PASSAT: Efficient SAT-based Test Pattern Generation for Industrial Circuits

J. Shi G. Fey R. Drechsler A. Glowatz F. Hapke J. Schlöffel

Institute of Computer Science
University of Bremen
28359 Bremen, Germany

Design Technology Center Hamburg
Philips Semiconductors GmbH
22502 Hamburg, Germany

Abstract *Automatic Test Pattern Generation (ATPG) based on Boolean Satisfiability (SAT) has been proposed as an alternative to classical search algorithms. SAT-based ATPG turned out to be more robust and more effective by formulating the problem as a set of equations.*

In this paper we present an efficient ATPG algorithm that makes use of powerful SAT-solving techniques. Problem specific heuristics are applied to guide the search. In contrast to previous SAT-based algorithms, the new approach can also cope with tri-states. The algorithm has been implemented as the tool PASSAT. Experimental results on large industrial circuits are given to demonstrate the quality and efficiency of the algorithm.

1 Introduction

Many algorithms for *Automatic Test Pattern Generation* (ATPG) have been proposed in the last 30 years. But due to the ever increasing design complexity new techniques have to be developed that can cope with todays circuits.

While classical approaches are based on backtracking on the circuit structure, since the early 80s several approaches based on Boolean Satisfiability (SAT) have been proposed (see e.g. [15, 14, 17, 16]). In [15] comparisons to more than 40 other “classical” approaches based on FAN, PODEM and the D-algorithm are provided showing the robustness and effectiveness of SAT-based techniques. In contrast to the early work in the 80s, where ATPG was often reduced to PROLOG or logic programming, the ATPG techniques developed by Larrabee [7] and Stephan et al. [15] used many simplification and learning approaches, like global implications [17]. By this, these algorithms combined the power of SAT with algorithms from the testing domain.

Recently, there is a renewed interest in SAT, and many improvements for proof engines have been proposed. SAT solvers make use of learning and implication procedures (see e.g. [10, 11]). These new proof techniques have led to breakthroughs in several applications, like formal hardware verification [2].

In this paper we present an efficient SAT-based ATPG algorithm. The algorithm has been implemented as the tool *PASSAT* (PAttern Search using SAT). It makes use of techniques recently developed for SAT solvers and so far only applied in the verification domain. In contrast to previous SAT approaches, that only considered Boolean values, we study the problem for a multi-valued logic encoding, i.e. our ATPG tool can also consider unknown values and tri-states. The influence of different branching heuristics is studied to tune the SAT solver towards test generation, i.e. variable selection strategies known from SAT and strategies applied in classical ATPG [4, 3]. Experimental results on large industrial circuits are given to demonstrate the efficiency of our approach.

2 SAT-based ATPG

In the following we briefly review the D-algorithm, its application to SAT-based test pattern generation in TEGUS [15], and the improvements of PASSAT over TEGUS. The technique to handle circuits with tri-state elements and to use unknown values is introduced in Section 2.2. Then, different decision schemes for variable selection are explained.

2.1 D-Algorithm

The D-Algorithm [12] was originally proposed for combinational test generation in classical ATPG. The basic ideas of the algorithm can be summarized as follows:

- An error is observed due to differing values at a line in the circuit with or without failure. Such a divergence is denoted by values D or $\overline{D}$ to mark differences $1/0$ or $0/1$, respectively.
- Instead of Boolean values, the set $\{0,1,D,\overline{D}\}$ is used to evaluate gates and carry out implications.
- A gate that is not on a path between the error and any output does never have a D-value.
- A necessary condition for testability is the existence of a path from the error to an output, where all intermediate gates either have a D-value or are not assigned, yet. Such a path is called a potential D-chain.
- A gate is on a D-chain, if it is on a path from the error location to an output and all intermediate gates have a D-value.

On this basis an ATPG algorithm can focus on propagating D-values and applying the error.

