

Chisel: Software Defined Hardware

Digital Design in the 21st Century

Martin Schoeberl

<mailto:masca@dtu.dk>

Technical University of Denmark

March 2, 2021

Motivating Example:

Lipsi: Probably the Smallest Processor in the World

- ▶ Tiny processor
- ▶ Simple instruction set
- ▶ Shall be small
 - ▶ Around 200 logic cells, one FPGA memory block
- ▶ Hardware described in Chisel
- ▶ Available at <https://github.com/schoeberl/lipsi>
- ▶ Usage
 - ▶ Utility processor for small stuff
 - ▶ In teaching for introduction to computer architecture
- ▶ The design took place on the island Lipsi

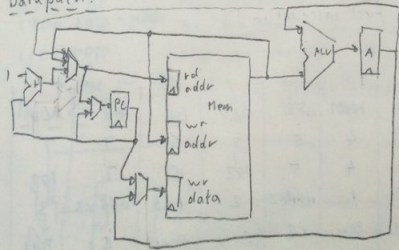
The Design of Lipsi on Lipsi

Lipsi: a Minimalistic Microcontroller

Levos, Lipsi
16.7.2010 81

- Single on-chip memory \Rightarrow 2 cycles/instruction
- ≈ 200 Ld! (ld reg indirect is 3 cycles)
- 8 bit datapath, 8 bit variable length instructions
- Accu + 8 (16) register in memory
- 256 byte instructions, 256 byte data

Datapath:



Lipsi Implementation

- ▶ Hardware described in Chisel
- ▶ Tester in Chisel
- ▶ Assembler in Scala
 - ▶ Core case statement about 20 lines
- ▶ Reference design of Lipsi as software simulator in Scala
- ▶ Testing:
 - ▶ Self testing assembler programs
 - ▶ Comparing hardware with a software simulator
- ▶ All in a single programming language!
- ▶ All in a single program
- ▶ How much work is this?

Chisel is Productive

- ▶ All coded and tested in less than 14 hours!
- ▶ The hardware in Chisel
- ▶ Assembler in Scala
- ▶ Some assembler programs (blinking LED)
- ▶ Simulation in Scala
- ▶ Two testers
- ▶ BUT, this does not include the design (done on paper)

Motivating Example: Lipsi, a Tiny Processor

- ▶ Show in IntelliJ

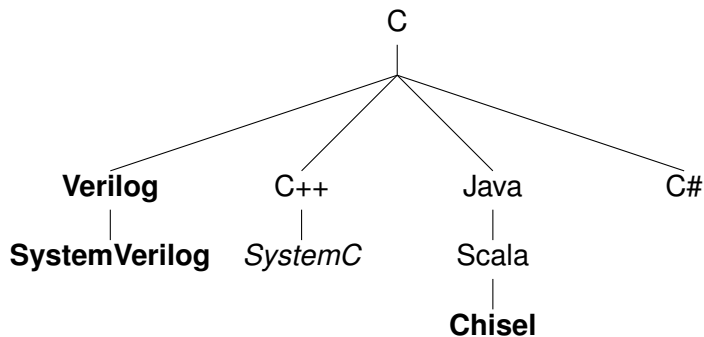
More on Chisel Success Stories

- ▶ CCC 2020 (in silicon valley)
- ▶ 90 participants
- ▶ More than 30 different (hardware) companies present
- ▶ Several companies are looking into Chisel
- ▶ IBM did an open-source PowerPC
- ▶ [SiFive](#) is a RISC-V startup success
 - ▶ High productivity with Chisel
 - ▶ Open-source Rocket chip
- ▶ Esperanto uses the BOOM processor in Chisel
- ▶ Google did a machine learning processor
- ▶ Intel is looking at Chisel
- ▶ Chisel is open-source, if there is a bug you can fix it
 - ▶ You can contribute to the Chisel ecosystem

Chisel

- ▶ A hardware *construction* language
 - ▶ Constructing Hardware In a Scala Embedded Language
 - ▶ If it compiles, it is synthesizable hardware
 - ▶ Say goodbye to your unintended latches
- ▶ Chisel is not a high-level synthesis language
- ▶ Single source two targets
 - ▶ Cycle accurate simulation (testing)
 - ▶ Verilog for synthesis
- ▶ Embedded in Scala
 - ▶ Full power of Scala available
 - ▶ But to start with, no Scala knowledge needed
- ▶ Developed at UC Berkeley

