


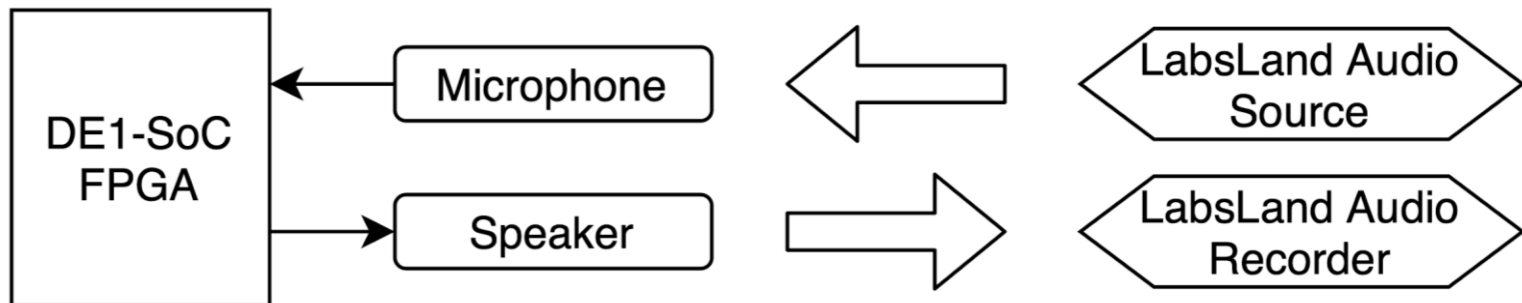
EE/CSE 371
Design of Digital Circuits and Systems
Lecture 5: Algorithmic State Machines

Relevant Course Information

1. Homework 2 due today (10/11)  Attempt HW3 problem1 on before starting Lab3.
2. Homework 3 released, due next Wednesday (10/18)
3. Lab 2 reports due Friday (10/13)
4. Lab 3 released later this week, due Friday (10/27)

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* (Labs → docs) for how to use the LabsLand Audio Interface to send audio input and record audio output:

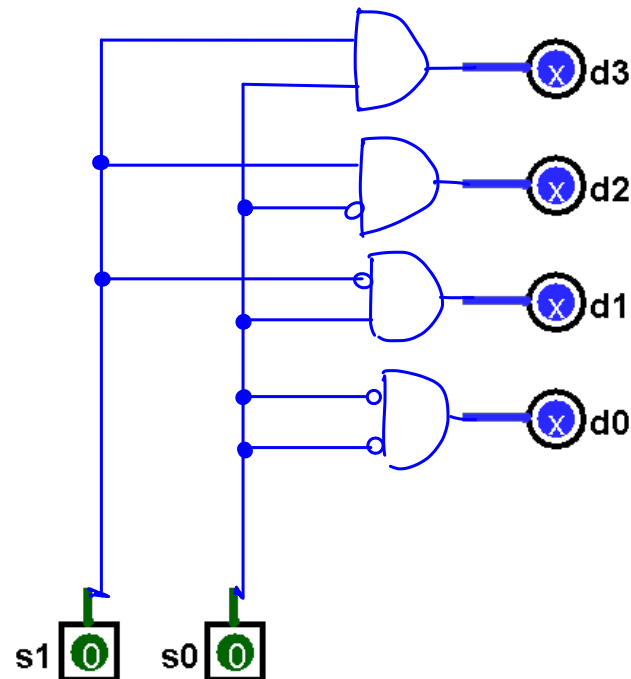
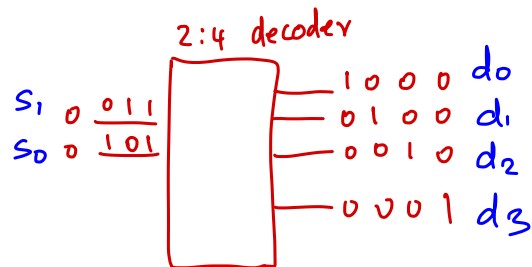


Lab 3 Notes

- ❖ 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
 - Don't use (send/receive) a signal if the receiver isn't ready yet!

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:

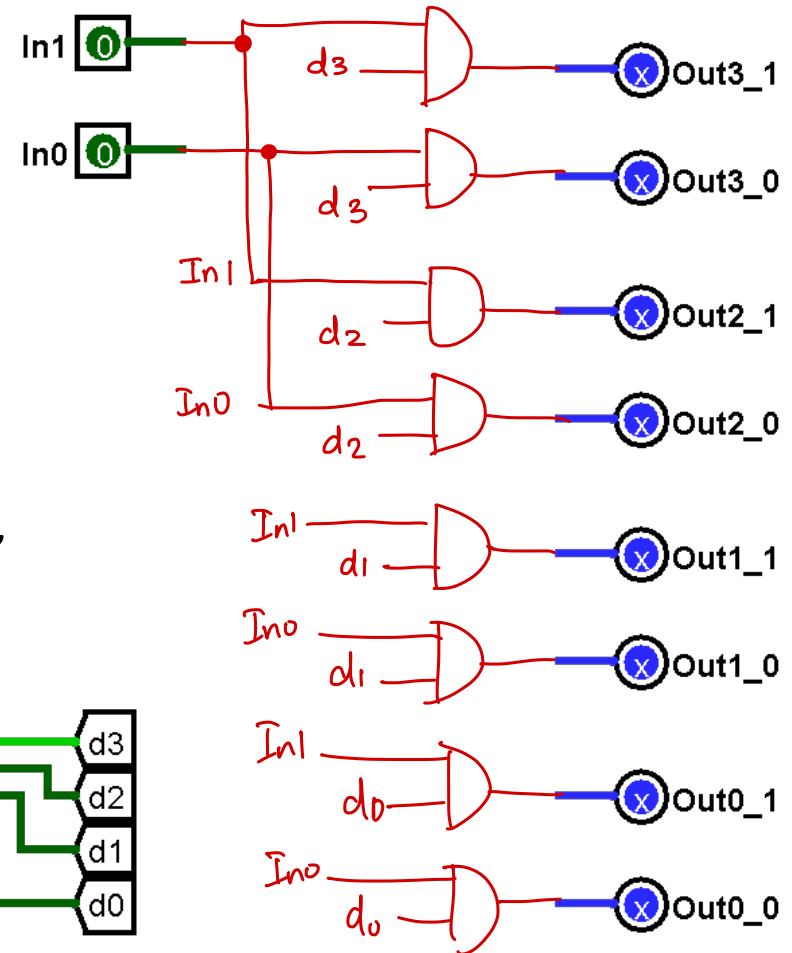
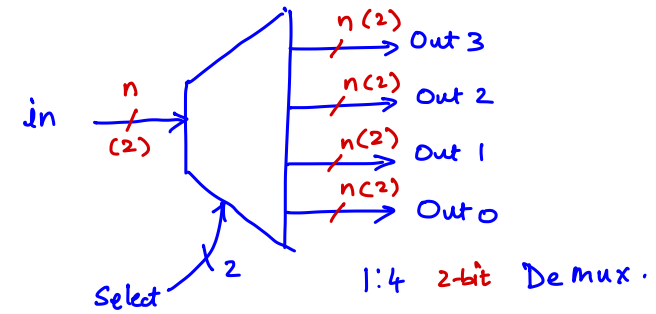
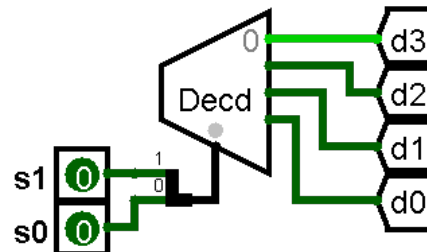


