S P I R I T: Satisfiability Problem Implementation for Redundancy Identification and Test Generation

Emil Gizdarski[1]
Department of Computer Systems
University of Rousse
7017 Rousse, Bulgaria
egizdarski@ieee.org

Hideo Fujiwara
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0101, Japan
fujiwara@is.aist-nara.ac.jp

Abstract. In this paper an efficient test pattern generation (TPG) algorithm for combinational circuits is presented. It is based on the Boolean satisfiability (SAT) and single path oriented propagation methods as well as static and dynamic learning techniques. We implement a new data structure for the complete implication graph reported in [4] and propose an improved backward justification procedure and some useful techniques for speeding up satisfying a characteristic formula. The resultant ATPG system, called SPIRIT, combines the flexibility of SAT-based TPG algorithms with the efficiency of structural TPG algorithms. Experimental results demonstrate the efficiency and robustness of the proposed TPG algorithm. Without fault simulation, SPIRIT is able to prove all redundant faults and generates complete test sets for the ISCAS'85 and ISCAS'89 benchmark circuits within 3 minutes on a 450MHz Pentium-III PC.

1. Introduction

In recent years substantial progress has been achieved in the fields of design automation, verification and test generation for integrated circuits using the Boolean satisfiability method. Although there are many effective ATPG algorithms, their performance is decreasing because of the increase in size and complexity of integrated circuits. This problem motivates us to work on the well-studied topic of test generation for combinational circuits. The goal is to increase efficiency and robustness of TPG algorithms. To do so, we study the well-known and most successful ATPG systems: (structural) PODEM[6], FAN[8], SOCRATES[15], ATOM[7] (algebraic) Nemesis [11], TRAN[3] and TEGUS[16], and (mixed) TIP[9]. Many techniques to increase efficiency of TPG algorithms have been proposed in the past three decades. The most important of techniques utilized in SPIRIT are briefly described below:

- An unjustified line [8] is an efficient concept for justification and an early detection of inconsistency.
- 9-V[13] and 16-V[14] algebra are used for more precise value assignment and search space reduction.
- Static learning [15] is introduced as a preprocessing phase that performs iterative state assignments through variables and finds new dependencies (global implications) between signals.
- Recursive learning [10] is the first complete learning procedure able to find all indirect implications at the current level of the decision tree (branch and bound search) by recursive AND-OR search on the unjustified lines.

[1] Currently visiting at Nara Institute of Science and Technology

- The Boolean satisfiability method [11] gives an elegant formulation of TPG problem. Until this time, the decision points of branch and bound search were limited to only the primary inputs [6], the head lines [8] or the implying nodes [12]. Actually, the SAT method transforms the search process from the primary inputs to satisfying a characteristic formula that includes all signals of the circuit. This increases the degree of freedom for TPG algorithms.

- Single path oriented propagation method [9] reduces the need for an analysis of the possible paths for fault effect propagation performed by X-path check (for structural TPG algorithms) and by extracting structural constraints (for SAT-based TPG algorithms).

The basic methods utilized in our TPG algorithm are static learning, a single cone (primary output circuit) processing and single path oriented fault effect propagation. We implement a new data structure for the complete implication graph reported in [4] that allows easily deriving and performing of global $\wedge$-implications as well as propose some useful auxiliary techniques like static 16-V search space reduction and propagation path pruning. We present a duality of learning phenomenon and propose an improved backward justification procedure as an alternative of PODEM algorithm. The resultant ATPG system SPIRIT (Satisfiability Problem Implementation for Redundancy Identification and Test generation) combines simplicity, efficiency and robustness.

The rest of the paper is organized as follows. In Section 2, a systems overview is presented. In Section 3, SAT-based static learning is provided. In Section 4, some useful techniques to improve the single path oriented propagation method are proposed. In Section 5, a basic backward justification procedure is described and duality of learning phenomenon is presented. In Section 6, an improved backward justification procedure is presented. Section 7 provides experimental results and Section 8 concludes the paper.

