Z-scale:
Tiny 32-bit RISC-V Systems
With Updates to the Rocket Chip Generator

Yunsup Lee, Albert Ou, Albert Magyar
2nd RISC-V Workshop
yunsup@eecs.berkeley.edu
6/30/2015
What is Z-scale? Why?

- Z-scale is a tiny 32-bit RISC-V core generator suited for microcontrollers and embedded systems
- Over the past months, external users have expressed great interest in small RISC-V cores
  - So we have listened to your feedback!
- Z-scale is designed to talk to AHB-Lite buses
  - plug-compatible with ARM Cortex-M series
- Z-scale generator also generates the interconnect between core and devices
  - Includes buses, slave muxes, and crossbars
Berkeley's RISC-V Core Generators

- **Z-scale**: Family of Tiny Cores
  - Similar in spirit to ARM Cortex M0/M0+/M3/M4
  - Integrates with AHB-Lite interconnect

- **Rocket**: Family of In-order Cores
  - Currently 64-bit single-issue only
  - Plans to work on dual-issue, 32-bit options
  - Similar in spirit to ARM Cortex A5/A7/A53
  - Will integrate with AXI4 interconnect

- **BOOM**: Family of Out-of-Order Cores
  - Supports 64-bit single-, dual-, quad-issue
  - Similar in spirit to ARM Cortex A9/A15/A57
  - Will integrate with AXI4 interconnect
  - BOOM talk right after this one
Z-scale Pipeline

- 32-bit 3-stage single-issue in-order pipe
- Executes RV32IM ISA, has M/U privilege modes
- I-bus and D-bus are AHB-Lite and 32-bits wide
- Interrupts are supported
- Will publish a “microarchitecture specification”
## ARM Cortex-M0 vs. Z-scale

<table>
<thead>
<tr>
<th>Category</th>
<th>ARM Cortex-M0</th>
<th>RISC-V Zscale</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA</td>
<td>32-bit ARM v6</td>
<td>32-bit RISC-V (RV32IM)</td>
</tr>
<tr>
<td>Architecture</td>
<td>Single-Issue In-Order 3-stage</td>
<td>Single-Issue In-Order 3-stage</td>
</tr>
<tr>
<td>Performance</td>
<td>0.87 DMIPS/MHz</td>
<td>1.35 DMIPS/MHz</td>
</tr>
<tr>
<td>Process</td>
<td>TSMC 40LP</td>
<td>TSMC 40GPLUS</td>
</tr>
<tr>
<td>Area w/o Caches</td>
<td>0.0070 mm²</td>
<td>0.0098 mm²</td>
</tr>
<tr>
<td>Area Efficiency</td>
<td>124 DMIPS/MHz/mm²</td>
<td>138 DMIPS/MHz/mm²</td>
</tr>
<tr>
<td>Frequency</td>
<td>≤50 MHz</td>
<td>~500 MHz</td>
</tr>
<tr>
<td>Voltage (RTV)</td>
<td>1.1 V</td>
<td>0.99 V</td>
</tr>
<tr>
<td>Dynamic Power</td>
<td>5.1 μW/MHz</td>
<td>1.8 μW/MHz</td>
</tr>
</tbody>
</table>

- Note: numbers are very likely to change in the future as we tune the design and add things to the core.
RV32E

- New base integer instruction set
  - Reduced version of RV32I designed for embedded systems
- Cut number of integer registers to 16
- Remove counters that are mandatory in RV32I
  - Counter instructions (rdcycle[h], rdtime[h], rdinstret[h]) are not mandatory
Building a Z-scale System

- Working on “platform specification”
Z-scale Generator is Written in Chisel

- Chisel is a new HDL embedded in Scala
  - Rely on good software engineering practices such as abstraction, reuse, object oriented programming, functional programming
  - Build hardware like software
- 604 Unique LOC written in Chisel
  - Control: 274 lines
  - Datapath: 267 lines (99 lines could be generalized)
  - Top-level: 63 lines
- 983 LOC in Chisel borrowed from Rocket
- Reuse and parameterization in Chisel and Rocket chip actually works!
(1, 2, 3) `map { n => n+1 }`
  – (2, 3, 4)
(1, 2, 3) `map { _+1 }`
(1, 2, 3) `zip (a, b, c)`
  – ((1, a), (2, b), (3, c))
(1, a)._1
  – 1
((1, a), (2, b), (3, c)) `map { _._1}`
((1, a), (2, b), (3, c)) `map { n => n._1}`
  – (1, 2, 3)
(0 until 3) `for each { println(_) }`
  – 0/1/2
class AHBXbar(n: Int, amap: Seq[UInt=>Bool]) extends Module {
    val io = new Bundle {
        val masters = Vec.fill(n){new AHBMasterIO}.flip
        val slaves = Vec.fill(amap.size){new AHBSlaveIO}.flip
    }

    val buses = io.masters map { m => Module(new AHBBus(amat)).io }
    val muxes = io.slaves map { s => Module(new AHBSlaveMux(n)).io }

