Open-Source Formal Verification for Chisel

Kevin Laeufer <laeufer@berkeley.edu>, Koushik Sen, Jonathan Bachrach

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

Chisel module

signal type

```
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

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? - 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!

Formal Verification Example

Formal Verification Example

```
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 Example

```
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 Example

```
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 Example

```
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)) {
  assert(stable(out))
 }


}
```

one cycle after hold was asserted

the delay register should not have changed

Formal Verification Example

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

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

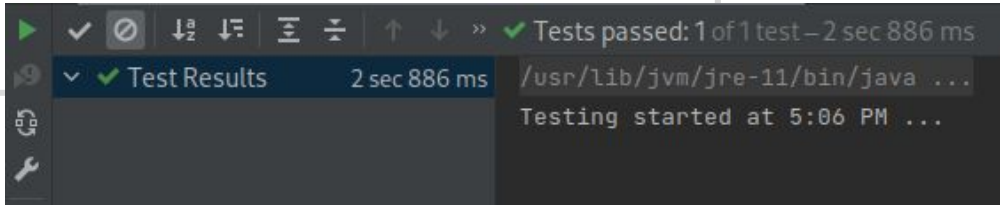
Formal Verification Example

```
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

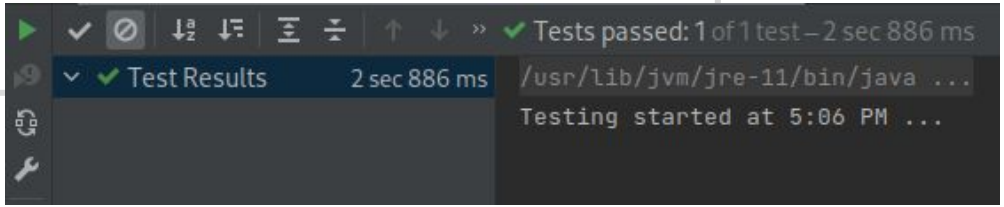
Formal Verification Example

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

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



Formal Verification Example

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

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

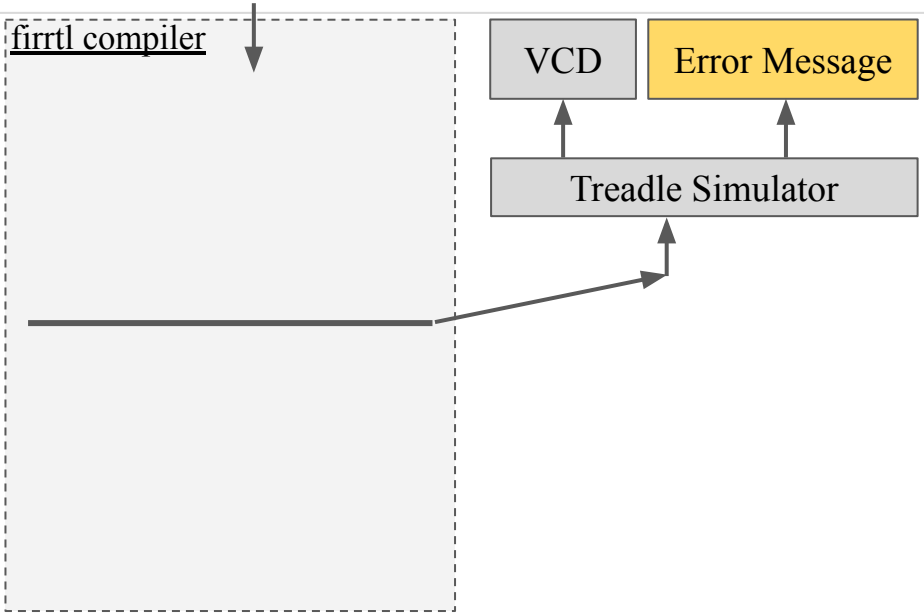


Same IDE Integration as Test Benches

Formal Verification: Behind the Scenes

Verification Flow

