Chipyard Fundamentals

Jerry Zhao

eley

jzh@berkele

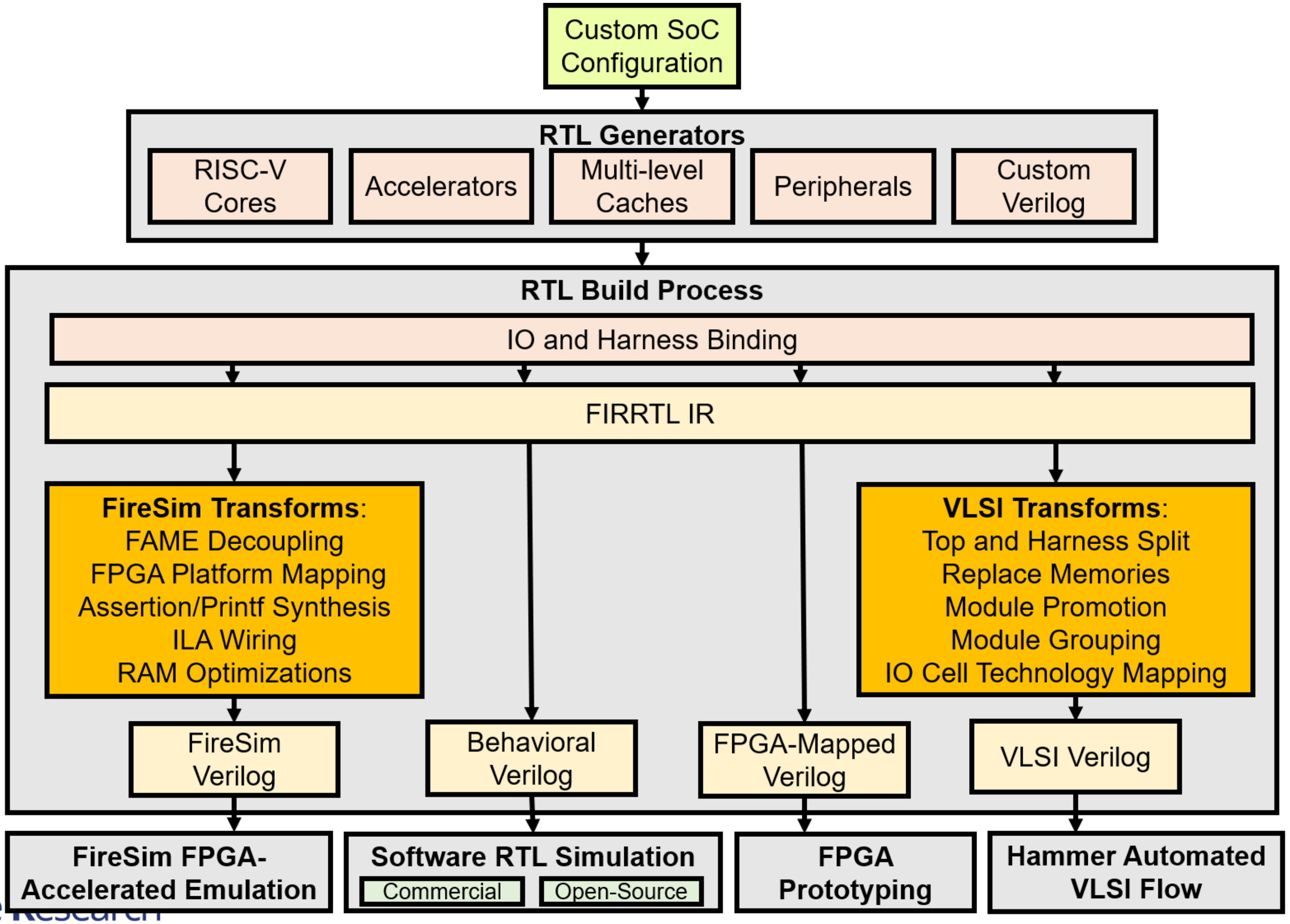
YA

Berkeley
Architecture
Research

Outline

- Chipyard Framework Architecture + Integration
- ML Tile
- GP Tile
- Memory subsystem

How is this integrated? Generators!

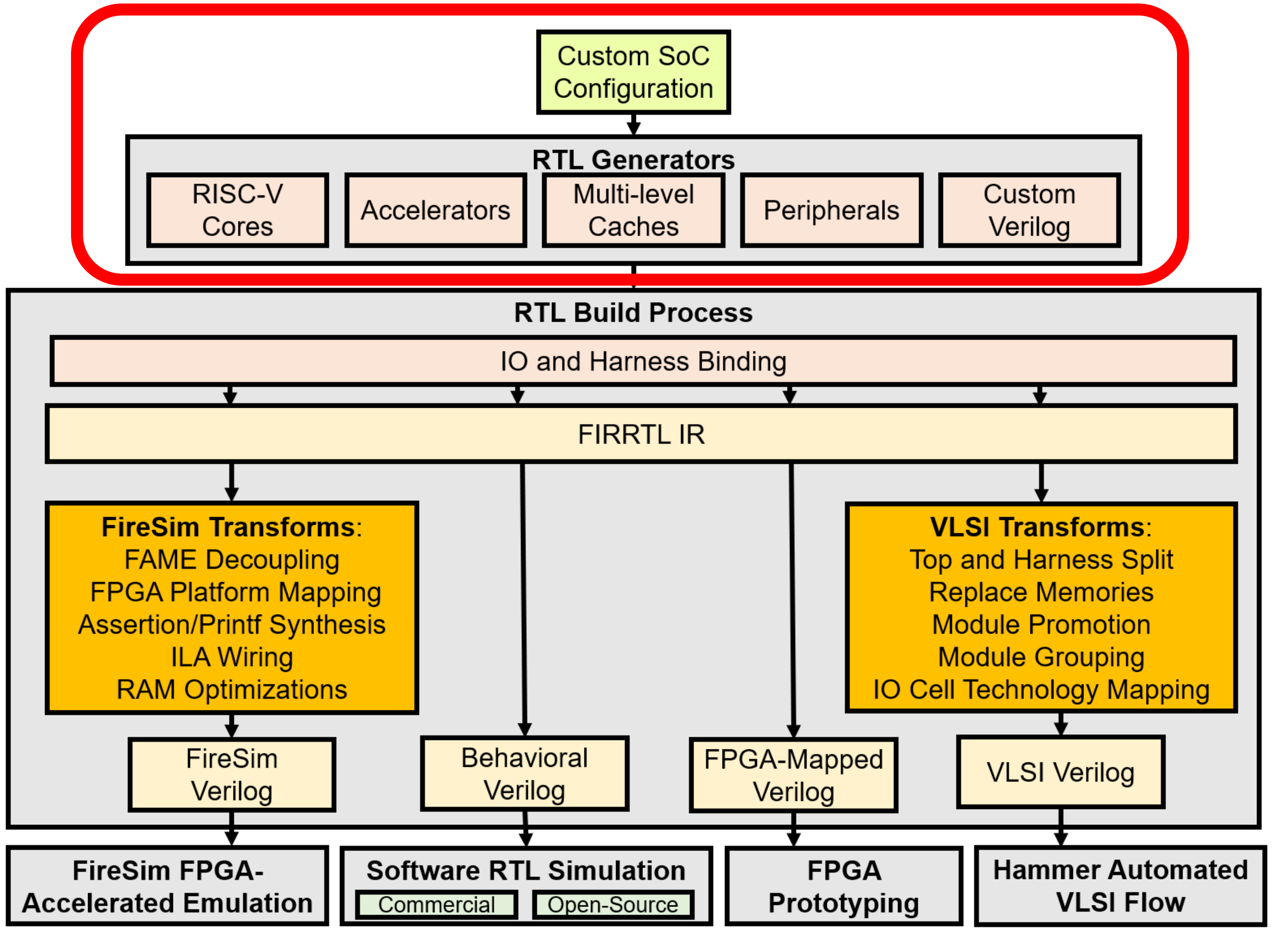


Custom SoC Configuration
RTL Generators
RISC-V Cores
Accelerators
Multi-level Caches
Peripherals
Custom Verilog
RTL Build Process
IO and Harness Binding
FIRRTL IR
FireSim Transforms: FAME Decoupling, FPGA Platform Mapping, Assertion/Printf Synthesis, ILA Wiring, RAM Optimizations
VLSI Transforms: Top and Harness Split, Replace Memories, Module Promotion, Module Grouping, IO Cell Technology Mapping
FireSim Verilog
Behavioral Verilog
FPGA-Mapped Verilog
VLSI Verilog
FireSim FPGA-Accelerated Emulation
Software RTL Simulation
Commercial
Open-Source
FPGA Prototyping
Hammer Automated VLSI Flow

Berkeley Architecture Research

How is this integrated? Generators!

