# Software-Defined Hardware: Digital Design in the 21st Century with Chisel

Martin Schoeberl

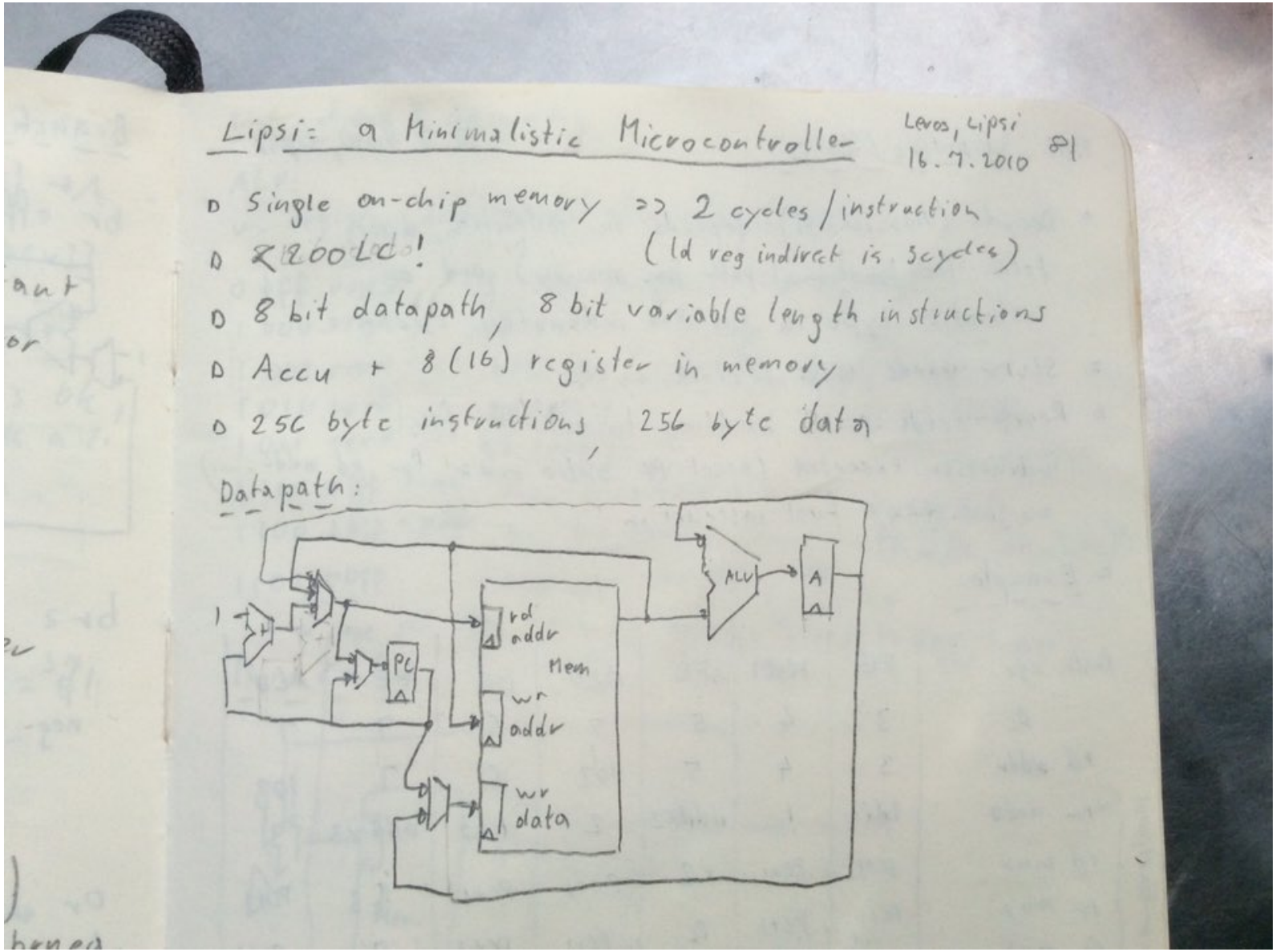
Technical University of Denmark

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 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

- Before the lockdown at 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
- Experanto 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

