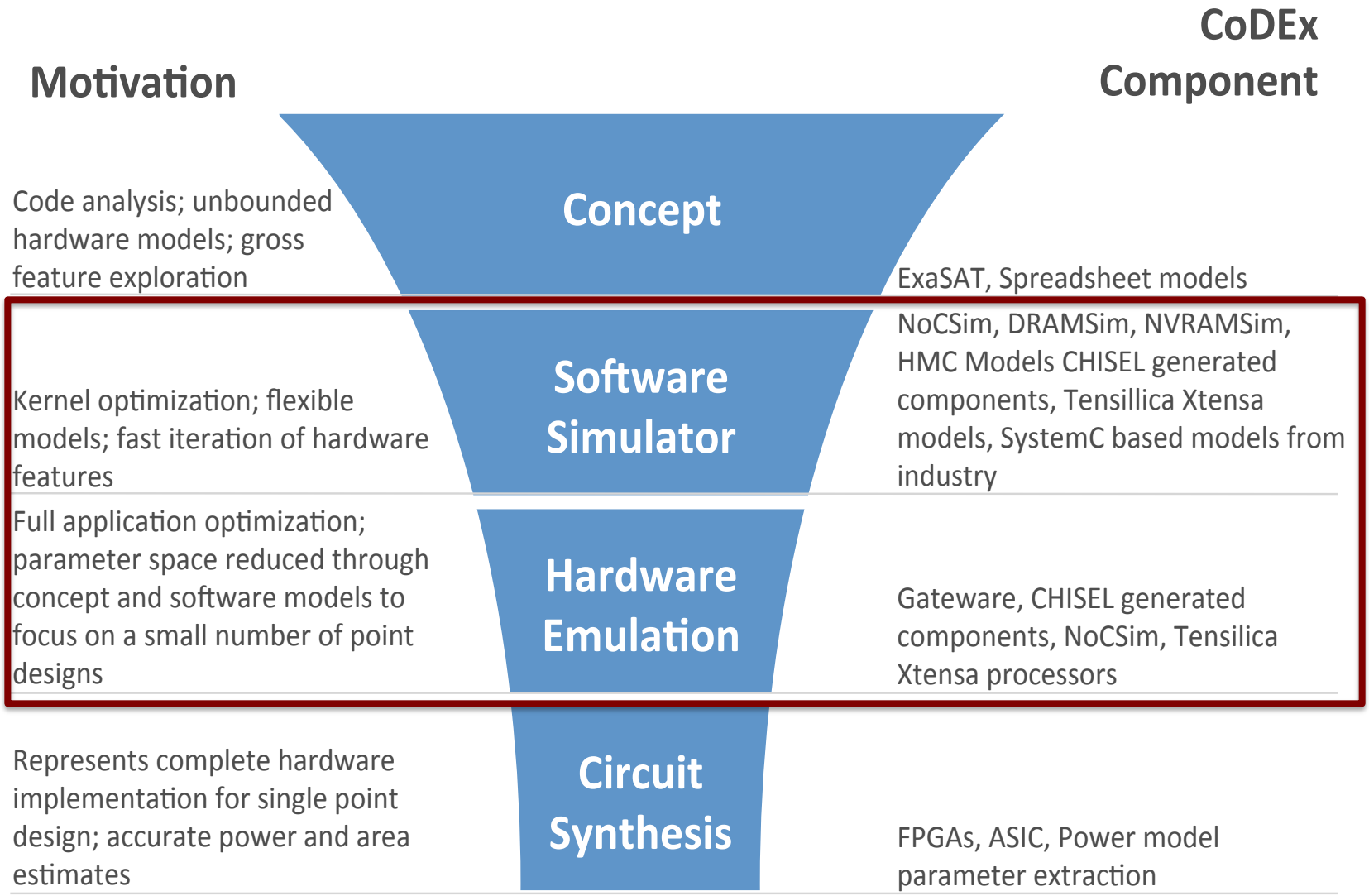
Emulating Future HPC SoC Architectures

Farzad Fatollahi-Fard, Dave Donofrio, John Shalf
Lawrence Berkeley National Lab

SC15 Emerging Technologies
November 17-19, 2015 – Austin, TX

What's Our Motivation?

Co-Design for Exascale (CoDEx)



Building an SoC from IP Logic Blocks

It's Legos with a some extra integration and verification cost

Processor Core (ARM, Tensilica, RISC-V)
With extra "options" like DP FPU, ECC

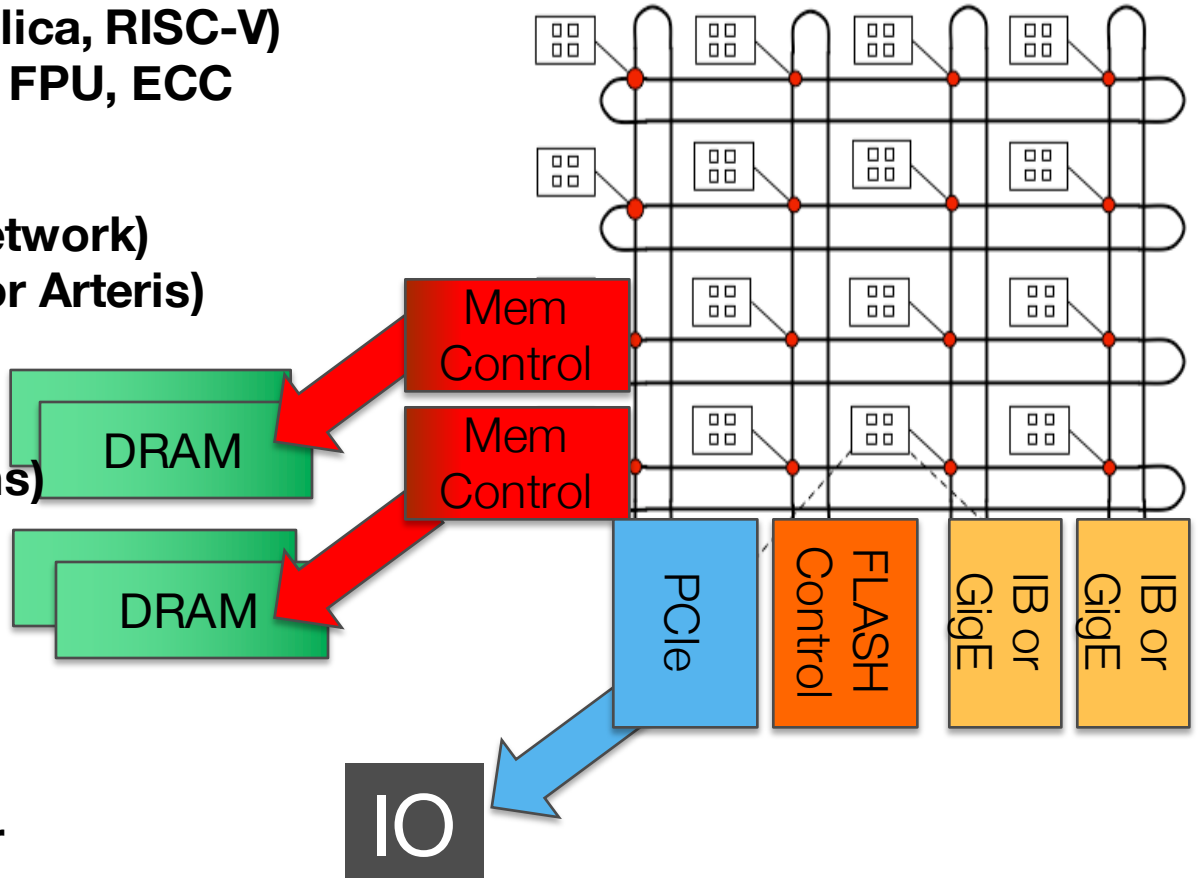
OpenSoC Fabric (on-chip network)
(currently proprietary ARM or Arteris)

