The Berkeley Out-of-Order Machine (**BOOM!**): An Open-source Industry-Competitive, Synthesizable, Parameterized RISC-V Processor

Christopher Celio, Krste Asanovic, David Patterson

2016 Jan

celio@eecs.berkeley.edu

Berkeley Architecture Research

What is BOOM?

- superscalar, out-of-order processor written in Berkeley's Chisel RTL
- It is synthesizable
- It is parameterizable
- We hope to use it as a platform for architecture research

One more thing...

What is BOOM?

- superscalar, out-of-order processor written in Berkeley's Chisel RTL
- It is synthesizable
- It is parameterizable
- **It is open-source!**

What is BOOM?

- superscalar, out-of-order processor written in Berkeley's Chisel RTL
- It is synthesizable
- It is parameterizable
- **It is open-source!**

BOOM is a work-in-progress.
Results shown in the talk are preliminary and subject to change!

Open-source Berkeley RISC-V Processors

- **Sodor** Collection
  - RV32I - tiny, educational, not synthesizable

Open-source Berkeley RISC-V Processors

- **Sodor** Collection
  - RV32I - tiny, educational, not synthesizable

Open-source Berkeley RISC-V Processors

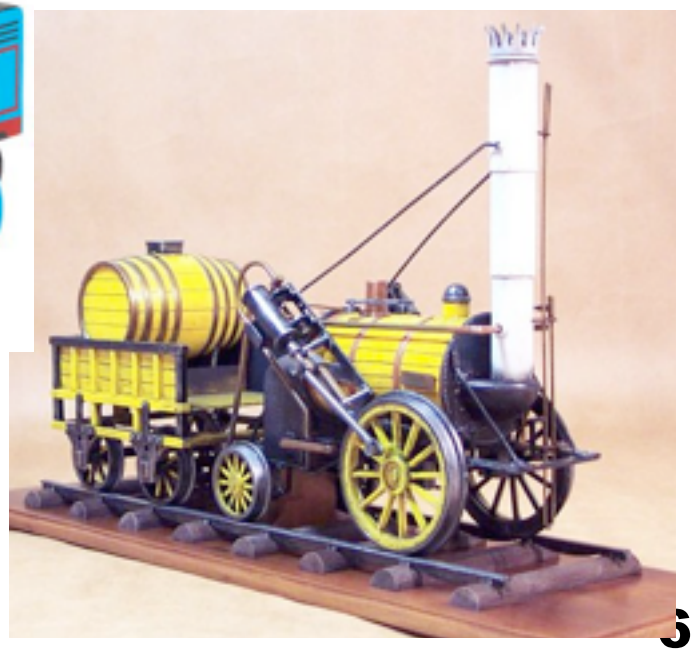
- **Sodor** Collection
  - RV32I - tiny, educational, not synthesizable
- **Rocket-chip** SoC generator
  - Z-scale
    - RV32IM - micro-controller
  - Rocket
    - RV64G - in-order, single-issue application core
  - BOOM
    - RV64G - out-of-order, superscalar application core

Open-source Berkeley RISC-V Processors

- **Sodor** Collection
  - RV32I - tiny, educational, not synthesizable
- **Rocket-chip** SoC generator
  - Z-scale
    - RV32IM - micro-controller
  - Rocket
    - RV64G - in-order, single-issue application core
  - BOOM
    - RV64G - out-of-order, superscalar application core

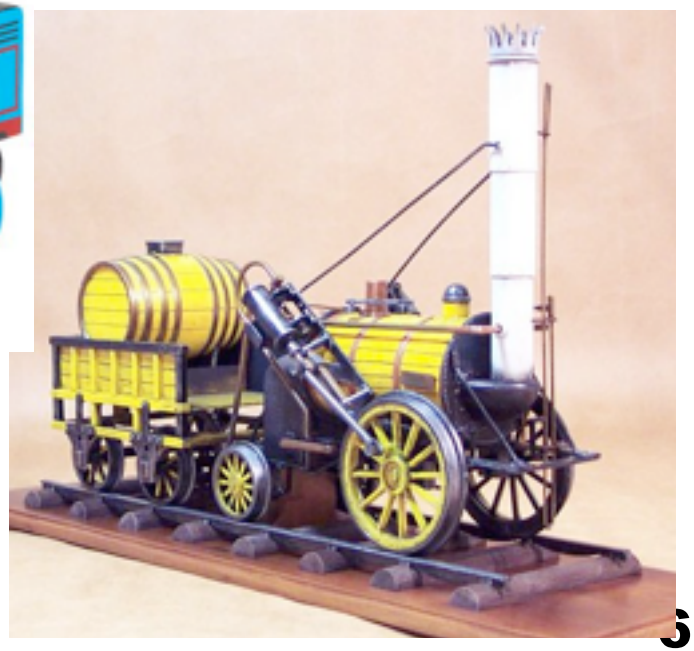
Open-source Berkeley RISC-V Processors

- **Sodor** Collection
  - RV32I - tiny, educational, not synthesizable
- **Rocket-chip** SoC generator
  - Z-scale
    - RV32IM - micro-controller
  - Rocket
    - RV64G - in-order, single-issue application core
  - BOOM
    - RV64G - out-of-order, superscalar application core



Open-source Berkeley RISC-V Processors

- **Sodor** Collection
  - RV32I - tiny, educational, not synthesizable
- **Rocket-chip** SoC generator
  - Z-scale
    - RV32IM - micro-controller
  - Rocket
    - RV64G - in-order, single-issue application core
  - BOOM
    - RV64G - out-of-order, superscalar application core



Why Out-of-Order?

```
add r1, r0,  r0
ld  r2,   M[r10]
sub r3, r2,  r13
mul r4, r14, r15
```

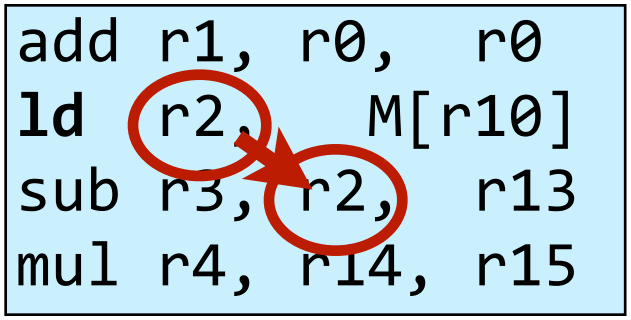
Why Out-of-Order?

```
add r1, r0,  r0
ld  r2,   M[r10]
sub r3, r2,  r13
mul r4, r14, r15
```

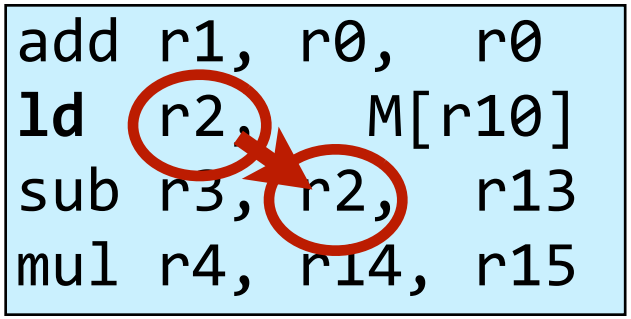
Why Out-of-Order?

```
add r1, r0,  r0
ld  r2,   M[r10]
sub r3, r2,  r13
mul r4, r14, r15
```

Why Out-of-Order?

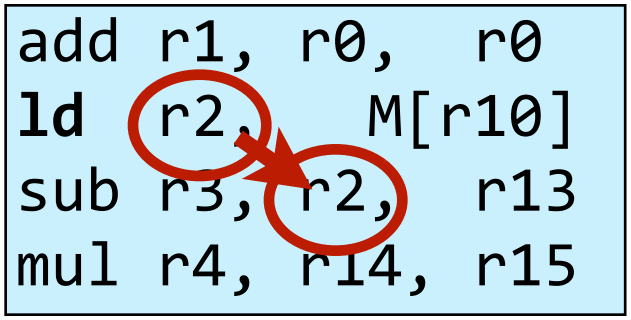


Why Out-of-Order?



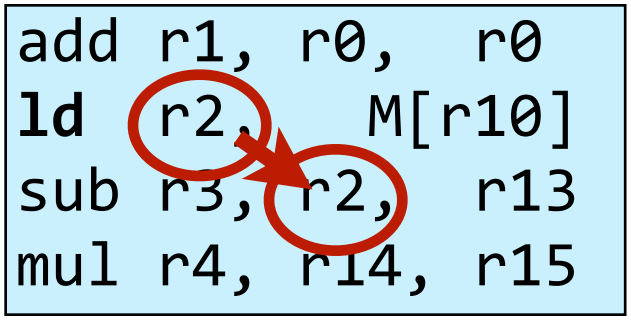
- Great for …

Why Out-of-Order?



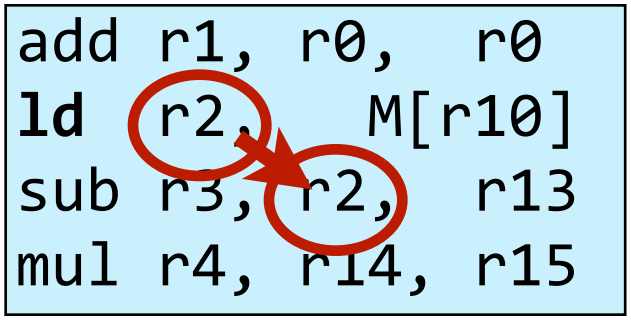
- Great for **…**
  - tolerating **variable** latencies

Why Out-of-Order?



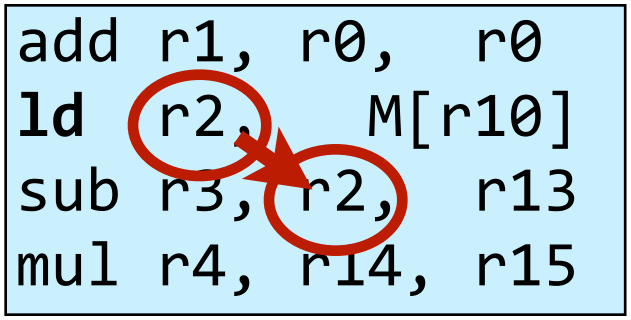
- Great for **…**
  - tolerating **variable** latencies
  - finding **ILP** in code (instruction-level parallelism)

Why Out-of-Order?



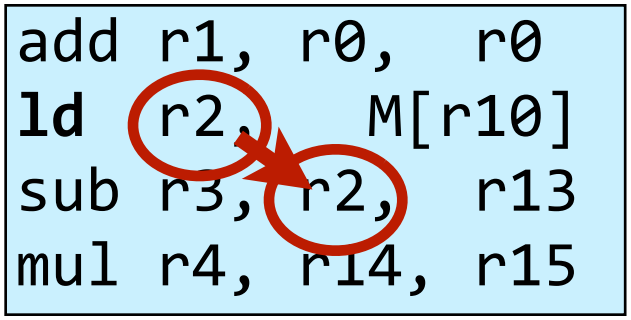
- Great for **…**
  - tolerating **variable** latencies
  - finding **ILP** in code (instruction-level parallelism)
  - complex method for fine-grain data **prefetching**

Why Out-of-Order?



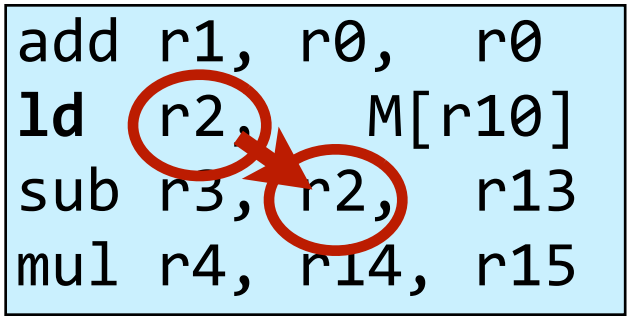
- Great for **…**
  - tolerating **variable** latencies
  - finding **ILP** in code (instruction-level parallelism)
  - complex method for fine-grain data **prefetching**
  - plays nicely with **poor** compilers and **lazily** written code

Why Out-of-Order?



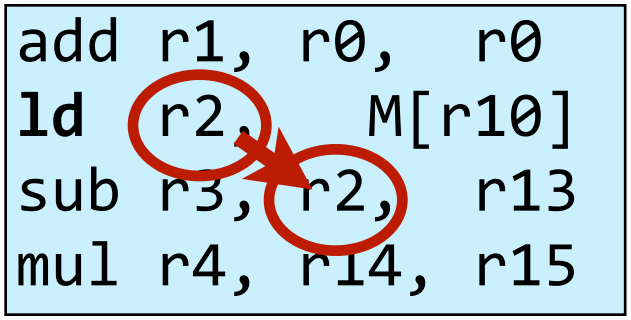
- Great for **…**
  - tolerating **variable** latencies
  - finding **ILP** in code (instruction-level parallelism)
  - complex method for fine-grain data **prefetching**
  - plays nicely with **poor** compilers and **lazily** written code
- Downsides

Why Out-of-Order?



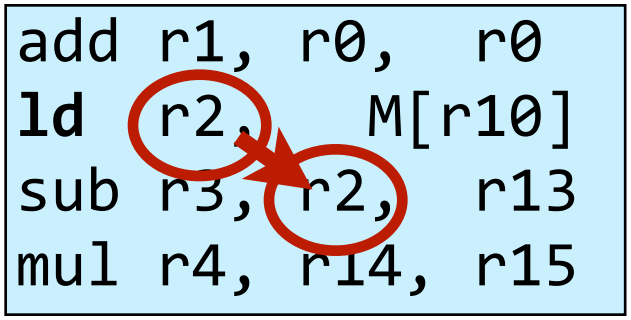
- Great for **…**
  - tolerating **variable** latencies
  - finding **ILP** in code (instruction-level parallelism)
  - complex method for fine-grain data **prefetching**
  - plays nicely with **poor** compilers and **lazily** written code
- Downsides
  - complicated

Why Out-of-Order?



- Great for **…**
  - tolerating **variable** latencies
  - finding **ILP** in code (instruction-level parallelism)
  - complex method for fine-grain data **prefetching**
  - plays nicely with **poor** compilers and **lazily** written code
- Downsides
  - complicated
  - expensive (area & power)

Why Out-of-Order?



- Great for **…**
  - tolerating **variable** latencies
  - finding **ILP** in code (instruction-level parallelism)
  - complex method for fine-grain data **prefetching**
  - plays nicely with **poor** compilers and **lazily** written code
- Downsides
  - complicated
  - expensive (area & power)

Performance! (and easy to program!)

OoO is widely used in industry

- Intel Xeon/i-series (10-100W)
- ARM Cortex mobile chips (1W)
- Sun/Oracle Niagra UltraSPARC
- Intel Atom
- Play Station

Goals

Goals

- build a prototypical OoO core
  - provide an open-source implementation for education, research, and industry
  - serve as a baseline for micro-architecture research

Goals

- build a prototypical OoO core
  - provide an open-source implementation for education, research, and industry
  - serve as a baseline for micro-architecture research
- enable studies that need an OoO core
  - research memory systems
  - research accelerators
  - research VLSI methodologies

Goals

- build a prototypical OoO core
  - provide an open-source implementation for education, research, and industry
  - serve as a baseline for micro-architecture research
- enable studies that need an OoO core
  - research memory systems
  - research accelerators
  - research VLSI methodologies
- use BOOM to get detailed performance, area, and power running real programs
  - most research uses sw simulators for performance (100 KIPS), RTL for area

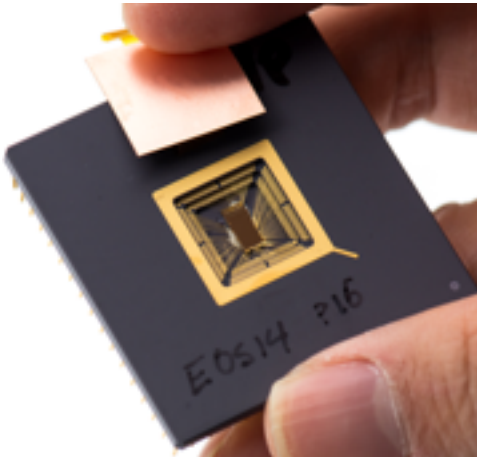
Berkeley Architecture Research Infrastructure

- RISC-V ISA
- Rocket-chip SoC generator
- Chisel HCL (hardware construction language)
- **bar.eecs.berkeley.edu**

Berkeley
Architecture
Research

BOOM Implements IMAFD

- "M" (mul/div/rem)
  - imul can be either pipelined or unpipelined
- "A"
  - AMOs+LR/SC
- "FD"
  - single, double-precision floating point
  - IEEE 754-2008 compliant FPU
  - SP, DP FMA with hw support for subnormals
  - parameterized latency
- **RV64G + privileged spec (page-based virtual memory)**



The RISC-V ISA is easy to implement!

The RISC-V ISA is easy to implement!

- relaxed memory model

The RISC-V ISA is easy to implement!

- relaxed memory model
- accrued FP exception flags

The RISC-V ISA is easy to implement!

- relaxed memory model
- accrued FP exception flags
- no integer side-effects (e.g., condition codes)

The RISC-V ISA is easy to implement!

- relaxed memory model
- accrued FP exception flags
- no integer side-effects (e.g., condition codes)
- no cmov or predication

The RISC-V ISA is easy to implement!

- relaxed memory model
- accrued FP exception flags
- no integer side-effects (e.g., condition codes)
- no cmov or predication
- no implicit register specifiers

The RISC-V ISA is easy to implement!

- relaxed memory model
- accrued FP exception flags
- no integer side-effects (e.g., condition codes)
- no cmov or predication
- no implicit register specifiers
  - JAL requires explicit rd

The RISC-V ISA is easy to implement!

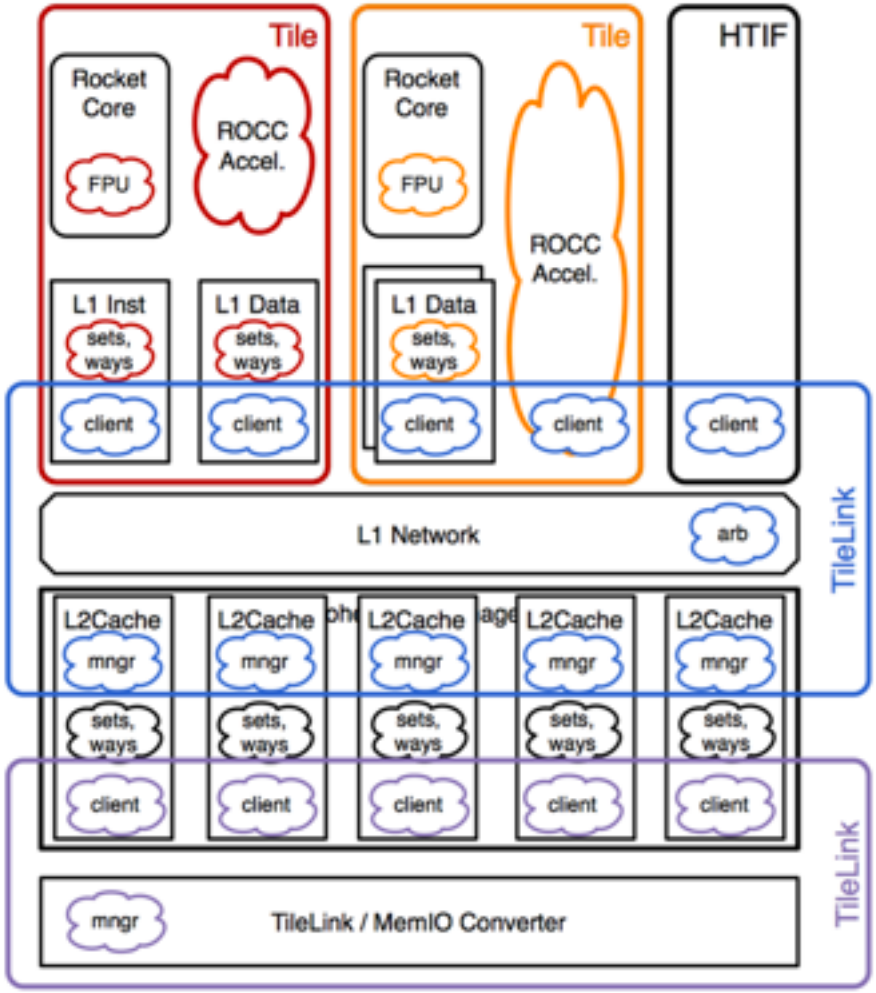
- relaxed memory model
- accrued FP exception flags
- no integer side-effects (e.g., condition codes)
- no cmov or predication
- no implicit register specifiers
  - JAL requires explicit rd
- rs1, rs2, rs3, rd always in same space

The RISC-V ISA is easy to implement!

