Interfacing and Memory

Martin Schoeberl

Technical University of Denmark
Embedded Systems Engineering

February 20, 2021

Overview

- Repeat FSMD (for the vending machine)
- Interfaces
- Memory (intern and extern)
- Serial interface (RS 232)
- 2 hours lab SRAM exercise
  - Exercise description is in DTU Inside
  - sram_exercise.pdf
  - SRAM data sheet: CYCC1041...
- The course evaluation is open for feedback

The Online Exam

- We will use the “old” online system
- Show it at http://onlineeksamen.dtu.dk/
- It will be two parts:
  - Multiple choice
  - Download and hand in a document (e.g., PDF generated from Word)
- You can play with the very short one today
- We will do a test exam next week, before the lab

Using an FSM and a Datapath

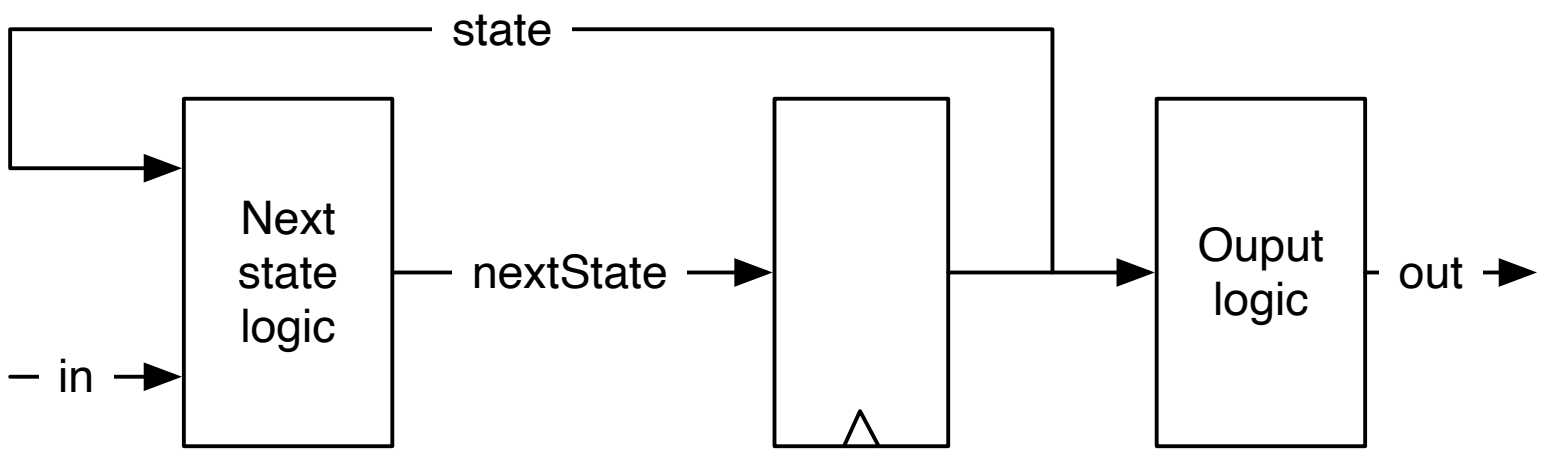
- About the design of the vending machine (VM)
- Some of you start coding the VM directly
  - This may work for small designs
  - But does not scale
- Better use a systematic approach
- Use a FSM that communicates with a datapath (FSMD)
- We will quickly repeat FSMD

Finite-State Machine (FSM)