10

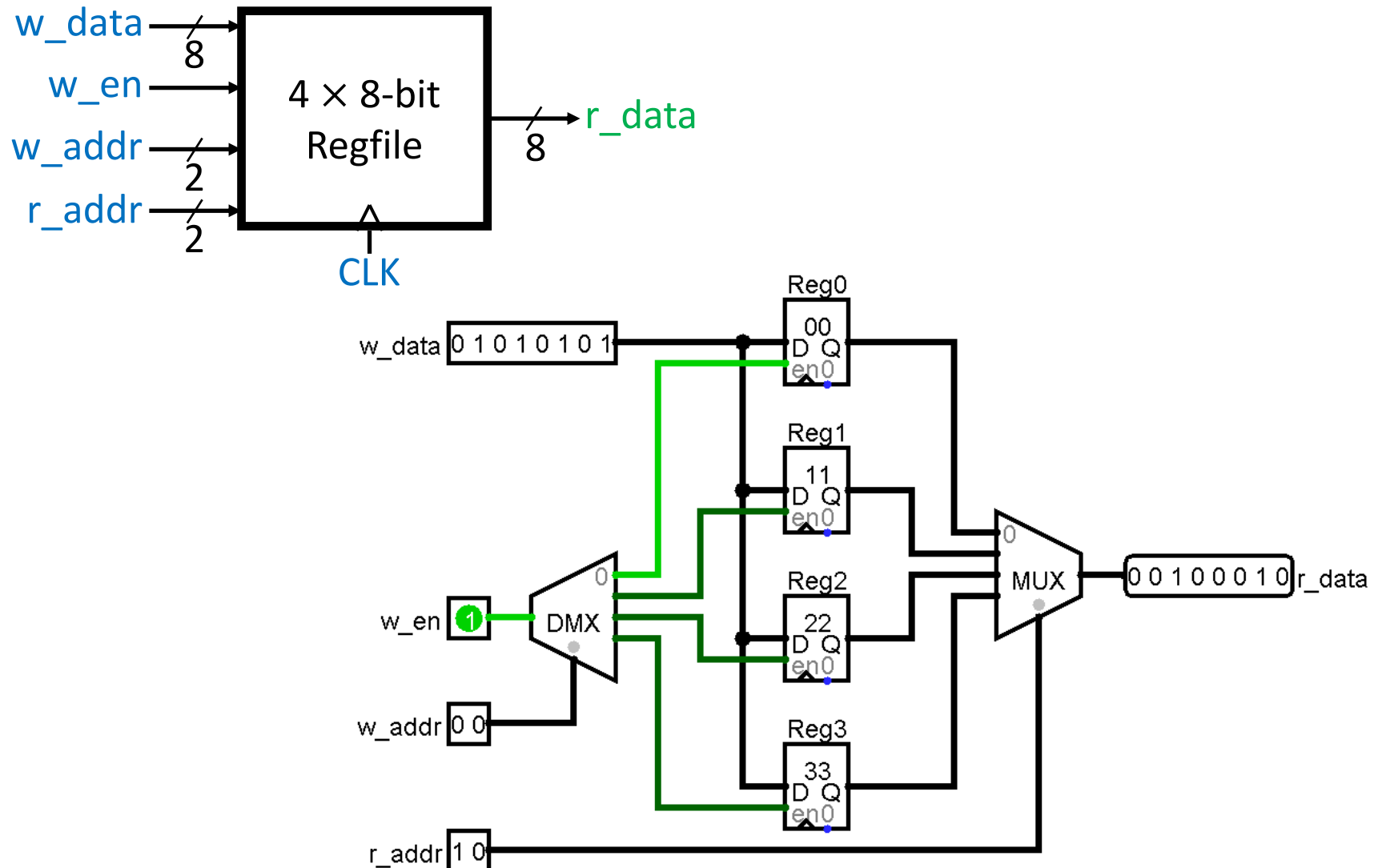
Review Question: DEMUX

❖ Implement a 2-bit, 1-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

(SDS)

❖ So far:

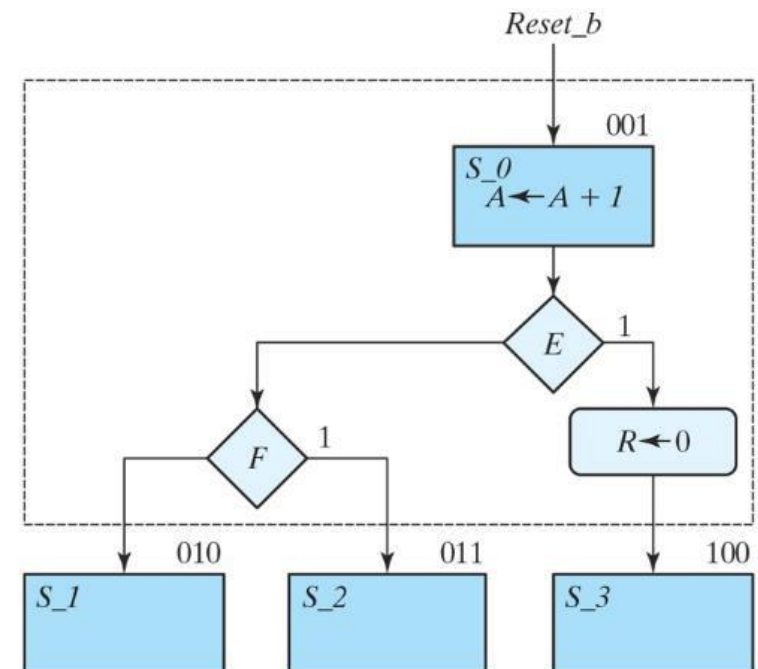
- SystemVerilog
- Block diagrams
- Finite State Machines
- Circuit/gate diagrams