The C Language Family



Other Language Families

Algol
|
Ada
|
VHDL

Python
|
MyHDL

Some Notes on Scala

- ▶ Object oriented
- ▶ Functional
- ▶ Strongly typed
 - ▶ With very good type inference
- ▶ Could be seen as Java++
- ▶ Compiled to the JVM
- ▶ Good Java interoperability
 - ▶ Many libraries available

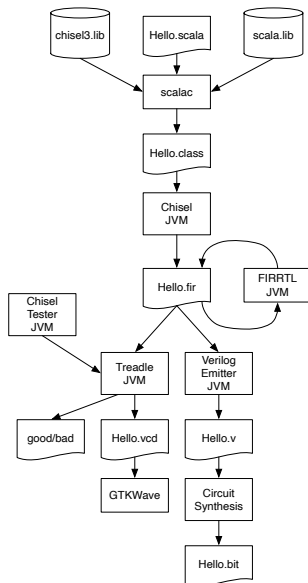
Chisel vs. Scala

- ▶ A Chisel hardware description is a Scala program
- ▶ Chisel is a Scala library
- ▶ When the program is executed it generates hardware
- ▶ Chisel is a so-called *embedded domain-specific language*

A Small Language

- ▶ Chisel is a *small* language
- ▶ On purpose
- ▶ Not many constructs to remember
- ▶ The [Chisel Cheatsheet](#) fits on two pages
- ▶ The power comes with Scala for circuit generators
- ▶ With Scala, Chisel can grow with you

Tool Flow for Chisel



Expressions are Combinational Circuits

$(a \mid b) \ \& \ \sim(c \wedge d)$

`val addVal = a + b`

`val orVal = a | b`

`val boolVal = a >= b`

- ▶ The usual operations
- ▶ Simple name assignment with `val`
- ▶ Width inference
- ▶ Type inference
- ▶ Types: `Bits`, `UInt`, `SInt`, `Bool`

Conditional Updates for Combinational Circuits

```
val w = Wire(UInt())  
  
when (cond) {  
    w := 1.U  
} .elsewhen (cond2) {  
    w := 2.U  
} .otherwise {  
    w := 3.U  
}
```

- ▶ Similar to VHDL process or SystemVerilog always_comb
- ▶ Chisel checks for complete assignments in all branches
- ▶ Latches give compile error

Registers

```
val cntReg = RegInit(0.U(32.W))
```

```
cntReg := cntReg + 1.U
```

- ▶ Type inferred by initial value (= reset value)
- ▶ No need to specify a clock or reset signal
- ▶ Also definition with an input signal connected:

```
val r = RegNext(nextVal)
```

Functional Abstraction

```
def addSub(add: Bool, a: UInt, b: UInt) =  
  Mux(add, a+b, a-b)
```

```
val res = addSub(cond, a, b)
```

```
def rising(d: Bool) = d && !RegNext(d)
```

- ▶ Functions for repeated pieces of logic
- ▶ May contain state
- ▶ Functions may return *hardware*

Bundles

```
class DecodeExecute extends Bundle {  
  val rs1 = UInt(32.W)  
  val rs2 = UInt(32.W)  
  val immVal = UInt(32.W)  
  val aluOp = new AluOp()  
}
```

- ▶ Collection of values in named fields
- ▶ Like struct or record

Vectors

```
val myVec = Vec(3, SInt(10.W))
```

```
myVec(0) := -3.S
```

```
val y = myVec(2)
```

- ▶ Indexable vector of elements
- ▶ Bundles and Vecs can be arbitrarily nested

IO Ports

```
class Channel extends Bundle {  
  val data = Input(UInt(8.W))  
  val ready = Output(Bool())  
  val valid = Input(Bool())  
}
```

- ▶ Ports are Bundles with directions
- ▶ Direction can also be assigned at instantiation:

```
class ExecuteIO extends Bundle {  
  val dec = Input(new DecodeExecute())  
  val mem = Output(new ExecuteMemory())  
}
```

Hello World in Chisel

```
class Hello extends Module {  
  val io = IO(new Bundle {  
    val led = Output(UInt(1.W))  
  })  
  val CNT_MAX = (500000000 / 2 - 1).U;  
  
  val cntReg = RegInit(0.U(32.W))  
  val blkReg = RegInit(0.U(1.W))  
  
  cntReg := cntReg + 1.U  
  when(cntReg === CNT_MAX) {  
    cntReg := 0.U  
    blkReg := ~blkReg  
  }  
  io.led := blkReg  
}
```

