

# **Finite State Machines**

**CS 64: Computer Organization and Design Logic  
Lecture #15  
Fall 2018**

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

# Administrative

- The Last 2 Weeks of CS 64:

Date	L #	Topic	Lab	Lab Due
11/26	13	Combinatorial Logic, Sequential Logic	8 (CL+SL)	Sun. 12/2
11/28	14	Sequential Logic		
12/3	15	FSM	9 (FSM)	Fri. 12/7
12/5	16	FSM, CS Ethics & Impact	10 (Ethics)	Fri. 12/7

# Administrative

---

- New Lab #9 issued tomorrow
  - Due Friday
- Practice Exam will be online v. soon
  - Look for Piazza announcement
- Your final exam is on  
**Wednesday, Dec. 12<sup>th</sup> at 8:00 AM**
  - In this classroom
  - Be ON TIME! Arrive 10 minutes EARLY!

# Lecture Outline

---

- Finite State Machines
  - Moore vs. Mealy types
  - State Diagrams
  - Figuring out a circuit for a FSM

If a combinational logic circuit is an implementation of a ***Boolean function***,

then a sequential logic circuit can be considered an implementation of a ***finite state machine***.

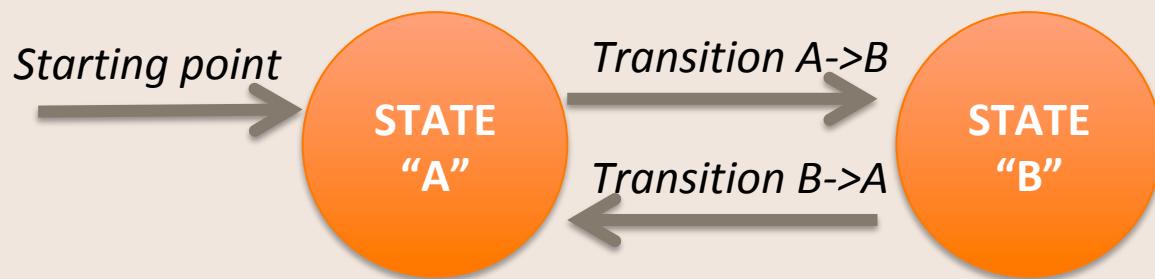
# Finite State Machines (FSM)

---

- An **abstract machine** that can be in **exactly one of a finite number of states at any given time**
- It's a very simple model of a computational machine, unlike Pushdown Automatons and Turing Machines
  - You'll discover these in other CS upper-div classes

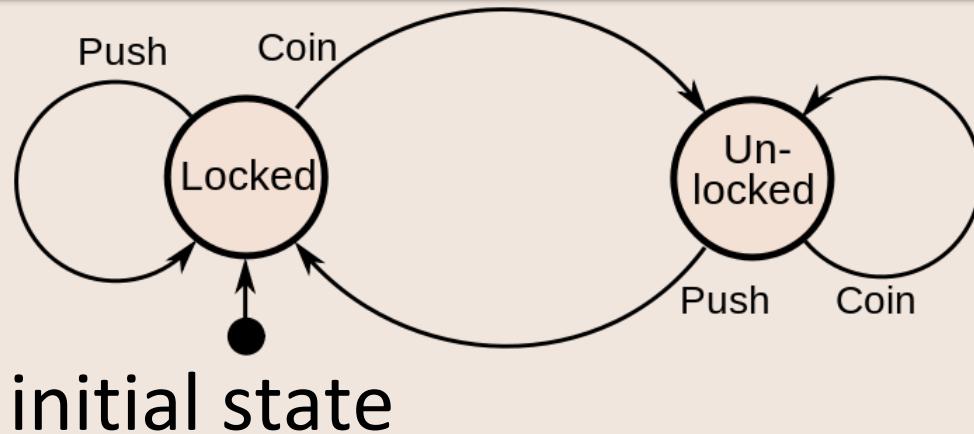
# Finite State Machines (FSM)

- The FSM can change from one state to another in **response to some external inputs**
- The change from one state to another is called a **transition**.



- An FSM is defined by a list of its states, its initial state, and the conditions for each transition.

# Example of a Simple FSM: The Turnstile

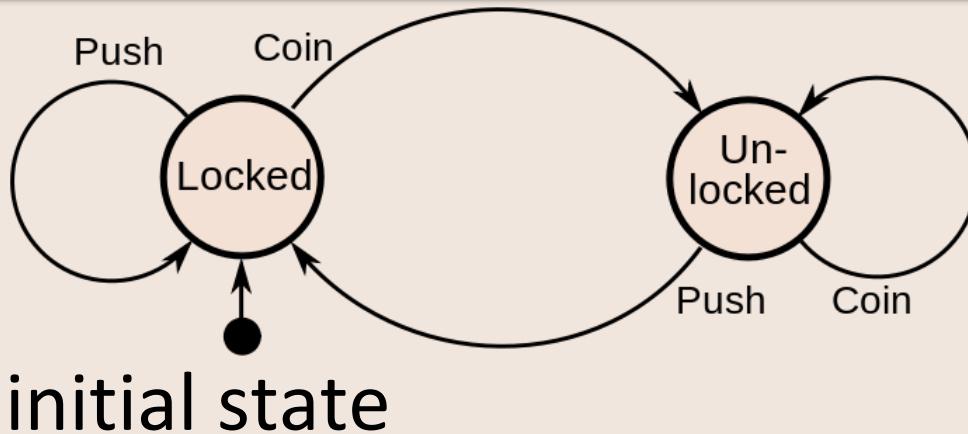


*State Transition Table*

Current State	Input	Next State	Output
Locked	Coin	Unlocked	Unlocks the turnstile so that the customer can push through.

