Bluespec IA-64 Modeling

Roland Wunderlich and James C. Hoe
Computer Architecture Lab (CALCM)
Carnegie Mellon University

Motivation

- Combining high-level description/synthesis and FPGAs for rapid processor prototyping
  an alternative to software simulation studies
- Allow direct evaluation of new mechanisms
  - Functionality: a fast emulator that can run real software but remains infinitely malleable
    completeness, correctness, …..
  - Implementation: a synthesized design gives hints about feasibility, design complexity and implementation cost
    area,cycle time, power, …..

Current Project

- Develop a high-level model of IA-64 microarchitectures
  - concise and malleable
  - detailed = *executable and synthesizable*
- Synthesize to FPGA
  - to target XC2V6000 FPGA board in a P6 processor slot
  - to execute binaries natively on the FPGA processor against a new PC environment
- *Current Modeling Challenges*
  - *processor complexity*
  - *trade-off between μarch realism and design effort*

Why Bluespec?

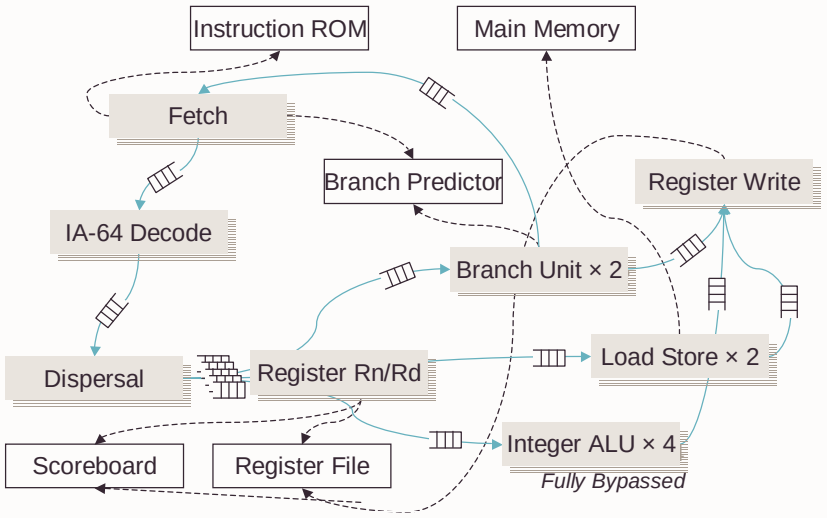
- Detailed
  - nothing left to interpretation
  - litmus test: can it be executed or synthesized automatically?
- Concise
  - compact and expressive
  - natural correspondence to HW structures and abstractions
- Maintainable
  - easy to understand
  - modular: composible and decomposible

IA64 Modeling

Current IA64 Model

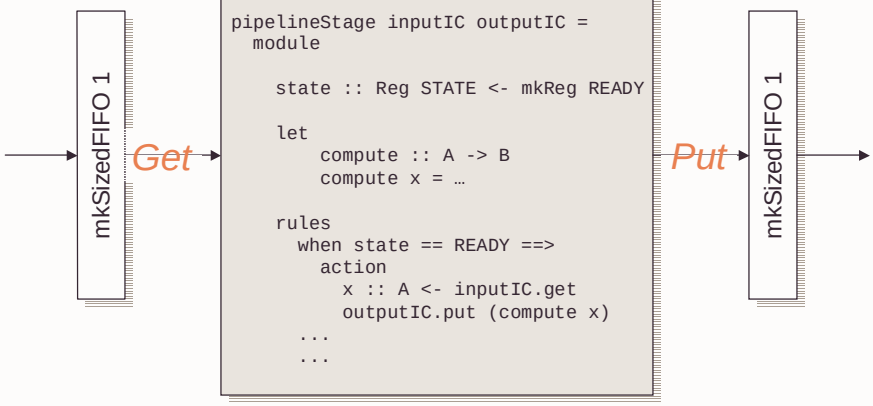
- Basic framework for a 6-wide Itanium datapath
  - decoding rules for all user-level, non-FP instructions
  - 2 br. unit, 2 mem/int unit, 4 integer unit
  - bypassing for integer unit, scoreboarding for remaining
  - currently supports execution of
    - alloc *(register stack but no rotation, no spill engine)*
    - mov *(branch registers to general registers and vice versa)*
    - cmp.*.unc *(all relations, all immediate forms, compare to zero)*
    - add, adds, addl *(all immediate forms)*
    - br.cond.*.*br.call.*.*br.ret.*.*
    - nop.* (all unit types)
- *Parameterized modeling, i.e. #'s can be changed easily*

