OpenSoC Fabric
An open source, parameterized, network generation tool
Farzad Fatollahi-Fard, Dave Donofrio, George Michelogiannakis, John Shalf

8th International Symposium on Networks-on-Chip (NOCS)
September 17-19, 2014. Ferrara, Italy.
<table>
<thead>
<tr>
<th>Motivation</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chisel Overview</td>
<td>2</td>
</tr>
<tr>
<td>Scala Crash Course</td>
<td>3</td>
</tr>
<tr>
<td>Chisel Deep Dive</td>
<td>4</td>
</tr>
<tr>
<td>Interactive Session</td>
<td>5</td>
</tr>
<tr>
<td>OpenSoC Walk Through</td>
<td>6</td>
</tr>
<tr>
<td>Using OpenSoC</td>
<td>7</td>
</tr>
<tr>
<td>Interactive Session</td>
<td>8</td>
</tr>
<tr>
<td>Future Work</td>
<td>9</td>
</tr>
<tr>
<td>Conclusion</td>
<td>10</td>
</tr>
</tbody>
</table>
Meet the OpenSoC Team

- Farzad Fatollahi-Fard
- David Donofrio
- George Michelogiannakis
- John Shalf

- Berkeley National Lab
- CoDEx
- CAL
A Radical Shift for the Future of Scientific Applications

“…exascale computing (will) revolutionize our approaches to global challenges in energy, environmental sustainability, and security.”

- E3 Report
Power: The New Design Constraint
Trends beginning in 2004 are continuing…

- Power densities have ceased to increase
- No power efficiency increase with smaller transistors
Power: The New Design Constraint
On-chip parallelism increasing to maintain performance increases...

- We have come to the end of clock frequency scaling
- Moore’s Law is alive and well
  - Now seeing core count increasing

Peter Kogge (DARPA 2008 “Exascale Challenges” Report)
<table>
<thead>
<tr>
<th></th>
<th>Franklin</th>
<th>Hopper</th>
<th>Edison</th>
<th>Cori (NERSC 8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Count</td>
<td>4</td>
<td>24</td>
<td>48 (logical)</td>
<td>&gt;60</td>
</tr>
<tr>
<td>Clock Rate</td>
<td>2.3GHz</td>
<td>2.1GHz</td>
<td>2.4 GHz</td>
<td>~1.5GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>8GB</td>
<td>32GB</td>
<td>64GB</td>
<td>64-128GB +On package</td>
</tr>
<tr>
<td>Peak Perf</td>
<td>.352 PF</td>
<td>1.288 PF</td>
<td>2.57 PF</td>
<td>&gt; 3 TF</td>
</tr>
</tbody>
</table>
Hierarchical Power Costs

Data movement is the dominant power cost

- **6 pJ**: Cost to move data 1 mm on-chip
- **100 pJ**: Typical cost of a single floating point operation
- **120 pJ**: Cost to move data 20 mm on chip
- **250 pJ**: Cost to move off-chip, but stay within the package (SMP)
- **2000 pJ**: Cost to move data off chip into DRAM
- **~2500 pJ**: Cost to move data off chip to a neighboring node
What Interconnect Provides the Best Power / Performance Ratio?

What tools exist to answer this question?
What tools exist for SoC research

What tools do we have to evaluate large, complex networks of cores?

- **Software models**
  - Fast to create, but plagued by long runtimes as system size increases

- **Hardware emulation**
  - Fast, accurate evaluate that scales with system size but suffers from long development time

A complexity-effective architecture for accelerating full-system multiprocessor simulations using FPGAs. FPGA 2008
Booksims
Cycle-accurate on-chip network simulator

- C++
- Cycle-accurate
- Verified against RTL
- Long runtimes limit simulation size
  - Few thousand cycles per second

A detailed and flexible cycle-accurate network-on-chip simulator. ISPASS 2013
Garnet
Event-driven on-chip network simulator

- C++
- Event-driven
- Verified against other network simulators
- Still not fast enough for thousand cores

GARNET: A detailed on-chip network model inside a full-system simulator. ISPASS 2009
Open-source NoC router RTL

- Parameterized Verilog
  - Configuration can be difficult
  - Adding new features high effort