2. System overview

The phases for the proposed ATPG system are shown in Figure 1. Phase 1 (circuit preprocessing) reads a circuit description, constructs a collapsed fault list, calculates justification coefficients, builds an implication graph and performs static learning. Phase 2 (cone preprocessing) marks a characteristic formula corresponding to a single cone (primary output) and calculates fault propagation coefficients. Phase 3 (test session) performs static 16-V search space reduction and satisfies a formula.

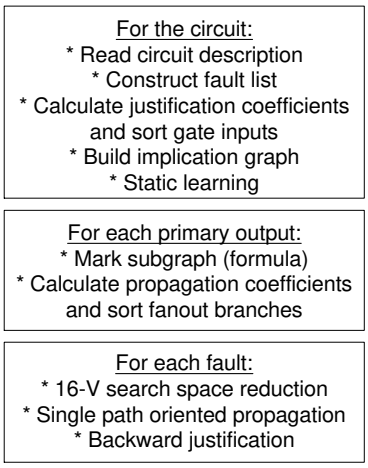


Figure 1. SPIRIT flowchart

In contrast to the most typical SAT-based TPG algorithms like Nemesis, TRAN and TEGUS, SPIRIT extracts the characteristic formula, builds the implication graph and performs static learning once for the whole circuit since static learning is prominent and time consuming step. Using the single path oriented propagation method [9], we avoid the extracting structural constraints (D-chain conditions) [3,11,16] or performing X-path check analysis [6,7,8,15]. This approach reduces considerably the preprocessing time but has some disadvantages. The first disadvantage is that this TPG algorithm is not feasible for redundancy removal because by removing redundant elements during test generation, the processed circuit is changed and the formula also has to be modified. In this case, some global

implications found during static learning become invalid. The second disadvantage is that static learning is restricted to a fault-free circuit and the formula cannot be minimized because an identity between the circuit and the formula must be kept. The first disadvantage is not critical because the best results for redundancy removal can be achieved by resynthesis. To address the second disadvantage, we propose static 16-V search space reduction and propagation path pruning techniques. We also utilize “Divide and Conquer” strategy by processing a single cone (primary output). This strategy may produce more that one test session for a redundant fault and even a testable fault because SPIRIT has to prove that the fault is untestable in respect to each primary output where the fault effect can be observed. On the other hand, application of this strategy together with static 16-V search space reduction allows higher reduction to be achieved.

3. SAT-based static learning

The SAT-based algorithms translate a problem into a characteristic formula that represents the constraints for the possible solutions [11]. The characteristic formula is usually written in 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. For instance, $(X \vee \overline{Z}) \wedge (Y \vee \overline{Z}) \wedge (\overline{X} \vee \overline{Y} \vee Z)$ is the CNF formula of an AND gate $Z = X \wedge Y$. It consists of one ternary and two binary clauses and evaluates to 1 iff the value of variables X, Y and Z is consistent with the truth table of the corresponding logic gate.

The first step in satisfying a CNF formula is to construct an implication graph. More formally, each variable X is represented by two nodes in the implication graph labeled X and $\overline{X}$. Next, each binary clause $(X \vee Y)$ can be viewed by two *local implications* $(\overline{X} \rightarrow Y)$ and $(\overline{Y} \rightarrow X)$ between these nodes, i.e., for each binary clause $(X \vee Y)$ there are two directed edges in the implication graph that represent these local implications.

Thus, the formula can be easily manipulated since a binding procedure requires only a partial traversal of the implication graph and a checking of the k-nary clauses [11]. Contrary to local implications that can be extracted by the structure of a circuit, detection of *global implications* requires a special analysis of the logic functions as they represent information that is not obvious from the circuit description. To find global implications, the static learning procedure iterates though the variables of the circuit performing both value assignments (0 and 1) until one full iteration produces no new implications. The derived global implications by static learning play an important role in avoiding an exhaustive backtracking during satisfying the CNF formula.

The leading static learning procedures [8,15] can find only a limited number of global implications. In [15] static learning is performed as a preprocessing phase based on the contradiction, Equation (1), where $A,B \in \{0,1\}$.

$$(X=A \rightarrow Y=B) \Leftrightarrow (Y=\overline{B} \rightarrow X=\overline{A}) \qquad (1)$$

Clearly, only the 2CNF portion of a CNF formula (only the binary clauses) fulfills priory the law of contraposition. This is not true when the k-nary clauses are also included. For example, it is possible that an assignment sets k-1 variables in a k-nary clause and the clause is still unsatisfied. In this case, the last variable should be set to a logic value so that this clause is satisfied. If the last variable of the clause is an output (or input) of the corresponding gate, this binding rule is called *a forward (or backward) local $\wedge$-implication*.