❖ Issues:

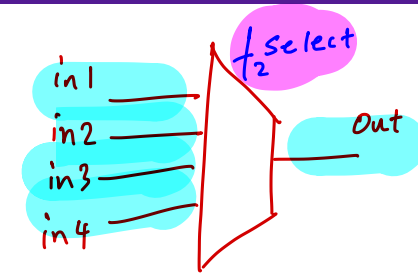
- SV is a specified language (rigid syntax) and can be very abstract (behavioral)
- Block diagrams can be vague or unspecified
- FSMs don't scale well (# of states + transitions)
- Gate-level is too detailed and specific

Algorithmic State Machine (ASM) \equiv FSM + datapath control

- ❖ 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

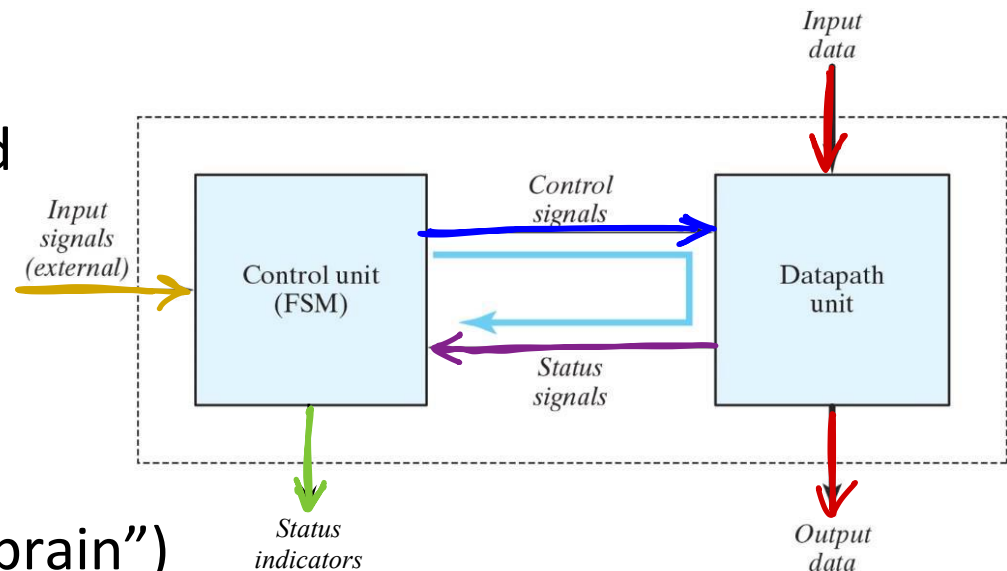


❖ Signal classification in a SDS:

- **Data:** information manipulated/processed by the system
- **Control:** signals that coordinate and execute the system operations

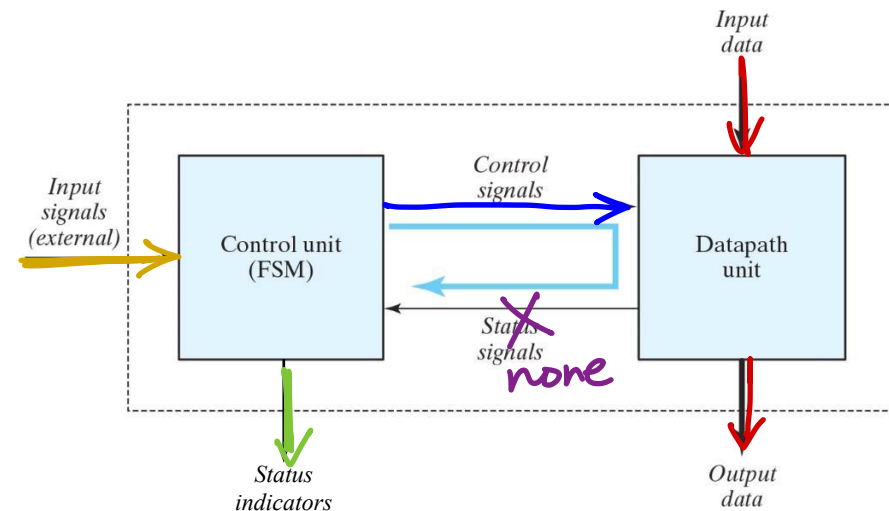
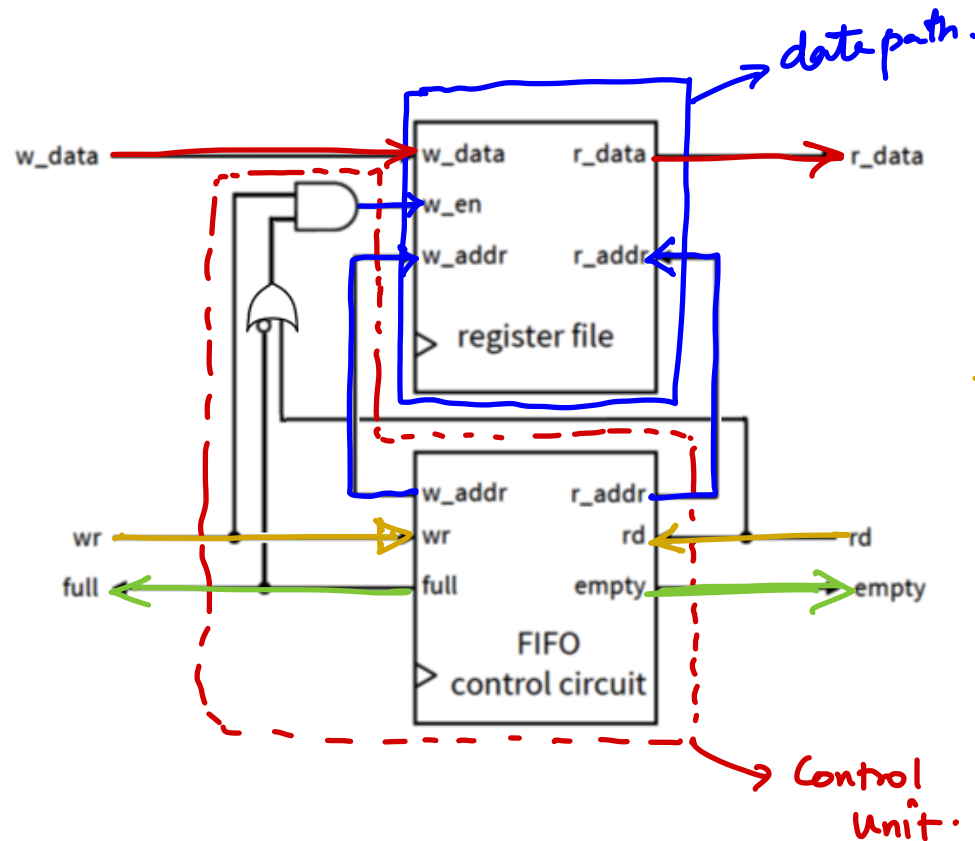
❖ We can logically separate a SDS into two distinct parts/circuits:

- **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?