- relaxed memory model
- accrued FP exception flags
- no integer side-effects (e.g., condition codes)
- no cmov or predication
- no implicit register specifiers
  - JAL requires explicit rd
- rs1, rs2, rs3, rd always in same space
  - allows decode, rename to proceed in parallel

Rocket-Chip SoC Generator

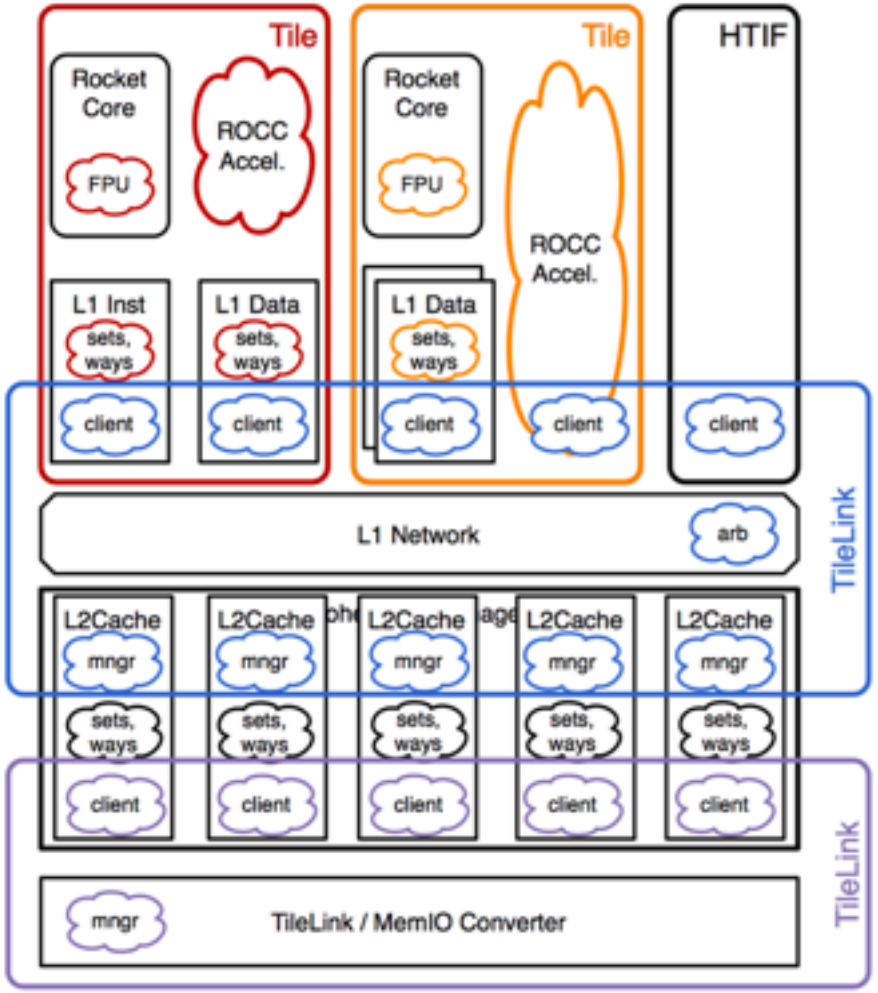
- open-source
- taped out **11** times by Berkeley
- runs at **1.6 GHz** in IBM 45nm
- makes for a great library of processor components!
- swap out Rocket tile and replace with a BOOM tile
- as Rocket-chip gets better, so does BOOM



*slightly out-of-date uncore
(now uses AXI instead of MemIO)

Rocket-Chip Updates

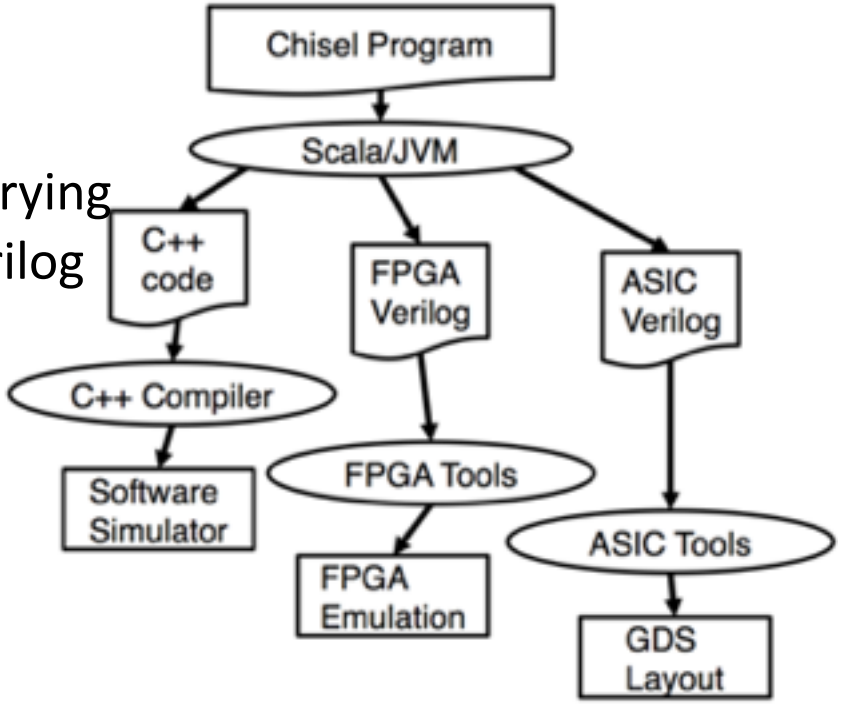
- uncached loads/stores
- memory-mapped IO
- replaced memIO with AXI
- **Work in Progress**
  - **remove HTIF (untether)**



*slightly out-of-date uncore
(now uses AXI instead of MemIO)

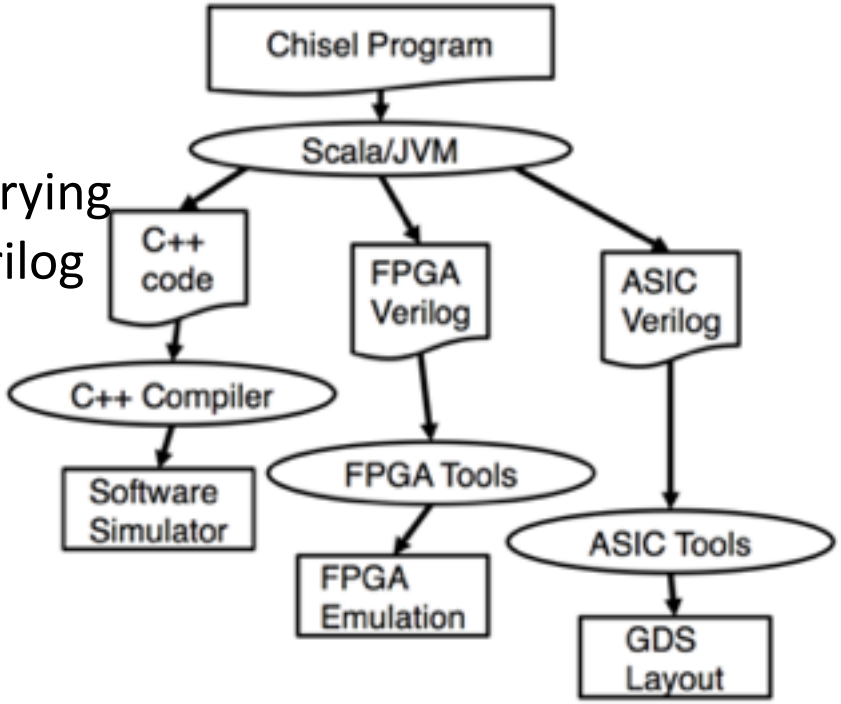
Chisel

- Hardware Construction Language embedded in **Scala**
- ***not*** a high-level synthesis language
- hardware module is a data structure in Scala
- Full power of Scala for writing generators
  - object-oriented programming
    - factory objects, traits, overloading
  - functional programming
    - high-order funs, anonymous funcs, currying
- generated C++ simulator is **1:1** copy of Verilog designs

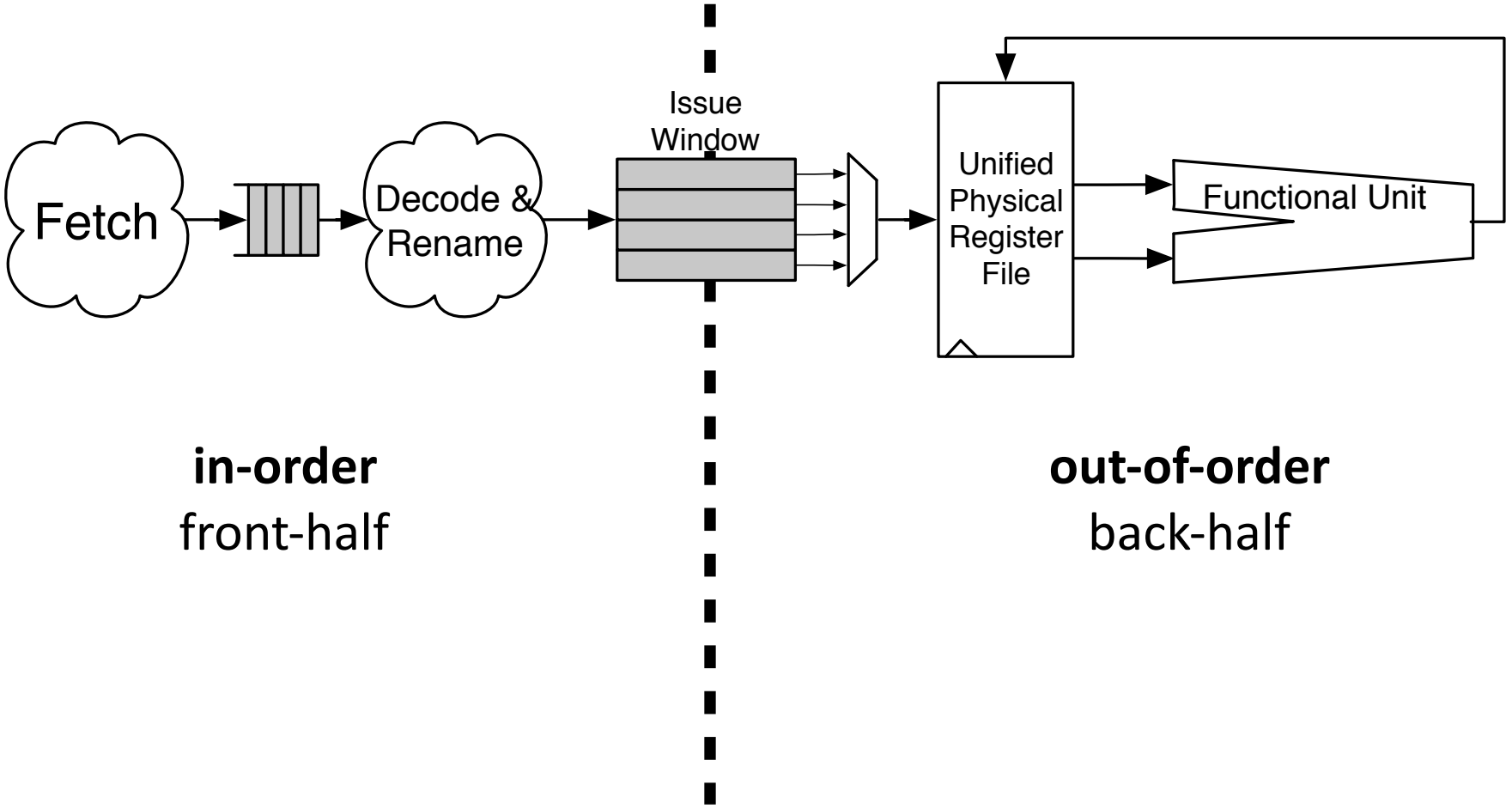


Chisel

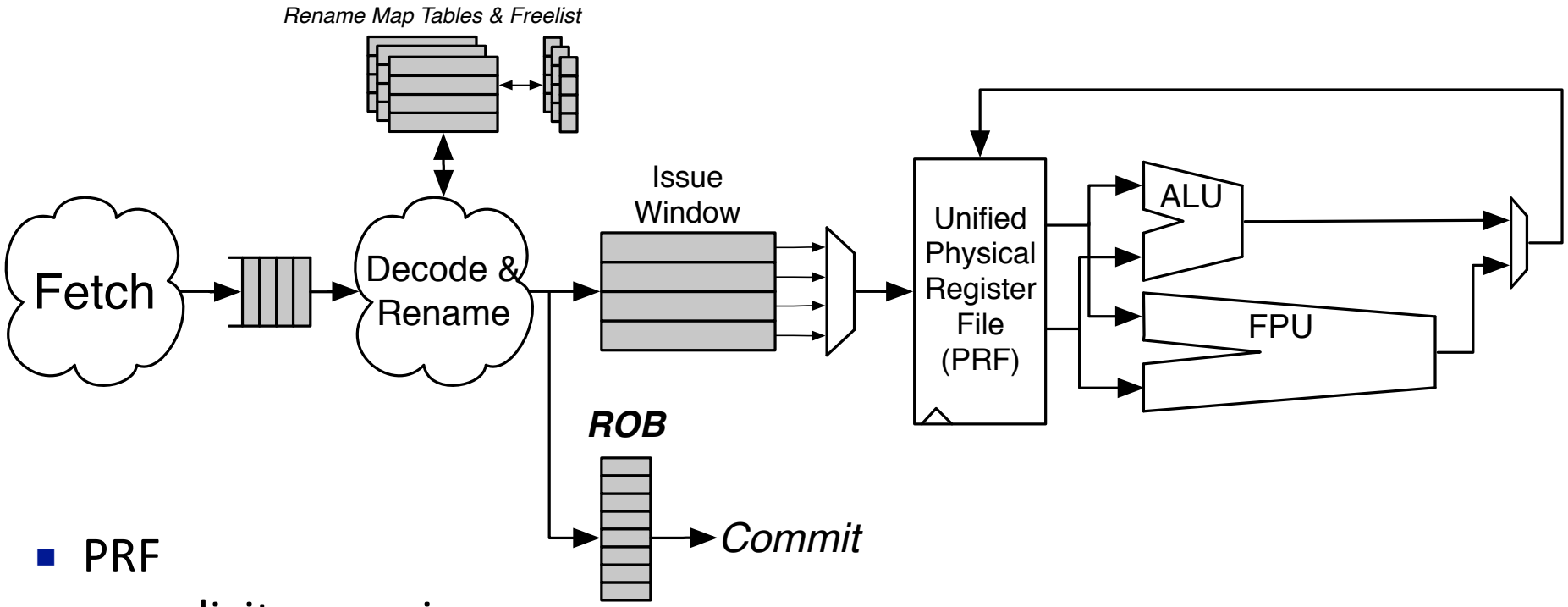
- Hardware Construction Language embedded in **Scala**
- ***not*** a high-level synthesis language
- hardware module is a data structure in Scala
- Full power of Scala for writing generators
  - object-oriented programming
    - factory objects, traits, overloading
  - functional programming
    - high-order funs, anonymous funcs, currying
- generated C++ simulator is **1:1** copy of Verilog designs
- **version 3.0 coming soon!**
  - **uses an IR called FIRRTL**



BOOM Pipeline



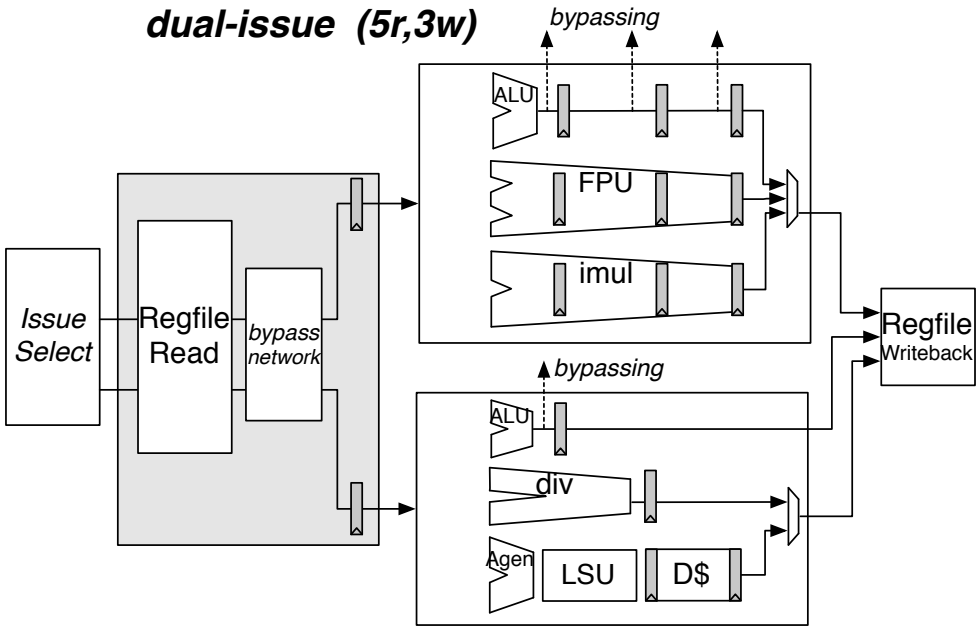
BOOM Pipeline



- PRF
  - explicit renaming
  - holds speculative and committed data
  - holds both x-regs, f-regs
- split ROB/issue window design
- Unified Issue Window
  - holds all instructions

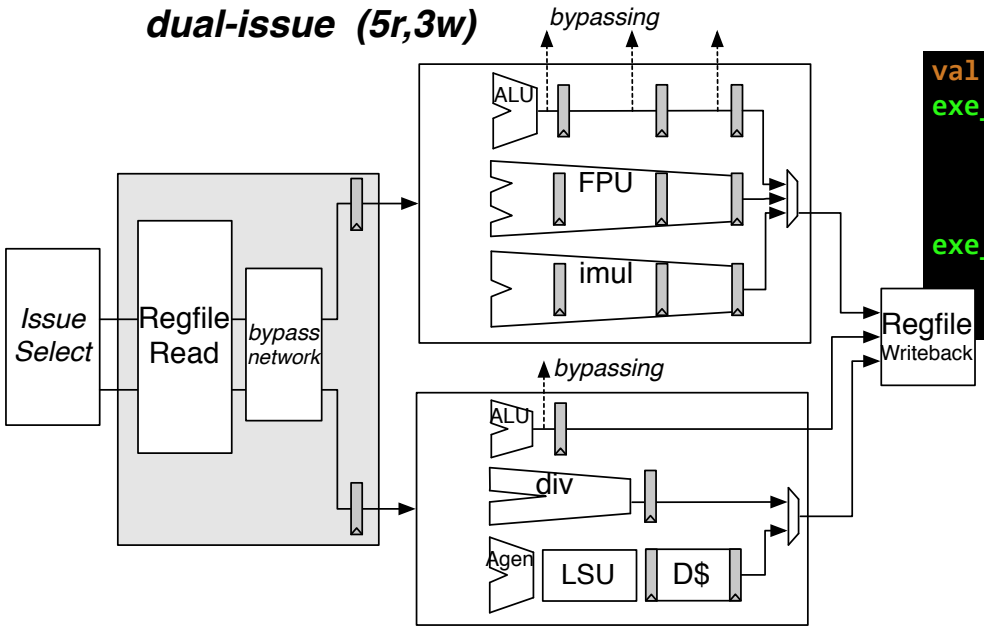
Parameterized Superscalar

dual-issue (5r,3w)



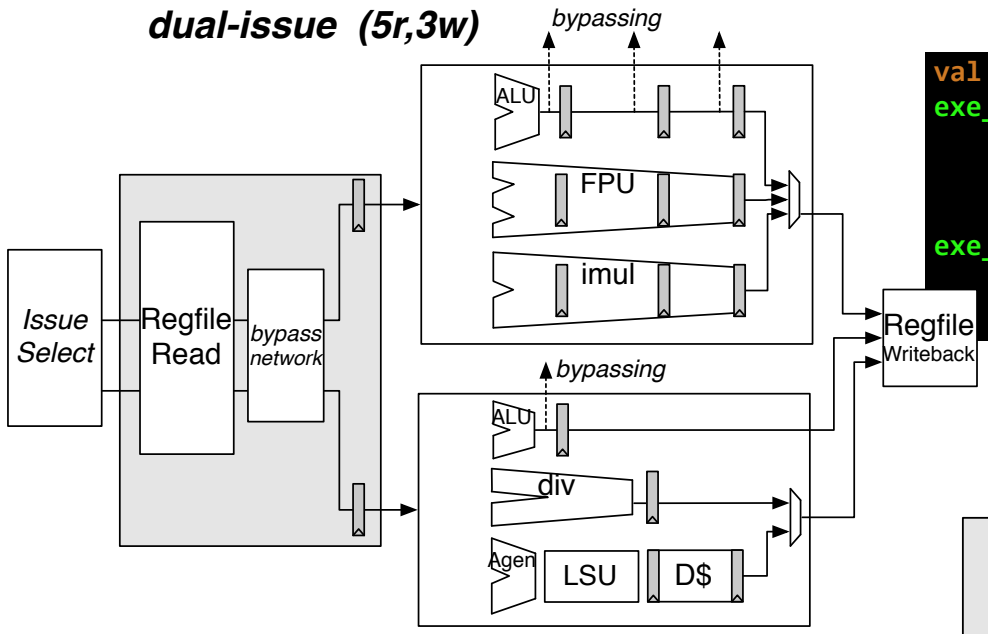
Parameterized Superscalar

dual-issue (5r,3w)



