


# COSC175 (Systems I): Computer Organization & Design



Professor Lillian Pentecost  
Fall 2024



# Warm-Up September 19

- Where we were
  - More complex, useful circuitry!
  - Our suite of tools for simplifications now includes K-Maps
- Where we are going
  - Not just building circuits, but building **faster** circuits
  - Not just reacting to inputs to produce an output, but incorporating **state**
- Logistics, Reminders
  - TA help 7-9PM on Sundays, Tuesdays, Thursdays in C107
  - LP Office Hours at 2:30-4PM Today
  - Weekly Exercises Due Friday 5PM
  - Lab 2 due Monday 10PM
- Textbook Tags: 2.9, 3.1, 3.2 (3.2.4), 3.4.1, 3.4.2



Application Software	
Operating Systems	
Architecture	
Micro-architecture	
Logic	
Digital Circuits	
Analog Circuits	
Devices	
Physics	

CS FALL RECEPTION

Connect with students and faculty and enjoy refreshments!

SEPT. 20TH 3-4PM  
SCCE-Basement/A013

Featuring Research Lightning Talks By:

Jesse Looney  
Hardware

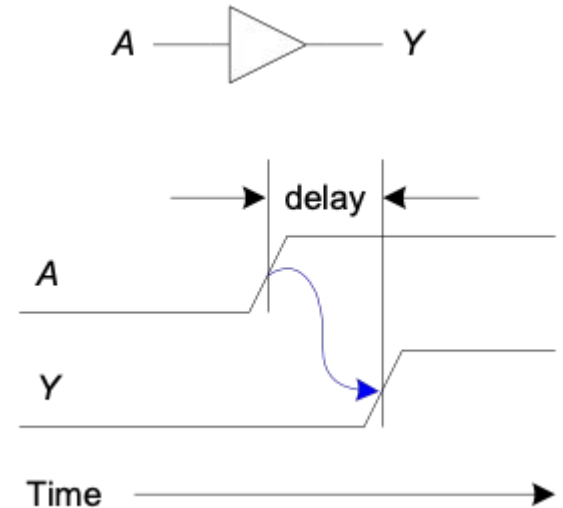
Soyon Choi  
Machine Learning

Madi Gudim  
Hardware

# Timing of circuits

**Delay** is the time between input change and output changing.

- Due to the speed of light
- Due to the physical properties (capacitance, resistance)
- Due to environmental factors (hotter circuits are slower, less reliable)