Connections

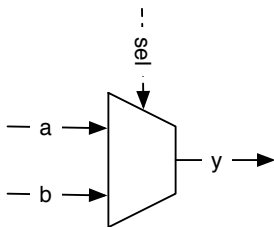
- ▶ Simple connections just with assignments, e.g.,

```
adder.io.a := ina  
adder.io.b := inb
```

- ▶ Automatic bulk connections between components

```
dec.io <> exe.io  
mem.io <> exe.io
```

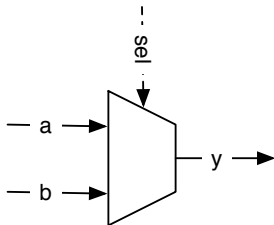
Chisel has a Multiplexer



```
val result = Mux(sel, a, b)
```

- ▶ So what?
- ▶ Wait... What type is a and b?
 - ▶ Can be any Chisel type!

Chisel has a Generic Multiplexer



```
val result = Mux(sel, a, b)
```

- ▶ SW people may not be impressed
- ▶ They have generics since Java 1.5 in 2004
 - ▶ `List<Flowers> != List<Cars>`

Generics in Hardware Construction

- ▶ Chisel supports generic classes with type parameters
- ▶ Write hardware generators independent of concrete type
- ▶ This is a multiplexer *generator*

```
def myMux[T <: Data](sel: Bool, tPath: T, fPath:
    T): T = {

    val ret = WireDefault(fPath)
    when (sel) {
        ret := tPath
    }
    ret
}
```

Put Generics Into Use

- ▶ Let us implement a generic FIFO
- ▶ Use the generic ready/valid interface from Chisel

```
class DecoupledIO[T <: Data](gen: T) extends  
  Bundle {  
    val ready = Input(Bool())  
    val valid = Output(Bool())  
    val bits  = Output(gen)  
  }
```

Define the FIFO Interface

```
class FifoIO[T <: Data](private val gen: T)
  extends Bundle {
  val enq = Flipped(new DecoupledIO(gen))
  val deq = new DecoupledIO(gen)
}
```

- ▶ We need enqueueing and dequeueing ports
- ▶ Note the Flipped
 - ▶ It switches the direction of ports
 - ▶ No more double definitions of an interface

But What FIFO Implementation?

- ▶ Bubble FIFO (good for low data rate)
- ▶ Double buffer FIFO (fast restart)
- ▶ FIFO with memory and pointers (for larger buffers)
 - ▶ Using flip-flops
 - ▶ Using on-chip memory
- ▶ And some more...
- ▶ This calls for object-oriented programming *hardware construction*

Abstract Base Class and Concrete Extension

```
abstract class Fifo[T <: Data](gen: T, depth: Int)
  extends Module {
    val io = IO(new FifoIO(gen))

    assert(depth > 0, "Number of buffer elements
      needs to be larger than 0")
  }
```

- ▶ May contain common code
- ▶ Extend by concrete classes

```
class BubbleFifo[T <: Data](gen: T, depth: Int)
  extends Fifo(gen: T, depth: Int) {
```

Select a Concrete FIFO Implementation

- ▶ Decide at hardware generation
- ▶ Can use all Scala/Java power for the decision
 - ▶ Connect to a web service, get Google Alphabet stock price, and decide on which to use ;-)
 - ▶ For sure a silly idea, but you see what is possible...
 - ▶ Developers may find clever use of the Scala/Java power
 - ▶ We could present a GUI to the user to select from
- ▶ We use XML files parsed at hardware generation time
- ▶ End of TCL, Python,... generated hardware

Binary to BCD Conversion for VHDL

[illegible]

Java Program

- ▶ Generates a VHDL table
- ▶ The core code is:

```
for (int i = 0; i < Math.pow(2, ADDRBITS); ++i) {  
    int val = ((i/10)<<4) + i%10;  
    // write out VHDL code for each line
```

- ▶ With all boilerplate 118 LoC

Chisel Version of Binary to BCD Conversion

```
val table = Wire(Vec(100, UInt(8.W)))
for (i <- 0 until 100) {
  table(i) := (((i/10)<<4) + i%10).U
}
val bcd = table(bin)
```

- ▶ Directly generates the hardware table as a Vec
- ▶ At hardware construction time
- ▶ In the same language

Use Functional Programming for Generators

```
def add(a: UInt, b: UInt) = a + b
```

```
val sum = vec.reduce(add)
```

```
val sum = vec.reduce(_ + _)
```

```
val sum = vec.reduceTree(_ + _)
```

- ▶ This is a simple example
- ▶ What about an arbiter tree with fair arbitration?

