

Configuring and Building Custom RISC-V SoCs in Chipyard

Jerry Zhao

UC Berkeley

jzh@berkeley.edu



Berkeley
Architecture
Research

CHIPYARD

Goals



- Get the basics of modifying a configuration
 - Create a heterogeneous BOOM and Rocket SoC
 - Configure a SoC with a custom accelerator
- Understand the process for generating Verilog for an SoC
- See how to run Verilator RTL simulations





Getting Started



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
```

Example Slide
“Follow Along”

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
```

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



Example



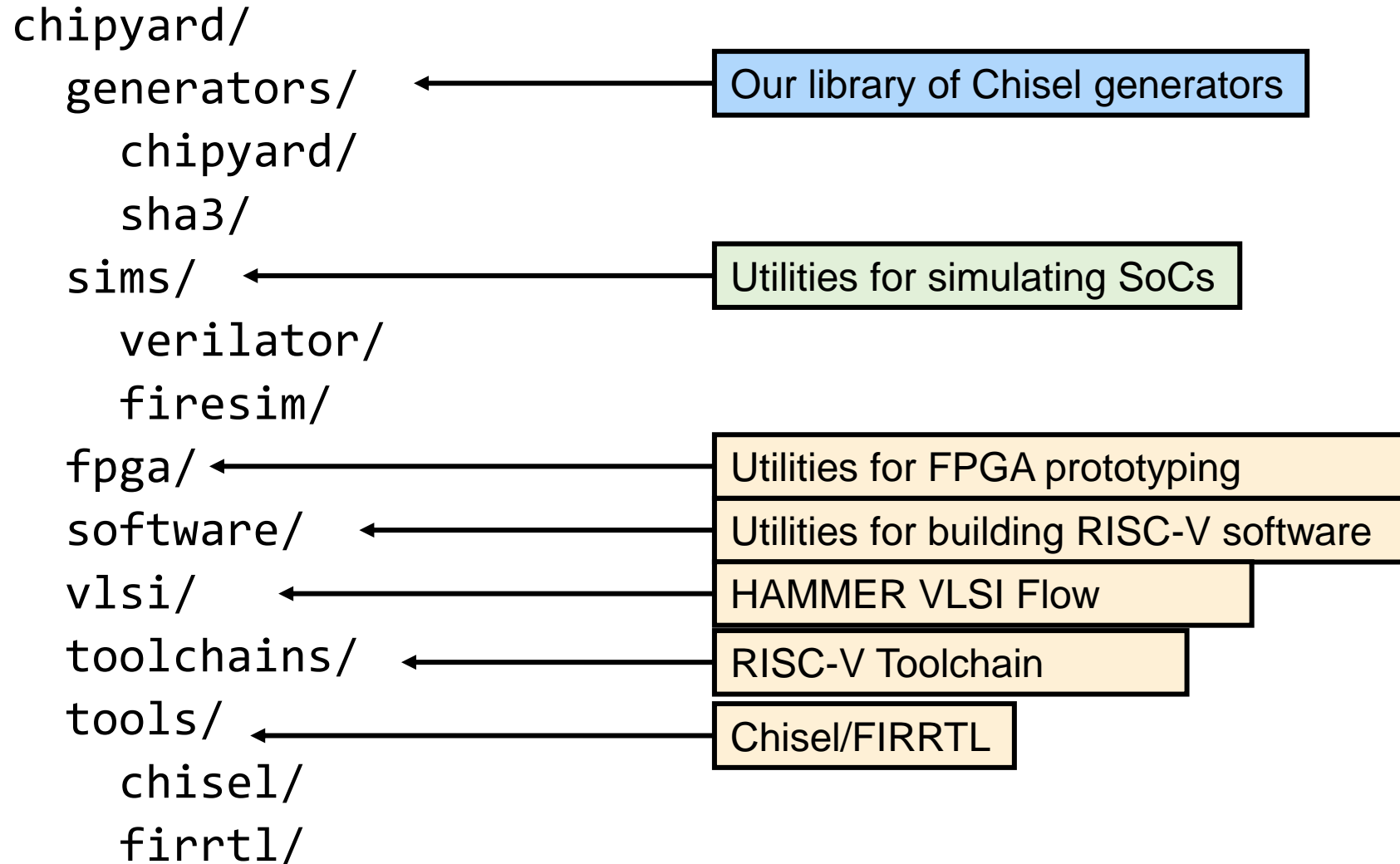
```
# Clone the Chipyard repository and setup submodules
> git clone https://github.com/ucb-bar/chipyard.git
> cd chipyard
> ./scripts/init-submodules-no-riscv-tools.sh

# Install a pre-built toolchain
> ./scripts/build-toolchains.sh esp-tools

# Source toolchain environment file
> source env-esp-tools.sh
```



Directory Structure





Build and simulate a heterogeneous BOOM + Rocket SoC



Example



```
# open up the heterogeneous configurations file
> cd generators/chipyard/src/main/scala/config
> vim HeteroConfigs.scala
```

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



Configuring a SoC with BOOM and Rocket



```
class TutorialConfig extends Config(  
  new boom.common.WithNSmallBooms(1) ++  
  new freechips.rocketchip.subsystem.WithNLargeCores(1) ++  
  new freechips.rocketchip.subsystem.WithInclusiveCache(capacityKB=256) ++  
  new chipyard.config.AbstractConfig)
```

- Bottom-to-top (right-to-left) config hierarchy
- AbstractConfig sets up “default” system configuration
- WithInclusiveCache configures the banked L2 cache
- WithNLargeCores adds Rocket Tiles
- WithNSmallCores adds small BOOM Tiles

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



Configuring a SoC with BOOM



```
class TutorialConfig extends Config(  
  new boom.common.WithNSmallBooms(1) ++  
  new freechips.rocketchip.subsystem.WithNLargeCores(1) ++  
  new freechips.rocketchip.subsystem.WithInclusiveCache(capacityKB=256) ++  
  new chipyard.config.AbstractConfig)
```

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

```
TilesKey      => Seq()  
BankedL2Key   => Module(new TLBroadcast)
```



