Hardware Verification with PVS

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

Conclusions

- Need *massive* automation to make industrial hardware verification practical
- General purpose verification tools (theorem provers) don't have enough automation
- Specialized tools (model checking, BDD-equivalence checking) are too brittle, solve only parts of the problem, don't support expressive specification languages
- Current project: combined approach
  - Specialized tools as decision procedures in a theorem prover
- Future: need a federated approach
  - Environment with many specialized components
  - Can be used together, new ones synthesized quickly
  - Theorem prover to coordinate them

PVS

- Verification system developed by SRI beginning in 1990, derived from earlier SRI systems, including (NSA-sponsored) EHDM
- Specification language is a higher-order logic with parameterized theories and a rich type system (predicate subtypes, dependent types, recursive abstract data types)
- Lowest levels of theorem prover do automated rewriting, provide decision procedures for ground equality with uninterpreted function symbols, linear arithmetic, arrays, propositional calculus
- Higher levels provide user-customizable strategies (tactics), interactive guidance using sequent calculus presentation
- ARPA-sponsored enhancements provide BDD-based propositional simplifier, decision procedure for μ-calculus and CTL model-checking

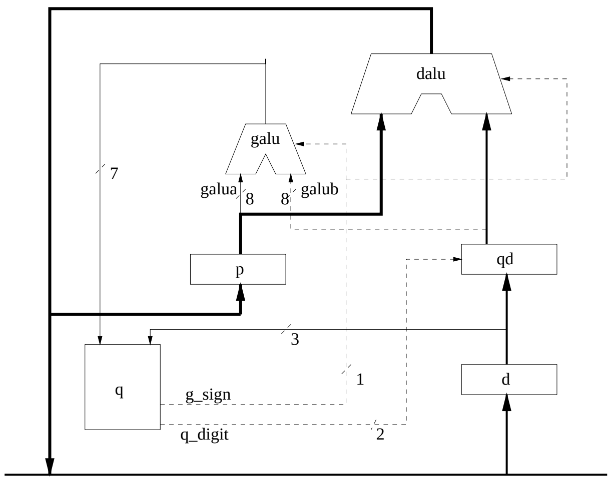
SRT Division: Derivation and Specification

- The famous Pentium FDIV bug was in the quotient lookup table of a radix-4 SRT divider
- SRT now something of a test-piece for formal verification
- PVS treatment specifies and verifies the general theory of SRT division and derives the conditions for the radix-4 lookup table
- Then verifies a particular table (similar to the Pentium's) due to Taylor (1981)
  - Need to show blank entries are inaccessible
    - ⋆ This was the FDIV bug
    - ⋆ PVS specification uses the `TABLE` construct, which automatically generates proof obligations (TCCs) for blank entries
  - And other entries have the right values

Quotient Lookup Table

```
q(D: bvec[3], (P: bvec[7] | estimation_bound?(valD(D), valP(P)))): subrange(-2, 2) =
LET a = -(2 - P(1) * P(0)), b = -(2 - P(1)), c = 1 + P(1), d = -(1 - P(1)), e = P(1),
    Dp:nat = bv2pattern(D), Ptruncp:nat = bv2pattern(P^(6,2))
IN TABLE Ptruncp, Dp
                |[ 000| 001| 010| 011| 100| 101| 110| 111]|
             %-------------------------------------------%
             |01010|    |    |    |    |    |    |    | 2 ||
             |01001|    |    |    |    |    | 2  | 2  | 2 ||
             |01000|    |    |    |    | 2  | 2  | 2  | 2 ||
             |00111|    |    | 2  | 2  | 2  | 2  | 2  | 2 ||
             |00110|    | 2  | 2  | 2  | 2  | 2  | 2  | 2 ||
             |00101| 2  | 2  | 2  | 2  | 2  | 2  | 2  | 1 ||
             |00100| 2  | 2  | 2  | 2  | c  | 1  | 1  | 1 ||
             |00011| 2  | c  | 1  | 1  | 1  | 1  | 1  | 1 ||
             |00010| 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1 ||
             |00001| 1  | 1  | 1  | 1  | e  | 0  | 0  | 0 ||
             |00000| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0 ||
             |11111| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0 ||
             |11110| -1 | -1 | d  | d  | 0  | 0  | 0  | 0 ||
             |11101| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 ||
             |11100| a  | b  | -1 | -1 | -1 | -1 | -1 | -1 ||
             |11011| -2 | -2 | -2 | b  | -1 | -1 | -1 | -1 ||
             |11010| -2 | -2 | -2 | -2 | -2 | -2 | b  | -1 ||
             |11001| -2 | -2 | -2 | -2 | -2 | -2 | -2 | -2 ||
             |11000|    |    | -2 | -2 | -2 | -2 | -2 | -2 ||
             |10111|    |    |    | -2 | -2 | -2 | -2 | -2 ||
             |10110|    |    |    |    |    | -2 | -2 | -2 ||
             |10101|    |    |    |    |    |    | -2 | -2 ||
             %-------------------------------------------%
    ENDTABLE
```


