Building Custom SoCs with RocketChip

Howard Mao
What is RocketChip?

- A library of reusable SoC components
  - Memory protocol converters
  - Arbiters and Crossbar generators
  - Clock-crossings and asynchronous queues
- A highly parameterizable SoC generator
  - Replace default Rocket core w/ your own core
  - Add your own coprocessor
  - Add your own SoC IP to uncore
- The largest open source CHISEL codebase
  - Scala allows advanced generator features
Can Generate a Variety of Systems

SiFive Freedom E310

UCB Hurricane-2
Used in Many a Tapeout
Accelarating Research

- Hwacha vector accelerator - Yunsup Lee, Colin Schmidt
- BOOM Out-of-Order Processor - Chris Celio
- Copy Accelerator - Howard Mao
- Garbage Collection Accelerator - Martin Maas
- CRAFT Agile Hardware Design - Stevo Bailey, Paul Rigge
- FireSim WSC Simulator - Sagar Karandikar, Howard Mao
- Page Fault Accelerator - Emmanuel Amaro, Nathan Pemberton
- HLS Accelerator Generator - Jenny Huang
Did We Mention It’s Open Source?
Project-Template

- A starting point for your RocketChip-based project.
- Includes a submodule for RocketChip, an example top-level, and some useful Makefiles
- Fork it and add your own Chisel IP
- https://github.com/ucb-bar/project-template
The Base System

- Rocket 64-bit in-order core
  - Single+Double precision FPU
  - Compressed instructions
  - L1 caches
- TileLink memory buses
- Programmable interrupt controller
- Real time clock
- Serial Interface
- AXI4 interface to DRAM
Possible Addons/Customizations

- RoCC coprocessor (e.g. Hwacha)
- TileLink MMIO peripherals
- DMA devices
- Additional top-level IO ports
- Custom CPU
Possible Addons/Customizations

- RoCC coprocessor (e.g. Hwacha)
- TileLink MMIO peripherals
- DMA devices
- Additional top-level IO ports
- Custom CPU
RoCC Accelerator Interface
RoCC Custom Instruction

<table>
<thead>
<tr>
<th></th>
<th>funct</th>
<th>rs2</th>
<th>rs1</th>
<th>xd</th>
<th>xs1</th>
<th>xs2</th>
<th>rd</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Diagram shows the structure and values of the RoCC Custom Instruction.
Example: Memcpy() Accelerator
Adding Accelerator to the Config

class WithDma extends Config((site, here, up) => {
    case DmaKey => DmaConfig()
    case RocketTilesKey => up(RocketTilesKey, site) map { r =>
        r.copy(rocc = Seq(
            RoCCParams(
                opcodes = OpcodeSet.custom2,
                generator = (p: Parameters) => LazyModule(new CopyAccelerator()((p)),
                nPTWPorts = 1))
        )
    }
})

class DmaConfig extends Config(new WithDma ++ new DefaultExampleConfig)
Possible Addons/Customizations

- RoCC coprocessor (e.g. Hwacha)
- TileLink MMIO peripherals
- DMA devices
- Additional top-level IO ports
- Custom CPU
Example: Network Interface Controller
Adding the NIC to your System

```scala
trait HasPeripheryIceNIC extends HasSystemBus {
  private val address = BigInt(0x10016000)
  val icenic = LazyModule(new IceNIC(address, sbus.beatBytes))
  icenic.mmionode := sbus.toVariableWidthSlaves
  sbus.fromSyncPorts() :=* icenic.dmanode
  ibus.fromSync := icenic.intnode
}

trait HasPeripheryIceNICModuleImp extends LazyModuleImp {
  val outer: HasPeripheryIceNIC
  val net = IO(new NICIO)
  net <> outer.icenic.module.io.ext
}

class ExampleTopWithIceNIC(implicit p: Parameters) extends ExampleTop
  with HasPeripheryIceNIC {
    override lazy val module = new ExampleTopWithIceNICModule(this)
}

class ExampleTopWithIceNICModule(outer: ExampleTopWithIceNIC)
  extends ExampleTopModule(outer)
  with HasPeripheryIceNICModuleImp
```
Possible Addons/Customizations

- RoCC coprocessor (e.g. Hwacha)
- TileLink MMIO peripherals
- DMA devices
- Additional top-level IO ports
- Custom CPU
Rocket vs. Boom

BOOMv2-2f4i
Replacing Rocket w/ BOOM

class DefaultBoomConfig extends Config((site, here, up) => {
  // Top-Level
  case BuildCore => (p: Parameters, e: freechips.rocketchip.tilelink.TLEdgeOut) => new BoomCore()(p, e)
  case XLen => 64

  // Rocket/Core Parameters
  case RocketTilesKey => up(RocketTilesKey, site) map { r => r.copy(
    core = r.core.copy(
      fWidth = 2,
      useCompressed = false,
      nPerfCounters = 29,
      nPMPs = 0,
      fpu = Some(freechips.rocketchip.tile.FPUParams(sfmaLatency=4, dfmaLatency=4, divSqrt=true)),
      btb = Some(BTBParams(nEntries = 0, updatesOutOfOrder = true)),
      dcache = Some(DCacheParams(rowBits = site(SystemBusKey).beatBits, nSets=64, nWays=8, nMSHRs=4, nTLBEntries=8)),
      icache = Some(ICacheParams(fetchBytes = 2*4, rowBits = site(SystemBusKey).beatBits, nSets=64, nWays=8))
    )
  )
  case BoomKey => BoomCoreParams(...)
})

class BoomConfig extends Config(new DefaultBoomConfig ++ new DefaultExampleConfig)
But don’t just take our word for it
Celerity SoC

Experiences Using the RISC-V Ecosystem to Design an Accelerator-Centric SoC in TSMC 16nm (CARRV ‘17)
Velour SoC

A Low Voltage RISC-V Heterogeneous System (CARRV ’17)
Memory Region Protector

Building Hardware Components for Memory Protection of Applications on a Tiny Processor
What will you make with RocketChip?
Credits

- Celerity SoC - Tutu Ajayi (University of Michigan), Khalid Al-Hawaj (Cornell University), Aporva Amarnath (University of Michigan), Steve Dai (Cornell University), Scott Davidson (University of California, San Diego), Paul Gao (University of California, San Diego), Gai Liu (Cornell University), Anuj Rao (University of California, San Diego), Austin Rovinski (University of Michigan), Ningxiao Sun (University of California, San Diego), Christopher Torng (Cornell University), Luis Vega (University of Washington), Bandhav Veluri (University of California, San Diego), Shaolin Xie (University of California, San Diego), Chun Zhao (University of California, San Diego), Ritchie Zhao (Cornell University), Christopher Batten (Cornell University), Ronald Dreslinski (University of Michigan), Rajesh Gupta (University of California, San Diego), Michael Taylor (University of Washington) and Zhiru Zhang (Cornell University)

- Velour SoC - Schuyler Eldridge (IBM T. J. Watson Research), Karthik Swaminathan (IBM T. J. Watson Research), Nandhini Chandramoorthy (IBM T. J. Watson Research), Alper Buyuktosunoglu (IBM T. J. Watson Research), Alec Roelke (University of Virginia), Xinfei Guo (University of Virginia), Vaibhav Verma (University of Virginia), Rajiv Joshi (IBM T. J. Watson Research), Mircea Stan (University of Virginia) and Pradip Bose (IBM T. J. Watson Research)

- Memory Region Protector - Hyunyoung Oh (Seoul National University), Yongje Lee (Seoul National University), Junmo Park (Seoul National University), Myonghoon Yang (Seoul National University) and Yunheung Paek (Seoul National University)