DDR memory controller
(Denali/Cadence, SiCreations)
+ Phy & Programmable PLL

PCIe Gen3 Root complex

Integrated FLASH Controller

10GigE or IB DDR 4x Channel



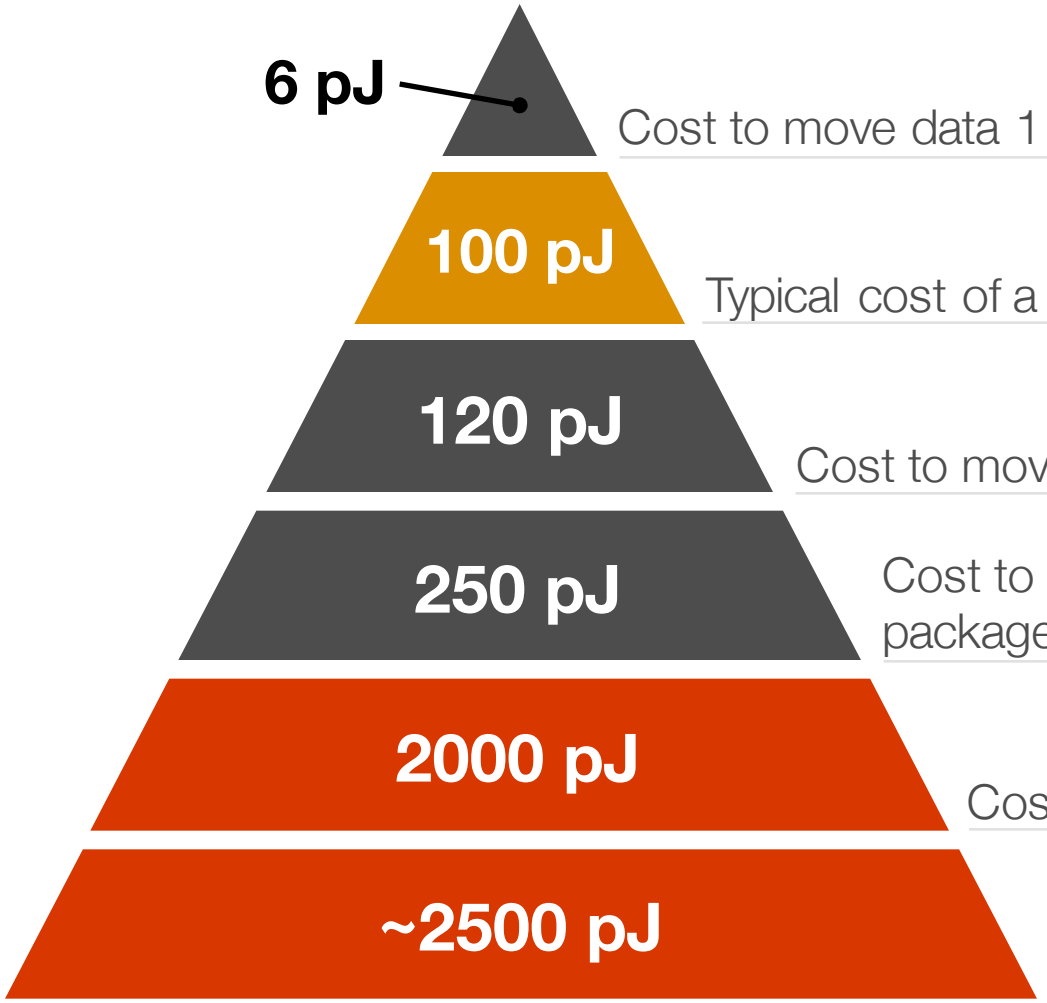
Parallelism increasing

NERSC Trends

	Franklin	Hopper	Edison	Cori (NERSC 8)
Core Count	4	24	48 (logical)	>60
Clock Rate	2.3GHz	2.1GHz	2.4 GHz	~1.5GHz
Memory	8GB	32GB	64GB	64-128GB +On package
Peak Perf	0.352 PF	1.288 PF	2.57 PF	> 3 TF

Hierarchical Power Costs

Data Movement is the Dominant Power Cost



Cost to move data 1 mm on-chip

Typical cost of a single floating point operation

Cost to move data 20 mm on chip

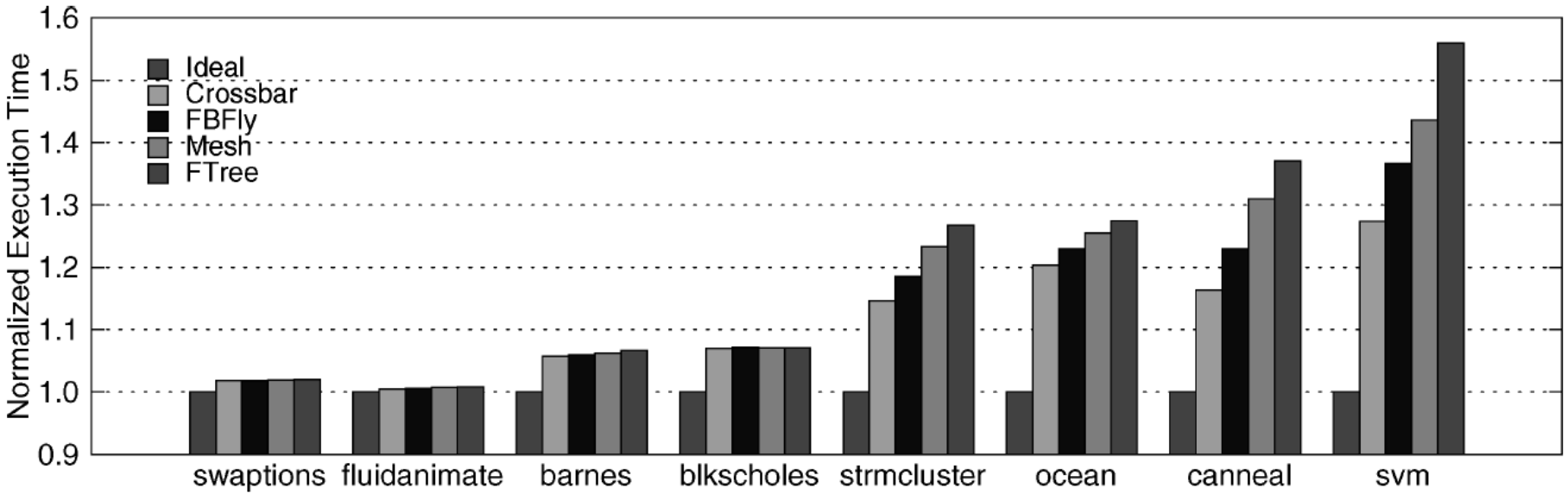
Cost to move off-chip, but stay within the package (SMP)

