Studies on Integrating SAT-based ATPG in an Industrial Environment

Daniel Tille Stephan Eggersglüß Görschwin Fey Rolf Drechsler
Institute of Computer Science, University of Bremen
28359 Bremen, Germany
{tille,segg,fey,drechsle}@informatik.uni-bremen.de

Andreas Glowatz Friedrich Hapke Jürgen Schlöffel
NXP Semiconductors GmbH
21147 Hamburg, Germany
{andreas.glowatz,friedrich.hapke,juegen.schloeffel}@nxp.com

Abstract

Due to ever increasing design sizes, more efficient tools for Automatic Test Pattern Generation *(ATPG) are needed. Recently, SAT-based approaches for test pattern generation have been shown to be very efficient even on large industrial circuits. But these SAT-based techniques are not always superior to classical ATPG approaches. An integration of SAT-based engines into the classical ATPG flow can improve the overall performance.*

In this paper we present a first approach to integrate a SAT-based engine into the industrial ATPG environment of NXP Semiconductors. Experimental results for large industrial benchmark circuits are presented that show the improvements achieved by the integration.

1 Introduction

The complexity of circuits increases rapidly. According to Moore's law the number of gates doubles every 18 months and this trend is going to last for at least another decade. As a result the size of problem instances that have to be handled by *Computer Aided Design* (CAD) tools also increases. One particular step in the design flow is the post production test. This test ensures the functional correctness of a chip and is therefore an important step in ensuring high quality products. In practice the post-production test is carried out by applying input stimuli – so called "test patterns" – to the circuit and controlling the output response with respect to its correctness. The test patterns are computed during *Automatic Test Pattern Generation* (ATPG). Therefore ATPG tools also have to cope with the increasing size of problem instances.

A number of sophisticated algorithms for ATPG have been proposed in the past. Usually, a fault model is used to model physical defects at the functional level in a Boolean representation of the circuit. Then, the space of input stimuli is searched to find a test pattern for a particular fault. Among the fault models the *Stuck-At Fault Model* (SAFM) is most frequently used in practice. The D-algorithm [10] was the first ATPG algorithm to carry out an efficient backtrack search steered by structural information from the circuit. The algorithms PODEM [4] and FAN [3] improved the branching heuristics to make the search more efficient. Using structural information to apply global implications during the search has been proposed for SOCRATES [12]. The more powerful recursive learning [7] and the integration with the FAN-algorithm have been proposed by the tool HANNIBAL [6]. All of these algorithms directly work on the circuit structure.

In contrast there also exist approaches based on Boolean Satisfiability (SAT) [8, 15, 16, 13]. These work on a representation of the problem instance in *Conjunctive Normal Form* (CNF). Early SAT-based approaches were not able to handle industrial instances. But the recent advances in algorithms for SAT solving [9, 1] made the application to ATPG feasible. The combination of these advances and structural knowledge into an ATPG tool provide an efficient and robust ATPG engine which has been shown by the tool PASSAT [13, 14].

Of course, it cannot be expected that a single SAT-based engine is faster on all ATPG instances than the sophisticated classical approaches. In industrial tools the combination of different techniques like random simulation, learning, and others is crucial to achieve robustness. To benefit from a powerful SAT-based engine in such an environment, the combination with other ATPG approaches has to be considered. But so far no tight integration of a SAT-based engine into an industrial framework has been proposed.

Here, we present the integration of a SAT-based engine into the pre-identification phase of the industrial ATPG environment from NXP Semiconductors. The

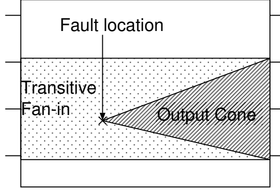


Figure 1. Extraction of the influenced circuit parts

industrial tool provides state of the art classical ATPG techniques, while the SAT-based engine combines advanced SAT techniques with problem specific improvements for ATPG. Which engine will be faster on a specific fault for a given circuit cannot be predicted. Therefore the parameters to control the overall flow are most important. A two phase approach is used, where the SAT-based engine specifically targets faults that are hard for classical approaches. In the following, the SAFM in combinatorial circuits is considered.

The paper is structured as follows: SAT-based ATPG is reviewed in the next section. The general flow in an industrial ATPG environment is explained in Section 3. The integration of a SAT-based engine into such an environment to handle hard-to-detect faults is discussed in Section 4. Experimental results that show the feasibility on industrial benchmark circuits are reported in Section 5. Finally, conclusions and the focus of future work are presented.

