Accelerating Systems with Programmable Logic Comp.

Lecture 8.5

The Hardware Description Language Chisel

April 27, 2020

Philipp Rümmer
Uppsala University
Philipp.Ruemmer@it.uu.se

Chisel/FIRRTL Hardware Compiler Framework

View on GitHub

Upcoming Events

Chisel/FIRRTL development meetings happen every Monday from 1100–1300 PT.

Call-in info and meeting notes are available here.

Outline

- High-level overview of Chisel
- Several examples

Chisel High-Level

- Chisel is a HDL embedded in Scala
- Tailored to large, complex hardware designs
- Development of Chisel started in 2012
- Used in projects including RISC-V
- Chisel can be compiled to various other languages:
    - FIRRTL, **Verilog**, VHDL, etc.

What is a DSL?

- Any language tailor-made for some domain, e.g.:
  - Query languages: SQL, XPath, ...
  - Markup/down: HTML, CSS, LaTeX, ...
  - Modeling: UML, SysML, AADL, ...
- Opposite:
  Generic languages: C, C++, Verilog, ...
- [Fowler, 2010]

What is a DSEL/eDSL?

- A DSL defined within some **host language**
  - Host language is usually a functional language: Haskell, ML, Scala, ...
- DSL takes the form of a library
- Easy to implement, extensible:
  - No parser needed
  - Features of host language are inherited

What is Scala?

- Modern multi-paradigm programming language
- Imperative features:
  Very similar to Java
- Functional features:
  ADTs, pattern matching, higher-order functions, closures, polymorphism
- Compiling to Java bytecode

