

Basic Digital Circuits in Chisel

Martin Schoeberl

Technical University of Denmark
Embedded Systems Engineering

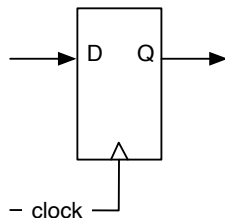
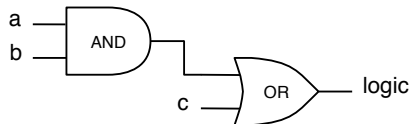
February 11, 2021

Overview

- ▶ Quick recap of last lecture
 - ▶ If something is unclear, please ask!
- ▶ Basic digital building blocks
- ▶ And the coding of it in Chisel
- ▶ Some coding style

The Digital Abstraction

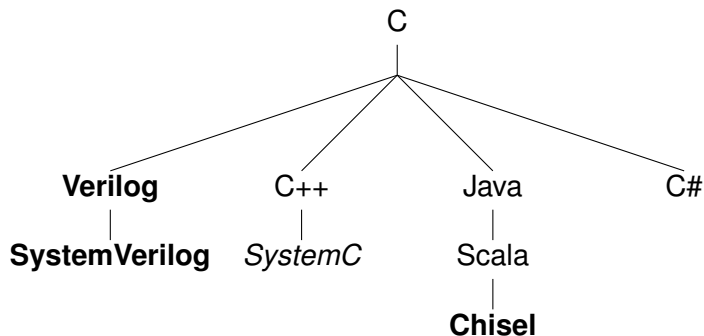
- ▶ Just two values: 0 and 1, or low and high
- ▶ Represented as voltage
- ▶ Digital signals tolerate noise
- ▶ Digital Systems are *simple*, just:
 - ▶ Combinational circuits and
 - ▶ Registers



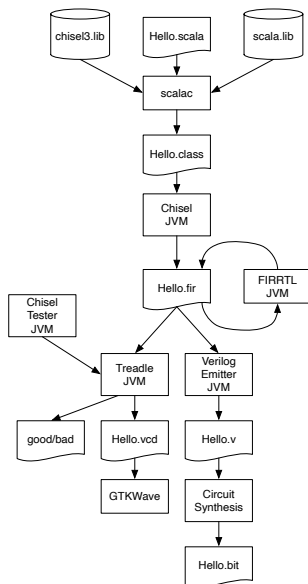
Chisel

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

Chisel is Part of the C Language Family



Tool Flow for Chisel Defined Hardware



Signal/Wire Types and Width

- ▶ 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)
- ▶ The number of bits is the width
- ▶ The width written as number followed by `.W`

`Bits(8.W)`

`UInt(8.W)`

`SInt(10.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

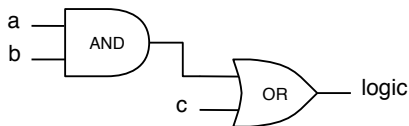
```
3.U(4.W) // An 4-bit constant of 3
```

- ▶ Use the string notation for a different base

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


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
```

Arithmetic and Logic Operations

- ▶ Same as in Java or C
- ▶ But this is *hardware*

```
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
```

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

Operators

- ▶ Operators precedence is the same as in Java
 - ▶ E.g., * has precedence over +
 - ▶ But different in VHDL or Verilog
 - ▶ Use parentheses when unsure (especially for logical expressions)
- ▶ + and - is relatively cheap
- ▶ * is expensive, know what you do
- ▶ / and % is VERY expensive, usually no direct use in hardware
 - ▶ Implement as a multi-cycle operation

Wires

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

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

```
w := a & b
```

Subfields and Concatenation

A single bit can be extracted as follows:

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

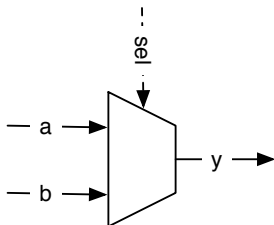
A subfield can be extracted from end to start position:

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

Bit fields are concatenated with Cat:

```
val word = Cat(highByte, lowByte)
```

A Multiplexer

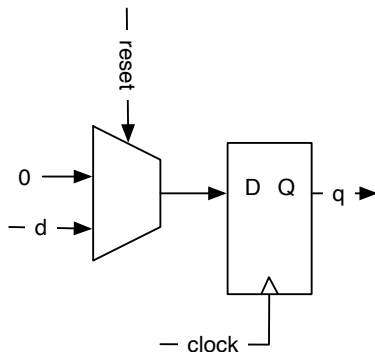


- ▶ A Multiplexer selects between alternatives
- ▶ So common that Chisel provides a construct for it
- ▶ Selects a when sel is true. B otherwise b

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

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



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
```


Reminder: We Construct Hardware

- ▶ Chisel code looks much like Java code
- ▶ But it is *not* a program in the usual sense
- ▶ It represents a circuit
- ▶ We should be able to *draw* that circuit
- ▶ The “program” constructs the circuit
- ▶ All statements are “executed” in parallel
- ▶ Statement order has mostly no meaning

Interlude

- ▶ Before we look at new material
- ▶ Sprinkle in some info on general development tools
- ▶ Get better at using your computer
- ▶ Learn some tools
- ▶ Don't be afraid of the command line ;-)
 - ▶ Show sbt usage
- ▶
- ▶
- ▶ Engineers are power users!

What is git?

- ▶ git is a distributed version-control system
 - ▶ What does that mean?
 - ▶ [Wikipedia on git](#)
- ▶ To manage source code or other documents
- ▶ Track changes in computer files
- ▶ Created by Linus Torvalds for Linux kernel development
- ▶ Good tool for cooperation
- ▶ Mostly used at the command line
- ▶ But graphical clients are available (i.e., with a GUI)
- ▶ Show the CLI commands

What is GitHub?

- ▶ **GitHub** is a git repository server
- ▶ GitHub is a classic startup, based in San Francisco
- ▶ Acquired 2018 by Microsoft for \$7.5 billion
- ▶ Many open-source projects are on GitHub (e.g., Chisel)
 - ▶ 85 million repositories, and 28 million developers
- ▶ Our DE2 material is hosted on GitHub
 - ▶ Lab material (you have used it)
 - ▶ The slides
 - ▶ The Chisel book
 - ▶ see <https://github.com/schoeberl>
 - ▶ Everyone can contribute via GitHub ;-)

Comment on Character Usage and Language

- ▶ Computers used for long time ASCII characters
- ▶ Show [ASCII table](#)
- ▶ Does NOT contain the special letters of DK, SE, AT,...
- ▶ Only a subset of ASCII was allowed for identifiers
- ▶ Languages such as Java or Scala are now more tolerant
 - ▶ You could use Chinese characters for your Java program!
- ▶ Please do not use any special characters
 - ▶ Also not in file names
- ▶ Programming is international
 - ▶ Use English identifiers and comments
- ▶ Avoid spaces in file names and folders

Coding Style

- ▶ Similar to Java
- ▶ Use readable, meaningful names
 - ▶ E.g., sum instead of y
- ▶ Use camelCase for identifiers
- ▶ Modules (classes) start with uppercase
 - ▶ E.g., VendingMachine
- ▶ Mark you register with a postfix Reg
 - ▶ E.g., countReg
- ▶ Use consistent indentation
 - ▶ Chisel style is 2 spaces (blanks)
- ▶ Use ASCII only ;-)

Combinational Circuits

- ▶ Simplest is a Boolean expression
- ▶ Assigned a name (e)
- ▶ This expression can be reused in another expression

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

Fixed Expression

- ▶ Expression is fixed
- ▶ Trying to reassign with = results in an error
- ▶ Trying the Chisel conditional update := results in runtime error

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

```
e := c & b
```


Combinational Circuit with Conditional Update

- ▶ Chisel supports conditional update
- ▶ Value first needs to be wrapped into a `Wire`
- ▶ Updates with the Chisel update operation `:=`
- ▶ With `when` we can express a conditional update
- ▶ The resulting circuit is a multiplexer
- ▶ The rule is that 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 “Else” Branch

- ▶ We can express a form of “else”
- ▶ Note the `.` in `.otherwise`

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

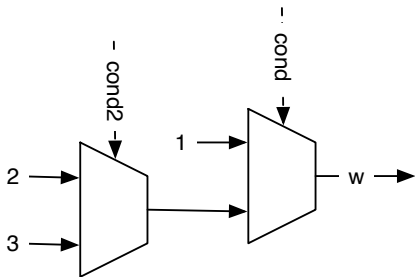
```
when (cond) {  
    w := 1.U  
} .otherwise {  
    w := 2.U  
}
```

A Chain of Conditions

- ▶ To test for different conditions
- ▶ Select with a priority order
- ▶ The first that is true counts
- ▶ The hardware is a chain of multiplexers

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

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



Default Assignment

- ▶ Practical for complex expressions
- ▶ Forgetting to assign a value on all conditions
 - ▶ Would describe a latch
 - ▶ Runtime error in Chisel
- ▶ Assign a default value is good practise

```
val w = WireDefault(0.U)
```

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

```
// ... and some more complex conditional  
    assignments
```

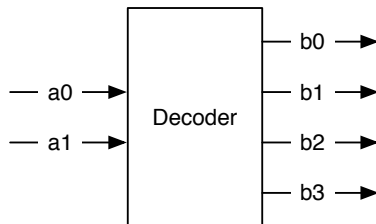
Logic Can Be Expressed as a Table

- ▶ Sometimes more convenient
- ▶ Still combinational logic (gates)
- ▶ Is converted to Boolean expressions
- ▶ Let the synthesize tool do the conversion!
- ▶ We use the `switch` statement

```
result := 0.U
```

```
switch(sel) {  
  is (0.U) { result := 1.U}  
  is (1.U) { result := 2.U}  
  is (2.U) { result := 4.U}  
  is (3.U) { result := 8.U}  
}
```

A Decoder



- ▶ Converts a binary number of n bits to an m -bit signal, where $m \leq 2^n$
- ▶ The output is one-hot encoded (exactly one bit is one)
- ▶ Building block for a m -way Mux
- ▶ Used for address decoding in a computer system

Truth Table of a Decoder

a	b
00	0001
01	0010
10	0100
11	1000

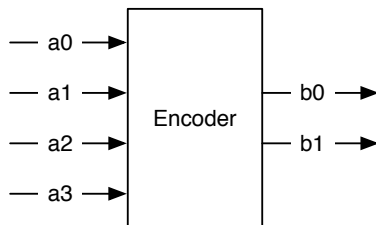
- Does this look like the table we have seen?

Decoder in Chisel

- Binary strings are a clearer representation

```
switch (sel) {  
  is ("b00".U) { result := "b00001".U}  
  is ("b01".U) { result := "b0010".U}  
  is ("b10".U) { result := "b0100".U}  
  is ("b11".U) { result := "b1000".U}  
}
```


An Encoder



- ▶ Converts one-hot encoded signal
- ▶ To binary representation

Truth Table of an Encoder

a	b
0001	00
0010	01
0100	10
1000	11
????	??

- Only defined for one-hot input

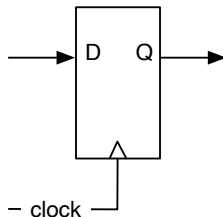
Encoder in Chisel

- ▶ We cannot describe a function with undefined outputs
- ▶ We use a default assignment of "b00"

```
b := "b00".U
switch (a) {
  is ("b0001".U) { b := "b00".U}
  is ("b0010".U) { b := "b01".U}
  is ("b0100".U) { b := "b10".U}
  is ("b1000".U) { b := "b11".U}
}
```

Register (Again)

- ▶ Sequential building blocks
 - ▶ Contains a register
 - ▶ Plus combinational circuits



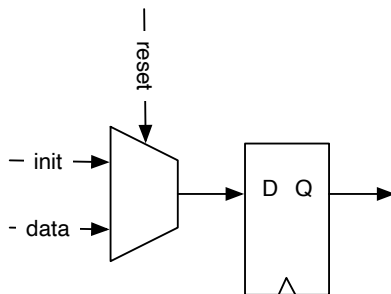
```
val q = RegNext(d)
```

Register in Two Steps

```
val delayReg = Reg(UInt(4.W))
```

```
delayReg := delayIn
```

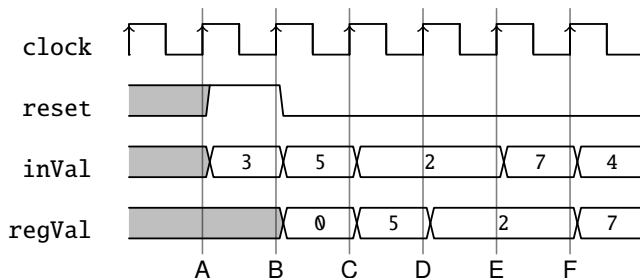
Register With Reset



```
val valReg = RegInit(0.U(4.W))
```

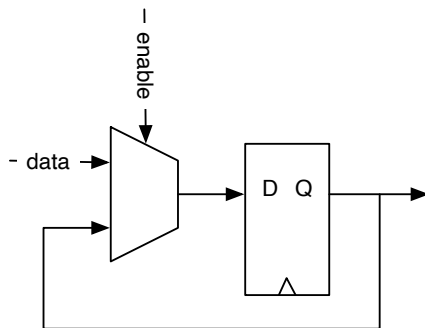
```
valReg := inVal
```

Timing Diagram of the Register with Reset



- ▶ Also called waveform diagram
- ▶ Logic function over time
- ▶ Can be used to describe a circuit function
- ▶ Useful for debugging

Register with Enable

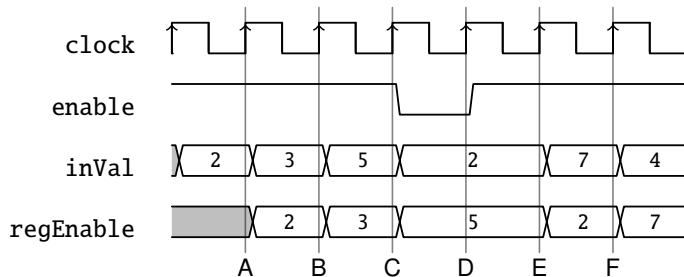


- Only when enable true is a value is stored

```
val enableReg = Reg(UInt(4.W))
```

```
when (enable) {  
  enableReg := inVal  
}
```


Timing Diagram for an Enable Register



More on Register

- ▶ We can combine initialization and enable

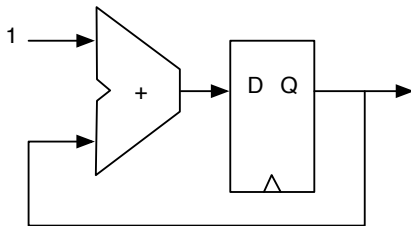
```
val resetEnableReg = RegInit(0.U(4.W))
```

```
when (enable) {  
    resetEnableReg := inVal  
}
```

- ▶ A register can also be part of an expression
- ▶ What does the following circuit do?

```
val risingEdge = din & !RegNext(din)
```

Combine a Register with an Adder



- ▶ Is a free running counter
- ▶ 0, 1, ... 14, 15, 0, 1, ...

```
val cntReg = RegInit(0.U(4.W))
```

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

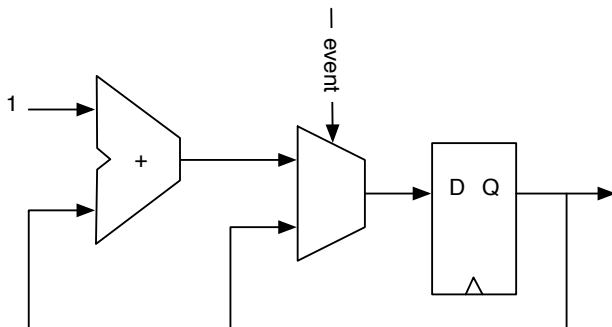
A Counter

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

```
cntReg := Mux(cntReg === 9.U, 0.U, cntReg + 1.U)
```

- ▶ This counter counts from 0 to 9
- ▶ And starts from 0 again after reaching 9
 - ▶ Starting from 0 is common in computer engineering
- ▶ A counter is the hardware version of a *for loop*
 - ▶ But runs forever (over and over again)
- ▶ Often needed
- ▶ Can we draw the schematic?

Counting Events



- ▶ The Chisel book has a slightly different figure

```
val cntEventsReg = RegInit(0.U(4.W))
when(event) {
  cntEventsReg := cntEventsReg + 1.U
}
```

Structure With Bundles

- ▶ A Bundle to groups signals
- ▶ Can be different types
- ▶ Defined by a class that extends Bundle
- ▶ List the fields as vals within the block

```
class Channel() extends Bundle {  
    val data = UInt(32.W)  
    val valid = Bool()  
}
```

Using a Bundle

- ▶ Create it with `new`
- ▶ Wrap it into a `Wire`
- ▶ Field access with *dot* notation

```
val ch = Wire(new Channel())  
ch.data := 123.U  
ch.valid := true.B  
  
val b = ch.valid
```

A Collection of Signals with Vec

- ▶ Chisel Vec is a collection of signals of the same type
- ▶ The collection can be accessed by an index
- ▶ Similar to an array in other languages

```
val v = Wire(Vec(3, UInt(4.W)))
```


Using a Vec

```
v(0) := 1.U
```

```
v(1) := 3.U
```

```
v(2) := 5.U
```

```
val idx = 1.U(2.W)
```

```
val a = v(idx)
```

- ▶ Reading from an Vec is a multiplexer
- ▶ We can put a Vec into a Reg

```
val registerFile = Reg(Vec(32, UInt(32.W)))
```

An element of that register file is accessed with an index and used as a normal register.

```
registerFile(idx) := dIn
```

```
val dOut = registerFile(idx)
```

Mixing Vecs and Bundles

- ▶ We can freely mix bundles and vectors
- ▶ When creating a vector with a bundle type, we need to pass a prototype for the vector fields. Using our `Channel`, which we defined above, we can create a vector of channels with:

```
val vecBundle = Wire(Vec(8, new Channel()))
```

- ▶ A bundle may as well contain a vector

```
class BundleVec extends Bundle {  
    val field = UInt(8.W)  
    val vector = Vec(4, UInt(8.W))  
}
```

Lab Today

- ▶ Combinational circuits in Chisel
- ▶ [Lab 2 Page](#)
- ▶ Each exercise contains a test, which initially fails
- ▶ `sbt test` runs them all
 - ▶ To just run a single test, run e.g.,
`sbt "testOnly MajorityPrinter"`

When all test succeed your are done ;-)

- ▶ Components contain a comment where you shall add your implementation
- ▶ The initial majority example has an optional implementation in an FPGA

Summary

- ▶ Think in hardware
 - ▶ Draw “boxes”
- ▶ Combinational logic (= Boolean function)
 - ▶ Logical and arithmetic expressions
 - ▶ Conditional update (*when*)
 - ▶ Function tables with *switch*
 - ▶ Large multiplexer with a *Vec*
- ▶ Registers
 - ▶ Define as *Reg*, *RegNext*, or *RegInit*

Summary

- ▶ We looked at basic digital circuit blocks
- ▶ Now you know all you need to build any digital circuit!
 - ▶ Digital controller
 - ▶ MP3 player
 - ▶ Microprocessor
 - ▶ Data center accelerator
 - ▶ ...
- ▶ Will show you some constructs for a more *elegant* style

Questions and Feedback

- ▶ Let us have a quick round in the breakout rooms