```
test(new Design()){ … }
```

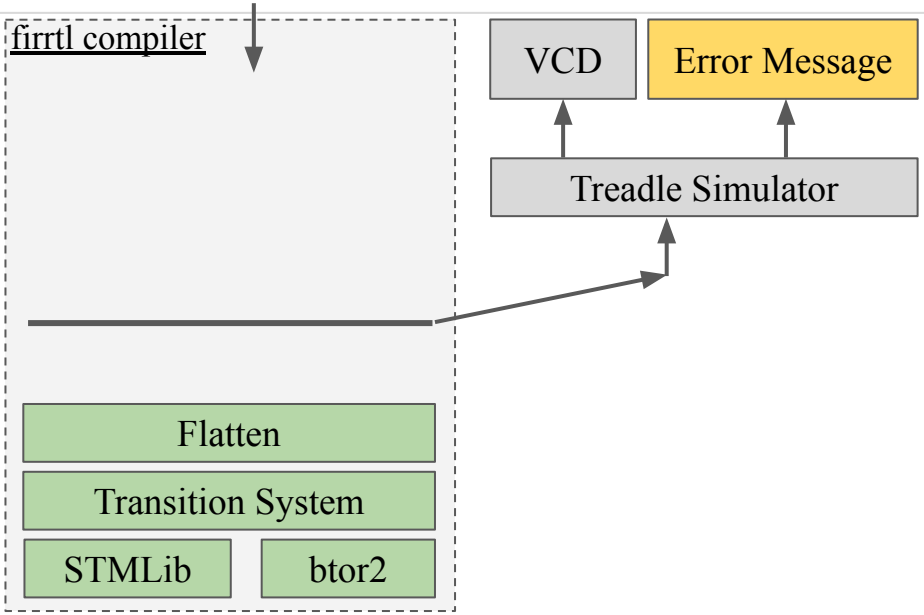


Part of the FIRRTL compiler

Part of the chiseltest library

Verification Flow

```
verify(new Design(), Seq(...)
```



Part of the FIRRTL compiler

Part of the chiseltest library

Inverter Transition System

```
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
}
```

Try it out yourself! https://scastie.scala-lang.org/SYu5jcy3Qe63NXikLfgexA

Inverter Transition System

```scala
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
}
```

```
Inverter
input reset : bv<1>
input in    : bv<1>
input hold  : bv<1>
node _T : bv<1> = eq(hold, 1'b0)
node _delay_T : bv<1> = eq(in, 1'b0)
output out : bv<1> = delay
state delay : bv<1>
  [next] ite(_T, _delay_T, delay)
```

Try it out yourself! https://scastie.scala-lang.org/SYu5jcy3Qe63NXikLfgexA

Inverter Transition System

```scala
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
}
```

```
Inverter
input reset : bv<1>
input in    : bv<1>
input hold  : bv<1>
node _T : bv<1> = eq(hold, 1'b0)
node _delay_T : bv<1> = eq(in, 1'b0)
output out : bv<1> = delay
state delay : bv<1>
  [next] ite(_T, _delay_T, delay)
```

Try it out yourself! https://scastie.scala-lang.org/SYu5jcy3Qe63NXikLfgexA

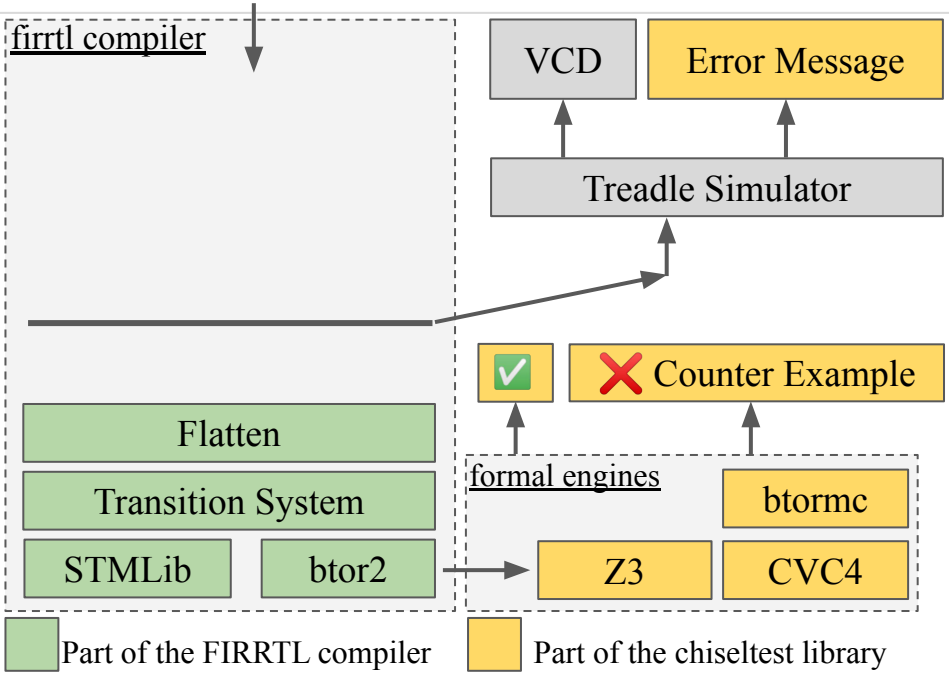
Inverter Transition System