- High effort for development

- Verilog simulation does not scale

```verilog
localparam flit_ctrl_width = (packet_format == `PACKET_FORMAT_HEAD_TAIL) ? (1 + vc_idx_width + 1 + 1) : (packet_format == `PACKET_FORMAT_TAIL_ONLY) ? (1 + vc_idx_width + 1) : (packet_format == `PACKET_FORMAT_EXPLICIT_LENGTH) ? (1 + vc_idx_width + 1) : -1;
```

https://noc.s.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/Router
Connect: config network creation

Hardware generator based on input parameters

- Verilog generator
- Optimized for FPGA based networks
- Highly configurable
  - Pre-defined options
  - Generator code in Bluspec

CONNECT: fast flexible FPGA-tuned networks-on-chip.
CARL 2012
Chisel: A New Hardware DSL
Using Scala to construct Verilog and C++ descriptions

- **Chisel provides both software and hardware models from the same codebase**
- **Object-oriented hardware development**
  - Allows definition of structs and other high-level constructs
- **Powerful libraries and components ready to use**
- **Working processors fabricated using chisel**
Recent Chisel Designs

Chisel code successfully boots Linux

• First tape-out in 2012
• Raven core just taped out in 2014 – 28nm
Chisel Overview

How does Chisel work?

- Not “Scala to Gates”
- Describe hardware functionality
- Chisel creates graph representation
  - Flattened
- Each node translated to Verilog or C++
Chisel Overview

How does Chisel work?

- All bit widths are inferred
- Clock and reset implied
  - Multiple clock domains possible
- IOs grouped into convenient bundles

```scala
class Max2 extends Module {
  val io = new Bundle {
    val x = UInt(INPUT, 8)
    val y = UInt(INPUT, 8)
    val z = UInt(OUTPUT, 8)
  }
  io.z := Mux(io.x > io.y, io.x, io.y)
}
```
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motivation</td>
<td>1</td>
</tr>
<tr>
<td>Chisel Overview</td>
<td>2</td>
</tr>
<tr>
<td>Scala Crash Course</td>
<td>3</td>
</tr>
<tr>
<td>Chisel Deep Dive</td>
<td>4</td>
</tr>
<tr>
<td>Interactive Session</td>
<td>5</td>
</tr>
<tr>
<td>OpenSoC Walk Through</td>
<td>6</td>
</tr>
<tr>
<td>Using OpenSoC</td>
<td>7</td>
</tr>
<tr>
<td>Interactive Session</td>
<td>8</td>
</tr>
<tr>
<td>Future Work</td>
<td>9</td>
</tr>
<tr>
<td>Conclusion</td>
<td>10</td>
</tr>
</tbody>
</table>
Scala Crash Course

- Developed specifically for DSLs
- Strong typing
- Large community
- Object Oriented
- Functional Semantics
- Compiled to JVM
Scala Basics

Variables, types

- **var vs val**

- **Common types**
  - Byte, Char, Int, Long, Float, Double

- **All types are classes**
  - So support operators, such as:
    - 1.to(10) -> (1,2,3,4,5,6,7,8,9,10)
  - `obj.method(arg)` *is equivalent to* `obj method arg`
Scala Basics
Control Structures

- **If / else**
  - If( x > 0) /* do stuff */ else /*do something else*/

- **For**
  - for (i <- 0 to n) // i traverses all values, including n
  - for (i <- 0 until n) // i traverses all values up to n-1
  - for (i <- "NoCs are Cool") // i traverses all values in an index or array
Scala basics
Control structures

- **While**
  - while (n > 0){ /* do something */ }

- **More interesting loop...**
  - for (i<-1 to 3; from = 4 – i; j <- from to 3)
    print( (10 * i + j ) + " ")
  - Prints: 13 22 23 31 32 33
  - Note the need for semicolons
Scala Basics
Function calls

- **Functions**
  - Last line is return function
  - def factorial( n : Int ) : Int =
    
    `{
    var r = 1
    for ( i <- 1 to n ) r = r * i
    r
    }

- Function calls
Scala Basics
Arrays and Maps

- **Fixed Size Arrays**
  - Declared as:
    - `val a = new Array[String][10]` //”10” is the size of the array
  - Accessed as
    - `a(3)`

