Generating and Simulating SoCs in CHIPYARD

Jerry Zhao
jzh@berkeley.edu
UC Berkeley

Tutorial Outline

1. Generate RTL for a simple SoC
2. Compile simple RISC-V baremetal software
3. Simulate Simple SoC running bare-metal software
4. Generate RTL for a large complex SoC
5. Simulate Complex SoC running bare-metal software

Chipyard Directory Structure

```
chipyard-morning/
  generators/     ← Our library of Chisel generators
    chipyard/
    sha3/
  sims/           ← Utilities for simulating SoCs
    verilator/
    firesim/
  fpga/           ← Utilities for FPGA prototyping
  software/       ← Utilities for building RISC-V software
  vlsi/           ← HAMMER VLSI Flow
  toolchains/     ← RISC-V Toolchain
```

```
> ls $MCYDIR
> tmux new –s buildrtl


> tmux a -t buildrtl
> cd $MCYDIR/generators/chipyard
> cd src/main/scala/config
> vim TutorialConfigs.scala
```

Designing a SoC Config

- Look for **TutorialLeanGemminiConfig** (line 121)
- A Config is composed of Config "fragments"
  - "fragments" set/override/adjust/clear keys in the Config object
  - Generators query this "Config" object at runtime to figure out what to do
- Try it yourself –
  - Set **`use_dedicated_tl_port=false`**
  - Change the number of BOOM/Rocket cores
  - Change the number of L2 banks (try 1/2/4)

Config Fragments In Depth

```
class WithNBanks(n: Int) extends Config((site, here, up) => {
  case BankedL2Key => up(BankedL2Key, site).copy(nBanks = n)
})
```

This fragment changes the **BankedL2Key** to set its **nBanks** field to the desired number of banks.

Config Fragments In Depth

```
class LeanGemminiConfig[T <: Data : Arithmetic, U <: Data, V <: Data](
  gemminiConfig: GemminiArrayConfig[T,U,V] = GemminiConfigs.leanConfig
) extends Config((site, here, up) => {
  case BuildRoCC => up(BuildRoCC) ++ Seq(
    (p: Parameters) => {
      implicit val q = p
      val gemmini = LazyModule(new Gemmini(gemminiConfig))
      gemmini
    }
  )
})
```

This fragment changes the **BuildRoCC** key to return a function that generates a Gemmini instance

```
> cd $MCYDIR/sims/xcelium
> make CONFIG=TutorialLeanGemminiConfig

...

> ls generated-src
> cd generated-src/ … LeanGemminiConfig/
> ls
```

Looking at what is generated

- **`<CONFIG>.dts`**
  - Device tree string - describes to software what's on the SoC
- **`<CONFIG>.fir`**
  - FIRRTL intermediate representation
- **`gen-collateral/`**
  - Directory containing output verilog files, harness files, etc.

```
> cd gen-collateral/
> ls *.cc
> ls *.sv
> less ChipTop.sv
> less TestHarness.sv
```

ChipTop and TestHarness

ChipTop.sv

- Contains definition of the ChipTop
- ChipTop defines a single die and its top-level IO
- This would get passed to VLSI tools as the target module
- This is the DUT for FireSim as well

TestHarness.sv

- Contains definition of the TestHarness
- TestHarness instantiates a ChipTop
- Also instantiates simulation models of I/O devices
  - Ex: model of off-chip DRAM

Compiling Software

Three common approaches

- **bare-metal**
  - No virtual memory
  - No system calls
  - Fast … minimal overhead before your `main` starts
- **proxy-kernel**
  - Compile RISC-V application as you would for Linux
  - PK “proxies” syscalls to the x86 host
  - Virtual memory
  - Slow, and only supports a subset of OS capabilities
- **linux**
  - User binary run as a normal program under OS
  - Run on FPGA prototypes or FireSim

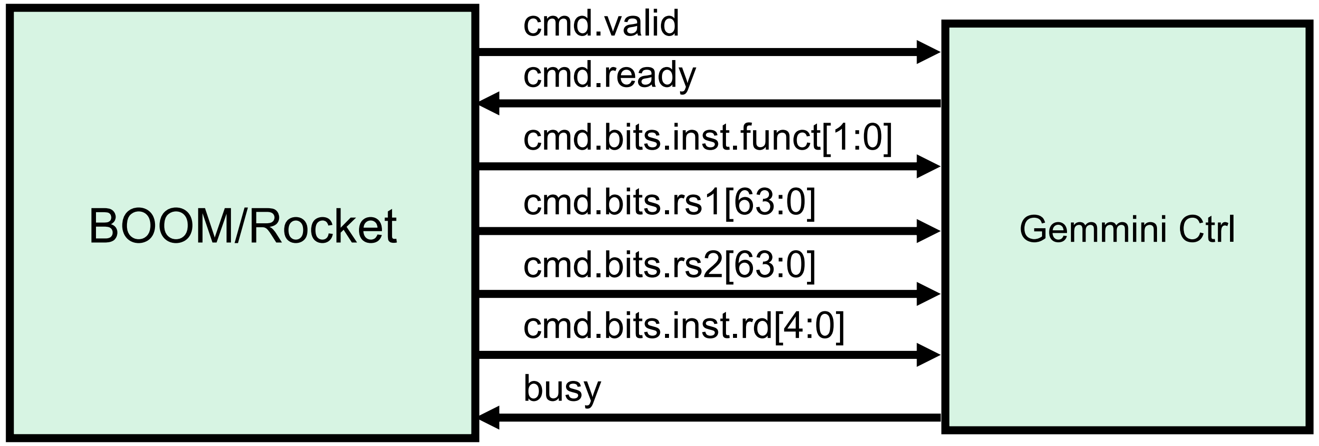
```
> cd $MCYDIR/tests
> make
> spike hello.riscv
> spike mt-hello.riscv
> spike -p4 mt-hello.riscv
```

RoCC Software

- RoCC instructions are typically inserted as inline-assembly in a C program
- For Gemmini, the **gemmini.h** header encapsulates all the inline-assembly, and presents a simple C interface to higher-level software
- Spike ISA simulator extensions can add functional models for new RoCC instructions to spike

RoCC Accelerators

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



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

```
> cd $MCYDIR/tests
> spike mt-gemmini.riscv
> spike –-extension=gemmini mt-gemmini.riscv
> spike --extension=gemmini –p4 mt-gemmini.riscv
```

Running tests on RTL simulators

- Chipyard uses make targets to invoke RTL simulators
  - **`run-binary`** - Runs a binary
  - **`run-binary-debug`** - Runs a binary with waveforms
  - **`run-binary-hex`** - Runs a binary with fast loadmem
  - **`run-binary-hex-debug`** - Runs a binary with fast loadmem + waves
- Make variables specify simulation options
  - **`CONFIG`** - What config to build
  - **`BINARY`** - Run this binary
  - **`timeout_cycles`** - Terminate simulation after this many cycles

```
> cd $MCYDIR/sims/xcelium
> make CONFIG=TutorialLeanGemminiConfig \
    BINARY=$MCYDIR/tests/hello.riscv \
    run-binary-hex

> make CONFIG=TutorialLeanGemminiConfig \
    BINARY=$MCYDIR/tests/mt-hello.riscv \
    run-binary-hex


> make CONFIG=TutorialLeanGemminiConfig \
    BINARY=$MCYDIR/tests/mt-gemmini.riscv \
    run-binary-hex
```

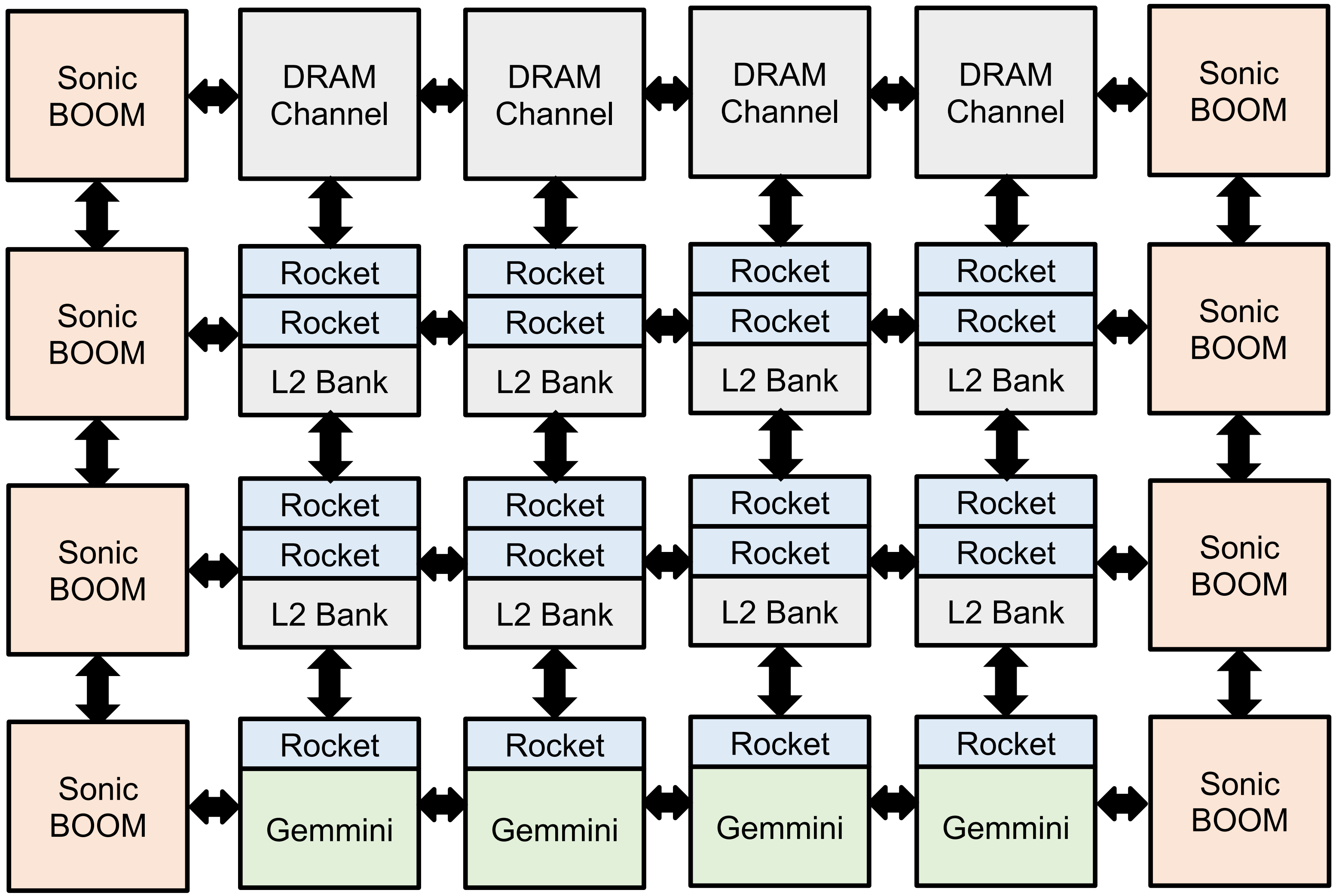
Building Giant SoCs

New in Chipyard 1.9.0 - improved support for very large designs

- **CIRCT** - up to 20x faster FIRRTL compilation
- **CloneTiles** - instantiate a Tile once, then stamp out many copies
- **Constellation** - scalable network-on-chip interconnect generator

```
> cd $MCYDIR/generators/chipyard
> cd src/main/scala/config
> vim TutorialConfigs.scala
```

TutorialManyCoreNoCConfig



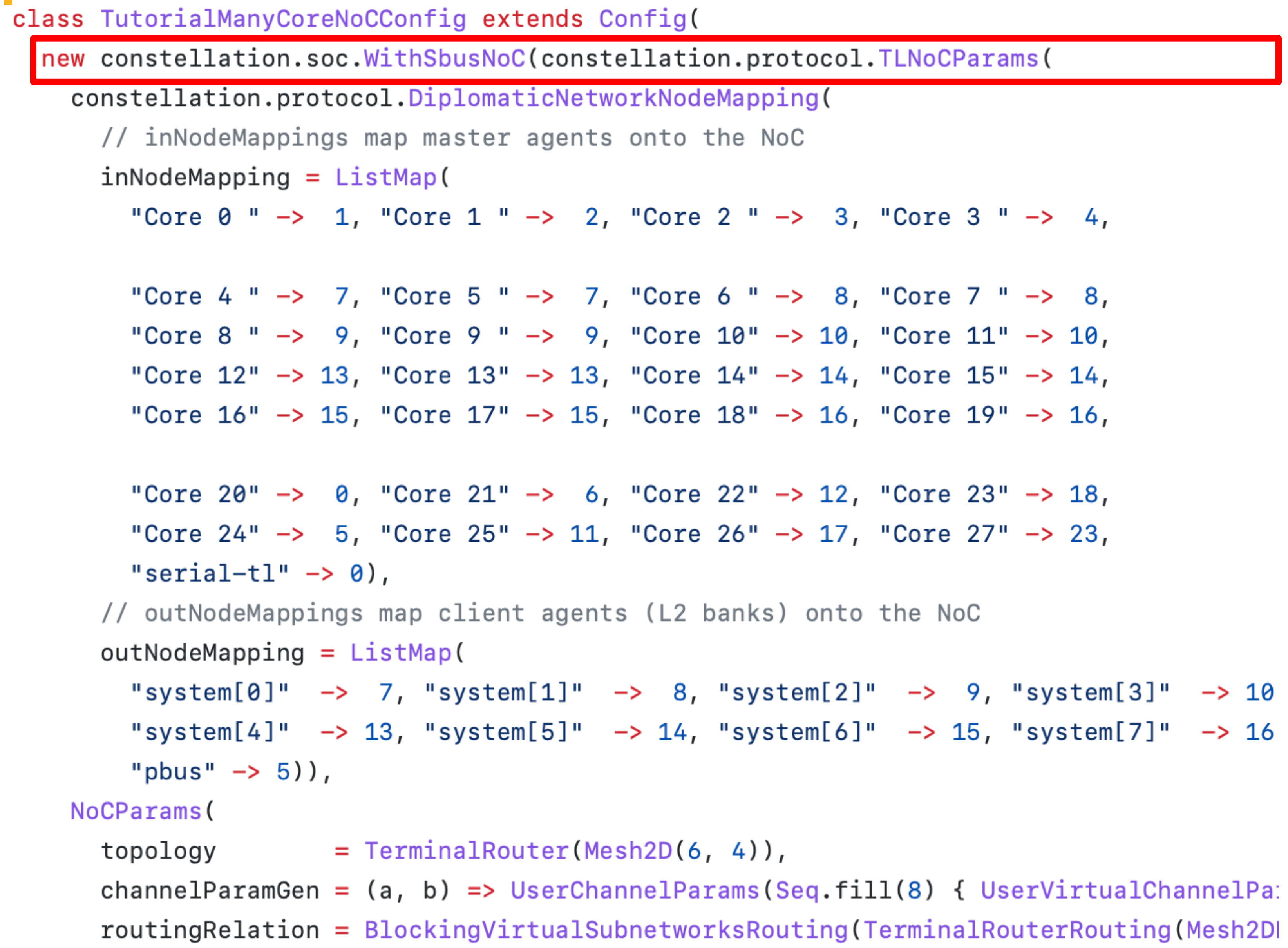
- 6 x 4 mesh NoC
- **4 x** Rocket + Gemmini Accelerator Tiles
- **16 x** Rocket Cores
- **8 x** 10-wide SonicBoom Tiles
- **8 x** Banks of L2 cache

- Total 28 coherent cores

```
> cd $MCYDIR/sims/xcelium
> make CONFIG=TutorialManyCoreNoCConfig


> make CONFIG=TutorialManyCoreNoCConfig \
    run-binary-hex \
    BINARY=../../tests/mt-hello.riscv
```

Configuring the NoC Topology



```
class TutorialManyCoreNoCConfig extends Config(
  new constellation.soc.WithSbusNoC(constellation.protocol.TLNoCParams(
    constellation.protocol.DiplomaticNetworkNodeMapping(
      // inNodeMappings map master agents onto the NoC
      inNodeMapping = ListMap(
        "Core 0 " ->  1, "Core 1 " ->  2, "Core 2 " ->  3, "Core 3 " ->  4,

        "Core 4 " ->  7, "Core 5 " ->  7, "Core 6 " ->  8, "Core 7 " ->  8,
        "Core 8 " ->  9, "Core 9 " ->  9, "Core 10" -> 10, "Core 11" -> 10,
        "Core 12" -> 13, "Core 13" -> 13, "Core 14" -> 14, "Core 15" -> 14,
        "Core 16" -> 15, "Core 17" -> 15, "Core 18" -> 16, "Core 19" -> 16,

        "Core 20" ->  0, "Core 21" ->  6, "Core 22" -> 12, "Core 23" -> 18,
        "Core 24" ->  5, "Core 25" -> 11, "Core 26" -> 17, "Core 27" -> 23,
        "serial-tl" -> 0),
      // outNodeMappings map client agents (L2 banks) onto the NoC
      outNodeMapping = ListMap(
        "system[0]"  ->  7, "system[1]"  ->  8, "system[2]"  ->  9, "system[3]"  -> 10
        "system[4]"  -> 13, "system[5]"  -> 14, "system[6]"  -> 15, "system[7]"  -> 16
        "pbus" -> 5)),
    NoCParams(
      topology        = TerminalRouter(Mesh2D(6, 4)),
      channelParamGen = (a, b) => UserChannelParams(Seq.fill(8) { UserVirtualChannelPa
      routingRelation = BlockingVirtualSubnetworksRouting(TerminalRouterRouting(Mesh2D
```

Request that the System Bus (SBus) should be a NoC

Configuring the NoC Topology

```
class TutorialManyCoreNoCConfig extends Config(
  new constellation.soc.WithSbusNoC(constellation.protocol.TLNoCParams(
    constellation.protocol.DiplomaticNetworkNodeMapping(
      // inNodeMappings map master agents onto the NoC
      inNodeMapping = ListMap(
        "Core 0 " ->  1, "Core 1 " ->  2, "Core 2 " ->  3, "Core 3 " ->  4,

        "Core 4 " ->  7, "Core 5 " ->  7, "Core 6 " ->  8, "Core 7 " ->  8,
        "Core 8 " ->  9, "Core 9 " ->  9, "Core 10" -> 10, "Core 11" -> 10,
        "Core 12" -> 13, "Core 13" -> 13, "Core 14" -> 14, "Core 15" -> 14,
        "Core 16" -> 15, "Core 17" -> 15, "Core 18" -> 16, "Core 19" -> 16,

        "Core 20" ->  0, "Core 21" ->  6, "Core 22" -> 12, "Core 23" -> 18,
        "Core 24" ->  5, "Core 25" -> 11, "Core 26" -> 17, "Core 27" -> 23,
        "serial-tl" -> 0),
      // outNodeMappings map client agents (L2 banks) onto the NoC
      outNodeMapping = ListMap(
        "system[0]"  ->  7, "system[1]"  ->  8, "system[2]"  ->  9, "system[3]"  -> 10
        "system[4]"  -> 13, "system[5]"  -> 14, "system[6]"  -> 15, "system[7]"  -> 16
        "pbus" -> 5)),
    NoCParams(
      topology        = TerminalRouter(Mesh2D(6, 4)),
      channelParamGen = (a, b) => UserChannelParams(Seq.fill(8) { UserVirtualChannelPa
      routingRelation = BlockingVirtualSubnetworksRouting(TerminalRouterRouting(Mesh2D
```

Map agents onto physical on the NoC

Configuring the NoC Topology

```
class TutorialManyCoreNoCConfig extends Config(
  new constellation.soc.WithSbusNoC(constellation.protocol.TLNoCParams(
    constellation.protocol.DiplomaticNetworkNodeMapping(
      // inNodeMappings map master agents onto the NoC
      inNodeMapping = ListMap(
        "Core 0 " ->  1, "Core 1 " ->  2, "Core 2 " ->  3, "Core 3 " ->  4,

        "Core 4 " ->  7, "Core 5 " ->  7, "Core 6 " ->  8, "Core 7 " ->  8,
        "Core 8 " ->  9, "Core 9 " ->  9, "Core 10" -> 10, "Core 11" -> 10,
        "Core 12" -> 13, "Core 13" -> 13, "Core 14" -> 14, "Core 15" -> 14,
        "Core 16" -> 15, "Core 17" -> 15, "Core 18" -> 16, "Core 19" -> 16,

        "Core 20" ->  0, "Core 21" ->  6, "Core 22" -> 12, "Core 23" -> 18,
        "Core 24" ->  5, "Core 25" -> 11, "Core 26" -> 17, "Core 27" -> 23,
        "serial-tl" -> 0),
      // outNodeMappings map client agents (L2 banks) onto the NoC
      outNodeMapping = ListMap(
        "system[0]"  ->  7, "system[1]"  ->  8, "system[2]"  ->  9, "system[3]"  -> 10
        "system[4]"  -> 13, "system[5]"  -> 14, "system[6]"  -> 15, "system[7]"  -> 16
        "pbus" -> 5)),
    NoCParams(
      topology        = TerminalRouter(Mesh2D(6, 4)),
      channelParamGen = (a, b) => UserChannelParams(Seq.fill(8) { UserVirtualChannelPa
      routingRelation = BlockingVirtualSubnetworksRouting(TerminalRouterRouting(Mesh2D
```

Specify topology/routing of NoC

```
> cd $MCYDIR/sims/xcelium
> make CONFIG=TutorialManyCoreNoCConfig



> make CONFIG=TutorialManyCoreNoCConfig \
    run-binary-hex \
    BINARY=../../tests/mt-hello.riscv
```

Lots more to try in Chipyard as well

- Chipyard reduces the complexity of SoC design-space-exploration
- Try different core/accelerator/memory-system/configurations
- **New in Chipyard 1.9.0: Config discovery**
  - Lists most of the available Config options across all Chipyard packages
  - `make find-config-fragments`

```
> cd $MCYDIR/sims/xcelium
> make find-config-fragments
```