Formal Analysis Techniques: A Basis for High-Quality Designs

(*Invited Talk*)

Stephan Eggersglüß*†, Rolf Drechsler*
*Institute of Computer Science, University of Bremen,
28359 Bremen, Germany
{segg, drechsle}@informatik.uni-bremen.de
†German Research Center for Artificial Intelligence (DFKI)

***Abstract*—Many problems of high complexity such as verification and test generation arise during the design and manufacturing flow of industrial circuits and processors. Solving these problems is often crucial to guarantee the absence of errors and eventually a high quality product. However, due to the increased complexity of today's circuits, classical methods often have problems to solve these problems. In this paper, we give an introduction to formal analysis techniques and show how the application of these techniques is able to increase the quality of the design for the particular problem of test generation.**

I. Introduction

Nowadays, the correctness of chip designs is of paramount importance. Especially since electronic devices are more and more part of safety-critical applications such as automotive or aerospace. Designers and manufacturing companies have to ensure that the design is free of errors or defects before shipping their products. However, *Moore's law* is still valid and the complexity of today's designs is steadily increasing. This also puts severe demands on the methods which should ensure the correctness. The efficiency of those methods has to keep up with the increased complexity of the design.

During chip design and manufacturing, different problems of high computational complexity arise at all stages of the design flow. Figure 1 shows a simplified design flow and introduces these problems. In order to verify that the design meets the requirements of the specification, *property checking* is performed – typically at a higher level of abstraction such as system level and RTL. If a property fails, the design has to be *debugged* and the error has to be fixed. This can be assisted by an analysis which pinpoints to potential fault locations.

During synthesis, *equivalence checking* verifies that the functionality of the design at a higher abstraction level corresponds to the functionality of the synthesized design at a lower abstraction level. *Test generation* for the post-production test which filters out defective devices is generally performed at gate level. If gate library information or layout information is available, *timing analysis* is conducted to verify the timing behavior of the chip. If the chip is found to be erroneous after manufacturing, *silicon debug* searches for the reason for the errors which have to be fixed.

All these problems have in common that the logic of the circuit or processor have to be considered during the solving process. Methods which address these problems have to be able to cope with the increased complexity of the logic or parts of the logic, e.g. a data path of a microprocessor.

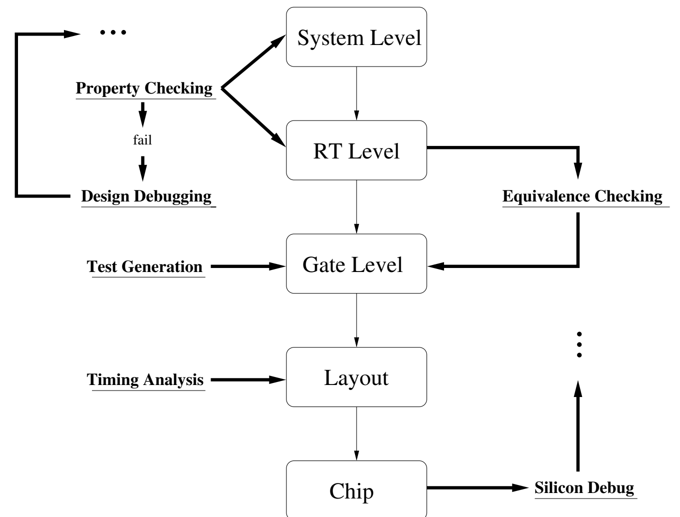

Fig. 1. Design flow

The fast and reliable solution of the arising tasks or problems is of high importance for the quality of the product. Additionally, the later an error is found in the design flow, the more expensive is resolving the error. One possibility is to use simulation-based techniques to solve these problems. However, due to the increased complexity, the overall search space is too large for a complete simulation-based approach. Therefore, these approaches are not able to provide sufficient quality.

Application-specific solutions which tackle the problem directly at the circuit level, e.g. structural ATPG algorithms, have problems with the increased complexity of today's designs. These approaches are not often not robust enough due to the heterogeneous structure of the problem.

A promising approach is the application of formal analysis techniques to resolve these problems. The next section gives an introduction to these techniques and describes their advantages.