2 SAT-based ATPG

In this section the application of SAT for ATPG is explained. The SAFM is reviewed. Then, the basic transformation of an ATPG instance into a SAT instance is presented. Also the improvements due to problem specific knowledge and SAT techniques are briefly discussed.

The SAFM is a static fault model. One line in the circuit is considered to be stuck at the constant value 0 or 1. Then, the value of this line does not depend on the primary input anymore. A particular stuck-at fault in a given circuit can easily be modeled. When comparing the faulty and the non-faulty circuit, a test pattern produces different output values at least at one output. A particular stuck-at fault is called *testable* iff such a test pattern exists. Otherwise the fault is called *untestable*.

Modern SAT solvers work on the problem represented as a CNF. A CNF is a set of clauses, a clause is a set of literals and a literal is a variable or a negated variable. The CNF is satisfied under a given assignment for the variables iff all clauses are satisfied. A clause is satisfied iff at least one literal is satisfied. A literal is satisfied, iff the variable is not negated and has assigned the value 1 or the variable is negated and has assigned the value 0. If there is a satisfying assignment, the CNF is called *satisfiable*. Otherwise the CNF is called *unsatisfiable*.

The transformation of an ATPG instance into a CNF representation has been explained in e.g. [8]. A variable is assigned to each line in the circuit. Then, the functionality of a single gate is described by a set of clauses. The conjunction of the clauses for all gates is a CNF representation for the circuit. This representation is not unique and therefore allows for improvements. The number of clauses in the CNF is linear in the number of gates in the circuit.

Based on this transformation, the creation of a CNF for a given ATPG problem is explained in the following. Consider the circuit in Figure 1. The fault location is marked. First, the output cone of the fault site is marked by a depth first traversal on the circuit. This determines all outputs that may be influenced by the fault. The transitive fan-in of these outputs influences the detection of the fault and must be contained in the SAT instance. This knowledge is used to create the SAT instance: A faulty version and a fault free version of the circuit are modeled. Different signal values may only occur in the fault site's output cone. Therefore the transitive fan-in is shared between both versions in order to reduce the size of the SAT instance. Finally, the outputs of the fault free version and the faulty version are compared by XOR-gates. An additional constraint ensures, that at least one XOR-gate assumes the value 1. In summary, this SAT instance is only satisfiable, iff a test pattern is found that yields a wrong output value if the fault is present. The transformation of this instance into CNF is done as explained above.

Classical approaches for ATPG are tuned for the particular problem and can exploit structural information that is present in the circuit. In contrast a SAT solver is a general purpose engine for Boolean satisfiability that heavily relies on efficient lerning techniques. Structural information is lost during the problem transformation. Partially, embedding this information accelerates the SAT search.

One improvement is the use of dedicated variable selection strategies. The branching points have been reduced to primary inputs in the PODEM algorithm [4] and to fanout points in the FAN algorithm [3]. This is also beneficial for SAT-based ATPG. Here, a combination of standard SAT decision strategies together with a restriction of the branching points is successful [13].

Embedding structural information in the SAT instance is a second improvement. Here, implications are coded into the CNF by adding some clauses. In particular, the observations used by the D-algorithm have been found to be valuable [15]. For example, a faulty value can only be propagated along a gate, if it is propagated along at least one succeeding gate. By adding such constraints to the CNF also structural information is available during SAT solving.

For an industrial application Boolean values are not sufficient to model the ATPG problem. Due to uncontrollable inputs, unknown values must be considered using the value U. Moreover, a Z-value is used for the high impedance state of tri-state elements. As a result a four-valued logic over $\{0, 1, U, Z\}$ has to be consid-