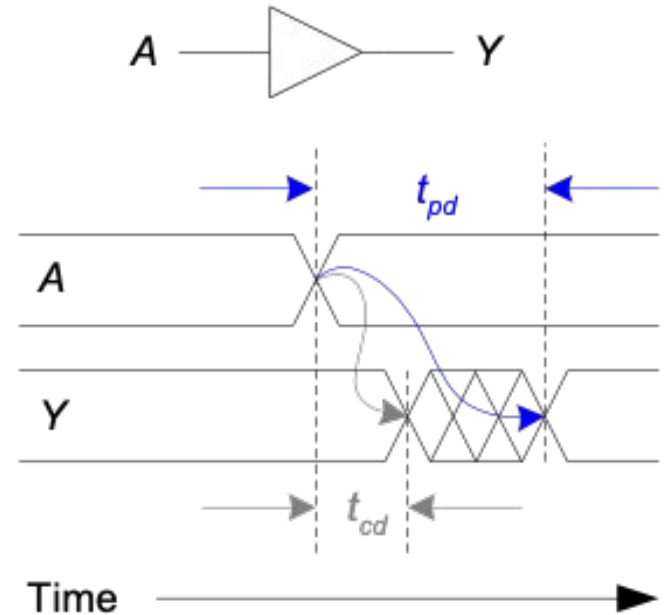


# Timing of circuits

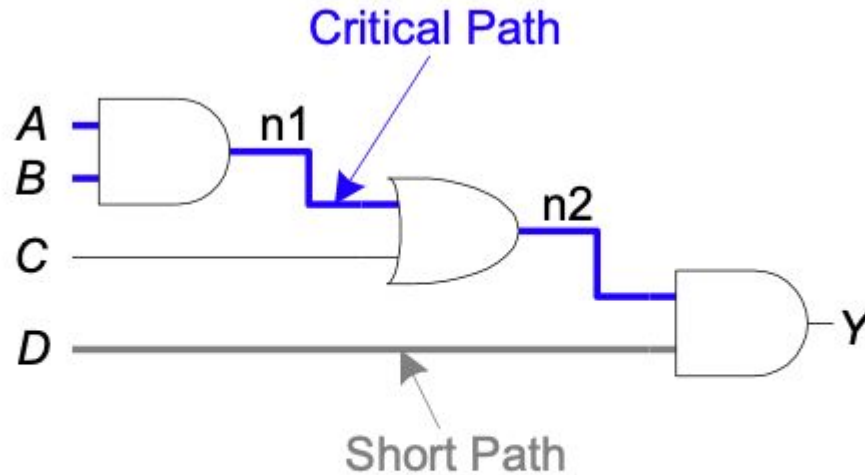
**Delay** is the time between input change and output changing

Delay can vary, and we define:

- **Propagation Delay:** the max delay from input to output
- **Contamination Delay:** the min delay from input to output



# Identifying a Critical Path vs. a Short Path

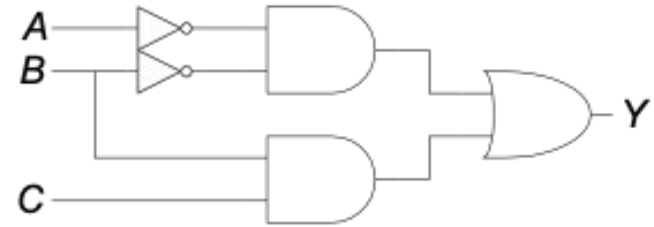


**Critical (Long) Path:**  $t_{pd} = 2t_{pd\_AND} + t_{pd\_OR}$  (max delay)

**Short Path:**  $t_{cd} = t_{cd\_AND}$  (min delay)

# Problems with delay

- **Glitches** can occur if a single input change causes an output to change multiple times
- What is the **critical path** of this circuit?
- What is the **short path** of this circuit?

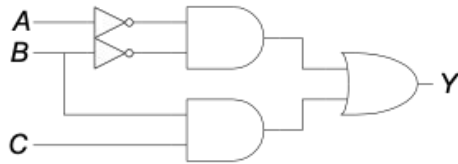


		AB			
		00	01	11	10
C	0	1	0	0	0
	1	1	1	1	0

$$Y = \bar{A}\bar{B} + BC$$

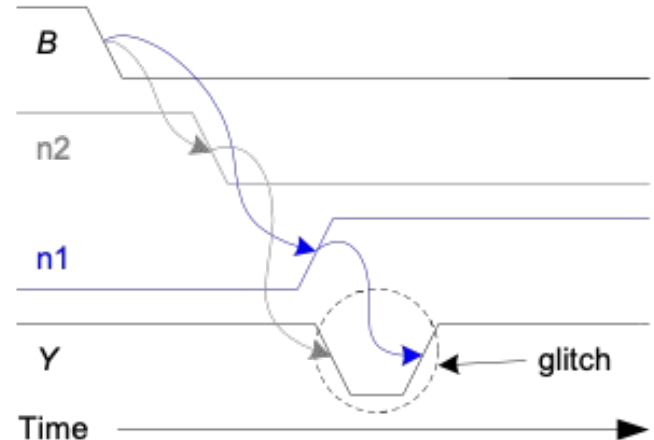
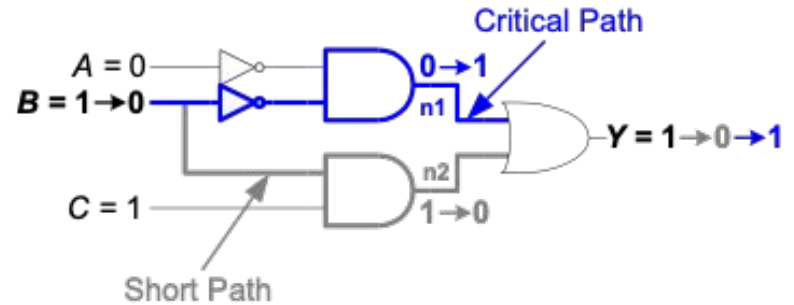
# Problems with delay

- Let's look at a **glitch** using a **timing diagram** – how can we fix it?



