Building Custom RISC-V SoCs in Chipyard

Abraham Gonzalez

UC Berkeley

abe.gonzalez@berkeley.edu

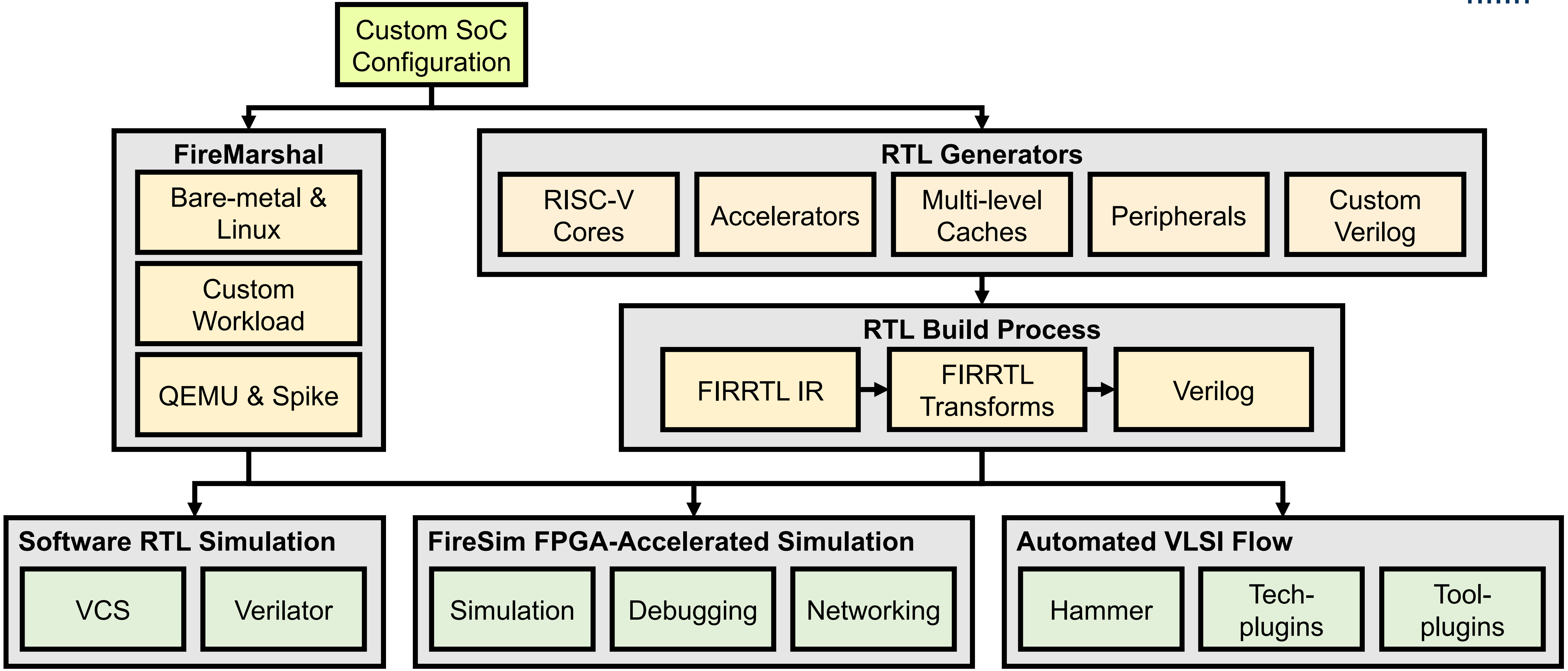
Berkeley Architecture Research

CHIPYARD

Goals

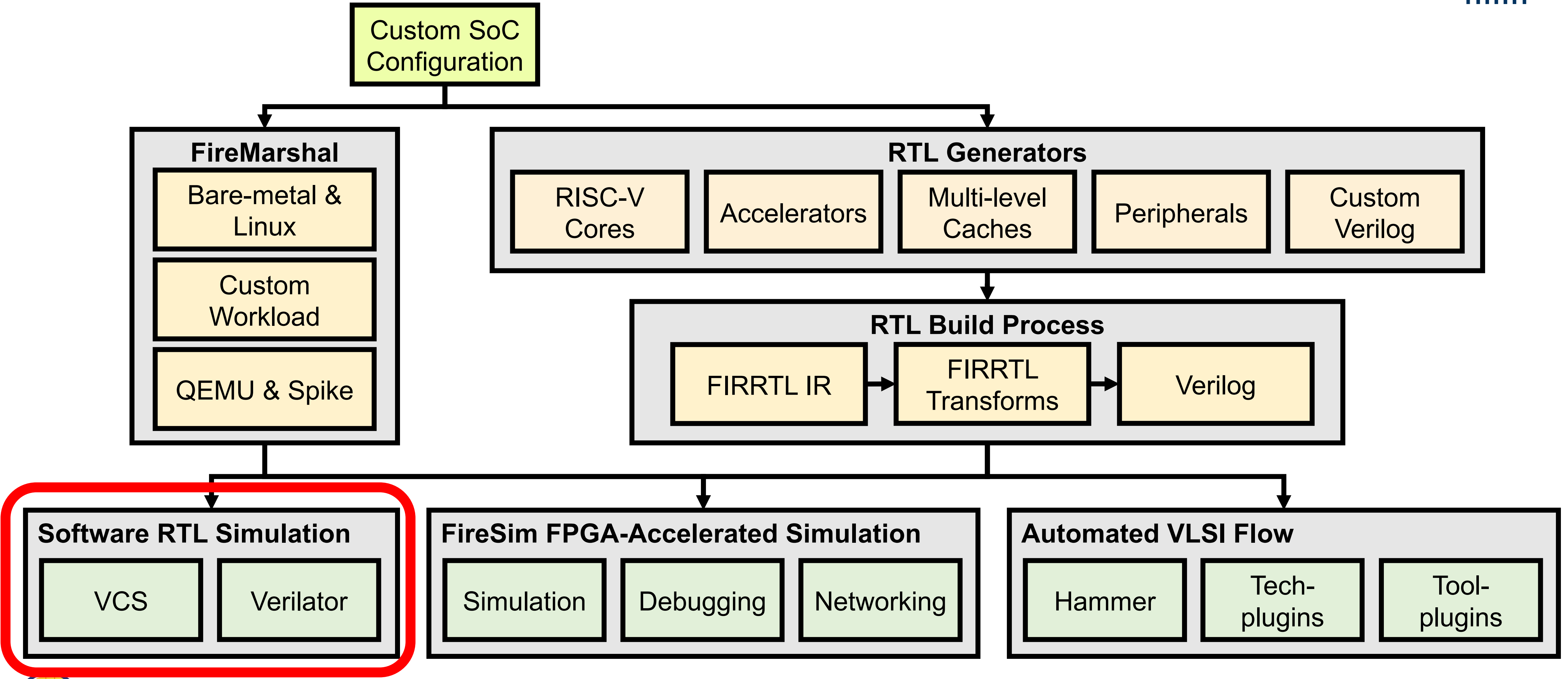
- Get the basics of modifying a configuration
    - Create a heterogeneous BOOM and Rocket RISC-V SoC
- Learn how to generate Verilog for an SoC
- Learn how to run Verilator RTL simulations
- Case Study: Integrating a SHA3 accelerator into a Chipyard SoC!
    - Add a SHA3 accelerator to Chipyard
    - Add the accelerator to a configuration!
    - Get Verilog and run Verilator simulations

Tutorial Roadmap



Berkeley Architecture Research

Tutorial Roadmap



Berkeley Architecture Research

Getting Started

Interactive Section!

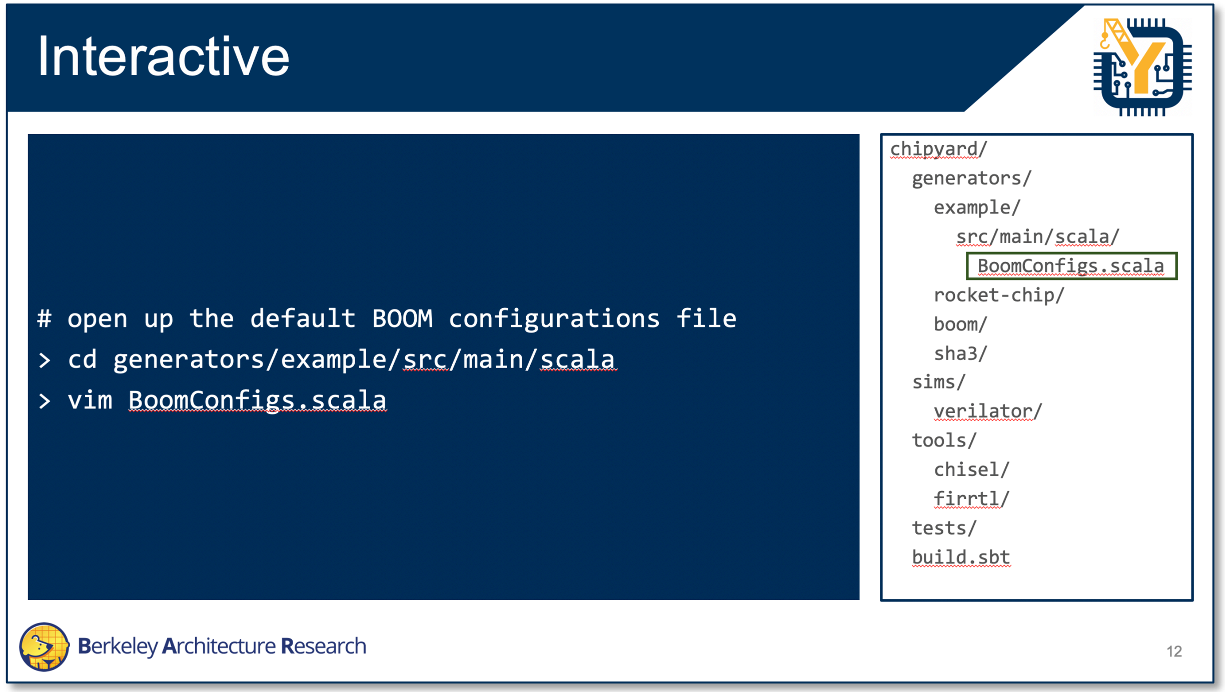
Having trouble? Raise a hand and someone will come help you.

