



# ECE TUTORING

**DROP-IN  
TUTORING  
AVAILABLE  
FOR A SELECTION  
OF EE  
UNDERGRADUATE  
CLASSES!**

**SCAN QR CODE TO VIEW THE  
TUTORING SCHEDULE**



**NOW LOCATED IN ECE RM 443!**

**Take the elevator to floor 4R and  
follow signage to 443.**



# Design of Digital Circuits and Systems

## Algorithmic State Machines

**Instructor:** Vikram Iyer

**Teaching Assistants:**

Ariel Kao

Josh Wentzien

Selim Saridede

Jared Yoder

Derek Thorp

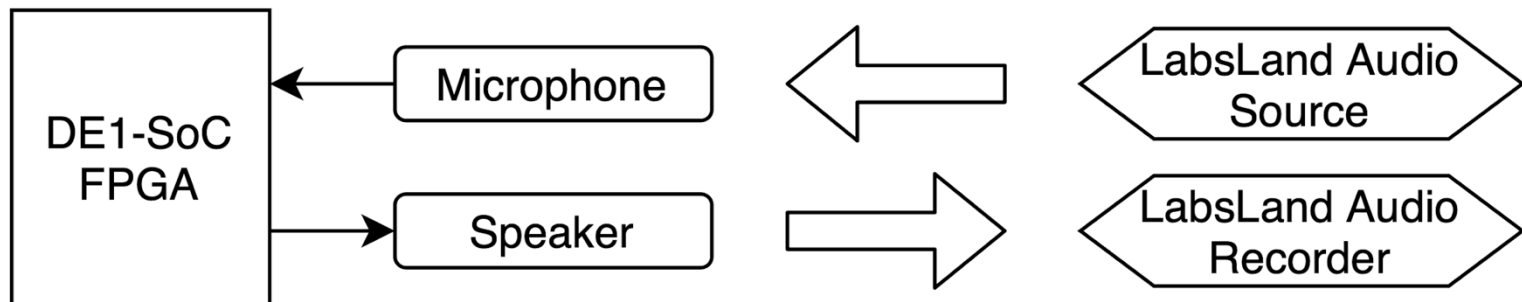
Adapted from material by Justin Hisa

# Relevant Course Information

- ❖ Homework 2 due Wednesday (4/16)
- ❖ Homework 3 released today, due next Friday (4/25)
  
- ❖ Lab 2 reports due 4/18, demos 4/21-25
- ❖ Lab 3 released today, due in two weeks (5/2)
  - Lab 3 + 4 are really ~1.5 weeks long, so don't wait!
  
- ❖ Quiz 2 not until *next* Thursday (4/24)
  - Spacing between material and quiz will get longer and longer; make sure to give time to review

# Lab 3 Notes

- ❖ More practical **applications of memory** on the DE1-SoC using **audio generation and filtering**
  - Task 2: ROM with MIF file to generate audio
  - Task 3: Use a FIFO buffer to implement a noise filter
- ❖ See *Audio\_Guide.pdf* in the spec for how to use the LabsLand Audio Interface to send audio input and record audio output:

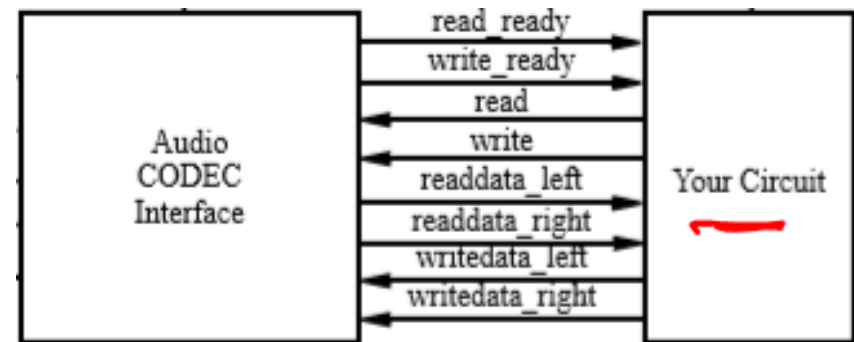


# Lab 3 Notes

- ❖ Example of *communication* as you interface with an audio CODEC (coder/decoder)

- Inputs: read,  
write,  
writedata\_left,  
writedata\_right

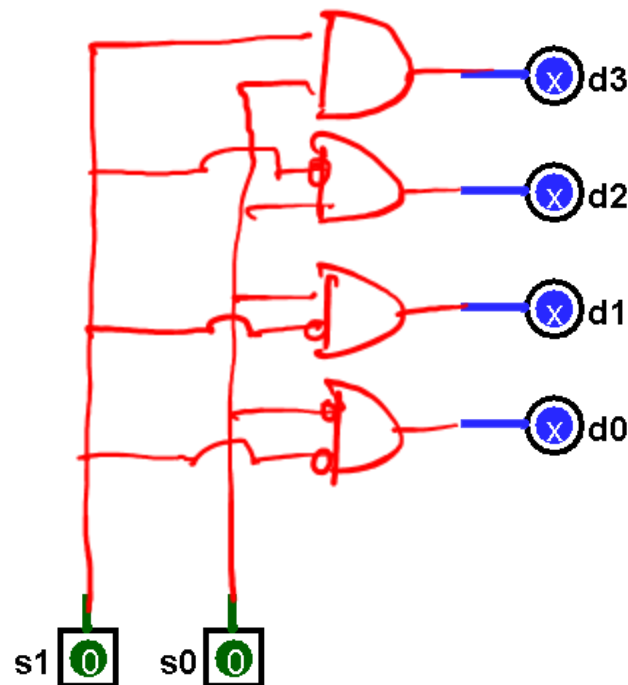
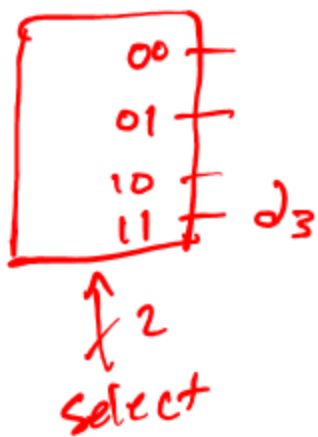
- Outputs: read\_ready,  
write\_ready,  
readdata\_left,  
readdata\_right



- Must wait for both sides (CODEC + your circuit) to be ready for data transmission in either direction!
  - Data is ready/generated and receiver is ready to accept

