# Chipyard Intro and Fundamentals

Jerry Zhao

UC Berkeley

jzh@berkeley.edu

Berkeley Architecture Research

CHIPYARD

# Outline

- **Introduction to Chipyard**
- Chipyard Tooling
- Chipyard SoC Structure and Organization
- Why Chipyard?

# Trends in Open-source Hardware

- Organization/Specifications: RISC-V, CHIPS Alliance, OpenHW
- Community: LowRISC, FOSSi
- Academia: PULP Platform, OpenPiton, ESP
- Government: DARPA POSH
- Industry: WD SWERVE, NVIDIA NVDLA
- Tools: Verilator, Yosys, OpenRoad
- Fabrication: Skywater 130nm

RISC-V

CHIPS ALLIANCE

DARPA launches POSH project for open source hardware blocks

Jul 26, 2018 — by Eric Brown — 1151 views

Please share:

DARPA

DARPA announced the first grants for its $1.5 billion Electronic Resurgence Initiative for accelerating chip development. More than $35 million went to a "Posh Open Source Hardware" project for developing and verifying hardware IP.

FOSSi Foundation

Western Digital's RISC-V "SweRV" Core Design Released For Free

14 Comments

+ Add A Comment

by Anton Shilov on February 15, 2019 11:30 AM EST

Posted in Storage CPUs SSDs Western Digital RISC-V

OpenHW Group Created and Announces CORE-V Family of Open-source Cores for Use in ...ne Production SoCs

...Executive Director of the RISC-V Foundation, leads ...oration, ecosystem development and open-source

OPENHW GROUP

PROVEN PROCESSOR IP

Berkeley Architecture | OpenHW Group → Jun 06, 2019, 04:00 ET

MICROPROCESSOR report

Insightful Analysis of Processor Technologies

Nvidia Shares Its Deep Learning

Xavier Neural-Network Accelerator Now Available as Open Source

March 26, 2018

By Mike Demler

PDF Version

VERILATOR

# Building an Open Source RISC-V System

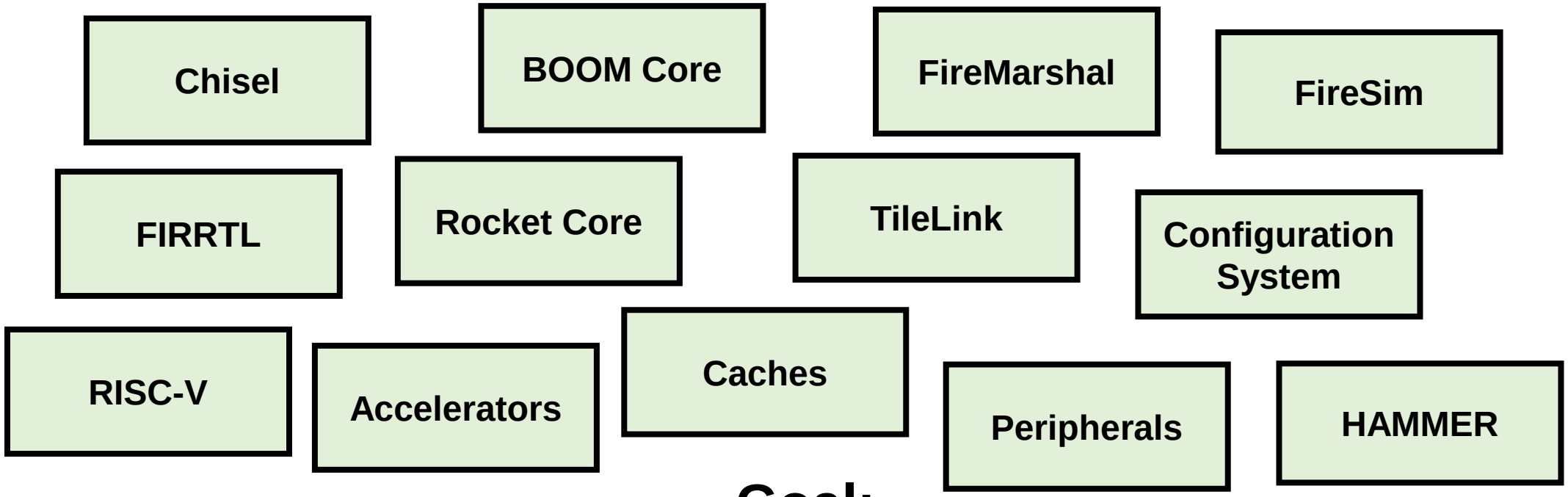
Cool! I want to build an Open-Source custom RISC-V SoC.
What do I need to do?

Have you heard of this Free and Open RISC-V thing? It should be so easy to build real systems now

Let me list all the cool projects you can use ….

# Motivation

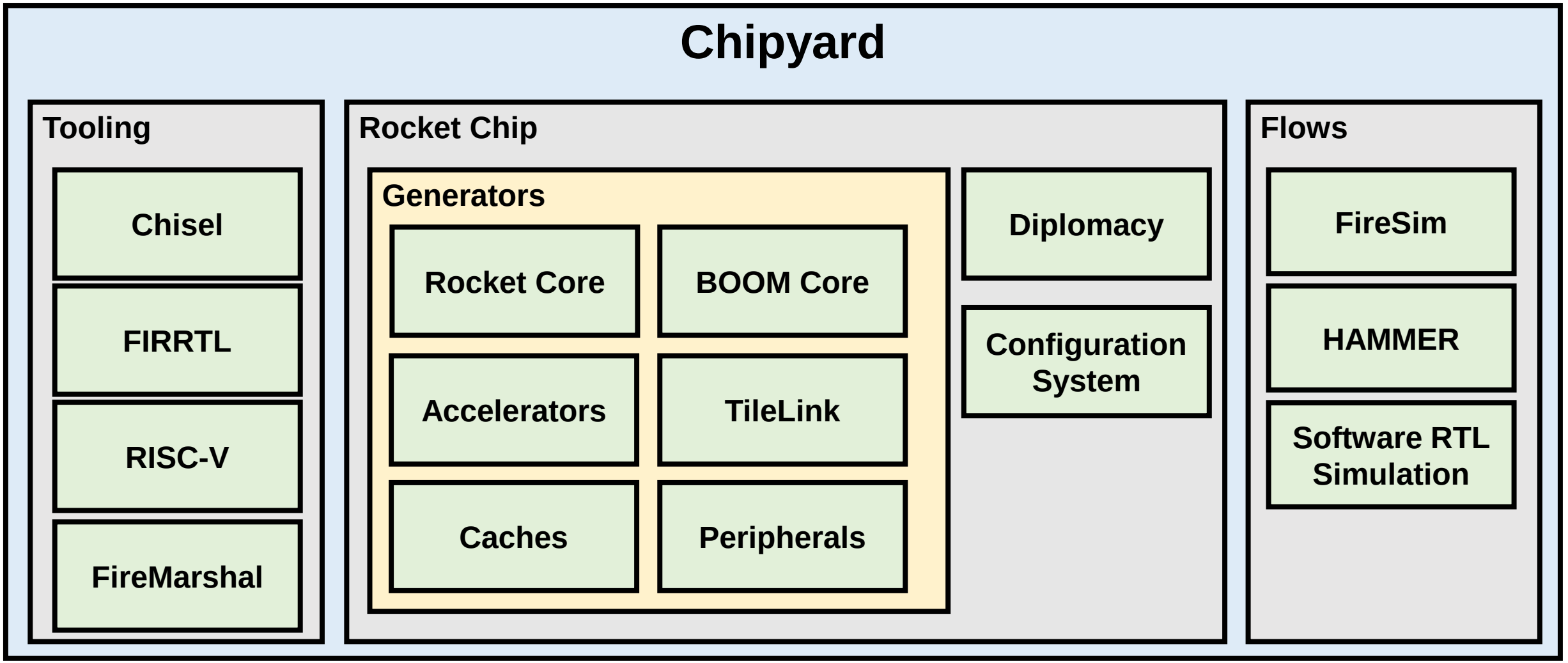
**Large library of open-source projects for RISC-V SoC development**



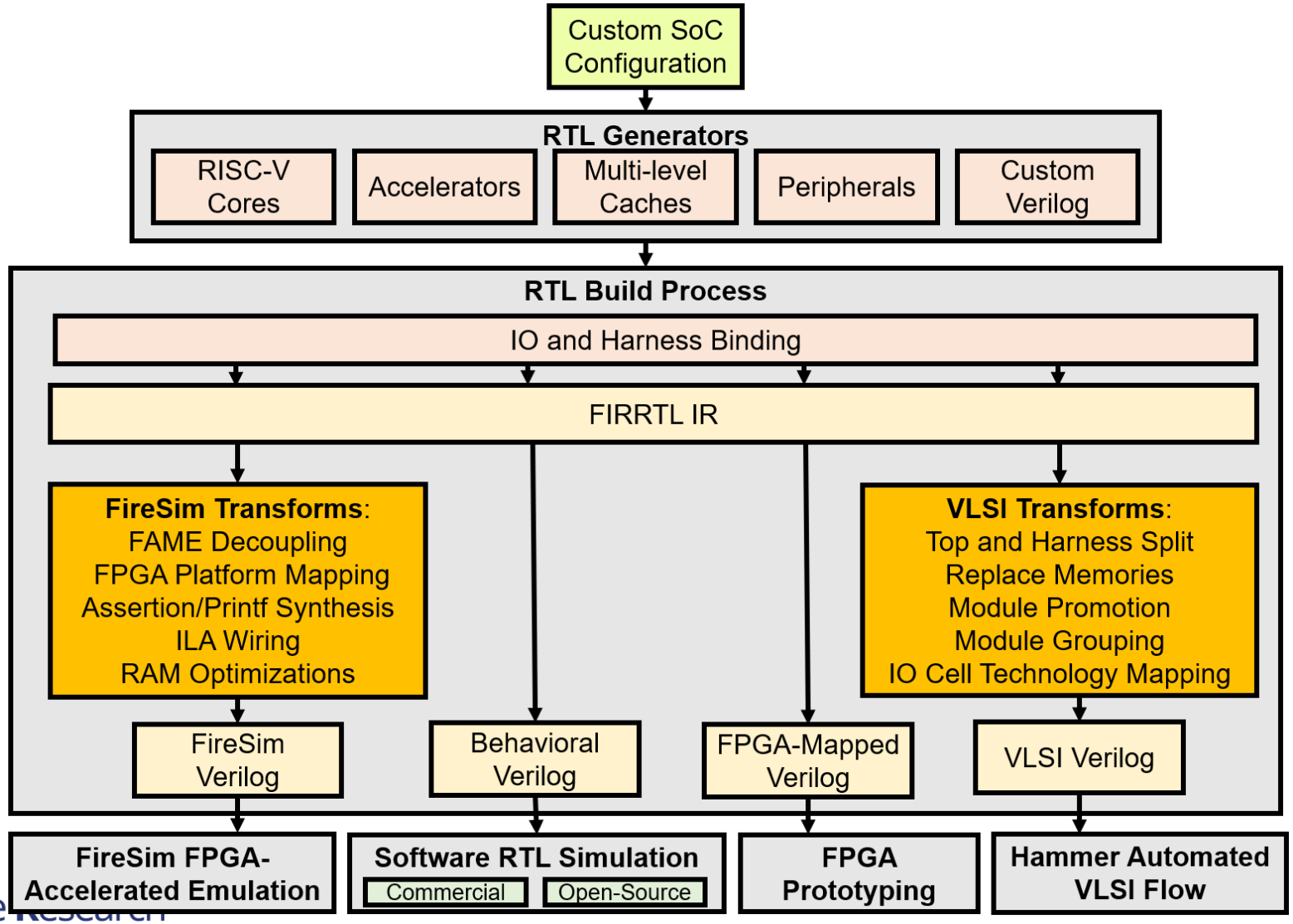

**Goal:**

Make it easy for small teams to
**design, integrate, simulate,** and **tape-out** a custom SoC

