

# Interfacing and Memory

Martin Schoeberl

Technical University of Denmark  
Embedded Systems Engineering

April 18, 2024

# Overview

- ▶ Repeat FSMD (for the vending machine)
  - ▶ I have (earlier) seen some intermix of FSM and datapath
  - ▶ Works only for small designs
- ▶ Quick reminder on FSMD
- ▶ Interfaces
- ▶ Memory (intern and extern)
  - ▶ You've asked for it during the midterm eval
- ▶ Busses
- ▶ The course evaluation should be open for feedback
- ▶ Lab is the vending machine

# Notes

- ▶ Luca's course
- ▶ Course evaluation is open
- ▶ Did you do a [Tiny Tapeout](#)?
  - ▶ Next deadline is 1st June

## I have seen similar code (not this year)

```
when(io.coin2) {  
  sum := sum + 2.U  
} .elsewhen(io.coin5) {  
  sum := sum + 5.U  
}
```

- ▶ This may work for tiny projects
- ▶ This style does not scale for larger designs
- ▶ I recommend using an FSM + datapath
  - ▶ Split responsibility
  - ▶ Can be individually tested
  - ▶ Can be developed in parallel

# Usage of an FSM

- ▶ Of course for your VM
  - ▶ The VM is a simple processor
  - ▶ But not Turing complete
  - ▶ Can *only* process coins of 2 and 5
- ▶ Have the FSM and the data path in two Modules
- ▶ An FSM can be used to build a processor
- ▶ Fine for simple processors
- ▶ E.g., [Lipsi](#)
- ▶ Pipelined processor topic of
  - ▶ Computer Architecture Engineering (02155)

# Test the FSM for the VM

- ▶ This is the main part your vending machine
- ▶ Can be design and tested just with Chisel testers (no FPGA board needed)
- ▶ See the given tester
  - ▶ Sets the price to 7
  - ▶ Adds two coins (2 and 5)
  - ▶ Presses the buy button
- ▶ Extend the test along the development
  - ▶ Remember test driven development?
  - ▶ Maybe test developer and FSM developer are not always the same person

# Testing Your Vending Machine

- ▶ Write (unit) tests for each component/module
- ▶ E.g., one for the data path, one for the state machine
- ▶ Then one (integration) test for the top-level component/module
- ▶ Maybe do some **extreme programming (XP)**
  - ▶ Write the test first
  - ▶ Then the code of the component
  - ▶ The test can serve as a specification
  - ▶ We can do **agile** hardware development

# Memory

- ▶ Registers are storage elements == memory
- ▶ Just use a Reg of a Vec
- ▶ This is 1 KiB of memory

```
val memoryReg = Reg(Vec(1024, UInt(8.W)))  
// writing into memory  
memoryReg(wrAddr) := wrData  
// reading from the memory  
val rdData = memoryReg(rdAddr)
```

- ▶ Simple, right?
- ▶ But is this a good solution?



# A Flip-Flop

- ▶ Remember the circuit of a register (flip-flop)?
- ▶ Two latches: **master and slave**
- ▶ One (enable) latch can be built with 4 NAND gates
- ▶ a NAND gate needs 6 transistors, an inverter 2 transistors
- ▶ A flip-flop needs 20 transistors (for a single bit)
- ▶ Can we do better?

# A Memory Cell

- ▶ A single bit can be stored in 6 transistors
- ▶ That is how larger memories are built
- ▶ FPGAs have this type of on-chip memories
- ▶ Usually many of them in units of 2 KiB or 4 KiB
- ▶ We need some Chisel code to represent it
- ▶ More memory needs an external chip
- ▶ Then we need to interface this memory from the FPGA

# SRAM Memory

- ▶ RAM stands for random access memory
- ▶ SRAM stands for static RAM
- ▶ There is also something called DRAM for dynamic RAM
  - ▶ Uses a capacitor and a transistor
  - ▶ DRAM is smaller than SRAM
  - ▶ But needs refreshes
  - ▶ Different technology than technology for logic
- ▶ All on-chip memory is SRAM (today)

