Digital Design in the 21st Century: Chisel

Martin Schoeberl

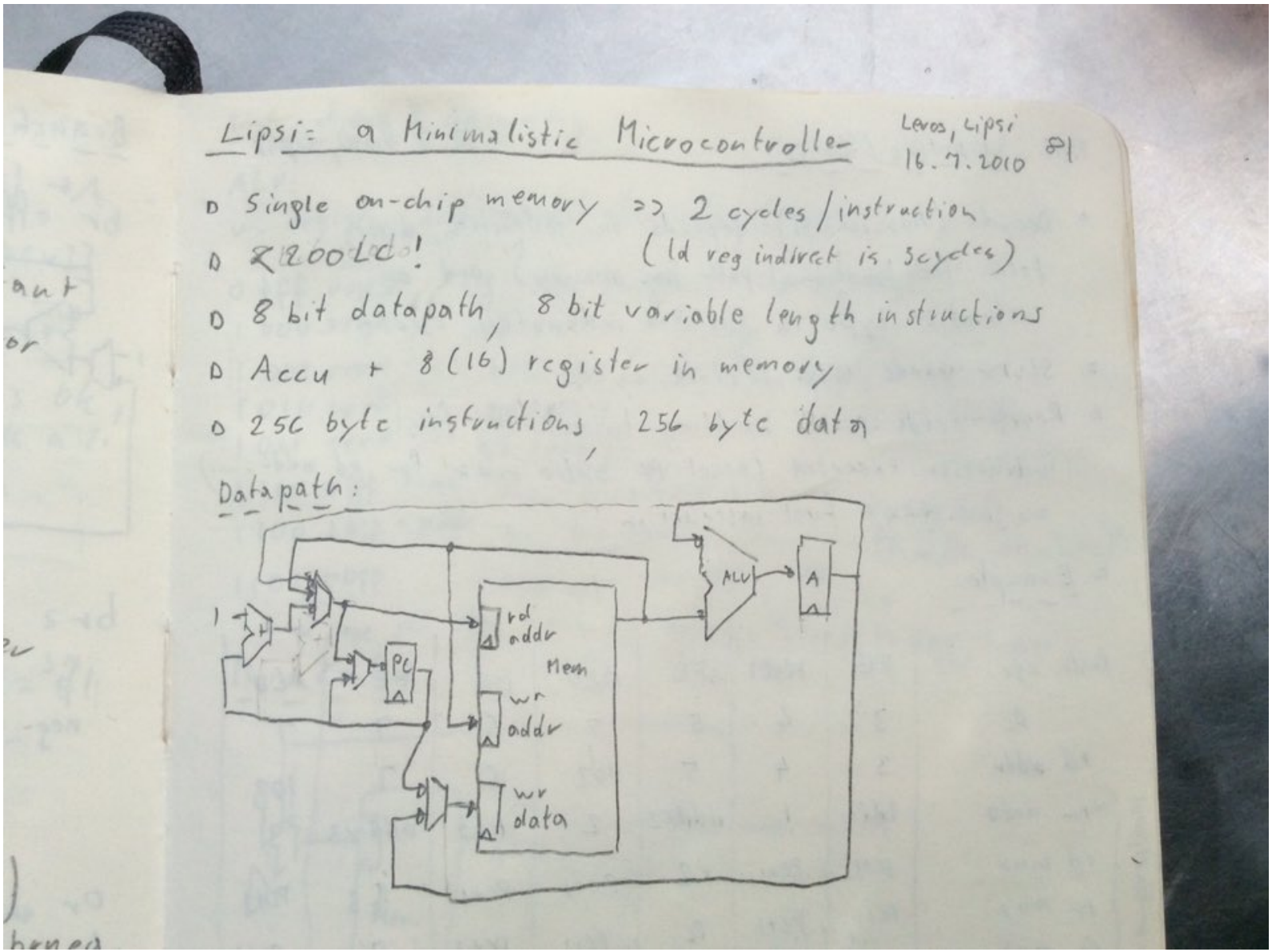
Technical University of Denmark

October 12, 2019

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 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 (if beamer allows)

The Slides are Online

- https://github.com/schoeberl/chisel-book/wiki

Goals for this Intro

- Get an idea what Chisel is
  - Will show you code snippets
- A first high level view of the main features of Chisel
- Reconsider how to describe hardware
- Some experience report from usage at DTU
- Pointers to more information
- Have your first Chisel design running in an FPGA!

Chisel

- A hardware *construction* language
  - Constructing Hardware In a Scala Embedded Language
  - If it compiles, it is synthesisable hardware
  - Say goodby 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

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*

Expressions are Combinational Circuits

```
(a | b) & ~(c ^ 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

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 arbitrarely 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())
}
```

Modules

```
class Adder extends Module {
  val io = IO(new Bundle {
    val a = Input(UInt(4.W))
    val b = Input(UInt(4.W))
    val result = Output(UInt(4.W))
  })

  val addVal = io.a + io.b
  io.result := addVal
}
```

- Organization of components
- IO ports defined as a Bundle named `io` and wrapped into an `IO()`
- Created (instantiated) with:

```
val adder = Module(new Adder())
```

Hello World in Chisel

```
class Hello extends Module {
  val io = IO(new Bundle {
    val led = Output(UInt(1.W))
  })
  val CNT_MAX = (50000000 / 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
```

Generic Components

```
val c = Mux(cond, a, b)
```

- This is a multiplexer
- Input can be any type

Testing

```
class CounterTester(c: Counter) extends
    PeekPokeTester(c) {
  for (i <- 0 until 5) {
    println(i.toString + ": " +
        peek(c.io.out).toString())
    step(1)
  }
}
```

