Verifying Chisel Circuits

Kevin Laeufer <laeufer@berkeley.edu>

Chisel Introduction

What is Chisel?

- Hardware Construction Language Embedded in Scala
- Allows you to write a Scala program that *generates* the description of a synchronous digital circuit
- Similar to a perl script that generates Verilog, but much better error reporting, auto-complete and well defined semantics
- **not** HLS, every state element is explicitly created by the designer

What is Chisel? - Basics

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = Reg(Bool())
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

What is Chisel? - Basics

Chisel module

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = Reg(Bool())
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

What is Chisel? - Basics

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = Reg(Bool())
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel module

signal type

What is Chisel? - Basics

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = Reg(Bool())
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel module

signal type

signal direction

What is Chisel? - Basics

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = Reg(Bool())
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel module

signal type

signal direction

signal is a port

What is Chisel? - Basics

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = Reg(Bool())
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel module

signal type

signal direction

signal is a port

register with undefined reset value

What is Chisel? - Basics

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = Reg(Bool())
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel module

signal type

signal direction

signal is a port

register with undefined reset value

condition

What is Chisel? - Basics

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = Reg(Bool())
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel module

signal type

signal direction

signal is a port

register with undefined reset value

condition

assign next state

What is Chisel? - Basics

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = Reg(Bool())
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel module

signal type

signal direction

signal is a port

register with undefined reset value

condition

assign next state

assign output

What is Chisel? - Bundles

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = Reg(Bool())
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

What is Chisel? - Bundles

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = Reg(Bool())
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

```
class InverterIO extends Bundle {
 val in = Input(Bool())
 val out = Output(Bool())
 val hold = Input(Bool())
}
```

What is Chisel? - Bundles

```
class Inverter extends Module {
 val io = IO(new InverterIO)



 val delay = Reg(Bool())
 when(!io.hold) {
   delay := !io.in
 }
 io.out := delay
}
```

```
class InverterIO extends Bundle {
 val in = Input(Bool())
 val out = Output(Bool())
 val hold = Input(Bool())
}
```

What is Chisel? - Bundles

```
class Inverter extends Module {
 val io = IO(new InverterIO)



 val delay = Reg(Bool())
 when(!io.hold) {
   delay := !io.in
 }
 io.out := delay
}
```

```
class InverterIO extends Bundle {
 val in = Input(Bool())
 val out = Output(Bool())
 val hold = Input(Bool())
}
```

```
class InvWrap extends Module {
 val io = IO(new InverterIO)
 val inv = Module(new Inverter)
 io <> inv.io
}
```

What is Chisel? - Generators

```
class Inverter extends Module {
 val io = IO(new InverterIO)

 val delay = Reg(Bool())
 when(!io.hold) {
   delay := !io.in
 }
 io.out := delay
}
```

What is Chisel? - Generators

```
class Inverter(ignoreHold: Boolean) extends Module {
 val io = IO(new InverterIO)

 val delay = Reg(Bool())
 when(!io.hold) {
   delay := !io.in
 }
 io.out := delay
}
```

What is Chisel? - Generators

```
class Inverter(ignoreHold: Boolean) extends Module {
 val io = IO(new InverterIO)

 val delay = Reg(Bool())
 when(!io.hold) {
   delay := !io.in
 }
 io.out := delay
}
```

Scala type!

What is Chisel? - Generators

```
class Inverter(ignoreHold: Boolean) extends Module {
 val io = IO(new InverterIO)

 val delay = Reg(Bool())
  if(ignoreHold) {
   delay := !in
  } else {
   when(!hold) {
     delay := !in
   }
  }
 io.out := delay
}
```

Scala type!

What is Chisel? - Generators

```
class Inverter(ignoreHold: Boolean) extends Module {
 val io = IO(new InverterIO)

 val delay = Reg(Bool())
  if(ignoreHold) {
   delay := !in
  } else {
   when(!hold) {
     delay := !in
   }
  }
 io.out := delay
}
```

Scala type!

