Introduction

Rolf Drechsler

With increasing design complexity, verification becomes a more and more important aspect of the design flow. Modern circuits contain up to several million transistors. In the meantime it has been observed that verification becomes the major bottleneck, i.e. up to 80% of the overall design costs are due to verification. This is one of the reasons why recently several methods have been proposed as alternatives to classical simulation, since it cannot guarantee sufficient coverage of the design. E.g. in [2] it has been reported that for the verification of the Pentium IV more than 200 billion cycles have been simulated, but this only corresponds to 2 CPU minutes, if the chip is run with 1 GHz.

Formal verification techniques have gained large attention, since they allow to prove the correctness of a circuit, i.e. they ensure 100% functional correctness. Besides being more reliable, formal verification approaches have also shown to be more cost effective in many cases, since test bench creation - usually a very time consuming and error prone task - becomes superfluous.

In this introduction, we first briefly describe some of the application domains, where formal techniques have successfully been used. We give some links to further literature where the interested reader can get more information. Then, a list of "challenging problems" is given, i.e. a list of topics that need further investigation in the context of formal hardware verification. Finally, the contributions to this book are briefly described.

1. Formal Verification

The main idea of formal hardware verification is to prove the functional correctness of a design instead of simulating some vectors. For the proof process different techniques have been proposed. Most of them work in the Boolean domain, like *Binary Decision Diagrams* (BDDs) or SAT solvers.

The typical hardware verification scenarios where formal proof techniques are applied are

Equivalence Checking (EC) and

Property Checking (PC), also called *Model Checking* (MC).

The goal of EC is to ensure the equivalence of two given circuit descriptions. These circuits might be given on different levels of abstraction, i.e. register transfer level or gate level. The main steps of an equivalence checker are as follows (see e.g. [12]):

1. Translate both designs to an internal format.

2. Establish the correspondence between the two designs in a matching phase.

3. Prove equivalence or inequivalence.

4. In case of an inequivalence a counter-example is generated and the debugging phase starts.

Notice that the circuit is considered as purely combinational by modeling the state elements as additional primary inputs and outputs. This modeling may result in counter-examples that are not reachable during normal circuit operation.

In contrast to EC, where two circuits are considered, for PC a single circuit is given and properties are formulated in a dedicated "verification language". It is then formally proven whether these properties hold under all circumstances. While "classical" CTL-based model checking [6] can only be applied to medium sized designs, approaches based on *Bounded Model Checking* (BMC) as discussed in [4] give very good results when used for complete blocks with up to 100k gates.

Nevertheless, all these approaches can run into problems caused by complexity, e.g. if the circuit becomes too large or if the function being represented turns out to be "difficult" for formal methods. The second problem often arises in cases of complex arithmetics, like multipliers.

Motivated by this, hybrid methods have been proposed, like e.g. *symbolic simulation* and *assertion checking*. These methods try to bridge the gap between simulation and correctness proofs. But these techniques also make use of formal proof techniques.

For more information on basics on formal verification techniques the reader is referred to [22].

2. Challenges

Even though formal verification techniques are very successfully applied and have become the state-of-the-art in many design flows, still many problems exist. In this section a list of these problems is given. The list is not complete in the sense that all difficulties are covered, but many important ones are mentioned. This gives a better understanding of current problems in hardware verification, motivates for the following chapters of the book and shows directions for future research.

Complexity: According to Moore's law the complexity of the circuits steadily increases. For this, the underlying data structures are very important. For EC and BMC often dedicated data structures are used. For representation of the state space BDDs have shown to work well, but if the size of the circuit becomes too large the BDDs often suffer from "memory explosion".

Proof technology: While BDDs and SAT are the most popular techniques in hardware verification and have also been applied to many domains, there is still a lot of research going on (see also Chapter 1 and 2). Besides the classical monolithic approaches modern EC tools make use of multi-engine approaches that combine different techniques, like SAT, BDD, term rewriting, ATPG, and random pattern simulation. How to successfully combine these - often orthogonal - approaches is not fully understood today.

Word-level approaches: Even though most proof techniques today work on the bit-level, many studies have shown that significant improvements can be achieved if the proof engine makes use of high-level information or even completely works on a higher level of abstraction. In this context also ILP solvers showed promise (see also Chapter 4).