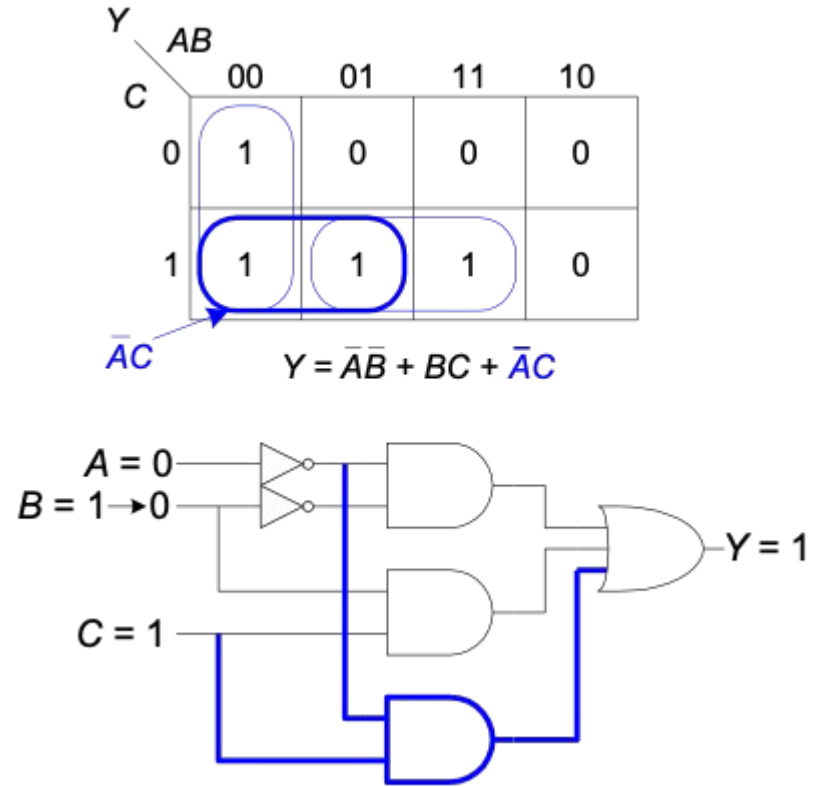
		AB			
C	AB	00	01	11	10
	0	1	0	0	0
1	0	1	1	1	0

$$Y = \bar{A}\bar{B} + BC$$



# Problems with delay

- Let's look at a **glitch** using a **timing diagram** – how can we fix it?
- One common strategy is **redundancy** to ensure consistent behavior
- But, we can't (and don't) get rid of all glitches – we should know they exist, and we should follow conventions so that they don't cause problems

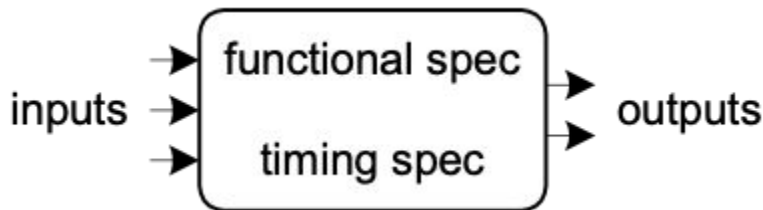




# Combinational vs. Sequential Logic

---

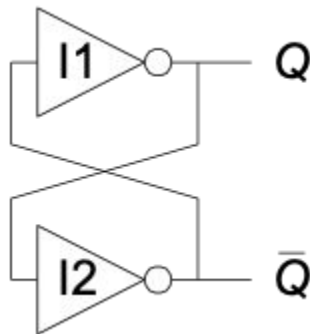
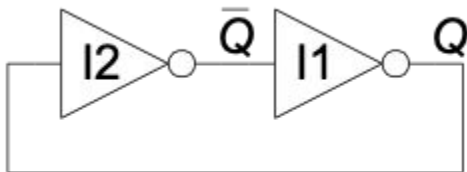
- Our example circuits have so far been **combinational**
  - Outputs determined by current values of inputs
- Moving forward, we will also consider **sequential** logic
  - These have “memory”
  - Outputs are determined by previous AND current values of inputs
  - All circuits that aren't combinational



