Finite State Machines

CS 64: Computer Organization and Design Logic
Lecture #16
Winter 2020

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

Administrative

- Lab 8 due tomorrow

- Final Exam Announcements
  - Will be 2 hours long – not 3 hours
  - From 7:30 PM – 9:30 PM

  - Practice Exam is on our website

- I will have office hours on Friday from 1 – 2 pm

What’s on the Final Exam?

What’s on It?

- Everything we’ve done this quarter, incl. this week’s lectures

What Should I Bring?

- Your pencil(s), eraser, MIPS Reference Card (printed on **1 page**)
- THAT’S ALL!

What Else Should I Know/Do?

- **The exam is 2 hours long!** **(DSP time will match accordingly)**
- ***IMPORTANT***: Come to the classroom 5-10 minutes EARLY
- **If you are late, I may not let you take the exam**
- ***IMPORTANT***: Use the bathroom before the exam – once inside, you cannot leave
- I will have some of you re-seated
- Bring your UCSB ID

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)

- A **State** = An output or collection of outputs of a digital "machine"

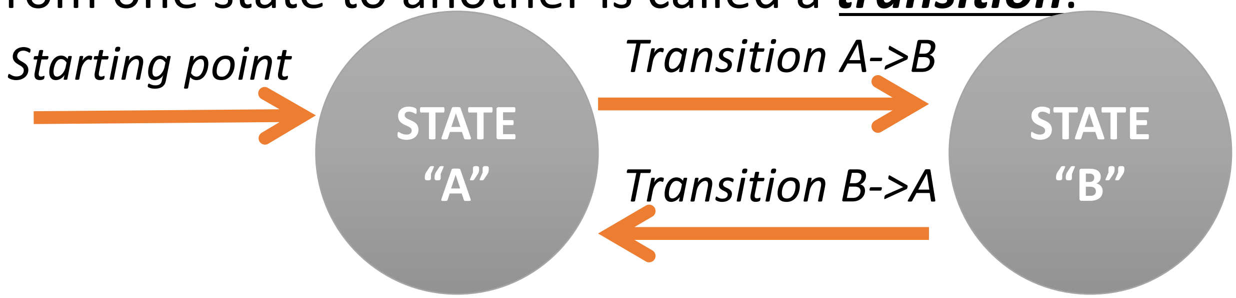
- A **Machine** = A computational entity that predictably works based on one or more input conditions and yields a logical output

- A Finite State Machine: An **abstract machine** that can be in **exactly one of a finite number of states at any given time**

Finite State Machines (FSM)

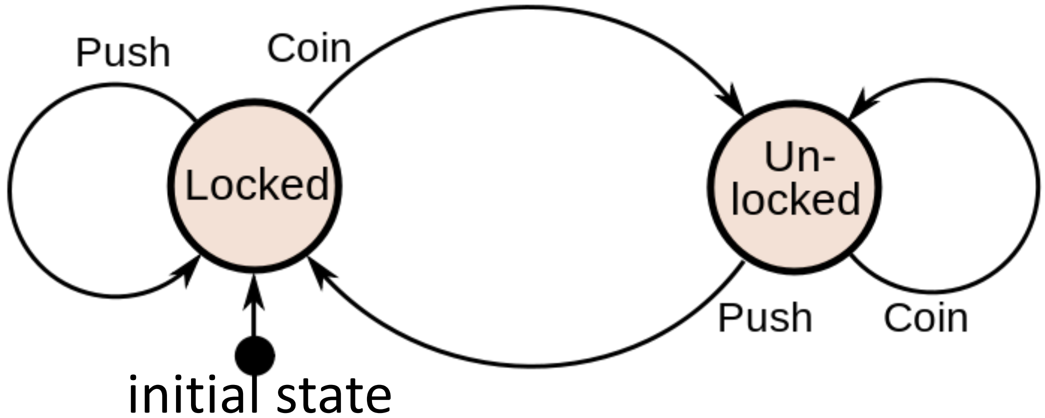
- The FSM can change from one state to another in **response to some *external inputs***