# Chipyard

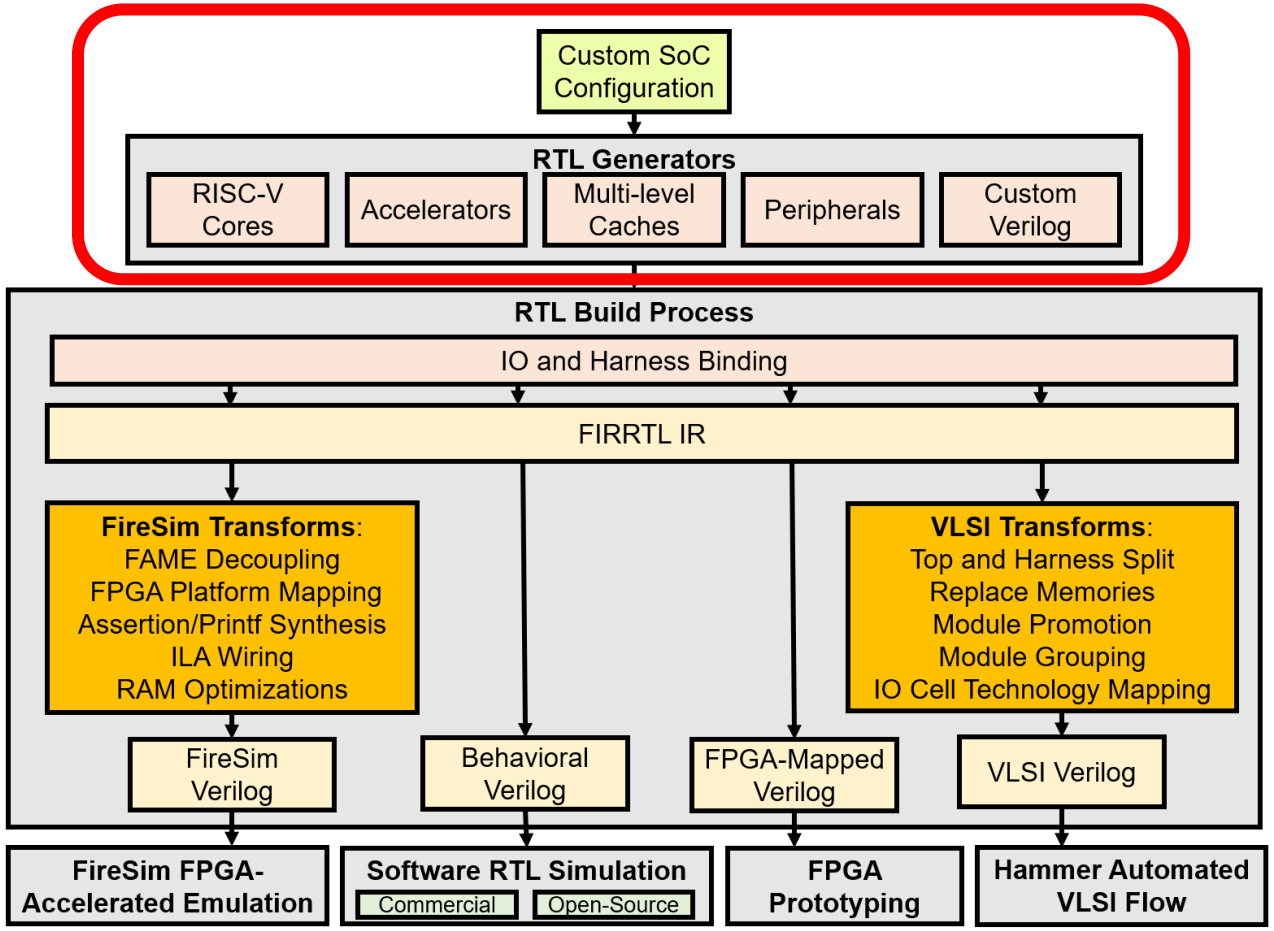

Chipyard
Tooling
Chisel
FIRRTL
RISC-V
FireMarshal
Rocket Chip
Generators
Rocket Core
BOOM Core
Accelerators
TileLink
Caches
Peripherals
Diplomacy
Configuration System
Flows
FireSim
HAMMER
Software RTL Simulation

# How is this integrated? Generators!



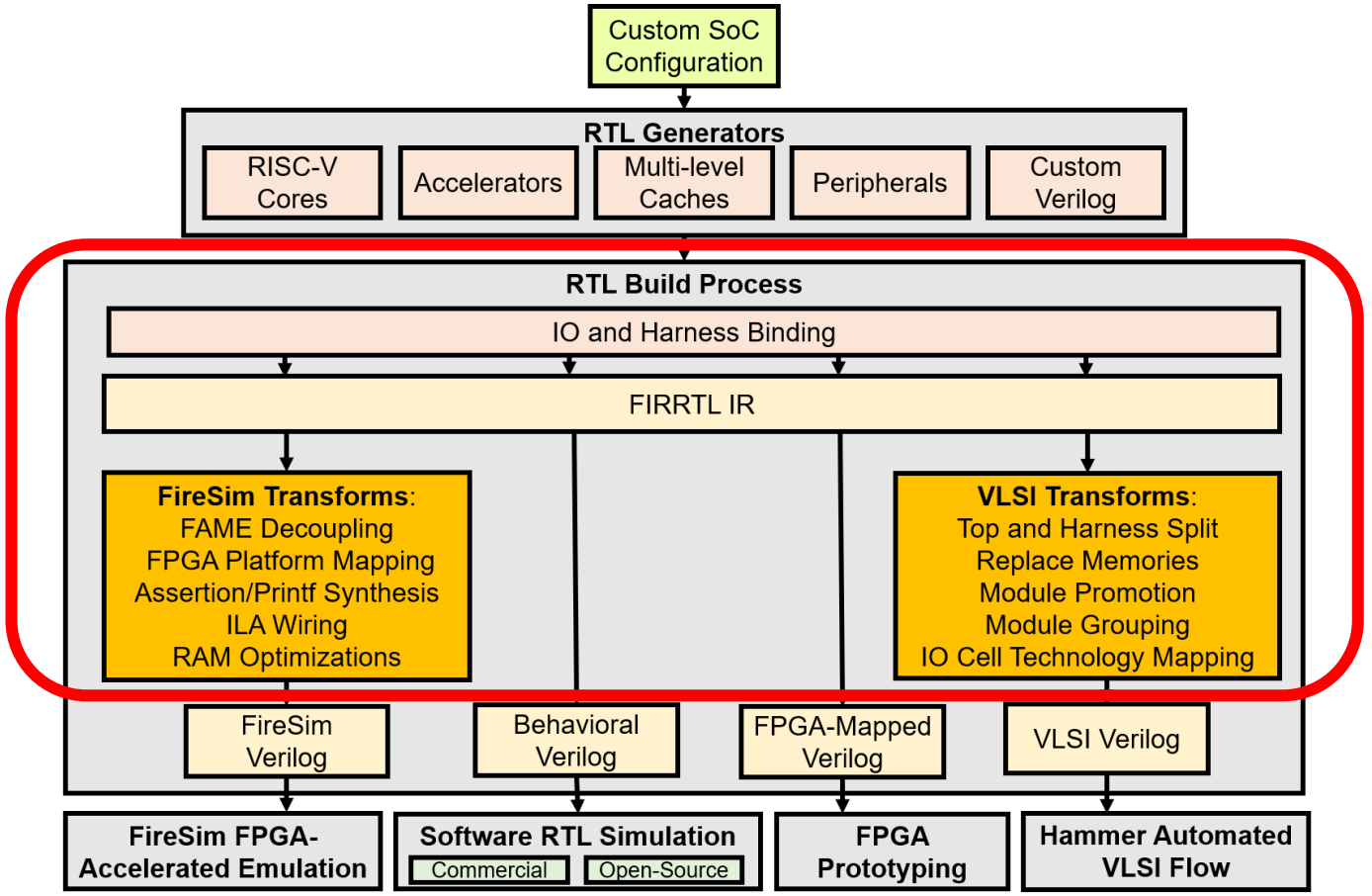
# How is this integrated? Generators!

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



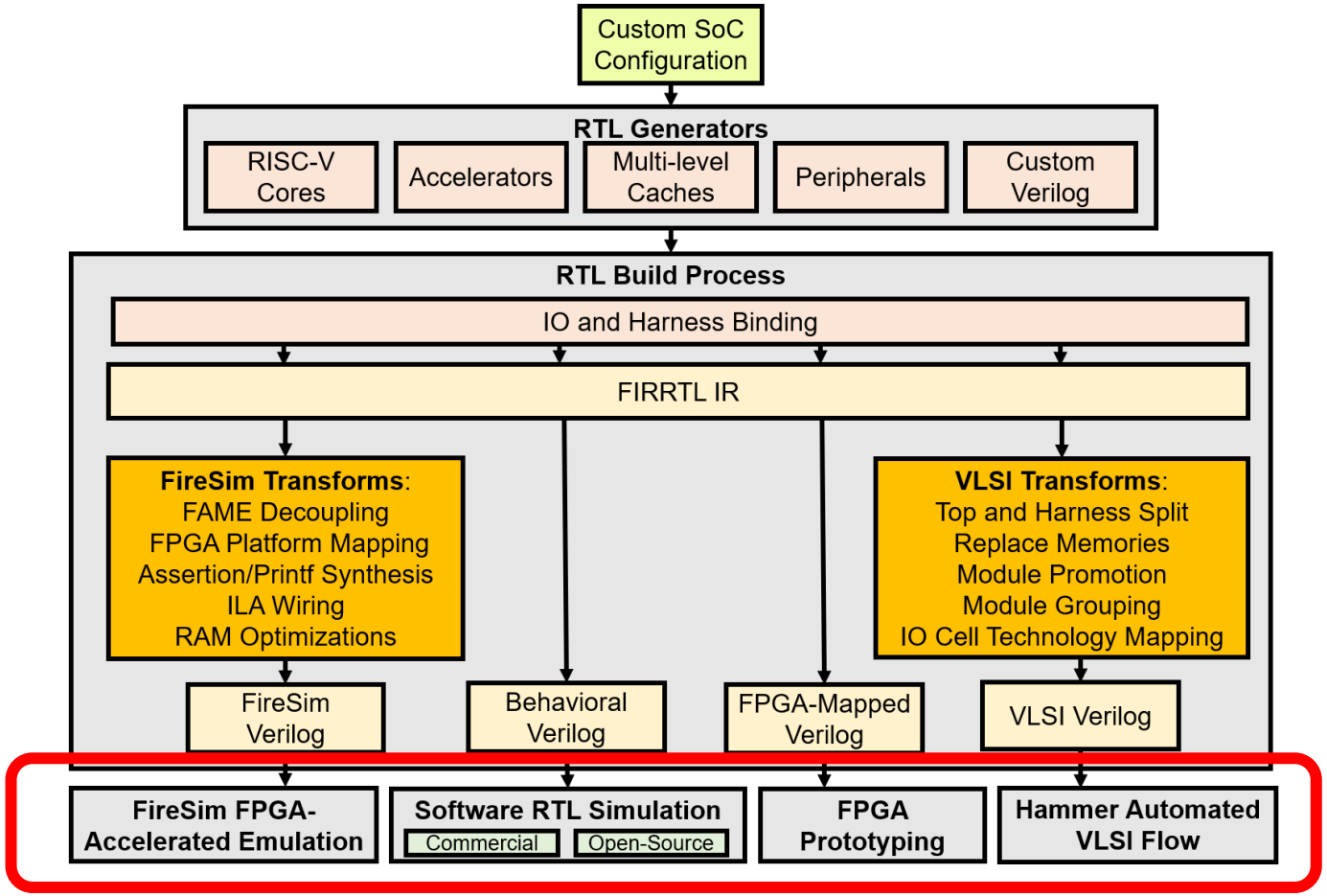
# How is this integrated? Generators!

- 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



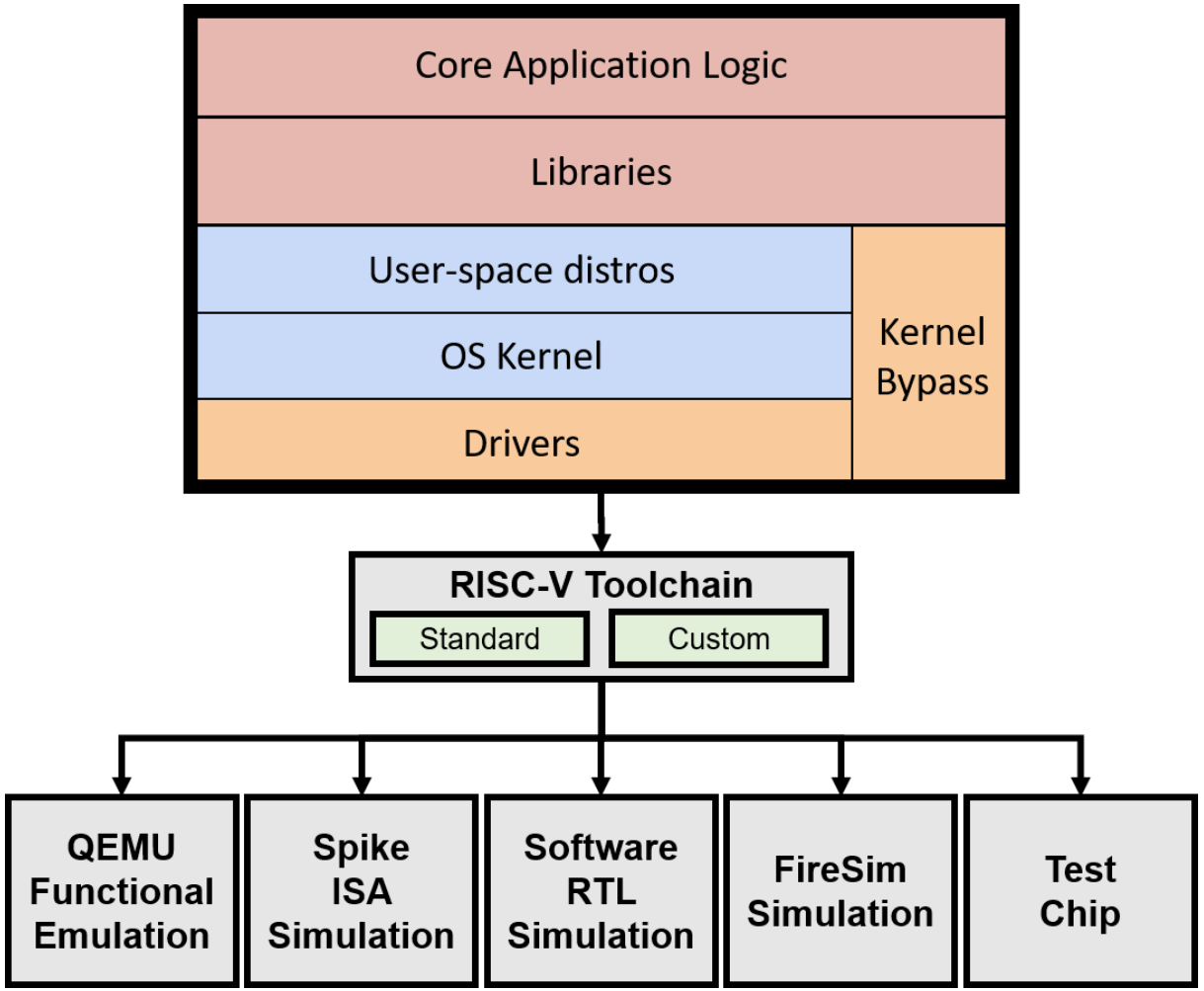
# How is this integrated? Generators!

