Introduction to Chisel/FIRRTL Hardware Compiler Framework

Boyang Han
yqszxx@gmail.com

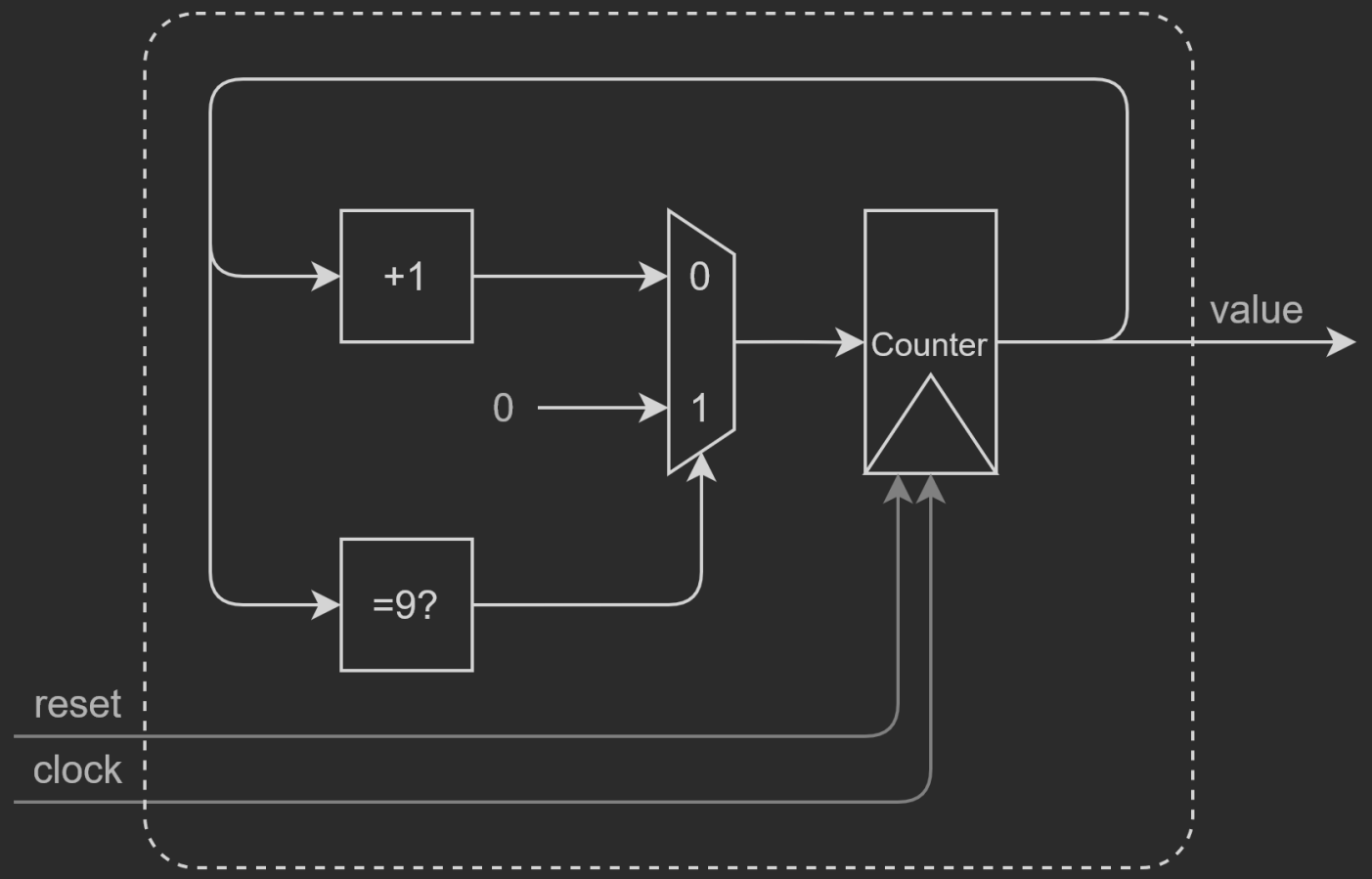
7/15/20

What is Chisel/FIRRTL HCF?