- 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

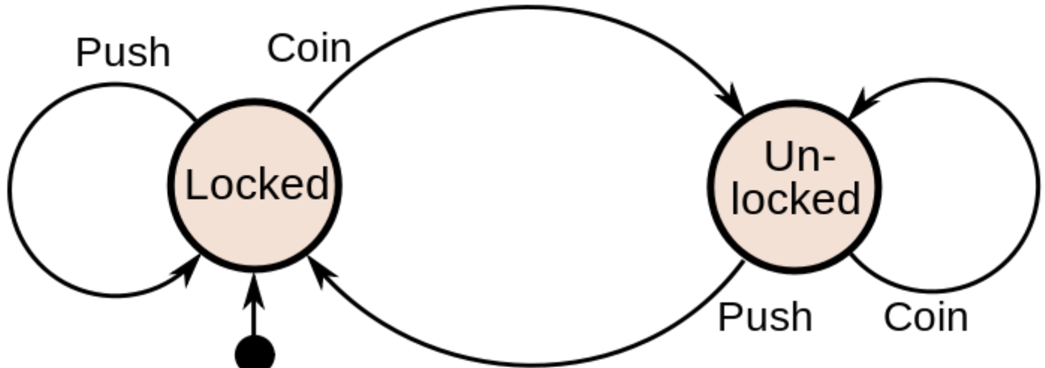


State Transition Table

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*** →

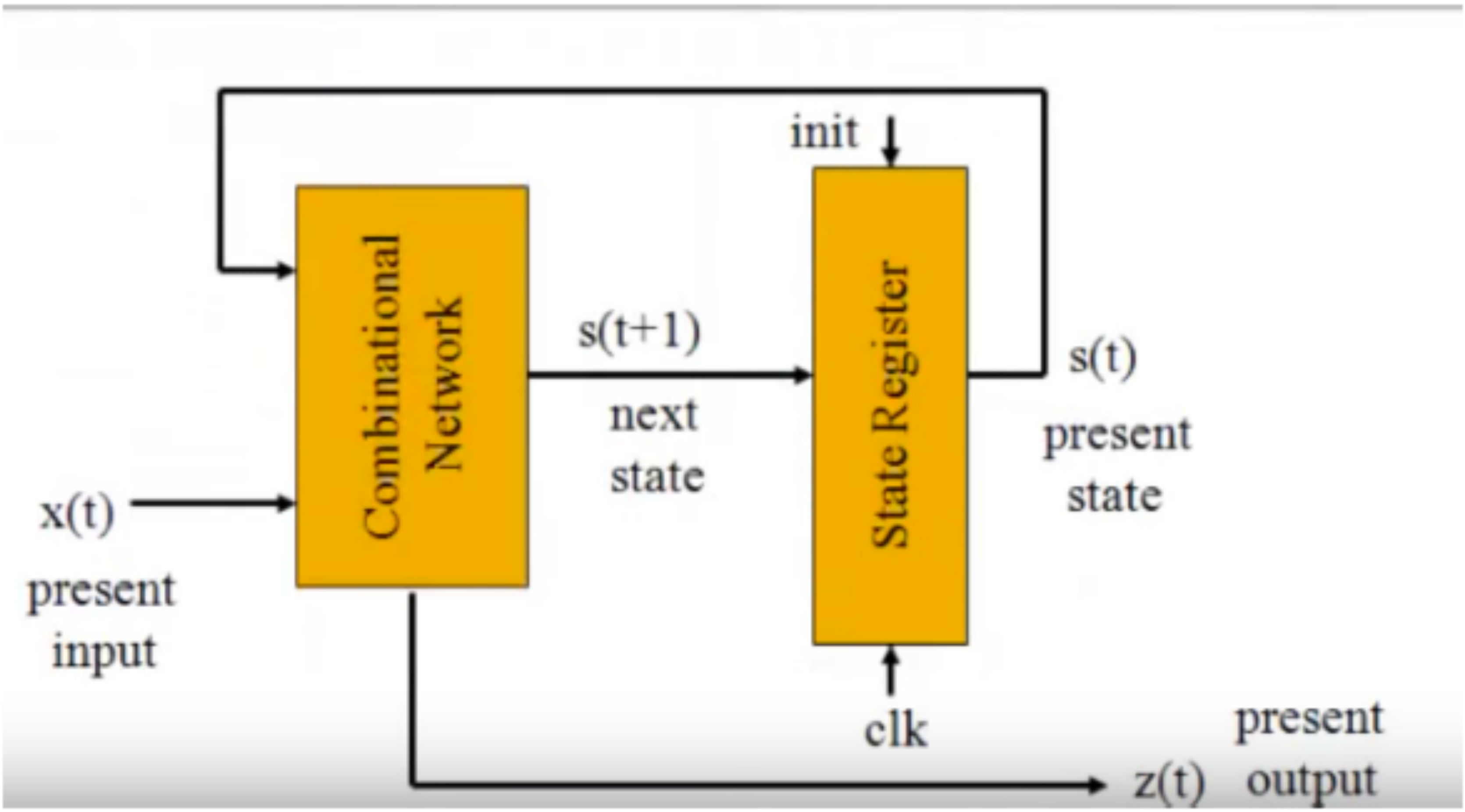


State Transition Table

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.

Source: Wikipedia

General Form of FSMs



FSM Types

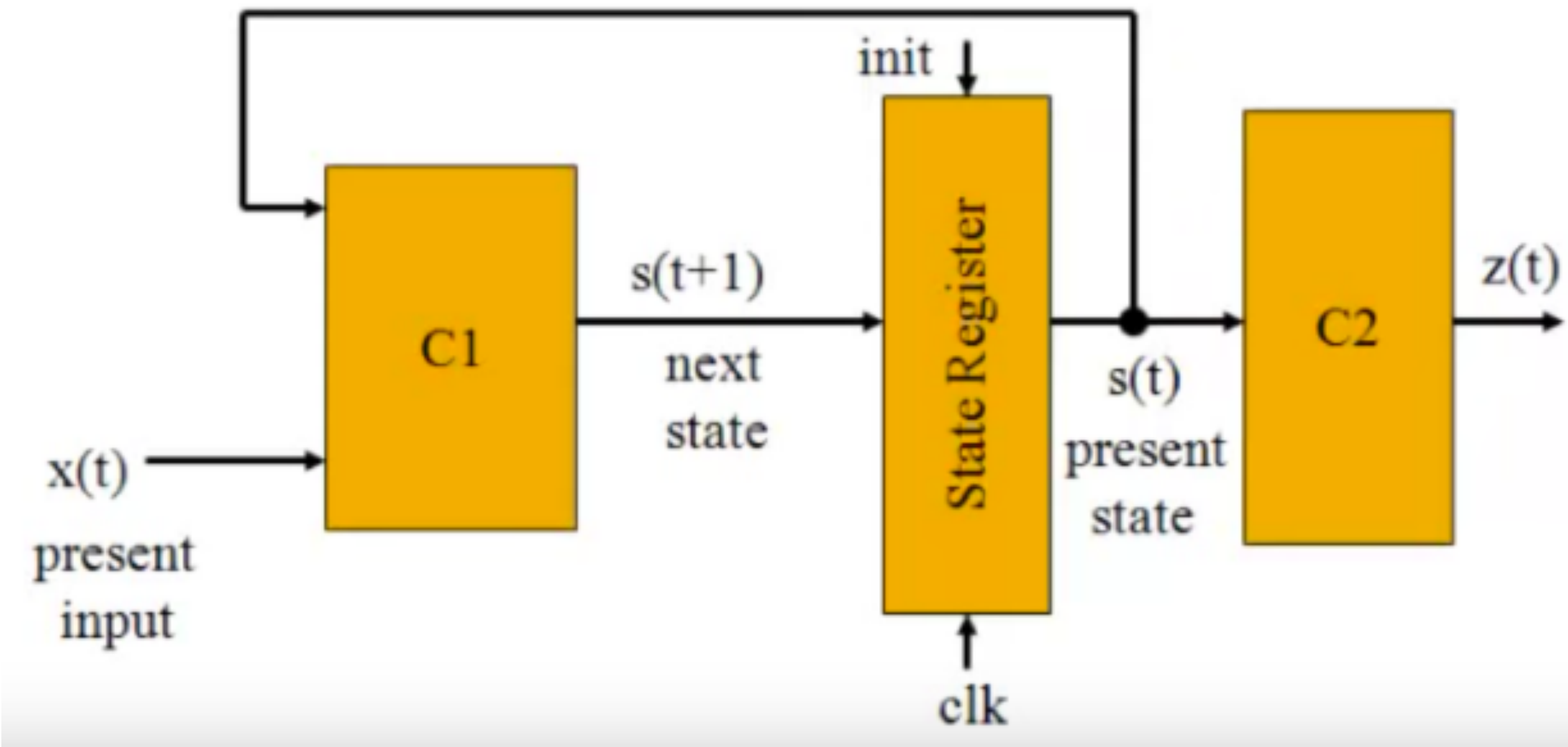
There are 2 types/models of FSMs:

- **Moore machine**
  - Output is function of present state only

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

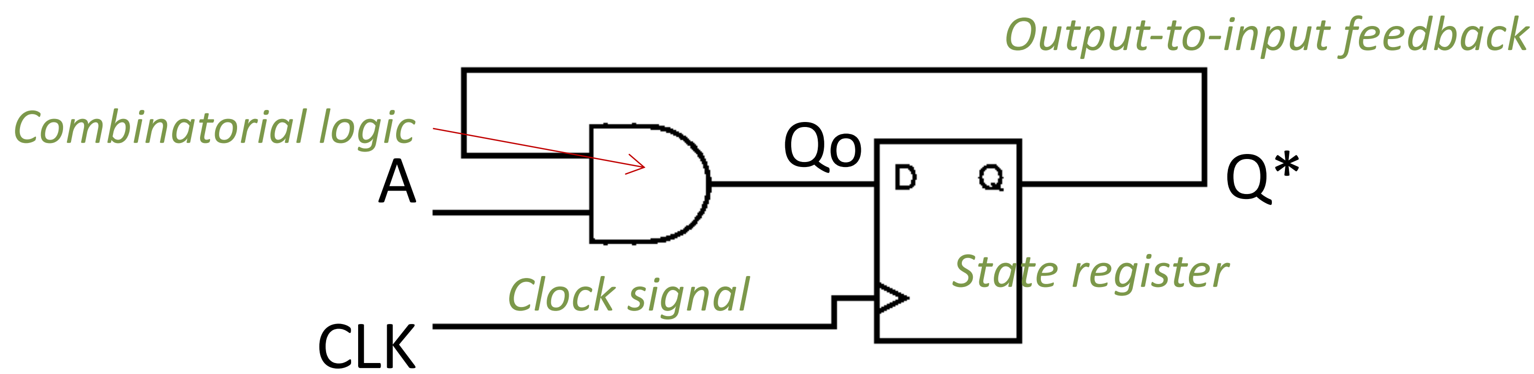
Moore Machine

Output is function of present state only



Represented by Q

Example of a Moore Machine *(with 1 state)*



$$\mathbf{Q^* = Q_O.A}$$

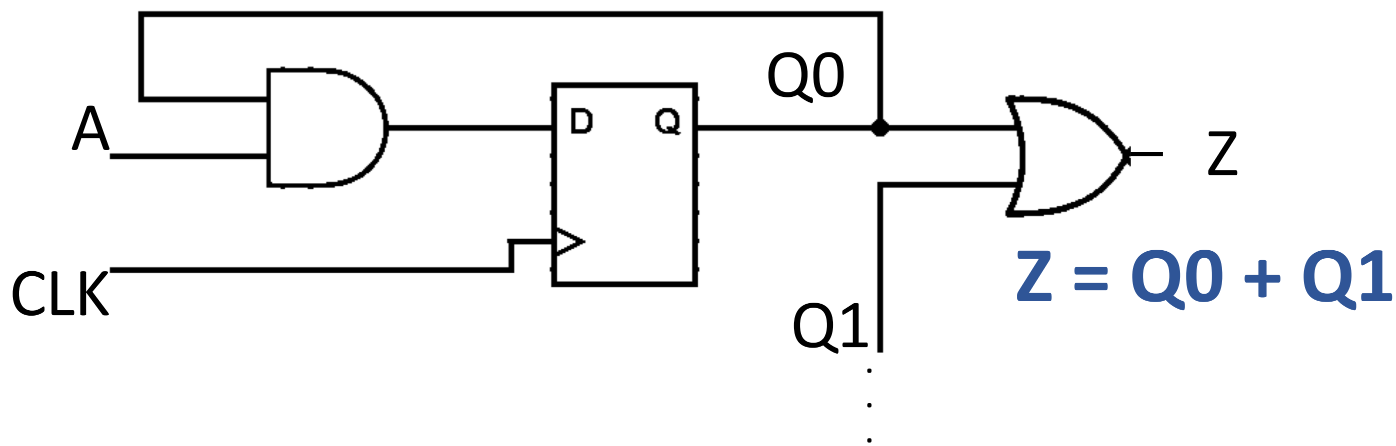
(*read as*: the **<u>next-state of Q</u>** will be Q_O.A)

i.e. ***On the next rising edge of the clock***, the output state – aka the output of D-FF (Q*) – will become the previous value of Q (Q_O) **AND** the value of input A

Example of a Moore Machine
(with 2 states + 1 output)