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



# Software

- Hardware alone is not enough
- Custom SoCs require custom software
- Different platforms require different firmware
- Chipyard codifies custom software handling
  - Toolchains
  - Reproducible software generation and management flows using FireMarshal



# Outline

- Introduction to Chipyard
- **Chipyard Tooling**
- Chipyard SoC Structure and Organization
- Why Chipyard?



- ***Simple***
  - Far smaller than other commercial ISAs
- ***Clean-slate design***
  - Clear separation between user and privileged ISA
  - Avoids μarchitecture or technology-dependent features
- A ***modular*** ISA designed for ***extensibility/specialization***
  - Small standard base ISA, with multiple standard extensions
  - Sparse and variable-length instruction encoding for vast opcode space
- ***Stable***
  - Base and standard extensions are frozen
  - Additions via optional extensions, not new versions
- ***Community designed***
  - With leading industry/academic experts and software developers

Market Research Telecast

RISC-V: No longer just microcontrollers, but also supercomputers

With a stronger focus on
have a future for superco
7 hours ago

Electropages

Are we seeing the takeover from RISC-V?www.electropages
...

Electropages

European Processor Initiative Tapes Out Their First RISC-V
...

Recently, the EPI announced that it has develop
RISC-V technology and is now in the stages of h
5 days ago

ith ARM, is making
RM differ, ...

Business Korea

Semiconductor Industry: Interest Strengthening in RISC-V

Representing a potential competitor to ARM, SiFive is a startup that is developing RISC-V architecture. The acquisition price under negotiation

The Next Platform

AI Is RISC-V's Trojan Horse into the Datacenter

If the workload-specific datacenter dominates in the near term, it could be RISC-V'
1 week a

IEEE Spectrum

RISC-V Star Rises Among Chip Developers Worldwide

The upstart RISC-V chip architecture has found international traction with its customizable open-source design and lack of licensing fees. By ...
Apr 7, 2021

Tom's Hardware

RISC-V Evolving to Address Supercomputers and AI

RISC-V going after AI, ML, DL, HPC, edge, and supercomputers.
5 days ago

Data Center Knowledge

RISC-V Is On a Roll. Is It Ready to Take a Seat Alongside Intel ...

RISC-V, the emerging open source instruction set architecture for processors that have so far been used mostly as accelerators, is suddenly on ...
Mar 1, 2021

# RISC-V Ecosystem

## Software

**Open-source software:**
Gcc, binutils, glibc, Linux, BSD, LLVM, QEMU, FreeRTOS, ZephyrOS, LiteOS, SylixOS, …

**Commercial software:**
Lauterbach, Segger, IAR, Micrium, ExpressLogic, Ashling, AntMicro, Imperas, UltraSoC …

RISC-V Foundation

ISA specification | Golden Model | Compliance

## Hardware

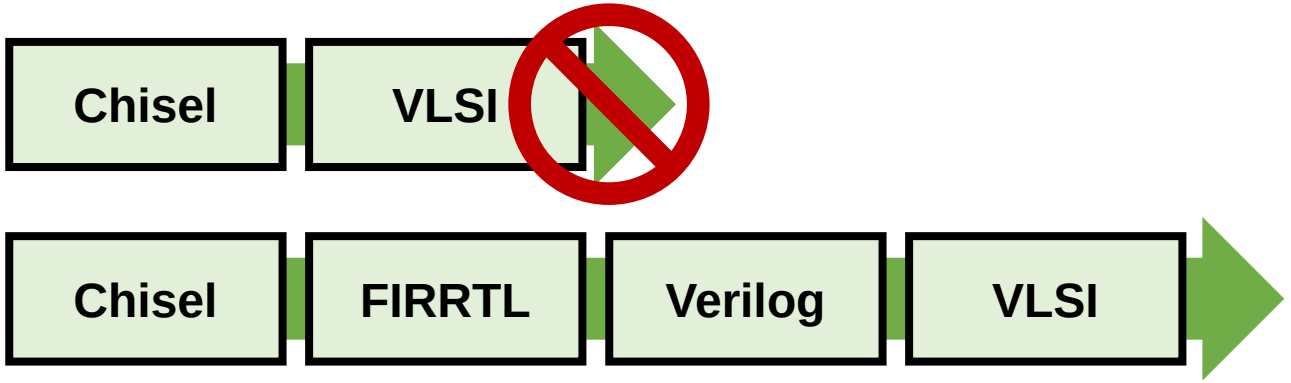
**Open-source cores:**
Rocket, BOOM, RI5CY, Ariane, PicoRV32, Piccolo, SCR1, Shakti, Swerv, Hummingbird, …

**Commercial core providers:**
Andes, Bluespec, Cloudbear, Codasip, Cortus, C-Sky, InCore, Nuclei, SiFive, Syntacore, …

**Inhouse cores:**
Nvidia, +others

# Chisel

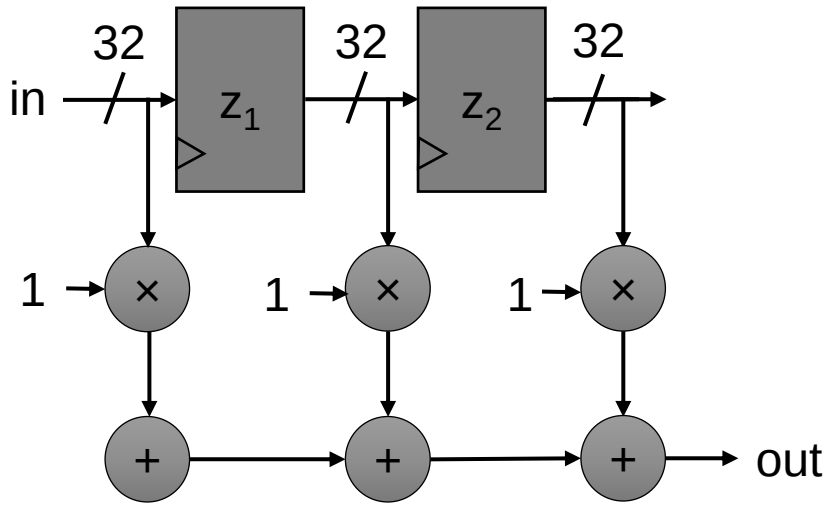
- Chisel – Hardware Construction Language built on Scala
- What Chisel **IS NOT:**
  - **NOT** Scala-to-gates
  - **NOT** HLS
  - **NOT** tool-oriented language
- What Chisel **IS:**
  - Productive language for **generating** hardware
  - Leverage **OOP/Functional programming** paradigms
  - Enables design of **parameterized generators**
  - **Designer-friendly**: low barrier-to-entry, high reward
  - **Backwards-compatible:** integrates with Verilog black-boxes



# Chisel Example

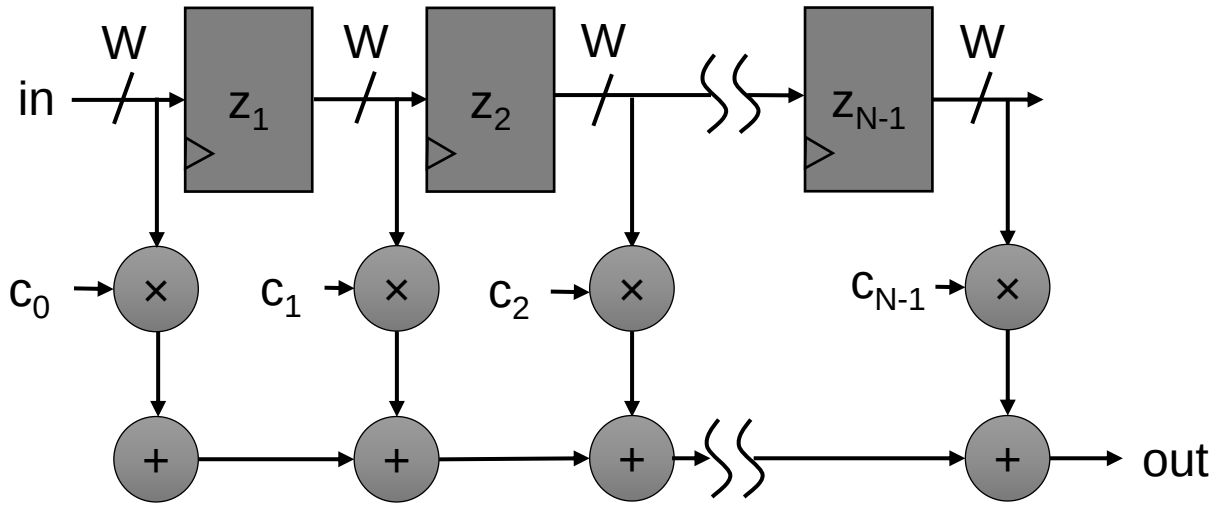
```
// 3-point moving average implemented in the
style of a FIR filter
class MovingAverage3 extends Module {
  val io = IO(new Bundle {
    val in = Input(UInt(32.W))
    val out = Output(UInt(32.W))
  })
  val z1 = RegNext(io.in)
  val z2 = RegNext(z1)

  io.out := io.in + z1 + z2
}
```



# Chisel Example

```
// Generalized FIR filter parameterized by coefficients
class FirFilter(bitWidth: Int, coeffs: Seq[Int]) extends Module {
  val io = IO(new Bundle {
    val in = Input(UInt(bitWidth.W))
    val out = Output(UInt(bitWidth.W))
  })
  val zs = Wire(Vec(coeffs.length, UInt(bitWidth.W)))
  zs(0) := io.in
  for (i <- 1 until coeffs.length) {
    zs(i) := RegNext(zs(i-1))
  }
  val products = zs zip coeffs map {
    case (z, c) => z * c.U
  }
  io.out := products.reduce(_ + _)
}
```



