Chisel Installation

Jonathan Bachrach
EECS Department, UC Berkeley
{jrb}@eecs.berkeley.edu

August 29, 2014

1 Introduction

This document is an installation guide for *Chisel* (Constructing Hardware In a Scala Embedded Language). Chisel is a hardware construction language embedded in the high-level programming language Scala.

1.1 Github

- Get an account on www.github.com
- Register your public key on github.com

1.2 Development Tool Installation

1.2.1 MacOSX

1. Install XCODE including console tools.
2. Install MacPorts from http://www.macports.org

From there install the following MacPorts packages:

1. git
2. openjdk6

using

```
sudo port install
```

1.2.2 Linux

To install Chisel on Linux, install the following packages:

1. git
2. g++
3. openjdk-7-jre
4. openjdk-7-jdk

using

```
sudo apt-get install
```

2 Setting Up Tutorial

cd above directory = $DIR you've chosen to place Chisel tutorial and type:

```
cd $DIR
git clone
    https://github.com/ucb-bar/chisel-tutorial.git
```

Your copy of the Chisel Tutorial repository will then be in $DIR/chisel-tutorial. Define this as a variable in your bash environment:

The following is the Chisel tutorial directory structure:

```
chisel-tutorial/
  src/
    problems/
      Accumulator.scala ...
    solutions/
      Accumulator.scala ...
  emulator/
    problems/
      Makefile
    solutions/
      Makefile
  verilog/
    problems/
      Makefile
    solutions/
      Makefile
  sbt/
    project/
      build.scala
```

The tutorial is split into problems and solutions, where the problems have some piece of the design to be filled in by the user and where the solutions are meant to be complete designs that should pass the given tests. In order to run either, you change directory into the appropriate subdirectory and type make of the particular lesson name:

```
cd $CHISEL/tutorial/emulator/solutions
make GCD
```

or you can run all tests using

```
cd $CHISEL/tutorial/emulator/solutions
make all
```

and the output should show that all tests have passed.

In order to produce Verilog, just do the following:

```
cd $CHISEL/tutorial/verilog/solutions
make all
```

3 Creating Your Own Projects

SBT has a particular directory structure that we adhere to and somewhat improve. Assuming that we have a project named *gpu*, then the following would be the directory structure template:

```
gpu/
  sbt/
    project/
      build.scala # edit this as shown below
  src/
    gpu.scala     # your source files go here
  emulator/       # your C++ target can go
      here
  verilog/        # your Verilog target can
      go here
```

and the following is the build.scala template:

```
import sbt._
import Keys._

object BuildSettings {
  val buildOrganization = "edu.berkeley.cs"
  val buildVersion = "1.1"
  val buildScalaVersion = "2.9.2"

  def apply(projectdir: String) = {
    Defaults.defaultSettings ++ Seq (
      organization := buildOrganization,
      version      := buildVersion,
      scalaVersion := buildScalaVersion,
      scalaSource in Compile :=
        Path.absolute(file(projectdir +
            "/src"))
      libraryDependencies +=
        "edu.berkeley.cs" %% "chisel" % "1.0"
    )
  }
}

object ChiselBuild extends Build {
  import BuildSettings._
  lazy val gpu =
    Project("gpu", file("gpu"),
            settings = BuildSettings(".."))
      dependsOn(chisel)
}
```

If you want to update your version of Chisel, all you have to do is change the version number for Chisel. For instance,

```
libraryDependencies += "edu.berkeley.cs" %%
    "chisel" % "1.0"
```

is using release 1.0, and

```
libraryDependencies += "edu.berkeley.cs" %%
    "chisel" % "1.0.1"
```

is using release 1.0.1