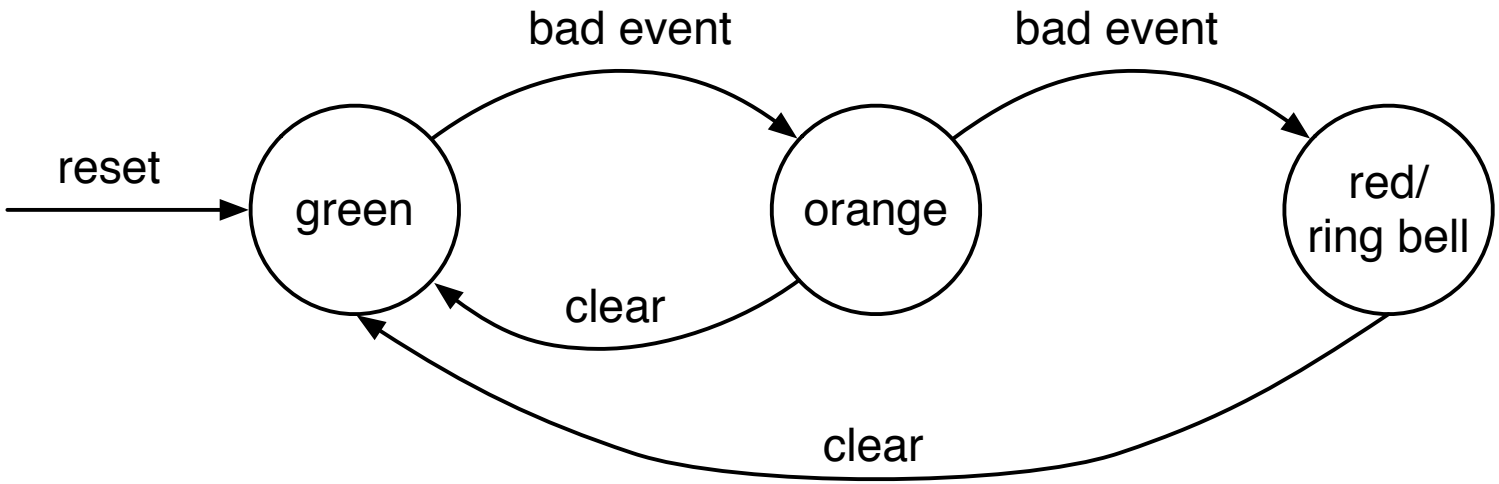
- Has a register that contains the state
- Has a function to computer the next state
  - Depending on current state and input
- Has an output depending on the state
- Use a Moore FSM for the VM

Basic Finite-State Machine

- A state register
- Two combinational blocks



State Diagram



- States and transitions depending on input values
- Example is a simple alarm FSM
- Nice visualization
- Draw the state diagram for your VM during the design
- Include a state diagram in the report

The Input and Output of the Alarm FSM

- Two inputs and one output

```
val io = IO(new Bundle{
  val badEvent = Input(Bool())
  val clear = Input(Bool())
  val ringBell = Output(Bool())
})
```

Encoding the State

- We can optimize state encoding
- Two common encodings are: binary and one-hot
- We leave it to the synthesize tool
- Use symbolic names with an `Enum`
- Note the number of states in the `Enum` construct
- We use a Scala list with the `::` operator

```
val green :: orange :: red :: Nil = Enum(3)
```

Start the FSM

- We have a starting state on reset

```
val stateReg = RegInit(green)
```

The Next State Logic

```
switch (stateReg) {
  is (green) {
    when(io.badEvent) {
      stateReg := orange
    }
  }
  is (orange) {
    when(io.badEvent) {
      stateReg := red
    } .elsewhen(io.clear) {
      stateReg := green
    }
  }
  is (red) {
    when (io.clear) {
      stateReg := green
    }
  }
}
```

The Output Logic

```
io.ringBell := stateReg === red
```

Summary on the Alarm Example

- Three elements:
  1. State register
  2. Next state logic
  3. Output logic
- This was a so-called Moore FSM
- There is also an FSM type called Mealy machine

FSM with Datapath

- A type of computing machine
- Consists of a finite-state machine (FSM) and a datapath
- The FSM is the master (the controller) of the datapath
- The datapath has computing elements
  - E.g., adder, incrementer, constants, multiplexers, ...
- The datapath has storage elements (registers)
  - E.g., sum of money payed, count of something, ...
- You VM design shall be a FSMD