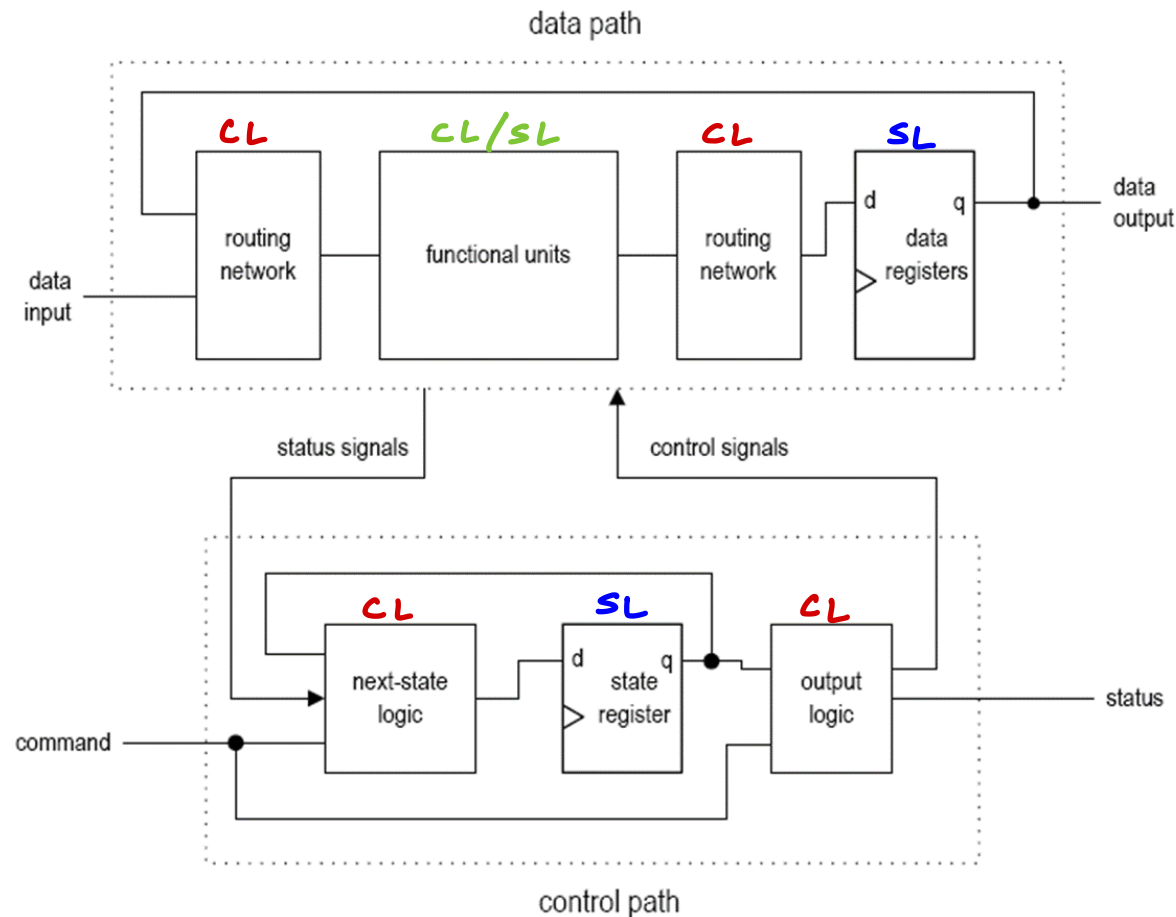


Algorithms for Hardware

- ❖ *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

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



RTL Operations

❖ Basic form:

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

replacement operator

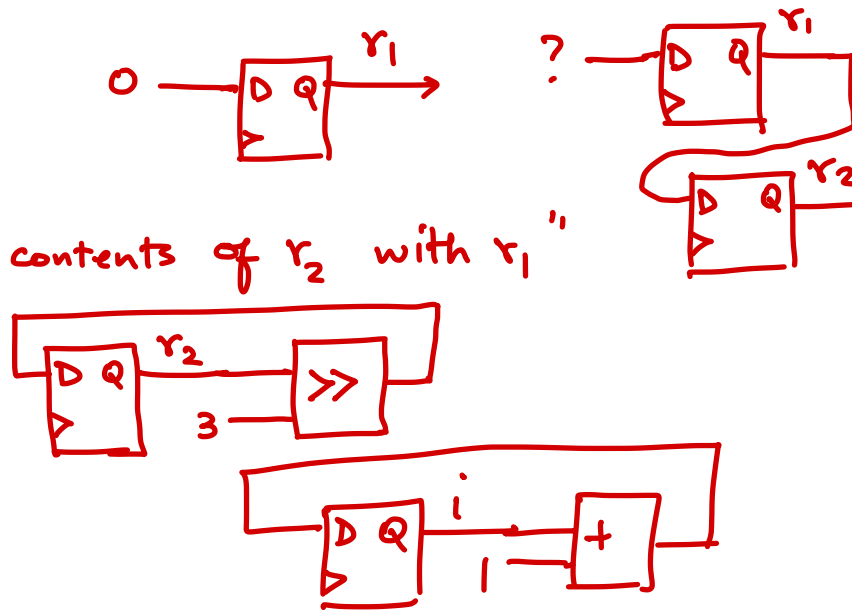
- r_i represent registers and $f()$ represents some combinational function

❖ Examples:

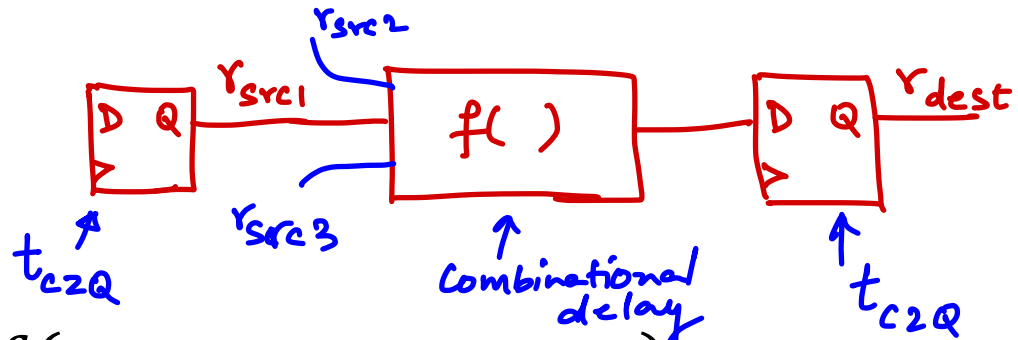
- $r_1 \leftarrow 0$
- $r_2 \leftarrow r_1$
- $r_2 \leftarrow r_2 \gg 3$
- $i \leftarrow i + 1$
- $d \leftarrow s_1 + s_2 + s_3$
- $y \leftarrow a * a$

"clear r_1 "

"replacing contents of r_2 with r_1 "



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_{dest}

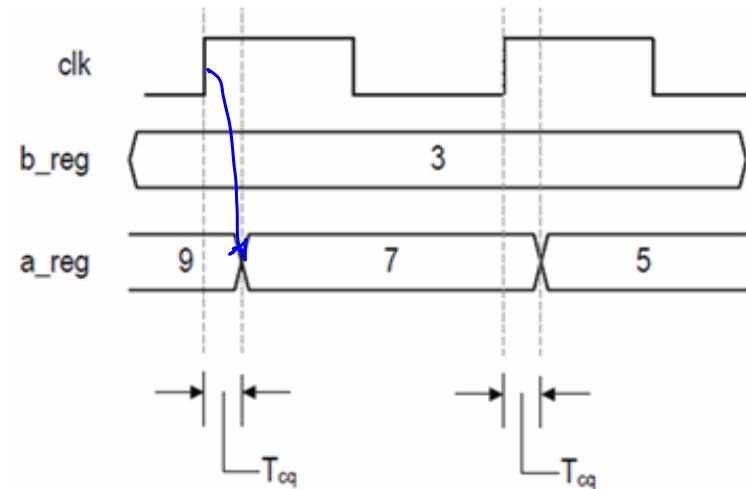
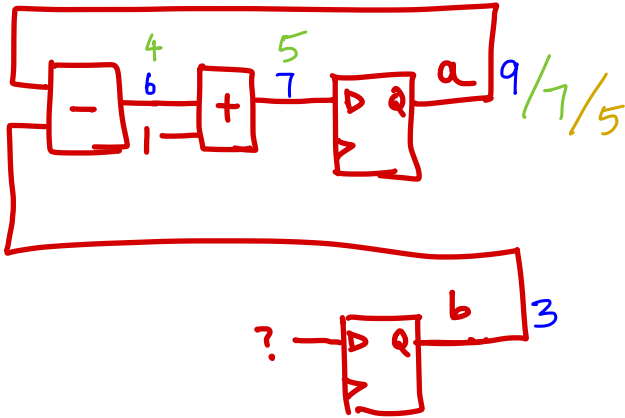
RTL Operations

❖ Basic form:

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

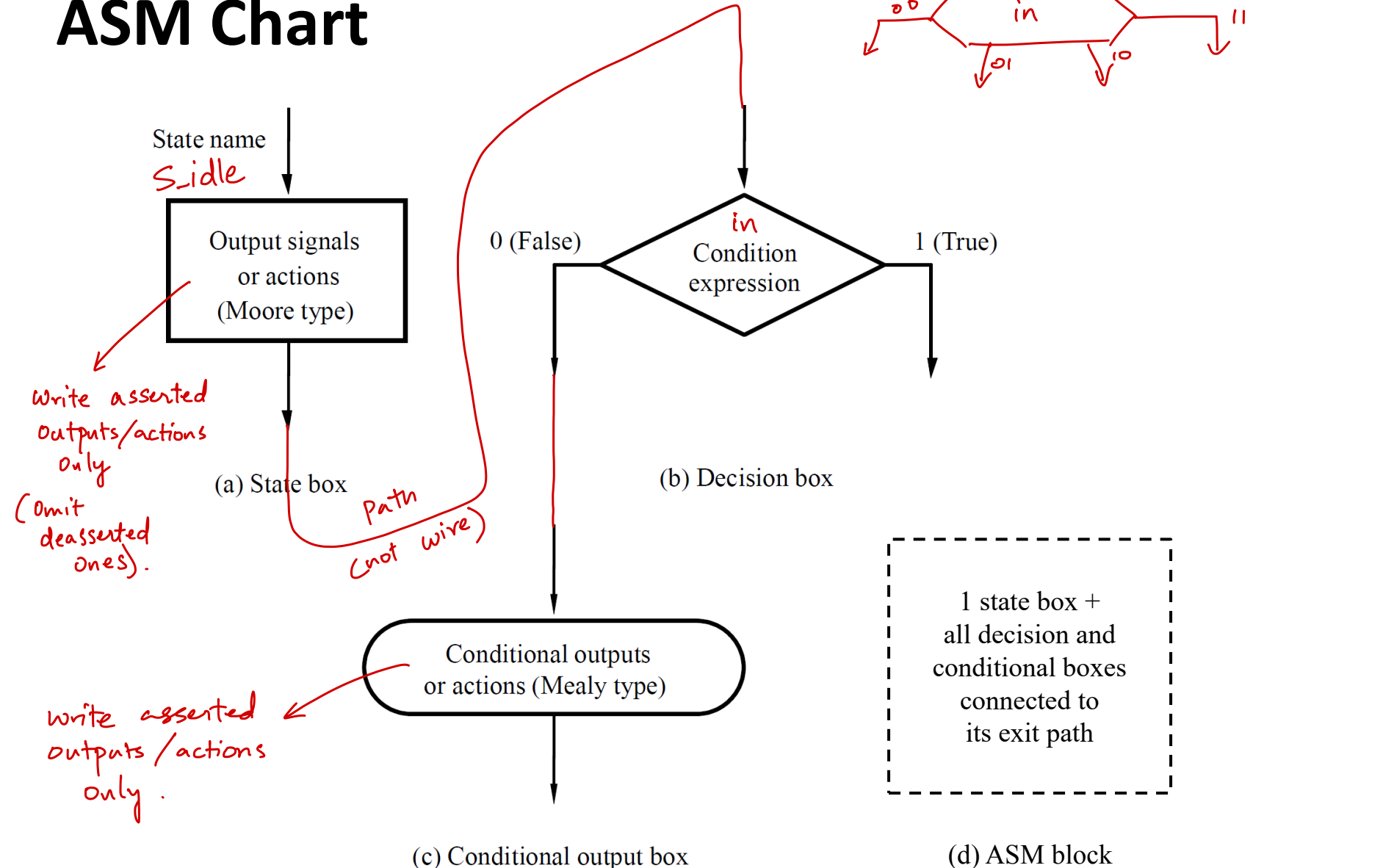
- r_i represent registers and $f()$ represents some combinational function