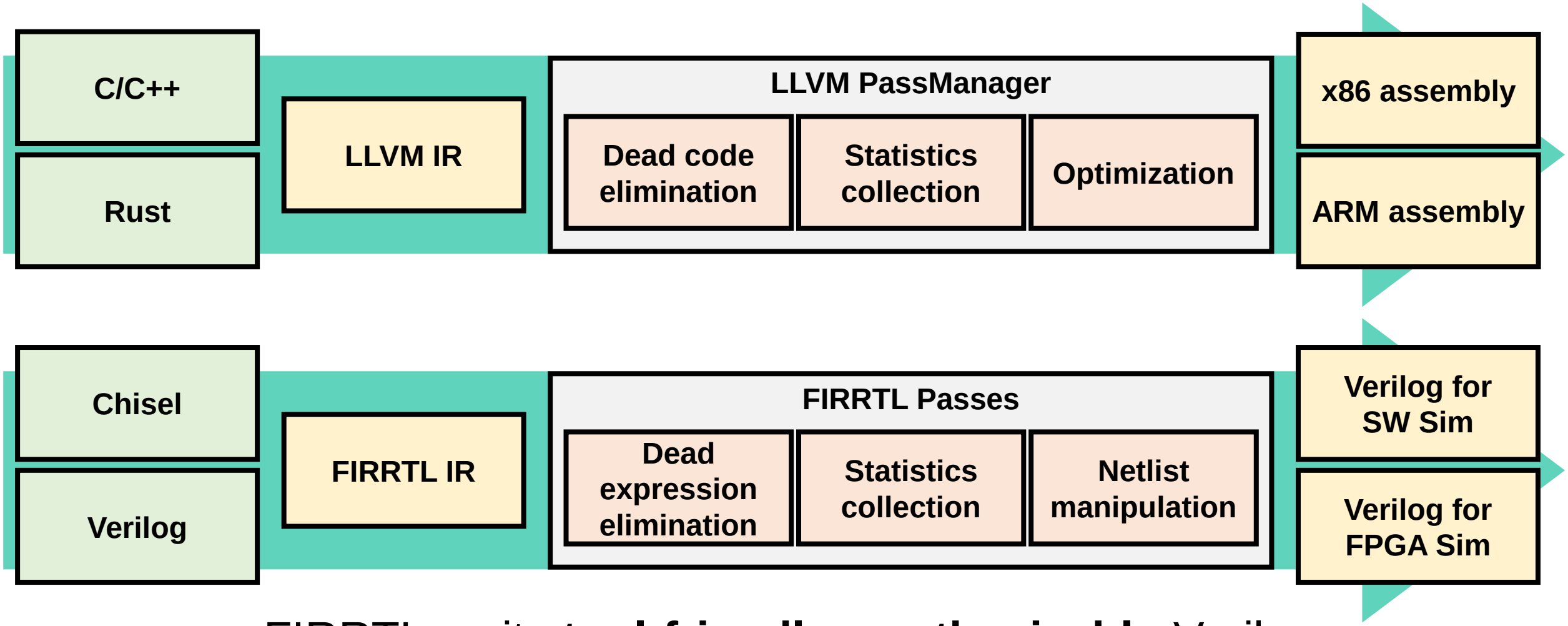
# Chisel Example

```
// Basic implementation
val basic3Filter = Module(new MovingAverage3)

// Parameterized implementation
val better3Filter = Module(new FirFilter(32, Seq(1, 1, 1)))

// Generator is reusable
val delayFilter = Module(new FirFilter(8, Seq(0, 1)))
val triangleFilter = Module(new FirFilter(8, Seq(1, 2, 3, 2, 1)))
```

# FIRRTL – LLVM for Hardware

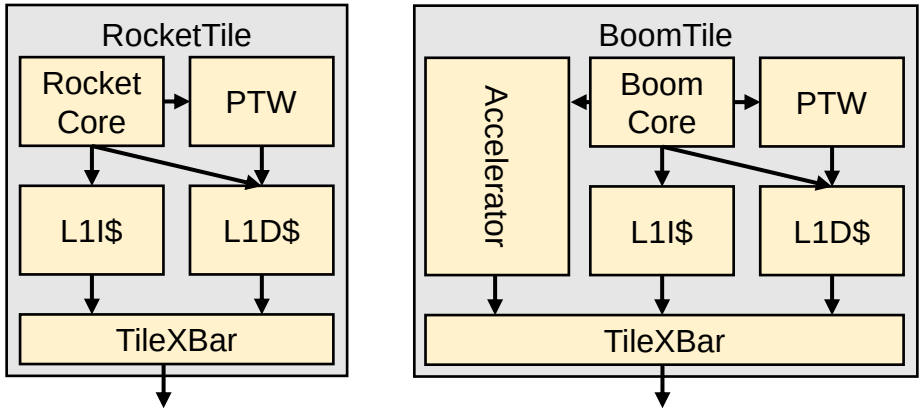



FIRRTL emits **tool-friendly, synthesizable** Verilog

# Outline

- Introduction to Chipyard
- Chipyard Tooling
- **Chipyard SoC Structure and Organization**
- Why Chipyard?

# SoC Organization: Tiles




**Tiles:** Units of replication in a multi-core SoC

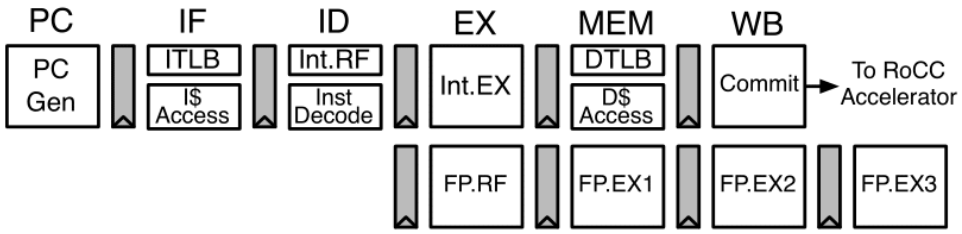
Contains:

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

Many varieties:

- Rocket “efficiency” core?
- SonicBOOM out-of-order “performance” core
- Sodor “educational” cores
- Your custom core?

# Rocket and BOOM

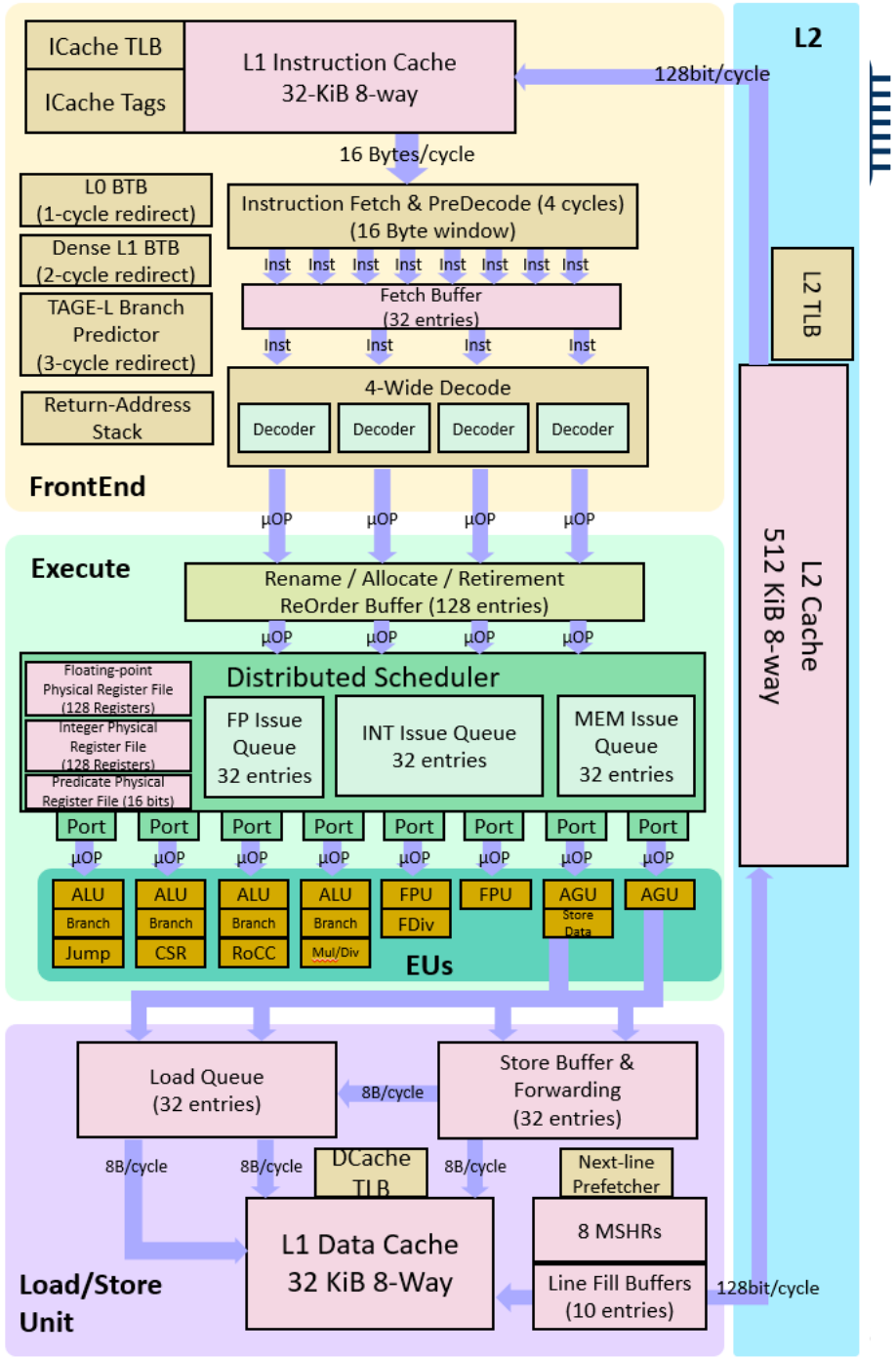



**Rocket:**

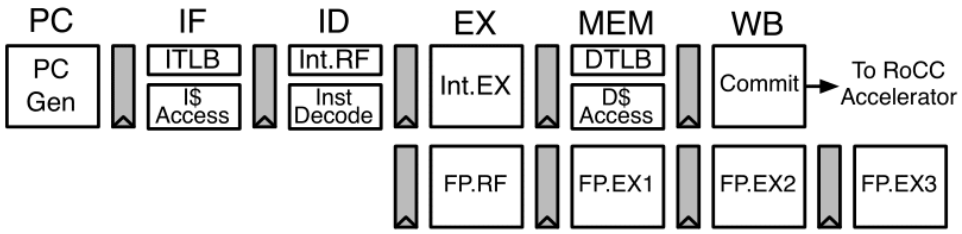
- First open-source RISC-V CPU
- In-order, single-issue RV64GC core
- Efficient design point for low-power devices

**SonicBOOM:**

- Superscalar out-of-order RISC-V CPU
- Advanced microarchitectural features to maximize IPC
- TAGE, out-of-order loads and stores, register renaming
- High-performance design point for general-purpose systems

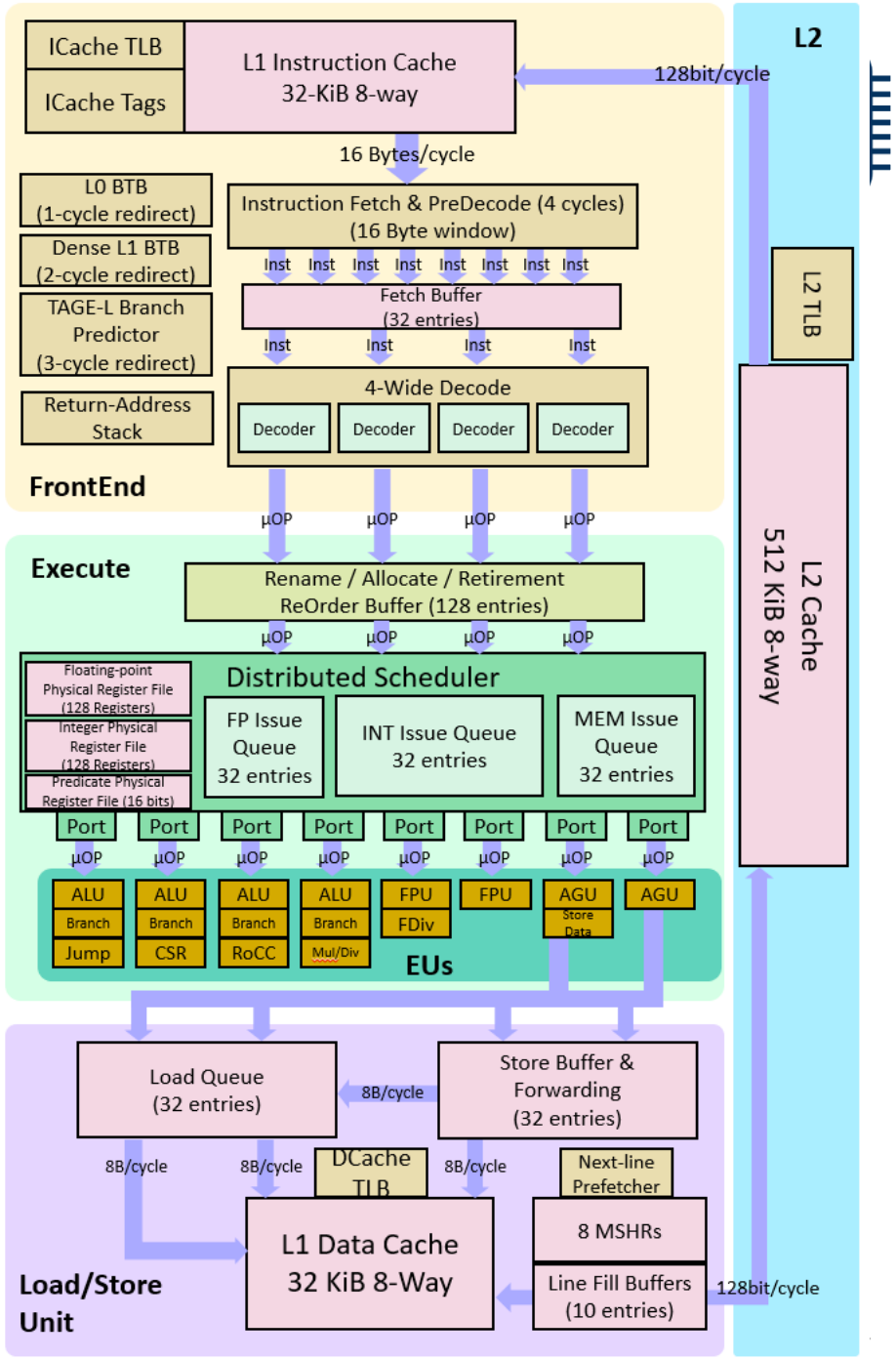


# Rocket and BOOM



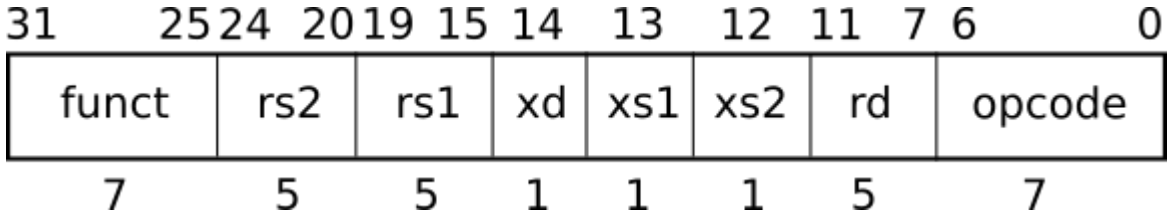

**Rocket and SonicBOOM:**

- Boots off-the-shelf RISC-V Linux distros (buildroot, Fedora, etc.)
- Supports floating point, virtual memory, supervisor mode, etc.
- Fully synthesizable, tapeout-proven
- Described in Chisel
- Fully open-sourced