Configuring a SoC with BOOM



```
class TutorialConfig extends Config(  
  new boom.common.WithNSmallBooms(1) ++  
  new freechips.rocketchip.subsystem.WithNLargeCores(1) ++  
  new freechips.rocketchip.subsystem.WithInclusiveCache(capacityKB=256) ++  
  new chipyard.config.AbstractConfig)
```

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

```
TilesKey      => Seq()  
BankedL2Key   => Module(InclusiveCache)
```



Configuring a SoC with BOOM



```
class TutorialConfig extends Config(  
  new boom.common.WithNSmallBooms(1) ++  
  new freechips.rocketchip.subsystem.WithNLargeCores(1) ++  
  new freechips.rocketchip.subsystem.WithInclusiveCache(capacityKB=256) ++  
  new chipyard.config.AbstractConfig)
```

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

```
TilesKey      => Seq(RocketTile)  
BankedL2Key   => Module(InclusiveCache)
```



Configuring a SoC with BOOM



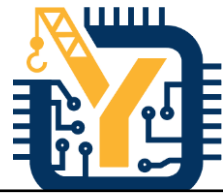
```
class TutorialConfig extends Config(  
  new boom.common.WithNSmallBooms(1) ++  
  new freechips.rocketchip.subsystem.WithNLargeCores(1) ++  
  new freechips.rocketchip.subsystem.WithInclusiveCache(capacityKB=256) ++  
  new chipyard.config.AbstractConfig)
```

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

```
TilesKey      => Seq(RocketTile, BoomTile)  
BankedL2Key   => Module(InclusiveCache)
```



Diving into the Config



```
// The HarnessBinders control generation of hardware in the TestHarness
new chipyard.harness.WithUARTAdapter ++ // add UART adapter to display UART on stdout, if uart is present
new chipyard.harness.WithBlackBoxSimMem ++ // add SimDRAM DRAM model for axi4 backing memory, if axi4 mem is enabled
new chipyard.harness.WithSimSerial ++ // add external serial-adapter and RAM
new chipyard.harness.WithSimDebug ++ // add SimJTAG or SimDTM adapters if debug module is enabled
new chipyard.harness.WithGPIOtiedOff ++ // tie-off chiptop GPIOs, if GPIOs are present
new chipyard.harness.WithSimSPIFlashModel ++ // add simulated SPI flash memory, if SPI is enabled
new chipyard.harness.WithSimAXIMMIO ++ // add SimAXIMem for axi4 mmio port, if enabled
new chipyard.harness.WithTieOffInterrupts ++ // tie-off interrupt ports, if present
new chipyard.harness.WithTieOffL2FBusAXI ++ // tie-off external AXI4 master, if present
new chipyard.harness.WithTieOffCustomBootPin ++
```

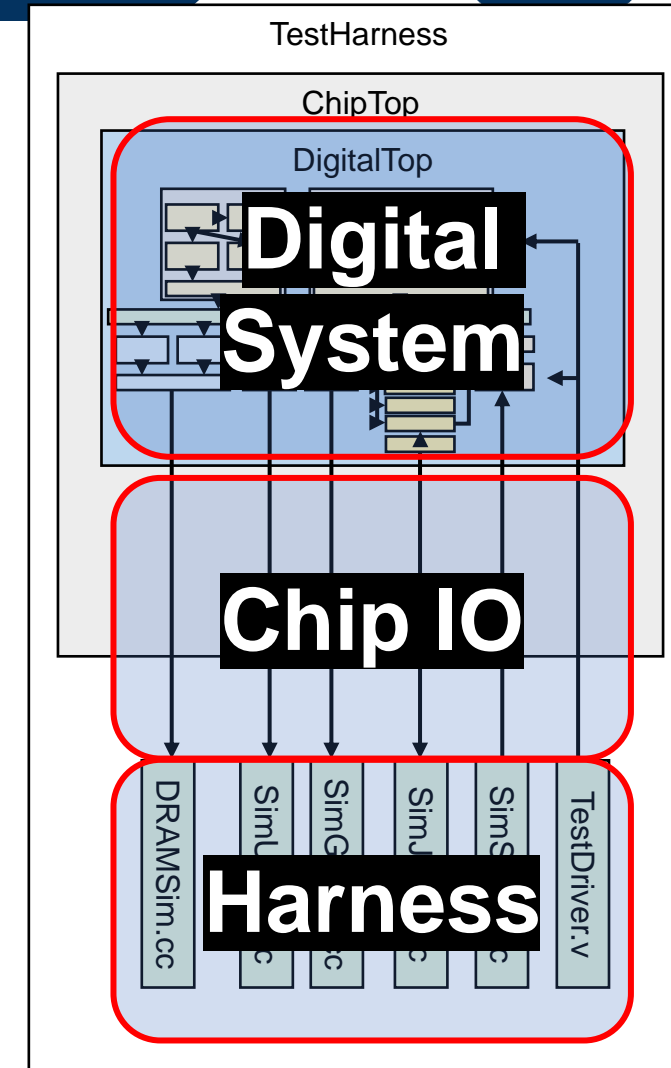
```
// The IOBinders instantiate Chiptop IOs to match desired digital IOs
// IOCells are generated for "Chip-like" IOs, while simulation-only IOs are directly punched through
new chipyard.iobinders.WithAXI4MemPunchthrough ++
new chipyard.iobinders.WithAXI4MMIOPunchthrough ++
new chipyard.iobinders.WithL2FBusAXI4Punchthrough ++
new chipyard.iobinders.WithBlockDeviceIOPunchthrough ++
new chipyard.iobinders.WithNICIOPunchthrough ++
new chipyard.iobinders.WithSerialTLIOCells ++
new chipyard.iobinders.WithDebugIOCells ++
new chipyard.iobinders.WithUARTIOCells ++
new chipyard.iobinders.WithGPIOCells ++
new chipyard.iobinders.WithUARTIOCells ++
new chipyard.iobinders.WithSPIOCells ++
new chipyard.iobinders.WithTraceIOPunchthrough ++
new chipyard.iobinders.WithExtInterruptIOCells ++
new chipyard.iobinders.WithCustomBootPin ++
```

```
new testchipip.WithDefaultSerialTL ++ // use serialized tilelink port to external serialadapter/harnessRAM
new chipyard.config.WithBootROM ++ // use default bootrom
new chipyard.config.WithUART ++ // add a UART
new chipyard.config.WithL2TLBs(1024) ++ // use L2 TLBs
new chipyard.config.WithNoSubsystemDrivenClocks ++ // drive the subsystem diplomatic clocks from Chiptop instead of using im
new chipyard.config.WithInheritBusFrequencyAssignments ++ // Unspecified clocks within a bus will receive the bus frequency if set
new chipyard.config.WithPeripheryBusFrequencyAsDefault ++ // Unspecified frequencies with match the pbus frequency (which is always s
new chipyard.config.WithMemoryBusFrequency(100.0) ++ // Default 100 MHz mbus
new chipyard.config.WithPeripheryBusFrequency(100.0) ++ // Default 100 MHz pbus
new freechips.rocketchip.subsystem.WithJtagDTM ++ // set the debug module to expose a JTAG port
new freechips.rocketchip.subsystem.WithNoMMIOPort ++ // no top-level MMIO master port (overrides default set in rocketchip)
new freechips.rocketchip.subsystem.WithNoSlavePort ++ // no top-level MMIO slave port (overrides default set in rocketchip)
new freechips.rocketchip.subsystem.WithInclusiveCache ++ // use Sifive L2 cache
new freechips.rocketchip.subsystem.WithNextTopInterrupts(0) ++ // no external interrupts
new chipyard.config.WithMultiClockCoherentBusTopology ++ // hierarchical buses including mbus+l2
new freechips.rocketchip.system.BaseConfig) // "base" rocketchip system
```