You can find these slides on

https://fires.im/micro-2019-tutorial/

Berkeley Architecture Research

How things will work

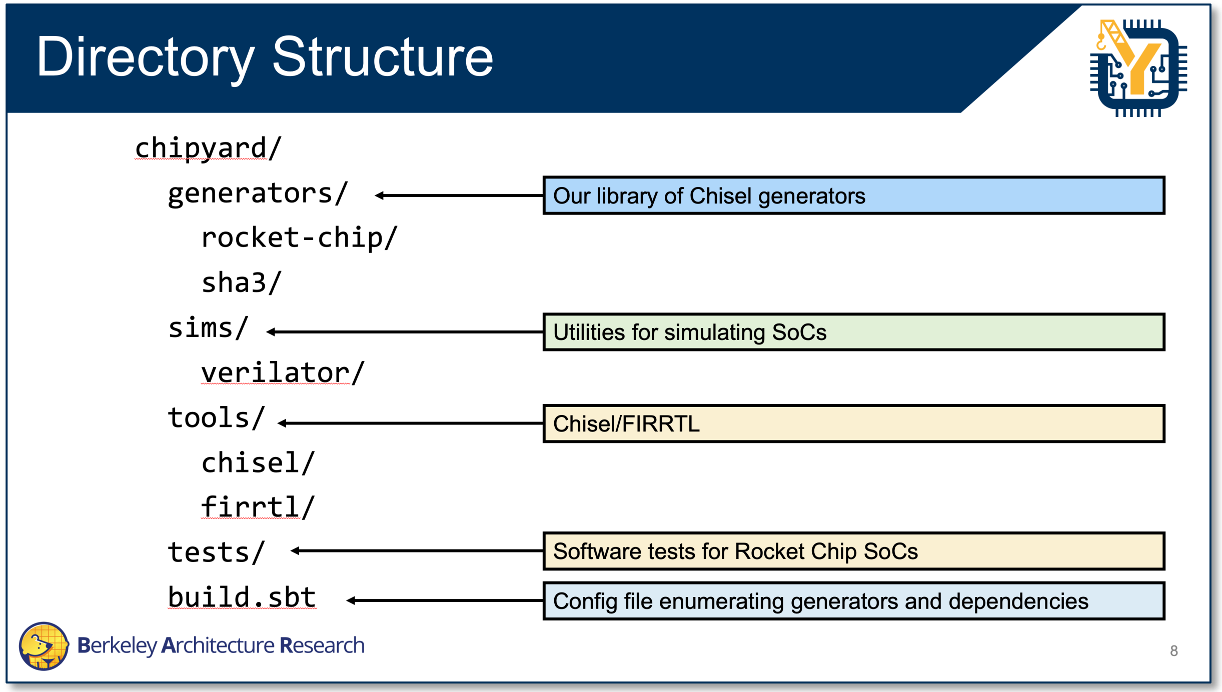


Interactive

```
# open up the default BOOM configurations file
> cd generators/example/src/main/scala
> vim BoomConfigs.scala
```

```
chipyard/
  generators/
    example/
      src/main/scala/
        BoomConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research 12



Directory Structure

```
chipyard/
  generators/    <- Our library of Chisel generators
    rocket-chip/
    sha3/
  sims/          <- Utilities for simulating SoCs
    verilator/
  tools/         <- Chisel/FIRRTL
    chisel/
    firrtl/
  tests/         <- Software tests for Rocket Chip SoCs
  build.sbt      <- Config file enumerating generators and dependencies
```

Berkeley Architecture Research 8

Interactive Slide

"Follow Along"

Explanation Slide

"What's happening?"

How things will work

```
# command 1
> echo “Chipyard Rules!”

# command 2
> do_this arg1 arg2
```

Terminal Section

```
// SOME COMMENT HERE
class SmallBoomConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

Inside-a-File Section

Interactive

```
# start a tmux session
> tmux new –s soc

# return to chipyard
> cd ~/chipyard-morning/
> ls
```

Berkeley Architecture Research

Directory Structure

```
chipyard-morning/
  generators/        <──── Our library of Chisel generators
    rocket-chip/
    sha3/
  sims/              <──── Utilities for simulating SoCs
    verilator/
  software/          <──── Utilities for building RISC-V software
    firemarshal/
  tools/             <──── Chisel/FIRRTL
    chisel/
    firrtl/
build.sbt            <──── Config file enumerating generators and dependencies
```

Build and simulate a heterogeneous BOOM + Rocket SoC

Berkeley Architecture Research

Default Chipyard Project

- You need a top-level project that combines all the generators wanted
  - Want Rocket Chip, BOOM, SHA3 accelerator, etc…
- In our case we are using the `` `example` `` project
  - Located in `` `generators/example` ``
- This holds things such as
  - Test Harness code
  - Top-level module (matches the top-level of the DUT)
  - SoC configurations
- Most of the work will be done in this project

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

BOOM Integration

- BOOM source code already added
  - Located in `` `generators/boom` ``
- Already has default configurations
  - Located in `` `generators/example/src/main/scala/BoomConfigs.scala` ``

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Interactive

```
# open up the default boom configurations file
> cd generators/example/src/main/scala
> less BoomConfigs.scala
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        BoomConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Interactive

```
# open up the default boom configurations file
> cd generators/example/src/main/scala
> less BoomConfigs.scala
```

Reminder: use q to quit less

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        BoomConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Configuring a SoC with BOOM

```
class SmallBoomConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

- Bottom-to-top (right-to-left) config hierarchy
- Configs can read/override other configs

```
BoomTilesKey   => Seq()
RocketTilesKey => Seq()
BankedL2Key    => Module(new TLBroadcast)
BootROMParams  => BootROMParams()
BuildTop       => None
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        BoomConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Configuring a SoC with BOOM

```
class SmallBoomConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

- Bottom-to-top (right-to-left) config hierarchy
- Configs can read/override other configs

```
BoomTilesKey    => Seq()
RocketTilesKey => Seq()
BankedL2Key     => Module(new TLBroadcast)
BootROMParams   => BootROMParams()
BuildTop        => None
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        BoomConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Configuring a SoC with BOOM

```
class SmallBoomConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

- Bottom-to-top (right-to-left) config hierarchy
- Configs can read/override other configs

```
BoomTilesKey   => Seq(BoomParams())
RocketTilesKey => Seq()
BankedL2Key    => Module(new TLBroadcast)
BootROMParams  => BootROMParams()
BuildTop       => None
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        BoomConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Configuring a SoC with BOOM

```
class SmallBoomConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

- Bottom-to-top (right-to-left) config hierarchy
- Configs can read/override other configs

```
BoomTilesKey   => Seq(BoomParams(Small))
RocketTilesKey => Seq()
BankedL2Key    => Module(new TLBroadcast)
BootROMParams  => BootROMParams()
BuildTop       => None
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        BoomConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Configuring a SoC with BOOM

```
class SmallBoomConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

- Bottom-to-top (right-to-left) config hierarchy
- Configs can read/override other configs

```
BoomTilesKey    => Seq(BoomParams(Small))
RocketTilesKey  => Seq()
BankedL2Key     => Module(new InclusiveCache)
BootROMParams   => BootROMParams()
BuildTop        => None
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        BoomConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Configuring a SoC with BOOM

```
class SmallBoomConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

- Bottom-to-top (right-to-left) config hierarchy
- Configs can read/override other configs

```
BoomTilesKey    => Seq(BoomParams(Small))
RocketTilesKey  => Seq()
BankedL2Key     => Module(new InclusiveCache)
BootROMParams   => BootROMParams("../bootrom.rv64.img")
BuildTop        => None
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        BoomConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Configuring a SoC with BOOM

```
class SmallBoomConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

- Bottom-to-top (right-to-left) config hierarchy
- Configs can read/override other configs

```
BoomTilesKey    => Seq(BoomParams(Small))
RocketTilesKey  => Seq()
BankedL2Key     => Module(new InclusiveCache)
BootROMParams   => BootROMParams("../bootrom.rv64.img")
BuildTop        => Module(new Top)
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        BoomConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Interactive

```
# open heterogeneous configs file
> vim HeteroConfigs.scala
```

Feel free to use another editor! These slides assume vim.

```
class TutorialSmallBoomConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        HeteroConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Interactive

```
# open heterogeneous configs file
> vim HeteroConfigs.scala
```

```
class TutorialSmallBoomConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

Modify

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        HeteroConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Interactive

```
# open heterogeneous configs file
> vim HeteroConfigs.scala
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        HeteroConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
```

```
class TutorialSmallBoomConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

Modify and add

```
class SmallBoomAndRocketConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithRenumberHarts ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.subsystem.WithNBigCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

VIM Reminders:
use i to insert text
use ESC to exit inserting text

Interactive

```
# open heterogeneous configs file
> vim HeteroConfigs.scala
```

```
class TutorialSmallBoomConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

Modify and add

```
class SmallBoomAndRocketConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithRenumberHarts ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.subsystem.WithNBigCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        HeteroConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
```

VIM Reminders:
use :q to quit
use :w to save
use :wq to save and quit

Interactive

```
# navigate to the verilator
> cd ~/chipyard-morning/sims/verilator

# start the verilator rtl simulator build
> make CONFIG=SmallBoomAndRocketConfig –j16

# this will take a while!
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Behind the Scenes: Config

```
class SmallBoomAndRocketConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithRenumberHarts ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.subsystem.WithNBigCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