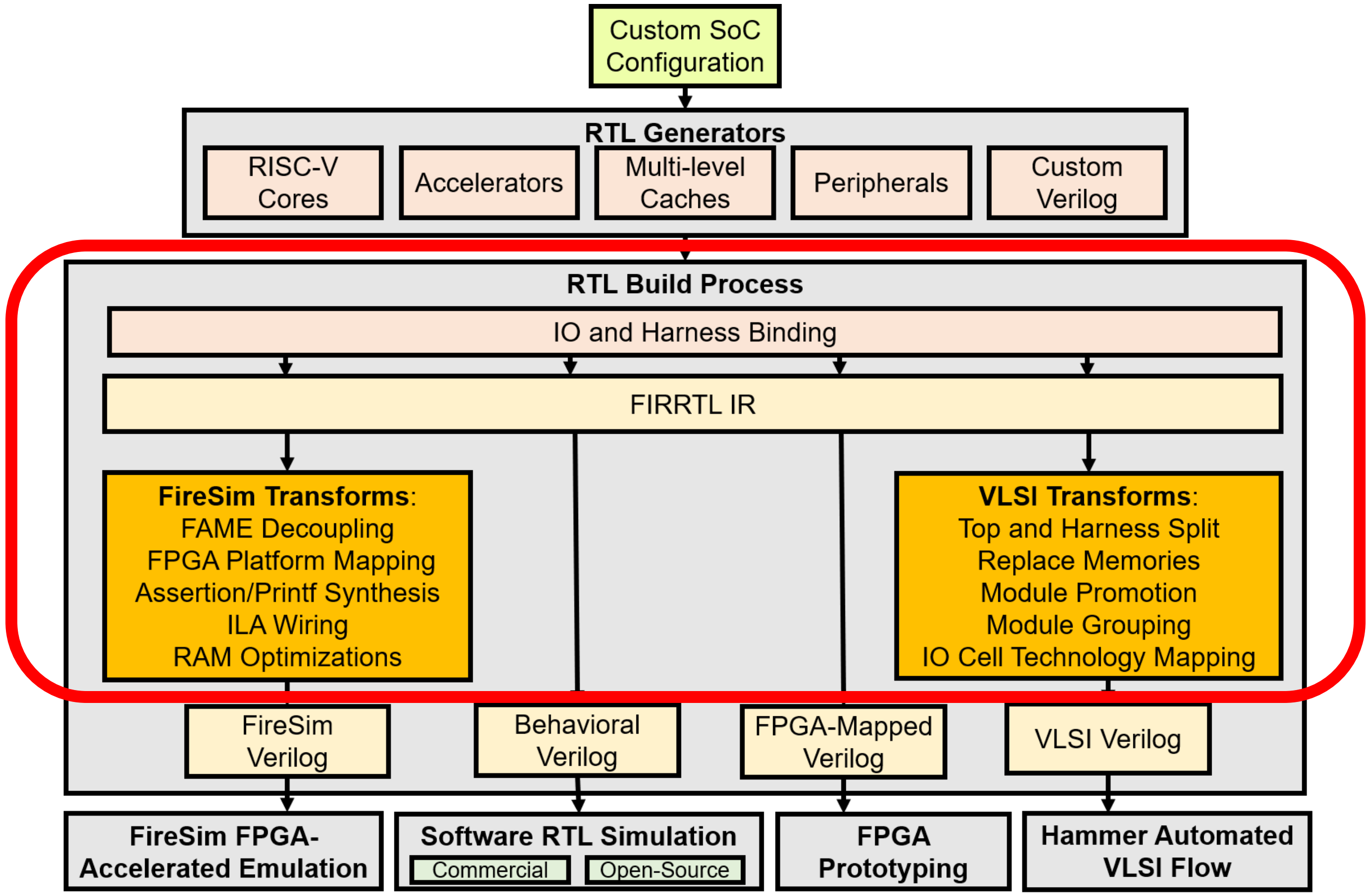
- Everything starts from a generator configuration
- Generators written in Chisel
- Generators can integrate third-party Verilog instance IP


Custom SoC Configuration
RTL Generators
RISC-V Cores
Accelerators
Multi-level Caches
Peripherals
Custom Verilog
RTL Build Process
IO and Harness Binding
FIRRTL IR
FireSim Transforms: FAME Decoupling, FPGA Platform Mapping, Assertion/Printf Synthesis, ILA Wiring, RAM Optimizations
VLSI Transforms: Top and Harness Split, Replace Memories, Module Promotion, Module Grouping, IO Cell Technology Mapping
FireSim Verilog
Behavioral Verilog
FPGA-Mapped Verilog
VLSI Verilog
FireSim FPGA-Accelerated Emulation
Software RTL Simulation
Commercial
Open-Source
FPGA Prototyping
Hammer Automated VLSI Flow

Berkeley Architecture Research

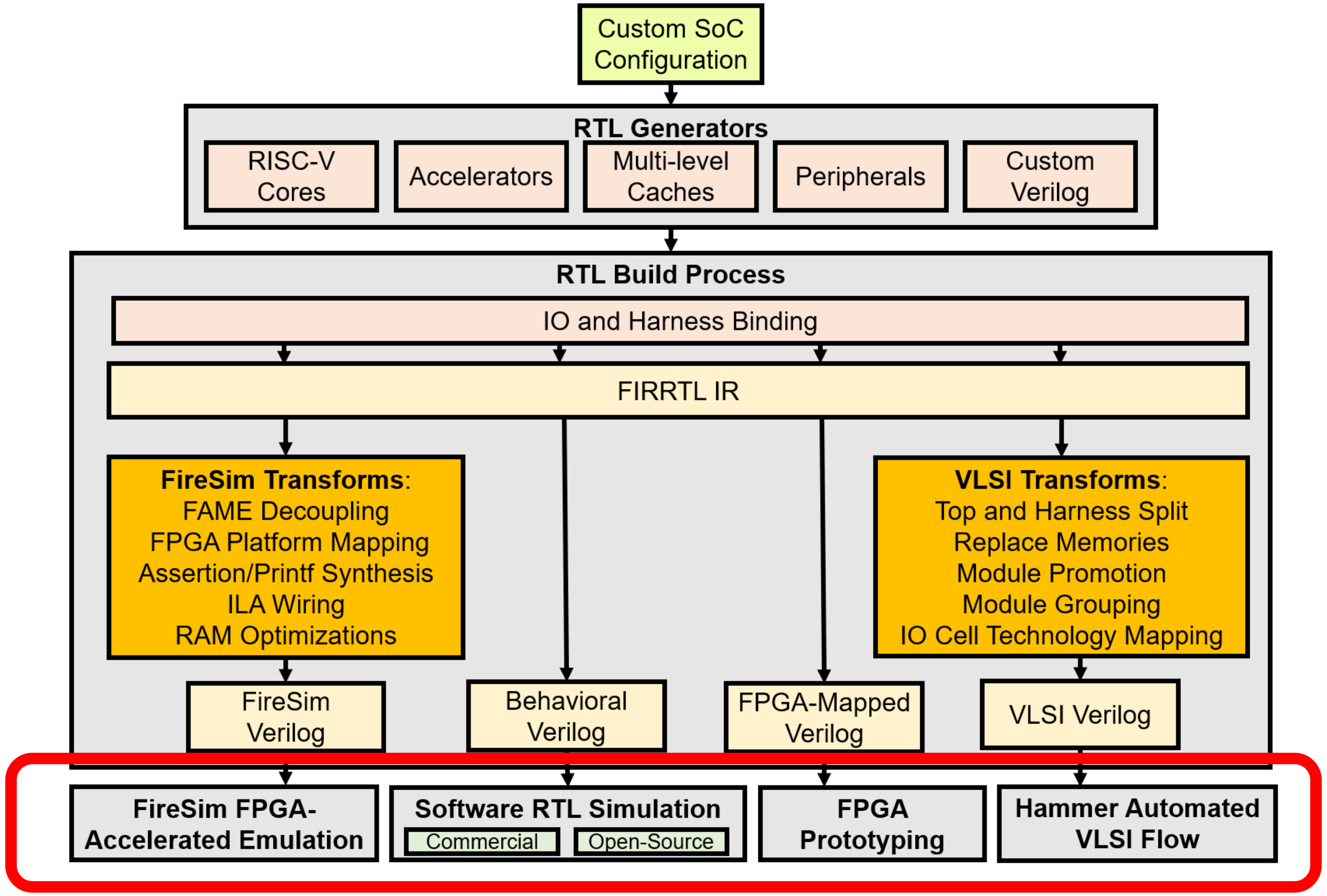
How is this integrated? Generators!

