Finite State Machines

CS 64: Computer Organization and Design Logic
Lecture #16

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

- *LESS THAN 2 weeks left!!!!!!* ***OMG****OMGOMGOMGOMGOMG!!!!!!!!!!!!!!*
- Final exam is on Wednesday, June 13th from 12:00 – 3:00 PM.
  - Details will be given next week

Lecture Outline

- Finite State Machines
  - Moore vs. Mealy types
  - State Diagrams
  - “One Hot” Method

Any Questions From Last Lecture?

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)

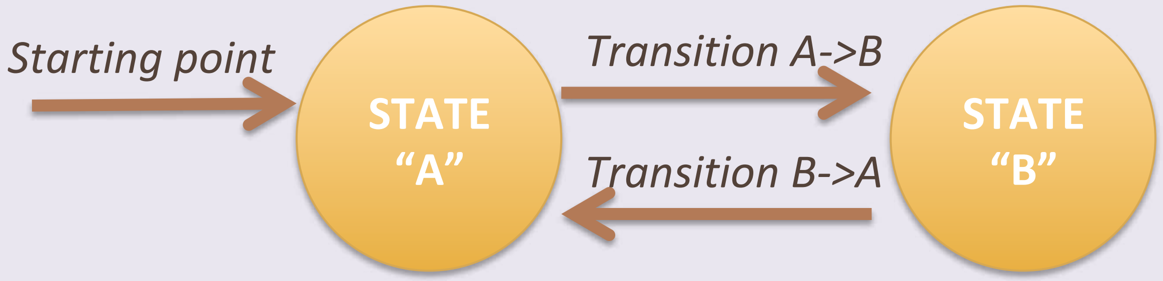
- An **abstract machine** that can be in **exactly one of a finite number of states at any given time**

- 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)

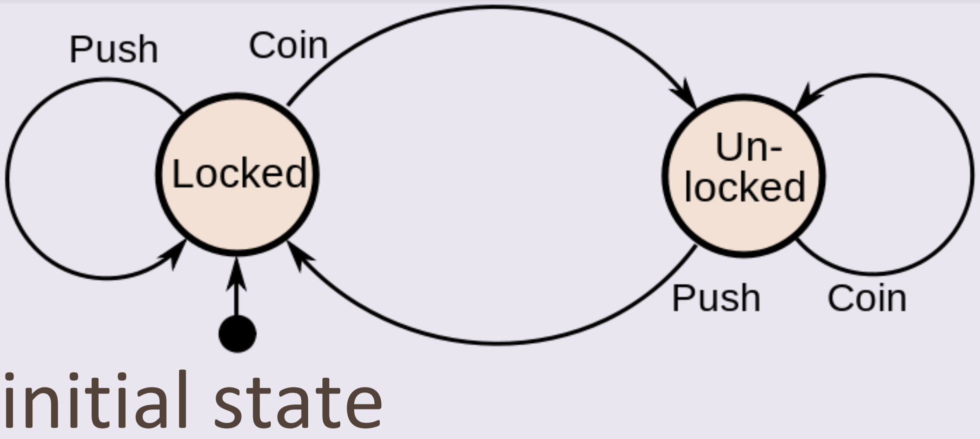
- The FSM can change from one state to another in **response to some *<u>external inputs</u>***

- The change from one state to another is called a ***<u>transition</u>***.