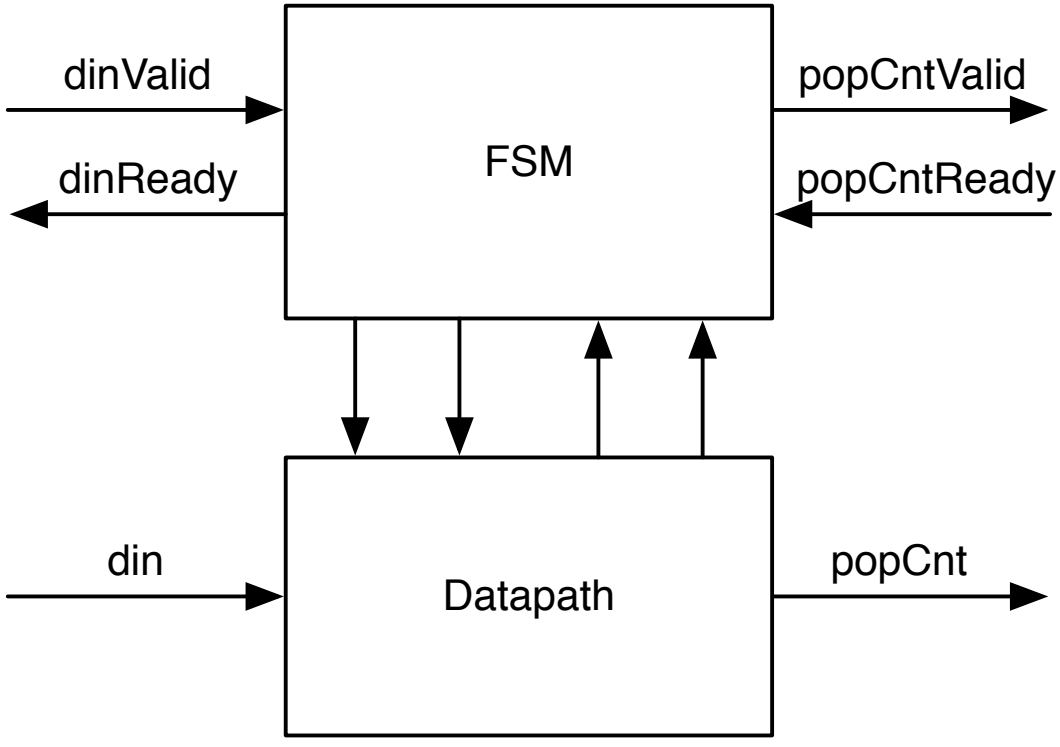
FSM-Datapath Interaction

- The FSM controls the datapath
  - For example, add 2 to the sum
- By controlling multiplexers
  - For example, select how much to add
  - Not adding means selecting 0 to add
- Which value goes where
- The FSM logic also depends on datapath output
  - Is there enough money payed to release a can of soda?
- FSM and datapath interact

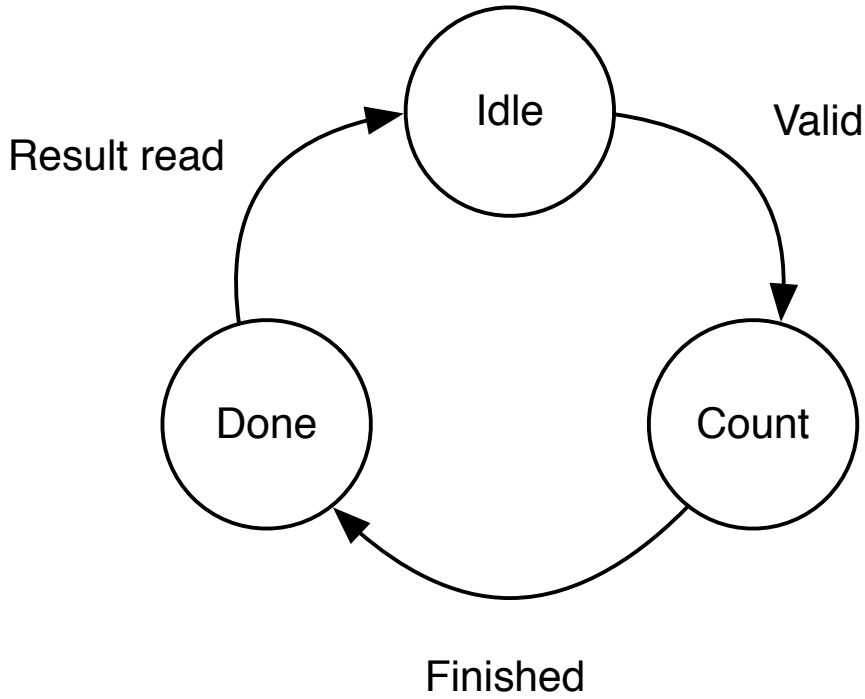
Popcount Example

- An FSMD that computes the popcount
- Also called the Hamming weight
- Compute the number of ‘1’s in a word
- Input is the data word
- Output is the count

