VisSAT: Visualization of SAT Solver Internals for Computer Aided Hardware Verification

Robert Wille **André Sülflow** **Rolf Drechsler**

Institute of Computer Science
University of Bremen
29359 Bremen, Germany
{rwille,suelflow,drechsle}@informatik.uni-bremen.de

Abstract— *Today, many applications for formal circuit verification exist that rely on solvers for Boolean satisfiability (SAT). Usually, these applications use the SAT solver as a black-box. However, exploiting information on the internals of the solving process can speed-up the treatment of the verification task.*

In this paper, we present the tool VisSAT. VisSAT *provides insights into the internals of SAT solvers. Statistics of the internal solving process are collected and visualized on the circuit. By this, a verification engineer gets insight about potential bottlenecks for formal verification. This can be used to reconfigure the SAT solver or to improve the encoding.*

Keywords: Boolean satisfiability, hardware verification, visualization

1. Introduction

Verification gains an increasing amount of design costs in modern computer aided design. With increasing complexity and continuous demands for correctness, the application of formal methods in verification becomes indispensable [1], [2]. SAT solvers [3] are essential tools building the basis of many formal verification approaches like equivalence checking [4], [5], property checking [6], [7], [8], or automated debugging [9].

Typically, the following flow is thereby applied: The problem is encoded into an instance of Boolean satisfiability which is passed to the SAT solver. Then, the SAT solver returns either a satisfying assignment or proves that no such assignment exists. From this result, a solution of the verification problem is deduced. In this sense, the SAT solver is utilized as a black-box.

However, since the SAT problem is proven to be NP-complete [10], SAT solvers may need a significant amount of time to produce results for large instances. In contrast, advanced problem encodings [11], [12], [13], [14], adjusted decision heuristics [15], [3], or specialized propagation strategies [16] may accelerate the solving process significantly. But choosing the proper problem encoding or solver configuration, respectively, requires a deep technical understanding of how the SAT solver processes the instance. Internal data structures of a SAT solver give valuable information concerning the traversal as well as the structure of the search space. Unfortunately, such information is hard to extract.

In this work, we present *VisSAT*, a graphical interface which visualizes internals of SAT solvers applied to hardware verification tasks. *VisSAT* collects statistical information about the search process (e.g. the number of decisions on certain variables, the variables most frequently involved in conflicts, etc.). Afterwards, these statistics are evaluated and correlated to the overlying hardware verification problem. Therewith, *VisSAT* pinpoints the verification engineer to critical parts of the problem instance, e.g. hotspots with large occurrences of conflicts. This feedback helps to reconfigure the solver or alter the problem formulation accordingly. The explicit choice and the application of an optimization technique stays in the hands of the engineer.

While in previous work visualization of SAT instances already have been shown to be helpful [17], for the first time statistical information about the solving process itself is intuitively highlighted using *VisSAT*. Since additionally our approach gives insights on the circuit level (and not in terms of SAT variables and clauses) even verification engineers not familiar with SAT solvers are able to perform the respective heuristic and parameter tunings.

The remaining tool presentation is structured as follows: The next section briefly introduces Boolean satisfiability and provides a brief overview about the application of SAT in formal hardware verification. Afterwards, *VisSAT* is introduced in Section 3. By means of small examples, possible use cases are illustrated in Section 4. Finally the tool presentation is concluded in Section 5.

2. Formal Hardware Verification Using Boolean Satisfiability

In the recent years, Boolean satisfiability has become an established technique in many fields of formal hardware verification. Property checking, equivalence checking, as well as sophisticated techniques for automated debugging and for proving the fault tolerance of circuits are only some of the typical applications [6], [5], [9], [18]. The performance of the underlying SAT solver is crucial for all these applications.

In a typical flow, the application encodes the problem instance in *Conjunctive Normal Form* (CNF), i.e. a product-of-sum representation, and passes the CNF instance to a SAT solver. Then, the SAT solver returns either a satisfying solution (i.e. a consistent assignment of variables) or provides a proof of unsatisfiability for the CNF instance. Afterwards, the application maps the results of the SAT solver to the overlying verification problem.

