtrace with X-path and conflict check, multiple primary input assignments, LookBack, and Support Regions. They improve the PODEM algorithm by reducing number of backtracks with a low computational cost. This is achieved by finding more necessary signal line assignments, by detecting conflicts earlier, and by avoiding unnecessary work during test generation.

To be able to assess the effectiveness of the proposed techniques, we have developed a deterministic ATPG system for combinational circuits, called ATOM, and incorporated these techniques into the test generator. The test generator uses the single stuck at fault model and it is based on the PODEM algorithm. In addition to the techniques we have proposed, it also employs some of the test generation techniques introduced in the literature.

The experimental results on the ISCAS85 and full scan version of the ISCAS89 benchmark circuits [2, 3] demonstrated the effectiveness of these techniques on the test generation performance. ATOM detected all the testable faults and proved all the redundant faults to be redundant with very small number of backtracks in a short amount of time. Since we did not use an initial random test generation phase and we obtained similar results both with and without using fault simulation, the results demonstrated the robustness of the techniques we have developed.

The rest of the paper is organized as follows. Section 2 introduces the test generation system. Section 3 discusses the proposed test generation techniques in detail. Section 4 presents the experimental results. Finally, section 5 presents the conclusions.

*This research was supported in part by the Semiconductor Research Corporation under contract SRC 96-DP-109 and in part by DARPA under contract DABT63-95-C-0069.

2 The Test Generation System

We have developed an ATPG system for combinational circuits, called ATOM. ATOM is composed of a parallel-pattern fault simulator and a PODEM based deterministic test pattern generator. To be able to assess the robustness of the test generation techniques we have developed, it does not use a random test generation phase. It is designed in a object oriented style and implemented in C++.

The fault simulator is a parallel-pattern simulator that simulates 32 patterns at a time [17]. It is also a parallel-fault simulator, since it simulates the faults in a fanout free region until the fanout stem, and it only simulates the faults on the fanout stem in the rest of the circuit. It also employs the fault simulation techniques used in PROOFS fault simulator [11].

The test generator is based on the PODEM algorithm. It uses the single stuck at fault model and the five-valued logic with logic values 0, 1, X, D, $\overline{D}$. In addition to the new test generation techniques that we have developed, the test generator uses the local implications [5], static global implications [12], X-path check [6], unique sensitization [5], and dynamic unique sensitization [13] techniques proposed in the literature. We did not use any compact test set generation techniques in the test generator.

In [8, 13, 14, 16], it is reported that using advanced test generation techniques for easy to detect faults degrades the overall performance of an atpg system. We have made the same observation. Therefore, ATOM runs in two phases. In the first phase simple test generation techniques, local implications, X-path check and unique sensitization, are used to test as much faults as possible with a maximum backtrack limit of 0, i.e. without any backtracks. Then in the second phase advanced test generation techniques are used to test the remaining faults.

3 New Test Generation Techniques

We will now describe the new techniques we have proposed to improve the test generation performance.

1. Improved Unique Sensitization: The dynamic unique sensitization technique introduced by the SOCRATES system [13] improved the effectiveness of the unique sensitization technique proposed by the FAN algorithm [5]. This technique identifies more necessary logic assignments due to fault propagation constraints that are not considered in the basic unique sensitization technique. It finds the necessary signal line assignments by taking the intersection of the dynamic dominators of the gates in the D-frontier. We have improved this technique by taking the intersection of the logic implications of the dynamic dominators.

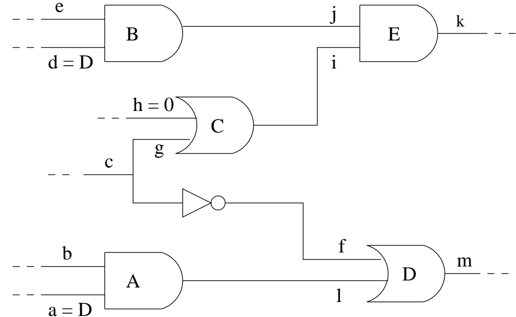

Figure 1: Improved Unique Sensitization

We have found that this new dynamic unique sensitization technique discovers a large number of necessary logic assignments that can not be discovered by SOCRATES. These necessary logic assignments speed up the test pattern generation by identifying conflicts earlier.