# Example of a Simple FSM: The Turnstile

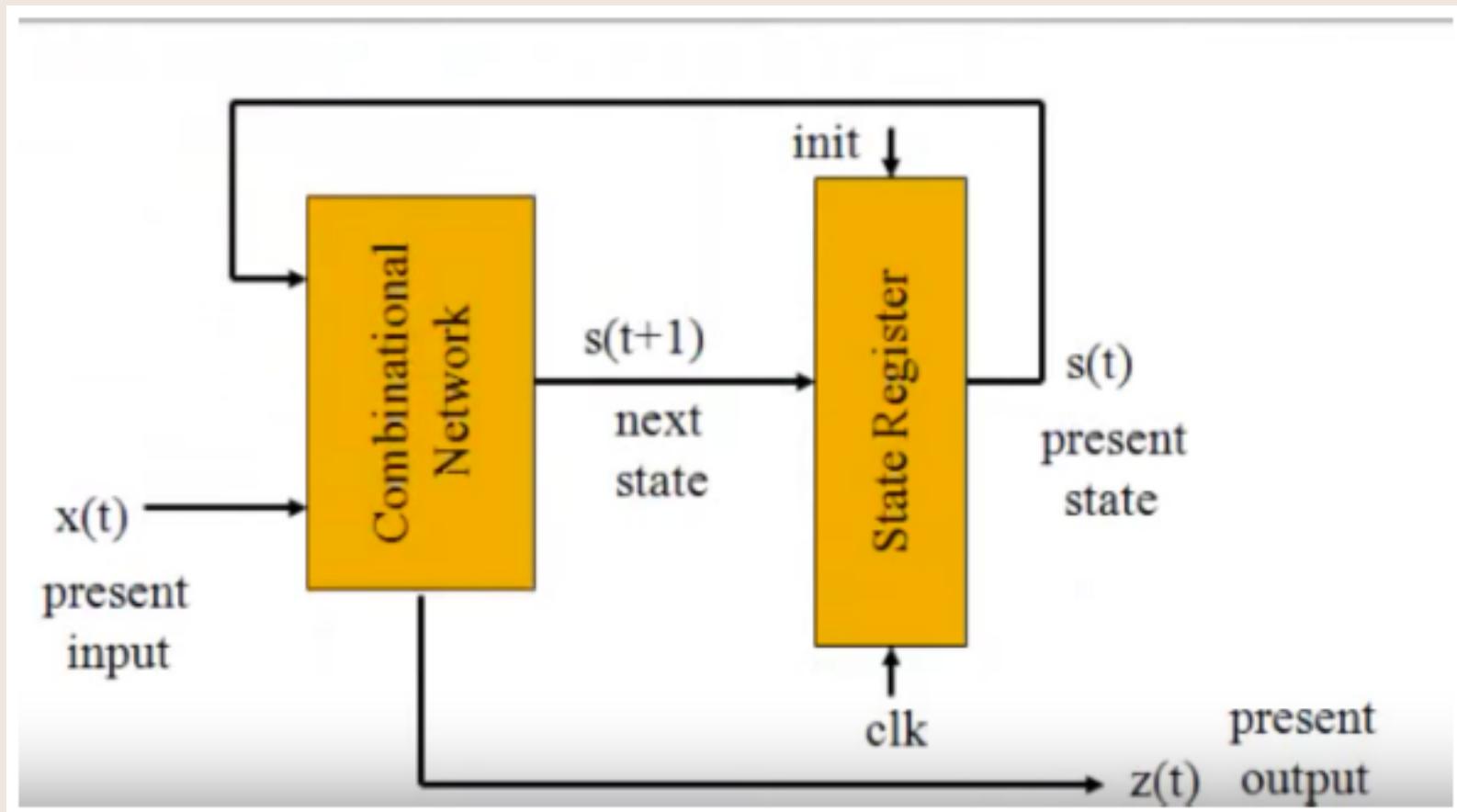
This is called a  
*state diagram*



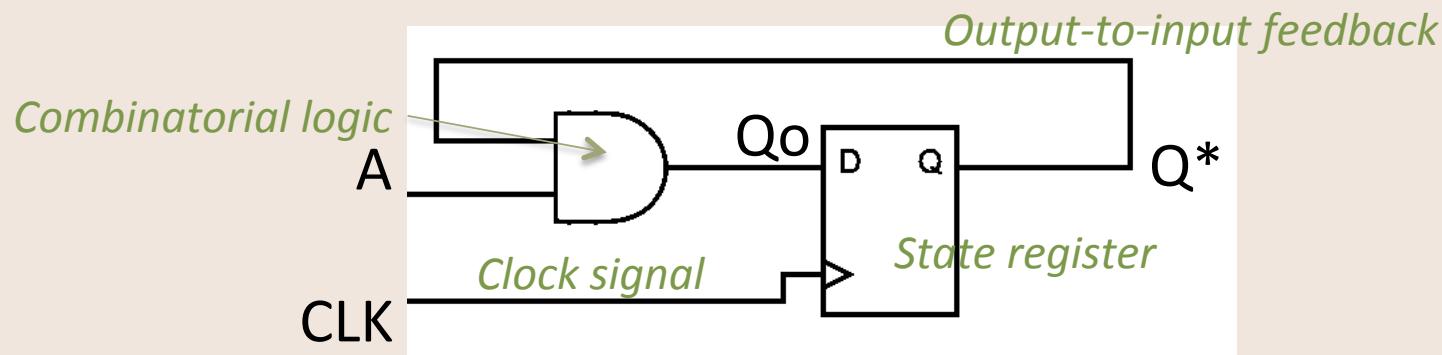
*State Transition Table*

Current State	Input	Next State	Output
Locked	Coin	Unlocked	Unlocks the turnstile so that the customer can push through.
Locked	Push	Locked	Nothing – you’re locked! ☺
Unlocked	Coin	Unlocked	Nothing – you just wasted a coin! ☺
Unlocked	Push	Locked	When the customer has pushed through, locks the turnstile.