For the circuit shown in Figure 2(a), an assignment B=0 sets variables D and E to 0 and a ternary clause (D $\vee$ E $\vee$ $\overline{F}$) corresponding to gate F is still unsatisfied. To satisfy this clause, forward local $\wedge$-implication is performed and variable F is set to 0. Thus, *backward global implication* (F=1 $\rightarrow$ B=1) is found by contradiction. For the circuit shown in Figure 2(b), an assignment D=1 sets variables A and C to 0, and a ternary clause (A $\vee$ B $\vee$ C) is still unsatisfied. To satisfy this clause, the binding procedure performs a backward local $\wedge$-implication and sets the last unspecified variable B to 1. Thus, *forward global implication* (B=0 $\rightarrow$ D=0) is found by contradiction.

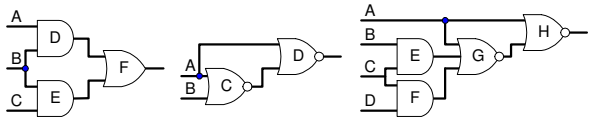


Figure 2: Network examples 1, 2 and 3

In [8] another concept, called an unjustified line, has been introduced. On this basis, some necessary assignments can be derived as an intersection of the conditions for satisfying a k-nary clause. This is called here *a global $\wedge$-implication*. Let us consider an example: how the static learning procedure derives backward global implication (H=1 $\rightarrow$ C=1) in the circuit shown in Figure 2(c). First, assignment H=1 sets variables A and G to 0 and k-nary clause (A $\vee$ E $\vee$ F $\vee$ G) is still unsatisfied. To satisfy this clause either variable E or variable F must be set to 1, but each one of these assignments implies that variable C should be set to 1. Therefore C=1 is a necessary assignment because it is intersection of the conditions to justify this clause. Thus, backward global implication (H=1 $\rightarrow$ C=1) is found. Clearly, this global implication cannot be found by contradiction. This global implication can be easily found by deriving and performing *global $\wedge$-implications* during static learning [4]. In [4], we also showed that some static global $\wedge$-implications derived during static learning can be transformed without any spare operations to *dynamic global implications* during the satisfying a formula.

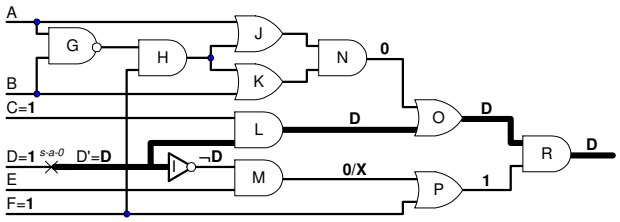
4. Single path oriented propagation method

The *Single Path Oriented Propagation* (SPOP) method has been introduced in [9]. Beginning from the fault location forward to 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 the SPOP method, the fanout stems are decision points and next path-segment is selected by the initial sorting of the fanout branches using fault propagation coefficients (see system chart in Figure 1). In this sense, the SPOP method is an alternative to the X-path check analysis (for the structural TPG algorithms) and the extracting structural constraints (for the SAT-based TPG algorithms). The main advantage of SPOP method is that it first sensitizes the easiest path for fault effect propagation according to the selected criteria. In contrast, the alternative methods can block the shortest propagation path by justification of the current objective (path-segment). The disadvantage of the SPOP method is that the number of unjustified lines increase considerably and it is possible that the sensitized propagation path is unjustifiable. To reduce these cases, we use static learning, static 16-V search space reduction and propagation path pruning techniques.

