Transforming the Theorem Prover into a Digital Design Tool: From Concept Car to Off-Road Vehicle

David Hardin, Matthew Wilding, and David Greve

Advanced Technology Center
Rockwell Collins
Cedar Rapids, IA 52498 USA
{dshardin, mmwildin, dagreve}@collins.rockwell.com

Abstract. As digital designs grow evermore complex and design cycles become ever shorter, traditional informal methods of design verification are proving inadequate. Design teams are increasingly turning to formal techniques to address this "verification crunch". The theorem prover, with its emphasis on establishing correctness, is arguably the dream design verification tool; however, theorem provers are rarely used in digital design. Much like automotive industry "concept cars", theorem provers provide a compelling vision of the future, but in the real world of industrial design they have proven to be difficult to drive and expensive to maintain. We suggest ways that the theorem prover "concept cars" of today can be adapted to become the "off-road vehicles" necessary to negotiate the rough-and-tumble terrain of digital design in the 21st century.

1 Introduction

The relentless march of semiconductor process technology over the last thirty years has given engineers exponentially increasing transistor budgets at constant recurring cost. This has encouraged increased functional integration onto a single die, as well as increased architectural sophistication of the functional units themselves. In CPU design, we have seen integration of memory management units, floating point units, cache, etc., as well as increased word length, pipelining, superscalar execution, and other architectural enhancements in the CPU core itself. Higher-end CPU designs now routinely encompass ten million transistors. Additionally, the lifetimes of designs are decreasing – Intel Pentium production lasted only two years – thus pressuring engineers to reduce design cycle time.

The increased scope of a typical digital chip design project, coupled with the desire for decreased cycle time, has caused design teams to increase in both size and number. It is not uncommon for a large company to be engaged in the design of several generations of a product simultaneously, with large teams of engineers working on each generation.

Not surprisingly, this rough-and-tumble environment has led to a number of uncaught design flaws. The most famous of these flaws is the Intel Pentium FDIV bug, although it is hardly alone; the errata sheet for a modern microprocessor may have more than one hundred entries.

Obviously, traditional simulation-based design verification has not kept up with the scale or pace of modern digital design. Increasingly, designers speak of a "verification crunch", and many design teams have looked to formal verification methods for help. Refutation-based techniques such as model checking are currently in use in a number of locations, and have been effective in finding some bugs. However, engineers aspire to create perfect designs; a verification tool that could establish that a design faithfully implements its specification would be ideal. In addition, the mere act of creating an unambiguous specification would have enormous benefit to the designers of the multiple upwardly compatible generations of a product, as well as to the user community.

Such correctness tools exist, in the form of theorem provers such as PVS [14] and ACL2 [10]. These tools have been used in the verification of industrial-scale designs [6, 5, 11, 12, 2, 15] but have not enjoyed much mainstream use. The reasons for the lack of success of theorem provers include:

- Insufficient automation
- Inefficient executability of formal models
- Lack of integration with the engineering environment
- Unfamiliar syntax and semantics
- Lack of support for digital design languages (e.g., VHDL, Verilog)
- Lack of infrastructure for reasoning about digital design (e.g., bitvectors)
- Sometimes gaping holes in basic mathematical libraries
- Speed, especially as a function of model size

To make an automotive analogy, theorem provers are much like "concept cars" that provide a compelling vision of the future but are difficult to drive and expensive to maintain in the real world. What is needed is an "off-road vehicle" to negotiate the rough-and-tumble terrain of digital design. In the next sections, we describe adaptations of theorem prover technology that can transform the theorem prover into a real-world digital design tool, such as might be used during our group's recent development of the JEM1 [9, 18]. We are currently evaluating many of these adaptations in the Rockwell Collins engineering design environment, using real development projects as our testbed.

2 Increasing Automation

Mistakes in proof development and changes to system design and specification are inevitable for real verifications. It is crucial that proofs we construct about our systems be robust in the face of changes. Large programming projects use software engineering techniques to make software robust despite inevitable changes. So too must large machine-checked proof projects use techniques to develop robust proofs. We've learned this lesson the hard way by building fragile proofs. For example, in the AAMP-FV verification effort a change was made in the formal model related to memory address decoding [11]. This change caused every previously-constructed instruction correctness proof to fail although the change had little to do with the substance of most of the proofs.