# General Form of FSMs



# Example



$$Q^* = Q_o \cdot A$$

On the next rising edge of the clock, the output of the D-FF  $Q$  ( $Q^*$ ) will become the previous value of  $Q$  ( $Q_o$ ) **AND** the value of input  $A$

# FSM Types

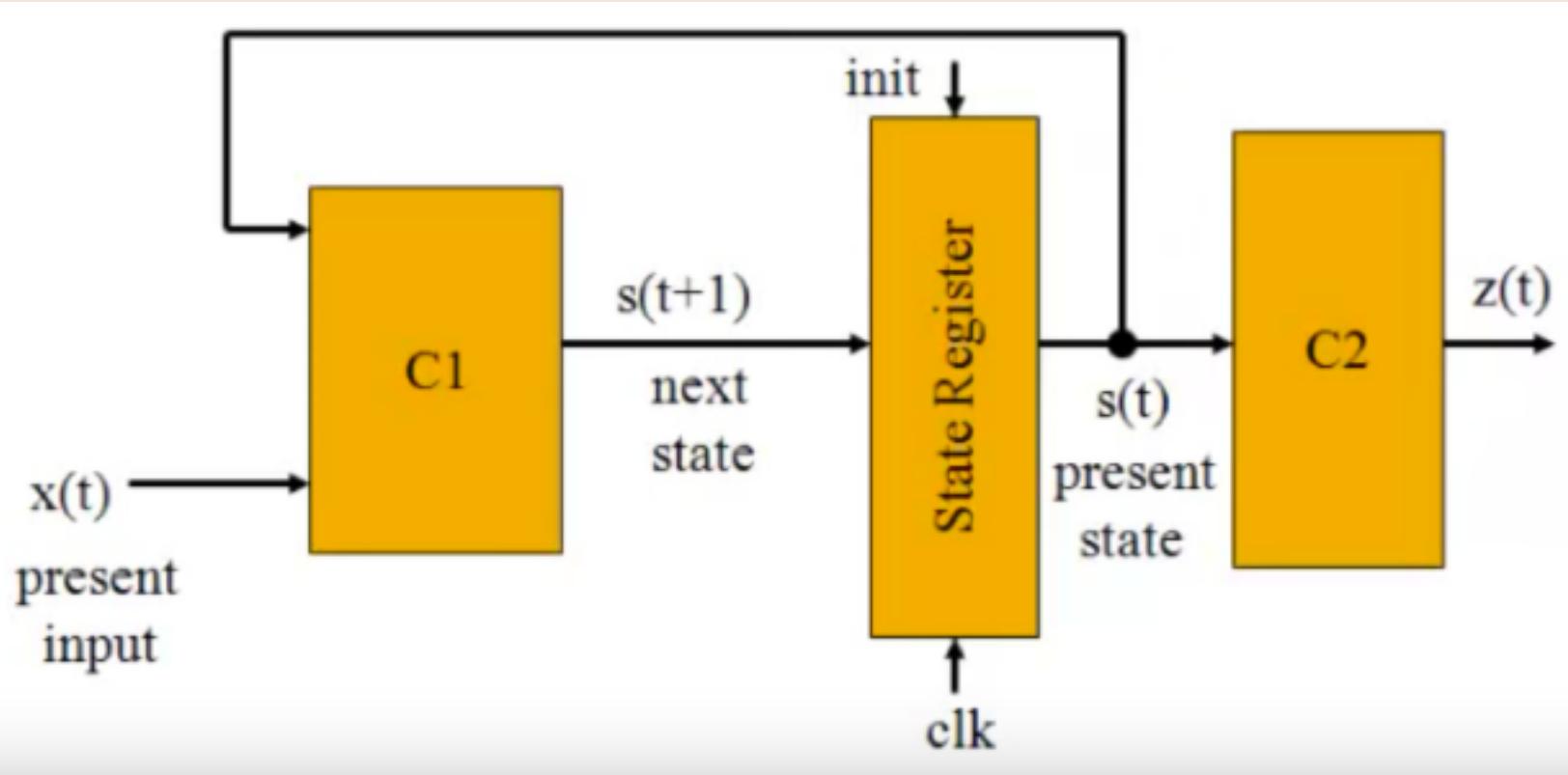
---

**There are 2 types/models of FSMs:**

- **Moore machine**
  - Output is function of present state only
- **Mealy machine**
  - Output is function of present state *and* present input