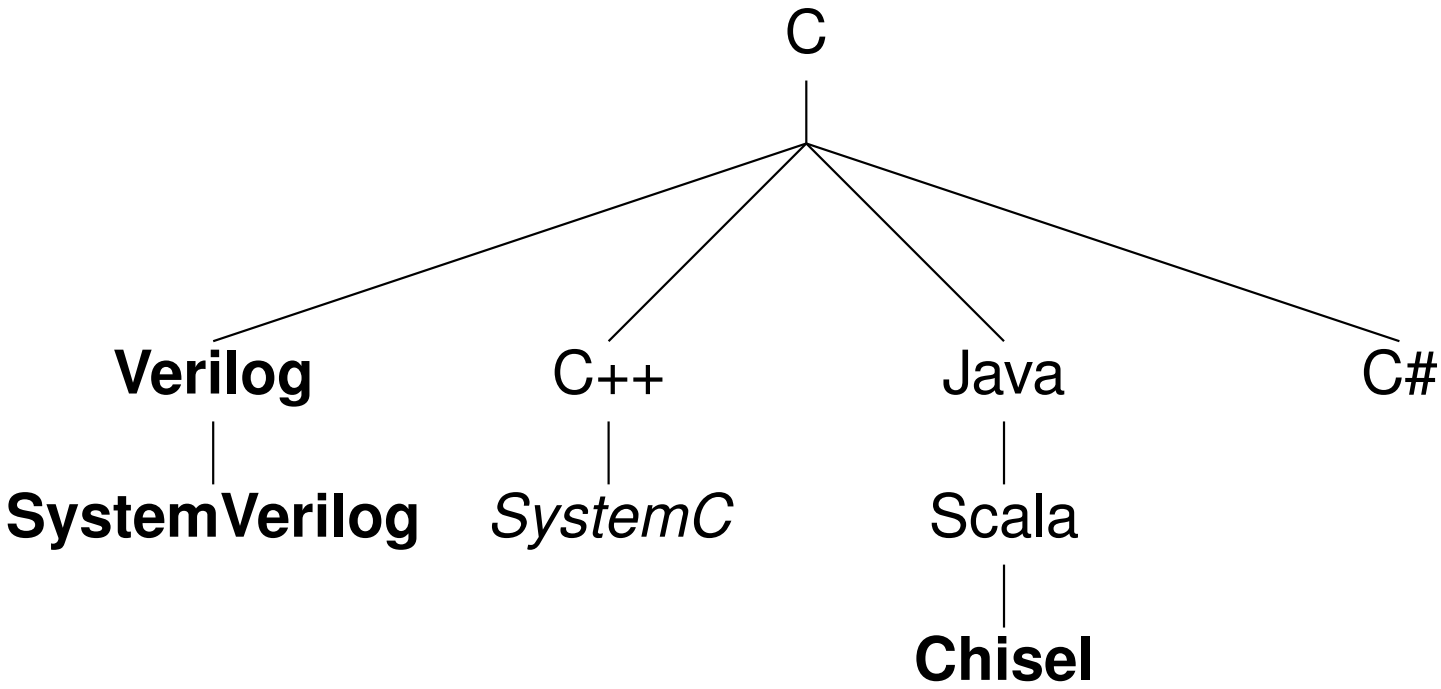
# Goals for this Intro

- 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!

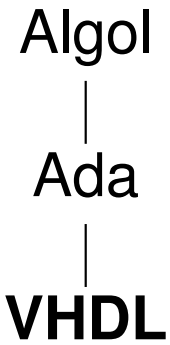
# Chisel

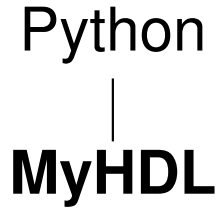
- A hardware *construction* language
  - Constructing Hardware In a Scala Embedded Language
  - If it compiles, it is synthesisable hardware
  - Say goodby to your unintended latches
- Chisel is not a high-level synthesis language
- Single source two targets
  - Cycle accurate simulation (testing)
 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

