Digital Electronics 2: Introduction

Martin Schoeberl

Technical University of Denmark
Embedded Systems Engineering

February 6, 2025

Overview

- Motivation and the digital abstraction
- Course organization
- Languages for digital hardware design
- A first round of Chisel
- Tools and tool setup
- Lab: a hardware “Hello World”

A BIG Chip

- https://singularityhub.com/2019/08/26/this-giant-ai-chip-is-the-size-of-an-ipad-and-holds-1-
- 1.2×10^{12} transistors
- If you design 1 gate (= 4 transistors) per second
  - It takes you 10 thousand years!
- This calls for some abstraction

Digital Systems are Everywhere

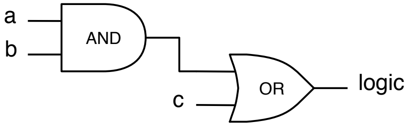
- Digital systems are all over in our live
- No more analog media
- CD, mobile phone, TV, DVD,... all digital now
- Analog circuits only at the edge
- The rest is processed in digital
- If performance allows, functions are moved to software
- But processor speedup has slowed down
- Algorithms are moved back into hardware

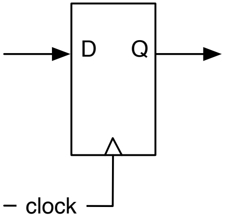
FPGAs in the Cloud

- High performance algorithm in an FPGA
- An FPGA in the cloud
- Intel offers FPGAs for servers
  - There was some reason why Intel bought Altera
  - And AMD bought Xilinx
- We need digital designers to make this work
- A good time to be a digital designer

The Digital Abstraction

- Just two values: 0 and 1, or low and hight
- Represented as voltage
- Digital signals tolerate noise
- Digital Systems are *simple*, just:
  - Combinational circuits and
  - Registers





Hardware Design in DK

- Demant (former Oticon)
- WSAudiology (former Widex)
- GN ReSound
- Microchip
- Intel (former Altera) Denmark
- SyoSil
- Comcores
- Synopsys
- Napatech
- Teledyn
- and some more...
-
- They are all hiring

Computer Engineering Education at DTU

- Between hardware (EE) and software (CS)
- Very well payed jobs :-)
- Now a new BSc in CE is available at DTU
- You can also start with a Bsc in EE
- Specialization in Indlejrede systemer og programmering
  - 02155: Computer Architecture and Engineering
  - 02105: Algoritmer og datastrukturer
- Take some of the new CE courses
- Continue as MSc. in Computer Science and Engineering
- Specialization in
  - Digital Systems

Digital Design within an CS or EE Master

- Select some of the following courses
  - 02155: Computer Architecture and Engineering
  - 02118: Introduction to Chip Design
  - 02203: Design of Digital Systems
  - 02211: Research Topics in Computer Architecture
  - 02205: VLSI Design
  - 022xx: Verification of Digital Designs
  - 02209: Test of Digital Systems

Web Resources

- DTU Learn
  - Group building
  - Project report hand in
- Course website
  - General information, starting point
- Lab website
  - Lab material on GitHub
- Chisel book website
  - Download the free PDF

Organization and Workload

- Usually 2 hours lectures and 2 hours supervised lab
- 5 ECTS is equivalent to 9 hours per week
- That means 5 hours work on your own
  - Do some reading, prepare for the lecture and lab
  - Get the tools installed on your laptop
  - You have an FPGA board, experiment with it
- You will learn a lot in this course, it will make you a better:
  - engineer
  - hardware designer
  - programmer, and
  - computer user in general
- Try to have fun with building stuff that is real!

Important Help

- We have two TAs for this semester
- David Wilson Jacobsen
- Ehsan Khodadad
- You will meet them in the lab
- Shall we start a Discord server for the course?

Lab Work

- A Vending Machine
  - At the end it shall run in your FPGA board
  - I am a big fan of running stuff in real hardware
- I know many groups have only one physical FPGA board
  - A lot can be done in simulation
  - I developed a simulation of the Basys3 board (during first lockdown)
- I assume you will find a solution for file sharing
  - GitHub is a popular one for source code
  - Can also be used if you plan to write your report in LaTeX

Communication and Getting Help

- Several sources of information:
  - The Internet, Google, and Stackoverflow
  - Your fellow students (e.g., via Discord)
  - The TAs
  - Me
- You can always just knock on my door or shoot me an email
- *Official* info will be sent via DTU Learn email

Cheating and Plagiarism

- It is ok and good practice to discuss problems and solutions with your fellow students
- But you need to hand in your own solution
- Copying stuff or offering stuff for copying is cheating
- Copying material from somewhere is plagiarism and copyright violation
- Cheating is handled quite rigorous at DTU, you might get expelled
- Using source code control (GitHub) is good practice
- However, keep it private. Otherwise you might contribute to cheating

AI, ChatGPT, and Co.

