# Exercises with Digital Logic

**CS 64: Computer Organization and Design Logic**
**Lecture #18**
**Winter 2019**

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

## Administrative

- Lab #8
  - Due on Wednesday
  - Paper copy drop off at HFH 2nd floor

# Administrative

- The Last 3 Weeks of CS 64:

| Date | L # | Topic | Lab | Lab Due |
|---|---|---|---|---|
| 2/26 | 14 | Combinatorial Logic, Sequential Logic 1 | 7 (CL+SL) | Wed. 3/6 |
| 2/28 | 15 | Sequential Logic 2 | | |
| 3/5 | 16 | FSM 1 | 8 (FSM) | Wed. 3/13 |
| 3/7 | 17 | FSM 2 | | |
| 3/12 | 18 | Digital Logic Review | 9 (Ethics) | Fri. 3/15 |
| 3/14 | 19 | CS Ethics & Impact Final Exam Review | | |

# FINAL IS COMING!

- **Thursday, 3/21** in this classroom
- **Starts at 4:00 PM **SHARP****
- Please start arriving 10 minutes early
- Please bring your UCSB IDs with you

- **Closed book**: no calculators, no phones, no computers

**STUDY GUIDE**
**NOW ONLINE!**

## What’s on the Final

- Everything

# Lecture Outline

***Exercises in Digital Logic***

- Ex. 1: Simplifying logic functions
- Ex. 2: Simplifying a logic circuit
- Ex. 3: Constructing a truth table
- Ex. 4: D-Latches and Multiplexers
- Ex. 5: FSM

# Exercise 1

| A | B | C | D | R | Q |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | X |
| 0 | 0 | 1 | 0 | X | 0 |
| 0 | 0 | 1 | 1 | X | 0 |
| 0 | 1 | 0 | 0 | X | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | X | 1 |
| 0 | 1 | 1 | 1 | X | 0 |
| 1 | 0 | 0 | 0 | X | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | X | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | X | X |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

Given this truth table for a **combinatorial circuit**, showing **4 inputs** (A,B,C,D) and **2 outputs** (R, Q), do the following:

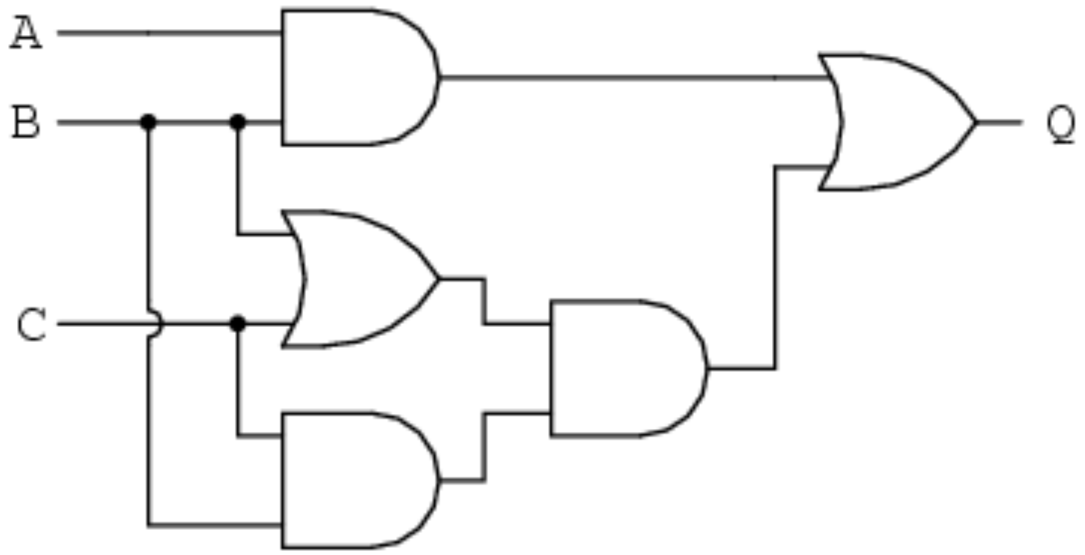
a) Draw the necessary K-Maps.

b) Write the resulting optimized logic functions.

c) Draw the simplified circuit that is described by this T.T. without unnecessary repeating logic blocks.