For example, property checking proves the correctness of a property (e.g. given in some temporal logic like PSL [19]) on a design (e.g. given in a hardware description language like Verilog or VHDL). A property checker reads in the property and the design as input, translates the verification problem into an instance of Boolean satisfiability, and passes the instance to a SAT solver [6], [7], [8]. A property is either proven to be correct on the design (i.e. the instance is unsatisfiable) or a counterexample is returned (i.e. the instance is satisfiable).

Most SAT solvers accept a CNF as input. A CNF is a set of clauses where each clause is a set of literals and each literal is a propositional variable or its negation. While translating the hardware verification problem into a CNF, each n-bit signal is represented by a set of n SAT variables. The relation between the circuit signals and the SAT variables additionally is stored in internal data structures. For example, a 32-bit output signal of an adder corresponds to 32 propositional SAT variables. A satisfying model, i.e. a set of consistent assignments to the SAT variables in the SAT instance, can easily be mapped back to assignments on signals and by this to the original problem instance.

The underlying search process of the SAT solver remains thereby as a black-box process for the application and, by this, for the verification engineer. However, depending on the complexity of the verification problem, the run time of the SAT solver may range from a few seconds to up to several hours or even days. Having knowledge about the internal search procedure (e.g. about decisions or conflicts on variables) enables the verification engineer to adjust the parameters of the SAT solver or to improve the encoding of the problem instance in order to gain a speed-up. Moreover, the information is also worthwhile to analyze the progress of the verification.

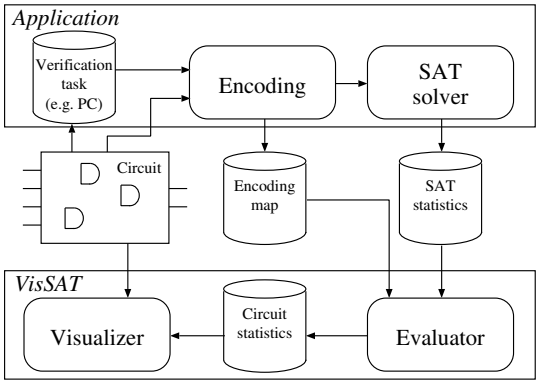

Fig. 1: Architecture of VisSAT

3. The VisSAT Tool

VisSAT collects statistical information about the solving process of a SAT solver and correlates this data to the overlying hardware verification problem. Accordingly, the tool is composed of two main components as illustrated in the lower part of Fig. 1[1].

The evaluator collects and processes statistical data of the solving process. The number of decisions on certain variables, the number of conflicts a SAT variable is involved, and further statistics provided by the SAT solver are thereby obtained[2]. Afterwards, these statistics are mapped to the respective hardware components of the circuit. Therefore, an encoding map is used to store which circuit signal is represented by which SAT variables. For example, if an one bit signal s is represented by a SAT variable x_{10} and, additionally, x_{10} was involved in four conflicts, then four conflicts are assigned to s. For signals representing multiple bits, the sum over all corresponding SAT variables is assigned to s. All these assignments are finally stored in a container.

The visualizer maps this data to a register transfer level or gate level schematic of the overlying circuit and, finally, visualizes both, the circuit and the obtained statistics. The visualization engine *RTLvision Pro* [20] is utilized which calculates a fitting layout that can be rendered. *RTLvision Pro* further allows cross-probing of the results to the source code of a design. The statistics are visualized by different color codes and, in order to obtain concrete values, tooltips. For example, if the conflict statistic is displayed, signals whose corresponding SAT variables were often involved in conflicts are highlighted in red, signals whose corresponding SAT variables were only occasionally involved in conflicts are highlighted in yellow, and signals whose corresponding SAT variables were never involved in a conflict are highlighted in green. Other statistics (e.g. the number of decisions) are displayed in a similar way.

[1]The upper part of Fig. 1 illustrates the SAT-based verification flow as described in Section 2.

[2]In this work, we extended MiniSat [3] to provide these statistics.