```
; BTOR description generated by firrtl 1.5.0-RC1 for module
Inverter.
1 sort bitvec 1
2 input 1 reset
3 input 1 in
4 input 1 hold
5 state 1 delay ; @[main.scala 15:17]
6 zero 1
7 eq 1 4 6 ; @[main.scala 16:7]
8 zero 1
9 eq 1 3 8 ; @[main.scala 17:13]
10 ite 1 7 9 5 ; @[main.scala 16:14 17:10 15:17]
11 output 5 ; out @[main.scala 19:6]
; delay.next
12 next 1 5 10
```

Try it out yourself! https://scastie.scala-lang.org/SYu5jcy3Qe63NXikLfgexA

Verification Flow

```
verify(new Design(), Seq(BoundedCheck(5))
```



Arbitrary Values

```
class AIs(value: Int) extends Module {
 val a = Wire(UInt(2.W))
 a := DontCare
 assert(a === value.U)
}
```

Arbitrary Values

```
class AIs(value: Int) extends Module {
 val a = Wire(UInt(2.W))
 a := DontCare
 assert(a === value.U)
}
```

```
verify(new AIs(1),Seq(BoundedCheck(2)))
verify(new AIs(2),Seq(BoundedCheck(2)))
verify(new AIs(3),Seq(BoundedCheck(2)))
verify(new AIs(0),Seq(BoundedCheck(2)))
```

Arbitrary Values

```
class AIs(value: Int) extends Module {
 val a = Wire(UInt(2.W))
 a := DontCare
 assert(a === value.U)
}
```

```
verify(new AIs(1),Seq(BoundedCheck(2))) ❌
verify(new AIs(2),Seq(BoundedCheck(2))) ❌
verify(new AIs(3),Seq(BoundedCheck(2))) ❌
verify(new AIs(0),Seq(BoundedCheck(2))) ✅
```

Arbitrary Values

```
class AIs(value: Int) extends Module {
 val a = Wire(UInt(2.W))
 a := DontCare
 assert(a === value.U)
}
```

by default firrtl *simplifies* a to 0

```
verify(new AIs(1),Seq(BoundedCheck(2))) ❌
verify(new AIs(2),Seq(BoundedCheck(2))) ❌
verify(new AIs(3),Seq(BoundedCheck(2))) ❌
verify(new AIs(0),Seq(BoundedCheck(2))) ✅
```

Arbitrary Values

```
class AIs(value: Int) extends Module {
 val a = Wire(UInt(2.W))
 a := DontCare
 assert(a === value.U)
}
```

```
verify(new AIs(1),Seq(BoundedCheck(2))) ❌
verify(new AIs(2),Seq(BoundedCheck(2))) ❌
verify(new AIs(3),Seq(BoundedCheck(2))) ❌
verify(new AIs(0),Seq(BoundedCheck(2)))
```

Invalid to Random

```
rand a_invalid : UInt<2>
assert(clock,
 eq(a_invalid, UInt(0)),
 not(reset), "")
```

Arbitrary Values

```
class AIs(value: Int) extends Module {
 val a = Wire(UInt(2.W))
 a := DontCare
 assert(a === value.U)
}
```

```
verify(new AIs(1),Seq(BoundedCheck(2))) ❌
verify(new AIs(2),Seq(BoundedCheck(2))) ❌
verify(new AIs(3),Seq(BoundedCheck(2))) ❌
verify(new AIs(0),Seq(BoundedCheck(2))) ❌
```

Invalid to Random