- Chisel: Constructing Hardware In a Scala Embedded Language
- https://github.com/freechipsproject/chisel3

- FIRRTL: Flexible Internal Representation for RTL
- https://github.com/freechipsproject/firrtl

Example of the Day



Decimal Counter

What does Chisel code look like?

```
1  import chisel3._
2
3  class DecCounter extends Module {
4    val io = IO(new Bundle{
5      val value = Output(UInt(4.W))
6    })
7
8    val counter = RegInit(0.U(4.W))
9
10   io.value := counter
11
12   when (counter === 9.U) {
13     counter := 0.U
14   } otherwise {
15     counter := counter + 1.U
16   }
17 }
```

Get me the hardware!

```scala
object main extends App {
  import stage._

  (new ChiselStage).execute(
    Array(
      "-X", "verilog",
    ),
    Seq(ChiselGeneratorAnnotation(() => new DecCounter))
  )
}
```

```verilog
module DecCounter(
  input        clock,
  input        reset,
  output [3:0] io_value
);
  reg [3:0] counter; // @[main.scala 8:24]
  wire  _T = counter == 4'h9; // @[main.scala 12:17]
  wire [3:0] _T_2 = counter + 4'h1; // @[main.scala 15:24]
  assign io_value = counter; // @[main.scala 10:12]
  always @(posedge clock) begin
    if (reset) begin
      counter <= 4'h0;
    end else if (_T) begin
      counter <= 4'h0;
    end else begin
      counter <= _T_2;
    end
  end
endmodule
```

Under the Hood

- Annotation: a piece of information about your design.
- AnnotationSeq: a sequence of annotations.

- Phase: a transform defined on AnnotationSeq and may have side effects.

  e.g. a function looked like this:

$$f: AnnotationSeq \rightarrow AnnotationSeq$$

- Stage: a group of phases.

ChiselStage

```
17 class ChiselStage extends Stage with PreservesAll[Phase] {
.. .....
21   val targets: Seq[Dependency[Phase]] =
22     Seq( Dependency[chisel3.stage.phases.Checks],
23          Dependency[chisel3.stage.phases.AddImplicitOutputFile],
24          Dependency[chisel3.stage.phases.AddImplicitOutputAnnotationFile],
25          Dependency[chisel3.stage.phases.MaybeAspectPhase],
26          Dependency[chisel3.stage.phases.Emitter],
27          Dependency[chisel3.stage.phases.Convert],
28          Dependency[chisel3.stage.phases.MaybeFirrtlStage] )
.. .....
111 }
```

src/main/scala/chisel3/stage/ChiselStage.scala @ 2f9f12a8

chisel3.stage.phases.Checks

```
14  class Checks extends Phase with PreservesAll[Phase] {
15
16    override val dependents = Seq(Dependency[Elaborate])
..  .....
111 }
```

src/main/scala/chisel3/stage/phases/Checks.scala @ 2f9f12a8

chisel3.stage.phases.Elaborate

```scala
class Elaborate extends Phase with PreservesAll[Phase] {

  def transform(annotations: AnnotationSeq): AnnotationSeq = annotations.flatMap {
    case a: ChiselGeneratorAnnotation => a.elaborate
    case a                            => Some(a)
  }

}
```

src/main/scala/chisel3/stage/phases/Elaborate.scala @ 2f9f12a8

chisel3.stage.ChiselGeneratorAnnotation

```
45 case class ChiselGeneratorAnnotation(gen: () => RawModule) extends NoTargetAnnotation
   with Unserializable {
.. .....
49   def elaborate: AnnotationSeq  = try {
50     val (circuit, dut) = Builder.build(Module(gen()))
51     Seq(ChiselCircuitAnnotation(circuit), DesignAnnotation(dut))
52   } catch {
.. .....
56   }
57
58 }
```

chisel3/stage/ChiselAnnotations.scala @ 2f9f12a8

Let's walk through

```
1  object main extends App {
2    import stage._
3
4    (new ChiselStage).execute(
5      Array(
6        "-X", "verilog",
7      ),
8      Seq(ChiselGeneratorAnnotation(() => new DecCounter))
9    )
10 }
```

```
CompilerAnnotation(compiler: Compiler =
new VerilogCompiler())
```

```
ChiselGeneratorAnnotation(gen: () => RawModule =
() => new DecCounter))
```

Let's walk through

```
ChiselGeneratorAnnotation(gen: () => RawModule = () => new DecCounter))
```

```
CompilerAnnotation(compiler: Compiler = new VerilogCompiler())
```

Elaborate

~~ChiselGeneratorAnnotation(gen: () => RawModule = () => new DecCounter))~~

```
ChiselCircuitAnnotation(circuit: Circuit = myCircuitDesign)
```

```
DesignAnnotation[DUT <: RawModule](design: DUT = myCircuitModule)
```

```
CompilerAnnotation(compiler: Compiler = new VerilogCompiler())
```

Entering FIRRTL domain

```
17 class ChiselStage extends Stage with PreservesAll[Phase] {
.. .....
21   val targets: Seq[Dependency[Phase]] =
22     Seq( Dependency[chisel3.stage.phases.Checks],
23          Dependency[chisel3.stage.phases.AddImplicitOutputFile],
24          Dependency[chisel3.stage.phases.AddImplicitOutputAnnotationFile],
25          Dependency[chisel3.stage.phases.MaybeAspectPhase],
26          Dependency[chisel3.stage.phases.Emitter],
27          Dependency[chisel3.stage.phases.Convert],
28          Dependency[chisel3.stage.phases.MaybeFirrtlStage] )
.. .....
111 }
```

```
a: ChiselCircuitAnnotation =>
Some(FirrtlCircuitAnnotation(Converter.convert(a.circuit))) ++ ...
```

src/main/scala/chisel3/stage/ChiselStage.scala @ 2f9f12a8
src/main/scala/chisel3/stage/phases/Convert.scala : 22-24 @ 2f9f12a8

