Circuit Capture of Chisel
Part 1. Correctness

Boyang Han
yqszxx@gmail.com

7/22/20

What is circuit capture?

```scala
import chisel3._

class DecCounter extends Module {
  val io = IO(new Bundle{
    val value = Output(UInt(4.W))
  })

  val counter = RegInit(0.U(4.W))

  io.value := counter

  when (counter === 9.U) {
    counter := 0.U
  } otherwise {
    counter := counter + 1.U
  }
}
```

```
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]
```

Problems involved

- Correctness
- Naming
- Debug info

Correctness

```scala
import chisel3._

class DecCounter extends Module {
  val io = IO(new Bundle{
    val value = Output(UInt(4.W))
  })

  val counter = RegInit(0.U(4.W))

  io.value := counter

  when (counter === 9.U) {
    counter := 0.U
  } otherwise {
    counter := counter + 1.U
  }
}
```

```scala
final def + (that: T): T = macro SourceInfoTransform.thatArg
```

src/main/scala/chisel3/Num.scala @ 4a0e828

```scala
override def do_+ (that: UInt)(···): UInt = this +% that
```

src/main/scala/chisel3/Bits.scala @ 4a0e828

```scala
def do_+% (that: UInt)(···): UInt =
          (this +& that).tail(1)
```

src/main/scala/chisel3/Bits.scala @ 4a0e828

```scala
def do_+& (that: UInt)(···): UInt =
 binop(···, UInt((this.width max that.width) + 1), AddOp, that)
```

src/main/scala/chisel3/Bits.scala @ 4a0e828

```scala
def binop[T <: Data](···, dest: T, op: PrimOp, other: Bits): T = {
  ···
  pushOp(DefPrim(sourceInfo, dest, op, this.ref, other.ref))
}
```

src/main/scala/chisel3/Bits.scala @ 4a0e828

```
node _T_1 = add(counter, UInt<1>("h1")) @[main.scala 15:24]
node _T_2 = tail(_T_1, 1) @[main.scala 15:24]
```

Correctness

```
47 final def + (that: T): T = macro SourceInfoTransform.thatArg
```

def *op* (that: T): T = macro SourceInfoTransform.thatArg

def *op* (that: T): T =
do_*op* (that) (**[Magically Filled SourceInfo]**)
}

UInt<1>
("h1")

UInt<?>
counter

src/main/scala/chisel3/Num.scala @ 4a0e828

Correctness

```
447 override def do_+ (that: UInt)(···): UInt = this +% that
```

Every Week Scala:
`.` and `()` can be omitted when calling methods.

```scala
class Cat {
  def say(word: String): Unit = println(word)
}

val kitty = new Cat
kitty say "Meow~"
```

```
defined class Cat

kitty: Cat = Cat@23f5da49
Meow~
```

```
def do_+ (that: UInt)(···): UInt = this.+%(that)
```

UInt<1>
("h1")

UInt<?>
counter

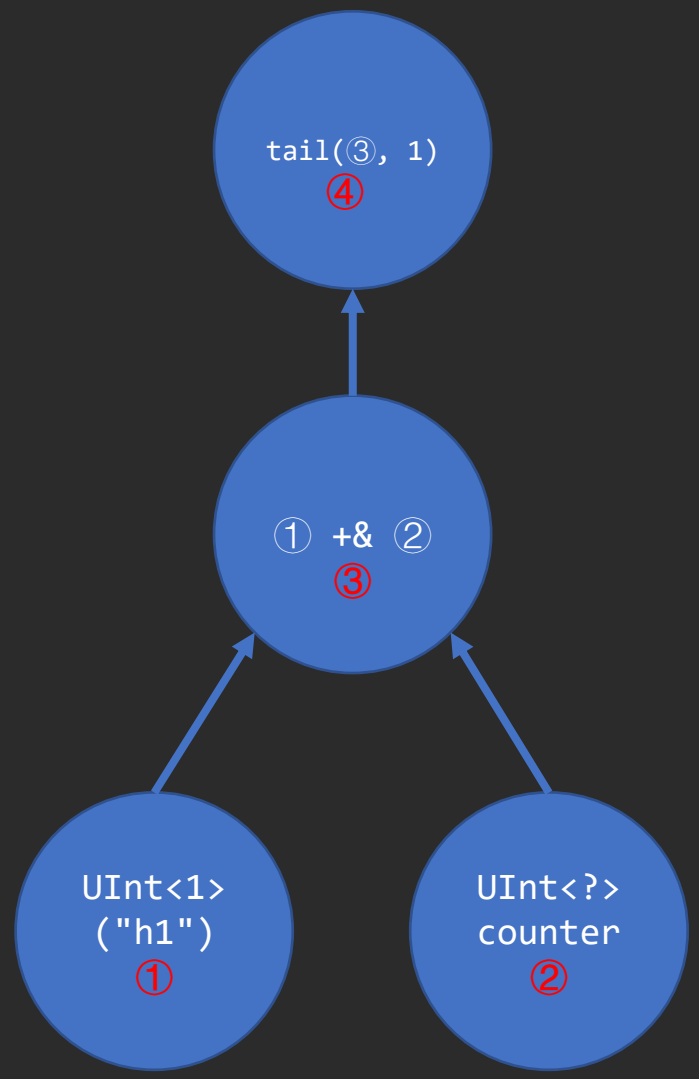
src/main/scala/chisel3/Bits.scala @ 4a0e828

