Introduction to Chisel

王淼

2021-06-17

Background

- 数字电路设计中常用的三种方法
  - HDL（Hardware Description Language）：Verilog HDL / VHDL
  - HLS（High Level Synthesis）：Vivado HLS / Vitis HLS / SDSoC / Intel HLS
  - HCL（Hardware Construction Language)：Chisel / BlueSpec / SpinalHDL
- HDL缺少抽象、重用、参数化生成等功能
- HLS适用于快速迭代且对硬件时序要求不高的场景

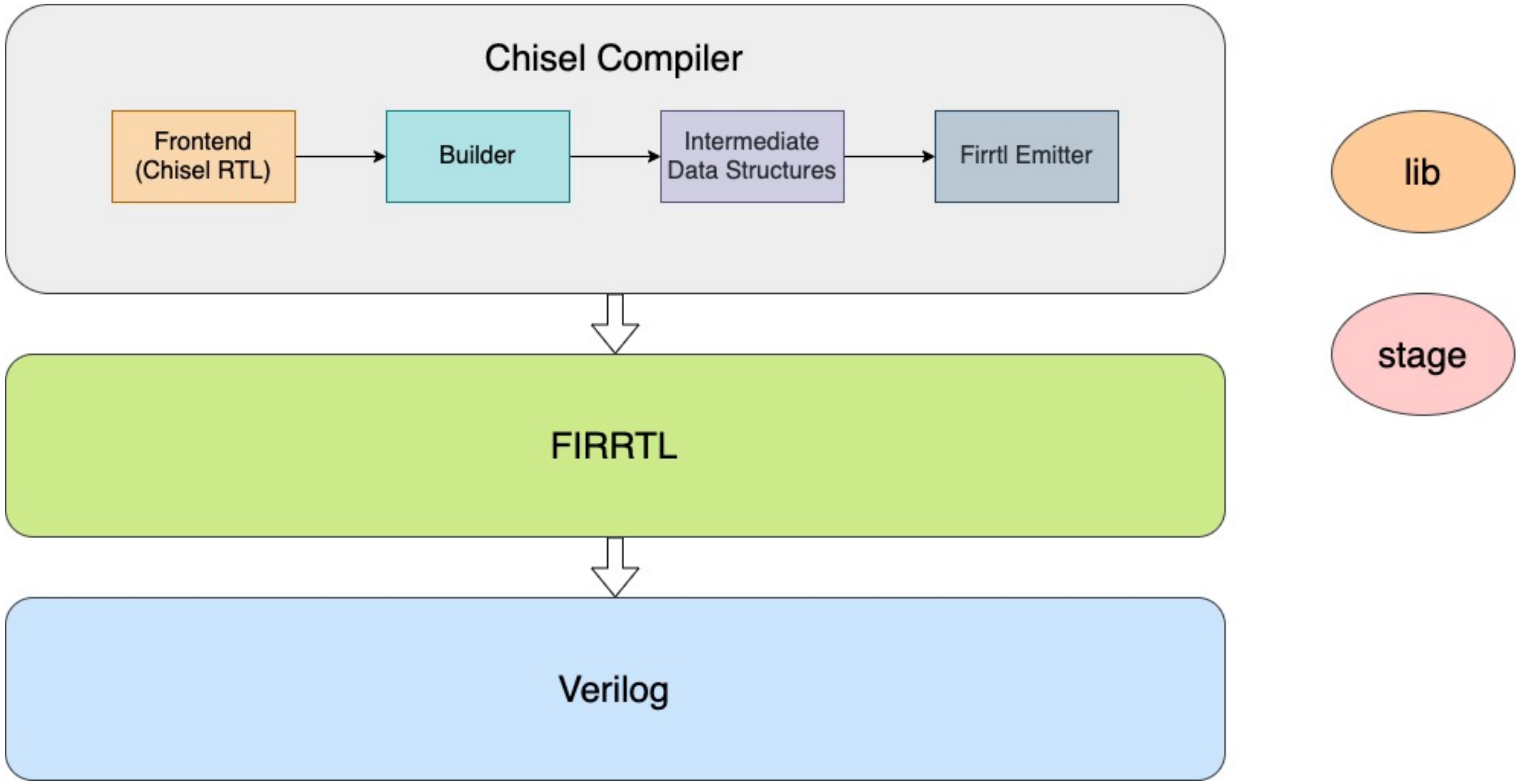
What is Chisel & Why use it?