SRT Division: Datapath



SRT Division: Correctness

Need to show quotient selection logic and lookup table satisfy the predicate

```
lookup_legitimate?(q,D,(P:rational | estimation_bound?(D, P))): bool =
  COND
   q =  2 -> 4/3 * (D + delta) <= P,
   q =  1 -> 1/3 * (D + delta) <= P & (P + eps) <= 5/3 * D,
   q =  0 -> -2/3 * D <= P & (P + eps) <= 2/3 * D,
   q = -1 -> -5/3 * D <= P & (P + eps) <= -1/3 * (D + delta),
   q = -2 -> (P + eps) <= -4/3 * (D + delta)
  ENDCOND
```


SRT Division: Verification

- The following PVS proof script generates 256 subgoals and proves all but one of them automatically

```
(then (skosimp* :preds? t)
      (install-rewrites :defs !)
      (case* "P!1(6) = 0" ...   "P!1(2) = 0")
      (case* "D!1(2) = 0" "D!1(1) = 0" "D!1(0) = 0")
      (repeat (forward-chain "bool_chain"))
      (replace*)
      (assert))
```

- Remaining subgoal has a nonlinearity and needs an extra case split on the remaining bits `P!1(1)` and `P!1(0)`
- TCCs are proved similarly

SRT Division: Lessons

- General design decisions in PVS work beautifully for this example
  - Compelling deriviation of overall theory and specification of specific algorithm (integration of language with prover)
  - Succinct, largely automated proofs for all steps (integration of rewriting, decision procedures, BDDs)
- However, proof for quotient lookup table runs for 3 hours on Sparc 10
- Need to retain basic design of language, high-level prover, and overall integration, while gaining three orders of magnitude speed improvement in low-level automation

Technology Transfer Projects

- SRI worked with Collins Commercial Avionics to apply formal methods to Collins avionics processors
- Projects sponsored by NASA Langley Research Center
- Goal: transfer SRI technology to Collins (and for SRI to learn how our technology scaled to real examples)
- First project (two years): specification and partial verification of microcode of (complex) AAMP5 microprocessor
  - SRI in the lead
- Second project (in progress): complete specification and verification of (simpler) AAMP-FV microprocessor
  - Collins in the lead

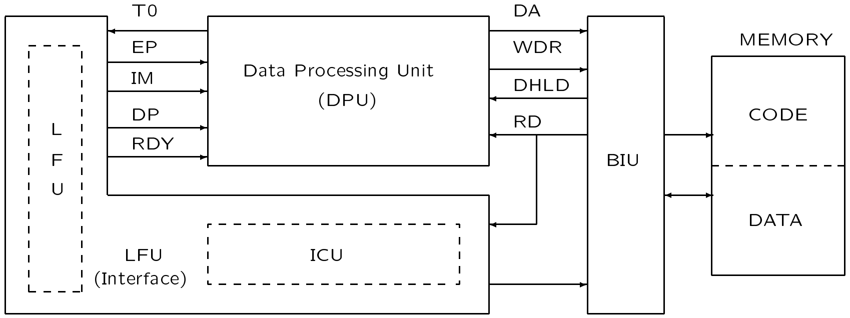
AAMP5 Microprocessor

- Not designed for formal methods, but to provide four-fold performance improvement over AAMP2
- Numerous applications—e.g., 30 AAMP2s on each 747-400
- Used for critical, but not ultra-critical, applications (e.g., instrumentation and displays, not autoland)
- Derived from CAPS architecture, dating back to 1974
  - When hardened memories were very small—needed high density for code and data
- So uses stack architecture (many zero-address instructions)
- Intended for embedded applications and code generated by high-level languages (Ada)—provides hardware support for runtime environment and real-time executive (209 instructions)
- Very different from RISC architectures

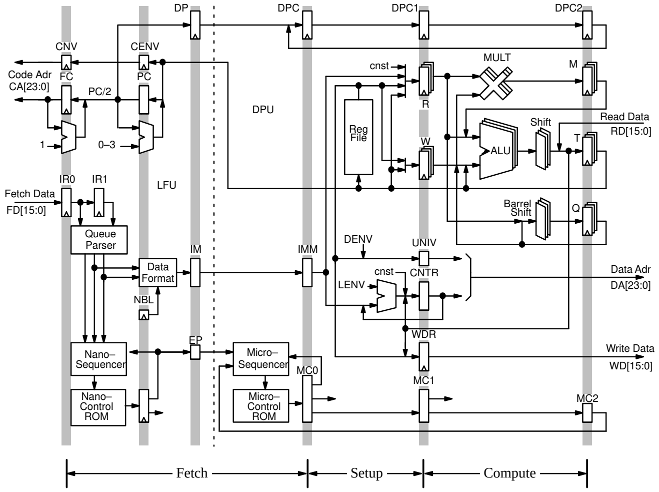
AAMP5 Microprocessor: Complexity

- Microcoded implementation
- Variable length instructions
- Also multiple word sizes
  - Complex bitvector operations outside ALU
- The upper portion of the stack is cached in processor registers
  - Automatically adjusted prior to each instruction–takes variable amount of time
- Three-stage pipeline with data and control hazards
- Semi-autonomous subunits for asynchronous instruction and data prefetch, and memory and bus transfers
- Implementation has 500,000 transistors