Popcount Block Diagram

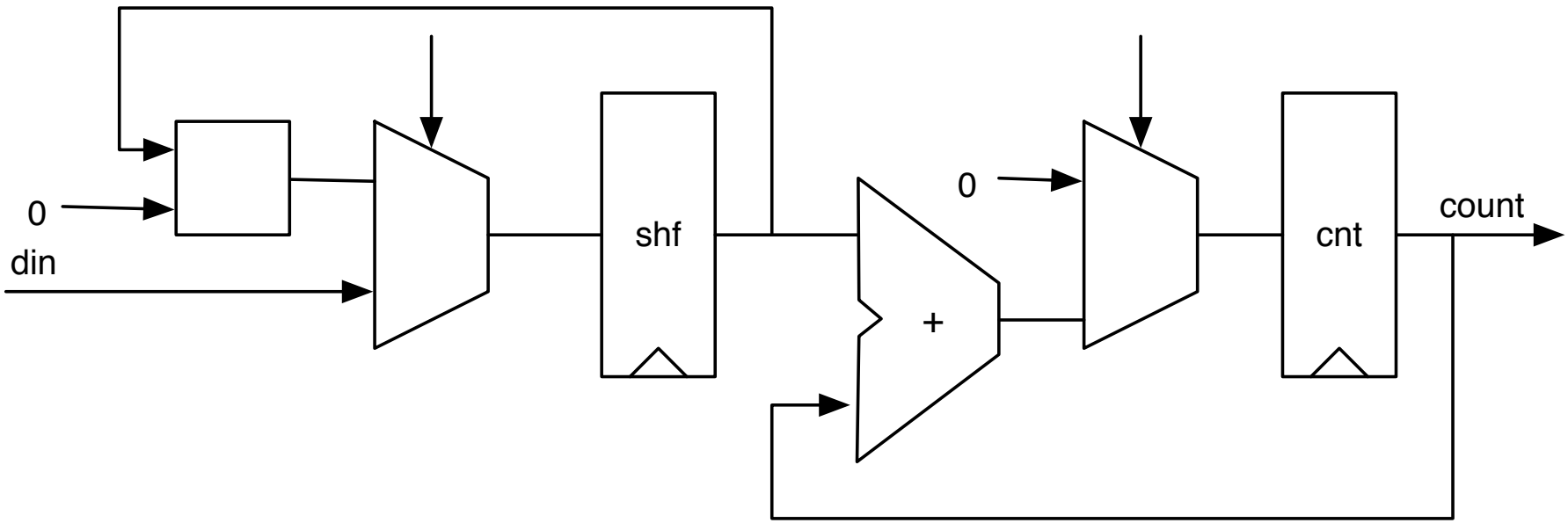


The FSM



- A Very Simple FSM
- Two transitions depend on input/output handshake
- One transition on the datapath output

The Datapath



Let's Explore the Code

- In PopCount.scala

Usage of an FSMD

- Of course for your VM
  - The VM is a simple processor
  - But not Turing complete
  - Can *only* process coins of 2 and 5
- An FSMD can be used to build a processor
- Fine for simple processors
- E.g., Lipsi
- Pipelined processor topic of
  - Computer Architecture Engineering (02155)

Use a FSMD 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 write the test before the implementation
  - Maybe test developer and FSMD developer are not always the same person

Code Snippets for the VM

```
val idle :: add2 ... :: Nil = Enum(?)
val stateReg = RegInit(idle)
...
switch (stateReg) {
  is (idle){
    when(coin2) {
      stateReg := ...
    }
    when(...) {
 ...
 switch(stateReg) {
   is (add2) { ... }  // drive the datapath for
       adding a coin of kr. 2
```

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 extra 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)

