Experimental Studies on SAT-Based Test Pattern Generation for Industrial Circuits

Junhao Shi Görschwin Fey Rolf Drechsler Andreas Glowatz Jürgen Schlöffel Friedrich Hapke

Institute of Computer Science
University of Bremen
28359 Bremen, Germany

Philips Semiconductors GmbH
22502 Hamburg, Germany

drechsle@informatik.uni-bremen.de

Abstract

Due to the ever increasing size of integrated circuits classical methods for Automatic Test Pattern Generation (ATPG) reach their limits. On the other hand recent advances in algorithms to solve the Boolean Satisfiability (SAT) problem allow the application to large instances. This suggests to exploit modern SAT techniques for ATPG.
Here, we discuss the SAT-based ATPG tool PASSAT that is applicable to large industrial circuits. The performance of different SAT solvers is experimentally evaluated and the potential for problem specific heuristics is shown. Further experiments show that most of the faults can be classified very efficiently independently of the circuit size.

1. Introduction

Due to the exponentially increasing size of integrated circuits tools for computer aided design have to cope with problems of rapidly growing complexity. This is especially true in the area of *Automatic Test Pattern Generation* (ATPG). Here, often a complete system of up to several million gates has to be considered. Therefore classical algorithms for ATPG, such as FAN [FS83] reach their limits.
On the other hand algorithms for *Boolean Satisfiability* (SAT) have been dramatically improved in the recent past. This is due to the use of several advanced techniques in SAT solvers:
- the DLL procedure [DLL62],
- dynamic learning [MS99],
- efficient implementation techniques [MZZ+01], and
- robust search heuristics [GN02].

These modern SAT solvers have been applied with success to search problems in different areas, e.g. artificial intelligence or formal verification. Also preliminary results have been presented for SAT-based ATPG. These tools transform the problem of ATPG into a SAT problem. This SAT instance is then solved by a dedicated SAT solver. The solver either proves unsatisfiability, i.e. the fault is redundant, or generates a satisfying assignment, i.e. a test pattern.
Here, we compare different SAT solvers with respect to ATPG and present experimental results for an improved version of the tool PASSAT [SFD+05] on large industrial circuits. So far mainly results for the ISCAS benchmarks and first promising results for larger circuits have been published. The modern SAT solver Zchaff [MZZ+01] is the core engine of PASSAT. A problem specific variable selection strategy is applied during the search process. Furthermore, structural information is embedded into the SAT problem in form of additional implications to aid the solver. Compared to the previous version of PASSAT the memory management during generation of the SAT instance for a given fault has been improved. As a result the SAT problem can be generated much more efficiently.

2. SAT-based ATPG

In the following we briefly review basic notations and definitions for the tool PASSAT [SFD+05]: the D-algorithm, its application to SAT-based test pattern generation in TEGUS [SBS96], and the improvements of PASSAT over TEGUS. The technique to handle circuits with tri-state elements and the use of unknown values is introduced. Then, different decision schemes for variable selection are explained.

A. D-Algorithm

The D-Algorithm [Rot66] 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 [SBS96]. Besides the structural information, also additional implications as proposed in the D-algorithm are encoded in the SAT instance. Additionally, the initial circuit is transformed into a circuit of AND gates to benefit from the resulting simplification during generation of the SAT problem and fault simulation.

This conversion of the test problem into an equation in *Conjunctive Normal Form* (CNF) is also facilitated by PASSAT. Then, the integrated SAT solver is used to solve the generated CNF formula.

B. Advances in SAT

Recently, SAT solvers based on the DLL procedure [DLL62] have been greatly improved due to three main techniques:

- Conflict analysis [MS99] 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 [MMZ+01] corresponds to implications carried out in classical ATPG. But SAT solvers only use simple implications to maintain efficiency.
- Variable selection strategies [GN02] 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 [MMZ+01]. 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.

C. Four-Valued Logic

So far only circuits working with Boolean logic have been considered. But due to environment restrictions in practice often three-state elements and unknown values have to be considered. Both is handled by PASSAT. Instead of only encoding each signal with one Boolean variable, two variables are used. This allows to encode the four values '0', '1', 'Z' and 'U'.