https://www.scala-lang.org/

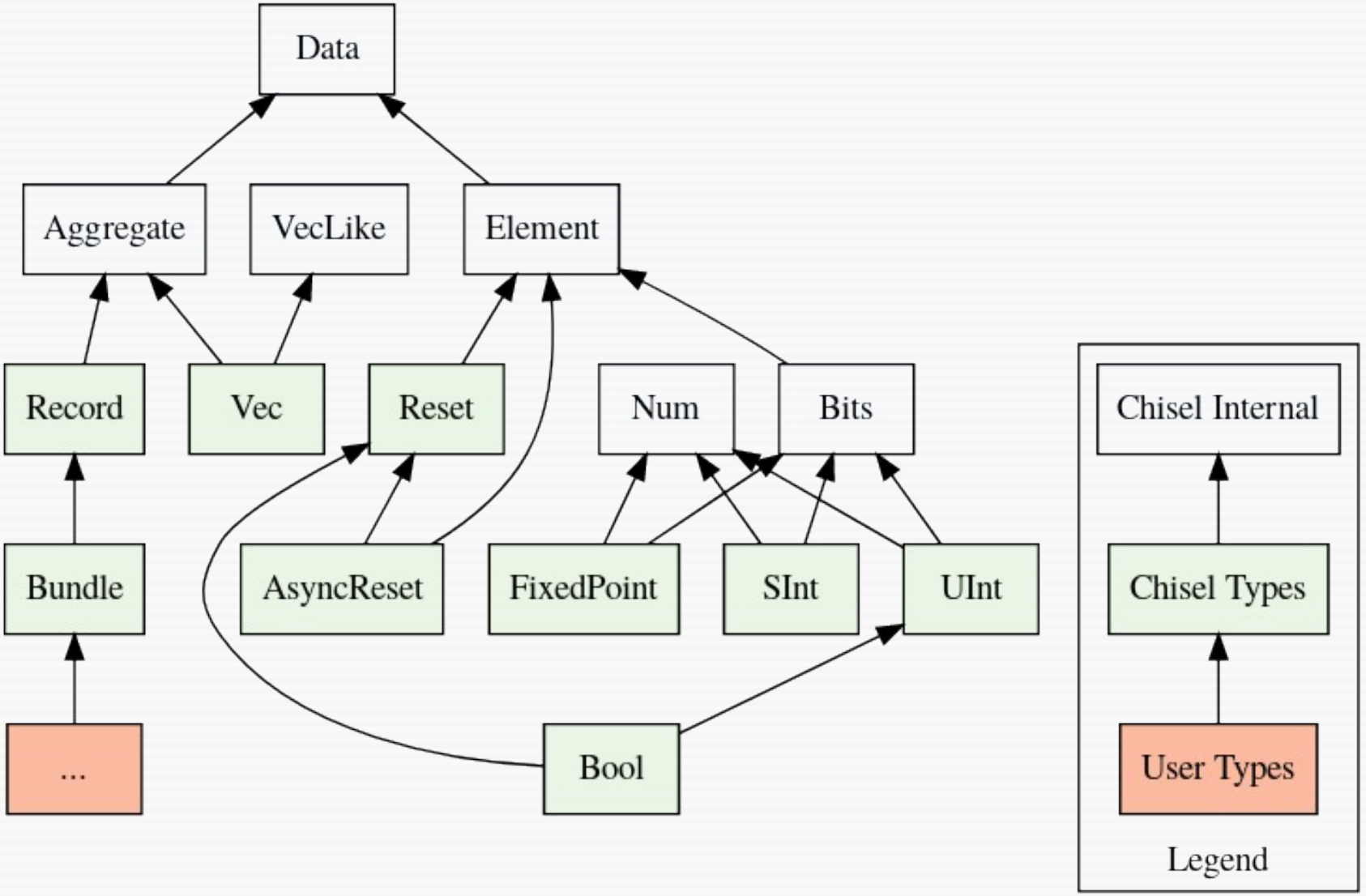
Introductory Example

- Multiplexer with 2 and 4 inputs

What have we seen?

- Modules, more or less like in Verilog
- Specification of module interfaces
- Instantiation of sub-modules
- Connection of wires

Data-types in Chisel



https://www.chisel-lang.org/api/latest/index.html

Data-types in Chisel (2)

- Data-types are Scala classes
- Each data-type comes with a set of operations (= methods)
- Bit-widths are often inferred automatically
- New types can be introduced
    - Similar to records/structs

Data-types in Chisel (3)

- Data can be marked as `Input` and `Output`


```
val muxIO = new Bundle {
    val sel = Input(UInt(1.W))
    val in0 = Input(UInt(1.W))
    val in1 = Input(UInt(1.W))
    val out = Output(UInt(1.W))
}
```

Modules

- Correspond closely to Verilog modules
- Each module comes with:
  - An interface, field `val io = IO(...)`
  - Instantiation of other modules
  - Circuit definitions
  - Methods, fields, etc., to structure the module definition
- Operator to connect wires: <>

Stateful Modules

- Different classes for registers:
  - Reg — *Standard register*
  - RegInit — *Register with reset value*
  - RegNext — *D-flipflop, 1-cycle delay*
  - RegEnable — *R. with update-enable*
  - ShiftRegister — *n-cycle delay*
  - …
- Memory: Mem

Registers (2)

- Updating registers: `:=`
- Conditional updates:

```
when (<condition 1>) {<register update 1>}
    .elsewhen (<condition 2>) {<register update 2>}
    ...
    .elsewhen (<condition N>) {<register update N>}
    ...
    .otherwise {<default updates>}
```

- Similar to *non-blocking* assig. in Verilog

Example with Registers

- Vending machine

Utilities

- Enum: create named constants

```
val state_on :: state_off :: Nil = Enum(2)
```

- Mux: if-then-else expression

```
x := Mux(<cond>, <then>, <else>)
```

- MuxCase: build a case expression

```
state := MuxCase(default, Array(c1 -> a, c2 -> b))
```

Utilities (2)

- switch: another kind of case expression

```
switch (myState) {
  is (state1) {
    // some logic that runs when myState === state1
  }
  is (state2) {
    // some logic that runs when myState === state2
  }
}
```

Clock and Resets

- Clocks are usually implicit in the Chisel design
    - But can be made explicit; e.g., for multiple clock domains
- Updates happen at positive clock edges
- Resets are usually also handled implicitly (e.g., using RegInit)
    - But can be made explicit as well

Generating Verilog from Chisel

- In the tutorial: add option

```
--backend-name verilator
```

- (and install the Verilator RTL simulator)

Advanced Modeling

- Chisel can be used like Verilog, to directly define some design
- But we can also see it as a **design generator:**
  *W*rite a (Scala/Chisel) program that will produce the design
    - “Meta-programming”
    - This is what makes Chisel quite powerful

Example: Mixing Chisel and Scala

- Computing maximum of a vector
- Using higher-order functions and polymorphism for abstraction

Further Reading

- Chisel tutorials:
  - https://mybinder.org/v2/gh/freechipsproject/chisel-bootcamp/master
  - https://github.com/ucb-bar/chisel-tutorial
- Wiki/User guide:
  - https://github.com/ucb-bar/chisel-tutorial/wiki/
- Paper introducing Chisel at DAC 2012:
  - https://dl.acm.org/doi/10.1145/2228360.2228584

Conclusions

- Many different ways of designing circuits:
  - Verilog
  - Chisel
  - HLS
  - *to be continued*
- **Which one is best? It depends ...**

Thank you!