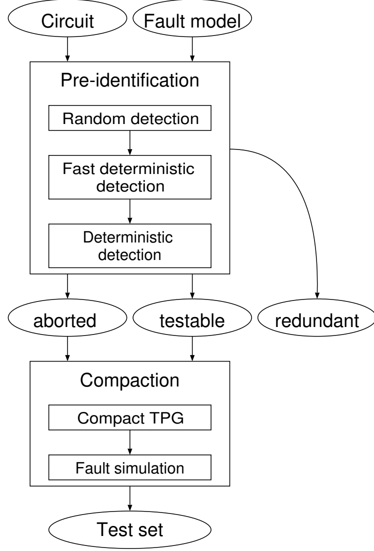


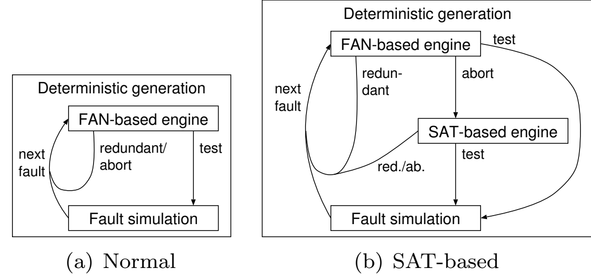
Figure 2. ATPG flow

ered. This can be efficiently handled by a SAT-based ATPG tool [13].

3 Industrial Environment

In principle it is sufficient to iterate over all faults with respect to the given fault model and generate a test pattern for each of them. But in an industrial environment like that of NXP Semiconductors this is not sufficient to retrieve a robust system. In the following only the overall flow in the system will be briefly reviewed to explain the problems that occur during the integration of a SAT-based engine. The particular improvements in the highly optimized testing tool AMSAL that was used for the experiments cannot be explained in detail. For a more detailed presentation on ATPG systems in general we refer to e.g. [5].

The major steps of the ATPG flow are shown in Figure 2. The inputs for the system are the circuit and the fault model to be considered. Here, the SAFM is assumed. Two main steps are carried out: the pre-identification phase to classify faults and the compaction phase to generate a small test set. The goal during pre-identification is the classification of faults. Here, three engines are used. First, random fault detection is applied to filter out easy-to-detect faults. For the remaining faults a fast deterministic fault detection is done. Finally, deterministic fault detection with increaded resources is applied to classify hard-to-detect faults. As a result four classes of faults are generated: untestable faults, easily testable faults, hard testable faults and non-classified aborted faults. Untestable and easy-to-detect faults are not further considered. Only the remaining testable faults are further treated in the compaction step. In the compaction step test patterns that detect as many faults as possible are generated. This is necessary because a small test set results in shorter test times during the post-production test.



(a) Normal (b) SAT-based

Figure 3. Deterministic fault detection

The main step considered here is the deterministic fault detection applied in the pre-identification phase. In this phase it is important to classify as many faults as possible. Only faults classified as testable are considered during the *compact test pattern generation*. Aborted faults are considered during fault simulation as well. Therefore untestable faults that were not classified, i.e. aborted, are an overhead in the compaction step. Testable faults that were not classified may not be detected by *compact TPG*.

The details of the deterministic fault detection are shown in Figure 3(a). Usually, a highly optimized FAN-based ATPG engine including learning techniques and a fault simulator are applied for deterministic test pattern generation. First, the FAN-based engine is used to classify a given fault. If a test pattern is produced by this engine, additional faults may be detected by the pattern besides the fault targeted in the first place. Therefore a fault simulator is used to calculate all faults detected by the test pattern. This consumes additional computation time, but often speeds up the overall process because other patterns can be removed from the fault list.

4 Integration

When integrating a SAT-based ATPG engine into an industrial environment, the goal is to improve the overall performance of the system. For improving the performance, two aspects have to be considered: the run time and the number of classified faults. The run time should be decreased. The number of faults that are classified by the system should be increased, i.e. the number of aborted fault classifications due to resource limits should be decreased. In the following we concentrate on integrating the SAT-based engine into the pre-identification phase because reducing the number of aborted faults in this step is beneficial for the succeeding compaction step as explained in the previous section.

The underlying problem is to determine how to interlace the engines. The integration of random simulation and deterministic algorithms is done by applying simulation in a fast pre-processing step. This ordering is also feasible when an additional SAT-based engine is available. Similar to a FAN-based approach the SAT-based engine needs time to generate the problem instance before solving can be started. This overhead

is much smaller when random simulation is applied. Therefore the integration into the deterministic test pattern generation between the FAN-based engine and fault simulation has been addressed.

A number of observations helps to set up the framework for the integration:

- There are faults that are easily classified by FAN while SAT needs a long run time and vice versa. This behavior is not predictable from the fault itself.
- A large number of faults can be classified efficiently using FAN.
- Often the SAT-based engine efficiently classifies untestable faults and faults that are hard for FAN, i.e. those faults where FAN needs long run times or aborts due to pre-defined resource limits.
- A SAT solver determines values for all inputs that are contained in the transitive fan-in of those outputs where the fault may be observed. This makes merging of multiple test-patterns during compaction difficult.
- The FAN-based algorithm directly runs on the circuit structure which is available in the system already.
- The SAT-based algorithm converts the problem into a CNF before starting the SAT solver. Therefore a larger overhead per fault is needed compared to FAN.