# Review Question: Decoder

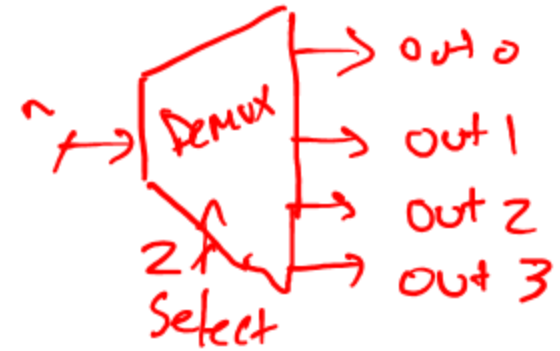
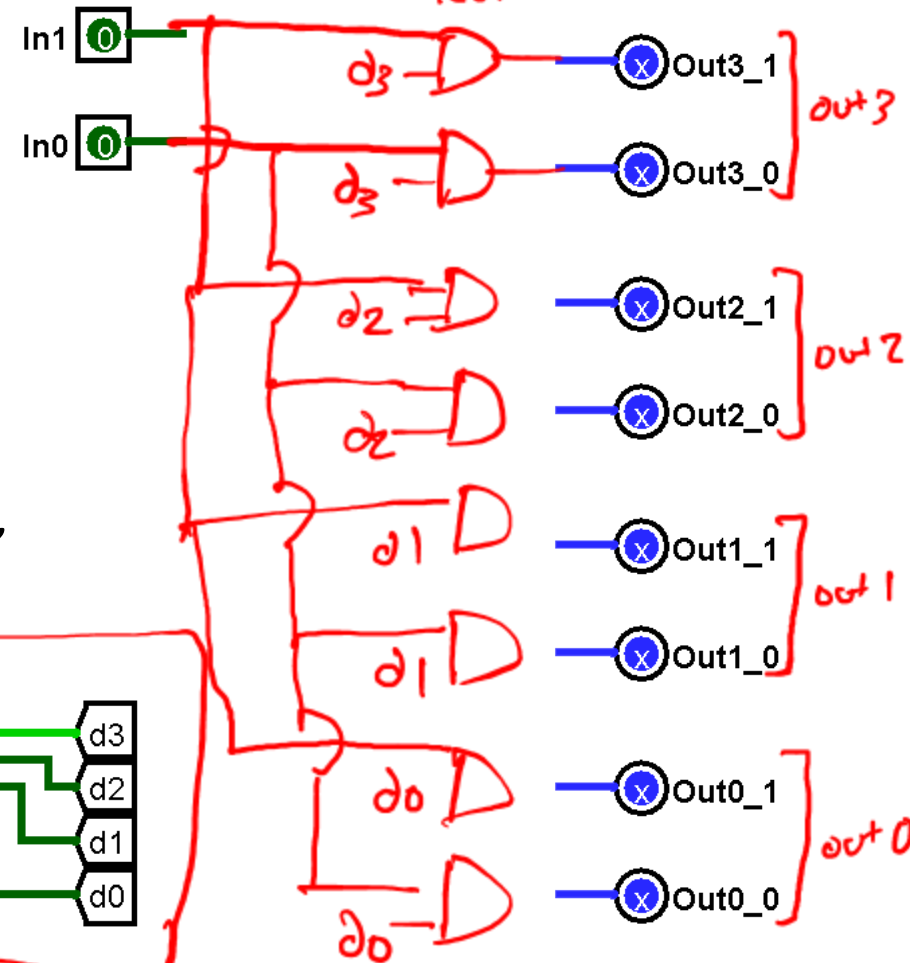
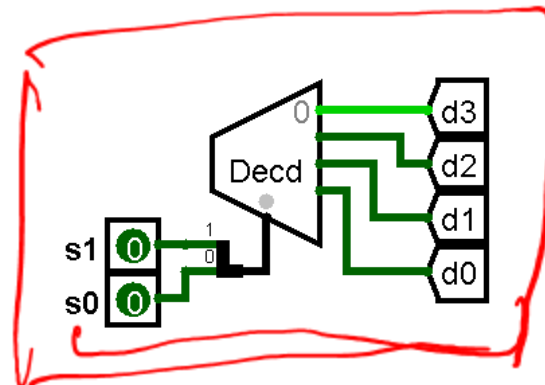
- ❖ 2:4 binary decoder has 2 select bits that specify which of 4 output bits is high (the others are low) – implement one below using only NOT, AND, and OR gates:



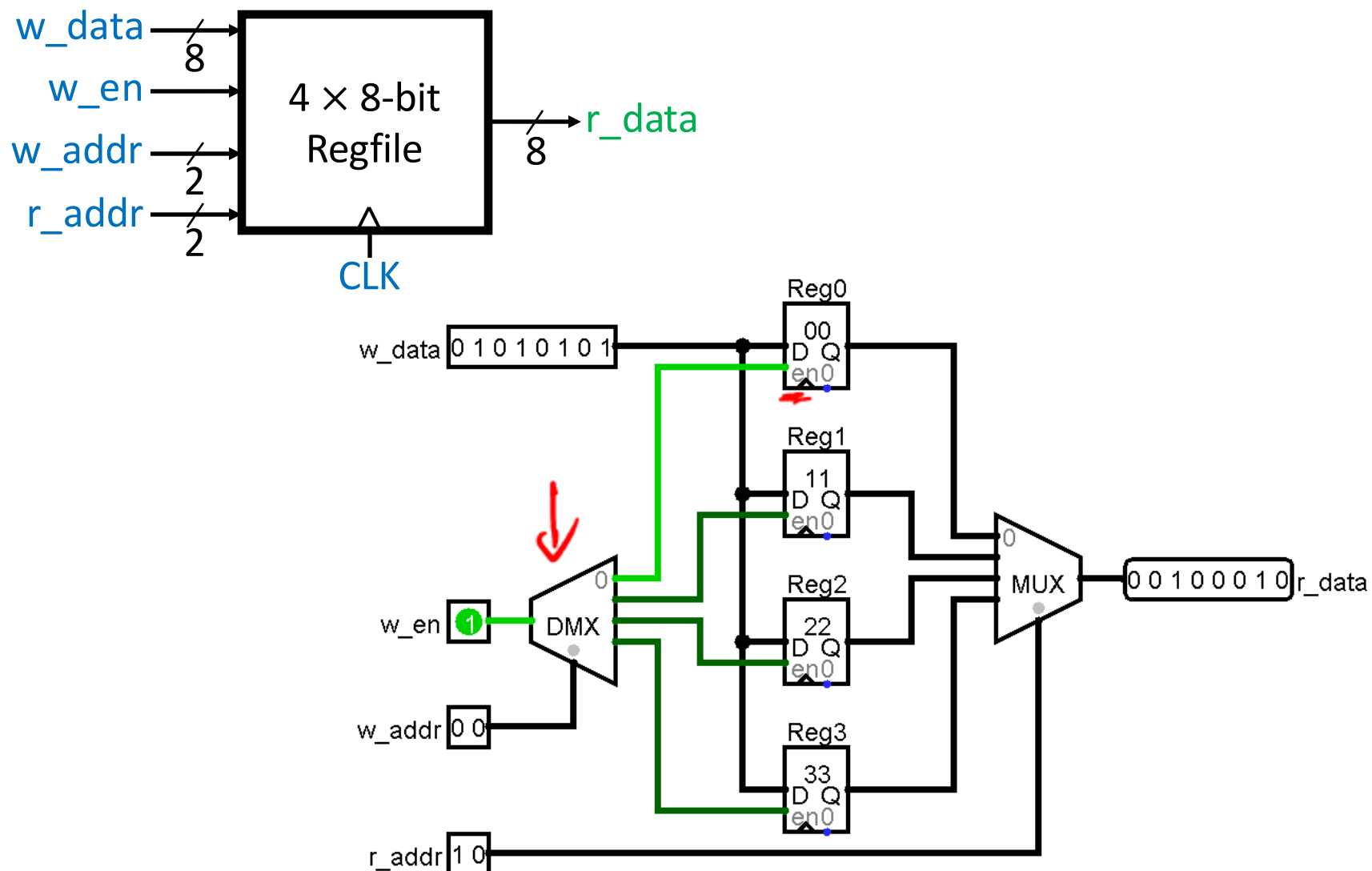
# Review Question: DEMUX

## ❖ Implement a 2-bit, 2-to-4 DEMUX:

- A DEMUX takes an input bus and connects to one of many output buses specified by selector bits
- Assume you have a working 2:4 binary decoder and write in the signals  $d_0$ ,  $d_1$ ,  $d_2$ , and  $d_3$  where needed.



# Simple Reg File uses DEMUX





# Specifying Synchronous Digital Systems

## ❖ So far:

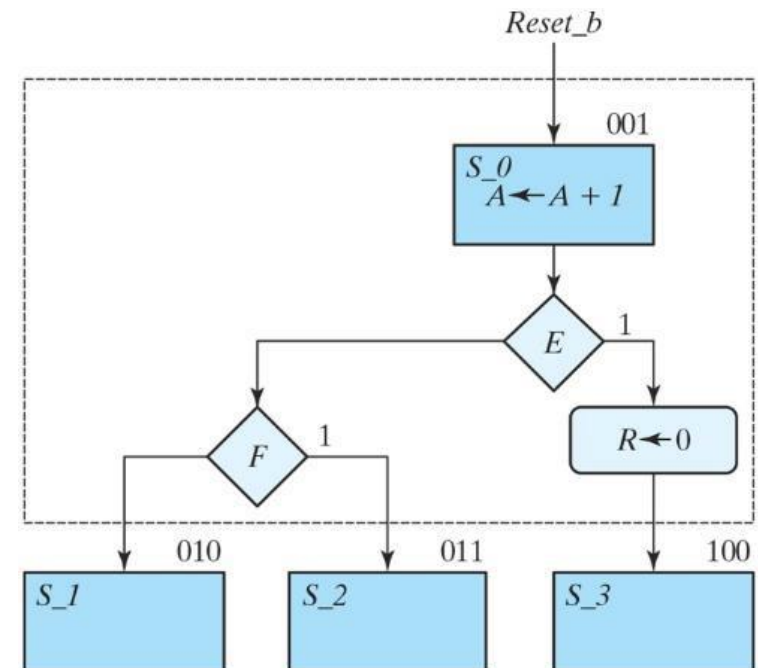
- SystemVerilog
- Block diagrams
- Finite State Machines
- Circuit/gate diagrams

## ❖ Issues:

- SV is a specified language (rigid syntax) and can be very abstract (behavioral)
- Block diagrams can be vague or unspecified
- FSMs don't scale well (# of states + transitions)
- Gate-level is too detailed and specific

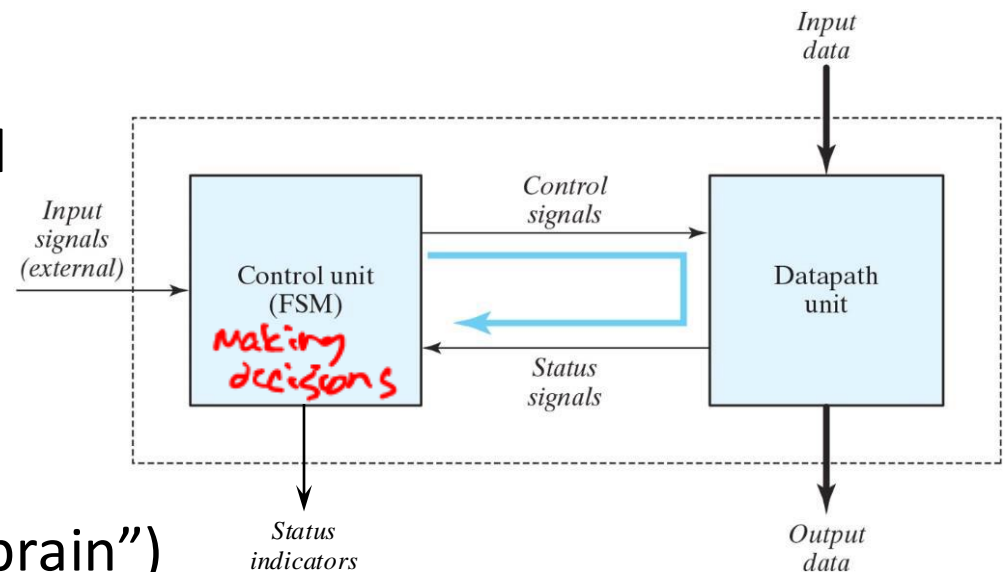
# Algorithmic State Machine (ASM)

- ❖ ASM charts are a method for designing and depicting synchronous digital systems
  - Use more generic syntax (RTL) than SystemVerilog
  - Contain more structured information than FSM state diagrams
  - Can more easily design your system from a *hardware algorithm*