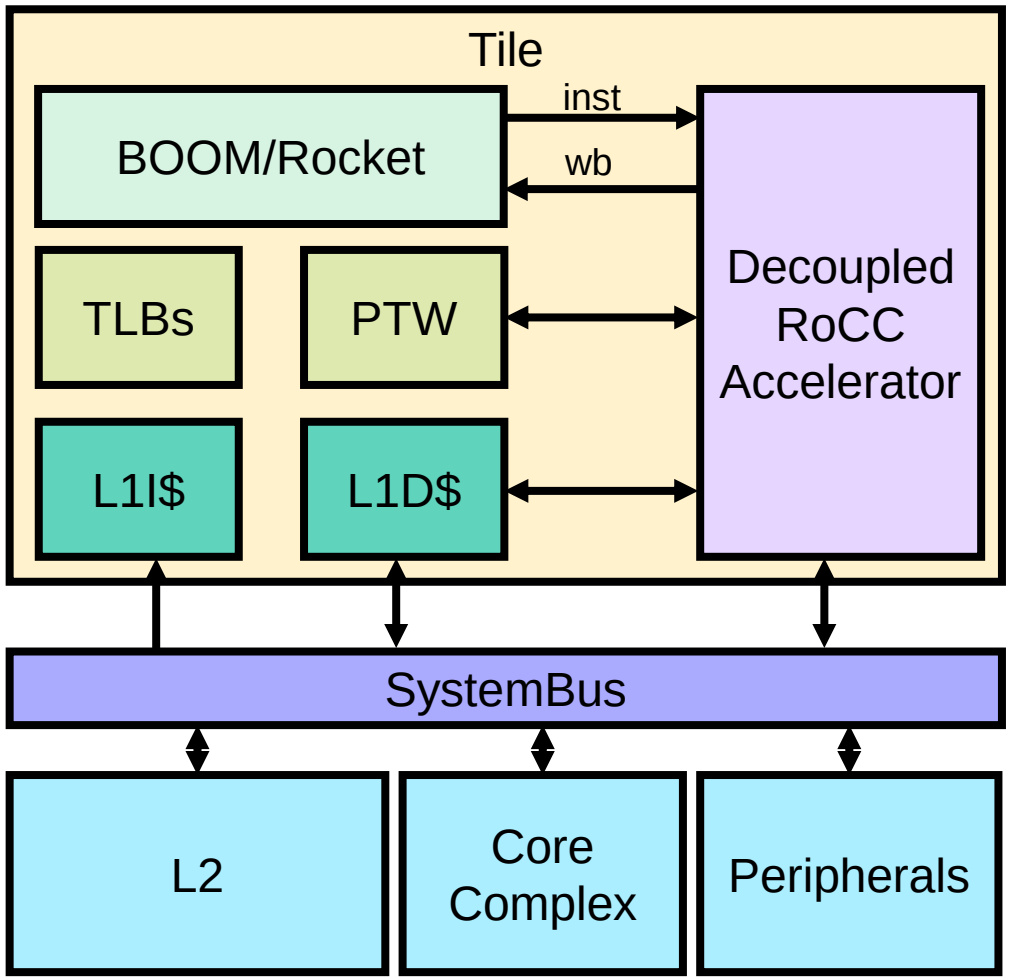


# RoCC Accelerators

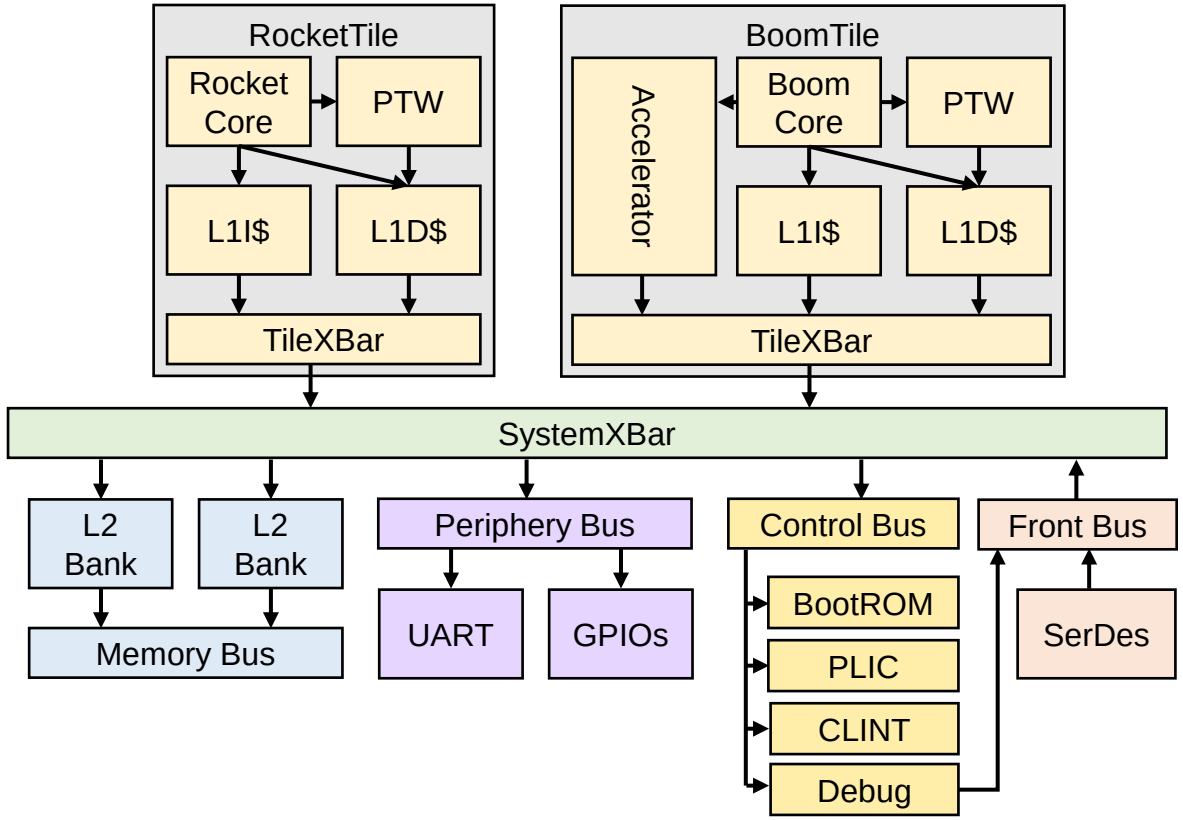
- **RoCC:** Rocket Custom Coprocessor
- Sits adjacent to Rocket or BOOM
- Execute custom RISC-V instructions for a custom extension




- Examples of RoCC accelerators in Chipyard
  - Hwacha vector accelerator
  - Gemmini matrix accelerator



# SoC Organization: Digital System



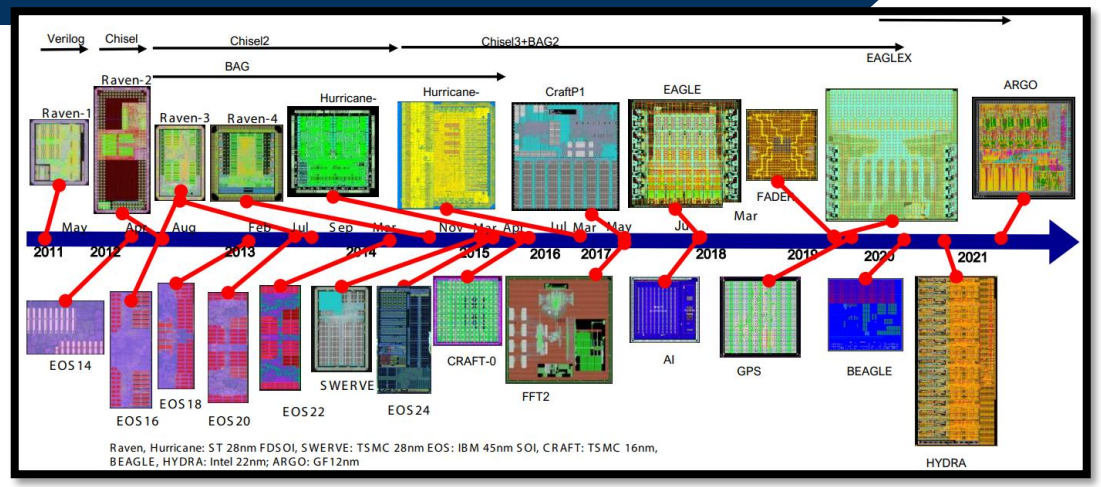

**RocketChip:** Library of digital components for an SoC subsystem

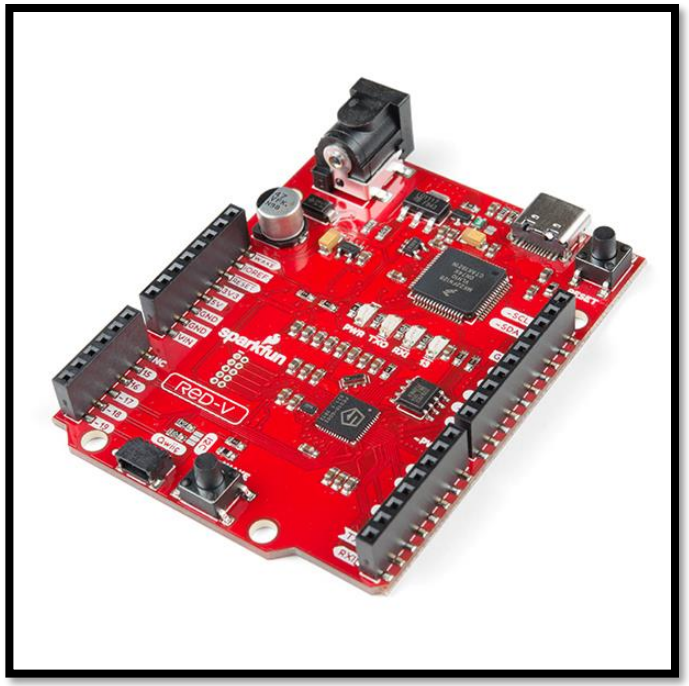
**TileLink:** Open-source chip interconnect protocol akin to AXI4

**Diplomacy:** Framework for describing connectivity of on-chip interconnects

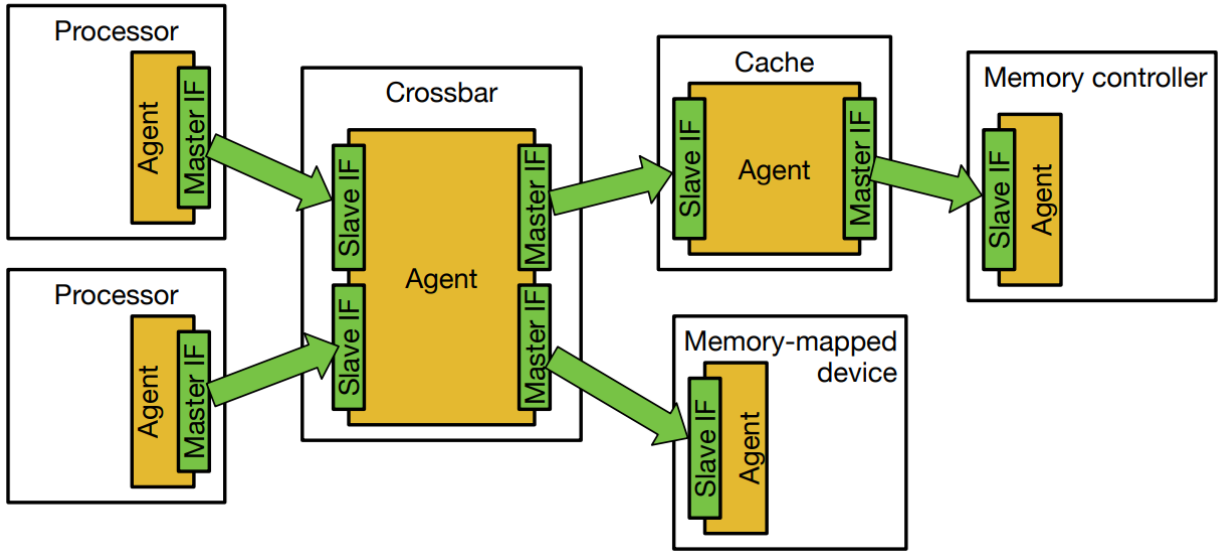
# What is Rocket Chip?