The above observations have been exploited to implement the SAT-based test pattern generator TEGUS [15]. Improvements on the justification and propagation have been proposed in [17, 16].

First a given circuit is converted into an equation in *Conjunctive Normal Form* (CNF). Then the algorithm uses this representation as the basis to carry out SAT-solving. Besides the circuit additional information is encoded in the CNF to benefit from the above observations and make them explicit in the CNF. Three variables are used for each gate G:

- G_f denotes the value in the faulty circuit.
- G_g denotes the value in the correct circuit.
- $G_d = 1$, iff G is on a D-chain.

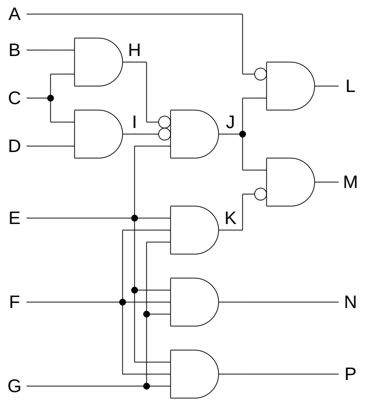

Figure 1: Example network fragment from [15]

Table 1: Representation of a gate in CNF

Constraint	CNF description
$J_g \equiv (\overline{H}_g \cdot \overline{I}_g \cdot E_g)$	$(J_g + H_g + I_g + \overline{E}_g) \cdot (\overline{H}_g + \overline{J}_g) \cdot (\overline{I}_g + \overline{J}_g) \cdot (\overline{E}_g + \overline{J}_g)$
$J_f \equiv (\overline{H}_f \cdot \overline{I}_f \cdot E_f)$	$(J_f + H_f + I_f + \overline{E}_f) \cdot (\overline{H}_f + \overline{J}_f) \cdot (\overline{I}_f + \overline{J}_f) \cdot (\overline{E}_f + \overline{J}_f)$
$J_d \rightarrow (J_g \neq J_f)$	$(\overline{J}_d + J_g + J_f) \cdot (\overline{J}_d + \overline{J}_g + \overline{J}_f)$
$J_d \rightarrow (L_d + M_d)$	$(\overline{J}_d + L_d + M_d)$

This notation allows to introduce additional implications, that lead to a fast evaluation of the CNF.

- If G is on a D-chain, the values in the faulty and the correct circuit are different: $G_d \rightarrow (G_f \neq G_g)$.
- If G is on a D-chain, at least one successor of G must be on the D-chain as well: Let $H^i, 1 \leq i \leq q$ be the successors of G, then $G_d \rightarrow \bigvee_{i=1}^{q} H_d^i$.
- If a gate G is not on a potential D-chain, the values in the faulty and the correct circuit are identical: $G_f = G_g$.

Additionally, the initial circuit is transformed into a circuit of AND gates to benefit from the resulting simplification during CNF generation and fault simulation (similar to the conversion into AND/INVERTER graphs in formal verification [6]). Then, the integrated SAT solver is used to solve the generated CNF formula.

Example 1 *Consider the network fragment shown in Figure 1 that has been presented in [15]. Applying the above descriptions to gate J yields the Boolean constraints given in Table 1. These constraints are translated into the CNF that is also shown in the table.*

Recently, SAT solvers have been greatly improved due to three main techniques:

- Conflict Analysis [9] allows to prune parts of the search space that do not yield solutions. Similar but often less powerful techniques for classical ATPG are often referred to as "learning".
- Boolean Constraint Propagation (BCP) [11] corresponds to implications carried out in classical ATPG. But SAT solvers only use simple implications to maintain efficiency.
- Variable selection strategies [5] have been tuned for robustness with respect to a wide range of problem instances. While decisions in ATPG are usually based on structural criteria, SAT solvers often collect run time statistics to carry out decisions.

