

CS137: Electronic Design Automation

Day 10: May 5, 2004
Processor Verification



CALTECH CS137 Spring2004 -- DeHon

Today

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

CALTECH CS137 Spring2004 -- DeHon

Specification

- Abstract from Implementation
- Describes observable/correct behavior

CALTECH CS137 Spring2004 -- DeHon

Implementation

- Some particular embodiment
- Should have **same** observable behavior
 - Same with respect to **important** behavior
- Many more details
 - How performed
 - Auxiliary/intermediate state

CALTECH CS137 Spring2004 -- DeHon

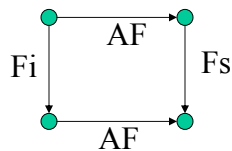
Important Behavior

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

CALTECH CS137 Spring2004 -- DeHon

Abstraction Function

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



CALTECH CS137 Spring2004 -- DeHon

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...

CALTECH CS137 Spring2004 -- DeHon

Memory AF

- Maps from
 - State of caches/WB/etc.
- To
 - Abstract state of memory
- Guarantee $AF(F_i(q,l)) == F_s(AF(q),l)$
 - Guarantee change to state always represents the correct thing

CALTECH CS137 Spring2004 -- DeHon

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)

CALTECH CS137 Spring2004 -- DeHon

Filter to Abstract Timing

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

CALTECH CS137 Spring2004 -- DeHon

Processors

- Pipeline is big difference between specification state and implementation state.
- Specification State:
 - Register contents (incl. PC)
 - Memory contents
- Implementation State:
 - + Instruction in pipeline

CALTECH CS137 Spring2004 -- DeHon

Observation

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

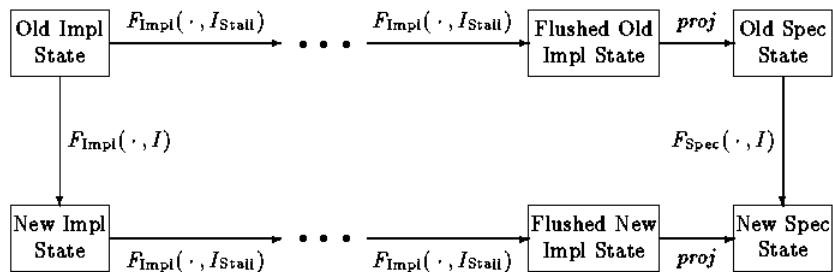
CALTECH CS137 Spring2004 -- DeHon

Pipelined Processor Correctness

- w = input sequence
- w_f = flush sequence
 - Enough NOPs to flush pipeline state
- For all states q and prefix w
 - $F_i(q, w w_f) \rightarrow F_s(q, w w_f)$
 - $F_i(q, w w_f) \rightarrow F_s(q, w)$
- FSM observation
 - Finite state in pipeline
 - only need to consider finite w

CALTECH CS137 Spring2004 -- DeHon

Pipeline Correspondence



[Burch+Dill, CAV'94]

CALTECH CS137 Spring2004 -- DeHon

Equivalence

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

CALTECH CS137 Spring2004 -- DeHon

Ideas

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

CALTECH CS137 Spring2004 -- DeHon

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

CALTECH CS137 Spring2004 -- DeHon

B&D Logic

- Formula = **ite**(formula, formula, formula)
 - | (term=term)
 - | psym(term,...term)
 - | pvar | **true** | **false**
- Term = **ite**(formula,term,term)
 - | fsym(term,...term)
 - | tvar

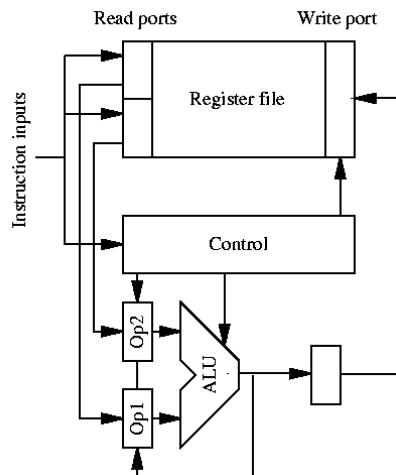
CALTECH CS137 Spring2004 -- DeHon

Sample

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

CALTECH CS137 Spring2004 -- DeHon

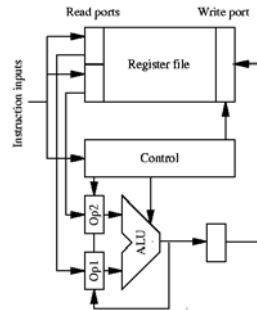
Sample Pipeline



CALTECH CS137 Spring2004 -- DeHon

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))



CALTECH CS137 Spring2004 -- DeHon

Symbolic Simulation

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

CALTECH CS137 Spring2004 -- DeHon

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

CALTECH CS137 Spring2004 -- DeHon

What have we done

- Add slide to pull this together
- ...review what done
- (before go on to next slide)

CALTECH CS137 Spring2004 -- DeHon

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

CALTECH CS137 Spring2004 -- DeHon

Self-Consistency

- w = instruction sequence
- $S(w)$ = w with no-ops
- Show: For all q, w
 - $F(q, w) = F(q, S(w))$

CALTECH CS137 Spring2004 -- DeHon

Key Idea

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

CALTECH CS137 Spring2004 -- DeHon

Big Ideas

- Proving Invariants
- Divide and Conquer
- Exploit structure

CALTECH CS137 Spring2004 -- DeHon