Cost to move data off chip into DRAM

Cost to move data off chip to a neighboring node

Other Parameters Impact Performance

For Example, Topology…



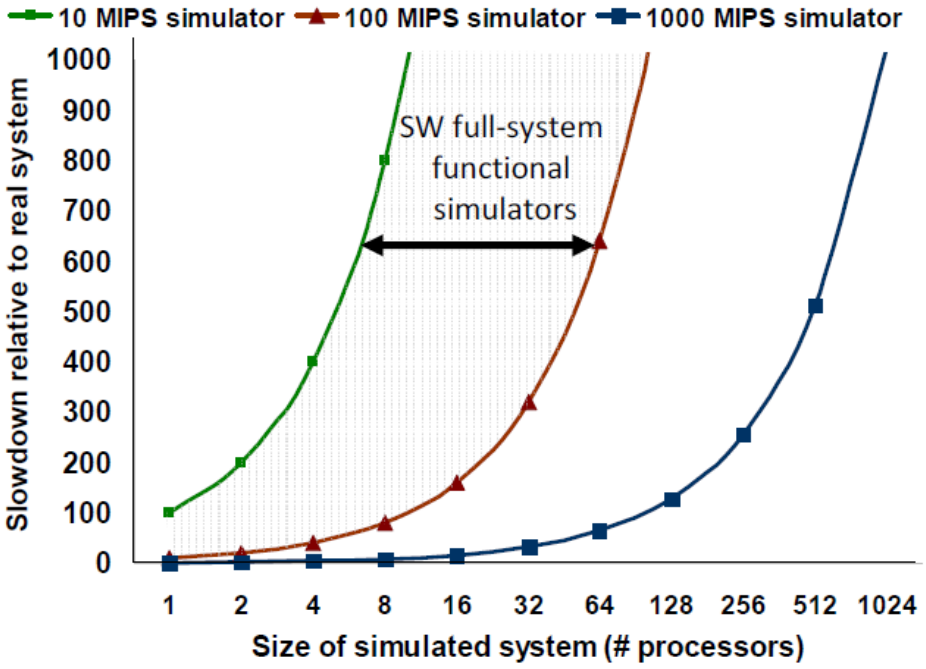
An analysis of on-chip interconnection networks for large-scale chip multiprocessors
ACM Transactions on computer architecture and code optimization (TACO), April 2010

- **Network topology can greatly influence application performance**

What tools exist for NoC research

What Tools Do We Have to Evaluate Large, Complex Networks of Cores?

- **Software models**
  - Fast to create, but plagued by long runtimes as system size increases
- **Hardware emulation**
  - Fast, accurate evaluate that scales with system size but suffers from long development time

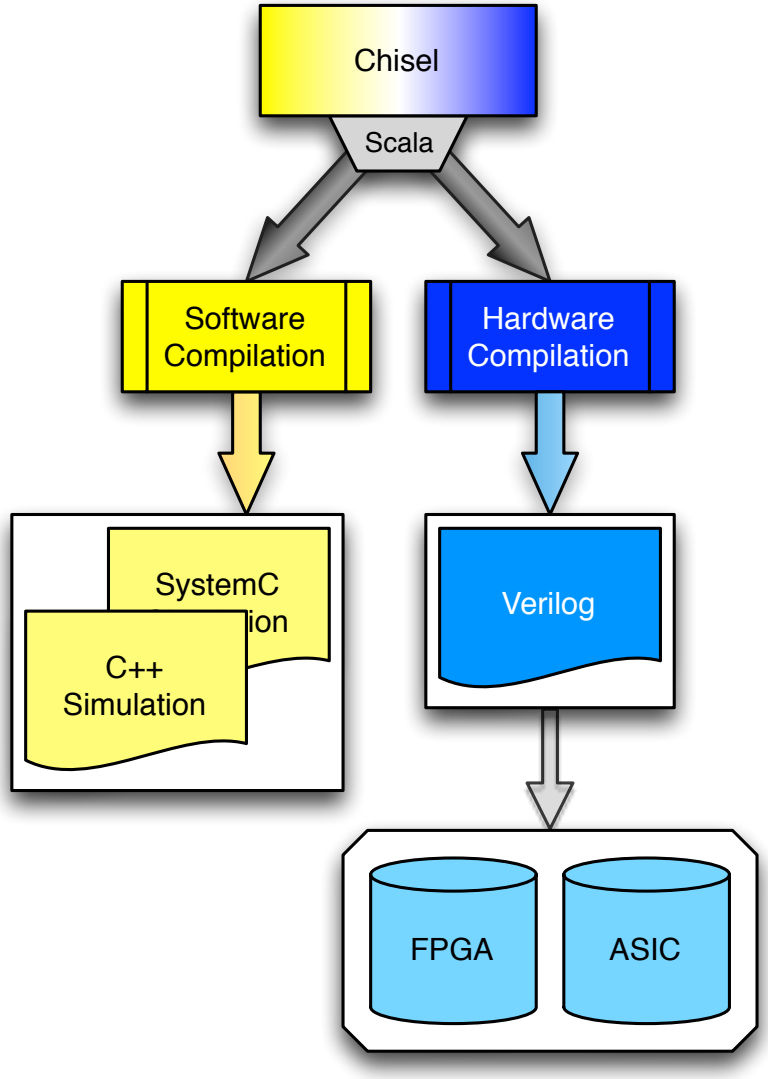


A complexity-effective architecture for accelerating full-system multiprocessor simulations using FPGAs. FPGA 2008

Chisel: A New Hardware DSL

Constructing Hardware In a Scala Embedded Language

- **Chisel provides both software and hardware models from the same codebase**
- **Object-oriented hardware development**
  - Allows definition of structs and other high-level constructs
- **Powerful libraries and components ready to use**
- **Working processors fabricated using Chisel**

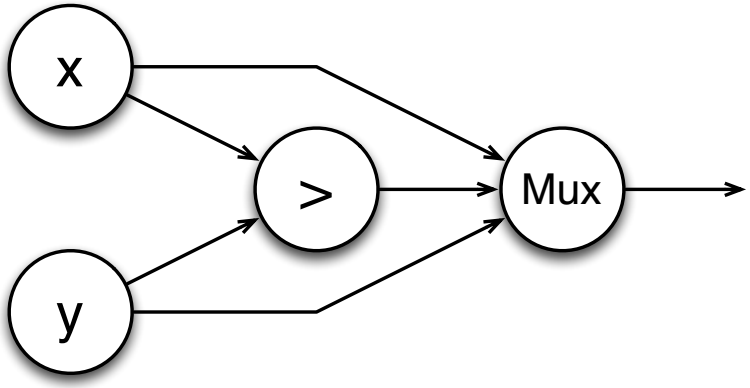


Chisel Overview

How Does Chisel Work?