Scala **if/else** is evaluated at generator runtime!

What is Chisel? - Generators

```
class Inverter(ignoreHold: Boolean) extends Module {
 val io = IO(new InverterIO)

 val delay = Reg(Bool())
  if(ignoreHold) {
   delay := !in
  } else {
   when(!hold) {
     delay := !in
   }
  }
 io.out := delay
}
```

Scala type!

Scala **if/else** is evaluated at generator runtime!

Chisel **when** becomes part of the circuit!

What is Chisel? - Advantages

- write **Reg** to get a register, no need to model the behavior of a register
- reset and clock are automatically connected
- Chisel can generate Verilog that is compatible with virtually all simulation and synthesis tools
- build powerful automation directly in Scala, like the RocketChip SoC generator
- take advantage of modern software development tools: IDE, package manager, unit test frameworks, continuous integration

Chisel Test Benches

Chisel Test Benches

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel Test Benches

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester {




}
```

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel Test Benches

standard
scalatest
framework

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester {



}
```

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel Test Benches

standard scalatest framework

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester {

}
```

our chiseltest extension

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel Test Benches

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester {
 behavior of "Inverter"

 it should "invert" in {



 }
}
```

one scalatest test

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel Test Benches

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester {
 behavior of "Inverter"

 it should "invert" in {
   test(new Inverter) { dut =>
```

handle to our design

```
   }
 }
}
```

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel Test Benches

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester {
 behavior of "Inverter"

 it should "invert" in {
   test(new Inverter) { dut =>
     dut.in.poke(true.B)
     dut.hold.poke(true.B)


   }
 }
}
```

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel Test Benches

```scala
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester {
 behavior of "Inverter"

 it should "invert" in {
   test(new Inverter) { dut =>
     dut.in.poke(true.B)
     dut.hold.poke(true.B)
     dut.clock.step()

   }
 }
}
```

```scala
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel Test Benches

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester {
 behavior of "Inverter"

 it should "invert" in {
   test(new Inverter) { dut =>
     dut.in.poke(true.B)
     dut.hold.poke(true.B)
     dut.clock.step()
     dut.out.expect(false.B)
   }
 }
}
```

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel Test Benches

```scala
class InverterTest extends AnyFlatSpec with ChiselScalatestTester {
  behavior of "Inverter"

  it should "invert" in {
    test(new Inverter) { dut =>
      dut.in.poke(true.B)
      dut.hold.poke(true.B)
      dut.clock.step()
      dut.out.expect(false.B)
    }
  }
}
```

Chisel Test Benches

```
65  class InverterTest extends AnyFlatSpec with ChiselScalatestTester {
66    behavior of "Inverter"
```

Tests failed: 1 of 1 test

- Test Results
  - InverterTest
    - Inverter
      - should invert

```
out=true (1, 0x1) did not equal expected=false (0, 0x0) (lines in BasicTest.scala: 69)
ScalaTestFailureLocation: chiseltest.tests.InverterTest at (BasicTest.scala:73)
Expected :expected=false (0, 0x0) (lines in BasicTest.scala: 69)
Actual   :out=true (1, 0x1)
<Click to see difference>
```

```
73        dut.out.expect(false.B)
74      }
75    }
76  }
```

Chisel Test Benches

```scala
65 class InverterTest extends AnyFlatSpec with ChiselScalatestTester {
66   behavior of "Inverter"
67
68   it should "invert" in {
69     test(new Inverter) { dut =>
70       dut.in.poke(true.B)
71       dut.hold.poke(true.B)
72       dut.clock.step()
73       dut.out.expect(false.B)
74     }
75   }
76 }
```

```
out=true (1, 0x1) did not equal expected=false (0, 0x0) (lines in BasicTest.scala: 69)
ScalaTestFailureLocation: chiseltest.tests.InverterTest at (BasicTest.scala:73)
```

