Noob Chisel Exp++

Wang Rui, YSYX-23060042

Group Meeting on Sun, 19 May 2024

Intro

前一段时间入门了 Chisel。如果说用一个词来形容这门语言对初学者的感受，那应该是**神秘感。** 如果用一个词来形容初学者学 Chisel 的心情，那就是 **一筹莫展。**

> 直接用 不要看 我已经看不懂了
>
> —— *节选自《微信 · Chisel 交流群》【现代】刘玖阳*

Part I: Way to Chisel

为什么学习 Chisel 让人感觉很难?

1. API 变动太快，害怕学习旧教程会落伍
2. Chisel 许多文档的缺失
3. 系统学习 Scala 与 Chisel 语法开销很大

学习 Chisel 的秘诀

1. 只用最新的资料
2. 先上手做，再去理解

搭建框架：使用 Mill

- chipalliance/chisel-template

不要太过于纠结 `build.sc` 写法。先复用框架的代码，之后可以一边做一边研究。

初学 Chisel 语法

不推荐：

- Chisel Bootcamp 教程
- GitHub: ucb-bar/chisel-tutorial

推荐：

- Martin Schoeberl, 2023, Digital Design with Chisel, 5th Edition
- GitHub: schoeberl/chisel-lab
- GitHub: schoeberl/chisel-examples
- Official Style Guide
- Official Documentation
- Chisel Source Code & GitHub ISSUE

更推荐的上手方式

1. 参考 chisel-examples 仓库与 chisel-lab 前两个实验，完成预学习讲义中的流水灯实验；
2. 进一步的，使用 Chisel 重写一遍预学习中要求的数字电路实验。

在粗略看完书的基础章节后，对着需求翻书，而不是没目的地读书。

后续学习方式

不要恐惧 Chisel 源代码，也不必强迫自己完全读懂每一行源代码

Tips 1：实用工具

LSP: Metals

JVM: Adoptium Eclipse Temurin, GraalVM

Formatter: Scalafmt

Tips 2: 向 Chisel 传 Flag

Chisel 代码不是在描述电路，而是在描述一个 **电路生成器软件**

Tips 3: 多交流

- Chisel 交流群
- 一生一芯小组
- 线下基地交流

Part II: Chisel Decoder

Lookup Table in SystemVerilog

```
parameter bit REGEN_TRUE = 1'b1;
parameter bit REGEN_FALSE = 1'b0;
parameter bit PCJEN_TRUE = 1'b1;
parameter bit PCJEN_FALSE = 1'b0;
parameter bit [1:0] MWEN_BYTE = 2'b01;
parameter bit PCREN_TRUE = 1'b1;
parameter bit [1:0] MREN_WORD = 2'b11;
parameter bit [2:0] ALUOP_ADD_BEQ = 3'b000;
parameter bit UNSIGN_ARITH_TRUE = 1'b1;
parameter bit [2:0] IMM_TYPE_UJ = 3'b111;

// Micro command format: [13]REGEN [12]PCJEN [11]PCREN [10:9]MWEN [8:7]MREN [6:4]ALUOP [3]UNSIGN [2:0]IMM_TYPE
localparam bit [PATTERN_LEN-1:0] LUIPattern = {7'b0000000, 3'b000, 5'b01101};
localparam bit [MICRO_LEN-1:0] LUIMicro = {
  REGEN_TRUE, PCJEN_FALSE, PCREN_FALSE, MWEN_NONE, MREN_NONE, ALUOP_ADD_BEQ, UNSIGN_LOGIC_FALSE, IMM_TYPE_U
};
localparam bit [PATTERN_LEN-1:0] AUIPCPattern = {7'b0000000, 3'b000, 5'b00101};
localparam bit [MICRO_LEN-1:0] AUIPCMicro = {
  REGEN_TRUE, PCJEN_FALSE, PCREN_TRUE, MWEN_NONE, MREN_NONE, ALUOP_ADD_BEQ, UNSIGN_LOGIC_FALSE, IMM_TYPE_U
};
```

Lookup Table in SystemVerilog

Cons:

- 逐行匹配遍历查找，性能较差；
- SV 匹配要使用 Mask，`?` 是不可综合语法；
- 在增加指令或者要重构生成的信号的时候难以维护。

Basic Decoder: Truth Table

```
class SimpleDecoder extends Module {
  val table = TruthTable(
    Map(
      BitPat("b001") -> BitPat("b?"),
      BitPat("b010") -> BitPat("b?"),
      BitPat("b100") -> BitPat("b1"),
      BitPat("b101") -> BitPat("b1"),
      BitPat("b111") -> BitPat("b1")
    ),
    BitPat("b0"))
  val input = IO(Input(UInt(3.W)))
  val output = IO(Output(UInt(1.W)))
  output := decoder(input, table)
}
```

Basic Decoder: Truth Table

Pros:

- 使用 `espresso` 进行优化，减少 Verilog 电路的复杂度；
- 可以使用强大的 `BitPat` 进行匹配，且生成可综合语法。
- 可以使用一些 Scala 代码辅助 Truth Table 构建。

Cons:

- 依然难以维护，无法输入结构化信息。

Decode Pattern

```
case class InstructionPattern(
    val func7: BitPat = BitPat.dontCare(7),
    val func3: BitPat = BitPat.dontCare(3),
    val opcode: BitPat
) extends DecodePattern {
  def bitPat: BitPat = pattern

  val genPattern =
    func7 ## BitPat.dontCare(10) ## func3 ## BitPat.dontCare(5) ## opcode

}
```

Decode Pattern

结构化传参，生成 Inputs Pattern

Decode Field

```
object MemLenField extends DecodeField[InstructionPattern, UInt] {
  def name: String = "memoryLenth"
  def chiselType = UInt(MemLen.getWidth.W)
  def genTable(op: InstructionPattern): BitPat = {
    op.func3.rawString match {
      // Maybe we should just use 2 bits for this field? Whatever, espresso will optimize it.
      case "000" => BitPat(MemLen.B.litValue.U(MemLen.getWidth.W))
      case "001" => BitPat(MemLen.H.litValue.U(MemLen.getWidth.W))
      case "010" => BitPat(MemLen.W.litValue.U(MemLen.getWidth.W))
      case "100" => BitPat(MemLen.B.litValue.U(MemLen.getWidth.W)) // LBU
      case "101" => BitPat(MemLen.H.litValue.U(MemLen.getWidth.W)) // LHU
      case _     => BitPat(MemLen.B.litValue.U(MemLen.getWidth.W))
    }
  }
}
```

Decode Field

结构化读取分析输入，有逻辑地生成 Outputs 片段

Decode Field

```
object BreakField extends BoolDecodeField[InstructionPattern] {
  def name: String = "break"

  // Only EBREAK has a break signal
  def genTable(op: InstructionPattern): BitPat = {
    if (
      op.pattern == BitPat.N(11) ## BitPat
        .Y(1) ## BitPat.N(13) ## BitPat("b1110011")
    ) BitPat(true.B)
    else BitPat(false.B)
  }
}
```

Decode Table

- 通过 `DecodePattern` 实例化所有输入，并做成 `Seq`
- 把所有 `Field` 对象做成 `Seq`
- 把两个序列作为 `DecodeTable` 的参数

Decode Table

```
  val possiblePatterns = Seq(
    InstructionPattern(
      opcode = BitPat("b0110111")
    ), // LUI

    InstructionPattern(
      opcode = BitPat("b0010111")
    ), // AUIPC
    ...
  )
```

Decode Table

```
  val allFields = Seq(
    ...
    Data1Field,
    ...
    BreakField,
    ...
  )
```

Decode Table

```
val decodeTable = new DecodeTable(possiblePatterns, allFields)
val decodeResult = decodeTable.decode(io.fromIFU.bits.inst)
```

Decode Result

```
io.toEXU.bits.instructionType := decodeResult(InstTypeField)
io.toEXU.bits.data1Type := decodeResult(Data1Field)
io.toEXU.bits.data2Type := decodeResult(Data2Field)
...
```

Chisel Decoder

Pros:

- 继承 Truth Table 的优点，非常灵活地使用，可维护性强

Cons:

- 文档不详细，要去代码里看 API。有一定学习成本。

Tips 1: 搭配 Chisel Enum 使用

chipalliance/chisel · Issue # 1871

```
object MemLenField extends DecodeField[InstructionPattern, UInt] {
  def name: String = "memoryLenth"
  def chiselType = UInt(MemLen.getWidth.W)
  def genTable(op: InstructionPattern): BitPat = {
    op.func3.rawString match {
      // Maybe we should just use 2 bits for this field? Whatever, espresso will optimize it.
      case "000" => BitPat(MemLen.B.litValue.U(MemLen.getWidth.W))
      case "001" => BitPat(MemLen.H.litValue.U(MemLen.getWidth.W))
      case "010" => BitPat(MemLen.W.litValue.U(MemLen.getWidth.W))
      case "100" => BitPat(MemLen.B.litValue.U(MemLen.getWidth.W)) // LBU
      case "101" => BitPat(MemLen.H.litValue.U(MemLen.getWidth.W)) // LHU
      case _     => BitPat(MemLen.B.litValue.U(MemLen.getWidth.W))
    }
  }
}
```

Tips 2: BitPat as Output

为什么在 Outputs 中还会有 `?` 通配——电路设计 Otherwise 和 Default 的秘密

Tips 2: BitPat as Output

```
object DecodeSupportField extends DecodeField[InstructionPattern, Bool] {
  def name: String = "decodeSupport"
  def chiselType = Bool()
  def genTable(op: InstructionPattern): BitPat = BitPat.Y(1)
  override def default: BitPat = BitPat.N(1)
}

...

  val decodeSupport = Wire(Bool())
  decodeSupport := decodeResult(DecodeSupportField)
  decodeSuuport := true.B // Or we can ...?
  dontTouch(decodeSupport)
```

