BROOM
An open-source out-of-order processor with resilient low-voltage operation in 28nm CMOS

Christopher Celio, Pi-Feng Chiu,
Krste Asanović, David Patterson, and Borivoje Nikolic
Hot Chips 2018
- Open-source superscalar out-of-order RISC-V core
- Resilient cache for low-voltage operation

BROOM Chip (Taped out Aug 2017)

TSMC 28 nm HPM
6 mm²
417k std cells
73 SRAM macros
1.0 GHz @ 0.9 V
Outline

- What is BROOM?
- The RISC-V BOOM Core
- Micro-architectural-level assist techniques
  - Line Disable (LD)
  - Line Recycle (LR)
  - Dynamic Column Redundancy (DCR)
  - Bit Bypass with SRAM (BB-S)
- The Agile Design Experience
- Chip Implementation
- Low Voltage Experimental Results
- Future Directions
What is BOOM?

- "Berkeley Out-of-Order Machine"
- out-of-order
- superscalar
- implements RV64G, boots Linux
- It is synthesizable
- it is open-source
- written in Chisel (16k loc)
- It is parameterizable generator
- built on top of Rocket-chip SoC Ecosystem

http://ucb-bar.github.io/riscv-boom
Timeline (the path to Hot Chips)

- 2011: TA
- 2012: TA
- 2013: Thesis

Chisel's first commit
- 2011: Verilog OOO engine
- 2011: Chisel's first commit
- 2012: RV64IM
- 2013: simple kernel
- 2013: super-scalar
- 2014: atomics
- 2014: FP
- 2014: VM
- 2015: boots
- 2015: Linux
- 2016: boots
- 2016: open-sourced
- 2016: TAGE predictor
- 2017: BROOM tapeout
- 2018: RISC-V 2.0
- 2010: bug fixing
- 2010: open-sourced

- 2011: toy BOOM
- 2012: 1-wide
- 2012: magic memory
- 2013: caches
- 2013: RV64IM
- 2014: RV64IM
- 2015: RV64IM
- 2016: RV64IM
- 2017: RV64IM
- 2018: RV64IM
The BOOM Core

Fetch (3 cycles) → Decode → Rename & Dispatch → Issue select → Register Read → Execute → Writeback

- I$ Buffer
- Branch predictor
- Fetch Buffer
- Decode
- Rename: tags
- Mem Issue Queue
- ALU Issue Queue
- FP Issue Queue
- Physical INT Register File (6R3W)
- Physical FP Register File (3R2W)

- SDQ
- SAQ
- LAQ
- ALU
- iDiv
- iMul
- To FP Regfile
- To Int Regfile
- F2I
- FD\text{v}
- FMA
Leveraging Open-source RTL

- The Rocket-chip SoC Generator
- Started in 2011
- Taped out 10 (13?) 17 times by Berkeley + many others
- 6,016 commits
- 64 contributors
- Commercial quality
- Replace standard in-order core with BOOM
- Leverage Rocket-chip as a library of processor components

https://github.com/freechipsproject/rocket-chip
# Core Comparison

<table>
<thead>
<tr>
<th>Processor</th>
<th>SiFive U54 Rocket (RV64GC)</th>
<th>Berkeley BOOMv2 (RV64G)</th>
<th>OpenSPARC T2</th>
<th>ARM Cortex-A9</th>
<th>Intel Xeon Ivy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Chisel</td>
<td>Chisel</td>
<td>Verilog</td>
<td>-</td>
<td>SystemVerilog</td>
</tr>
<tr>
<td>Core LoC</td>
<td>8,000</td>
<td>16,000</td>
<td>290,000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SoC LoC</td>
<td>34,000</td>
<td>50,000</td>
<td>1,300,000</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
## Core Comparison

<table>
<thead>
<tr>
<th>Processor</th>
<th>SiFive U54 Rocket (RV64GC)</th>
<th>Berkeley BOOMv2 (RV64G)</th>
<th>OpenSPARC T2</th>
<th>ARM Cortex-A9</th>
<th>Intel Xeon Ivy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Chisel</td>
<td>Chisel</td>
<td>Verilog</td>
<td>-</td>
<td>SystemVerilog</td>
</tr>
<tr>
<td>Core LoC</td>
<td>8,000</td>
<td>16,000</td>
<td>290,000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SoC LoC</td>
<td>34,000</td>
<td>50,000</td>
<td>1,300,000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Foundry</td>
<td>TSMC</td>
<td>TSMC</td>
<td>TI</td>
<td>TSMC</td>
<td>Intel</td>
</tr>
<tr>
<td>Technology</td>
<td>28 nm (HPC)</td>
<td>28 nm (HPM)</td>
<td>65 nm</td>
<td>40 nm (G)</td>
<td>22 nm</td>
</tr>
<tr>
<td>Core+L1 Area</td>
<td>0.54 mm²</td>
<td>0.52 mm²</td>
<td>~12 mm²</td>
<td>~2.5 mm²</td>
<td>~12 mm²</td>
</tr>
<tr>
<td>Coremark/MHz</td>
<td>2.75</td>
<td>3.77</td>
<td>1.64*</td>
<td>3.71</td>
<td>5.60</td>
</tr>
<tr>
<td>Frequency</td>
<td>1.5 GHz</td>
<td>1.0 GHz</td>
<td>1.17 GHz</td>
<td>1.4 GHz</td>
<td>3.3 GHz</td>
</tr>
</tbody>
</table>