- DTU has changed their rules
  - I do not know the exact rules currently
  - I, as teacher, can override those rules
- AI is part of our live, it is just another tool in our toolbox
- I use ChatGPT, MS Copilot, GitHub Copilot, and DeepSeek
- You are allowed to use AI as well
- However, be careful to learn some coding by yourself (exam without Internet)
- If you use it:
  - Cite it in your report
  - Write a short section on reflecting how useful it was

This is an Open-Access/Open-Source Course

- Almost all material is public visible
- Slides are open source
- Lab material is open source
- The Chisel book is open source
- Hosted on GitHub
  - **You** can contribute with a pull request
  - Becoming an author of this course :-)

Lab Work

- Some paper and pencil exercises
- Two personal hand-ins of typical exam problems
- Most work on designing digital circuits with a hardware description language
- Builds up to the final project: a vending machine
- The hand-ins, vending machine, and the report are graded

Questions?

- On lectures
- On the group/lab work

A Vending Machine from 1952

Source: Minnesota Historical Society, CC BY-SA 2.0

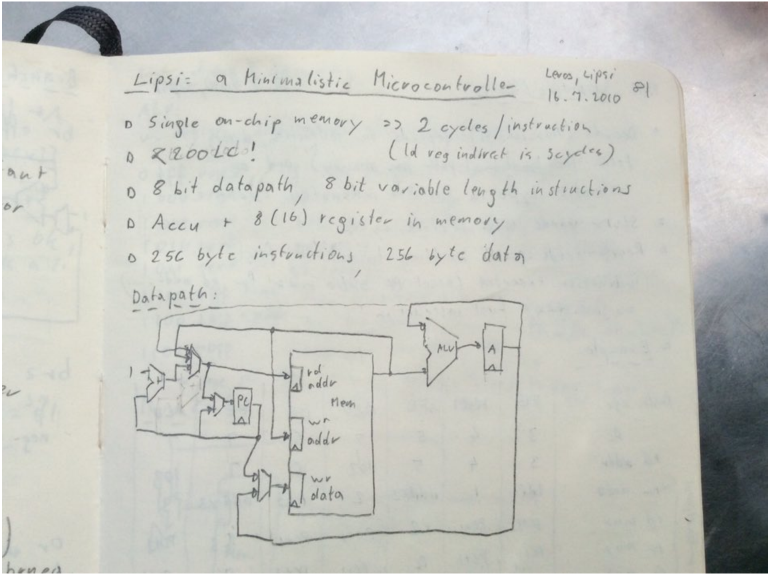
The Vending Machine

- Final project is a vending machine
- Inputs: coins, buy
- Display: price and current amount
- Output: release can or error
- Small challenge to multiplex the display
- State machine with data path is the *brain* of the VM
- Will be guided step by step over several weeks
- More details next week
- VM in hardware versus VM in software
    - This is an exercise that you can solve with reasonable effort

Motivating Example for Chisel: 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
  - Could be used for your vending machine
  - 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

- http://www2.imm.dtu.dk/courses/02139/
- https://github.com/schoeberl/chisel-book/tree/master/slides

10 Minutes Break

Why Chisel Instead of VHDL/Verilog/SystemVerilog?

- Company O does Verilog, company W does VHDL
  - Why Chisel?
- We learn principles of digital design, not tools
  - We pick a language that is modern and productive
- When knowing principles, switching the language is a matter of a week
- You are the future engineers and shall learn new tools
- You may then bring Chisel into the company

More on Chisel Success Stories

- Last live conference CCC 2020 in silicon valley
- 90 participants
- More than 30 different chip 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 even contribute to the Chisel ecosystem :-)

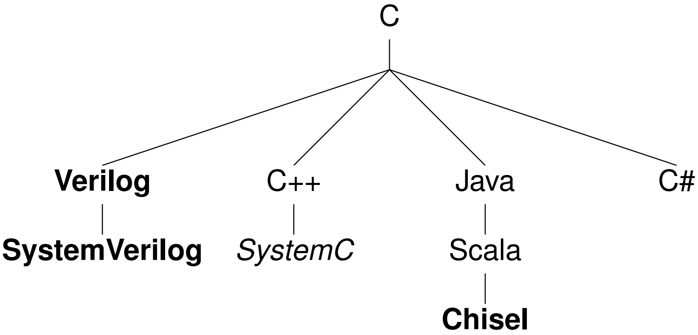
Introduction to Chisel

- Get an idea what Chisel is
  - Will show you code snippets
- Basic hardware constructs in Chisel
- Pointers to more information
- Have your first Chisel design running in an FPGA!
  - From 0 to 100 in one afternoon

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 for 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 68
|
Ada
|
VHDL

Python
|
MyHDL

What Language do You Already Know?