- **C**onstructing **H**ardware **I**n a **S**cala **E**mbedded **L**anguage
- 建构在Scala语言之上的领域专用语言
- 与HDL相比
  - 效率高
  - 一致性检查
  - 参数化支持
- 与HLS相比
  - 更加可控
  - 设计思路完全不同

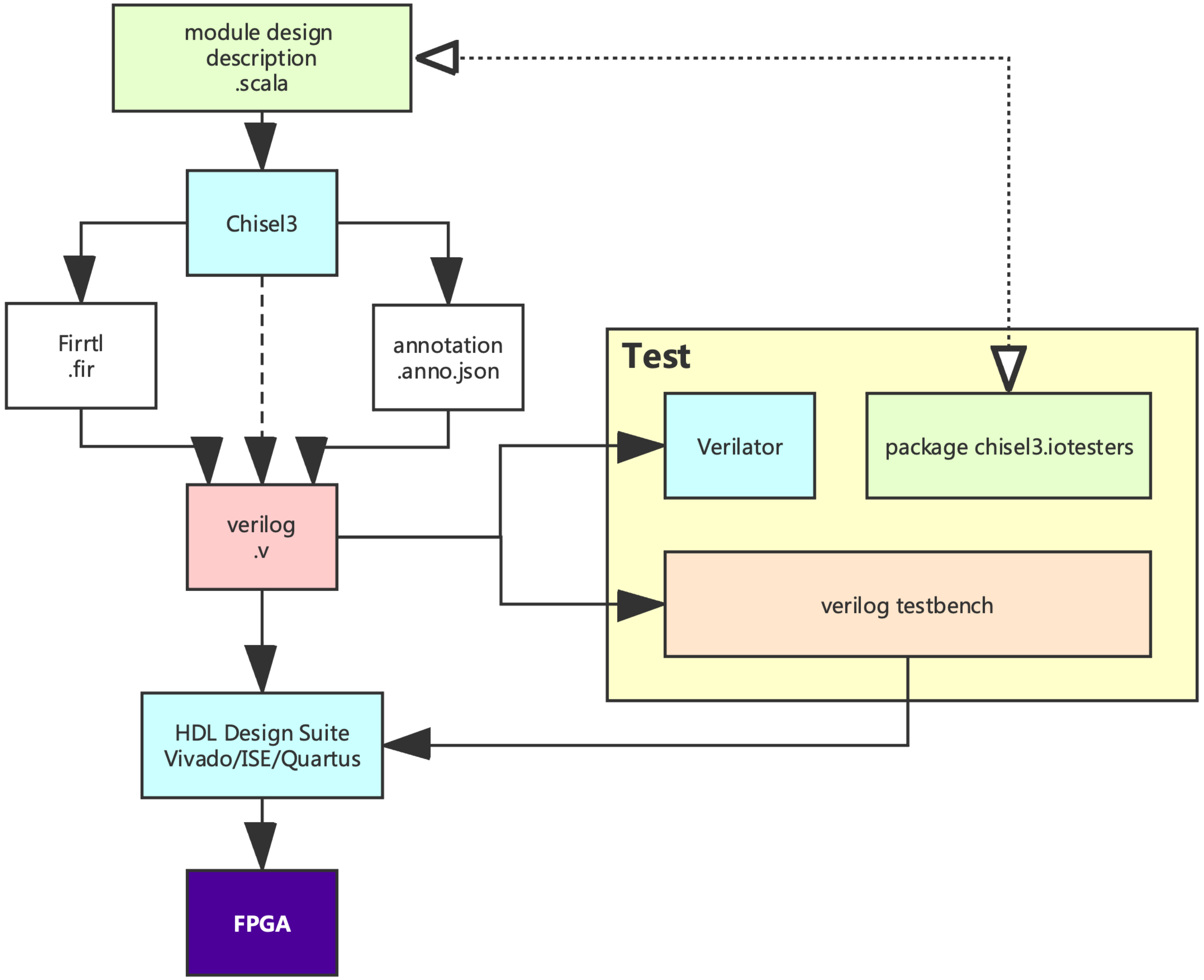
HDL vs HLS vs HCL

	HDL	HLS	HCL
对HW开发者友好度	高	低	高
对SW开发者友好度	低	高	中
设计可复用性	低	中	高
电路可控性	高	低	中
调试效率	高	低	中

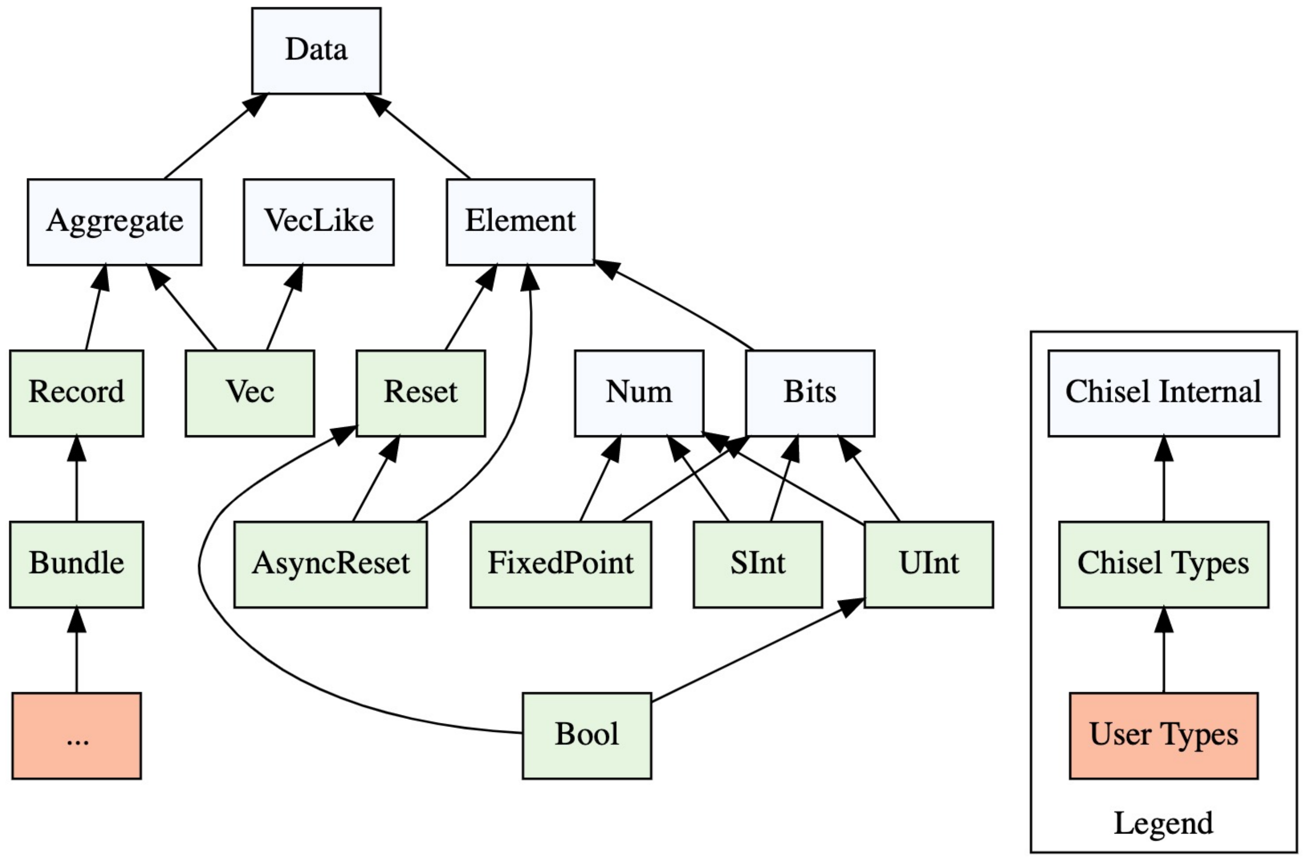