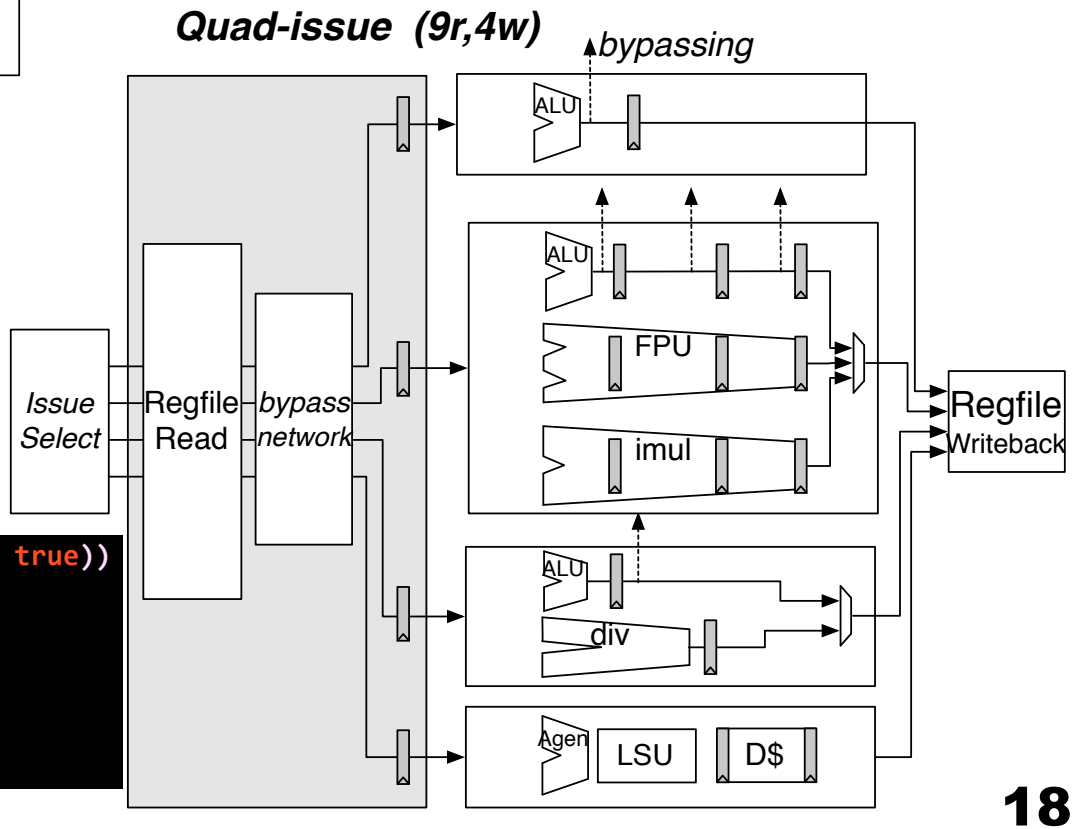

```
val exe_units = ArrayBuffer[ExecutionUnit]()
exe_units += Module(new ALUExeUnit(is_branch_unit    = true
                                  , has_fpu          = true
                                  , has_mul          = true
                                  ))
exe_units += Module(new ALUMemExeUnit(fp_mem_support = true
                                  , has_div          = true
                                  ))
```

Parameterized Superscalar



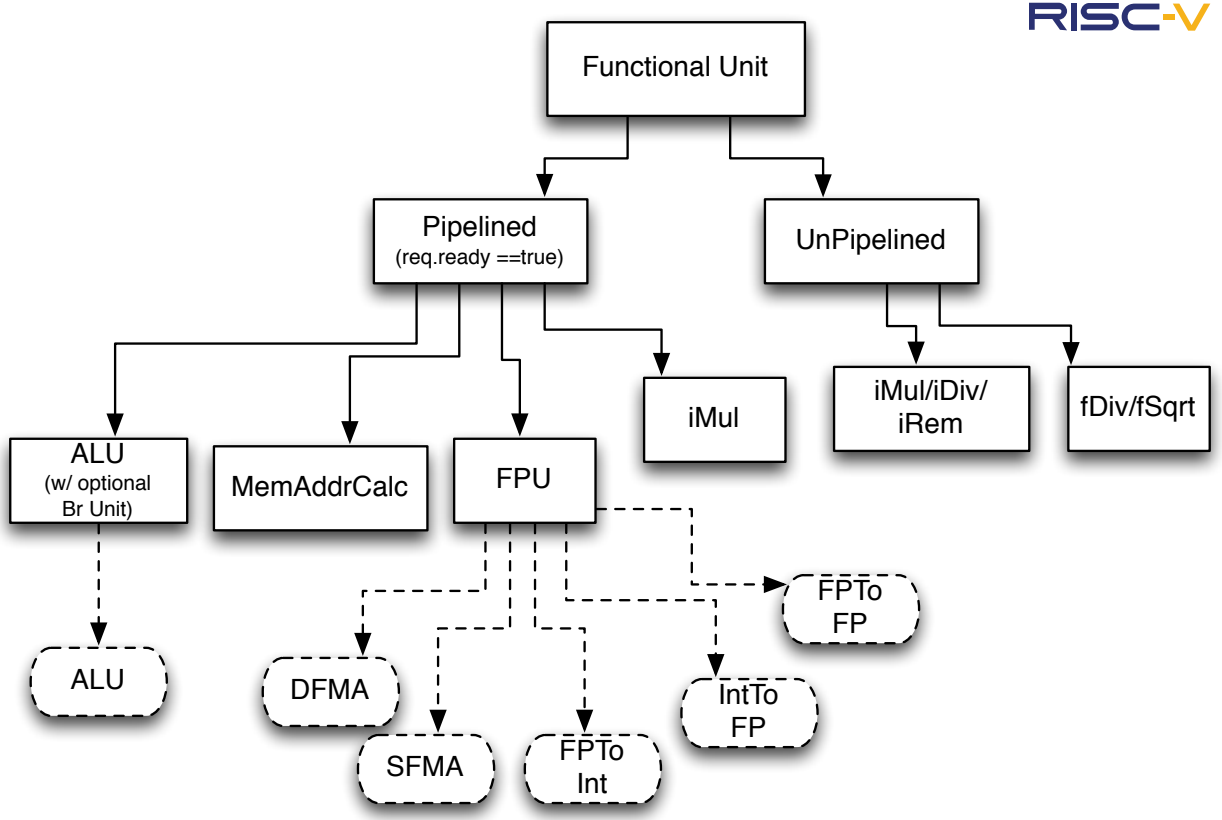

```
val exe_units = ArrayBuffer[ExecutionUnit]()
exe_units += Module(new ALUExeUnit(is_branch_unit    = true
                                  , has_fpu          = true
                                  , has_mul          = true
                                  ))
exe_units += Module(new ALUMemExeUnit(fp_mem_support = true
                                  , has_div          = true
                                  ))
```

OR



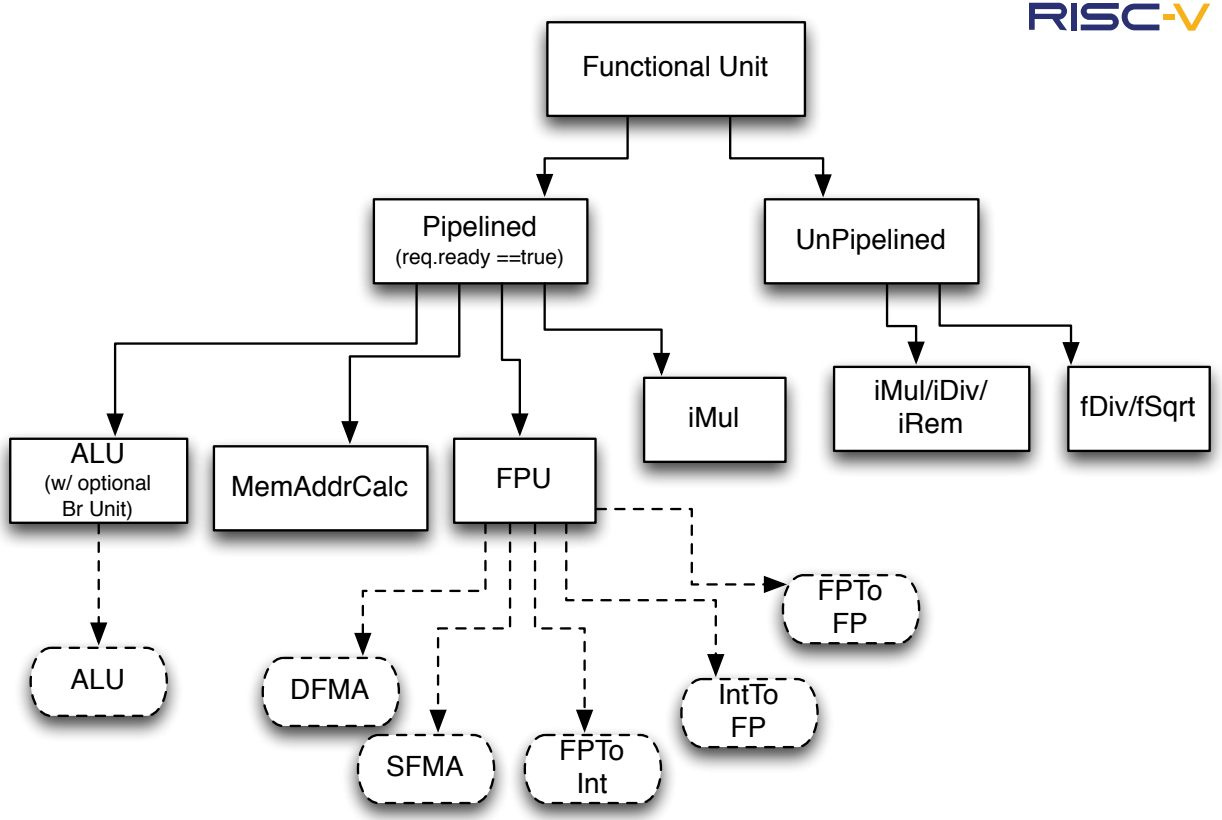

```
exe_units += Module(new ALUExeUnit(is_branch_unit = true))
exe_units += Module(new ALUExeUnit(has_fpu = true
                                 , has_mul = true
                                 ))
exe_units += Module(new ALUExeUnit(has_div = true))
exe_units += Module(new MemExeUnit())
```

A Functional Unit Hierarchy



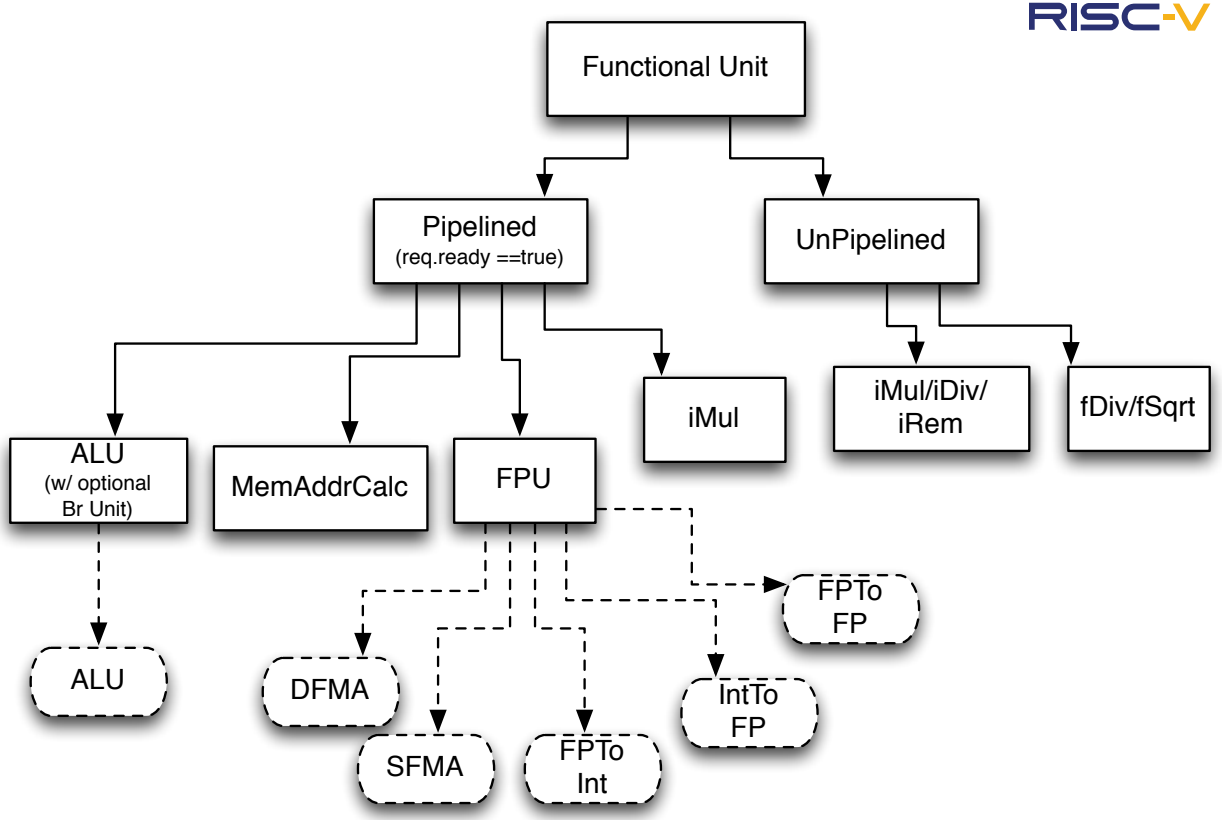
A Functional Unit Hierarchy

- **Abstract FunctionalUnit**



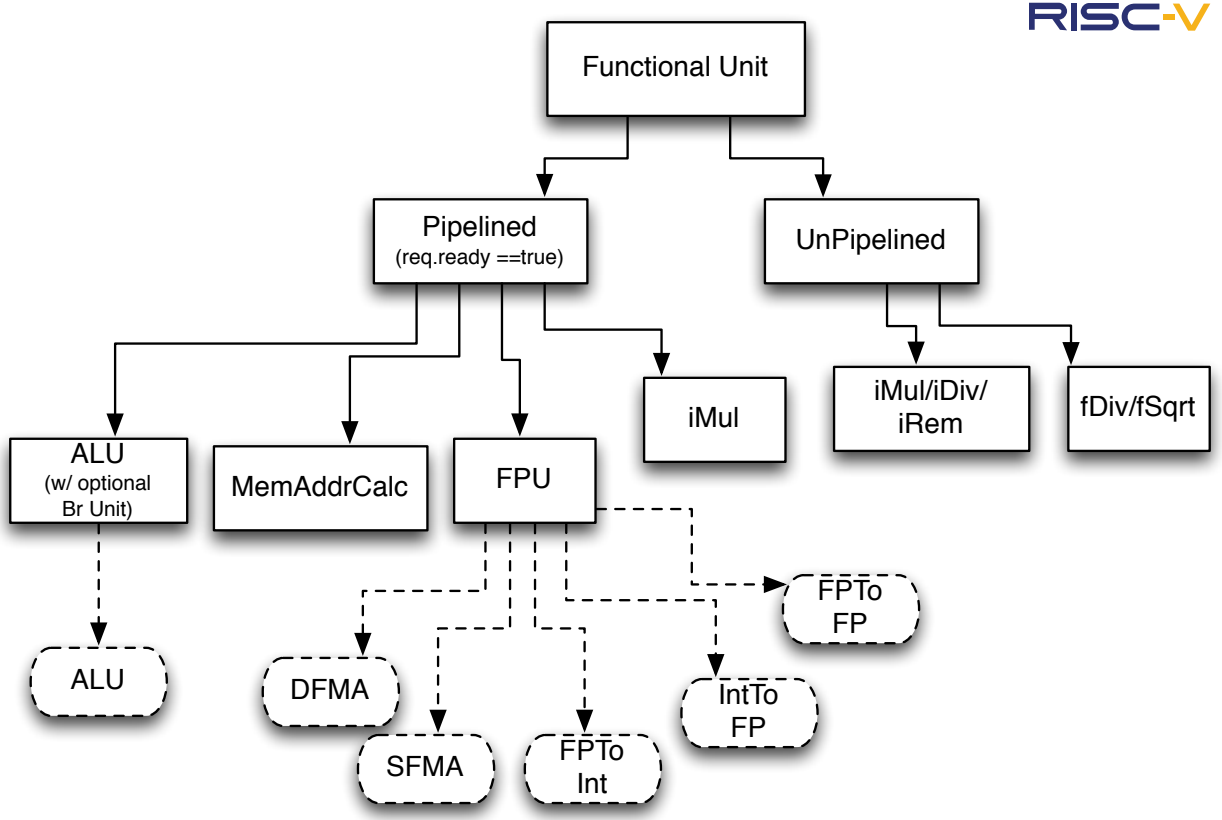
A Functional Unit Hierarchy

- **Abstract FunctionalUnit**
  - describes common IO



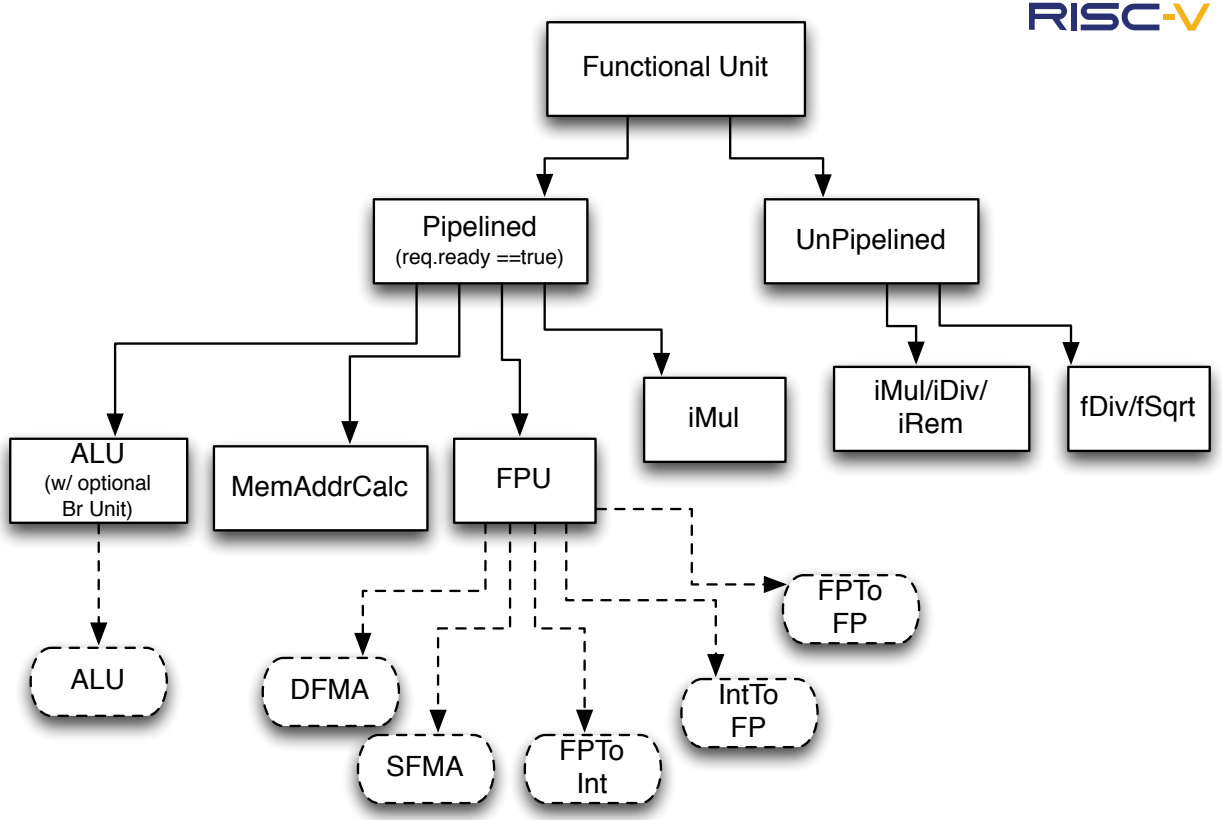
A Functional Unit Hierarchy

- **Abstract FunctionalUnit**
  - describes common IO
- **Pipelined/Unpipelined**