Prototype Microarchitecture



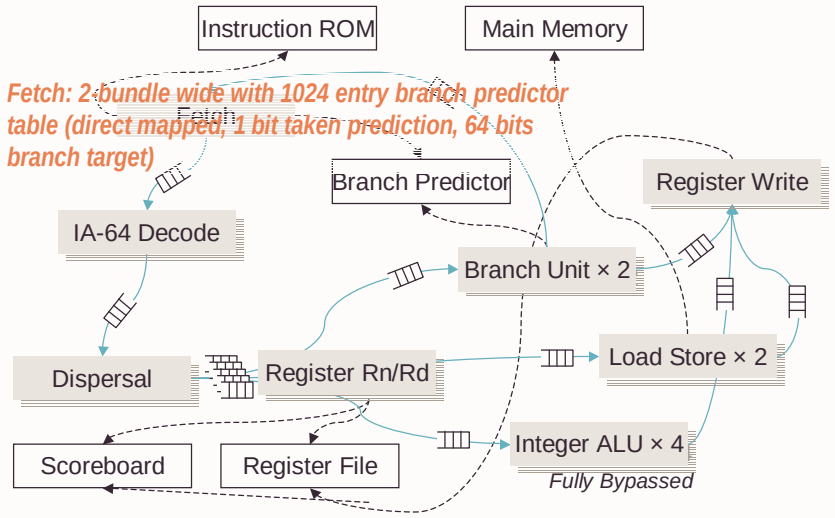
About 70 rules (corresponds to many more instances)

Pipeline Stage Detail

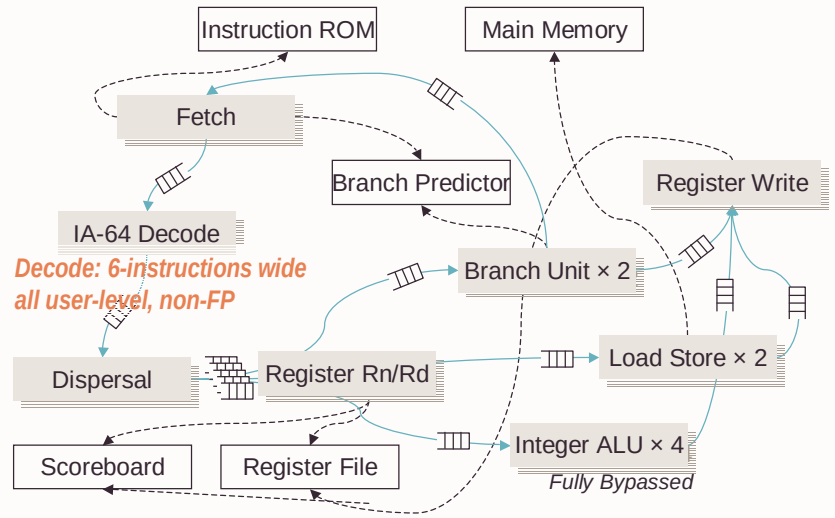



```
pipelineStage :: Get A -> Put B
  -> Module Empty

pipelineStage inputIC outputIC =
  module

    state :: Reg STATE <- mkReg READY

    let
        compute :: A -> B
        compute x = …

    rules
      when state == READY ==>
        action
          x :: A <- inputIC.get
          outputIC.put (compute x)
      ...
      ...
```