Configuring RTL in the TestHarness

Configuring IOs in the ChipTop

Configuring DigitalSystem components



Example



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

# start the Verilator RTL simulator build
> make CONFIG=TutorialConfig

# this will take a few minutes!
```

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



Behind the Scenes: Make



- ``make`` is the main 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. ``chipyard.TestHarness.SmallBoomAndRocketConfig``



Building a SW Simulator

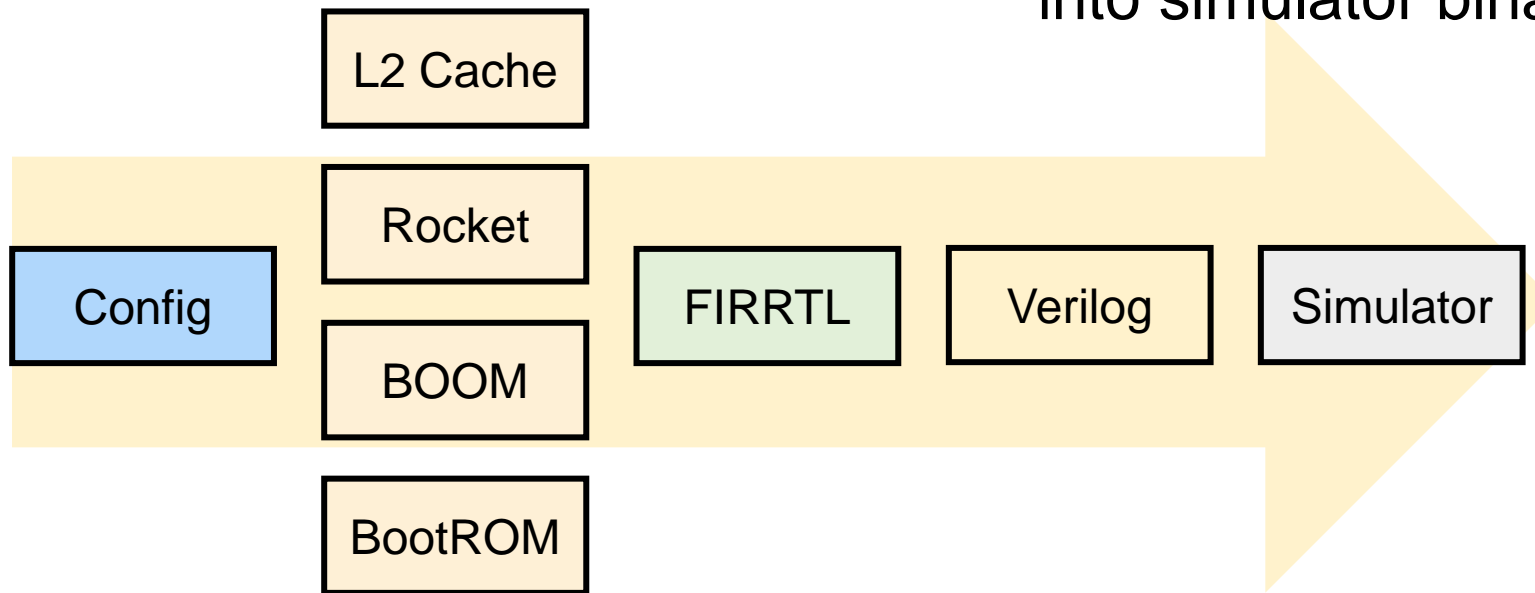


1. Configs parameterize Chisel generators

2. Chisel elaborates into FIRRTL

3. FIRRTL elaborates into Verilog

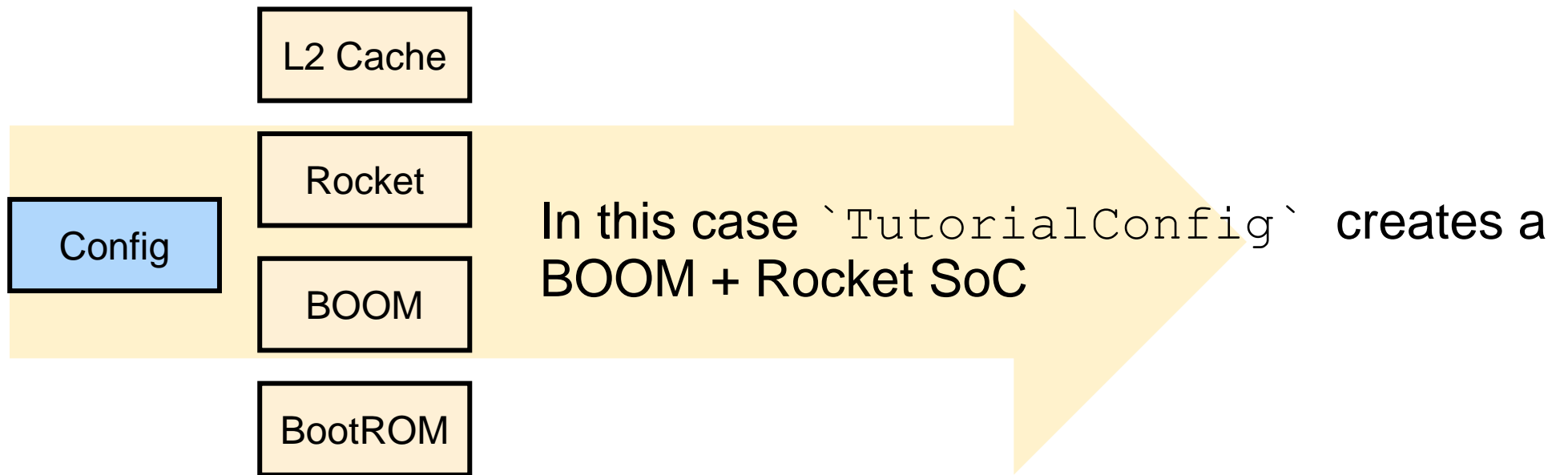