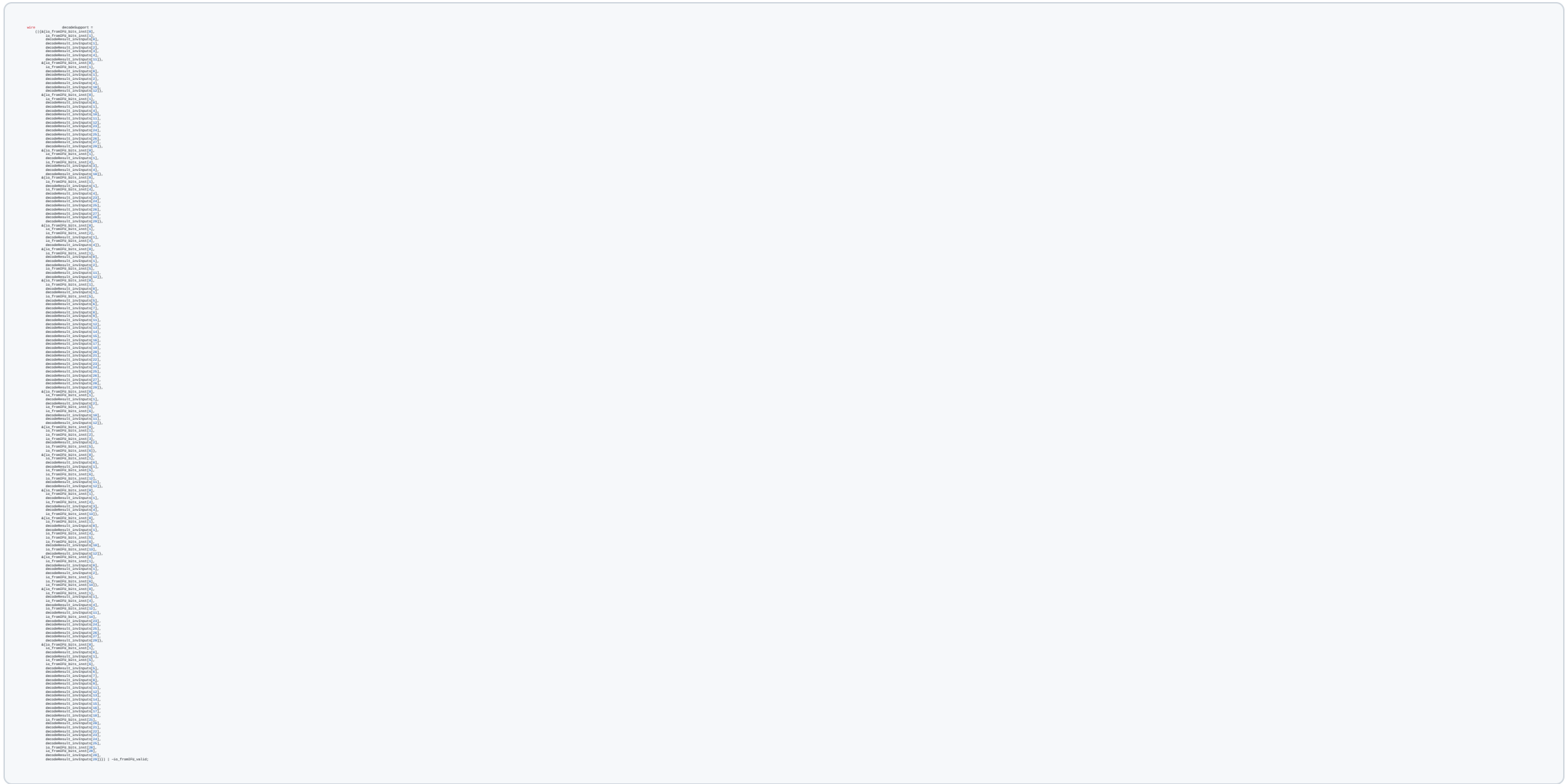
Tips 2: BitPat as Output

`decodeSupport := true.B`

```
wire decodeSupport = 1'h1;      // src/main/scala/IDU/IDU.scala:96:27
```

Tips 2: BitPat as Output

```
decodeSupport := decodeResult(DecodeSupportField)
```



Tips 2: BitPat as Output

在 Output 中 `Dontcare` 的意义，就是给 `espresso` 优化的空间；在电路中定义 Default 行为，开销往往是巨大的。

> 启示：拒绝电路行为编程，心中还是要有电路

关于 Chisel Decoder 使用的其他问题，欢迎来《微信 · Chisel 交流群》和大家沟通。

Group Members(374)

sequencer

Seque...

Group Members(374)

decoder

decoder

谢谢大家！