The new technique finds the necessary signal line assignments as follows.

> Suppose that the D-frontier consists of the gates $d_1, d_2, ..., d_n$, and sla_{d1} denotes the set of signal line assignments that are necessary to propagate the fault from gate d_1. Signal line assignments in sla_{d1} can be determined by applying the static and dynamic unique sensitization techniques to gate d_1. Then, CO = $Logic_Implications(sla_{d1}) \cap ... \cap Logic_Implications(sla_{dn})$ is the set of necessary signal line assignments.

Example: Figure 1 illustrates a situation in which this new technique discovers a necessary logic assignment that can not be discovered by SOCRATES. In this example, let us assume that the D-frontier consists of the gates A and B. In this situation, new technique will determine that $c = 1$ is a necessary logic assignment. Because to propagate the fault from gate A, signal f should be set to 0, and $f = 0$ implies $c = 1$. Similarly to propagate the fault from gate B, signal i should be set to 1, and $i = 1$ implies $c = 1$. Therefore signal c should be set to 1 regardless of through which gate the fault will be propagated. SOCRATES will not be able to find this necessary logic assignment, because $c = 1$ is not a dynamic dominator of either gate A or gate B.

2. Improved Backtrace with X-path and Conflict Check: After selecting a gate for fault propagation, PODEM carries out X-path check for this gate to determine whether there is a path from this gate to at least one of the primary outputs on which all lines are at X. If there is no such path then it selects another gate. This X-path check is carried out before the backtrace procedure. Although an X-path

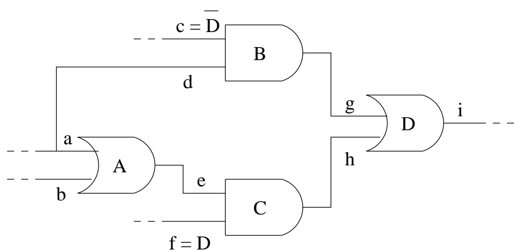

Figure 2: Backtrace with X-Path Check

may exist before the backtrace starts, an OR-choice selected during backtrace may destroy these X-paths. We have improved the backtrace procedure to detect these situations during backtrace before committing to a primary input assignment.

Example: Figure 2 illustrates such a situation. In this case, let us assume that the objective is to propagate the fault from gate C. Therefore signal e should be set to 1. This can be justified by either setting signal a or b to 1. However, setting signal a to 1 implies that signal i will be 1. If signal i becomes 1, then there will not be any X-paths remaining for the gate C. Thus if the backtrace procedure continues by selecting $a = 1$ choice, the resulting primary input assignment may not satisfy the initial objective, propagating the fault from gate C to a primary output. Therefore our test generator discards $a = 1$ OR-choice, and it continues to backtrace by selecting $b = 1$ OR-choice.

PODEM uses only the controllability heuristics to select an OR-choice at an OR decision point in backtrace. However, during backtrace sometimes an OR-choice conflicts with the choices selected earlier in the backtrace. We have improved the backtrace procedure with an extensive conflict check to determine these conflicts before committing to a primary input assignment. This avoids the test generator from going into a nonsolution area in the search space which might result in a significant number of backtracks.

Example: Figure 3(a) illustrates a situation in which a later OR-choice directly conflicts with an earlier AND-choice. In this example, let us assume that $e = 0$ is the current objective of the backtrace procedure. Justifying this objective requires setting the signals c and d to 1. If the backtrace procedure chooses the AND-choice $c = 1$, then this can be justified by either setting signal a or signal b to 0. The OR-choice $b = 0$ clearly conflicts with the $d = 1$ AND-choice. In this example, the improved backtrace procedure will determine this conflict, and it will continue to backtrace by choosing the $a = 0$ OR-choice.