- Elaboration and Transformation
- Internals: FIRRTL – IR enables automated manipulation of the hardware description
- Externals: I/O and Harness Binders – pluggable interface functions enable automated targeting of different external interface requirements



How is this integrated? Generators!

- Design flows
  - Software RTL Simulation
  - FPGA-Accelerated Emulation
  - FPGA Prototyping
  - VLSI Fabrication


Custom SoC Configuration
RTL Generators
RISC-V Cores
Accelerators
Multi-level Caches
Peripherals
Custom Verilog
RTL Build Process
IO and Harness Binding
FIRRTL IR
FireSim Transforms: FAME Decoupling FPGA Platform Mapping Assertion/Printf Synthesis ILA Wiring RAM Optimizations
VLSI Transforms: Top and Harness Split Replace Memories Module Promotion Module Grouping IO Cell Technology Mapping
FireSim Verilog
Behavioral Verilog
FPGA-Mapped Verilog
VLSI Verilog
FireSim FPGA-Accelerated Emulation
Software RTL Simulation
Commercial
Open-Source
FPGA Prototyping
Hammer Automated VLSI Flow

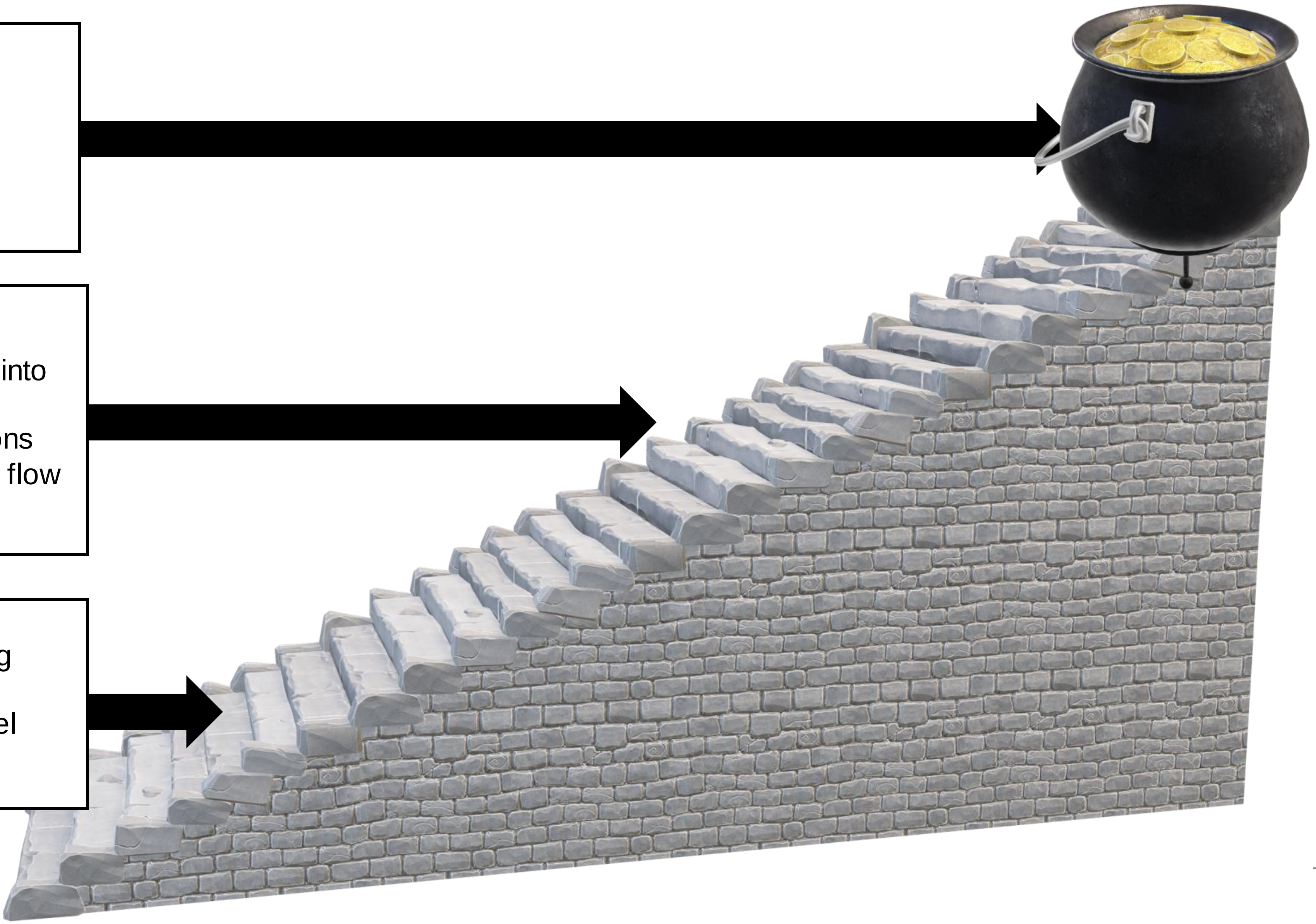
Berkeley Architecture Research

Chipyard Learning Curve

Advanced-level
- Configure custom IO/clocking setups
- Develop custom FireSim extensions
- Integrate and tape-out a complete SoC

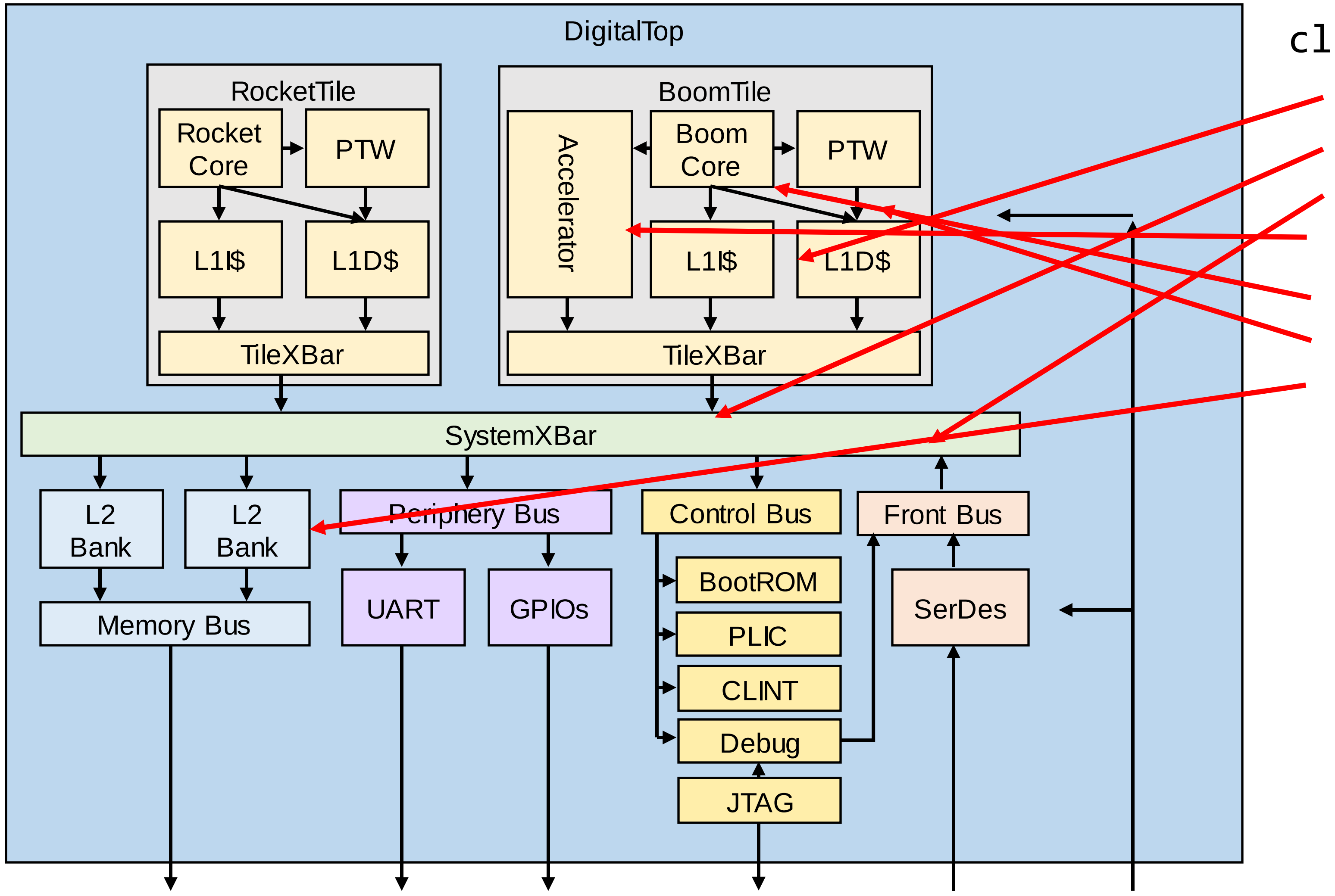
Evaluation-level
- Integrate or develop custom hardware IP into Chipyard
- Run FireSim FPGA-accelerated simulations
- Push a design through the Hammer VLSI flow
- Build your own system