- Within Chisel with a tester (= Scala program)
- May include waveform generation
- peek and poke to read and set values
  - Remember the BASIC days ;-)
- printf in simulation on rising edge

```
printf("Counting %x\n", r1)
```

Component Generation

```
val cores = new Array[Module](32)

for (j <- 0 until 32)
  cores(j) = Module(new CPU())
```

- Use Scala array to collect components
- Generation with a Scala loop

Conditional Component Generation

```
val icache =
  if (TYPE == METHOD)
    Module(new MCache())
  else if (TYPE == LINE)
    Module(new ICache())
  else
    ChiselError.error("Unsupported Type")
```

- Use Scala if/else for conditional component types
- Code example from Patmos
- We parse an XML file for the configuration

Logic Generation

- Read a file into a table
  - E.g., to read in ROM content for a processor
- Generate a truth table algorithmically
  - E.g., generate binary to BCD translation
- Use the full power of Scala

```
val byteArray =
    Files.readAllBytes(Paths.get(fileName))
val arr = new Array[Bits](byteArray.length)
for (i <- 0 until byteArray.length) {
  arr(i) = Bits(byteArray(i), 8)
}
val rom = Vec[Bits](arr)
```

An IDE for Chisel

- Eclipse or IntelliJ
- Scala plugin
- For an Eclipse project: sbt eclipse
- For IntelliJ: File - New - Project from Existing Sources..., open build.sbt
- Show it

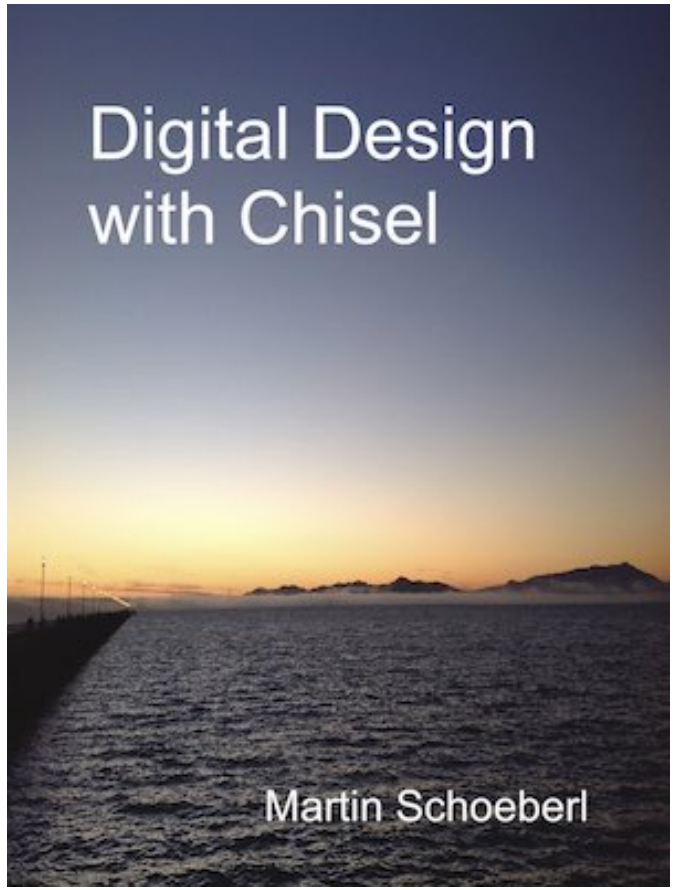
Chisel in the T-CREST Project

- 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 – Chisel was relative easy
  - TDM based memory arbiter – trouble with Chisel
  - RISC stack cache – no issues with Chisel
  - and more ongoing

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
- Switch Digital Electronics 2 at DTU to Chisel (spring semester 2020)
- Issue of *writing a program* instead of describing hardware remains

A Chisel Book



- Available in open access (PDF)
- In paper from Amazon

Further Information

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

Hello World in Chisel and Examples

```
git clone
    https://github.com/schoeberl/chisel-examples.git
```

- or download from
  https://github.com/schoeberl/chisel-examples
- This contains a minimal Chisel project with the blinking LED
- Has ready to use project files for:
  - Altera DE0
  - Altera DE1
  - Altera DE2-115
  - Altera DE10-Nano
  - BeMicro
- Plus a simple ALU for HW test and showing Chisel Tester
- Plus some more examples to explore

What is a Minimal Chisel Project?

- Scala class (e.g., `Hello.scala`)
- Build info in `build.sbt` for `sbt`:

```
scalaVersion := "2.11.7"

libraryDependencies += "edu.berkeley.cs" %%
   "chisel3" % "3.1.2"
```

- Run the process manually (look into the Makefile)

Additional Files in the Hello World Example

- A `Makefile`
- Optional Verilog or VHDL top level file
- Quartus project files `quartus/altde2-115/hello.q{p|s}f`
  - List of source files, device and pin assignments
  - plain text files, look into hello.qsf

Summary

- We need a modern language for hardware/systems design
- Chisel builds on the power of object-oriented and functional Scala
- Chisel is good for hardware generators

Lab Time

- Get a blinking LED working on the FPGA board
- Clone or download the repository now

```
git clone \\
   https://github.com/schoeberl/chisel-examples.git
cd chisel-examples/hello-world
make
```

- Start Quartus
- Open the project at `chisel-examples/hello-world/quartus/altde2-115`
- Synthesize with the Play button
- Configure the FPGA with the Programmer button
- **You have your first Chisel design running!**

Change the Design

- Use `gedit`, or the editor you like most
- Source is in `.../src/main/scala/Hello.scala`
- Change blinking frequency
- Rerun the example
- Optional:
  - Change to an asymmetric blinking, e.g., 200 ms on every second
  - Setup IntelliJ with a Scala project