Represented by
Q0 and Q1 (states)
and Z (output)

Output is function of present state only

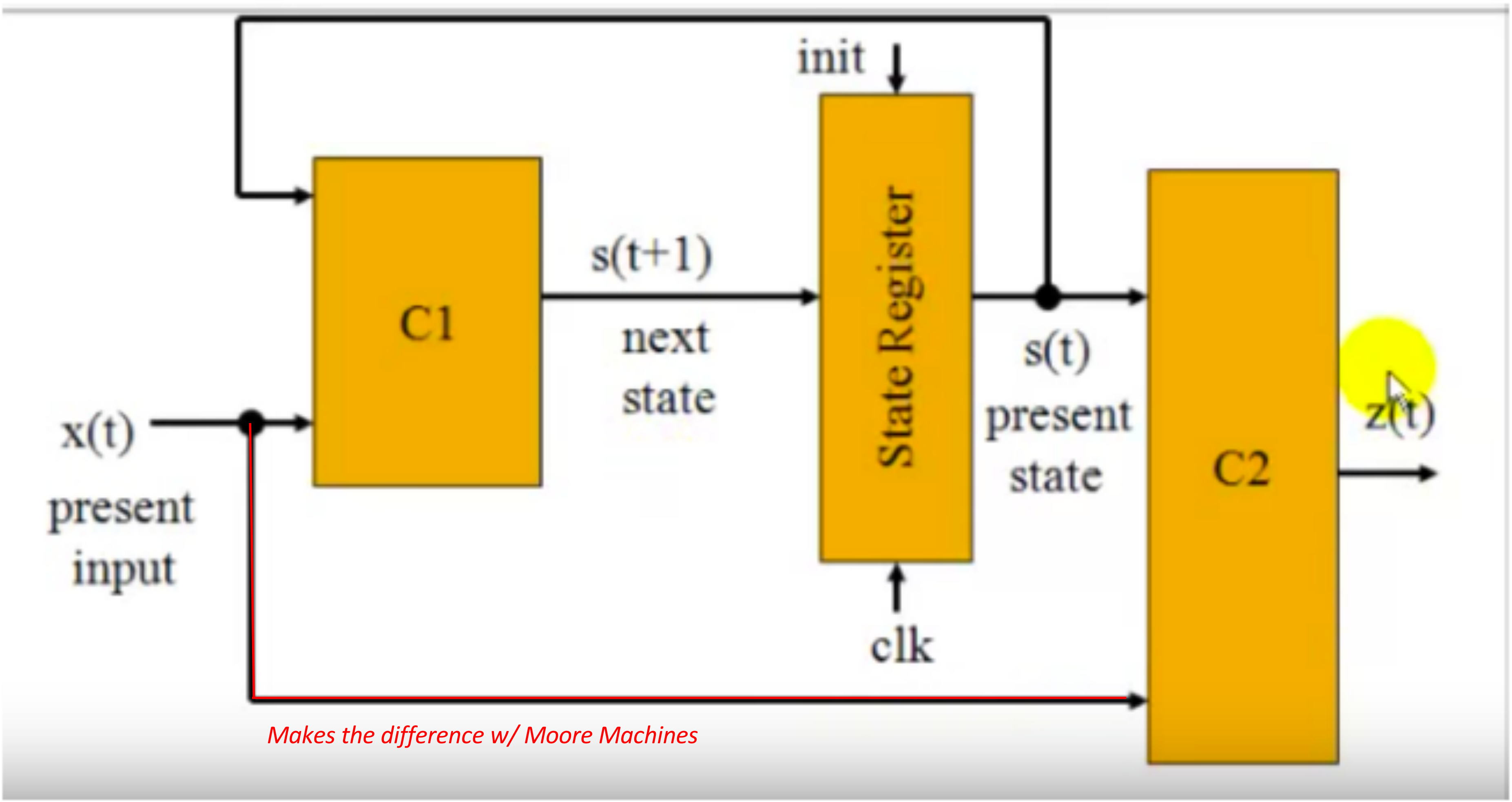


On the next rising edge of the clock, the output state Q0 will be Q0.A and the output state Q1 will be … (not shown here, but you get the idea)
Also, the circuit output Z will become Q0 + Q1

***NOTE*:** **CLK is <u>NOWHERE</u> IN THE EQUATION!!!**

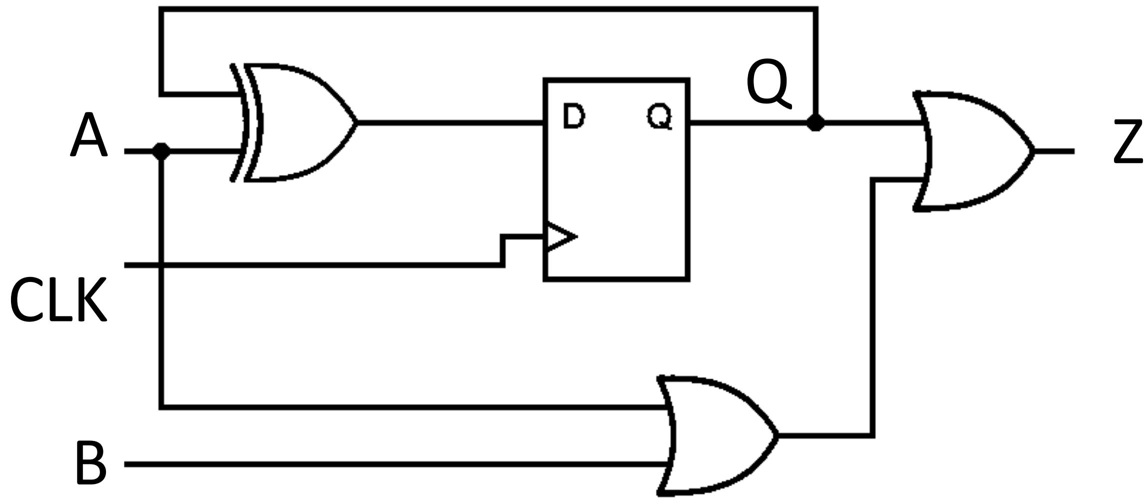
Mealy Machine

Output is function of present state and present input



Example of a Mealy Machine

Output is function of present state and present input



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

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

Example of a Moore FSM

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:

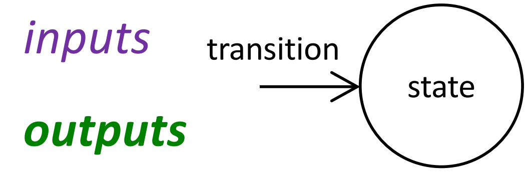
"Done" indicator

Machine Design

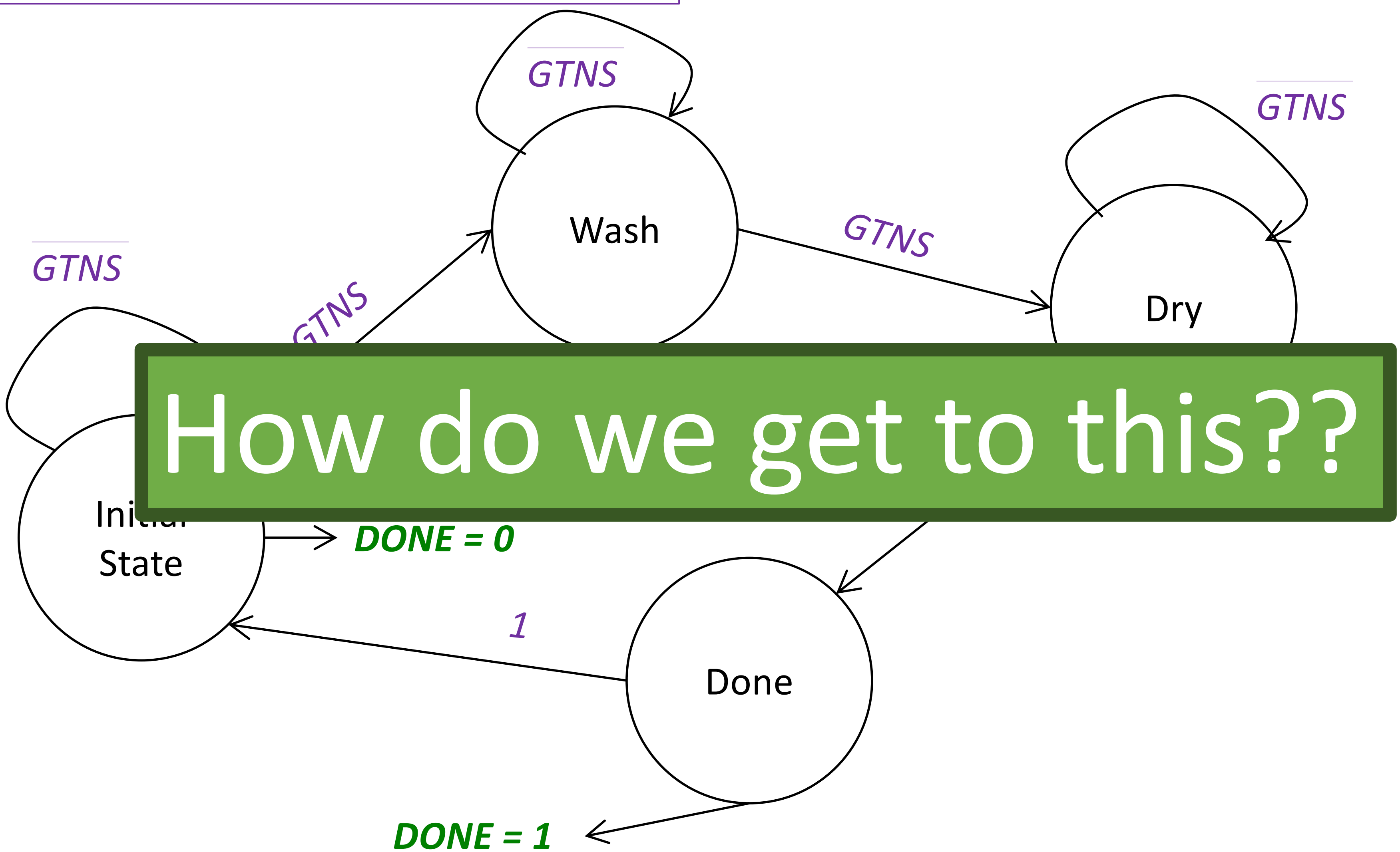
- We want this machine to have **4 distinct states** that we go from one to the next in this sequence:

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**

State Diagram for Washer-Dryer Machine



GTNS = COIN_IN + NO_SOAP + CLTHS_DRY



How do we get to this??

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 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)

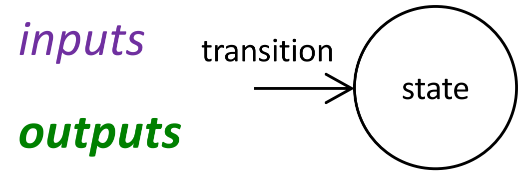
Clothes are now dry (vs clothes are still wet)