Matching in EC: As described above, before the proof process starts the correspondence between the circuits has to be established. Here, several techniques exist, like name-based, structural or prover-based, but still for large industrial designs these methods often fail. This results in very time consuming user defined matching.

Reachability of counter-examples: In EC and BMC the generated counter-example might not be reachable in normal circuit operation. This results from the modeling of the circuit, i.e. instead of a FSM only the combinational part is considered. Thus, it has to be checked that the counter-example is "valid" after it has been generated, or the prover has to ensure that it is reachable. Techniques have to be developed how this can be ensured without a complete reachability analysis of the FSM, that is usually not feasible due to complexity reasons.

Arithmetic: Industrial practice has shown that today's proof techniques, like BDD and SAT, have difficulties with arithmetic circuits, like multipliers. Word-level approaches have been proposed as an alternative, but these methods turned out to often be difficult to integrate in fully automatic tools. For this, arithmetic circuits - often occurring in circuit design - are still difficult to handle (see Chapter 4).

System integration: PC works best on the module level, i.e. for blocks with up to 100k gates. But in multi-chip modules many of these blocks are integrated to build a system. Due to complexity the modules cannot be verified as one large block and for this models and approaches are needed.

Hybrid approaches: For complex blocks or on the system level PC might be a very complex task and for this simpler alternatives have been studied, i.e. techniques that are more powerful than classical simulation but need less resources than PC. Techniques, like symbolic simulation or assertion-based verification, in this context also make use of formal verification techniques (see also Chapter 5).

Checker synthesis: The specified properties can also be synthesized and added to the design. In this way, they can also be used for on-line test after the circuit has been fabricated.

Analog/mixed signal: Most EC and PC models assume that the circuit is purely digital, while in modern system-on-chip designs many analog components are integrated. For this, also models and proof mechanisms need to be developed for analog and mixed signal devices (see Chapter 6).

Retiming: For EC retimed circuits are still difficult to handle, since in this case the state matching cannot be performed. Thus, the problem remains sequential and by this becomes far too complex.

Multiple clocks: Many circuits have different clocking domains, while verification tools can often only work with a single clock.

Coverage: To check the completeness of a verification process coverage metrics have to be defined. While typical methods, like state coverage, are much too weak in the context of formal verification, there still does not exist a good measure that is comfortable to use for PC.

Diagnosis: After a fault has been identified by a formal verification tool a counter-example is generated. The next step is to identify the fault location or a reason for the failing proof process. Here, also formal proof techniques can be applied.

Most solutions to these problems are still in a very early stage of development, but these fields have to be addressed to make formal hardware verification successful in industrial applications. To orient the reader, some recent references are provided to give a starting point for further studies: [25, 17, 22, 16, 9, 26, 13, 1, 7, 23, 21, 15, 5, 19, 24, 20, 11, 18, 27, 3, 14, 10, 8]

3. Contributions to this Book

The book consists of six chapters that cover most of the aspects described above. Examples of proof technology are described and the latest developments in this field are presented. But also contributions from industrial practice show the importance of formal verification approaches in today's design flows. Each chapter provides experimental results and for each application domain open problems and directions for future work are outlined.

In Chapter 1, Eugene Goldberg analyses the core problem in formal techniques, i.e. the satisfiability problem. Resolution-based SAT solvers are analyzed and a new way of testing satisfiability is proposed.

Properties of SAT and BDDs are studied in Chapter 2 by Gianpiero Cabodi and Stefano Quer. Based on this analysis, the integration of the two currently most successful proof techniques is discussed.

As mentioned above, formal proof techniques often have difficulties in handling arithmetic circuits. This issue is addressed in Chapter 3 by Dominik Stoffel, Evgeny Karibaev, Irina Kufareva and Wolfgang Kunz, where EC approaches are presented.

New innovative proof techniques that make use of word-level information are described by Raik Brinkmann, Peer Johannsen and Klaus Winkelmann, and an industrial property checking flow is presented in Chapter 4.

In Chapter 5, Claudionor Nunes Coelho Jr. and Harry D. Foster focus on assertion-based verification and in this context introduce a formal property language. The underlying methodology is introduced and implications for the user are addressed.

