Chisel: Combinational Circuits

Muhammad Tahir

Lecture 2

Contents

Supported Operations

- All supported operations are implemented as Combinational Circuits
- Can be performed with single or bundle of wires (with few exceptions)
- Operator precedence is determined by the evaluation order of the circuit (and follows Scala operator precedence)

Chisel Hardware Operations

Table: Different groups of hardware operations.

Operator Symbol	Description	Operand Type
&& || !	AND, OR, NOT (logical)	Bool
& | ~ ^	AND, OR, NOT, XOR (bitwise)	UInt, SInt, Bool
<< >>	shift left, shift right (sign extend for SInt)	UInt, SInt
+ −	addition, subtraction	UInt, SInt
* / %	multiplication, division, modulus	UInt, SInt
=== =/=	equal, not equal (returns Bool)	UInt, SInt
> >= < <=	different comparisons (returns Bool)	UInt, SInt

Chisel Hardware Operations Cont'd

More on arithmetic operations

```
// Arithmetic operations

// Addition without width expansion
val sum = x + y // OR
val sum = x +% y

// Addition with width expansion
val sum = x +& y

// Subtraction without width expansion
val sum = x - y // OR
val sum = x -% y

// Subtraction with width expansion
val sum = x -& y
```

Chisel Hardware Operations Cont'd

Bitfield manipulations and reductions

```
// Bitfield manipulations
val xMSB = x(31)          // when x is 32-bit
val yLowByte = y(7, 0)    // y is atleat 8-bit

// concatenates bitfields with first operand on left
val address = Cat(highByte, lowByte)

// replicate a string multiple times
val duplicate = Fill(2, "b1010".U) // "b10101010".U

// Bitfield reductions
val data = "b00111010".U
val allOnes = data.andR              // performs AND reduction
val anyOne = data.orR               // performs OR reduction
val parityCheck = data.xorR         // performs XOR reduction
```

Width Inference for Hardware Operations

Rules to infer width at the output of hardware block

Table: Bit Width Inference*.

Operation	Bit Width
out = in1 + in2	W(out) = max{W(in1), W(in2)}
out = in1 +& in2	W(out) = max{W(in1), W(in2)} + 1
out = in1 & in2	W(out) = max{W(in1), W(in2)}
out = in1 * in2	W(out) = W(in1) + W(in2)
out = in1 $\ll$ shift	W(out) = W(in1) + max(shift)
out = in1 $\gg$ shift	W(out) = W(in1) - min(shift)
out = Cat(in1 , in2)	W(out) = W(in1) + W(in2)

* where W(x) is the bit width of signal or wire x.

