19th IEEE VLSI Test Symposium, Los Angeles, CA, USA, pp. 346-351, April 29 - May 3, 2001 .

SPIRIT: A Highly Robust Combinational Test Generation Algorithm[1]

Emil Gizdarski*[2] and Hideo Fujiwara**

* Department of Computer Systems, University of Rousse, 7017 Rousse, Bulgaria
** Nara Institute of Science and Technology, Ikoma, Nara 630-0101, Japan
E-mail: egizdarski@ieee.org, fujiwara@is.aist-nara.ac.jp

Abstract

In this paper, we present a robust test generation algorithm for combinational circuits based on the Boolean satisfiability method called SPIRIT. We elaborate some well-known techniques as well as present new techniques that improve the performance and robustness of test generation algorithms. As a result, SPIRIT achieves 100% fault efficiency for a full scan version of the ITC'99 benchmark circuits in a reasonable amount of time.

1. Introduction

In recent years substantial progress has been achieved in the field of Electronic Design Automation (EDA) using the Boolean satisfiability (SAT) method. Originally motivated by the work of T.Larrabee [13] in test pattern generation (TPG), the SAT method has been applied to many other EDA applications. Test generation itself plays a key role in other processes such as logic optimization, verification, design-for-testability, and built-in self-testing where the efficiency of the combinational TPG algorithms is an important issue. Therefore, understanding and improving test generation has not only practical importance, but it might also offer insight into several other related SAT-based EDA applications. The basic parameters of the ATPG systems are *performance* and *robustness* estimated by CPU time and fault efficiency. The ultimate goal of this work is to propose efficient techniques for improving the robustness of the TPG algorithms. To do so, we study the well-known and most successful ATPG systems: (structural) PODEM[5], FAN[7], SOCRATES[20] and ATOM[9], (algebraic) Nemesis[13], TRAN[1] and TEGUS[21], and (mixed) TIP[10,22]. The most important techniques implemented in SPIRIT are described briefly below:

• 9-V[17] and 16-V[19] algebra for more precise value assignment and search space reduction • X-path check [5] as an efficient technique for early detection of inconsistency during propagation • Unique sensitization [7] and dominators [20] as efficient concepts for dynamic learning during propagation • An unjustified line [7,8] as an efficient concept for justification and an early detection of inconsistency • Static learning [20] as an efficient technique for deriving new dependencies between signals during preprocessing • Recursive learning [12] as efficient technique for dynamic learning during propagation and justification • Boolean satisfiability as a method giving an elegant formulation of the TPG problem [13] • Single cone processing [15] and single path oriented propagation [10] as efficient approaches for reducing the search space of the TPG problem • Backward justification [11] as an alternative of PODEM algorithm making decisions on only the primary inputs • A new data structure of the complete implication graph [3] as an alternative of the complete implication graph used in [10,22] • Duality of learning [4] as an efficient concept for improving justification and avoiding overspecification.

In previous work [4], we examined some not so popular approaches such as single cone processing [15], single path oriented propagation [10] and backward justification [11]. We showed that two techniques, static learning [20] and duality of learning [4], are sufficient to achieve both high performance and robustness for the ISCAS'85 and ISCAS'89 benchmark circuits. Without fault simulation, SPIRIT generated complete test sets for these benchmark circuits within 3 minutes on a 450MHz Pentium-III PC. In this paper, we propose new techniques and heuristics to increase the efficiency of these approaches.

The rest of the paper is organized as follows. In Section 2, a system overview of SPIRIT is provided. In Section 3, new techniques and heuristics are presented. Section 4 provides experimental results and Section 5 concludes the paper.

---

[1] *This work was supported by JSPS under grant P99747*

[2] *Currently visiting at Nara Institute of Science and Technology*

2. System overview

The most typical SAT-based algorithms [1,13,21] translate the TPG problem into a characteristic formula that represents both the *logical* and *structural* constraints for the possible solutions. The formula is usually written in a conjunctive normal form (CNF) where one sum is called a clause. Clauses with one, two, or more variables are called unary, binary, and k-nary clauses respectively.