Chisel Architecture



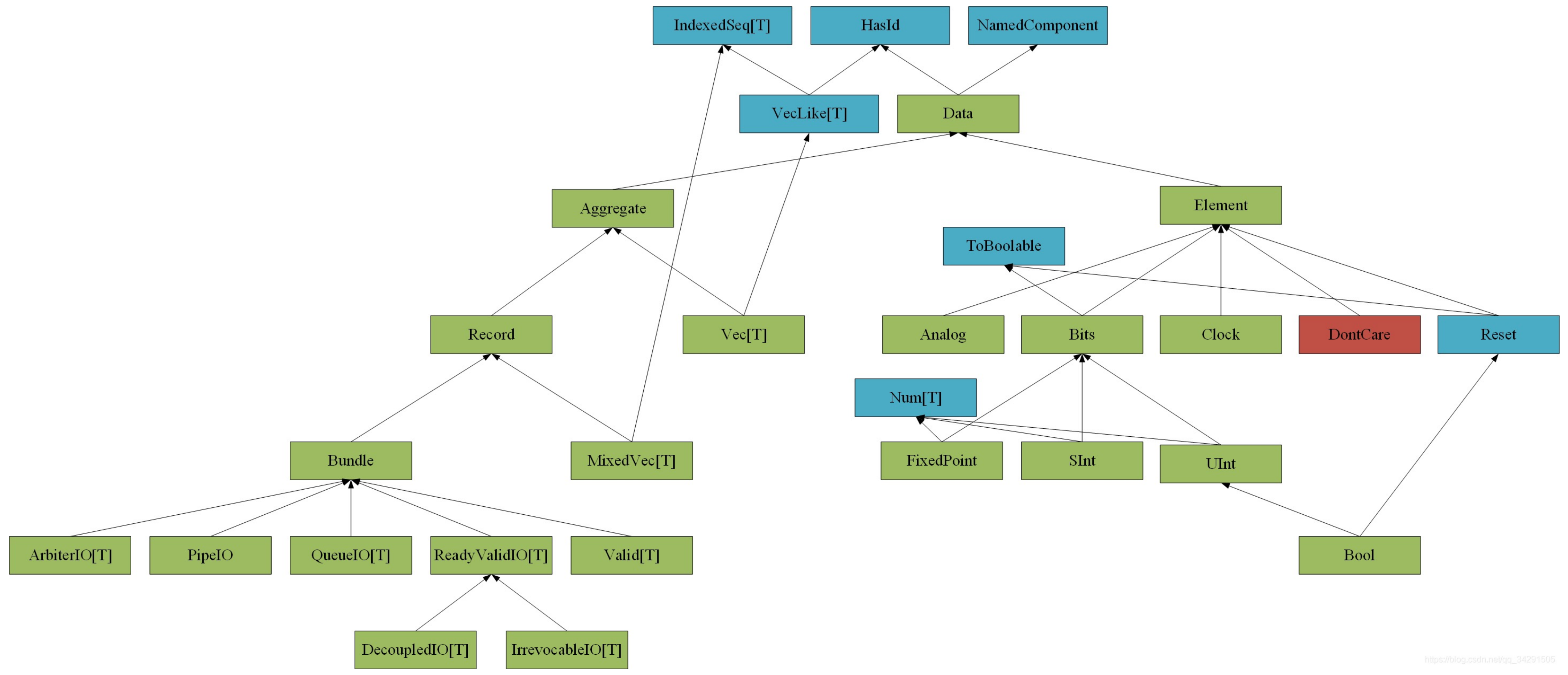
Chisel Design Flow



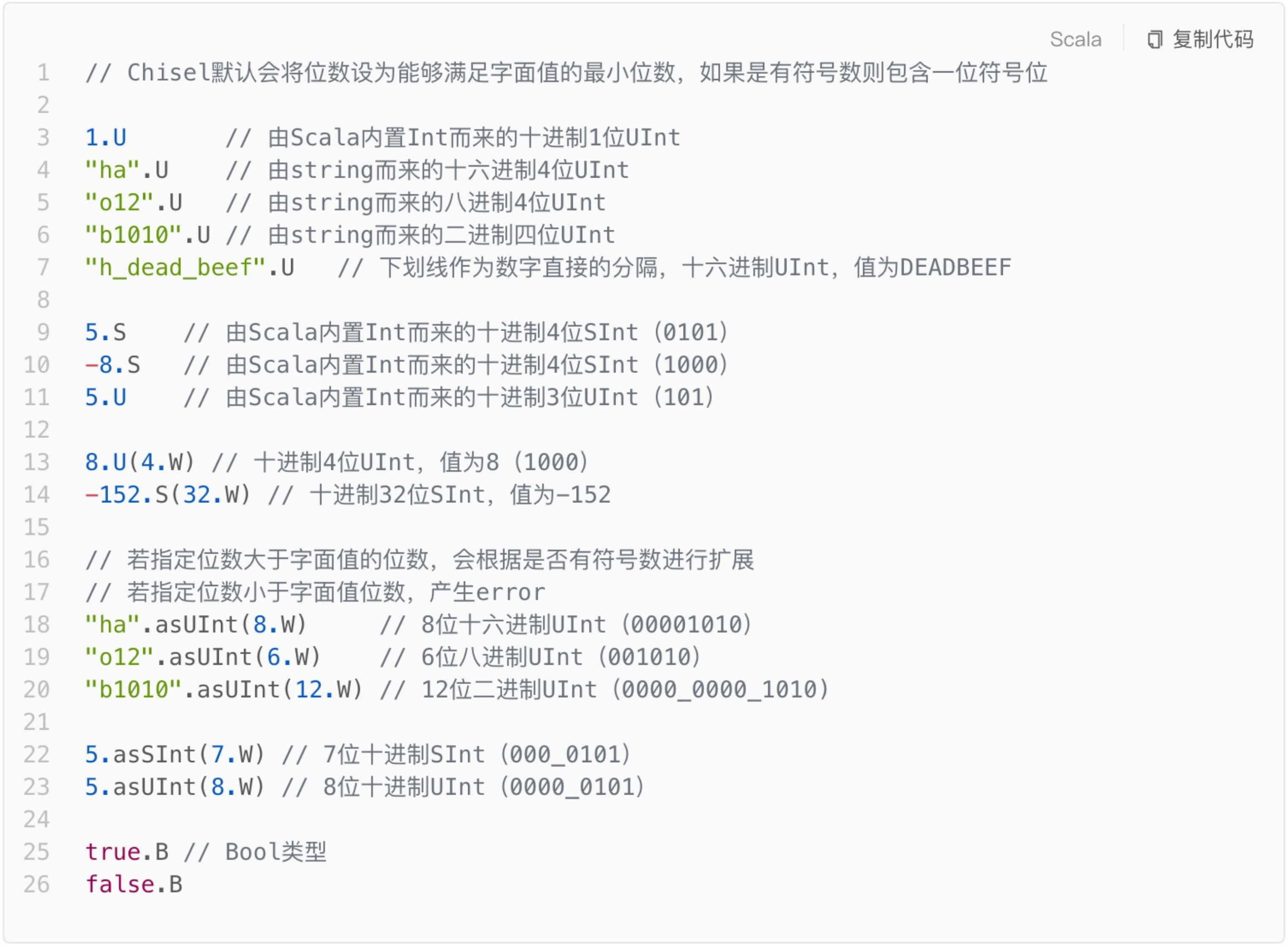
Data type in Chisel



Data type in Chisel (Cont.)



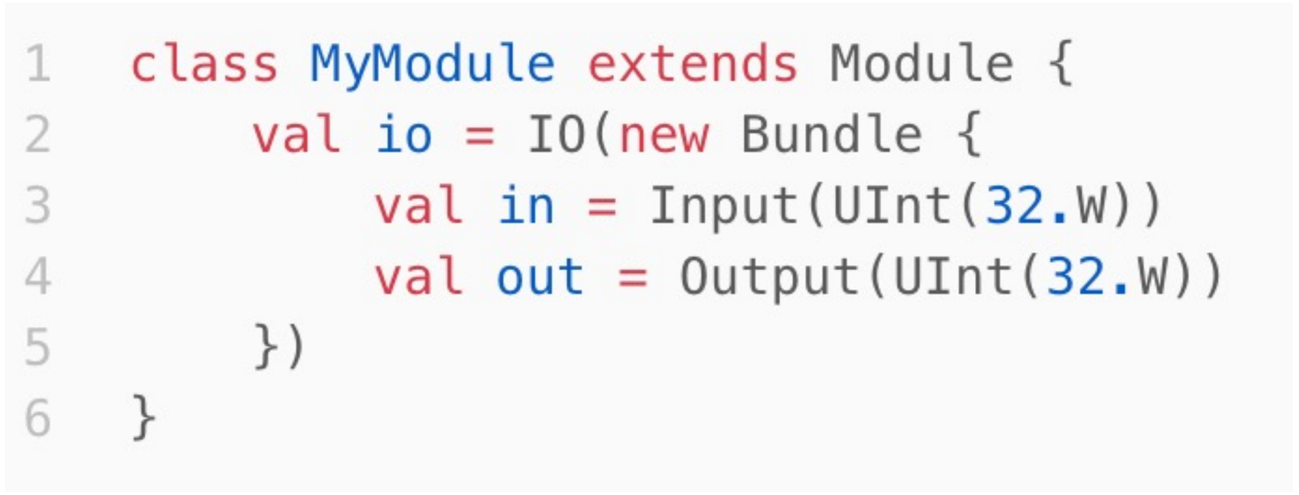
Literal Value Representation



```scala
// Chisel默认会将位数设为能够满足字面值的最小位数，如果是有符号数则包含一位符号位

1.U       // 由Scala内置Int而来的十进制1位UInt
"ha".U    // 由string而来的十六进制4位UInt
"o12".U   // 由string而来的八进制4位UInt
"b1010".U // 由string而来的二进制四位UInt
"h_dead_beef".U   // 下划线作为数字直接的分隔，十六进制UInt，值为DEADBEEF

5.S    // 由Scala内置Int而来的十进制4位SInt (0101)
-8.S   // 由Scala内置Int而来的十进制4位SInt (1000)
5.U    // 由Scala内置Int而来的十进制3位UInt (101)

8.U(4.W) // 十进制4位UInt，值为8 (1000)
-152.S(32.W) // 十进制32位SInt，值为-152

// 若指定位数大于字面值的位数，会根据是否有符号数进行扩展
// 若指定位数小于字面值位数，产生error
"ha".asUInt(8.W)     // 8位十六进制UInt (00001010)
"o12".asUInt(6.W)    // 6位八进制UInt (001010)
"b1010".asUInt(12.W) // 12位二进制UInt (0000_0000_1010)

5.asSInt(7.W) // 7位十进制SInt (000_0101)
5.asUInt(8.W) // 8位十进制UInt (0000_0101)

true.B // Bool类型
false.B
```