The above example illustrates the basic improvement. However, there are more complicated cases in

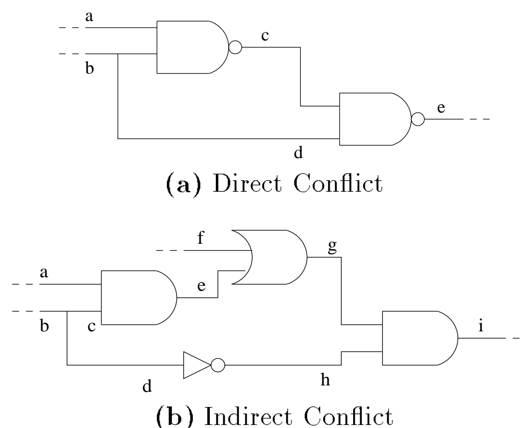

Figure 3: Backtrace with Conflict Check

which a later OR-choice does not directly conflict with the earlier choices, rather the logic implications of this OR-choice conflicts with the earlier choices.

Example: Figure 3(b) illustrates such a situation. In this example, let us assume that $i = 1$ is the current objective of the backtrace procedure. Justifying this objective requires setting the signals g and h to 1. If the backtrace procedure chooses the AND-choice $g = 1$, then this can be justified by either setting signal e or signal f to 1. Although $e = 1$ does not directly conflict with the earlier choices, setting signal e to 1 implies that $h = 0$ which conflicts with the $h = 1$ AND-choice. The improved backtrace algorithm will determine this conflict, and it will continue to backtrace by choosing $f = 1$ OR-choice.

The improved backtrace procedure works as follows. During the backtrace a backtrace-stack is built to keep track of the earlier OR-choices. At each OR decision point, we push all the choices and the logic implications of the selected choice to the stack marking the selected choice as tried. At each OR decision point the backtrace procedure checks whether the OR-choice destroys all the X-paths of the initial objective and whether it directly or indirectly conflicts with the choices selected earlier in the backtrace. If an OR-choice causes a conflict, one of the untried choices is selected next. If all the choices at an OR decision point cause a conflict, the backtrace procedure backtracks until the last OR-choice. This way it tries all the possible OR-choices until it backtraces to a primary input without any conflicts or there is no untried OR-choice left in which case the test generator backtracks.

For some circuits, this technique may cause a large number of backtracks during a backtrace degrading the overall performance of the test generator. To avoid this situation, we use a maximum backtrack limit of 20 in our test generator. If this limit is exceeded during a

```
LOOKBACK(DT: decision-tree, LookBack-Backtrack_Limit: int)
{
  While DT is not empty
  {
     current_pi = last primary input in DT
     Complement the logic value of current_pi
     no_of_backtracks = 0
     DT' = DT    // save the DT
     While (no_of_backtracks < LookBack-Backtrack_Limit)
     {
          Perform branch and bound search starting from DT
          If test vector is found then return SUCCESS
          If conflict then
          {
             no_of_backtrack = no_of_backtracks + 1
             backtrack to the previous primary input decision in DT
          }
     }
     DT = DT'    // restore the DT
     backtrack to the previous primary input decision in DT
  }
  return FAILURE
}
```

Figure 4: LookBack Algorithm

backtrace, this technique is no longer used for this circuit, i.e. the backtrace procedure is executed without any conflict checks. Among the ISCAS circuits, this maximum backtrack limit is exceeded only during the test generation for c6288 when the test generator is executed without fault simulation.

3. Multiple Primary Input Assignments: We have improved the single backtrace technique of PODEM to assign logic values to more than one primary input at the end of a backtrace. This technique speeds up the test pattern generation by reducing the number of backtraces.

Multiple primary input assignments is achieved by exploiting the backtrace-stack constructed during the backtrace procedure. The backtrace-stack contains the OR-choices selected during backtrace and their logic implications. We have improved the test pattern generator to check the backtrace-stack after a backtrace to determine whether any of the OR-choices selected during backtrace implies a logic value for a primary input which is not assigned a logic value by the backtrace procedure and which is not already in the primary input decision tree. If such a logic implication exist, we then put the corresponding primary input into the decision tree by assigning the implied logic value.