Mux Block

One of the most widely used hardware blocks

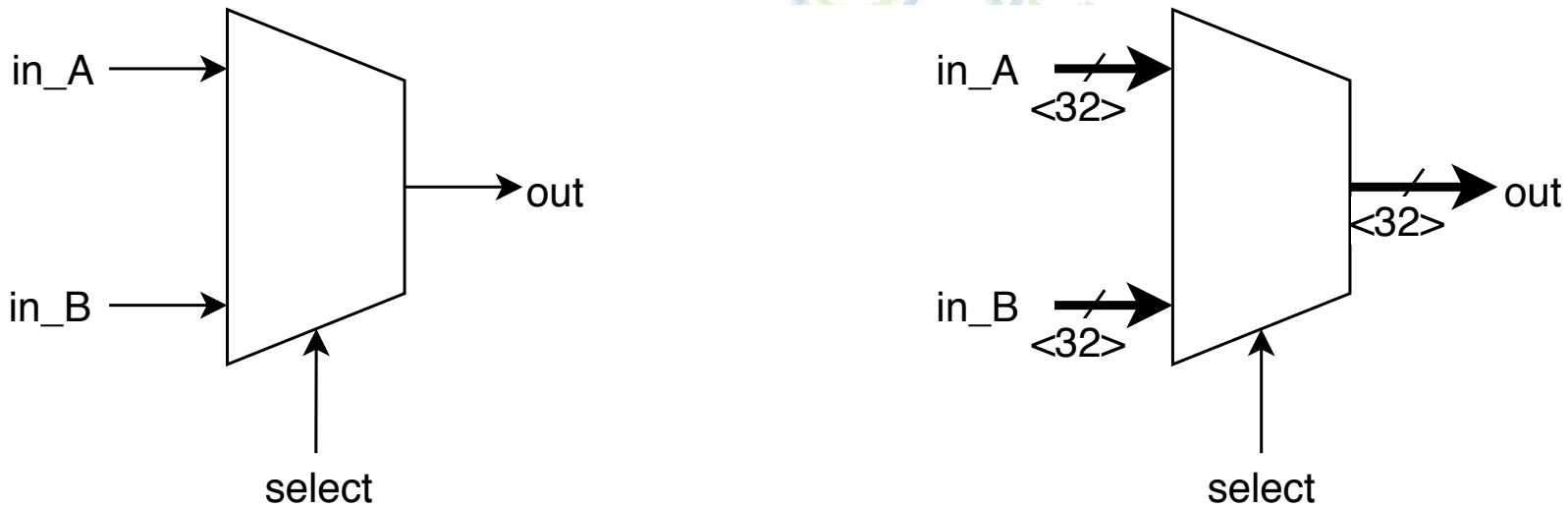


Figure: A simple 2 to 1 Mux.

Mux Block Cont'd

Mux 2 to 1 (scalar (Bool) inputs)

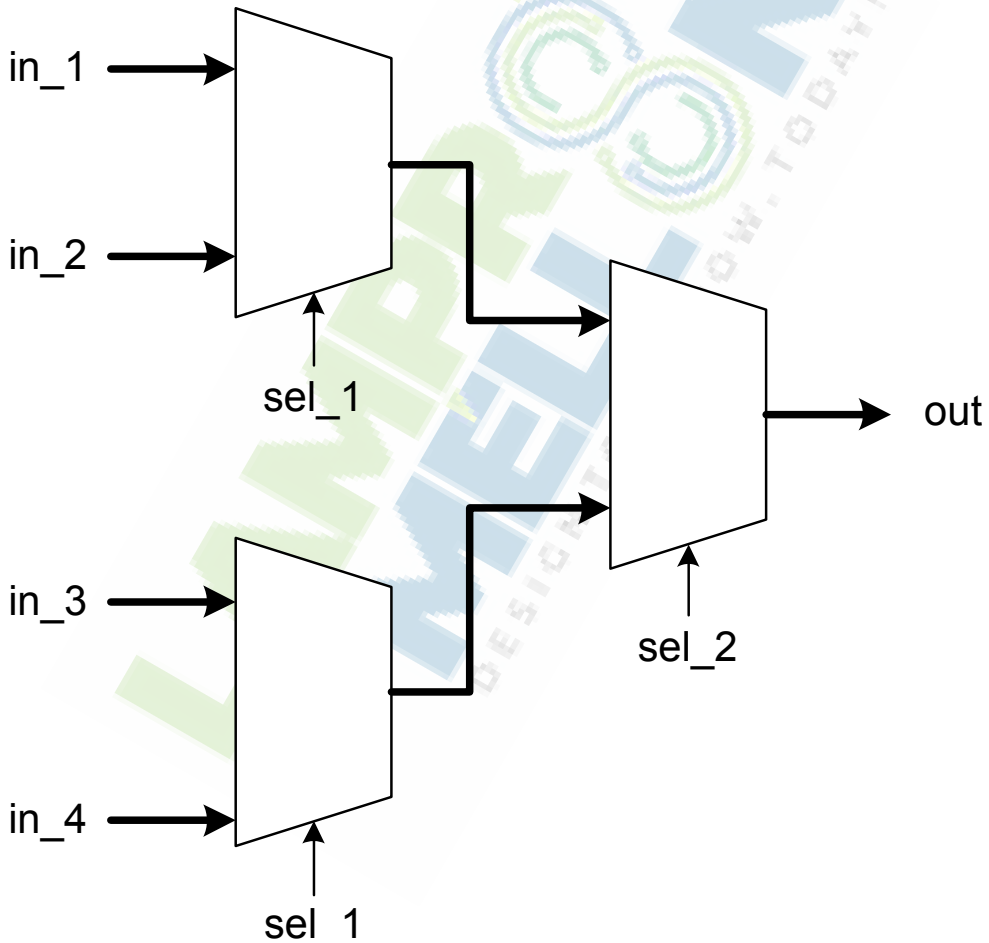
```
import chisel3._

// Mux IO interface class
class Mux_2to1_IO extends Bundle {
      val in_A   = Input(Bool())
      val in_B   = Input(Bool())
      val select = Input(Bool())
      val out    = Output(Bool())
}
// 2 to 1 Mux implementation
class Mux_2to1 extends Module {
      val io = IO(new Mux_2to1_IO)

      // update the output
      io.out := io.in_A & io.select | io.
        in_B & (~io.select)
}
println((new chisel3.stage.ChiselStage).
      emitVerilog(new Mux_2to1()))
```

