Today

- Specification/Implementation
- Abstraction Functions
- Correctness Condition
- Verification
- Self-Consistency

Specification

- Abstract from Implementation
- Describes observable/correct behavior

Implementation

- Some particular embodiment
- Should have same observable behavior
  - Same with respect to important behavior
- Includes many more details than spec.
  - How performed
  - Auxiliary/intermediate state

Important Behavior

- Same output sequence for input sequence
  - Same output after some time?
- Timing?
  - Number of clock cycles to/between results?
  - Timing w/in bounds?
- Ordering?

Abstraction Function

- Map from implementation state to specification state
  - Use to reason about implementation correctness
  - Want to guarantee: AF(Fi(q,i))=Fs(AF(q),i)
Familiar Example

- Memory Systems
  - Specification:
    - \( W(A,D) \)
    - \( R(A) \rightarrow D \) from last \( D \) written to this address
  - Specification state: contents of memory
  - Implementation:
    - Multiple caches, VM, pipelined, Write Buffers...
    - Implementation state: much richer...

Memory AF

- Maps from
  - State of caches/WB/etc.
- To
  - Abstract state of memory
- Guarantee \( AF(Fi(q,I)) = Fs(AF(q),I) \)
  - Guarantee change to state always represents the correct thing

Abstract Timing

- For computer memory system
  - Cycle-by-cycle timing not part of specification
  - Must abstract out
- Solution:
  - Way of saying "no response"
    - Saying "skip this cycle"
    - Marking data presence
      - (tagged data presence pattern)

Filter to Abstract Timing

- Filter input/output sequence
  - \( Os(in) \rightarrow out \)
- FilterStall(\( Impl_{in} \)) = in
- FilterStall(\( Impl_{out} \)) = out
- For all sequences \( Impl_{in} \)
  - FilterStall(Oi(\( Impl_{in} \))) = Os(FilterStall(\( Impl_{in} \)))

DLX Datapath

- DLX unpipelined datapath from H&P (Fig. 3.1 e2, A.17 e3)

Processors

- Pipeline is big difference between specification state and implementation state.
- Specification State:
  - Register contents (incl. PC)
  - Memory contents
Revised Pipeline

DLX repipelined datapath from H&P (Fig. 3.22 c2, A.24 e3)

Processors

- Pipeline is big difference between specification state and implementation state.
- Specification State:
  - Register contents (incl. PC)
  - Memory contents
- Implementation State:
  + Instruction in pipeline
  + Lots of bits
  - Many more states
  - State-space explosion to track

Observation

- After flushing pipeline,
  - Reduce implementation state to specification state
- Can flush pipeline with series of NOPs or stall cycles

Pipelined Processor Correctness

- w = input sequence
- w_f = flush sequence
  - Enough NOPs to flush pipeline state
- For all states q and prefix w
  - $F_i(q, w_{\text{w}f}) \Rightarrow F_s(q, w_{\text{w}f})$
  - $F_i(q, w_{\text{w}f}) \Rightarrow F_s(q, w)$
- FSM observation
  - Finite state in pipeline
  - Only need to consider finite w

Pipeline Correspondence

[Burch+Dill, CAV’94]
Equivalence

• Now have a logical condition for equivalence
• Need to show that it holds
  – Is a Tautology
• Or find a counter example

Ideas

• Extract Transition Function
• Segregate datapath
• Symbolic simulation on variables
  – For q, w's
• Case splitting search
  – Implication pruning

Extract Transition Function

• From HDL
• Similar to what we saw for FSMs

Segregate Datapath

• Big state blowup is in size of datapath
  – Represent data symbolically/abstractly
    • Independent of bitwidth
  – **Not verify** datapath/ALU functions as part of this
    • Can verify ALU logic separately using combinational verification techniques
    • Abstract/uninterpreted functions for datapath

Burch&Dill Logic

• Quantifier-free
• Uninterpreted functions (datapath)
• Predicates with
  – Equality
  – Propositional connectives