4. LookBack Technique: When a conflict occurs in the test generation process, PODEM backtracks to the last primary input decision. The dependency-directed backtracking technique introduced by the LEAP system [14] improved the PODEM algorithm to nonchronologically backtrack to an earlier primary input decision. When a conflict occurs in the test generation process, LEAP carries out an extensive analysis to determine the cause of the conflict and it backtracks to the last primary input decision that is responsible from the conflict. However, the conflict analysis is a quite costly process.

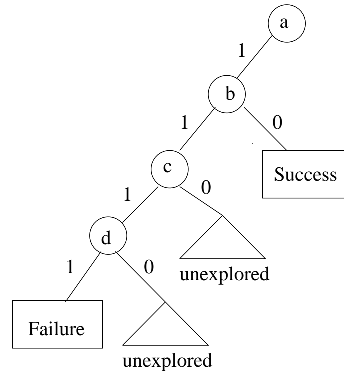

Figure 5: LookBack Example

We have improved the PODEM algorithm to nonchronologically backtrack to an earlier primary input decision using a new search technique called LookBack. LookBack is a probing technique which is triggered after the maximum backtrack limit of the test generator is reached. In this case, instead of continuing with the branch and bound search technique, we save the current search state, i.e. the primary input decision tree, and use the LookBack technique. LookBack is effective in the situations where an earlier primary input decision causes the test generator to go into a large nonsolution area.

As it is shown in Figure 4, LookBack procedure takes two parameters, the current decision tree and a lookback-backtrack limit. In our test generator we used a lookback-backtrack limit of 1. Starting from the last primary input decision in the decision tree LookBack procedure complements the logic value of the primary input and runs the PODEM algorithm until the given lookback-backtrack limit is reached. If no solution is found within this lookback-backtrack limit, it backtracks to the previous primary input decision in the original decision tree. This way instead of spending a large amount of time in the nonsolution area, it quickly backtracks to the earlier primary input decision that caused the test generator to enter into the nonsolution area. Since LookBack is not a complete algorithm, if it fails to find a test vector then the test generator continues to the branch and bound search starting from the original decision tree.

Example: Figure 5 illustrates a situation where LookBack technique helps the test generator to quickly get rid of the nonsolution area and detect the fault. In

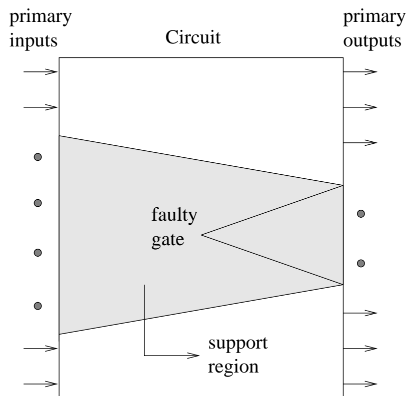

Figure 6: Support Region Example

this example, let us assume that the current decision tree consists of the decisions $a = 1$, $b = 1$, $c = 1$ and $d = 1$. The triangles in the figure represent the unexplored nonsolution areas, one of the square boxes represents a conflict, i.e. failure, and the other one represents a solution. In this example, the standard branch and bound technique will take a large number of backtracks to come back to the decision tree node b and find the solution. Since LookBack technique only explores a small portion of each triangle, the size of which is determined by the lookback-backtrack limit given to the LookBack procedure, it quickly backtracks to the decision tree node b and finds the solution.

5. Support Regions:

For a given fault the *support region* of a circuit is defined to be the part of the circuit that may affect the test generation process for this fault. As illustrated in Figure 6, for a given faulty gate this region consists of the gates in the fanin cones of the primary outputs that are in the fanout cone of this faulty gate. Since the gates outside the support region do not fanout to the same primary outputs with the faulty gate, they do not effect the test generation process for this faulty gate. Therefore, even though a logic value of a gate in the support region may imply a logic value for a gate outside the support region, during test generation it is not necessary to carry out forward implications outside the support region of the target fault. This technique avoids considerable amount of unnecessary work during the test generation of a fault.