D. Variable Selection

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 [Goe81] 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 [MMZ+01] is applied to benefit from the feedback of conflict analysis and current position in the search space.

Restricting the variable selection to fanout gates only has been proposed in FAN [FS83] 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. If fanout gates and primary inputs are selected instead of only primary inputs, conflict detection due to local inconsistencies becomes possible. Thus, internal conflicts are detected with less effort. PASSAT applies the VSIDS strategy to select fanout gates or primary inputs.

The experiments in Section 4 show that some heuristics are quite robust, i.e. they can classify all faults, while others are fast for some faults but abort on others. Therefore an iterative approach turned out to be most 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 ensures, that a fast test pattern generation is carried out where possible, while a more sophisticated search is done for the remaining faults.

3. Memory Management

The main improvement of the advanced version of PASSAT over the one presented in [SFD+05] is the memory management.

The naive way to register clauses in the clause database of the solver is to produce them one by one while traversing the circuit structure. When clauses are generated for a particular gate the corresponding memory has to be allocated.

More efficient is the allocation of large blocks of memory for the storage of clauses as implemented in the new version of PASSAT. When storage space for clauses is needed, a block of 64KB is allocated. Clauses in this block are stored in dynamic lists, the management of these lists is done by PASSAT instead of the operating system. While traversing the circuit the clauses are stored in the current memory block. If no more space is available a new block is allocated.

This memory is not freed between runs for different faults, i.e. in successive runs often no additional memory has to be allocated.

In summary, only a few system calls for memory allocation are necessary. This memory management does not directly improve the SAT solving process, but reduces the time needed for CNF generation.

4. Experimental Results

So far only experimental data for the ISCAS benchmarks and first promising results for industrial circuits have been reported for PASSAT [SFD+05]. Further experimental studies on large industrial benchmarks from Philips Semiconductors GmbH are presented in the following. These studies open new insights in properties specific to SAT-based ATPG. A set of industrial circuits has been used for benchmarking. The name of each circuit indicates the number of gates of the corresponding circuit, e.g. circuit p44k has more than 44.000 gates. The experiments were carried out on an AMD Athlon 3300+ (2.2GHz, 1GB, Linux).

A. Different SAT solvers

In the first series of experiments different SAT solvers have been applied: Berkmin (v5.61) [GN02], Grasp (v2004) [MS99], Minisat (v1.13) [ES03], Walksat (v45) [JT96], and Zchaff [MMZ+01]. The SAT problem has been written into a data-file, then the different SAT solvers were applied to this problem. For all SAT solvers the default heuristics for variable selection, clause

Table 1: Run times of different SAT solvers

Method / Fault	Time(s)						
	Zchaff (inputs)	Zchaff (all)	Berkmin	Minisat	Grasp	Walksat	Result
p49k, no. 1	14.78	Abort	6078.00	101.00	Abort	Too large	SAT
p49k, no. 2	14.54	Abort	6215.00	100.00	Abort	Too large	SAT
p49k, no. 3	0.85	Abort	Abort	644.00	Abort	Too large	SAT
p49k, no. 4	0.88	Abort	Abort	891.00	Abort	Too large	SAT
p44k, no. 1	1.38	0.11	0.29	0.15	24.95	10.08 undec.	UNSAT
p44k, no. 2	0.65	0.29	90.63	3.19	4361	10.06 undec.	SAT