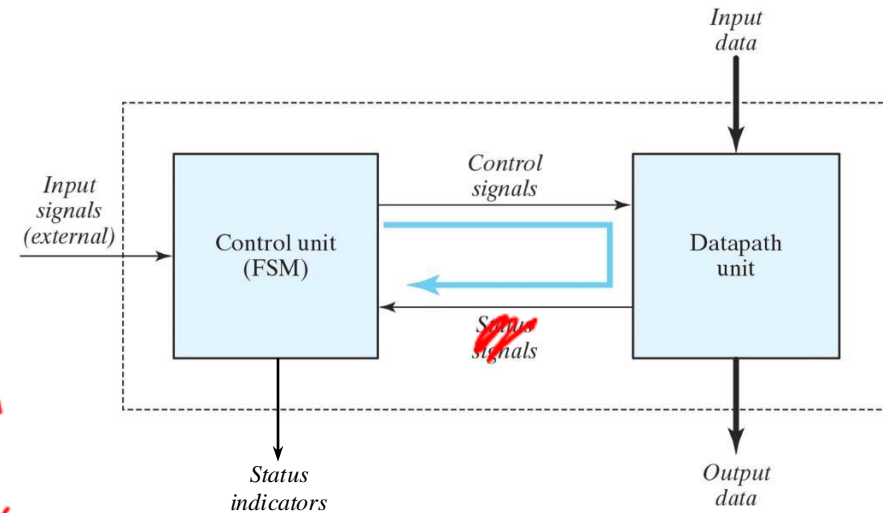
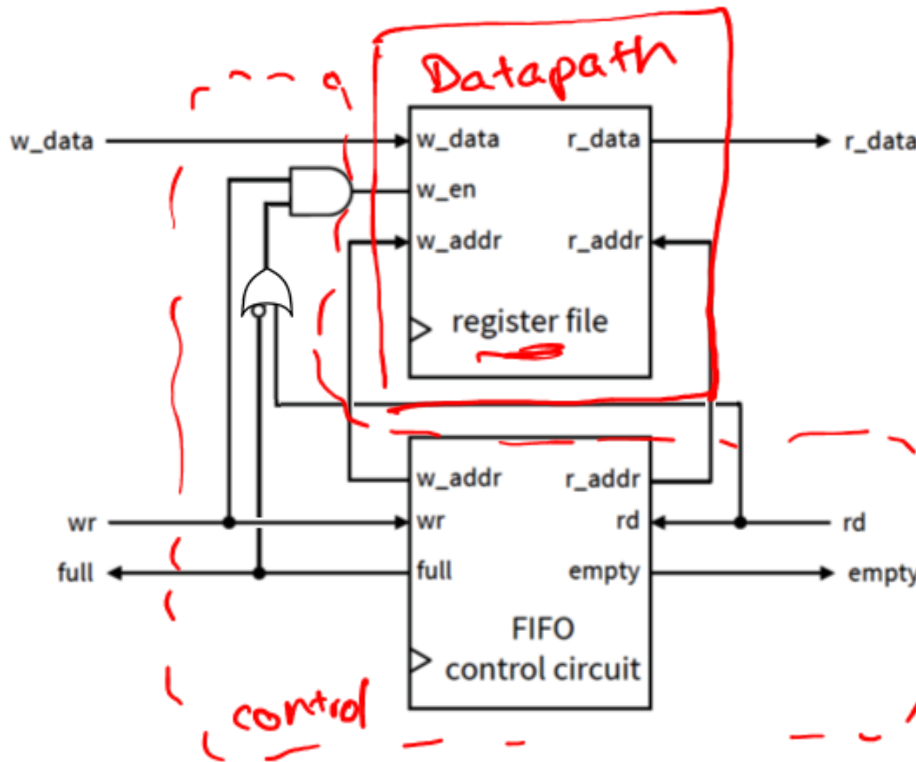
# Control and Datapath

- ❖ Signal classification in a SDS:
  - *Data*: information manipulated/processed by the system
  - *Control*: signals that coordinate and execute the system operations
- ❖ We can logically separate a SDS into two distinct parts/circuits:
  - **Datapath**: parts needed for data manipulation (“the brawn”)
  - **Control**: logic that tells the datapath what needs to be done (“the brain”)



# Control and Datapath: FIFO Buffer

- ❖ Circular queue implementation from last lecture:
  - Datapath and control split?



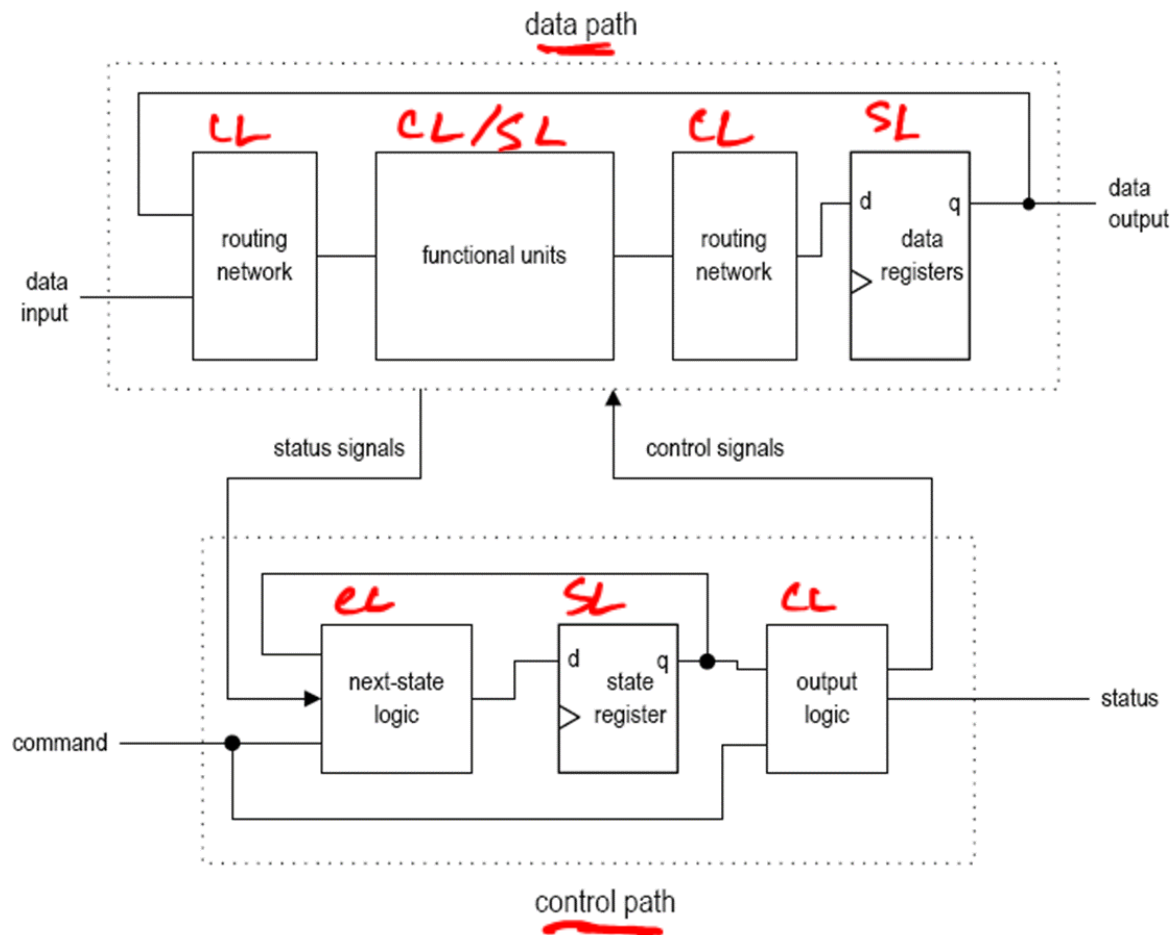
# Algorithms for Hardware

- ❖ *Sequential* algorithms:
  - Variables used as symbolic memory locations
  - Sequential execution dictates the ordering of operations
- ❖ Hardware implementation:
  - Registers store intermediate data (variables)
  - Datapath implements all necessary register operations (computations attached to register inputs)
  - A control path FSM specifies the ordering of register operations
- ❖ This design scheme sometimes referred to as **register-transfer level (RTL)** design