Although this technique requires the computation of the support regions, the cost of this computation can be reduced by computing this information only once for a fanout free region in a demand-driven way. Since the support region for all the faults in a fanout free region is the same, this information can be computed once for a fanout free region, and it can be used for all the faults in the same fanout free region. Moreover, since we do not use this technique in the fault simulator, it is not necessary to compute the support regions for the faults that are detected by fault simulation. Therefore, we compute the support regions only for the faults that are targeted by the test generator.

4 Experimental Results

ATOM is tested on the ISCAS85 and full scan version of the ISCAS89 benchmark circuits on a 200 MHz Pentium Pro PC with 128MB RAM running Linux 2.0.0 using GNU CC version 2.8.0. Table 1 presents the performance results. In all these experiments a maximum backtrack limit of 6 is used. The LookBack technique is triggered only after 6 backtracks, and this only happened for the circuits c5315, c6288 and s13207 when ATOM is executed without fault simulation.

The columns Det, Red, Abt, Vect present the number of faults detected, the number of faults proven to be redundant, number of aborted faults and the number of test vectors generated respectively. We did not present the the number of faults detected, the number of faults proven to be redundant and the number of test vectors generated for the without fault simulation case. Because the number of faults detected and the number of faults proven to be redundant are same for both with and without fault simulation cases, and the number of test vectors for the without fault simulation case is same as the number of detected faults, i.e. one test vector is generated for each fault. The results show that ATOM detected all testable faults and proved all redundant faults to be redundant. Since we used a parallel-pattern fault simulator, and we didn't use any compact test set generation techniques, the test set sizes generated by ATOM are relatively large. In this paper, we do not address the compact test set generation problem.

The TB column in the table presents the total number of backtracks done during the test generation for the corresponding circuit. The MDB column shows the maximum number of backtracks required to detect a fault and the MRB column shows the maximum number of backtracks required to prove that a fault is redundant. The LT1B column shows the number of faults that required more than one backtrack to be detected or to be proven as redundant. The results show that ATOM detected all testable faults and proved all redundant faults to be redundant with a very small number of backtracks. This shows that the new techniques are quite successful in reducing the number of backtracks.

