A Little Bit of Scala and Chisel Background

Martin Schoeberl

Technical University of Denmark

October 12, 2019

Chisel and Scala

- Chisel is a library written in Scala
  - Import the library with `import chisel3._`
- Chisel code is Scala code
- When it is run is *generates* hardware
  - Verilog for synthesize
  - Scala netlist for simulation (testing)
- Chisel is an embedded domain specific language
- Two languages in one can be a little bit confusing

Scala

- Is object oriented
- Is functional
- Strongly typed with very good type inference
- Runs on the Java virtual machine
- Can call Java libraries
- Consider it as Java++
  - Can almost be written like Java
  - With a more lightweight syntax
  - Scala for Java Refugees is a nice tutorial
  - `http://www.codecommit.com/blog/scala/roundup-scala-for-java-refugees`

Scala Hello World

```
object HelloWorld extends App {
  println("Hello, World!")
}
```

- Compile with `scalac` and run with `scala`
- You can even use Scala as scripting language
- Show both

Scala Values and Variables

```
// A value is a constant
val i = 0
// No new assignment; this will not compile
i = 3

// A variable can change the value
var v = "Hello"
v = "Hello World"

// Type usually inferred, but can be declared
var s: String = "abc"
```

Simple Loops

```
// Loops from 0 to 9
// Automatically creates loop value i
for (i <- 0 until 10) {
  println(i)
}
```

Conditions

```
for (i <- 0 until 10) {
  if (i%2 == 0) {
    println(i + " is even")
  } else {
    println(i + " is odd")
  }
}
```

Scala Arrays and Lists

```
// An integer array with 10 elements
val numbers = new Array[Integer](10)
for (i <- 0 until numbers.length) {
  numbers(i) = i*10
}
println(numbers(9))


// List of integers
val list = List(1, 2, 3)
println(list)
// Different form of list construction
val listenum = 'a' :: 'b' :: 'c' :: Nil
println(listenum)
```

Scala Classes

```
// A simple class
class Example {
  // A field, initialized in the constructor
  var n = 0

  // A setter method
  def set(v: Integer) = {
    n = v
  }

  // Another method
  def print() = {
    println(n)
  }
}
```

Scala (Singleton) Object

```
object Example {}
```

- For *static* fields and methods
  - Scala has no static fields or methods like Java
- Needed for `main`
- Useful for helper functions

Singleton Object for the `main`

```
// A singleton object
object Example {

  // The start of a Scala program
  def main(args: Array[String]): Unit = {

    val e = new Example()
    e.print()
    e.set(42)
    e.print()
  }
}
```

- Compile and run it with sbt (or within Eclipse/IntelliJ):

```
sbt "runMain Example"
```

Scala Build Tool (sbt)

- Downloads Scala compiler if needed
- Downloads dependent libraries (e.g., Chisel)
- Compiles Scala programs
- Executes Scala programs
- Does a lot of magic, maybe too much
- Compile and run with:

```
sbt "runMain simple.Example"
```

- Or even just:

```
sbt run
```

Build Configuration

- Defines needed Scala version
- Library dependencies
- File name: `build.sbt`

```
scalaVersion := "2.11.7"

resolvers ++= Seq(
  Resolver.sonatypeRepo("snapshots"),
  Resolver.sonatypeRepo("releases")
)

libraryDependencies += "edu.berkeley.cs" %%
   "chisel3" % "3.1.2"
libraryDependencies += "edu.berkeley.cs" %%
   "chisel-iotesters" % "1.2.2"
```

File Organization in Scala/Chisel

- A Scala file can contain several classes (and objects)
- For large classes use one file per class with the class name
- Scala has packages, like Java
- Use folders with the package names for file organization
- `sbt` looks into current folder and `src/main/scala/`
- Tests shall be in `src/test/scala/`

Chisel in Scala

- Chisel components are Scala classes
- Chisel code is in the constructor
- Executed at object creation time
- Builds the network of hardware objects
- Testers are written in Scala to drive the tests

Chisel Main

- Create one top-level Module
- Invoke the Chisel driver from the Scala `main` (or App)
- Pass some parameters and the top module
- Following code generates Verilog code

```
object Hello extends App {
  chisel3.Driver.execute(Array[String](), () =>
      new Hello())
}
```