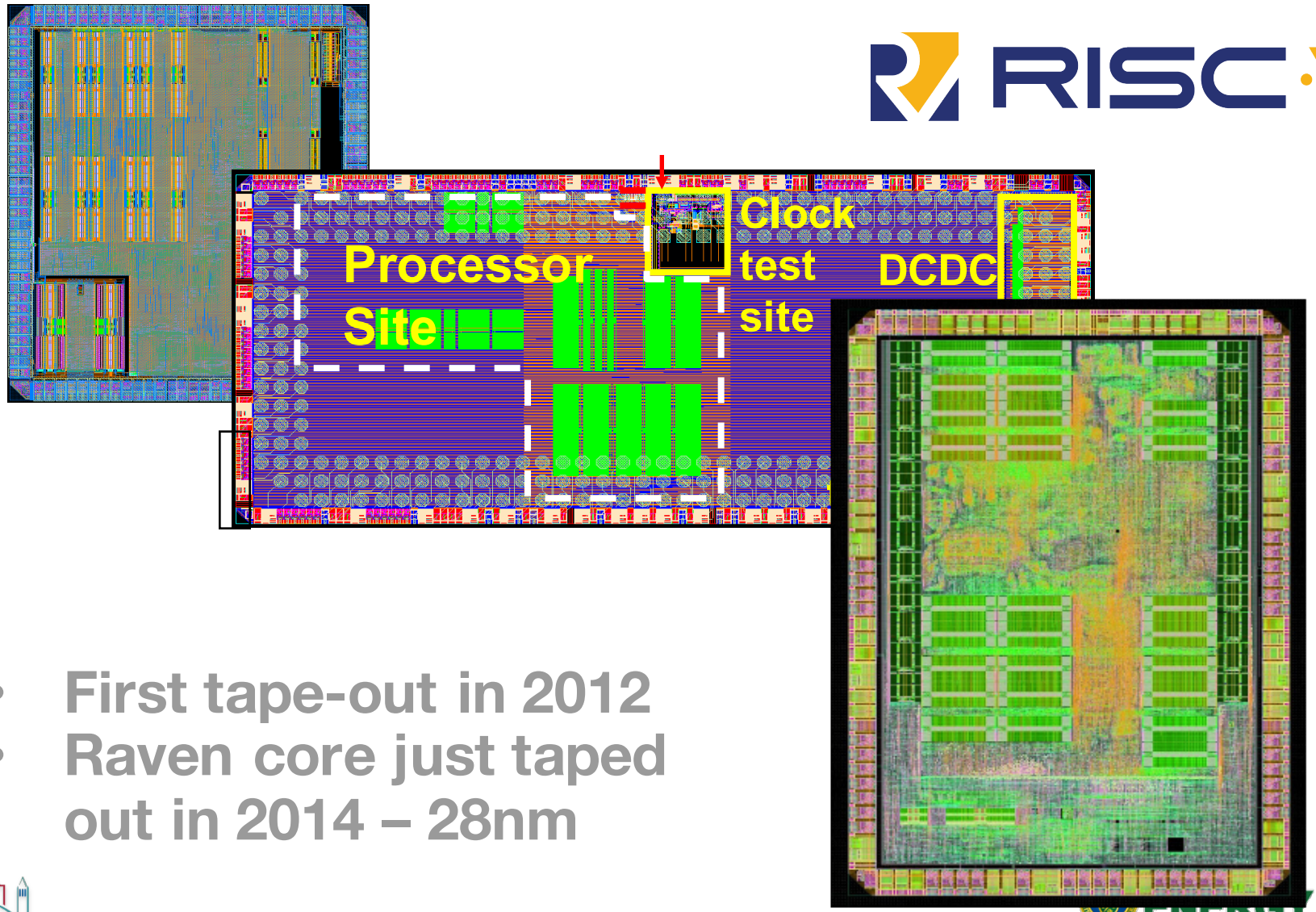
- **Not "Scala to Gates"**
- **Describe hardware functionality**
- **Chisel creates graph representation**
  - Flattened
- **Each node translated to Verilog or C++**

```
Mux(x > y, x, y)
```



Recent Chisel Designs

Chisel Code Successfully Boots Linux



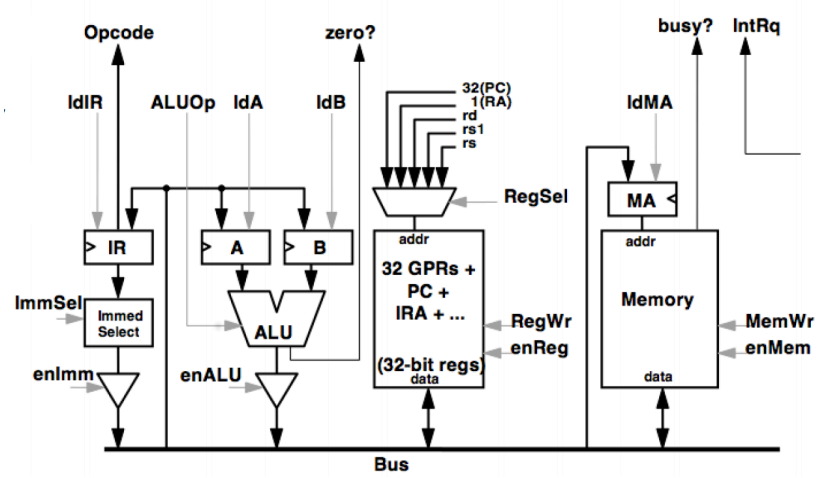
- First tape-out in 2012
- Raven core just taped out in 2014 – 28nm

RISC-V

A Completely *Open* ISA That Is *Free!*

- A completely *open* ISA that is freely available to academia and industry
- A *real* ISA suitable for direct native hardware implementation, not just simulation or binary translation
- 32-bit, 64-bit, and 128-bit address space variants for applications, operating system kernels, and hardware implementations.

