From *An Invitation to Formal Methods*, a collection of short papers edited by Hossein Saiedian, IEEE Computer, April 1996, Vol. 29, No. 4, pp. 16–30.

Acceptance of Formal Methods: Lessons from Hardware Design*

David L. Dill
Computer Science Department
Stanford University
Stanford CA 94305

Dill@cs.stanford.edu

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025

Rushby@csl.sri.com

Despite years of research, the overall impact of formal methods on mainstream software design has been disappointing. By contrast, formal methods are beginning to make real inroads in commercial hardware design. This penetration is the result of sustained progress in automated hardware verification methods, an increasing accumulation of success stories from using formal techniques, and a growing consensus among hardware designers that traditional validation techniques are not keeping up with the increasing complexity of designs. For example, validation of a new microprocessor design typically requires as much manpower as the design itself, and the size of validation teams continues to grow. This manpower is employed in writing test cases for simulations that run for months on acres of high-powered workstations.

In particular, the notorious FDIV bug in the Intel Pentium processor [13], has galvanized verification efforts, not because it was the first or most serious bug in a processor design, but because it was easily repeatable and because the cost was quantified (at over $400 million).

Hence, hardware design companies are increasingly looking to new techniques, including formal verification, to supplement and sometimes replace conventional validation methods. Indeed, many companies, including industry leaders such as AT&T, Cadence, Hewlett-Packard, IBM, Intel, LSI Logic, Motorola, Rockwell, Texas Instruments, and Silicon Graphics have created formal verification groups to help with ongoing designs.[1] In many cases, these groups began by demonstrating the effectiveness of formal verification by finding subtle design errors that were overlooked by months of simulation.

*This work was partially sponsored by DARPA through NASA Ames Research Center Contract NASA-NAG-2-891.

[1]See the attendance list for FMCAD [15] at http://www.csl.sri.com/FMCAD96/attend.html: there were 190 attendees, with more than half coming from industry.

Why have formal methods been more successful for hardware than for software? We believe that the overriding reason is that applications of formal methods to hardware have become cost-effective.

The decision to use a new methodology is driven by *economics*: do the benefits of the new method exceed the costs of converting to it and using it by a sufficient margin to justify the risks of doing so? The benefits may include an improved product (e.g., fewer errors), but those most keenly desired are reduced validation costs and reduced time-to-market (for the same product quality). The chief impediments to applying traditional formal methods are that the costs are thought to be high (e.g., large amounts of highly skilled labor) or even unacceptable (a potential increase in time-to-market), while the benefits are uncertain (a possible increase in product quality). Formal hardware verification has become attractive because it has focussed on reducing the cost and time required for validation rather than pursuit of perfection.

Of course, hardware has some intrinsic advantages over software as a target for formal methods. In general, hardware has no pointers, no potentially unbounded loops or recursion, and no dynamically created processes, so its verification problem is more tractable. Furthermore, hardware is based on a relatively small number of major design elements, so that investment in mastering the formal treatment of, say, pipelining or cache coherence can pay off over many applications. And the cost of fabricating hardware is much greater than software, so the financial incentive to reduce design errors is much greater.

However, we believe there are some lessons and principles from hardware verification that can be transferred to the software world. Some of these are listed below.

Provide Powerful Tools

Technology is the primary source of increased productivity in most areas, and especially this one. In particular, tools that use formal specifications as the starting point for mechanized formal *calculations* are the primary source of cost-effective applications of formal methods. This is exactly analogous to the use of mathematical modeling and calculation in other engineering disciplines. Without tools to deliver tangible benefits, formal specifications are just documentation, and there is little incentive for engineers to construct them, or to keep them up to date as the design evolves.

For hardware, a spectrum of tools has evolved to perform formal calculations at different levels of the design hierarchy and with different benefits and costs. At the lowest level are tools that check Boolean equivalence of combinational circuits (this is useful for checking manual circuit optimizations). Techniques based on Ordered Binary Decision Diagrams (BDDs) are able to check large circuits quite efficiently,

and are now incorporated in commercial CAD tools [3]. At a higher level, designs can often be represented as interacting finite state machines, and tools that systematically explore the combined state space can check that certain desired properties always hold, or that undesired circumstances never arise. Tools based on explicit state enumeration can explore many millions of states in a few hours; tools that represent the state space symbolically (using BDDs) can sometimes explore vast numbers of states (e.g., 10^{100}) in the same time, and can check richer properties (e.g., those that can be specified in a temporal logic, in which case the technique is called "temporal logic model checking") [10]. At the highest levels, or when very complex properties or very large (or infinite) state spaces are involved, highly automated theorem proving methods can be used to compare implementations with specifications. These theorem proving methods include decision procedures for propositional calculus, equality, and linear arithmetic, combined with rewriting, and induction [7, 8, 2]. In all cases, the tools concerned are highly engineered so that they can deal with very large formulas, and require little or no user interaction when applied in familiar domains.

Use Verification to Find Bugs

A tool that simply blesses a design at the end of a laborious process is not nearly as impressive to engineers as a tool that finds a bug. Finding bugs is computationally easier than proving correctness, and a potential cost can be attached to every bug that is found, making it easy to see the payoff from formal verification. Traditional validation methods already are used primarily as bug-finders, so formal methods are very attractive if they find different bugs from those traditional methods—a much more achievable goal than trying to guarantee correctness.

