SPIRIT: A High 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 an efficient and 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 some new techniques that improve robustness of test generation algorithms. As a result, SPIRIT achieves 100% fault efficiency for 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 in test pattern generation (TPG), SAT method has been applied to many others EDA applications. The test generation itself plays a key role in other processes as logic optimization, verification, design for testability and logic built-in self-test where a high robustness of combinational TPG algorithms is an important issue. Therefore, understanding and improving TPG algorithms has no only practical importance but it might offer insight into several others SAT-based EDA applications. This problem motivates us to work on the well-studied topic - test generation for combinational circuits. The goal is to improve performance and robustness of TPG algorithms. To do so, we study the well-known and most successful ATPG systems: (structural) PODEM[5], FAN[7], SOCRATES[20], [26], ATOM[9] and STAR-ATPG[25] (algebraic) Nemesis[13], TRAN[1], TEGUS[21] and CGRASP[6], and (mixed) TIP[10,23]. The most important techniques implemented in SPIRIT are briefly described below:

- 9-V[17] and 16-V[19] algebra allows 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] as an efficient technique for learning during propagation.
- An unjustified line [7,8] as an efficient concept for justification and an early detection of inconsistency.
- Static learning [20] as a preprocessing phase that performs iterative both value assignments (0 and 1) through the variables and finds new dependencies (global implications) between signals.
- Recursive learning [12] is the first complete learning procedure able to find all implications (necessary assignments) at the current level of the decision tree by recursive AND-OR search on the unjustified lines.
- The Boolean satisfiability method gives an elegant formulation of TPG problem [13]. Until that time, the decision points of search were limited to the primary inputs [5], head lines [7] or implying nodes [24]. Actually, the Boolean satisfiability method translates TPG problem to a formula that includes all signals of the circuit. This increases the degree of freedom for TPG algorithms.
- Single path oriented propagation [10] simplifies propagation (for structural TPG algorithms) and reduces the need for extracting structural constraints (for SAT-based TPG algorithms).
- Single cone processing [15] reduces the size of problem and allows more effective application of many test generation techniques.
- Backward justification [11] as an alternative solution of PODEM algorithm making decisions on the primary inputs only.
- A new data structure of the complete implication graph presented in [3] as an alternative of the complete implication graph used in [10,23].
- Duality of learning [4] as an efficient concept for improving justification and avoiding overspecification.

The rest of the paper is organized as follows. In Sections 2 a systems overview is provided. In Section 3, techniques and heuristics to improve performance and robustness of TPG algorithms are presented. Section 4 provides experimental results and Section 5 concludes the paper.

[1] *This work was supported in part 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 a problem into a formula that represents both the *logical* and *structural* constraints for the possible solutions. The characteristic formula is usually written in Conjunctive Normal Form (CNF) where one sum is called a clause. Clauses with one, two, three or more variables are called unary, binary and k-nary clauses respectively.

SPIRIT represents a CNF formula using a new data structure of the complete implication graph presented in [3]. More formally, a node in the implication graph represents each variable in CNF formula. Each binary clause ($X \vee Y$) is represented by two local implications, ($X=0 \rightarrow Y=1$) and ($Y=0 \rightarrow X=1$). Each k-nary clause is represented by 2^k $\wedge$-nodes, k local $\wedge$-implications and k-bit key dynamically calculated by the binding procedure. Figure 1 sketches a data structure of ternary clause corresponding to a two-input AND gate. In contrast to the complete implication graph used in [10,23], this data structure allows a unified representation of both ternary and k-nary clause as well as both local $\wedge$-implications and derived global $\wedge$-implications during static learning.