Exploratory-level
- Configure a custom SoC from pre-existing components
- Generate RTL, and simulate it in RTL level simulation
- Evaluate existing RISC-V designs



Berkeley Architecture Research

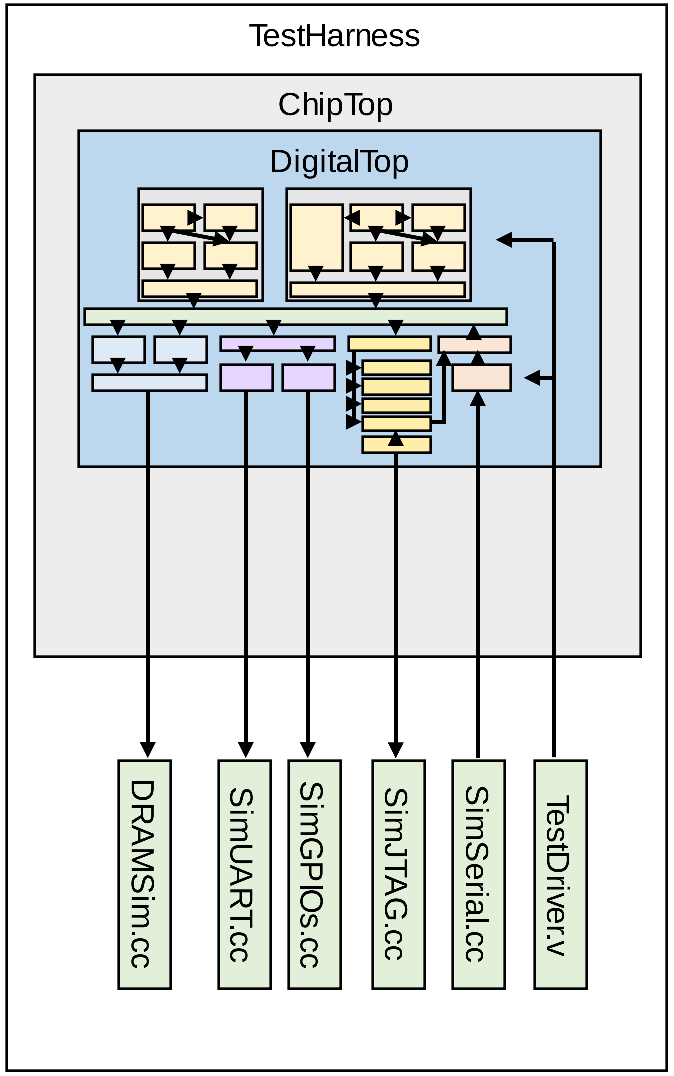
Highly Parameterized Configurations

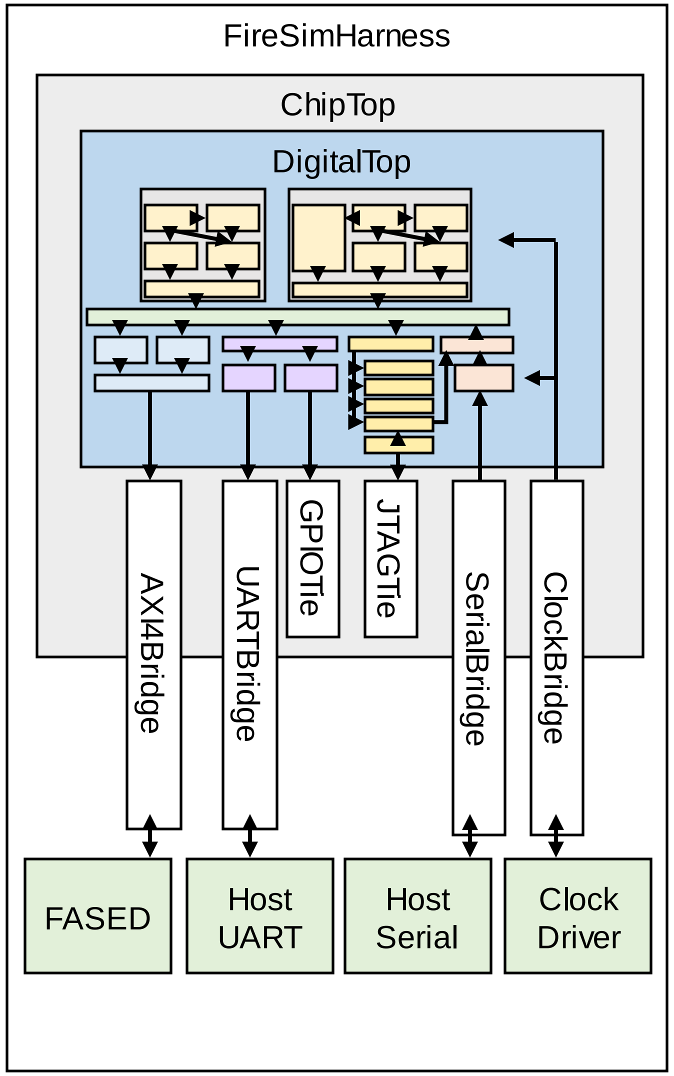


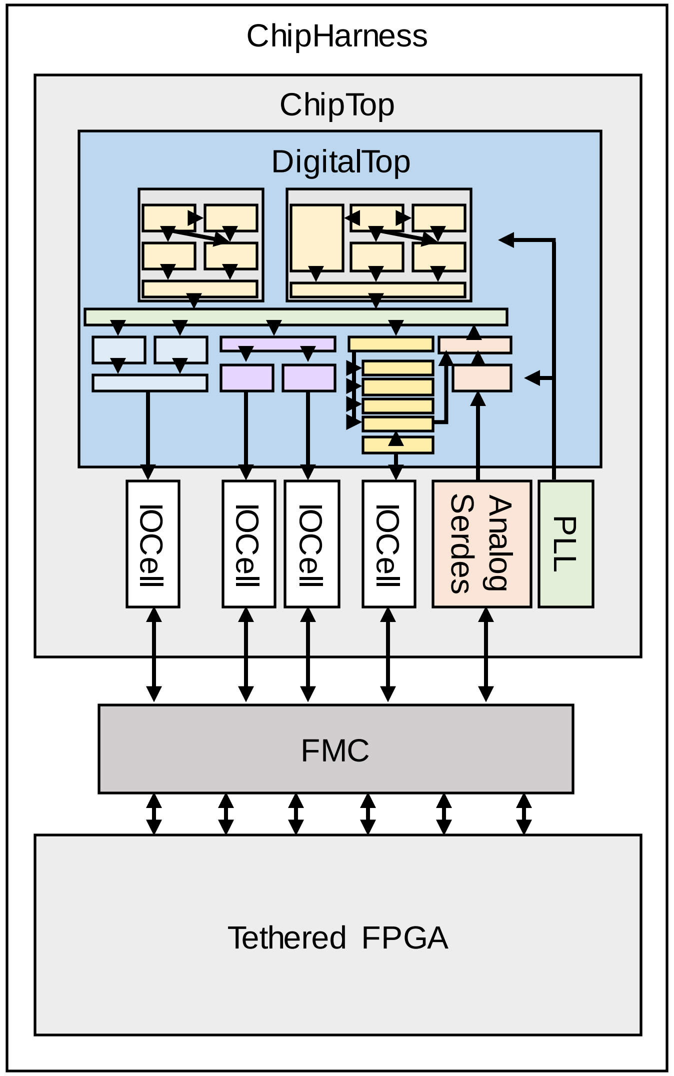
```
class CustomConfig extends Config(
  new WithL1CacheWays(4) ++
  new WithAsyncTiles ++
  new WithRingSystemBus +
  new WithFPGemmini ++
  new With3WideBooms ++
  new WithL2TLBs(512) ++
  new WithL2Banks(4) ++

  new WithDefaultGemmini ++
  new WithNRocketCores(1) ++
  new WithNBoomCores(1) ++
  new WithBootROM ++
  new WithUART ++
  new WithJtagDTM ++
  new WithGPIOs ++
  new WithInclusiveCache(512) ++
```