AAMP5 Block Diagram



AAMP5 Datapath



AAMP5 Microprocessor: Comparison with RISC

- The AAMP verifications are technically challenging
  - Variable instruction timings
  - Stack cache not completely transparent
  - Complex instructions, microcode
  - Complex bitvector operations outside ALU
- A RISC processor (single pipe) would actually be easier
  - Our automation would deal with the deeper pipe
- But difficult to convince industry of that—they see AAMP as an idiosyncratic design and assume its verification does not address their concerns

Technical Goals

- Formally specify as much of the AAMP5 macroarchitecture and instruction set as feasible (actually did 108 of 209 instructions)
- Formally specify the microarchitecture of the AAMP5 DPU at the RTL level
  - Abstract the DPU environment: interaction with LFU and BIU characterized by rely-guarantee assertions
- Formally verify a core set of 11 instructions representing the major instruction classes
  - I.e., prove that execution of microcode by microarchitecture achieves specified behavior for that instruction
  - Collins seeded subtle errors in microcode for two instructions (unknown to SRI)

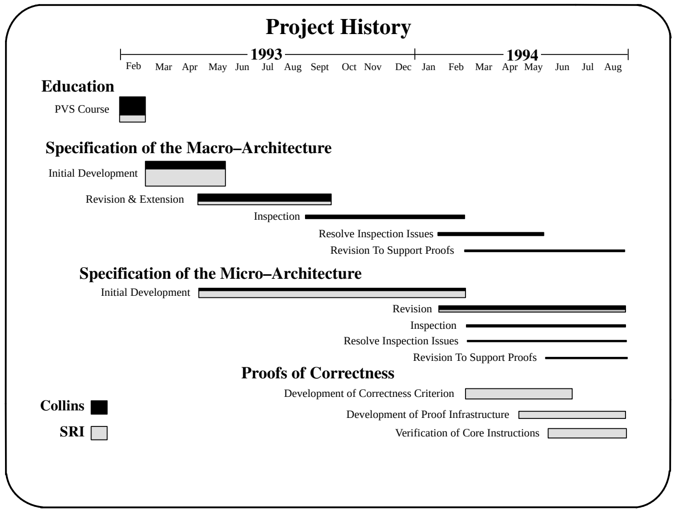
Programmatic Goals

- Assess feasibility of formal verification for industrial use
- Determine cost and value
- Learn how to integrate with traditional process
- Transfer technology to Collins' engineers
- Learn what was needed to achieve true "industrial strength" if formal hardware verification

Process

- Run as a "shadow" project
- Undertaken incrementally in order to involve Collins' engineers and to provide feedback to traditional development
  - SRI would make first pass at each step in the process
  - Collins would polish it, and then do more of the same
- Collins subjected formal specifications to their traditional inspection process

Timeline and Effort



Formal Specifications

Macroarchitecture: 480 hours by SRI, 675 by Collins

- Initial development done by SRI
  - 1,595 Lines of PVS and comments in 25 theories
- Revised and extended by Collins
  - Specified most of the executive service routines
  - Completed 108 of the AAMP's 209 instructions
  - 2,550 lines of PVS and comments in 48 theories

Microarchitecture: 640 hours by SRI, 479 by Collins

- Initially done by SRI, revised by Collins
- 2,679 lines of PVS and comments organized into 20 theories

Bitvector library: Initially by SRI, taken over by NASA Langley

- 2,030 lines of PVS and comments in 31 theories

Inspections of Specifications

- Formal inspections performed at Collins by between 3 and 5 practicing electrical engineers
- Inspectors had little trouble reading PVS (learned by sitting in)
  - Average preparation rate 150 lines of PVS and comments per hour
  - Inspection rate about 330 lines per hour
- Performed 11 inspections of macroarchitecture specification
  - Found 53 style errors, 28 correctness errors
- Performed 10 inspections of microarchitecture specification
  - Found 64 style errors, 19 correctness errors