- ???

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
  - You can write your testing code in Java

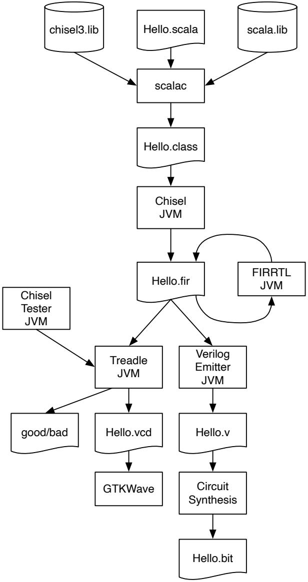
Chisel vs. Scala

- 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 Defined Hardware



Signal Types

- All types in hardware are a collection of bits
- The base type in Chisel is `Bits`
- `UInt` represents an unsigned integer
- `SInt` represents a signed integer (in two's complement)

```
Bits(8.W)
UInt(8.W)
SInt(10.W)
```

Number of Bits: n.W

- A collection of bits has a *width*
- The width is the number of bits
- Is written as number followed by `.W`
- Following example shows the width of `n`

```
n.W
Bits(n.W)
```

Constants

- Constants can represent signed or unsigned numbers
- We use `.U` and `.S` to distinguish

```
0.U  // defines a UInt constant of 0
-3.S // defines a SInt constant of -3
```

- Constants can also be specified with a width

```
3.U(4.W) // An 4-bit constant of 3
```

Hexadecimal and Binary Representation

- We can specify constants with a different base
- May come handy sometimes

```
"hff".U         // hexadecimal representation of
   255
"o377".U        // octal representation of 255
"b1111_1111".U // binary representation of 255
```

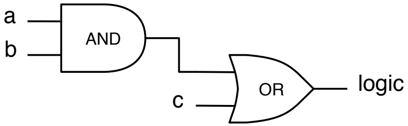
Boolean Values

- Type for logical values
- Can be `true` or `false`
- Almost exchangeable with `UInt(1.W)`
- Sometimes a signal, such as `valid`, may be better represented by a Boolean type

```
Bool()
true.B
false.B
```

Combinational Circuits

- Chisel uses Boolean operators, similar to C or Java
- & is the AND operator and `|` is the OR operator
- The following code is the same as the schematics
- `val logic` gives the circuit/expression the name `logic`
- That name can be used in following expressions




```
val logic = (a & b) | c
```

Standard Logic Operations

```
val and = a & b // bitwise and
val or = a | b  // bitwise or
val xor = a ^ b // bitwise xor
val not = ~a    // bitwise negation
```

- Note that we do not need to define the width of the values
- Note also that this is *hardware*
- All expressions are evaluated in parallel
- Order does not matter

Arithmetic Operations

- Same as in Java or C
- The width of the result is automatically computed
- E.g., the width of the multiplication is the sum of the width of `a` and the width of `b`

```
val add = a + b // addition
val sub = a - b // subtraction
val neg = -a    // negate
val mul = a * b // multiplication
val div = a / b // division
val mod = a % b // modulo operation
```

Wires

- A signal (or wire) can be first defined
- And later assigned an expression with `:=`

```
val w = Wire(UInt())

w := a & b
```

Chisel Defined Hardware Operators

Operator	Description	Data types
`* / %`	multiplication, division, modulus	UInt, SInt
`+ -`	addition, subtraction	UInt, SInt
`=== =/=`	equal, not equal	UInt, SInt, returns Bool
`> >= < <=`	comparison	UInt, SInt, returns Bool
`<< >>`	shift left, shift right (sign extend on SInt)	UInt, SInt
`~`	NOT	UInt, SInt, Bool
`& | ^`	AND, OR, XOR	UInt, SInt, Bool
`!`	logical NOT	Bool
`&& ||`	logical AND, OR	Bool

Subfields and Concatenation

A single bit can be extracted as follows:

```
val sign = x(31)
```

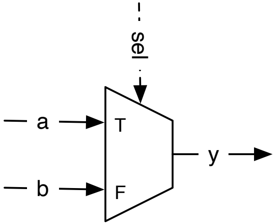
A subfield can be extracted from end to start position:

```
val lowByte = largeWord(7, 0)
```

Bit fields are concatenated with the `##` operator:

```
val word = highByte ## lowByte
```