Chisel Test Benches

```
65  class InverterTest extends AnyFlatSpec with ChiselScalatestTester {
66    behavior of "Inverter"
67
68    it should "invert" in {
69      test(new Inverter) {
70        dut.in.poke(true.B)
71        dut.hold.poke(true.
72        dut.clock.step()
73        dut.out.expect(false.B)
74      }
75    }
76  }
```

Waveform?

```
out=true (1, 0x1) did not equal expected=false (0, 0x0) (lines in BasicTest.scala: 69)
ScalaTestFailureLocation: chiseltest.tests.InverterTest at (BasicTest.scala:73)
```

Chisel Test Benches

```scala
65 class InverterTest extends AnyFlatSpec with ChiselScalatestTester {
66   behavior of "Inverter"
67
68   it should "invert" in {
69     test(new Inverter).withAnnotations(Seq(WriteVcdAnnotation)) { dut =>
70       dut.in.poke(true.B)
71       dut.hold.poke(true.B)
72       dut.clock.step()
73       dut.out.expect(false.B)
74     }
75   }
76 }
```

```
out=true (1, 0x1) did not equal expected=false (0, 0x0) (lines in BasicTest.scala: 69)
ScalaTestFailureLocation: chiseltest.tests.InverterTest at (BasicTest.scala:73)
```

Chisel Test Benches

```
65  class InverterTest extends AnyFlatSpec with ChiselScalatestTester {
66    behavior of "Inverter"
67
68    it should "invert" in {
69      test(new Inverter).withAnnotations(Seq(WriteVcdAnnotation)) { dut =>
70        dut.in.poke(true.B)
71        dut.hold.poke(true.B)
72        dut.clock.step()
73        dut.out.expect(false.B)
74      }
75    }
76  }
```

options passed to our testing framework

```
out=true (1, 0x1) did not equal expected=false (0, 0x0) (lines in BasicTest.scala: 69)
ScalaTestFailureLocation: chiseltest.tests.InverterTest at (BasicTest.scala:73)
```

Chisel Test Benches

```
65 class InverterTest extends AnyFlatSpec with ChiselScalatestTester {
66   behavior of "Inverter"
67
68   it should "invert" in {
69     test(new Inverter).withAnnotations(Seq(WriteVcdAnnotation)) { dut =>
70       dut.in.poke(true.B)
71       dut.hold.poke(true.B)
72       dut.clock.step()
73       dut.out.expect(false.B)
74     }
75   }
76 }
```

the output folder is automatically derived from the test name

```
out=true (1, 0x1) did not equal expected=false (0, 0x0) (lines in BasicTest.scala: 69)
ScalaTestFailureLocation: chiseltest.tests.InverterTest at (BasicTest.scala:73)
```

Chisel Test Benches

```
65  class InverterTest extends AnyFlatSpec with ChiselScalatestTester {
66    behavior of "Inverter"
67
68    it should "invert" in {
69      test(new Inverter).withAnnotations(Seq(WriteVcdAnnotation)) { dut =>
70
71
72
73
74      }
75    }
76  }
```