Correctness

```
508 def do_+% (that: UInt)(···): UInt =
509             (this +& that).tail(1)
```

```
 79 def do_tail(n: Int)(···): UInt = {
 79   val w = width match {
 79     case KnownWidth(x) =>
 79       require(x >= n, [message])
 79       Width(x - n)
 79     case UnknownWidth() => Width()
 79   }
 79   binop(···, UInt(width = w), TailOp, n)
 79 }
```

```
195 def binop[T <: Data](···, dest: T, op: PrimOp, other: BigInt): T = {
196   ···
197   pushOp(DefPrim(···, dest, op, this.ref, ILit(other)))
198 }
```

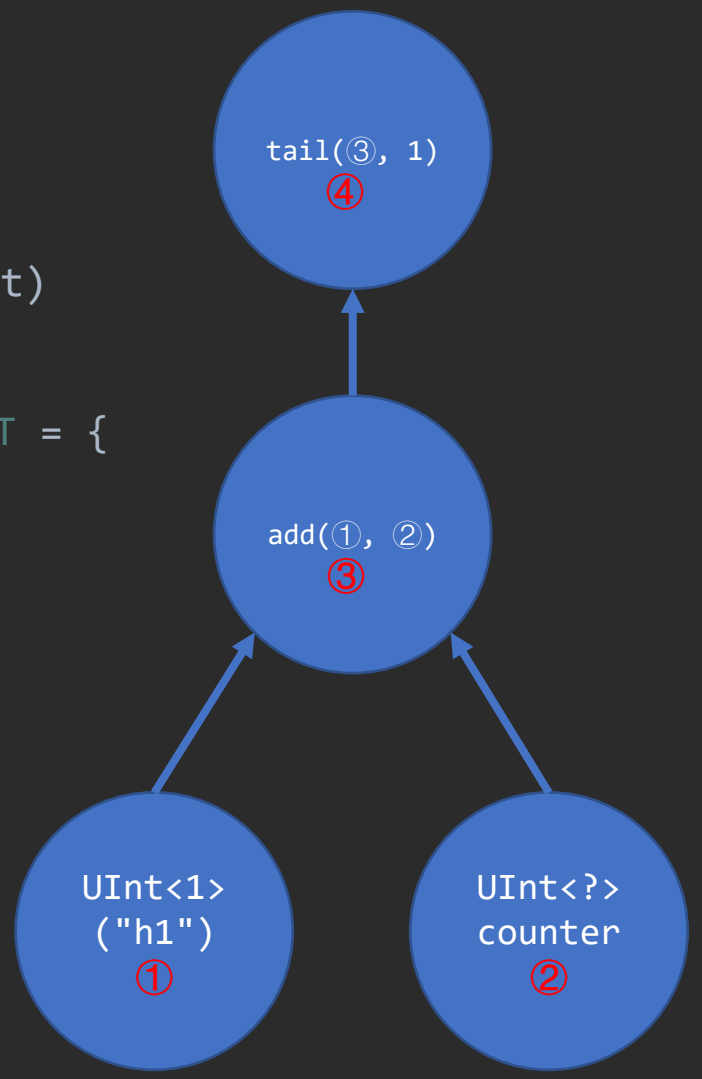


src/main/scala/chisel3/Bits.scala @ 4a0e828

Correctness

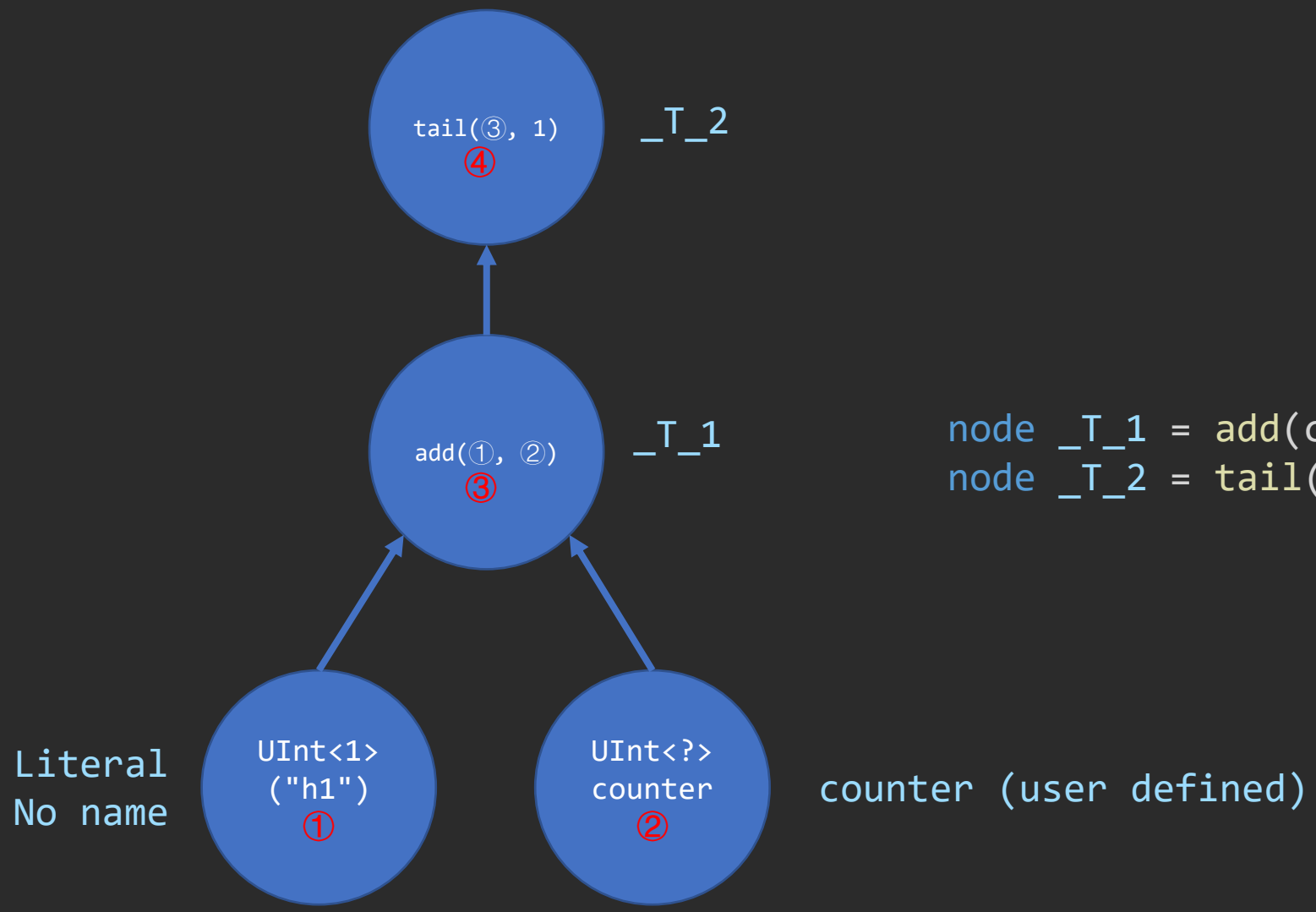
```
505 def do_+& (that: UInt)(···): UInt =
506  binop(···, UInt((this.width max that.width) + 1), AddOp, that)
```

```
199 def binop[T <: Data](···, dest: T, op: PrimOp, other: Bits): T = {
···   ···
202   pushOp(DefPrim(sourceInfo, dest, op, this.ref, other.ref))
203 }
```



src/main/scala/chisel3/Bits.scala @ 4a0e828

Correctness




```
node _T_1 = add(counter, UInt<1>("h1"))
node _T_2 = tail(_T_1, 1)
```

Result

```scala
import chisel3._

class DecCounter extends Module {
  val io = IO(new Bundle{
    val value = Output(UInt(4.W))
  })

  val counter = RegInit(0.U(4.W))

  io.value := counter

  when (counter === 9.U) {
    counter := 0.U
  } otherwise {
    counter := counter + 1.U
  }
}
```

```
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]
```

Q & A

Thanks!

Boyang Han
yqszxx@gmail.com