- A library of RISC-V SoC hardware components
  - Protocol converters
  - TileLink components
  - Clock crossings
- Tapeout-proven in industry and academia
- All open-sourced, built on Chisel
- Maintained by SiFive, Berkeley, ChipsAlliance



# TileLink Interconnect




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

# Diplomacy

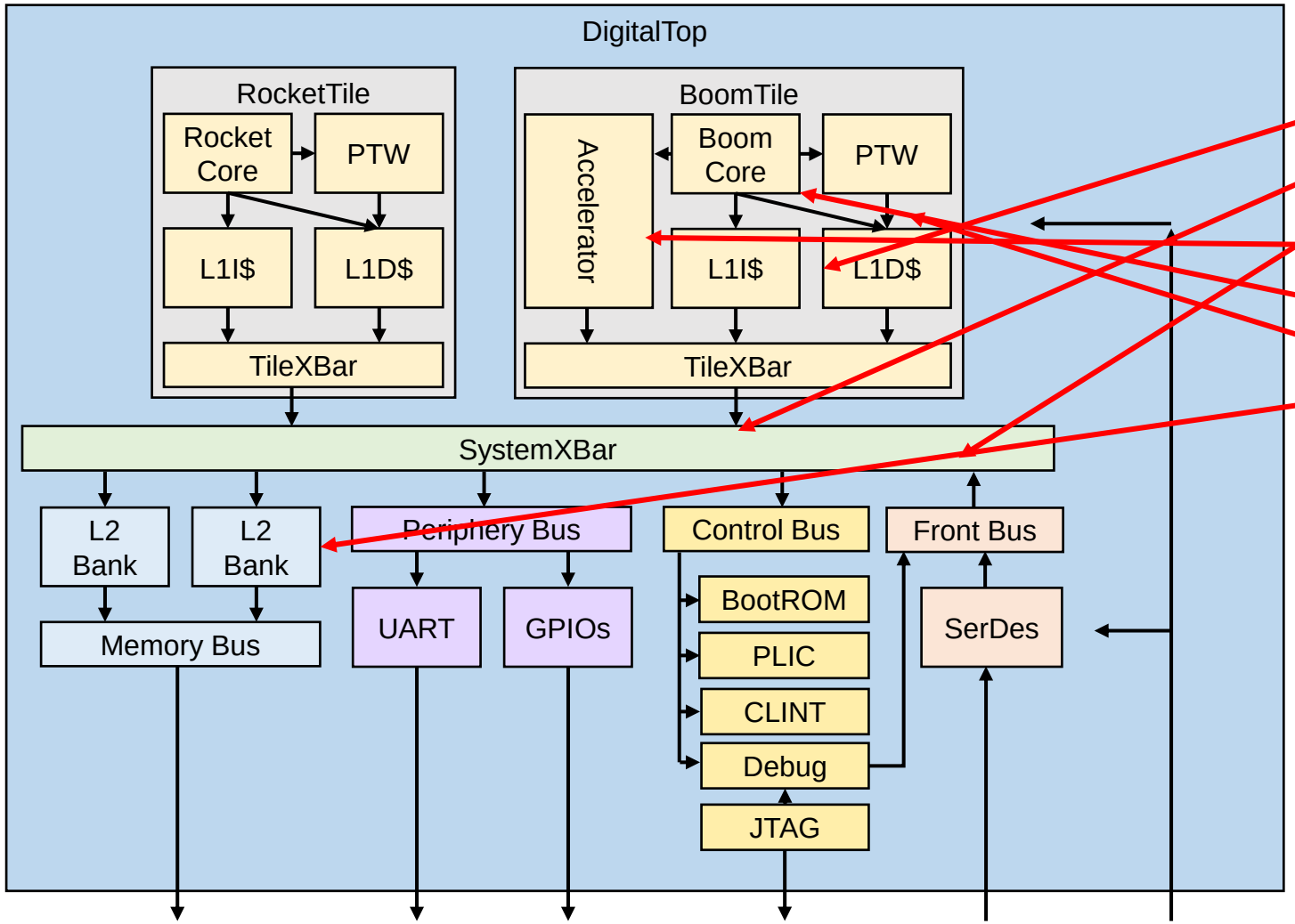
**Problem:** Interconnects are difficult to parameterize correctly

- Complex interconnect graph with many nodes
- Nodes are independently parameterized

**Diplomacy:** Framework for negotiating parameters between Chisel generators

- Graphical abstraction of interconnectivity
- Diplomatic lazy modules follow two-phase elaboration
  - **Phase one**: nodes exchange configuration information with each other and decide final parameters
  - **Phase two**: Chisel RTL elaborates using calculated parameters
- Used extensively by RocketChip TileLink generators