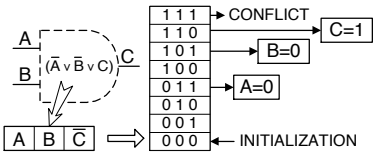

Figure 1. *The representation of local ∧-implications*

SPIRIT represents the CNF formula using a new data structure for the complete implication graph proposed in [3]. More formally, a node in the implication graph represents each variable in the CNF formula. Each binary clause (X ∨ Y) is represented by two local implications, (X=0 → Y=1) and (Y=0 → X=1). Each k-nary clause is represented by 2^k ∧-nodes, k local ∧-implications and a k-bit key dynamically calculated by the binding procedure. Figure 1 sketches the data structure of a ternary clause of a 2-input AND gate. In contrast to the complete implication graph used in [10,22], this data structure allows unified representation of both ternary and k-nary clauses as well as both local ∧-implications and derived global ∧-implications during static learning.

Since a test pattern for a fault is *an input vector* that sensitizes the fault and propagates the fault effect to a primary output, we consider that: *a test pattern is found iff both the fault under consideration and a propagation path are sensitized and all unjustified lines are justified*. If one of these conditions cannot be satisfied, the fault is proved as undetectable in respect to the current primary output. In this way, we avoid extracting the structural constraints and simplify the satisfying the logical constraints for test generation. This definition considerably increases the efficiency of the SAT-based TPG algorithms. For example, the premier SAT-based TPG algorithms [1,13,21] consider that a test pattern is found when the formula is satisfied, i.e., every clause in that formula evaluates to 1. This approach potentially increases the complexity of the TPG problem. The recent SAT-based TPG algorithms [4,6,22] also use justification by checking for an empty J-frontier instead of checking whether all the clauses in the formula are satisfied.

The phases of SPIRIT are shown in Figure 2. In contrast to the most typical SAT-based TPG algorithms [1,13,21], SPIRIT builds the implication graph and performs static learning once for the whole circuit since these are prominent and time-consuming steps. Using single path oriented propagation, we avoid extracting structural constraints [1,13,21]. Using single cone processing, we apply a "divide-and-conquer" strategy and keep the size of the TPG problem as small as possible. This approach may produce more that one test session for a redundant or even detectable fault because SPIRIT has to prove that the fault is undetectable in respect to each primary output where the fault effect can be observed. On the other hand, this approach allows more effective application of many TPG techniques.

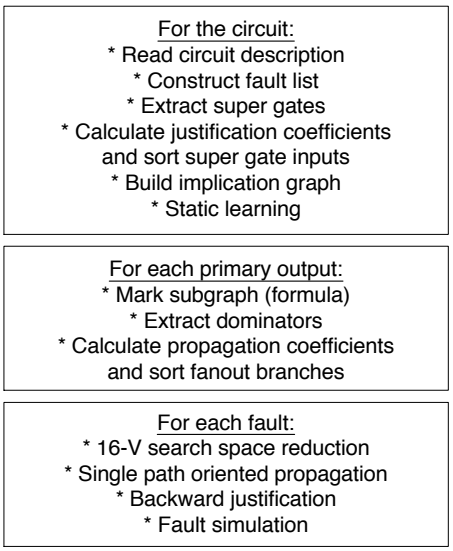

Figure 2. *SPIRIT flowchart*

3. New test generation techniques

In this section, we present techniques and heuristics for improving the robustness of the TPG algorithms.