A Multiplexer



- A Multiplexer selects between alternatives
- So common that Chisel provides a construct for it
- Selects `a` when `sel` is `true.B` otherwise `b`

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

Conditional Update

- With `when` we can express a conditional update
- The resulting circuit is a multiplexer
- In contrast to the `Mux` component, we can have several assignments in the `when` block
- The rule is the the last enabled assignment counts
  - Here the order of statements has a meaning

```
val w = Wire(UInt())

w := 0.U
when (cond) {
  w := 3.U
}
```

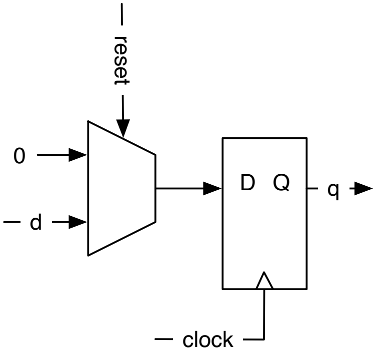
The World of Combinational Logic

- With the shown operations (logic, arithmetic, Mux) all possible combinational circuits can be described
- Even the `Mux` is already *syntactic sugar*
  - A `Mux` is basically: `(a & sel) | (b & !sel)`
- But Chisel provides further constructs for more elegant description of circuits
- Stay tuned!

Register

- A register is a collection of flip-flops
- Updated on the rising edge of the clock
- May be set to a value on reset
- Clock and reset are implicitly connected to the register
- A register can be any Chisel type that can be represented as a collection of bits

A Register with Reset



A Register with Reset

Following code defines an 8-bit register, initialized with 0 at reset:

```
val reg = RegInit(0.U(8.W))
```

An input is connected to the register with the `:=` update operator and the output of the register can be used just with the name in an expression:

```
reg := d
val q = reg
```

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
}
```

Chisel is a Hardware Construction Language

- The code I showed you looks much like Java code
- But it is *not* a program in the usual sense
- It represents a circuit
- The "program" constructs the circuit
- All statements are "executed" in parallel
- Statement order has *mostly* no meaning

Free Tools for Chisel and FPGA Design

- Java OpenJDK 8 (or later) already installed for Java course
- sbt, the Scala (and Java) build tool
- IntelliJ (the free Community version)
- GTKWave
- Vivado WebPACK already installed from DE1
- Nice to have:
  - make, git

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
  - Use an Ubuntu VM to run Vivado
- Linux/Ubuntu
  - `sudo apt install openjdk-8-jdk git make gtkwave`
  - Install sbt
  - IntelliJ as from the website
- Instruction details: https://github.com/schoeberl/chisel-lab/blob/master/Setup.md

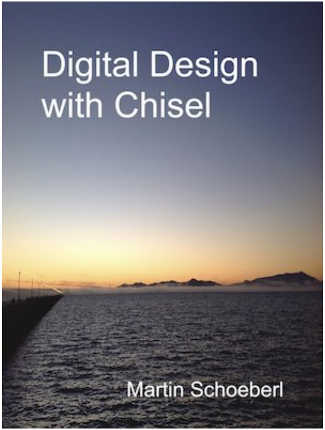
Virtual Machine Setup for Chisel

- If setup fails, we have you covered with a Virtual Machine
- Ubuntu based
- 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)
  - Use the free VMWare Fusion for macOS

An IDE for Chisel

- IntelliJ
- Install the Scala plugin
- For IntelliJ: File - New - Project from Existing Sources..., open build.sbt
- Show it

A Chisel Book



- Available in open access (as PDF)
  - Optimized for reading on a tablet (size, hyper links)
- Amazon can do the printout

Further Information

- https://www.chisel-lang.org/
- https://github.com/freechipsproject/chisel-cheatsheet/releases/latest/download/chisel_cheatsheet.pdf
- 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

Lab Time: Hello World in Chisel

- Get a blinking LED working on your 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!**
  - There is also a simulation version available

Change the Design

- Use IntelliJ, `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

Tiny Tapeout Workshop

- What is Tiny Tapeout?
- Tiny Tapeout can have your design on a real chip
- For $ 50 instead of $ 10000
- We have a TT workshop at DTU Fr. 15th Feb.
- Given by Matt Venn
- https://edu4chip.github.io/ttw2025DTU.html
- DTU will buy some PCBs
- test your chip on a bringup party

Summary

- The world is digital
- Processors do not get much faster – we need to design custom hardware
- We need a modern language for hardware/systems design for efficient/fast development
- Chisel builds on the power of object-oriented and functional Scala

Let’s have a Chat

- I will join you in the lab