International Journal on Software Tools for Technology (2006) 8(4/5): 355–358
DOI 10.1007/s10009-005-0212-y

SPECIAL SECTION ON RECENT ADVANCES IN HARDWARE VERIFICATION

Enrico Tronci

Introductory paper

Published online: 3 April 2006

Abstract In today's competitive market designing of digital systems (hardware as well as software) faces tremendous challenges. In fact, notwithstanding an ever decreasing project budget, time to market and product lifetime, designers are faced with an ever increasing system complexity and customer expected quality. The above situation calls for better and better formal verification techniques at all steps of the design flow. This special issue is devoted to publishing revised versions of contributions first presented at the 12th Advanced Research Working Conference on *Correct Hardware Design and Verification Methods* (CHARME) held 21–24 October 2003 in L'Aquila, Italy. Authors of well regarded papers from CHARME'03 were invited to submit to this special issue. All papers included here have been suitably extended and have undergone an independent round of reviewing.

Keywords Formal verification · Model checking

1 Introduction

In today's competitive market designing of digital systems (hardware as well as software) faces tremendous challenges. In fact, notwithstanding an ever-decreasing project budget, time to market and product lifetime, designers are faced with an ever-increasing system complexity and customer expected quality. For example, according to National Institute of Standards and Technology [1], software bugs cost the US economy an estimated $59.5 billion each year, with more than half of the cost borne by end users and the remainder by developers and vendors. Improvements in testing and verification technology could reduce this cost by about a third.

As for digital hardware (e.g. ICs, ASICs), a 2002 study by Collett International Research [2, 3] revealed that first silicon success rate has fallen to 39% from about 50% (its value in 2000). The total cost of re-spins is about $100,000 plus months of additional development time. Thus, companies that are able to curb this trend have a huge advantage over their competitors, both in terms of the ensuing reduction in engineering cost and the business advantage of being to market sooner and with high-quality products.

Taking into account that the testing phase may presently account for 30–70% of the product final cost [1], it is easy to see that even keeping the cost of testing below 50% of the final cost is a formidable challenge for new products. Failing to meet such challenges means being out of the market. This is why many hi-tech companies have their own research labs working on formal verification methods.

To help focus verification efforts it is useful to understand what are the main sources of errors. The research in [3] identified the following sources of errors in chip design.

Design errors About 82% of designs with re-spins resulting from logic/functional flaws had design errors. This means that particular *corner* cases simply were not covered during the testing process, and bugs remained hidden in the design all the way through tape-out.

Specification errors About 47% of designs with re-spins resulting from logic/functional flaws had incorrect/incomplete specifications. Moreover, 32% of designs with re-spins resulting from logic/functional flaws had changes in specifications. Clearly, improved specification techniques are required.

Re-used modules and imported IP About 14% of all chips that failed had bugs in reused components or imported IP.

The above situation calls for better formal verification techniques at all steps of the design flow.

This special section is devoted to publishing revised versions of contributions first presented at the 12th Advanced Research Working Conference on *Correct Hardware Design and Verification Methods* (CHARME) [4] held 21–24 October 2003 in L'Aquila, Italy. Authors of well-regarded papers from CHARME'03 were invited to submit to this special

E. Tronci (✉)
Dipartimento di Informatica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Rome, Italy
E-mail: tronci@di.uniroma1.it

issue. All papers included here have been suitably extended and have undergone an independent round of reviewing.

2 Hardware model checking

Automatic Formal Verification's (namely, model checking [5]) goal is to check correctness of the system at hand with respect to given specifications. The system to be verified is typically defined using a high-level language (e.g. Verilog, VHDL) whereas the property to be verified is typically defined using a temporal logic (e.g. CTL [5]) or the same language used to define the system under verification.

Automatic verification always entails (and in many cases is equivalent to) exploring all system states that are reachable from a given initial state and checking that each reachable state satisfies the given specification (invariant). From this follows that State Space Exploration (Reachability Analysis) algorithms are at the very heart of all model-checking techniques. Note that, as for coverage, formal verification is equivalent to testing with 100% coverage.