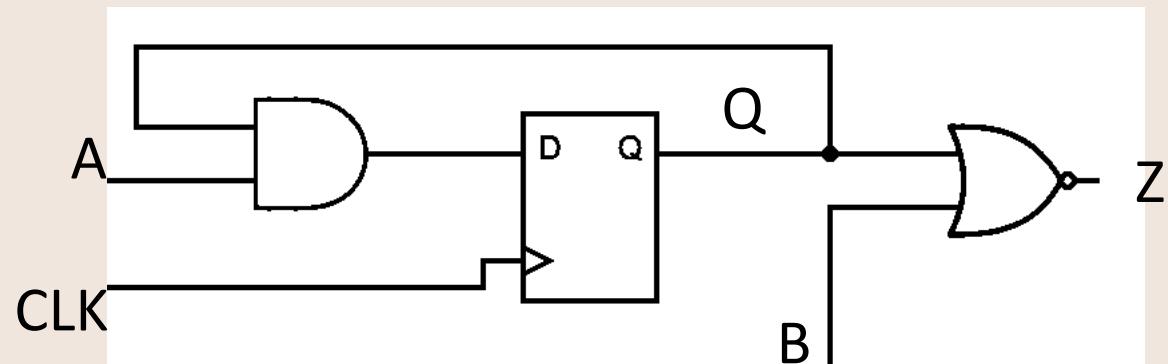
# Moore Machine

***Output is function of present state only***



# Example of a Moore Machine (with 1 state)

***Output is function of present state only***



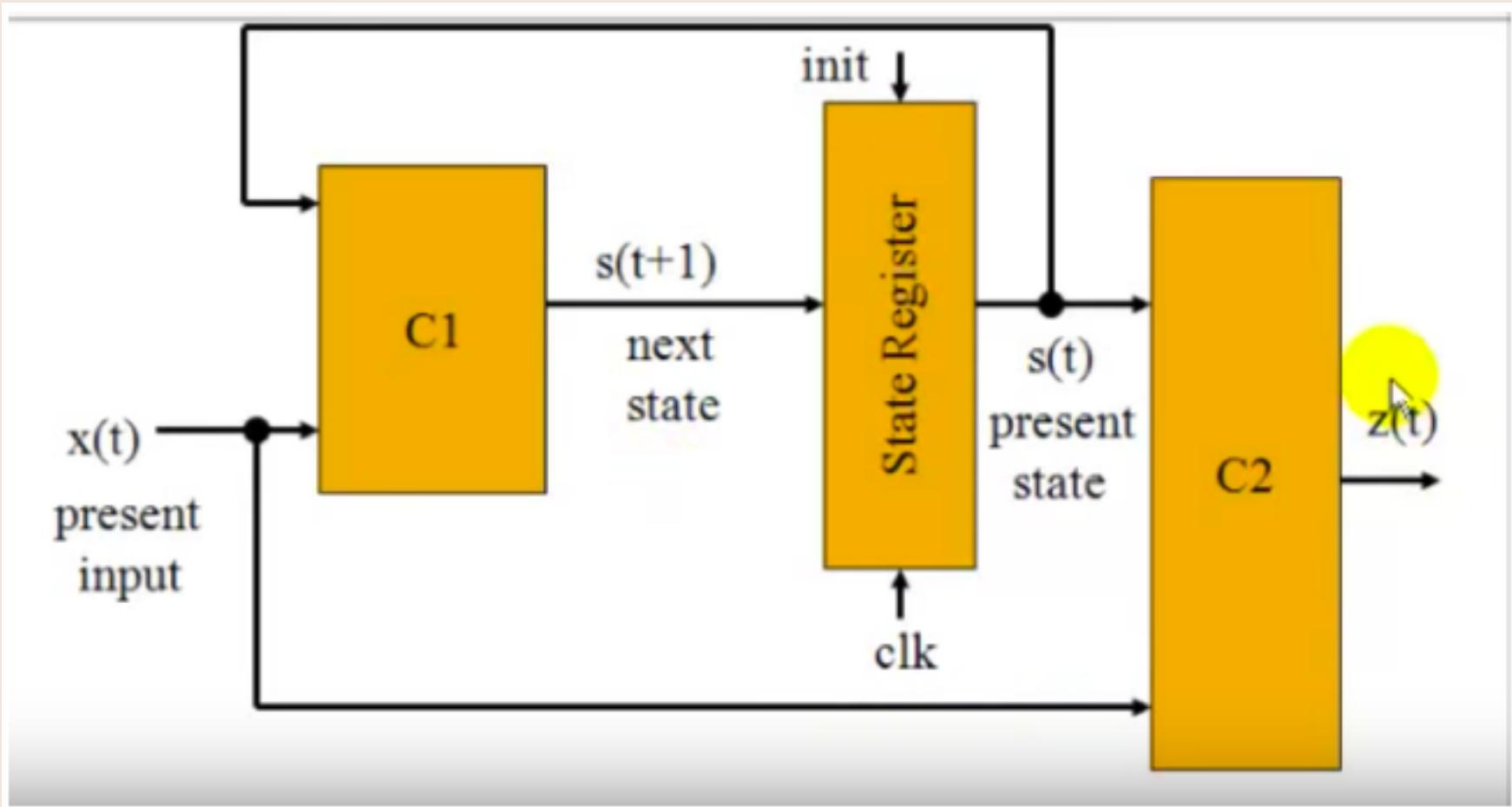
$$Z = \overline{(Q^* + B)} = \overline{(Q_0 \cdot A + B)}$$

On the next rising edge of the clock, the output of the entire circuit (Z) will become  
(the previous value of Q ( $Q_0$ ) **AND** the value of input A) **NOR** B