We are working toward more robust proofs by relying on automated proof techniques. An important advance in this area is the interpreter style of proofs that has been used in a variety of verification projects, including [1, 3, 4, 13, 16]. This approach involves specifying the semantics of a computer system with an interpreter and deriving symbolic results using automatic reasoning. We have adapted this approach for use in PVS [17] and believe that its usefulness transcends the particularities of different theorem proving systems.

3 Achieving Efficient Execution of Formal Models

It is often the case when proving theorems about computer systems that one is faced with an expression composed entirely of functions and constants. Perhaps the word in memory pointed to by the PC is an "add" instruction, or the carry flag holds, or the microcode uses the bit mask 0xFFFF. Proofs about *executable* functions are often easier to construct because function execution can be used to simplify expressions.

Using executable functions in specifications has another, easily overlooked, advantage. An issue that arises in formal verification work is the validation of the processor model used to support the reasoning. Model validation is crucial to ensure that formal analysis applies to the actual machine. Function execution provides an avenue for model validation. Model inspections can be enhanced by using the model to execute test vectors and comparing the results to the result of running the tests on the actual processor. Examples of this approach include the 68020 verification work [4], FM9001 verification [6], and Russinoff's recent floating-point verification work [15].

We are investigating how a model of a processor can be crafted that is amenable to formal microcode analysis and supports efficient simulation. We want the formal model to be validated by having microcode developers use it. We are using ACL2 for this work because its logic is a real programming language – an applicative subset of Common Lisp – that we hope can support our simulation needs. Our initial results seem promising: a prototype processor model written in ACL2 runs at about 90% of a similar model written in C with some optimizations we have developed [7].

The integration of simulator and analysis models has several advantages beyond simpler proofs and model validation. There is potential for the integration of symbolic simulation results into the development cycle. We are experimenting with this using a PVS-based symbolic simulation system that we have developed [8]. The availability of symbolic results may enable us to perform regression tests of designs more effectively. We hope in the longer term that our creation and use of formally analyzable models will allow us ultimately to verify formally aspects of our designs.

4 Integrating with the Engineering Environment

We are currently integrating formal models as "engines" into an existing development and debugging environment, replacing existing simulator artifacts written in C. Ultimately, we would like to have only one design artifact that can be used for synthesis, simulation, and formal analysis. In our experience, it is not difficult to teach a competent engineer to use formal languages such as PVS or ACL2 to describe digital systems, but we would rather avoid the time spent and errors introduced by manual translation of the design.

Hardware description languages such as VHDL and Verilog are commonly used for synthesis, but do not produce very efficient simulators, and do not have a formal basis. Recent work by Mark Bickford at ORA, however, promises to bridge the gap between VHDL and formal analysis [2]. An intriguing additional feature of the ORA toolset is its ability to excise an element of the design, and produce new VHDL with the element removed.

5 Improving the Hardware Reasoning Infrastructure

Many of the pioneering efforts in the use of theorem provers for industrial-scale digital design verification suffered from the lack of basic infrastructure needed to reason about hardware. For example, much effort in the AAMP-FV effort [11] was spent on developing an efficient bitvector library for PVS.

Engineers do not have much patience for proving basic mathematics facts in order to get their work done. We note for example that although ACL2 provides many benefits over Nqthm, there appears to be less automatic support for basic arithmetic. The research community could do a great service here by consolidating the basic work that has already been done, and making it part of the standard distributions for the various theorem provers.

6 Improving Speed and Scalability

The industrial theorem proving projects using PVS that we have undertaken at Rockwell Collins have all managed to tax its capabilities, especially as the size of the model grows, or the number of repetitions of a model execution increases. The highly automated, GRIND-heavy operations that we tend to employ in order to promote proof robustness appear to be atypical of most PVS users. However, the PVS developers have stepped up to the challenge, and delivered marked improvements in performance and scalability over the past year. We look forward to further improvements in areas such as rewriting speed as this tool continues to develop.

