Chisel: Testers

Muhammad Tahir

Lecture 4

Contents

Testing in Chisel

- Chisel offers simulation based verification tools
- Most common tool in Chisel is **iotesters** with three harnesses available currently
  - PeekPokeTester (most common)
  - SteppedHWIOTester
  - OrderedDecoupledHWIOTester
- Another (under development) tool is **tester2**

Test Project

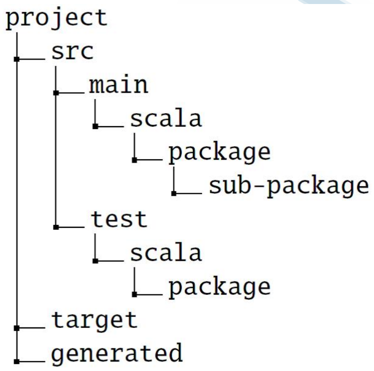


Figure: Directory structure for running the tests.

PeekPokeTester

- Test harness provides the following four interactions with the device-under-test (DUT)
  - `poke`: to set/drive the DUT's inputs
  - `peek`: to observe the DUT's outputs (may also be used to observe inputs)
  - `expect`: to test the DUT's outputs for specific response
  - `step`(n): to advance the DUT's clock by n cycle
- The tester is constructed by sub-classing PeekPokeTester
- Similar to non-synthesizable Verilog testbench

Example Program

```
package LM_Chisel
import chisel3._

class MuxTreeIO extends Bundle {
      val in_1    = Input(UInt(32.W))
      val in_2    = Input(UInt(32.W))
      val in_3    = Input(UInt(32.W))
      val sel_1   = Input(Bool())
      val sel_2   = Input(Bool())
      val out     = Output(UInt())
}

// 3 to 1 MuxTree implementation
class MuxTree extends Module {
      val io = IO(new MuxTreeIO)

      // update the output
      io.out := Mux(io.sel_2, io.in_3, Mux(io.sel_1, io.in_2, io.in_1))
}
```

Writing a Test

```
package LM_Chisel
import chisel3._
import chisel3.iotesters.{
     Driver, PeekPokeTester
}

class TestMuxT(c: MuxTree) extends PeekPokeTester(c) {
     val in1  = 0x11111111
     val in2  = 0x22222222
     val in3  = 0x33333333

     poke(c.io.in_1, in1.U)
     poke(c.io.in_2, in2.U)
     poke(c.io.in_3, in3.U)

     poke(c.io.sel_1, false.B)
     poke(c.io.sel_2, false.B)
     expect(c.io.out, in1.U)
     step(1)

     poke(c.io.sel_1, true.B)
     poke(c.io.sel_2, false.B)
     expect(c.io.out, in2.U)
     step(1)

     poke(c.io.sel_1, true.B)
     poke(c.io.sel_2, true.B)
     expect(c.io.out, in3.U)
     step(3)
```

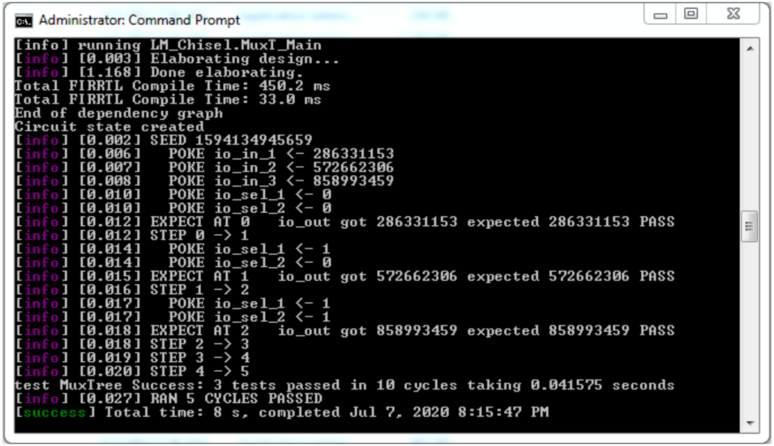
Wrapping the Tester

```
// object for tester class
object MuxT_Main extends App {
    iotesters.Driver.execute(Array("--is-verbose", "--
     generate-vcd-output",
    "on"), () => new MuxTree) {
        c => new TestMuxT(c)
    }
}
```

Generating Verilog and Running Test

- Scala build tool (sbt) is used to generate Verilog and run the tests
- Verilog code is generated by the following command
  $>$ sbt run
- The tester is run by the following command
  $>$ sbt test:run