In the same amount of time Automatic Formal Verification algorithms explore many more system states than traditional testing. For this reason automatic formal verification is supported by many digital hardware CAD tools. In such tools the model checking machinery is typically hidden behind the curtains of otherwise "traditional" Verilog or VHDL simulators. This allows digital hardware designers with little or no knowledge about formal methods to use model checking effortless. What designers see is just a simulator that explores a much larger fraction of the state space than a "traditional" simulator in the same amount of time.

Effectiveness of the method together with its short learning curve are two key ingredients of model checking wide diffusion, both in academia and industry.

Essentially there are two main model-checking techniques: explicit and symbolic. *Explicit model checking* (e.g. see [6–9]) uses a hash table to store the set of visited states whereas *symbolic model checking* (e.g. see [10, 11]) represents the set of visited states with its characteristic function which, in turn, is represented and manipulated using *Ordered Binary Decision Diagrams* (OBDDs) [12].

If we only look for error states reachable in at most k steps from an initial state than the model checking problem can be transformed into a satisfiability problem. This leads to the *Bounded Model Checking* approach [13]. The satisfiability problem thus obtained can then be solved using a SAT solver (e.g. [14]). Since set of states are represented with Boolean expressions bounded model checking is to be considered a symbolic model-checking technique.

Model checkers represent the dynamics of the system to be verified using a `nextstate` function. When using explicit model checking the `nextstate` function takes a system state and returns the set of system states reachable in one step. When using symbolic model checking the `nextstate` function takes as input (the characteristic function of) a set X of states and returns (the characteristic function of) the set X' of states reachable in one step from a state in X.

Usually explicit model checkers are effective on software-like or protocol-like systems [15] as well as on system which `nextstate` definition entails complex arithmetical operations on the state variables (e.g. as in hybrid systems) [16]. Intuitively, the latter is because arithmetical operations make life very hard for OBDDs, thus making usage of a symbolic model checkers on hybrid systems rather problematic. Here are some examples of *explicit model checkers*: SPIN [9, 17], Murφ [8, 18], Bandera [19, 20], JPF [21, 22], FeaVer [23, 24], Caching Murφ [25, 26].

Usually symbolic model checking works better for *control dominated* systems. Many digital hardware systems as well as low level software (e.g. drivers, embedded systems) fall in this category. Here are some examples of *symbolic model checkers*: SMV [10, 27], NuSMV [28, 29], VIS [30, 31], UPPAAL [32, 33] BMC [13, 27], SLAM [34, 35].

Probabilistic protocols can be verified using *Probabilistic Model Checkers* taking as input Markov Chains rather than finite state automata. A state of the art probabilistic model checker is PRISM [36–38]. PRISM implements symbolic, explicit as well as hybrid probabilistic verification techniques.

3 Recent advancements in hardware model checking

Hardware model checking technology is used in the standard design flow of many leading industries. This is a recognition of the success of this technology as well as a never ending source of harder and harder verification problems.

For the above reasons the quest for algorithms and techniques that can improve the state of the art of the verification technology is never ending.

This special section focuses on six successful techniques that are capable of improving the state of the art of today's formal verification tools.

The paper *Inductive Assertions and Operational Semantics*, by Moore [39], shows how classic inductive assertions can be used in conjunction with a formal operational semantics to prove partial correctness properties of programs. The proposed method imposes only the proof obligations that would be produced by a verification condition generator without asking for the definition of a verification condition generator. All that is required is a theorem prover, a formal operational semantics, and the object program with appropriate assertions at user-selected cut-points. The verification conditions are generated in the course of the theorem proving process by straightforward symbolic evaluation of the formal operational semantics. The proposed technique is demonstrated by proving the partial correctness of simple bytecode programs with respect to a pre-existing operational model of the Java Virtual Machine.

