Verification of Digital Designs: Week 2

Martin Schoeberl

Technical University of Denmark
Embedded Systems Engineering

September 7, 2021

Overview

- More on tooling
- Some more Scala (+ repetition)
- An example test of an ALU
- Regression tests and continuous integration
- Presentation of reading material
- Lab: test an accumulator (from the Leros processor)

Tools: git and GitHub

- git is a distributed version-control system
- You *may* also use GitHub as code host
- `git clone path` local copy from a server
- `git pull` get updates
- `git commit -a -m "message"` add your changes locallye
- `git push` push to the server
- `git add filename` add a file to the repo
- Always start with git pull
- Concurrent edits are merged on a line-by-line bases
- When sharing text, have manual line breaks to minimize chances of conflict

Tools: make

- `make` is for automation
- A list a dependencies and commands to run
- Can become complex
- Keep it simple just to remember commands
- Look into chisel-example

Using make

- Run it with:
  - `make`
- Run a target:
  - `make target`
- Targets are described in a `Makefile`
  - `target: dependency`
  - `[tab] command`
- Let us explore it now
- Install `make` and write a simple `Makefile`

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)
}
```

Scala Classes

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

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

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

Functions

```
class Example {

  def compute(a: Int, b: Int): Int = {
    a + b
  }
}
```

- Last expression is the return value

Functions

```
class Example {

  def complexCompute(a: Int, b: Int, c: Int):
      Int = {

    def add(x: Int, y: Int) {
      x + y
    }

    add(a, b) + c
  }
}
```

- We can define local functions
- To better organize our code

Functions

```
class Example {

  def complexCompute(a: Int, b: Int, c: Int):
      Int = {

    def add() {
      a + b
    }

    add() + c
  }
}
```

- Local functions have access to outer variables

Regression Tests

- Tests are collected over time
- When a bug is found, a test is written to reproduce this bug
- Collection of tests increases
- Runs every night to test for *regression*
  - Did a code change introduce a bug in the current code base?

Continuous Integration (CI)

- Next logical step from regression tests
- Run all tests whenever code is changed
- Automate this with a repository, e.g., on GitHub
- Run CI on Travis (with GitHub integration)
- Show about this on the Chisel book
  - Show `sbt test`
  - Mails from travis
  - Live demo on travis
- https://travis-ci.com/schoeberl/chisel-book

Lab Time I: Write a ScalaTest

- Setup a Scala project with `build.sbt`
- ScalaTest: write a test suit to test `Int` that 2 + 3 = 5, and two more integer tests
- Maybe add a `Makefile`

Lab Time II: Test the accumulator

- ALU + register
- Exhaustive testing is not an option
- Corner cases plus random

Lab Time III: CI with Travis

- Create a repo on GitHub
- Add the ALU plus your tester
- Connect it with Travis
- you need a `.travis.yml`
- Look into the Chisel book source for an example