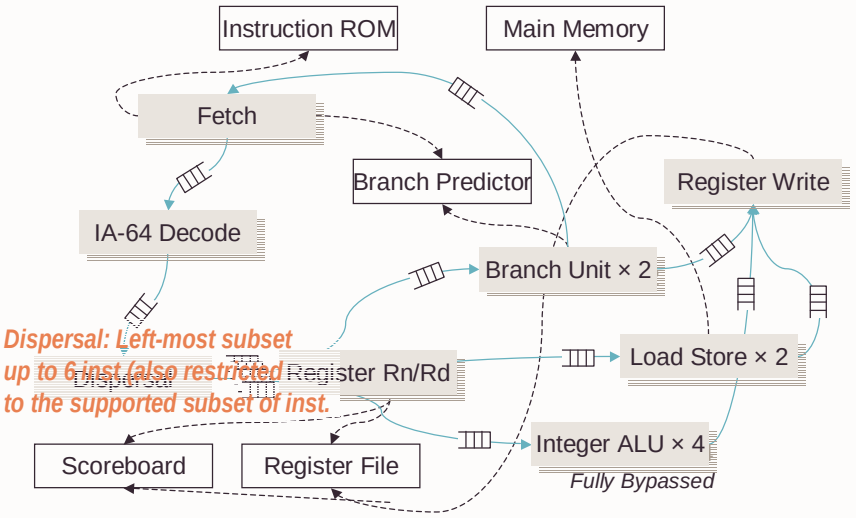
Prototype Microarchitecture



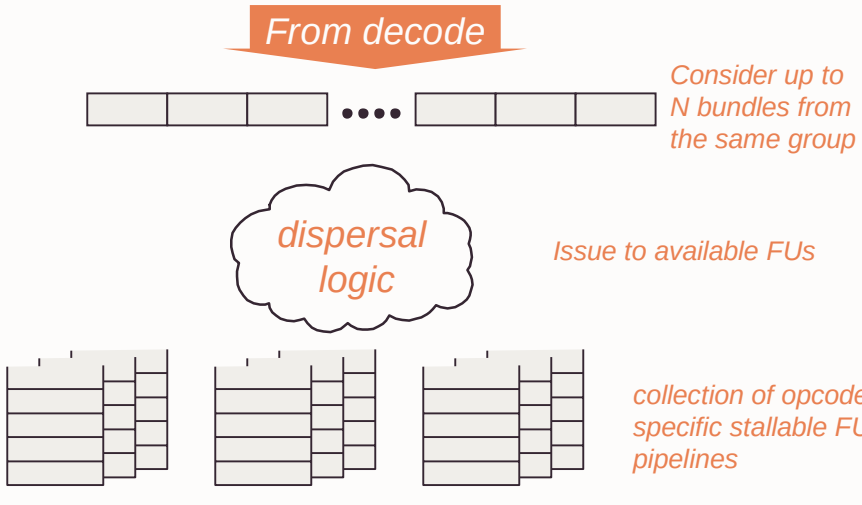
Prototype Microarchitecture



Prototype Microarchitecture



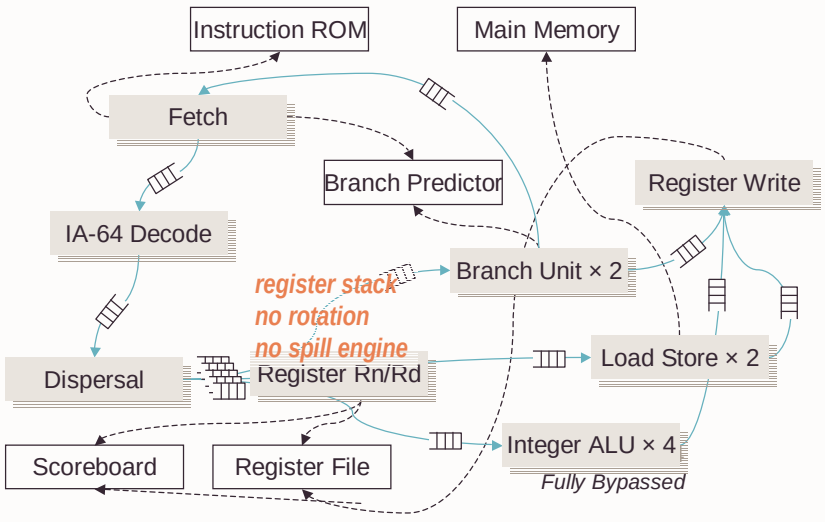
Dispersal Stage Logic



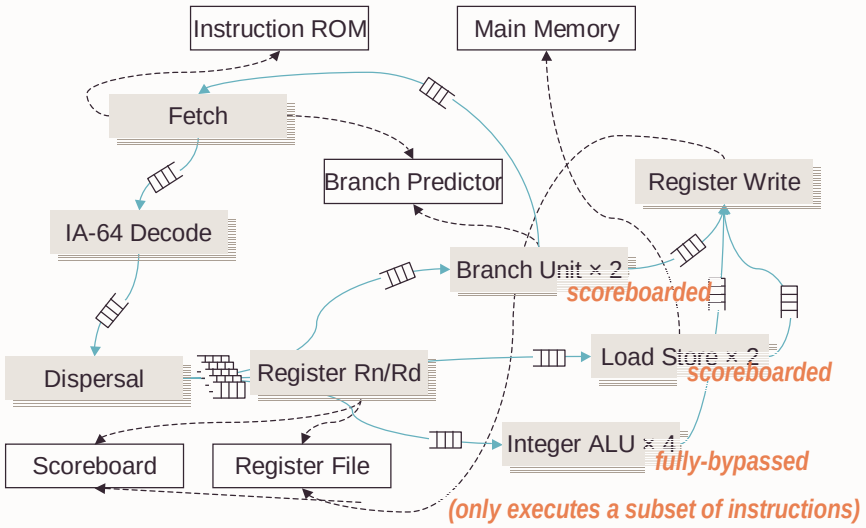
Dispersal Stage Bluespec

```
dispersal decodeIC (brPorts, memPorts, intPorts) =
  module
    instructionQueue :: Reg WideDispersalInstructions <- mkReg
    let
      wideDisperse :: WideDispersalInstructions -> Action
      wideDisperse instructions =
        let
          unitTypes = getUnitTypes (selectExecutionGroup instructions)
          availablePorts = map isAvailable (brPorts :> memPorts :> intPorts)
          pattern = dispersalPattern availablePorts unitTypes
        in
          joinActions (map (\x -> disperseAction pattern) instructions)
          instructionQueue := map (\x -> splitIssue pattern) instructions
    rules
      "New Dispersal": when instructionQueue == Nothing ==>