Adds a “Big” Rocket core to the SoC

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        HeteroConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Behind the Scenes: Config

```
class SmallBoomAndRocketConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new boom.common.WithRenumberHarts ++
  new boom.common.WithSmallBooms ++
  new boom.common.WithNBoomCores(1) ++
  new freechips.rocketchip.subsystem.WithNBigCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

Renumbers the core ID's so that there are no duplicates

Adds a "Big" Rocket core to the SoC

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        HeteroConfigs.scala
    rocket-chip/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Behind the Scenes: Make

- `make` is the top-level build system to combine everything
  - Invokes the Scala Build Tool (sbt)
    - Used in Chisel, FIRRTL, and other Chipyard tools
  - Invokes all of the simulator builds (VCS and Verilator)
  - Automatically keeps track of file dependencies for you!
- Powerful `make` commands
  - Can just `make verilog` for Verilog only
  - Can run pre-added or unique tests
    - `make CONFIG=<YOUR CONFIG> BINARY=<YOUR BINARY> run-binary`
    - `make CONFIG=<YOUR CONFIG> run-bmark-tests`
  - Keeps all outputs organized based of a unique name of the SoC
    - `<PROJECT>.<DUT MODULE>.<CONFIG>`
    - Ex. `example.TestHarness.SmallBoomAndRocketConfig`

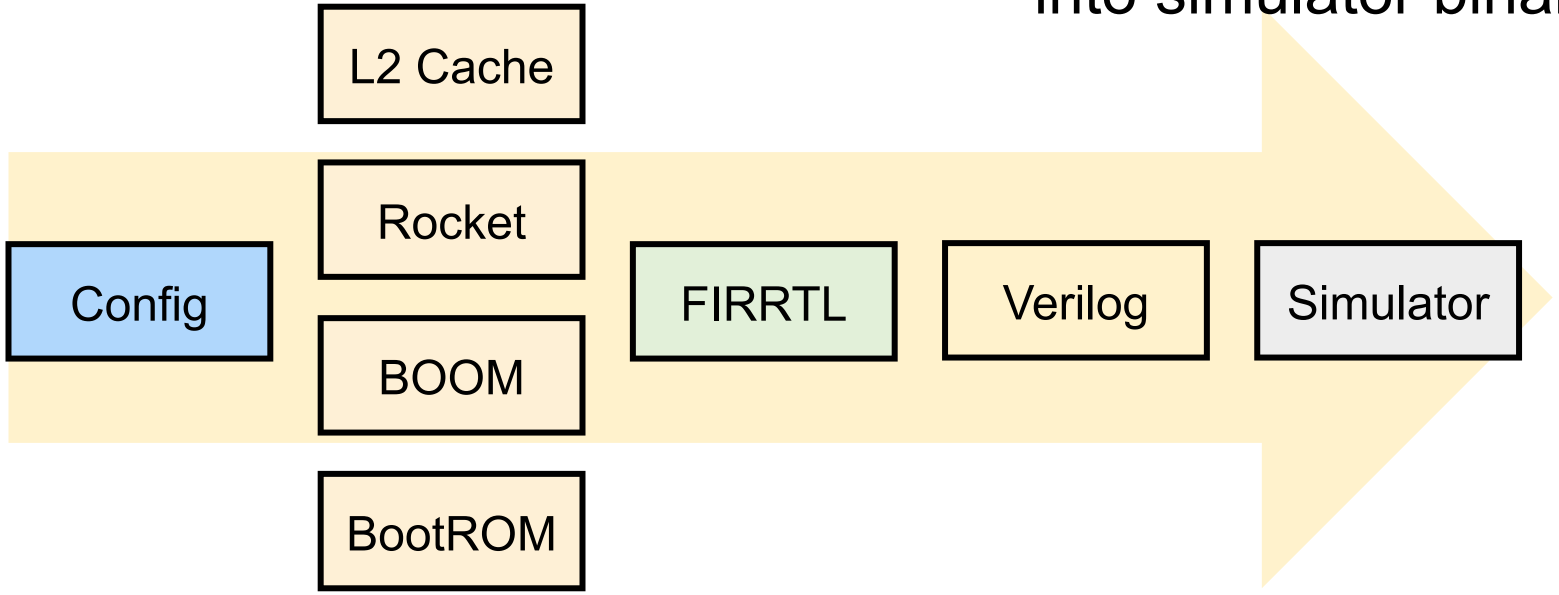