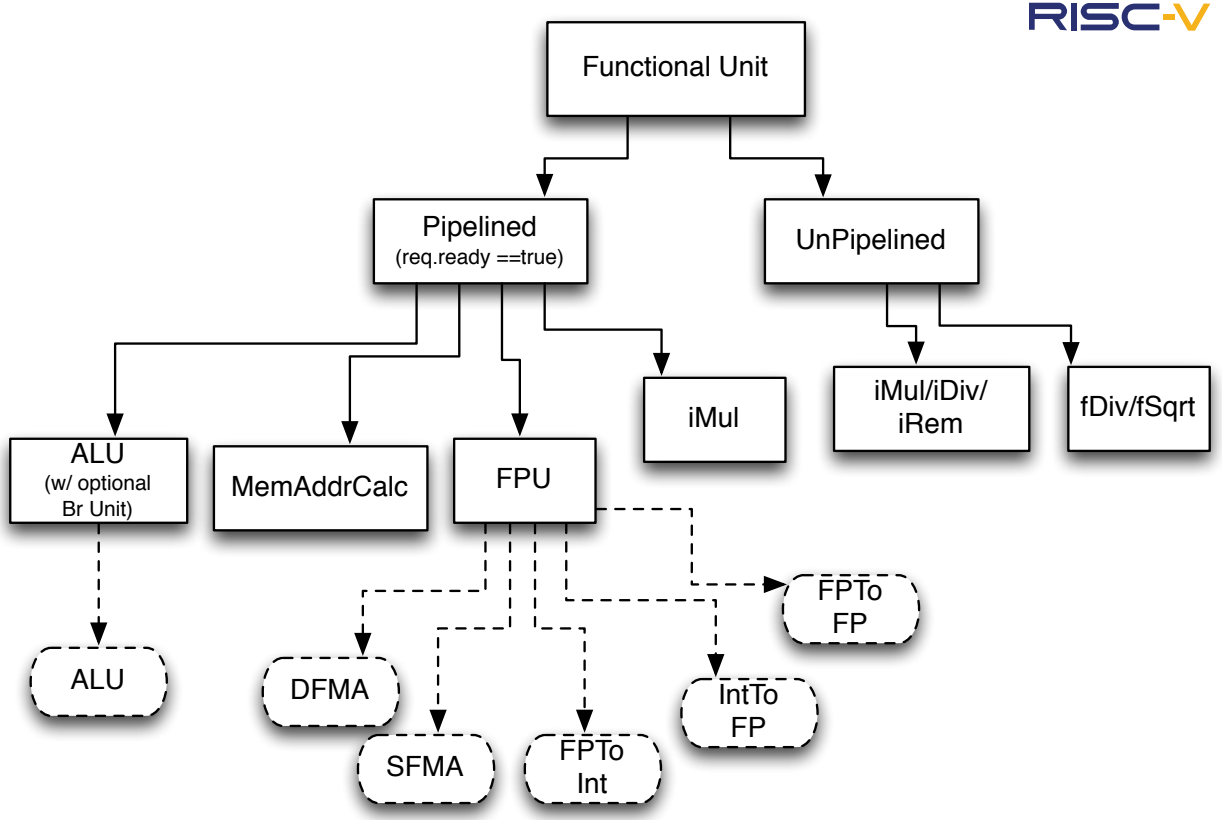
A Functional Unit Hierarchy

- **Abstract FunctionalUnit**
  - describes common IO
- **Pipelined/Unpipelined**
  - handles storing uop metadata, branch resolution, branch kills



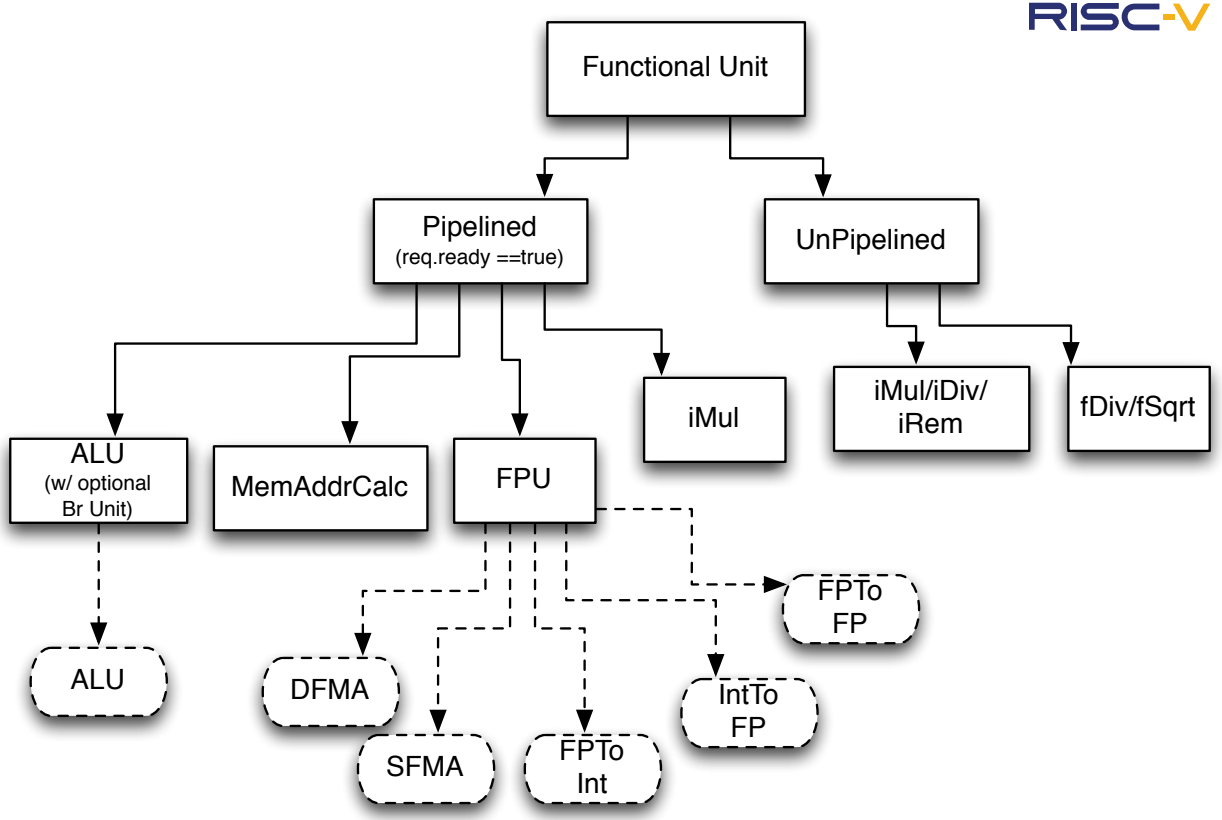
A Functional Unit Hierarchy

- **Abstract FunctionalUnit**
  - describes common IO
- **Pipelined/Unpipelined**
  - handles storing uop metadata, branch resolution, branch kills
- **Concrete Subclasses**



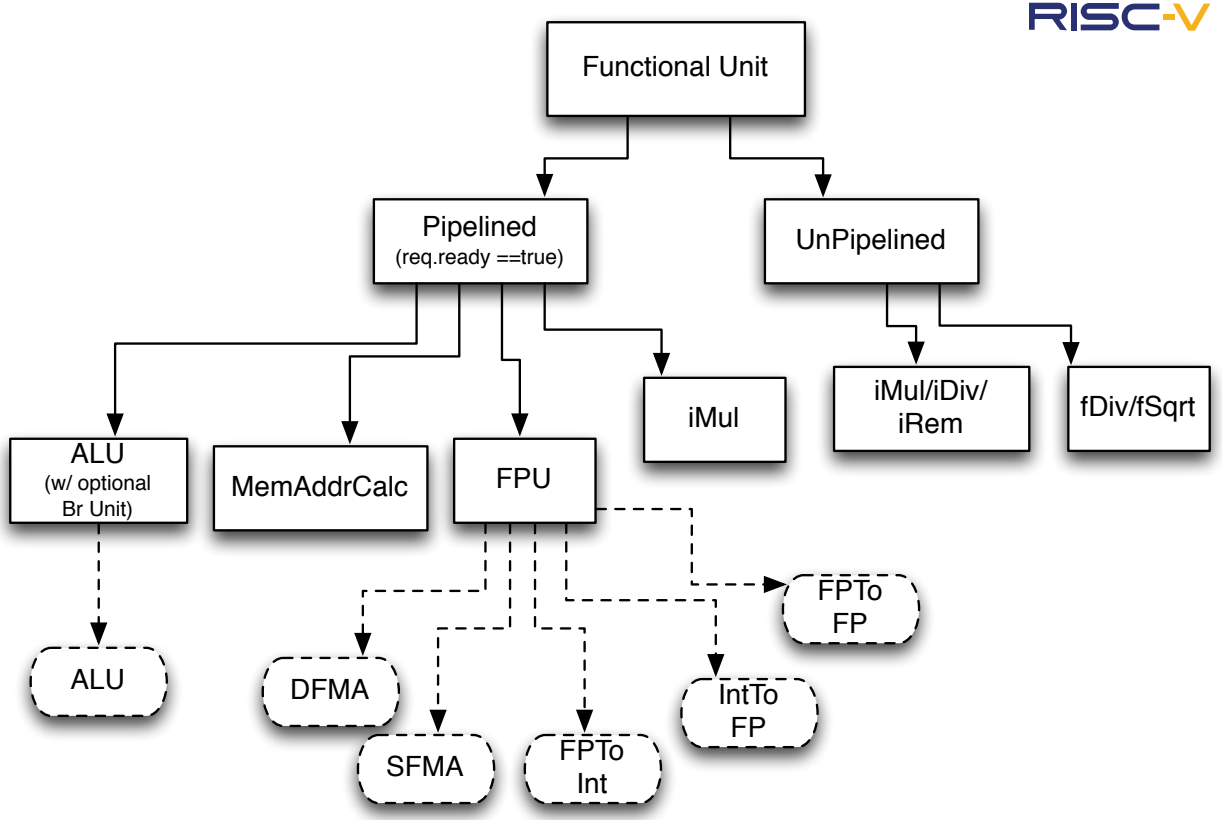
A Functional Unit Hierarchy

- **Abstract FunctionalUnit**
  - describes common IO
- **Pipelined/Unpipelined**
  - handles storing uop metadata, branch resolution, branch kills
- **Concrete Subclasses**
  - instantiates the actual expert-written FU



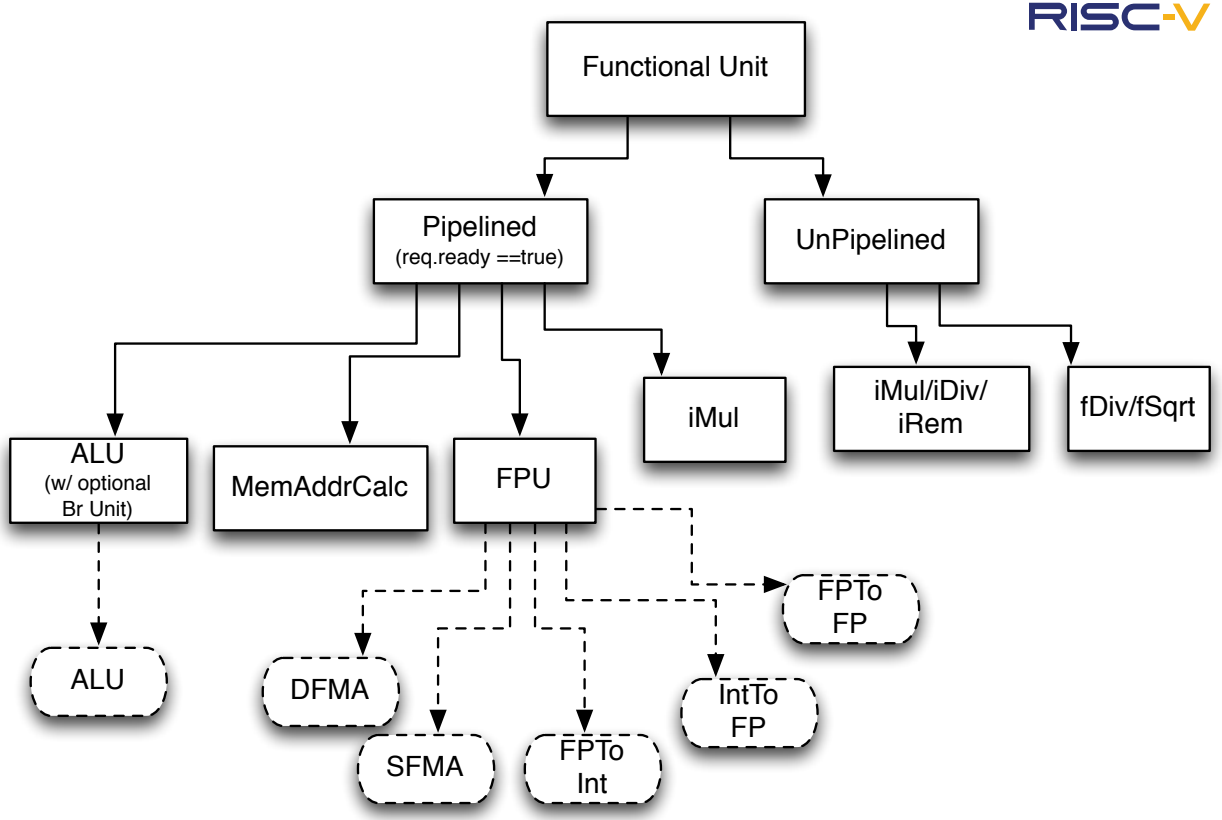
A Functional Unit Hierarchy

- **Abstract FunctionalUnit**
  - describes common IO
- **Pipelined/Unpipelined**
  - handles storing uop metadata, branch resolution, branch kills
- **Concrete Subclasses**
  - instantiates the actual expert-written FU
  - no modifications required to get FU working with speculative OoO



A Functional Unit Hierarchy

- **Abstract FunctionalUnit**
  - describes common IO
- **Pipelined/Unpipelined**
  - handles storing uop metadata, branch resolution, branch kills
- **Concrete Subclasses**
  - instantiates the actual expert-written FU
  - no modifications required to get FU working with speculative OoO
  - allows easy "stealing" of external code

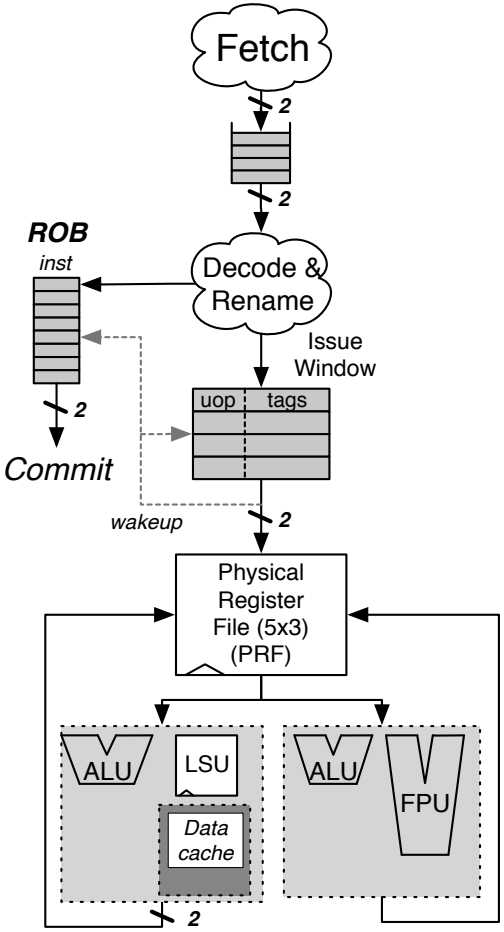


Some of the Parameters

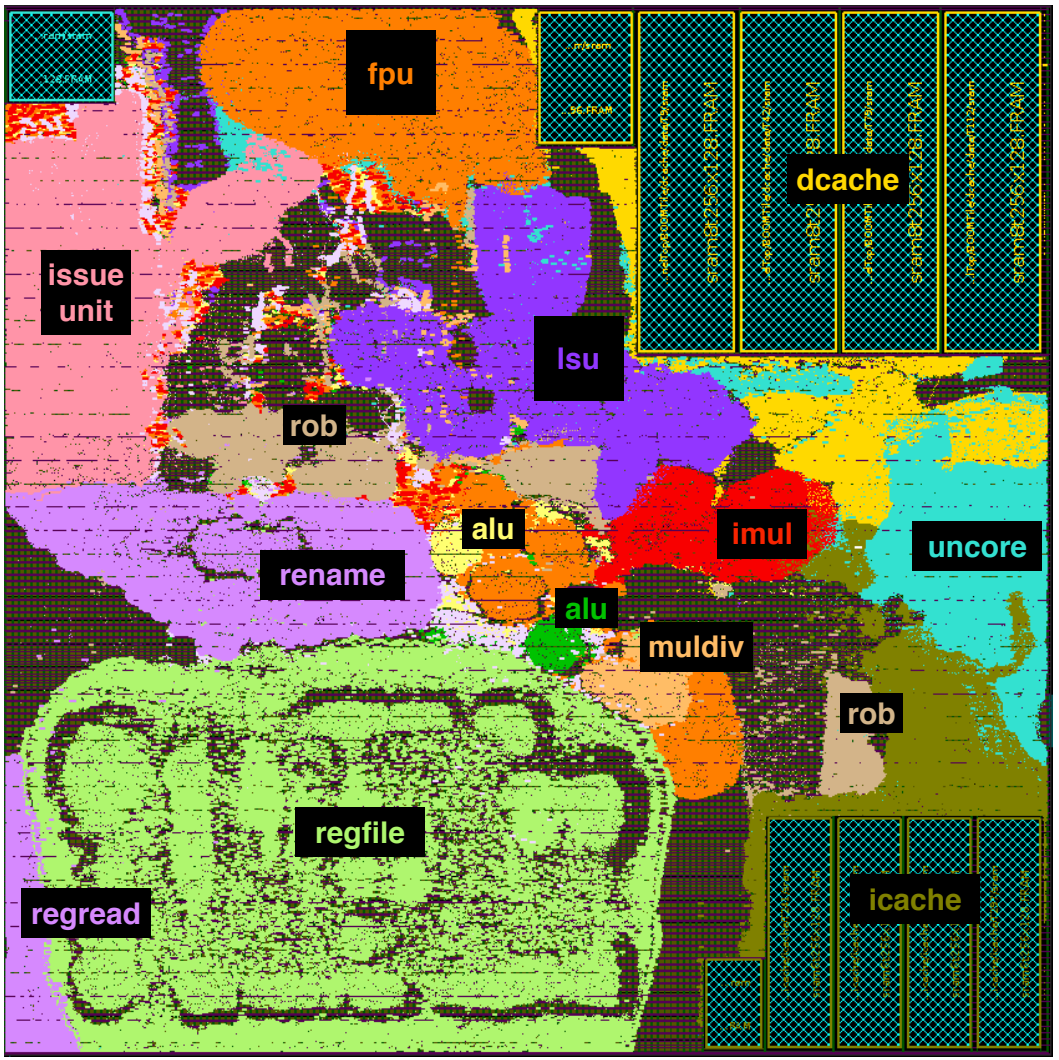
- fetch/decode/commit width (1,2,4)
- issue width
  - functional unit mix
- un-ordered vs age-ordered issue scheduler (R10k vs R12k)
- enable commit map table (R10k roll-back vs. Alpha 21264 single-cycle flush)
- rob size
- issue window size
- lsu size
- number of physical registers
- max inflight branches
- FPU latencies, iMul latencies, unpipelined mul/div bits per stage
- fetch buffer size
- flow-through fetch buffer
- enable backing branch predictor (and it's set of parameters...)
- enable uarch counters
- RAS size, BTB size
- L1 cache sets, ways, mshrs
- enable L2 (and L2 parameters...)

Synthesizable

- Targeting ASIC
- Runs on FPGA
  - Zynq zc706



TSMC 45nm floorplan



2-wide BOOM, 16kB L1 caches

1.2 mm²

preliminary results

How fast can BOOM be clocked?

- depends on parameters
- +1.5 GHz for two-wide
- (currently) designed for single-cycle SRAM access as critical path

How fast can BOOM be clocked?

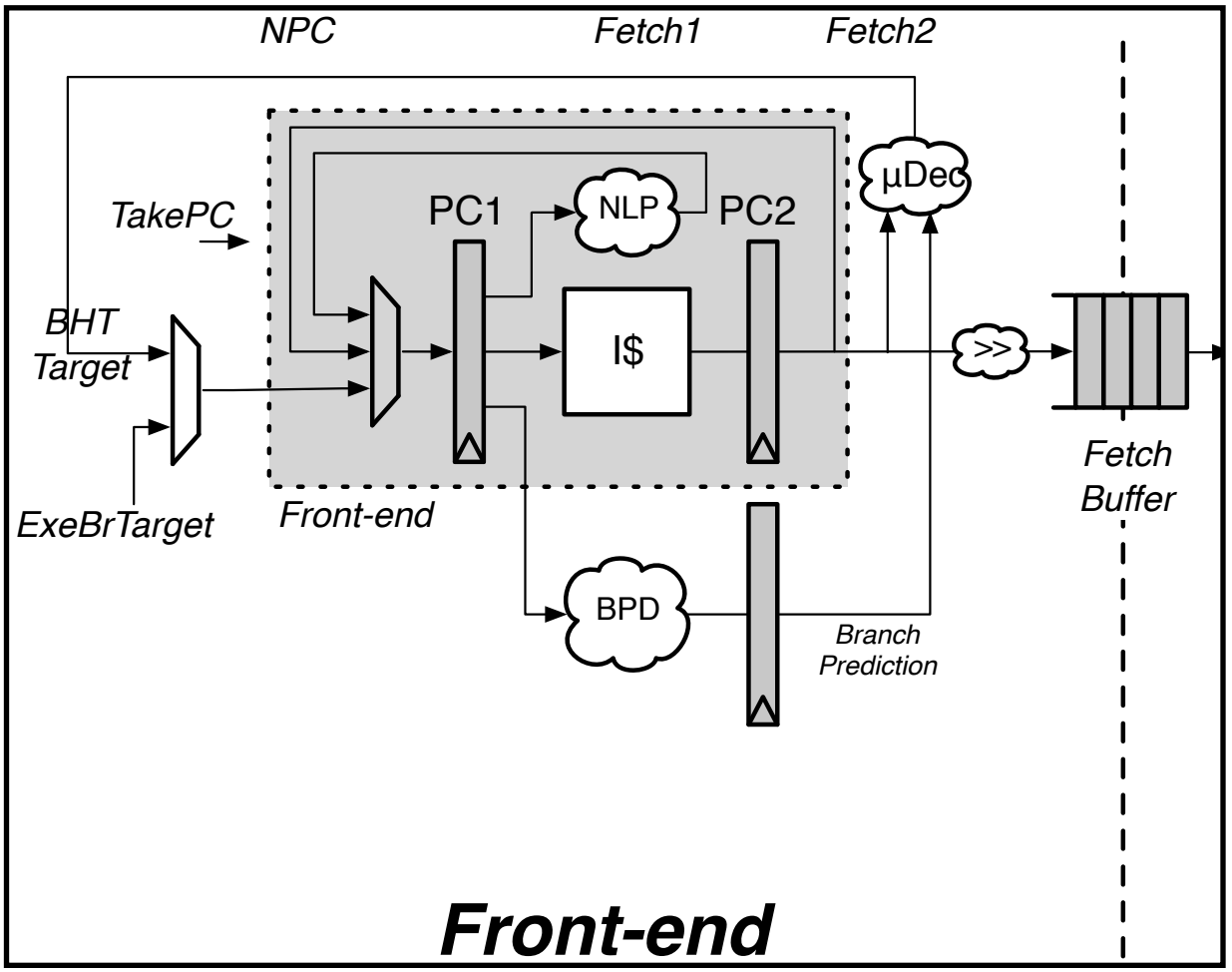
- depends on parameters
- +1.5 GHz for two-wide
- (currently) designed for single-cycle SRAM access as critical path
- upcoming tape-out will keep us honest!