Finally, an approach to formal verification of analog circuits is proposed in Chapter 6 by Walter Hartong, Ralf Klausen and Lars Hedrich. MC and EC techniques for nonlinear analog systems are discussed.

References

[1] L. Bening and H. Foster. *Principles of Verifiable RTL Design.* Kluwer Academic Publishers, 2001.

[2] B. Bentley. Validating the Intel Pentium 4 microprocessor. In *Design Automation Conf.*, pages 244–248, 2001.

[3] J. Bergeron. *Writing Testbenches: Functional Verification of HDL Models.* Kluwer Academic Publishers, 2003.

[4] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Design Automation Conf.*, pages 317–320, 1999.

[5] R. Brinkmann and R. Drechsler. RTL-datapath verification using integer linear programming. In *ASP Design Automation Conf.*, pages 741–746, 2002.

[6] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Sequential circuit verification using symbolic model checking. In *Design Automation Conf.*, pages 46–51, 1990.

[7] H. Chockler, O. Kupferman, R. Kurshan, and M. Vardi. A practical approach to coverage in model checking. In *Computer Aided Verification*, volume 2102 of *LNCS*, pages 66–77. Springer Verlag, 2001.

[8] F. Copty, A. Irron, O. Weissberg, N. Kropp, and G. Kamhi. Efficient debugging in a formal verification environment. *Software Tools for Technology Transfer*, 4:335–348, 2003.

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

[10] R. Drechsler. Synthesizing checkers for on-line verification of system-on-chip designs. In *Int'l Symp. Circ. and Systems*, pages IV:748–IV:751, 2003.

[11] R. Drechsler and N. Drechsler. *Evolutionary Algorithms for Embedded System Design.* Kluwer Academic Publisher, 2002.

[12] R. Drechsler and S. Höreth. Gatecomp: Equivalence checking of digital circuits in an industrial environment. In *Int'l Workshop on Boolean Problems*, pages 195–200, 2002.

[13] R. Drechsler and D. Sieling. Binary decision diagrams in theory and practice. *Software Tools for Technology Transfer*, 3:112–136, 2001.

[14] H. Foster, A. Krolnik, and David J. Lacey. *Assertion-Based Design*. Kluwer Academic Publishers, 2003.

[15] S. Hassoun and T. Sasao. *Logic Synthesis and Verification*. Kluwer Academic Publishers, 2001.

[16] P.-H. Ho, T. Shiple, K. Harer, J. Kukula, R. Damiano, V. Bertacco, J. Taylor, and J. Long. Smart simulation using collaborative formal and simulation engines. In *Int'l Conf. on CAD*, pages 120–126, 2000.

[17] Y. Hoskote, T. Kam, P. Ho, and X. Zhao. Coverage estimation for symbolic model checking. In *Design Automation Conf.*, pages 300–305, 1999.

[18] Y.-C. Hsu, B. Tabbara, Y.-A. Chen, and F. Tsai. Advanced techniques for RTL debugging. In *Design Automation Conf.*, pages 362–367, 2003.

[19] P. Johannsen and R. Drechsler. Formal verification on register transfer level – utilizing high-level information for hardware verification. In *IFIP Int'l Conf. on VLSI*, pages 127–132, 2001.

[20] R. Jones. *Symbolic Simulation Methods for Industrial Formal Verification*. Kluwer Academic Publishers, 2002.

[21] A. Kölbl, J. Kukula, and R. Damiano. Symbolic RTL simulation. In *Design Automation Conf.*, pages 47–52, 2001.

[22] Th. Kropf. *Introduction to Formal Hardware Verification*. Springer, 1999.

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

[24] J. Mohnke, P. Molitor, and S. Malik. Limits of using signatures for permutation independent Boolean comparison. *Formal Methods in System Design: An International Journal*, 2(21):167–191, 2002.

[25] D. Moundanos, J. Abraham, and Y. Hoskote. Abstraction techniques for validation coverage analysis and test generation. *IEEE Trans. on Comp.*, pages 2–14, January 1998.

[26] P. Rashinkar, P. Paterson, and L. Singh. *System-on-a-Chip Verification*. Kluwer Academic Publishers, 2000.

[27] A. Veneris, A. Smith, and M. S. Abadir. Logic verification based on diagnosis techniques. In *ASP Design Automation Conf.*, 2003.