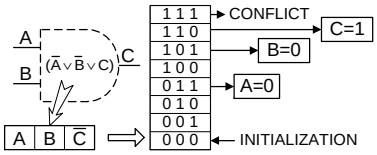

Figure 1. *The representation of local $\wedge$-implications*

Since a test pattern for a fault is *an input vector* that sensitizes the fault and propagates the fault effect to a primary output, hereafter we will consider that:

- *a test pattern is found iff both the fault under consideration and propagation path are sensitized and all unjustified lines are justified*

If one of these conditions cannot be satisfied, the fault under consideration is proved as an undetectable in respect to the current primary output. In this way, we eliminate the deriving of structural constrains and simplify the satisfying of logical constrains for test generation. In fact, this definition considerably increases efficiency of SAT-based algorithms. The premier SAT-based TPG algorithms consider that test pattern is found when formula is satisfied, i.e. all clauses evaluate to 1. This approach potentially increases the complexity of test generation problem. The recent SAT-based TPG algorithms [4,6,23] also use justification by checking for empty J-frontier instead of checking weather all clauses are satisfied.

The phases for the proposed ATPG system are shown in Figure 2. In contrast to the most typical SAT-based TPG algorithms [1,13,21], SPIRIT builds the complete implication graph (extracts logical constrains) and performs static learning once for the whole circuit since extracting of the formula and static learning are prominent and time-consuming steps. Using single path oriented propagation, we avoid the extracting of structural constraints (D-chain conditions) [1,13,21]. Using a single cone processing, we apply "Divide and Conquer" strategy and keep the size of TPG problem as small as possible. This approach may produce more that one test session for a redundant or 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 all implemented techniques.

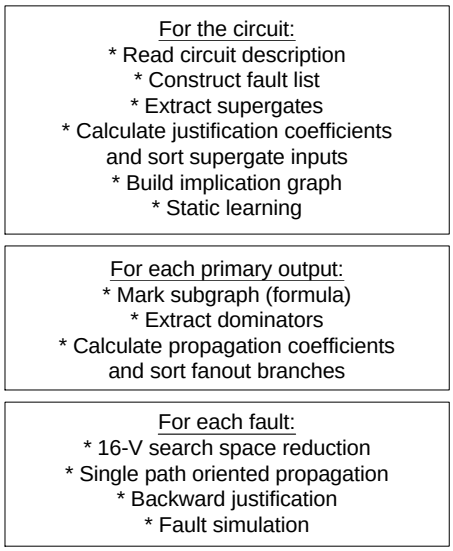

Figure 2. *SPIRIT flowchart*

3. New test generation techniques

In this section we present some techniques that improve performance and robustness of TPG algorithms. The first two techniques are oriented to improve propagation while the second two techniques are oriented to improve justification.

3.1. Augmented unique sensitization

Since our propagation procedure is based on the single path oriented propagation (SPOP) approach introduced in [10], we will briefly describe this approach. Beginning from the fault location forward to the primary output, the SPOP approach 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 the SPOP approach, the fanout stems are decision points and next path-segment is selected by initial sorting of the fanout branches using propagation coefficients calculated during cone

preprocessing (see Figure 2). During propagation, an inconsistency can be easily found by deriving as many as possible implications (necessary value assignments) at each level of decision tree. To do so, we apply static learning [20], static 16-V search space reduction [4], and unique sensitization [7], structural dominators [20] and dynamic learning based on first level recursive learning [12]. Many potential conflicts producing a deep backtracking during propagation can be avoided by dynamic learning restricted to an area near to the primary output. The improved propagation procedure based on this assumption has two extra steps:

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

Step 2: This step is based on *restricted dominators*, i.e. the structural 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 derives some dynamic implications by first level recursive learning on the alternative decisions for sensitization of 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 $\overline{D}$ from the restricted dominator to the primary output. Dynamic implications, i.e. a set of necessary value assignments for sensitization of this path segment, is derived as an intersection of implications produced by the alternative decisions, propagate value D or propagate value $\overline{D}$ to the primary output.

3.2. Improved X-path check

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

Conjunction 1: Let a stuck-at-0 or stuck-at-1 fault in line X be undetectable in respect to the current primary output then all paths propagating value D or $\overline{D}$ via this line cannot be sensitized or justified.