Using this visualization, the verification engineer is pinpointed to critical parts of the problem instance, e.g. hotspots with a large occurrence of conflicts. This feedback enables conclusions on why a verification tasks is hard to solve using SAT engines. The next section illustrates this by means of examples.

4. Use Cases

4.1 Using Information on Conflicts

Fig. 2 shows the distribution of conflicts occurred while solving a property checking instance. More precisely, the correct behavior of a multiplication in an ALU circuit was verified. The visualization shows the considered circuit highlighting the number of conflicts.

The coloring intuitively differentiates parts with a large number of conflicts (highlighted in red) from parts with a smaller number of conflicts (highlighted in yellow) or from parts without conflicts (highlighted in green), respectively. As indicated by the red signal, the multiplier module frequently causes conflicts. With this information, the designer can apply changes to the SAT solver parameters (e.g. preferring signals of the multiplier in order to address all these conflicts first during the solve process) or modify the design (e.g. replacing the multiplier by shifters), respectively.

4.2 Using Information on Decisions

Fig. 3 shows information on the number of decisions made for each signal while checking the correct behavior of the ADD instruction of another ALU. As expected, most decisions have been made for the output of an adder that was also involved in many conflicts, too. Thus, decisions on the output of the adder should have higher priority than on signals in the fan-in of the adder.

Moreover, decisions have been made for signals that do not influence the output of the property check, e.g. on the output of the subtraction module. In order to avoid this and, thus, to improve the solving process either, decisions on such signals can be deactivated or the verification model itself can be simplified.

5. Conclusions

In this paper, we presented *VisSAT*, a tool for the visualization of SAT solver internals within computer aided hardware verification. *VisSAT* collects statistics of the internal solving process and visualizes them on the considered circuit structure. By this, verification engineers are pinpointed to critical parts of the problem instance. This can be used to reconfigure the SAT solver or to improve the encoding. The application of VisSAT was illustrated by two use cases.

6. Acknowledgments

We thank Concept Engineering, in particular Gerhard Angst and Lothar Linhard, for providing us with the *RTLvision Pro* tool. This work was supported in part by the European Union (project DIAMOND, FP7-2009-IST-4-248613).

References

[1] R. Drechsler, *Formal Verification of Circuits*. Kluwer Academic Publishers, 2000.

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

[3] N. Eén and N. Sörensson, "An extensible SAT solver," in *SAT 2003*, ser. LNCS, vol. 2919, 2004, pp. 502–518.

[4] D. Brand, "Verification of large synthesized designs," in *Int'l Conf. on CAD*, 1993, pp. 534–537.

[5] S. Huang and K. Cheng, *Formal Equivalence Checking and Design Debugging*. Kluwer Academic Publisher, 1998.

[6] 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. LNCS, vol. 1579. Springer Verlag, 1999, pp. 193–207.

[7] K. Winkelmann, H.-J. Trylus, D. Stoffel, and G. Fey, "Cost-efficient block verification for a UMTS up-link chip-rate coprocessor," in *Design, Automation and Test in Europe*, vol. 1, 2004, pp. 162–167.

[8] M. Nguyen, M. Thalmaier, M. Wedler, J. Bormann, D. Stoffel, and W. Kunz, "Unbounded protocol compliance verification using interval property checking with invariants," *IEEE Trans. on CAD*, vol. 27, no. 11, pp. 2068–2082, 2008.

[9] A. Smith, A. Veneris, M. Fahim Ali, and A.Viglas, "Fault diagnosis and logic debugging using boolean satisfiability," *IEEE Trans. on CAD*, vol. 24, no. 10, pp. 1606–1621, 2005.

[10] S. Cook, "The complexity of theorem proving procedures," in *3. ACM Symposium on Theory of Computing*, 1971, pp. 151–158.

[11] O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of boolean cardinality constraints," in *Principles and Practice of Constraint Programming*, ser. LNCS, no. 2833, 2003, pp. 108–122.

[12] C. Sinz, "Towards an optimal CNF encoding of boolean cardinality constraints," in *Principles and Practice of Constraint Programming*, ser. LNCS, no. 3709, 2005, pp. 827–831.