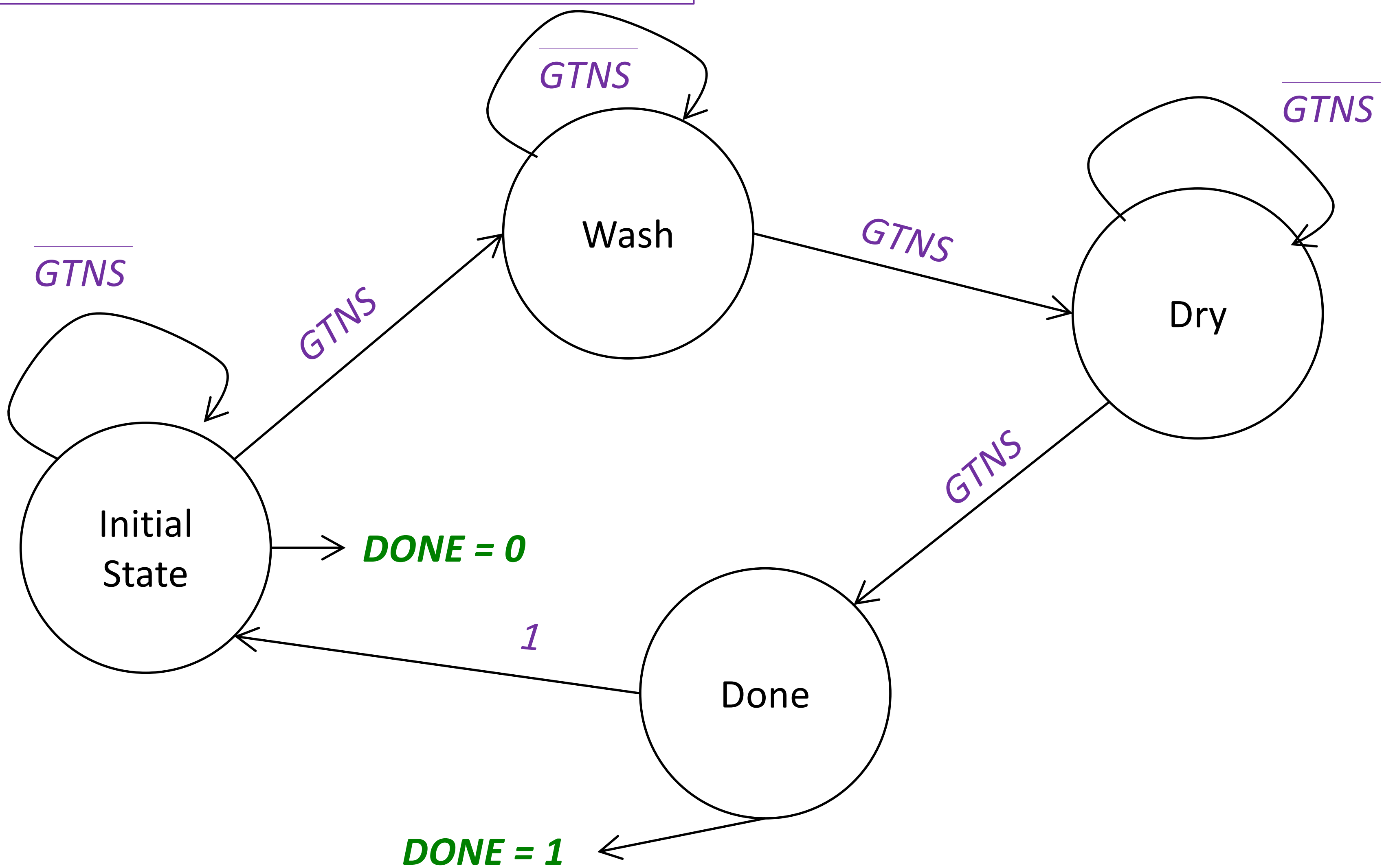
- This is followed by the next state, “**Dry**”
  - This new state sets an output that triggers a timer
  - The input “**Soap is no longer present**” goes to 0, so **GTNS** is 0 also
  - While the input “**Clothes are now dry**” is 0 , **GTNS** remains at 0 too
  - When the input “**Clothes are now dry**” 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