Entering FIRRTL domain

```
ChiselCircuitAnnotation(circuit: Circuit = myCircuitDesign)
```

```
.....
```

```
CompilerAnnotation(compiler: Compiler = new VerilogCompiler())
```

Convert

~~ChiselCircuitAnnotation(circuit: Circuit = myCircuitDesign)~~

```
FirrtlCircuitAnnotation(circuit: Circuit = myCircuitDesignInFIRRTL)
```

```
.....
```

```
CompilerAnnotation(compiler: Compiler = new VerilogCompiler())
```

Entering FIRRTL domain

```scala
class MaybeFirrtlStage extends Phase with PreservesAll[Phase] {

  override val prerequisites = Seq(Dependency[Convert])

  def transform(annotations: AnnotationSeq): AnnotationSeq = annotations
    .collectFirst { case NoRunFirrtlCompilerAnnotation => annotations }
    .getOrElse    { (new FirrtlStage).transform(annotations)          }

}
```

src/main/scala/chisel3/stage/phases/MaybeFirrtlStage.scala @ 2f9f12a8

FIR

- Serialized form: .fir file, https://github.com/freechipsproject/firrtl/tree/master/spec

  Or formal lexical & syntax definition @src/main/antlr4/FIRRTL.g4
- Internal representation: `firrtl.ir.Circuit`

```
923 case class Circuit(info: Info, modules: Seq[DefModule], main: String) extends FirrtlNode with HasInfo {
... .....
933 }
```

- CircuitState: a wrapper of `firrtl.ir.Circuit` and its annotations.

src/main/scala/firrtl/ir/IR.scala @ 16037c7
src/main/scala/firrtl/Compiler.scala : 36-87 @ 16037c7

FIRRTL Compiler Internals

- Pass / Transform: a transform defined on CircuitState.

  e.g. a function looked like this:

$$f: CircuitState \rightarrow CircuitState$$

Circuit after `chisel3.stage.phases.Convert`

```
circuit DecCounter :
  module DecCounter :
    input clock : Clock
    input reset : UInt<1>
    output io : { value : UInt<4>}

    reg counter : UInt<4>, clock with :
      reset => (reset, UInt<4>("h0")) @[main.scala 8:24]
    io.value <= counter @[main.scala 10:12]
    node _T = eq(counter, UInt<4>("h9")) @[main.scala 12:17]
    when _T : @[main.scala 12:26]
      counter <= UInt<1>("h0") @[main.scala 13:13]
    else :
      node _T_1 = add(counter, UInt<1>("h1")) @[main.scala 15:24]
      node _T_2 = tail(_T_1, 1) @[main.scala 15:24]
      counter <= _T_2 @[main.scala 15:13]
```

Circuit after `firrtl.passes.ExpandWhensAndCheck`

```
circuit DecCounter :
  module DecCounter :
    input clock : Clock
    input reset : UInt<1>
    output io : { value : UInt<4>}

    reg counter : UInt<4>, clock with :
      reset => (reset, UInt<4>("h0")) @[main.scala 8:24]
    node _T = eq(counter, UInt<4>("h9")) @[main.scala 12:17]
    node _T_1 = add(counter, UInt<1>("h1")) @[main.scala 15:24]
    node _T_2 = tail(_T_1, 1) @[main.scala 15:24]
    node _GEN_0 = mux(_T, UInt<1>("h0"), _T_2) @[main.scala 12:26]
    io.value <= counter @[main.scala 10:12]
    counter <= _GEN_0 @[main.scala 13:13 main.scala 15:13]
```