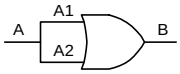

Figure 3*. Contra example of Conjunction 1*

Figure 3 depicts a contra example where Conjunction 1 is not valid. In this case, stuck-at-0 faults on line A1 and line A2 are undetectable but paths propagating value D via line A1 or line A2 can be sensitized and justified. Clearly, Conjunction 1 is not valid in this case. Obviously, such cases may exist but they can be easily identified by simple structural analysis on the undetectable faults. This analysis identifies and ignores the undetectable faults that invalidate Conjunction 1. The remaining undetectable faults are used for path pruning according to Conjunction 1. More formally, X-path check considers that the value D and $\overline{D}$ cannot be propagated via a line with undetectable stuck-at-0 or stuck-at-1 fault.

The main advantage of this technique is that it reduces not only the number of backtracks during propagation but also the number of 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 output to the primary inputs.

3.3. Improved backward justification

The justification procedure is based on justification coefficients calculated during circuit preprocessing (see Figure 2). These coefficients take into account the logical function and structure of the circuit and measure the relative difficulty for justification of each line. Duality of learning is other important concept utilized in the justification procedure. Accordingly, the justification procedure avoids justification of some 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 J-frontier [4]. The justification procedure also uses the following strategies (heuristics) to select next unjustified line for justification:

S1: *Depth-first search* strategy is a basic strategy of the backward justification procedure. This strategy gives a higher priority to the unjustified lines included in the J-frontier more recently.

S2: *First-difficult/First-easy* are two orthogonal strategies that give priority to unjustified lines more

difficult/easier for justification according to justification coefficients. This strategy is used for initial ordering of the J-frontier.

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

S4: *Decision tree reduction*. This strategy minimizes the size of decision tree. Clearly, the unjustified lines corresponding to two-input gates are more difficult for justification than the unjustified lines corresponding to gates having many inputs because the second case supposes many decisions for justification, i.e. it is sufficient that just one of the inputs can be set to a certain value. On the other hand, giving a higher priority to the unjustified lines corresponding to the gates having less inputs reduces the size of the decision tree by decreasing the number of branches in the highest levels of the decision tree. This strategy also may reduce the size of the decision tree by decreasing the number of levels.

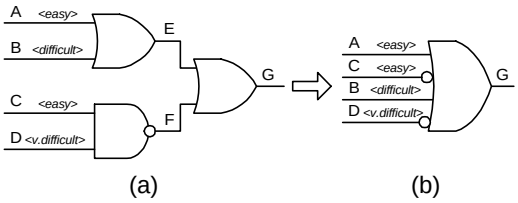

Figure 4. *Supergate extraction*

To increase efficiency and precision of the presented strategies, a supergate extraction is performed. By sorting the inputs of each supergate 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 supergate. Here, we consider a logical supergates extraction, i.e. a supergate of a gate is derived by direct backward justification of this gate [25]. In contras, structural supergates are derived by reconvergence-based analysis [22]. Figure 4, depicts an example of supergate extraction. To determine a supergate of gate G shown in Figure 4(a), we set line G to 0 and perform all backward implications. As a result, lines A and B are set to 0, and lines C and D are set to 1. The resultant supegate is shown in Figure 4(b). Let lines A, B, C and D be easy, difficult, easy and very difficult for justification, respectively. In the original circuit, the decisions for justification of line G are ordered according to justification coefficients and will be performed top-down. Without supergate extraction, the second decision for justification of line G will be B=1 estimated as difficult. While after supergate extraction, the second decision for justification of line G will be C=0 estimated as easy. Therefore supergate extraction might improve justification of line G if the first decision for justification, A=1, is inconsistent.