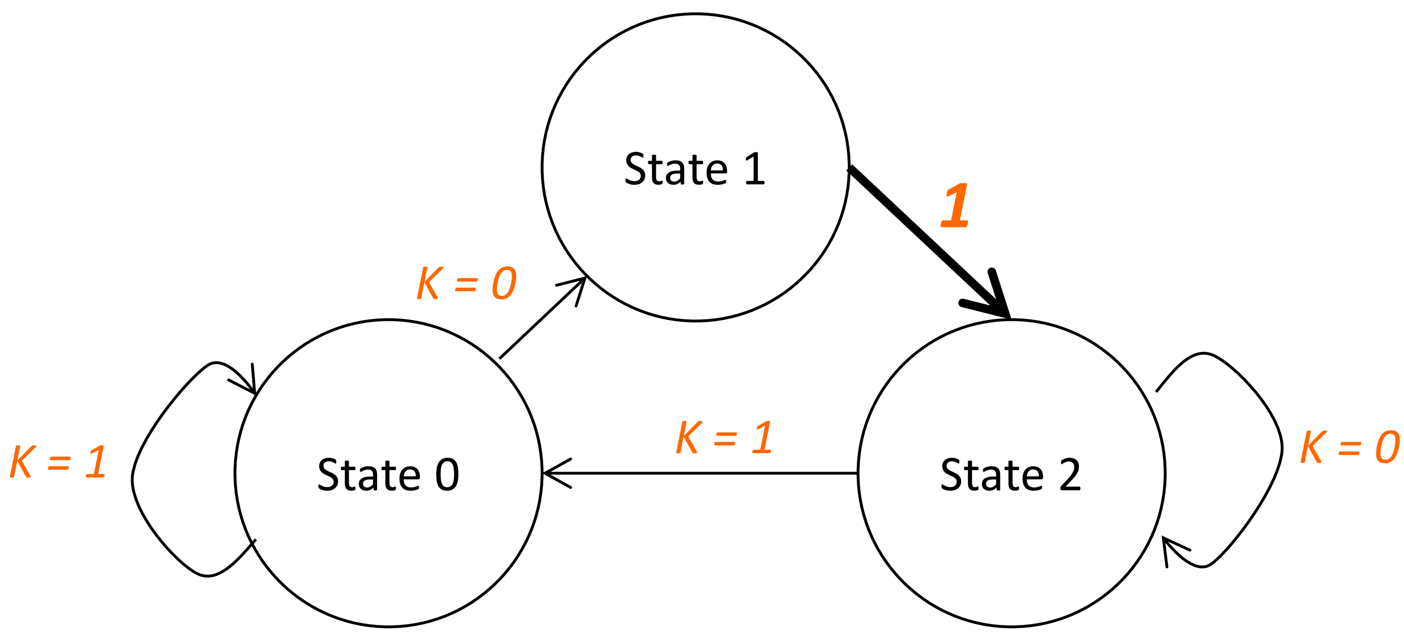




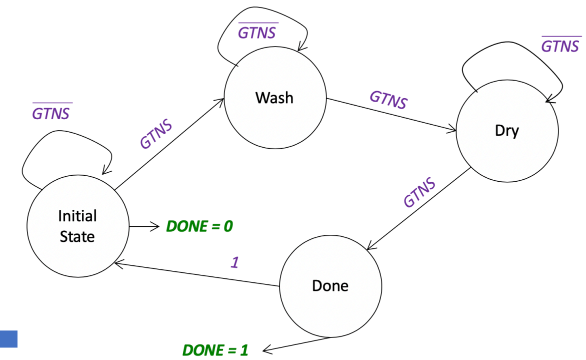
Unconditional Transitions

- Sometimes the transition is unconditional
  - Does not depend on any input –
    you go from **State X** to **State Y** regardless…

- 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"
  - Will be explained later…

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

Example of a Moore FSM 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 (S0)
- We'll first look for a **1**. We'll call that "State 1" (S1)
  - Don't go to S1 if all we find is a **0**!

- We'll then keep looking for another **1**. We'll call that "State 11" (S2)

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

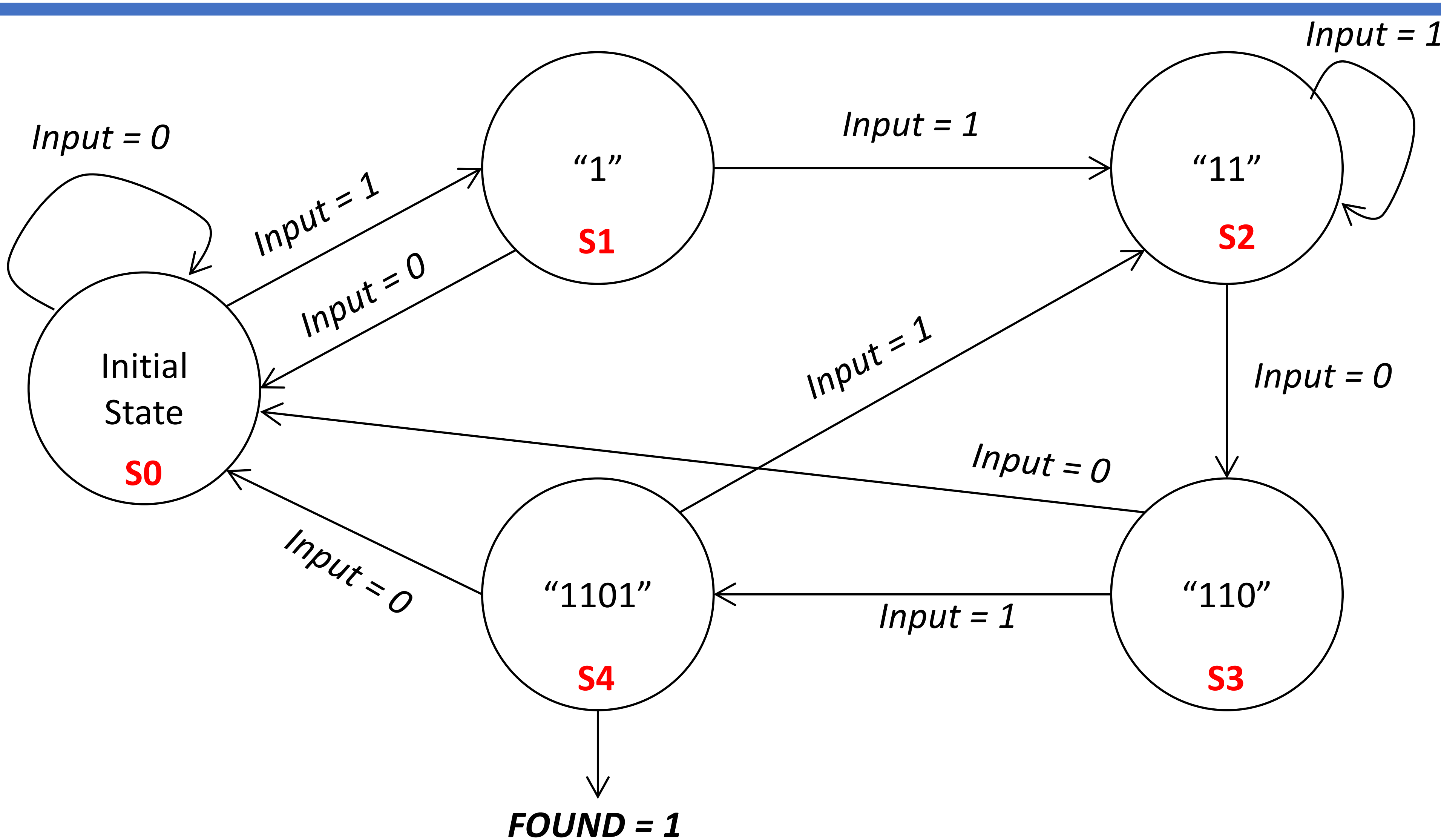
- We will always be detecting “1101” (it doesn’t end)