How fast can BOOM be clocked?

- depends on parameters
- +1.5 GHz for two-wide
- (currently) designed for single-cycle SRAM access as critical path
- upcoming tape-out will keep us honest!
- ... and 50 MHz on FPGA
  - bottleneck is FPGA tools which can't register-retime the FPU

Full Branch Speculation Support

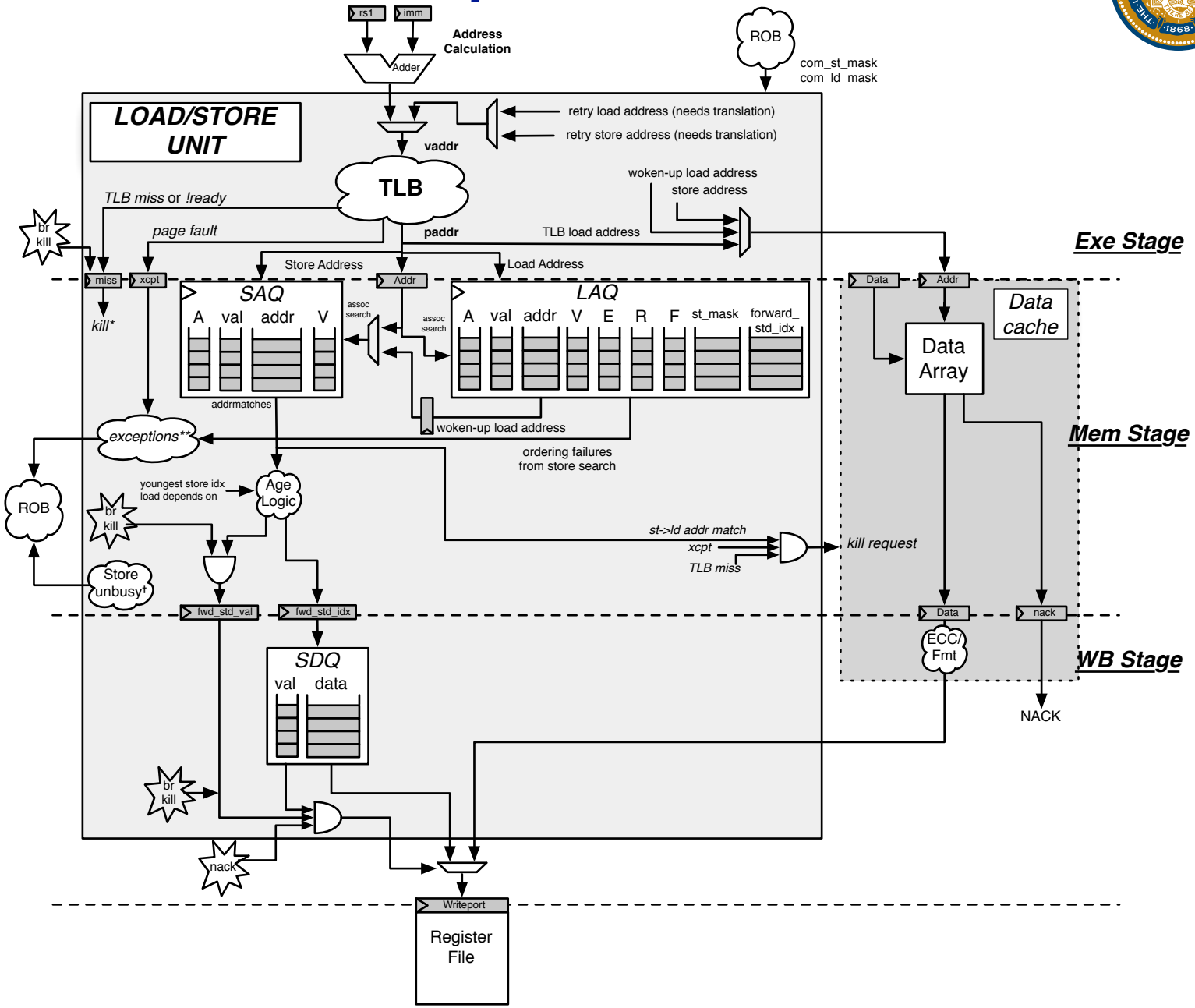
- next-line predictor (NLP)
  - BTB, BHT, RAS
  - combinational
- backing predictor (BPD)
  - global history predictor
  - SRAM (1 r/w port)



Load/Store Unit

- talks to Rocket's non-blocking data cache
- load/store queue with store ordering
  - loads execute fully **out-of-order** wrt stores, other loads
    - (based on current understanding of the RISC-V memory model)
  - store-data forwarded to loads as required

Load/Store Unit

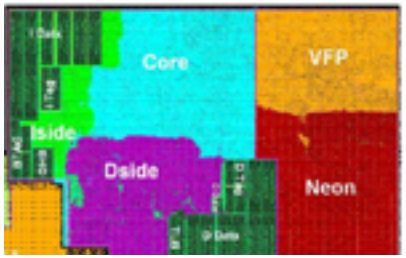


Feature Summary

Feature	BOOM
ISA	RISC-V (RV64G)
Synthesizable	√
FPGA	√
Parameterized	√
floating point	√
AMOs+LR/SC	√
caches	√
VM	√
Boots Linux	√
Multi-core	√
lines of code	**9k + 11k**

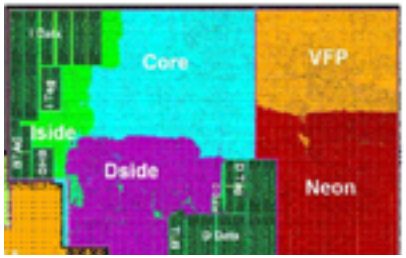
Category	ARM Cortex-A9	RISC-V BOOM-2w
ISA	32-bit ARM v7	64-bit RISC-V v2 (RV64G)
Architecture	2 wide, 3+1 issue Out-of-Order 8-stage	2 wide, 3 issue Out-of-Order 6-stage

note:
not to scale



Category	ARM Cortex-A9	RISC-V BOOM-2w
ISA	32-bit ARM v7	64-bit RISC-V v2 (RV64G)
Architecture	2 wide, 3+1 issue Out-of-Order 8-stage	2 wide, 3 issue Out-of-Order 6-stage
Performance	**3.59** CoreMarks/MHz	**3.91** CoreMarks/MHz
Process	TSMC 40GPLUS	TSMC 40GPLUS
Area with 32K caches	~2.5 mm^2	~1.00 mm^2
Area efficiency	**1.4** CoreMarks/MHz/mm^2	**3.9** CoreMarks/MHz/mm^2
Frequency	1.4 GHz	1.5 GHz

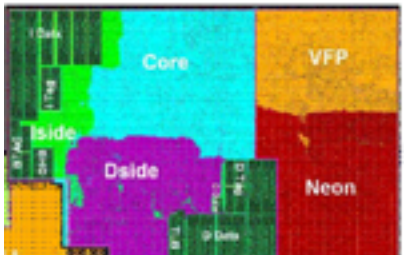
note:
not to scale



Category	ARM Cortex-A9	RISC-V BOOM-2w
ISA	32-bit ARM v7	64-bit RISC-V v2 (RV64G)
Architecture	2 wide, 3+1 issue Out-of-Order 8-stage	2 wide, 3 issue Out-of-Order 6-stage
Performance	**3.59** CoreMarks/MHz	**3.91** CoreMarks/MHz
Process	TSMC 40GPLUS	TSMC 40GPLUS
Area with 32K caches	~2.5 mm^2	~1.00 mm^2
Area efficiency	**1.4** CoreMarks/MHz/mm^2	**3.9** CoreMarks/MHz/mm^2
Frequency	1.4 GHz	1.5 GHz

+9%!

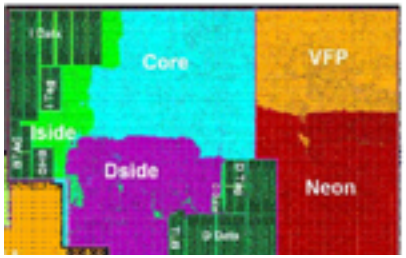
note:
not to scale



Category	ARM Cortex-A9	RISC-V BOOM-2w
ISA	32-bit ARM v7	64-bit RISC-V v2 (RV64G)
Architecture	2 wide, 3+1 issue Out-of-Order 8-stage	2 wide, 3 issue Out-of-Order 6-stage
Performance	**3.59** CoreMarks/MHz	**3.91** CoreMarks/MHz
Process	TSMC 40GPLUS	TSMC 40GPLUS
Area with 32K caches	~2.5 mm^2	~1.00 mm^2
Area efficiency	**1.4** CoreMarks/MHz/mm^2	**3.9** CoreMarks/MHz/mm^2
Frequency	1.4 GHz	1.5 GHz
Power	**0.5-1.9 W** (2 cores + L2) @ TSMC 40nm, 0.8-2.0 GHz	**0.25 W** (1 core + L1) @ TSMC 45nm, 1 GHz

+9%!

note:
not to scale



Challenges executing SPECint

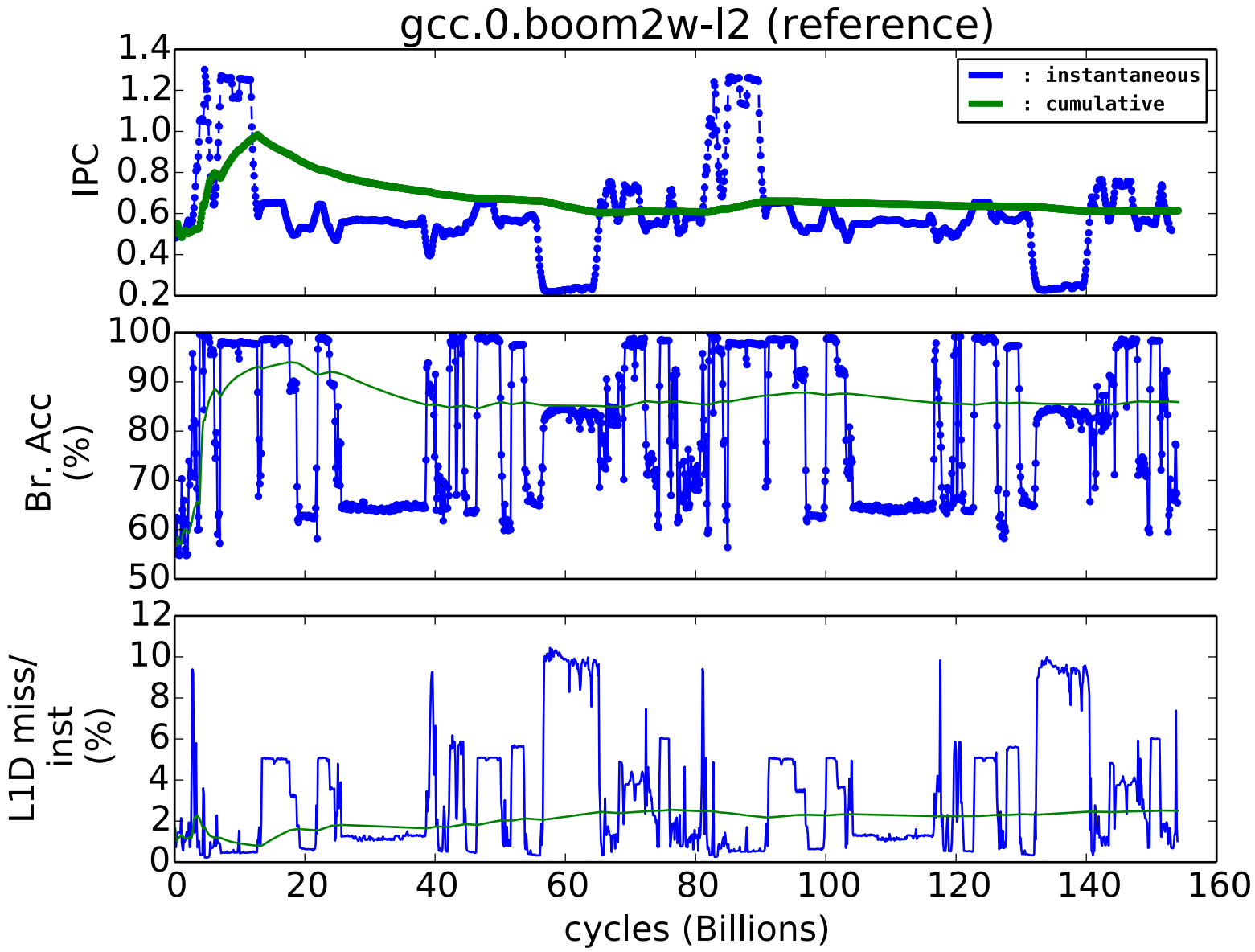
Challenges executing SPECint

- 12 benchmarks in CINT2006
  - 35 workloads written in C or C++ (--reference)
  - average 2 trillion instructions
  - ~24T instructions total
    - (**2 hours** for a 3 GHz Intel processor at CPI=1)
    - (**7 years** for a sw simulator @ 100 KIPS)

Challenges executing SPECint

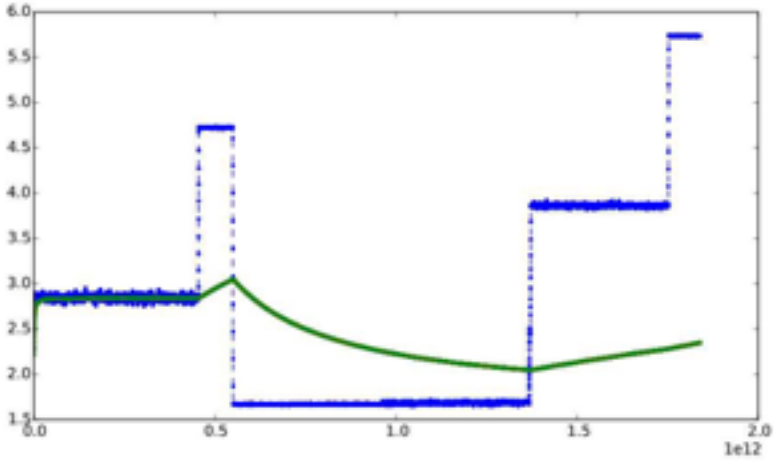
- 12 benchmarks in CINT2006
  - 35 workloads written in C or C++ (--reference)
  - average 2 trillion instructions
  - ~24T instructions total
    - (**2 hours** for a 3 GHz Intel processor at CPI=1)
    - (**7 years** for a sw simulator @ 100 KIPS)
- RISC-V
  - requires libc (riscv64-unknown-linux-gnu*)
  - run on Linux
  - **https://github.com/ccelio/Speckle**
    - useful for generating portable SPEC directories