3.5. Avoiding critical area

This technique was inspirited by the work in [8,14,18]. Clearly, the justification is the critical phase of test generation because in the literature there is no proof that the propagation is NP-complete. For example in [8], the proof of NP-completeness for test generation problem is based on polynomial transformation of justification into the 3CNF SAT problem. Hereafter, we suppose a similar transformation, and associate the justification with a dynamically undated CNF formula. In [14], T.Larrabee et.al. used randomly generated 3CNF formulas and showed that for each clause to variable ratio bigger than 4.2, the percentage of satisfiable formulas decrease as a function of number of clauses. Also around this ration the random 3CNF formulas need much more time to be satisfied or prove as unsatifiable. Hereafter, we call this *critical area*. In [18] a relation between the cut-width property of a circuit and the worst case complexity for test generation is studied. Clearly, each value assignment during search process changes the ratio between variable and clauses as well as cut-width properties of the formula. Until this point, we used these characteristics by choosing the SPOP approach. Let us compare the effectiveness of 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 critical instances (hard to prove redundant faults). We may consider that the first approach is more effective for justification. The reasons are: (1) the application of orthogonal strategies (S2 and S3) presented in section 3.3 is restricted for the second approach and (2) the 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 formulas as unsatifiable within the critical area. In [4], we experimentally show that many redundant faults can be identified during propagation phase, i.e., without justification. However, some redundant faults may produce unjustifiable propagation paths that are difficult to be proved because the formula falls into critical area near to the primary output. To avoid critical area, the justification procedure selects up to 5 variables and proves the formula as unjustifiable for all possible value assignments of these variables. To improve cut-width properties of the formula, the justification procedure selects variables corresponding to fanout stems having a maximum number of fanout branches. In this way, many or all value assignments for the selected variables are inconsistent. The remaining instances are easily proved unsatisfiable during dynamic learning (first

level recursive learning on unjustified lines in the J-frontier) or need much less backtracks because of the improved cut-width properties of the formula.

4. Experimental results

We implemented the presented TPG algorithm in ATPG system SPIRIT[4] and ran experiments on full scan version of the ITC'99[2] benchmark circuits on a 450 MHz Pentium-III PC. The basic characteristics of the ITC'99 benchmark circuits are provided in Table 1. 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 number of supergate inputs.

SPIRIT ran in two passes. For the first pass, the maximum number of sensitized unjustifiable paths during propagation and backtracks during justification was set at 3, and the maximum number of performed value assignments was set at 10000. For the second pass, all these parameters were set 100 times higher as well as the technique for avoiding critical area was applied to improve robustness of our TPG algorithm. The experimental results are provided in Table 2. Columns 2-7 give the number of faults, test patterns, aborted faults as well as preprocessing and test generation time (TPG), fault simulation time (FS) and total time in seconds. The next columns give the experimental results for test generation of an available commercial ATPG system. Two experiments, with a time limit 10 and 50 seconds per fault, were ran on Sun UltraSparc 60/2360. In the first experiment, the performance of commercial ATPG systems was higher than this of SPIRIT. The main reasons was that SPIRIT used a compiled fault simulator based on the single-pattern parallel fault propagation approach that is inefficient for large circuits. In second experiment, the total time of the commercial ATPG system was much longer than this of SPIRIT but still too many aborted faults left, 3268 (0.86%).

In this work, we experimentally examined many other TPG techniques but found that they were ineffective with this benchmark set. Some of these techniques were dynamic headlines, dependency-directed backtracking [16] and deriving and performing global $\wedge$-implications [3]. Actually, we achieved the best performance without static learning. The reason was that a high dependency between signals in ITC'99 benchmark circuits exists and too many global implications and global $\wedge$-implications were derived. In general, this increased robustness of our TPG algorithm but decreased its performance.