[13] O. Bailleux, Y. Boufkhad, and O. Roussel, "A translation of pseudo-boolean constraints to SAT," in *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, 2006, pp. 191–200.

[14] J. Marques-Silva and I. Lynce, "Towards robust cnf encodings of cardinality constraints," in *Principles and Practice of Constraint Programming*, ser. LNCS, no. 4741, 2007, pp. 483–497.

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

[16] R. Wille, G. Fey, D. Große, S. Eggersglüß, and R. Drechsler, "SWORD: A SAT like Prover Using Word Level Information," in *Int'l Conference on Very Large Scale Integration*, 2007, pp. 88–93.

[17] C. Sinz and E.-M. Dieringer, "DPvis - a tool to visualize structured SAT instances," in *Proc. of the 8th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT 2004)*. St. Andrews, Scotland: Springer-Verlag, June 2005, pp. 257–268.

[18] G. Fey, A. Sülflow, and R. Drechsler, "Computing bounds for fault tolerance using formal techniques," in *Design Automation Conf.*, 2009, pp. 190–195.

[19] Accellera, *Property Specification Language – Reference Manual*. Accellera Organization Inc., 2004, available at http://www.accellera.org/home.

[20] Concept Engineering GmbH, *RTLvision PRO*, http://www.concept.de, 2011.