Bundle

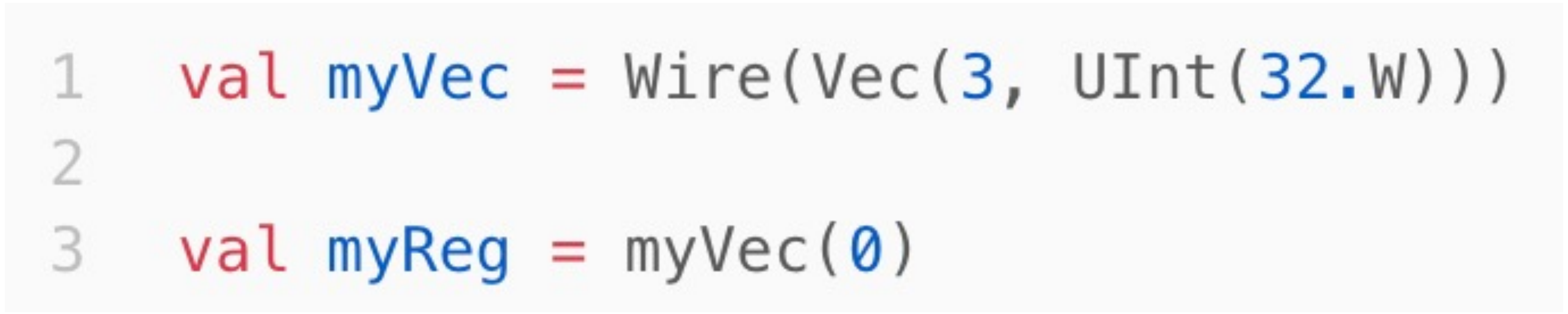
- 类似struct in C/C++
- 可以协助构建线网或寄存器
- 最常见的用途：包裹IO，构建一个模块的端口列表，或者一部分端口



```
class MyModule extends Module {
    val io = IO(new Bundle {
        val in = Input(UInt(32.W))
        val out = Output(UInt(32.W))
    })
}
```

Vec

- 构造函数：两个参数
  - Int，表示元素的个数
  - 元素数据类型
- 可索引，下标从0开始



```
val myVec = Wire(Vec(3, UInt(32.W)))

val myReg = myVec(0)
```

Hardware type in Chisel

- IO
- Module
- Reg
- Wire
- …

Flipped io & connection

- 反转IO方向
- 端口整体连接，连接同名端口
  - 同一级的端口方向：输入连输出、输出连输入
  - 父级和子级的端口方向：输入连输入、输出连输出

```
class MyIO extends Bundle {
  val in  = Input(Vec(5, UInt(32.W)))
  val out = Output(UInt(32.W))
}

......
val io = IO(new Bundle {
  val x = new MyIO
  val y = Flipped(new MyIO)
})

io.x <> io.y  // 相当于 io.y.in := io.x.in;
              //        io.x.out := io.y.out
......
```