The *16-V search space reduction* (SSR) is performed as a preprocessing phase of each test session. SSR starts from the fault location forward to the primary output, and calculates the possible assignments in a set of {0, 1, D, $\overline{D}$} for each variable in the CNF formula. SSR also takes into account the number of propagation

paths and reduces the values {0,1} when only one propagation path is possible. Using directly observable points, some propagation paths can be reduced and even the CNF formula can be proved as unsatisfiable. For example, if a propagation path passes through a line that is directly observable for another primary output, then the values {D, $\overline{D}$} for the corresponding variable and therefore all propagation paths that pass via this line can be reduced. If the set of possible assignments of a variable is empty then the CNF formula is proved as unsatisfiable during SSR. In this way, SSR determines the logical constraints for a test session and finds some new global implications valid for the faulty circuit. However, both static learning and SSR cannot guarantee that the successfully sensitized propagation path is justifiable. In some cases, it is possible that a propagation path is sensitized but one or more unjustified lines cannot be justified at a time. We avoid these cases using a propagation path pruning technique.

The *propagation path pruning (PPP)* is performed after the first sensitized propagation path of a test session is proved as unjustifiable. PPP checks whether each unjustified line can be justified. If an unjustifiable line is found then the PPP procedure determines (by spare value assignments) and backtracks to the nearest level of SPOP where this unjustified line can be justified. Figure 3 shows an example of all techniques described above.



(a)

SAT variables	possible assignments	SAT variables	possible assignments
A	{0,1}	I	{¬D}
B	{0,1}	J	{0,1}
C	{0,1}	K	{0,1}
D	{1}	L	{0,D}
D'	{D}	M	{0,¬D}
E	{0,1}	N	{0,1}
F	{0,1}	O	{0,1,D}
G	{0,1}	P	{0,1,¬D}
H	{0,1}	R	{D,¬D}

(b)

Figure 3: Network example 4

Example 1: The circuit is depicted in Figure 3(a) and the fault under consideration is D:s-a-0. Figure 3(b) shows the possible assignments for the variables included in the CNF formula. Let us consider that propagation path *D'-L-O-R* is selected and sensitized. In this case, N is only an unjustified line but this line is unjustifiable. If line R is not a primary output that can produce sensitization of many propagation paths until the alternative propagation path, *D'-I-M-P-R* is sensitized. This particular case can be easily solved if static learning is performed. For example, if static learning is performed then four global implications will be found: (F=1→J=1), (F=1→K=1), (N=0→H=0) and (R=0→H=0). These new implications find inconsistency during SPOP by assignment O=D. In the few cases where the global implications derived by static learning are not enough, the application of the propagation path pruning technique can also reduce the number of sensitized unjustifiable propagation paths.

5. Backward justification and duality of learning phenomenon

Justification is performed after a successful sensitization of propagation path. It has to justify all unjustified lines [8], defined as a complete set of all unsatisfied k-nary clauses corresponding to gates with specified outputs. In this way, we assume that the CNF formula is satisfied when a test pattern is found, i.e., when a fault effect is propagated to the primary output and all unjustified lines are justified. Many SAT-based TPG algorithms try to satisfy all clauses in the CNF formula. This approach can produce a massive backtracking and even aborting after a test pattern is found. On the other hand, this also increases the number of specified primary

inputs. The goal here is to minimize both the efforts for satisfying the CNF formula and the number of specified primary inputs. To do so, we propose *backward justification* based on static justification coefficients calculated during preprocessing (see the systems chart in Figure 1). The justification coefficients take into account the structure of the circuit and measure the relative difficulty for justification of each line in the circuit. By sorting the inputs of each logic gate using justification coefficients before extracting the implication graph, each unjustified line can be relatively easily justified by assigning a certain/controlling value to the first unspecified variable/input of the corresponding k-nary clause/gate.

During the SPOP phase all unjustified lines are included dynamically in a justification list which is sorted in increasing order of justification coefficients before backward justification (heuristic BJ1). During backward justification this list is also dynamically updated by adding new unjustified lines at the end of the list when the unjustified lines are processed (justified) starting from the end to the beginning of the justification list. Actually, this is a dynamic reordering of the variables based on the depth first search strategy. This strategy is more efficient than static order typical for the SAT-based TPG algorithms [3,11,16]. In case that a backtrack limit is exceeded, the backward justification is repeated with the reverse ordered justification list (heuristic BJ2). In case that both heuristics, BJ1 and BJ2, cannot justify or prove unjustifiable the sensitized propagation path, the backward justification performs heuristics BJ1* and BJ2*, i.e., heuristics BJ1 and BJ2 using dynamic learning (first level recursive learning [10]).