Table 2: Blockwise clause allocation

Circuit	No MM		MM		Speed-up
	Eqn	SAT	Eqn	SAT	
c0432	1.48	1.64	1.19	1.82	1.04
c0499	5.03	13.00	3.57	11.10	1.23
c0880	1.36	0.31	1.04	0.36	1.19
c1355	8.97	9.25	6.36	9.51	1.15
c1908	9.08	16.30	7.53	16.30	1.07
c2670	6.83	7.83	5.24	8.09	1.10
c3540	39.60	16.50	31.10	14.70	1.22
c6288	691	643	697	638	1.00
c7552	50.50	24.30	41.20	26.70	1.10
s01494	1.38	0.52	1.03	0.52	1.22
s05378	4.30	1.48	3.42	1.47	1.18
s09234	7.18	2.48	5.71	2.48	1.18
s13207	49.90	45.60	40.00	43.60	1.14
s15850	61.30	20.10	51.30	20.90	1.13
s35932	31.70	5.74	28.30	6.26	1.08
s38417	83.40	34.20	68.80	35.70	1.13
s38584	43.30	16.40	36.50	17.50	1.11
p88k	16142	4726	11215	4639	1.32

deletion etc. have been used. In Zchaff the branching variables have been restricted to inputs only in one run. All variables were allowed for branching in a second run. The results are shown in Table 1. The SAT solver Walksat that is based on random search can not handle the problem or takes too long to return the result. The other SAT solvers, that are based on the DLL procedure are similar in performance. When the other SAT solvers are ranked by considering the default strategy, the number of abortions, and the run times, the order would be: Minisat, Berkmin, Zchaff, Grasp – this corresponds to the publication dates, i.e. Minisat is the newest solver. On the other hand Zchaff performs best, when only input variables are allowed for branching. This shows the potential for improvements that can be gained by adopting problem specific heuristics.

B. *Improved Clause Allocation*

An important improvement over the earlier version of PASSAT is the memory management as introduced in Section 3. The speed-up gained from the memory management can be seen in Table 2. All faults of the given benchmarks have been considered. Given are the run times to generate the problem instances in CNF and the run times to solve all the SAT instances in columns Eqn and SAT, respectively. Already for the small ISCAS benchmarks a significant speed-up was achieved. The time for SAT solving only varies slightly. The speed-up is due to the faster generation of the CNF descriptions. The same effects as for the small circuits are also observed for the industrial benchmark p88k. For such large circuits an improvement in generating the CNF is of even more importance. But also speeding up the SAT solver itself is desirable.

Table 3: Time to classify faults

Circuit	Time for classification				
	<0.1	0.1-1	1-10	10-20	abort
p44k	0	57	19	0	0
p49k	0	0	385	0	1581
p80k	9	207	0	0	0
p88k	106	167	7	0	0
p177k	137	119	58	5	13
p565k	961	440	8	0	0
p1330k	2053	486	21	34	8

C. *Run Time Spectrum*

Table 3 shows the run time spectrum for sets of faults. The faults considered in this experiment were aborted by the industrial FAN-based ATPG tool AMSAL from Philips Semiconductors GmbH using the default settings. Therefore these faults can be considered as being hard to classify. The table shows the number of faults that were handled within a given amount of time. The run time comprises the time for generating and solving the SAT problem. Each column gives the number of faults that were handled within a given time interval. After 20 seconds the process was aborted. In most cases the faults were classified very efficiently. The only exception is circuit p49k. In this case a large number of faults could not be handled within 20 seconds. For all other circuits most of the faults were classified within 1 second. Especially remarkable is the largest circuit p1330k, where 79% of the faults that are hard for classical ATPG were classified within less than 0.1 second.

D. *Decision Strategies*

Based on the previous results adapted decision strategies for variable selection have been investigated. The four decision strategies introduced in Section 2.D are

Table 5: All stuck-at faults

Circuit	cnt	red	ab	Eqn	SAT	All	Mem	Cls	Var
p44k	61230	823	0	17821	30797	49515	171.7M	330770	102085
p77k	126338	0	0	1156	334	1491	211.4M	815921	240910
p80k	176159	5	9	7420	5591	13012	279.6M	1308017	396964
p88k	126929	2354	169	2985	9044	12030	231.2M	499783	150628
p99k	131913	759	4548	4364	36965	41327	526.4M	522160	160026
p565k	1175605	26372	28343	1456	3073	4546	691.6M	3378721	1039140

Table 4: Decision strategies (on p49k)