The paper *Putting it all together – Formal verification of the VAMP*, by Beyer et al. [40], presents the design, functional verification and synthesis of a processor with full

DLX instruction set, delayed branch, Tomasulo scheduler, maskable nested precise interrupts, pipelined fully IEEE compatible dual precision floating point unit with variable latency, and separate instruction and data caches. This work has been carried out within the VAMP (Verified Architecture MicroProcessor) project and verification has been carried out with the theorem proving system PVS. Moreover, the processor has been implemented on a Xilinx FPGA.

The paper *Coverage Metrics for Formal Verification*, by Chockler et al. [41], investigates how various notion of coverage metrics can be defined in a formal verification framework. More specifically, in formal verification, usually one verifies that a system is correct with respect to a given specification. However, even when the system is proven to be correct, there is still a question of how complete the specification is, and whether it really covers all the behaviors of the system. The challenge of making the verification process as exhaustive as possible is even more crucial in simulation-based verification, where the infeasible task of checking all input sequences is replaced by checking a test suite consisting of a finite subset of them. It is very important to measure the exhaustiveness of the test suite, and indeed, there has been an extensive research in the simulation-based verification community on coverage metrics, which provide such a measure. It turns out that no single measure can be absolute, leading to the development of numerous coverage metrics whose usage is determined by industrial verification methodologies. On the other hand, prior research of coverage in formal verification has focused solely on state-based coverage. This paper adapts the work done on coverage in simulation-based verification to the formal-verification setting in order to obtain new coverage metrics. Thus, for each of the metrics used in simulation-based verification, it presents a corresponding metric that is suitable for the formal verification setting, and describes an algorithmic way to check it.

The paper *Efficient Distributed SAT and SAT based Distributed Bounded Model Checking*, by Ganay et al. [42], investigates distributed approaches to SAT-based model checking. In fact, SAT-based Bounded Model Checking (BMC), though a robust and scalable verification approach, is computationally intensive, requiring large memory and time. Interestingly, with the recent development of improved SAT solvers, it is frequently the memory limitation of a single server rather than time that becomes a bottleneck for doing deeper BMC search. Distributing computing requirements of BMC over a network of workstations can overcome the memory limitation of a single server, albeit at increased communication cost. This paper presents (a) a method for distributed-SAT over a network of workstations using a Master/Client model where each Client workstation has an exclusive partition of the SAT problem and uses knowledge of partition topology to communicate with other Clients, (b) a method for distributing SAT-based BMC using the distributed-SAT. For the sake of scalability, at no point in the BMC computation does a single workstation have all the information. This paper presents experiments on a network of heterogeneous workstations interconnected with a standard Ethernet LAN. For example, on an industrial design with about 13 K FFs and about 0.5 M gates, the non-distributed BMC on a single workstation (with 4 GB memory) ran out of memory after reaching a depth of 120. On the other hand, the paper SAT-based distributed BMC over five similar workstations was able to go up to 323 steps with a communication overhead of only 30%.

The paper *Finite Horizon Analysis of Markov Chains with the Murφ Verifier*, by Della Penna et al. [43], presents an explicit disk-based verification algorithm for Probabilistic Systems defining discrete time/finite state Markov Chains. That is, given a Markov Chain and an integer k (horizon), the algorithm presented in this paper checks whether the probability of reaching an error state in at most k-steps is below a given threshold. The authors present an implementation of their algorithm within a suitable extension of the Murφ verifier and call the resulting probabilistic model checker FHP-Murφ (Finite Horizon Probabilistic Murφ). This paper also present experimental results comparing FHP-Murφ with (a finite horizon subset of) PRISM, a state-of-the-art symbolic model checker for Markov Chains. Their experimental results show that FHP-Murφ can handle systems that are out of reach for PRISM, namely those involving arithmetic operations on the state variables (e.g. hybrid systems).

The paper *Towards Diagrammability and Efficiency in Event Sequence Languages*, by Fisler [44], investigates the problem of devising event sequence languages for hardware verification. Indeed, many industrial verification teams are developing event sequence languages for hardware verification. Such languages must be expressive, designer friendly, and hardware specific, as well as efficient to verify. While the formal verification community has formal models for assessing the efficiency of an event sequence language, none of these models also accounts for designer friendliness. This paper proposes an intermediate language for event sequences that addresses both concerns. The language achieves usability through a correlation to timing diagrams; its efficiency arises from its mapping into deterministic weak automata. The author present the language, relate it to existing event sequence languages, and prove its relationship to deterministic weak automata. These results indicate that timing diagrams can become more expressive while remaining more efficient for symbolic model checking than LTL.