# The C Language Family



# Other Language Families



# 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

# 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

# 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 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")
```

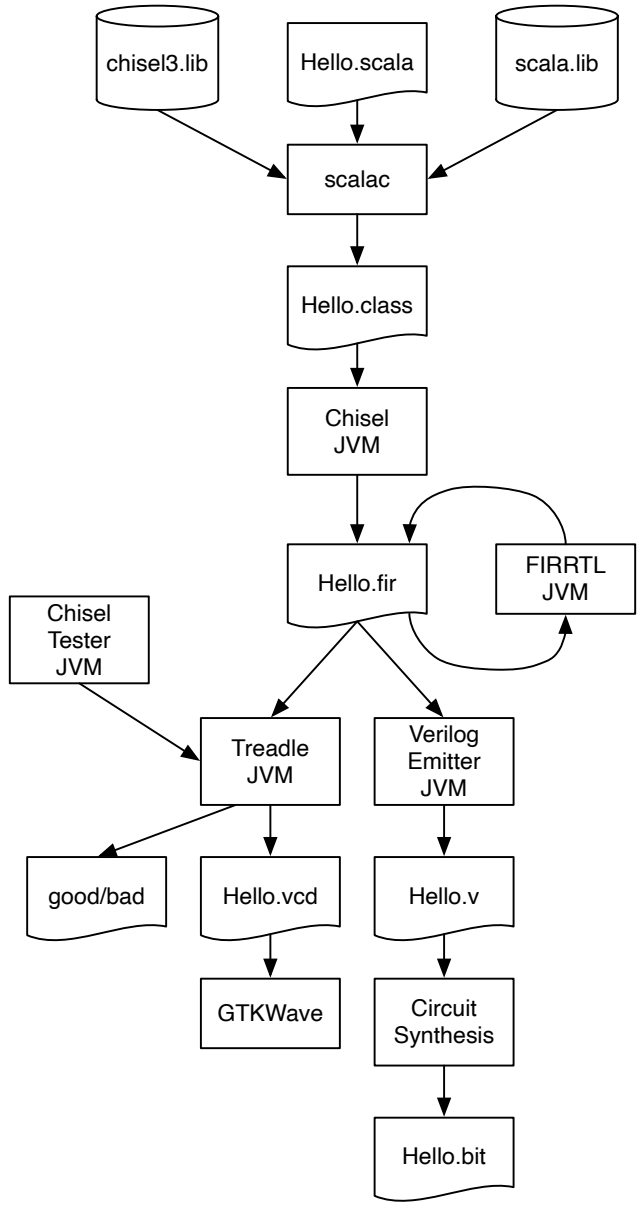
- Use Scala if/else for conditional component types
- Code example from Patmos
- We parse an XML file for the configuration

# 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)
```

# Tool Flow for Chisel



# 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
- Setup instructions: `https://github.com/schoeberl/chisel-lab/blob/master/Setup.md`

# Tool Setup for Different OSs

- Windows
  - Use the installers from the websites
- macOS
  - `brew install sbt`
  - For the rest, use the installer from the websites
- Linux/Ubuntu
  - `sudo apt install openjdk-8-jdk git make gtkwave`
  - Install sbt, see https://github.com/schoeberl/chisel-lab/blob/master/Setup.md
  - IntelliJ as from the website

# Virtual Machine Setup for Chisel

- Ubuntu based
- Ubuntu VM with Quartus uid: patmos, pwd: patmos
- Ubuntu VM with Vivado uid: de2lab, pwd: de2lab
  - But this is VERY large (40 GB for the .zip file)
- Use the VMWare Workstation Player (free for Linux and Windows)

# An IDE for Chisel

- IntelliJ (Eclipse Scala support is/was shaky)
- 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

# Chisel in Digital Electronic 2

- Basic RTL level digital design wit 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)