Microcode Errors Found During Formal Specification

- Discovered two new microcode errors
- Found while specifying behavior of the AAMP in unusual cases
- One was due to a missing requirement
  - Would have been found during Ada validation testing
- Other was a coding error
  - Required specific stack configuration, an improperly sized stack, and a specific instruction sequence
- Both specific to the AAMP5 and corrected before first fabrication
- Collins had not completed traditional verification at that time

Formal Verification

- Performed entirely by SRI
- 800 hours
- Developed reusable proof infrastructure
- Formally verified 11 instructions representing the major instruction classes
- Found errors in specifications that had escaped inspections
- Also found the two seeded bugs
  - First was designed to be difficult to detect
  - Second was an actual error that had escaped detection by simulations and walkthroughs; discovered by Collins while running application code on an early prototype of AAMP5
- Discovery was very systematic and SRI was able to explain how to correct both errors

Follow-On: AAMP-FV

- Collins were enthusiastic about formal specification
- But initially skeptical of formal verification
- The way the seeded bugs were found persuaded them it was feasible and had real value
- But decided to undertake new project rather than push through with AAMP5 verification
  - Better suited Collins business case
  - Move tech transfer to a new stage: Collins engineers take the lead, SRI researchers serve as consultants (also do the complex instructions)
  - With startup and learning costs written off to AAMP5, clean start would allow better estimate of costs on real project

AAMP-FV

- Simpler processor, closer to one for ultra-critical applications (e.g., autolanders, 777 autopilot)
- No pipelining, nor autonomous subunits
- 80 instructions
  - 55 have straightline microcode
  - 5 are complex (very long microcode–OS functions etc.)
  - 20 are challenging (iterative microcode for shifts, divide, complex intermediate states for multiply, etc.)
- About 100,000 transistors

AAMP-FV Status

- Microarchitecture formal specification completed by Collins in 50 hours (cf. over 1,100 for AAMP5)
- Macroarchitecture specification also completed by Collins
- Formal specifications are the principal design documentation
- Formal verification is the principal debugging method
- 50 (of 55) straightline instructions verified so far, by Collins following strategies developed by SRI
- 3 (of 20) challenging instructions verified by SRI in developing suitable strategies

Lessons Learned: Programmatic

- It takes a long time to develop mutual understanding and trust
- On the job training better than courses in formal methods
- Target problem has to be big and real to interest engineers
- Likely to require methodology development, tool enhancements
- Must do this concurrently with tech-transfer
- High risk: difficult to predict schedule and outcome
- Governm't funding essential to get through this difficult period
- And a committed champion in the receiving organization
- Initial skepticism about feasibility of proof is a big barrier
- But eventually discover mechanized proof is the primary source of benefit

Lessons Learned: Technical

- Again, general design decisions in PVS worked well
  - Collins engineers accept PVS specifications
  - Able to undertake very large proofs, provided approach is well-developed
- However process still requires too much skill and patience
  - Proofs are far too slow (5–15 minutes)
  - Far too much effort goes into controlling size of intermediate terms
  - Insufficiently powerful automation makes challenging instructions too hard
- Need to retain basic design of language, high-level prover, and overall integration, while gaining three orders of magnitude speed improvement and scaling in low-level automation

The Future

- Existing decision procedures need massive speedups and better scaling
- Also need new decision procedures (bitvectors)
- Need richer control of low-level automation (when can let rip, vs. when must hold on tight leash)
- Need new automation to control large problems (help with invariants, abstraction, use of model checking inside proofs)
- Overall: a federation of powerful and efficient individual techniques (decision procedures, model checking)
  - That work together
  - Coordinated by a theorem prover

References

Process: *Formal Verification of the AAMP5 Microprocessor: A Case Study in the Industrial Use of Formal Methods*, by Steven P. Miller and Mandayam Srivas. In proceedings of WIFT '95, Boca Raton, Florida USA, April 5-8, 1995, pp. 30–43. `http://www.csl.sri.com/wift95.html`

Technology: *Applying Formal Verification to a Commercial Microprocessor*, by Mandayam Srivas and Steven P. Miller. In proceedings CHDL '95 Languages, Chiba, Japan, August 1995, pp. 597–603. (Best Paper award). `http://www.csl.sri.com/chdl95.html`

Full details: *Formal Verification of the AAMP5 Microprocessor* by Mandayam K. Srivas and Steven P. Miller, pp. 125–180 in *Applications of Formal Methods*, Michael G. Hinchey and Jonathan P. Bowen Eds., Prentice Hall 1995. `http://www.csl.sri.com/csl-95-4.html`(Tech Report)

People Who Did The Work

PVS was built by Sam Owre and Natarajan Shankar

SRT divider: Harald Ruess (Ulm), Natarajan Shankar, and Mandayam Srivas

AAMP processors:

SRI: Mandayam Srivas

Collins: Steve Miller (Collins engineer of the Year), Al Mass, Dave Greve, with others participating in reviews

For more information about PVS: `http://www.csl.sri.com/pvs.html`