Viewing Output on Command Line


Administrator: Command Prompt

```
[info] running LM_Chisel.MuxT_Main
[info] [0.003] Elaborating design...
[info] [1.168] Done elaborating.
Total FIRRTL Compile Time: 450.2 ms
Total FIRRTL Compile Time: 33.0 ms
End of dependency graph
Circuit state created
[info] [0.002] SEED 1594134945659
[info] [0.006]   POKE io_in_1 <- 286331153
[info] [0.007]   POKE io_in_2 <- 572662306
[info] [0.008]   POKE io_in_3 <- 858993459
[info] [0.010]   POKE io_sel_1 <- 0
[info] [0.010]   POKE io_sel_2 <- 0
[info] [0.012] EXPECT AT 0   io_out got 286331153 expected 286331153 PASS
[info] [0.012] STEP 0 -> 1
[info] [0.014]   POKE io_sel_1 <- 1
[info] [0.014]   POKE io_sel_2 <- 0
[info] [0.015] EXPECT AT 1   io_out got 572662306 expected 572662306 PASS
[info] [0.016] STEP 1 -> 2
[info] [0.017]   POKE io_sel_1 <- 1
[info] [0.017]   POKE io_sel_2 <- 1
[info] [0.018] EXPECT AT 2   io_out got 858993459 expected 858993459 PASS
[info] [0.018] STEP 2 -> 3
[info] [0.019] STEP 3 -> 4
[info] [0.020] STEP 4 -> 5
test MuxTree Success: 3 tests passed in 10 cycles taking 0.041575 seconds
[info] [0.027] RAN 5 CYCLES PASSED
[success] Total time: 8 s, completed Jul 7, 2020 8:15:47 PM
```

Figure: Command line output in verbose mode.

Viewing Output using Waveform Viewer

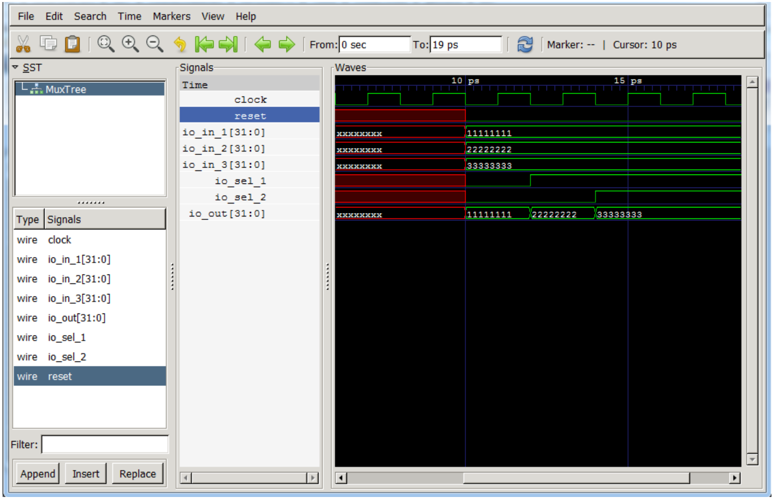


Figure: Viewing input/output signals using GTKWave.

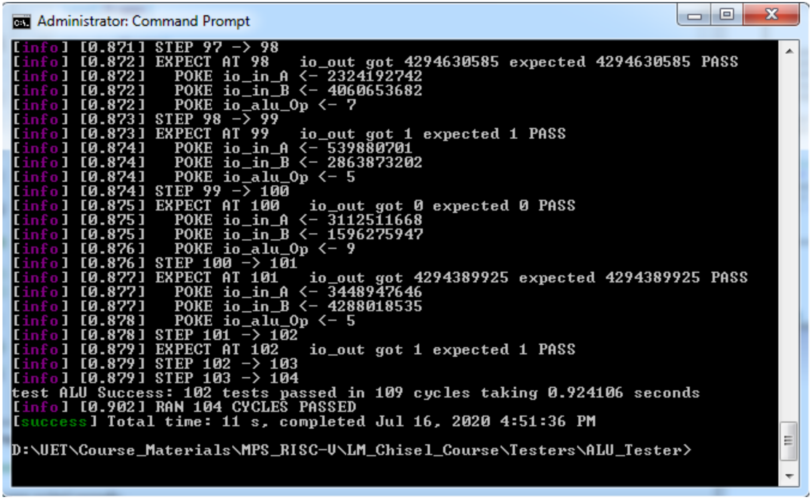
Testing ALU

```
class TestALU(c: ALU) extends PeekPokeTester(c) {
     // ALU operations
     val array_op = Array(ALU_ADD , ALU_SUB, ALU_AND, ALU_OR, ALU_XOR,
     ALU_SLT, ALU_SLL, ALU_SLTU, ALU_SRL, ALU_SRA, ALU_COPY_A, ALU_COPY_B,
     ALU_XXX)

     for (i <- 0 until 100) {

          val src_a = Random.nextLong()& 0xFFFFFFFFL
          val src_b = Random.nextLong()& 0xFFFFFFFFL
          val opr   = Random.nextInt(12)
          val aluop = array_op(opr)

          // ALU functional implementation using Scala match
          val result = aluop match {
               case ALU_ADD  => src_a + src_b
               case ALU_SUB  => src_a - src_b
               case ALU_AND  => src_a & src_b
               case ALU_OR   => src_a | src_b
               case ALU_XOR  => src_a ^ src_b
               case ALU_SLT  => (src_a.toInt < src_b.toInt).toInt
               case ALU_SLL  => src_a  << (src_b & 0x1F)
               case ALU_SLTU => (src_a < src_b).toInt
               case ALU_SRL  => src_a >> (src_b & 0x1F)
               case ALU_SRA  => src_a.toInt >> (src_b & 0x1F)
               case ALU_COPY_A => src_a
               case ALU_COPY_B => src_b
               case _        => 0
          }
```