Berkeley Architecture Research

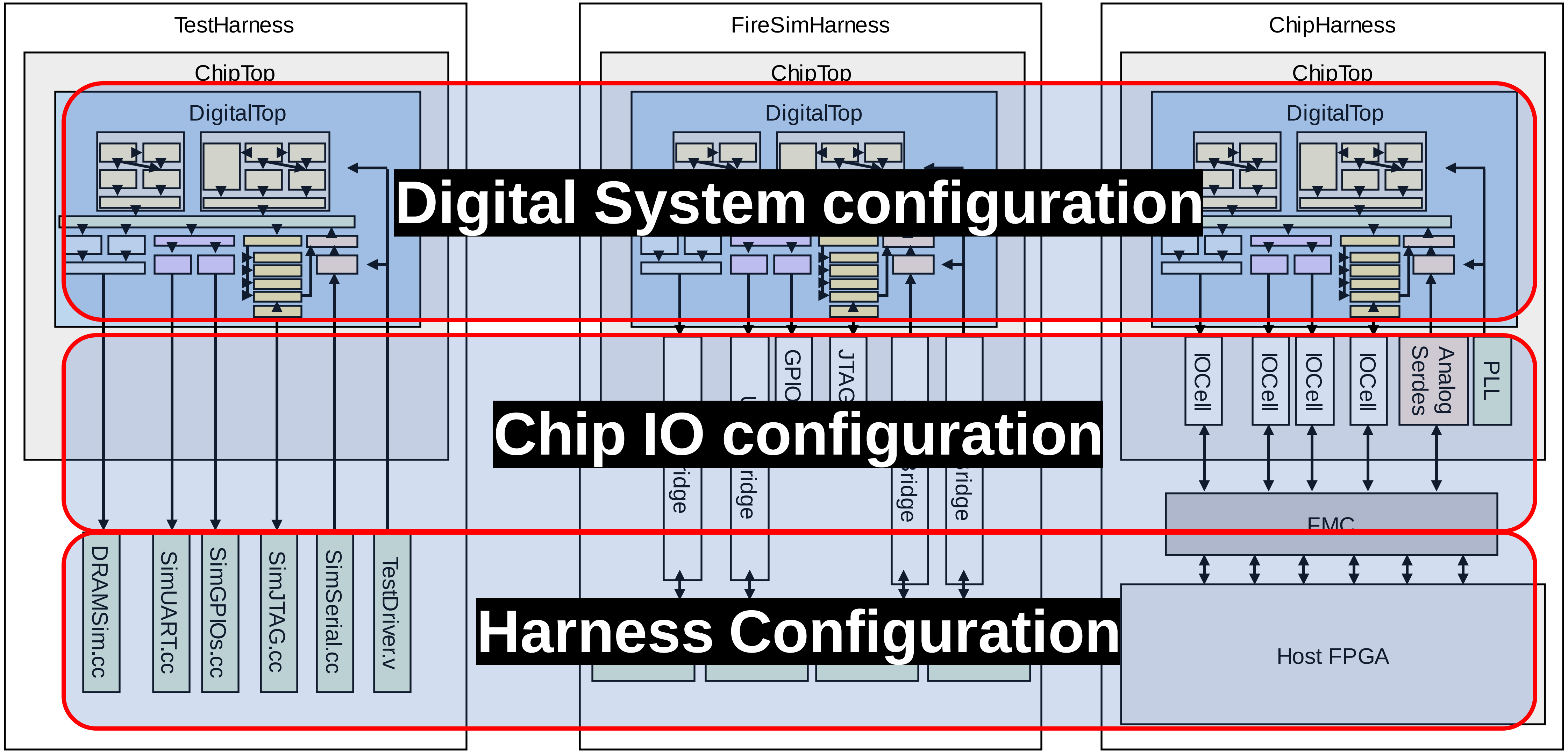
Multipurpose



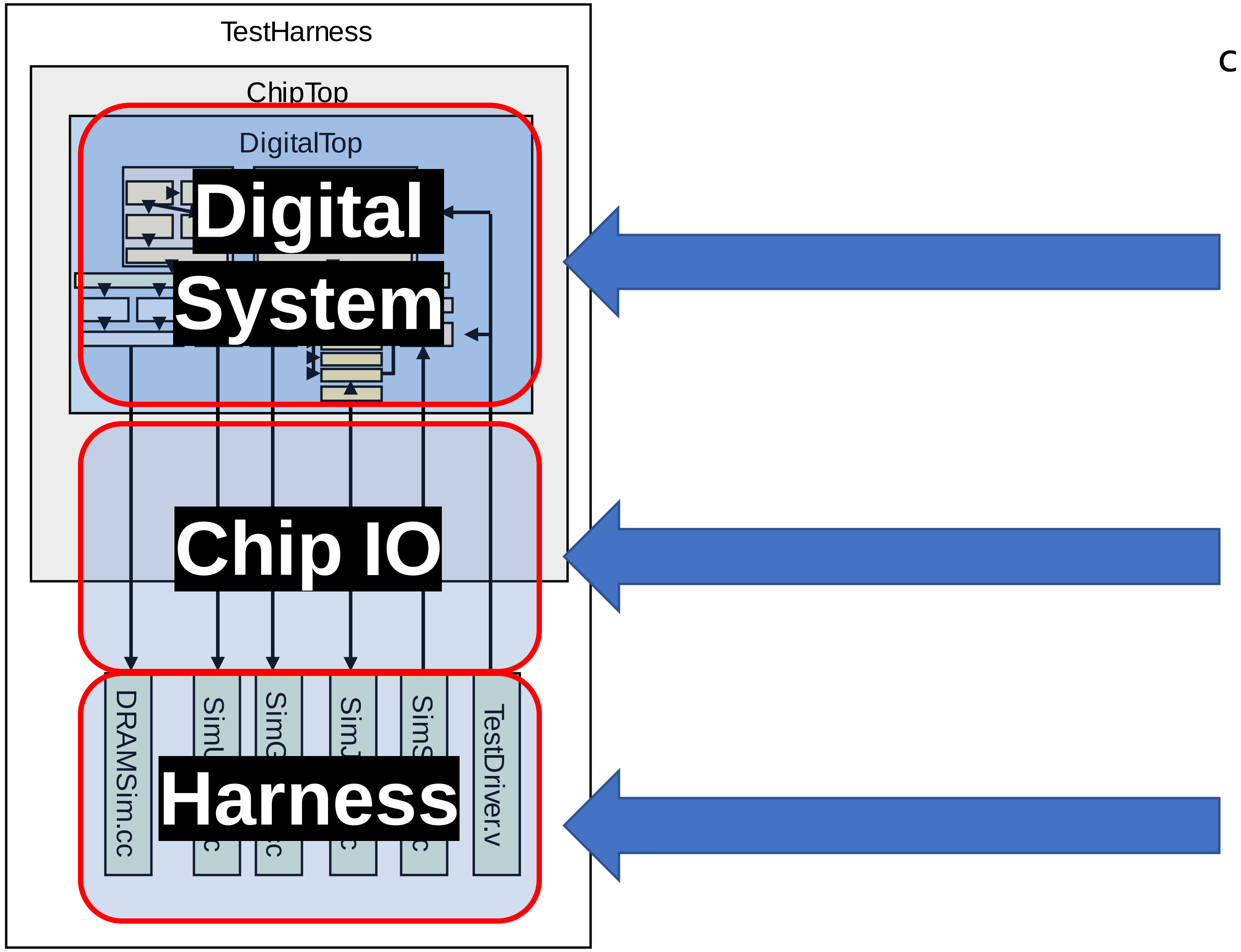




Multipurpose



A Complete Config



```
class CustomConfig extends Config(
  new WithDefaultGemmini ++
  new WithNRocketCores(1) ++
  new WithNBoomCores(1) ++
  new WithBootROM ++
  new WithUART ++
  new WithJtagDTM ++
  new WithGPIOs ++
  new WithInclusiveCache(512) ++

  new WithPassThroughIOs ++


  new WithDRAMSim ++
  new WithSimUART ++
  new WithSimJTAG ++
  new WithSimSerial

)
```