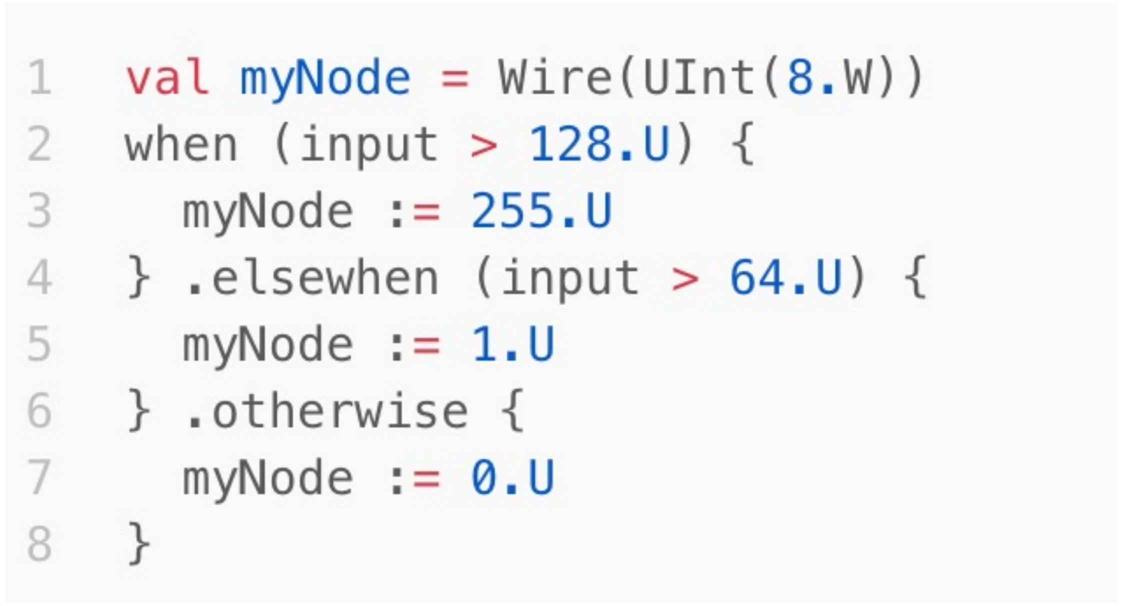
Module

- 三个特点
  - 继承自Module类
  - 需要实现一个抽象字段名为io，该字段必须引用端口对象IO
  - 在类的主构造方法里进行内部电路连线

- 隐式字段
  - clock
  - reset

Assign values & Control flows

- 使用:=进行赋值
- 控制流：when...elsewhen...otherwise...



```
val myNode = Wire(UInt(8.W))
when (input > 128.U) {
  myNode := 255.U
} .elsewhen (input > 64.U) {
  myNode := 1.U
} .otherwise {
  myNode := 0.U
}
```

What does Chisel code look like?

- Combinational

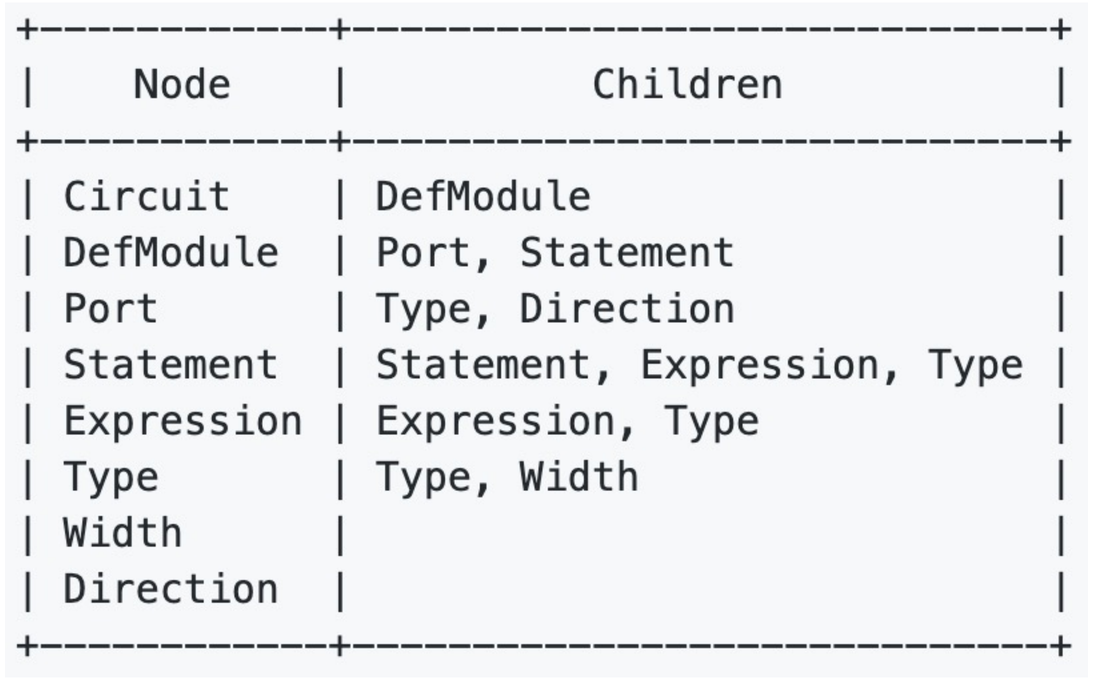
```
class Combinational extends Module {
  val io = IO(new Bundle {
    val x   = Input(UInt(16.W))
    val y   = Input(UInt(16.W))
    val z   = Output(UInt(16.W))
  })
  io.z := io.x + io.y
}
```