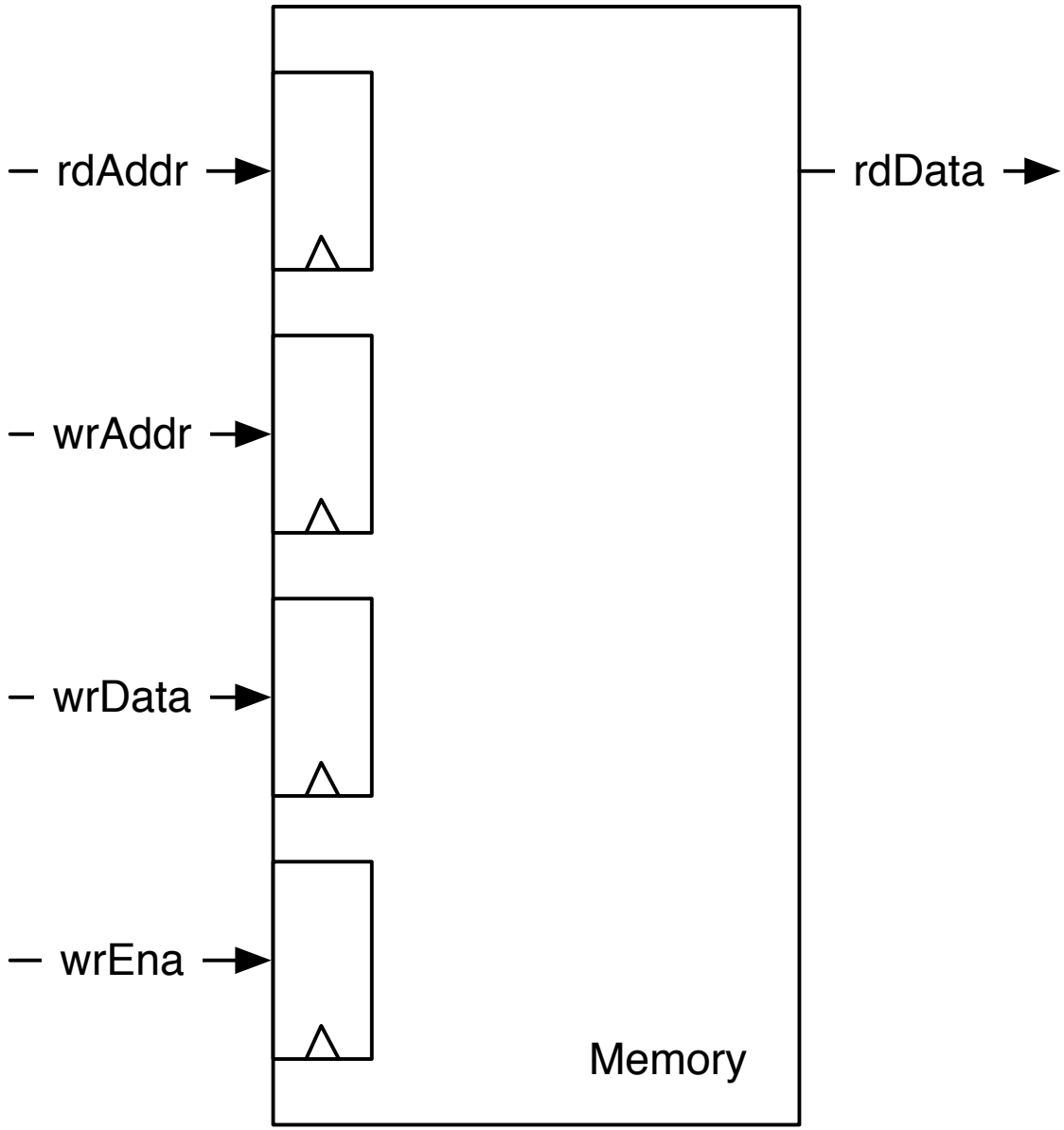
Memory Interface

- 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 only synchronous memories
- FPGAs usually have dual-ported memories
- This means the result of a read is available on clock cycle after the address is given
  - This is different from the use of flip-flops (`Reg(Vec(..))`)

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 wrEna = Input(Bool())
    val wrData = Input(UInt(8.W))
    val wrAddr = Input(UInt(10.W))
  })

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

  io.rdData := mem.read(io.rdAddr)