References

1. National Institute of Standards and Technology: The economic impacts of inadequate infrastructure for software testing. Planning Report 02–3. National Institute of Standards and Technology (2002)
2. Fitzpatrick, T., Schutten, R.: Design for verification: Blueprint for productivity and product quality. Technical report, Synopsy Inc., 2003. http://www.synopsys.com/products/simulation/dfv_wp.html

3. Collett International Research Inc. IC/ASIC functional verification study. Technical report, Collett International Research Inc. (2002) http://www.collett.com/reports.htm
4. Geist, D., Tronci, E. (eds.): Correct Hardware Design and Verification Methods (CHARME), vol. 2860 of Lecture Notes in Computer Science, L'Aquila, Italy. 12th IFIP WG 10.5 Advanced Research Working Conference. Springer, Berlin (2003)
5. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, MA (1999)
6. Clarke, E.M., Emerson, E.A.: Synthesis of synchronization skeletons for branching time temporal logic. In: Logic of Programs: Workshop, vol. 131 of Lecture Notes in Computer Science, Springer-Verlag, Yorktown Heights, NY (1981)
7. Emerson, E.A., Clarke, E.M., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Languages Syst. **8**(2), 244–263 (1986)
8. Dill, D.L., Drexler, A.J., Hu, A.J., Yang, C.H.: Protocol verification as a hardware design aid. In: Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer and Processors, pp. 522–525. IEEE Computer Society, Washington, DC (1992)
9. Holzmann, G.J.: The SPIN Model Checker, Primer and Reference manual. Addison-Wesley, Reading, MA (2003)
10. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Inf. Comput. **98**(2), 142–170 (1992)
11. McMillan, K.L.: Symbolic Model Checking. Kluwer, Dordretch (1993)
12. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput **C-35**(8), 677–691 (1986)
13. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without bdds. In: Cleaveland, W.R. (ed.) Proceedings of 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS'99, vol. 1579 of LNCS. Springer, Amsterdam, The Netherlands (1999)
14. Zhang, L., Malik, S.: The quest for efficient boolean satisfiability solvers. In: Proceedings of 14th Conference on Computer Aided Verification (CAV), Lecture Notes in Computer Science. Springer-Verlag, Copenhagen, Denmark (2002)
15. Hu, A.J., York, G., Dill, D.L.: New techniques for efficient verification with implicitly conjoined BDDs. In: 31st ACM/IEEE Design Automation Conference (DAC). San Diego Convention Center, San Diego, CA (1994)
16. Della Penna, G., Intrigila, B., Melatti, I., Minichino, M., Ciancamerla, E., Parisse, A., Tronci, E., Zilli, M.V.: Automatic verification of a turbogas control system with the murφ verifier. In: Maler, O., Pnueli, A. (eds.) Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3–5, 2003, Proceedings, vol. 2623 of Lecture Notes in Computer Science, pp. 141–155. Springer, Berlin (2003)
17. Spin web page: http://spinroot.com
18. Murphi web page: http://sprout.stanford.edu/dill/murphi.html
19. Corbett, J., Dwyer, M., Hatcliff, J., Pasareanu, R.C., Laubach, S., Zheng, H.: Bandera: Extracting finite-state models from java source code. In: Proceedings of the 22nd International Conference on Software Engineering (2000)
20. Bandera web page: http://bandera.projects.cis.ksu.edu/index.shtml.
21. Havelund, K., Pressburger, T.: Model checking java programs using java pathfinder. Int. J. Software Tools Technol. Transfer **2**(4), April (2000)
22. Java pathfinder web page: http://ase.arc.nasa.gov/havelund/jpf.html