Preliminary Data

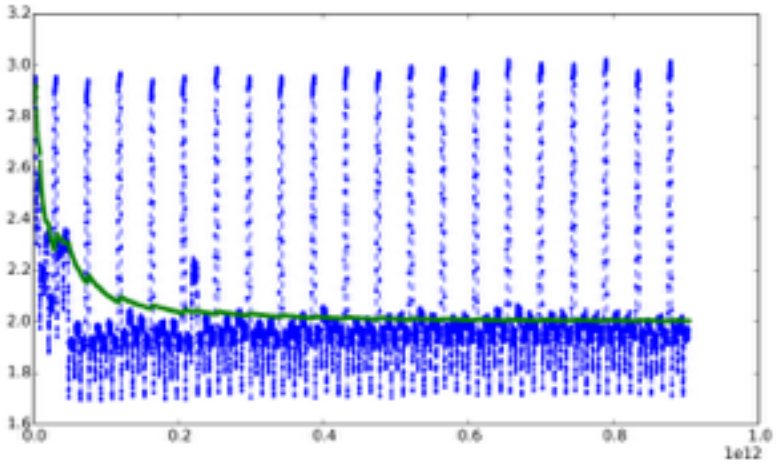


SPEC Score

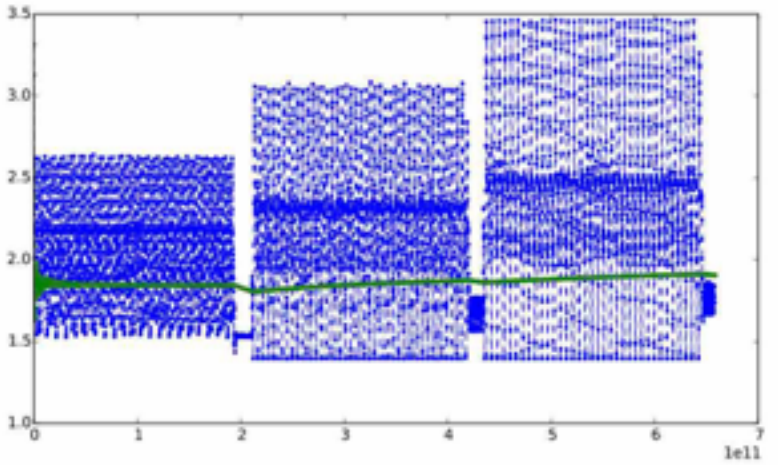
astar.1



h264.1



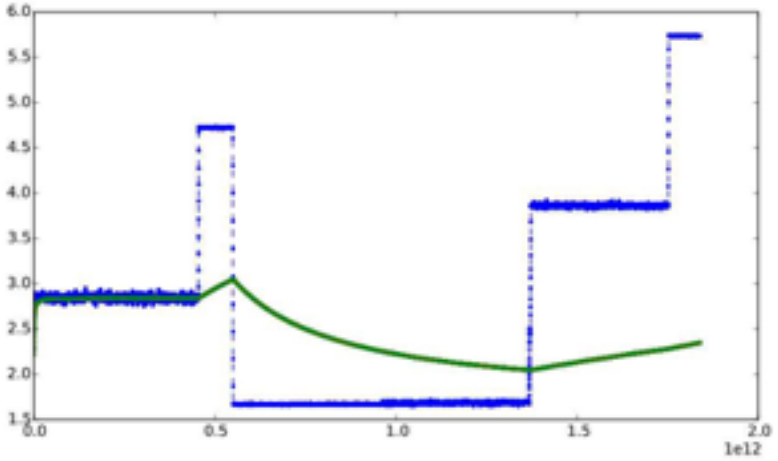
bzip2.2



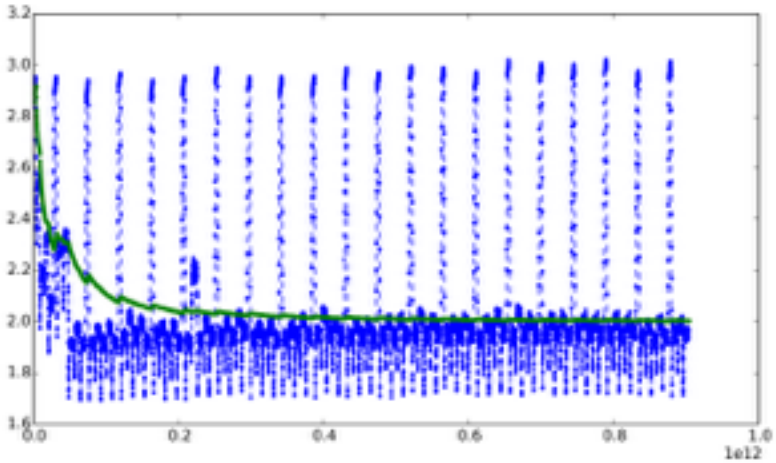
SPEC Score

- TODO :(

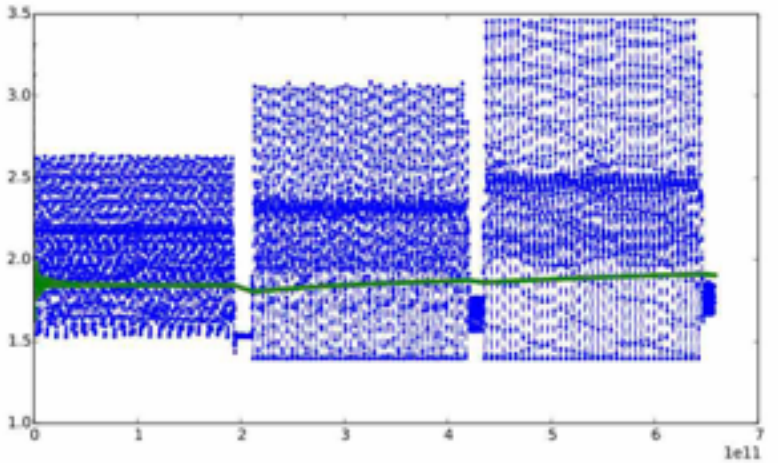
astar.1



h264.1



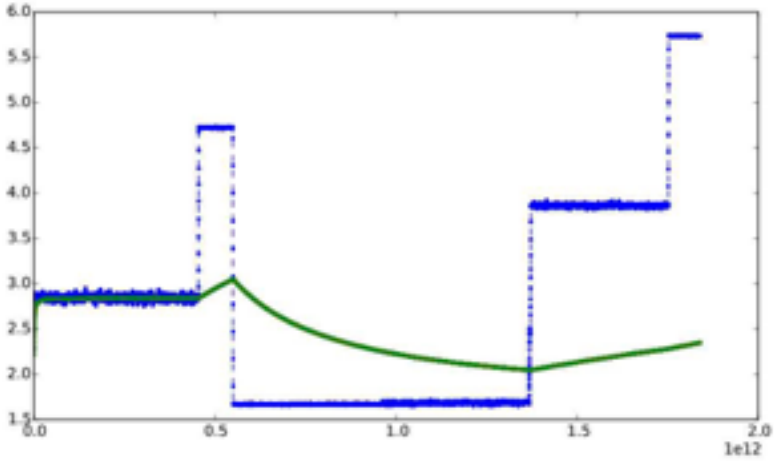
bzip2.2



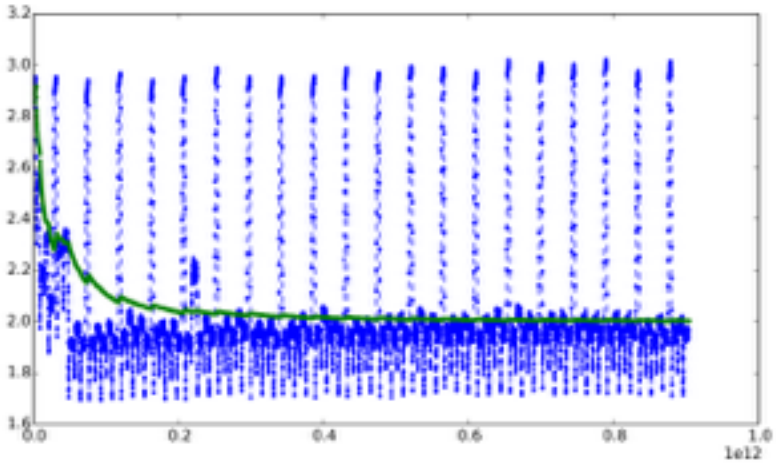
SPEC Score

- TODO :(
- need more DRAM

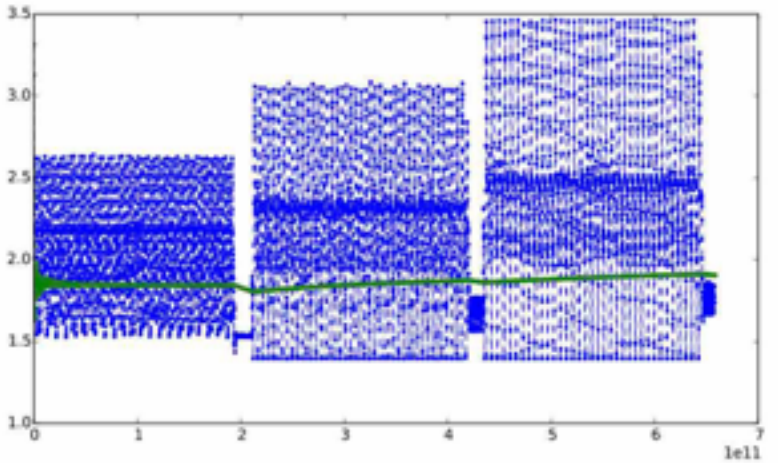
astar.1



h264.1



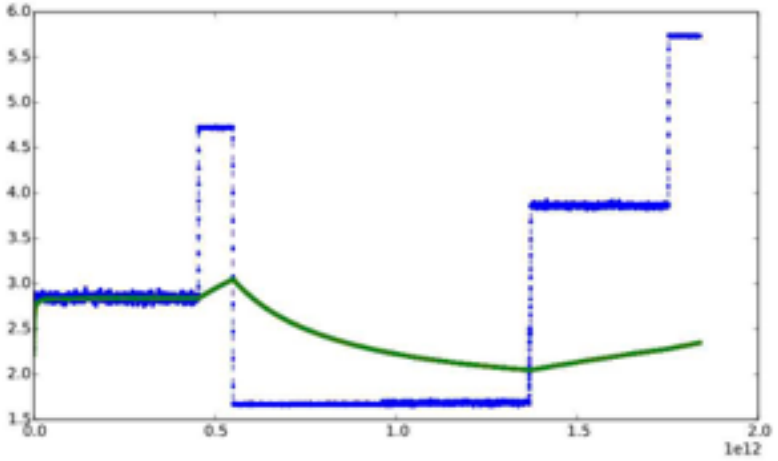
bzip2.2



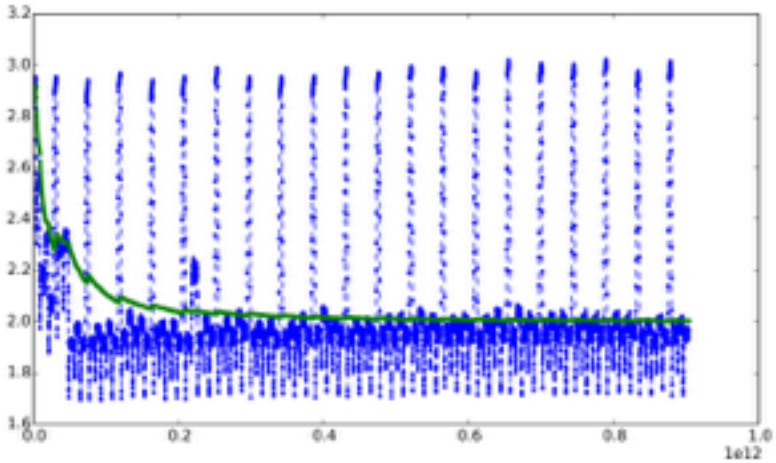
SPEC Score

- TODO :(
- need more DRAM
- need DRAM emulation

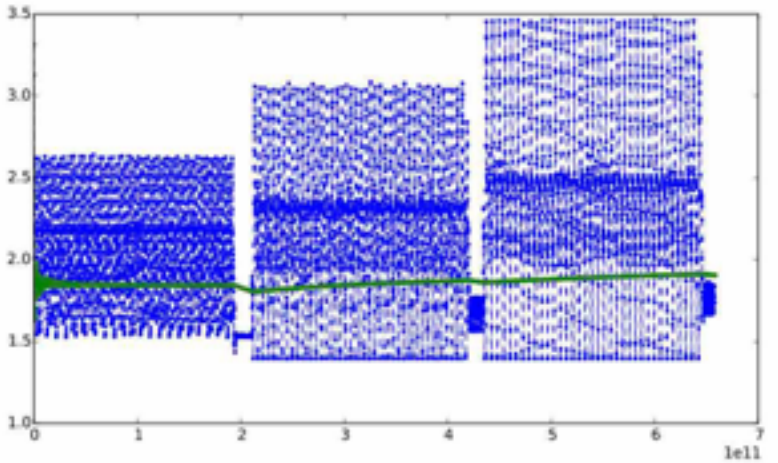
astar.1



h264.1



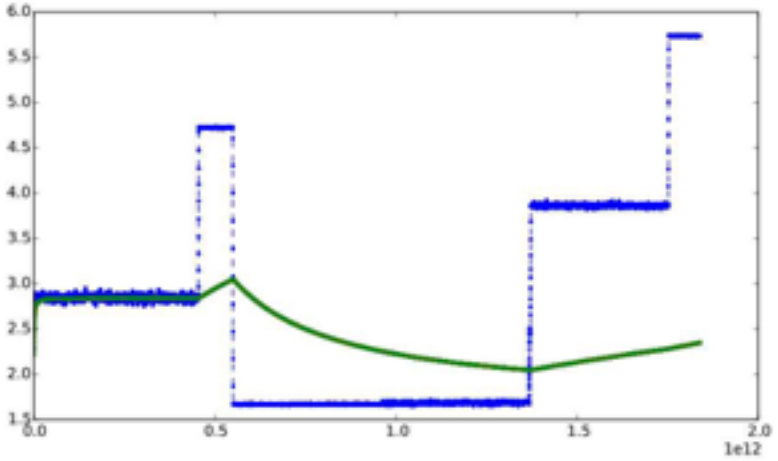
bzip2.2



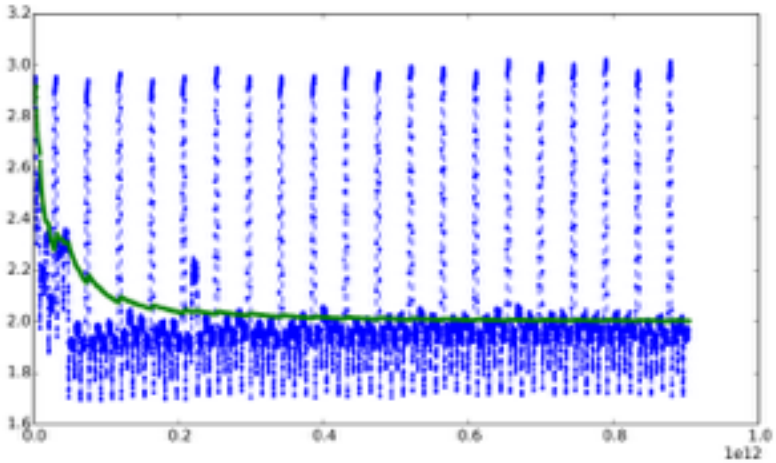
SPEC Score

- TODO :(
- need more DRAM
- need DRAM emulation
- will take ~1 day with cluster

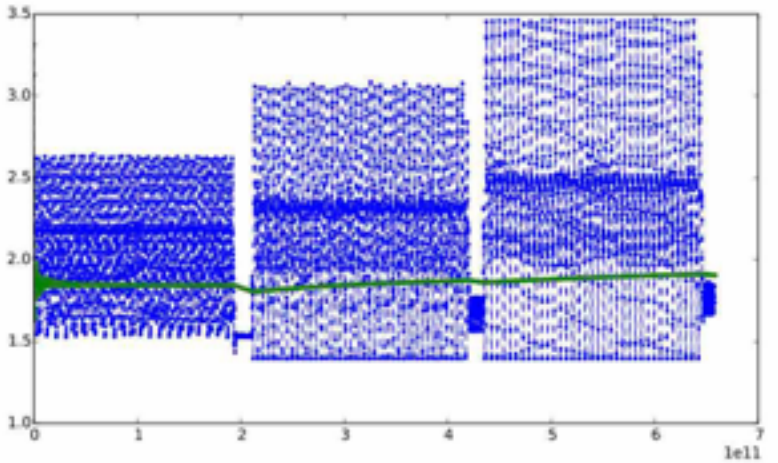
astar.1



h264.1



bzip2.2



How to use BOOM...

Repositories

- BOOM
  - https://github.com/ucb-bar/riscv-boom
  - just the BOOM core code
- Rocket-chip
  - https://github.com/ucb-bar/rocket-chip
  - the rest of the SoC
  - chisel, rocket, uncore, junctions, riscv-tools, fpga-zynq, etc.
  - generates C++ emulator, verilog for FPGA
  - http://riscv.org/tutorial-hpca2015/riscv-rocket-chip-tutorial-bootcamp-hpca2015.pdf (for more information)

BOOM Quick-start

```
$ export ROCKETCHIP_ADDONS="boom"
$ git clone https://github.com/ucb-bar/rocket-chip.git
$ cd rocket-chip
$ git checkout boom
$ git submodule update --init
$ cd riscv-tools
$ git submodule update --init --recursive riscv-tests
$ cd ../emulator; make run CONFIG=BOOMCPPConfig
```

BOOM Quick-start

```
$ export ROCKETCHIP_ADDONS="boom"
$ git clone https://github.com/ucb-bar/rocket-chip.git
$ cd rocket-chip
$ git checkout boom
$ git submodule update --init
$ cd riscv-tools
$ git submodule update --init --recursive riscv-tests
$ cd ../emulator; make run CONFIG=BOOMCPPConfig
```

- "BOOM-chip" is (currently) a branch of rocket-chip

BOOM Quick-start

```
$ export ROCKETCHIP_ADDONS="boom"
$ git clone https://github.com/ucb-bar/rocket-chip.git
$ cd rocket-chip
$ git checkout boom
$ git submodule update --init
$ cd riscv-tools
$ git submodule update --init --recursive riscv-tests
$ cd ../emulator; make run CONFIG=BOOMCPPConfig
```

- "BOOM-chip" is (currently) a branch of rocket-chip
- "make run" builds and runs riscv-tests suite

BOOM Quick-start

```
$ export ROCKETCHIP_ADDONS="boom"
$ git clone https://github.com/ucb-bar/rocket-chip.git
$ cd rocket-chip
$ git checkout boom
$ git submodule update --init
$ cd riscv-tools
$ git submodule update --init --recursive riscv-tests
$ cd ../emulator; make run CONFIG=BOOMCPPConfig
```

- "BOOM-chip" is (currently) a branch of rocket-chip
- "make run" builds and runs riscv-tests suite
- there are many different CONFIGs available!
  - rocket-chip/src/main/scala/PrivateConfigs.scala
  - rocket-chip/boom/src/main/scala/configs.scala
  - slightly different configs for different targets (CPP, FPGA, VLSI)

How do you verify and debug BOOM?

How do you verify and debug BOOM?

- parameterization makes this tough!

How do you verify and debug BOOM?

- parameterization makes this tough!
- tested with:
  - riscv-tests
    - (assembly functional tests + bare-metal micro-benchmarks)
  - CoreMark + riscv-pk
  - SPEC + Linux
  - riscv-torture

riscv-torture

- **now open-source!**
  - https://github.com/ucb-bar/riscv-torture
- torture generates random tests to stress the core pipeline
- runs test on Spike and your processor
  - architectural register state dumped to memory on program termination
  - diff state
  - if error found, finds smallest version of test that exhibits an error

riscv-torture quick-start

Run Rocket-chip (BOOM-chip?)

```
$ git clone https://github.com/ucb-bar/rocket-chip.git
$ cd rocket-chip
$ git submodule update --init
$ cd riscv-tools
$ git submodule update --init --recursive riscv-tests
$ cd ../emulator; make run CONFIG=BOOMCPPConfig
```

Run Torture

```
$ cd rocket-chip
$ git clone https://github.com/ucb-bar/riscv-torture.git
$ cd riscv-torture
$ git submodule update --init
$ vim Makefile  # change RTL_CONFIG=BOOMCPPConfig
$ make igentest # test that torture works, gen a single test
$ make cnight   # run C++ emulator overnight
```

Commit Logging

- BOOM can generate commit logs
  - (priv level, PC, inst, wb raddr, wb data)
- Spike supports generating commit logs!
  - configure Spike with "--enable-commitlog" flag
  - outputs to stderr

```
# modify the build.sh in riscv-tools
build_project riscv-isa-sim --prefix=$RISCV --with-fesvr=$RISCV --enable-commitlog
```

Commit Logging

- BOOM can generate commit logs
  - (priv level, PC, inst, wb raddr, wb data)
- Spike supports generating commit logs!
  - configure Spike with "--enable-commitlog" flag
  - outputs to stderr
- only semi-automated
  - valid differences from Spike and hw
  - e.g., spinning on LR+SC

```
# modify the build.sh in riscv-tools
build_project riscv-isa-sim --prefix=$RISCV --with-fesvr=$RISCV --enable-commitlog
```

What documentation is available?

- A design document is in progress
  - https://github.com/ccelio/riscv-boom-doc
- Wiki
  - https://github.com/ucb-bar/riscv-boom/wiki

Future Plans

Future Plans

- tape-out this year

Future Plans

- tape-out this year
- uncached loads/stores

Future Plans

- tape-out this year
- uncached loads/stores
- update to Chisel 3.0

Future Plans

- tape-out this year
- uncached loads/stores
- update to Chisel 3.0
- SPEC score
  - improve branch prediction
  - memory ordering speculation (support ld->ld ordering?)

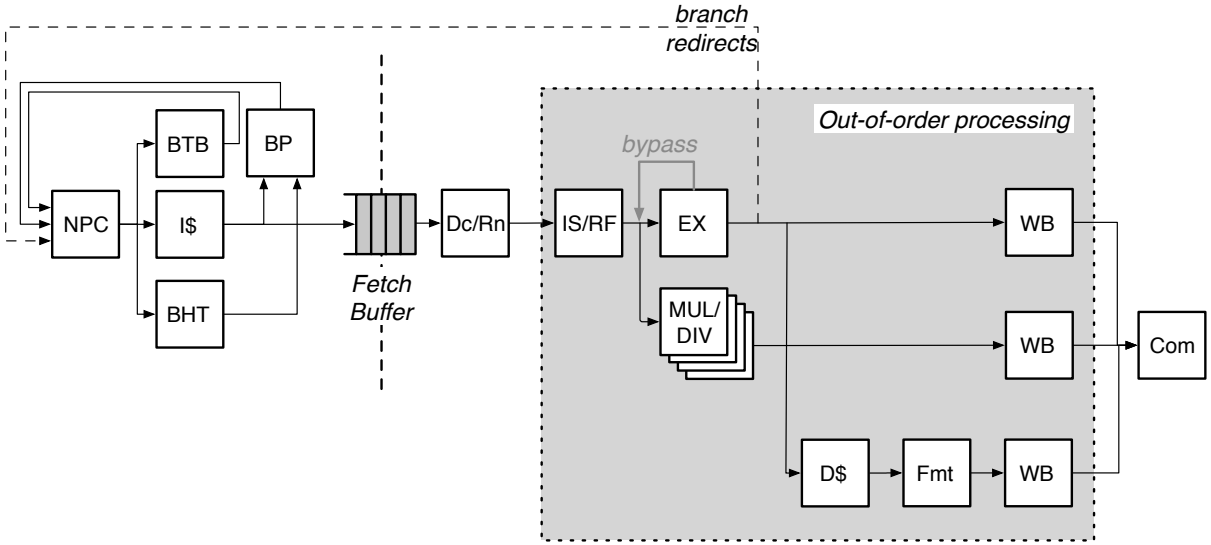
Future Plans

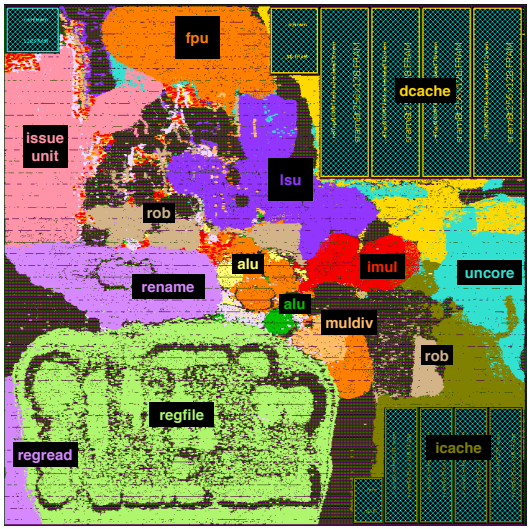
- tape-out this year
- uncached loads/stores
- update to Chisel 3.0
- SPEC score
  - improve branch prediction
  - memory ordering speculation (support ld->ld ordering?)
- finish documentation

Future Plans

- tape-out this year
- uncached loads/stores
- update to Chisel 3.0
- SPEC score
  - improve branch prediction
  - memory ordering speculation (support ld->ld ordering?)
- finish documentation
- build a community of Baby Boomers?