II. Formal Analysis Techniques

Formal analysis techniques differ from application-specific techniques. These techniques do not work directly on the original problem representation but on a formal model. On this model, they are able to produce a formal (mathematical)

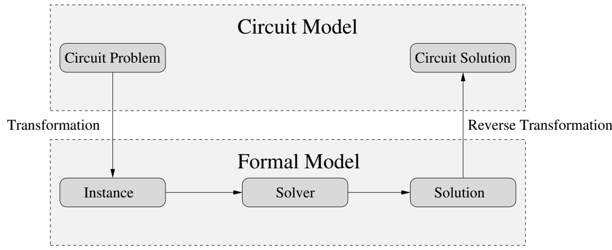

Fig. 2. Formal analysis - application flow

proof for the given problem. Common formal analysis techniques are for example based on *Binary Decision Diagrams* (BDDs), *Boolean Satisfiability* (SAT), *Satisfiability Modulo Theory* (SMT) or *Quantified Boolean Formulas* (QBF).

The application flow is shown in Figure 2. At first, the original circuit problem has to be transformed into a formal problem representation. Then, this problem is solved by a dedicated solver. The resulting (formal) solution is then transformed back to the circuit level.

The transformation seems to be an overhead at the first sight. However, the following advantages outweigh the costs of the transformation in most cases:

- Efficient solver technology - The restriction to a special formal model typically leads to a very homogeneous problem formulation. This allows for the application of efficient proof and learning techniques. Especially in the last decade, significant advances in formal solving technology have been made and effective solvers were constructed which are able to cope with problems of high complexity.
- Problem encapsulation - Solver technology is a very active field of research. Therefore, the efficiency of solvers is expected to increase in the next decade. The transformation of the original problem to a formal problem has to be done only once. The underlying solver can simply be substituted by an improved solver. By this, the application can easily benefit from the advances made in the field of solver technology.

The ongoing improvement of today's solver technology and the encapsulation of problem formulation and solving process provide a good basis for the development of high quality products. As a result, the methods which are responsible to ensure high quality have the potential to keep up with the increased complexity of modern designs. Examples for the application of formal analysis techniques in the design flow are given in [1], [2] (verification) [3], [4] (equivalence checking), [5], [6] (debugging), [7] (timing analysis), [8] (silicon debug) and [9], [10] (test generation).

In the following, we show by the example of test generation how formal analysis techniques can be integrated into the industrial test flow in order to increase the overall quality of industrial circuits and processors.

III. Application in Test Generation

In order to guarantee the absence of defects in manufactured chips, every chip is subjected to a post-production test to filter out defective devices. In this test, stimuli are applied which are generated by *Automatic Test Pattern Generation* (ATPG) algorithms. The task of an ATPG algorithm is to generate for each fault of a fault model, e.g. stuck-at or path delay, a test pattern which makes the fault observable at an output. The ATPG problem was proven to be NP-complete [11]. Classical ATPG algorithms typically work on a structural gate-level netlist. Due to the increased complexity of modern circuits, these algorithms have problems to cope with hard faults. This leads to the fact that an increasing number of faults remains undetected by the generated test set. As a result, the high-quality demands of the industry are compromised since faults which are not covered by the test set will not be detected. We will show how this quality gap can be compensated by the use of formal analysis techniques, i.e. by formulating the problem as a *Boolean Satisfiability* (SAT) problem.

SAT-based ATPG works differently to classical structural ATPG. In order to make use of the efficient SAT solving techniques, the ATPG problem has to be represented as a Boolean formula in *Conjunctive Normal Form* (CNF). The original problem which is based on a circuit model must be transformed into a formal model, i.e. a SAT instance in CNF (see Figure 2). Then, the SAT solver is applied to the SAT instance to solve the formula. Finally, the SAT solution must be re-transformed to the original circuit model, i.e. to obtain the test pattern.

A CNF Φ in m Boolean variables is a conjunction of n clauses. Each clause is a disjunction of literals. A literal is a Boolean variable (x) or its complement ($\overline{x}$). The CNF Φ is *satisfied* if all clauses are satisfied. A clause is satisfied if at least one literal of the clause is satisfied. The CNF Φ is said to be *unsatisfiable* iff no solution can be found that satisfies Φ. The task of a SAT solver for a given Φ is to find a satisfying assignment or to prove that no such assignment exists. Due to the homogeneity of the CNF, the SAT solver is able to use very efficient implication procedures and learning schemes.