These observations lead to the conclusion that the SAT-based engine should be used to target those faults that cannot be classified by the FAN-algorithm within a short timeout. This reduces the overhead for initializing the SAT-based engine on faults that are easy to classify by FAN already. Then, the SAT-based TPG may classify additional faults which, in turn, helps to remove other faults from the fault list as well.

This leads to the framework shown in Figure 3(b) when a particular fault is targeted. The FAN-based engine is started at first with the default parameter set. If a test pattern is generated, fault simulation is carried out as usual and may identify additional faults as being testable by the same test pattern. Otherwise, the SAT-based engine is applied to classify the fault in a second step.

The experiments show that this combined approach classifies more faults with almost no overhead for the additional runs of the SAT-based engine.

5 Experimental Results

Experimental results are reported in this section. The proposed integration of a SAT-based engine into the industrial environment was applied to the NXP Semiconductors ATPG tool AMSAL. As SAT solver we used MiniSat [1]. AMSAL provides very compact production and field test patterns to raise the quality of large and complex digital circuits. Since 1986 AMSAL has been continuously developed and covers faults like stuck-at, bridges, gate- and path-delay transitions as well as Iddq faults. Together with the possibilities of a root cause analysis, layout based pattern generation and test point insertion with extremely high test compaction is performed, resulting in a minimum set of test patterns. As in SOCRATES [11] and HANNIBAL [6], the main ATPG engine of AMSAL is a highly optimized FAN-algorithm. The SAT-based engine is integrated into the system in a prototypic manner as explained in Section 4. The resource limits of the FAN-based algorithm were set to the default settings of AMSAL since these parameters have been determined on a large range of circuits. The time out for the SAT-based engine was set to 20 seconds per fault. This is a quite high run time limit, but the experimental results show that this is useful to classify very hard faults [13]. This saves run time afterwards. As a result, a loose coupling between the engines is achieved. All faults that would not be classified in the classical flow are targeted by the SAT-based engine afterwards. Thereby, more faults can be classified in total. The SAT-based engine runs in two steps: First the CNF is generated, then, the fault is classified; afterwards the CNF is completely dropped. This approach is acceptable in the present setting, because the SAT-based engine is only applied to aborted classifications that are "randomly" distributed. Therefore identifying structural overlapping of CNF instances is difficult. For all other settings in the flow, e.g. the number of random patterns to be simulated, the default parameters of AMSAL were used. All of the following experiments were run on an AMD Athlon XP 3500+ (2.2 GHz, 1 GByte, GNU/Linux).

Table 1. Pre-identification results

		FAN(de)		FAN(long)		SAT		FAN+SAT	
Circuit	Targets	ab.	time	ab.	time	ab.	time	ab.	time
p44k	64105	12	7:44m	0	7:46m	0	3:44h	0	7:57m
p77k	163310	0	0:24m	0	0:24m	0	0:36m	0	0:45m
p80k	197834	218	43:04m	79	44:06m	0	1:03h	0	43:23m
p88k	147742	195	12:21m	51	13:56m	0	15:55m	0	12:44m
p99k	162019	1398	9:04m	615	16:27m	0	13:28m	0	11:56m
p177k	268176	270	21:40m	60	22:01m	808	10:43h	9	27:27m
p462k	673949	1383	1:53h	928	1:58h	136	4:49h	1	1:57h
p565k	1026851	1398	2:35h	154	2:43h	0	2:38h	0	2:40h
p1330k	1516144	*aborted*		*aborted*		1	5:21h	*aborted*	

A set of industrial circuits (provided by NXP) was considered as benchmarks. These circuits have been found to be difficult cases for ATPG. The name of the circuit also gives information about the number of gates contained in the circuit, e.g. p88k means that the circuits consists of about 88,000 gates. The largest circuit p1330k contains more than 1.3 million gates. The number of faults that have to be considered for a circuit is even larger than the number of gates as can be seen in Table 1; Column 'Targets' gives the number of faults after the fault collapsing step.

Four different approaches are compared in the following:

- "FAN (de)": Using the FAN-based engine with default parameters as explained above.