To benefit from these advances PASSAT relies on the advanced SAT-solver zchaff [11]. The basic clause generation is the same as proposed in the TEGUS approach. This clause generation has been interfaced with zchaff. As a result the clause database can directly be accessed during clause generation to provide an efficient ATPG flow even for a large number of test patterns.

Table 2: 4-valued AND gate

AND	0	1	Z	U
0	0	0	0	0
1	0	1	U	U
Z	0	U	U	U
U	0	U	U	U

Table 3: Size of encodings

#encodings	#clauses
7	8
9	9
8	11

Table 4: Encoding of the 4-valued domain

X	Encode		Interpretation
	c_x	c_x^*	
0	0	0	Signal X is 0
1	1	0	Signal X is 1
U	1	1	Signal X is unknown
Z	0	1	Signal X is at 'Z'

Table 5: Clauses for the 4-valued AND-gate

$$(\overline{c}_a+\overline{c}_b+c_c)\cdot(c_a^*+c_b^*+\overline{c}_c^*)\cdot(c_c+\overline{c}_c^*)\cdot$$
$$(c_a+c_a^*+\overline{c}_c)\cdot(c_b+c_b^*+\overline{c}_c)\cdot(\overline{c}_a^*+\overline{c}_b+c_c^*)\cdot$$
$$(\overline{c}_a+\overline{c}_b^*+c_c^*)\cdot(\overline{c}_a^*+\overline{c}_b^*+c_c^*)$$

Table 6: CNF size for a 2-input AND-gate

	CNF description			
	2-valued		4-valued	
Constraint	#cls	#lit	#cls	#lit
$C_g \equiv (A_g \cdot B_g)$	3	7	8	23
$C_f \equiv (A_f \cdot B_f)$	3	7	8	23
$C_d \rightarrow (C_g \neq C_f)$	2	6	5	16
$C_d \rightarrow (D_d + E_d)$	1	3	1	3
Overhead	1	1	2.4	2.8

2.2 Encoding

For simplicity SAT approaches have not modeled high impedance values so far, but circuits may contain tri-state elements to describe the access of multiple drivers to a bus. Therefore the new value Z is introduced to model such circuits. This value has to be respected correctly during test pattern generation as well. Additionally it is beneficial to know which assignments in a test pattern are necessary and which are optional to exploit this information for compaction of a set of test patterns. Optional assignments are denoted by a value U. Therefore the circuit operates on a 4-valued domain: $\{0,1,U,Z\}$. The function of any gate can be extended to this domain. Table 2 exemplary shows the truth table of an AND-gate.

In order to use a Boolean SAT solver on such a circuit the 4-valued domain has to be encoded in a binary representation. This can be done using two binary variables to encode one 4-valued variable. (A similar encoding for a 3-valued domain has been introduced in [17].) There exist $4! = 24$ different encodings. All encodings have been applied to describe the AND-gate as a CNF formula. The CNF formula has been optimized by using ESPRESSO [13] to minimize the Sum-of-Product representation of the OFF-set. The efficiency of the solutions varies. Table 3 lists the number of encodings, that led to 8, 9 and 11 clauses for the AND-gate. During generation of the CNF formula of a circuit a small representation for a single gate is beneficial, as clauses are generated per gate. Therefore the encoding scheme in Table 4 that leads to a minimal description with 8 clauses has been chosen. The resulting clauses for the AND-gate are shown in Table 5. Only the AND-gate is considered due to the transformation of the circuit into a network of AND-gates (see above).

After choosing an encoding the derivation of the CNF description from the truth table of

(1) *Variable Selection*: Choose an unassigned variable and assign a new value.

(2) *Boolean Constraint Propagation (BCP)*: Carry out implications resulting from the previous assignment.

(3) *Solution*: If all clauses are satisfied, output the current variable assignment and return "satisfiable". If all assignment have been exhausted, return "unsatisfiable".

(4) If there is no unsatisfied clause due to the current assignment proceed with step (1). *Conflict analysis*: If the current assignment leads to at least one unsatisfied clause, carry out conflict analysis and add conflict clauses.