```
rand a_invalid : UInt<2>
assert(clock,
 eq(a_invalid, UInt(0)),
 not(reset), "")
```

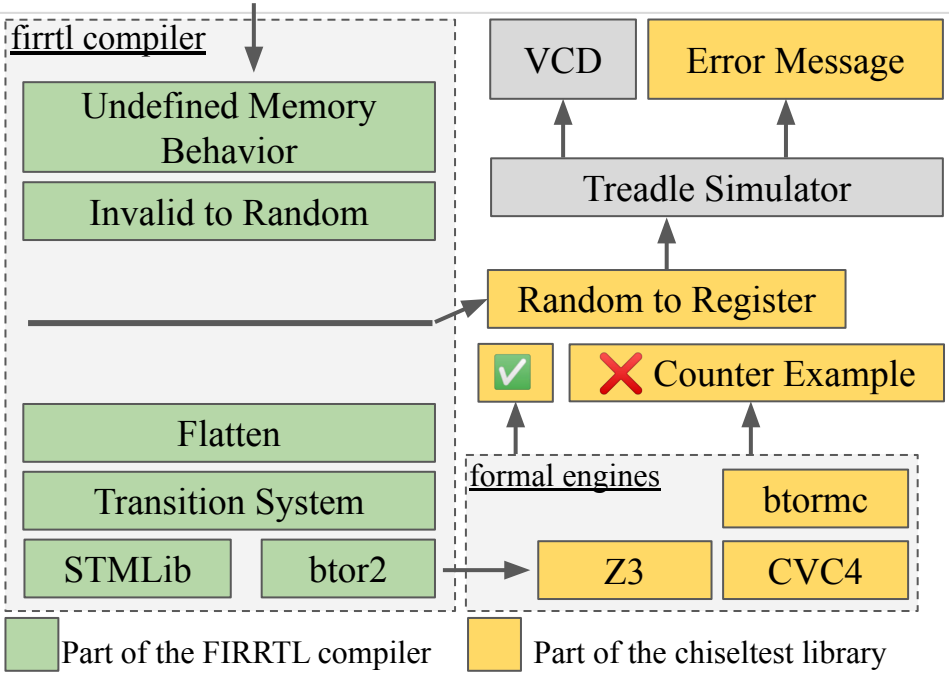
Memory Behavior

- A similar approach is used to model arbitrary values in memories
- Example: read data when read enable is false

https://github.com/ucb-bar/chisel-testers2/blob/master/src/test/scala/chiseltest/formal/MemoryTests.scala

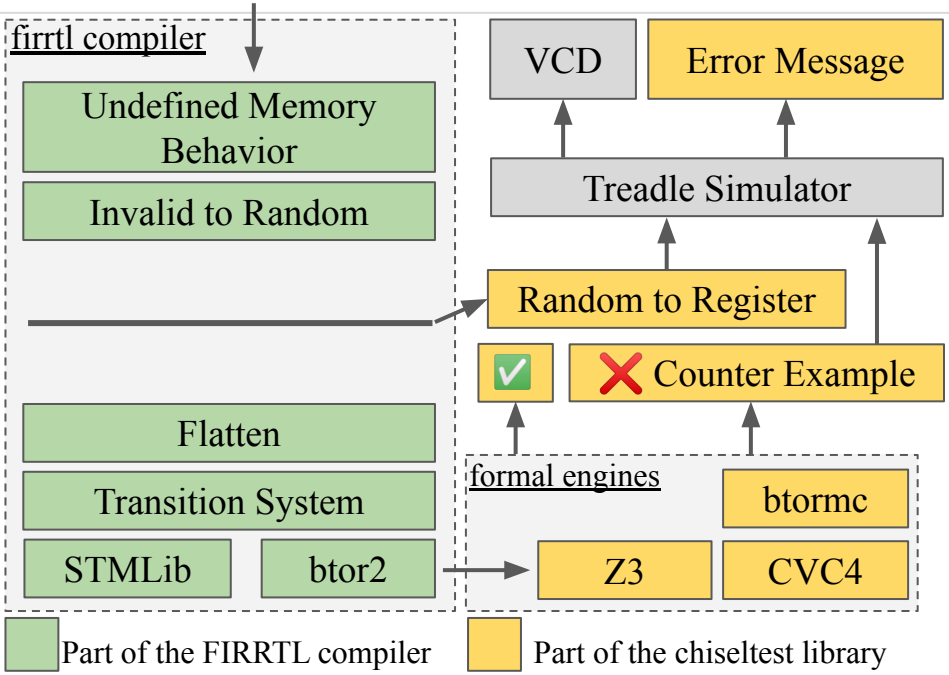
Verification Flow