Building a SW Simulator

1. Configs parameterize Chisel generators

2. Chisel elaborates into FIRRTL

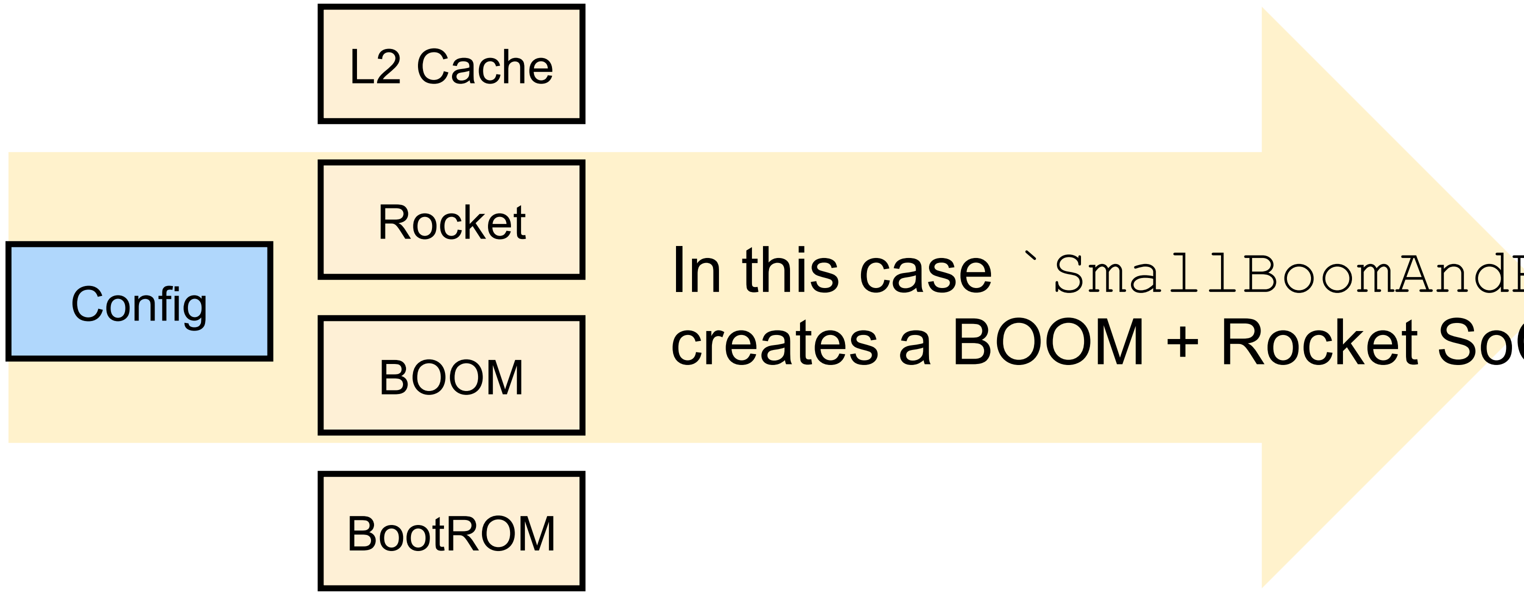
3. FIRRTL elaborates into Verilog

4. Verilator compiles Verilog into simulator binary



Building a SW Simulator

1. Configs parameterize Chisel generators

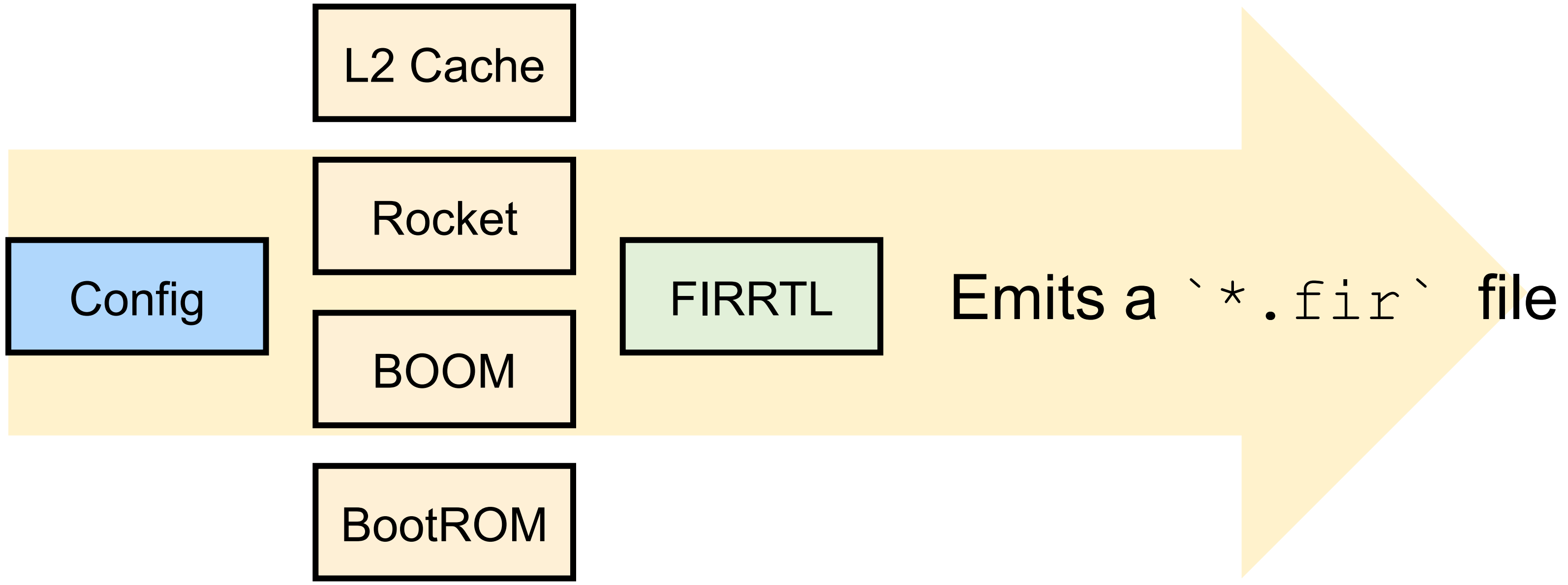


In this case `SmallBoomAndRocketConfig` creates a BOOM + Rocket SoC

Building a SW Simulator

1. Configs parameterize Chisel generators

2. Chisel elaborates into FIRRTL

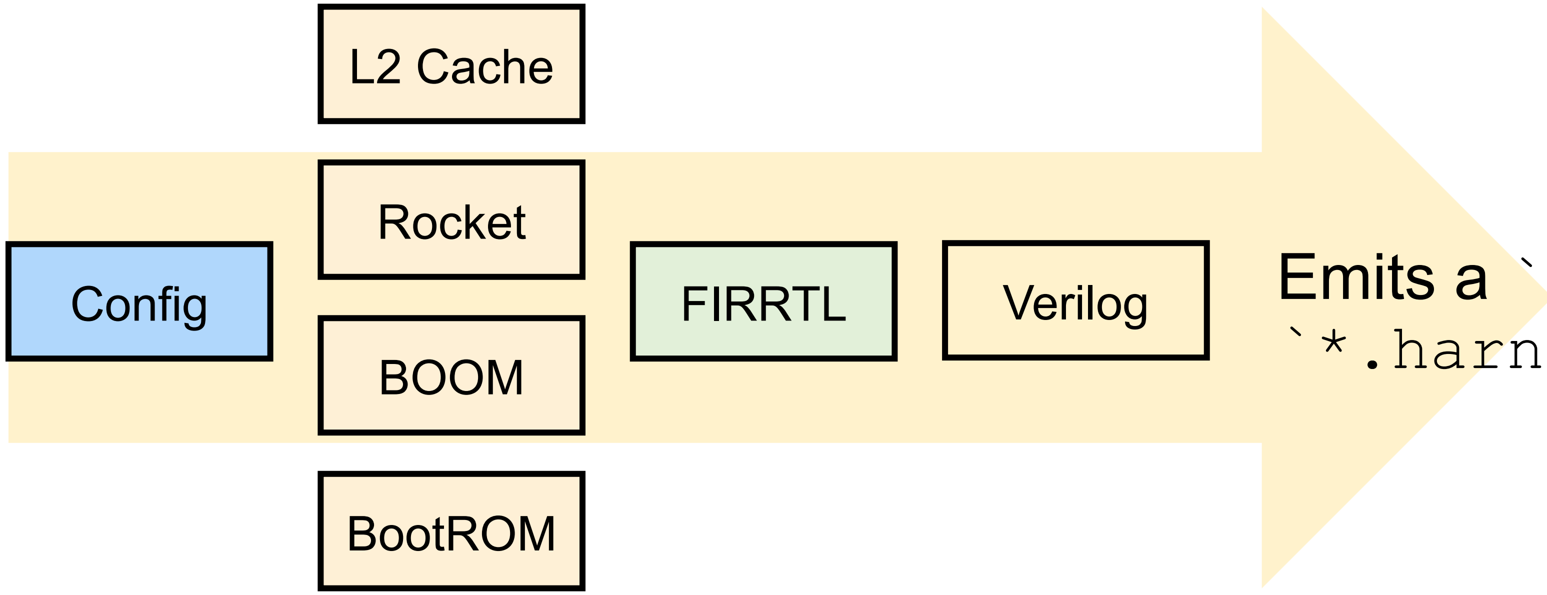


Building a SW Simulator

1. Configs parameterize Chisel generators

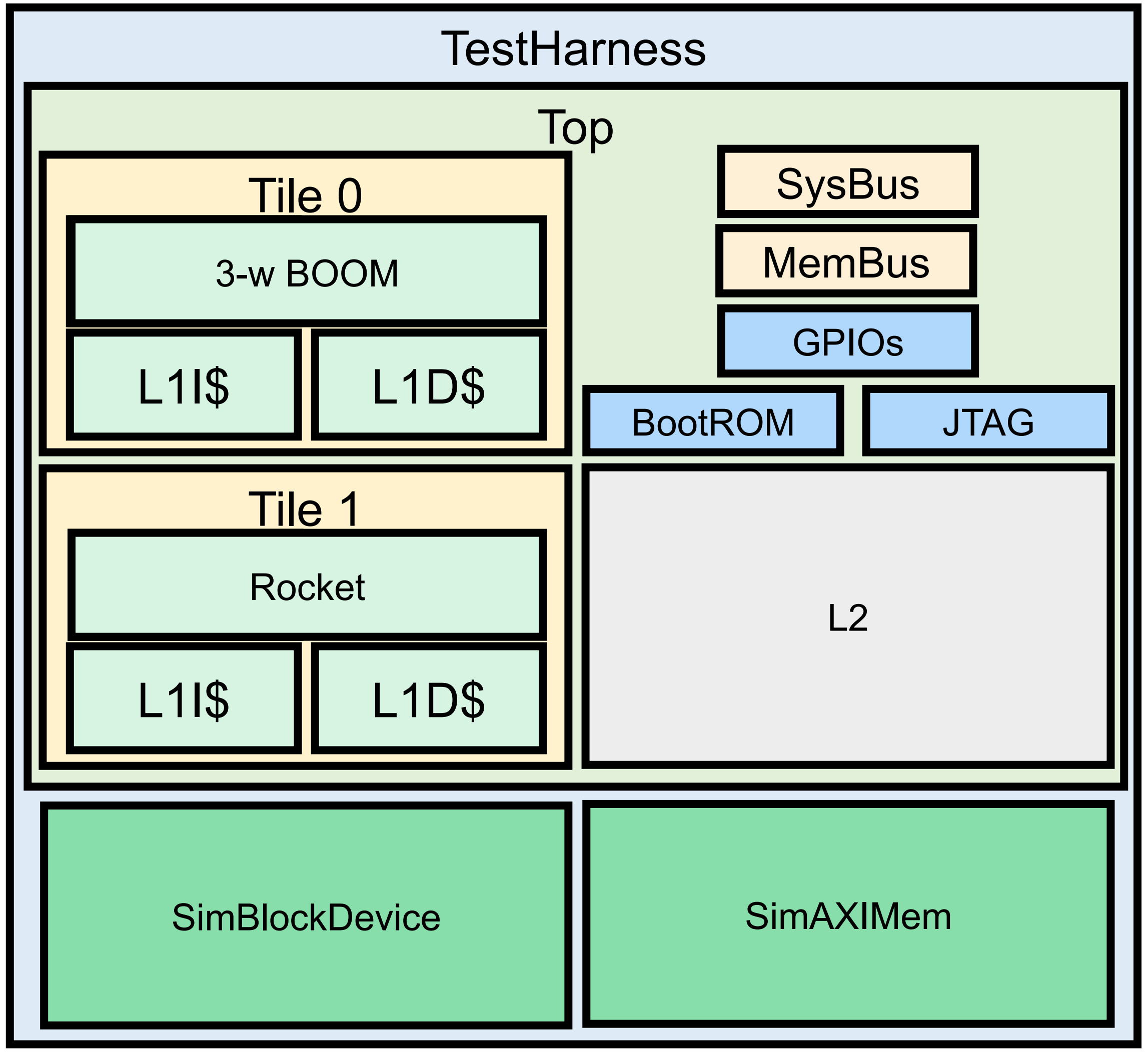
2. Chisel elaborates into FIRRTL

3. FIRRTL elaborates into Verilog



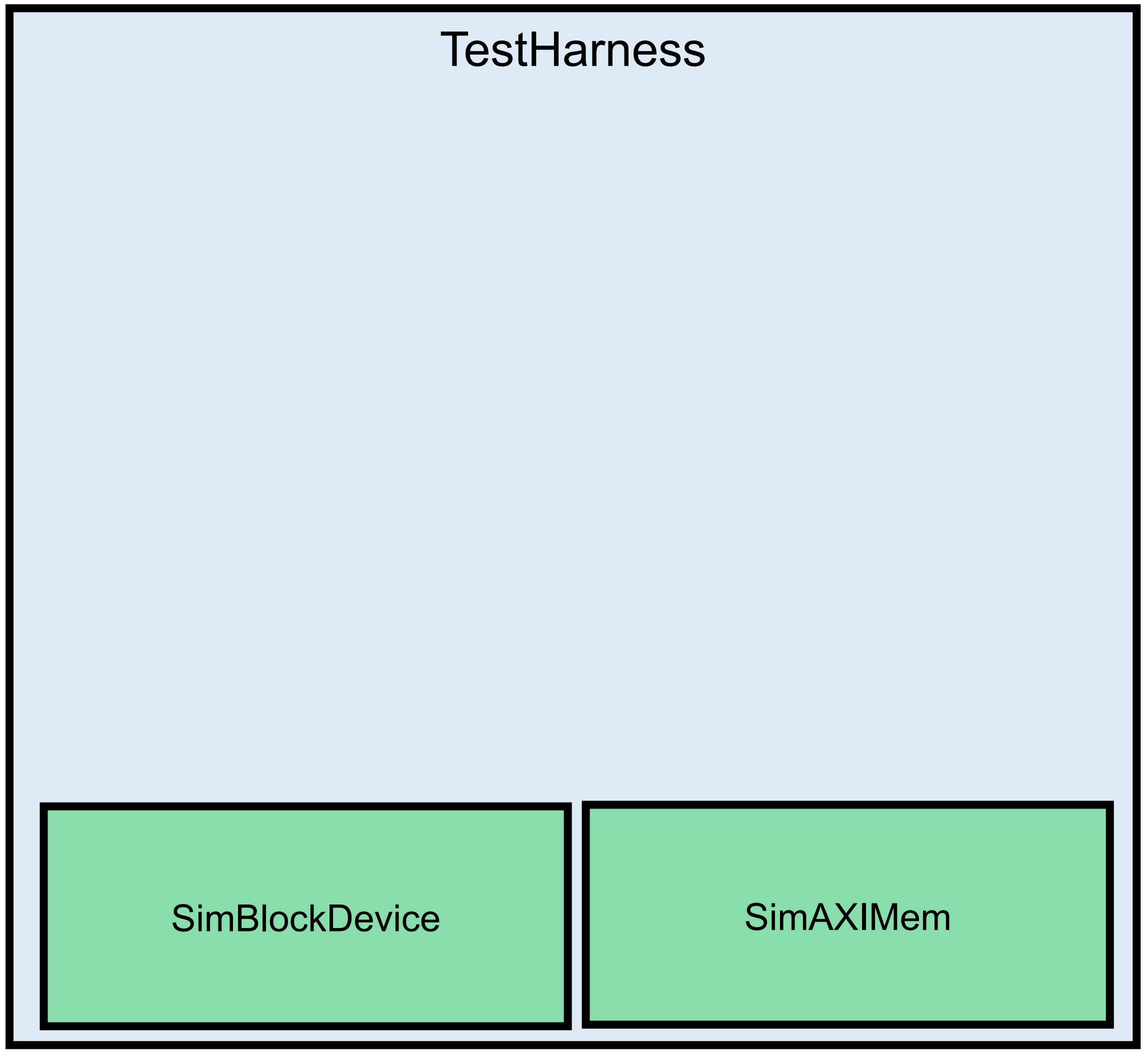
Emits a `` `*.top.v` `` and `` `*.harness.v` `` file

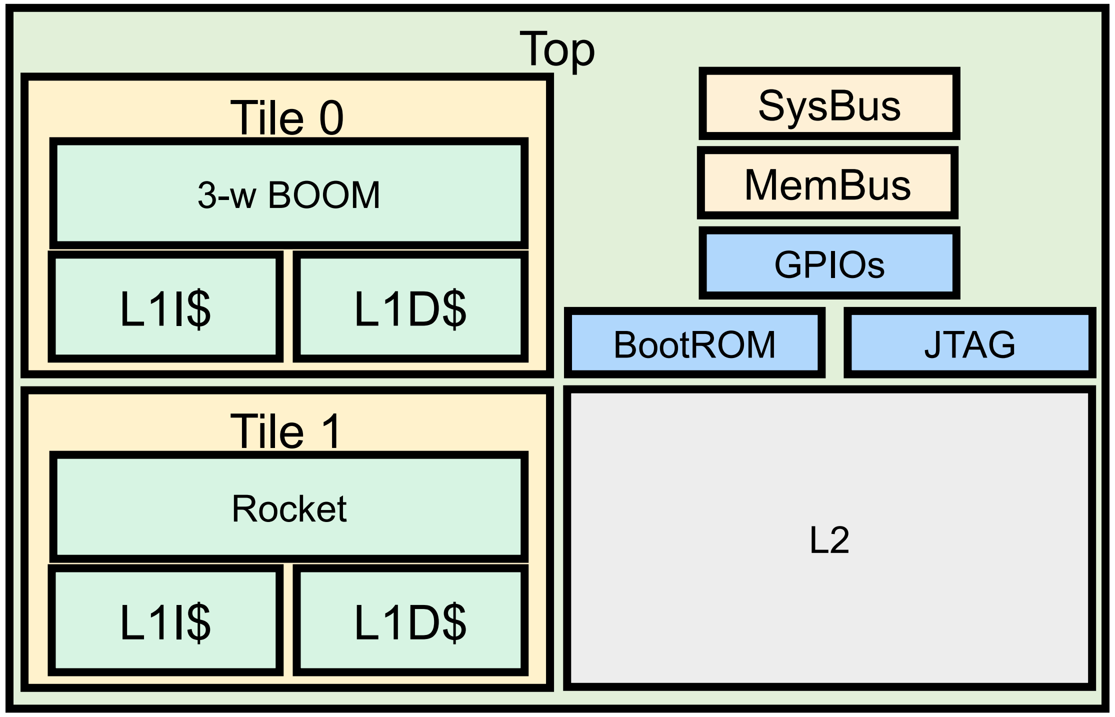
Why `*.top.v` and `*.harness.v`



Berkeley Architecture Research

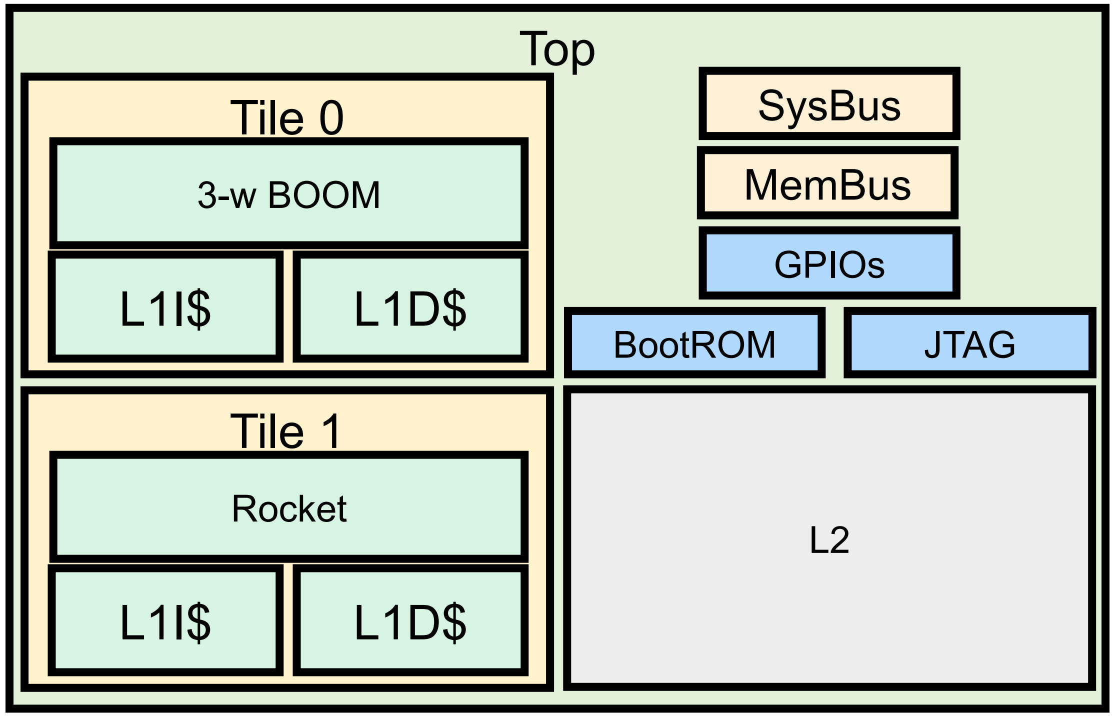
Why `*.top.v` and `*.harness.v`





Why `*.top.v` and `*.harness.v`

Pass to your VLSI flow!



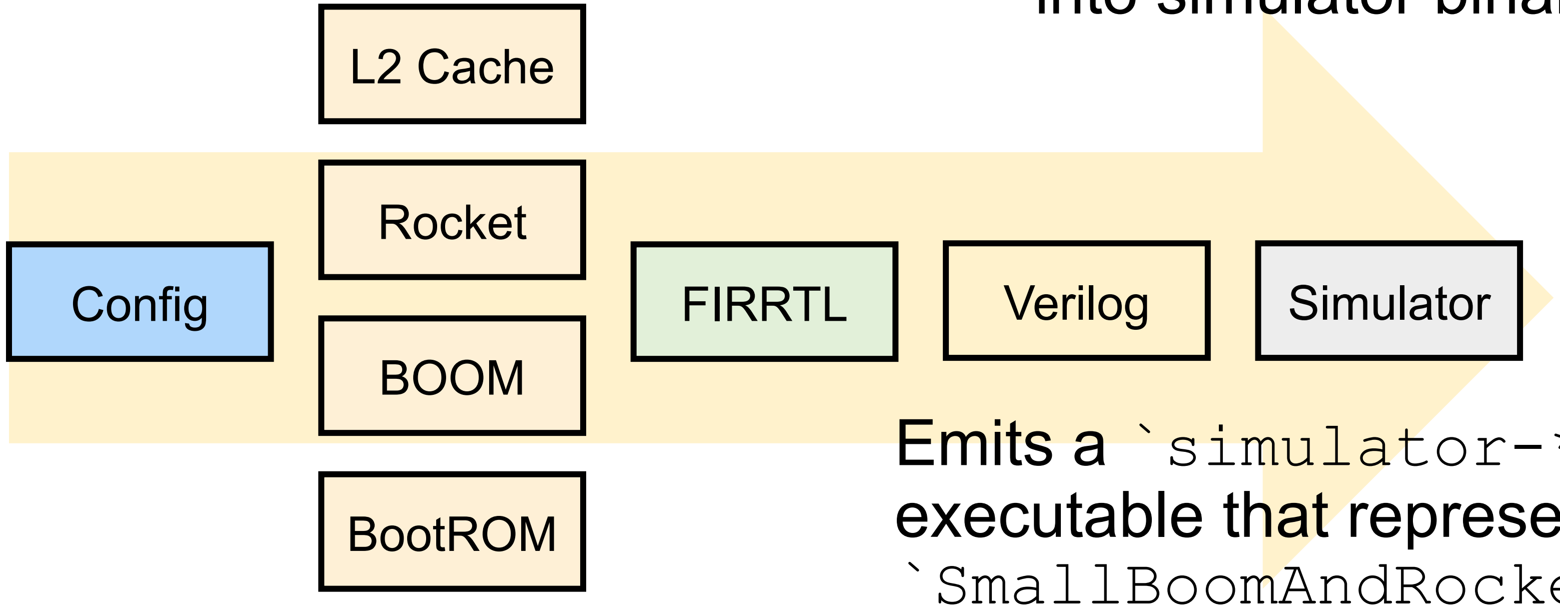
Building a SW Simulator

1. Configs parameterize Chisel generators

2. Chisel elaborates into FIRRTL

3. FIRRTL elaborates into Verilog

4. Verilator compiles Verilog into simulator binary



Emits a `` `simulator-*` `` executable that represents the `` `SmallBoomAndRocketConfig` ``

Where is my Verilog!

- Once completed the build outputs are located here