- 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

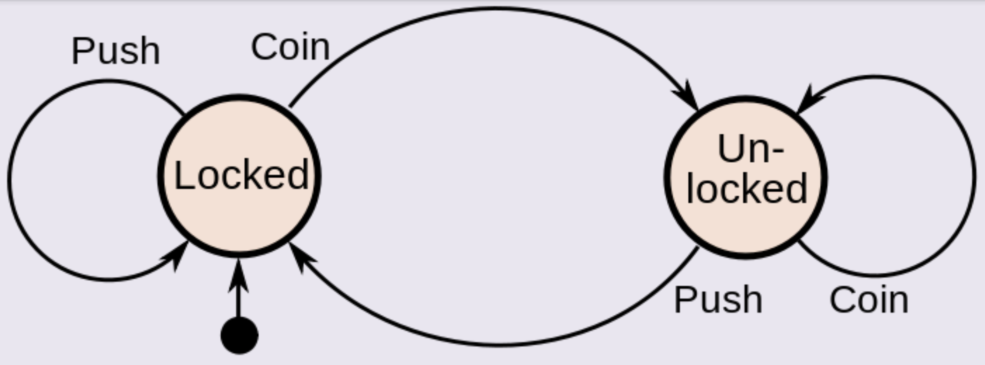


State Transition Table

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

Source: Wikipedia

Example of a Simple FSM: The Turnstile



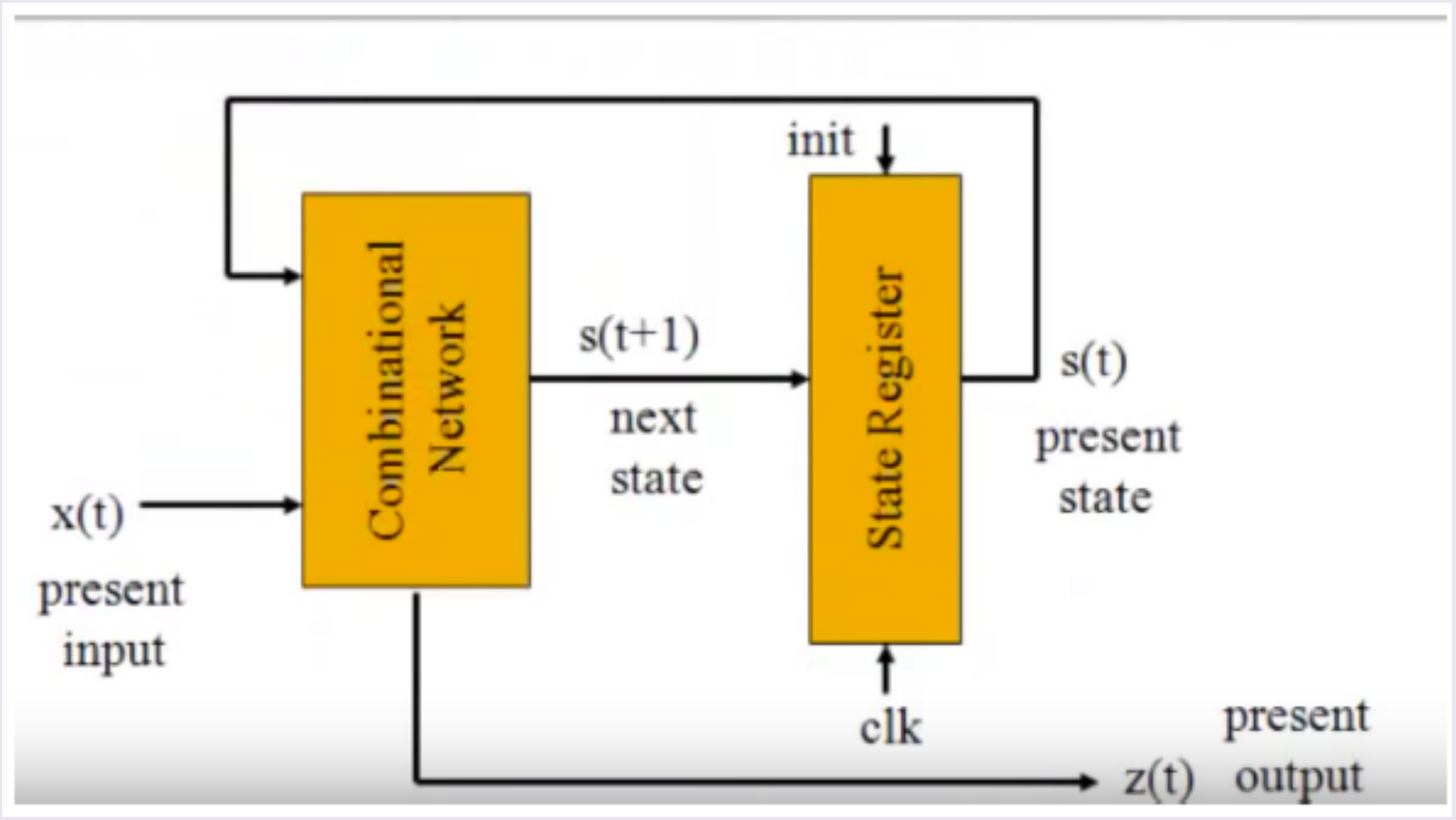
initial state

State Transition Table

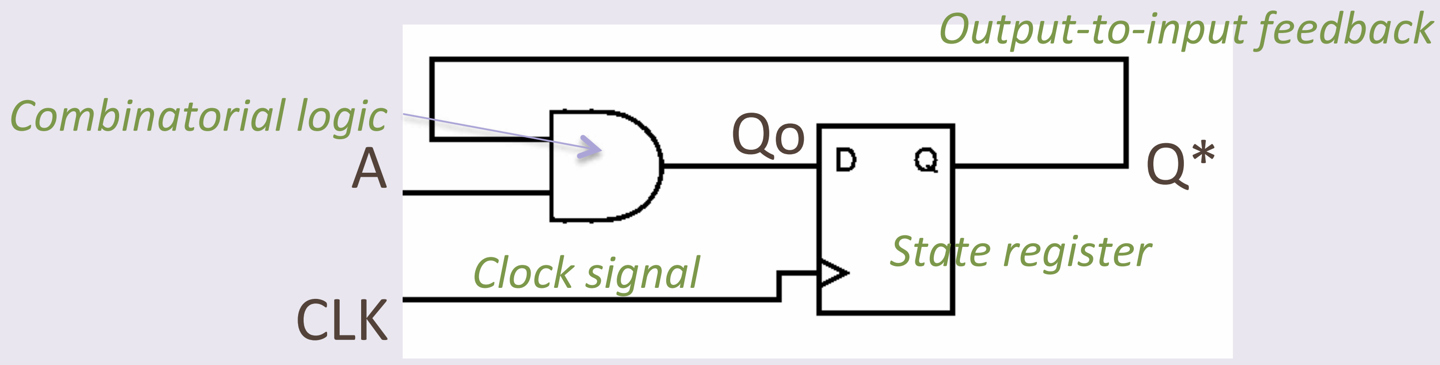
Current State	Input	Next State	Output
Locked	Coin	Unlocked	Unlocks the turnstile so that the customer can push through.
Locked	Push	Locked	Nothing – you're locked! ☺
Unlocked	Coin	Unlocked	Nothing – you just wasted a coin! ☺
Unlocked	Push	Locked	When the customer has pushed through, locks the turnstile.