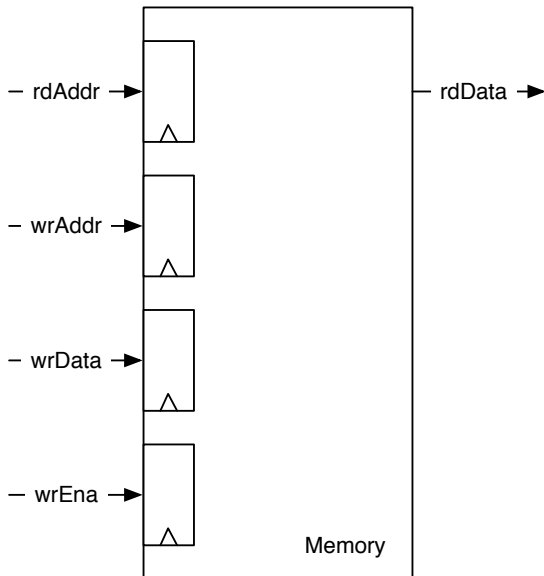
# Memory Interface

- ▶ Interface
  - ▶ Address input (e.g., 10 bits for 1 KiB)
  - ▶ Write signal (e.g., we)
  - ▶ Data input
  - ▶ Data output
- ▶ May share pins for the data input and output (tri-state)
- ▶ May have read and write addresses
  - ▶ A so-called dual ported memory
  - ▶ Can do a read and a write in the same clock cycle

# On-Chip Memory

- ▶ SRAM by itself is asynchronous
- ▶ No clock, just the correct timing
- ▶ Apply the address and after some time the data is valid
- ▶ But one can add input registers, which makes it a synchronous SRAM
- ▶ Current FPGAs have synchronous memories only
- ▶ This means the result of a read is available one clock cycle after the address is given
  - ▶ This is different from the use of flip-flops (`Reg(Vec(. .))`)
- ▶ FPGAs usually have dual-ported memories

# Synchronous Memory



## Use of a Chisel SyncReadMem

```
class Memory() extends Module {  
  val io = IO(new Bundle {  
    val rdAddr = Input(UInt(10.W))  
    val rdData = Output(UInt(8.W))  
    val wrAddr = Input(UInt(10.W))  
    val wrData = Input(UInt(8.W))  
    val wrEna = Input(Bool())  
  })  
  
  val mem = SyncReadMem(1024, UInt(8.W))  
  
  io.rdData := mem.read(io.rdAddr)  
  
  when(io.wrEna) {  
    mem.write(io.wrAddr, io.wrData)  
  }  
}
```

# Read-During-Write

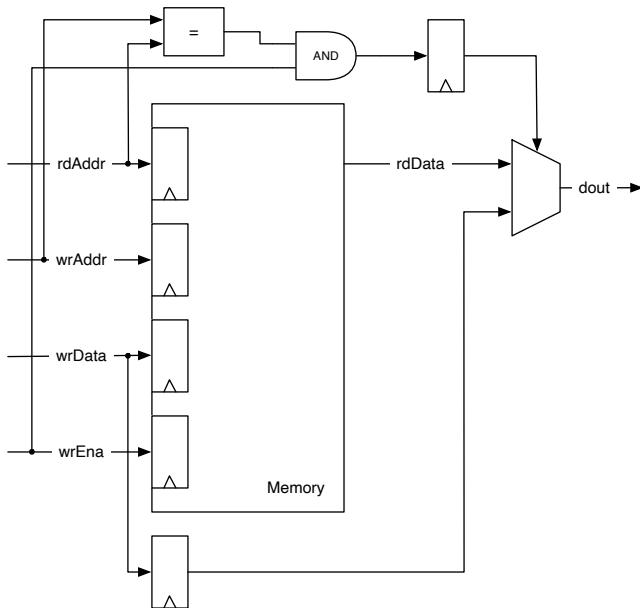
- ▶ What happens when one writes to and reads from the same address?
- ▶ Which value is returned?
- ▶ Three possibilities:
  1. The newly written value
  2. The old value
  3. Undefined (mix of old and new)
- ▶ Depends on technology, FPGA family, ...
- ▶ We want to have a defined read-during-write
- ▶ We add hardware to *forward* the written value



# Condition for Forwarding

- ▶ If read and write addresses are equal
- ▶ If write enable is true
- ▶ Multiplex the output to take the new write value instead of the (old) read value
- ▶ Delay that forwarded write value to have the same timing

# Memory with Forwarding



## Forwarding in Chisel

```
val mem = SyncReadMem(1024, UInt(8.W))

val wrDataReg = RegNext(io.wrData)
val doForwardReg = RegNext(io.wrAddr ===
    io.rdAddr && io.wrEna)

val memData = mem.read(io.rdAddr)

when(io.wrEna) {
    mem.write(io.wrAddr, io.wrData)
}

io.rdData := Mux(doForwardReg, wrDataReg,
    memData)
```

# External Memory

- ▶ On-chip memory is limited
- ▶ We can add an external memory chip
  - ▶ Is cheaper than FPGA on-chip memory
- ▶ Sadly the Basys 3 board has no external memory
- ▶ Simple memory is an asynchronous SRAM

# External SRAM

- ▶ We *buy* a CY7C1041CV33
- ▶ Let us look into the [data sheet](#)

# Interfacing the SRAM

- ▶ FPGA output drives address, control, and data (sometimes)
- ▶ FPGA reads data
- ▶ The read signal is asynchronous to the FPGA clock
- ▶ Do we need an input synchronizer?