Chisel Main for Testing

- Tests can be written in Scala/Chisel
- Tester and device under test (DUT) are two processes
- Invoke `execute` with some parameters, the DUT, and a tester

```
object CounterTester extends App {

  iotesters.Driver.execute(Array[String](), () =>
      new Counter(2)) {
    c => new CounterTester(c)
  }
}
```

A Chisel Tester

- Extends class `PeekPokeTester`
- Has the DUT as parameter
- Testing code can use all features of Scala

```
class CounterTester(dut: Counter) extends
    PeekPokeTester(dut) {

  // Here comes the Chisel/Scala code
  // for the testing
}
```

Testing

- Set input values with `poke`
- Advance the simulation with `step`
- Read the output values with `peek`
- Compare the values with `expect`

Testing Example

```
// Set input values
poke(dut.io.a, 3)
poke(dut.io.b, 4)
// Execute one iteration
step(1)
// Print the result
val res = peek(dut.io.result)
println(res)

// Or compare against expected value
expect(dut.io.result, 7)
```

A Tiny ALU: IO Connection

```
class Alu extends Module {
  val io = IO(new Bundle {
    val fn = Input(UInt(2.W))
    val a = Input(UInt(4.W))
    val b = Input(UInt(4.W))
    val result = Output(UInt(4.W))
  })

  // Use shorter variable names
  val fn = io.fn
  val a = io.a
  val b = io.b
```

A Tiny ALU: The Function

```
  val result = Wire(UInt(4.W))
  // some default value is needed
  result := 0.U

  // The ALU selection
  switch(fn) {
    is(0.U) { result := a + b }
    is(1.U) { result := a - b }
    is(2.U) { result := a | b }
    is(3.U) { result := a & b }
  }

  // Output on the LEDs
  io.result := result
}
```

Testing the ALU

- Compute the expected result in Scala

```
// This is exhaustive testing,
// which usually is impossible
for (a <- 0 to 15) {
  for (b <- 0 to 15) {
    for (op <- 0 to 3) {
      val result =
        op match {
          case 0 => a + b
          case 1 => a - b
          case 2 => a | b
          case 3 => a & b
        }
      val resMask = result & 0x0f
```

Testing the ALU

- Compare the Scala computed result with the hardware result

```
      poke(dut.io.fn, op)
      poke(dut.io.a, a)
      poke(dut.io.b, b)
      step(1)
      expect(dut.io.result, resMask)
    }
  }
}
```

Generating Wave Forms

- Additional parameters: `"--generate-vcd-output", "on"`
- IO signals and registers are dumped
- Option `--debug` puts all wires into the dump
- Generates a .vcd file
- Viewing with gtkwave or ModelSim
- See the example with `make fifo`
  - Show it
- BubbleFifo contains also longer testing code

Functions

- Circuits can be encapsulated in functions
- Each *function call* generates hardware
- Simple functions can be a single line

```
def adder(v1: UInt, v2: UInt) = v1 + v2

val add1 = adder(a, b)
val add2 = adder(c, d)
```

More Function Examples

- Functions can also contain registers

```
def addSub(add: Bool, a: UInt, b: UInt) =
  Mux(add, a + b, a - b)

val res = addSub(cond, a, b)

def rising(d: Bool) = d && !RegNext(d)

val edge = rising(cond)
```

The Counter as a Function

- Longer functions in curly brackets
- Last value is the return value

```
def counter(n: UInt) = {

  val cntReg = RegInit(0.U(8.W))

  cntReg := cntReg + 1.U
  when(cntReg === n) {
    cntReg := 0.U
  }
  cntReg
}

val counter100 = counter(100.U)
```

Functions

- Example from Patmos execute stage

```
def alu(func: Bits, op1: UInt, op2: UInt): Bits = {
  val result = UInt(width = DATA_WIDTH)
  // some more lines...
  switch(func) {
    is(FUNC_ADD) { result := sum }
    is(FUNC_SUB) { result := op1 - op2 }
    is(FUNC_XOR) { result := (op1 ^ op2).toUInt }
    // some more lines
  }
  result
}
```

Lab Session

- Explore the ALU example
  - It contains a tester
  - It can also be tested with the switches and LEDs on the FPGA
- Explore the bubble FIFO example for wave form viewing
- Do an example project from scratch
  - But reuse Quartus files (from the ALU or knight example)
  - With a Chisel main
  - Some simple function with LEDs and switches
    - E.g., moving light with direction change
    - Remember Knight Rider?
  - Running it in the FPGA