Source: Wikipedia

General Form of FSMs



Example



$$Q^* = Q_O.A$$

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

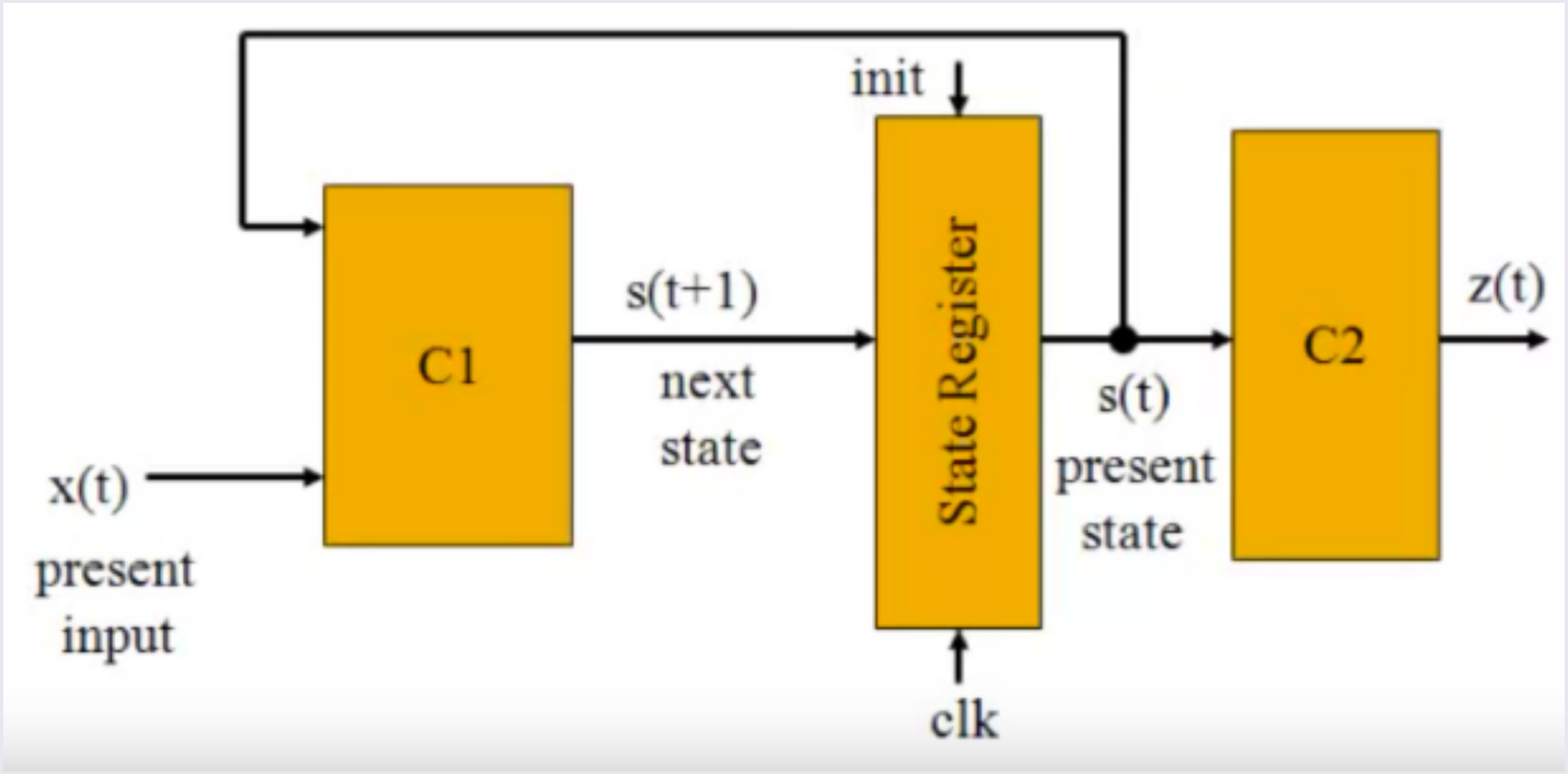
FSM Types

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

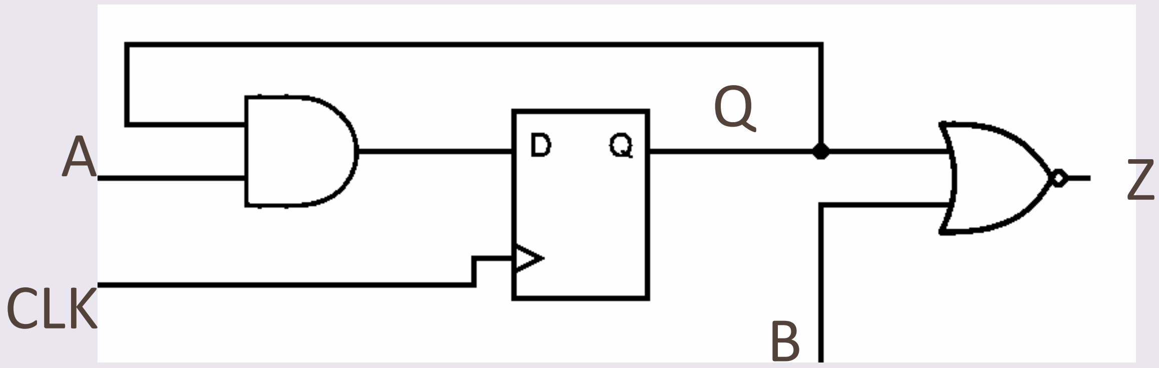
Moore Machine

Output is function of present state only



Example of a Moore Machine
(with 1 state)

