Formal Verification of Microprocessors at AMD

April 6, 2002

Arthur Flatau
Matt Kaufmann (speaker)
David F. Reed
David Russinoff
Eric Smith
Rob Sumners

Advanced Micro Devices, Inc.

AMD, the AMD logo and combinations thereof, and AMD Athlon are trademarks of Advanced Micro Devices, Inc.

⌊ Today at AMD, page 1 ⌋

- Traditional simulation-based methods:
  - Block-level "whackers"
  - Full-chip directed tests written by hand
  - Full-chip test programs written by pseudo-random test generators
  - Various "checkers" monitoring simulation for potential bugs
- Boolean equivalence checking for comparing RTL (Register-Transfer Logic) models with custom gate-level models
  - Synthesis alone does not meet all our needs

⌊ Today at AMD, page 2 ⌋

• Formal verification using the ACL2 theorem-proving system

– Proofs of correctness of RTL floating-point modules

◦ Specifically, proofs are done on the output from translation tools applied to the RTL.

– Proofs of correctness of higher-level algorithms implemented in RTL

– Ongoing improvement of ACL2 itself and libraries of lemmas used to "program" ACL2

This talk focuses on theorem proving and the consideration of more automatic formal methods.

⌊ ACL2 ⌋

• ACL2 [1] is "A Computational Logic for Applicative Common Lisp"
(descendant of Boyer-Moore theorem prover)

• Authors: Matt Kaufmann and J Moore

• Interactive prover with induction, conditional rewriting, and decision procedures (arithmetic, equality, Boolean logic)

– "Programmed" with theorems proved by the user, usually stored as rewrite rules.

• Publicly available at:

`http://www.cs.utexas.edu/users/moore/acl2`

– Includes numerous papers and proof scripts, and links to ongoing work

• *[Plug]* ACL2 Workshop immediately follows this DCC Workshop.

⌊ Some AMD Formal Verification History ⌋

We have emphasized automated theorem proving.

• 1995–96: Division and square root algorithms for AMD-K5 microcode[3, 5]

• 1997–present: Proofs of floating-point algorithms and actual RTL that use ACL2 on the AMD Athlon^{TM} processor and its derivatives [6, 7, 8]

– We have a translator from our proprietary RTL to ACL2 [7] that enables RTL proofs.

• 2001: Completed some protocol-level proofs

⌊ Floating-point Verification, page 1 ⌋

A natural target for theorem provers [10, 4]

• Concise formal specifications relating outputs to inputs

• The RTL is *relatively* tractable.

– While the size of an FPU may be substantial, the logic tends to decompose by operation.

– The interfaces with other modules are smaller and simpler.

• Complexity of floating-point designs causes problems for other verification approaches.

– Testing alone may be inadequate.

– Decision procedures used in formal verification traditionally have capacity limitations, for example for multiplication and shifting.

⌊ Floating-point Verification, page 2 ⌋

We have addressed the verification of RTL models with increasing levels of complexity.

• Started with simple pipeline-based designs

• Conditional pipelines [2] allowed more complicated signal dependencies and the sharing of hardware among operations of different latencies.

• Current work involves RTL with feedback (especially state machines, which are used in the implementation of iterative algorithms).

⌊ Floating-point Verification, page 3 ⌋

Various tools besides ACL2 are involved in this verification effort.

• "Translator" (written in flex/bison/C++ and ACL2) takes RTL as input and generates forms in a Lisp-like target language for specifying state machine transitions.

– We have also written high-level specs directly in this target language.

• "Compiler" (written in ACL2) analyzes signal dependencies and pipeline structures and produces ACL2 definitions.

⌊ Floating-point Verification, page 4 ⌋

• Tools (written in ACL2) automate repetitive tasks by generating lemmas automatically from the RTL:

– Lemmas about bit-vector widths

– Lemmas used in reasoning about conditional pipelines [2]

– Lemmas connecting different models (combinational and executable)

• ACL2 library of general reusable lemmas [9] has been designed to simplify terms built from RTL operations, in many cases automatically.

– Development continues on the RTL library, with users inside/outside of AMD [10].

⌊ Protocol-level Verification, page 1 ⌋

Formal verification of non-floating-point RTL can be considerably more difficult.

- Unclear and incomplete (or nonexistent) specs
- Decomposition of verification task is far more difficult.
  - Sufficient invariants often involve every state variable, and significant and complex environment assumptions are required.
- Experimental formal analysis of a bus interface unit (many thousands of lines of RTL)
  - instrumental in resolving a subtle liveness issue
  - limited practical value
- Higher-level proof attempt on cache correctness
  - Partially completed, but appeared to have limited payoff relative to the effort involved

⌊ Protocol-level Verification, page 2 ⌋

We completed proofs when the effort seemed justified.