The Time column for the with fault simulation

Circuit	ATOM with Fault Simulation									ATOM without Fault Simulation					
	Det	Red	Abt	Vect	TB	MDB	MRB	LT1B	Time	Abt	TB	MDB	MRB	LT1B	Time
c432	520	4	0	110	0	0	0	0	0.1	0	0	0	0	0	0.3
c499	750	8	0	127	0	0	0	0	0.3	0	0	0	0	0	1.1
c880	942	0	0	133	0	0	0	0	0.1	0	0	0	0	0	0.3
c1355	1566	8	0	192	3	1	0	0	2.0	0	66	3	0	12	12.7
c1908	1870	9	0	210	3	2	0	1	0.8	0	7	2	0	2	2.8
c2670	2630	117	0	242	4	0	4	1	1.1	0	4	0	4	1	3.9
c3540	3291	137	0	264	0	0	0	0	1.9	0	7	3	0	2	9.9
c5315	5291	59	0	216	0	0	0	0	1.2	0	28	17	0	3	6.7
c6288	7710	34	0	64	0	0	0	0	1.3	0	12478	32	0	1371	86.3
c7552	7419	131	0	393	7	1	0	0	4.6	0	83	4	0	5	23.6
s208	217	0	0	65	0	0	0	0	0.0	0	0	0	0	0	0.0
s298	308	0	0	52	0	0	0	0	0.0	0	0	0	0	0	0.1
s344	342	0	0	62	0	0	0	0	0.0	0	0	0	0	0	0.1
s349	348	2	0	65	0	0	0	0	0.0	0	0	0	0	0	0.1
s382	399	0	0	72	0	0	0	0	0.0	0	0	0	0	0	0.1
s386	384	0	0	109	0	0	0	0	0.1	0	0	0	0	0	0.1
s400	418	6	0	71	0	0	0	0	0.0	0	0	0	0	0	0.1
s420	455	0	0	98	0	0	0	0	0.1	0	0	0	0	0	0.1
s444	460	14	0	77	0	0	0	0	0.0	0	0	0	0	0	0.1
s510	564	0	0	90	0	0	0	0	0.1	0	0	0	0	0	0.2
s526	754	1	0	107	0	0	0	0	0.1	0	0	0	0	0	0.1
s641	467	0	0	99	0	0	0	0	0.1	0	0	0	0	0	0.2
s713	543	38	0	100	0	0	0	0	0.1	0	0	0	0	0	0.2
s820	850	0	0	190	0	0	0	0	0.1	0	0	0	0	0	0.3
s832	856	14	0	200	0	0	0	0	0.2	0	0	0	0	0	0.3
s838	857	0	0	183	0	0	0	0	0.2	0	0	0	0	0	0.5
s953	1079	0	0	138	0	0	0	0	0.2	0	0	0	0	0	0.6
s1196	1242	0	0	227	0	0	0	0	0.4	0	0	0	0	0	0.9
s1238	1286	69	0	240	9	1	6	1	0.6	0	12	2	6	2	1.2
s1423	1501	14	0	135	0	0	0	0	0.2	0	0	0	0	0	0.6
s1488	1486	0	0	196	0	0	0	0	0.2	0	0	0	0	0	0.9
s1494	1494	12	0	191	0	0	0	0	0.3	0	0	0	0	0	0.9
s5378	4563	40	0	358	0	0	0	0	0.8	0	0	0	0	0	2.6
s9234	6475	452	0	660	37	0	2	12	7.0	0	43	2	2	14	13.8
s13207	9664	151	0	709	18	0	6	5	6.6	0	51	12	6	9	20.8
s15850	11336	389	0	643	0	0	0	0	7.0	0	0	0	0	0	23.9
s35932	35110	3984	0	129	0	0	0	0	6.4	0	128	1	0	0	74.5
s38417	31015	165	0	1458	0	0	0	0	15.9	0	0	0	0	0	46.6
s38584	34797	1506	0	989	0	0	0	0	14.1	0	0	0	0	0	40.9
Total	181259	7364	0	9664	81	2	6	20	74.2	0	12907	32	6	1421	378.4

Table 1: ATPG Results

case presents the total test generation time in seconds for the corresponding circuit, including the time for computing the dominators and the static global implications, the fault simulation time and all the input/output times. The Time column for the without fault simulation case also presents the total test generation time in seconds, but without including the time for writing the generated test vectors to a file. Since for each of the largest three ISCAS89 circuits the size of the test vector file is around 60MB, there is a considerable input/output time for these circuits. Since this input/output time is the same independent of the test generation algorithm used and it is determined by the performance of the input/output subsystem of the machine used, we did not include it in the total time presented for the without fault simulation case. The results show that ATOM, both with and without fault simulation, detected all testable faults and proved all redundant faults to be redundant in a short amount of time without aborting any faults. These performance results demonstrate the effectiveness and robustness of the test generation techniques we have developed.

We have compared the performance of ATOM with one of the fastest atpg systems reported, TEGUS [16] which uses a satisfiability based algorithm for test generation. Table 2 presents the comparison. The performance of TEGUS is reported for 18 ISCAS circuits, 10 ISCAS85 circuits and the 8 largest ISCAS89 circuits, using a DEC 7000/610 AXP. The Abt column presents the total number of aborted faults for these

	With Fault Sim		Without Fault Sim	
	Abt	Time/fault (ms)	Abt	Time/fault (ms)
TEGUS [16]	0	0.59	0	3.38
ATOM	0	0.41	0	2.14

Table 2: Comparison of ATPG Results

18 circuits, and the Time/fault column presents the average time per fault in milliseconds for these 18 circuits. Both systems demonstrated their robustness by detecting all the testable faults and proving all the redundant faults to be redundant both with and without using fault simulation. Although the execution times are reported in two different machines, the two machines has comparable performances; DEC 7000/610 AXP has a 182 MHz 64-bit DEC alpha chip and a fast memory subsystem including a 4MB second level cache, and our machine has a 200 MHz 32-bit Pentium Pro chip and a 256KB second level cache. Based on these features, it is fair to conclude that ATOM is faster than TEGUS.