Testing ALU

```
// process the ALU result for -ve values
  val result1: BigInt = if (result < 0)
                             (BigInt(0xFFFFFFFF)+result+1) & 0xFFFFFFFFL
                        else result & 0xFFFFFFFFL

    poke(c.io.in_A, src_a.U)
    poke(c.io.in_B, src_b.U)
    poke(c.io.alu_Op, aluop)
    step(1)
    expect(c.io.out, result1.asUInt)
 }
  step(1)
  step(1)
}

// object for tester class
object ALU_Main extends App {
 iotesters.Driver.execute(Array("--is-verbose", "--generate-vcd-output",
   "on", "--backend-name", "firrtl"), () => new ALU) {
     c => new TestALU(c)
  }
}
```

ALU Test Results



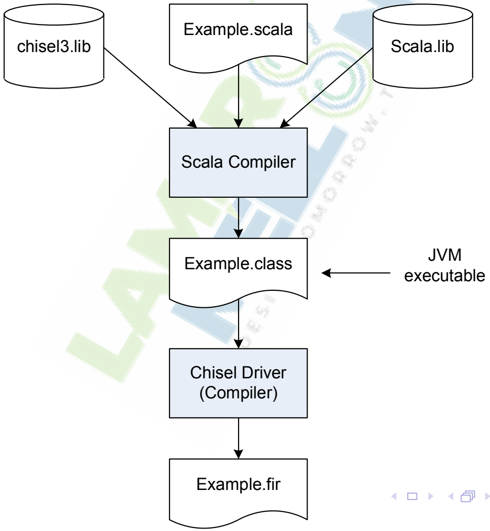
```
[info] [0.871] STEP 97 -> 98
[info] [0.872] EXPECT AT 98   io_out got 4294630585 expected 4294630585 PASS
[info] [0.872]   POKE io_in_A <- 2324192742
[info] [0.872]   POKE io_in_B <- 4060653682
[info] [0.872]   POKE io_alu_Op <- 7
[info] [0.873] STEP 98 -> 99
[info] [0.873] EXPECT AT 99   io_out got 1 expected 1 PASS
[info] [0.874]   POKE io_in_A <- 539880701
[info] [0.874]   POKE io_in_B <- 2863873202
[info] [0.874]   POKE io_alu_Op <- 5
[info] [0.874] STEP 99 -> 100
[info] [0.875] EXPECT AT 100   io_out got 0 expected 0 PASS
[info] [0.875]   POKE io_in_A <- 3112511668
[info] [0.875]   POKE io_in_B <- 1596275947
[info] [0.876]   POKE io_alu_Op <- 9
[info] [0.876] STEP 100 -> 101
[info] [0.877] EXPECT AT 101   io_out got 4294389925 expected 4294389925 PASS
[info] [0.877]   POKE io_in_A <- 3448947646
[info] [0.877]   POKE io_in_B <- 4288018535
[info] [0.878]   POKE io_alu_Op <- 5
[info] [0.878] STEP 101 -> 102
[info] [0.879] EXPECT AT 102   io_out got 1 expected 1 PASS
[info] [0.879] STEP 102 -> 103
[info] [0.879] STEP 103 -> 104
test ALU Success: 102 tests passed in 109 cycles taking 0.924106 seconds
[info] [0.902] RAN 104 CYCLES PASSED
[success] Total time: 11 s, completed Jul 16, 2020 4:51:36 PM

D:\UET\Course_Materials\MPS_RISC-V\LM_Chisel_Course\Testers\ALU_Tester>
```

Figure: ALU test results.