# Highly Parameterized Configurations



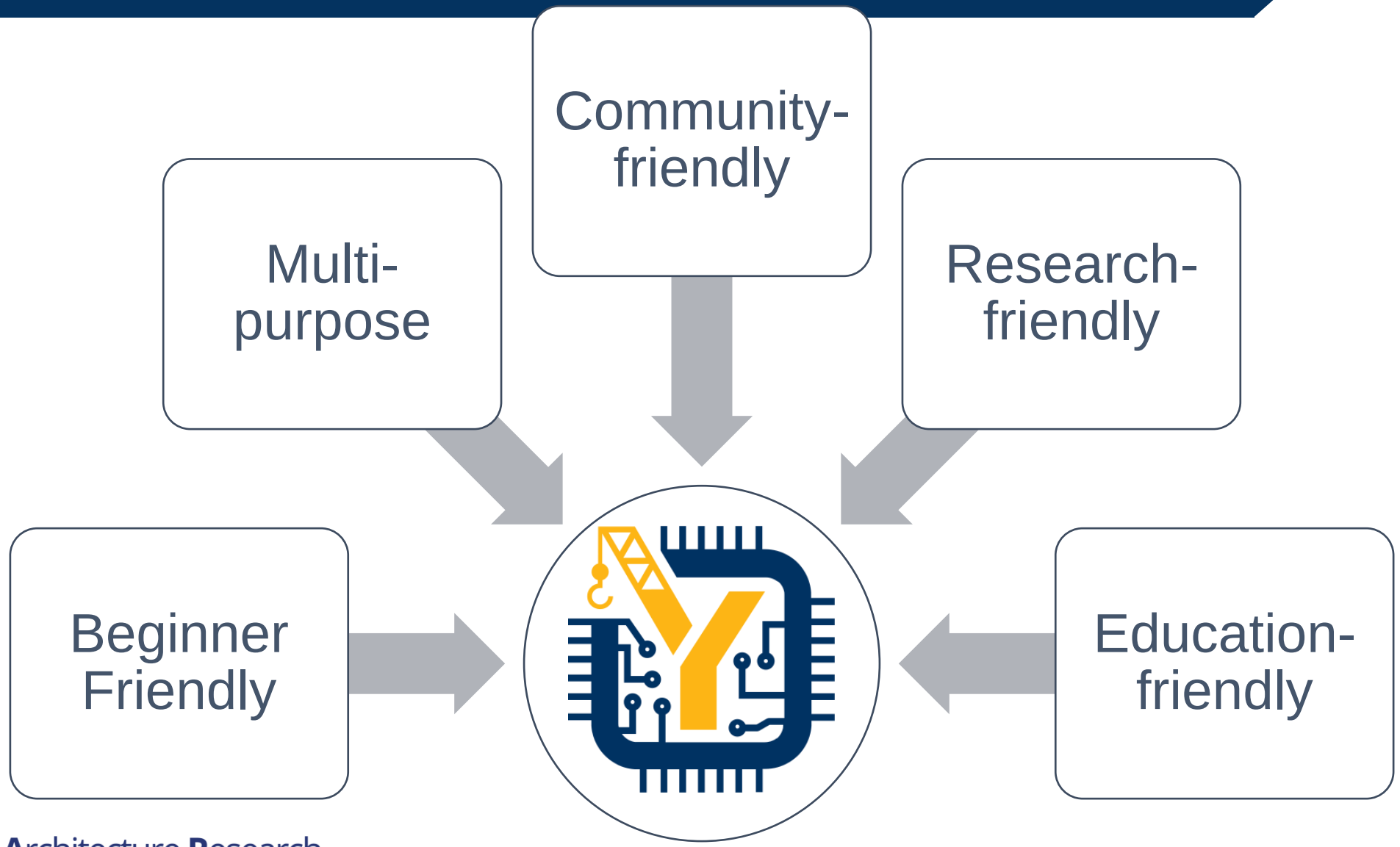

```
class CustomConfig extends Config(
  new WithL1CacheWays(4) ++
  new WithAsyncTiles ++
  new WithRingSystemBus +
  new WithFPGemmini ++
  new With3WideBooms ++
  new WithL2TLBs(512) ++
  new WithL2Banks(4) ++

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

# Outline

- Introduction to Chipyard
- Chipyard Tooling
- Chipyard SoC Structure and Organization
- **Why Chipyard?**

# Chipyard Goals

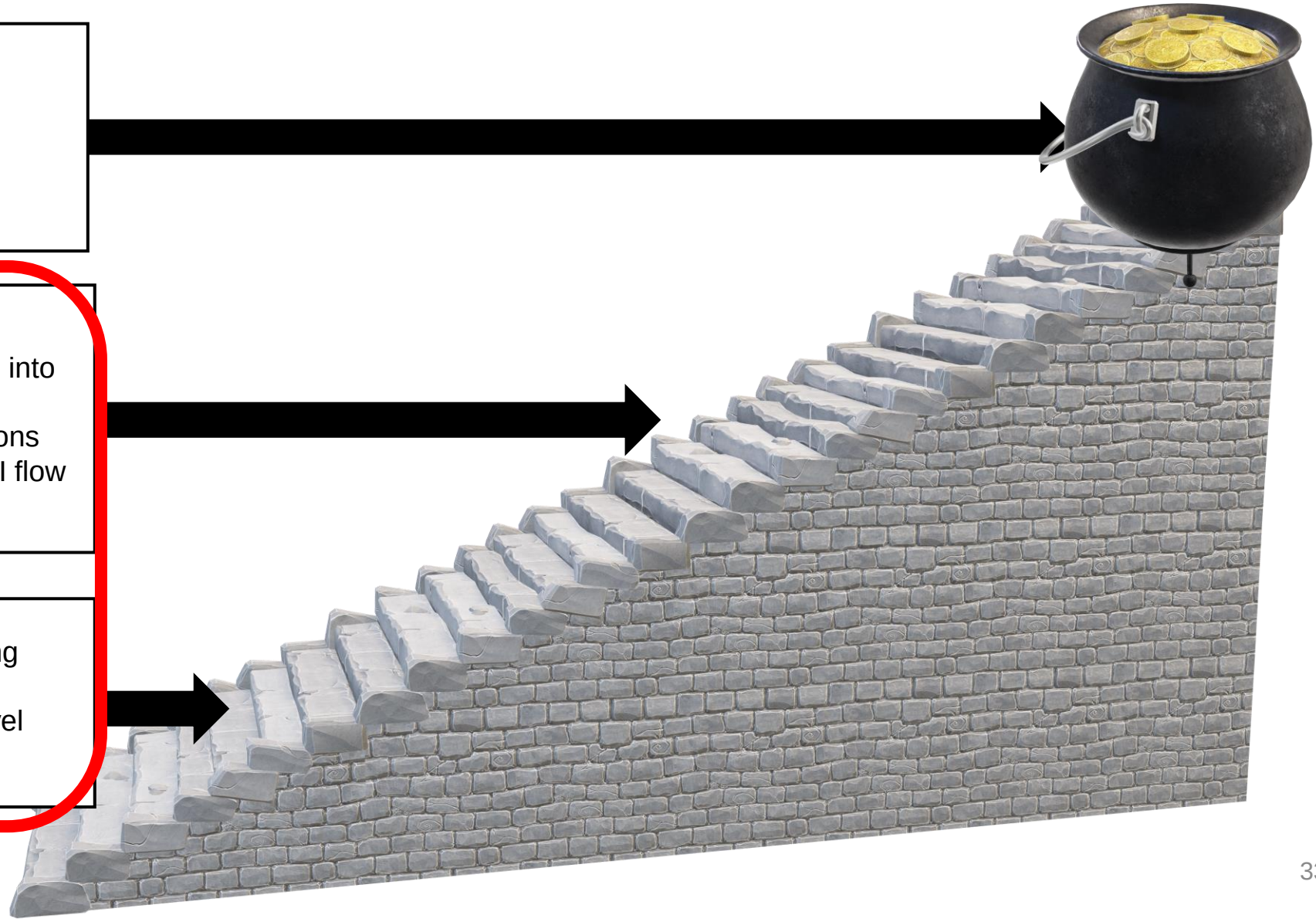


# Chipyard Learning Curve

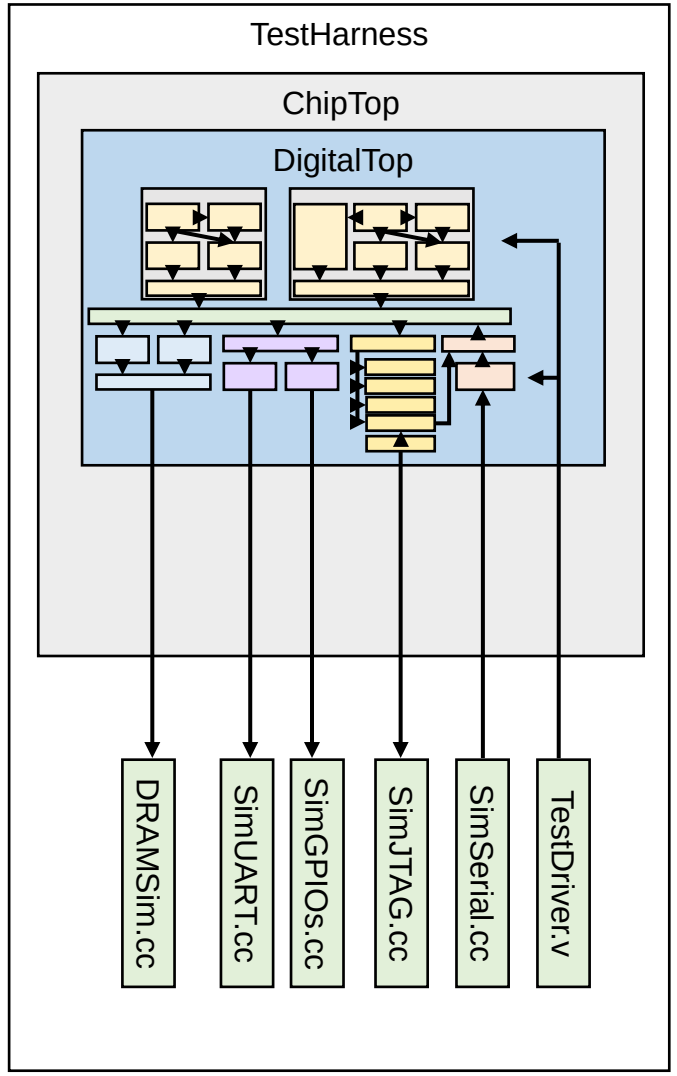
**Advanced-level**
- Configure custom IO/clocking setups
- Develop custom FireSim extensions
- Integrate and tape-out a complete SoC

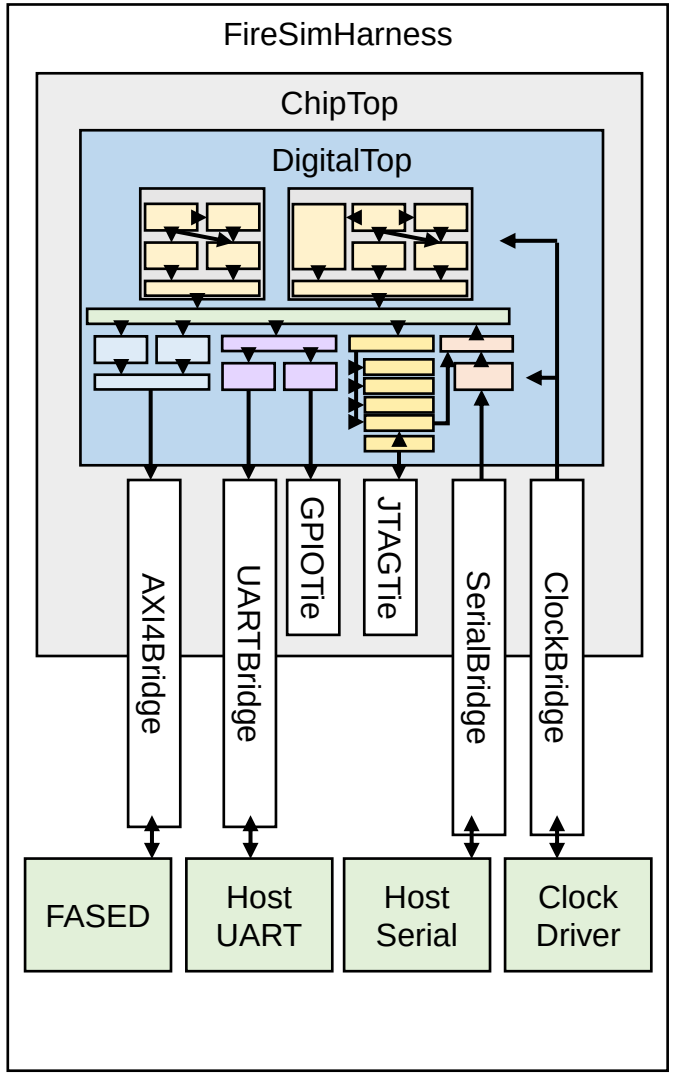
**Evaluation-level**
- Integrate or develop custom hardware IP into Chipyard
- Run FireSim FPGA-accelerated simulations
- Push a design through the Hammer VLSI flow
- Build your own system

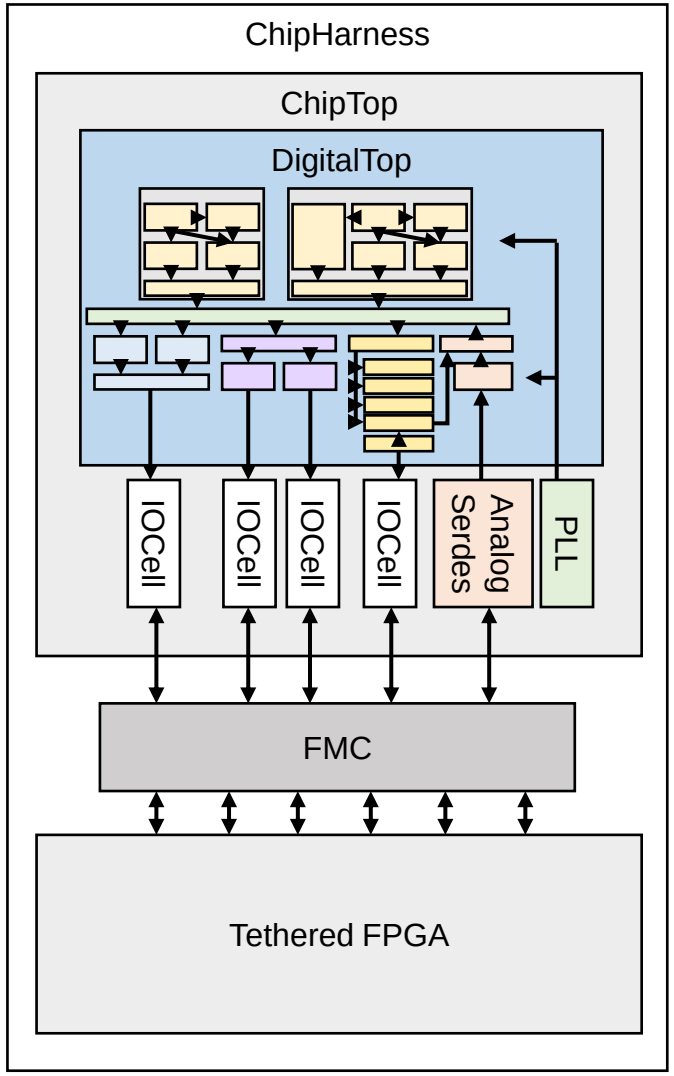
**Exploratory-level**
- Configure a custom SoC from pre-existing components
- Generate RTL, and simulate it in RTL level simulation
- Evaluate existing RISC-V designs



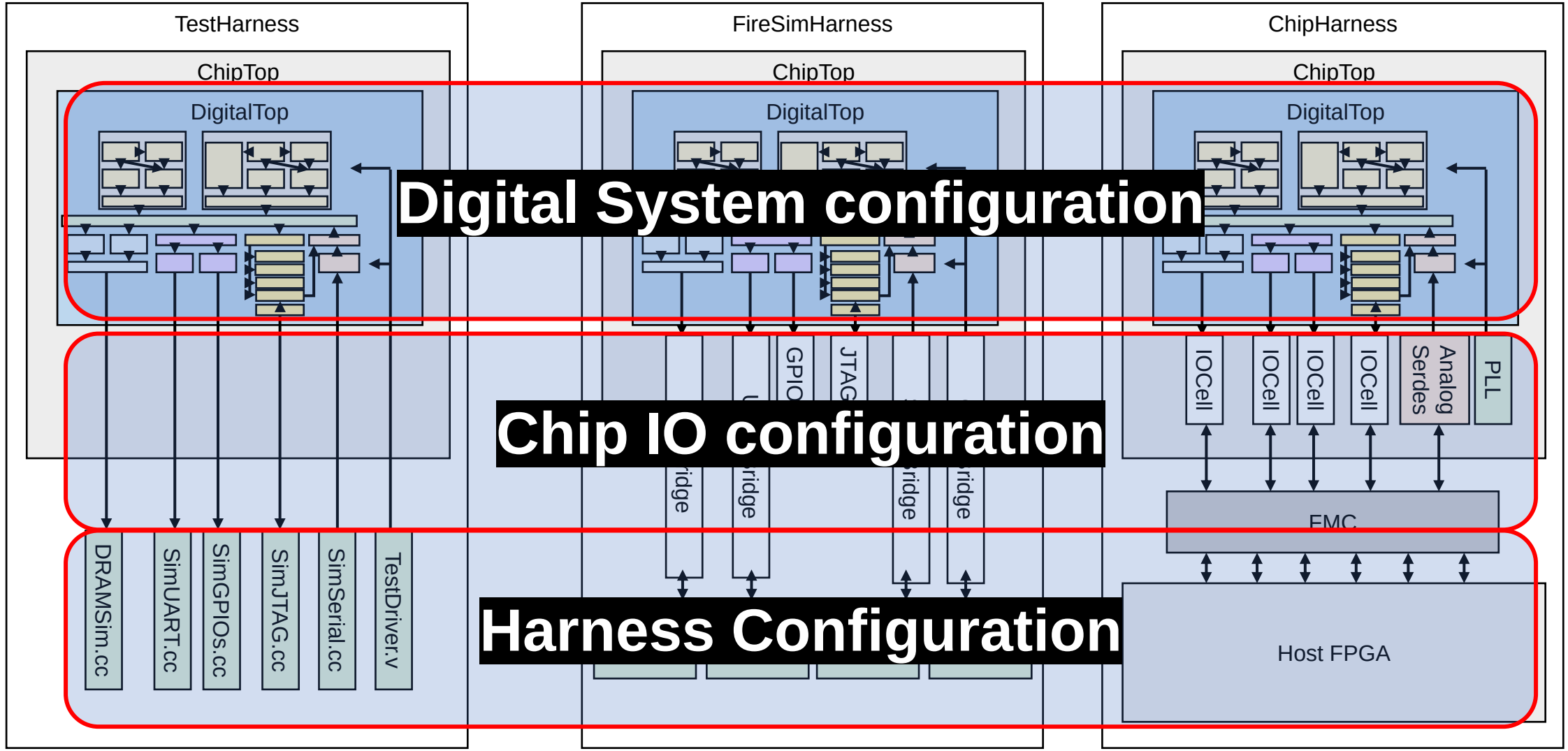
# Multipurpose



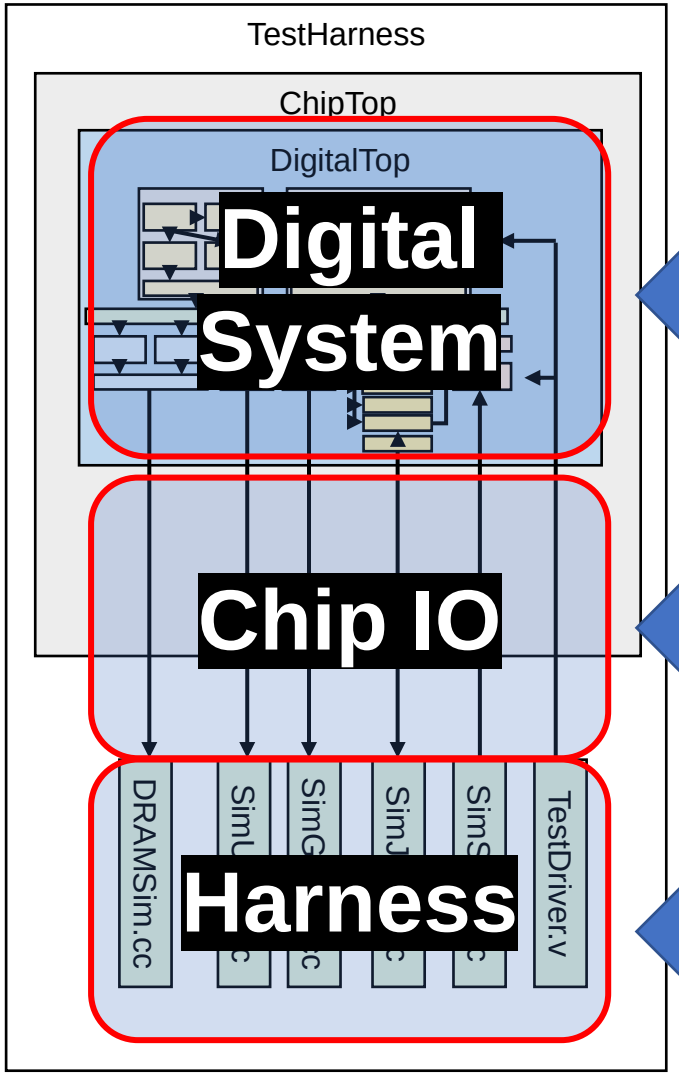



# Multipurpose



# 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

)
```