```
> ls generated-src/*/

generated-src/*/
  *.harness.v
  *.top.v
  *.dts
  *.json
  ...
```

Verilog used for testing

Main Verilog of the design

Device Tree used for SW

MMIO files (where and what is connected)

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
      generated-src/*/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Interactive

```
# navigate to the output sources
> cd generated-src/*/
> ls

# take a look at the following files
> less example.TestHarness.*.top.v
> less example.TestHarness.*.harness.v
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
      generated-src/*/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

How does the Verilog look?

example.TestHarness.SmallBoomAndRocketConfig.top.v

```
// Rocket Tile (includes a Rocket Core + L1$’s)
module RocketTile( // @[:example.TestHarness.SmallBoomAndRocketConfig.fir@347233.2]
  input         clock, // @[:example.TestHarness.SmallBoomAndRocketConfig.fir@347234.4]
  input         reset, // @[:example.TestHarness.SmallBoomAndRocketConfig.fir@347235.4]
. . .
// BOOM Tile (includes a BOOM Core + L1$’s)
module BoomTile( // @[:example.TestHarness.SmallBoomAndRocketConfig.fir@293433.2]
  input         clock, // @[:example.TestHarness.SmallBoomAndRocketConfig.fir@293434.4]
  input         reset, // @[:example.TestHarness.SmallBoomAndRocketConfig.fir@293435.4]
. . .
// Top-level DUT that includes the BOOM and Rocket Tiles
module Top( // @[:example.TestHarness.SmallBoomAndRocketConfig.fir@384841.2]
  input         clock, // @[:example.TestHarness.SmallBoomAndRocketConfig.fir@384842.4]
  input         reset, // @[:example.TestHarness.SmallBoomAndRocketConfig.fir@384843.4]
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
      generated-src/*/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

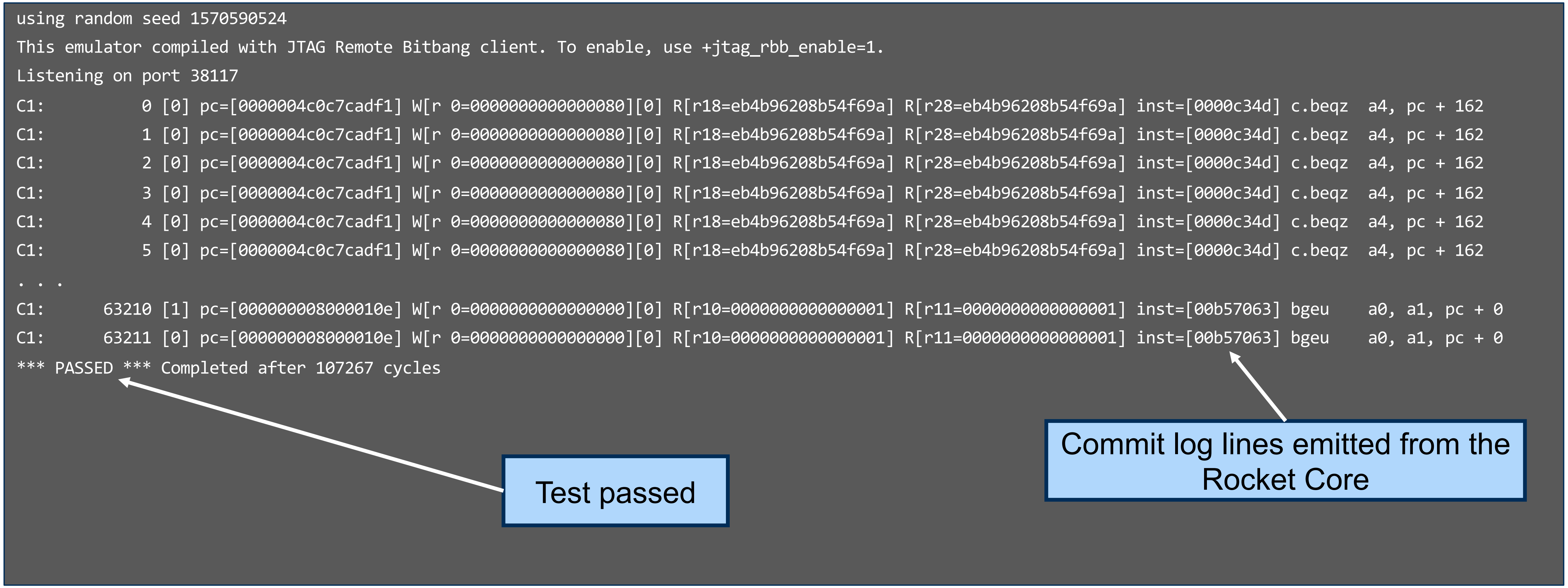
Interactive