  So, as SOON as S4 is done, we keep looking for 1s or 0s

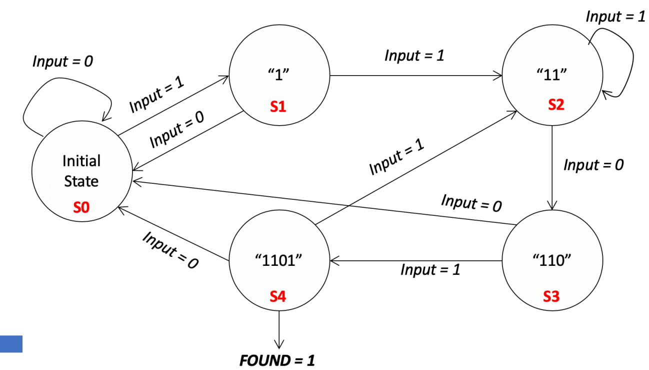
- Example: if the input stream is **1111011101011010000111111011011**

  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
N/A	1	0	1
	1	1	X

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

Note: We are ignoring the N/A states

1. The Truth Table (The State Transition Table)

	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

2. K-Maps for B2* and B1*

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

You need to do this for **all** state outputs

*B2**

B2.B1 B0.I	00	01	11	10
00				
01				
11		1		
10				

- B2* = !B2.B1.B0.I
  - No further simplification

*B1**

B2.B1 B0.I	00	01	11	10
00		1		
01		1		1
11	1			
10				

- B1* = !B2.!B1.B0.I
  + B2.!B1.!B0.I
  + !B2.B1.!B0

2. K-Map for B0*
Output FOUND

- B0* = !B2.!B1.!B0.I
  + !B2.B1.!B0.!I

B0*

B2.B1 **B0.I**	00	01	11	10
00		**1**		
01	**1**			
11				
10				

- FOUND = B2.!B1.!B0
  - 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 B2=1, B1=0, B0=0)

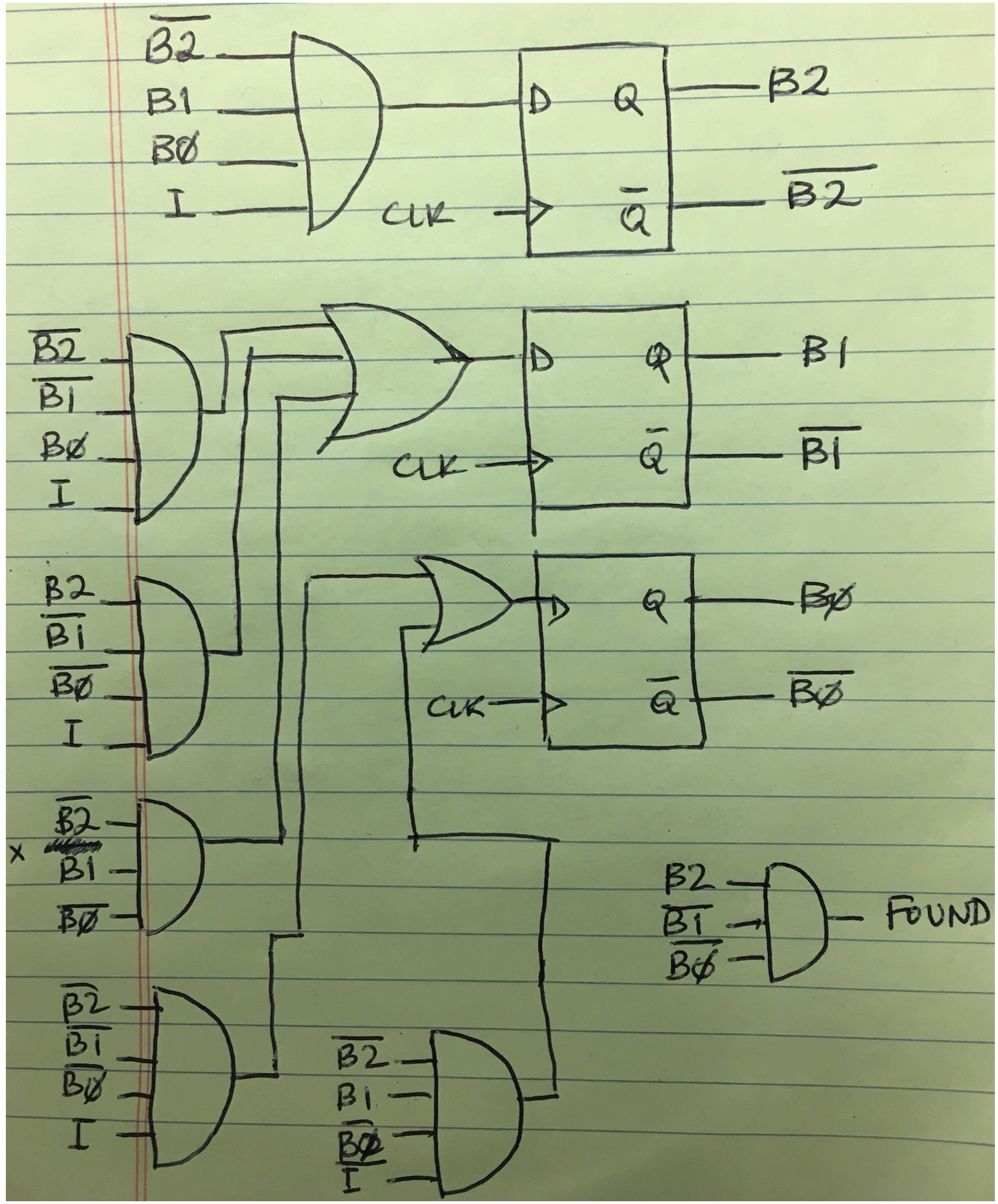
3. Design the Circuit

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!**



YOUR TO-DOs

- Review this FSM stuff!
- Finish Lab #8!

</LECTURE>