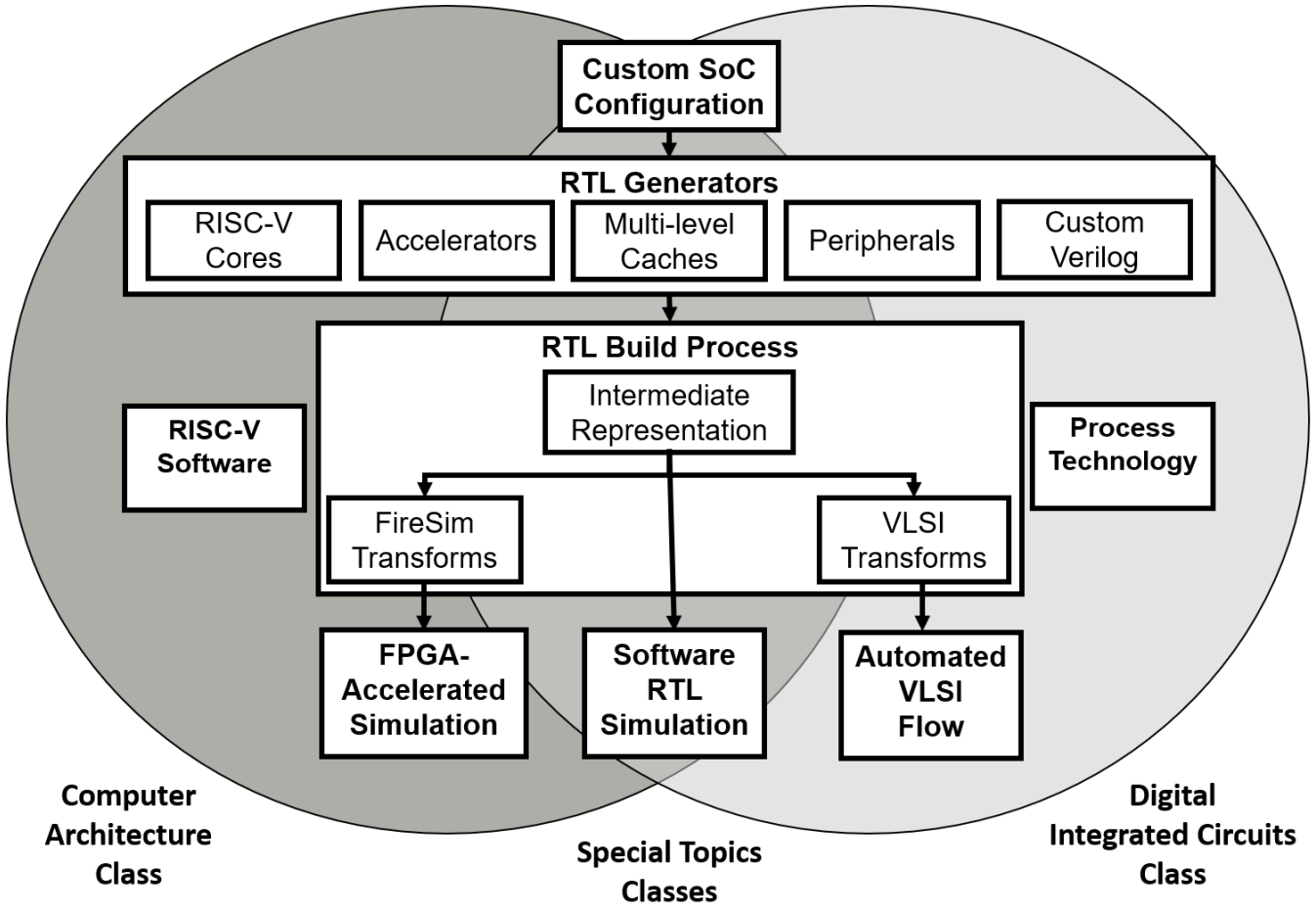
# Chipyard is Education Friendly

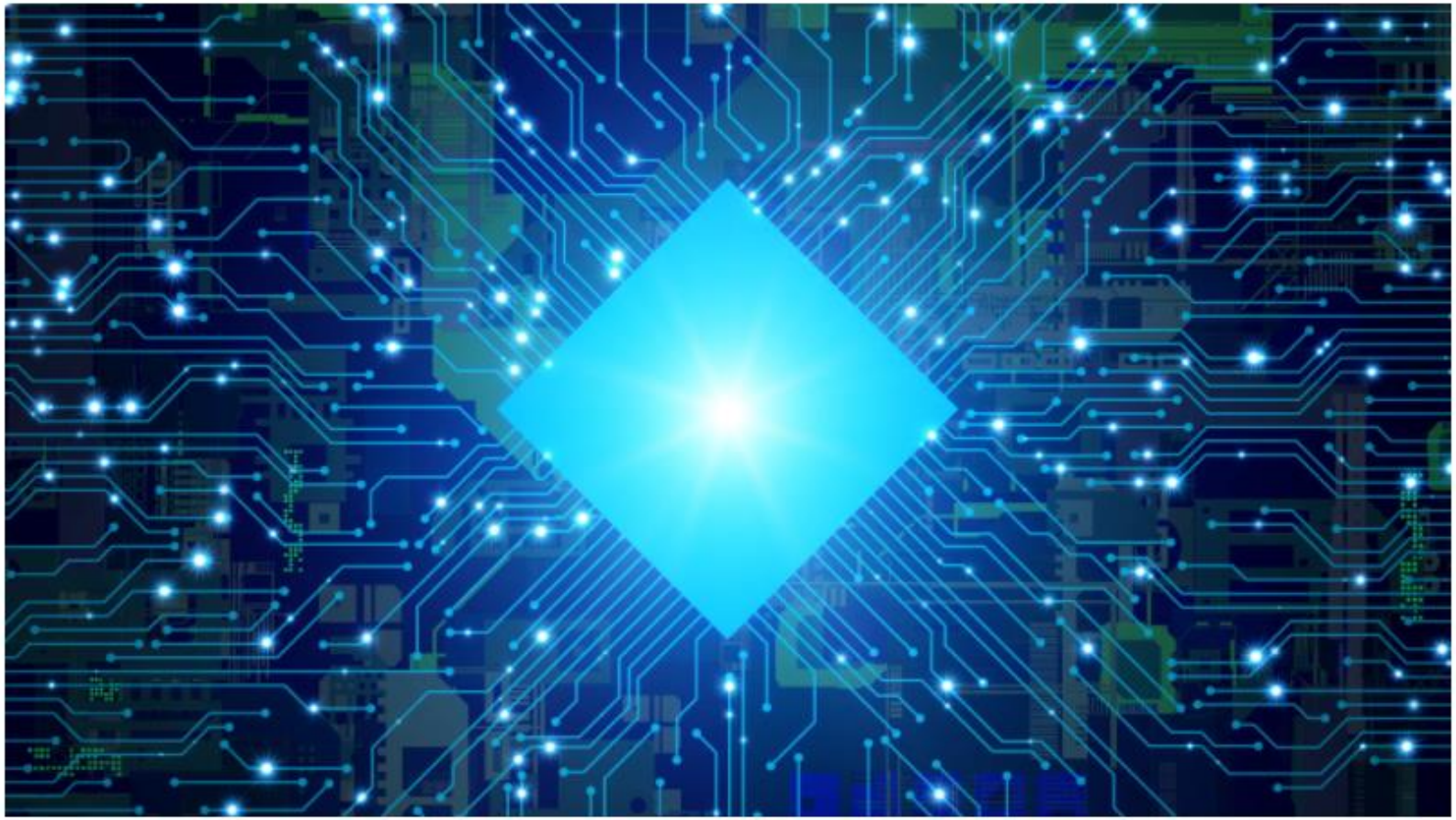
Proven in many Berkeley Architecture courses

- Hardware for Machine Learning
- Undergraduate Computer Architecture
- Graduate Computer Architecture
- Advanced Digital ICs
- Tapeout HW design course

Advantages of common shared HW framework

- Reduced ramp-up time for students
- Students learn framework once, reuse it in later courses
- Enables more advanced course projects (tapeout a chip in 1 semester)





*Berkeley Engineering students pull off novel chip design in a single semester. The class shows successful model for expanding entry into field of semiconductor design*

# Berkeley engineering students pull off novel chip design in a single semester

Class shows successful model for expanding entry into field of semiconductor design

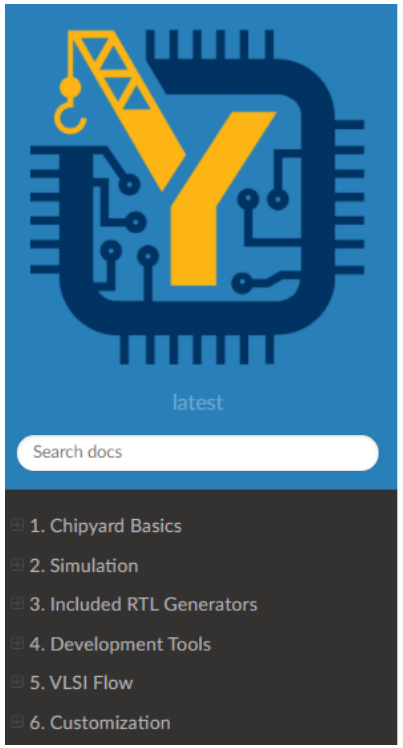
# Chipyard is Research-Friendly

- Add new accelerators/custom instructions
- Modify OS/driver/software
- Perform design-space exploration across many parameters
- Test in software and FPGA-sim before tape-out

# Chipyard is Community-Friendly

**Documentation:**

- https://chipyard.readthedocs.io/en/dev/
- 133 pages
- Most of today's tutorial content is covered there



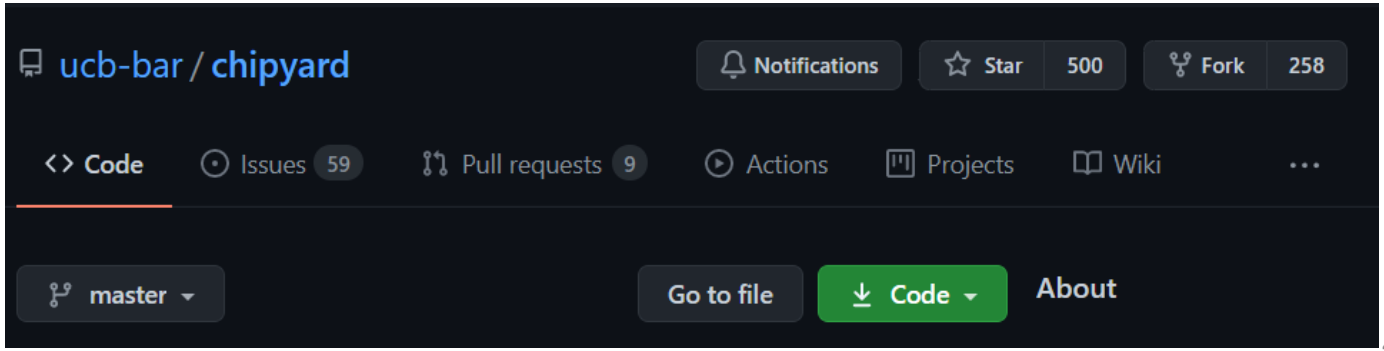


**Mailing List:**

- google.com/forum/#!forum/chipyard

**Open-sourced:**

- All code is hosted on GitHub
- Issues, feature-requests, PRs are welcomed



# Conclusion

**Chipyard:** An open, extensible research and design platform for RISC-V SoCs

- Unified framework of parameterized generators
- One-stop-shop for RISC-V SoC design exploration
- Supports variety of flows for multiple use cases
- Open-sourced, community and research-friendly

**Questions?**