Heuristic	cnt	red	ab	Eqn	SAT	All
Set A						
Input+All	187	68	1	1288	795	2084
All	0	1	255	1075	2770	3847
Input	187	67	2	1184	601	1787
Fanout	0	0	256	1295	1272	2568
Set B						
Input+All	0	0	171	26	6798	6826
All	0	0	171	28	8713	8745
Input	0	0	171	27	7463	7496
Fanout	0	0	171	25	4749	4781

considered. Table 4 shows the experimental results for two sets of faults (A and B) of circuit p49k. Shown are the number of faults (cnt), the number of faults classified as redundant (red) and the number of aborted faults (ab). The run times for creating (Eqn) and solving (SAT) all CNF instances as well as the total run time (All) are given. Set A contained only a few “hard” faults, i.e. faults that are difficult to classify using the SAT approach. In contrast all the faults in Set B were too hard to be classified. In general the strategies *Input* or *Fanout* were the fastest. But only the default strategy of Zchaff was robust enough to classify one of the redundant faults efficiently. This leads to the good results of the combined strategy *Input+All*. Branching only on the inputs for a short time efficiently filters and classifies “simple” faults. Afterwards the “harder” faults are classified by the default strategy that branches on all variables.

E. Industrial Benchmarks

The individual techniques that were empirically validated in the previous sections yield a robust SAT-based ATPG tool. The performance of PASSAT on industrial benchmarks is shown in Table 5. The data columns are the same as explained previously. Furthermore, the peak memory requirements (Mem), the number of clauses (Cls) and variables (Var) are reported. As can be seen all faults of most circuits are classified efficiently. Even for the largest circuit p565k more than 97% of the faults were classified. Run times and memory requirements were moderate in all cases. In our experiments no direct relation between run time and circuit size can be observed. The run time is subject to the inherent complexity of the individual fault classification problems circuit. In contrast in the experiments the memory requirements are directly related to the circuit size.

5. Conclusions

An improved version of the SAT-based ATPG tool PASSAT has been presented. Techniques to improve the performance of CNF generation and SAT solving have been proposed and empirically validated. The overall performance of the tool even on large industrial benchmarks is remarkable.

References

[DLL62] M. Davis, G. Logeman, D. Loveland: A machine program for theorem proving, Comm. of the ACM, Vol. 5, No. 7, pp. 394-397, 1962.

[ES03] N. Eén, N. Sörensson: An extensible SAT solver. In. Proc. of SAT 2003, LNCS, Vol. 2919, pp. 502-518, 2004.

[FS83] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. IEEE Trans. on Computers, Vol. 32, No. 12, pp. 1137-1144, 1983.

[GN02] E.Goldberg, Y.Novikov: BerkMin: A fast and robust SAT solver. In Proc. of DATE, pp. 142-149, 2002.

[Goe81] P. Goel. An implicit enumeration algorithm to generate test for combinational logic. IEEE Trans. on Computers, Vol. 30, No. 3, pp. 215-222, 1981.

[JT96] D. S. Johnson, M. A. Trick, ed.: Appendix summarizing results of walksat on challenge instances, in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26, AMS, 1996.

[MMZ+01] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik: Zchaff: Engineering an efficient SAT solver. In Proc. of Design Automation Conference, pp. 530-535, 2001.

[MS99] J. P. Marques-Silva, K. A. Sakallah : GRASP: A search algorithm for propositional satisfiability. In IEEE Trans. on Computers, Vol. 48, No. 5, pp. 506-521, 1999.

[Rot66] J. Roth. Diagnosis of automata failures: A calculus and a method. IBM J. Res. Dev., Vol. 10, pp. 278-281, 1966.

[SBS96] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. IEEE Trans. on CAD, Vol. 15, No. 9, pp. 1167-1176, 1996.

[SFD+05] J. Shi, G. Fey, R. Drechsler, A. Glowatz, J. Schlöffel, F. Hapke: PASSAT: Efficient SAT-based test pattern generation for industrial circuits. In Proc. of ISVLSI, pp. 212-217, 2005.