4. Verilator compiles Verilog into simulator binary



Building a SW Simulator



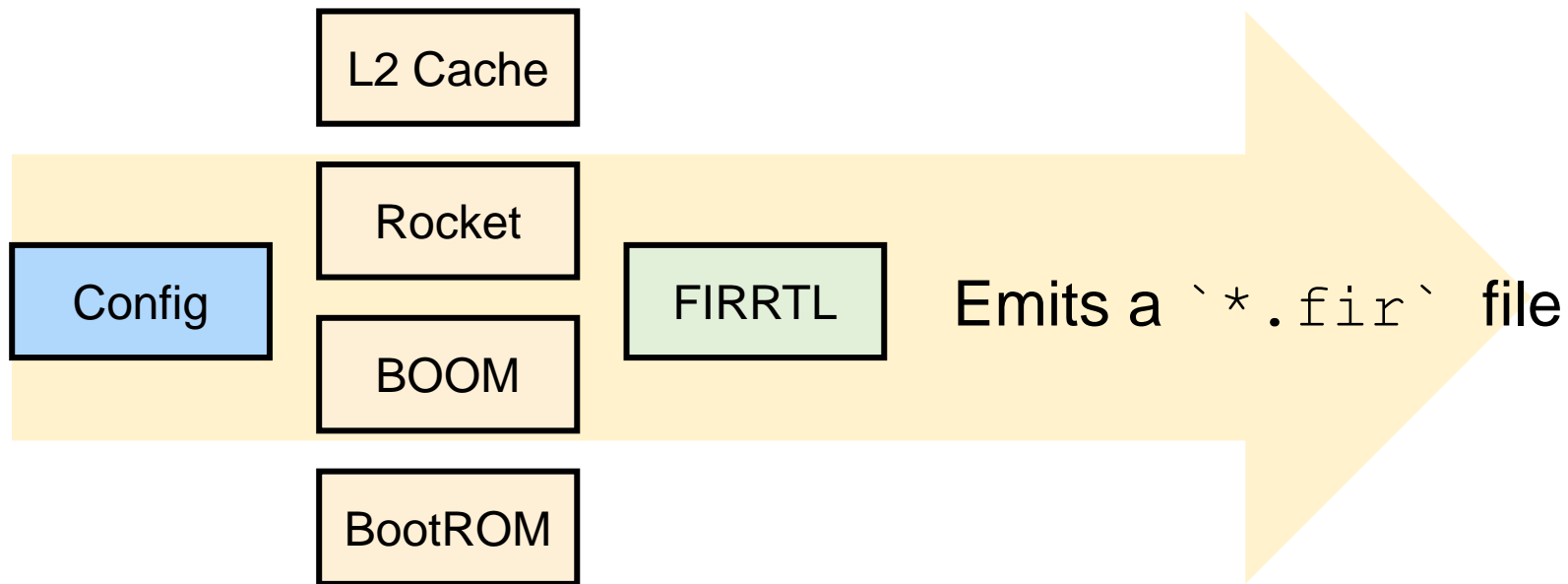
1. Configs parameterize Chisel generators



Building a SW Simulator



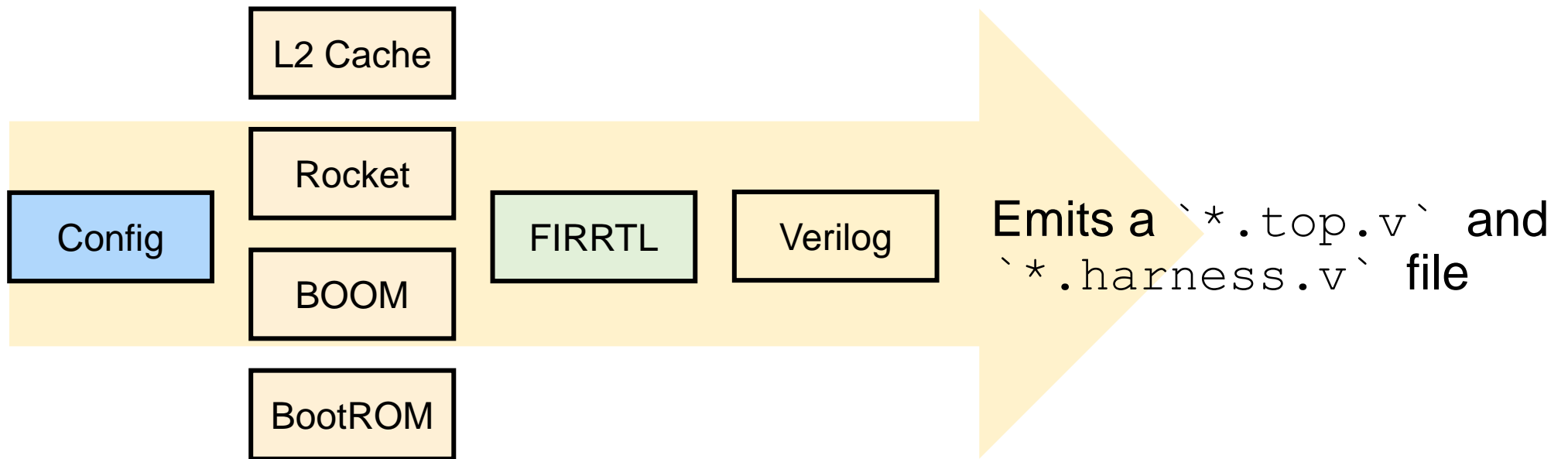
1. Configs parameterize Chisel generators
2. Chisel elaborates into FIRRTL



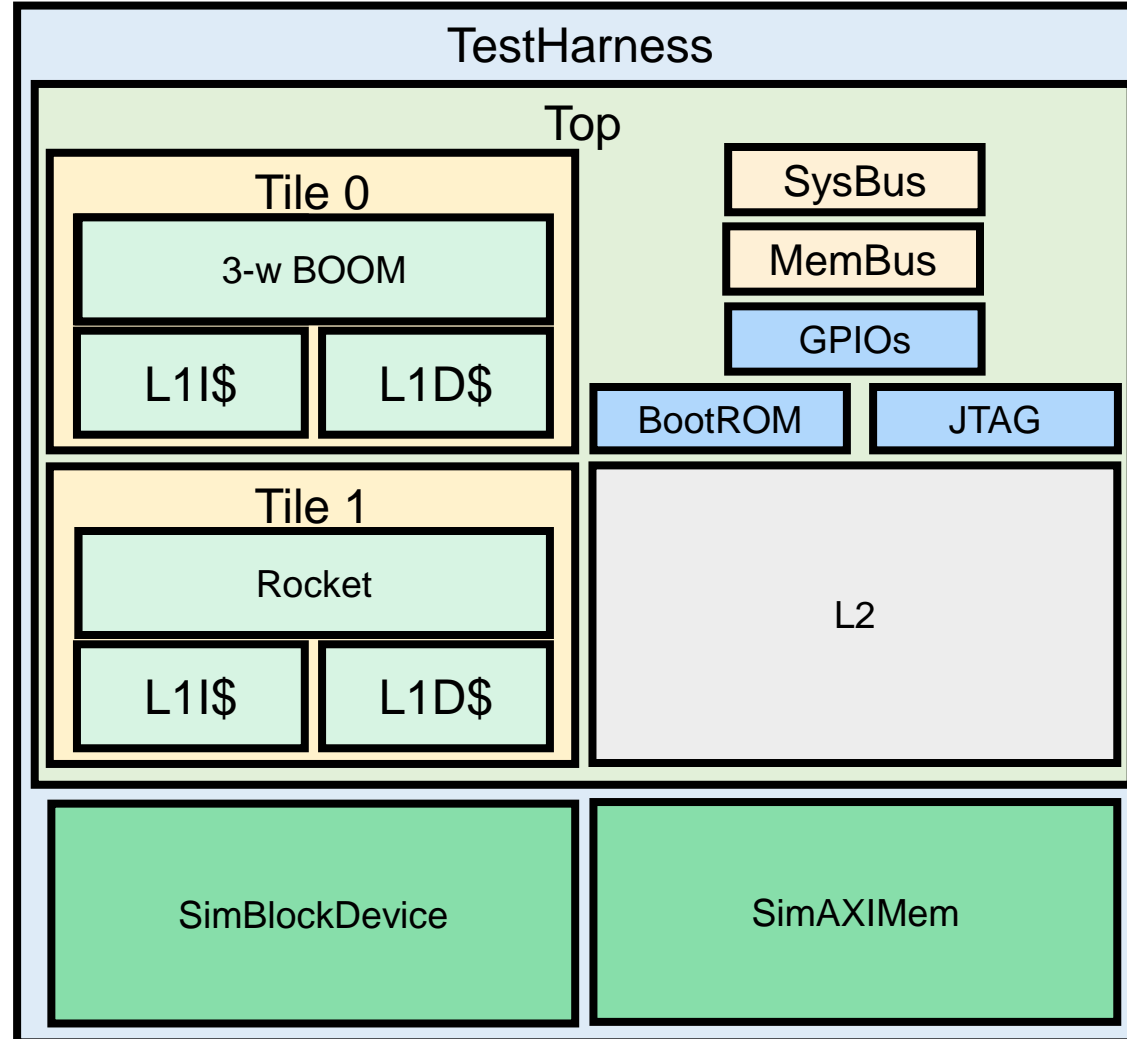