    (buses.map(_.master) zip io.masters) foreach { case (b, m) => b <> m }
    (0 until n) foreach { m => (0 until amap.size) foreach { s =>
        buses(m).slaves(s) <> muxes(s).ins(m) }
    }

    (io.slaves zip muxes.map(_.out)) foreach { case (s, x) => s <> x }
}
Z-scale in Verilog

- Talked to many external users, and perhaps the #1 reason why they can’t use our stuff is because it’s written in Chisel
  - So we have listened to your feedback!
- We have implemented the same Z-scale core in Verilog
- 1215 LOC
- No more excuses for adoption!
  - If there still is any reason why you can’t use RISC-V, please do let us know
Z-scale FPGA DEMO System

JTAG Debugger
Z-Scale Core
AHB-Lite Crossbar

J-Bus
I-Bus
D-Bus

Boot ROM
DRAM
SPI FLASH
AHB APB

S-Bus

P-Bus
AHB-Lite
APB

CORE RESET (1KB)
GPIO LED (1KB)
Empty
SPI FLASH (16KB)
DRAM (64MB)
Empty
Boot ROM (16KB)

0x8000_0000
0x8000_0400
0x8000_0800
0x2000_0000
0x2000_0400
0x2000_0800
0x2400_0000
0x2400_0400
0x2400_0800
0x0000_0000
0x0000_0400
0x0000_0800
Z-scale FPGA DEMO System Mapped to Xilinx Spartan6 LX9

- **Avnet LX9 Microboard**
  - $89
  - Xilinx Spartan6 LX9
  - 64MB LPDDR RAM
  - 16MB SPI FLASH
  - 10/100 Ethernet
  - USB-to-UART
  - USB-to-JTAG
  - 2x Pmod headers
  - 4x LEDs
  - 4x DIP switches
  - RESET/PROG buttons

- **4 boards for raffle!**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Used</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>2,329</td>
<td>20%</td>
</tr>
<tr>
<td>LUTs</td>
<td>4,328</td>
<td>75%</td>
</tr>
<tr>
<td>RAM16</td>
<td>8</td>
<td>25%</td>
</tr>
<tr>
<td>RAM8</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

Test program is stored in bootrom. It is a memory test program, which writes 32-bit words generated from an LFSR to 64MB of DRAM, and checks it by reading 64MB of data, and toggles LED if it succeeds.
Z-scale Use Cases

- **Microcontrollers**
  - Implement your simple control loops
  - If code density matters

- **Embedded Systems**
  - Build your system around Z-scale

- **Validation of Tiny 32-bit RISC-V Systems**
  - You don’t need to use our code, just consider Z-scale as an existence proof and implement your own RV32I core

- **Both Chisel and Verilog versions of Z-scale is open-sourced under the BSD license**
  - [https://github.com/ucb-bar/zscale](https://github.com/ucb-bar/zscale)
  - [https://github.com/ucb-bar/fpga-spartan6](https://github.com/ucb-bar/fpga-spartan6)
What is the Rocket Chip Generator?

- Parameterized SoC generator written in Chisel
- Generates n Tiles
  - (Rocket) Core
  - RoCC Accelerator
  - L1 I$
  - L1 D$
- Generates Uncore
  - L1 Crossbar
  - Coherence Manager
  - Shared L2$ with directory bits
  - Exports a simple memory interface
Rocket Chip Generator Updates
Since the 1st RISC-V Workshop

- Implemented L2$ with directory bits
- RoCC coprocessor has a memory port directly into the L2$
- Main development will happen on the rocket-chip repository
- Moving towards standardized memory interfaces
Important Memory Interfaces

- **TileLink**
  - Our cache-coherent interconnect
  - For more details, watch my talk from last workshop
- **NASTI (pronounced nasty)**
  - Not A STandard Interface
  - Our implementation of the AXI4 standard
- **HASTI (pronounced hasty)**
  - Highly Advanced System Transport Interface
  - Our implementation of the AHB-Lite standard
- **POCI (pronounced pokey)**
  - Peripheral Oriented Connection Interface
  - Our implementation of the APB standard
Conclusion, Future Work, and Raffle

- Z-scale is a RISC-V tiny core generator suited for microcontrollers and embedded systems
- Z-scale
  - Microarchitecture document will be released first
  - Improve performance
  - Implement “C” extension as an option
  - Add MMU option to boot Linux
  - More devices on the LX9 board to come
- Rocket Chip Generator
  - JTAG debug interface (get rid of HTIF)
  - Move to standardized interfaces (NASTI/HASTI/POCI)
  - Add Z-scale option
- Raffle time!