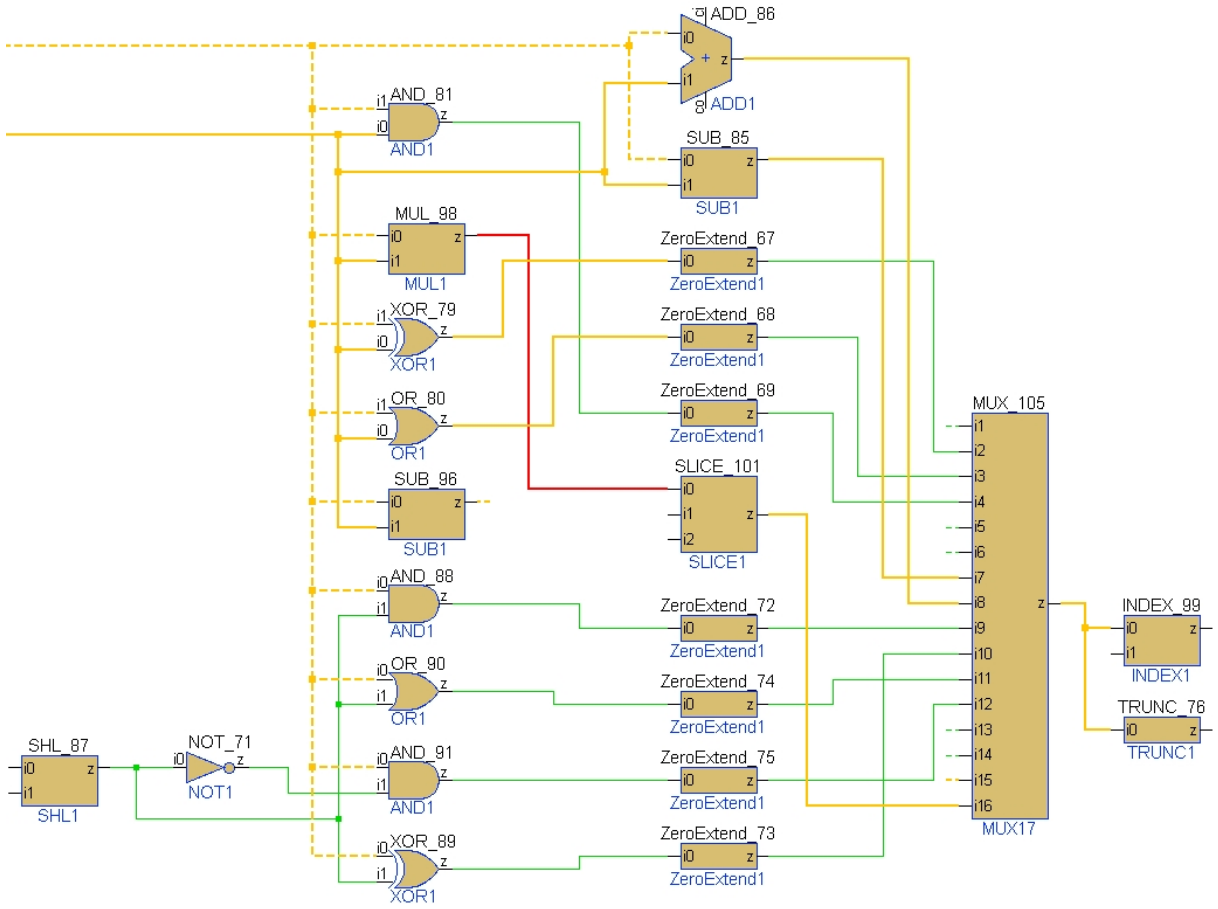

Fig. 2: Visualization of conflict statistics

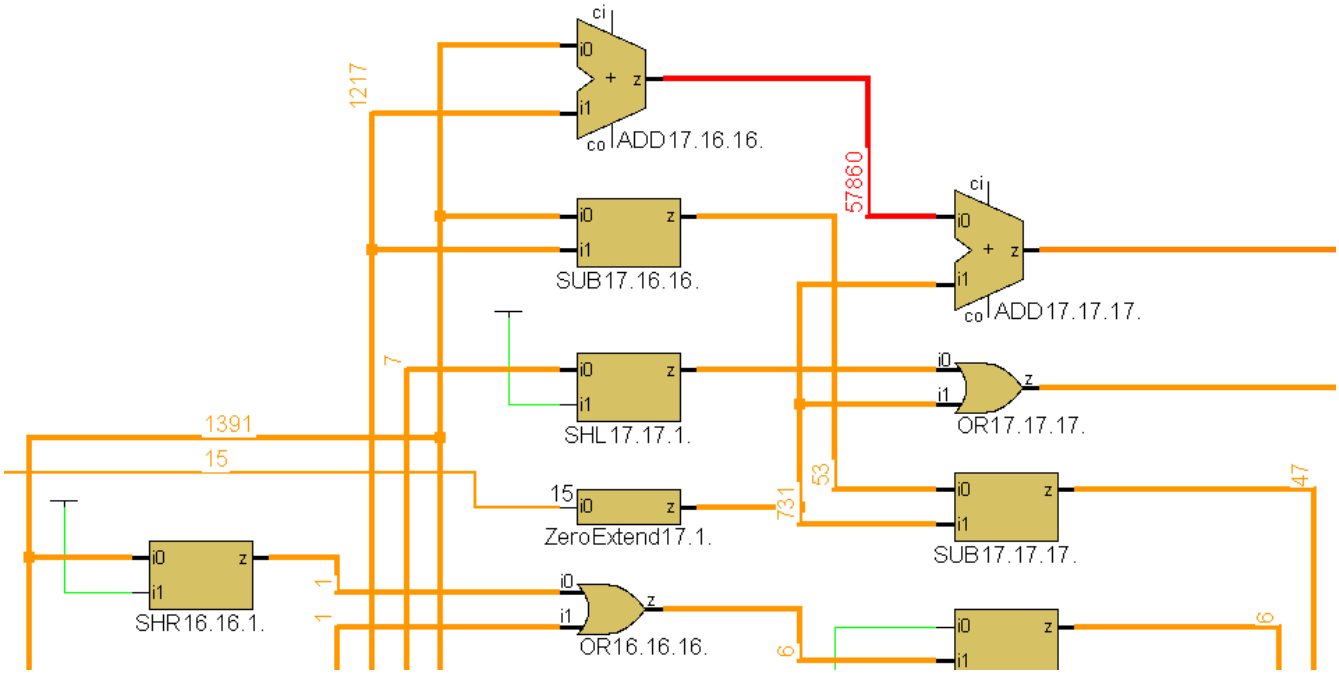

Fig. 3: Visualization of decision statistics