7 Conclusions

At Rockwell Collins, we are currently exploring ways in which formal models can improve our computer system development work. The proofs of correctness offered by theorem provers are an important capability, and we have discovered other cost-effective benefits of formal model development, including creation of unambiguous specifications, use of formal models as efficient simulators, integration of symbolic results into the design process, and use of symbolic results to aid regression testing. In the long term we hope to verify formally the most difficult aspects of our designs as part of the basic design cycle.

References

1. William R. Bevier, Warren A. Hunt Jr., J Strother Moore, and William D. Young. An approach to systems verification. *Journal of Automated Reasoning*, 5(4):411–428, December 1989.
2. Mark Bickford and Damir Jamsek. Formal specification and verification of VHDL. In Mandayam Srivas and Albert Camilleri, editors, *Formal Methods in Computer-Aided Design – FMCAD*, volume 1166 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.
3. Robert S. Boyer and J Strother Moore. Mechanized formal reasoning about programs and computing machines. In R. Veroff, editor, *Automated Reasoning and Its Applications: Essays in Honor of Larry Wos*. MIT Press, 1996.
4. Robert S. Boyer and Yuan Yu. Automated proofs of object code for a widely used microprocessor. *Journal of the ACM*, 43(1):166–192, January 1996.
5. Bishop Brock, Matt Kaufmann, and J Strother Moore. ACL2 theorems about commercial microprocessors. In Mandayam Srivas and Albert Camilleri, editors, *Formal Methods in Computer-Aided Design – FMCAD*, volume 1166 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.
6. Bishop C. Brock and Jr. Warren A. Hunt. The DUAL-EVAL hardware description language and its use in the formal specification and verification of the FM9001 microprocessor. *Formal Methods in System Design*, 11(1):71–104, July 1997.
7. David Greve, Matthew Wilding, and David Hardin. Efficient simulation using a simple formal processor model. Technical report, Rockwell Collins Advanced Technology Center, April 1998. (submitted for publication).
8. David A. Greve. Symbolic simulation of the JEM1 microprocessor. Technical report, Rockwell Collins, Inc., Cedar Rapids, IA, 1998. (submitted for publication).
9. David A. Greve and Matthew M. Wilding. Stack-based Java a back-to-future step. *Electronic Engineering Times*, page 92, January 12, 1998.
10. M. Kaufmann and J S. Moore. An industrial strength theorem prover for a logic based on Common Lisp. *IEEE Transactions on Software Engineering*, 23(4):203 – 213, April 1997.
11. Steven P. Miller, David A. Greve, Matthew M. Wilding, and Mandayam Srivas. Formal verification of the AAMP-FV microcode. Technical report, Rockwell Collins, Inc., Cedar Rapids, IA, 1996.
12. Steven P. Miller and Mandayam Srivas. Formal verification of the AAMP5 microprocessor: A case study in the industrial use of formal methods. In *WIFT'95: Workshop on Industrial-Strength Formal specification Techniques*, Boca Raton, FL, 1995. IEEE Computer Society.
13. J Strother Moore. *Piton – A Mechanically Verified Assembly-Level Language*. Kluwer Academic Publishers, 1996.
14. S. Owre, N. Shankar, and J. M. Rushby. *User Guide for the PVS Specification and Verification System (Beta Release)*. Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993.

15. David M. Russinoff. A mechanically checked proof of IEEE compliance of the floating point multiplication, division, and square root algorithms of the AMD-K7 processor. Available at www.onr.com/user/russ/david/, January 28 1998.
16. Matthew Wilding. A mechanically verified application for a mechanically verified environment. In Costas Courcoubetis, editor, *Computer-Aided Verification – CAV '93*, volume 697 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.
17. Matthew M. Wilding. Robust computer system proofs in PVS. In C. Michael Holloway and Kelly J. Hayhurst, editors, *LFM97: Fourth NASA Langley Formal Methods Workshop*. NASA Conference Publication, 1997. (http://atb-www.larc.nasa.gov/Lfm97/).
18. Alexander Wolfe. First Java-specific MPU rolls. *Electronic Engineering Times*, page 1, September 22, 1997.