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

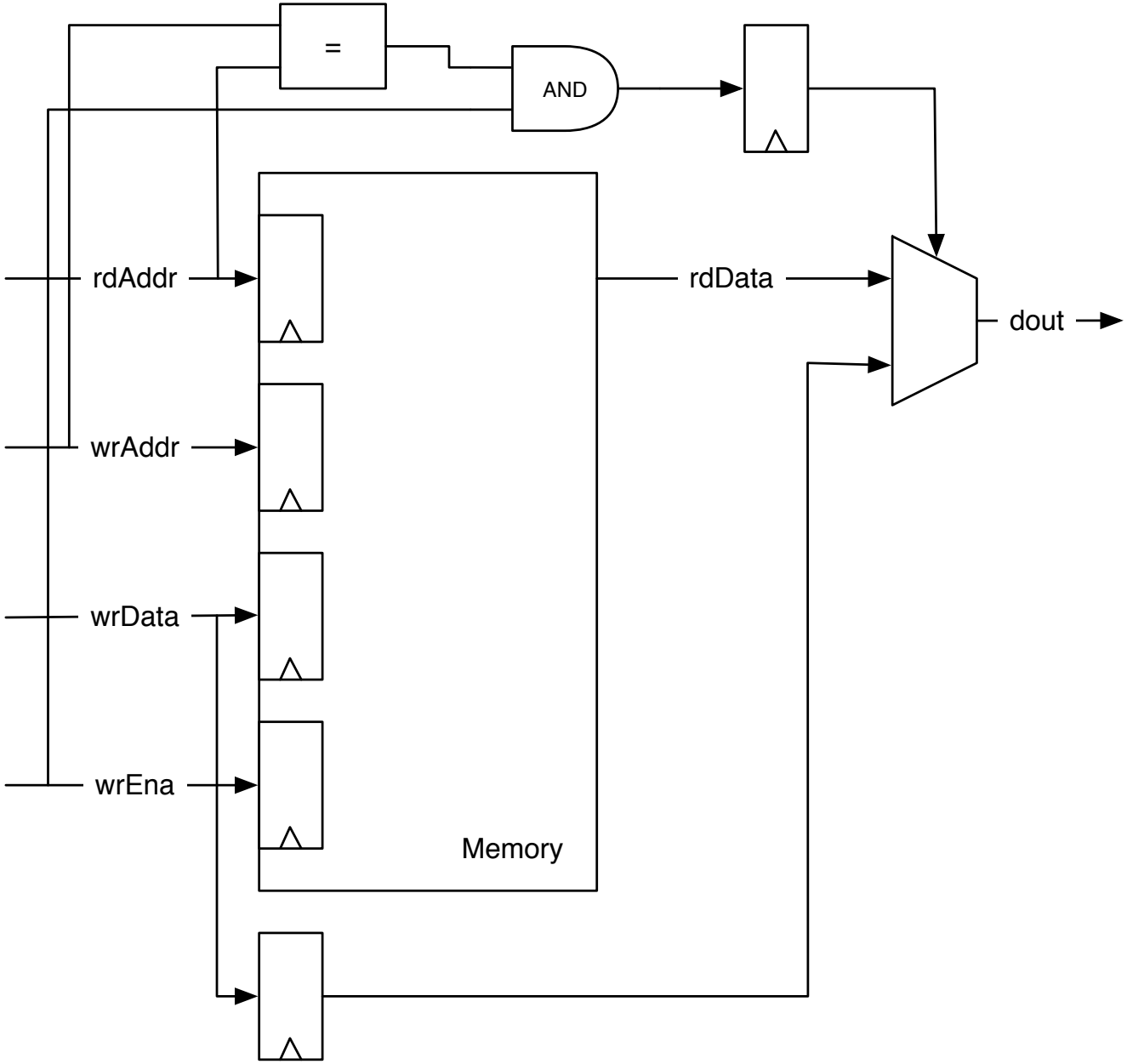
Read-During-Write

- 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 Basys3 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 synchroous
- Memory is asynchronous
  - How to interface?
- Output signals
  - Generate timing with synchronous circuit
  - Small FSM
- Asynchronous input signale
  - Usually 2 register for input synchronization
  - Really needed for the SRAM interface?
  - We would loose 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
- Dra 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
- Is your today's lab exercise

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
- Sketch a small microprocessor system

Serial I/O Interface

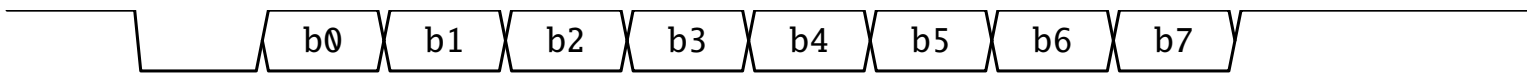
- Use only one wire for data transfer
  - Bits are serialized
  - That is where you need your shift register
- Shared wire or dedicated wires for transmit and receive
- Self timed
  - Serial UART (RS 232)
  - Ethernet
  - USB
- With a clock signal
  - SPI, I2C, ...

RS 232

- Old, but still common interface standard
  - Was common in 90' in PCs
  - Now substituted by USB
  - Still common in embedded systems
  - Your Basys3 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
  - 8 data bits
  - Stop bit(s)
- Common baud rate 115200 bits/s



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 Basys3

- Basys3 has an FTDI chip for the USB interface
- USB interface for FPAG programming
- But also to provide a RS 232 to the FPGA
- You can talk with the Laptop
- Your VM could write out some text
- Use the Chisel code I showed you
- Open a terminal to watch (show it)

Today's Lab

- Paper & pencil exercises on SRAM interfacing
- On paper or in a plain text editor
- As usual, show and discuss with a TA
- Exercise description is in DTU Inside file sharing
- sram_exercise.pdf
- SRAM data sheet: CYCC1041...

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 Basys3 board to the laptop