```
# navigate to the output sources
> cd ../..
> ls

# run a multi-threaded vector-vector add
> make CONFIG=SmallBoomAndRocketConfig BINARY=mt-vvadd.riscv run-binary

# take a look at the output file
> less mt-vvadd.*.out
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Output file

mt-vvadd.example.TestHarness.SmallBoomAndRocketConfig.top.v



```
using random seed 1570590524
This emulator compiled with JTAG Remote Bitbang client. To enable, use +jtag_rbb_enable=1.
Listening on port 38117
C1:          0 [0] pc=[0000004c0c7cadf1] W[r 0=0000000000000080][0] R[r18=eb4b96208b54f69a] R[r28=eb4b96208b54f69a] inst=[0000c34d] c.beqz  a4, pc + 162
C1:          1 [0] pc=[0000004c0c7cadf1] W[r 0=0000000000000080][0] R[r18=eb4b96208b54f69a] R[r28=eb4b96208b54f69a] inst=[0000c34d] c.beqz  a4, pc + 162
C1:          2 [0] pc=[0000004c0c7cadf1] W[r 0=0000000000000080][0] R[r18=eb4b96208b54f69a] R[r28=eb4b96208b54f69a] inst=[0000c34d] c.beqz  a4, pc + 162
C1:          3 [0] pc=[0000004c0c7cadf1] W[r 0=0000000000000080][0] R[r18=eb4b96208b54f69a] R[r28=eb4b96208b54f69a] inst=[0000c34d] c.beqz  a4, pc + 162
C1:          4 [0] pc=[0000004c0c7cadf1] W[r 0=0000000000000080][0] R[r18=eb4b96208b54f69a] R[r28=eb4b96208b54f69a] inst=[0000c34d] c.beqz  a4, pc + 162
C1:          5 [0] pc=[0000004c0c7cadf1] W[r 0=0000000000000080][0] R[r18=eb4b96208b54f69a] R[r28=eb4b96208b54f69a] inst=[0000c34d] c.beqz  a4, pc + 162
. . .
C1:      63210 [1] pc=[000000008000010e] W[r 0=0000000000000000][0] R[r10=0000000000000001] R[r11=0000000000000001] inst=[00b57063] bgeu    a0, a1, pc + 0
C1:      63211 [0] pc=[000000008000010e] W[r 0=0000000000000000][0] R[r10=0000000000000001] R[r11=0000000000000001] inst=[00b57063] bgeu    a0, a1, pc + 0
*** PASSED *** Completed after 107267 cycles
```

Berkeley Architecture Research

Case Study: Create and test a SHA3-accelerated SoC!

Why do the case study?

- “We are in a golden age of computer architecture”
- Domain Specific Accelerators are extremely popular
  - Vector Accelerators
  - ML Accelerators
    - Training
    - Inference
  - Compression Accelerators
  - More!

- Goal is to answer: “But how do I add my own accelerator?”

SHA3 accelerator?

- SHA3 accelerator is a pre-implemented Chisel accelerator
- Implements the Secure Hash Algorithm 3 (SHA3)
  - Rough specification in 2012, released in late 2015
  - Uses variable length messages with a sponge function
- Designed to be more efficient when implemented in HW!
  - Want to improve the hashes/sec and hashes/Watt

Let’s begin adding it!

Interactive

```
# add sha3 to chipyard
> cd ~/chipyard-morning/generators
> git submodule add https://github.com/ucb-bar/sha3.git

# open the dependencies file
> cd ..
> vim build.sbt
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Interactive

```
//lazy val sha3 = (project in file("generators/sha3"))
//  .dependsOn(rocketchip, chisel_testers)
//  .settings(commonSettings)

lazy val example = (project in file("generators/example"))
  .dependsOn(boom, hwacha, sifive_blocks, sifive_cache, utilities)
  .settings(commonSettings)
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Interactive

```
~~//~~lazy val sha3 = (project in file("generators/sha3"))
~~//~~  .dependsOn(rocketchip, chisel_testers)
~~//~~  .settings(commonSettings)
. . .
lazy val example = (project in file("generators/example"))
  .dependsOn(boom, hwacha, sifive_blocks, sifive_cache, utilities)
  .settings(commonSettings)
```

```
lazy val sha3 = (project in file("generators/sha3"))
  .dependsOn(rocketchip, chisel_testers)
  .settings(commonSettings)
. . .
lazy val example = (project in file("generators/example"))
  .dependsOn(sha3, boom, hwacha, sifive_blocks, sifive_cache, utilities)
  .settings(commonSettings)
```

Uncomment

Add sha3

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

What happened?

- There are three steps to add an accelerator to Chipyard
  1. Add the source code to `` `generators` ``
  2. Add the accelerator project to the build system in `` `build.sbt` ``
     - Need to tell the build system it exists
     - Need to add it to a top-level project (in our case `` `example` ``)
  3. Create a new configuration that adds it!

What happened?

- There are three steps to add an accelerator to Chipyard
  1. Add the source code to `` `generators` ``
  2. Add the accelerator project to the build system in `` `build.sbt` ``
     - Need to tell the build system it exists
     - Need to add it to a top-level project (in our case `` `example` ``)
  3. Create a new configuration that adds it!

Berkeley Architecture Research

What happened?

- There are three steps to add an accelerator to Chipyard ✓
  1. Add the source code to `` `generators` ``
  2. Add the accelerator project to the build system in `` `build.sbt` `` ✓
     - Need to tell the build system it exists
     - Need to add it to a top-level project (in our case `` `example` ``)
  3. Create a new configuration that adds it!

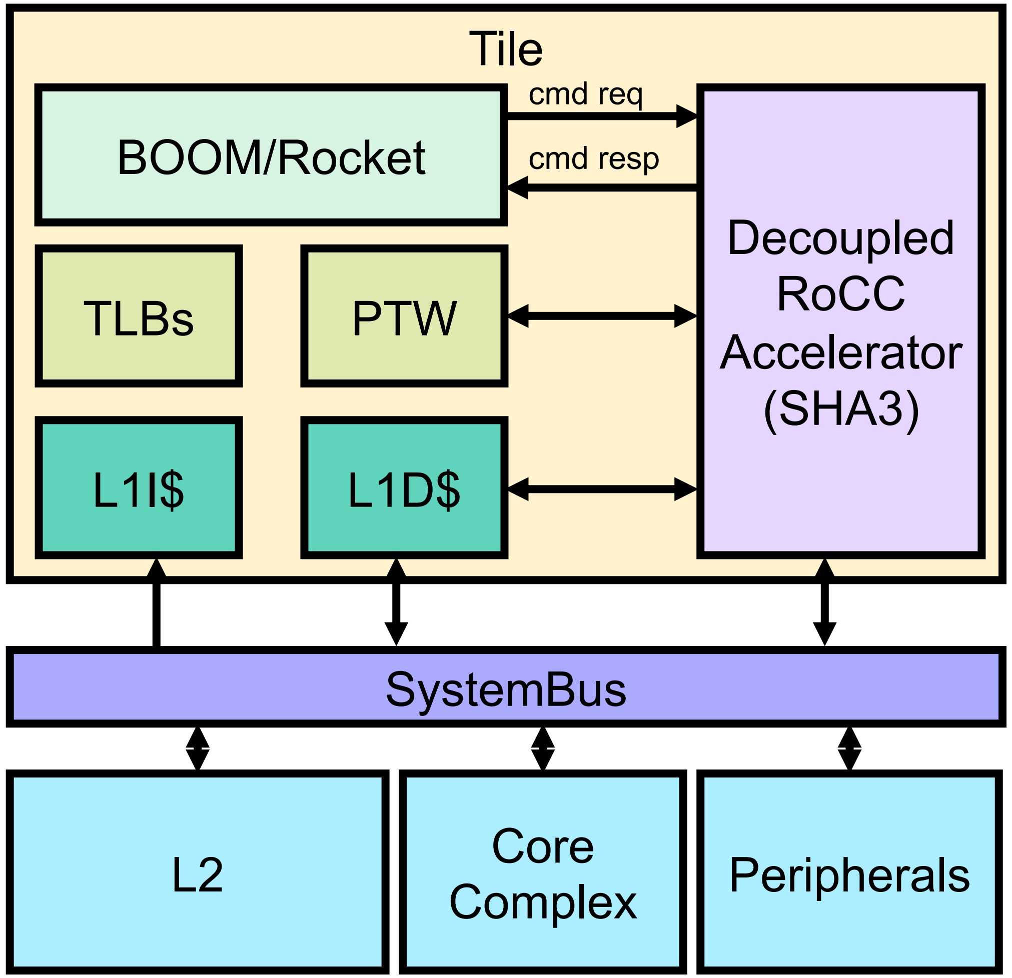
First a quick-background on
RoCC accelerators

RoCC Accelerators

- **RoCC:** Rocket Chip Coprocessor
- 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

- Examples of RoCC accelerators
  - Hwacha vector accelerators
  - Memcpy accelerator
  - Machine-learning accelerators
  - Java GC accelerator