Building a SW Simulator



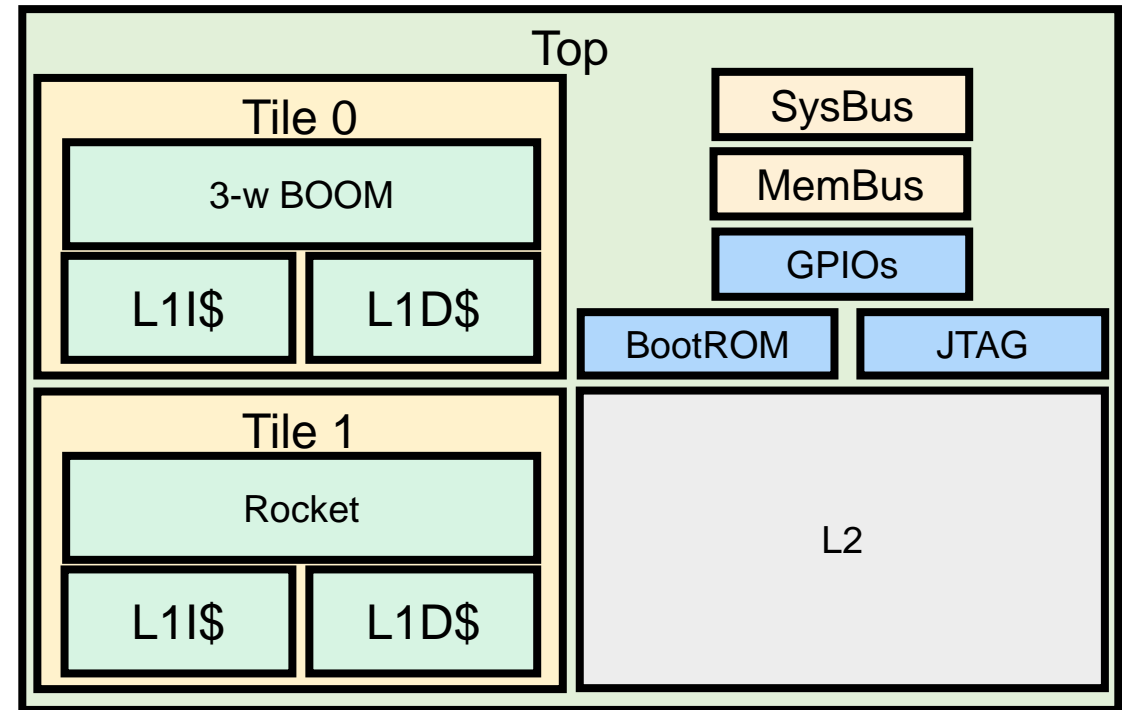
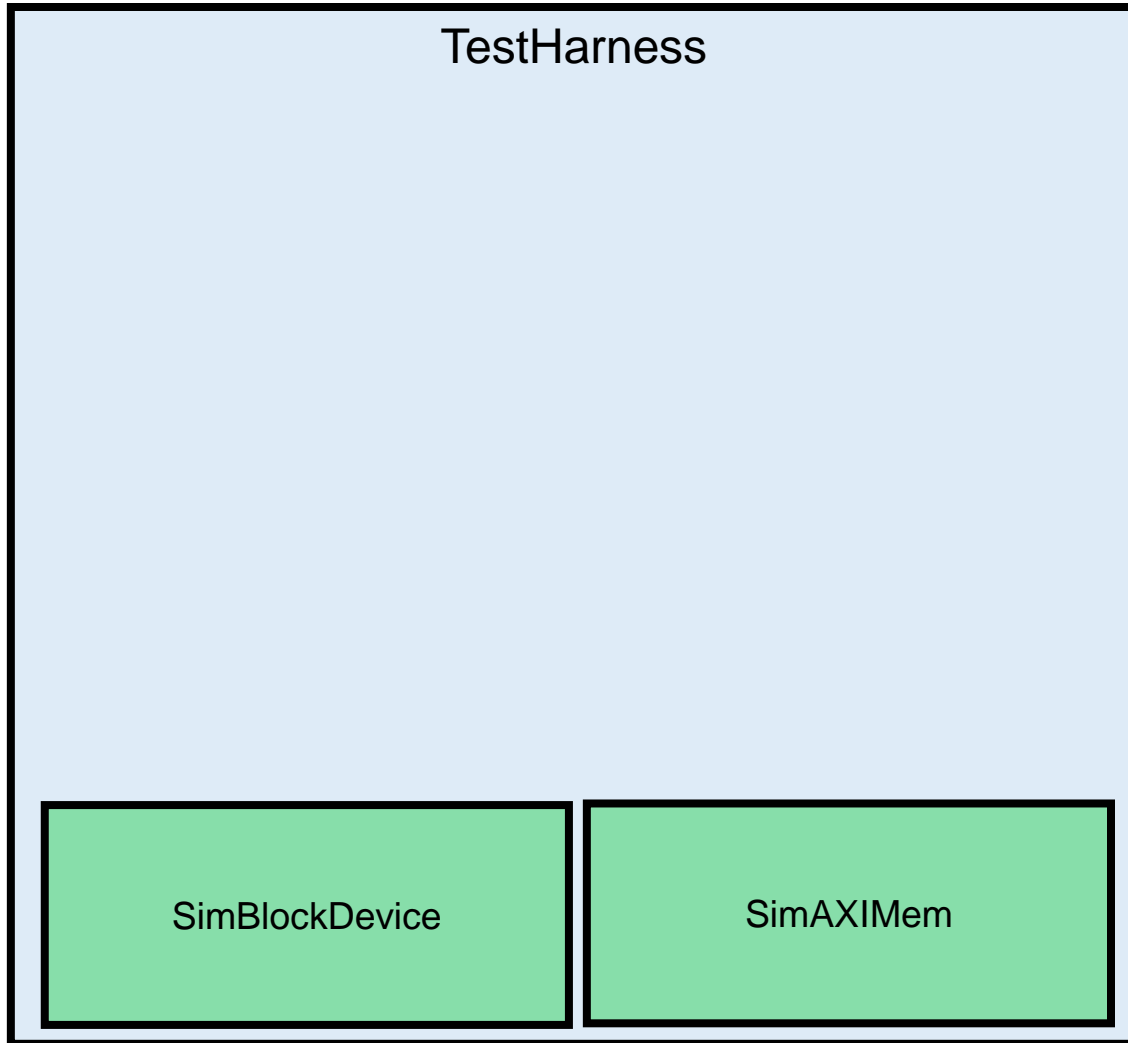
1. Configs parameterize Chisel generators
2. Chisel elaborates into FIRRTL
3. FIRRTL elaborates into Verilog



Why top.v and harness.v



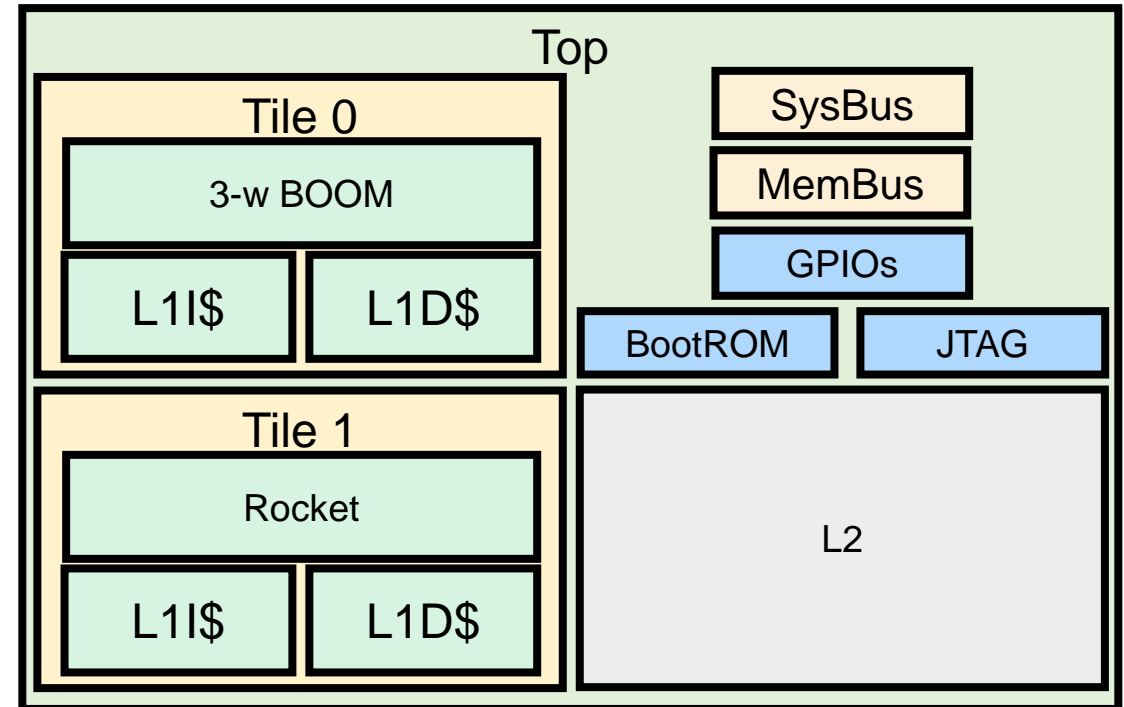
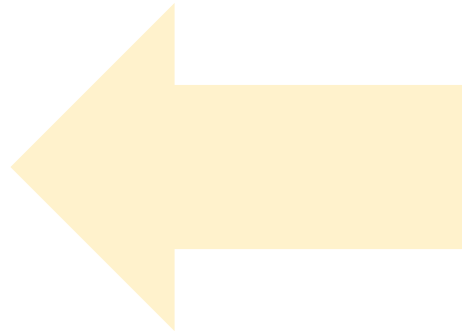
Why top.v and harness.v