Output is function of present state only

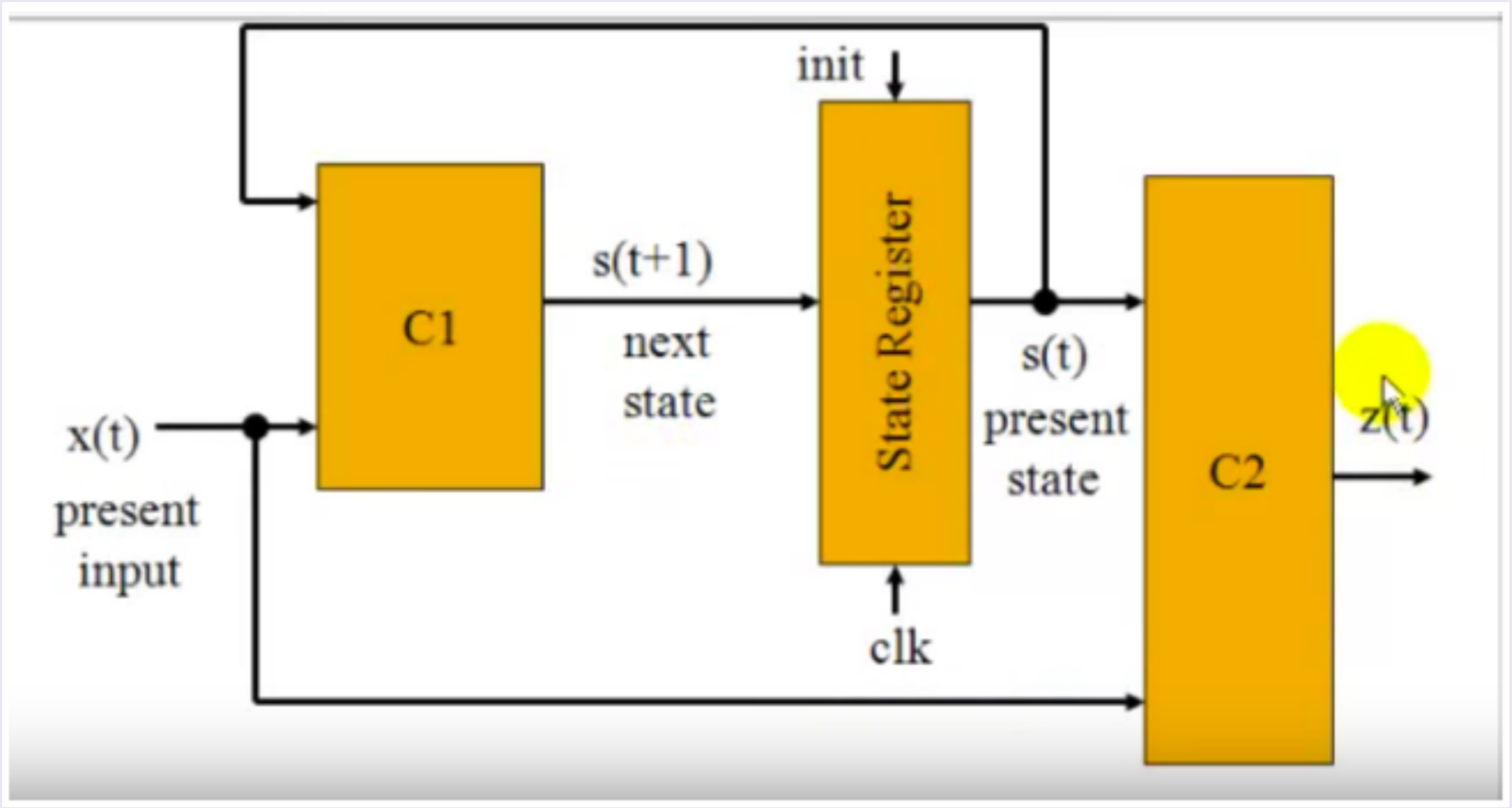


$$\mathbf{Z = \overline{(Q^* + B)} = \overline{(Q_O.A + B)}}$$

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

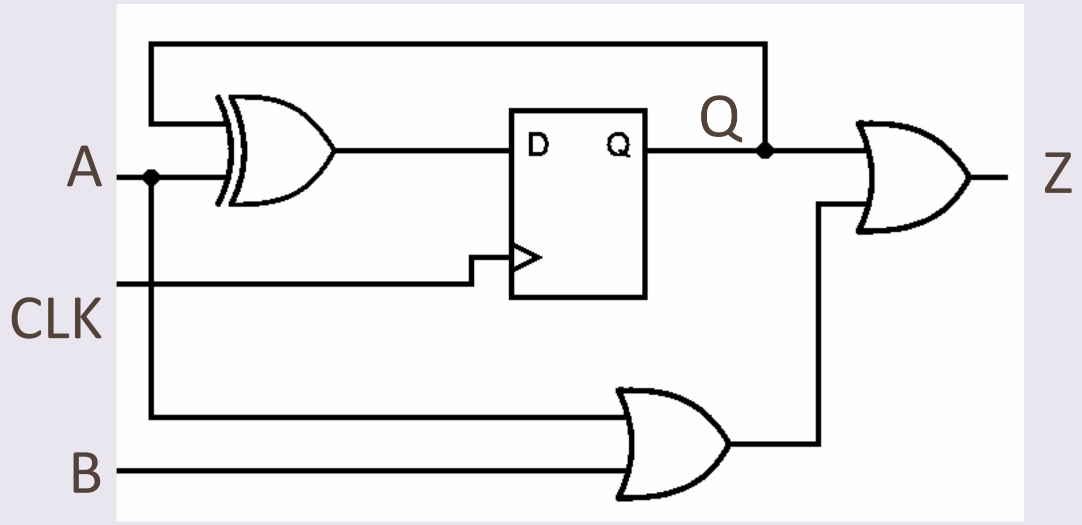
Mealy Machine

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



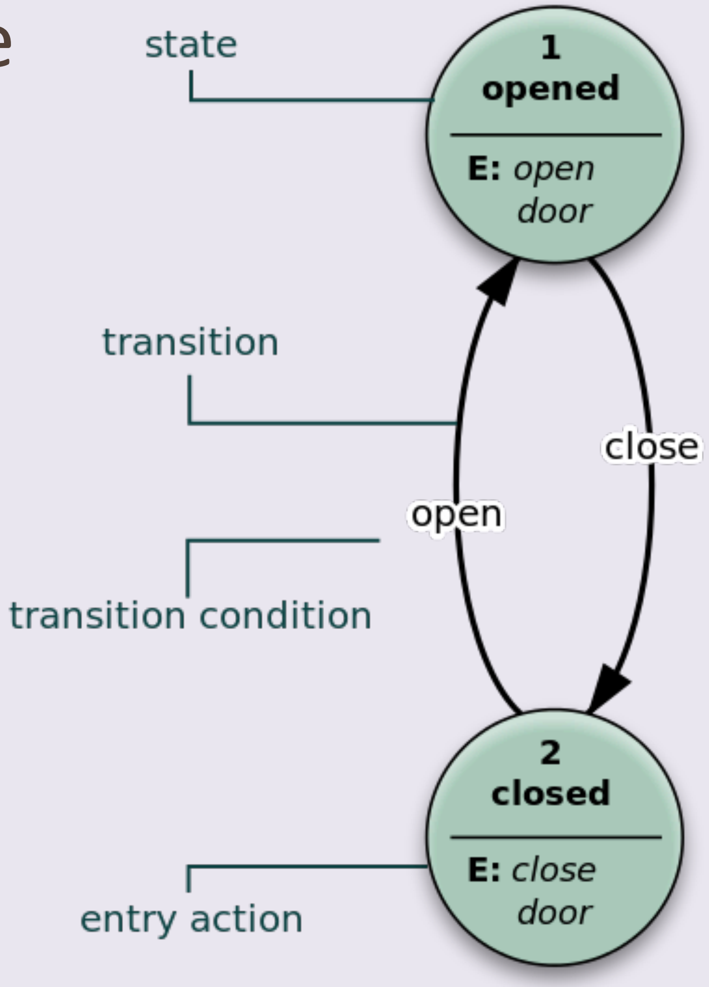
$Z = (Q^* + A + B) = (Q_O \text{ XOR } A) + (A + B)$

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

Diagraming State Machines

- A simple FSM example
- 2 states:
  - Door opened
  - Door closed

- This is called a **state diagram**



Example of a Moore Machine 1

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 put in (vs it isn't)	*Timer is under 30 minutes (vs it isn't)*
Soap is present (vs used up)	*Wetness indicator (vs clothes are dry)*

- This machine also issues outputs while running:

"Timer Begin" indicator	"Fill-Water" indicator
"Drain-Then-Refill" indicator	"Drain Completely" indicator

Example of a Moore Machine 1

WASHER_DRYER

- Before we begin, the machine is in an initial state that is waiting for you to insert a coin. We'll call that the
  "Initial State"

- The machine will start a washer timer as soon as a coin is inserted.
  - By issuing an output, "Timer Begin"

- The timer is controlled by a signal (i.e. input var) called **TIMER_LT_30**, which is always initialized to be 1 (i.e. True).

- We thus transition from "Initial State" to "Wash State"

Example of a Moore Machine 1

WASHER_DRYER

- Wash State: This will output a signal to fill the washer with water (**FILL_WATER**).
- As long as the timer is below 30 mins (**TIMER_LT_30 = 1**), the cycle continues.
- When the timer surpasses 30 mins (i.e. **TIMER_LT_30 = 0**), this state will end
- We thus transition from "Wash State" to "Rinse State"

Example of a Moore Machine 1

WASHER_DRYER

- Rinse State: This will output a signal to drain the water and then refill with new water (**DRAIN_REFILL = 1**).
- As long as the soap sensor is on (**SOAP = 1**), the cycle continues.
- When the soap sensor turns off (i.e. **SOAP= 0**), this state will end
- We thus transition from “Rinse State” to “Dry State”

Example of a Moore Machine 1

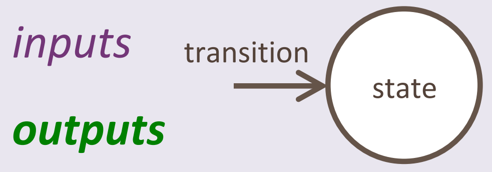
WASHER_DRYER

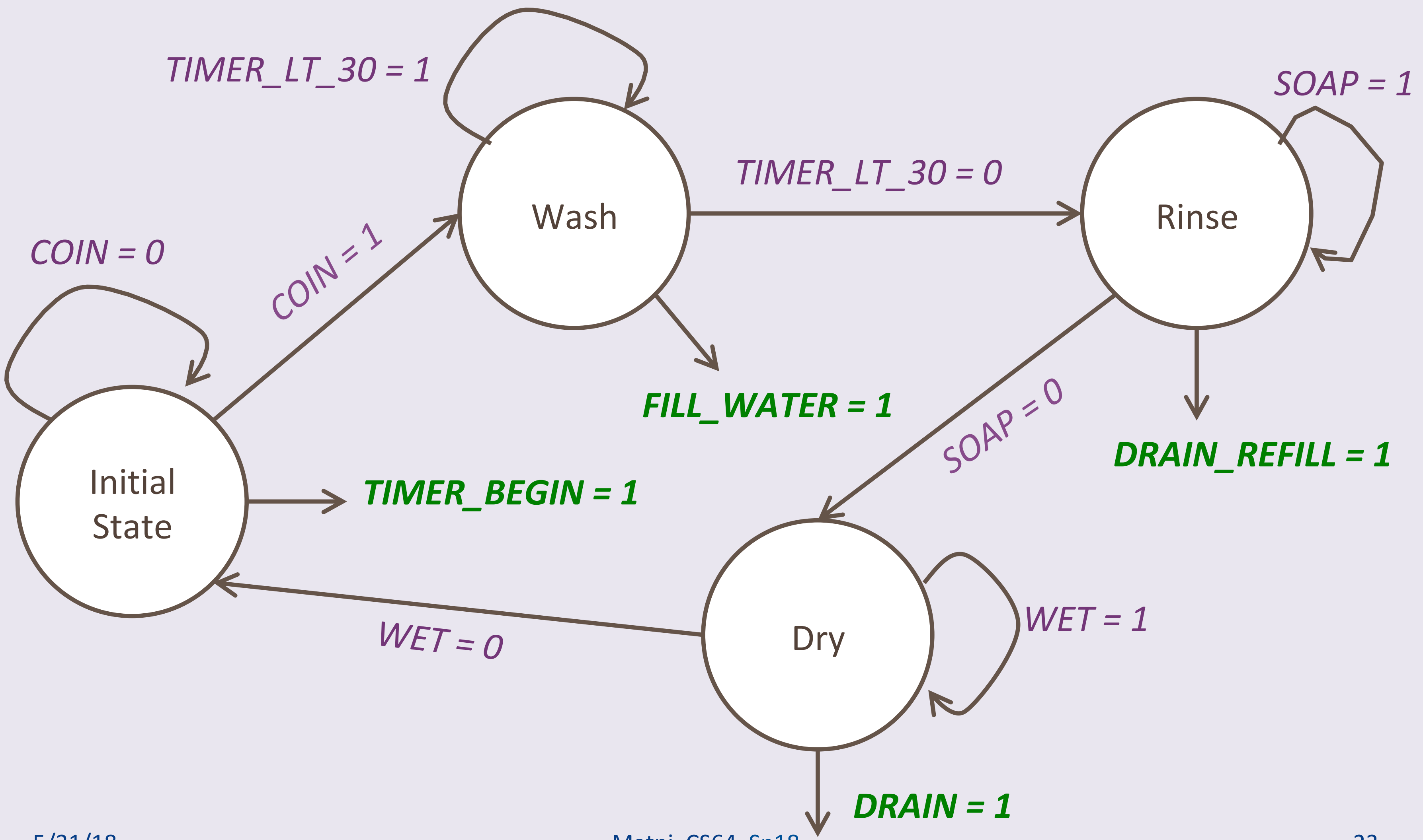
- Dry State: This will output a signal to drain the water for good and begin drying (**DRAIN = 1**).

- As long as the wet clothes sensor is on (**WET = 1**), the cycle continues.

- When the wet clothes sensor is off (**WET = 0**), we will stop!
- This means going back to the "**Initial State**"