# Exercise 2

Given this circuit diagram:

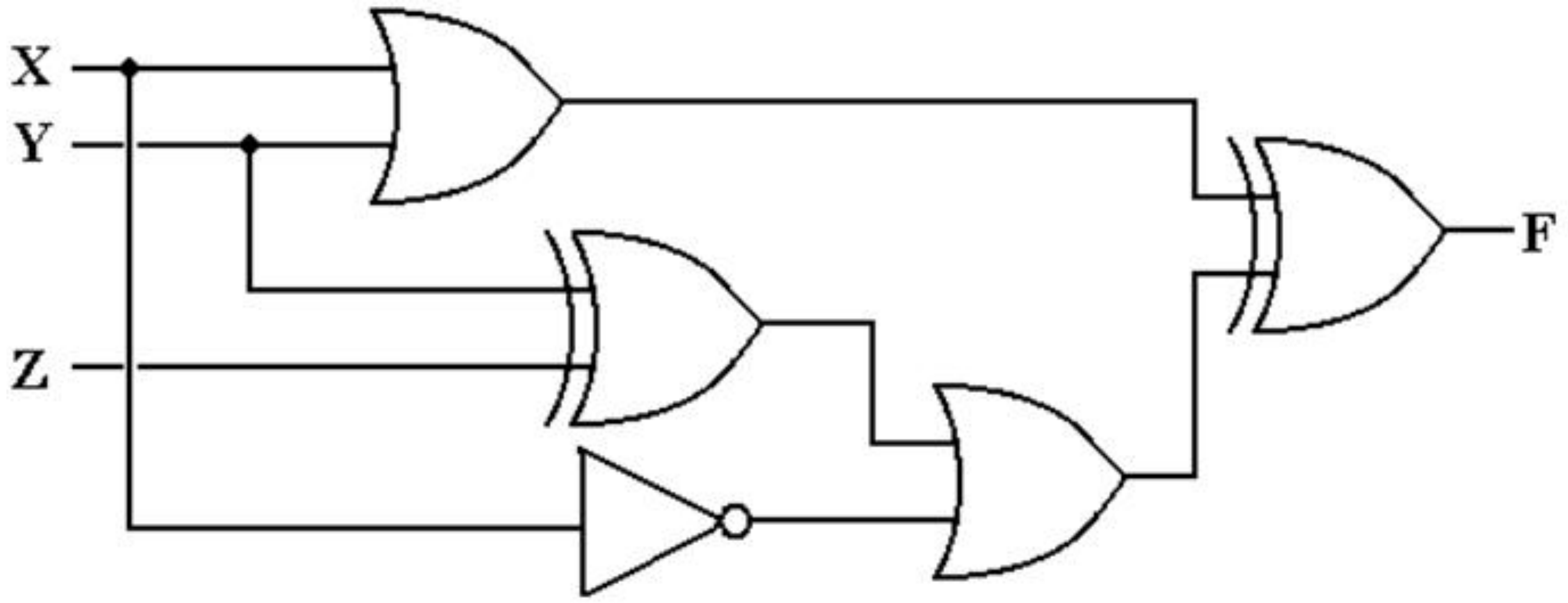


a) Write logic functions that describe the diagram, but in the form of "sum-of-product".

b) Re-draw the circuit based on your answers in part (a).

# Exercise 3

Given this circuit diagram:



a) Write simplified logic functions that describe the diagram in a "sum-of-product" format.

b) Construct the truth table for this circuit

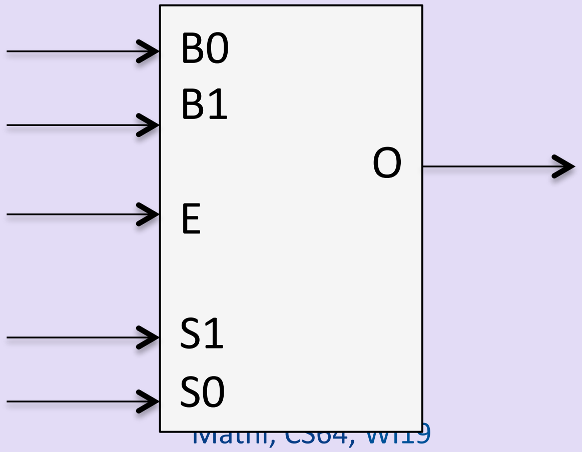
c) Further optimize the logic function based on the T.T.