Mux 2 to 1 (vector inputs)

```
import chisel3._

// Mux IO interface class
class Mux_2to1_IO extends Bundle {
      val in_A   = Input(UInt(32.W))
      val in_B   = Input(UInt(32.W))
      val select = Input(Bool())
      val out    = Output(UInt())
}
// 2 to 1 Mux implementation
class Mux_2to1 extends Module {
      val io = IO(new Mux2to1_IO)

      // update the output
      io.out := Mux(io.select, io.in_A, io.
        in_B)
}
println((new chisel3.stage.ChiselStage).
      emitVerilog(new Mux_2to1()))
```

Mux Block: Mux4to1

Mux 4 to 1 with 2 selection lines



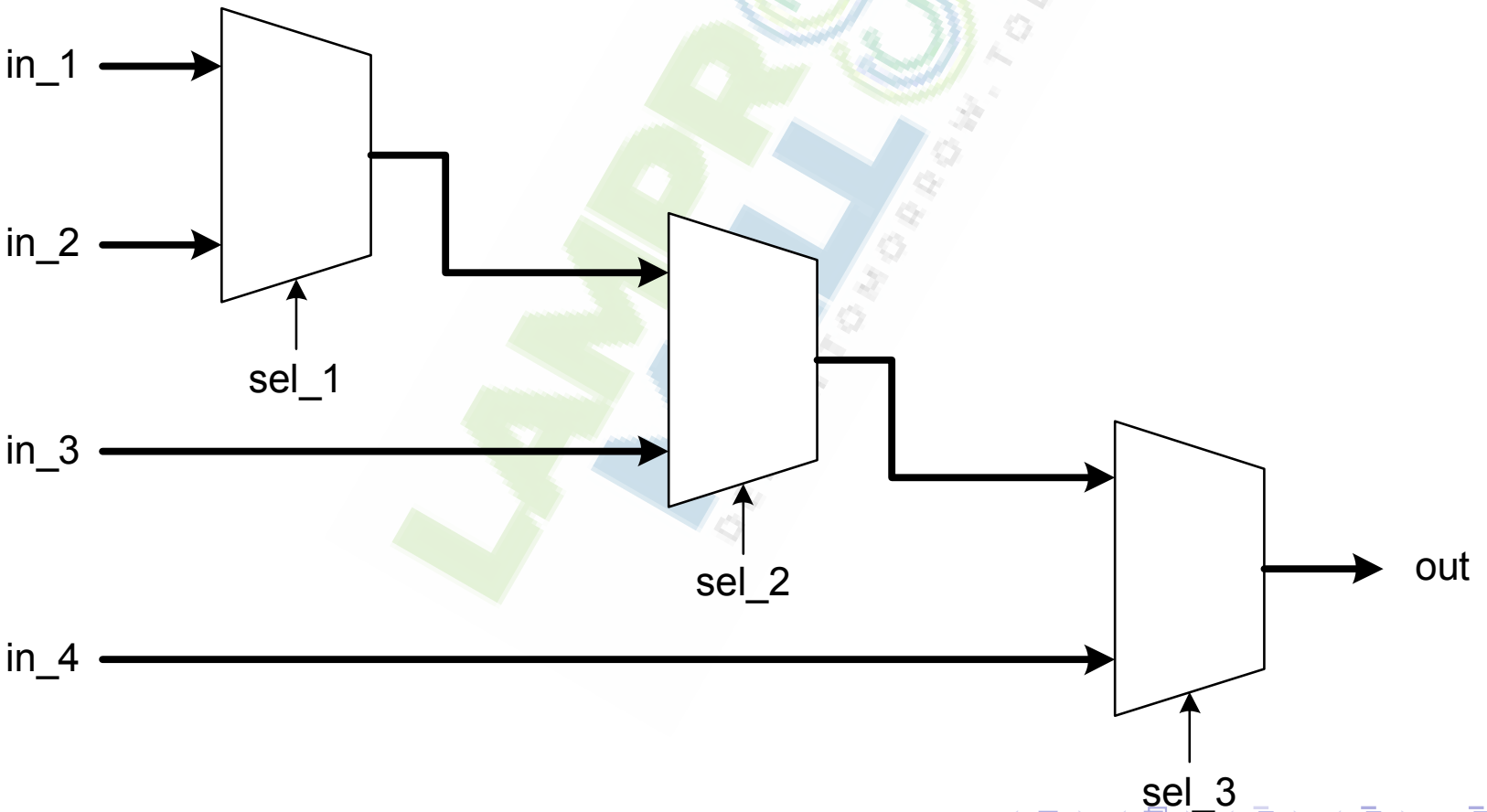
Mux Block: Mux4to1

Mux 4 to 1 implementation

```
// Mux4to1 example
import chisel3._

class IO_Interface extends Bundle {
    val in  = Input(UInt(4.W))
    val s1  = Input(Bool())
    val s2  = Input(Bool())
    val out = Output(Bool())     // UInt(1.W))
}

class Mux_4to1 extends Module {