Why top.v and harness.v



Passed to the VLSI flow



Building a SW Simulator

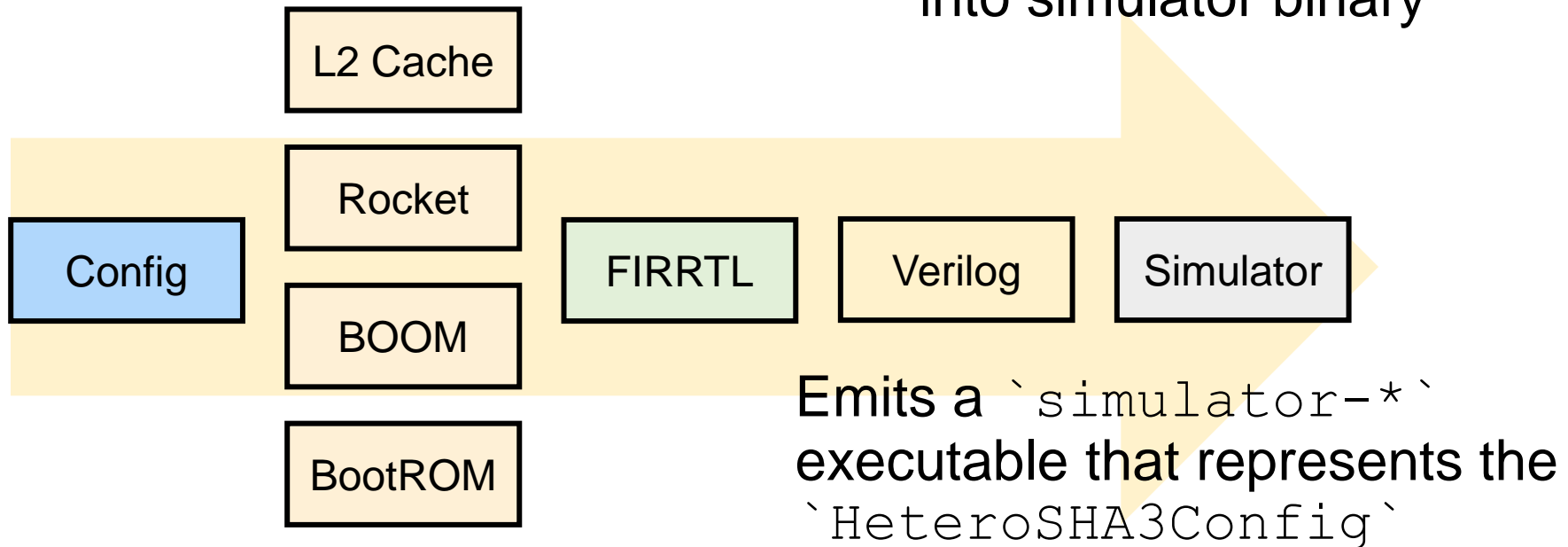


1. Configs parameterize Chisel generators

2. Chisel elaborates into FIRRTL

3. FIRRTL elaborates into Verilog

4. Verilator compiles Verilog into simulator binary



Where is my Verilog!



- Once completed the build outputs are located here

