Verification of Digital Designs: Introduction

Martin Schoeberl

Technical University of Denmark
Embedded Systems Engineering

August 26, 2021

Overview

- Motivation
- Course organization
- Languages for digital hardware design
- Debugging, Testing, and Verification
- A little bit of Scala
- An Exercise

Motivation

- We had a meeting with DK industry in June
- Missing design verification
- For one developer there are 2–3 verification engineers
- Testing is considered boring, but it does not have to be
- Best is to be a developer and a verification engineer
- Change roles, we will do in this course

Course Organization

- This is a special course
- Not just me giving talks and preparing exercises
  - But I will bring in some
- This is a lot about self study
- You will bring up material
- We will use GitHub for material, exercises, project
  - Slides are there as well
  - https://github.com/chisel-uvm/class2020
  - Let's signup right now ;-)
  - Shall we use Slack?

Technicalities

- We will use Chisel/Scala for testing
- VHDL, SystemVerilog, UVM are optional
- You need to setup your laptop
  - see: https://github.com/schoeberl/chisel-lab/blob/master/Setup.md
  - We will not use an FPGA

Reading Material

- No *good* book on DV
- We need to find
  - Web sites
  - Blogs
  - Articles (popular, e.g., EE Times)
  - Paper (conferences)
  - Look what software people do
  - ...
- Your homework: search literature till next week
- Present what you found

This is an Open-Access/Open-Source Course

- Almost all material is public visible
- Slides are open access
- Lab material is open access
- Hosted on GitHub
  - **You** can contribute with a pull request
  - Becoming an author of this course :-)
- The Chisel book is freely available

Chisel Overview

- A hardware *construction* language
  - Constructing Hardware In a Scala Embedded Language
  - If it compiles, it is synthesisable hardware
  - Say goodby to your unintended latches
- Chisel is not a high-level synthesis language
- Single source for two targets
  - Cycle accurate simulation (testing)
  - Verilog for synthesis
- Embedded in Scala
  - Full power of Scala available
  - But to start with, no Scala knowledge needed
- Developed at UC Berkeley

Chisel vs. Scala

- A Chisel hardware description is a Scala program
- Chisel is a Scala library
- When the program is executed it generates hardware
- Chisel is a so-called *embedded domain-specific language*

Free Tools for Chisel and FPGA Design

- Java OpenJDK 8 already installed for Java course
- sbt, the Scala (and Java) build tool
- IntelliJ (the free Community version)
- GTKWave
- Vivado WebPACK already installed from DE1
- Nice to have:
  - make, git

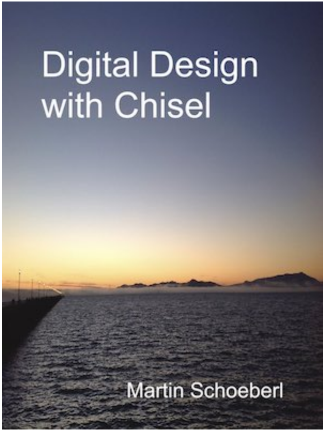
Tool Setup for Different OSs

- Windows
  - Use the installers from the websites
- macOS
  - `brew install sbt`
  - For the rest, use the installer from the websites
- Linux/Ubuntu
  - `sudo apt install openjdk-8-jdk git make gtkwave`
  - Install sbt, see https://github.com/schoeberl/chisel-lab/blob/master/Setup.md
  - IntelliJ as from the website

An IDE for Chisel

- IntelliJ
- Scala plugin
- For IntelliJ: File - New - Project from Existing Sources..., open build.sbt
- Show it (down to the Basys3)

A Chisel Book



- Available in open access (as PDF)
  - Optimized for reading on a tablet (size, hyper links)
- Amazon can do the printout

Further Information

- https://www.chisel-lang.org/
- https://github.com/freechipsproject/chisel-cheatsheet/releases/latest/download/chisel_cheatsheet.pdf
- https://github.com/ucb-bar/chisel-tutorial
- https://github.com/ucb-bar/generator-bootcamp
- http://groups.google.com/group/chisel-users
- https://github.com/schoeberl/chisel-book

Testing and Debugging

- Nobody writes perfect code ;-)
- We need a method to improve the code
- In Java we can simply print the result:
  - `println("42");`
- What can we do in hardware?
  - Describe the whole circuit and hope it works?
  - We can switch an LED on or off
- We need some tools for debugging
- Writing testers in Chisel

Testing with Chisel

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

```
import chisel3._
import chisel3.iotesters._
```

Using peek, poke, and expect

```
// 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 Chisel Tester

- Extends class `PeekPokeTester`
- Has the device-under test (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
}
```