# Synchronous Interface

- ▶ Logic is synchronous
- ▶ Memory is asynchronous
  - ▶ How to interface?
- ▶ Output signals
  - ▶ Generate timing with synchronous circuit
  - ▶ Small FSM
- ▶ Asynchronous input signal
  - ▶ Usually 2 register for input synchronization
  - ▶ Really needed for the SRAM interface?
  - ▶ We would lose 2 clock cycles

# SRAM Read

- ▶ Asynchronous timing definition (data sheet)
- ▶ But, we know the timing and we trigger the SRAM address from our synchronous design
- ▶ No need to use synchronization registers
- ▶ *Just* get the timing correct
- ▶ Draw the example
  - ▶ Address - SRAM - data
  - ▶ Relative to the FPGA clock



# Read Timing Continued

- ▶ Add all time delays
  - ▶ Within FPGA
  - ▶ Pad to pin
  - ▶ PCB traces
  - ▶ SRAM read timing
  - ▶ PCB traces back
  - ▶ Pin to pad
  - ▶ Into FPGA register
- ▶ Setup and hold time for FPGA register

# Connecting to the World

- ▶ Logic in the FPGA
  - ▶ Described in Chisel
  - ▶ Abstracting away electronic properties
- ▶ Interface to the world
  - ▶ Simple switches and LEDs
  - ▶ Did we think about timing?
- ▶ FPGA is one component of the system
- ▶ Need interconnect to
  - ▶ Write outputs
  - ▶ Read inputs
  - ▶ Connect to other chips

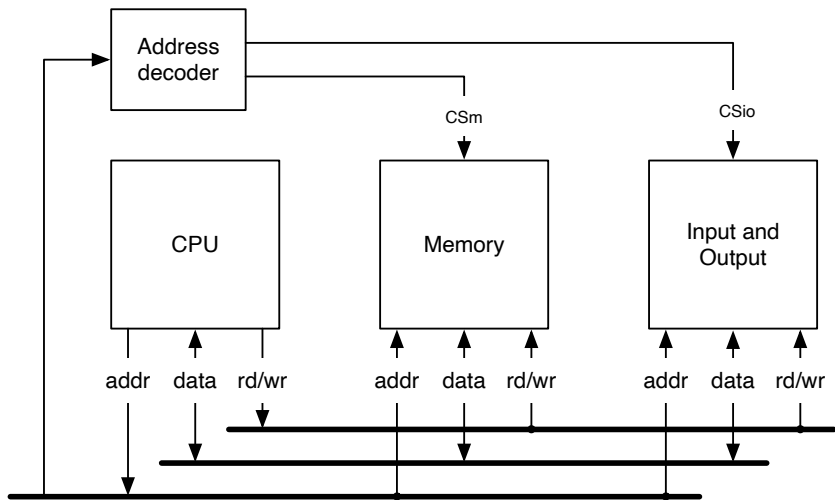
# Bus Interface

- ▶ Memory interface can be generalized
- ▶ We use a so-called bus to connect several devices
- ▶ Usually a microprocessor connected to devices (memory, IO)
- ▶ The microprocessor is the master
- ▶ A bus is an interface definition
  - ▶ Logic and timing
  - ▶ Electrical interface
- ▶ Parallel or serial data
- ▶ Asynchronous or synchronous
  - ▶ But interface clock is usually not the logic clock

# Bus Properties

- ▶ Address bus and data bus
- ▶ Control lines (read and write)
- ▶ Several devices connected
  - ▶ Multiple outputs
  - ▶ Use tri-state to avoid multiple driver
- ▶ Single or multiple master
  - ▶ Arbitration for multiple master