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



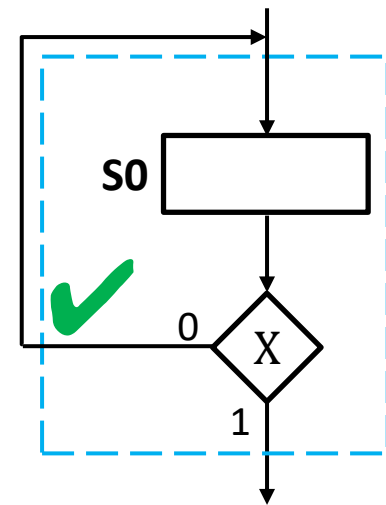
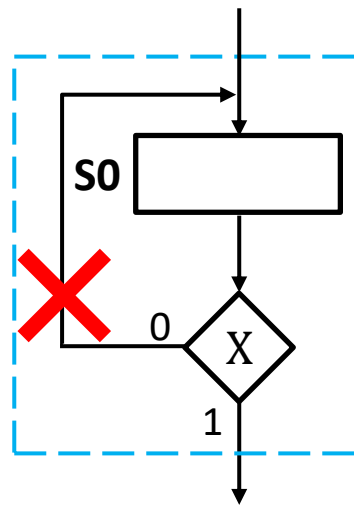
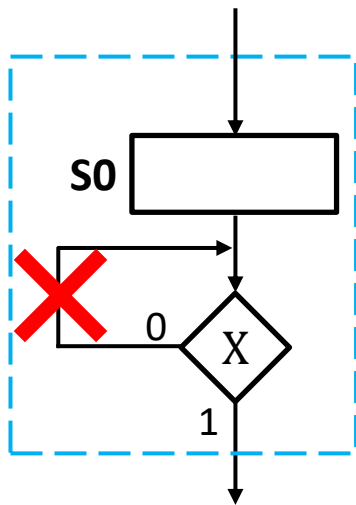
Technology Break

ASM Chart



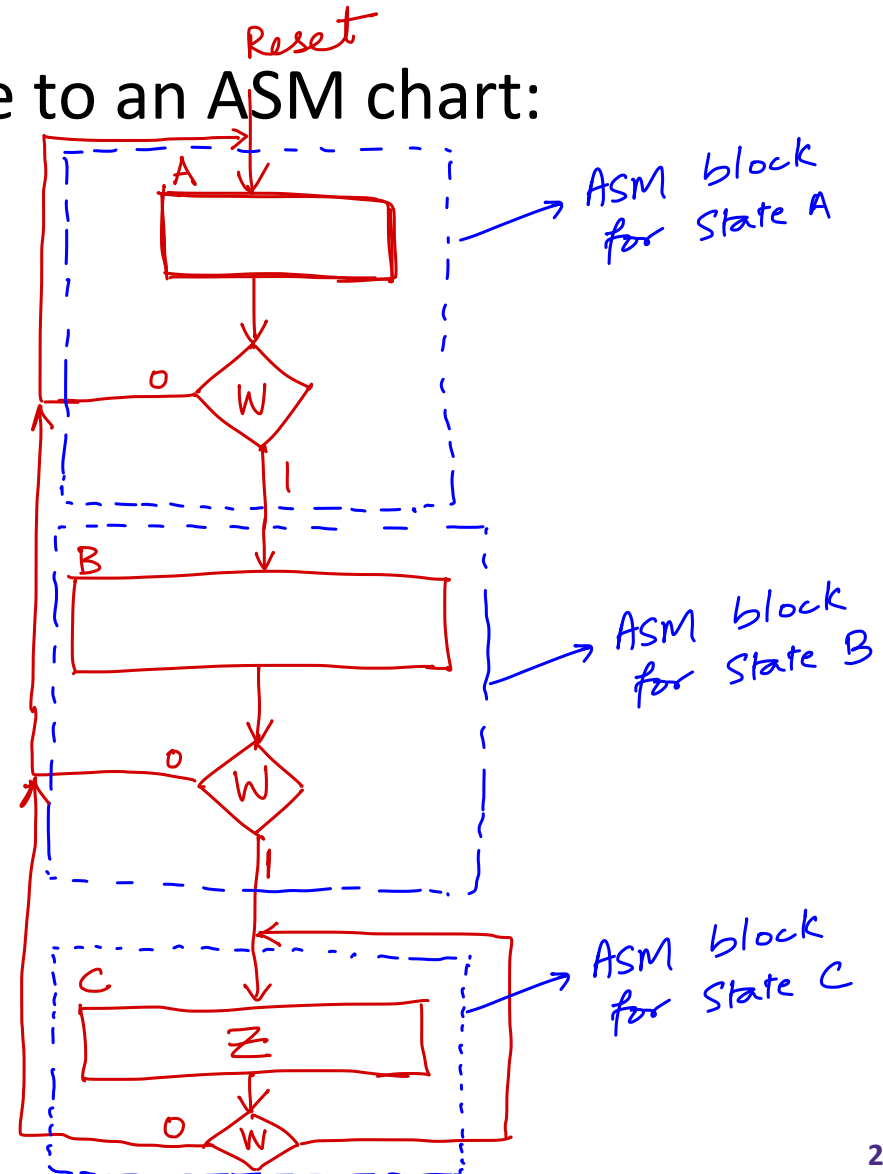
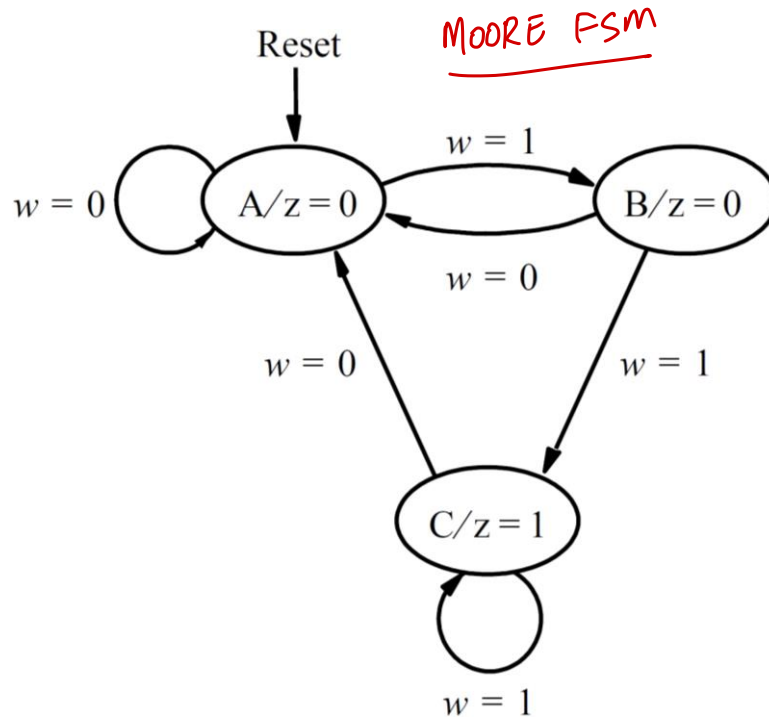
ASM Blocks