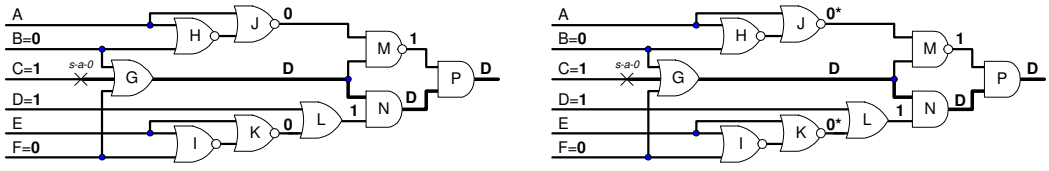


Figure 4: Network example 5

Example 2: Let us consider a fault C:s-a-0 for the circuit shown in Figure 4(a). During static learning four global implications are found: forward implications (B=0 →J=0) and (F=0→K=0) and backward implications (M=1→J=0) and (N=0→K=0). After assignment C'=D, the values are shown in Figure 4(a) and fault effect is propagated to the primary output. As a result, four primary inputs are specified and two unjustified lines J and K are included in the justification list. During backward justification, variables A and E are set to value 1 in order to justify lines J and K. However, both these unjustified lines are already justified therefore primary inputs A and E can be left unspecified.

This phenomenon is called here *duality of learning*. Obviously, static learning is important to decrease the number of backtracks. On the other hand, static learning may produce some spare unjustified lines that have to be justified. For example without static learning, K will be unspecified after SPOP since variable K is set to 0 by forward global implication (F=0→K=0). In this case, J and L will be included in the justification list. To justify these lines, variables A and D will be set to 1, i.e., primary input E will be left unspecified. We propose two solutions to improve the backward justification procedure:

Solution 1: After static learning, all forward global implications are removed since these implications produce some spare unjustified lines.

Solution 2: The 9-V composite algebra is extended to a composite algebra that can identify and take into account these special cases during backward justification.

6. Improved back justification

In the previous section, we presented duality of learning phenomenon and showed that by removing all forward global implications a reduction of the number of unjustified lines can be expected. In general, this approach avoids justification of some spare unjustified lines produced by forward global implications. This decreases the depth of the decision tree and the number of backtracks. We also showed that some spare unjustified lines (already justified) may exist even when all global implications are removed. In this section, we solve this problem by introducing a problem oriented 11-V composite algebra. For simplicity, we restrict application of forward global implications in a fault-free region. Table 1 shows notation and meaning of the well-known 9-V composite algebra and its extension to 11-V composite algebra. Two new values 0* and 1* are introduced to indicate that the line is set to 0/0 or 1/1 and can be considered as justified, i.e. if an unjustified line is set to a value in the set {0*,1*} then no justification of this line is necessary.

Table 1: Notations of 11-V composite algebra

9-V composite algebra						Extension	
Notation	Good/Faulty	Notation	Good/Faulty	Notation	Good/Faulty	Notation	Good/Faulty
X	X/X	D	1/0	G1	1/X	0*	0/0
0	0/0	$\overline{D}$	0/1	F0	X/0	1*	1/1
1	1/1	G0	0/X	F1	X/1		

Example 3: Let us consider again fault C:s-a-0 for the circuit shown in Figure 4(b). Two forward global implications, (B=0 →J=0) and (F=0→K=0), and two backward global implications, (M=0→J=1) and (N=0→K=0), are found during static learning. After assignment C'=D, the fault effect is propagated to the primary output and the assigned values are shown in Figure 4(b). As a result, variables J and K are set to 0* by performing forward implications (B=0 →J=0) and (F=0→K=0) respectively. Therefore these lines do not need justification. Since the justification list is empty, a test pattern is found.

The advantages of Solution 2 are obvious. It allows forward global implications (about 16% of global implications) to be used for a reduction of the justification list. In general, this can decreases the depth of the decision tree, the number of backtracks and also the number of specified primary inputs. The last parameter is important for test compaction, test generation for sequential circuit and other applications of TPG algorithms.