# Algorithms for Hardware

CL = combinational  
SL = sequential

- ❖ The resulting system is called an algorithmic state machine (ASM) or FSM with a datapath (FSMD):



# RTL Operations

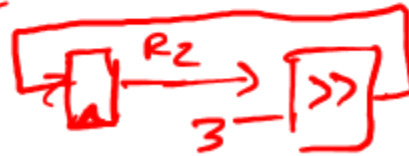
## ❖ Basic form:

$$r_{\text{dest}} \leftarrow f(r_{\text{src1}}, r_{\text{src2}}, \dots, r_{\text{srcn}})$$

- $r_i$  represent registers and  $f()$  represents some combinational function

## ❖ Examples:

- $r_1 \leftarrow 0$  "clear  $r_1$ "
- $r_2 \leftarrow r_1$  "replace  $r_2$  with value in  $r_1$ "
- $\underline{r_2} \leftarrow \underline{r_2} \gg 3$
- $i \leftarrow i + 1$
- $d \leftarrow s_1 + s_2 + s_3$
- $y \leftarrow a * a$



# RTL Operations

## ❖ Basic form:

$$r_{\text{dest}} \leftarrow f(r_{\text{src1}}, r_{\text{src2}}, \dots, r_{\text{srcn}})$$

- $r_i$  represent registers and  $f()$  represents some combinational function

## ❖ Timing Interpretation:

- After the start of a clock cycle, the outputs of all registers update and become available
- During the rest of the clock cycle, these outputs propagate through the combinational circuit that performs  $f()$
- At the *next* clock trigger/cycle, the result is stored into  $r_{\text{dest}}$



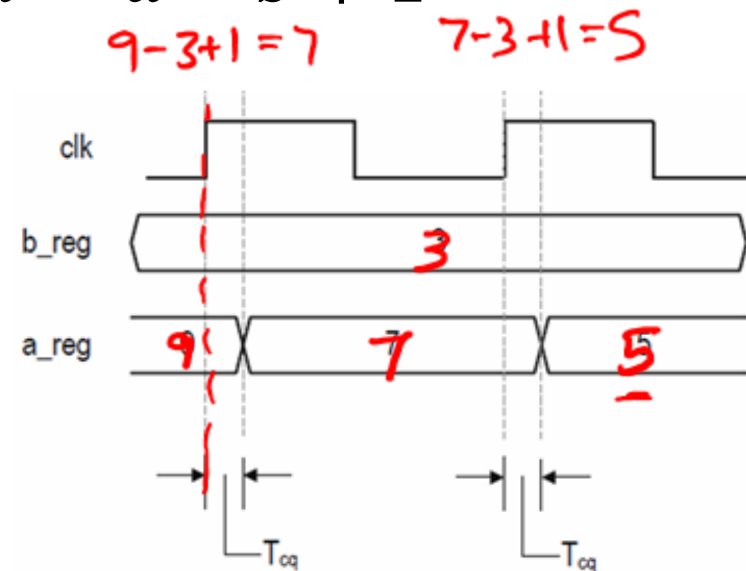
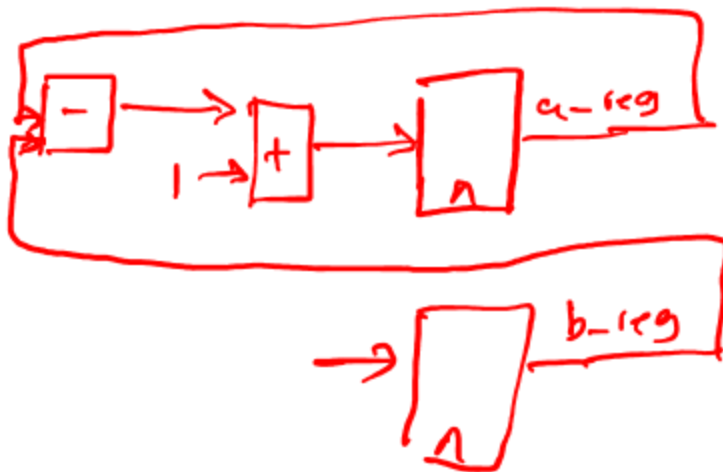
# RTL Operations

## ❖ Basic form:

$$r_{\text{dest}} \leftarrow f(r_{\text{src1}}, r_{\text{src2}}, \dots, r_{\text{srcn}})$$