(5) *(Non-chronological) Backtracking*: Backtrack to the most recent variable assignment, where switching the variable could lead to a solution. Proceed with step (2).

Figure 2: Basic loop of a SAT solver

an arbitrary gate is straightforward. This includes Busses and Busdrivers. Again, the number of clauses needed for a single gate can be minimized using 2-level logic optimization. This enables to model elements such as tri-state elements and buses in the binary domain.

Table 6 shows the number of clauses and literals needed for an AND-gate C with inputs A,B and successors D,E in the 2-valued and 4-valued model, respectively. The size of the CNF increases by roughly 2.5 times. Additionally the implication $C_d \rightarrow (C_g \neq C_f)$ is weakened: In the 2-valued domain the assignment $C_d = 1, C_g = 1$ directly implies $C_f = 0$ but the same assignment in the 4-valued domain only implies $C_f \in \{0, Z\}$. Therefore more decisions are necessary to solve the 4-valued SAT instance.

2.3 Variable Selection

A SAT solver traverses the search space by a back-tracking scheme. The basic loop carried out by the solver is shown in Figure 2 (see e.g. [11] for more details).

While BCP and conflict analysis have greatly improved the speed of SAT solvers, the variable selection strategy in step (1) is crucial for an efficient traversal of the search space. But no general way to choose the best variable is known, as the decision about satisfiability of a given CNF formula is NP-complete [1]. Therefore SAT solvers have sophisticated heuristics to select variables.

2.3.1 Heuristics for SAT Solving

Often the heuristic accumulates some statistics about the CNF formula during run time of the SAT solver to use this data as the basis for decisions. This leads to a trade-off between the quality of a decision and the overhead needed to update the statistics. Also, the quality of a given heuristic often depends on the problem domain.

The default variable selection strategy applied by zchaff is the quite robust *Variable State Independent Decaying Sum* (VSIDS) strategy. Conceptually the idea is to focus on those variables, that have the strongest influence on the current region of the search space. In practice this is realized, by counting the occurrences of variables in conflict clauses. The counters are periodically divided by two in order to emphasize the recent past of a run. For more elaborate discussions and comparisons of variable selection strategies for SAT-solving see [8, 5].

2.3.2 Heuristics for ATPG

Decisions based on variable selection also occur in classical test pattern generation. Here, usually structural methods are employed to determine a good choice for the next selection. Besides the default variable selection strategy from zchaff PASSAT provides two strategies similar to strategies known from classical ATPG: selecting primary inputs only or selecting fanout points only.

Making decisions on primary inputs only was the improvement of PODEM [4] over the D-algorithm. Any other internal value can be implied from the primary inputs. This yields a reduction of the search space and motivates to apply the same strategy for SAT-based test pattern generation as well. This is done by restricting the variable selection of the SAT solver to those variables corresponding to primary inputs or state bits of the circuit. Within these variables the VSIDS strategy is applied to benefit from the feedback of conflict analysis and current position in the search space.

Restricting the variable selection to fan-out gates only has been proposed in FAN [3] for the first time. Again, the idea is to restrict the search space while getting a large number of implications from a single decision. Conflicts resulting from a decision are often due to a small region within the circuit. These conflicts are detected with less effort, if fan-out gates and primary inputs are selected instead of only primary inputs. PASSAT applies the VSIDS strategy to select fan-out gates or primary inputs.

The experiments in Section 3 show that some heuristics are quite robust, i.e. can classify all faults, while others are fast for some faults but abort on other faults. Therefore an iterative approach turned out to be more effective:

1. One strategy is run with a given time out.
2. If the first strategy does not yield a test pattern a second, more robust, strategy is applied.

This approach assures, that a fast test pattern generation is carried out where possible, while a more sophisticated search is done for the remaining faults. For our experiments selecting only inputs was used as the first strategy, and selecting any variable as the second.