7. Experimental results

We implement this TPG algorithm as well as the new data structure for the complete implication graph presented in [4] in the SPIRIT ATPG system [5] and ran experiments on the ISCAS'85[1] and full scan version of ISCAS'89[2] benchmark circuits on a 450 MHz Pentium-III PC (SpecInt'95=18.7). For all experiments, the maximal number of the sensitized propagation paths during SPOP and the maximal number of backtracks during backward justification was set at 10. As a result, there were no aborted test sessions, i.e., without fault simulation SPIRIT was able to generate complete test set and prove all redundant faults in the ISCAS'85 and ISCAS'89 benchmark circuits.

Table 2 presents the number of sensitized propagation paths for successful and unsuccessful test sessions, i.e. test patterns exist and no test patterns exist respectively. Accordingly, 99.97% of the successful test sessions generated test patterns by justification of the first sensitized propagation path. Also, 99.83% of the

unsuccessful test sessions were identified during SSR or SPOP, i.e., without justification. Only 89 test sessions (0.05%) sensitized one or more unjustifiable propagation paths.

Table 2: The number of sensitized propagation paths for successful/unsuccessful test sessions

Circuit	Successful test sessions					Unsuccessful test sessions				#test	#test
	paths=1	=2	=3	=4	=5	SSR	paths=0	=1	=2	sessions	patterns
C432	517	3	0	0	0	0	27	0	0	547	520
C499	750	0	0	0	0	0	156	0	0	906	750
C880	942	0	0	0	0	0	0	0	0	942	942
C1355	1566	0	0	0	0	0	156	0	0	1722	1566
C1908	1870	0	0	0	0	0	177	0	0	2047	1870
C2670	2627	3	0	0	0	220	90	9	0	2949	2630
C3540	3291	0	0	0	0	0	649	0	0	3940	3291
C5315	5287	4	0	0	0	84	177	9	8	5569	5291
C6288	7707	3	0	0	0	0	380	0	0	8090	7710
C7552	7400	12	4	2	1	86	1294	0	0	8799	7419
S1494	1494	0	0	0	0	0	13	0	0	1507	1494
S5378	4563	0	0	0	0	83	235	0	0	4881	4563
S9234	6471	3	1	0	0	130	833	0	0	7438	6475
S13207	9653	11	0	0	0	74	1020	16	0	10774	9664
S15850	11336	0	0	0	0	765	9968	0	0	22069	11336
S35932	35110	0	0	0	0	0	5376	0	0	40486	35110
S38417	31015	0	0	0	0	36	1052	0	0	32103	31015
S38584	34797	0	0	0	0	267	964	0	0	36028	34797
Total:	**166396**	**39**	**5**	**2**	**1**	**1745**	**22567**	**34**	**8**	**190797**	**166443**

Table 3 shows the results of the application of Solution 1 (all forward global implications are removed after static learning). In this case, we took advantage of static learning but avoided the justification of some spare unjustified lines. Columns 2-5 show the number of local implications, forward and backward global implications and the number of reduced indirect implications by removing the forward global implications. Columns 6-11 show the number of aborted justifications using heuristics BJ1, BJ2 and both BJ1&BJ2 in two cases: (1) when all global implications were used; and (2) when only backward global implications were used (Solution 1). As a result of the application of Solution 1, we observed a significant reduction in the number of aborted justifications 81.11%, 71.66% and 97.75%, respectively.

Table 3: The number of implications and aborted test sessions using heuristics BJ1, BJ2 and BJ1&BJ2