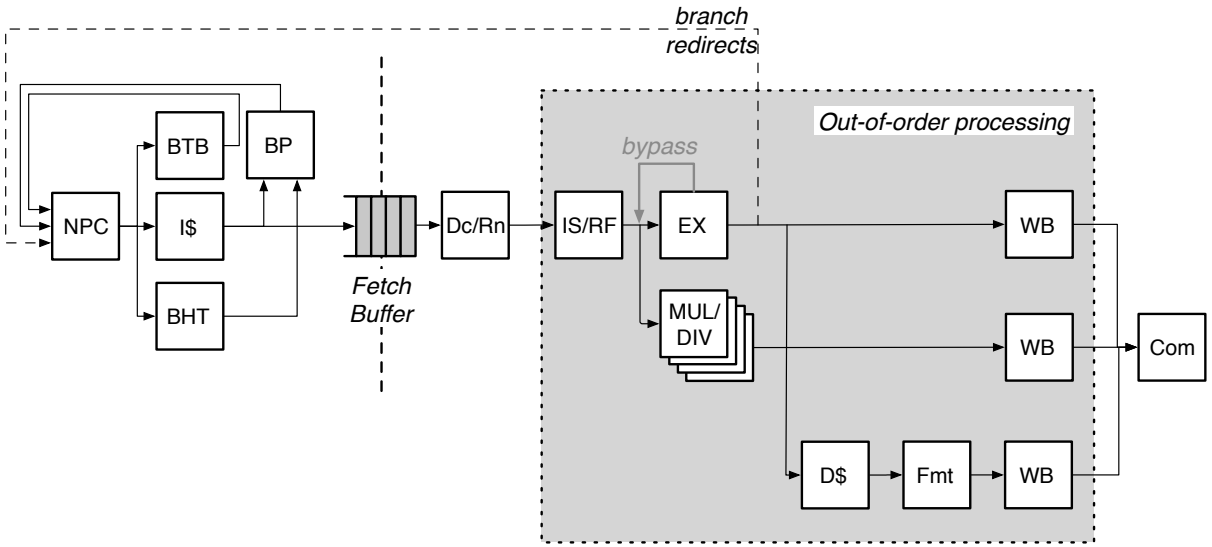
Conclusion

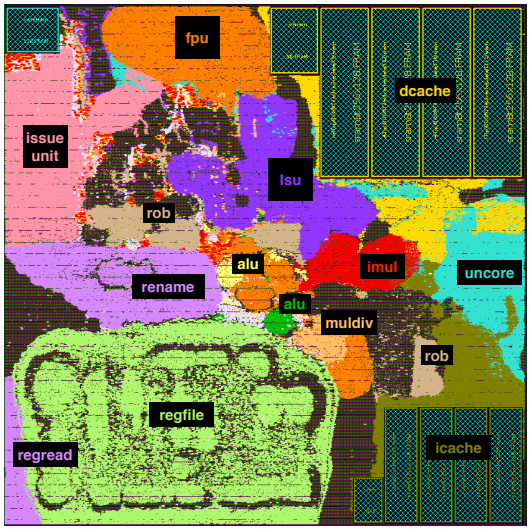

branch
redirects
Out-of-order processing
bypass
BTB
BP
NPC
I$
BHT
Fetch
Buffer
Dc/Rn
IS/RF
EX
MUL/
DIV
D$
Fmt
WB
WB
WB
Com


fpu
dcache
issue
unit
lsu
rob
alu
imul
uncore
rename
alu
muldiv
rob
regfile
regread
icache

Conclusion

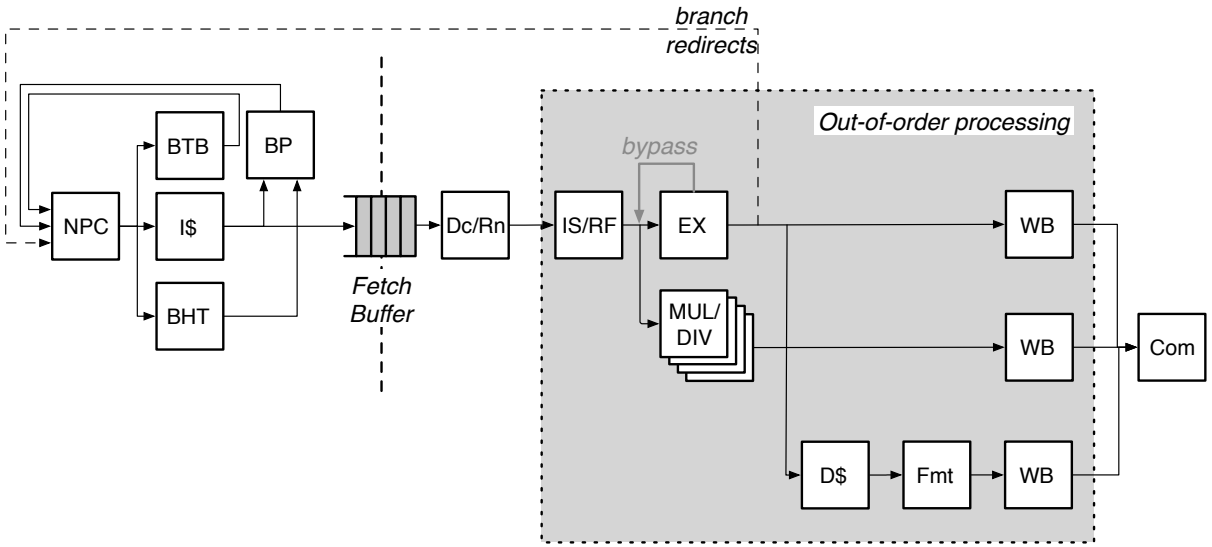
- BOOM is RV64G, runs SPEC on Linux on an FPGA

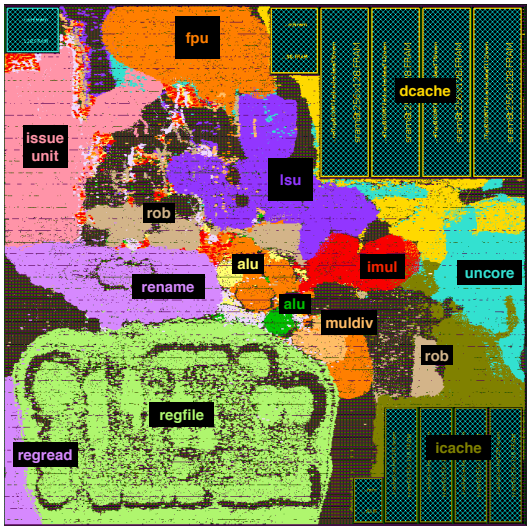




Conclusion

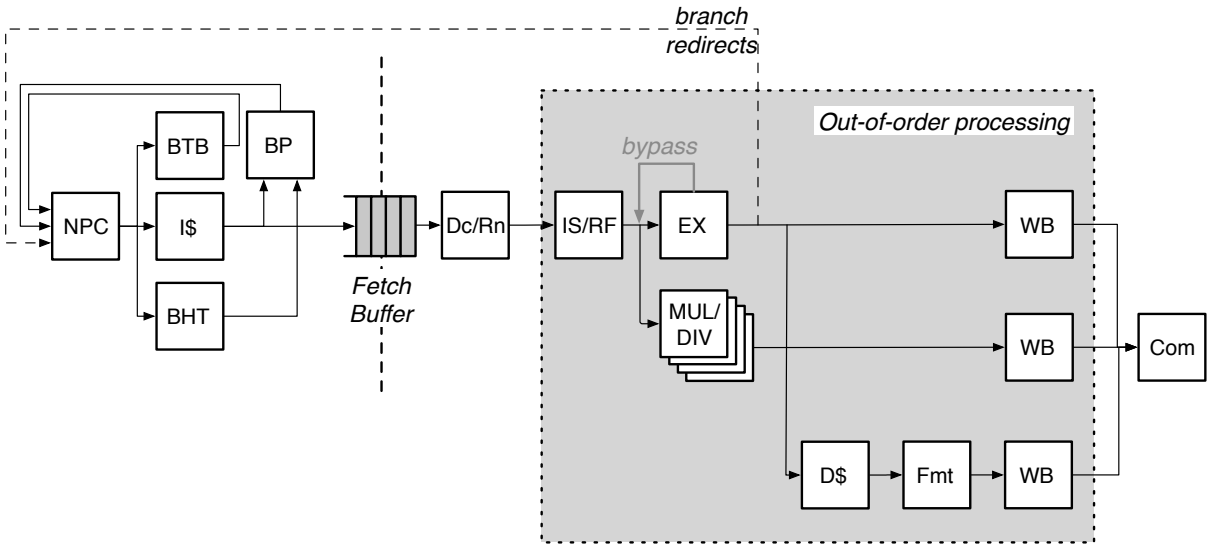
- BOOM is RV64G, runs SPEC on Linux on an FPGA
- BOOM is ~10k loc and 4 person-years of work

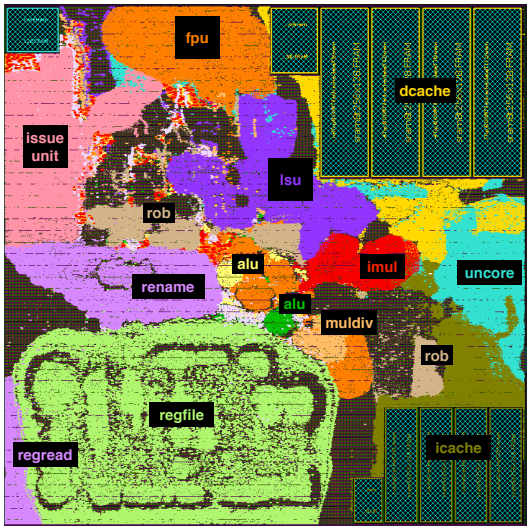




Conclusion

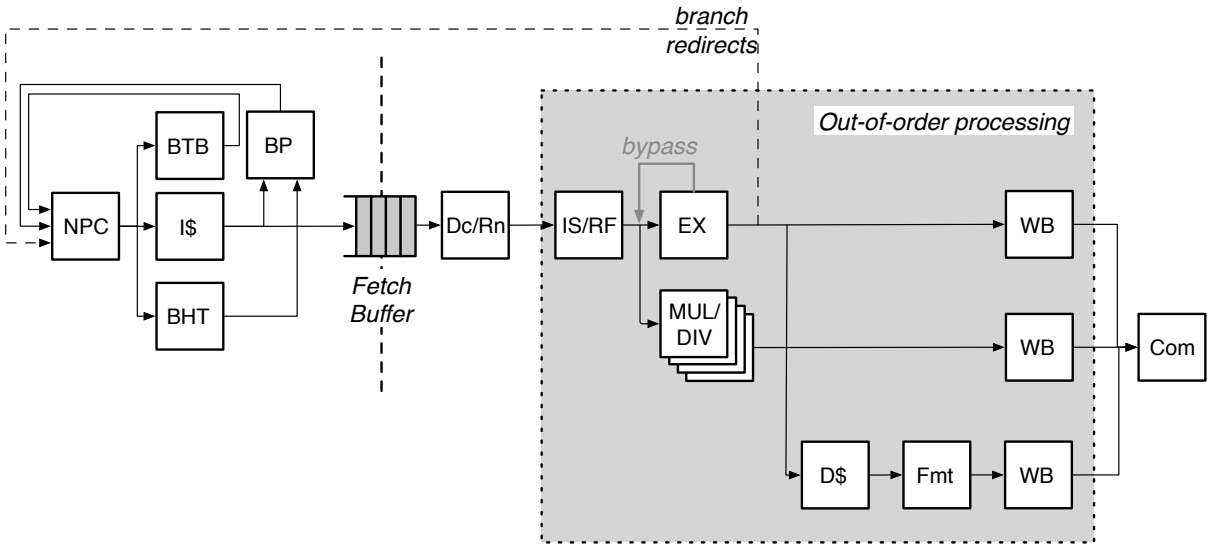
- BOOM is RV64G, runs SPEC on Linux on an FPGA
- BOOM is ~10k loc and 4 person-years of work
- excellent platform for prototyping new ideas

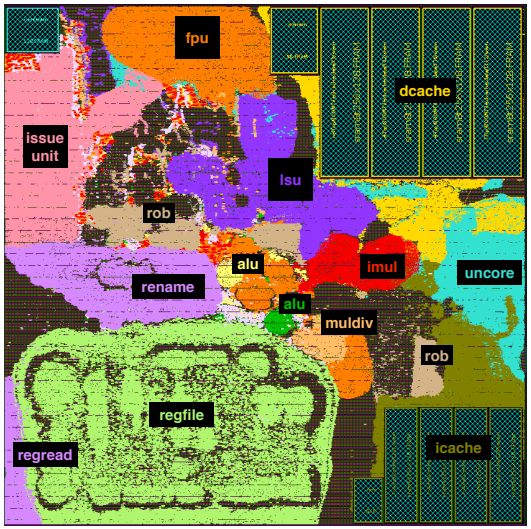




Conclusion

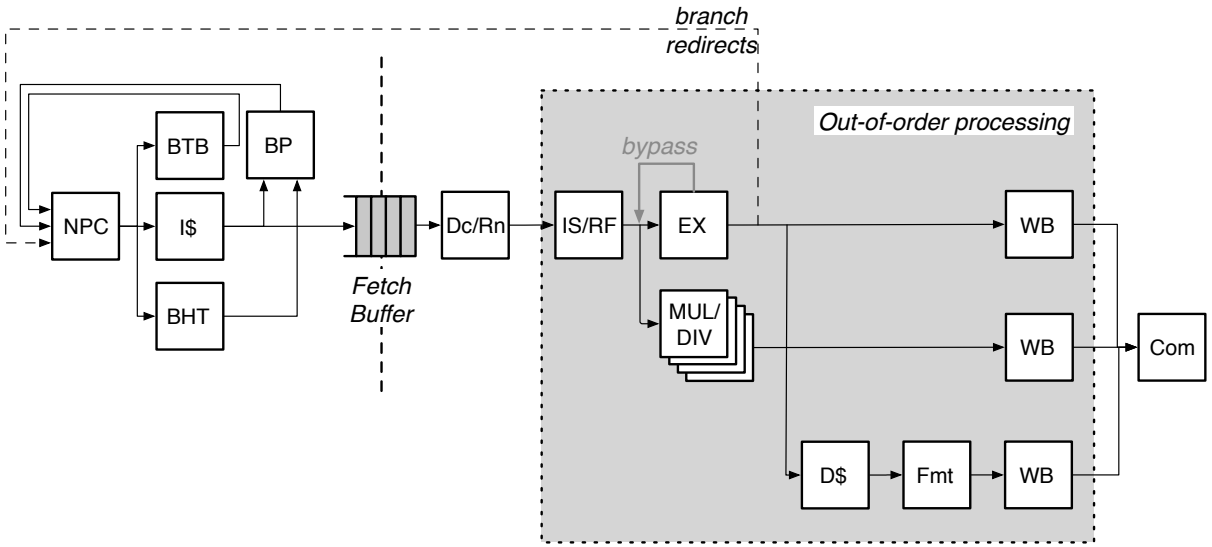
- BOOM is RV64G, runs SPEC on Linux on an FPGA
- BOOM is ~10k loc and 4 person-years of work
- excellent platform for prototyping new ideas
- now **open-source**!

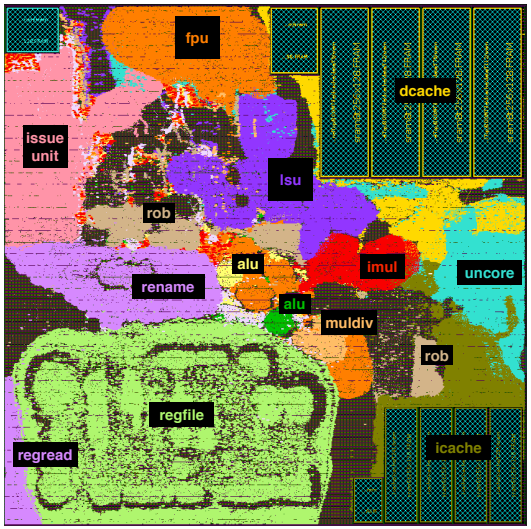




Conclusion

- BOOM is RV64G, runs SPEC on Linux on an FPGA
- BOOM is ~10k loc and 4 person-years of work
- excellent platform for prototyping new ideas
- now **open-source**!
- we'll continue to support and improve BOOM





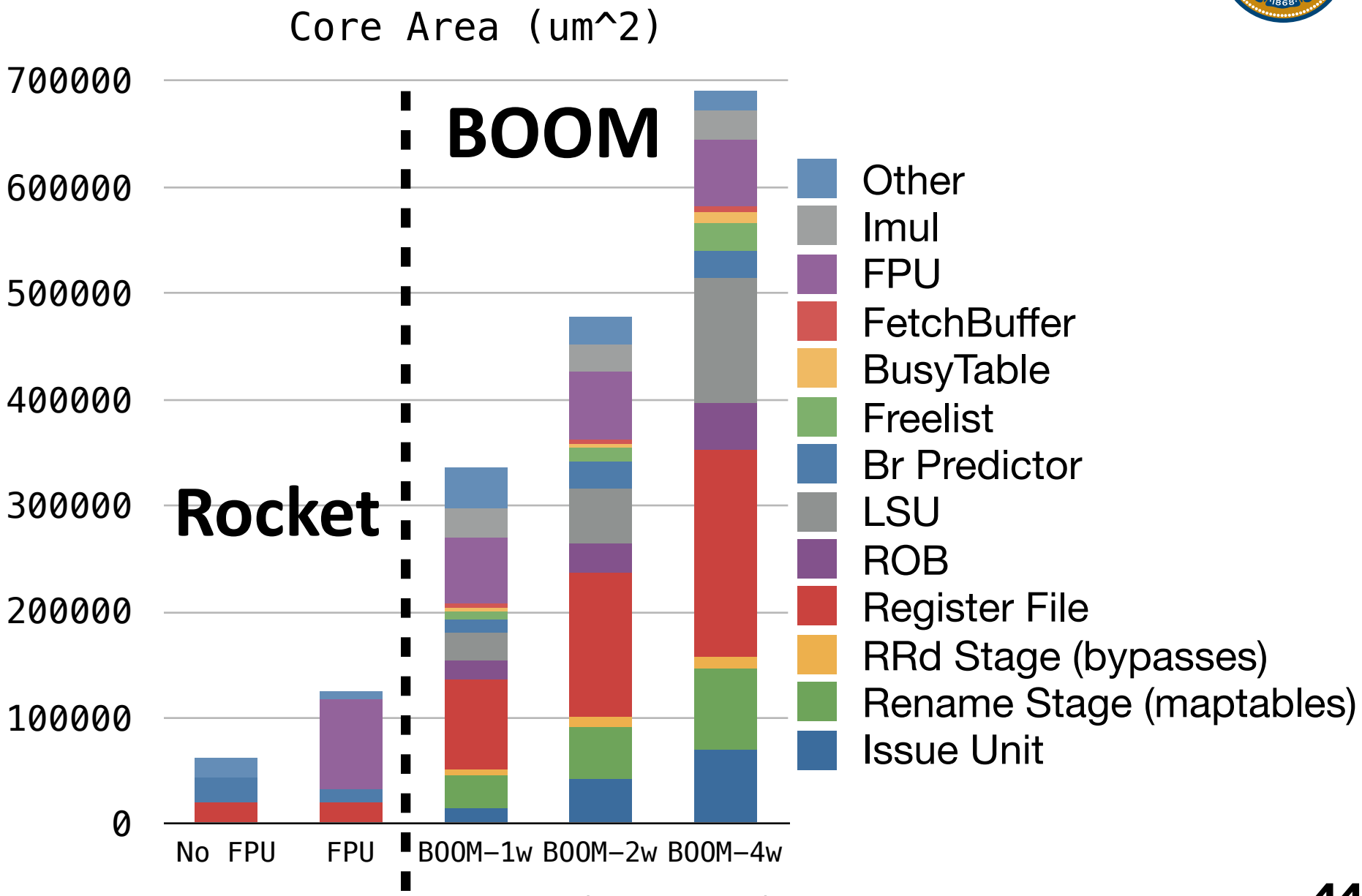
Questions?