3.1. Augmented unique sensitization

Since the propagation procedure is based on the single path oriented propagation (SPOP) approach [10], we briefly describe this approach. Beginning from the fault location forward to the primary output, SPOP sensitizes path-segment by path-segment where a *path-segment* is defined as a subpath starting at the fault location or a fanout branch and ending at a fanout stem or the primary output. For SPOP, the fanout stems are decision points and the next path-segment is selected by sorting the fanout branches using propagation coefficients during cone preprocessing (Figure 2). During propagation, an inconsistency can be easily found by deriving as many as possible implications at each level of the decision tree. To do so, we apply static learning [20], static 16-V search space reduction [4], unique sensitization [7], dominators [20], and dynamic learning based on first level recursive learning [12]. Many potential conflicts can be avoided by dynamic learning restricted to an area near to the primary

output. The propagation procedure based on this assumption has two extra steps:

Step 1: This step is performed after the fault and all unique gates are sensitized, and first level recursive learning on the unjustified lines is performed. In general, the propagation procedure checks all alternative decisions for the end of the propagation path. First, some propagation paths near to the primary output are invalidated by assigning value D and $\overline{D}$ to certain lines. Next, a convergent gate for all valid propagation paths is determined. Let us assume that A1,...,An are the alternative decisions for the end of the propagation path, then some implications can be found by first level recursive learning, i.e., as an intersection of the implications produced by these alternative decisions.

Step 2: This step is based on *restricted dominators*, i.e., the dominators in the last path segment starting from a fanout branch and ending at the primary output. If the next path segment has a restricted dominator then the propagation procedure finds some dynamic implications by first level recursive learning on the alternative decisions for sensitization of the path segment starting from the restricted dominator and ending at the primary output. More formally, the propagation procedure sensitizes this path segment by propagating both value D and value $\overline{D}$ to the primary output. Dynamic implications are derived as an intersection of the implications produced by both alternative decisions, propagate value D or value $\overline{D}$ to the primary output.

3.2. Improved X-path check

Since SPOP does not suppose justification of each sensitized path, it is possible that a propagation path is sensitized but cannot be justified. Such a path is called an *unjustifiable* path. Early identification and reduction of the unjustifiable paths is important to improve the robustness of the TPG algorithms. To achieve this, we propose a path pruning technique based on an analysis of the undetectable faults in respect to the current primary output and applied during X-path check.

Assumption 1: If a stuck-at-0 (1) fault in line X be undetectable then all paths propagating value D ($\overline{D}$) via this line cannot be sensitized and justified.

Clearly, Assumption 1 is not true, but it may accelerate the test generation. More formally, the cases where Assumption 1 is not valid are easily identified and the remaining undetectable faults are used for path pruning. Accordingly, X-path-check considers that the value D and $\overline{D}$ cannot be propagated via a line with undetectable stuck-at-0 or 1 fault respectively.

The advantage of this technique is that it reduces not only the number of backtracks during propagation but also the number of the sensitized unjustifiable paths. The disadvantage of this technique is that to be effective, the fault lists should be processed in reverse order, i.e., from the primary outputs to the primary inputs.

3.3. Improved backward justification

Justification is performed after a propagation path to the primary output is sensitized. The justification procedure is based on justification coefficients calculated during circuit preprocessing (see Figure 2). These coefficients take into account the functions and structure of the circuit and measure the relative difficulty of justification for each line. Duality of learning is another important technique used by the justification procedure. Accordingly, the justification procedure avoids justification of spare unjustified lines by removing all forward global implications and $\wedge$-implications after static learning as well as by restricting dynamic learning to the unjustified lines in the J-frontier [4]. The justification procedure also uses the following strategies (heuristics) to select the next unjustified line for justification:

S1: *Depth-first search* strategy gives a higher priority to the unjustified lines more recently included in the J-frontier.

S2: *First-difficult/First-easy* are two orthogonal strategies giving a higher priority to the unjustified lines more difficult/easier for justification according to the justification coefficients.

S3: *First-local/first-global* are two orthogonal strategies giving a higher priority to the unjustified lines located near/far from the last processed unjustified line. Clearly, the first-local strategy is oriented to finding an inconsistency at the nearest fanout stems, while the first-global strategy is oriented to finding an inconsistency at the primary inputs.