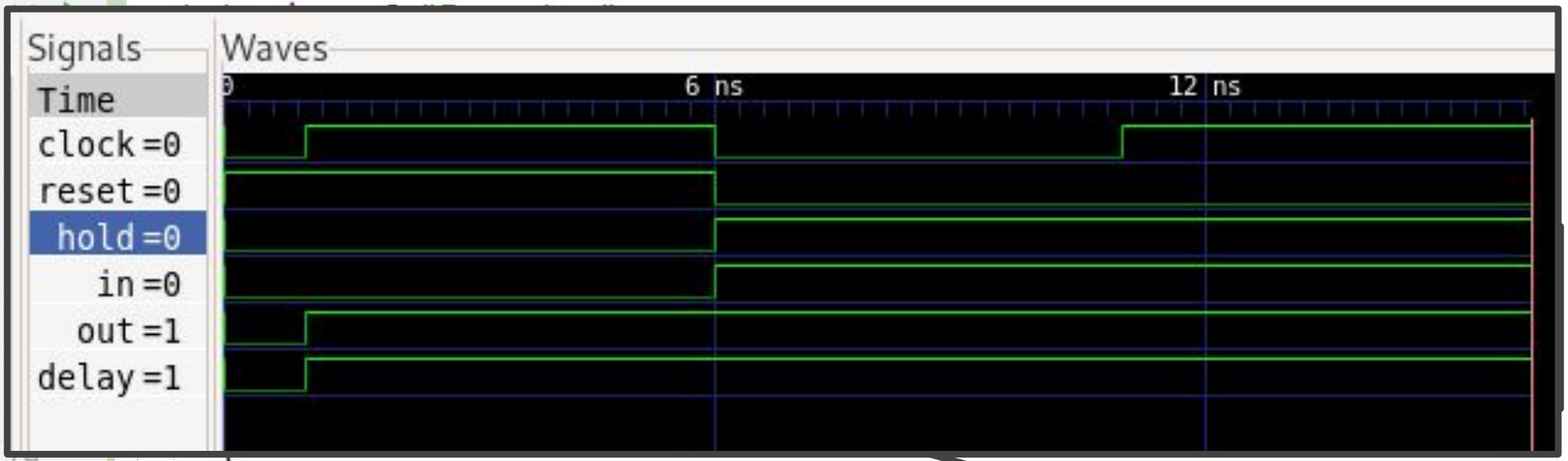
the output folder is automatically derived from the test name

```
> gtkwave test_run_dir/Inverter_should_invert/Inverter.vcd
```

```
out=true (1, 0x1) did not equal expected=false (0, 0x0) (lines in BasicTest.scala: 69)
ScalaTestFailureLocation: chiseltest.tests.InverterTest at (BasicTest.scala:73)
```

Chisel Test Benches

```
65  class InverterTest extends AnyFlatSpec with ChiselScalatestTester {
```




```
75      }
76  }
```

```
out=true (1, 0x1) did not equal expected=false (0, 0x0) (lines in BasicTest.scala: 69)
ScalaTestFailureLocation: chiseltest.tests.InverterTest at (BasicTest.scala:73)
```

Chisel Test Benches

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester {
 behavior of "Inverter"

 it should "invert" in {
   test(new Inverter) { dut =>
     dut.in.poke(true.B)
     dut.hold.poke(true.B)
     dut.clock.step()
     dut.out.expect(false.B)
   }
 }
}
```

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel Test Benches

```scala
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester {
 behavior of "Inverter"

 it should "invert" in {
   test(new Inverter) { dut =>
     dut.in.poke(true.B)
     dut.hold.poke(false.B)
     dut.clock.step()
     dut.out.expect(false.B)
   }
 }
}
```

```scala
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 when(!hold) {
   delay := !in
 }
 out := delay
}
```

Chisel Test Benches

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester {
 be
 it
 }
}
```