*From eembc.org. 32 threads/8 cores achieve 13 Cm/MHz.
Case study: how agile is the BOOM generator?

<table>
<thead>
<tr>
<th></th>
<th>BOOMv1</th>
<th>BOOMv2</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTB entries</td>
<td>40 (fully-assoc)</td>
<td>64 x 4 (set-assoc)</td>
</tr>
<tr>
<td>Fetch Width</td>
<td>2 insts</td>
<td>2 insts</td>
</tr>
<tr>
<td>Issue Width</td>
<td>3 micro-ops</td>
<td>4 micro-ops</td>
</tr>
<tr>
<td>Issue Entries</td>
<td>20</td>
<td>16/20/10</td>
</tr>
<tr>
<td>Regfile</td>
<td>7r3w (unified)</td>
<td>6r3w (inst), 3r2w (fp)</td>
</tr>
<tr>
<td>Exe Units</td>
<td>iALU+iMul+FMA</td>
<td>iALU+iMul+iDiv</td>
</tr>
<tr>
<td></td>
<td>iALU+fDiv</td>
<td>iALU</td>
</tr>
<tr>
<td></td>
<td>Load/Store</td>
<td>FMA+fDiv</td>
</tr>
</tbody>
</table>

**BOOM v1 (April 2017)**

**BOOM v2 (Aug 2017)**
Frontend Design Changes

BOOMv1

- BTB in SRAM
  - set-associative
  - partially tagged
  - Checker to verify integrity

- BPD (Conditional Predictor)
  - hash gets entire stage
  - redirect based on BTB
  - redirect pushed back (I$)

BOOMv2
Building a Register File (the first P&R)

- BOOMv1 -- 7r3w with 110 registers (INT/FP)
- Initial Regfile design was infeasible for layout
- Critical paths in issue-select and register read
- Not DRC/LVS clean
Splitting the Regfile and Issue Queues

Fetch (3 cycles) — Decode — Rename & Dispatch — Issue select — Register Read — Execute — Writeback

I$ — Branch predictor — Fetch Buffer — Decode — Rename — Mem Issue Queue — Physical INT Register File (6R3W) — SDQ — To FP RF

BTB — ALU Issue Queue — Physical FP Register File (3R2W) — iDiv — iMul — To FP Regfile

ALU — FP Issue Queue — ALU — i2F — To Int Regfile

FMA —
Multi-port Register File for Design Exploration

**Transistor-level**

![Diagram of multi-port register file]

**Advantage**
- Compact area
- Higher performance

**Challenge**
- Long design cycle
- Difficult for architecture design exploration
Multi-port Register File for Design Exploration

**Transistor-level**

**Advantage**
- Compact area
- Higher performance

**Challenge**
- Long design cycle
- Difficult for architecture design exploration

**RTL**

**Advantage**
- Low design effort
- Rapid design exploration

**Challenge**
- Large area
- Bad performance
- Routing congestion

- 70x64 flip-flops
- 4480
- 7-level deep logic tree
- Congestion
Multi-port Register File for Design Exploration

**Transistor-level**

**Advantage**
- Compact area
- Higher performance

**Challenge**
- Long design cycle
- Difficult for architecture design exploration

**Gate-level**

**Advantage**
- Rapid design exploration
- Shared read wires solve routing congestion

**Challenge**
- Guided place-and-route for area/performance optimization

**RTL**

**Advantage**
- Low design effort
- Rapid design exploration

**Challenge**
- Large area
- Bad performance
- Routing congestion
Resilient Cache for Energy Efficiency

- Low-voltage operation improves energy efficiency
- SRAM-based cache fails at low voltages
- 50 mV reduction in VDD increases BER by 10x
- Architecture-level assist techniques can tolerate errors to reduce Vmin
- Only require RTL changes
Line Recycling (LR)

- Line Disable (LD) avoids errors by disabling the faulty cache line.
- Group three faulty lines with no error at the same column.
- Two patch lines (line P1/P2) used to repair the recycled line (line R).
- A majority vote of line R/P1/P2 corrects the data output.