- “FAN (long)”: Using the FAN-based engine with drastically increased resources, i.e. the backtrack limit set to 1024 and up to 5 seconds per fault.
- “SAT”: Using only the SAT-based engine, i.e. replacing the FAN-based engine by the SAT-based engine in the flow.
- “FAN + SAT”: Using the combined approach as explained in Section 4.

Table 1 shows the results of the pre-identification phase. For each approach the number of aborted fault classifications (column ‘ab.’) and the run time (column ‘time’) are given. Most critical are aborted faults because these may not be targeted adequately in the compaction step as explained in Section 3.

The number of faults aborted by the default approach FAN (de) is quite large. This number can be reduced by using FAN (long), where the resources are increased. However, there are still too many aborted faults. On circuit p1330k, the classification using FAN even aborts due to memory explosion, i.e. the circuit representation does not fit into the available main memory and the test pattern generation process aborts. Using the SAT-based engine, many circuits, where aborts occur at the FAN approach, are now fully testable.

However, regarding the run time, the usage of the SAT-based engine as a stand alone algorithm causes too much overhead (a CNF formula has to be created for each fault). Only three circuits are not fully testable with the SAT-based approach: While in p462k the number of aborts decreases drastically, in p177k more aborts occur. Actually, the representation of p1330k as SAT instance is compact enough to fit into the main memory, i.e. the entire test pattern generation process can be performed.

Regarding the FAN+SAT approach, the run time is similar to that of FAN (de). However, once again, the number of aborts decreases in this combined approach. This is possible because many easy-to-detect fault are quickly classified by FAN while difficult faults are classified by the SAT-based engine.

In summary, the combined approach is able to fully classify 6 out of 9 benchmarks while the resources needed remain similar to that of a classical approach.

6 Conclusions and Future Work

The integration of a SAT-based ATPG engine into an industrial environment has been shown. The reason for applying the SAT-based engine as a second deterministic ATPG step to aborted faults was explained in detail. Experimental results show the improved robustness achieved by the combination of classical ATPG algorithms with a SAT-based approach. Even on large industrial circuits that are hard to test the proposed combined approach performs better than classical engines alone.

Further research is going to address the reuse of learned information as a promising improvement [2]. Another main focus will be the integration of a SAT-based approach into the compaction step.

Acknowledgements

Parts of this research work were supported by the German Federal Ministry of Education and Research (BMBF) in the Project MAYA under the contract number 01M3172B. Furthermore, the authors would like to thank Junhao Shi for helpful discussions.

References

[1] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

[2] G. Fey, T. Warode, and R. Drechsler. Using structural learning techniques in SAT-based ATPG. In *VLSI Design Conf.*, pages 69–74, 2007.

[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 tests for combinational logic. *IEEE Trans. on Comp.*, 30:215–222, 1981.

[5] N. Jha and S. Gupta. *Testing of Digital Systems*. Cambridge University Press, 2003.

[6] W. Kunz. HANNIBAL: An efficient tool for logic verification based on recursive learning. In *Int'l Conf. on CAD*, pages 538–543, 1993.

[7] W. Kunz and D. Pradhan. Recursive learning: A new implication technique for efficient solutions of CAD problems: Test, verification and optimization. *IEEE Trans. on CAD*, 13(9):1143–1158, 1994.

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

[9] 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.

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

[11] M. Schulz, E. Trischler, and T. Sarfert. SOCRATES: A highly efficient automatic test pattern generation system. In *Int'l Test Conf.*, pages 1016–1026, 1987.

[12] M. H. Schulz, E. Trischler, and T. M. Sarfert. SOCRATES: A highly efficient automatic test pattern generation system. *IEEE Trans. on CAD*, 7(1):126–137, 1988.

[13] J. Shi, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schlöffel. PASSAT: Effcient SAT-based test pattern generation for industrial circuits. In *IEEE Annual Symposium on VLSI*, pages 212–217, 2005.

[14] J. Shi, G. Fey, R. Drechsler, A. Glowatz, J. Schlöffel, and F. Hapke. Experimental studies on SAT-based test pattern generation for industrial circuits. In *Int'l Conf. on ASIC*, pages 967–970, 2005.

[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, A. Ganz, and K. Antreich. Igraine - an implication graph based engine for fast implication, justification, and propagation. *IEEE Trans. on CAD*, 19(8):907–927, 2000.