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
}
```

Run: InverterTest

Tests passed: 1 of 1 test – 2 sec 910 ms

Test Results 2 sec 910 ms

/usr/lib/jvm/jre-11/bin/java ...

Testing started at 4:02 PM ...

chiseltest simulator backends

Treadle

- default
- fast compilation
- runs on JVM
- slow on larger designs
- no Verilog blackboxes

chiseltest simulator backends

Treadle	Verilator
• default • fast compilation • runs on JVM • slow on larger designs • no Verilog blackboxes	• long compile times • fastest simulator • supports Verilog blackboxes • JVM <-> native communication overhead

chiseltest simulator backends

Treadle	Verilator	VCS
• default • fast compilation • runs on JVM • slow on larger designs • no Verilog blackboxes	• long compile times • fastest simulator • supports Verilog blackboxes • JVM <-> native communication overhead	• commercial • event based • four state • not as well tested

chiseltest simulator backends

Treadle	Verilator	VCS
• default • fast compilation • runs on JVM • slow on larger designs • no Verilog blackboxes	• long compile times • fastest simulator • supports Verilog blackboxes • JVM <-> native communication overhead	• commercial • event based • four state • not as well tested

much faster with JNA in Chisel 3.5!

chiseltest simulator backends

Treadle	Verilator	VCS
• default • fast compilation • runs on JVM • slow on larger designs • no Verilog blackboxes	• long compile times • fastest simulator • supports Verilog blackboxes • JVM <-> native communication overhead	• commercial • event based • four state • not as well tested

much faster with JNA in Chisel 3.5!

smaller wave dumps with FST in Chisel 3.5

chiseltest simulator backends

chiseltest API

Treadle

- default
- fast compilation
- runs on JVM
- slow on larger designs
- no Verilog blackboxes

Verilator

- long compile times
- fastest simulator
- supports Verilog blackboxes
- JVM <-> native communication overhead

VCS

- commercial
- event based
- four state
- not as well tested

much faster with JNA in Chisel 3.5!

smaller wave dumps with FST in Chisel 3.5

chiseltest simulator backends

chiseltest API

Peek Poke Tester

Treadle

- default
- fast compilation
- runs on JVM
- slow on larger designs
- no Verilog blackboxes

Verilator

- long compile times
- fastest simulator
- supports Verilog blackboxes
- JVM <-> native communication overhead

VCS

- commercial
- event based
- four state
- not as well tested

much faster with JNA in Chisel 3.5!

smaller wave dumps with FST in Chisel 3.5

chiseltest simulator backends

chiseltest API	Peek Poke Tester	Synthesizable Tester
Treadle	Verilator	VCS
• default • fast compilation • runs on JVM • slow on larger designs • no Verilog blackboxes	• long compile times • fastest simulator • supports Verilog blackboxes • JVM <-> native communication overhead	• commercial • event based • four state • not as well tested

much faster with JNA in Chisel 3.5!

smaller wave dumps with FST in Chisel 3.5

Formal Verification

Formal Verification with chiseltest

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 when(!hold) {
   delay := !in
 }
 out := delay




}
```

Formal Verification with chiseltest

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 [...]





}
```

Formal Verification with chiseltest

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 [...]

 when(past(hold)) {

 }


}
```

one cycle after hold was asserted

Formal Verification with chiseltest

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 [...]

 when(past(hold)) {
   verification.assert(stable(out))
 }


}
```

one cycle after hold was asserted

the delay register should not have changed

Formal Verification with chiseltest

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 [...]

 when(past(hold)) {
   verification.assert(stable(out))
 }.otherwise {
   verification.assert(out === !past(in))
 }
}
```

otherwise we expect the output to be the inverse of the previous input

Formal Verification with chiseltest

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester {
 behavior of "Inverter"

 it should "invert" in {
   test(new Inverter) { dut =>
     dut.in.poke(true.B)
     dut.hold.poke(true.B)
     dut.clock.step()
     dut.out.expect(false.B)
   }
 }
}
```

Formal Verification with chiseltest

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester with Formal {
 behavior of "Inverter"

 it should "invert" in {
   test(new Inverter) { dut =>
     dut.in.poke(true.B)
     dut.hold.poke(true.B)
     dut.clock.step()
     dut.out.expect(false.B)
   }
 }
}
```

Formal Verification with chiseltest

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester with Formal {
 behavior of "Inverter"

 it should "invert" in {
   verify(new Inverter,
          Seq(BoundedCheck(10)))




 }
}
```

check for 10 cycles after reset

Formal Verification with chiseltest

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester with Formal {
 behavior of "Inverter"

 it should "invert" in {
   verify(new Inverter,
          Seq(BoundedCheck(10)))

 }
}
```

Run: InverterTest.Inverter should invert

Tests passed: 1 of 1 test – 3 sec 514 ms

Test Results 3 sec 514 ms

```
/usr/lib/jvm/jre-11/bin/java ...
Testing started at 4:58 PM ...
```