23. Holzmann, G., Smith, M.: A practical method for verifying event driven software. In: Proceedings of ACM/IEEE International Conference on Software Engineering (ICSE). ACM/IEEE, New York (1999)
24. Feaver web page: http://cm.bell-labs.com/cm/cs/what/feaver/
25. Della Penna, G., Intrigila, B., Melatti, I., tronci, E., Zilli, M.: Exploiting transition locality in automatic verification of concurrent systems. International Journal on Software Tools for Technology Transfer **6**(4) (2004)
26. Caching murphi web page: http://www.dsi.uniroma1.it/∼tronci/cached.murphi.html
27. Smv web page: http://www.cs.cmu.edu/∼modelcheck/
28. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In: Proceedings of International Conference on Computer-Aided Verification (CAV 2002), vol. 2404 of LNCS. Springer, Copenhagen, Denmark (2002)
29. Nusmv web page: http://nusmv.irst.itc.it
30. Brayton, R.K., Hachtel, G.D., Sangiovanni-Vincentelli, A., Somenzi, F., Aziz, A., Cheng, S.-T., Edwards, S., Khatri, S., Kukimoto, Y., Pardo, A., Qadeer, S., Ranjan, R.K., Sarwary, S., Shiple, T.R., Swamy, G., Villa, T.: Vis: A system for verification and synthesis. In: Proceedings of Computer Aided Verification (CAV), Lecture Notes in Computer Science. Springer, Berlin (1996)
31. Vis web page: http://vlsi.colorado.edu/ vis
32. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL: Status and developments. In: Grumberg, O. (ed.) Proc. of Computer 9th Intern. Conf. on Computer Aided Verification (CAV), vol. 1254 of Lecture Notes in Computer Science, pp. 456–459. Springer, Haifa, Israel (1997)0
33. Uppaal web page: http://www.uppaal.com/
34. Ball, T., Rajamani, S.K.: Automatically validating temporal safety properties of interfaces. In: Dwyer, M.B. (ed.) Proceedings of the 8th International SPIN Workshop on Model Checking Software, vol. 2057 of LNCS, pp 103–122, Springer, Toronto, Canada (2001)
35. Slam web page: http://research.microsoft.com/projects/slam/
36. Baier, C., Clarke, E.M., Hartonas-Garmhausen, V., Kwiatkowska, M., Ryan, M.: Symbolic model checking for probabilistic processes. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) Automata, Languages and Programming, 24th International Colloquium, ICALP'97, Bologna, Italy, Proceedings, vol. 1256 of Lecture Notes in Computer Science, pp. 430–440. Springer, Berlin (1997)
37. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: A hybrid approach. In: Katoen, J.P., Stevens, P. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, Proceedings, vol. 2280 of Lecture Notes in Computer Science, pp. 52–66. Springer, Berlin (2002)
38. Prism web page: http://www.cs.bham.ac.uk/∼dxp/prism/
39. Moore, J.S.: Inductive assertions and operational semantics. Int. J. Software Tools Technol. Transfer XX, XX (XXXX)
40. Beyer, S., Jacobi, C., Kröning, D., Leinenbach1, D., Paul, W.J.: Putting it all together—formal verification of the vamp. Int. J. Software Tools Technol. Transfer XX, XX (XXXX)
41. Chockler, H., Kupfermann, O., Vardi, M.Y.: Coverage metrics for formal verification. Int. J. Software Tools Technol. Transfer XX, XX (XXXX)
42. Ganay, M.K., Gupta, A., Yang, Z., Ashar, P.: Efficient distributed sat and sat based distributed bounded model checking. Int. J. Software Tools Technol. Transfer XX, XX (XXXX)
43. Della Penna, G., Intrigila, B., Melatti, I., Tronci, E., Zilli, M.V.: Finite horizon analysis of markov chains with the murφ verifier. Int. J. Software Tools Technol. Transfer XX, XX (XXXX)
44. Fisler, K.: Towards diagrammability and efficiency in event sequence languages. Int. J. Software Tools Technol. Transfer XX, XX (XXXX)