- **Variable Sized Arrays – Array Buffers**
  - `val b = new ArrayBuffer[Int]()`
  - Using:
    - Insert, remove, trim, etc functions available
    - `b += 3` //add element to the end
Scala Basics

Arrays and Maps

- **Maps**
  - `val myMap = (key1 -> value1, key2 -> value2)`
  - `val myMutableMap = collection.mutable.Map(key1 -> val…`
  - `Val myEmptyMap = new collection.mutable.HashMap[KeyType][ValueType]`
  - Access as: `myMap(key)`

- **Interesting functions for arrays (and other collections)**
  - `sum()` `product()` `sortWith()`
Scala Basics
Classes and Objects

- Classes
  - Default to public
  - Get / set functions auto created
  - class Counter {
    
    private var Value : Int = 0
    
    def increment() { value += 1 }
  
    def current() = value
  }

- Use
  - myCounter.increment()
  - myCounter.current
Scala Basics

Objects

- Objects are singletons
  - Defines a single instance of a class with features you define
  - Often are *companion objects* to an identically named class
  - Example below will create a new unique account number

```scala
object Accounts {
  private var lastNumber = 0
  def newUniqueNumber() {lastNumber += 1; lastNumber}
}
```
Scala Basics
A few more things to know…

- Inheritance supported through `extends` keyword
- *Abstract classes* can be created to enforce an interface in derived classes
  - Think virtual functions in C++
- Types inferred at runtime but can be checked using `.isInstanceOf`
- Casting can be done using the `.asInstanceOf`
Scala basics

A few gotchas…

› Type is always written after the variable

  • `val myStr : String = "NoCs are cool"`
  • But type is typically not required – it is inferred by the compiler

› The `apply` function

  • "NoCs are cool"(2) returns "C"

› No ternary function

  • if / else used in place since "if" statement returns a value

› No semicolons needed (usually)
Scala Exercises

Use the Scala REPL to try out a few Scala concepts

- <This is highly time dependent – would require participants to get their VM up and running. This may be a better time to do this than during the OpenSoC Example session>
Chisel Crash Course

A few things to remember…

- Chisel acts as a hardware generator
  - Think about how data would flow through the hardware described
  - This is a different mindset then describing an algorithm with C/C++

```scala
class Max2 extends Module {
  val io = new Bundle {
    val x = UInt(INPUT, 8)
    val y = UInt(INPUT, 8)
    val z = UInt(OUTPUT, 8)
  }
  io.z := Mux(io.x > io.y, io.x, io.y)
}
```
Chisel Basics
Some preliminary concepts

- **Modules** are at the top of the hierarchy
  - Similar to Verilog modules
  - Can create sub-modules

- **Wires used to connect Modules are Bundles**
  - Compile time configurable
  - Do not hold state, simply specify an interface

- **Simple state elements** – Reg, Mem, Queue included
Chisel Deep Dive

Basic Data Types

- `Bool([x:Boolean])`

- `Bits/UInt/SInt([x:Int/String], [width:Int])`
  - `x (optional)` create a literal from Scala type/ passed String, or declare unassigned if missing
  - `width (optional)` bit width (inferred if missing)
Chisel Deep Dive
Aggregate Types

- **Vec**
  - Indexable vector of *Data* types
  - `val myVec = Vec(elts:Iterable[Data])`
    - `elts` - initial element *Data* (vector depth inferred)
  - `val myVec = Vec.fill(n:Int) {gen:Data}`
    - `n` - vector depth (elements)
    - `gen` - initial element *Data*, called once per element

- **Usage**
  - Elements can be dynamically or statically indexing
    - `readVal := myVec(ind:Data/idx:Int)`
    - `myVec(ind:Data/idx:Int) := writeVal`
Chisel Deep Dive

Aggregate Types

- **Bundle**
  - Contains *Data* types indexed by name
    - class MyBundle extends Bundle {
      val a = Bool ()
      val b = UInt(width = 32)
    }

- **Using**

```
val my_bundle = new MyBundle()
val bundleVal = my_bundle.a
my_bundle.a := Bool(true)
```
Chisel Deep Dive