**NOTE: CLK is NOWHERE IN THE EQUATION!!!**

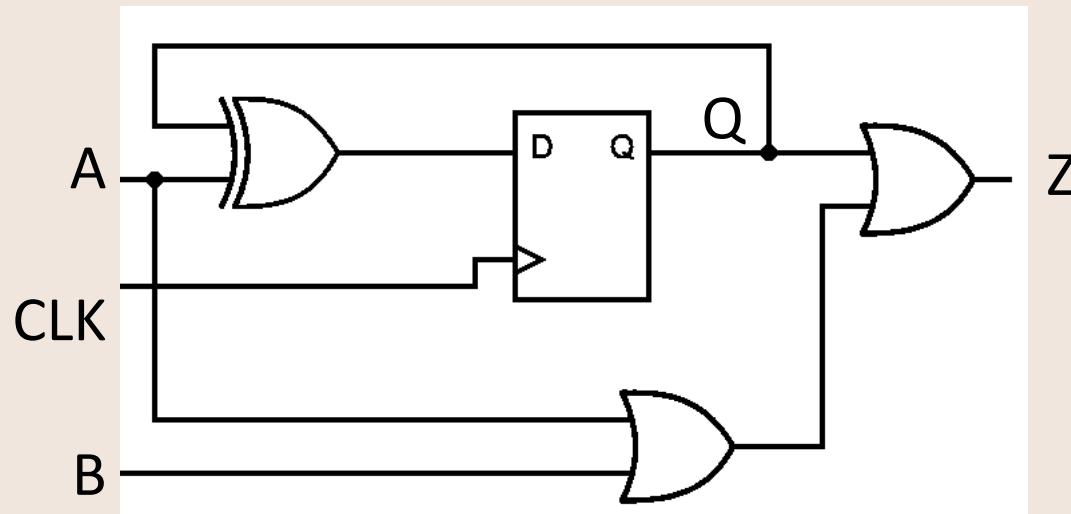
# Mealy Machine

*Output is function of present state and present input*



# Example of a Mealy Machine (with 1 state)

*Output is function of present state and present input*



$$Z = (Q^* + A + B) = (Q_0 \text{ XOR } A) + (A + B)$$

On the next rising edge of the clock, the output of the entire circuit (Z) will become ...etc...

# Example of a Moore Machine

## WASHER\_DRYER

- Let's "build" a sequential logic FSM that acts as a controller to a *simplistic* washer/dryer machine
- This machine takes in various inputs in its operation (we'll only focus on the following sensor-based ones):

*Coin is in (vs it isn't in)*

*Soap is present (vs it's used up)*

*Clothes are still wet (vs clothes are dry)*

- This machine also issues 1 output while running:

"Done" indicator

# Machine Design

---

- We want this machine to have **4 distinct states** that we go from one to the next in this sequence:

## 1. Initial State

- Where we are when we are waiting to start the wash

## 2. Wash

- Where we wash with soap and water

## 3. Dry

- Where we dry the clothes

## 4. Done

# Combining the Inputs

*Coin is in (vs it isn't in)*

*Soap is no longer detected (vs it's still there)*

*Clothes are now dry (vs clothes are still wet)*

- Let's create a variable called **GTNS** (i.e. Go To Next State)
- GTNS is 1 if **any** of the following is true:
  - Coin is in
  - Soap is no longer detected
  - Clothes are now dry
  - **I assume that these 3 inputs to be mutually exclusive**

# What's Going to Happen? 1/2

*Coin is in (vs it isn't in)*  
*Soap is no longer detected (vs it's still there)*  
*Clothes are now dry (vs clothes are still wet)*

- We start at an “**Initial**” state whenever we start up the machine
  - Let’s also assume this stage is when you’d put in the soap and clothes
  - Once input “Coin is in” is 1, GTNS is now 1
  - This event triggers leaving the current state to go to the next state
- This is followed by the next state, “**Wash**”
  - “Coin inserted” is now 0 at this point (so GTNS goes back to 0)
  - While soap is still present, GTNS goes back to 0
  - When the input “Soap is no longer present” goes to 1, GTNS goes to 1
  - This event triggers leaving the current state to go to the next state

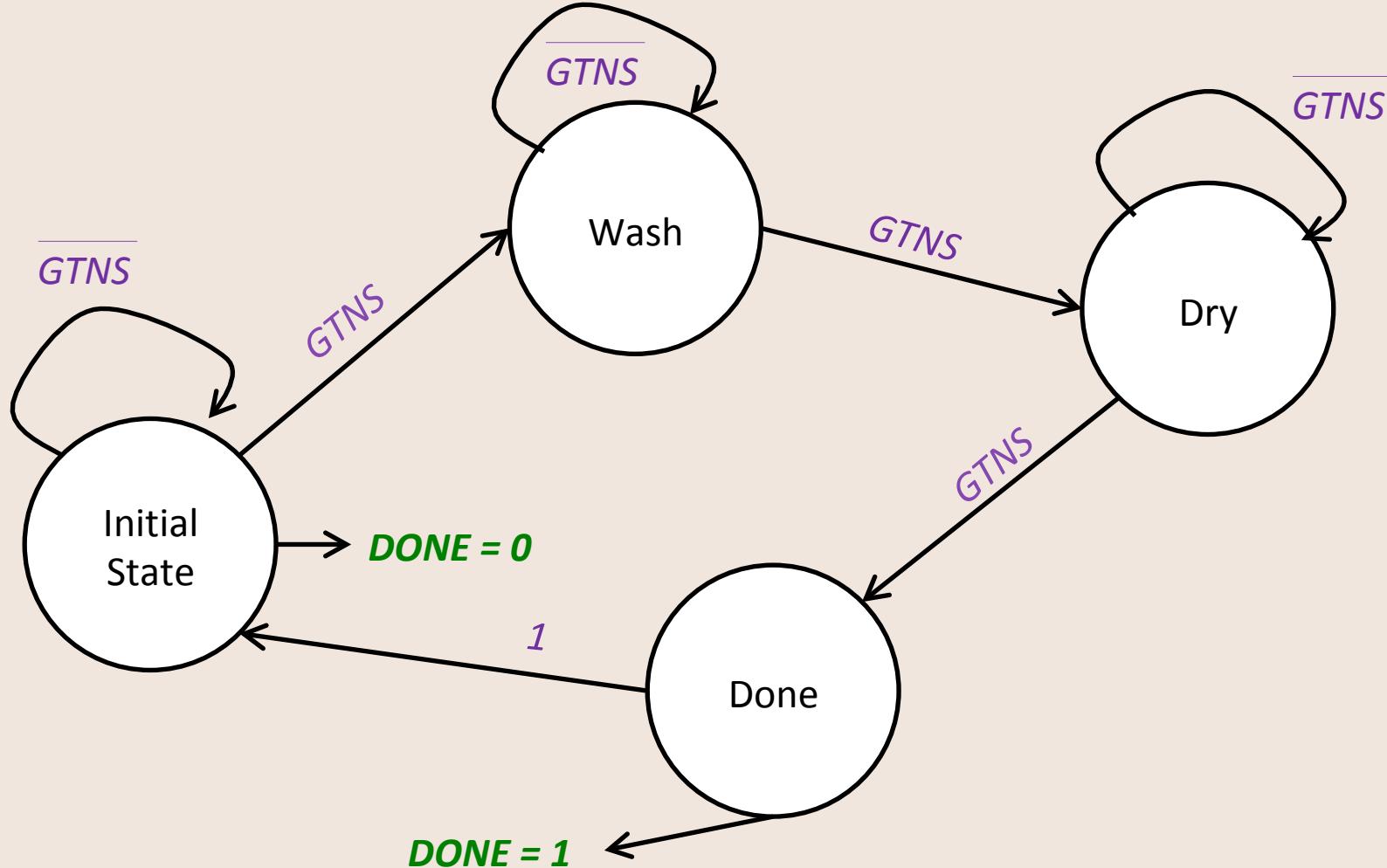
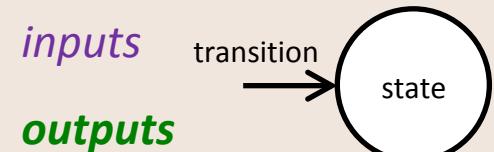
# What's Going to Happen? 2/2