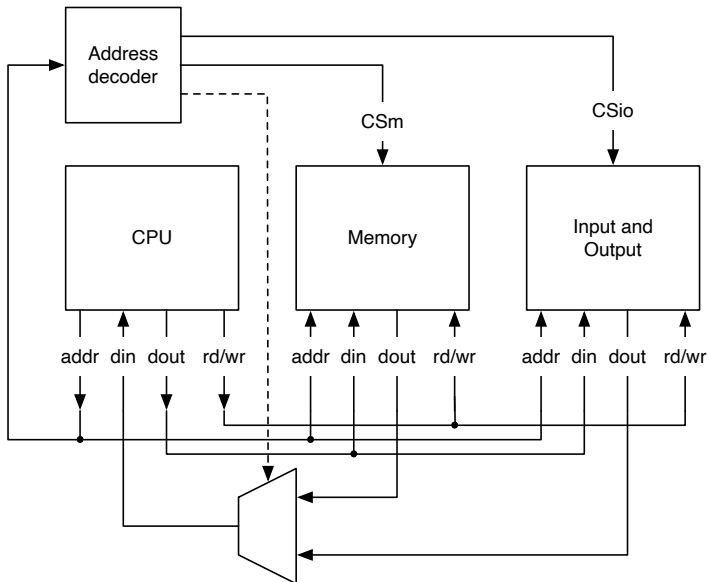
# A Classic Microprocessor Bus



# Does This Work On-Chip?

- ▶ Just mapping that bus on-chip?
- ▶ Tri-state not so easy
- ▶ Wires are cheap
- ▶ Use dedicated connections and a Mux

# An On-Chip Bus



# Serial I/O Interface

- ▶ Use only one wire for data transfer
  - ▶ Bits are serialized
  - ▶ That is where you need your shift register
- ▶ Shared wire or dedicated wires for transmit and receive
- ▶ Self timed
  - ▶ Serial UART (RS 232)
  - ▶ Ethernet
  - ▶ USB
- ▶ With a clock signal
  - ▶ SPI, I2C, ...

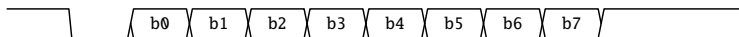


# RS 232

- ▶ Old, but still common interface standard
  - ▶ Was common in 90' in PCs
  - ▶ Now substituted by USB
  - ▶ Still common in embedded systems
  - ▶ Your Basys 3 board has a RS 232 interface
- ▶ Standard defines
  - ▶ Electrical characteristics
  - ▶ '1' is negative voltage (-15 to -3 V)
  - ▶ '0' is positive voltage (+3 to +15 V)
  - ▶ Converted by a RS 232 driver to *normal* logic voltage levels

# Serial Transmission

- ▶ Transmission consists of
  - ▶ Start bit (low)
  - ▶ 8 data bits
  - ▶ Stop bit(s) (high)
- ▶ Common baud rate is 115200 bits/s



# RS 232 Interface

- ▶ How would we implement this?
- ▶ Baud rate is 115200 bit/s (ca. 10 us)
- ▶ Clock on Basys 3 is 100 MHz (10 ns)
- ▶ Output (transmit)
- ▶ Input (receive)
- ▶ Let us do it now together

# RS 232 Interface

- ▶ Generate bit clock with with counter
  - ▶ Like clock tick generation for display multiplexer
- ▶ Output (transmit)
  - ▶ Use shift register for parallel to serial conversion
  - ▶ Small FSM to generate start bit, data bits, and stop bits
- ▶ Input (receive)
  - ▶ Detect start with the falling edge of the start bit
  - ▶ *Position* into middle of start bit
  - ▶ Sample individual bits
  - ▶ Serial to parallel conversion with a shift register

## Chisel Code for RS 232

- ▶ More explanation can be found in section 11.2
- ▶ The code is in the Chisel book
- ▶ [uart.scala](#)
- ▶ Also see example usage in [chisel-examples](#) repo

## RS 232 on the Basys 3

- ▶ Basys 3 has an FTDI chip for the USB interface
- ▶ USB interface for FPAG programming
- ▶ But also provides a RS 232 to the FPGA
- ▶ You can talk with your laptop
- ▶ You have used it two weeks ago

## Today's and next Lab

- ▶ Work on your Vending Machine
- ▶ As usual, show and discuss with a TA
- ▶ Get a tick from a TA when done. This is VERY important!
- ▶ Add features and show again to the TA
- ▶ Next week: guest lecture from Peter Jensen (SyoSil) on verification

# Summary

- ▶ Use an FSMD for the vending machine and simple processors
- ▶ We need to connect to the world
- ▶ FPGA (or any chip) is only part of a system
- ▶ Bus interface to external devices (e.g., memory)
- ▶ Serial interface to connect systems
  - ▶ E.g., your Basys 3 board to the laptop
- ▶ Testing is more than looking at waveforms