Chisel Wire Operators

- `val x = UInt()`
  - Allocate as wire of type `UInt()`

- `x := y`
  - Assign (connect) wire `y` to wire `x`

- `x <> y`
  - Connect `x` and `y` (mostly for aggregate types)
  - Wire directionality is automatically inferred
Chisel Deep Dive
Conditional Operators

- **When**
  - Executes blocks conditionally by Bool
  - Equivalent to Verilog `if`

- **Switch**
  - Executes blocks conditionally by data

```
when (condition1) {
  // run if condition1 true and skip rest
}
.elsewhen (condition2) {
  // run if condition2 true and skip rest
}
.unless (condition3) {
  // run if condition3 false and skip rest
}
.otherwise {
  // run if none of the above ran
}

switch(x) {
  is(value1) {
    // run if x === value1
  }
  is(value2) {
    // run if x === value2
  }
}
```
# Chisel Deep Dive

## Bool Operators

<table>
<thead>
<tr>
<th>Chisel</th>
<th>Explanation</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>!x</td>
<td>Logical NOT</td>
<td>1</td>
</tr>
<tr>
<td>x &amp;&amp; y</td>
<td>Logical AND</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>y</td>
</tr>
<tr>
<td>Chisel</td>
<td>Explanation</td>
<td>Width</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>x(n)</td>
<td>Extract bit, 0 is LSB</td>
<td>1</td>
</tr>
<tr>
<td>x(n, m)</td>
<td>Extract bitfield</td>
<td>n - m + 1</td>
</tr>
<tr>
<td>x &lt;&lt; y</td>
<td>Dynamic left shift</td>
<td>Width(x)+MaxVal(y)</td>
</tr>
<tr>
<td>x &gt;&gt; y</td>
<td>Dynamic right shift</td>
<td>Width(x)-MaxVal(y)</td>
</tr>
<tr>
<td>x &lt;&lt; n</td>
<td>Static left shift</td>
<td>Width(x) + n</td>
</tr>
<tr>
<td>x &gt;&gt; n</td>
<td>Static right shift</td>
<td>Width(x) - n</td>
</tr>
<tr>
<td>Fill(n, x)</td>
<td>Replicate x, n times</td>
<td>n * Width(x)</td>
</tr>
<tr>
<td>Cat(x, y)</td>
<td>Concatenate bits</td>
<td>Width(x) + Width(y)</td>
</tr>
<tr>
<td>Mux(c, x, y)</td>
<td>If c, then x; else y</td>
<td>MaxWidth(x, y)</td>
</tr>
</tbody>
</table>
# Chisel Deep Dive

## Bits Operators

<table>
<thead>
<tr>
<th>Chisel</th>
<th>Explanation</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>(~x)</td>
<td>Bitwise NOT</td>
<td>Width(x)</td>
</tr>
<tr>
<td>(x &amp; y)</td>
<td>Bitwise AND</td>
<td>MaxWidth(x, y)</td>
</tr>
<tr>
<td>(x \mid y)</td>
<td>Bitwise OR</td>
<td>MaxWidth(x, y)</td>
</tr>
<tr>
<td>(x ^\wedge y)</td>
<td>Bitwise XOR</td>
<td>MaxWidth(x, y)</td>
</tr>
<tr>
<td>(x === y)</td>
<td>Equality</td>
<td>1</td>
</tr>
<tr>
<td>(x != y)</td>
<td>Inequality</td>
<td>1</td>
</tr>
<tr>
<td>(\text{andR}(x))</td>
<td>AND-reduce</td>
<td>1</td>
</tr>
<tr>
<td>(\text{orR}(x))</td>
<td>OR-reduce</td>
<td>1</td>
</tr>
<tr>
<td>(\text{xorR}(x))</td>
<td>XOR-reduce</td>
<td>1</td>
</tr>
</tbody>
</table>
## Chisel Deep Dive

### UInt/SInt Operators

<table>
<thead>
<tr>
<th>Chisel</th>
<th>Explanation</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>x + y</td>
<td>Bitwise NOT</td>
<td>MaxWidth(x, y)</td>
</tr>
<tr>
<td>x - y</td>
<td>Bitwise AND</td>
<td>MaxWidth(x, y)</td>
</tr>
<tr>
<td>x * y</td>
<td>Bitwise OR</td>
<td>Width(x) + Width(y)</td>
</tr>
<tr>
<td>x / y</td>
<td>Bitwise XOR</td>
<td>Width(x)</td>
</tr>
<tr>
<td>x % y</td>
<td>Equality</td>
<td>MaxVal(y) - 1</td>
</tr>
<tr>
<td>x &gt; y</td>
<td>Inequality</td>
<td>1</td>
</tr>
<tr>
<td>x &gt;= y</td>
<td>AND-reduce</td>
<td>1</td>
</tr>
<tr>
<td>x &lt; y</td>
<td>OR-reduce</td>
<td>1</td>
</tr>
<tr>
<td>x &lt;= y</td>
<td>XOR-reduce</td>
<td>1</td>
</tr>
</tbody>
</table>
## Chisel Deep Dive
### UInt/SInt Operators

<table>
<thead>
<tr>
<th>Chisel</th>
<th>Explanation</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x &gt;&gt; y$</td>
<td>Arithmetic right shift</td>
<td>$\text{Width}(x) - \text{MinVal}(y)$</td>
</tr>
<tr>
<td>$x &gt;&gt; n$</td>
<td>Arithmetic right shift</td>
<td>$\text{Width}(x) - n$</td>
</tr>
</tbody>
</table>
Chisel Deep Dive
State Elements

- **Registers**
  - `val my_reg = Reg([outType:Data], [next:Data], [init:Data])`
  - `outType (optional)` - register type (or inferred)
  - `next (optional)` - update value every clock
  - `init (optional)` - initialization value on reset

- **Updating**
  - `my_reg := next_val`
  
  Assign to latch new value on next clock
Chisel Deep Dive
State Elements

- **Read-Write Memory**
  - val my_mem = Mem(out:Data, n:Int, seqRead:Boolean)
    out - memory element type
    n - memory depth (elements)
    seqRead - only update reads on clock edge

- **Using**
  - Reads: val readVal = mem(addr:UInt/Int)
  - Writes: mem(addr:UInt/Int) := y
Chisel Deep Dive

Creating a Module

- Class must extend Module class
  - Abstract classes legal
- IO ports listed as a Bundle with directions and widths specified

```scala
class Max2 extends Module {
  val io = new Bundle {
    val x = UInt(INPUT, 8)
    val y = UInt(INPUT, 8)
    val z = UInt(OUTPUT, 8)
  }
  io.z := Mux(io.x > io.y, io.x, io.y)
}
```
Chisel Deep Dive

Connecting Modules – Simplest way is to assign each individual element in the IO bundle

```scala
val m1 = Module(new Max2())
m1.io.x := a
m1.io.y := b
val m2 = Module(new Max2())
m2.io.x := c
m2.io.y := d
val m3 = Module(new Max2())
m3.io.x := m1.io.z
m3.io.y := m2.io.z
```
Using the Chisel bulk connection interface to connect routers

```scala
class Router extends Module {
  val io = new Bundle {
    val InChannel = new Channel()
    val OutChannels = new Channel()
  }
}

class Network extends Module {
  val Router1 = new Router
  val Router2 = new Router

  Router1.io.inChannel <> Router2.io.outChannel
  Router1.io.outChannel <> Router2.io.inChannel
```
Chisel Deep Dive

Hardware Generation

- **Functions**
  - Provide block abstractions for code
    - ```python
def Adder(op_a:UInt, op_b:UInt): UInt = { op_a + op_b }
    ```
  - Hardware is instantiated when called
    - ```python
    sum := Adder(UInt(1), some_data)
    ```

- **If/For**
  - Used to control hardware generation
  - Equivalent to Verilog `generate if/for`
Chisel Deep Dive
Standard Library

- **Function Blocks**
  - Stateless: UIntToOH, OHToUInt, Reverse, PopCount, etc
  - Stateful: LFSR16, ShiftRegister