- Proof of a write-ordering property with respect to a fairly sophisticated mechanism

  – Proof performed at algorithm level. Abstracted numerous uninteresting details. Formal analysis more effectively focused at subtle cases.

  – Informal statement: If processor P1 performs write Wr(addr1) followed by write Wr(addr2), and processor P2 performs reads at address addr2 and then addr1, then if the read at addr2 gets the new value, so does the write at addr1.

- Proof of progress for a routing module

  – The proof was performed on a model which generalized the RTL (i.e., the RTL was functionally equivalent to an instance of the model). The model was defined with recursive functions and data structures, which provided a much more expressive "language" for defining invariants, refinement maps, etc.

⌊ Attempts at Model Checking ⌋

We have begun looking at model checking and symbolic simulation, but initial results are lackluster.

• Our designs are in a proprietary language, which is not an input language for existing Model Checkers.

  – Translator output is often difficult for a human to read.

• Attempts at using symbolic simulation were ineffective due to incompleteness of search.

  – In order to expose bugs, we need to simulate for hundreds of cycles and simulation becomes inefficient much sooner than this.

• Attempts at using Bounded Model Checking have been more effective, but the property definition complexity is considerably higher.

  – Must explicitly define strengthened invariants.

  – Multiple modules expose expressiveness and capacity issues.

⌊ Problems ⌋

- Modules are large (many thousands of lines).

- RTL is not written in a standard language.

- It takes effort to develop meaningful specifications, which are not always readily supplied by the RTL developers.

- Is formal verification cost-effective?

  – RTL writers have told us that any value added would appear to be in verification involving interfaces among multiple large modules.

  – Time to write specs is a real issue, but has some support among the RTL designers.

  – We developed a simple checker (written in ACL2) for some sorts of typos that have been seen during pre-silicon RTL.

- Capacity, Capacity, Capacity...

References

[1] Kaufmann, M, Manolios, P, Moore, J S. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Press, 2000.

[2] Kaufmann, M. and Russinoff, D., "Verification of Pipeline Circuits," in Proceedings of ACL2 Workshop 2000. Available at URL `http://www.cs.utexas.edu/users/moore/acl2/-workshop-2000/final/russinoff-kaufmann/paper.pdf`.

[3] Moore, J, Lynch, T., and Kaufmann, M., "A Mechanically Checked Proof of the Correctness of the Kernel of the $AMD5_K86$ Floating Point Division Algorithm", *IEEE Transactions on Computers*, 47:9, September, 1998.

[4] O'Leary, J., Zhao, X., Gerth, R., and Seger, C-J.H.,"Formally Verifying IEEE Compliance of Floating-Point Hardware", *Intel Technology Journal*, 1999.

[5] Russinoff, D., "A Mechanically Checked Proof of IEEE Compliance of the AMD-K5 Floating Point Square Root Microcode", *Formal Methods in System Design* 14:1, January 1999. See URL `http://www.onr.com/user/russ/david/fsqrt.html`.

[6] Russinoff, D., "A Mechanically Checked Proof of IEEE Compliance of the AMD-K7 Floating Point Multiplication, Division, and Square Root Algorithms", *Journal of Computation and Mathematics* 1, London Math. Society, December 1998. See URL `http://www.onr.com/user/russ/david/-k7-div-sqrt.html`.

[7] Russinoff, D. and Flatau, A., "RTL Verification: A Floating-Point Multiplier", in Kaufmann, M., Manolios, P., and Moore, J, eds., *Computer-Aided Reasoning: ACL2 Case Studies*, Kluwer Academic Press, 2000. See URL `http://www.onr.com/user/russ/david/acl2.html`.

[8] Russinoff, D., "A case study in formal verification of register-transfer logic with ACL2: The floating point adder of the AMD Athlon^{TM} Processor, in Hunt, Warren A. Jr. and Johnson, Steven D., eds., *FMCAD 2000, Proceedings of Third International Conference on Formal Methods in Computer-Aided Design*, Springer, November, 2000.

[9] Russinoff, D. and Smith, E., "An ACL2 Library of Floating-Point Arithmetic", to appear (earlier version at URL `http://www.cs.utexas.edu/users/moore/-publications/others/fp-README.html`).

[10] Sawada, J., "Formal Verification of Divide and Square Root Algorithms Using Series Calculation", to appear in Proceedings of ACL2 Workshop 2002.