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) )
```


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))
```

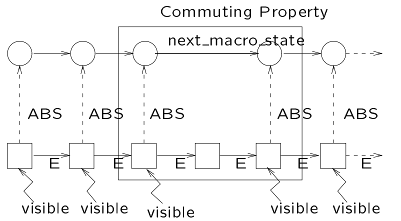

Sample Register-Transfer Level Specification

```
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
```


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]
- ...

General Microprocessor Correctness



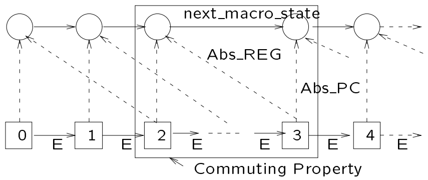
(a) Infinite Trace Correspondence

- Abstraction function (ABS)
- Visible state

Complications posed by AAMP5

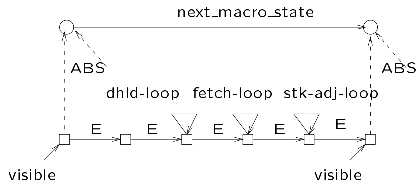
- Pipelining
- Autonomous prefetch and data transfers
- The "stack cache" abstraction

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

AAMP5 Pipeline Correctness



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"

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 =
  MC0(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)
```


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*

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"

Instruction-specific Verification Conditions

```
T0_correctness: LEMMA
   visible_state_with(ADD)(t) & SysInv(t) =>
      T0(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" ?