# Combinational vs. Sequential Logic

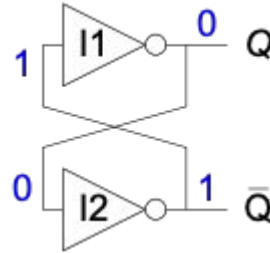
---

- Our example circuits have so far been **combinational**
  - Outputs determined by current values of inputs
- Moving forward, we will also consider **sequential** logic
  - These have “memory”
  - Outputs are determined by previous AND current values of inputs
  - All circuits that aren't combinational

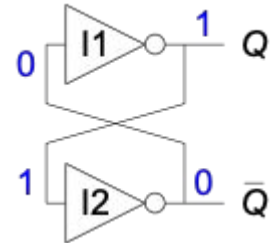


# Bistable Circuits: holding 1 bit of state!

- If  $Q = 0$ :
  - then  $\bar{Q} = 1$ ,  $Q = 0$  (consistent)



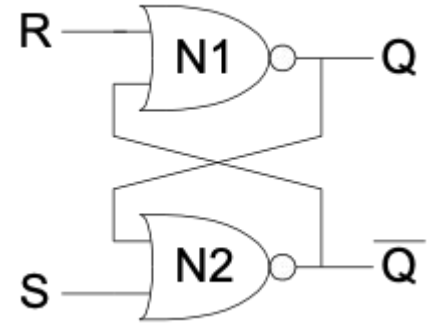
- If  $Q = 1$ :
  - then  $\bar{Q} = 0$ ,  $Q = 1$  (consistent)



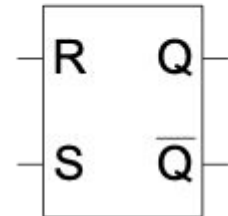
- This circuit **stores 1 bit of state** in the state variable,  $Q$  (or  $\bar{Q}$ )
- But there are no inputs to control the state

# We can store data!! Now let's make sure we can change it, too, with an ***SR Latch***

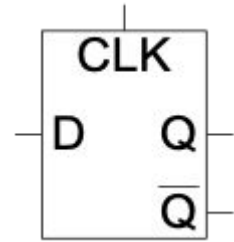
- **SR** stands for Set/Reset Latch
  - Stores one bit of state (Q)
- Control what value is being stored with S, R inputs
- **Set:** Make the output 1
  - $S = 1, R = 0, Q = 1$
- **Reset:** Make the output 0
  - $S = 0, R = 1, Q = 0$
- **Memory:** Retain value
  - $S = 0, R = 0, Q = Q_{\text{prev}}$
- **Avoid invalid state (when  $S = R = 1$ )**



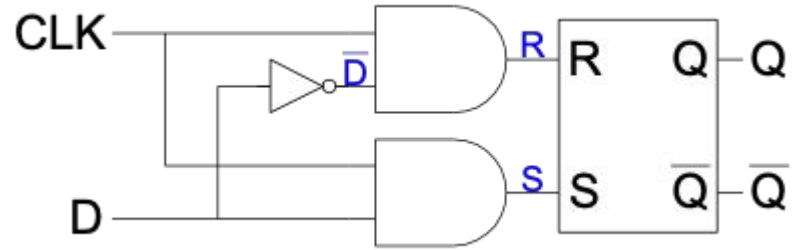
SR Latch  
Symbol



# A better, more foolproof bit: D latch



- The **clock signal** (CLK) determines when the output changes
- **D** controls what the output changes to
- When CLK is 1, Q becomes D
- When CLK is 0, Q holds previous value
- We've removed the possibility of S=1, R=1 → stable!
- ***Study this on your own!***  
***Appreciate a stable bit!***