Shortcuts can be taken when formal verification is used for finding bugs rather than proving correctness. For example, a system can be "down-scaled"—the number or sizes of components can be drastically reduced. For example, a directory-based cache-coherence protocol can be checked with just four processors, one cache line, and two data values—such a down-scaled description will still have many millions of states, but will be within reach of state exploration and model checking methods. These methods can check the reduced system *completely*; in contrast, simulation checks the full system very incompletely. Both techniques find some bugs and miss others, but the formal methods often detect bugs that simulation does not. For example, cache-coherence bugs have been found in the IEEE standard FutureBus+ [6] and Scalable Coherent Interface (SCI) protocols [17] in just this way. Some researchers are now applying these techniques to software.

Formal Techniques Must Be Targeted

In hardware, experience shows that control-dominated circuits are much harder to debug than data paths. So effort has gone into developing formal verification techniques for protocols and controllers rather than for data paths. Targeting maximizes the potential payoff of formal methods by solving problems that are not handled by other means. Notice that the targeted problems often concern the *hardest* challenges in design: cache coherence, pipeline (and now superscalar) correctness, and floating point arithmetic. For software, correspondingly difficult and worthwhile challenges include those where local design decisions have complex global consequences, such as the fault-tolerance and real-time properties of concurrent distributed systems.

Researchers Should Apply Their Work To Real Problems

Our research priorities are completely different from what they would have been, had we not exercised our ideas on realistic problems. Such efforts have frequently raised interesting new theoretical problems, as well as highlighting the need for improvements in tools.

Of course, applying verification strategies to real problems is also crucial for building credibility. There is now a long string of success stories from academia and industry where finite-state verification techniques have been applied to hardware and protocols. A few documented examples include the protocol bugs mentioned above in FutureBus+ (found using symbolic model checking with a version of CMU's SMV system [10]) and SCI (found using explicit state enumeration with Stanford's Murphi verifier [9]), and formal verification of the microarchitecture and microcode of the Collins AAMP5 [16] and AAMP-FV avionics processors (using theorem proving with SRI's PVS system [12]). Several groups have also demonstrated the ability to detect bugs in the quotient-prediction tables of SRT division algorithms (similar to the Pentium FDIV bug), and some have been able to verify specific SRT circuits and tables [5, 14, 4, 11]. There have also been many unpublicized examples of problems found by industrial formal verification groups, which have helped them build credibility among designers and managers in their companies.

Conclusion

We attribute the growing acceptance of formal methods in commercial hardware design to the power and effectiveness of the tools that have been developed, to the pragmatic character of the ways in which those tools have been applied, and to the overall cost-effectiveness and utility that has been demonstrated. We believe that formal methods can achieve similar success in selected software applications by following the same principles.

References

[1] Rajeev Alur and Thomas A. Henzinger, editors. *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[2] Clark Barrett, David Dill, and Jeremy Levitt. Validity checking for combinations of theories with equality. In Srivas and Camilleri [15], pages 187–201.

[3] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[4] Randal E. Bryant. Bit-level analysis of an SRT divider circuit. In *Proceedings of the 33rd Design Automation Conference*, pages 661–665, Las Vegas, NV, June 1996.

[5] E. M. Clarke, S. M. German, and X. Zhao. Verifying the SRT division algorithm using theorem proving techniques. In Alur and Henzinger [1], pages 111–122.

[6] Edmund M. Clarke, Orna Grumberg, Hiromi Haraishi, Somesh Jha, David E. Long, Kenneth L. McMillan, and Linda A. Ness. Verification of the Futurebus+ cache coherence protocol. *Formal Methods in System Design*, 6(2):217–232, March 1995.

[7] D. Cyrluk, S. Rajan, N. Shankar, and M. K. Srivas. Effective theorem proving for hardware verification. In Ramayya Kumar and Thomas Kropf, editors, *Theorem Provers in Circuit Design (TPCD '94)*, volume 910 of *Lecture Notes in Computer Science*, pages 203–222, Bad Herrenalb, Germany, September 1994. Springer-Verlag.

[8] David Cyrluk, Patrick Lincoln, and N. Shankar. On Shostak's decision procedure for combinations of theories. In M. A. McRobbie and J. K. Slaney, editors, *Automated Deduction—CADE-13*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 463–477, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[9] David L. Dill. The Murϕ verification system. In Alur and Henzinger [1], pages 390–393.

[10] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, MA, 1993.

[11] Paul S. Miner and James F. Leathrum, Jr. Verification of IEEE compliant subtractive division algorithms. In Srivas and Camilleri [15], pages 64–78.

[12] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[13] Vaughan Pratt. Anatomy of the Pentium bug. In *TAPSOFT '95: Theory and Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 97–107, Aarhus, Denmark, May 1995. Springer-Verlag.

[14] H. Rueß, N. Shankar, and M. K. Srivas. Modular verification of SRT division. In Alur and Henzinger [1], pages 123–134.

[15] Mandayam Srivas and Albert Camilleri, editors. *Formal Methods in Computer-Aided Design (FMCAD '96)*, volume 1166 of *Lecture Notes in Computer Science*, Palo Alto, CA, November 1996. Springer-Verlag.

[16] Mandayam K. Srivas and Steven P. Miller. Formal verification of the AAMP5 microprocessor. In Michael G. Hinchey and Jonathan P. Bowen, editors, *Applications of Formal Methods*, Prentice Hall International Series in Computer Science, chapter 7, pages 125–180. Prentice Hall, Hemel Hempstead, UK, 1995.

[17] Ulrich Stern and David L. Dill. Automatic verification of the SCI cache coherence protocol. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 21–34. IFIP WG10.5, 1995.