```
verify(new Design(), Seq(BoundedCheck(5))
```



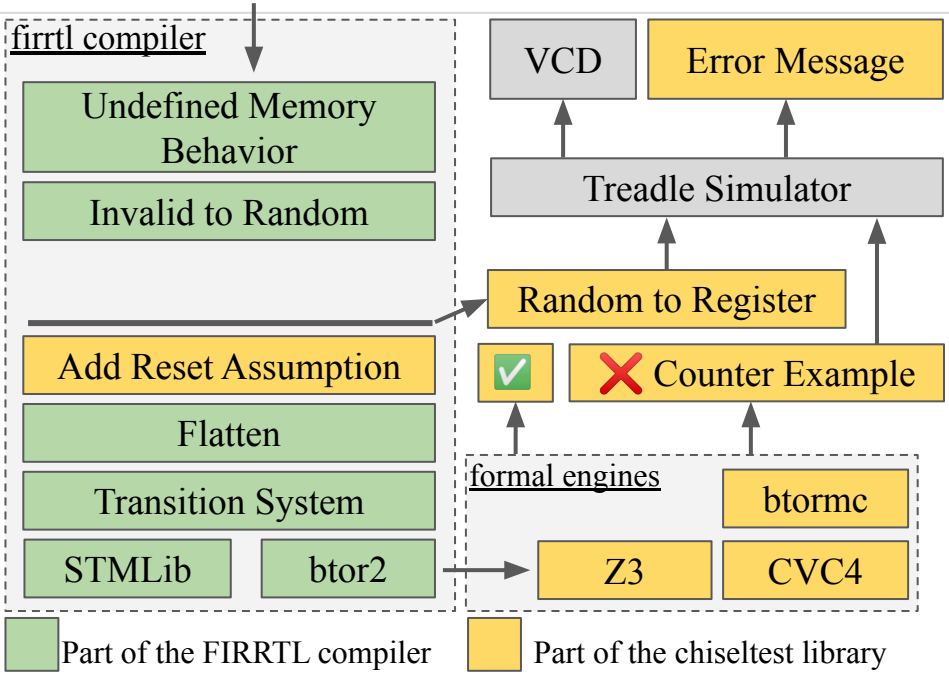
Verification Flow

```
verify(new Design(), Seq(BoundedCheck(5))
```



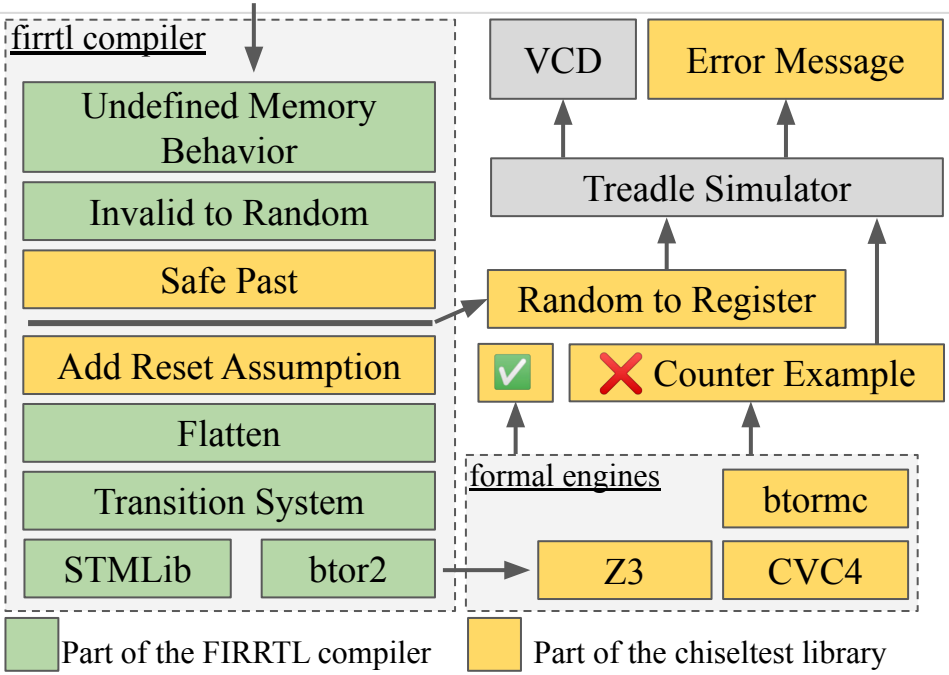
Verification Flow

```
verify(new Design(), Seq(BoundedCheck(5))
```



Verification Flow

```
verify(new Design(), Seq(BoundedCheck(5))
```



Formal Verification Example

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

Formal Verification Example

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

Formal Verification Example

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

Formal Verification Example

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

Formal Verification Example

Run: InverterTest.Inverter should fail a bounded check w...

Tests failed: 1 of 1 test

- Test Results
  - InverterTest
    - Inverter
      - should fail a bounded ch

```
Assertion failed
    at Inverter.scala:41 assert(out === RegNext(in))
```

```
  [...]