Funding Acknowledgements

- *Research partially funded by DARPA Award Number HR0011-12-2-0016, the Center for Future Architecture Research, a member of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and ASPIRE Lab industrial sponsors and affiliates Intel, Google, Huawei, Nokia, NVIDIA, Oracle, and Samsung.*
- *Approved for public release; distribution is unlimited. The content of this presentation does not necessarily reflect the position or the policy of the US government and no official endorsement should be inferred.*
- *Any opinions, findings, conclusions, or recommendations in this paper are solely those of the authors and does not necessarily reflect the position or the policy of the sponsors.*

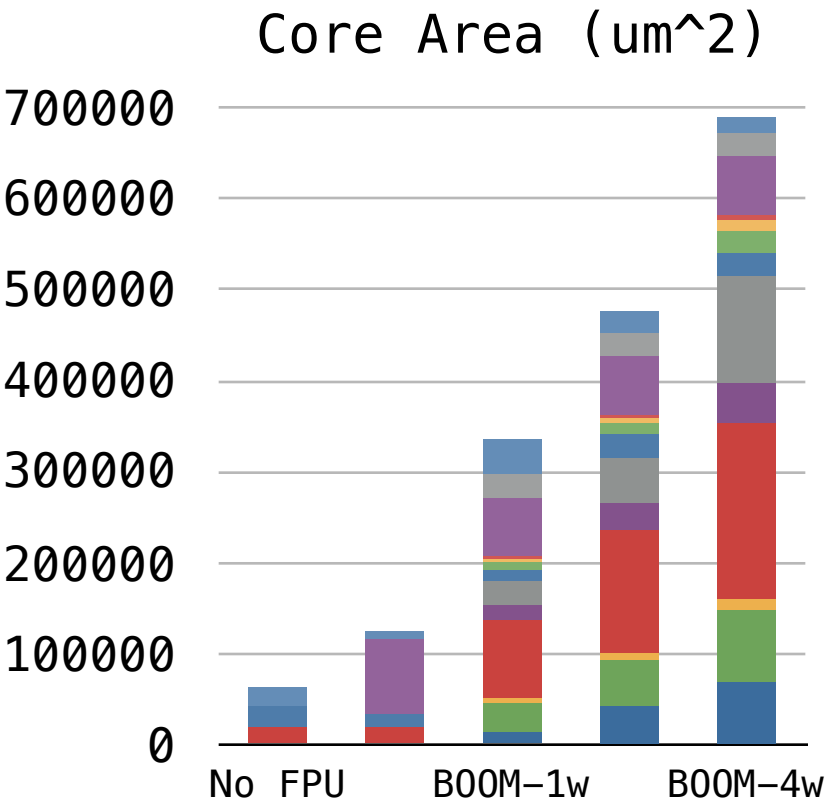
Extra Slides

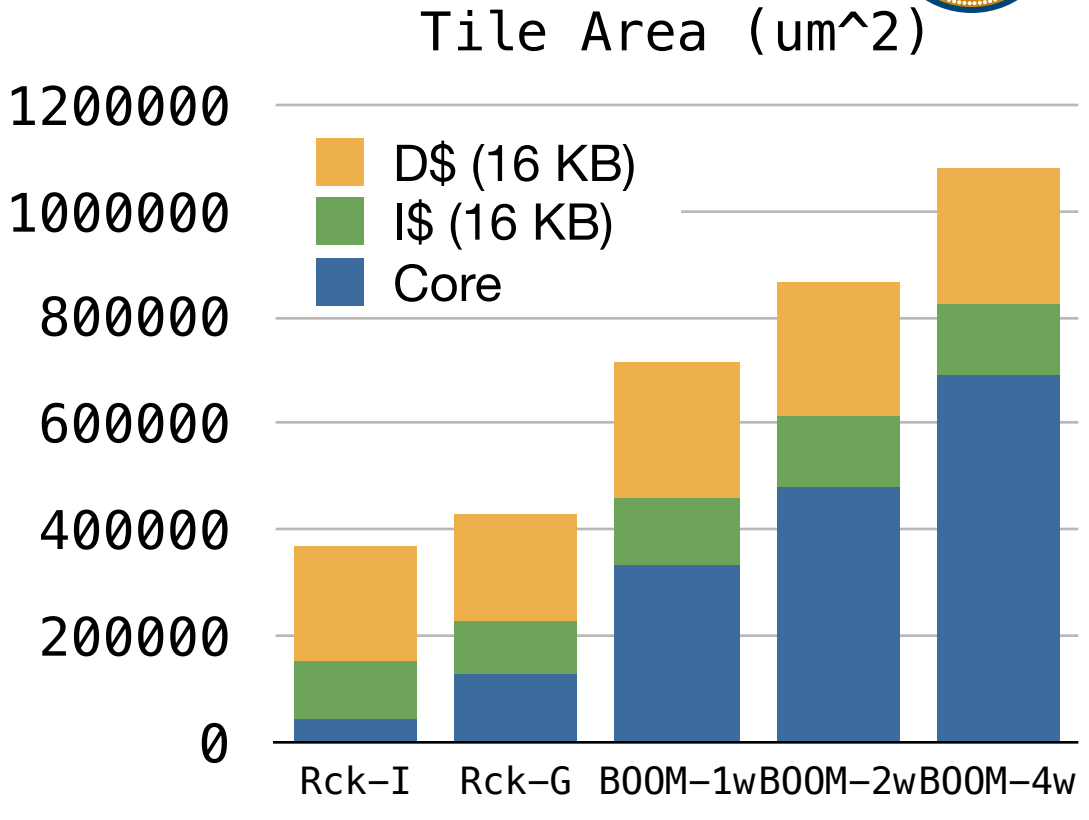
Synthesis Results



preliminary results

Synthesis Results

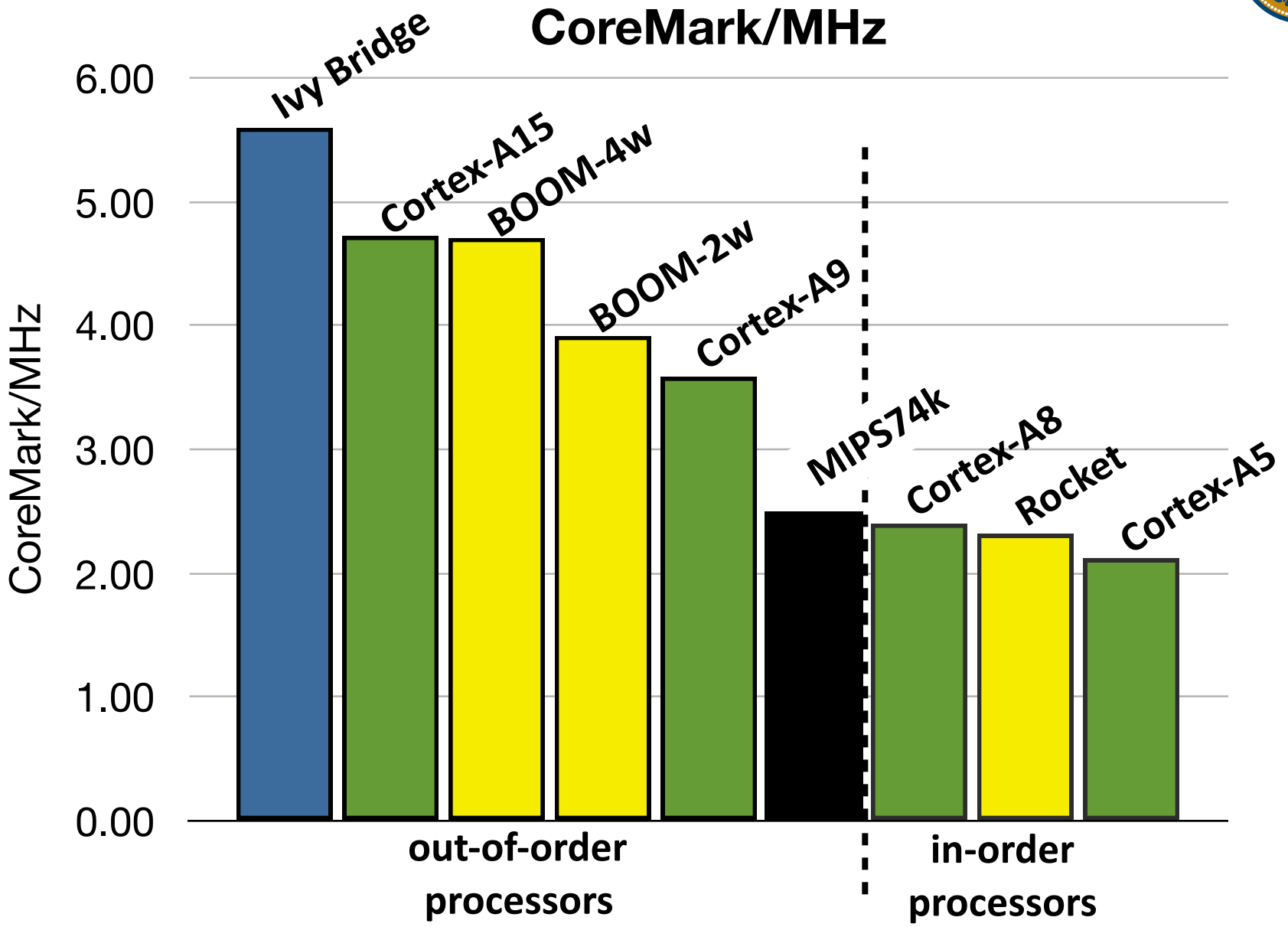




- Issue Unit
- RRd Stage (bypasses)
- ROB
- Br Predictor
- BusyTable
- FPU
- Other
- Rename Stage (maptables)
- Register File
- LSU
- Freelist
- FetchBuffer
- Imul

preliminary results

Industry Comparisons



preliminary results

Industry Comparisons

Processor	Core Area	CoreMark/ MHz	Freq (MHz)	IPC
Intel Xeon E5 2668 (Ivy)	~12 mm^2@22nm	5.6	3,300	1.96
ARM Cortex-A15	2.8 mm^2@28nm	4.72	2,116	1.5
BOOM-4wide	1.1 mm^2@45nm	4.7	1,000	1.5
BOOM-2wide	0.8 mm^2@45nm	3.91	1,500	1.26
ARM Cortex-A9	2.5 mm^2@40nm	3.59	1,400	1.27
MIPS 74K	2.5 mm^2@65nm	2.5	1,600	-
Rocket (RV64G)	0.5 mm^2@45nm	2.32	1,500	0.76
ARM Cortex-A5	0.5 mm^2@40nm	2.13	-	-

preliminary results

Industry Comparisons

Processor	Core Area	CoreMark/ MHz	Freq (MHz)	IPC
Intel Xeon E5 2668 (Ivy)	~12 mm^2@22nm	5.6	3,300	1.96
ARM Cortex-A15	2.8 mm^2@28nm	4.72	2,116	1.5
BOOM-4wide	1.1 mm^2@45nm	4.7	1,000	1.5
BOOM-2wide	0.8 mm^2@45nm	3.91	1,500	1.26
ARM Cortex-A9	2.5 mm^2@40nm	3.59	1,400	1.27
MIPS 74K	2.5 mm^2@65nm	2.5	1,600	-
Rocket (RV64G)	0.5 mm^2@45nm	2.32	1,500	0.76
ARM Cortex-A5	0.5 mm^2@40nm	2.13	-	-

preliminary results

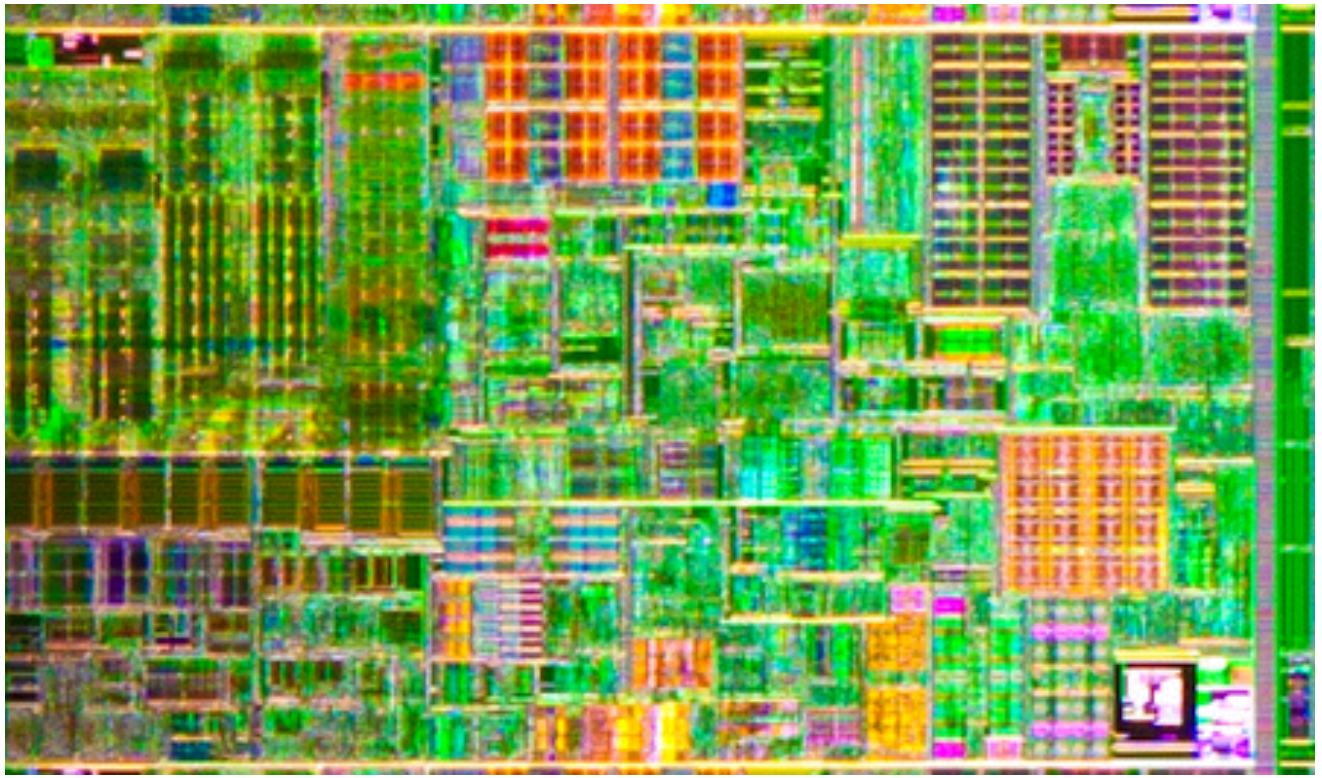
Industry Comparisons

Processor	Core Area	CoreMark/ MHz	Freq (MHz)	IPC
Intel Xeon E5 2668 (Ivy)	~12 mm^2@22nm	5.6	3,300	1.96
ARM Cortex-A15	2.8 mm^2@28nm	4.72	2,116	1.5
BOOM-4wide	1.1 mm^2@45nm	4.7	1,000	1.5
BOOM-2wide	0.8 mm^2@45nm	3.91	1,500	1.26
ARM Cortex-A9	2.5 mm^2@40nm	3.59	1,400	1.27
MIPS 74K	2.5 mm^2@65nm	2.5	1,600	-
Rocket (RV64G)	0.5 mm^2@45nm	2.32	1,500	0.76
ARM Cortex-A5	0.5 mm^2@40nm	2.13	-	-

48x

preliminary results

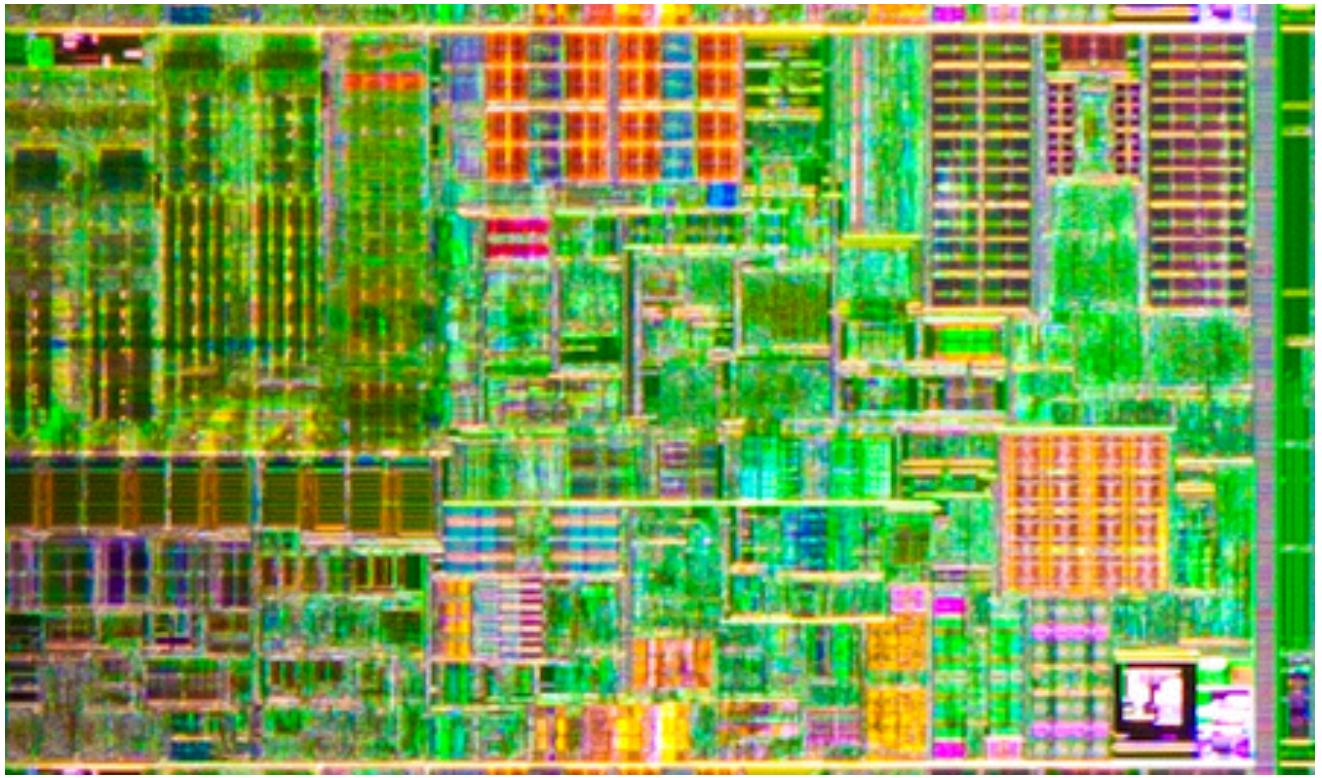
Ivy Bridge Tile Comparison

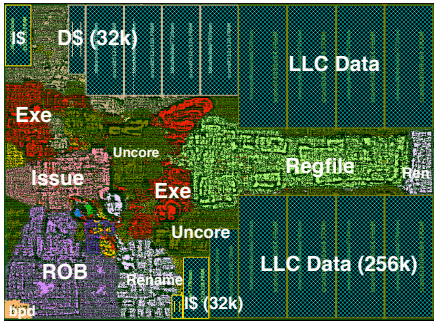


Ivy Bridge-EP Tile
(32kB/32kB + 256kB caches)
~12nm @ 22nm

preliminary results

Ivy Bridge Tile Comparison



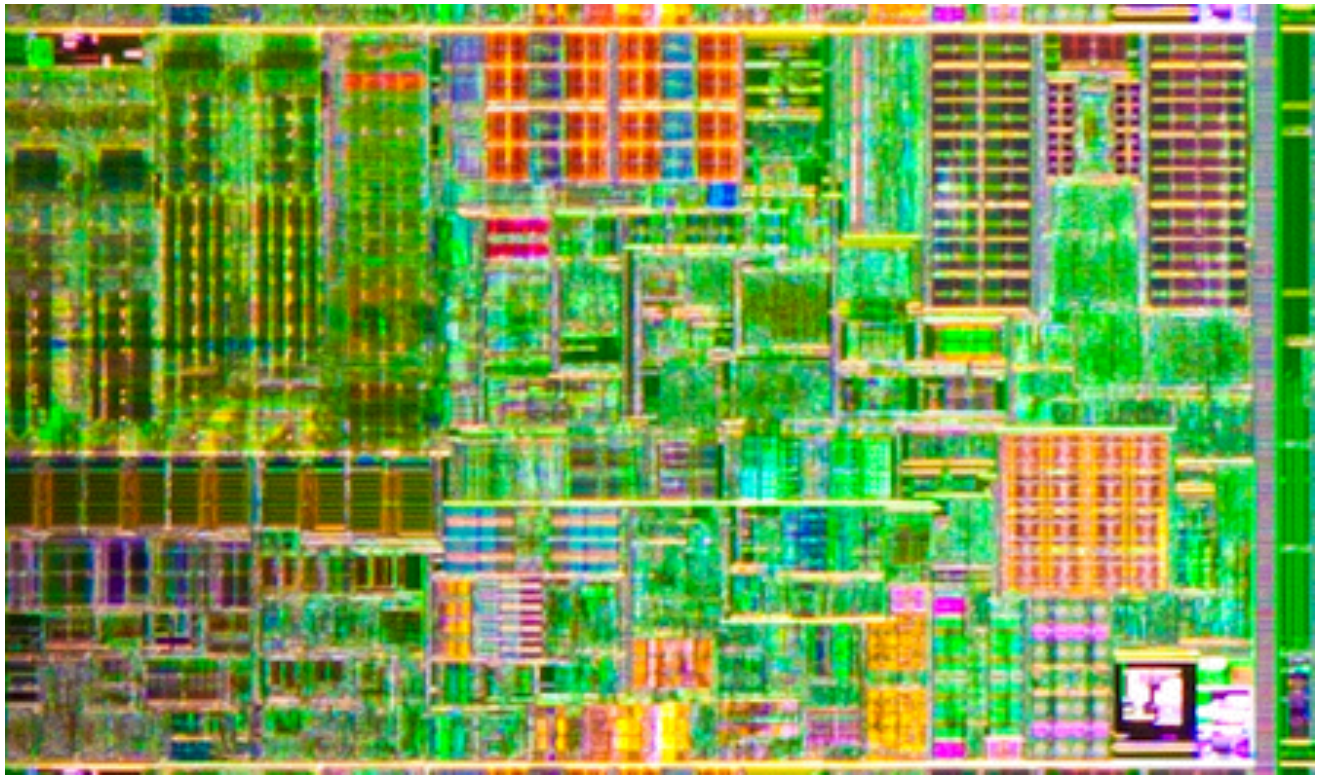


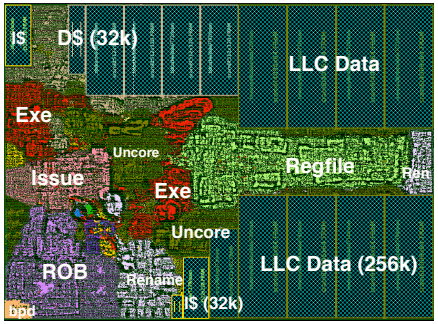
BOOM-2w Chip
(32kb/32kB + 256kB caches)
1.7mm² @ 45nm

Ivy Bridge-EP Tile
(32kB/32kB + 256kB caches)
~12nm @ 22nm

preliminary results

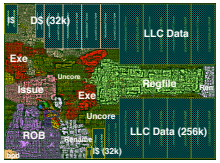
Ivy Bridge Tile Comparison





BOOM-2w Chip
(32kb/32kB + 256kB caches)
1.7mm² @ 45nm

Ivy Bridge-EP Tile
(32kB/32kB + 256kB caches)
~12nm @ 22nm



BOOM-2w Chip
scaled to **0.4mm² @ 22nm**
preliminary results