S4: *Decision tree reduction*. This strategy tries to minimize the size of the decision tree. Clearly, the justification of 2-input gates is more difficult than the justification of k-input gates where $k \geq 3$ because in the second case, many alternatives for justification exist, i.e., it is sufficient that just one of the inputs of these gates can be set to the control value. Giving a higher priority to the unjustified lines corresponding to gates having less inputs can reduce the size of the decision tree by decreasing the number of the branches in the highest levels of the decision tree.

Strategy S1 is basic for the justification procedure. Strategy S2 is used for an initial ordering of the J-frontier after a successful propagation, while strategies S3 and S4 are used for a reordering of the J-frontier during justification. For example, strategy S1 is applied by a stack-type J-frontier using the first-in-first-out model. To

increase effectiveness of strategy S4, a super gate extraction is performed during circuit preprocessing. By sorting the inputs of each super gate using the justification coefficients, we suppose that each unjustified line can be easily justified by assigning a certain value to the first unspecified input of the corresponding super gate. Here, we consider that super gates are extracted by direct backward justification [23]. Figure 3, is an example for super gate extraction. To determine the super gate of gate G in Figure 3(a), we set line G to 0 and performed all direct backward implications. As a result, lines A and B are set to 0, and lines C and D are set to 1. The resultant super gate is shown in Figure 3(b). Let us suppose that lines A, B, C and D be very easy, difficult, easy and difficult for justification, respectively. In the original circuit, the decisions for justification of line G are sorted according to the justification coefficients and will be performed top-down. Without super gate extraction, the second decision for justification of line G will be B=1 declared as difficult for justification. After super gate extraction, the second decision for justification of line G will be C=0 declared as easy for justification. Therefore, the super gate extraction may improve justification of line G if the first decision, A=1, is inconsistent.

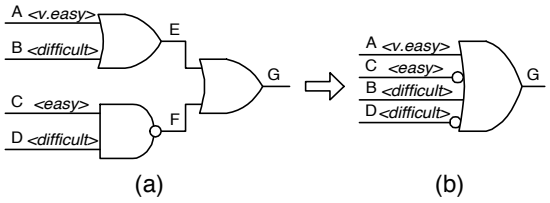

Figure 3. *Super gate extraction*

3.4. Avoiding critical area

This heuristic was inspired by the work in [8,14,18]. Clearly, the justification is the critical phase of test generation. For example in [8], the proof of NP-completeness of the TPG problem is based on a polynomial transformation of the justification into the 3CNF-SAT problem. Hereafter, we suppose a similar transformation, and represent the justification by a dynamically updated formula (J-formula). Clearly, the J-formula includes a part of the characteristic formula corresponding to a subcircuit involved in the justification process because justifying and satisfying the CNF formula are not equivalent. In [14], T.Larrabee showed that for each clause to variable ratio bigger than 4.2, the percentage of satisfiable randomly generated 3CNF formulas decreases as a function of the number of clauses. Also, around this ration the random 3CNF formulas need much more CPU time to be satisfied or to be proved as unsatisfiable. We call this area *critical*. In [18], a relation between cut-width propertys of a circuit and the worst case complexity for test generation is discussed. Let us associate the unjustified lines and the primary inputs involved in the justification process with the clauses and the variables of the J-formula. Clearly, each value assignment during the search process changes the clause to variable ratio and the cut-width properties of the J-formula. Until this point, we used these characteristics informally by choosing SPOP. Let us compare the two alternative approaches for justification: (1) justification of the whole propagation path (the SPOP approach) and (2) justification of each path segment in respect to the critical instances (hard to prove redundant faults). We may consider the first approach as more effective because: (1) the application of the orthogonal strategies S2 and S3, presented in Section 3.3, is restricted for the second approach; and (2) the J-formula can fall into a critical area somewhere between the fault location and the primary output. In this case, the second approach for justification needs more time to prove the J-formulas as unjustifiable within the critical area. In [4], we experimentally showed that many redundant faults can be identified during the propagation phase, i.e., without justification. However, SPOP may sensitize one or more propagation paths for redundant faults that are difficult to be proved as unjustifiable because the J-formula falls into a critical area during justification. To avoid the critical area in these cases, the justification procedure selects up to 5 variables and proves the sensitized propagation path as unjustifiable for all possible value assignments of the selected variables. To improve the cut-width properties, the justification procedure selects variables corresponding to the fanout stems having a maximum number of fanout branches. As a result, many or all value assignments of the selected variables are inconsistent. The remaining instances need fewer backtracks because of the improved cut-width properties of the J-formula.