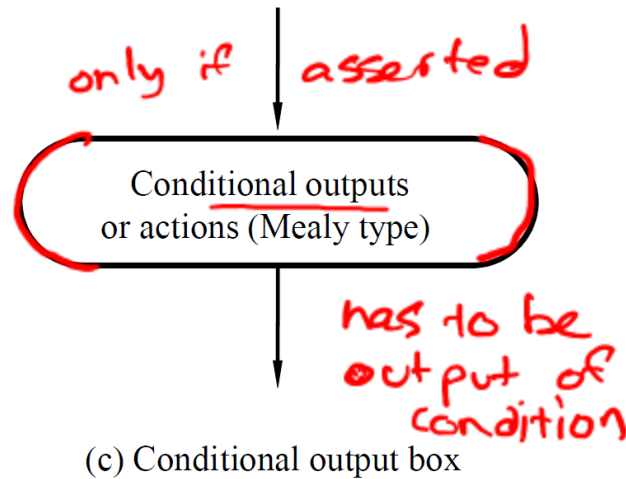
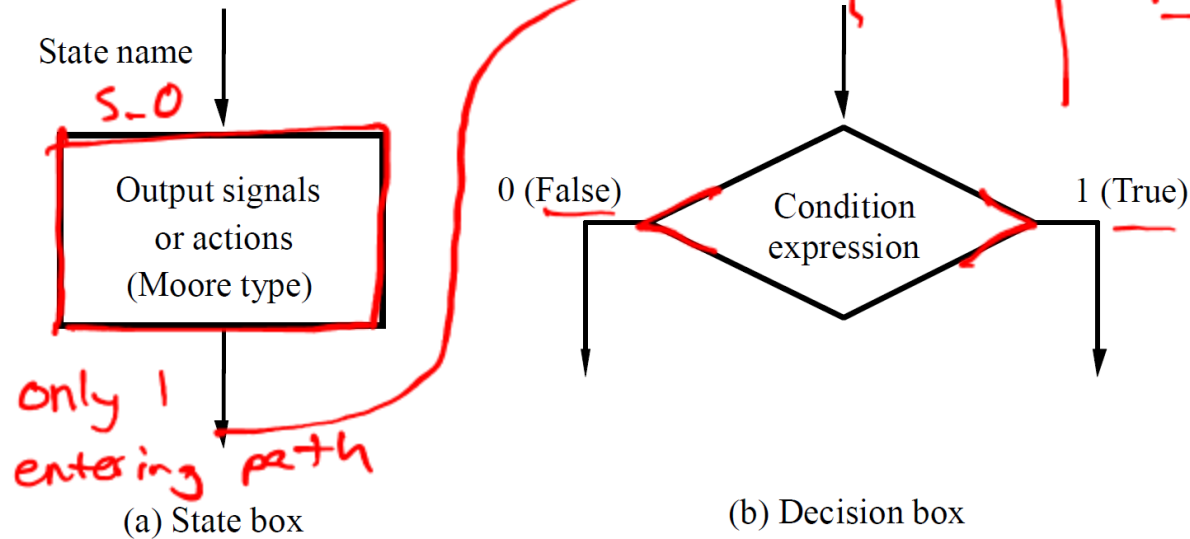
- $r_i$  represent registers and  $f()$  represents some combinational function

## ❖ Implementation Example: $a \leftarrow a - b + 1$



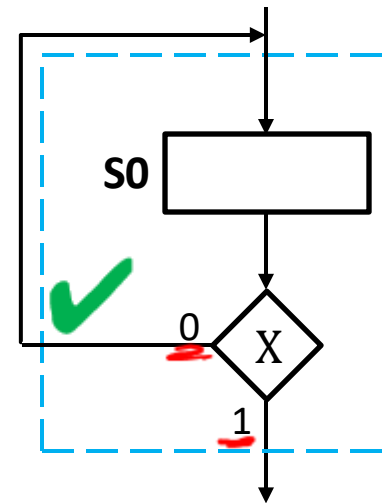
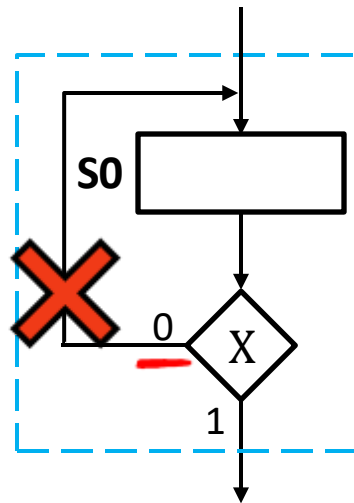
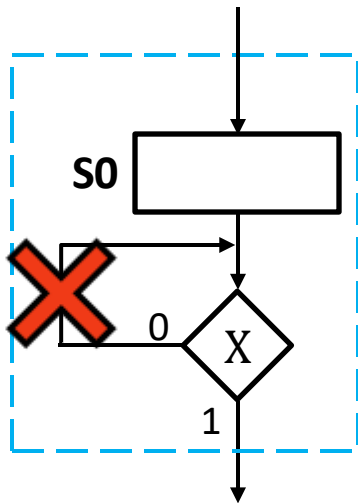
# Technology Break

# ASM Chart



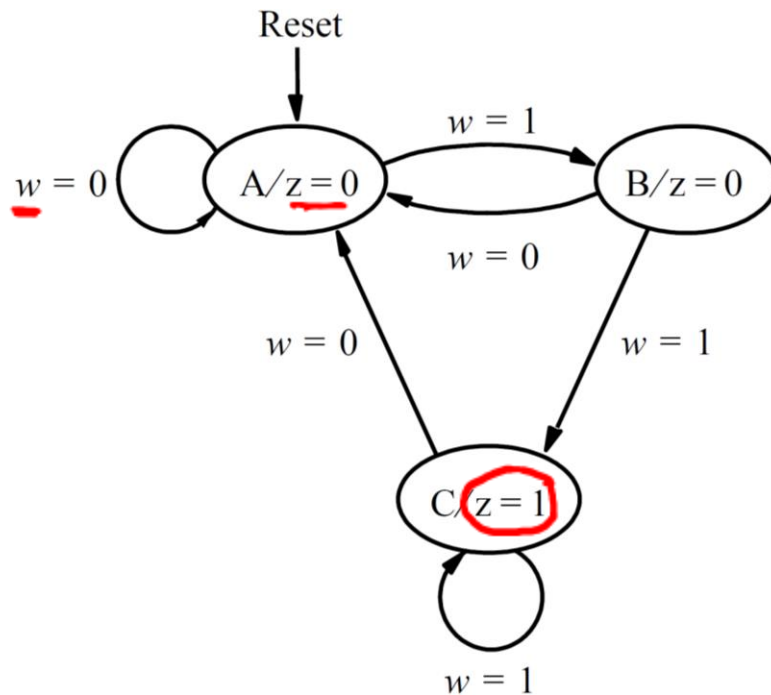
# ASM Blocks

- ❖ Each block describes the state machine operation in a given state
  - For every valid combination of inputs, there must be **exactly one** exit path
  - There should be **no internal feedback**