Understanding the SHA3 Accelerator

- SHA3 is a minimal example of a RoCC-based accelerator
  - Executes custom “sha3” instructions sent by the Rocket or BOOM core
- sha3.scala
  - Note the `WithSha3Accel` mixin, which plugs into the Rocket Chip config system
  - `Sha3AccelImp` implements the Chisel-based accelerator

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    sha3/
      src/main/scala/
        sha3.scala
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

The SHA3 Accelerator

- The SHA3 mixin

```
class WithSha3Accel extends Config ((site, here, up) => {
  case Sha3WidthP => 64
  case Sha3Stages => 1
  case Sha3FastMem => true
  case Sha3BufferSram => false
  case BuildRoCC => Seq(
    (p: Parameters) => {
      val sha3 = LazyModule.apply(
        new Sha3Accel(OpcodeSet.custom2)(p)
      )
      sha3
    }
  )
})
```

SHA3 Parameters

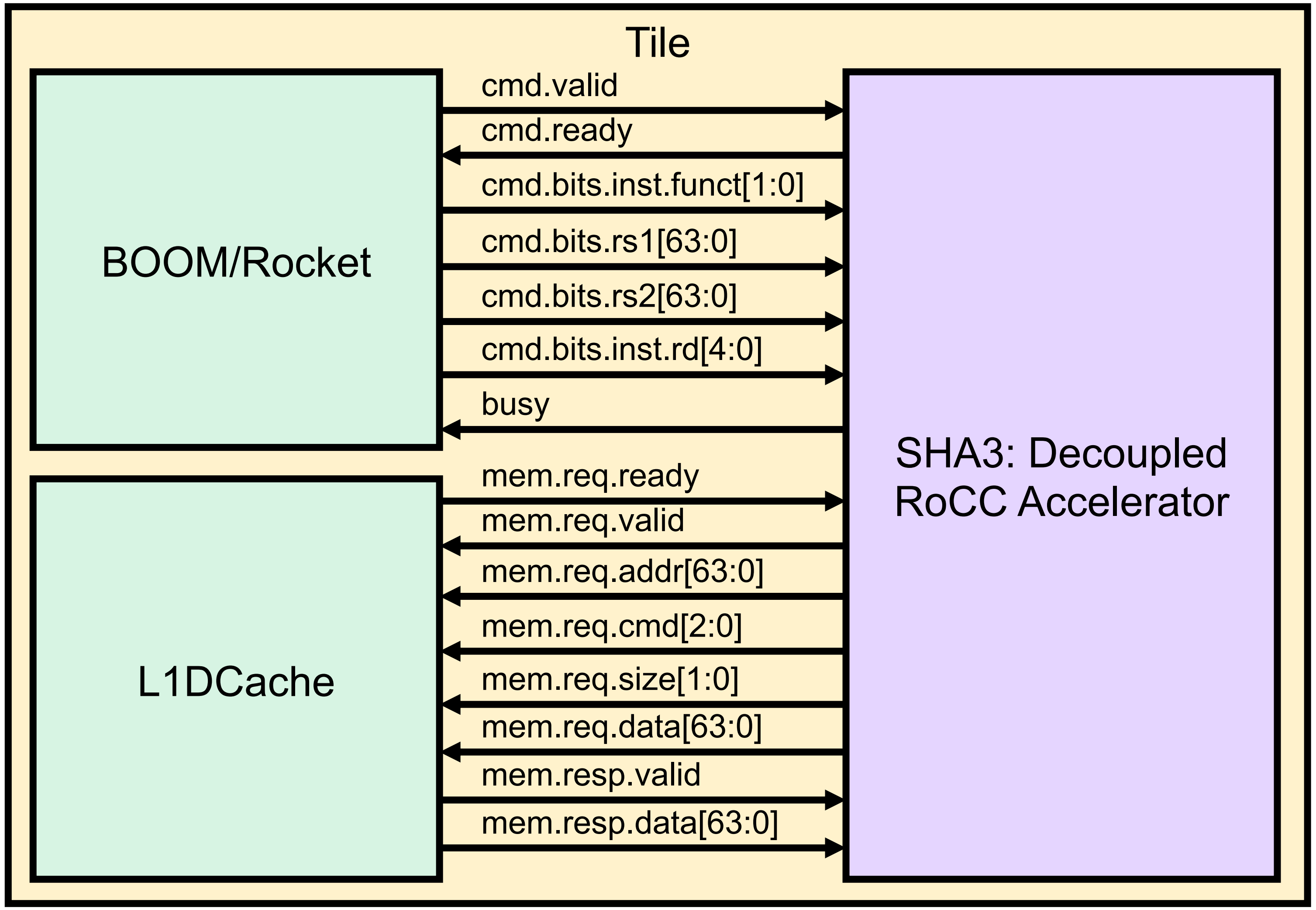
Rocket Chip uses the “BuildRoCC” key to figure out which accelerator to build

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    sha3/
      src/main/scala/
        sha3.scala
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

The SHA3 Accelerator

- RoCC Command Stream



```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    sha3/
      src/main/scala/
        sha3.scala
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Interactive

```
# open rocket configs file
> cd generators/example/src/main/scala
> vim RocketConfigs.scala
```

```
class TutorialRocketConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new freechips.rocketchip.subsystem.WithNBigCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        RocketConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Interactive

```
# open rocket configs file
> cd generators/example/src/main/scala
> vim RocketConfigs.scala
```

```
class TutorialRocketConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new freechips.rocketchip.subsystem.WithNBigCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

Modify

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        RocketConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Interactive

```
# open rocket configs file
> cd generators/example/src/main/scala
> vim RocketConfigs.scala
```

```
class TutorialRocketConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new freechips.rocketchip.subsystem.WithNBigCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

Modify and add

```
class Sha3RocketConfig extends Config(
  new WithTop ++
  new WithBootROM ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new sha3.WithSha3Accel ++
  new freechips.rocketchip.subsystem.WithNBigCores(1) ++
  new freechips.rocketchip.system.BaseConfig)
```

```
chipyard-morning/
  generators/
    example/
      src/main/scala/
        RocketConfigs.scala
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Interactive

```
# navigate to the verilator
> cd ~/chipyard-morning/sims/verilator

# start the verilator rtl simulator build
> make CONFIG=Sha3RocketConfig debug -j16

# this will take a while!
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Reminder: Where is my Verilog!

- Once completed the build outputs are located here

```
> ls generated-src/example.TestHarness.Sha3RocketConfig/

generated-src/example.TestHarness.Sha3RocketConfig/
  *.harness.v
  *.top.v
  *.dts
  *.json
  ...
```

Verilog used for testing

Main Verilog of the design

Device Tree used for SW

MMIO files (where and what is connected)

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
      generated-src/*/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Coffee Break until 11am

Coming up…

Running customized software on the SHA3 accelerated SoC

Slides here:
https://fires.im/micro-2019-tutorial/

Berkeley Architecture Research

Interactive

```
# re-enter the tmux session (only if you exited it)
> tmux attach -t soc

# navigate to firemarshal
> cd ~/chipyard-morning/software/firemarshal

# view sha3 accelerated program
> vim workloads/sha3/benchmarks/src/sha3-rocc.c
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  software/
    firemarshal/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Accelerator Software

sha3-rocc.c

```c
printf("Start basic test 1.\n");
// BASIC TEST 1 - 150 zero bytes

start = rdcycle();

asm volatile ("fence");

// setup accelerator with addresses of input and output
ROCC_INSTRUCTION_SS(2, &input, &output, 0);

// Set length and compute hash
ROCC_INSTRUCTION_S(2, sizeof(input), 1);

asm volatile ("fence" ::: "memory");

end = rdcycle();
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  software/
    firemarshal/
      workloads/sha3/benchmarks/src/
        sha3-rocc.c
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Accelerator Software

sha3-rocc.c

```
printf("Start basic test 1.\n");
// BASIC TEST 1 - 150 zero bytes

start = rdcycle();

asm volatile ("fence");

// setup accelerator with addresses of input and output
ROCC_INSTRUCTION_SS(2, &input, &output, 0);

// Set length and compute hash
ROCC_INSTRUCTION_S(2, sizeof(input), 1);

asm volatile ("fence" ::: "memory");

end = rdcycle();
```

Expanded C macro

```
                     opcode   rd  rs1            rs2           funct
asm volatile ("custom2 x0, %[msg_addr], %[hash_addr], 0"
        :: [msg_addr] "r" (&input), [hash_addr] "r" (&output));
```

Accelerator Software

sha3-rocc.c

```
printf("Start basic test 1.\n");
// BASIC TEST 1 - 150 zero bytes

start = rdcycle();

asm volatile ("fence");

// setup accelerator with addresses of input and output
ROCC_INSTRUCTION_SS(2, &input, &output, 0);

// Set length and compute hash
ROCC_INSTRUCTION_S(2, sizeof(input), 1);

asm volatile ("fence" ::: "memory");

end = rdcycle();
```

Expanded C macro

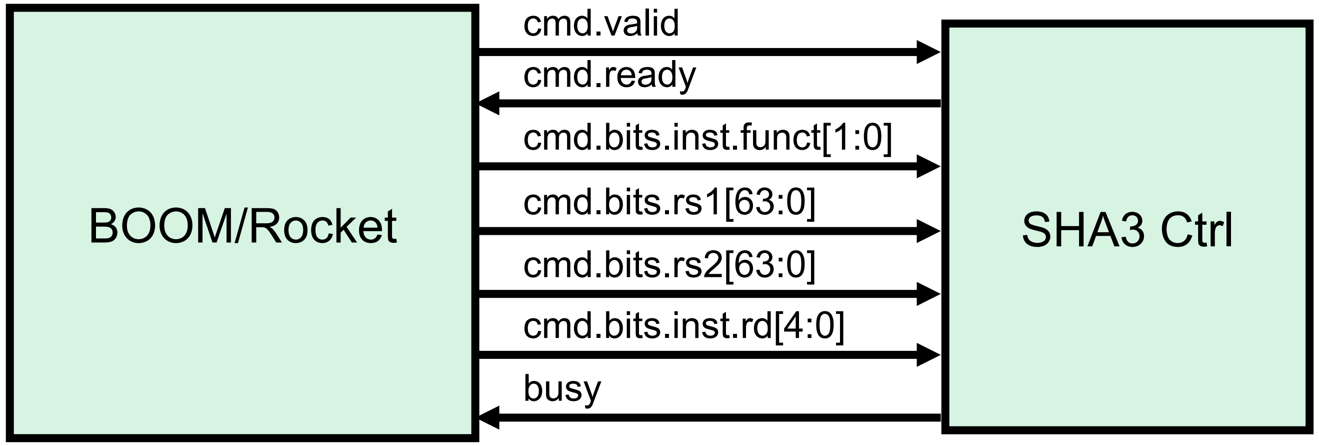
```
                  opcode   rd  rs1           rs2            funct
asm volatile ("custom2 x0, %[msg_addr], %[hash_addr], 0"
        :: [msg_addr] "r" (&input), [hash_addr] "r" (&output));
```