Free Tools for Chisel and FPGA Design

- ▶ Java OpenJDK 8
- ▶ sbt, the Scala (and Java) build tool
- ▶ IntelliJ (the free Community version)
- ▶ GTKWave
- ▶ Vivado WebPACK or
- ▶ Quartus
- ▶ Nice to have:
 - ▶ make, git

Chisel in the T-CREST Project

- ▶ Patmos processor rewritten in Chisel
 - ▶ As part of learning Chisel
 - ▶ 6.4.2013: Chisel: 996 LoC vs VHDL: 3020 LoC
 - ▶ But VHDL was very verbose, with records maybe 2000 LoC
- ▶ Memory controller, memory arbiters, IO devices in Chisel
- ▶ Several Phd, master, and bachelor projects:
 - ▶ Patmos stack cache
 - ▶ Method cache for Patmos
 - ▶ TDM based memory arbiter
 - ▶ RISC stack cache
 - ▶ and some more

Chisel in Teaching

- ▶ Using/offering it in Advanced Computer Architecture
- ▶ Spring 2016–2018 all projects have been in Chisel
- ▶ Several Bachelor and Master projects
- ▶ Students pick it up reasonable fast
- ▶ For software engineering students easier than VHDL
- ▶ Switched Digital Electronics 2 at DTU to Chisel (spring semester 2020)
- ▶ Issue of *writing a program* instead of describing hardware remains

Chisel in Digital Electronic 2

- ▶ Basic RTL level digital design with Chisel
- ▶ Chisel testers for debugging
- ▶ Very FPGA centric course
- ▶ Final project is a vending machine
- ▶ All material (slides, book, lab material) in open source
- ▶ Tried to coordinate with introduction to programming (Java)
 - ▶ But sometimes I was ahead with Chisel constructs (e.g., classes)

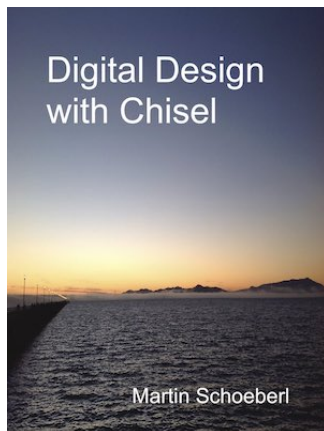
Then there was the Lockdown

- ▶ Usually one FPGA board per group
- ▶ No group meetings
- ▶ Just virtual labs
- ▶ Can I do something about it with Chisel?

Teaching Feedback

- ▶ General positive feedback of the course
- ▶ Most students liked Chisel
- ▶ They also liked the (free) Chisel book
- ▶ Better link to Java programming (same JVM)
 - ▶ Similar setup (IDE)
- ▶ Lab finish about the same time as last year with VHDL
 - ▶ So Chisel is not more productive than VHDL?
 - ▶ But we had the Corona lockdown

A Chisel Book



- ▶ Available in open access (PDF)
- ▶ In paper from Amazon
- ▶ see <http://www.imm.dtu.dk/~masca/chisel-book.html>

What May Happen with an Open-Source Book



- ▶ A free Chinese translation

Furthermore, I got This

Chiselを使ったデジタル・デザイン

第二版(日本語版)

マーチン・シェーベル著

Chisel勉強会訳

- ▶ A Japanese translation

Further Information

- ▶ <https://www.chisel-lang.org/>
- ▶ <http://www.imm.dtu.dk/~masca/chisel-book.html>
- ▶ <https://github.com/ucb-bar/chisel-tutorial>
- ▶ <https://github.com/ucb-bar/generator-bootcamp>
- ▶ <http://groups.google.com/group/chisel-users>
- ▶ <https://github.com/schoeberl/chisel-book>
- ▶ <https://github.com/schoeberl/chisel-lab>

Summary

- ▶ Processors do not get much faster – we need to design custom hardware
- ▶ We need a modern language for hardware/systems design
- ▶ Chisel is a small language
- ▶ Embedding it in Scala gives the power
- ▶ We can write circuit generators
- ▶ We can do co-simulation
- ▶ I can provide further introduction into Chisel including labs