# Exercise 4

Using one or more **D-Latch**, one or more **2-to-1 mux**, and any number of **basic logic blocks** (i.e. AND, OR, NOT, etc.), construct a circuit that takes 2 bits as inputs (B0, B1) and writes them to 2 registers called REG0 and REG1, respectively, *but only* when another input, E = 1.

The registers retain their values if E = 0.

The output of this circuit is the value stored in one the registers as chosen by inputs S1 and S0. If S1=1 and S0=1, we should see the value in REG1 appear on the output (O) and if S1=0 and S0=0, we should see the value in REG0 appear at O. Any other combination of S1 and S0 should leave the output unchanged, that is the previous selection should remain the same.

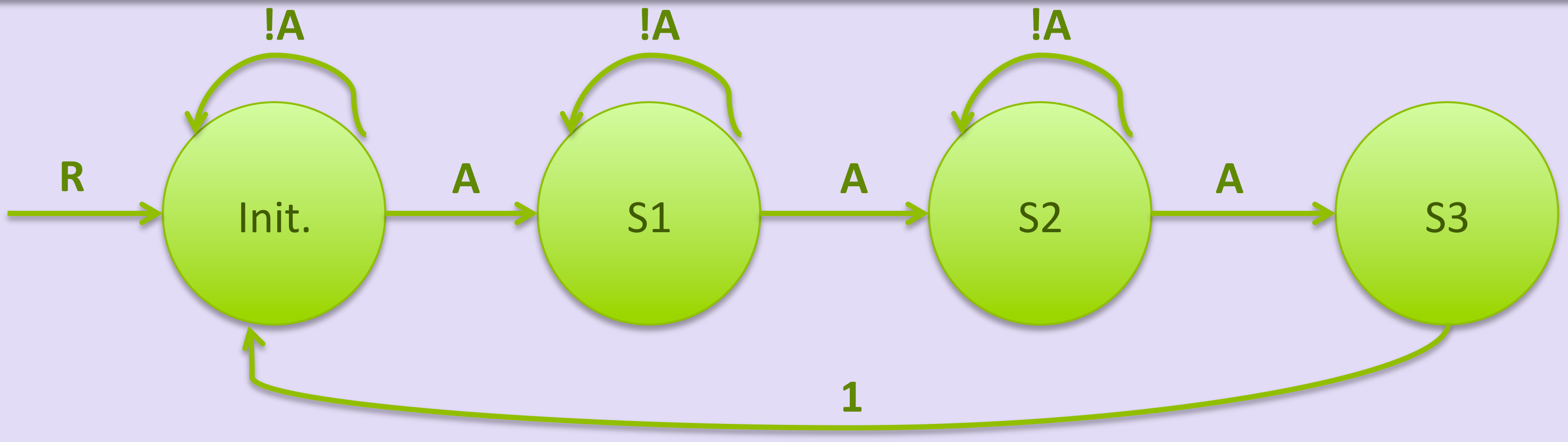


## Exercise 5

- Design a FSM that takes in one input, A, and on every rising clock edge, switches from initial state to State1 to State2 to State3, when A is set.
- When it gets to the last state, it simply goes to back the initial state again (regardless of input A).
- There is a second input R, that, when set, forces the state machine to go back to the initial state, regardless of where it currently is.

A. Draw the state diagram.
B. Using the "**regular method**", how many bits do we need to represent all the states in this FSM?
C. Using the "**one-hot method**", how many bits do we need to represent all the states in this FSM?
D. Write the next-state functions for this FSM using the approach in C.
E. Design the dig. logic circuit to implement the FSM using your results so far

# Exercise 5 Partial Solution



Using “one hot” method, we’d use 4 D-FFs. Let’s call Init. = S0. The next state functions are:

S0* = S0.!A + S3 + R
S1* = S0.A + S1.!A
S2* = S1.A + S2.!A
S3* = S2.A

## YOUR TO-DOs

- Lab 8

</LECTURE>