N96-10031

The Formal Verification Used for the AAMP5 and AAMP-FV

59515

P-7

It is becoming increasingly evident within the VLSI design industry that the complexity of many current hardware designs is outstripping the capability of traditional simulation-based tools to adequately verify them. This situation was well-illustrated by the recent floating point bug discovered in Intel's Pentium processor. The industry is beginning to look at formal verification as a technological alternative to simulation for obtaining higher assurance than is currently possible.

Recently, SRI International and Collins Commercial Avionics, a division of Rockwell International, undertook a project to explore how formal techniques for specification and verification could be introduced into an industrial process. The project, sponsored by the Systems Validation Branch of NASA Langley and Collins Commercial Avionics, consisted of specifying in the PVS language a portion of a Rockwell proprietary microprocessor, the AAMP5, at both the instruction set and register-transfer levels and using the PVS interactive proof-checker to show that the microcode correctly implemented the specified behavior for a representative subset of instructions.

The main goal of the project was two-fold: First, to investigate the feasibility of formally specifying and verifying a complex commercial microprocessor that was not expressly designed for formal verification. Second, to explore effective ways to transfer the technology to an industrial setting. The choice of the AAMP5 satisfied the first goal since the AAMP5 was not designed for formal verification, but to provide a more than threefold performance improvement while remaining object-code-compatible with the earlier AAMP2, which is used in numerous avionics applications, including the Boeing 737, 747, 757, and 767.

To satisfy the technology transfer objective, we had to develop a suitable verification methodology and a formal infrastructure to make the technology usable by practicing engineers. This infrastructure includes techniques for decomposing the microprocessor verification problem into a set of verification conditions that the engineers can formulate and strategies to automate the proof of the verification conditions. The development of the infrastructure was one of the key accomplishments of the project. Most of the infrastructure and methodology are general enough to be reused for other microprocessors, certainly in the verification of another member of the AAMP family. This methodology was used to formally specify the entire microarchitecture and more than half of the instruction set and to verify a core set of eleven AAMP5 instructions representative of several instruction classes. However, the methodology and the formal machinery developed are adequate to cover most of the remaining AAMP5 instructions. Although PVS was the vehicle of the experiment, the methodology is applicable to other sufficiently powerful theorem provers.

Another key result of the project was the discovery of both actual and seeded errors. Two actual microcode errors were discovered during development of the formal specification, illustrating the value of simply creating a precise specification. Both were specific to the AAMP5 and were corrected before first fabrication. Two additional errors seeded by Collins in the microcode were systematically uncovered by SRI, who knew that bugs had been seeded, but not their location or identity, while doing correctness proofs. One of these was an actual error that had been discovered by Collins after first fabrication but left in the microcode provided to SRI. The other error was designed to be unlikely to be detected by walk-throughs, testing, or simulation.

Steve Miller's talk earlier in the workshop, gave an overview of the AAMP5 project emphasizing the technology transfer process with its administrative and managerial aspects. This talk describes the technical approach used in verifying the AAMP5. Please refer to Steve Miller's slides for the AAMP5 design figures.

The Formal Verification Technology Used for the AAMP5 and AAMP-FV*

Mandayam Srivas

Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
(srivas@csl.sri.com)
(http:\\www.csl.sri.com)

*Joint work with Steve Miller, Rockwell International, Collins Commercial Avionics, Cedar Rapids, Iowa 52498

Organization of the Talk

- Overview of AAMP5 Processor
- What did we verify?
- How did we manage complexity of verification?
- Mechanization of proofs
- Conclusions

The Stack Memory Model

Macromachine Specfication

```
ADD_lemma_1: LEMMA
  LET X = current_data_env(st)(tos(st)+1),
      Y = current_data_env(st)(tos(st))
  IN current_opcode(st) = ADD & NOT arith_update.exception(st) =>
     normal_macro_machine.next_macro_state(st) =
       st WITH [(dmem)(word2denv(denv(st)))(tos(st)+1) := X + Y,
                (pc)  := pc(st)  + 1,
                (tos) := tos(st) + 1]
```

DPU Environment Assumptions

- "If RDY is false then RDY will eventually become true provided the DPU obeys its end of the protocol."

```
CLR_RDY_wait: AXIOM
   NOT RDY(t) & normal_fetching(t) & CLR_RDY?(NCO(t)) =>
      (EXISTS (t1 | t1 > t):
          stays_same(NCO(t))(t, t1)  =>
            stays_same(RDY)(t, t1-1) & RDY(t1) &
               normal_fetching(t1) &
                 DP(t1) = DP(t)+length_of_instruction(opcode) )
```

5

DPU Environment Assumptions

- "If RDY is true then decoded outputs presented by the LFU correspond to the instruction stored at the PC presented by the LFU."

```
% Invariant constraints on the outputs when LFU is RDY
RDYinv: AXIOM
   RDY(t) =>
     LET opcode = opcode_at(CENV(t), DP(t), CODE_MEMORY(t))
     IN  EP(t) = EP_ROM(opcode) & (has_imm_data(opcode)
           => IM(t) = first_unit_of_imm_data(t))
```

6

Sample Register-Transfer Level Specification

```
SVax: AXIOM SV(t+1) = IF DHLD(t) THEN SV(t) ELSE NEXT_SV(t) ENDIF

SV1ax: AXIOM SV1(t+1) = IF DHLD(t) THEN SV1(t) ELSE SV(t) ENDIF

TVax: AXIOM TV(t+1) = IF DHLD(t) THEN TV(t) ELSE NEXT_TV(t) ENDIF

TV1ax: AXIOM TV1(t+1) = IF DHLD(t) THEN TV1(t) ELSE TV(t) ENDIF

END SVTVpipeline_axioms
```