5 Conclusions

In this paper, we have introduced new techniques for speeding up the deterministic test pattern generation for VLSI circuits. These techniques improve the PODEM algorithm by reducing number of backtracks with a low computational cost. This is achieved by finding more necessary signal line assignments, by detecting conflicts earlier, and by avoiding unnecessary work during test generation.

To be able to assess the effectiveness of the proposed techniques, we have developed a deterministic ATPG system for combinational circuits, called ATOM, and incorporated these techniques into the test generator. The performance results for the ISCAS85 and full scan version of the ISCAS89 benchmark circuits demonstrated the effectiveness of these techniques on the test generation performance. ATOM detected all the testable faults and proved all the redundant faults to be redundant with very small number of backtracks in a short amount of time. Since these results are achieved without using an initial random test generation phase, and similar results are obtained both with and without fault simulation, the results demonstrated the robustness of the techniques we have developed.

References

[1] M. Abramovici, J. J. Kulikowski, P. R. Menon, and D. T. Miller, "SMART and FAST: Test generation for VLSI scan-design circuits", *IEEE Design and Test of Computers*, pp 43-54, August 1986.

[2] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran", in *Proc. of the Int. Symp. on Circuits and Systems*, June 1985.

[3] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits", in *Proc. of the Int. Symp. on Circuits and Systems*, pp. 1929-1934, May 1989.

[4] H. Fujiwara and S. Toida, "The complexity of fault detection problems for combinational logic circuits", *IEEE Trans. on Computers*, pp. 555-560, June 1982.

[5] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms", *IEEE Trans. on Computers*, pp. 1137-1144, December 1983.

[6] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits", *IEEE Trans. on Computers*, pp. 21-222, March 1981.

[7] T. Kirkland and M. R. Mercer, "A topological search algorithm for ATPG", in *Proc. of the Design Automation Conf.*, pp. 502-508, June 1987.

[8] S. Kundu, L. M. Huisman, I. Nair, V. Iyengar, and L. N. Reddy, "A Small Test Generator for Large Designs",*Proc. of the Int. Test Conf.*, pp. 30-40, September 1992.

[9] W. Kunz and D. Pradhan, "Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation in Digital Circuits", *Proc. of the Int. Test Conf.*, pp. 816-825, September 1992.

[10] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability", *IEEE Trans. on Computer-Aided Design*, pp. 4-15, January 1992.

[11] T. H. Niermann, W. Cheng, and J. H. Patel, "PROOFS: A fast, memory-efficient sequential circuit fault simulator", *IEEE Trans. on Computer-Aided Design*, pp. 198-207, February 1992.

[12] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system", *IEEE Trans. on Computer-Aided Design*, pp. 126-137, January 1988.

[13] M. H. Schulz and E. Auth, "Improved deterministic test pattern generation with applications to redundancy identification", *IEEE Trans. on Computer-Aided Design*, pp. 811-816, July 1989.

[14] J. P. M. Silva and K. A. Sakallah, "Dynamic Search-Space Pruning Techniques in Path Sensitization", in *Proc. of the Design Automation Conf.*, pp. 705-711, June 1994.

[15] J. P. M. Silva and K. A. Sakallah, "Robust Search Algorithms for Test Pattern Generation", in *Proc. of the Fault-Tolerant Computing Symp.*, pp. 152-161, June 1997.

[16] P. Stephan, R. K. Brayton and A. L. Sagiovanni-Vincentelli, "Combinational Test Generation Using Satisfiability", In *IEEE Trans. on Computer-Aided Design*, pp. 1167-1176, September 1996.

[17] J. A. Waicukauski, P. A. Shupe, D. J. Giramma, and A. Matin, "ATPG for Ultra-Large Structured Designs", in *Proc. of the Int. Test Conf.*, pp. 44-51, August 1990.