Berkeley Architecture Research

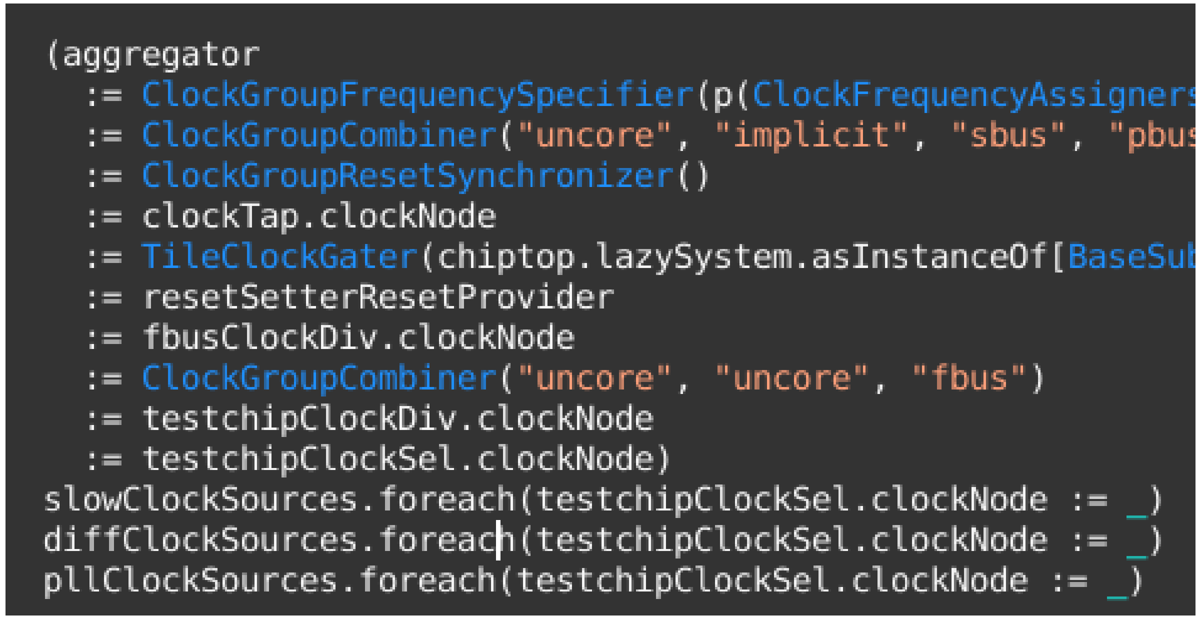
Using Chipyard Diplomatic Clocking

Clock and reset distribution is hard

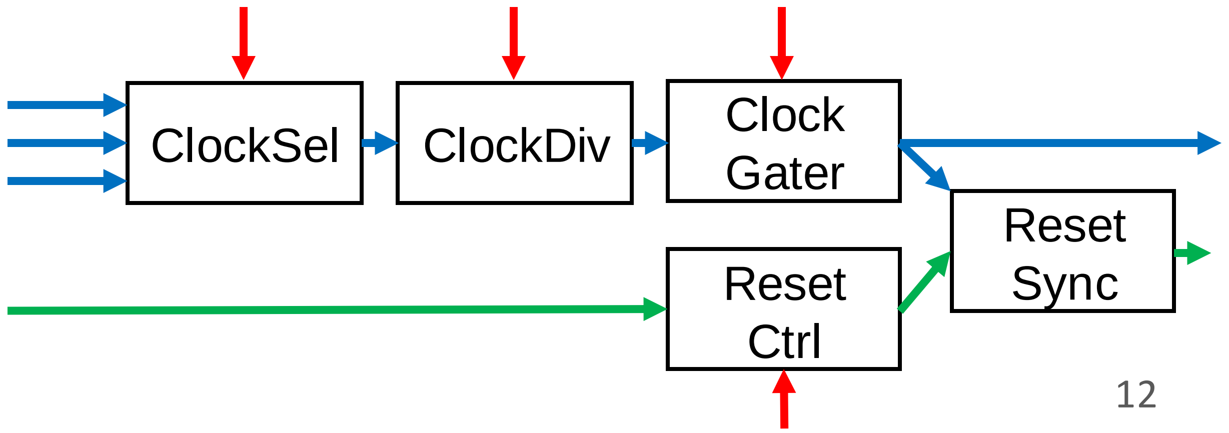
- Many clock sources
- Programmable clock/reset control
- Dangerous source of bugs

Chipyard Diplomatic Clocking

- Leverage RocketChip diplomatic clock graphs
- Pull out clock graph out of digital core logic
- Describe clock/reset distribution as graph of nodes



```
(aggregator
  := ClockGroupFrequencySpecifier(p(ClockFrequencyAssigners
  := ClockGroupCombiner("uncore", "implicit", "sbus", "pbus
  := ClockGroupResetSynchronizer()
  := clockTap.clockNode
  := TileClockGater(chiptop.lazySystem.asInstanceOf[BaseSub
  := resetSetterResetProvider
  := fbusClockDiv.clockNode
  := ClockGroupCombiner("uncore", "uncore", "fbus")
  := testchipClockDiv.clockNode
  := testchipClockSel.clockNode)
slowClockSources.foreach(testchipClockSel.clockNode := _)
diffClockSources.foreach(testchipClockSel.clockNode := _)
pllClockSources.foreach(testchipClockSel.clockNode := _)
```



Bringup Flow – JTAG vs TSI

JTAG – standard interface for debug

- Writes program byte-by-byte into target memory using standard JTAG commands
- Send commands to interrupt core, direct core to PC, read core registers, etc.
- Slow, but standard

TSI – tethered serial interface

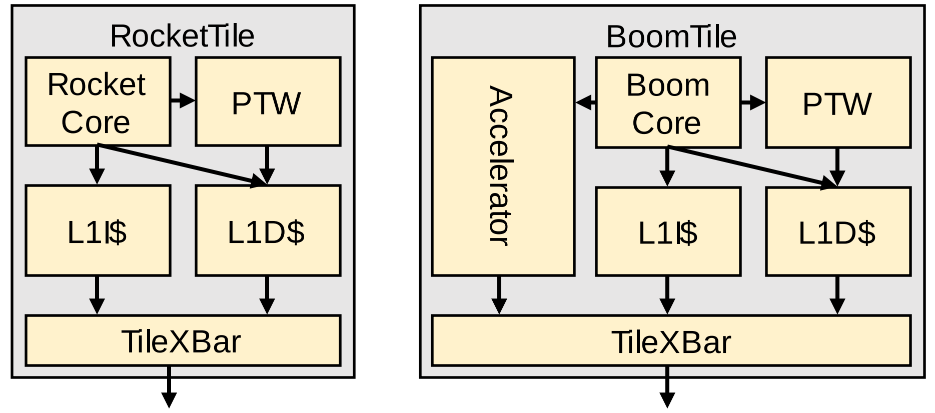
- Simple fast test chip interface
- Chip communicates with "frontend-server" running on FPGA soft-core

Chipyard Build System

- Based on Make
- **Variables:**
  - RISCV: specifies directory of riscv-tools installation, should be set by environment file
  - CONFIG: references Scala config describing design
- **RTL-sim Targets:**
  - run-binary-debug: runs BINARY on simulator of CONFIG
- **VLSI Targets:**
  - buildfile/syn/par/drc/lvs: runs steps in VLSI flow
  - redo-syn/par/drc/lvs: runs steps in VLSI flow (does not rerun prereq steps)

Chipyard Tiles

SoC Organization: Tiles



Tiles: Units of replication in a multi-core SoC

Contains:

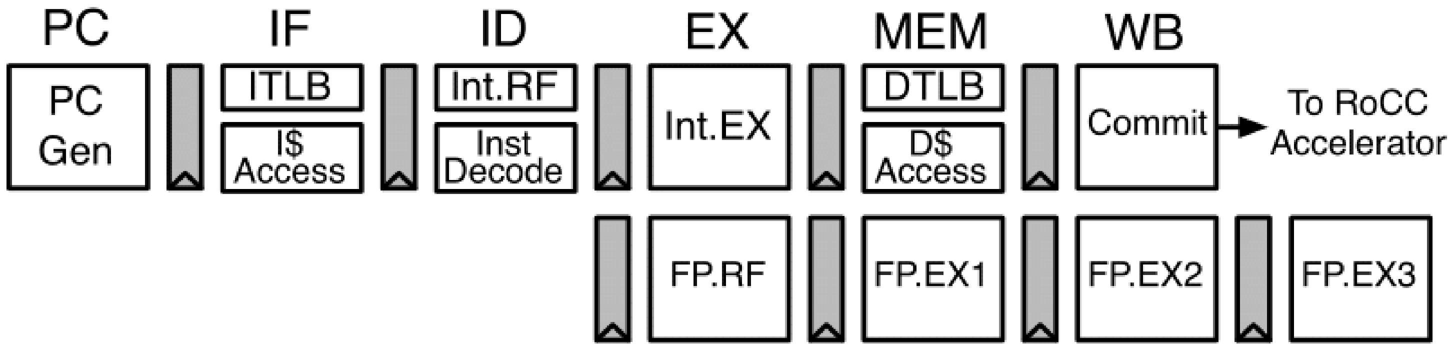
- RISC-V core
- Private L1 caches
- TLBs, PTW
- RoCC accelerator?