Example DUT

- A device-under test (DUT)

```
class DeviceUnderTest extends Module {
  val io = IO(new Bundle {
    val a = Input(UInt(2.W))
    val b = Input(UInt(2.W))
    val out = Output(UInt(2.W))
  })

  io.out := io.a & io.b
}
```

A Simple Tester

- Just using println for manual inspection

```
class TesterSimple(dut: DeviceUnderTest)
   extends PeekPokeTester(dut) {

  poke(dut.io.a, 0.U)
  poke(dut.io.b, 1.U)
  step(1)
  println("Result is: " +
     peek(dut.io.out).toString)
  poke(dut.io.a, 3.U)
  poke(dut.io.b, 2.U)
  step(1)
  println("Result is: " +
     peek(dut.io.out).toString)
}
```

The Main Program for the Test

- Extend an App and invoke the `iotesters` driver
- With the DUT and the tester

```
object TesterSimple extends App {
  chisel3.iotesters.Driver(() => new
      DeviceUnderTest()) { c =>
    new TesterSimple(c)
  }
}
```

A Real Tester

- Poke values and `expect` some output

```
class Tester(dut: DeviceUnderTest) extends
    PeekPokeTester(dut) {

  poke(dut.io.a, 3.U)
  poke(dut.io.b, 1.U)
  step(1)
  expect(dut.io.out, 1)
  poke(dut.io.a, 2.U)
  poke(dut.io.b, 0.U)
  step(1)
  expect(dut.io.out, 0)
}
```

ScalaTest

- Testing framework for Scala
- `sbt` understands ScalaTest
- Run all tests: `sbt test`
- When all `expects` are ok, the test passes
- A little bit funny syntax
- Add library to `build.sbt`

```
libraryDependencies += "org.scalatest" %%
   "scalatest" % "3.0.5" % "test"
```

- Import ScalaTest library

```
import org.scalatest._
```

ScalaTest Version of our Tester

```
class SimpleSpec extends FlatSpec with Matchers {

  "Tester" should "pass" in {
    chisel3.iotesters.Driver(() => new
        DeviceUnderTest()) { c =>
      new Tester(c)
    } should be (true)
  }
}
```

Generating Waveforms

- Waveforms are timing diagrams
- Good to see many parallel signals and registers
- 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

Call the Tester

- Using here ScalaTest
- Note `Driver.execute`
- Note `Array("--generate-vcd-output", "on")`

```
class Count6WaveSpec extends
  FlatSpec with Matchers {

  "CountWave6 " should "pass" in {
    chisel3.iotesters.Driver.
    execute(Array("--generate-vcd-output",
        "on"),() => new Count6)
    { c => new Count6Wave(c) }
    should be (true)
  }
}
```

Test Driven Development (TDD)

- Software development process
  - Can we learn from SW development for HW design?
- Writing the test first, then the implementation
- Started with extreme programming
  - Frequent releases
  - Accept change as part of the development
- Not used in its pour form
  - Writing all those tests is simply considerer too much work

Testing versus Debugging

- Debugging is during code development
- Waveform and println are easy tools for debugging
- Debugging does not help for regression tests
- Write small test cases for regression tests
- Keeps your code base *intact* when doing changes
- Better confidence in changes not introducing new bugs

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
  - Compiled to the JVM
  - Good Java interoperability
  - Many libraries available
- `https://docs.scala-lang.org/tour/tour-of-scala.html`

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 has a REPL, show it

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`

```scala
// 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

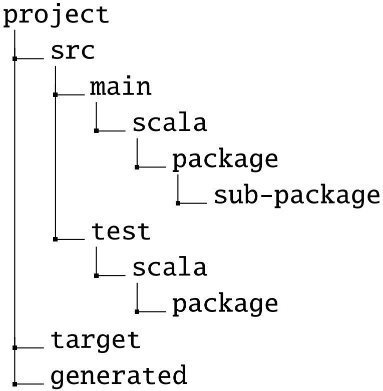
- Defines needed Scala version
- Library dependencies
- File name: `build.sbt`

```
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/`

File Organization in Scala/Chisel



```
project
    src
        main
            scala
                package
                    sub-package
        test
            scala
                package
    target
    generated
```

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
- You can write a reference simulation in Scala and compare with Chisel

Summary

- This is a special course
- We will work together to learn about testing and verification
- You will present reading material next week
- We meet again next Tuesday 13:00 in 322/123

Lab Time: Hello World Testing

- Write test/verification code for a 5:1 multiplexer
- Project and DUT in GitHub lab1
- `https://github.com/chisel-uvm/class2020`