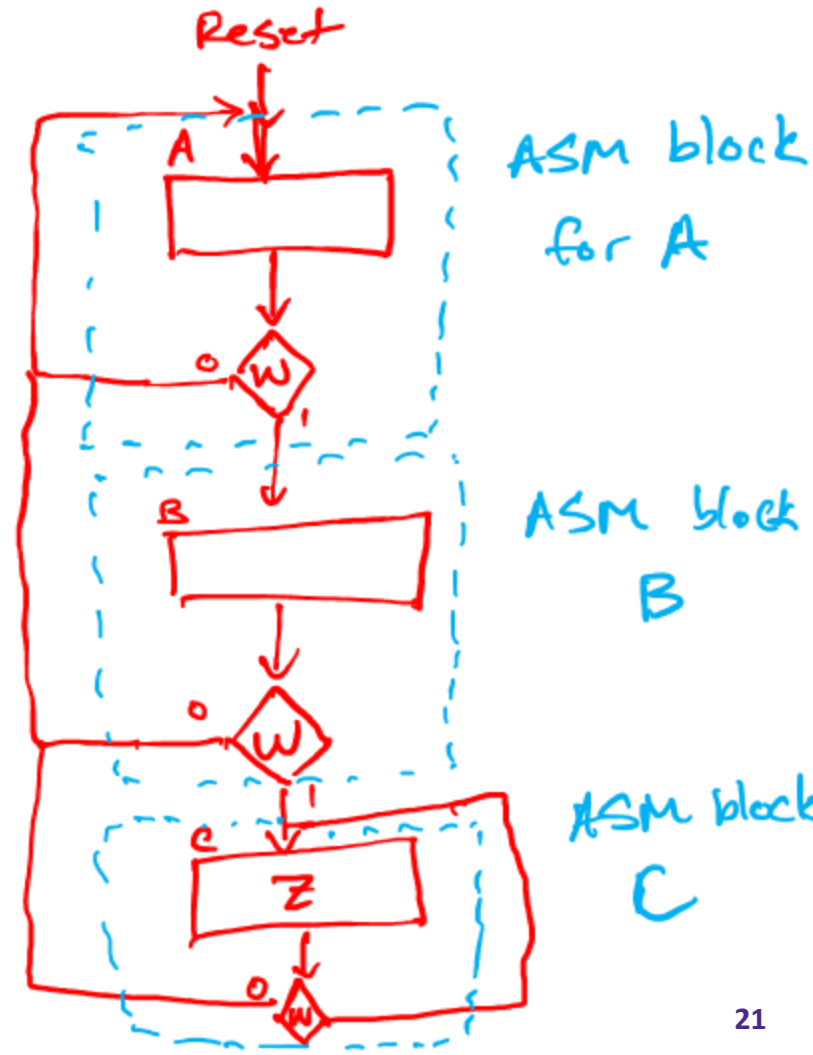
# Worked Example #1

- ❖ Convert this state machine to an ASM chart:



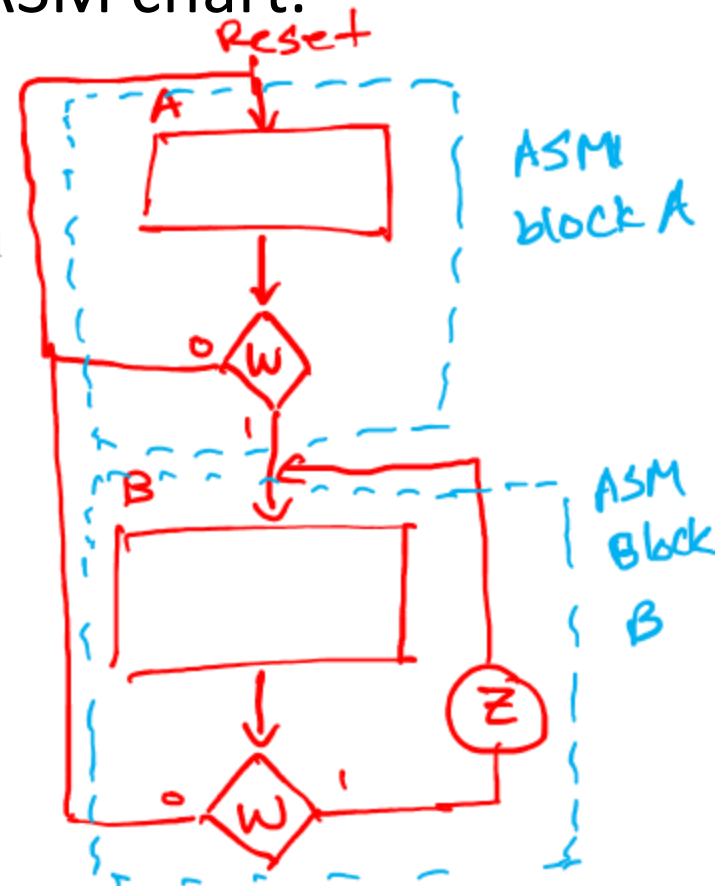
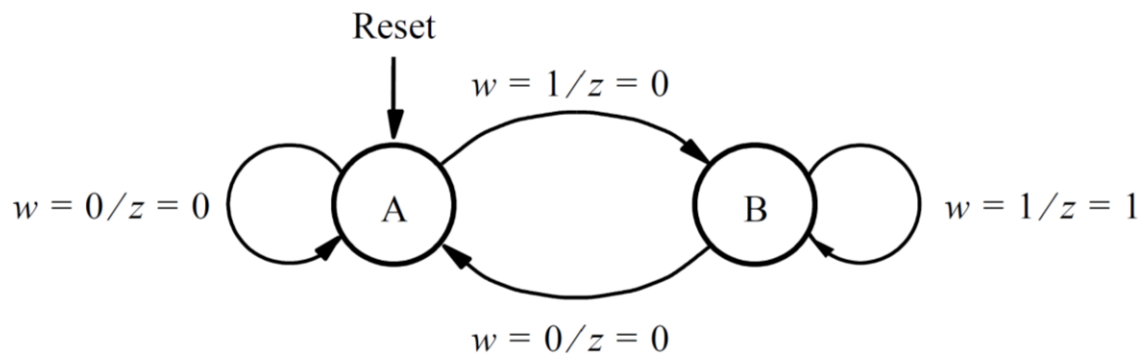
Moore