    val io = IO(new IO_Interface)

    io.out := Mux(io.s2, Mux(io.s1, io.in(3), io.in(2)),
    Mux(io.s1, io.in(1), io.in(0)))
}

println((new chisel3.stage.ChiselStage).emitVerilog(new
    Mux_4to1()))
```

Mux Tree: Mux4to1

- Mux 4 to 1 with 3 selection lines
- Available as PriorityMux from utilities



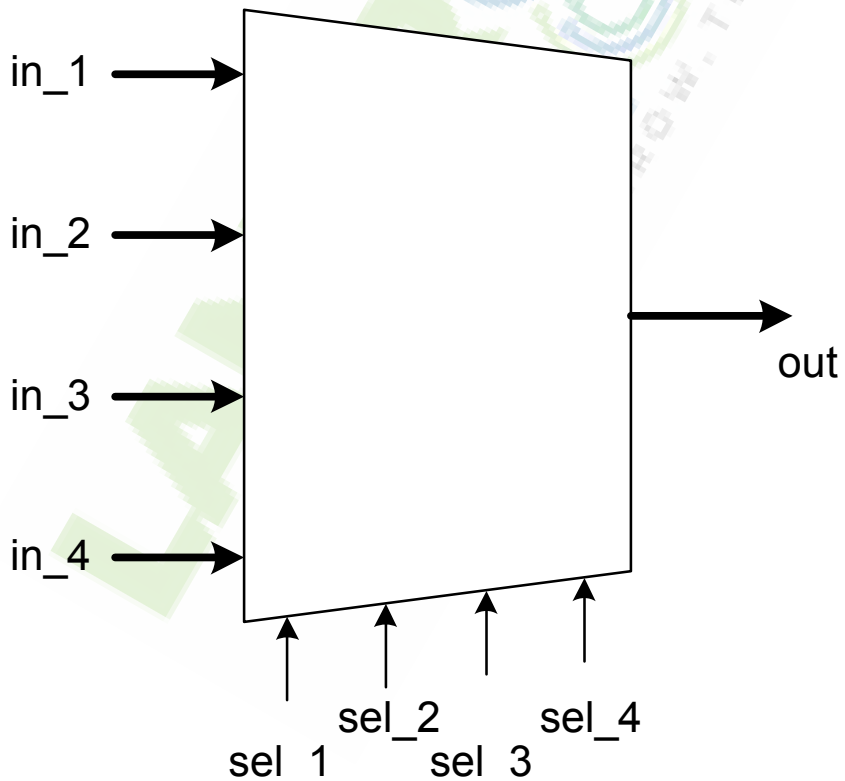
Mux Tree: Mux4to1

Mux 4 to 1 using priority implementation

```
// Mux with input priority
import chisel3._

class IO_Interface extends Bundle {
    val in  = Input(UInt(4.W))
    val s1  = Input(Bool())
    val s2  = Input(Bool())
    val s3  = Input(Bool())
    val out = Output(Bool())     // UInt(1.W))
}

class Mux_Tree extends Module {
    val io = IO(new IO_Interface)

    io.out := Mux(io.s3, io.in(3), Mux(io.s2, io.in(2),
    Mux(io.s1, io.in(1), io.in(0))))
}
println((new chisel3.stage.ChiselStage).emitVerilog(new
    Mux_Tree()))
```

Mux1H

- 4 to 1 Mux with 4 selection lines
- Available as Mux1H from utilities



MuxCase

```
// 4 to 1 mux using MuxCase
import chisel3._
import chisel3.util._

class MuxCase_Example extends Module {
      val io = IO(new Bundle{
            val in0 = Input(Bool())
            val in1 = Input(Bool())
            val in2 = Input(Bool())
            val in3 = Input(Bool())
            val sel = Input(UInt(2.W))
            val out = Output(Bool())
      })

      io.out := MuxCase(false.B, Array(
      (io.sel === 0.U)  ->      io.in0,
      (io.sel === 1.U)  ->      io.in1,
      (io.sel === 2.U)  ->      io.in2,
      (io.sel === 3.U)  ->      io.in3
      )
      )
}

println((new chisel3.stage.ChiselStage).emitVerilog(new MuxCase_Example ()))
```

MuxLookup

MuxLookup is similar to MuxCase with slightly different syntax

```
// MuxLookup example
val raw_data = io.dmem.rdata
val load_data = MuxLookup(ld_type, 0.U,
                          Seq(LD_LW  -> raw_data.zext,
                              LD_LH  -> raw_data(15, 0).asSInt,
                              LD_LB  -> raw_data(7, 0).asSInt,
                              LD_LHU -> raw_data(15, 0).zext,
                              LD_LBU -> raw_data(7, 0).zext))
```

Bundles, Ports and Vec

```
//Example using Asynchronous memory
import chisel3._
import chisel3.util._

class IO_Interface extends Bundle{
      // input using a Vector of 4 values
      val data_in = Input(Vec(4,(UInt(32.W))))

      val data_selector = Input(UInt(2.W))   // vector slection lines
      val data_out = Output(UInt(32.W))      // output data
      val addr = Input(UInt(5.W))            // address lines
      val wr_en = Input(Bool())              // high for write
}

class Mem_bundle_intf extends Module {
      val io = IO(new IO_Interface)
      val memory = Mem(32, UInt(32.W))       // Make memory of 32x32

      io.data_out := 0.U

      when(io.wr_en) {
            // Write at addr, with selected data from data_in vector
            memory.write(io.addr, io.data_in(io.data_selector))
      } .otherwise {
            // Asyncronous read from addr location
            io.data_out := memory.read(io.addr)
      }
}

println((new chisel3.stage.ChiselStage).emitVerilog(new Mem_bundle_intf()))
```

Flipped and Bulk Connection

```
import chisel3._
import chisel3.util._

class Interface extends Bundle {
      val in_data  = Input(UInt(6.W))
      val valid    = Output(Bool())
      val out_data = Output(UInt(6.W))
}

class MS_interface extends Bundle {
      val s2m  = Input(UInt(6.W))
      val m2s  = Output(UInt(6.W))
}

class Top_module extends Module {
      val io = IO(new Interface)

      val master = Module(new Master)
      val slave  = Module(new Slave)
      //connecting top with master => same direction, same name connects
      io <> master.io.top_int
      //connecting master with slave => opposite direction, same name connects
      master.io.MS <> slave.io
}
```

Flipped and Bulk Connection Cont'd

```
class Master extends Module {
    val io = IO(new Bundle {
      val top_int = new Interface
      val MS = new MS_interface
    })

   io.MS.m2s := io.top_int.in_data
   io.top_int.valid := true.B
   io.top_int.out_data := io.MS.s2m
}

class Slave extends Module {
    val io = IO(Flipped(new MS_interface))

  io.s2m := io.m2s + 16.U
}

println(chisel3.Driver.emitVerilog(new Top_module))
```

Reading List

- Read the relevant sections from Chapters 2 and 4 of [Schoeberl, 2019]
- Also consult the [chisel3, 2020] for further details

References

chisel3 (2020).

Chisel3 library reference.

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

Schoeberl, M. (2019).

Digital Design with Chisel.

Kindle Direct Publishing.