*Coin is in (vs it isn't in)*  
*Soap is no longer detected (vs it's still there)*  
*Clothes are now dry (vs clothes are still wet)*

- This is followed by the next state, “**Dry**”
  - This new state sets an output that triggers a timer
  - The input “Soap is no longer present” goes to 0, so GTNS is 0 also
  - While the input “Clothes are now dry” is 0 , GTNS remains at 0 too
  - When the input “Clothes are now dry” is 1, GTNS changes to 1
  - This event triggers leaving the current state to go to the next state
- This is followed by the next and last state, “**Done**”
  - When you’re here, you go back to the “initial” state
  - No inputs to consider: you do move this regardless

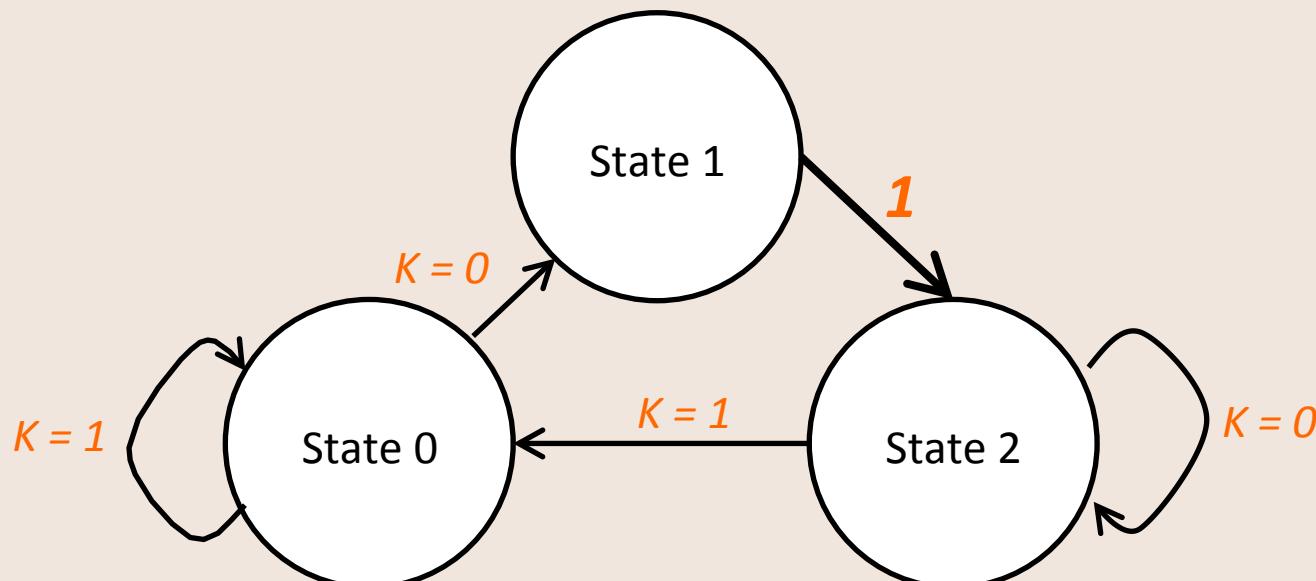
# State Diagram for Washer-Dryer Machine

$GTNS = COIN\_IN + NO\_SOAP + CLTHS\_DRY$



# Unconditional Transitions

- Sometimes the transition is unconditional
  - Does not depend on any input – it just happens
- We then diagram this as a “1” (for “always does this”)



# Representing The States

- How many bits do I need to represent all the states in this Washer-Dryer Machine?
- There are 4 unique states (including “init”)
  - So, 2 bits
- If my state machine will be built using a memory circuit (most likely, a D-FF), how many of these should I have?
  - 2 bits = 2 D-FFs
- There is another scheme to do this called “One Hot Method”.
  - More on this later...