5. Conclusions

In the previous work [4], we examined some not popular approaches like single cone processing, single path oriented propagation, backward justification and showed that they are effective. Without fault simulation and some basic techniques like X-path check [5] and unique sensitization [7], we was able to achieved 100% fault efficiency for the ISCAS'85 and ISCAS'89 benchmark circuits in short amount of time (within 3 minutes on a 450 MHz Pentium-III PC). In this paper, we presented some new techniques to improve performance and robustness of TPG algorithms. As a result, SPIRIT was able to generate complete test set and prove all redundant faults in the ITC'99 benchmark circuits in a reasonable amount of time. In this way, SPIRIT is the first reported TPG system that achieves 100% fault efficiency for the ITC'99 benchmark circuits.

Acknowledgments

Authors would like to thanks Dr.Satoshi Ohtake for his valuable help in processing description of the ITC'99 benchmark circuits and getting experimental results by commercial software.

References:

[1] S.Chakradhar, V.D.Agarval 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. IEEE ITC, 1999, pp.1125.
(Gate-level descriptions from*: 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-TR2000001, January 2000, pp.18. *(http://isw3.aist-nara.ac.jp/IS/TechReport2/report/2000001.ps)*

[4] E.Gizdarski and H.Fujiwara, "SPIRIT: Satisfiability Problem Implementation for Redundancy Identification and Test Generation," Proc. IEEE ATS, 2000, will appear.

[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.Guerra e Silva, L.M.Silveira and J.Marques-Silva, "Algorithms for Solving Boolean Satisfiability in Combinational Circuits," Proc. IEEE DATE, 1999, pp.526-530.

[7] H.Fujiwara and T.Shimono, "On 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. IEEE/ACM ICCAD, 1995, pp.304-309.
[11] S.Kundu, L.M.Huisman, I.Nair, V.Iyengar and L.N.Ready "A Small Test Generation for Large Designs," Proc. IEEE ITC, 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. 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. IEEE/ACM ICCAD, 1990, pp.222-225.
[16] J.Marques-Silva and K.A.Sakallah, "Dynamic Search Pruning Techniques in Path Sensitization," Proc. IEEE/ACM DAC, 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. IEEE/ACM DAC, 1999, pp.22-28.
[19] J.Rajski and H.Cox, "A Method to Calculate Necessary Assigmnets in Algorithmic Test Generation", Proc. IEEE ITC, 1990, pp.25-34.
[20] M.Schulz, E.Trischler and T.Sarfert, "SOCRATES: A High Efficient Automatic Test Pattern Generation System," IEEE Trans. on CAD, vol.7, No.1., Jan. 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] S.C.Seth and V.D.Agrawal, "A New Model for Computation of Probabilistic Testability in Combinational Circuits," Integration, the VLSI journal, vol.-7, July 1989, pp.49-75.
[23] P.Tafertshofer and A.Ganz, "SAT Based ATPG Using Fast Justification and Propagation in the Implication Graph," Proc. IEEE/ACM ICCAD, 1999, pp.139-146.
[24] M.Teramoto, "A Method for Reducing the Search Space in Test Pattern Generation," Proc. IEEE ITC, 1993, pp.429-435.
[25] K.-H.Tsai, R.Tompson, J.Rajski and M.Marek-Sadowska, "STAR-ATPG: A High Speed Test Pattern Generator for Large Scan Designs," Proc. IEEE ITC, 1999, pp.1021-1030.
[26] J.A.Waicukauski, P.A.Shupe, D.J.Giramma and A.Martin, "ATPG for Ultra-Large Structural Designs," Proc. IEEE ITC, 1990, pp.44-51.

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

Circuit	#gates	#inputs	#outputs	#lines	#faults		Cone size		Supergate 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	42	558	23316	929	282	253	31735
B17s	65464	2803	0	1262	465	1727	65324	2682	1039	740	91354
B18s	188538	7305	0	2782	4721	7503	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	71	269	40200	628	212	270	25348
Total:	**382523**	**17542**	**0**	**5007**	**5365**	**10372**	**381731**	**11423**	**5262**	**3268**	**412606**