State Diagram for Washer-Dryer Machine

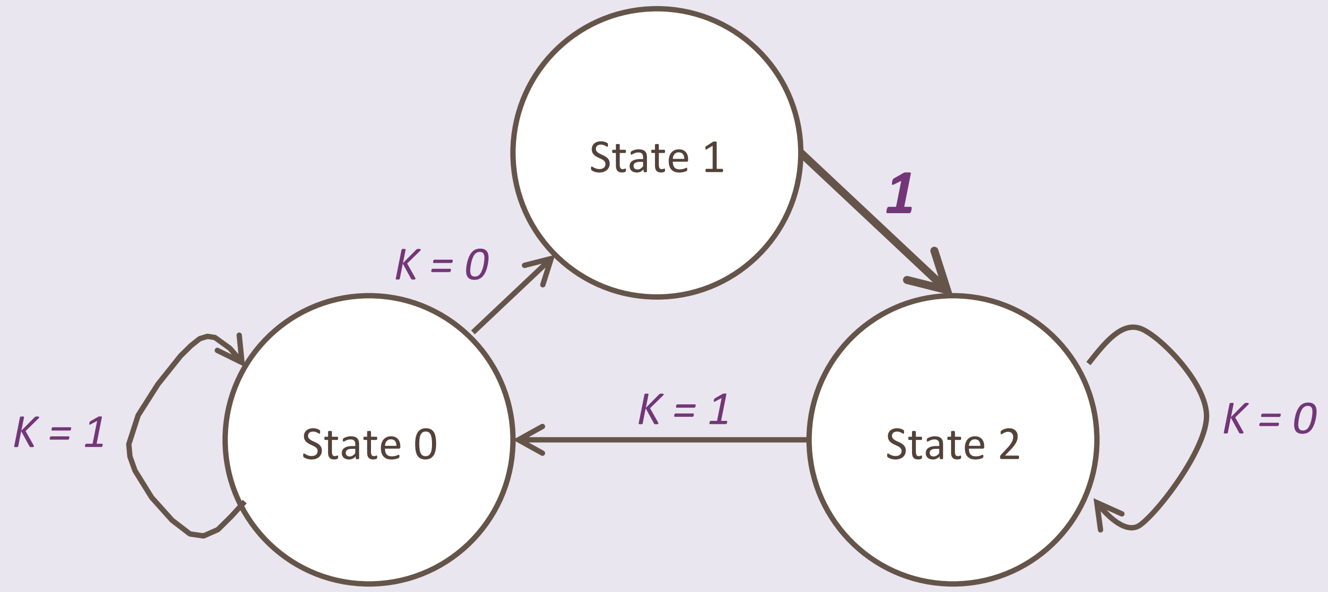




Unconditional Transitions

- 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

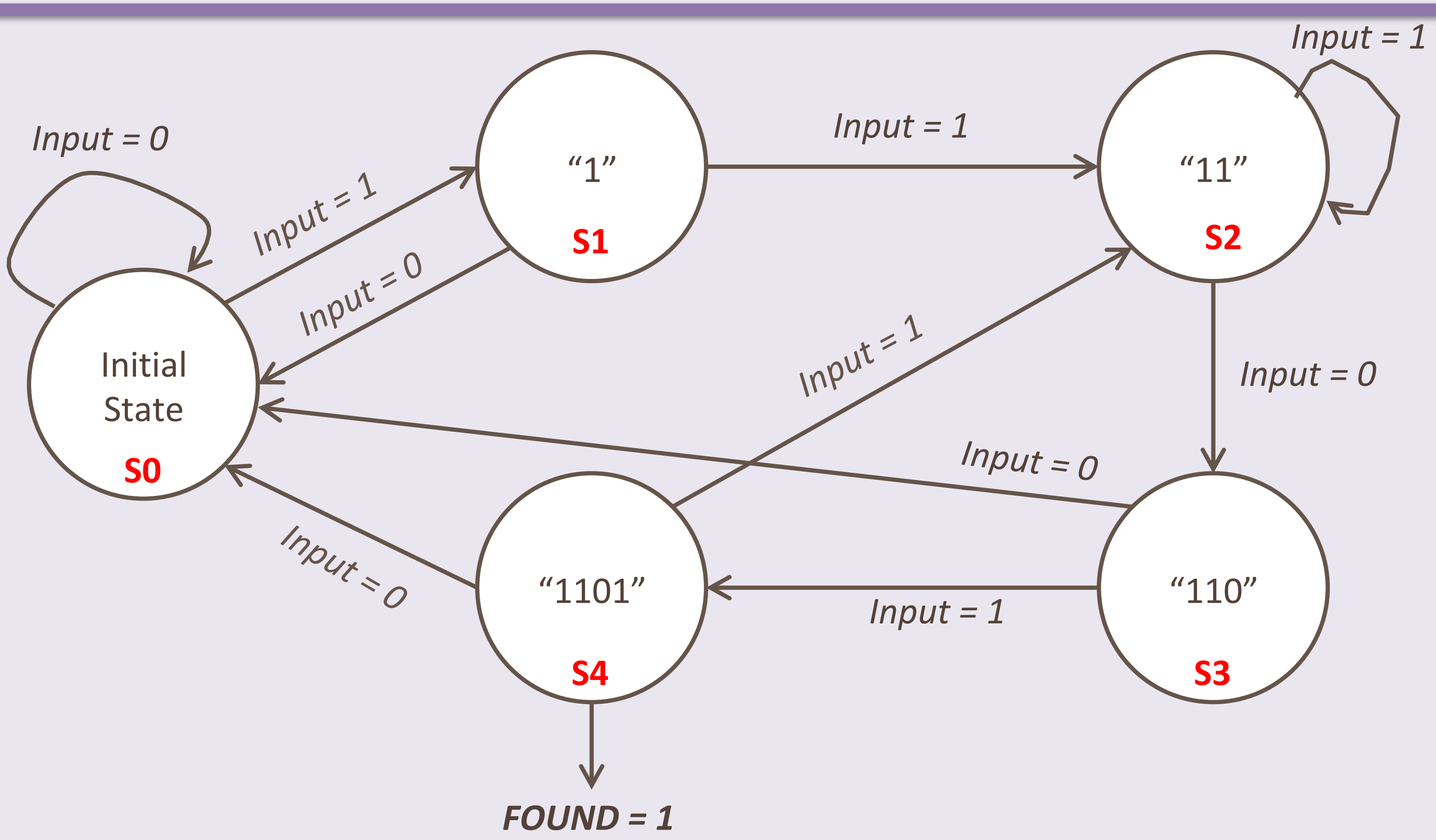
- Let’s build a sequential logic FSM that always detects a specific serial sequence of bits: ***1101***

- We’ll start at an “Initial” state (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)
- Then… a **0**. We’ll call that “State 110” (S3)
- Then another **1**.
  We’ll call that “State 1101”(S4) – this will output a **FOUND** signal

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

```
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	S2	S1	S0
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
2. Make K-Maps and simplify
3. Design the circuit

1. The Truth Table

State	S2	S1	S0	I	S2*	S1*	S0*	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	1	1	0	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

- Example: for S2*
  - You need to do this for **all** 4 outputs

S2.S1 / S0.I	00	01	11	10
00		1		
01				
11		1		
10				

```
S2* =  !S2.S1.!S0.!I
     + !S2.S1.S0.I
     = !S2.S1.(!S0.!I + S0.I)
     = !S2.S1.!(S0 ⊕ I)
```

3. Design the Circuit

- See blackboard…

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 FFs are combinatorial logic that can simplified into a “sum-of-products” type of Boolean expression

- Current CAD software can do this automatically
- Implementation is usually done on a simulator (software), or prototype hardware Integrated Circuit (FPGA)

Encoding our States

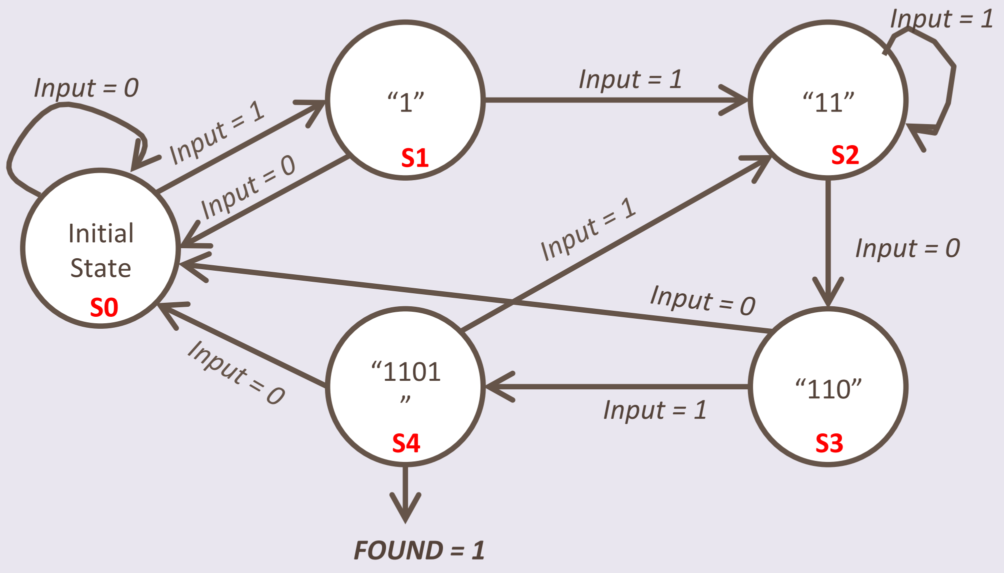
Per the last example: We had 5 separate states:

NAME		Binary 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!!):



S1* = $S0.I$

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

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

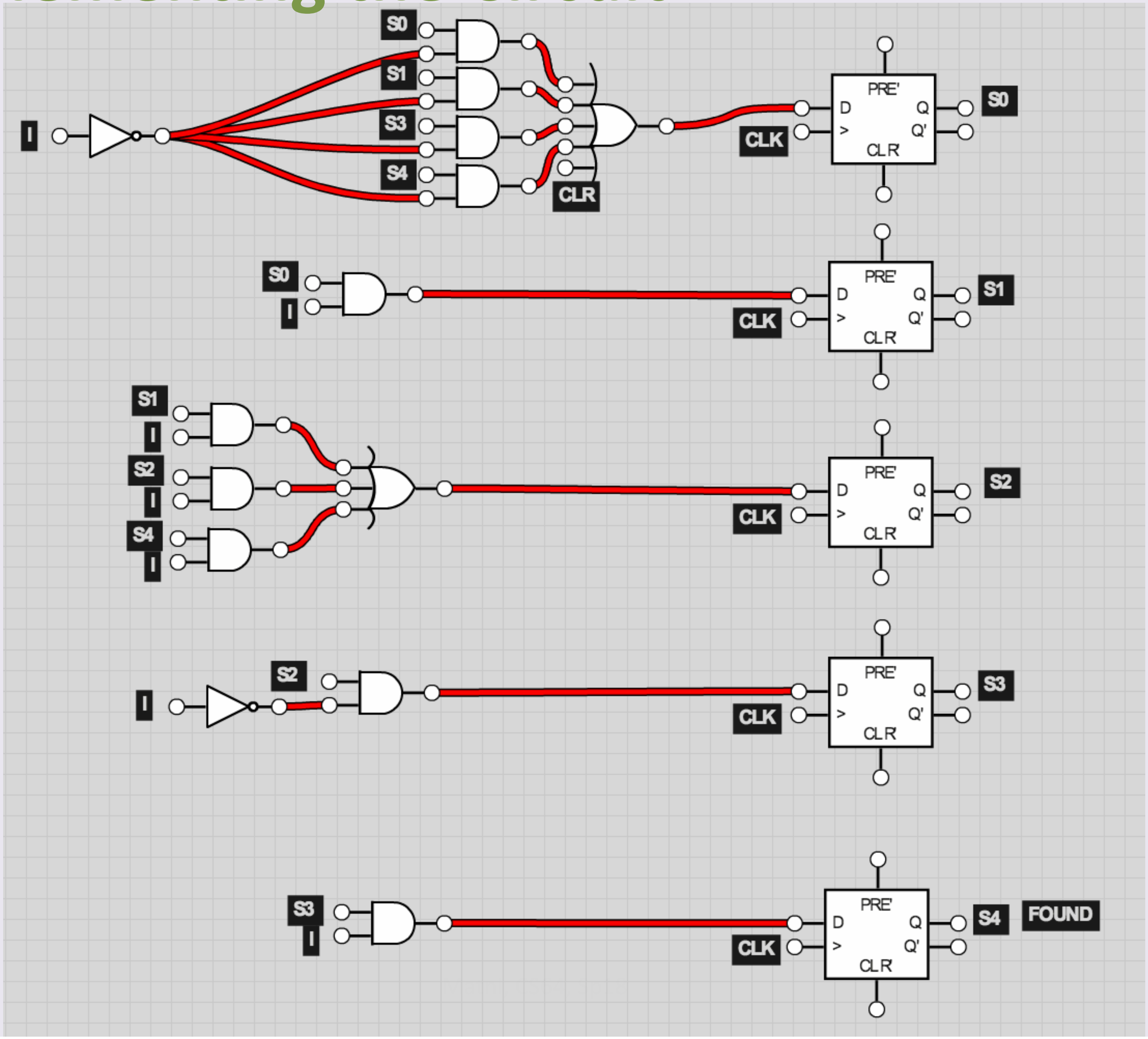
S4* = $S3.I$

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

Also, when S4 happens, FOUND = 1, i.e. **FOUND = S4**

We have now described ALL the outputs of the machine as combinations of certain inputs

Implementing the Circuit



Your To Dos

- Lab #8 is due Monday!
- Lab #9 is on FSM and will be issued tomorrow
  - Due on the last day of classes

</LECTURE>