        wideDisperse decodeIC.get
      "Split Issue Dispersal": when (Just partial) <- instructionQueue ==>
        wideDisperse partial
```

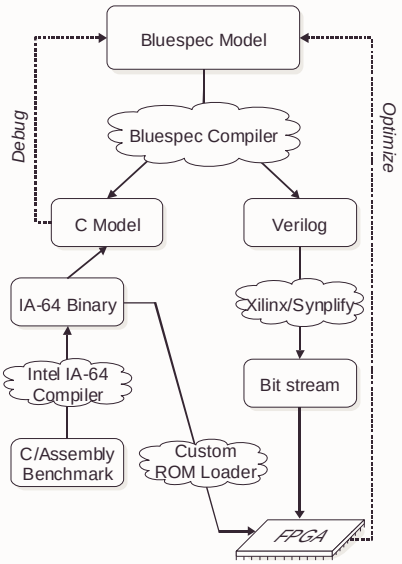

Prototype Microarchitecture



Prototype Microarchitecture



Development Flow



Synthesis Results

- Synplify and Xilinx ISE for a XC2V6000 –6
  - *LUTs: 43,408 of 74,900 (58%)*
  - *Critical path: 23.450 ms (42.6 MHz)*
- Cost breakdown *(synthesized in pieces)*

	LUT (max 38,400)	Critical Path (ns)	Freq. (MHz)
Fetch	6%	14.8	67.3
Decode	8%	23.0	43.5
Dispersal+ Rn+Rd	24%	17.2	58.1
Execute	11%	19.5	51.4

Some Questions

- What has been the real advantage of using Bluespec?

  operation-centric abstraction vs language expressiveness?
- Are high-level transformations a good thing?

  do I trust the compiler when I don't know what its doing behind my back?
- What about synchronous/deterministic designs?

Computer Architecture Lab at Carnegie Mellon (CALCM)
http://www.ece.cmu.edu/~jhoe
jhoe@ece.cmu.edu