4. Experimental results

We implemented the techniques presented in Section 3 in the ATPG system SPIRIT[4], and ran experiments on a full scan version of the ITC'99[2] benchmark circuits on a 450 MHz Pentium-III PC. Table 1 provides the basic characteristics of the ITC'99 benchmark circuits. Columns 2 to 7 give the number of gates (excluding INV and EQU gates), primary inputs, primary outputs, lines, detectable and redundant faults. Columns 8 and 9 give the average and maximum cone size estimated by the number of variables (#primary inputs + #gates). Columns 10 and 11 give the average and maximum size of super gates.

SPIRIT was run in two passes. In the first pass, the maximum number of sensitized unjustifiable paths during propagation and backtracks during justification was set at 3. Likewise, the maximum number of value assignments was set at 10000. In the second pass, all these parameters

were set 100 times higher and the heuristic for avoiding the critical area was applied to improve the robustness of SPIRIT. The experimental results are provided in Table 2. Columns 2-7 give the number of faults, test patterns, aborted faults as well as the preprocessing and test generation time (TPG), fault simulation time (FS) and total time in seconds. The next columns give the experimental results of an available commercial ATPG system. Two experiments, with a time limit per fault of 10 and 50 seconds, were run on Sun UltraSparc 60/2360. In the first experiment, the commercial ATPG system was faster than SPIRIT. The main reason was that SPIRIT used a compiled fault simulator based on the single-pattern parallel-fault propagation approach that is ineffective for large circuits. In the second experiment, the CPU time of the commercial ATPG system was much longer than that of SPIRIT but still too many aborted faults remained, 3268 (0.86%). In both experiments, the test sets of the commercial ATPG system were smaller than those of SPIRIT, 7165 and 13797 test patterns respectively. However, we should take into account that these test sets are incomplete, i.e., all aborted faults are potentially detectable.

In this work, we examined many other TPG techniques, but they were ineffective with this benchmark set. Some of these techniques were dynamic head lines [7], dependency-directed backtracking [16], and deriving and performing global $\wedge$-implications [3]. Actually, SPIRIT achieved the best performance without static learning. The reason is that a high dependency between signals in the ITC'99 benchmark circuits exists and too many global implications and $\wedge$-implications were derived during static learning. In general, static learning increases the robustness of SPIRIT but decreases the performance in some cases. Table 3 presents an impact of the proposed techniques on the efficiency of SPIRIT. This experiment showed that the application of the SPOP approach without both static and dynamic learning might be quite time-consuming (rows 3 and 4 in Table 3). Also, the heuristic for avoiding the critical area is essential to achieve 100% fault efficiency.

To assess the negative impact of the implemented techniques, we ran SPIRIT without simulation for the ISCAS'85 and ISCAS'89 benchmark circuits (the same experiment as in [4]). The total time was 298 sec., i.e., the performance of SPIRIT was degraded by 73.2 percents.

5. Conclusions

In the previous work [4], we examined some not so popular approaches such as single cone processing, single path oriented propagation, and backward justification, and showed that they are efficient. In this paper, we proposed efficient techniques for these approaches. In fact, some of the proposed techniques elaborate the well-know techniques, while other techniques present new ideas for improving the robustness of the TPG algorithms. As a result, SPIRIT generated complete test sets for the ITC'99 benchmark circuits in a reasonable amount of time. In this way, SPIRIT is the first reported ATPG system able to achieve 100% fault efficiency for the ITC'99 benchmark circuits.

Acknowledgments

The authors would like to thank Dr.Satoshi Ohtake for his valuable help in preprocessing a description of the ITC'99 benchmark circuits as well as for obtaining the experimental results of the commercial ATPG system.

References:

[1] S.Chakradhar, V.D.Agrawal and S.Rothweiler, "A Transitive Closure Algorithm for Test Generation," IEEE Trans. on CAD, vol.12, No.7, July 1993, pp.1015-1028.

[2] S.Davidson, "ITC'99 Benchmark Circuits – Preliminary results," Proc. of International Test Conference, 1999, pp.1125. (Gate-level description*: http://www.cad.polito.it)*

[3] E.Gizdarski and H.Fujiwara, "A New Data Structure for Complete Implication Graph with Application for Static Learning," Technical report NAIST-IS-TR2000-001, January 2000, pp.18.

[4] E.Gizdarski and H.Fujiwara, "SPIRIT: Satisfiability Problem Implementation for Redundancy Identification and Test Generation," Proc. of Asian Test Symposium, 2000, pp.171-178.

[5] P.Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. on Computers, vol. C-30, No.3, March 1981, pp.215-222.

[6] L.G.Silva, L.M.Silveira and J.Marques-Silva, "Algorithms for Solving Boolean Satisfiability in Combinational Circuits," Proc. of DATE-Conference, 1999, pp.526-530.

[7] H.Fujiwara and T.Shimono, "On the Acceleration of Test Generation Algorithms," IEEE Trans. on Computers, vol. C-32, No.12, Dec.1983, pp.1137-1144.

[8] H.Fujiwara and S.Toida, "The Complexity of Fault Detection for Combinational Logic Circuits," IEEE Trans. on Computers, vol. C-31, No.6, June 1982, pp.555-560.

[9] I.Hamzaoglu and J.H.Patel, "New Techniques for Deterministic Test Pattern Generation," Journal of Electronic Testing: Theory and Applications, vol.-15, No.1/2, 1999, pp.63-73.

[10] M.Henftling, H.Wittmann and K.Antreich, "A Single-Path-Oriented Fault-Effect Propagation in Digital Circuits Considering Multiple-Path Sensitization," Proc. of International Conference on Computer-Aided Design, 1995, pp.304-309.

[11] S.Kundu, L.M.Huisman, I.Nair, V.Iyengar and L.Reddy "A Small Test Generation for Large Designs," Proc. of International Test Conference, 1992, pp.30-40.

[12] W.Kunz and D.K.Pradhan, "Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems – Test, Verification, and Optimization," IEEE Trans. on CAD, vol.13, No.9, Sept.1994, pp.1143-1158.

[13] T.Larrabee, "Test Pattern Generation Using Boolean Satisfiability," IEEE Trans. on CAD, vol.11, No.1, Jan.1992, pp.4-15.
[14] T.Larrabee and Y.Tsuji, "Evidence for a Satisfiability Threshold for Random 3CNF Formulas," Proc. of AAAI Symp. on AI NP-Hard Problems, 1992.
[15] U.Mahlstedt, T.Gruning, C.Ozcan and W.Daehn, "CONTEST: A Fast ATPG Tool for Very Large Combinational Circuits," Proc. of International Conference on Computer-Aided Design, 1990, pp.222-225.
[16] J.Marques-Silva and K.A.Sakallah, "Dynamic Search Pruning Techniques in Path Sensitization," Proc. of Design Automation Conference, 1994, pp.705-711.
[17] P.Muth, "A Nine-Value Logic Model for Test Generation," IEEE Trans. on Computers, vol.C-25, No.6, June 1976, pp.630-636.
[18] M.Prasad, P.Chong and K.Keutzer, "Why is ATPG easy?," Proc. of Design Automation Conference, 1999, pp.22-28.
[19] J.Rajski and H.Cox, "A Method to Calculate Necessary Assignments in Algorithmic Test Generation," Proc. of International Test Conference, 1990, pp.25-34.
[20] M.Schulz, E.Trischler and T.Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," IEEE Trans. on CAD, vol.7, No.1, January 1988, pp.126-137.
[21] P.Stephan, R.K.Brayton and A.L.Sangiovanni-Vincentelli, "Combinational Test Generation using Satisfiability," IEEE Trans. on CAD, vol.15, No.9, Sept.1996, pp.1167-1176.
[22] P.Tafertshofer and A.Ganz, "SAT Based ATPG Using Fast Justification and Propagation in the Implication Graph," Proc. of International Conference on Computer-Aided Design, 1999, pp.139-146.
[23] K.Tsai, R.Thompson, J.Rajski and M.Marek-Sadowska, "STAR-ATPG: A High Speed Test Pattern Generator for Large Scan Designs," Proc. of International Test Conference, 1999, pp.1021-1030.

Table 1: *Characteristics of the ITC'99 benchmark circuits*

Circuit	#Gates	#Inputs	#Outputs	#Lines	#Faults		Cone size		Super gate inputs	
					Detectable	Redundant	Average	Max	Average	Max
1	2	3	4	5	6	7	8	9	10	11
B14s	4124	277	299	11346	12537	274	486	2519	4.01	38
B15s	7844	485	519	21285	23020	508	1166	4046	8.02	115
B17s	21556	1452	1512	58741	64108	1356	1207	4022	8.48	115
B18s	61636	3357	3343	167559	185205	3333	1483	8120	6.87	115
B20s	8189	522	512	22533	24852	486	688	3071	4.20	38
B21s	8544	522	512	23541	26084	496	753	3070	4.02	38
B22s	13162	767	757	35950	39488	776	739	3083	3.97	39
Total:	**125055**	**7382**	**7454**	**340955**	**375304**	**7229**	**1185**	**8120**	**6.45**	**115**

Table 2: *Experimental results for the ITC'99 benchmark circuits*

Circuit	SPIRIT						Commercial TPG system				
	#Faults	#Test patterns	Aborted faults	Time, s			#Faults	Limit 10 sec.		Limit 50 sec.	
				TPG	FS	Total		Aborted	Time, s	Aborted	Time, s
1	2	3	4	5	6	7	8	9	10	11	12
B14s	12811	939	0	44	8	52	12643	261	59	102	8495
B15s	23528	1179	0	516	38	554	23316	929	282	253	31735
B17s	65464	2803	0	1262	348	1610	65324	2682	1039	740	91354
B18s	188538	7305	0	2782	3214	5996	188458	6093	3439	1546	220482
B20s	25338	1590	0	93	29	122	25274	437	121	196	19074
B21s	26580	1505	0	112	29	141	26516	393	110	161	16118
B22s	40264	2221	0	198	69	267	40200	628	212	270	25348
Total:	**382523**	**17542**	**0**	**5007**	**3735**	**8742**	**381731**	**11423**	**5262**	**3268**	**412606**

Table 3: *Impact of each technique on the efficiency of SPIRIT*

	SPIRIT	Pass 1		Pass 2		#Test patterns	Total time, s
		Aborted	Time, s	Aborted	Time, s		
1	All techniques without static learning	**93**	**8438**	**0**	**304**	**17542**	**8742**
2	All techniques with static learning	46	10358	0	278	-15	+1894
3	Augmented unique sensitization without step 1	120	10968	12	45301	+10	+47527
4	Augmented unique sensitization without step 2	108	9322	9	30191	-1	+30771
5	X-path check without path pruning	136	9117	0	343	-1557	+718
6	Backward justification without strategies S1,..,S4	132	9162	0	1639	-123	+2059
7	Second pass without avoiding critical area	93	8438	22	2734	-1	+2430