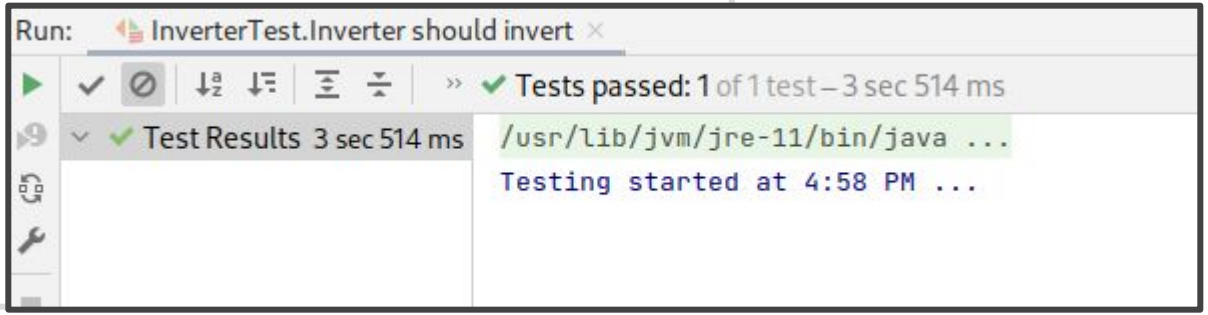
Formal Verification with chiseltest

```
class InverterTest extends AnyFlatSpec
with ChiselScalatestTester with Formal {
 behavior of "Inverter"

 it should "invert" in {
   verify(new Inverter,
          Seq(BoundedCheck(10)))

 }
}
```

Same IDE Integration as Test Benches



Formal Verification with chiseltest

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 [...]

 when(past(hold)) {
   verification.assert(stable(out))
 }.otherwise {
   verification.assert(out === !past(in))
 }
}
```

Formal Verification with chiseltest

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 [...]

 when(RegNext(hold)) {
   verification.assert(stable(out))
 }.otherwise {
   verification.assert(out === !past(in))
 }
}
```

Formal Verification with chiseltest

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 [...]

 when(RegNext(hold)) {
   verification.assert(stable(out))
 }.otherwise {
   verification.assert(out === !RegNext(in))
 }
}
```

Formal Verification with chiseltest

```
class Inverter extends Module {
 val in = IO(Input(Bool()))
 val out = IO(Output(Bool()))
 val hold = IO(Input(Bool()))

 val delay = RegInit(false.B)
 [...]

 when(RegNext(hold)) {
   verification.assert(out === RegNext(out))
 }.otherwise {
   verification.assert(out === !RegNext(in))
 }
}
```

Formal Verification with chiseltest

Run: InverterTest.Inverter should invert

Tests failed: 1 of 1 test

- Test Results
  - InverterTest
    - Inverter
      - should invert

```
Failure Stop:SIM_1:(65) at  @[BasicTest.scala 68:24]

[Inverter] found an assertion violation 0 steps after reset!
```

```
  [...]

  when(RegNext(hold)) {
    verification.assert(out === RegNext(out))
  }.otherwise {
    verification.assert(out === !RegNext(in))
  }
}
```

Formal Verification with chiseltest

Run: InverterTest.Inverter should invert

Tests failed: 1 of 1 test

- Test Results
  - InverterTest
    - Inverter
      - should invert

```
Failure Stop:SIM_1:(65) at  @[BasicTest.scala 68:24]