**Example:** write data 1001 to the recycled group

<table>
<thead>
<tr>
<th></th>
<th>way0</th>
<th>way1</th>
<th>way2</th>
<th>way3</th>
</tr>
</thead>
<tbody>
<tr>
<td>set0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set1</td>
<td>⭐</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set3</td>
<td></td>
<td></td>
<td>⭐ ⭐</td>
<td></td>
</tr>
</tbody>
</table>

- ⭐: failing bit
- Store three identical copies

```
Line R 1101
Line P1 1011
Line P2 0001
```

**corrected output:**

```
1001
```
Dynamic Column Redundancy (DCR)

- DCR dynamically selects a different multiplexer shift to avoid the error according to the redundancy address (RA)
- Fix 1 bit per set
- Require LD/LR to handle multi-bit errors
Bit Bypass with SRAM (BB-S) for tag

- Expand tag arrays to store error entries
- Correct more error with less area
  - Fix 2 bits per row
  - An 8T SRAM bitcell is 6x smaller than a flip-flop
  - Shared decoder and peripheral circuit
- Area overhead: 8.6% in tag arrays
Cache Resiliency Techniques

<table>
<thead>
<tr>
<th>Technique</th>
<th>Protected cache</th>
<th>Timing overhead</th>
<th>Area overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>L2 data</td>
<td>Small §</td>
<td>0.77%</td>
</tr>
<tr>
<td>LD</td>
<td>L1/2 data</td>
<td>Small</td>
<td>0.2%</td>
</tr>
<tr>
<td>DCR</td>
<td>L1/2 data</td>
<td>Small</td>
<td>1.1% †</td>
</tr>
<tr>
<td>BB-S</td>
<td>L1/2 tag</td>
<td>Small</td>
<td>0.9% †</td>
</tr>
<tr>
<td>SECDED</td>
<td>L1/2</td>
<td>Large</td>
<td>10.9% ‡</td>
</tr>
</tbody>
</table>

§ Require 3 additional cycles
† Numbers are reported for L2
‡ 1 repair/64 bit

* Data portion is 86.2% of cache area, tag portion is 11% of cache area.
4 months of agile tape-out

*ignore the Y-axis:
-- too many parameters/variables changing between each run
-- doesn't capture DRC violations
Agile Hardware Development

- **RTL hacking can be very agile**
  - ~6 minutes to compile, build, and run "riscv-tests" regression suite (10 KHz for Verilator simulator)
  - Chisel allows for quick, far-reaching changes
  - generator approach allows for late-binding design decisions
  - small changes, improvements (that don't affect floor plan) are agile

- **Physical design is a bottleneck**
  - 2-3 hours for synthesis results
  - 8-24 hours for p&r results
  - RTL and PD are tightly coupled

- **Verification is a bottleneck**
  - I can write bugs faster than I can find them
Verification

- Directed tests and a randomized torture generator.
- Verilator/VCS/FPGA simulation at RTL.
- VCS for post-gl/par simulation.
- Speculative OOO pipelines are difficult to get good coverage on.
  - Need tests that build up a lot of speculative state.
  - Need tests that cover OS- and platform-level use-cases.
- Assertions are king.
DESSERT: Debugging RTL Effectively with State Snapshotting for Error Replays across Trillions of Cycles

- Co-simulate, find bugs, and get waveforms from Cloud FPGA-based simulation!
- Donggyu Kim, et. al. CARRV 2018
Incorrect Jump Target

- 401.bzip2 (assertion error at 500 billion cycles)
  - JAL jumps to wrong target.
  - Due to improper signed arithmetic.
  - 2-3 year old bug.
  - 3 hours of FPGA time.
Incorrect Jump Target

- 401.bzip2 (assertion error at 500 billion cycles)
  - JAL jumps to wrong target.
  - Due to improper signed arithmetic.
  - 2-3 year old bug.
  - 3 hours of FPGA time.
  - Would require 39 years of Verilator simulation to find.
  - DESSERT found this via a synthesized assertion.
## Chip Implementation

### Chip Summary

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ISA</strong></td>
<td>RISC-V RV64IMAFD with Sv39</td>
</tr>
<tr>
<td><strong>Fetch Width</strong></td>
<td>2 insts</td>
</tr>
<tr>
<td><strong>Issue Width</strong></td>
<td>3 micro-ops</td>
</tr>
<tr>
<td><strong>Issue Entries</strong></td>
<td>16 (i) 20 (m) 10 (f)</td>
</tr>
<tr>
<td><strong>Regfile</strong></td>
<td>6R3W (int), 3R2W (fp)</td>
</tr>
<tr>
<td><strong>Exe Units</strong></td>
<td>iALU+iMul+FMA, iALU+fDiv, Load/Store</td>
</tr>
<tr>
<td><strong>L1 I/D Cache</strong></td>
<td>4-way, 16KB</td>
</tr>
<tr>
<td><strong>L2 Cache</strong></td>
<td>8-way, 1MB</td>
</tr>
</tbody>
</table>