3 Experimental Results

The methods described above have been implemented in C as the tool PASSAT. The structure of TEGUS was used as the basic frame work. Zchaff was taken as the SAT solver. All experiments were carried out on an Athlon XP 2200+ workstation with 512 MByte of main memory. Run times are given in CPU seconds. In the following results for the different variable selection stategies are given. Then results for large industrial circuits including bus structures are given. Due to page limitation only a limited number of results is reported.

3.1 Variable Selection

First, we study the influence of the different variable selection strategies introduced in Section 2.3. The 4-valued circuit model is considered. Table 7 shows the results for the default strategy known from zchaff, using fanout-gates only, using inputs only, and the iterative strategy. For each strategy the number of aborted faults, the run-time needed to create the SAT problem and the run-time to solve the problem are reported in Columns *Ab*, *Eqn*, and *SAT*, respectively. Column *Imp* gives the speed-up of the iterative strategy over the default strategy from zchaff. For the iterative strategy a time out of 1 sec while selecting only inputs was allowed per fault, then the default strategy was applied. In case of the default strategy

Table 7: PASSAT with different variable selection strategies

	Default			Fanout			Inputs			Input, Default			
Circ.	Ab	Eqn	SAT	Ab	Eqn	SAT	Ab	Eqn	SAT	Ab	Eqn	SAT	Imp
c432	0	1.41	1.01	0	1.64	0.79	0	1.49	1.53	0	1.47	1.53	0.7
c880	0	1.43	0.86	0	1.44	0.94	0	1.25	0.43	0	1.42	0.28	3.1
c1355	0	8.79	38.8	0	8.70	33.3	1	8.72	8.60	0	8.92	8.72	4.4
c1908	0	8.59	12.6	2	8.83	17.8	13	8.63	16.0	0	9.90	15.8	0.8
c2670	0	5.96	4.09	0	6.30	4.09	5	6.01	6.83	0	6.61	7.70	0.5
c3540	0	38.2	33.8	2	38.4	37.3	2	39.1	15.6	0	39.8	15.3	2.2
c5315	0	24.6	33.3	0	25.2	42.5	0	24.5	8.51	0	24.6	8.31	4.0
c6288	0	229	634	169	238	485	517	234	635	0	715	636	1.0
c7552	0	49.2	141	12	51.0	153	2	49.6	24.0	0	50.1	23.5	6.0
s1494	0	1.30	0.51	0	1.30	0.59	0	1.28	0.54	0	1.29	0.55	0.9
s5378	0	4.10	3.26	0	4.65	2.94	0	4.47	1.24	0	4.17	1.48	2.2
s9234.1	0	7.73	5.43	0	8.06	5.60	0	7.90	2.25	0	8.19	2.01	2.7
s13207.1	0	31.6	92.9	30	33.8	100	19	32.0	40.0	0	55.4	39.9	2.3
s15850.1	0	62.9	79.7	0	67.4	72.2	0	63.5	20.5	0	63.9	20.7	3.9
s35932	0	31.9	32.6	0	66.0	11.6	0	31.4	5.32	0	32.0	5.34	6.1
s38417	0	80.7	64.6	0	100	66.3	0	82.2	31.8	0	82.6	32.5	2.0
s38584.1	0	40.8	26.3	0	68.1	27.5	0	40.5	14.1	0	40.8	14.3	1.8