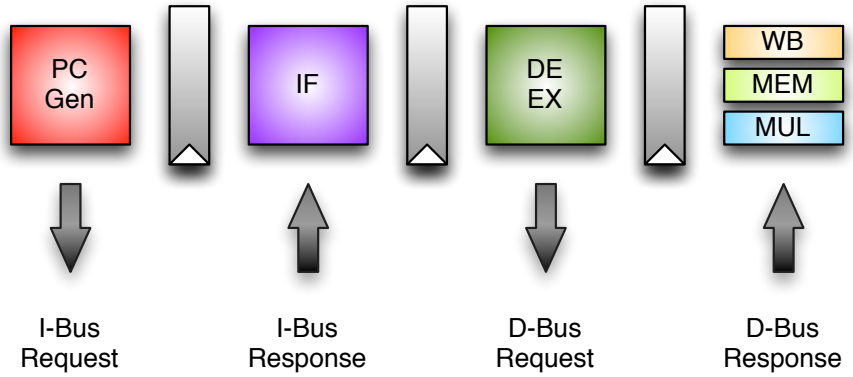
ASPIRE
UC BERKELEY



Z-Scale

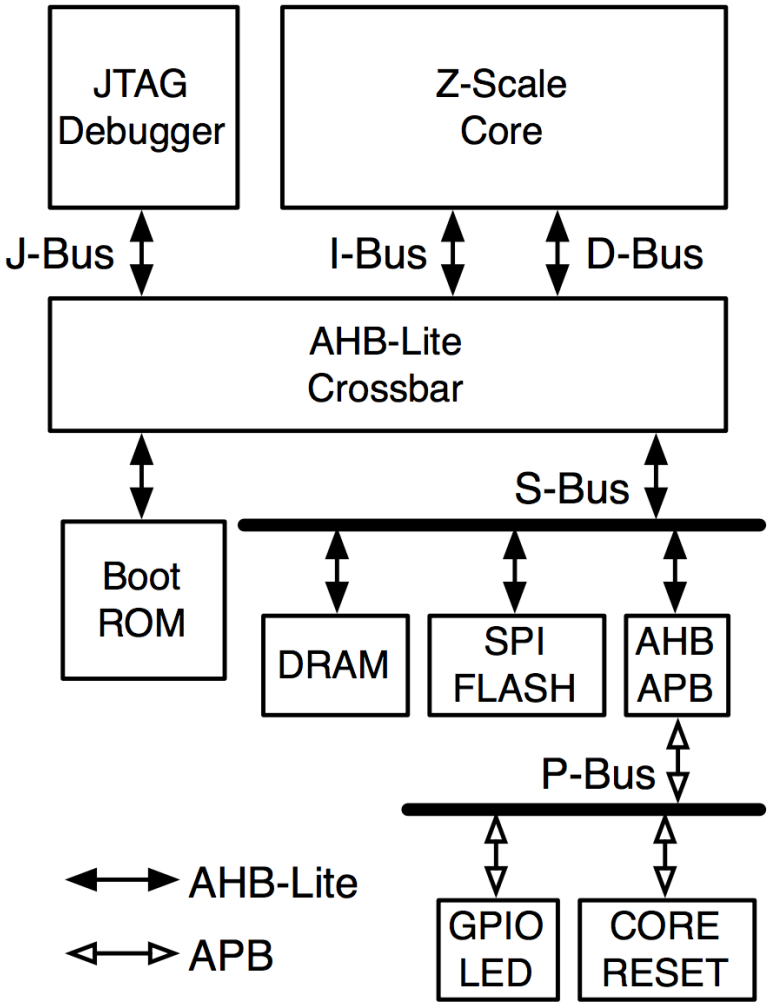
Tiny 32-bit RISC-V System

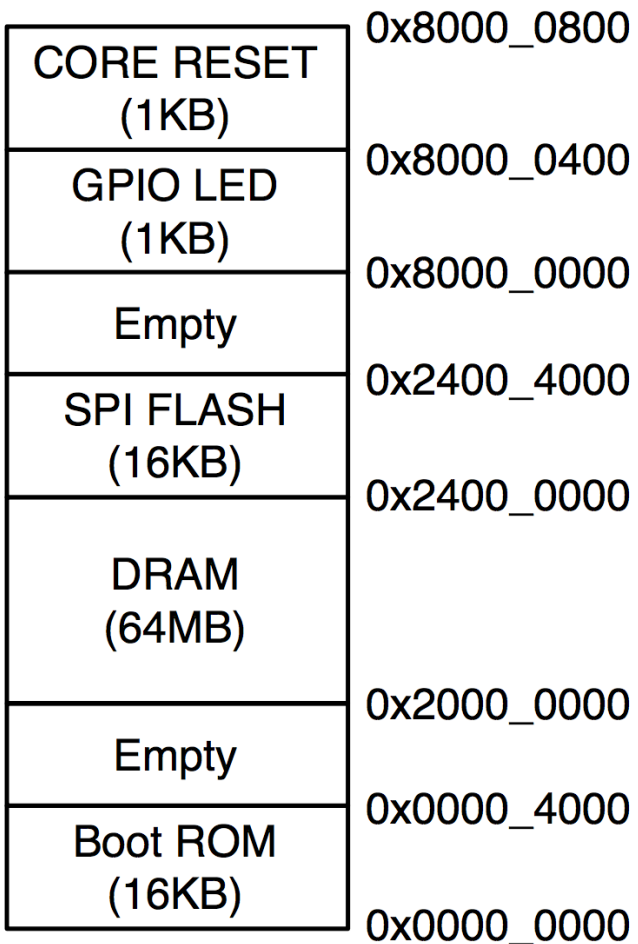
- **A tiny 32-bit 3-stage RISC-V core generator suited for microcontrollers and embedded systems**
- **Z-scale is designed to talk to AHB-Lite buses**
- **Z-scale generator also generates the interconnect between core and devices**
  - Includes buses, slave muxes, and crossbars