Circuit	Implications				All global implications 9-V algebra			Only backward global implications		
	local	forward global	backward global	reduced Indirect	BJ1	BJ2	BJ1&BJ2	BJ1	BJ2	BJ1&BJ2
1	2	3	4	5	6	7	8	9	10	11
C432	1624	1	68	3	3	0	0	3	0	0
C499	1504	128	8	608	8	0	0	5	0	0
C880	3384	0	76	0	0	0	0	0	0	0
C1355	5344	336	112	3168	9	8	4	1	2	0
C1908	7854	18	404	201	0	2	0	0	2	0
C2670	10162	31	552	656	0	0	0	0	0	0
C3540	14588	93	1156	4364	7	0	0	7	0	0
C5315	21880	38	1413	185	30	21	0	30	15	0
C6288	25024	1018	930	4125	466	575	216	29	142	3
C7552	30440	115	3005	1835	22	15	2	22	15	2
S1494	6296	5	5716	19	0	0	0	0	0	0
S5378	21182	38	1168	954	0	0	0	0	0	0
S9234	36642	299	3332	5929	0	0	0	0	0	0
S13207	51198	520	6629	14034	32	0	0	12	0	0
S15850	61794	953	4567	24847	0	0	0	0	0	0
S35932	135396	3798	2811	442431	0	0	0	0	0	0
S38417	148980	1053	6186	18124	0	0	0	0	0	0
S38584	151982	861	9935	13019	0	0	0	0	0	0
Total:	**735274**	**9305**	**48068**	**534502**	**577**	**621**	**222**	**109**	**176**	**5**

Table 4 shows the final results. Columns (2-3) give the average and maximal number of binary variables in CNF formula (size of problem), columns (4-6) and columns (7-9) give the number of value assignments, the number of levels of the decision trees (DT) and the number of specified primary inputs (PI) for Solution 1 and Solution 2. The last column (10) presents the CPU time in seconds. In some cases, the CPU time required for static learning was almost two times the required time for test generation (circuits S35932 and S38584) [4]. To decrease preprocessing time, we applied a restricted static learning when a huge dependency between signals was found (the average number of set variables by value assignment is greater than 100). These benchmark circuits are marked by (*) in Table 4.

Table 4: Experimental results on the ISCAS'85 and ISCAS'89 benchmark circuits

Circuit	Size of problem		Solution 1			Solution 2			CPU
	Average	Max	#assign.	#levels of DT	#spec. PI	#assign.	#levels of DT	#spec. PI	time, s
1	2	3	4	5	6	7	8	9	10
C432	124.8	204	9362	8472	6305	+0	+0	+0	0.14
C499	150.8	190	24617	23563	25375	+0	+0	+0	0.26
C880	102.2	187	9089	8082	9126	+0	+0	+0	0.17
C1355	381.5	462	49235	47007	56204	+0	+0	+0	1.67
C1908	532.5	784	31115	26650	37617	+0	+0	+0	2.08
C2670	306.9	953	36111	39159	40477	+0	+0	+1	1.56
C3540*	581.7	1832	40278	35693	44126	-74	-67	+19	5.07
C5315	272.0	1106	66344	57207	70319	+18	+6	+0	2.97
C6288	1327.2	3526	269940	230701	193171	+12910	+4778	+2542	49.17
C7552*	547.2	1293	230532	211407	230551	-1160	-1097	-1158	10.91
S1494*	74.0	122	5606	3579	10479	-3	-3	-5	0.80
S5378*	134.8	571	23589	18513	35816	-119	-130	-54	1.39
S9234*	354.8	1238	45323	38066	75560	-98	-98	-45	3.88
S13207*	188.9	1573	43150	31281	74320	-12	-13	-25	7.04
S15850*	625.4	1750	71521	62834	93388	-154	-595	-48	18.65
S35932*	77.8	134	144035	108996	167312	-320	-320	-320	27.41
S38417*	197.1	641	357558	320978	406429	-99	-99	-274	21.66
S38584*	93.2	897	147167	110223	211007	-44	-36	-8	17.50
Total:	**285.1**	**3526**	**1604572**	**1382411**	**1787582**	**+10845**	**+2326**	**+625**	**172.33**