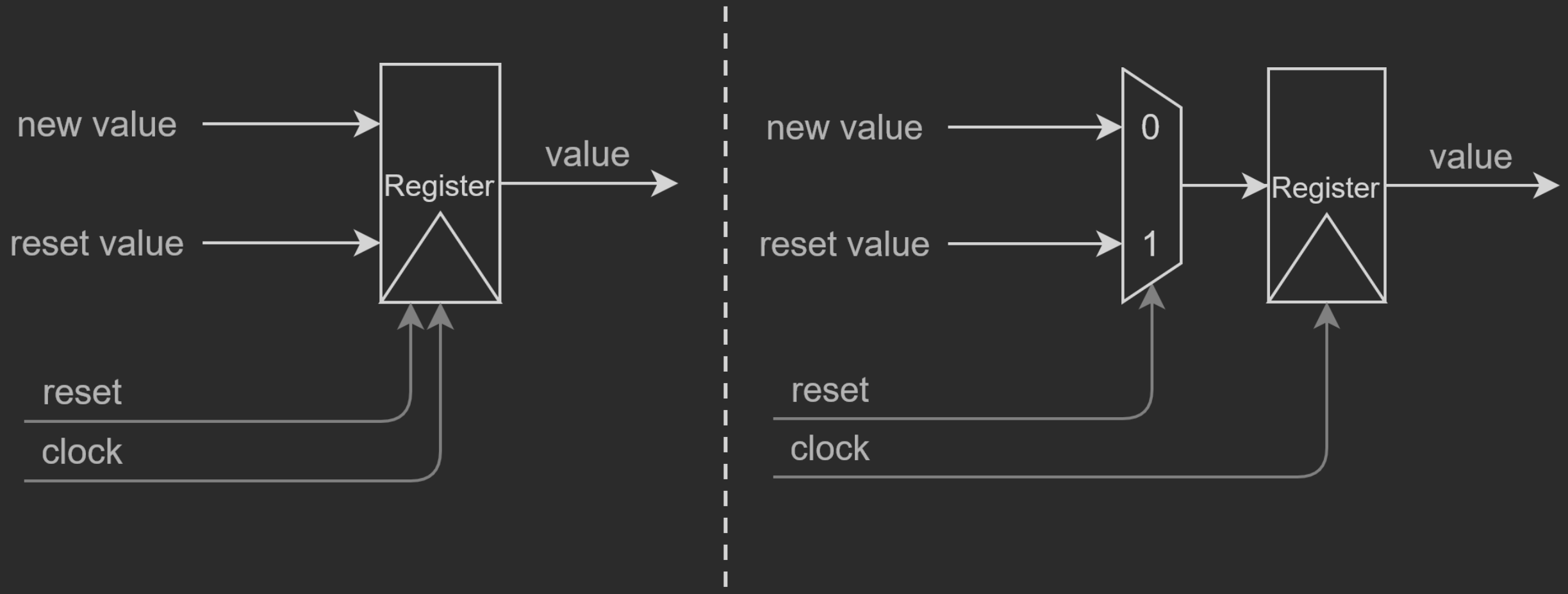
Circuit after `firrtl.passes.LowerTypes`

```
circuit DecCounter :
  module DecCounter :
    input clock : Clock
    input reset : UInt<1>
    output io_value : UInt<4>

    reg counter : UInt<4>, clock with :
      reset => (reset, UInt<4>("h0")) @[main.scala 8:24]
    node _T = eq(counter, UInt<4>("h9")) @[main.scala 12:17]
    node _T_1 = add(counter, UInt<1>("h1")) @[main.scala 15:24]
    node _T_2 = tail(_T_1, 1) @[main.scala 15:24]
    node _GEN_0 = mux(_T, UInt<1>("h0"), _T_2) @[main.scala 12:26]
    io_value <= counter @[main.scala 10:12]
    counter <= _GEN_0 @[main.scala 13:13 main.scala 15:13]
```

Circuit after `firrtl.transforms.RemoveReset`

```
circuit DecCounter :
  module DecCounter :
    input clock : Clock
    input reset : UInt<1>
    output io_value : UInt<4>

    reg counter : UInt<4>, clock with :
      reset => (UInt<1>("h0"), counter) @[main.scala 8:24]
    node _T = eq(counter, UInt<4>("h9")) @[main.scala 12:17]
    node _T_1 = add(counter, UInt<1>("h1")) @[main.scala 15:24]
    node _T_2 = tail(_T_1, 1) @[main.scala 15:24]
    node _GEN_0 = mux(_T, UInt<1>("h0"), _T_2) @[main.scala 12:26]
    io_value <= counter @[main.scala 10:12]
    counter <= mux(reset, UInt<4>("h0"), _GEN_0) @[main.scala 13:13 main.scala 15:13]
```

firrtl.transforms.RemoveReset



firrtl.transforms.RemoveReset

Create an empty $Map[String \rightarrow Reset], \mathrm{M}$

For all registers r with reset:

$$\mathrm{M}[r.name] \leftarrow r.reset$$

$$r.reset \leftarrow 0$$

For all connections c which is an input of a register with reset:

$$originalReset \leftarrow \mathrm{M}[c.out.name]$$

replace c with
$Mux(originalReset.active, c.in, originalReset.value)$

src/main/scala/firrtl/transforms/RemoveReset.scala : 38-78 @ 16037c7

One step further

- How does Chisel capture my design?
- How was dependencies between Chisel Phases, FIRRTL passes / transforms resolved?
- How annotations, FIR serialization & deserialization works?
- .....

Q & A

Thanks!

Boyang Han
yqszxx@gmail.com