no time out was used, for the remaining strategies 2 seconds were used to guarantee that a single fault does not block the test generation for remaining faults. The table shows that the default strategy is the most robust strategy among the non-iterative strategies. No abortions occur for this heuristic for the ISCAS benchmarks considered. The selection of fanouts only yields improvements in some cases (e.g. “s35932”), but can not classify all faults in several cases. The reason here is that the reduction of the search space does not compensate for the weak influence of conflict analysis on further decisions. The search space is more drastically pruned, when only inputs are allowed for selection. This yields a large speed up of up to 6 times in case of “s35932”. On the other hand difficult faults are often aborted, because internal conflicts during SAT-solving are detected too late. This is the case for “c6288”. Therefore the iterative strategy was applied to retrieve a fast solution where possible, while using the more sophisticated VSIDS strategy on all variables, where a fast solution was not possible. As a result the iterative strategy does not lead to any aborted faults. In most cases even a remarkable speed-up compared to the default strategy is gained. The last column of Table 7 shows the percentage of run time needed by SAT-solving using the iterative strategy compared to the default strategy. Only for a few cases a penalty occurs. But the improvement often is significant, e.g. less than 20% are needed in case of “c7552” and “s35932”. Especially for the larger benchmarks significant run time improvements were achieved.

3.2 Industrial Circuits

Finally, a set of industrial circuits from Philips has been considered, i.e. circuits that also contain buses and have up to 560k gates. We gave a 20 second CPU time limit to the SAT solver. The results are given in Table 8. As can be seen for these circuits only very few faults remain unclassified, i.e. only 204 for circuit “p565k” that consists of more than 565k gates. For the same benchmark TEGUS gives up on more than 340,000 faults (!) demonstrating how hard the circuit is to test.

Table 8: PASSAT for industrial benchmarks

Circ.	Ct	Red	Ab	Eqn	SAT
p88k	127084	2354	14	12186	9584
p565k	1203690	26426	204	5291	8155

4 Conclusions

For the first time an approach for SAT-based test pattern generation aiming at circuits that include tri-state elements has been shown. Due to recent advances in SAT-solving the approach is more robust than previous ones. Complete test sets for the circuits from ISCAS89 have been calculated. The influence of the variable selection strategy on the efficiency of test pattern generation is crucial. But applying an iterative approach to combine a fast and a robust strategy yields significant improvements. Large industrial circuits have been studied including buses. Also for circuits with more then 500k gates for almost all faults test pattern have been generated in very small run time.

References

[1] S. Cook. The complexity of theorem proving procedures. In *3. ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[2] R. Drechsler. *Advanced Formal Verification*. Kluwer Academic Publishers, 2004.

[3] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Trans. on Comp.*, 32:1137–1144, 1983.

[4] P. Goel. An implicit enumeration algorithm to generate test for combinational logic. *IEEE Trans. on Comp.*, 30:215–222, 1981.

[5] E. Goldberg and Y. Novikov. BerkMin: a fast and robust SAT-solver. In *Design, Automation and Test in Europe*, pages 142–149, 2002.

[6] A. Kuehlmann, M. Ganai, and V. Paruthi. Circuit-based Boolean reasoning. In *Design Automation Conf.*, pages 232–237, 2001.

[7] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on CAD*, 11:4–15, 1992.

[8] J. Marques-Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *9th Portuguese Conference on Artificial Intelligence (EPIA)*, 1999.

[9] J. Marques-Silva and K. Sakallah. GRASP – a new search algorithm for satisfiability. In *Int'l Conf. on CAD*, pages 220–227, 1996.

[10] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Comp.*, 48(5):506–521, 1999.

[11] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

[12] J. Roth. Diagnosis of automata failures: A calculus and a method. *IBM J. Res. Dev.*, 10:278–281, 1966.

[13] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, University of Berkeley, 1992.

[14] J. M. Silva and K. Sakallah. Robust search algorithms for test pattern generation. In *Int'l Symp. on Fault-Tolerant Comp.*, pages 152–, 1997.

[15] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Trans. on CAD*, 15:1167–1176, 1996.

[16] P. Tafertshofer and A. Ganz. SAT based ATPG using fast justification and propagation in the implication graph. In *Int'l Conf. on CAD*, pages 139–146, 1999.

[17] P. Tafertshofer, A. Ganz, and M. Henftling. A SAT-based implication engine for efficient ATPG, equivalence checking, and optimization of netlists. In *Int'l Conf. on CAD*, pages 648 – 655, 1997.