Many config options:

- Different core options
- Accelerator interface
- Cache sizes
- Etc.

Berkeley Architecture Research

ML Tile – Rocket Core



Rocket:

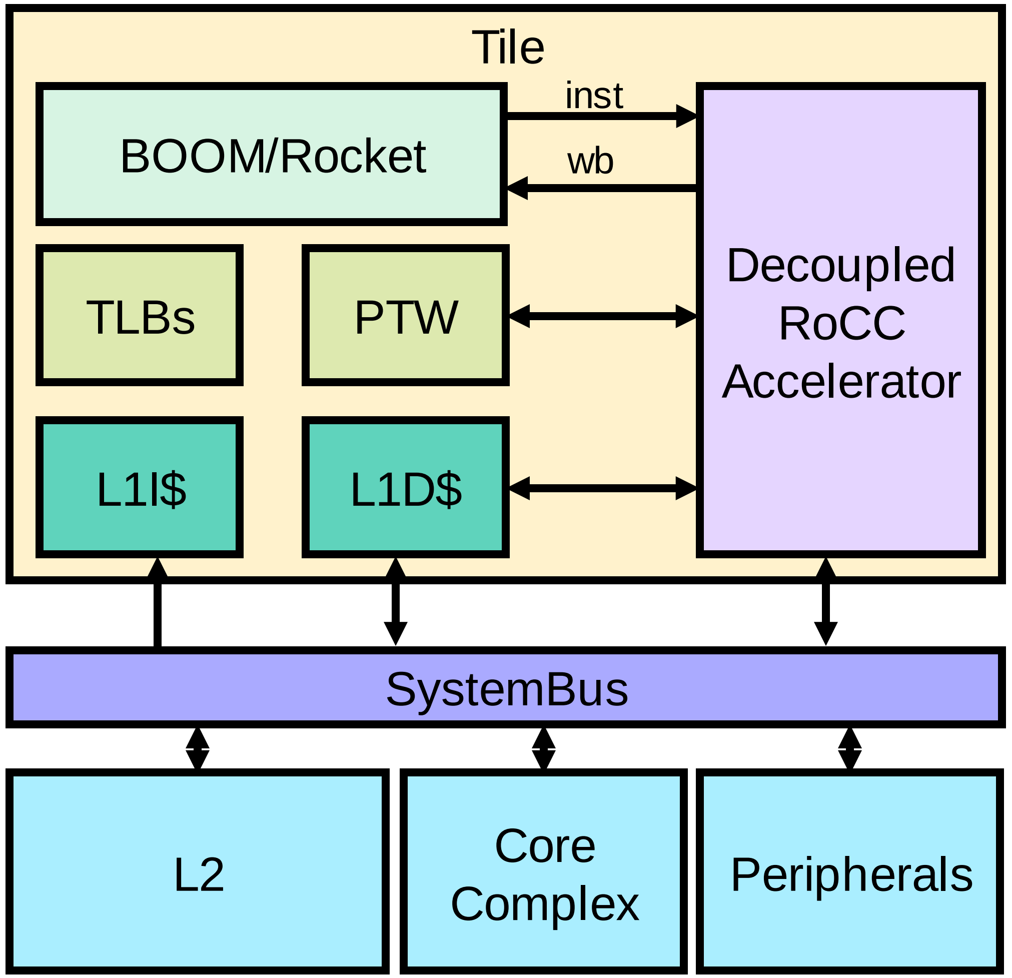
- First open-source RISC-V CPU
- In-order, single-issue RV64GC core
- Efficient design point for low-power devices
- Virtual memory, floating point, caches, etc.
- Not the focus of this tile, focus on RoCC accelerator interface

ML Tile RoCC Interface

- **RoCC:** Rocket Custom Coprocessor
- Sits adjacent to Rocket
- Execute custom RISC-V instructions for a custom extension

31 25	24 20	19 15	14	13	12	11 7	6 0
funct	rs2	rs1	xd	xs1	xs2	rd	opcode
7	5	5	1	1	1	5	7

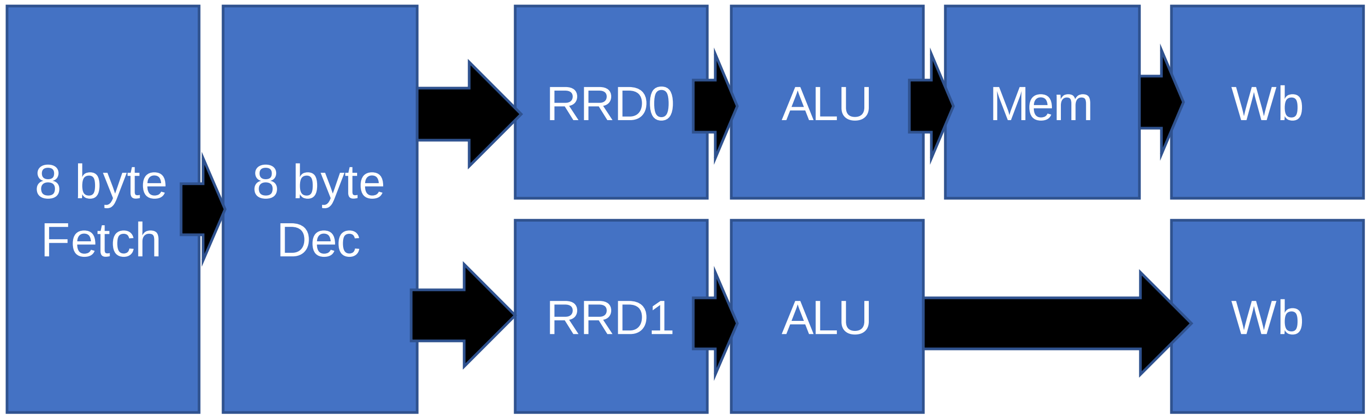
- Implement an accelerator as RoCC
  - Receive commands/data from core
  - Access memory through L1D$
  - Develop SW using RoCC (compare accelerator vs CPU)



Berkeley Architecture Research

GP Tile – Saturn Core

- Superscalar in-order core (fetches/decodes/executes multiple instructions per cycle)
- Baseline: dual-issue (2 instructions per cycle)
- Opportunities for customization
  - Three-issue, four-issue, etc.
  - Advanced cache
  - Custom functional unit
  - Prefetching

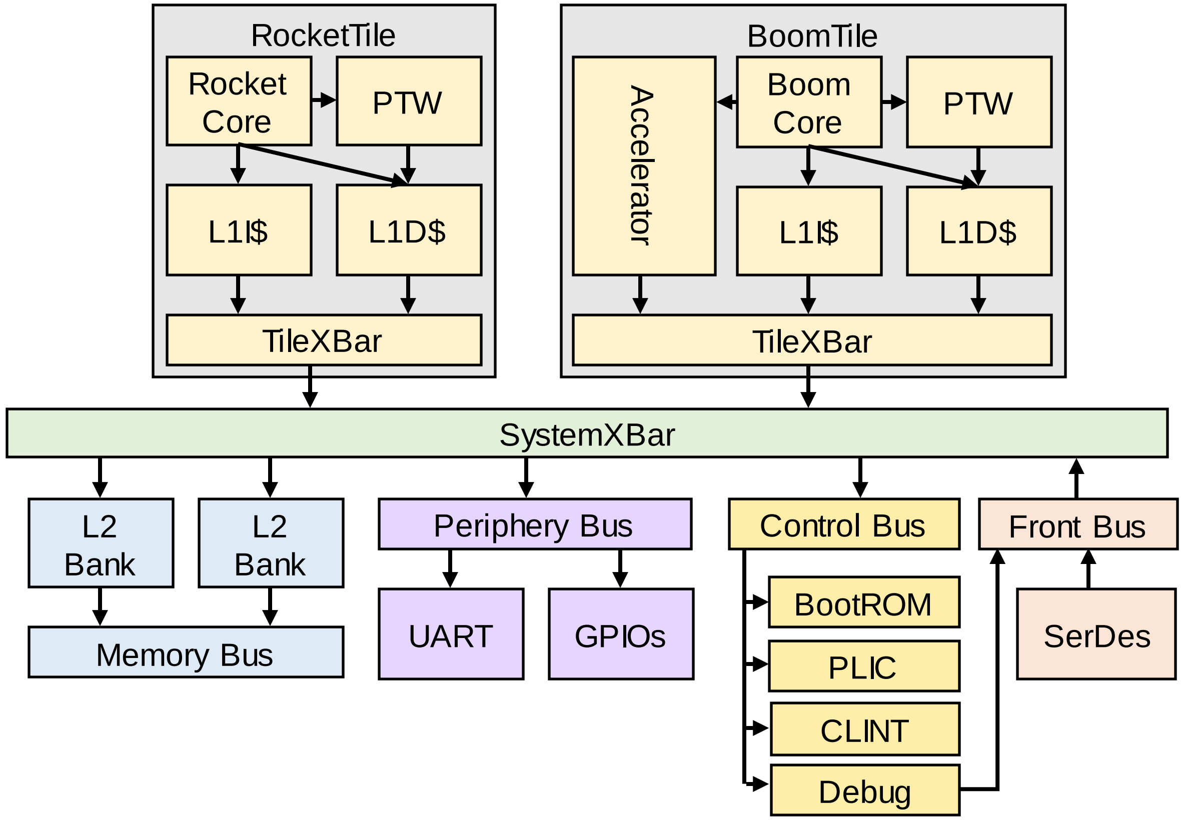


Berkeley Architecture Research

Memory Subsystem

Berkeley Architecture Research

SoC Organization: Digital System



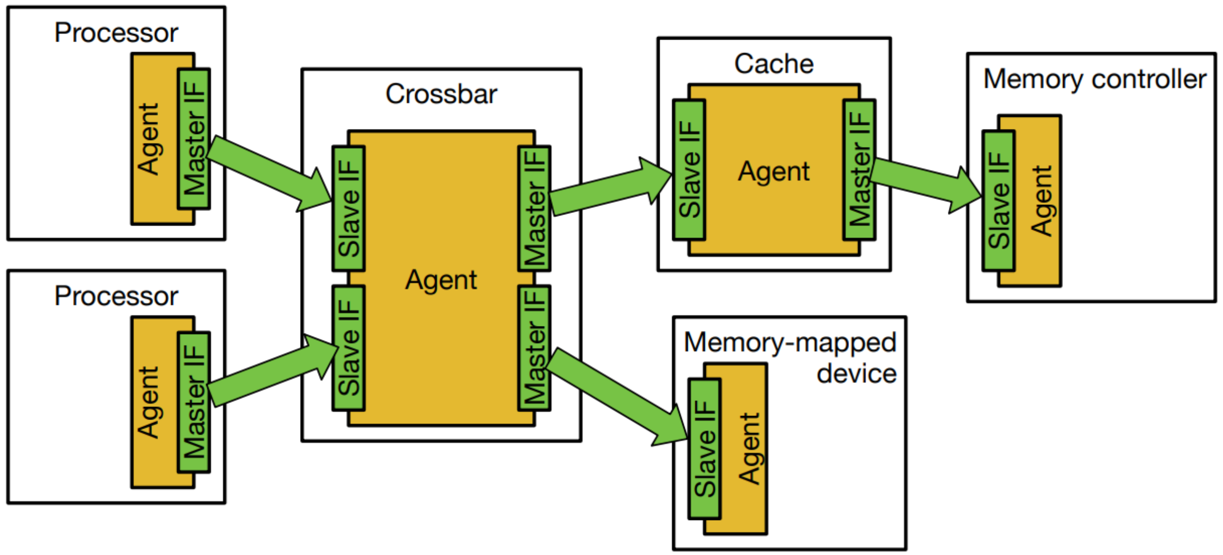
L2 Banks: Distributed banks of L2 cache, each bank manages subset of the address space

MMIO devices: Hang off the system bus or periphery bus

TileLink: Coherent interconnect standard (like AXI)

Interconnect: Physical transport for interconnect protocol

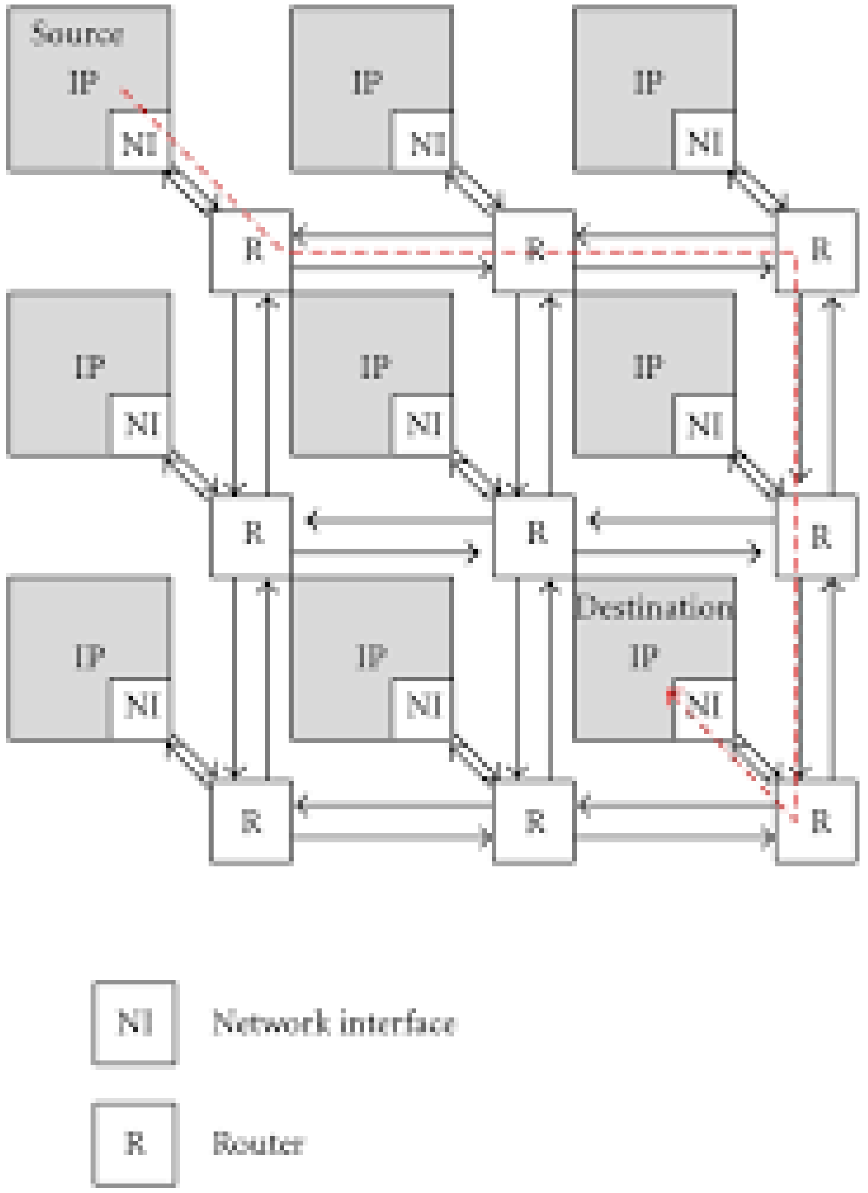
TileLink Protocol



- Free and open chip-scale interconnect protocol
- Supports multiprocessors, coprocessors, accelerators, DMA, peripherals, etc.
- Provides a physically addressed, shared-memory system
- Supports cache-coherent shared memory, MOESI-equivalent protocol

Coherent Interconnect

- Purpose: manage routing of messages between clients (cores/accelerators/DMAs) and managers (caches/scratchpads/MMIO)
- Baseline: Fully-connected all-to-all crossbar (xbar)
  - Good for small, simple designs
  - Does not scale to large systems
- Goal: Tape-out a network-on-chip
  - Multiplexes message routing onto fewer physical resources
  - We will provide NoC generator



Questions?

Berkeley Architecture Research