In the following, the circuit-to-CNF transformation is briefly described. More information can be found in [12]. A Boolean variable is assigned to each connection in circuit $\mathcal{C}$. The CNF Φ_g for each gate $g \in \mathcal{C}$ is derived from the characteristic function which can be constructed using the truth table. The CNF $\Phi_{\mathcal{C}}$ representing the circuit's function is then constructed by the conjunction of the CNFs of all gates $g_1, \ldots, g_n \in \mathcal{C}$:

$$\Phi_{\mathcal{C}} = \prod_{i=1}^{n} \Phi_{g_i}$$

The CNF $\Phi_{\mathcal{C}}^F$ has to be extended by the fault-specific constraints Φ_F for generating a test for fault F. The fault-specific constraints include the fault site and the faulty circuitry. Additionally, structural information is used to encode the concept of D-chains into CNF [9].

More formally, a test for F is generated by evaluating the formula:

$$\Phi_{test}^F = \Phi_{\mathcal{C}}^F \cdot \Phi_F$$

If Φ_{test}^F is unsatisfiable, the fault is untestable. A test can be easily derived from the satisfying assignment if Φ_{test}^F is satisfiable.

A. Multiple-Valued Logic

For practical purposes, considering only the Boolean values 0 and 1 during test generation is insufficient. In order to be applicable in industrial practice, additional values are needed. The value Z describes the state of high impedance occurring in so-called tri-state elements, e.g. bus elements. The value U describes an unknown state caused by e.g. non-controllable inputs. This results in the following 4-valued logic [10]:

$$\mathcal{L}_4 = \{0, 1, U, Z\}$$

It is mandatory to consider these additional values during test generation. However, the SAT problem is only defined over Boolean logic and cannot directly applied to a problem represented in multiple-valued logic.

Therefore, a Boolean encoding has to be used to transform the multiple-valued problem into a Boolean problem. The value of each signal is represented by one Boolean variable in a purely Boolean circuit. In a multiple-valued circuit representation, one Boolean variable is insufficient. Two Boolean variables c, c^* are used to encode all four values and represent the signal's value. Clearly, the CNF representation is an overhead compared to the pure Boolean formulation. Typically, using $\mathcal{L}_4$ results in significantly increased run time. This gets worse when considering the more complex delay fault models. In order to make SAT algorithms feasible for industrial application, the following SAT instance generation flow has to be used.

B. SAT Instance Generation Flow

Test generation in an industrial context requires a multiple-valued logic $\mathcal{L}_4$. This logic has to be extended for delay test generation since two time frames t_1, t_2 are necessary. Therefore, the Cartesian product of all values in $\mathcal{L}_4$ is needed to represent all possible value combinations on a connection. This leads to the 16-valued logic $\mathcal{L}_{16}$. Additionally, delay tests can be of different quality [13]: non-robust and robust. Robust tests which are of higher quality require the modeling of static values. Incorporating the static values $S0, S1, SZ$ results in the 19-valued logic $\mathcal{L}_{19s}$ [14]:

$$\mathcal{L}_{19s} = \{S0, 00, 01, 10, 11, S1, 0U, 1U, U0, U1, UU,$$
$$0Z, 1Z, Z0, Z1, UZ, ZU, ZZ, SZ\}$$

In principle, $\mathcal{L}_{19s}$ can be used to model the circuit for SAT-based ATPG. However, logics with less values are generally more compact in their CNF representation than logics with more values. In fact, the exclusive use of $\mathcal{L}_{19s}$ would result in excessively large SAT instances and typically in run times too large for practical application. For solving this problem, the usage of several multiple-valued logics is proposed.

Typically, only a few connections in a circuit can assume all values contained in $\mathcal{L}_{19s}$ or $\mathcal{L}_4$. For example, there are only very few gates that are able to assume the value Z. Modeling all gates with $\mathcal{L}_{19s}$ or $\mathcal{L}_4$ would be correct but would also unnecessarily blow up the CNF representation since some values are known to be never assumed in the majority of all elements. These elements can be modeled in a multiple-valued logic with a smaller number of values.