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

Formal Verification Example

Run: InverterTest.Inverter should fail a bounded check w...

Tests failed: 1 of 1 test

- Test Results
  - InverterTest
    - Inverter
      - should fail a bounded ch

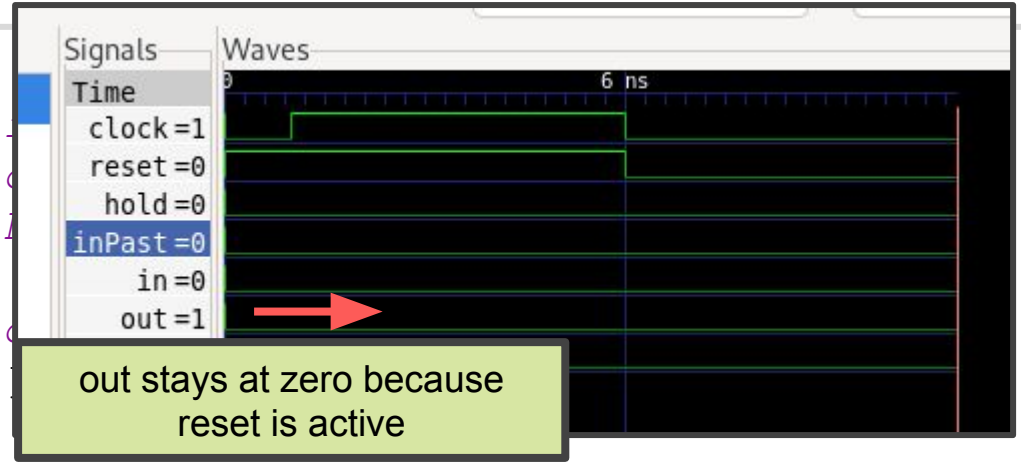
```
Assertion failed
    at Inverter.scala:41 assert(out === RegNext(in))
```

```
[...]
```

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

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

Formal Verification Example

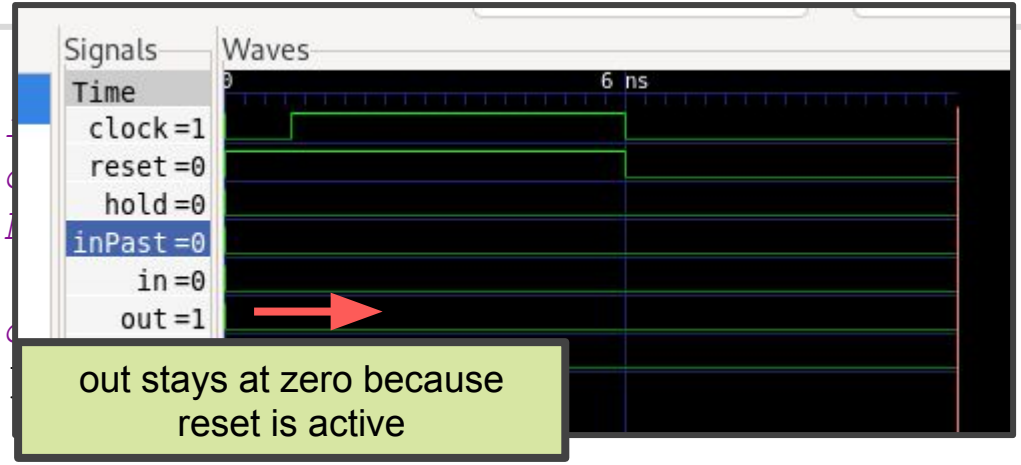


out stays at zero because reset is active

```
class
 val
 val
 val

 val
 [...

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

Formal Verification Example

```
class
 val
 val
 val

 val
 [...

 when(RegNext(hold)) {
  assert(out === RegNext(out))
 }.otherwise {
  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 Example

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

Open-Source Formal Verification for Chisel

Kevin Laeufer
<laeufer@berkeley.edu>

github.com/ekiwi

- New formal backend to firrtl
- Accurate modelling of arbitrary values
- Safe and simple temporal assertions
- Easy verification setup and counterexample replay from a Scala unit test
- Scala embedding + firrtl compiler → prototype you formal apps!

Please join the community!

https://github.com/chipsalliance/chisel3

https://gitter.im/freechipsproject/chisel3

Try out all our examples:

https://github.com/ekiwi/open-source-formal-verification-for-chisel