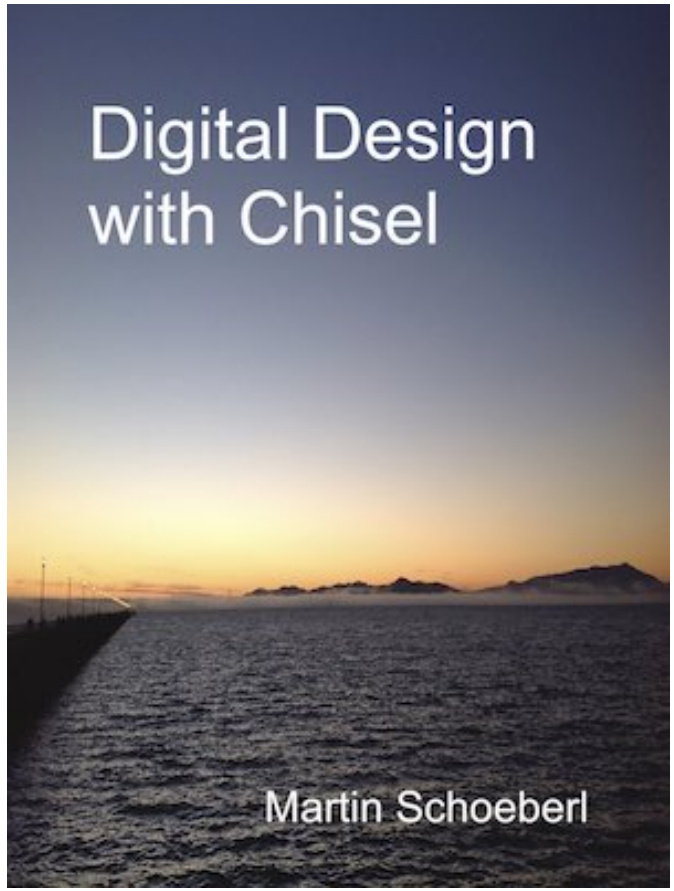
# Then there was the Lockdown

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

# Teaching Feedback

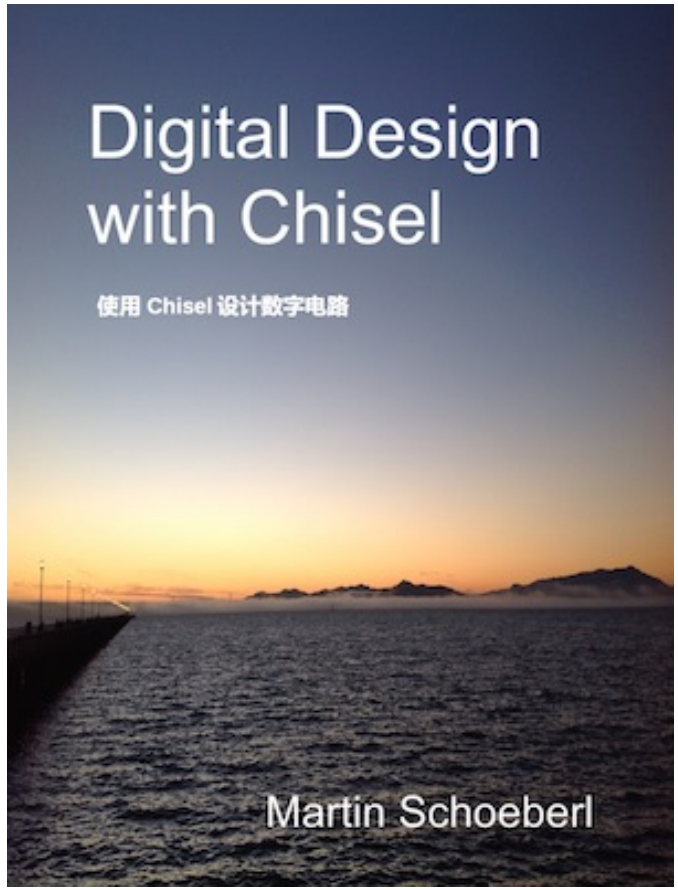
- General positive feedback of the course
- Most students liked Chisel
- They also liked the (free) Chisel book
- Better link to Java programming (same JVM)
  - Similar setup (IDE)
- Lab finish about the same time as last year with VHDL
  - So Chisel is not more productive than VHDL?
  - But we had the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

# A Chisel Book




- Available in open access (PDF)
- In paper from Amazon
- see http://www.imm.dtu.dk/~masca/chisel-book.html
- Source at https://github.com/schoeberl/chisel-book

# What May Happen with an Open-Source Book




- A free Chinese translation

# Several Weeks ago 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://github.com/schoeberl/chisel-book
- https://github.com/schoeberl/chisel-lab
- 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

# 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

- Processors do not get much faster – we need to design domain specific hardware accelerators
- We need a modern language for hardware/systems design
- Chisel is a small language
- Embedding it in Scala gives the power
- Chisel is good for hardware generators
- We can do co-simulation

# Lab Time (Simulation)

- Get a blinking LED working in simulation
- Clone or download the repository from:
  - https://github.com/schoeberl/chisel-lab
- Follow the instructions from the lab page
  - Start IntelliJ and follow the instructions from the lab page
  - `sbt run`
  - Observe the generated Verilog file
  - For simulation change the factor for the counter and run the simulation
  - `val CNT_MAX = (50000 / 2 - 1).U;`
  - `sbt test`
- **You have your first Chisel design running in simulation!**

# Lab Time (Basys3)

- Get a blinking LED working on the FPGA board
- Clone or download the repository from:
  - https://github.com/schoeberl/chisel-lab
- Follow the instructions from the lab page
  - Start IntelliJ and follow the instructions from the lab page
  - `sbt run`
  - Create a Vivado project
  - Synthesize with the Play button
  - Configure the FPGA with the Programmer button
- **You have your first Chisel design running in an FPGA!**

# Lab Time (DE2-115)

- 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 in an FPGA!**

# 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