Since SPIRIT processes a single cone, the number of test sessions was usually bigger than the number of faults in the collapsed fault list. For example, the number of test sessions for circuit S15850 was 88% more than the number of processed faults when the average ratio was 9.9% (Table 2). On the other hand using this approach, SPIRIT keeps the size of problem as small as possible and also allows more effective application of static 16-V search space reduction. It is also interesting to mention that SPIRIT does not apply any techniques that need a special analysis such as X-path check, unique sensitization, multiple primary input assignments, search pruning diagnostics or transitive closure (equivalent to the first level recursive learning). After static learning and 16-V search space reduction, the proposed TPG algorithm selects a variable in the CNF formula and assigns a proper value according to the current objective, fault effect propagation or justification. Therefore, the number of value assignments gives a clear estimation of the efforts spent for satisfying the CNF formula. In this sense, the ratio between columns (4) and (5) and between (4) and (6) assesses the spare assignments and the number of assignments necessary to determine the value of one primary input. Except for the circuits C432 and C6288 the number of specified primary inputs was larger than the number of assignments which demonstrated the effectiveness of the improved backward justification procedure in respect to the PODEM approach. For the most of the considered benchmark circuits Solution 2 for the improved backward justification gave better results than Solution 1. This is not always valid because in some cases the justification of the spare unjustified lines gives a better solution for the justification list. For this benchmark set, the total CPU time was 172.33 seconds which made SPIRIT competitive to the best published ATPG systems TEGUS [16] (477

seconds on a 200 MHz Pentium Pro PC) and ATOM [7] (371.6 seconds on a 200 MHz Pentium Pro PC). We believe that the advantages of SPIRIT will be more visible with more difficult benchmark circuits. For example, LookBack is the basic technique of ATOM since without using LookBack, 999 faults in the circuits C5315, C6288 and S13207 will be aborted. However, this technique is ineffective in respect to redundant faults. If there were a few hard to prove redundant faults in the processed benchmark circuits, then the LookBack technique could degrade the overall performance of ATOM.

8. Conclusions

In this paper we presented an efficient TPG algorithm based on the Boolean Satisfiability method. We applied some not so popular methods such as a single cone processing, single path oriented fault effect propagation [9] and deriving and performing of global $\wedge$-implications [4]. We proposed some efficient techniques such as static 16-V search space reduction and propagation path pruning as well as the improved backward justification using 9-V composite algebra and a new problem oriented 11-V composite algebra. To assess the effectiveness of the proposed new techniques as well as the new data structure for the complete implication graph, we implemented this TPG algorithm in ATPG system SPIRIT [5] using a static variable order typical for the SAT-based TPG algorithms. As a result, a considerable impact on the TPG efficiency and robustness has been achieved. Without fault simulator, SPIRIT was able to generate complete test sets and prove all redundant faults in the ISACAS'85 and ISCAS'89 benchmark circuits in reasonable amount of time. These results could be achieved even without dynamic learning but we kept this powerful technique to guarantee the robustness of our TPG algorithm for future applications as combinational test generation for megagate circuits and design for testability of sequential circuits.

Acknowledgment

This paper was supported by JSPS under grant P99747.

References:

[1] F.Brglez and H.Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," in Proc. IEEE ISCAS, 1985, pp. 663-698.

[2] F.Brglez, D.Bryan and K.Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," in Proc. IEEE ISCAS, 1989, pp.1929-1934.

[3] 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.

[4] 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.

[5] E.Gizdarski, "Test Pattern Generation using Boolean Satisfiability," Automatica and Informatics, vol-32, No.2, 1998, pp. 33-40.

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

[7] 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, August 1999, pp.63-73.

[8] 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.

[9] M.Henftling, H.Wittmann and K.Antreich, "A Single-Path-Oriented Fault-Effect Propagation in Digital Circuits Considering Multiple-Path Sensitization," in Proc. IEEE/ACM ICCAD, 1995, pp.304-309.

[10] 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.

[11] T.Larrabee, "Test Pattern Generation Using Boolean Satisfiability," IEEE Trans. on CAD, vol.11, No.1, Jan.1992, pp.4-15.

[12] M.Teramoto, “A Method for Reducing the Search Space in Test Pattern Generation,” in Proc. ITC, 1993, pp.429-435.

[13] P.Muth, "A Nine-Value Logic Model for Test Generation," IEEE Trans. on Computers, vol.C-25, No.6, June 1976, pp.630-636.

[14] J.Rajski and H.Cox, “A Method to Calculate Necessary Assigmnets in Algorithmic Test Generation”, in Proc. IEEE ITC, 1990, pp.25-34.

[15] 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.

[16] P.Stephan, R.K.Brayton and A.L.Sangiovanni-Vincentelli, ”Combinational Test Generation using Stisfiability,” IEEE Trans. on CAD, vol.15, No.9, Sept. 1996, pp.1167-1176.