Z-Scale

Tiny 32-bit RISC-V System

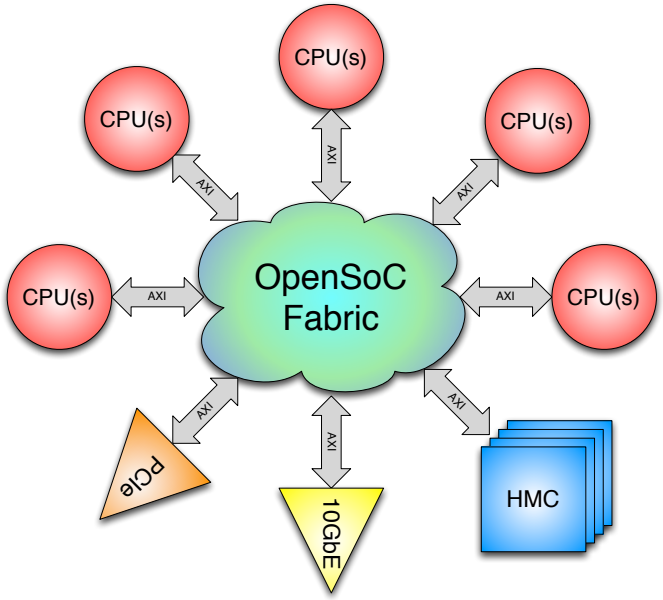




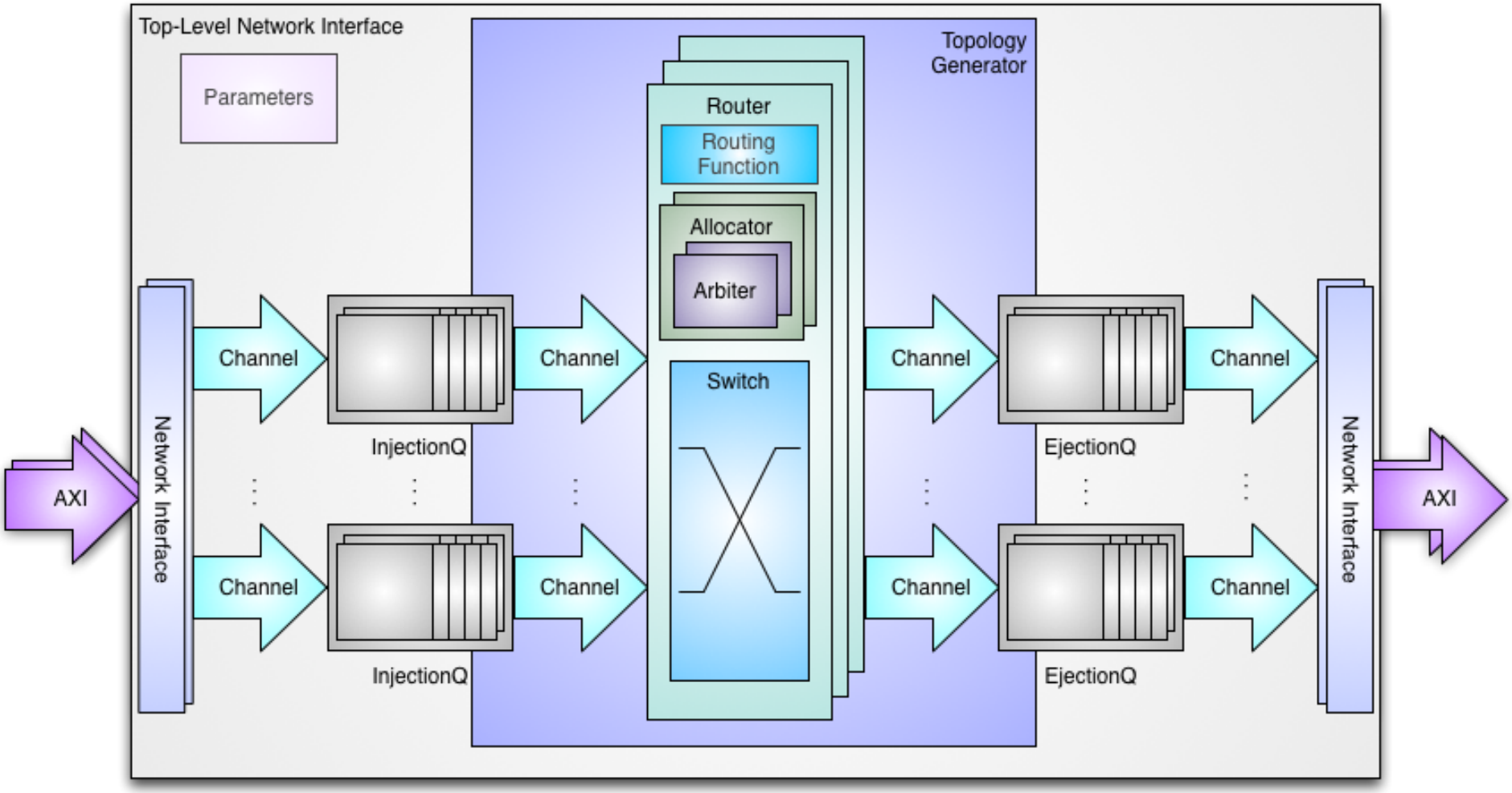
OpenSoC Fabric

An Open-Source, Flexible, Parameterized, NoC Generator

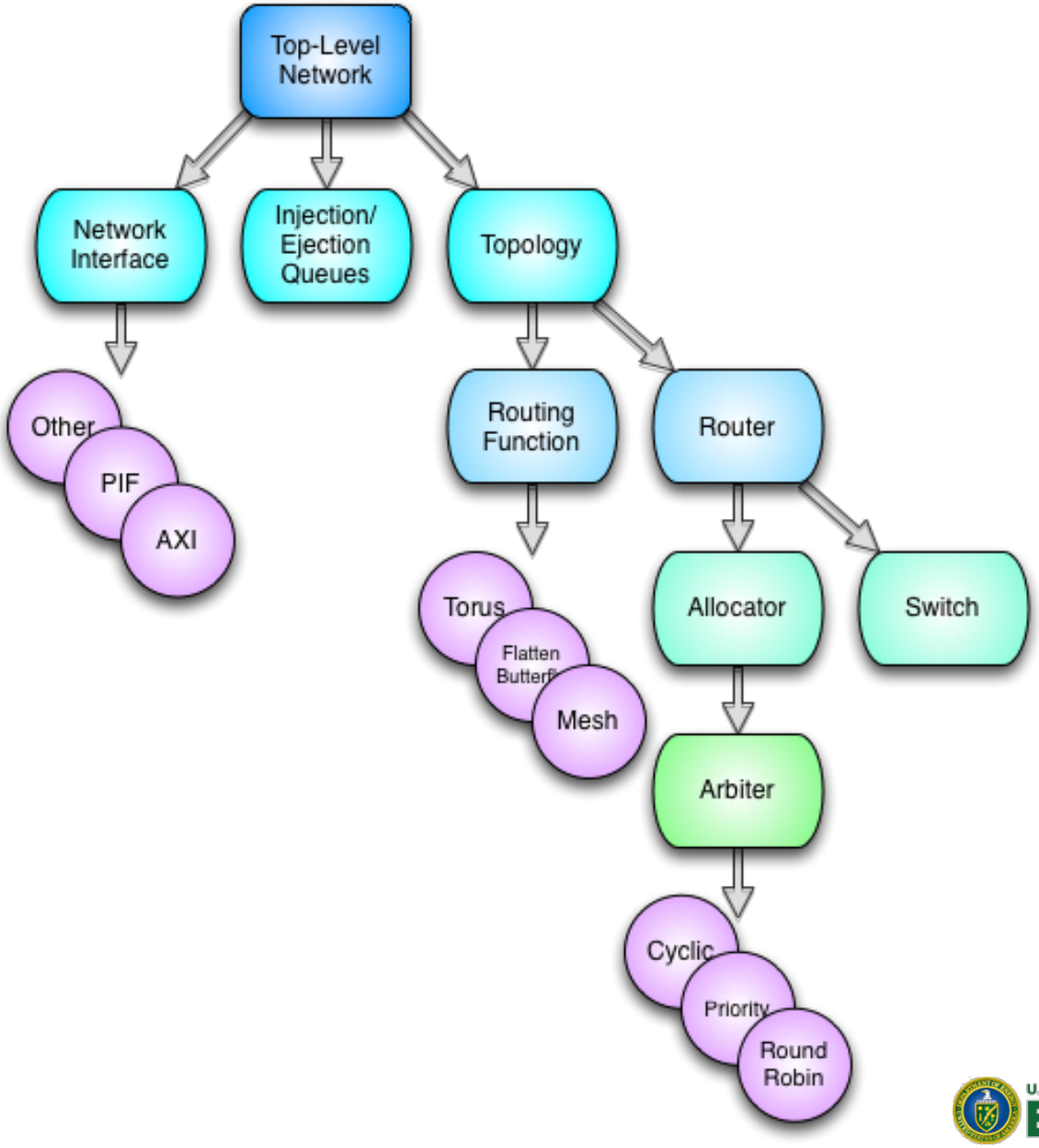
- **Part of the CoDEx tool suite**
- **Written in Chisel**
- **Dimensions, topology, VCs all configurable**
- **Fast functional C++ model for functional validation**
- **Verilog based description for FPGA or ASIC**
  - Synthesis path enables accurate power / energy modeling



Top Level Diagram



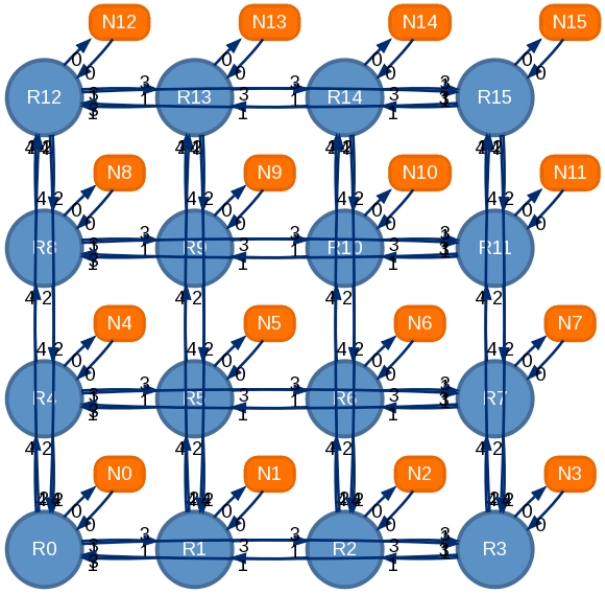
Functional Hierarchy



Top Level Modules

Topology

- Stiches routers together
- Assigns routers individual ID
- Assigns Routing Function to routers
- Connections Injection and Ejection Queues for network endpoints



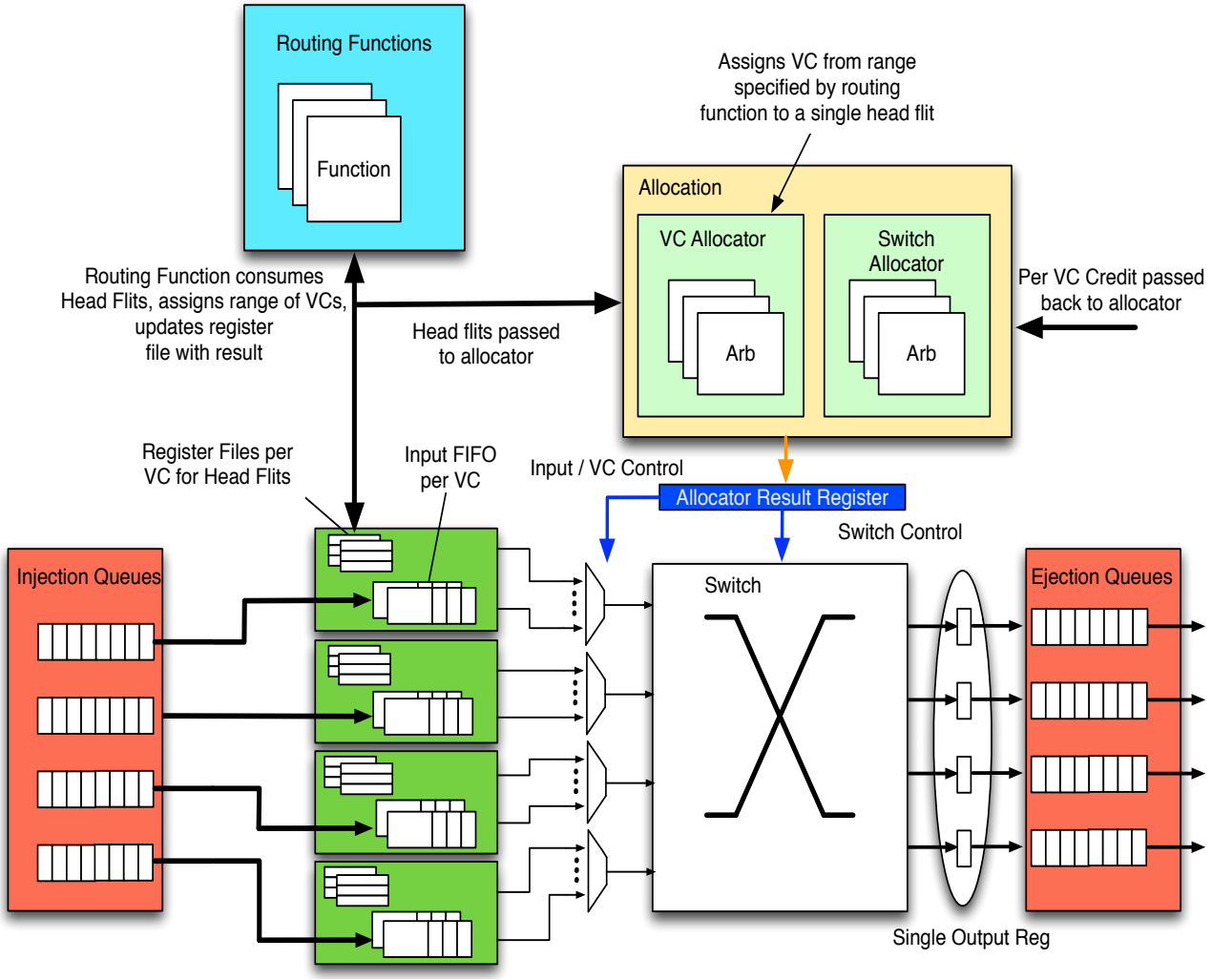
OpenSoC Top Level Modules

Router

- **Created and connected by Topology module**
- **Instantiates and connects:**
  - Routing Function
  - Allocators
  - Switch
- **Pipelined**
  - 3 stage pipeline for Wormhole
  - 4 stage pipeline for VCs
  - Includes state storage for each sub-module
- **Connects to Injection / Ejection Queues**

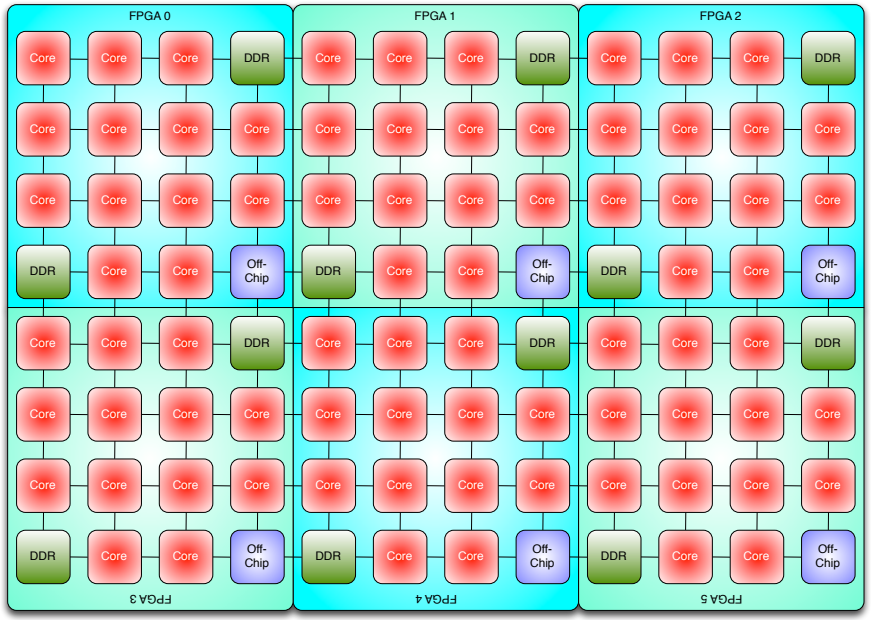
OpenSoC Top Level Modules

VC Router



The Demo

An SoC Design for HPC



- Z-Scale processors connected in a Concentrated Mesh
- 4 Z-scale processors
- 2x2 Concentrated mesh with 2 virtual channels

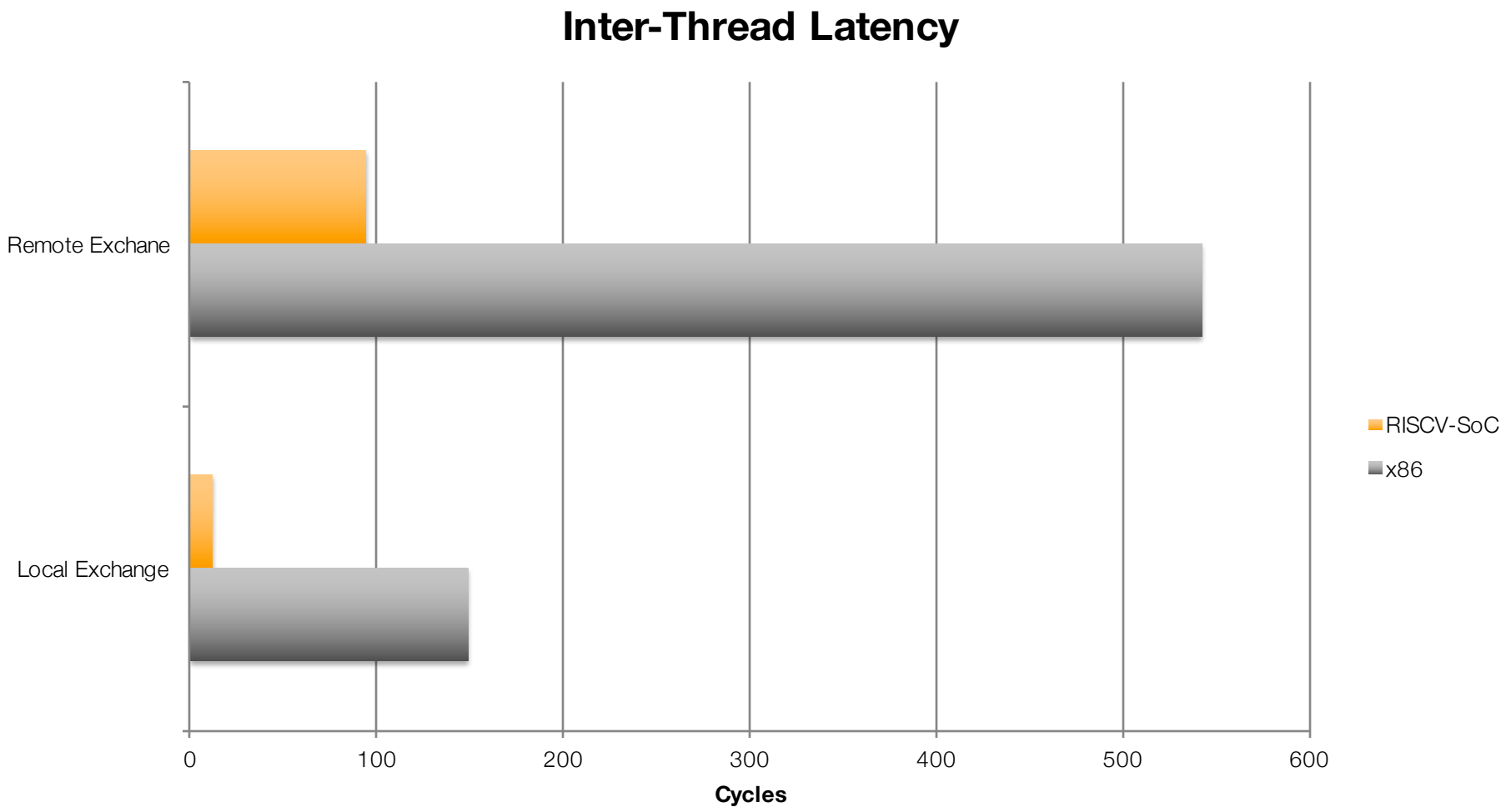
The Code

ZscaleChip.scala

```
class ZscaleTop_withOpenSoC_2x2 extends Module {
  val K = Vector(2,2)
    val numProcessors = K.product

    val io = new Bundle {
      val host  = Vec.fill(numProcessors) {new HTIFIO        }
      val dram  = Vec.fill(numProcessors) {new HASTISlaveIO().flip }
    val routersBusyPacked = UInt(OUTPUT, width=128)
      val fpgaID      = UInt(INPUT, width=32)
      val done        = Bool(OUTPUT)
    }

  var parms   = OpenSoC.Parameters.empty
    parms   = parms.child("MyOpenSoC_CMesh", Map(
                    ("TopologyDimension"->OpenSoC.Hard(2)),
                    ("RoutersPerDim"->OpenSoC.Hard(K)),
                    ("Concentration"->OpenSoC.Hard(1)),
                    ("numVCs"->OpenSoC.Hard(2)),
                    ("credThreshold"->OpenSoC.Hard(1)),

                    ("queueDepth"->OpenSoC.Soft(16)),

                    ("packetIDWidth"->OpenSoC.Hard(16)),
                    ("packetMaxLength"->OpenSoC.Hard(16)),
                    ("packetWidth"->OpenSoC.Hard(32)),

                    ("packetTypeWidth"->OpenSoC.Hard(4)),
                    ("destCordWidth"->OpenSoC.Hard(3)),
                    ("destCordDim"->OpenSoC.Hard(3)),
                    ("payloadWidth"->OpenSoC.Hard(32)),

                    ("flitIDWidth"->OpenSoC.Hard(4)),
                    ("payloadWidth"->OpenSoC.Hard(32)),
                                                ("breadCrumbCount"->OpenSoC.Soft(1)),
                    ("InputFlitizer"->OpenSoC.Soft((parms: OpenSoC.Parameters) => new OpenSoC.NetworkPacketToFlit(parms))),
                    ("OutputPacketizer"->OpenSoC.Soft((parms: OpenSoC.Parameters) => new OpenSoC.FlitToNetworkPacket(parms))),
                    ("Decoupled"->OpenSoC.Hard(true)),

                    ("numPriorityLevels"->OpenSoC.Hard(1))
                    ))

  var dmaParms  = OpenSoC.Parameters.empty
  val network  = Module(new OpenSoC.OpenSoC_CMesh[OpenSoC.NetworkPacket](parms, (p)=>new OpenSoC.NetworkPacket(p)))


  val doneSignals = Vec.fill(numProcessors){ Bool()}
  for (proc <- 0 until numProcessors){
    val sys     =  Module(new ZscaleSystem_Simple)
    val networkDMA  =  Module(new OpenSoC.DMA(parms.child( "myDMA", Map(
                ("DMAPortID"->OpenSoC.Hard(proc)),
                ("payloadWidth"->OpenSoC.Hard(32)),
                ("queueDepth"->OpenSoC.Soft(16))
              ))
            ))
    sys.io.host  <> io.host(proc)
    io.dram(proc)  <> sys.io.dram

    networkDMA.io.slave <> sys.io.dma
    network.io.ports(proc).in <> networkDMA.io.injPacket
    network.io.ports(proc).out <> networkDMA.io.ejcPacket

    networkDMA.io.fpgaID <> io.fpgaID
    doneSignals(proc) := networkDMA.io.done
  }

  io.done := orR(doneSignals.toBits)

  io.routersBusyPacked := network.io.cyclesRouterBusy.toBits
}
```

Line 185, Column 1 | Spaces: 2 | Scala

Results


Inter-Thread Latency
Remote Exchane
Local Exchange
0
100
200
300
400
500
600
Cycles
RISCV-SoC
x86

More Information

http://www.codexhpc.org