- **Interfaces**
  - DecoupledIO, ValidIO, Queue, Pipe, Arbiter
Chisel Deep Dive

Tester

- Class with functions for testing Modules, connecting and communicating with a simulator
  - reset
  - step
  - poke
  - peek
  - expect
<table>
<thead>
<tr>
<th>Motivation</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chisel Overview</td>
<td>2</td>
</tr>
<tr>
<td>Scala Crash Course</td>
<td>3</td>
</tr>
<tr>
<td>Chisel Deep Dive</td>
<td>4</td>
</tr>
<tr>
<td>Interactive Session</td>
<td>5</td>
</tr>
<tr>
<td>OpenSoC Walk Through</td>
<td>6</td>
</tr>
<tr>
<td>Using OpenSoC</td>
<td>7</td>
</tr>
<tr>
<td>Interactive Session</td>
<td>8</td>
</tr>
<tr>
<td>Future Work</td>
<td>9</td>
</tr>
<tr>
<td>Conclusion</td>
<td>10</td>
</tr>
</tbody>
</table>
Build a NxN Switch

- Open VM
- Navigate to
  - ~/opensoc-demo/switch-demo/
- Switch Code and tester in:
  - src/main/scala/switch.scala
- To test your code run:
  sbt run
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motivation</td>
<td>1</td>
</tr>
<tr>
<td>Chisel Overview</td>
<td>2</td>
</tr>
<tr>
<td>Scala Crash Course</td>
<td>3</td>
</tr>
<tr>
<td>Chisel Deep Dive</td>
<td>4</td>
</tr>
<tr>
<td>Interactive Session</td>
<td>5</td>
</tr>
<tr>
<td>OpenSoC Walk Through</td>
<td>6</td>
</tr>
<tr>
<td>Using OpenSoC</td>
<td>7</td>
</tr>
<tr>
<td>Interactive Session</td>
<td>8</td>
</tr>
<tr>
<td>Future Work</td>
<td>9</td>
</tr>
<tr>
<td>Conclusion</td>
<td>10</td>
</tr>
</tbody>
</table>
OpenSoC Fabric

An open source, flexible, parameterized, NoC generator

- Part of the CoDEx tool suite, written in Chisel
- Dimensions, topology, VCs all configurable
- Fast functional C++ model for functional validation
  - SystemC ready
- Verilog based description for FPGA or ASIC
  - Synthesis path enables accurate power / energy modeling
- AXI Based endpoints
  - Ready for ARM integration
OpenSoC Fabric
An open source, flexible, parameterized, NoC generator
OpenSoC: Current Status
Projected v1.0 release date of October 1st

- **On your Flash Drive:**
  - 2-D concentrated mesh network of arbitrary size
  - Wormhole routing

- **Included in 1.0 Release**
  - Virtual Channels
  - AXI Interface
  - Additional Topologies
OpenSoC – Top Level Diagram
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motivation</td>
<td>1</td>
</tr>
<tr>
<td>Chisel Overview</td>
<td>2</td>
</tr>
<tr>
<td>Scala Crash Course</td>
<td>3</td>
</tr>
<tr>
<td>Chisel Deep Dive</td>
<td>4</td>
</tr>
<tr>
<td>Interactive Session</td>
<td>5</td>
</tr>
<tr>
<td>OpenSoC Walk Through</td>
<td>6</td>
</tr>
<tr>
<td>Using OpenSoC</td>
<td>7</td>
</tr>
<tr>
<td>Interactive Session</td>
<td>8</td>
</tr>
<tr>
<td>Future Work</td>
<td>9</td>
</tr>
<tr>
<td>Conclusion</td>
<td>10</td>
</tr>
</tbody>
</table>
OpenSoC Top Level Modules
OpenSoC – Functional Hierarchy
OpenSoC – Top Level Modules

- Stiches routers together
- Assigns routers individual ID
- Assigns Routing Function to routers
- Connections Injection and Ejection Queues for network endpoints
OpenSoC – Functional Hierarchy
OpenSoC Top Level Modules

Router

- Created and connected by Topology module
- Instantiates and connects:
  - Routing Function
  - Allocators
  - Switch
- Created as a 3 stage pipeline
  - Includes state storage for each sub-module
- Connects to Injection / Ejection Queues
OpenSoC – Top Level Modules

Allocator

- Collection of Arbiters
  - Currently all arbiters are round-robin, locking
  - Arbitration policy configurable

- Interface to credit logic

- Wormhole router has single allocator

- VC router has two allocators
  - Same module, controls both switch allocation and VC allocation
OpenSoC - Configuring

Parameters

- OpenSoC configured at run time through *Parameters* class
  
  - Declared at top level, sub modules can add / change parameters tree
  - *Hard* parameters may not be changed by sub-modules
  - *Soft* parameters may be changed by sub-modules

- Not limited to just integer values
  
  - Leverage Scala to pass functions to parameterize module creation
    
    - Example: Routing Function constructor passed as parameter to router
OpenSoC - Configuring

Parameters

- All OpenSoC Modules take a Parameters class as a constructor argument

- Setting parameters:
  - `parms.child("MySwitch", Map( ("numInPorts"-&gt;Soft(8)), ("numOutPorts"-&gt;Soft(3)) ))`

- Getting a parameter:
  - `val numInPorts = parms.get[Int]("numInPorts")`
OpenSoC Data Formats
Flits – All data widths inferred or described at runtime

- **Head Flit**
  - Packet ID
  - Is Tail
  - Vc Port
  - Packet Type
  - Destination
    - Chisel Vec

- **Body Flit**
  - Packet ID
  - Is Tail
  - Vc Port
  - Flit ID
  - Payload
# OpenSoC Data Formats

## Packets

<table>
<thead>
<tr>
<th>Data Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Address</td>
</tr>
<tr>
<td>Dest Address</td>
</tr>
<tr>
<td>Total Length</td>
</tr>
<tr>
<td>Packet Length</td>
</tr>
<tr>
<td>Additional Flags</td>
</tr>
<tr>
<td>PacketID</td>
</tr>
<tr>
<td>Command</td>
</tr>
<tr>
<td>Command Options</td>
</tr>
<tr>
<td>Reserved for Debug</td>
</tr>
<tr>
<td>Payload Phase 0</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Payload Phase N</td>
</tr>
</tbody>
</table>
OpenSoC Demo
Build a Network
For a concentration of 1

- Open VM
- Navigate to
  - ~/opensoc-demo/opensoc/
- Two Tests to Run
  - ./OpenSoC_CMesh_Random_C1.sh
    - Packets go to a random destination
  - ./OpenSoC_CMesh_Neighbor_C1.sh
    - Packets go to neighboring router
Build a Network
For a concentration of 2

- Open VM
- Navigate to
  - ~/opensoc-demo/opensoc/
- Change concentration of network to 2
  - Edit src/main/scala/main.scala to update val C
- Test to Run
  - ./OpenSoC_CMesh_Random_C2.sh
    - Packets go to a random destination
Future additions
Towards a full set of features

- A collection of topologies and routing functions
- An easy way to adjust router pipeline stages
- Validation against other RTL or simulators
- Standardized interfaces at the endpoints (e.g., AXI)
- More powerful synthetic traffic and trace replay support
- Power modeling in the C++ model
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motivation</td>
<td>1</td>
</tr>
<tr>
<td>Chisel Overview</td>
<td>2</td>
</tr>
<tr>
<td>Scala Crash Course</td>
<td>3</td>
</tr>
<tr>
<td>Chisel Deep Dive</td>
<td>4</td>
</tr>
<tr>
<td>Interactive Session</td>
<td>5</td>
</tr>
<tr>
<td>OpenSoC Walk Through</td>
<td>6</td>
</tr>
<tr>
<td>Using OpenSoC</td>
<td>7</td>
</tr>
<tr>
<td>Interactive Session</td>
<td>8</td>
</tr>
<tr>
<td>Future Work</td>
<td>9</td>
</tr>
<tr>
<td>Conclusion</td>
<td>10</td>
</tr>
</tbody>
</table>
Conclusion

- This is an open-source community-driven infrastructure
  - We are counting on your contributions
Acknowledgements

› UCB Chisel
› US Dept of Energy
› Ke Wen
› Columbia LRL
› John Bachan
› Dan Burke
› BWRC
More Information

http://opensocfabric.org