B&D Logic

• Formula = \( \text{ite}(\text{formula}, \text{formula}, \text{formula}) \)
  | (\(\text{term}=\text{term}\))
  | \(\text{psym}(\text{term},...\text{term})\)
  | \(\text{pvar} | \text{true} | \text{false}\)
• Term = \(\text{ite}(\text{formula},\text{term},\text{term})\)
  | \(\text{fsym}(\text{term},...\text{term})\)
  | \(\text{tvar}\)
Sample

• Regfile:
  – (ite stall
    regfile
    (write regfile
     dest
     (alu op
      (read regfile src1)
      (read regfile src2)))))

Example Logic

• arg1:
  – (ite (or bubble-ex
           (not (= src1 dest-ex)))
       (read
        (ite bubble-wb
         regfile
         (write regfile dest-wb result))
        src1)
       (alu op-ex arg1 arg2))

Symbolic Simulation

• Create logical expressions for outputs/state
  – Taking initial state/inputs as variables

Case Splitting Search

• Satisfiability Problem
• Pick an unresolved variable
• Branch on true and false
• Push implications
• Bottom out at consistent specification
• Exit on contradiction
• Pragmatic: use memoization to reuse work

Review: What have we done?

• Reduced to simpler problem
  – Simple, clean specification
• Abstract Simulation
  – Explore all possible instruction sequences
• Abstracted the simulation
  – Focus on control
  – Divide and Conquer: control vs. arithmetic
• Used Satisfiability for reachability in search in abstract simulation
Achievable

• Burch&Dill: Verify 5-stage pipeline DLX
  – 1 minute in 1994

Self Consistency

• Compare same implementation in two different modes of operation
  – (which should not affect result)
  – Compare pipelined processor
    – To self w/ NOPs separating instructions
      • So only one instruction in pipeline at a time

Sample Result

<table>
<thead>
<tr>
<th>Circuit</th>
<th># Gates</th>
<th>Latches</th>
<th>Simulated Variables</th>
<th>Execution Time (s)</th>
<th>Equivalent Simulation Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>842</td>
<td>2500</td>
<td>40</td>
<td>2</td>
<td>$2 \times 10^4$</td>
</tr>
<tr>
<td>B</td>
<td>7203</td>
<td>11709</td>
<td>111</td>
<td>10</td>
<td>$2 \times 10^5$</td>
</tr>
</tbody>
</table>

Table 1. Self-consistency checking results.

[Jones, Seger, Dill/FMCAD 1996]

n.b. Jones&Seger at Intel

Sample Result

<table>
<thead>
<tr>
<th>IMPL. RAS Verification</th>
<th>IMPL. Rescheck</th>
<th>IMPL. RAS Verification</th>
<th>IMPL. Rescheck</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (sec)</td>
<td>Core Split</td>
<td>CPU (sec)</td>
<td>Core Split</td>
</tr>
<tr>
<td>Case</td>
<td></td>
<td>Case</td>
<td></td>
</tr>
<tr>
<td>Base Case</td>
<td>1.9</td>
<td>10</td>
<td>0.7</td>
</tr>
<tr>
<td>Core</td>
<td>63.0</td>
<td>28.21 (6)</td>
<td>71.0</td>
</tr>
<tr>
<td>Debug</td>
<td>49.1</td>
<td>12.03 (6)</td>
<td>41.02</td>
</tr>
<tr>
<td>Unhacked</td>
<td>35.0</td>
<td>88.2</td>
<td>42.1</td>
</tr>
<tr>
<td>Jitter</td>
<td>29.5</td>
<td>3.38</td>
<td>6.90</td>
</tr>
</tbody>
</table>

Verification running on P2-200MHz

[Skakkebæk, Jones, and Dill / CAV 1998]
Key Idea

• Implementation State reduces to Specification state after finite series of operations
• Abstract datapath to avoid dependence on bitwidth

Admin

• Last Class
• Final office hours T4pm
• Assignment 7 due May 13th
• I did make it clear you cannot pass class without completing the programming assignments (3,4,5).

Big Ideas

• Proving Invariants
• Divide and Conquer
• Exploit structure