```
> ls generated-src/*/
```

```
generated-src/*/
```

```
*.harness.v
```

Verilog used for testing

```
*.top.v
```

Main Verilog of the design

```
*.dts
```

Device Tree used for SW

```
*.json
```

MMIO files (where and what is connected)

```
...
```

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



Example



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

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

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



How does the Verilog look?



chipyard.TestHarness.TutorialConfig.top.v

```
// Rocket Tile (includes a Rocket Core + L1$'s)
module RocketTile( // @[ :chipyard.TestHarness.HeteroSHA3Config.fir@347233.2]
  input      clock, // @[ :chipyard.TestHarness.HeteroSHA3Config.fir@347234.4]
  input      reset, // @[ :chipyard.TestHarness.HeteroSHA3Config.fir@347235.4]
  . . .
// BOOM Tile (includes a BOOM Core + L1$'s)
module BoomTile( // @[ :chipyard.TestHarness.HeteroSHA3Config.fir@293433.2]
  input      clock, // @[ :chipyard.TestHarness.HeteroSHA3Config.fir@293434.4]
  input      reset, // @[ :chipyard.TestHarness.HeteroSHA3Config.fir@293435.4]
  . . .
// Top-level DUT that includes the BOOM and Rocket Tiles
module ChipTop( // @[ :chipyard.TestHarness.HeteroSHA3Config.fir@384841.2]
  input      clock, // @[ :chipyard.TestHarness.HeteroSHA3Config.fir@384842.4]
  input      reset, // @[ :chipyard.TestHarness.HeteroSHA3Config.fir@384843.4]
```

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



Example



```
# navigate to the output sources
> cd ~/chipyard/sims/verilator
> ls

# run basic RISC-V assembly tests
> make CONFIG=TutorialConfig run-asm-tests

# take a look at an output file
> vim output/chipyard.TestHarness.TutorialConfig/rv64ui-p-simple.out
```

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



Output file



```
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
```

Test passed

Commit log lines emitted from the Rocket Core





Case Study: Configure and Test SHA3-accelerated SoC!



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
- Costly when running on general purpose CPU!
 - Want to improve the hashes/sec and hashes/Watt
- Drastically reduced runtimes with the accelerator

Let's see what integration looks like!



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



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



Example



```
# open rocket configs file
> cd generators/chipyard/src/main/scala/configs
> vim HeteroConfigs.scala
```

```
class TutorialConfig extends Config(
  new boom.common.WithNSmallBooms(1) ++
  new freechips.rocketchip.subsystem.WithNLargeCores(1) ++
  new freechips.rocketchip.subsystem.WithInclusiveCache ++
  new chipyard.config.AbstractConfig)
```

```
class TutorialSHA3Config extends Config(
  new sha3.WithSha3Accel ++
  new TutorialConfig)
```

Add

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



Running customized software on the SHA3 Accelerated SoC



Example



```
# navigate to firemarshal
> cd ~/chipyard/software/firemarshal

# view SHA3 accelerated program
> vim workloads/sha3/benchmarks/src/sha3-rocc.c
```

```
chipyard/
  generators/
    chipyard/
      rocket-chip/
        boom/
          sha3/
  sims/
    verilator/
  software/
    firemarshal/
  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();
```

```
chipyard/
  generators/
    chipyard/
      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

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

Legend: opcode (cyan), rd (green), rs1 (yellow), rs2 (red), funct (orange)



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

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

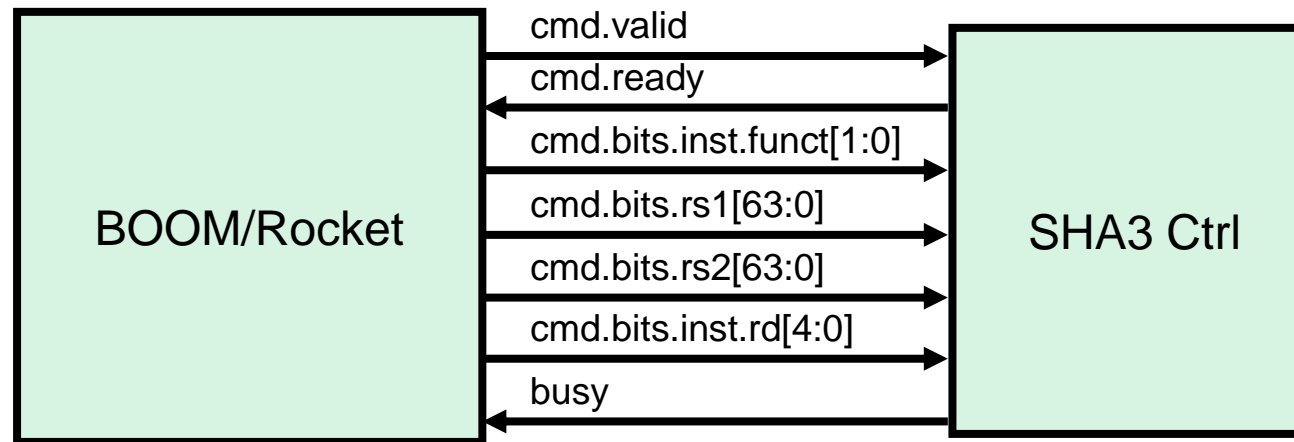
Setup and run the
accelerator



Understanding SHA3 Software



- Send necessary information to the accelerator
 - Source(s), Destination, and what function to run
- Accelerator accesses memory, performs SHA3



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



Example



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

```
chipyard/  
  generators/  
    chipyard/  
      rocket-chip/  
        boom/  
          sha3/  
sims/  
  verilator/  
software/  
  firemarshal/  
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();
```

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



Example



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

```
chipyard/  
  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`



Example



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

# run accelerated program
> make CONFIG=TutorialSHA3Config run-binary-
debug BINARY=../../software/firemarshal/workloads/sh
a3/tests/bare/sha3-rocc.riscv

# run non-accelerated program
> make CONFIG=TutorialSHA3Config run-binary-debug
BINARY=../../software/firemarshal/workloads/sha3/test
s/bare/sha3-sw.riscv
```

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

The non-accelerated version takes longer



More Information



- chipyard.readthedocs.io
 - Talks about heterogeneous SoCs
 - Talks about adding Verilog IP
 - Talks about adding accelerators
 - . . .

Covers what we did in
this talk and more!

Docs » Welcome to Chipyard's documentation! [Edit on GitHub](#)

Welcome to Chipyard's documentation!



Chipyard is 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