- **OoO core**
- **iregfile**
- **1MB L2 Cache**
- **Data array**
- **L2 tag, RIT, PAT**

- **BPD, BTB**

*Figure credits: UC Berkeley ASPIRE*
Experimental Setup

- Chip-on-board (COB) package
- Voltage and clock generation on the motherboard
- Cortex A9 on ZC706 works as the front-end server
- Boot Linux

Performance

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock frequency</td>
<td>1GHz @0.9V</td>
</tr>
<tr>
<td></td>
<td>320MHz @0.6V</td>
</tr>
<tr>
<td>Coremark/MHz</td>
<td>3.77</td>
</tr>
<tr>
<td>Instruction Per Cycle</td>
<td>1.11 (@Coremark)</td>
</tr>
</tbody>
</table>
Bit Error Rate and Vmin reduction

Bit Error Rate (BER) vs. VDD (a.u.)

- L2 tag
- L2 data
- L1 tag
- L1 data

LR 5%  LR 1%

Vmin reduction: 21.7%

43% energy reduction

*: estimated
With LR and 5% loss of L2 cache capacity, Vmin is reduced to 0.47V@70MHz
2.3% increase in L2 misses, but only 0.2% degradation in IPC
Future Directions

- Pi-Feng and Chris have graduated!
- BOOM will continue to be supported and improved.
  - [github.com/ucb-bar/riscv-boom](https://github.com/ucb-bar/riscv-boom)
Lots of opportunities for using the BOOM core
Lots of opportunities for using the BOOM core

- Agile Methodology Research
  - How to verify complex IP?
  - How do you measure/predict RTL performance, area, power?
Lots of opportunities for using the BOOM core

- Agile Methodology Research
  - How to verify complex IP?
  - How do you measure/predict RTL performance, area, power?
- Software Studies
  - Hardware/software co-design.
  - High visibility of very long-running applications.
Lots of opportunities for using the BOOM core

- Agile Methodology Research
  - How to verify complex IP?
  - How do you measure/predict RTL performance, area, power?
- Software Studies
  - Hardware/software co-design.
  - High visibility of very long-running applications.
- Security Research
  - New class of speculation-based attacks.
  - How to attack?
  - How to defend?
  - Evaluate cost of changes to branch predictors, caches, and more.
Lots of opportunities for using the BOOM core

- Agile Methodology Research
  - How to verify complex IP?
  - How do you measure/predict RTL performance, area, power?
- Software Studies
  - Hardware/software co-design.
  - High visibility of very long-running applications.
- Security Research
  - New class of speculation-based attacks.
  - How to attack?
  - How to defend?
  - Evaluate cost of changes to branch predictors, caches, and more.
- New RISC-V extensions
  - Variable-length vector.
  - Managed-language support.
FireSim now supports BOOM!

- FireSim is a open-source cycle-accurate FPGA-accelerated simulation tool that runs on Amazon EC2 F1
- Chisel RTL is automatically transformed into cycle-accurate FPGA simulator
- Peripheral device support:
  - UART, Disk, Ethernet NIC, easy to add more
- Boot Linux on a multi-core BOOM with 16 GB DDR3, UART, Ethernet NIC in the cloud for 50 cents/hour at ~100 MHz
- FireSim is available at:
  - [https://fires.im](https://fires.im)
- ISCA 2018 Paper:
  - [https://sagark.org/assets/pubs/firesim-isca2018.pdf](https://sagark.org/assets/pubs/firesim-isca2018.pdf)
A 2-person tapeout takes a village!

- RISC-V ISA
  - very out-of-order friendly!
- Chisel hardware construction language
  - object-oriented, functional programming
- FIRRTL
  - exposed RTL intermediate representation (IR)
- Rocket-chip
  - A full working SoC platform built around the Rocket in-order core
- Thanks to:
  - Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Henry Cook, Palmer Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, Jack Koenig, Jim Lawson, Yunsup Lee, Richard Lin, Eric Love, Martin Maas, Chick Markley, Albert Magyar, Howard Mao, Miquel Moreto, Quan Nguyen, Albert Ou, Brian Richards, Colin Schmidt, Wenyu Tang, Stephen Twigg, Huy Vo, Andrew Waterman, Angie Wang, Jerry Zhao, and more...
Thank you!

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.