↳ no conditional  
output boxes



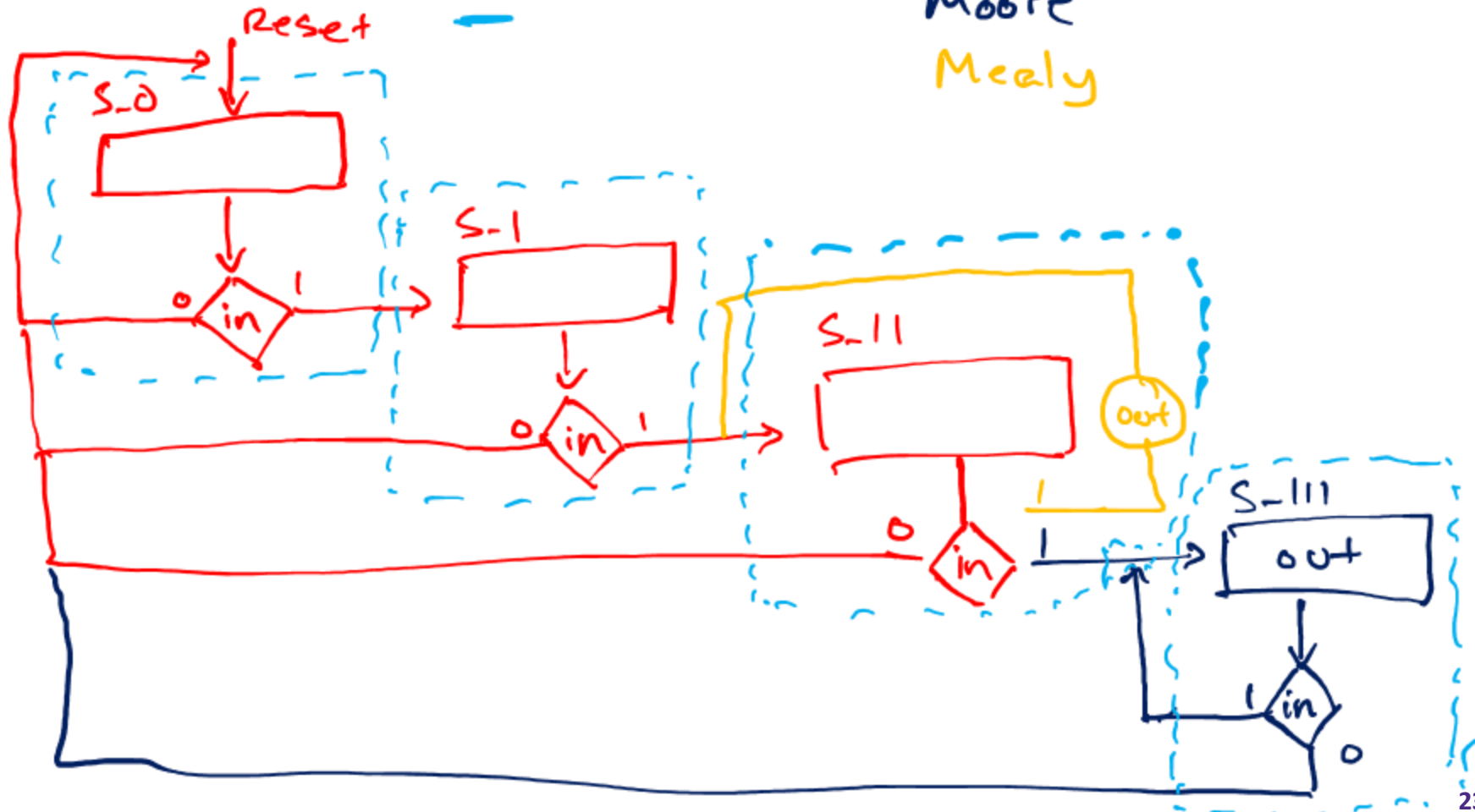
# Worked Example #2

- ❖ Convert this state machine to an ASM chart:



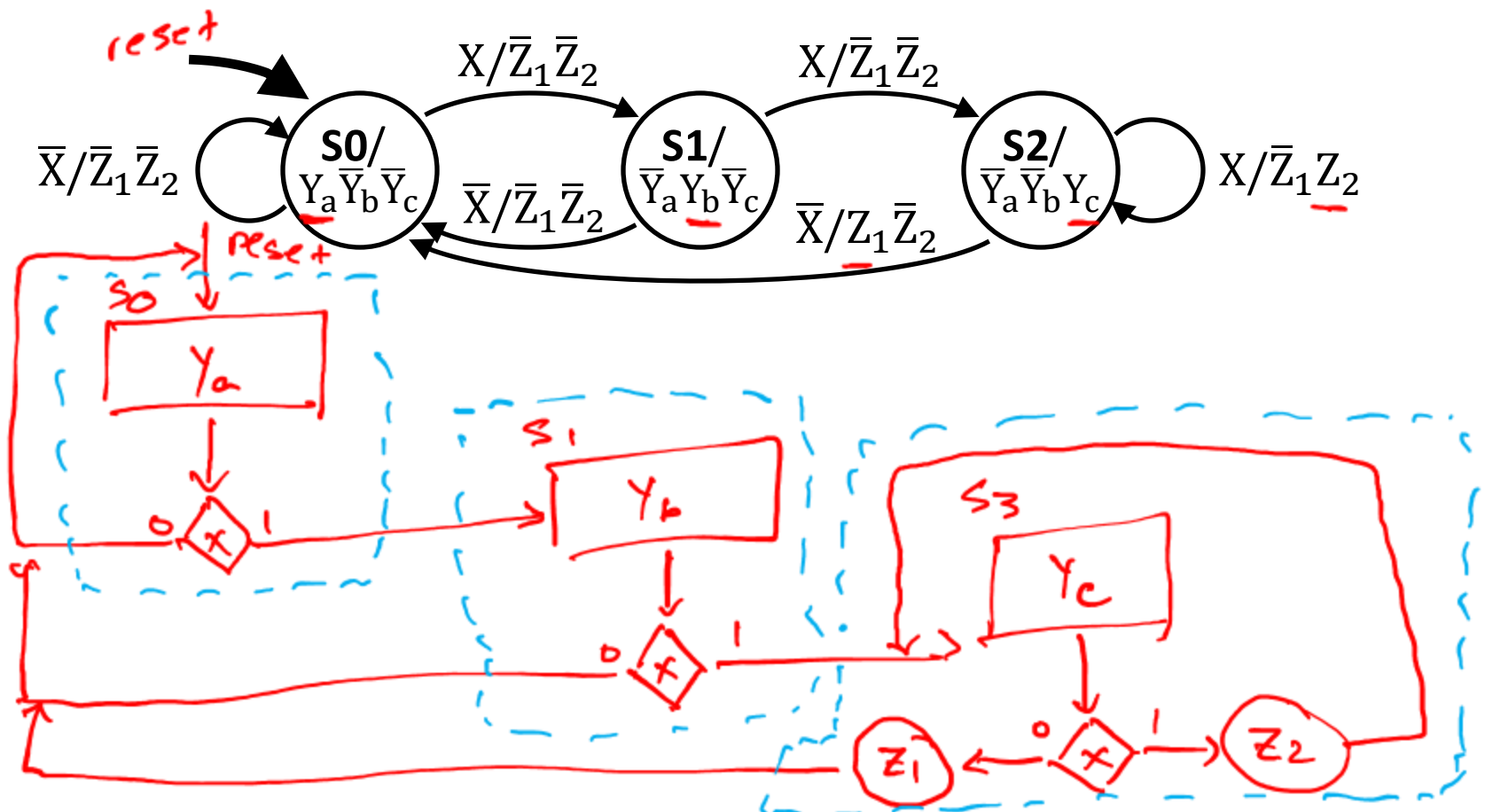
## Example #3

- ❖ Draw an ASM chart for `threeOnes`: asserts out iff last 3 values of in were all 1's.



# Example #4

- ❖ Convert this state machine to an ASM chart:
- 1 input:  $X$ , 5 outputs:  $Y_a$ ,  $Y_b$ ,  $Y_c$  (Moore),  $Z_1$ ,  $Z_2$  (Mealy)





# Worked Example #5 (Preview)

- ❖ Convert the ASM chart for a control circuit shown in figure (b) to a state diagram:

