Exercises with Finite State Machines

CS 64: Computer Organization and Design Logic
Lecture #17
Winter 2019

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

Administrative

- Lab #8
  - Due next week 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		

Designing the Circuit for the FSM

1. We start with a T.T
   - Also called a “State Transition Table”
2. Make K-Maps and simplify
   - Usually give your answer as a “sum-of-products” form
3. Design the circuit
   - Have to use D-FFs to represent the state bits

1. The Truth Table (The State Transition Table)

Next state: K-Maps!

Outputs: Relate to state input bits

	CURRENT STATE			INPUT(S)	NEXT STATE			OUTPUT(S)
State	**B2**	**B1**	**B0**	**I**	**B2***	**B1***	**B0***	**FOUND**
Initial	0	0	0	0	0	0	0	0
				1	0	0	1	0
Found “1”	0	0	1	0	0	0	0	0
				1	0	1	0	0
Found “11”	0	1	0	0	0	1	1	0
				1	0	1	0	0
Found “110”	0	1	1	0	0	0	0	0
				1	1	0	0	0
Found “1101”	1	0	0	0	0	0	0	**1**
				1	0	1	0	**1**

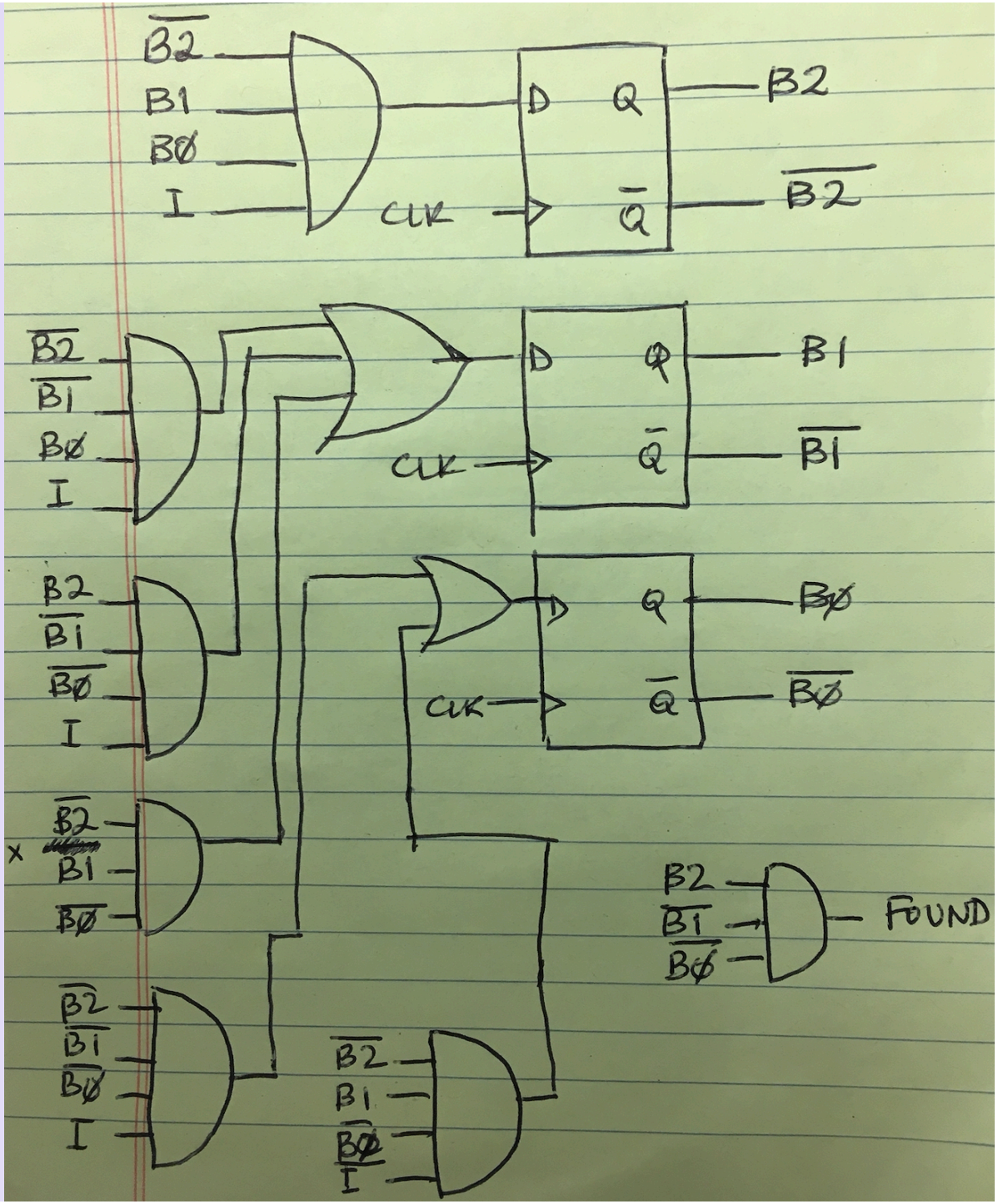
3. Design the Circuit

Note that CLK is the input to ALL the D-FFs' clock inputs. This is a ***synchronous machine***.

Note the use of labels (example: B2 or B0-bar) instead of routing wires all over the place!

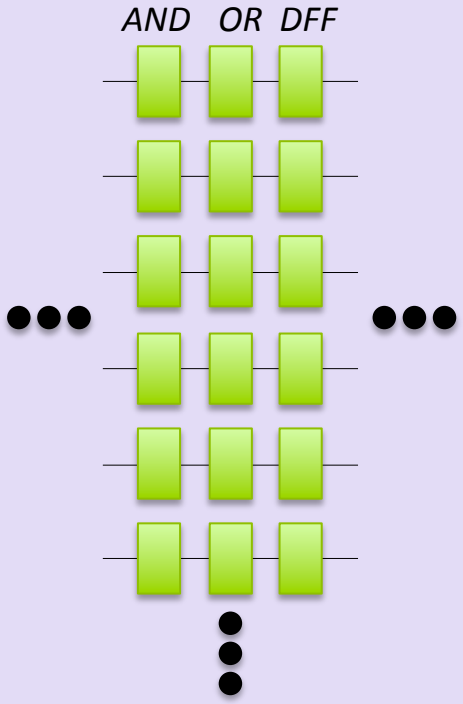
Note that I issued both B*n* and B*n*-bar from all the D-FFs − it makes it easier with the labeling and you won't have to use NOT gates!

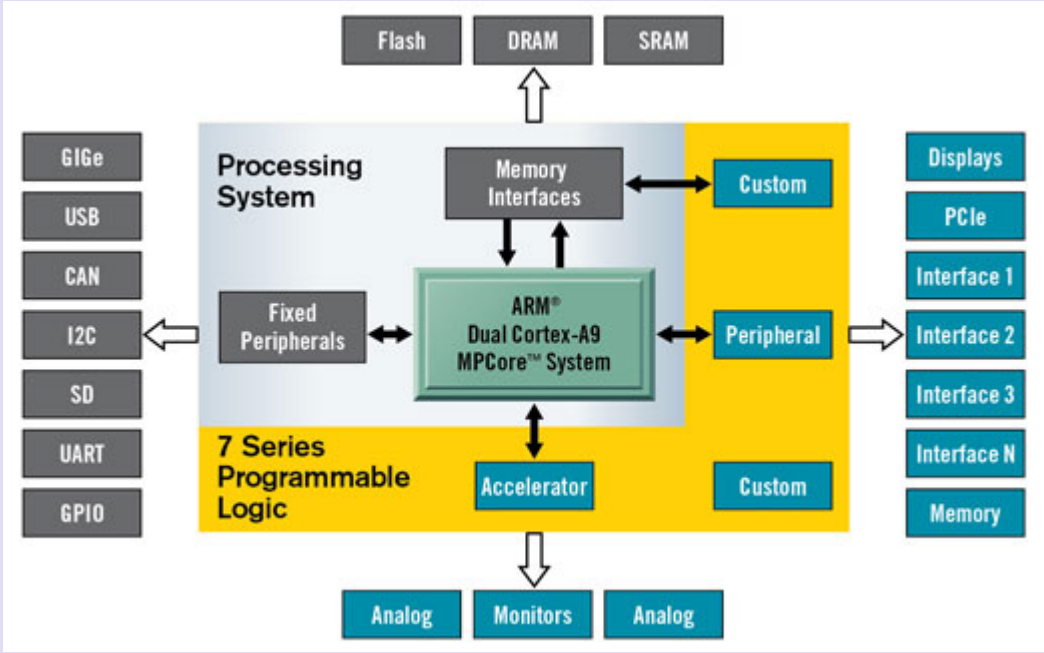
Note that the sole output (FOUND) does **not** need a D-FF because it is **NOT A STATE BIT!**



FPGAs and Programmable Logic

Field-Programmable Gate Arrays





The "One Hot" Method

- Most popularly used in building FSMs
- Give each state it's own D-FF output
  - **# of FFs needed = # of states**
  - You end up using MORE D-FFs, but the implementation is easier to automate
- Inputs to the D-FFs are combinatorial logic that can simplified into a "sum-of-products" type of Boolean expression
  - **No need to go through T.T.s and K-Maps**
- Current CAD software can do this automatically
- Implemented with FPGA integrated circuits

Encoding our States

Per the last example ("1101 Detector"): We had 5 separate states, so we're going to need 5 bits (i.e. 5 DFFs) to describe the states:

NAME		"Regular" Code	"One Hot" Code	OUTPUTS
Initial State	S0	000	00001	
"1"	S1	001	00010	
"11"	S2	010	00100	
"110"	S3	011	01000	
"1101"	S4	100	10000	FOUND

- Advantage of this "One Hot" approach?
  - When we implement the machine with circuits, we can use a D-FF for every state (so, in this example, we'd use 5 of them)

Using the "One Hot" Code to Determine the Circuit Design

- Every state has 1 D-FF
- We can see that (follow the arrows!!):

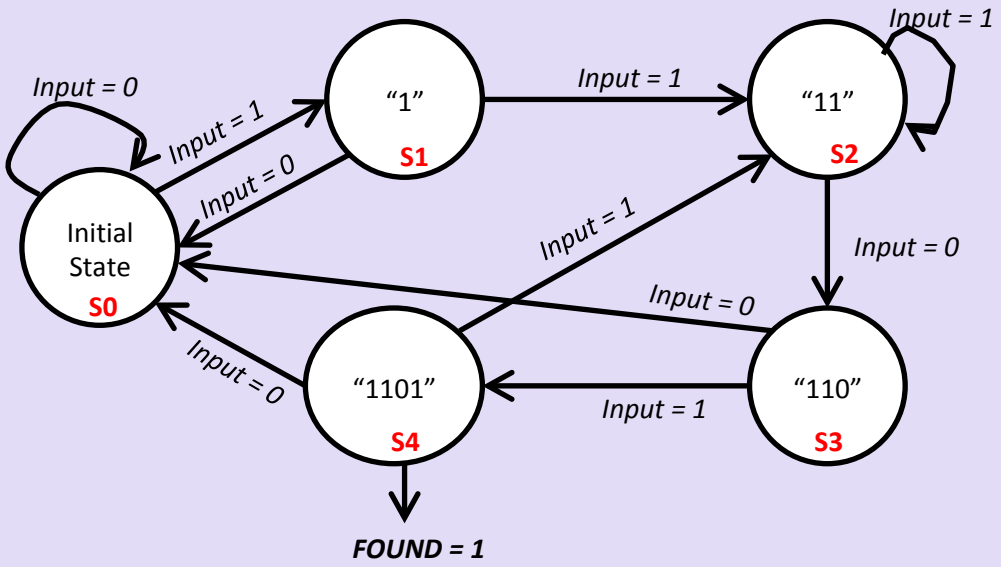
S0* = S0.$\overline{I}$ + S1.$\overline{I}$ + S3.$\overline{I}$ + S4.$\overline{I}$

S1* = S0.I

S2* = S1.I + S2.I + S4.I

S3* = S2.$\overline{I}$

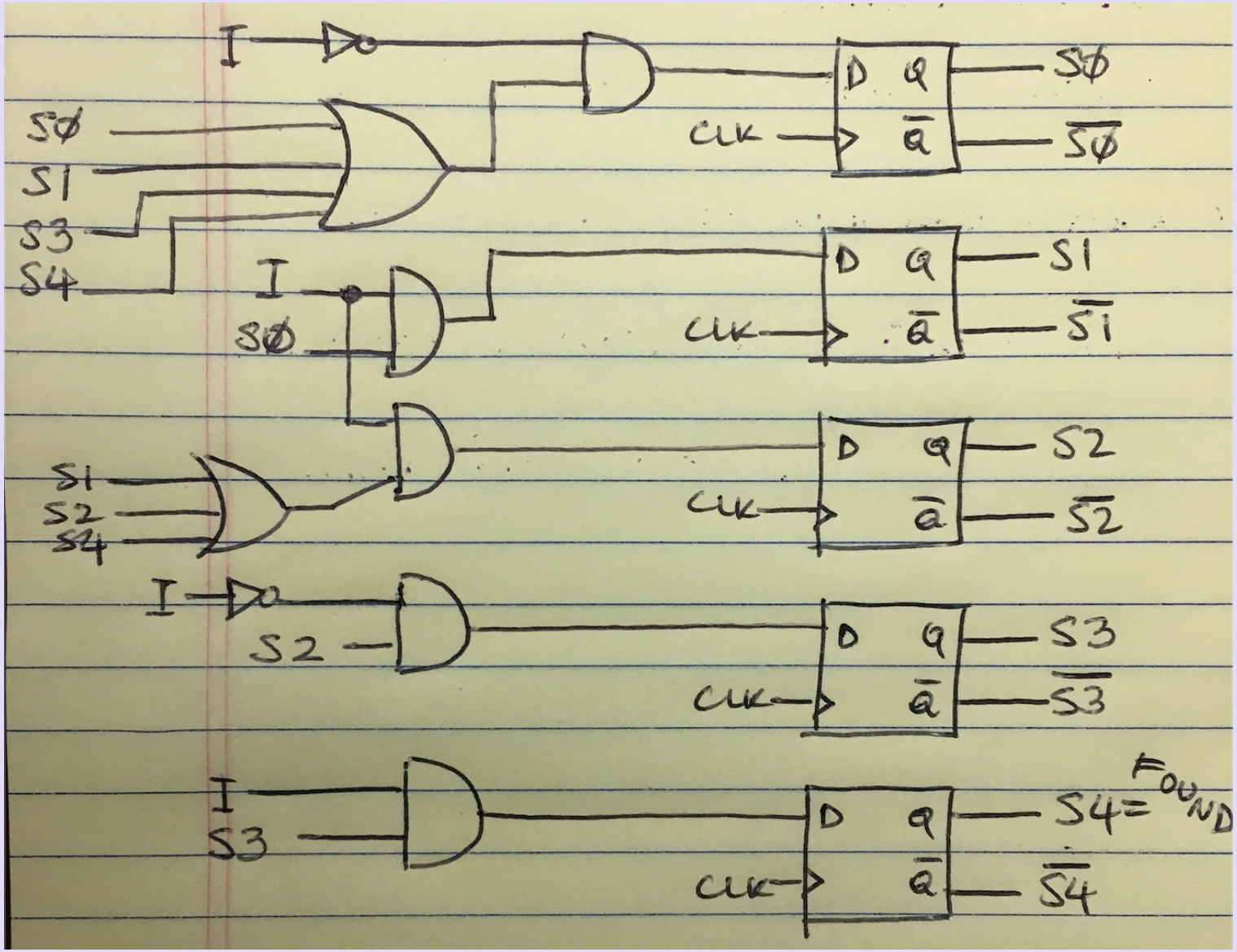
S4* = S3.I



Also, when S4 is True, FOUND is True, i.e. **FOUND = S4**

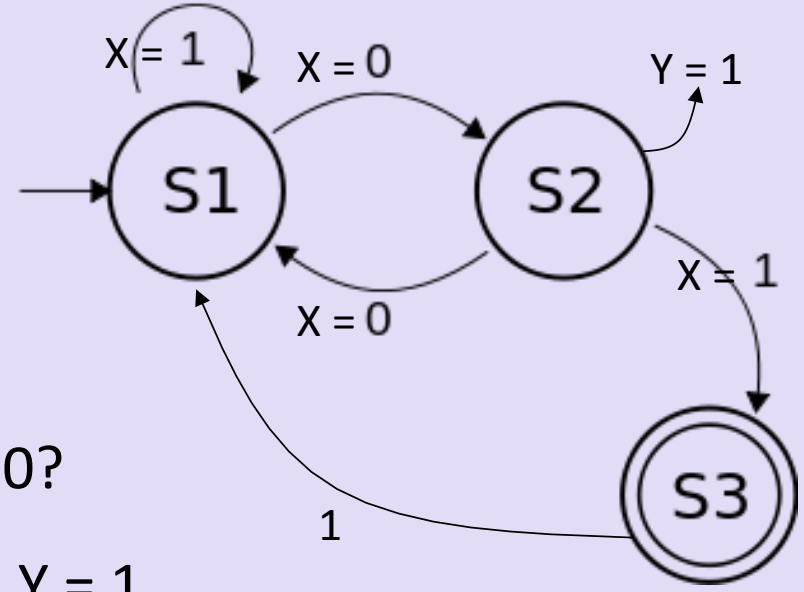
We have now described ALL the outputs of the machine as combinations of certain inputs WITHOUT needing to do T.T. & K-Maps!

Implementing the Circuit For "Detect 1101" FSM Using the "One Hot" Method



FSM Exercise 1

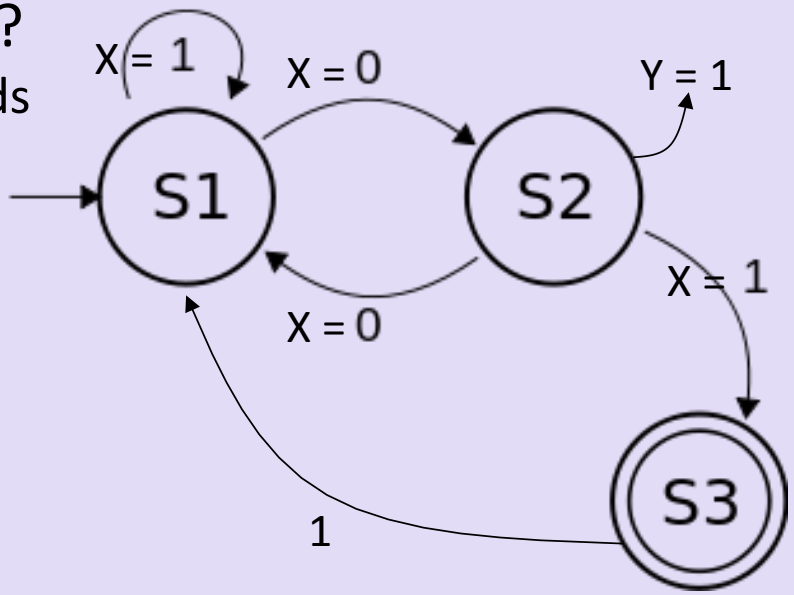
- Given a FSM described with the following state diagram where:
  - The initial state is S1
  - There is only 1 input, X
  - There is only 1 output, Y, and it is initialized to 0



- What state do you end up in if X takes on the sequential values 0110?
- Which of these inputs will result in Y = 1 at the end of their sequences?
  - **A. 0110**
  - **B. 1111101**
  - **C. 0101010**

FSM Exercise 1b

- How many bits do we need to represent all the states in this FSM?
  - Using regular, non-alternative methods
- Write the T.T. for this FSM
- Write the next-state functions for this FSM
- Design the digital logic circuit to implement this FSM, showing all inputs and all outputs



FSM Exercise 2

- Design a FSM that takes in 2 single-bit binary inputs, A and B and always is reset to (begins in) an initial state.

- The machine will move from the initial state only if **A&&B** is true. Once it does that, however, it will go through N states sequentially, once for every time **A||B** is true.

- On the last state, it simply goes to the initial state again and repeats.

A. Draw the state diagram for N = 3.
B. Using the "one-hot method", how many bits do we need to represent all the states in this FSM?
C. Write the next-state functions for this FSM using the approach in B.
D. Design the dig. logic circuit to implement the FSM using your results so far

FSM Exercise 3

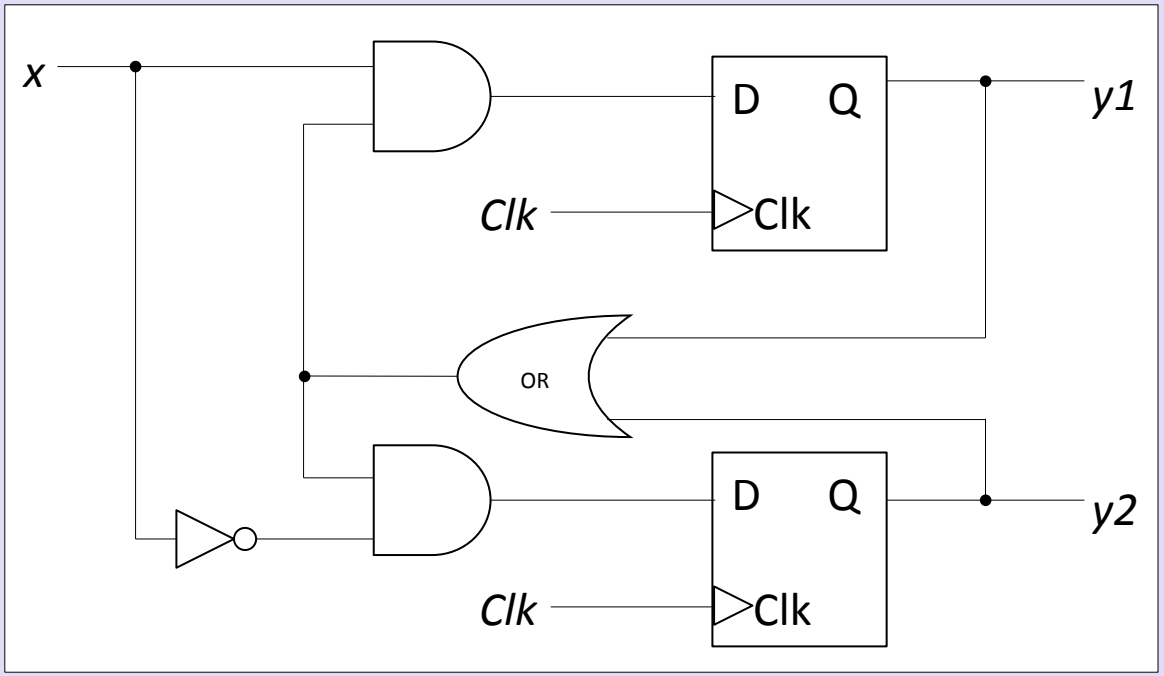
- Consider the FSM circuit, shown here using the "one-hot method"

A. Identify the non-clock inputs and outputs

B. Write the next state equations

C. Write the T.T. for this FSM

D. Draw the state diagram for this FSM



YOUR TO-DOs

- Lab 8

</LECTURE>