TABLE I
MULTIPLE-VALUED LOGICS FOR CIRCUIT REPRESENTATION

Logic	Value set
$\mathcal{L}_{19s}$	$\mathcal{L}_{16} \cup$ {S0,S1,SZ}
$\mathcal{L}_{16}$	$\mathcal{L}_9 \cup$ {0Z,1Z,Z0,Z1,UZ,ZU,ZZ}
$\mathcal{L}_{11s}$	$\mathcal{L}_9 \cup$ {S0,S1}
$\mathcal{L}_9$	$\mathcal{L}_6 \cup$ {U0,U1,UU}
$\mathcal{L}_{8s}$	$\mathcal{L}_6 \cup$ {S0,S1}
$\mathcal{L}_6$	$\mathcal{L}_{4B} \cup$ {0U,1U}
$\mathcal{L}_{6s}$	$\mathcal{L}_{4B} \cup$ {S0,S1}
$\mathcal{L}_{4B}$	{00,01,10,11}
$\mathcal{L}_4$	{0,1,U,Z}
$\mathcal{L}_B$	{0,1}

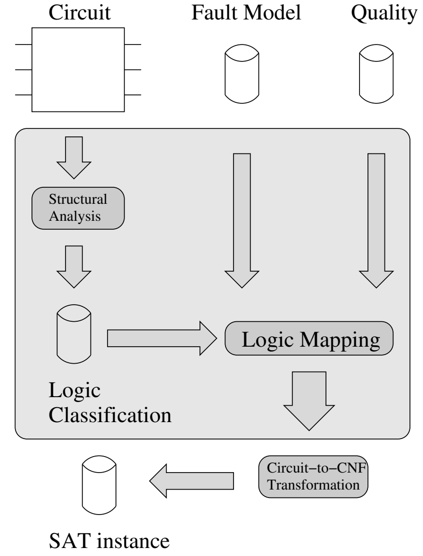

Fig. 3. SAT instance generation flow

Therefore, a structural analysis [14] is applied as a pre-processing step to determine for each gate which values can be assumed. This step has to be done only once and the results can be reused for each fault model and each quality level. The aim of the structural analysis is to exclude as many values as possible for each part of the circuit so that a multiple-valued logic with fewer values can be used. This typically promises a more efficient CNF representation of the circuit, i.e. SAT solvers generally need less time for smaller SAT instances.

Table I shows the set of logics identified to be necessary to model the correct circuit behavior for ATPG. Additionally, the corresponding value sets are denoted. Using the results of the structural analysis, it is decided for each gate during circuit-to-CNF transformation – according to the fault model used and desired quality – which logic has to be used for CNF generation.

The complete flow is illustrated in Figure 3. At first, the circuit is analyzed and the structural analysis is done for each element. Then, according to the fault model and the desired quality, this information is used to select the specific logic which has to be used for the circuit-to-CNF transformation. This procedure results in SAT instances with significant decreased size well suited for high-quality test generation for industrial circuits.

Additionally to the improved SAT instance generation flow, it has been shown that it is advantageous to include struc-

tural information into the search process itself to boost the search. However, this has to be done in a manner which does not modify the efficient SAT solving techniques, since the methodologies and data structures are highly optimized and very susceptible to changes. Therefore, an approach has been developed which uses the regular SAT solving techniques but work on dynamically growing SAT instance [15]. Additionally, the efficient learning schemes of a SAT solver can be leveraged by transferring learned information from one fault to subsequent faults [16]. Both modifications do not violate the principle of problem encapsulation that the actual SAT solver itself can be easily substituted.

C. Experiments

The described techniques were implemented and integrated into the industrial test environment of NXP Semiconductors as part of a 3-years research project with the ultimate goal to show the applicability in industrial practice. The approach was extensively evaluated on industrial circuits of practical size for several fault models, i.e. stuck-at, transition, path delay (non-robust, robust).

Table II shows the impact of the application of formal analysis techniques. The proposed approach *SAT* is compared to a highly optimized industrial structural ATPG approach (*FAN*). Column *%FC* gives the fault coverage which could be achieved and column *%FE* presents the fault efficiency. The fault efficiency is defined as the percentage of testable faults in faults not identified as untestable. Fault efficiency is commonly taken as a measurement to compare the effectiveness of ATPG algorithms. Column *%FC Inc.* gives the fault coverage increase of SAT compared to FAN. The run time improvement compared to FAN is given in column *Imp.F*. The run time of both approaches are in most cases comparable. However, if the run time of a pure SAT-based approach is too high, e.g. for p456k, there is the possibility that structural and SAT-based ATPG are combined [10] and SAT-based ATPG is only started for the hard faults.

The results show that the SAT approach has a very high fault efficiency being either 100% or between 99% and 100%. This signifies a considerable increase compared to FAN and shows the robustness of the proposed approach. Nearly all faults for which structural ATPG algorithms could not find tests due to the high complexity of the problem can be solved with formal analysis techniques. As a result, the fault coverage increases of up to 1.87% which is very important for the high quality demands of the industry.

Overall, the experiments clearly show the applicability and benefits of formal analysis techniques in industrial practice.

TABLE II
IMPACT ON FAULT COVERAGE / FAULT EFFICIENCY – TRANSITION

	FAN		SAT			
Circ.	%FC	%FE	Imp.F	%FC	%FE	%FC Inc.
p44k	55.15	99.40	0.96x	55.36	99.98	+0.21
p57k	96.36	98.71	0.97x	97.22	99.97	+0.86
p77k	34.46	67.62	1.59x	34.46	100.00	+0.00
p80k	94.86	98.58	1.25x	96.06	100.00	+1.20
p88k	92.33	97.56	2.04x	94.00	100.00	+1.67
p99k	89.91	95.95	1.86x	90.90	99.98	+0.99
p177k	76.13	96.56	0.36x	77.57	99.96	+1.44
p456k	84.17	94.43	0.08x	86.04	99.17	+1.87
p462k	57.68	97.48	1.01x	57.95	100.00	+0.27
p565k	94.81	99.44	1.38x	95.02	99.99	+0.21
p1330k	90.44	99.54	0.96x	90.57	100.00	+0.13

IV. CONCLUSIONS

In order to ensure the correctness of today's designs, many problems of high computational complexity has to be solved during the design and manufacturing flow. This paper has shown that formal analysis techniques provide a good basis to manage the increasing complexity of the designs and that these techniques have the potential to substitute or assist classical methods in industrial practice. As an example, it has been shown which efforts have to be undertaken to make the use of formal analysis techniques applicable in the field of industrial test generation. Experiments have shown that these techniques are able to increase the quality of the generated test set significantly due to the increased fault coverage.

REFERENCES

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, vol. 1579, 1999, pp. 193–207.

[2] M. R. Prasad, A. Biere, and A. Gupta, "A survey of recent advances in SAT-based formal verification," *Software Tools for Technology Transfer*, vol. 7, no. 2, pp. 156–173, 2005.

[3] E. I. Goldberg, M. R. Prasad, and R. K. Brayton, "Using SAT for combinational equivalence checking," in *Proceedings of Design, Automation and Test in Europe*, 2001, pp. 114–121.

[4] S. Disch and C. Scholl, "Combinational equivalence checking using incremental SAT solving, output ordering, and resets," in *Proceedings of the ASP Design Automation Conference*, 2007, pp. 938–943.

[5] G. Fey, S. Staber, R. Bloem, and R. Drechsler, "Automatic fault localization for property checking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 6, pp. 1138–1149, 2008.

[6] H. Mangassarian, A. G. Veneris, and M. Benedetti, "Robust qbf encodings for sequential circuits with applications to verification, debug and test," vol. 7, no. 59, pp. 981–994, 2010.

[7] M. Palla, J. Bargfrede, S. Eggersglüß, W. Anheier, and R. Drechsler, "Timing arc based logic analysis for false noise reduction," in *Proceedings of the International Conference on Computer-Aided Design*, 2009, pp. 225–230.

[8] Y.-S. Yang, B. Keng, N. Nicolici, A. G. Veneris, and S. Safarpour, "Automated silicon debug data analysis techniques for a hardware data acquisition environment," in *Proceedings of the International Symposium on Quality Electronic Design*, 2010, pp. 675–682.

[9] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1167–1176, 1996.

[10] R. Drechsler, S. Eggersglüß, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and D. Tille, "On acceleration of SAT-based ATPG for industrial designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1329–1333, 2008.

[11] H. Fujiwara and S. Toida, "The complexity of fault detection problems for combinational logic circuits," *IEEE Transactions on Computers*, vol. 31, no. 6, pp. 555–560, 1982.

[12] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, 1992.

[13] A. Krstić and K.-T. Cheng, *Delay Fault Testing for VLSI Circuits*. Kluwer Academic Publishers, Boston, MA, 1998.

[14] S. Eggersglüß, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and R. Drechsler, "MONSOON: SAT-based ATPG for path delay faults using multiple-valued logics," *Journal of Electronic Testing: Theory and Applications*, vol. 26, no. 3, pp. 307–322, 2010.

[15] S. Eggersglüß and R. Drechsler, "Efficient data structures and methodologies for SAT-based ATPG providing high fault coverage in industrial application," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2011, accepted.

[16] ——, "Increasing robustness of SAT-based delay test generation using efficient dynamic learning techniques," in *Proceedings of the IEEE European Test Symposium*, 2009, pp. 81–86.