Setup and run the accelerator

Berkeley Architecture Research

Understanding SHA3 Software

- Send necessary information to the accelerator
  - Source(s), Destination, and what function to run
- Accelerator returns
  - Busy… a.k.a. “Is the accelerator done?”



```
                    opcode   rd   rs1            rs2            funct
asm volatile ("custom2 x0, %[msg_addr], %[hash_addr], 0"
              :: [msg_addr] "r" (&input), [hash_addr] "r" (&output));
```

Berkeley Architecture Research

Interactive

```
# view sw implementation of SHA3 program
> vim workloads/sha3/benchmarks/src/sha3-sw.c
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  software/
    firemarshal/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Software SHA3

sha3-rocc.c

```
printf("Start basic test 1.\n");
// BASIC TEST 1 - 150 zero bytes

start = rdcycle();

// run sw to compute the SHA3 hash
sha3ONE(input, sizeof(input), output);

end = rdcycle();
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  software/
    firemarshal/
      workloads/sha3/benchmarks/src/
        sha3-sw.c
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Software SHA3

sha3-rocc.c

```
printf("Start basic test 1.\n");
// BASIC TEST 1 - 150 zero bytes

start = rdcycle();

// run sw to compute the SHA3 hash
sha3ONE(input, sizeof(input), output);

end = rdcycle();
```

SW implementation of SHA3

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  software/
    firemarshal/
      workloads/sha3/benchmarks/src/
        sha3-sw.c
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Interactive

```
# make sure you are in firemarshal directory
> cd ~/chipyard-morning/software/firemarshal

# build both binaries
> ./marshal build workloads/sha3-bare-*.json
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  software/
    firemarshal/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Building SHA3 Software

- Used the FireMarshal utility to build the binaries
  - Tool that takes in `` `.json` `` description of build and emits the `` `.riscv` `` binary
  - More in-depth view after lunch
- What was done… built two binaries
  - `` `sha3-sw.riscv` `` - software version of SHA3 computation
  - `` `sha3-rocc.riscv` `` - sends SHA3 computation to the accelerator
- Both binaries created in
  - `` `workloads/sha3/benchmarks/bare/sha3-*.riscv` ``

Interactive

```
# navigate to the verilator directory
> cd ~/chipyard-morning/sims/verilator


# run accelerated program
> make CONFIG=Sha3RocketConfig run-binary-debug
BINARY=../../software/firemarshal/workloads/sha3/benc
hmarks/bare/sha3-rocc.riscv


# run non-accelerated program
> make CONFIG=Sha3RocketConfig run-binary-debug
BINARY=../../software/firemarshal/workloads/sha3/benc
hmarks/bare/sha3-sw.riscv
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  software/
    firemarshal/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

The non-accelerated version takes longer

Berkeley Architecture Research

Outputs

- You should see the following after both tests

```
# you just ran: “make CONFIG=… BINARY=… run-binary-debug”
# this is the output:

Start basic test 1.
output[0]:221 ==? results[0]:221
. . .
output[31]:238 ==? results[31]:238
Success!
SHA execution took M cycles
```

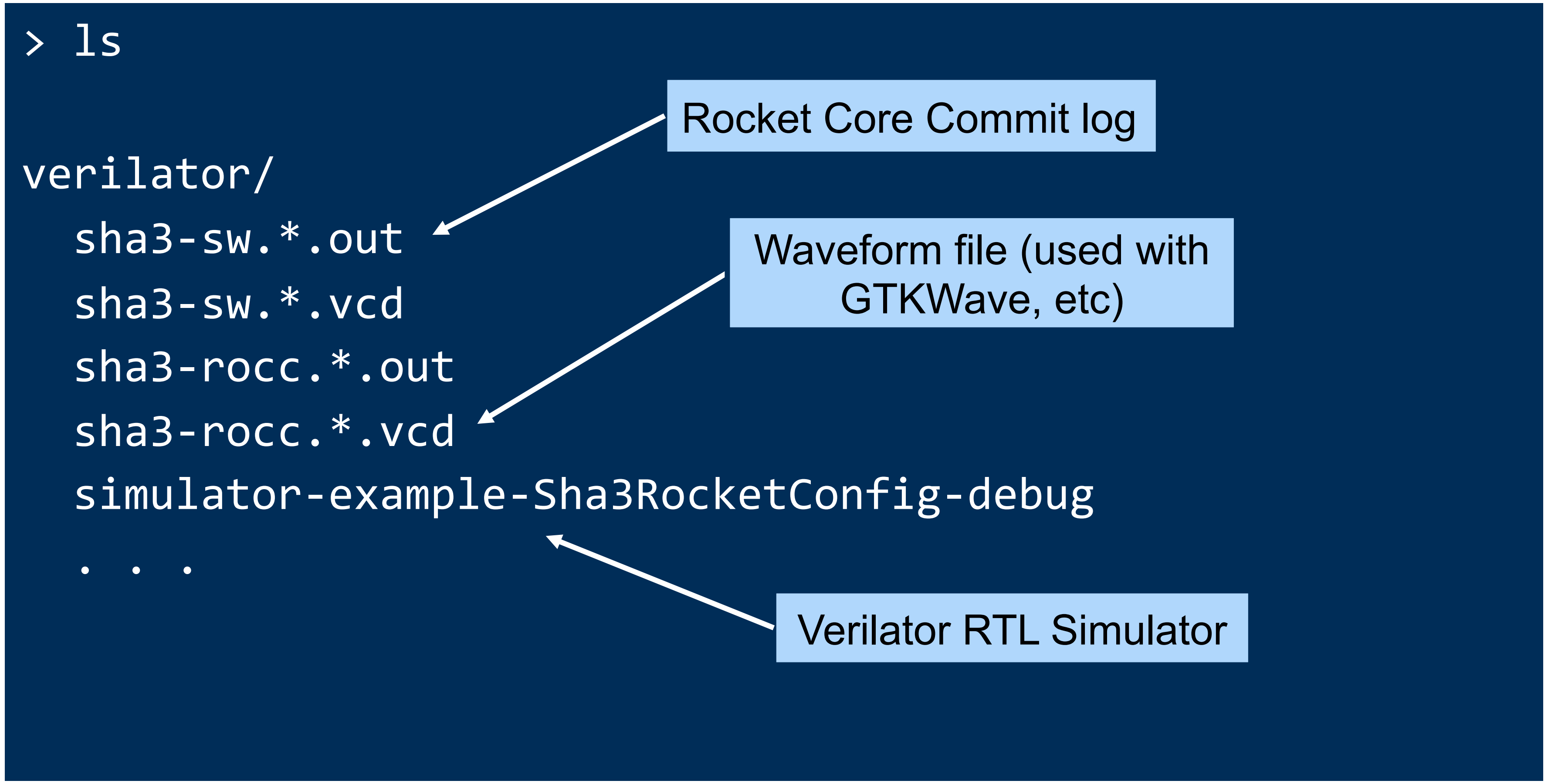
Compare the cycle time of both runs.
Which is faster?

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  software/
    firemarshal/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

All Outputs

- Once completed the build outputs are located here



```
> ls

verilator/
  sha3-sw.*.out
  sha3-sw.*.vcd
  sha3-rocc.*.out
  sha3-rocc.*.vcd
  simulator-example-Sha3RocketConfig-debug
  . . .
```

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    boom/
    sha3/
  sims/
    verilator/
  software/
    firemarshal/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Interactive

Delete the tmux session

```
# exit the terminal
CTRL-D
```

Berkeley Architecture Research

More Information

- https://chipyard.readthedocs.io/en/latest
  - Talks about heterogeneous SoCs
  - Talks about adding Verilog IP
  - Talks about adding accelerators
  - . . .

Covers what we did in this talk and more!

dev

Search docs

Docs » Welcome to Chipyard's documentation!

Edit on GitHub

Welcome to Chipyard's documentation!

CHIPYARD

Chipyard is a a framework for designing and evaluating full-system hardware using agile teams. It is composed of a collection of tools and libraries designed to provide an intergration between open-source and commercial tools for the development of systems-on-chip. New to Chipyard? Jump to the Chipyard Basics page for more info.

Quick Start

Requirements

Chipyard is developed and tested on Linux-based systems.

Warning

Berkeley Architecture Research

That’s it!

Coming up…

Using Hammer to Speed Up the VLSI Flow

Backup Slides

Exploring the Module Hierarchy

example.Sha3RocketConfig.(harness/top).v:

- Generated Verilog for Top and TestHarness

example.Sha3RocketConfig.core.config:

- Configuration information for core

example.Sha3RocketConfig.dts:

- Device tree string

example.Sha3RocketConfig.fir:

- FIRRTL IR

example.Sha3RocketConfig.graphml:

- Diplomacy graph

```
chipyard-morning/
  generators/
    example/
    rocket-chip/
    sha3/
  sims/
    verilator/
      generated-src/
        example.Sha3RocketConfig/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

TileLink PWM Custom Module

- Custom PWM module driving a top-level IO pin
- Changes to base SoC
  - Additional top-level pin for PWM out
  - Additional memory-mapped register for configuring PWM
  - Additional top-level module to drive PWM

```
chipyard-morning/
  generators/
    example/
      PWM.scala
      RocketConfigs.scala
      Top.scala
    rocket-chip/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

Berkeley Architecture Research

Instantiating a PWM Top

PWMModule:

- Module which instantiates the PWM functionality

PWMTL:

- Module which creates a TileLink node, and inherits functionality of the PWMModule

HasPeripheryPWMTL:

- Connects PWM TileLink node to Diplomacy graph of the BaseSubsystem

HasPeripheryPWMTLModuleImp

- Adds additional IO pin for PWMOut, and drives it with the PWMModule

```
chipyard-morning/
  generators/
    example/
      PWM.scala
      RocketConfigs.scala
      Top.scala
    rocket-chip/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```

PWM Custom Components

TopWithPWMTL:

- Adds the HasPeripheryPWMTL trait to add the PWM node to the Top's Diplomacy graph
- Instantiates a TopWithPWMTLModule

TopWithPWMTLModule:

- Adds the HasPeripheryPWMTLModuleImp trait to get the additional PWM IO pin and PWM Module

```
chipyard-morning/
  generators/
    example/
      PWM.scala
      RocketConfigs.scala
      Top.scala
    rocket-chip/
    sha3/
  sims/
    verilator/
  tools/
    chisel/
    firrtl/
  tests/
  build.sbt
```