State	S1	S0
Initial	0	0
Wash	0	1
Rinse	1	0
Dry	1	1

# Example of a Moore Machine 2

---

## **DETECT\_1101**

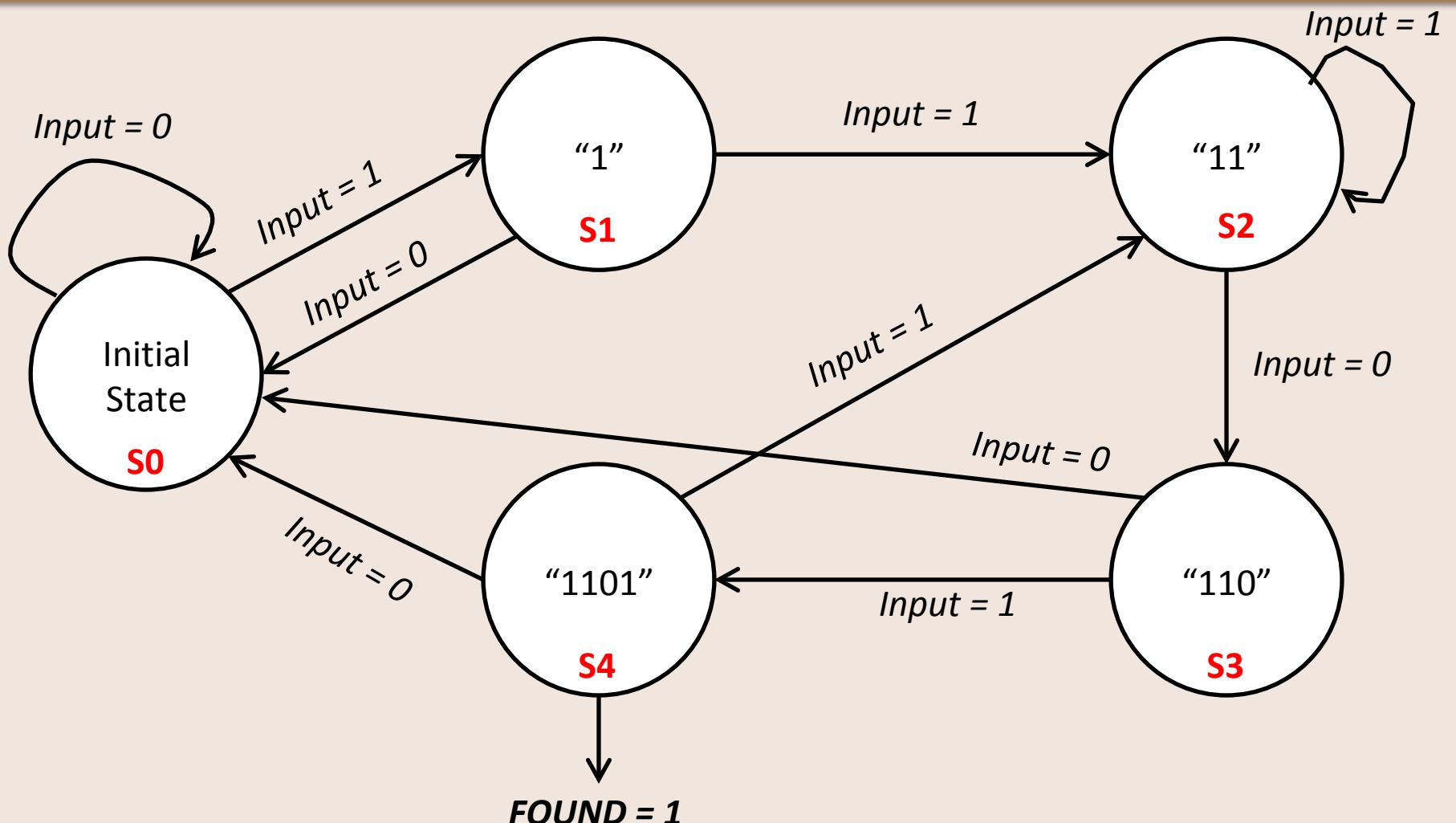
- Let's build a sequential logic FSM that **always** detects a specific serial sequence of bits: **1101**
- We'll start at an "Initial" state ( $S_0$ )
- We'll first look for a **1**. We'll call that "State 1" ( $S_1$ )
  - Don't go to  $S_1$  if all we find is a **0**!
- We'll then keep looking for another **1**. We'll call that "State 11" ( $S_2$ )

# Example of a Moore Machine 2

## DETECT\_1101

- Then... a **0**. We'll call that "State 110" (S3)
- Then another **1**.  
We'll call that "State 1101"(S4) – this will also output a **FOUND** signal
- We will always be detecting "1101" (it doesn't end)  
So, as SOON as S4 is done, we keep looking for 1s or 0s
- Example: if the input stream is **111101110101101000011111011011**  
we detect "1101" at                          ↑      ↑      ↑                          ↑      ↑

# State Diagram 2



# Representing The States

- How many bits do I need to represent all the states in this “Detect 1101” Machine?
- There are 5 unique states (including “init”)
  - So, 3 bits
- How many D-FFs should I have to build this machine?
  - 3 bits = 3 D-FFs

State	B2	B1	B0
Initial	0	0	0
Found “1”	0	0	1
Found “11”	0	1	0
Found “110”	0	1	1
Found “1101”	1	0	0

# 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)

	CURRENT STATE			<i>INPUT(S)</i>	NEXT STATE			<i>OUTPUT(S)</i>
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

