Chisel 使用经验

Singularity K Chen

Nanyang Technological University

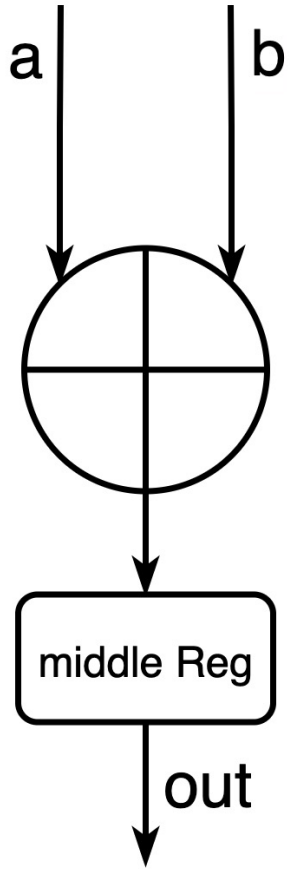
TOC

- Chisel 开发环境
- 用 Chisel 写个简单的模块：Adder
- 创建这个模块的端口类
- 创建这个模块的伴生对象
- 创建这个模块的 Chisel-Testers 2 TB
- 继承这个模块的端口类
- 高阶实例：Adder Tree (map, foreach, case, fork join)
- 优化命令 (suggestName, dontTouch, dontCare)
- 几个个人习惯

Chisel 开发环境

- 使用 GitHub repo 模版 (mill)
  - https://github.com/sequencer/chipsalliance-playground 大杂烩，需要自己删去一部分不需要submodule
  - https://github.com/SingularityKChen/chipsalliance-playground 没有冗余submodule
- 使用 EasySoC Chisel插件 (sbt)
  - https://plugins.jetbrains.com/plugin/14269-easysoc-chisel

用 Chisel 写个简单的模块——Adder




```scala
import chisel3._

class Adder extends Module {
  val io = IO(new Bundle {
    val a: UInt = Input(UInt(32.W))
    val b: UInt = Input(UInt(32.W))
    val out: UInt = Output(UInt(32.W))
  })
  protected val middleReg: UInt = RegInit(0.U(32.W))
  middleReg := io.a + io.b
  io.out := middleReg
}
```

创建这个模块的端口类

- 为了好看（为了IDEA不报warning），我们把这几个端口单独创建一个类
- 当然，这不仅仅是为了好看
  - 同一协议模块多处引用，减少代码量、方便维护
  - 继承、隐参传递
  - ……

```
1 import chisel3._
2
3 class AdderIOs extends Bundle {
4   val a: UInt = Input(UInt(32.W))
5   val b: UInt = Input(UInt(32.W))
6   val out: UInt = Output(UInt(32.W))
7 }
```

创建这个模块的伴生对象

- 让我们在例化模块的时候就让他们自己连上叭
  - 类型检查，在写代码的时候就初步检查端口连接

```
1 import chisel3._
2
3 object Adder {
4   def apply(inPorts: AdderIOs): Adder = {
5     val adder = Module(new Adder)
6     adder.io.a := inPorts.a
7     adder.io.b := inPorts.b
8     adder
9   }
10
11  def apply(a: UInt, b: UInt): Adder = {
12    val adder = Module(new Adder)
13    adder.io.a := a
14    adder.io.b := b
15    adder
16  }
17 }
```

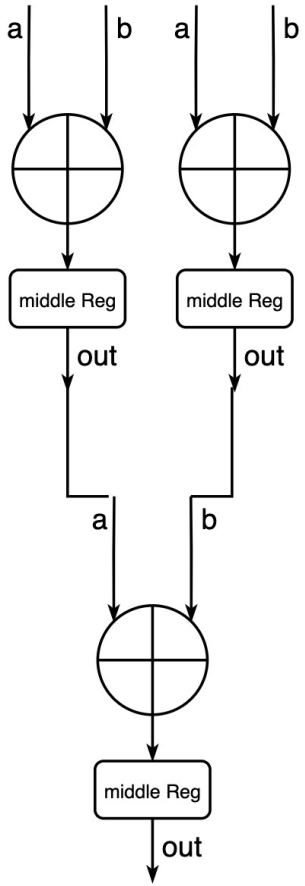
创建这个模块的 Chisel-Testers 2 TB

- The core primitives are similar to non-synthesizable Verilog: input pin assignment (**poke**), pin value assertion (**expect**), and time advance (**step**). Threading concurrency is also supported with the use of **fork** and **join**, and concurrent accesses to wires are checked to prevent race conditions.

```
1 import chisel3._
2 import chiseltest._
3 import org.scalatest.flatspec.AnyFlatSpec
4 import org.scalatest.matchers.should.Matchers
5
6 import scala.math.pow
7 import scala.util.Random
8
9 class AdderTest extends AnyFlatSpec
10       with Matchers with ChiselScalatestTester {
11 …
12 }
```

高阶实例：Adder Tree

- RegInit
- Map, Foreach, case
- Fork join



优化命令

- suggestName: 给例化后的模块里信号加个prefix
  - 也可以尝试使用chisel3-plugin
- desiredName: 给本 module 命名
- dontTouch: 不相信工具的优化
- dontCare: 想不连接就不连接

个人习惯一：protected val

- 在声明内部硬件的时候（reg/wire/module 等等）我习惯用 protected 去修饰
  - public: 在任何地方都可以被访问
  - protected: 创建类对象后不能访问到内部变量，继承后可以访问
  - private: 只有在当前类下可以访问
- 在硬件里，内部的 reg/wire 等等不能被外部访问：protected
  - 使用 private 不利于继承，且无法把变量名保留在生成出的 Verilog 中

个人习惯二：经常 refactor

- 整理代码结构
- 合并重复代码
- …

- 经常添加注释
- Scaladoc

更多技巧……

- Trait
- Optional 端口
- 握手端口及其专有测试
- 多态和参数化类/函数/端口/信号等
- 多时钟域
- 黑盒 BlackBox
- TB 的多进程 fork join
- TB 的时间域 timescope
- 使用 optional 端口 debug