- Sequential

```
class ShiftRegister extends Module {
  val io = IO(new Bundle {
    val in  = Input(UInt(1.W))
    val out = Output(UInt(1.W))
  })
  val r0 = RegNext(io.in)
  val r1 = RegNext(r0)
  val r2 = RegNext(r1)
  val r3 = RegNext(r2)
  io.out := r3
}
```

FIRRTL

- **Flexible Internal Representation for RTL**
- 通用电路转换过程中使用的IR
- 数据结构：AST Nodes



```
+------------+------------------------------+
|    Node    |           Children           |
+------------+------------------------------+
| Circuit    | DefModule                    |
| DefModule  | Port, Statement              |
| Port       | Type, Direction              |
| Statement  | Statement, Expression, Type  |
| Expression | Expression, Type             |
| Type       | Type, Width                  |
| Width      |                              |
| Direction  |                              |
+------------+------------------------------+
```

An example of FIRRTL

- Chisel

```
class RegisterModule extends Module {
  val io = IO(new Bundle {
    val in  = Input(UInt(12.W))
    val out = Output(UInt(12.W))
  })

  val register = Reg(UInt(12.W))
  register := io.in + 1.U
  io.out := register
}
```

- FIRRTL

```
circuit cmd2HelperRegisterModule :
  module cmd2HelperRegisterModule :
    input clock : Clock
    input reset : UInt<1>
    output io : {flip in : UInt<12>, out : UInt<12>}

    reg register : UInt<12>, clock @[cmd2.sc 7:21]
    node _T = add(io.in, UInt<1>("h01")) @[cmd2.sc 8:21]
    node _T_1 = tail(_T, 1) @[cmd2.sc 8:21]
    register <= _T_1 @[cmd2.sc 8:12]
    io.out <= register @[cmd2.sc 9:10]
```

An example of FIRRTL (Cont.)

- Verilog

```
module cmd2HelperRegisterModule(
  input         clock,
  input         reset,
  input  [11:0] io_in,
  output [11:0] io_out
);
  reg [11:0] register; // @[cmd2.sc 7:21]
  reg [31:0] _RAND_0;
  assign io_out = register; // @[cmd2.sc 9:10]
  // ...
  // generate random reg init
  always @(posedge clock) begin
    register <= io_in + 12'h1;
  end
endmodule
```

Why does Chisel use Scala?

- is a very powerful language with features we feel are important for building circuit generators
- is specifically developed as a base for domain-specific languages
- compiles to the JVM
- has a large set of development tools and IDEs
- has a fairly large and growing user community

Chisel: Constructing Hardware in a Scala Embedded Language, DAC 2012

Things Chisel cannot do

- X or Z
- High readability generated code
- Level trigger
- Inline Verilog
- Inout

Resources

- https://www.chisel-lang.org/
- https://www.yuque.com/wuangmoyao/keh90g
- https://github.com/chipsalliance/chisel3
- https://github.com/chipsalliance/firrtl
- https://stackoverflow.com/questions/53007782/what-benefits-does-chisel-offer-over-classic-hardware-description-languages

An ad

- Chisel Community Conference 2021, Shanghai, China. 6/26/2021(CST)
- Join through Zoom
- Info: https://docs.google.com/document/d/1BLP2DYt59DqI-FgFCcjw8Ddl4K-WU0nHmQu0sZ_wAGo/edit
- Schedule: https://docs.google.com/spreadsheets/d/1Gb4mMGRhs9exJW-l3NOS0IFi7fDplNQQnkXyAKzHlTg/edit

Thank you