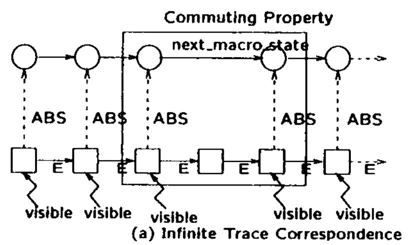
7

Other Microprocessor Verification Efforts

- FM8501 [Hunt: 1986]
- Viper [Cohn: 1988]
- Tamarack [Joyce: 1988]
- MiniCayuga [Srivas & Bickford: 1990]
- Saxe's Pipeline [Saxe, et. al. 1991]
- DLX pipeline [Burch & Dill: 1993]
- UNITA [Windley: 1994]
- ...

8

General Microprocessor Correctness



(a) Infinite Trace Correspondence

- Abstraction function (ABS)
- Visible state

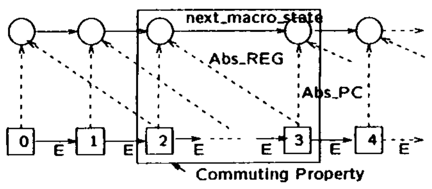
9

Complications posed by AAMP5

- Pipelining
- Autonomous prefetch and data transfers
- The "stack cache" abstraction

10

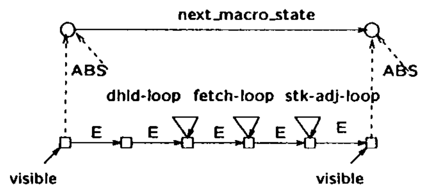
Pipelined Microprocessor Correctness



- Abstraction function must be suitably "skewed"
- Length of instruction cycle can be idefinite not necessarily a function of the current visible state

11

AAMP5 Pipeline Correctness



12

Specializing Correctness Criterion to the AAMP5

- Visible state:
  - The first microinstruction of the *current* macroinstruction is loaded into MC0 (microregister)
  - The current (compute-stage) instruction is guaranteed to advance
  - The previous (write-stage) instruction in the pipeline is guaranteed to complete in the next cycle
- Abstraction function
  - A projection function except for data memory
  - The data memory at the macro level is real memory with the overlaid "stack registers"

13

Formal Correctness Statement

```
commuting_property: LEMMA
  visible_state(t) & proper_instrns_in_pipe(t)
          & no_logical_stack_overflow(t)
    => EXISTS (tp: time | tp > t):
          stays_low(t+1, tp-1)(visible_state) &
             visible_state(tp) & ABS(tp) = next_macro_state(ABS(t))

visible_state(t): bool =
  MCO(t) = ROM(zero_extend[8](11)(EP_ROM(current_op(t))(t))) &
     & NOT(DHLD(t)) & NOT(nextDJMP(t))
         & entry_cond_met(current_op(t))(t)
```

14

Our Approach to the Verification Task

- Incremental verification: Structure it so as to
  - get early feedback
  - Collins staff can participate in the proof process
- Abstract the DPU environment: A set of "rely-guarantee" assertions characterizing the interactions for the DPU with the other units.
- Decompose the proof into
  - an automatic part: *instruction-specific verification conditions*
  - interactive part: *general verification conditions*

15

General Verification Conditions

- `dhld_lemma`:
  "NOT DHLD $\Rightarrow$
  $\diamond$ (DHLD & macrostate unchanged)"
- `LFU_not_rdy_lemma`:
  "NOT RDY $\Rightarrow$
  $\diamond$ (RDY & next instruction moves in)"
- `stack_adjust_lemma`:
  "NOT entry_condition_met $\Rightarrow$
  $\diamond$ (entry_condition_met & macrostate unchanged"

16

Instruction-specific Verification Conditions

```
TO_correctness: LEMMA
   visible_state_with(ADD)(t) & SysInv(t) =>
      TO(t+2) =
        (sign_extend[16](48)(ith_top_of_scache(t+1)(1))
          + sign_extend[16](48)(ith_top_of_scache(t+1)(0)))^(15,0)

i: VAR below[10]
REG_correctness: LEMMA
   visible_state_with(ADD)(t) & SysInv(t) =>
                  REG(t + 2)(i) = REG(t + 1)(i)
```

Mechanization of proofs of Verification Conditions

- Overview of PVS
- Our core hardware strategy

The Core Startegy Used

For proving theorems of the form:

"Precond(s0) $\Rightarrow$ $f(s0) \;=\; g(E^i(s0))$, for some constant i"

- Rewrite expressions in the theorem using the functions in the design specification as auto-rewrite rules.
  `((repeat (do-rewrite)))`
- "Lift" all the if-then-else structures to the topmost level
  `((repeat (lift-if)))`
- Simplify using a BDD simplifier, arithmetic, and other decision procedures along with using a library of bit-vector properties as auto-rewrite rules.
  `((then* (bddsimp) (assert)))`

Mechanization overview

- Instruction-specific VCs: Core hardware strategy
- General VCs: Inductions with core strategy for the base and inductive steps.
- Main correctness theorem: Chaining of VCs with intruction-specific VCs, definition of abstraction function, and macro specification used as rewrite rules.

Conclusions

- Is commercial processor verification technically feasible?
  Yes, if carefully planned and executed.
- Can practicing engineers use this technology?
  Yes, with appropriate training and expert help.
- Is it "cost-effective" ?