Tools Invocation

- The digital circuit is described in Chisel (**Example.scala**)
- Scala compiler compiles **Example.scala**, together with the Chisel and Scala libraries, and generates **Example.class** (a Java class)
- The **Example.class** is compiled by Chisel Driver to generate **Example.fir**

Tools Invocation Cont'd



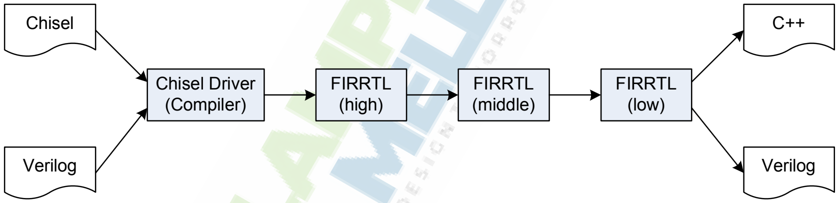
What is FIRRTL

- FIRRTL is a Flexible Intermediate Representation (IR) for RTL (the circuit)
- It is the file format emitted by the Chisel compiler (.fir file)
- The *.fir file is the input to FIRRTL compiler (written in Scala)
- FIRRTL compiler passes the circuit through a series of circuit-level transformations
- A FIRRTL transform can be applied at one of three levels, Hi/Mid/Low

FIRRTL Cont'd

- Transforms can be standalone or can take annotations as input
- Annotations are used to pass information to FIRRTL compiler when applying transforms
- Chisel driver automatically serialize the annotations as json snippet (.json file)
- JSON (JavaScript Object Notation) is a syntax for storing and exchanging data

FIRRTL Cont'd



Tools Invocation in Testing

When running a test

- The tester is invoked
- Tester invokes chisel3 to generate the circuit
- The chisel3 invokes firrtl to compile the circuit into low firrtl
- The low firrtl invokes the firrtl-interpreter to execute the test on the DUT

Tester Options

```
                       Selected Tester Options
                       -----------------------

-tbn, --backend-name <firrtl|treadle|verilator|ivl|vcs>
                     backend to use with tester, default is treadle

-tiv, --is-verbose   set verbose flag on PeekPokeTesters, default is false

-twffn, --wave-form-file-name <value>
                     wave form file name

-tts, --test-seed <value>
                     provides a seed for random number generator

-tgvo, --generate-vcd-output <value>
                     set this flag to "on" or "off", otherwise it defaults to on
                     for verilator, off for scala backends

-td <target-directory>, --target-dir <target-directory>
                     defines a work directory for intermediate files, default is
                     current directory
```

Chisel: dontTouch

- Labels the signal, so that it is not removed by Chisel and Firrtl optimization

```
import chisel3._
import chisel3.stage.ChiselStage

class DontTouch extends Module {
  val io = IO(new Bundle {
    val a = Input(UInt(32.W))
    val b = Output(UInt(32.W))
  })

  io.b := io.a
  val reg1 = RegInit(18.U)
  val unused = io.a + reg1 // 'unused' will be eliminated if not
  dontTouch(unused)        // preserved with dontTouch
}

println((new ChiselStage).emitVerilog(new DontTouch))
```

dontTouch Cont'd

Output with `dontTouch`

```
module DontTouch(
  input         clock,
  input         reset,
  input  [31:0] io_a,
  output [31:0] io_b
);
  wire [31:0] unused; // @[main.scala 14:21]
  assign unused = io_a + 32'h12;
  assign io_b = io_a; // @[main.scala 12:8]
endmodule
```

Output without `dontTouch`

```
module DontTouch(
  input         clock,
  input         reset,
  input  [31:0] io_a,
  output [31:0] io_b
);
  assign io_b = io_a; //@[main.scala 12:8]
endmodule
```

Reading List I

- Read Chapters 1 to 3 of [Odersky et al., 2016] for an introduction to Scala
- Read Chapter 2 of [Schoeberl, 2019] for basic know how of Chisel and Chapter 1 for tools installation

References

Odersky, M., Spoon, L., and Venners, B. (2016).
Programming in Scala.
Artima Incorporation.

Schoeberl, M. (2019).
Digital Design with Chisel.
Kindle Direct Publishing.