## 2. K-Maps for $B_2^*$ and $B_1^*$

- You need to do this for all 3 state outputs
- $B_2^* = \neg B_2 \cdot B_1 \cdot B_0 \cdot I$ 
  - No further simplification
- $B_1^* = \neg B_2 \cdot \neg B_1 \cdot B_0 \cdot I + B_2 \cdot \neg B_1 \cdot \neg B_0 \cdot I + \neg B_2 \cdot B_1 \cdot \neg B_0$

$B_2 \cdot B_1$ $B_0 \cdot I$	00	01	11	10
00				
01				
11			1	
10				

$B_2 \cdot B_1$ $B_0 \cdot I$	00	01	11	10
00		1		
01		1		1
11	1			
10				

## 2. K-Map for $B_0^*$ Output FOUND

- $$\begin{aligned}B_0^* &= \overline{B_2} \cdot \overline{B_1} \cdot \overline{B_0} \cdot I \\&+ \overline{B_2} \cdot B_1 \cdot \overline{B_0} \cdot \overline{I}\end{aligned}$$

$B_2 \cdot B_1$ $B_0 \cdot I$	00	01	11	10
00			1	
01		1		
11				
10				

- $$\begin{aligned}FOUND &= B_2 \cdot \overline{B_1} \cdot \overline{B_0} \\&- \text{Note that FOUND does not need a K-Map. It is always "1" (i.e. True) when we are in state S4 (i.e. when } B_2=1, B_1=0, B_0=0\text{)}\end{aligned}$$

### 3. Design the Circuit

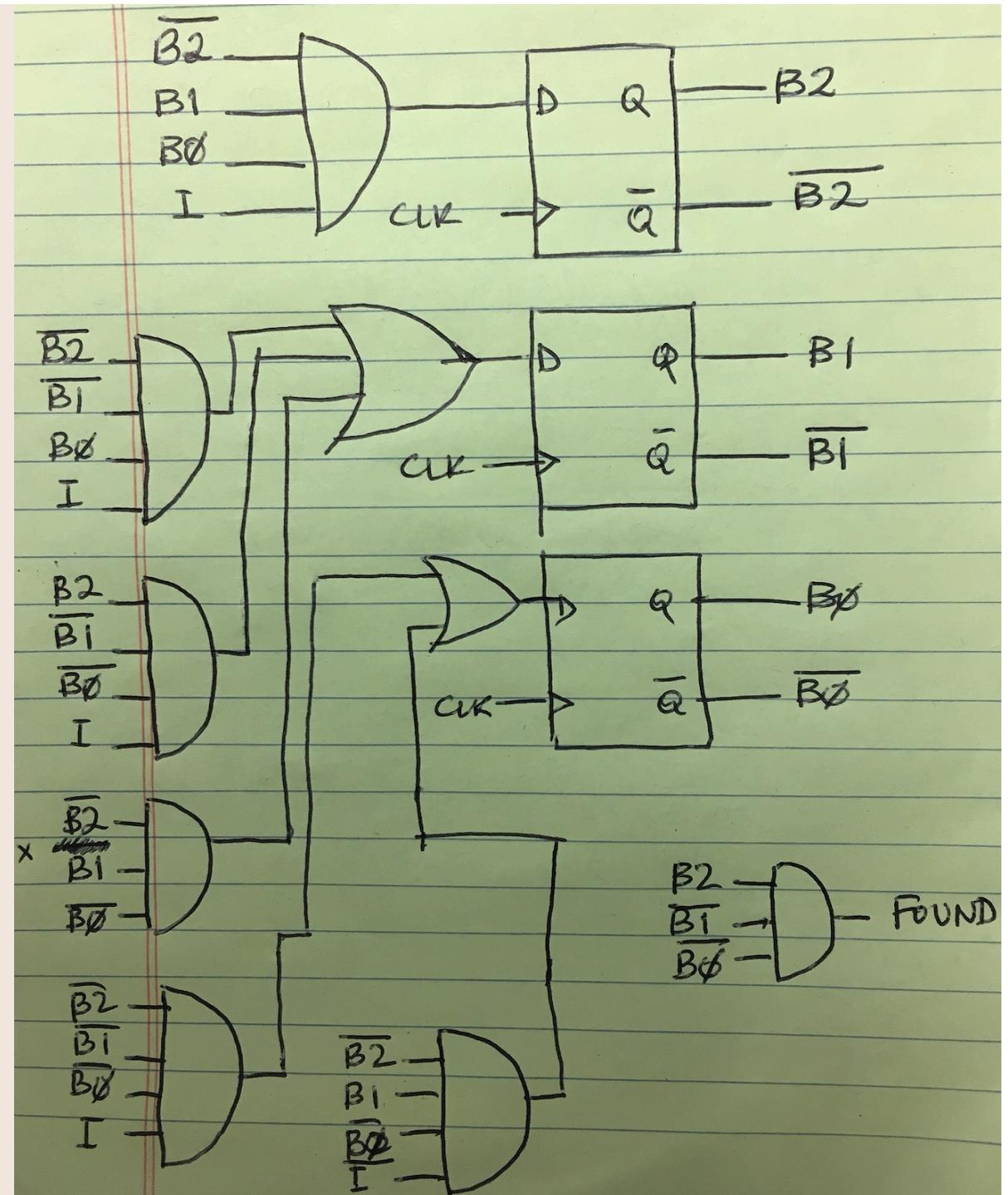
Note that CLK is the input to ALL the D-FFs' clock inputs. This is a **synchronous machine**.

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!**

12/3/18



</LECTURE>