[Inverter] found an assertion violation 0 steps after reset!
```

```
> gtkwave test_run_dir/Inverter_should_invert/Inverter.vcd
```

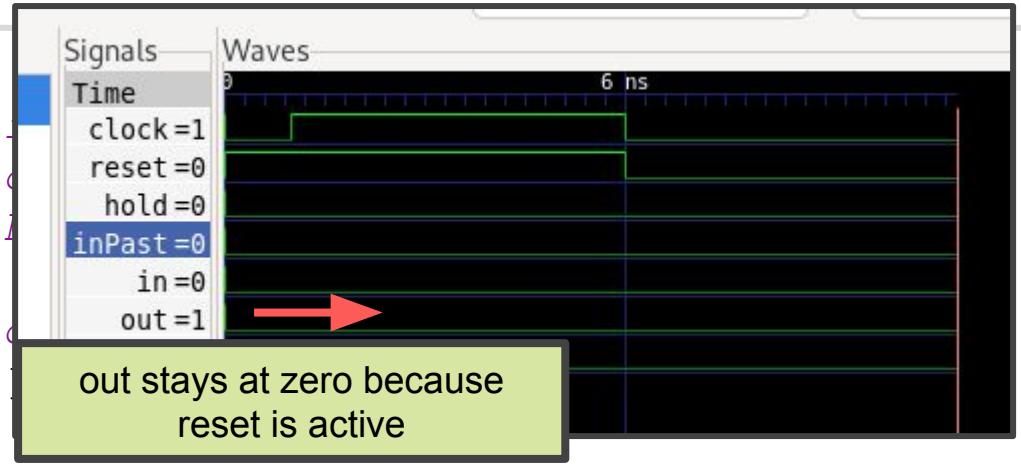
```
    verification.assert(out === !RegNext(in))
  }
}
```

Formal Verification with chiseltest

```
class
 val
 val
 val

 val
 [...

 when(RegNext(hold)) {
   verification.assert(out === RegNext(out))
 }.otherwise {
   verification.assert(out === !RegNext(in))
 }
}
```

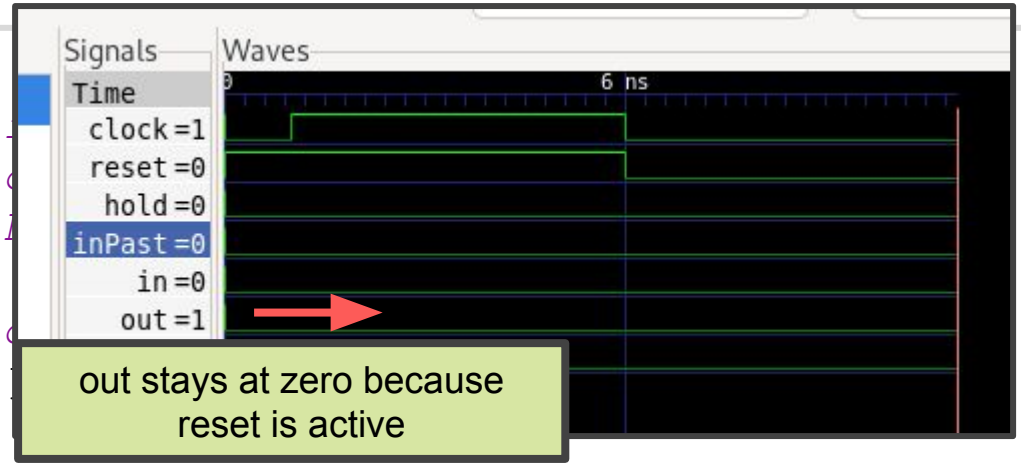


Formal Verification with chiseltest

```
class
 val
 val
 val

 val
 [...

 when(RegNext(hold)) {
   verification.assert(out === RegNext(out))
 }.otherwise {
   verification.assert(out === !RegNext(in))
 }
}
```



out stays at zero because reset is active

We can solve this issue by delaying the temporal assertion until 1 cycle after reset!

Formal Verification with chiseltest

- our new **past** statement adds a RegNext to delay the signal
- it also annotates the register and schedules a compiler pass to be run
- the compiler pass analyzes the longest **past** delay in the fan-in of the assertion
- it then adds a saturating reset counter and appropriately delays assertions

Formal Verification with chiseltest

- all features presented here will be part of the Chisel 3.5 release
- currently we only have support for bounded checks with Z3 or CVC4 since they are the easiest to install engines
- more advanced academic model checkers with unbounded proof support could be targeted (PRs welcome!)
- cover statement support is work in progress