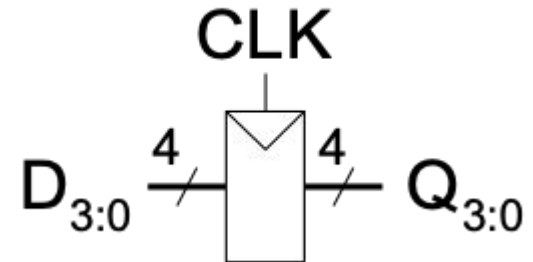


<i>CLK</i>	<i>D</i>	$\overline{D}$	<i>S</i>	<i>R</i>	<i>Q</i>	$\overline{Q}$
0	X	$\overline{X}$	0	0	$Q_{prev}$	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

# Sequential Logic Principles & Terms

---

- Give sequence to events
  - Have memory (short-term)
  - Use feedback from output to input to store information
  - The **state** is all the information necessary to explain a circuit's future behavior
  - We will define ***latches*** and ***flip-flops*** as state elements that store one bit of state
  - A **register** as an N-bit collection of flip-flops storing N bits, updated on CLK
- 
- We will work to build ***synchronous sequential circuits***:
    - sequential circuits using flip-flops sharing a common ***clock***

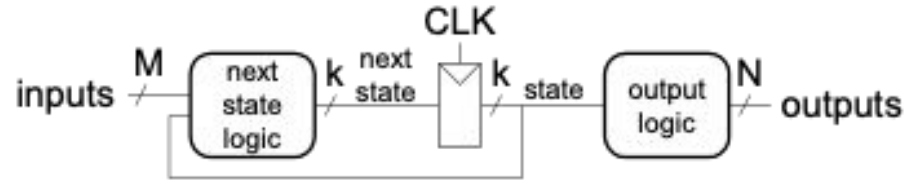


# Finite State Machines (FSMs)

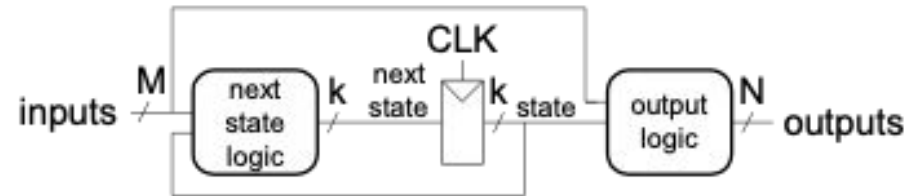
- **State register** to store *current state*
  - The register is updated to a *next state* at the clock edge
- **Combinational Logic** to compute the next state and outputs based on the current state and inputs

*More broadly though:* A FSM is an expressive and meaningful way to describe the behavior of a system, and framing questions and system behavior

Moore FSM



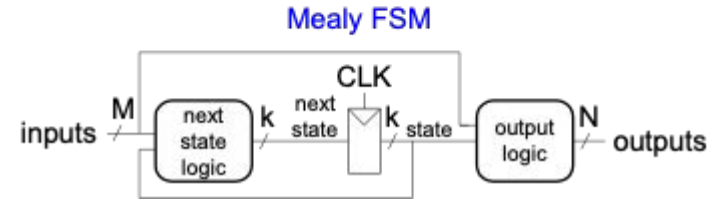
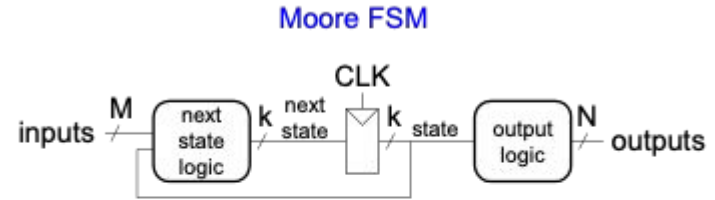
Mealy FSM



# FSM Design Procedure!

---

1. Identify inputs and outputs
2. Sketch **state transition diagram**
3. Write state transition table and output table
  - a. Moore FSM: write separate tables
  - b. Mealy FSM: write combined state transition and output table
4. Select state encodings
5. Rewrite state transition table and output table with state encodings
6. Write Boolean equations for next state and output logic
7. Sketch the circuit schematic



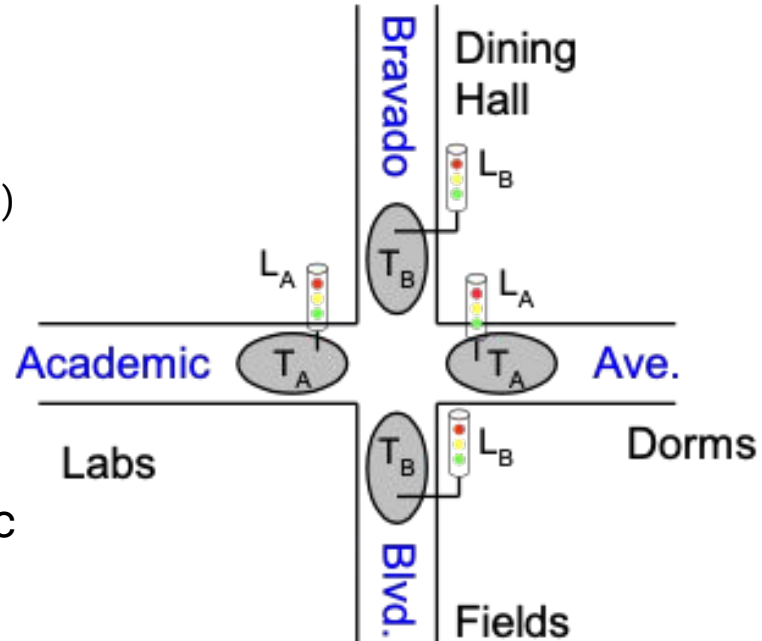


# So squishy! Let's try an example.

- Traffic sensors:
  - TA, TB (1 when there are cars present)
- Lights:
  - LA, LB can be green (go), yellow (slow), or red (stop)

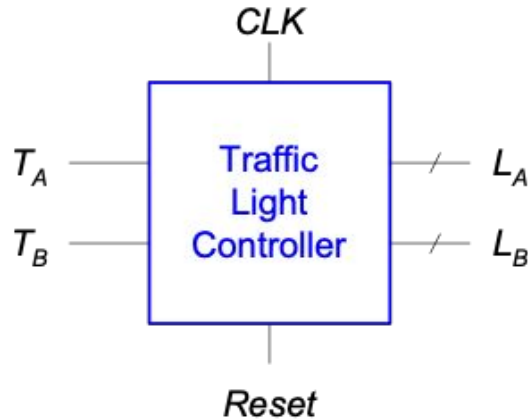
We would like to control the traffic light so that:

- The two lights do not cause accidents (can't both be green!)
- The appropriate light turns green when traffic is present

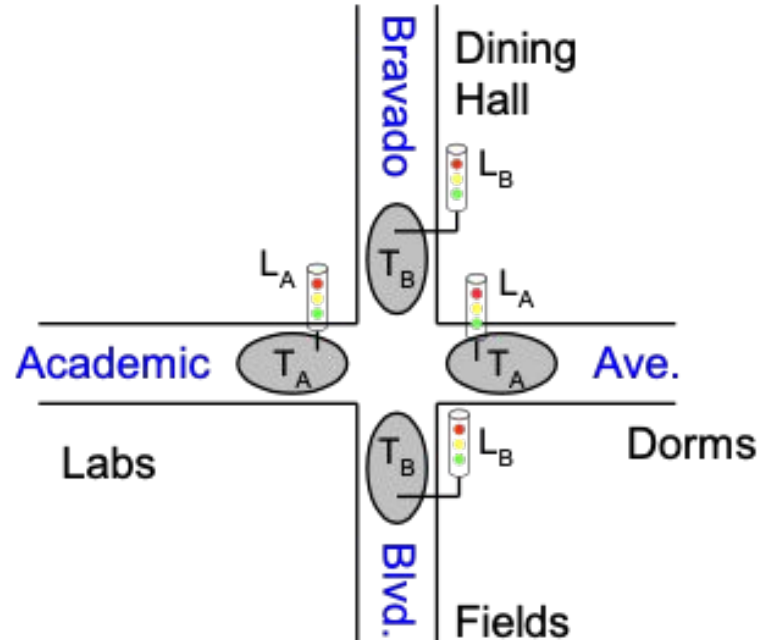


# So squishy! Let's try an example.

Here is our basic system:



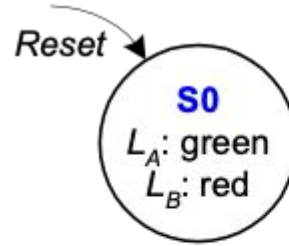
... now we need to define the **state transitions** that should occur under each circumstance



# FSM State Transition Diagrams

---

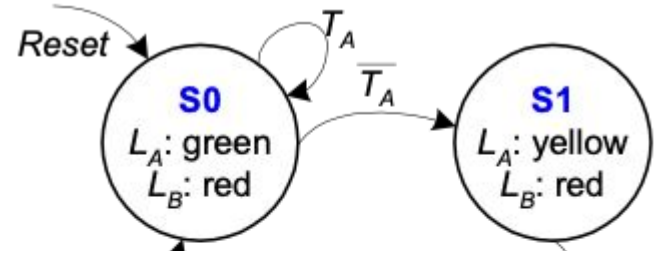
- We will start with a **Moore**-style FSM, and learn the differences later
- Each possible **state** is represented by a circle
- **Transitions** between states are drawn as arcs



# FSM State Transition Diagrams

---

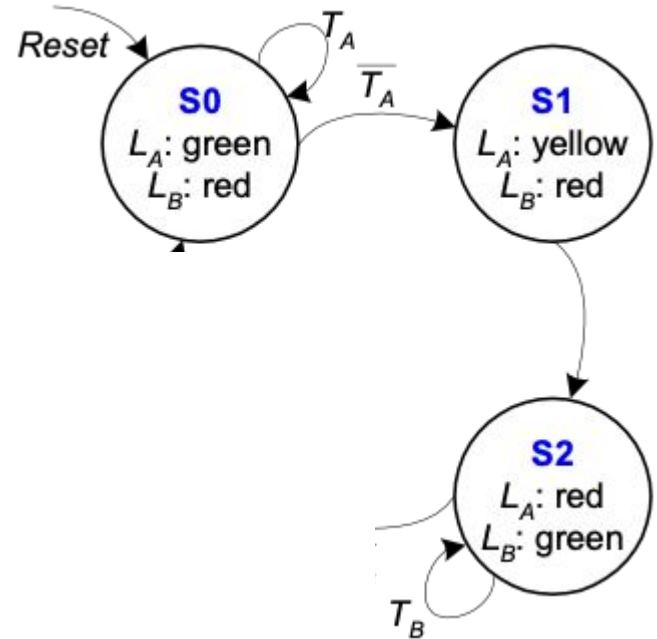
- We will start with a **Moore**-style FSM, and learn the differences later
- Each possible **state** is represented by a circle
- **Transitions** between states are drawn as arcs



# FSM State Transition Diagrams

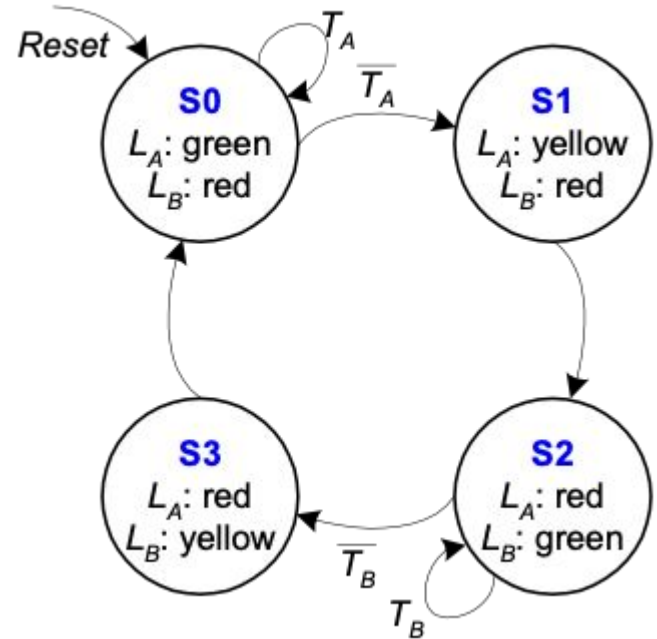
---

- We will start with a **Moore**-style FSM, and learn the differences later
- Each possible **state** is represented by a circle
- **Transitions** between states are drawn as arcs

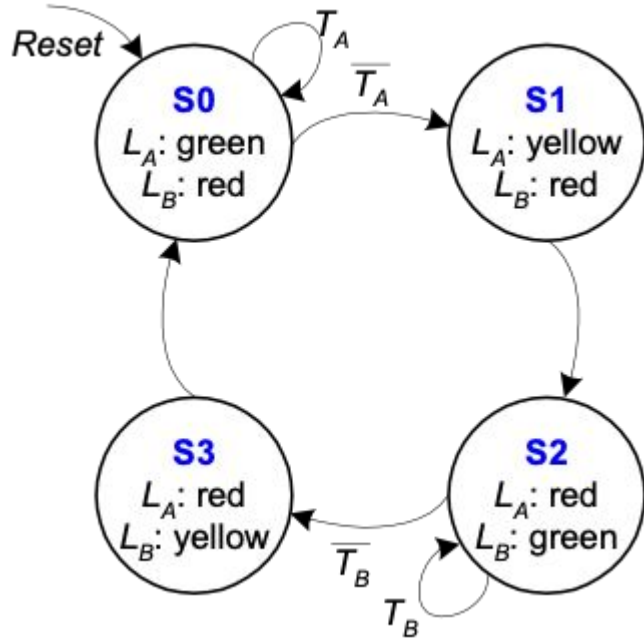


# FSM State Transition Diagrams

- We will start with a **Moore**-style FSM, and learn the differences later
- Each possible **state** is represented by a circle
- **Transitions** between states are drawn as arcs

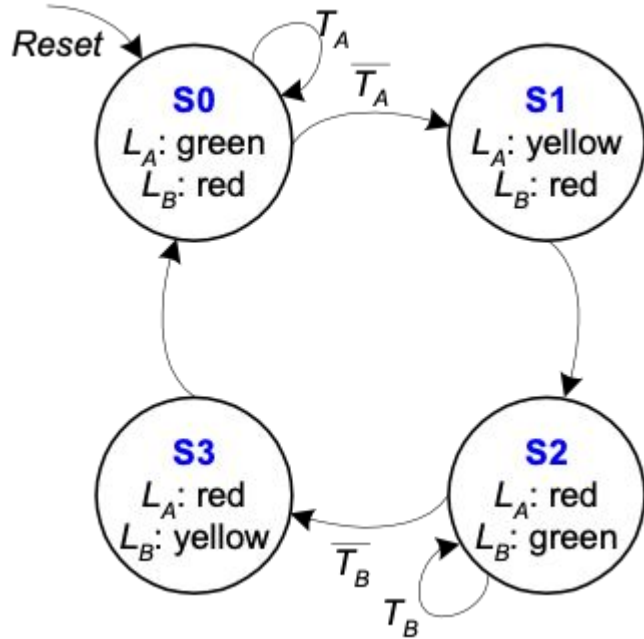


# Translate behavior to a *State Transition Table*



Current State	Inputs		Next State
$S$	$T_A$	$T_B$	$S'$
S0	0	X	
S0	1	X	
S1	X	X	
S2	X	0	
S2	X	1	
S3	X	X	

# Translate behavior to a *State Transition Table*



Current State	Inputs		Next State
$S$	$T_A$	$T_B$	$S'$
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0



# Wrap-Up September 19

---

- Coming up next!
  - FSMs and Sequential Logic for complex designs with feedback
- Logistics, Reminders
  - Evening help sessions 7-9PM on Sundays, Tuesdays, Thursdays in C107
  - LP Office Hours at 2:30-4PM Today
  - Weekly Exercises Due Friday 5PM
  - Lab 2 due Monday 10PM
  - Lab 0 Feedback via Moodle by end-of-week (expect a ~1.5-2 week turnaround as labs grow in complexity)
- FEEDBACK
  - <https://forms.gle/5Aafcm3iJthX78jx6>



SCAN ME



**CS FALL RECEPTION**

Connect with students and faculty and enjoy refreshments!

**SEPT. 20TH 3-4PM**  
**SCCE-Basement/A013**

**Featuring Research Lightning Talks By:**

-   
**Jesse Looney**  
Hardware
-   
**Soyon Choi**  
Machine Learning
-   
**Madi Gudín**  
Hardware