- ❖ 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**



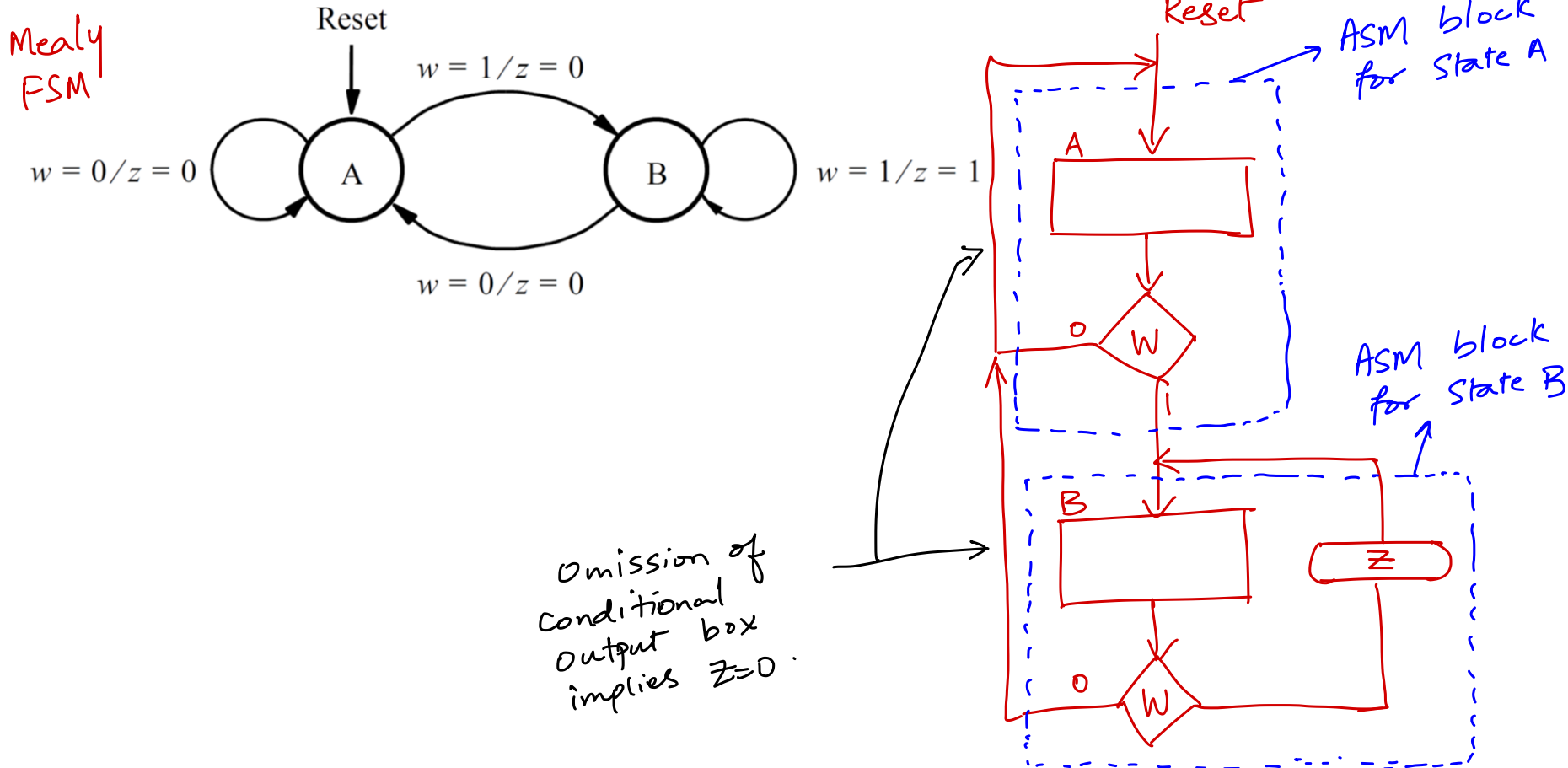
Worked Example #1

❖ Convert this state machine to an ASM chart:



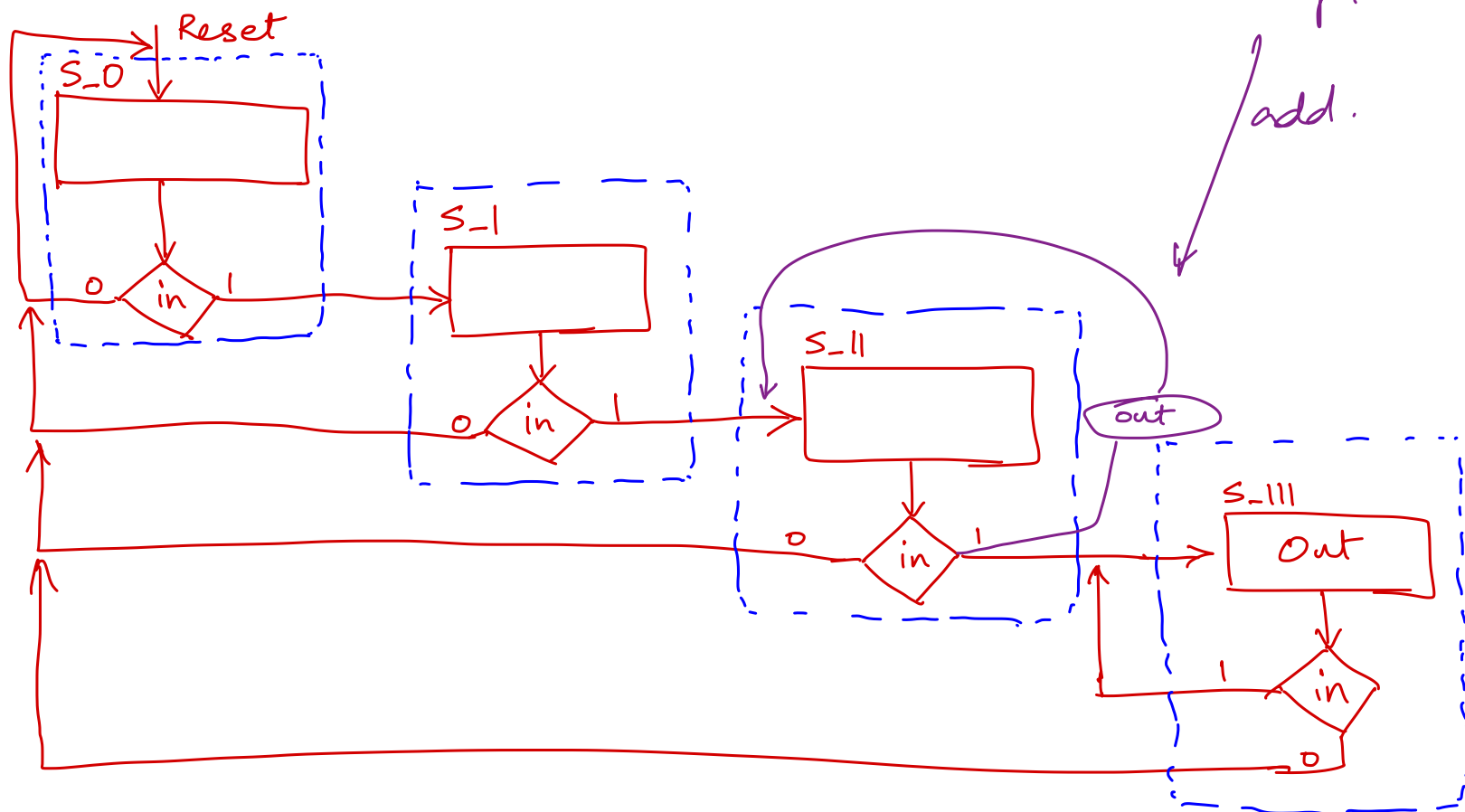
Worked Example #2

❖ Convert this state machine to an ASM chart:



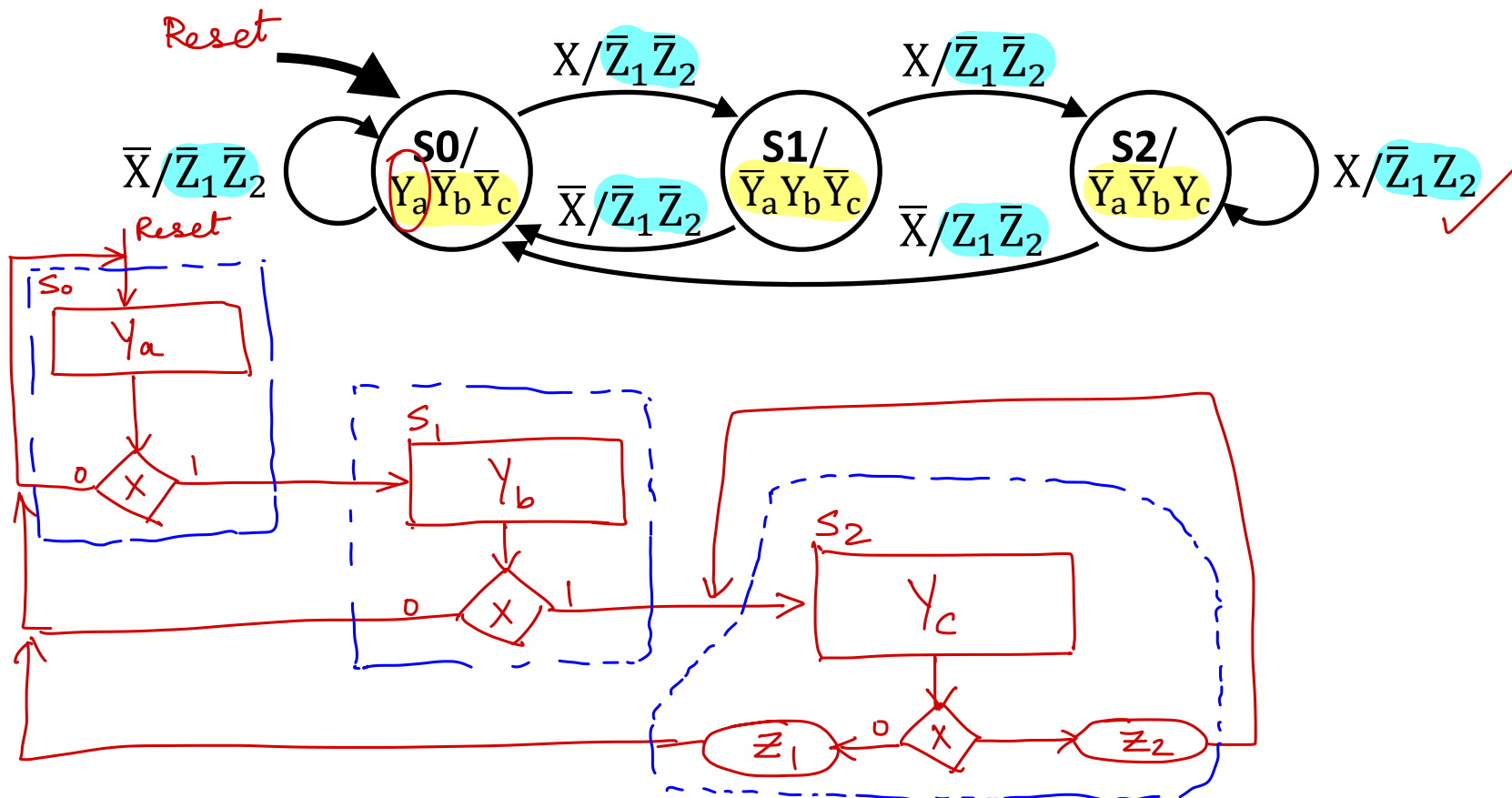
Example #3

- ❖ 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:

