Finite State Machines

Computer Architecture

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CS3501 - 2025I
PROF.: JGONZALEZ@UTEC.EDU.PE
SRC: HARRIS, HARRIS - DIGITAL DESIGN AND COMPUTER ARCHITECTURE

Executive Summary

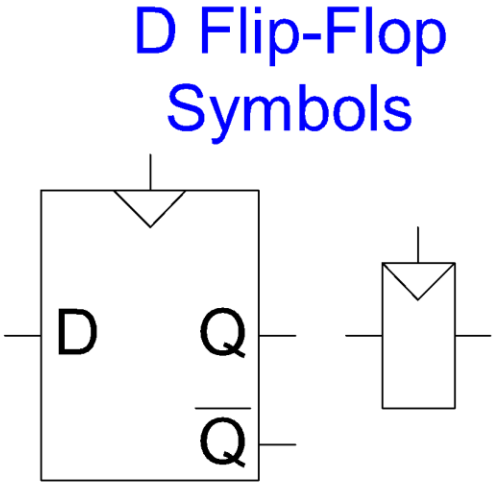
- **Motivation: Combinational circuits with sequential circuits allow to implement a complex design**
- **Problem: We need a procedure to transform our design idea to a circuit**
- **Overview:**
  - **State definitions**.
  - Finite State Machine (FSM) design.
  - FSM examples.
- **Conclusion: A complex idea can be implemented as a digital design using sequential and combinational building blocks**

Outline

- Introduction
- Definitions
- Finite State Machines
- Conclusion

Recall: D Flip-Flop

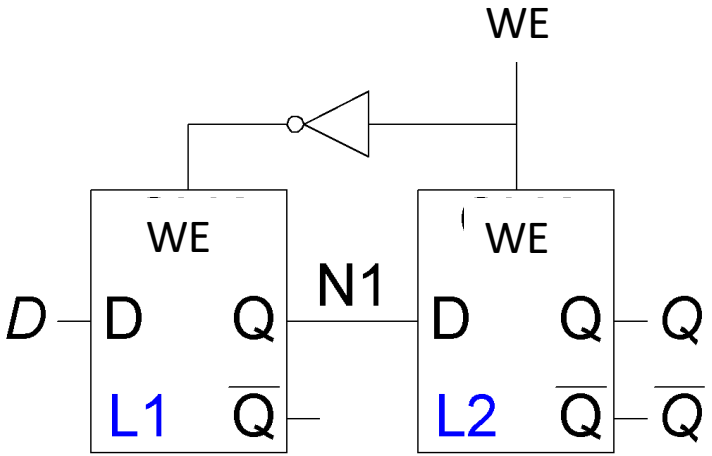
- **Inputs:** WE, D
- **Function**
  - Samples D on rising edge of WE
    - When **WE rises from 0 to 1, D passes through to Q**
    - Otherwise, Q holds its previous value
  - **Q changes only on rising edge of WE**
- **Activated on the WE edge** (edge-triggered).



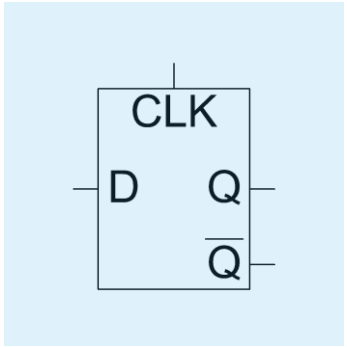
D Flip-Flop operation is associated with a **CLK (clock) instead of a WE**.

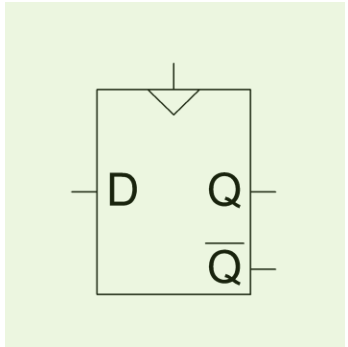
Recall: D Flip-Flop Internal Circuit

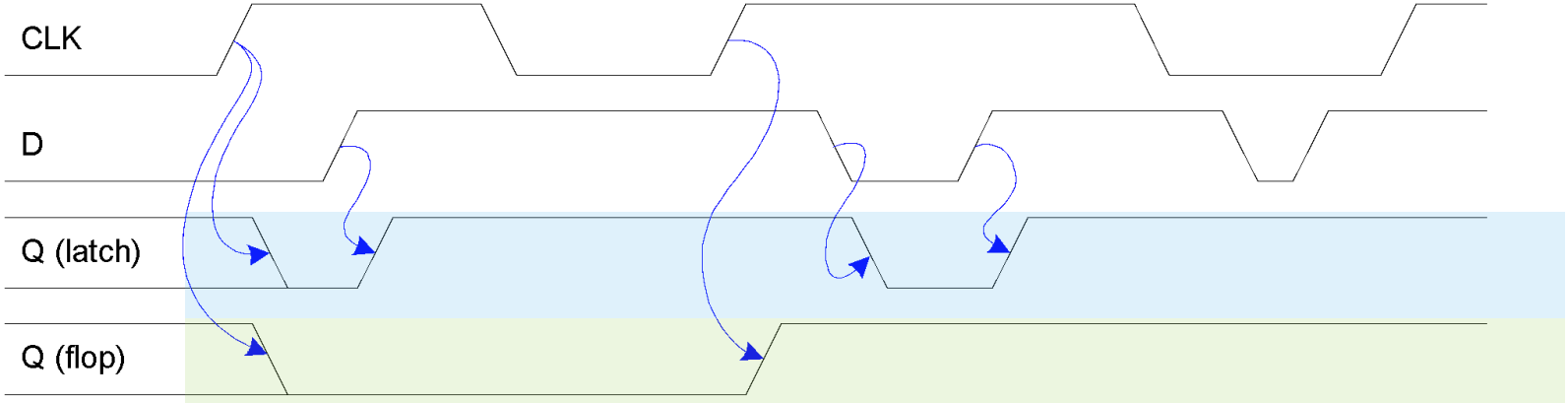
- **Two back-to-back latches** (L1 and L2) **controlled by complementary WE**
- When **WE = 0**
  - L1 is transparent
  - L2 is opaque
  - D passes through to N1
- When **WE = 1**
  - L2 is transparent
  - L1 is opaque
  - N1 passes through to Q
- Thus, on the edge of the WE (when **WE rises from 0 → 1**)
  - D passes through to Q



Recall: Gated D Latch vs D Flip-Flop







At the lab we use a clock (clk) to change the Q value using D input

Outline

Introduction

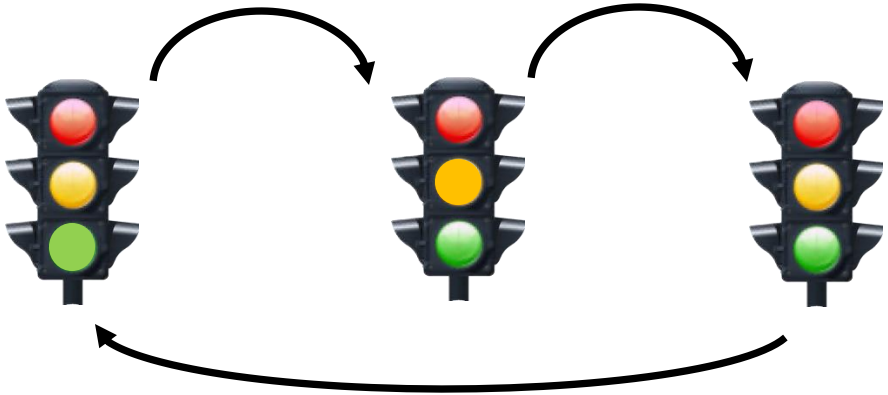
Definitions

Finite State Machines

Conclusion

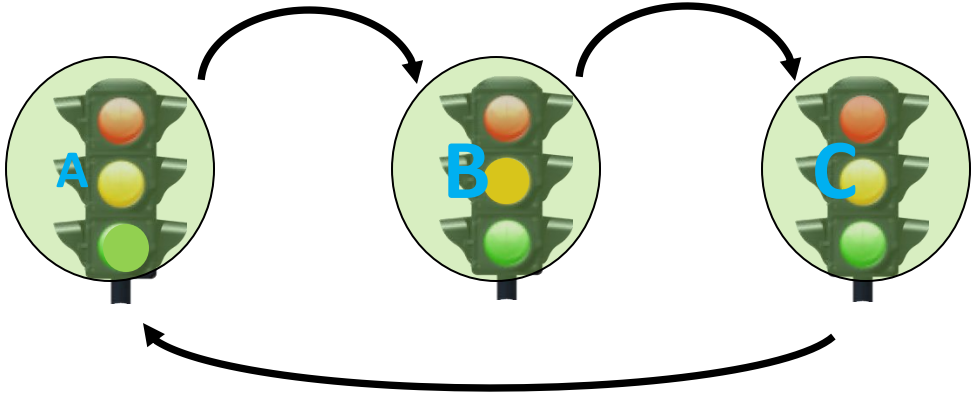
State Definitions

- **State:**
  - **In a system**, it is a **snapshot** of all the important elements (at the time of taking the snapshot).
  - **In a circuit, all the information about a circuit** necessary to define its future behavior.
    - **State elements:** "store the state" or store one bit (e.g.: latches, flip-flops).
- **Diagram example:** a semaphore

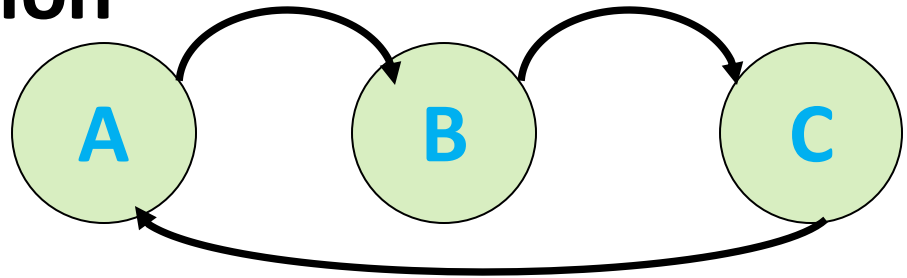


State Definitions

- **State:**
  - **In a system**, it is a **snapshot** of all the important elements (at the time of taking the snapshot).
  - **In a circuit, all the information about a circuit** necessary to define its future behavior.
    - **State elements:** "store the state" or store one bit (e.g.: latches, flip-flops).
- **Diagram example:** a semaphore



State Transition



- **Clock** is a general mechanism that **triggers transition from one state to another** in a sequential circuit
  - Clock **synchronizes state changes** across many sequential circuit elements
  - Combinational logic evaluates for the length of the clock cycle

CLK: 0

- **At the start of a clock cycle** ( ), system state changes
  - **During a clock cycle, the state stays constant**
  - In the semaphore example, **we are assuming the traffic light stays in each state an equal amount of time**

Outline

Introduction

Definitions

Finite State Machines

Conclusion

Finite State Machine (FSM)

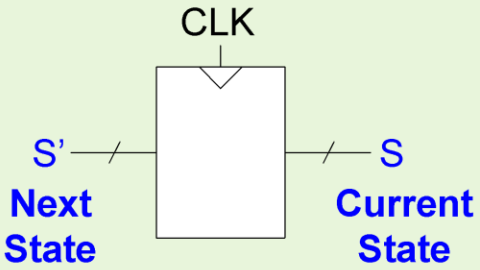
- A **discrete-time model of a stateful system**
- **Each state represents a snapshot** of the system at a given time
- **Can model** a semaphore, washing machine, a computer processor, etc.

- **An FSM state diagram can visually represent**:
  1. All possible states
  2. Transition conditions

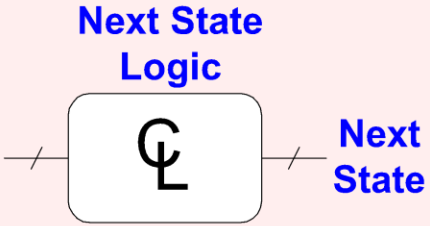
- **It is defined as finite because** it has:
  1. A **finite** number of states (snapshot)
  2. A **finite** number of external inputs
  3. A **finite** number of external outputs
  4. An explicit specification of all state transitions
  5. An explicit specification of what determines each external output value

FSM Elements

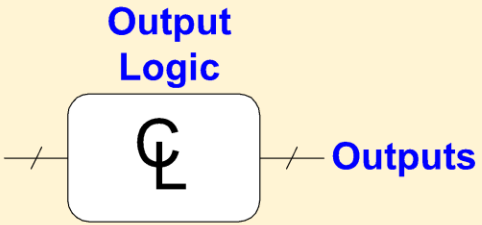
- **Sequential Circuits**
  - **State register**
    - Stores current state
    - Loads next state at clock edge



- **Combinational Circuits**
  - **Next state logic:**
    - Computes the next state



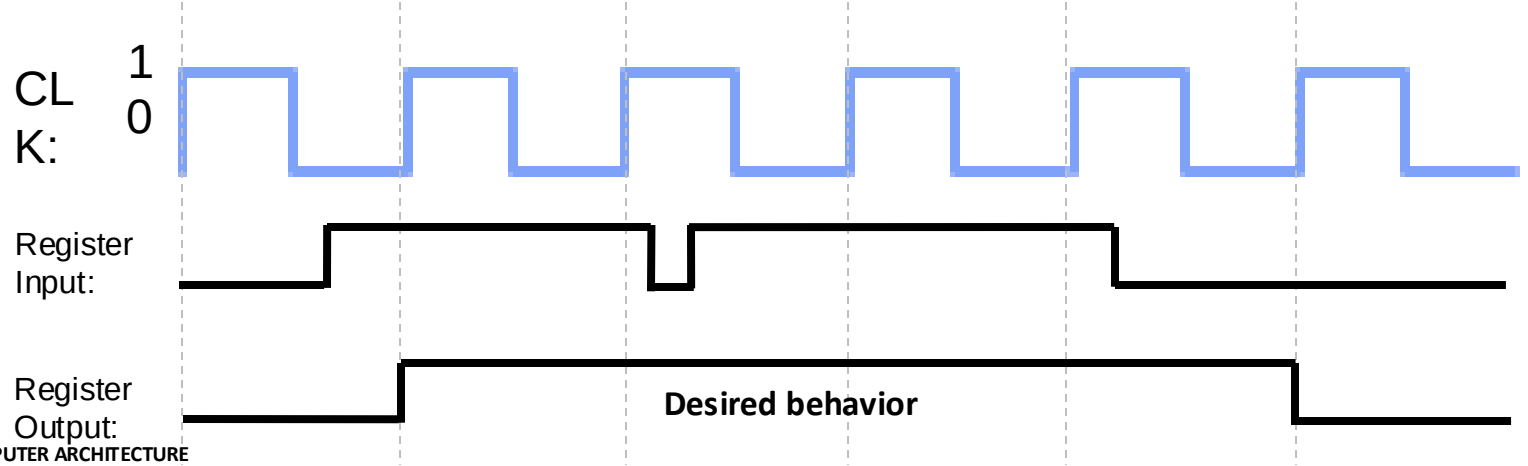
  - **Ouput logic:**
    - Computes the outputs



State Register Implementation

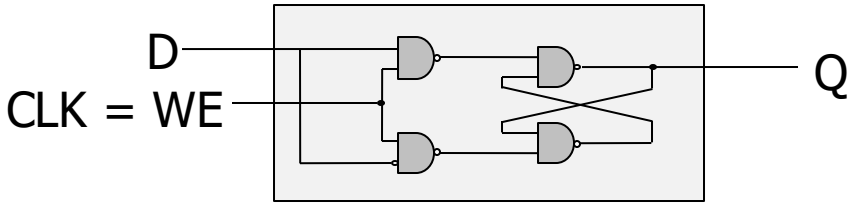
- How can we implement a **state register**? Two properties:
  1. We need to store data at the **beginning** of every clock cycle

  2. The data must be **available** during the entire clock cycle

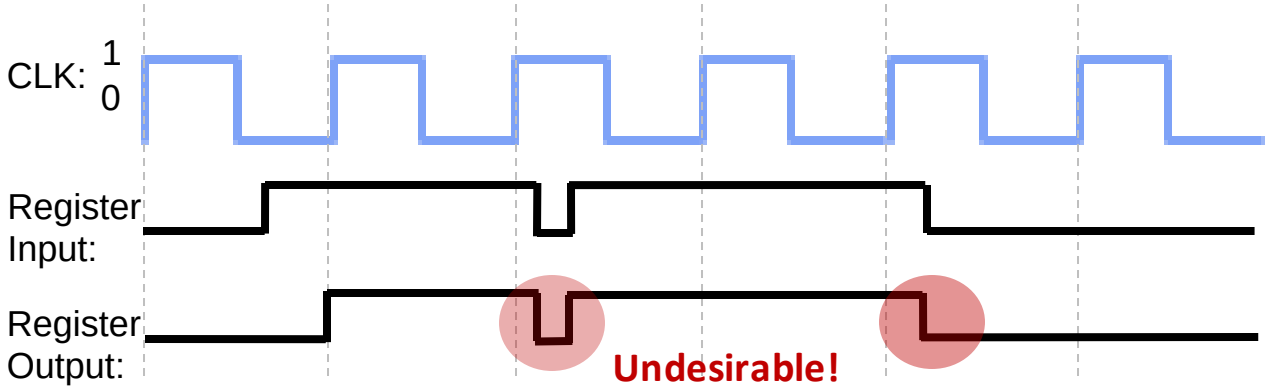


Problem: State Register with Latches

Recall the Gated D Latch

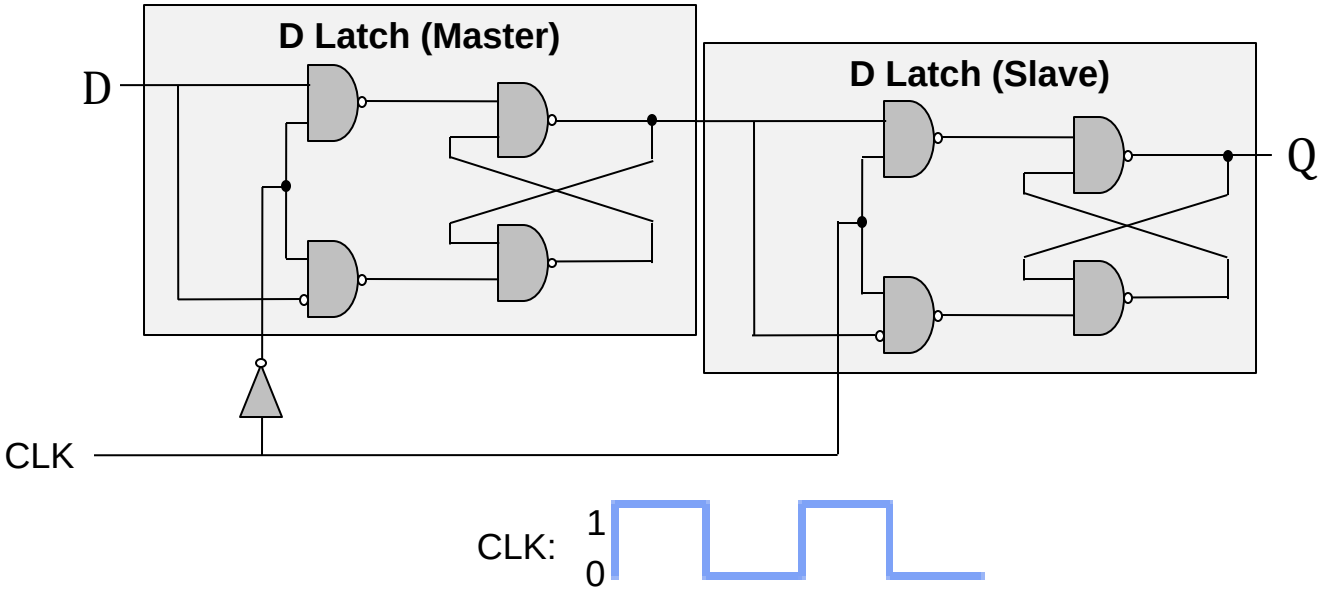


- If we wire a clock to WE of a latch
  - **Whenever the clock is high,** the latch propagates **D** to **Q**
  - **The latch is transparent**



Solution: State Register with D Flip-Flop

- 1) state change on clock edge, 2) data available for full cycle

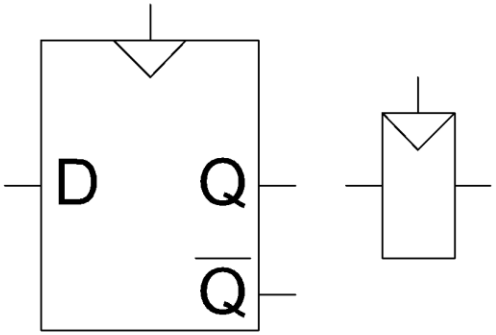


- **When the clock is low,** master propagates **D** to the input of slave (Q unchanged)
- **Only when the clock is high,** slave latches **D (Q stores D)**
  - At the rising edge of clock (clock going from 0->1), Q gets assigned D

Recall: D Flip-Flop

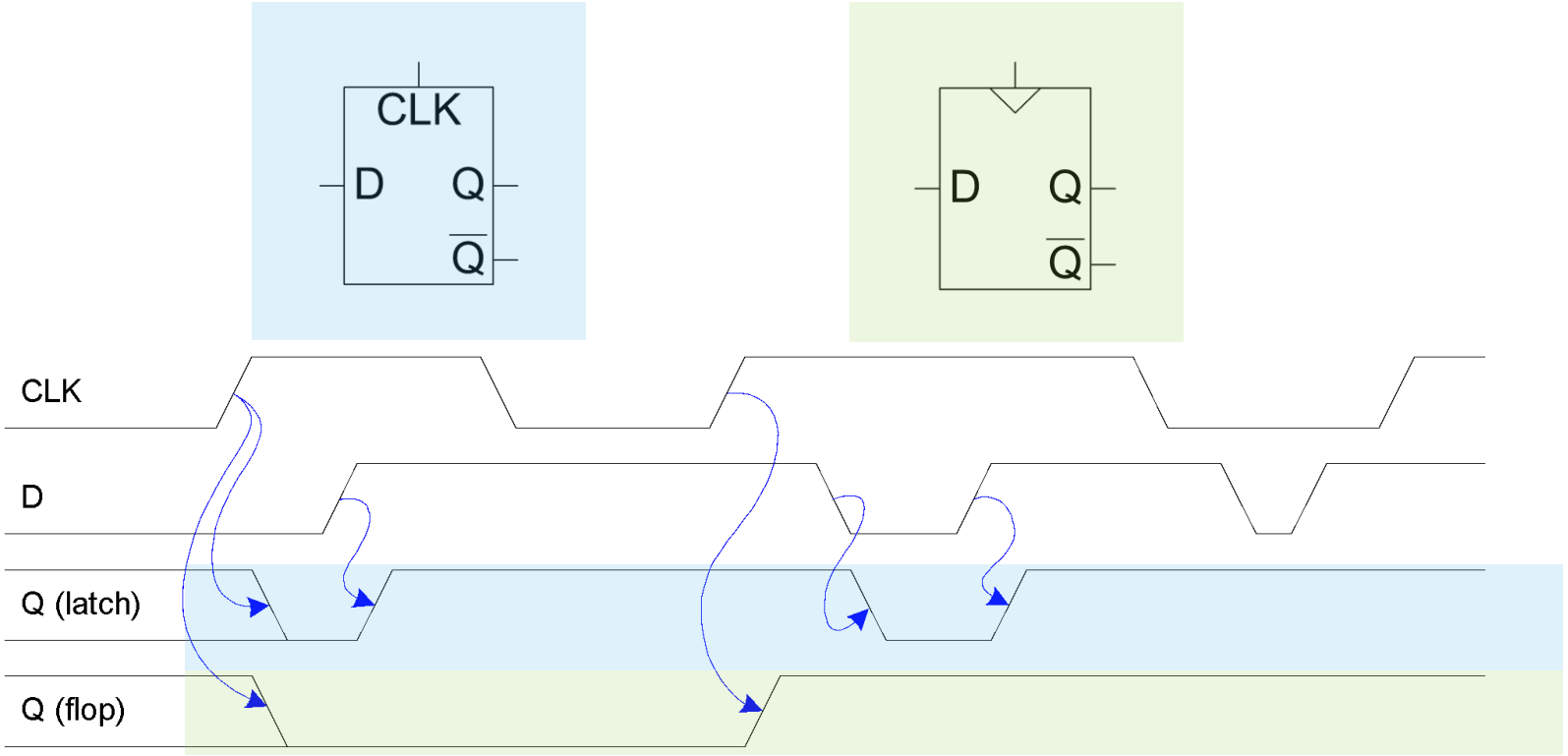
- **Inputs:** WE, D
- **Function**
  - Samples D on rising edge of WE
    - When **WE rises from 0 to 1, D passes through to Q**
    - Otherwise, Q holds its previous value
  - **Q changes only on rising edge of WE**
- **Activated on the WE edge** (edge-triggered).

D Flip-Flop Symbols



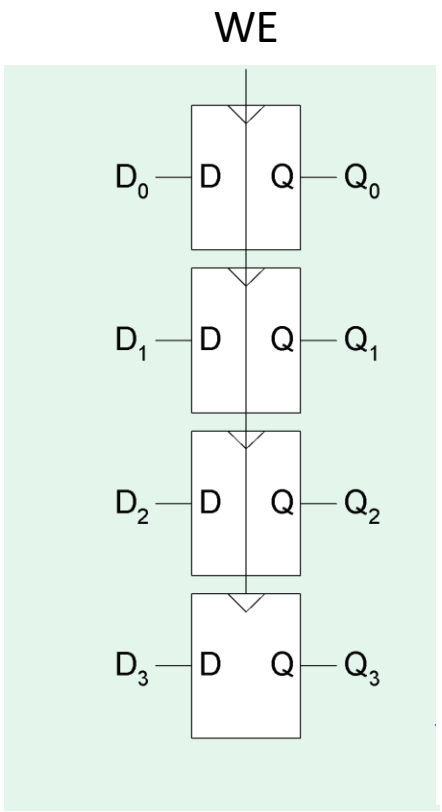
D Flip-Flop operation is associated with a **CLK (clock) instead of a WE**.
We **will explain the clock in later lectures.**

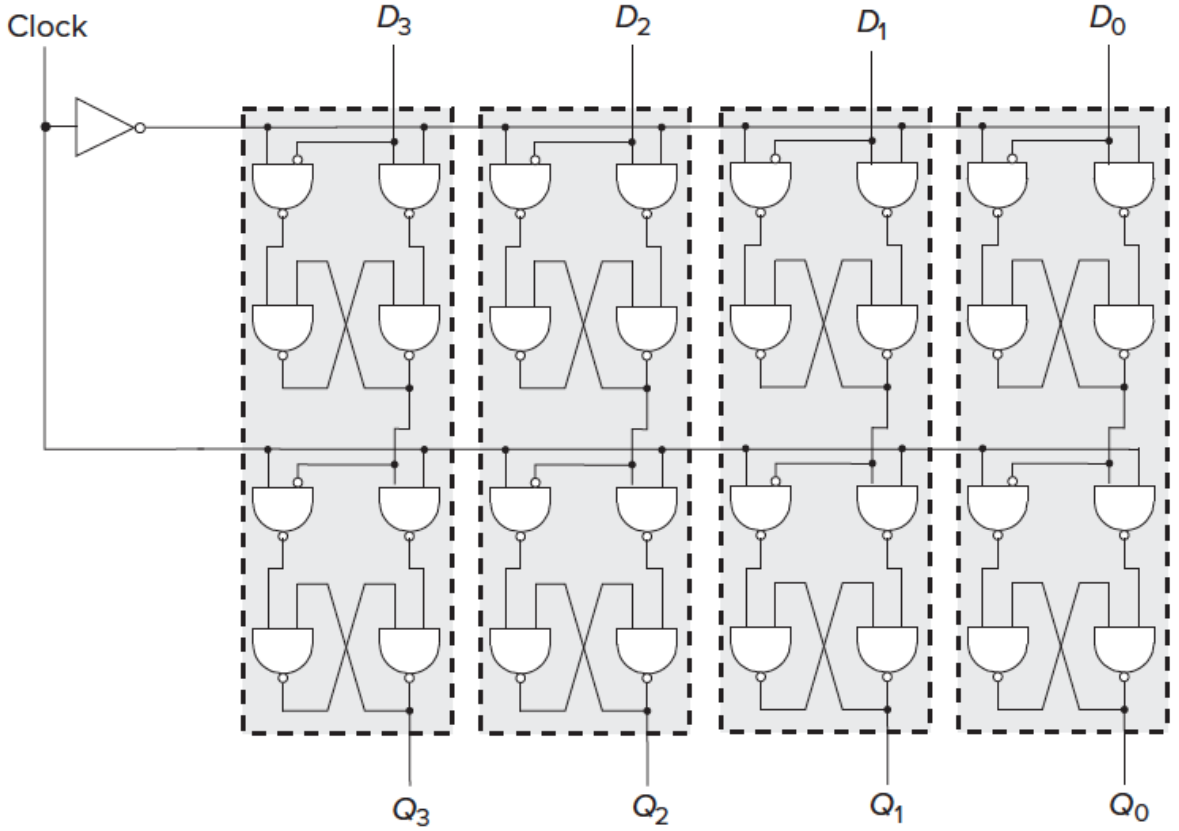
Recall: Gated D Latch vs D Flip-Flop



At the lab we use a clock (clk) to change the Q value using D input

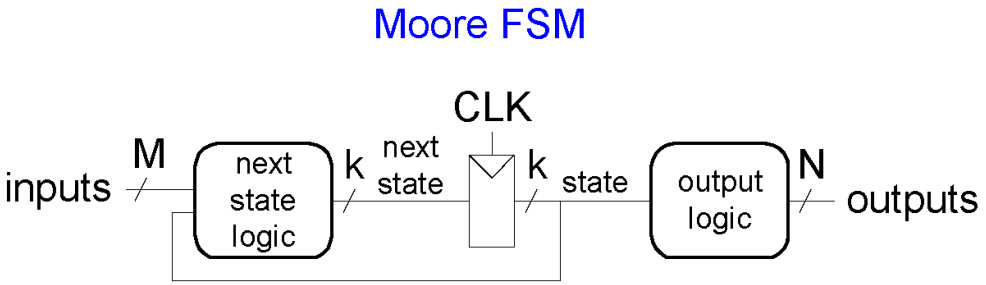
Recall: 4-bit Register with D Flip Flop



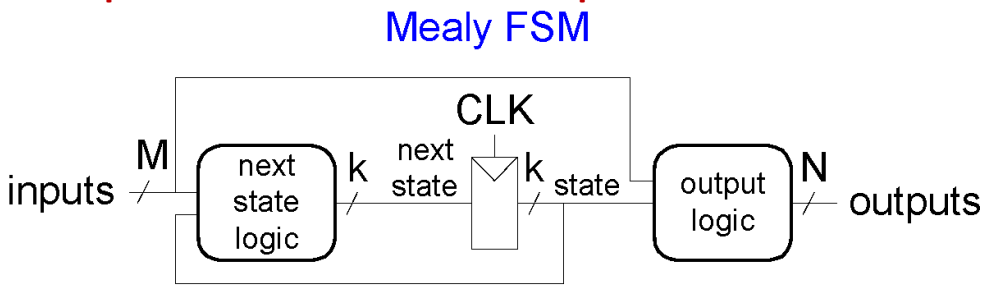


Types of FSM

- **Next state** determined by current state and inputs
- Two types of finite state machines differ in **output logic**:
  1. Moore FSM: **outputs depend only on current state**



  1. Mealy FSM: **outputs depend on current state *and* inputs**



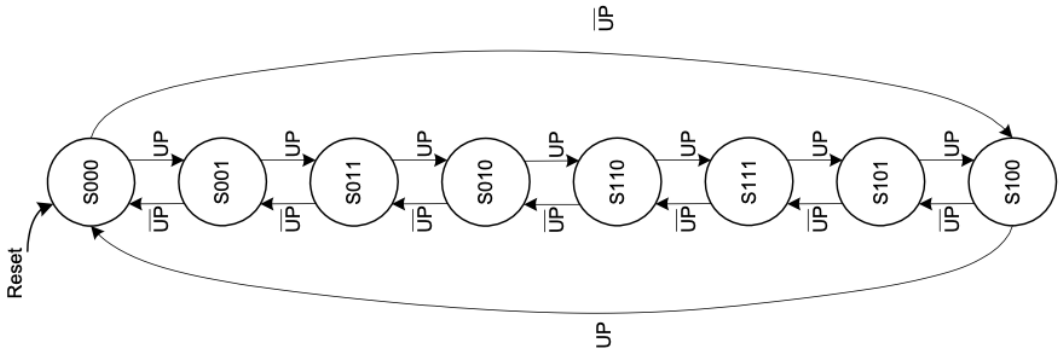
FSM example

- Design a modulo 8 Gray code counter with an UP/DOWN counter. Define an UP input. If UP= 1, the counter advances to the next number. If UP= 0, the counter retreats to the previous number.
- Add an output Y, where Y=1 every time the Gray count has only one zero.

FSM Black Box

- **Inputs:** *?*
- **Outputs:** *?*

1. Design the state diagram



2. Write down the next state logic

current state $s_{2:0}$	input up	next state $s'_{2:0}$
000	1	001
001	1	011
011	1	010
010	1	110
110	1	111
111	1	101
101	1	100
100	1	000
000	0	100
001	0	000
011	0	001
010	0	011
110	0	010
111	0	110
101	0	111
100	0	101

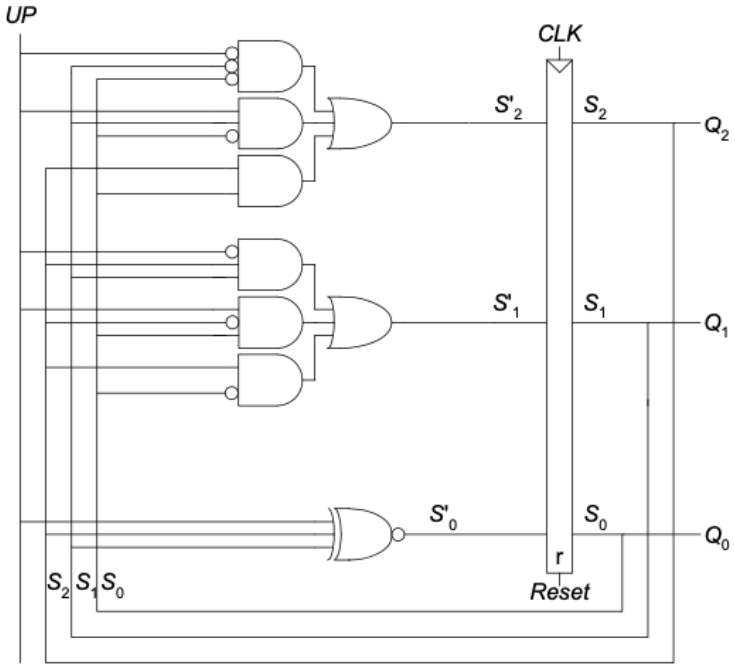
$$S'_2 = UPS_1\overline{S_0} + \overline{UP}\,\overline{S_1}\,\overline{S_0} + S_2S_0$$

$$S'_1 = S_1\overline{S_0} + UP\overline{S_2}S_0 + \overline{UP}S_2S_1$$

$$S'_0 = UP \oplus S_2 \oplus S_1$$

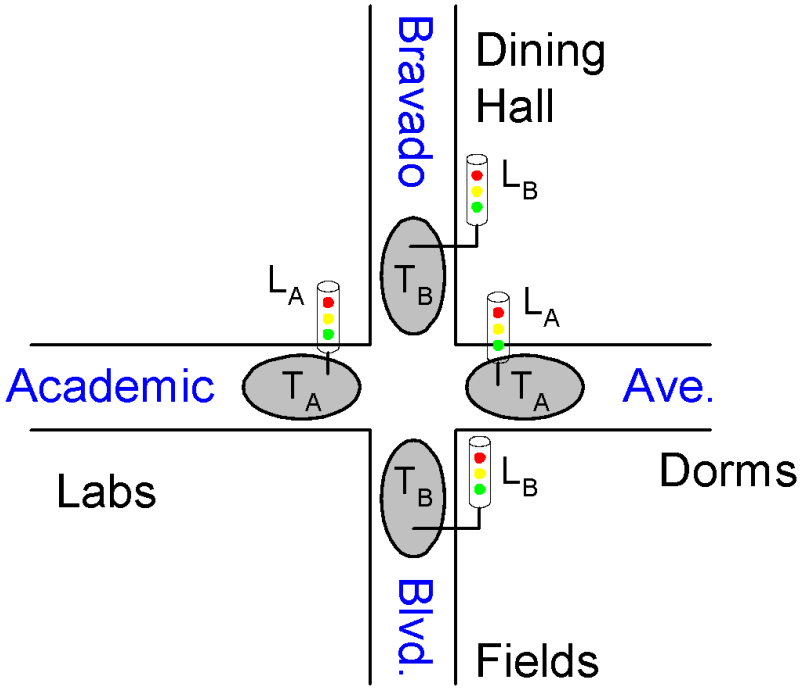
3. Write down the output logic

4.Draw the schematic



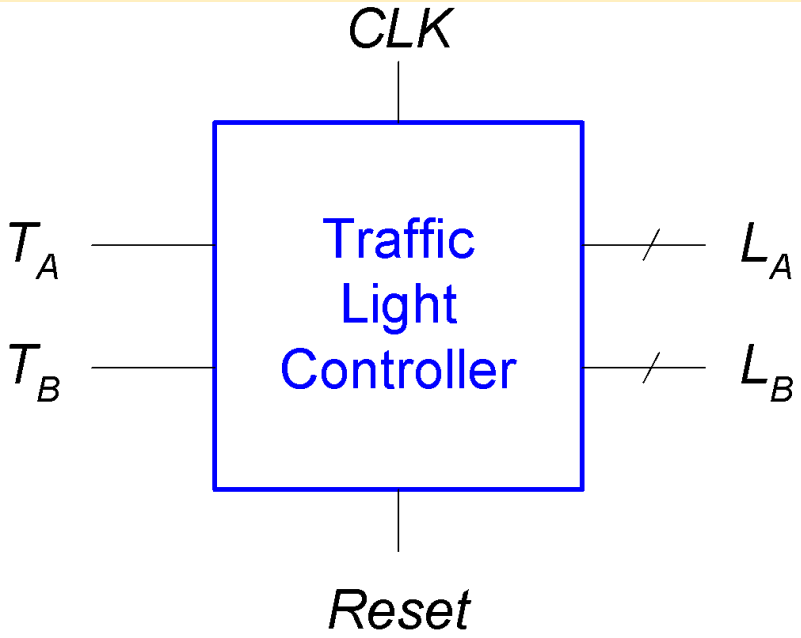
Another FSM Example

- Traffic light controller
  - **Inputs:**
    - Traffic sensors: T_A, T_B (TRUE when there's traffic)
  - **Outputs:**
    - Lights: L_A, L_B
  - **States:** transitions each 5 seconds



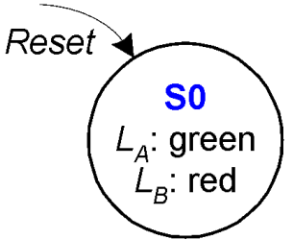
FSM Black Box

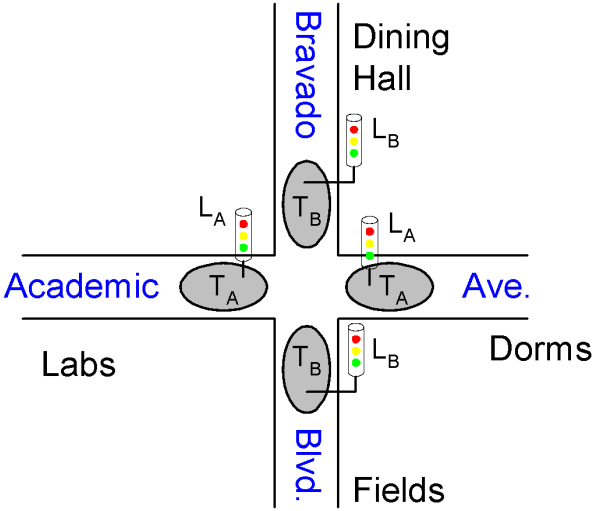
- **Inputs:** *CLK*, *Reset*, T_A, T_B
- **Outputs:** L_A, L_B



FSM State Transition Diagram

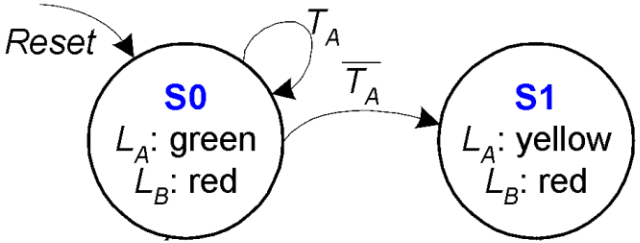
- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs

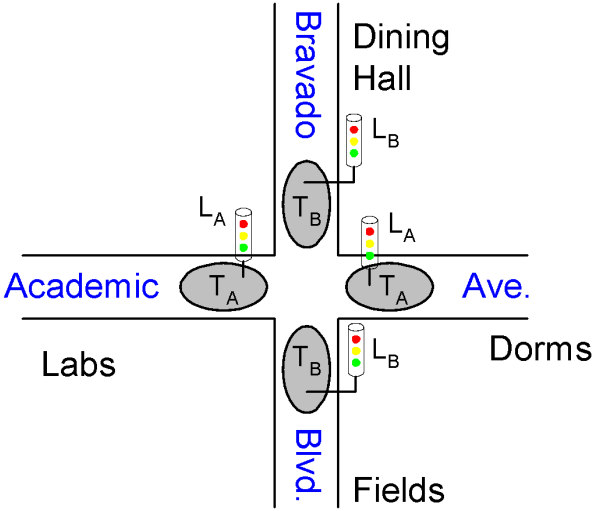




FSM State Transition Diagram

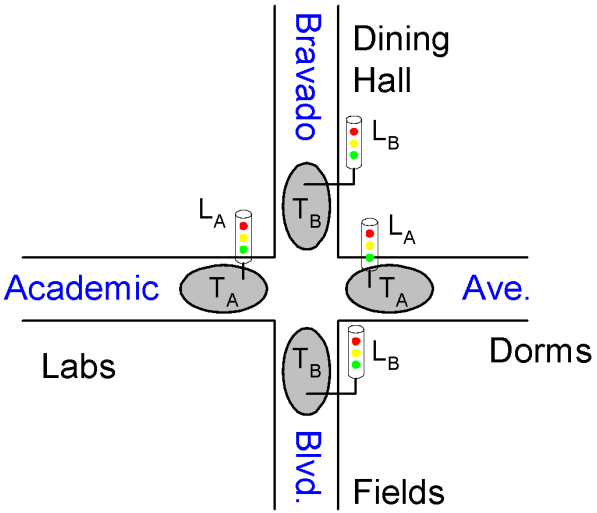
- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs

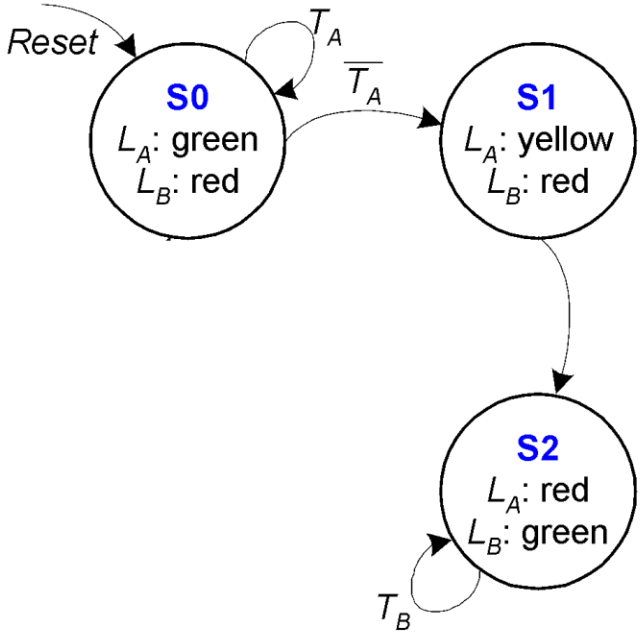




FSM State Transition Diagram

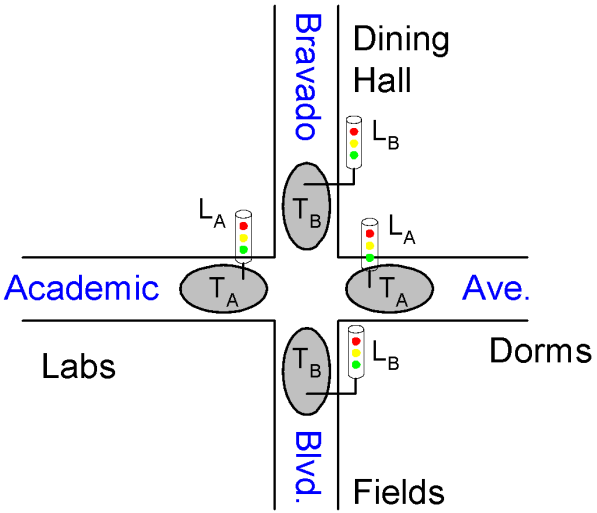
- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs

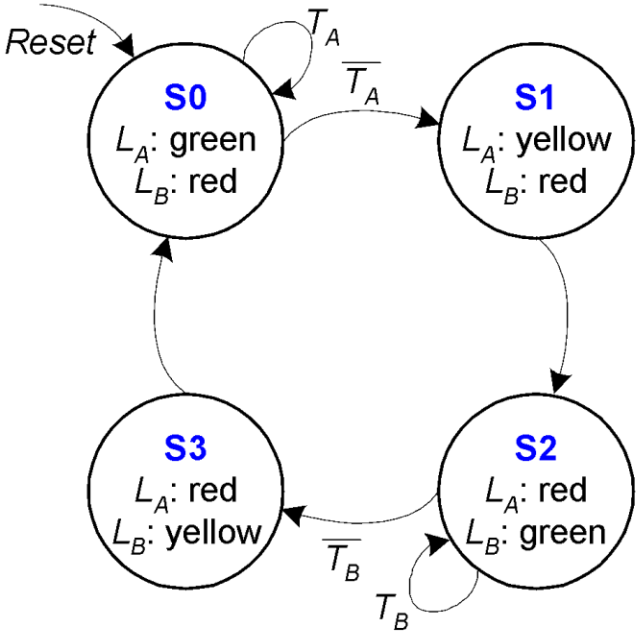




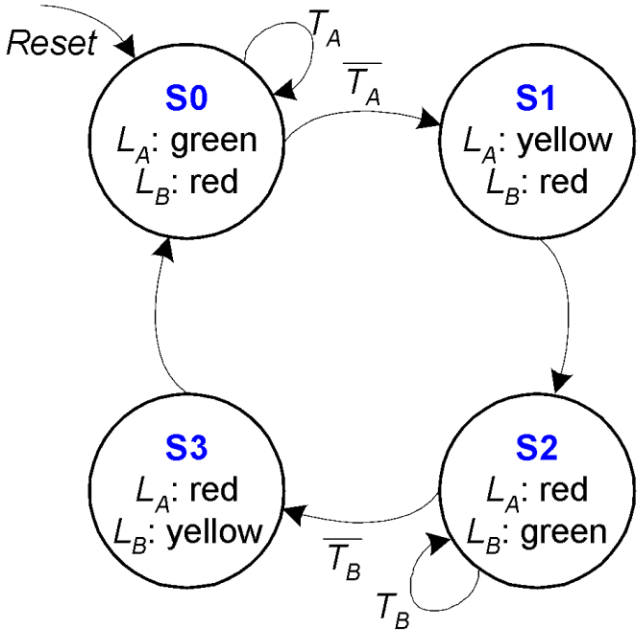
FSM State Transition Diagram

- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs



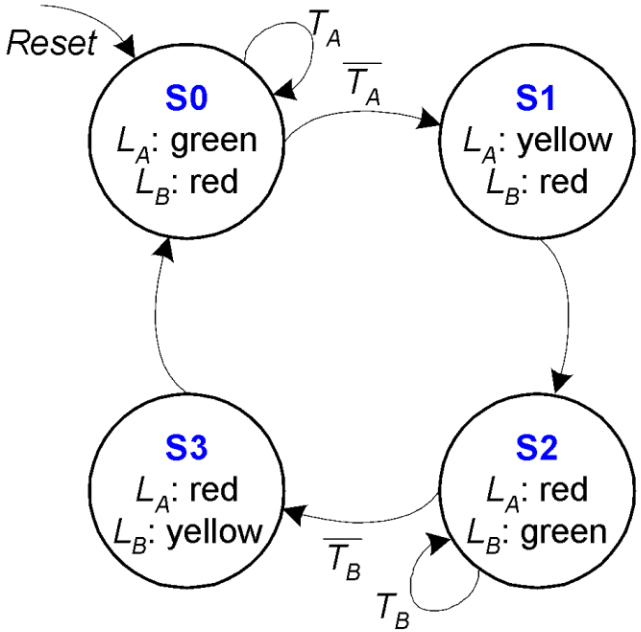


FSM State Transition Table



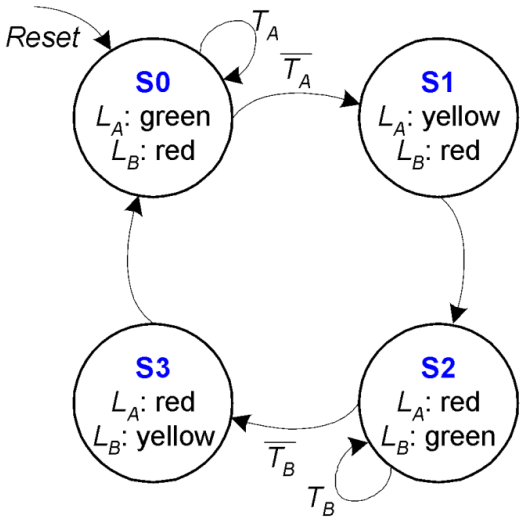
Current State	Inputs		Next State
S	T_A	T_B	S'
S0	0	X	
S0	1	X	
S1	X	X	
S2	X	0	
S2	X	1	
S3	X	X	

FSM 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

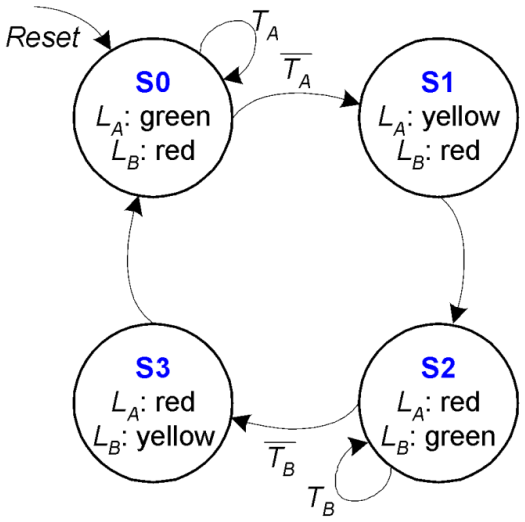
FSM Encoded State Transition Table



Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X		
0	0	1	X		
0	1	X	X		
1	0	X	0		
1	0	X	1		
1	1	X	X		

State	Encoding
S0	00
S1	01
S2	10
S3	11

FSM Encoded State Transition Table



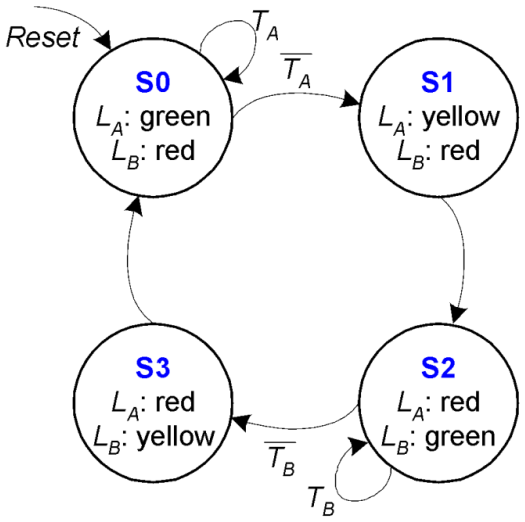
Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

State	Encoding
S0	00
S1	01
S2	10
S3	11

$$S'_1 = (S_1 \cdot S_0) + (S_1 \cdot S_0 \cdot T_B) + (S_1 \cdot S_0 \cdot T_B)$$

$$S'_0 = (S_1 \cdot S_0 \cdot T_A) + (S_1 \cdot S_0 \cdot T_B)$$

FSM Encoded State Transition Table



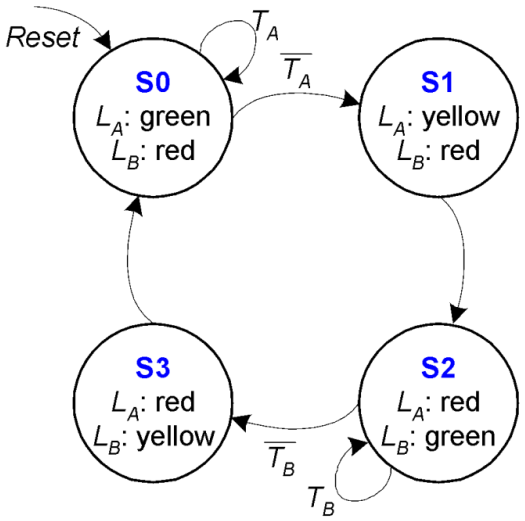
Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

State	Encoding
S0	00
S1	01
S2	10
S3	11

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

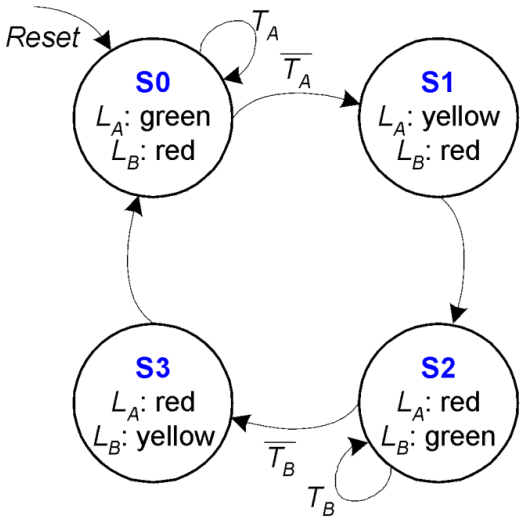
FSM Output Table



Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0				
0	1				
1	0				
1	1				

Output	Encoding
green	00
yellow	01
red	10

FSM Output Table

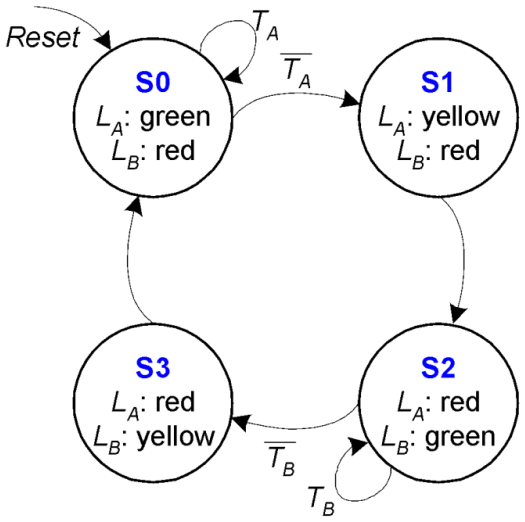


Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

Output	Encoding
green	00
yellow	01
red	10

$$L_{A1} = S_1$$
$$L_{A0} = S_1 \cdot S_0$$

FSM Output Table

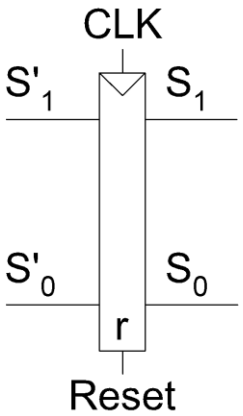


Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

Output	Encoding
green	00
yellow	01
red	10

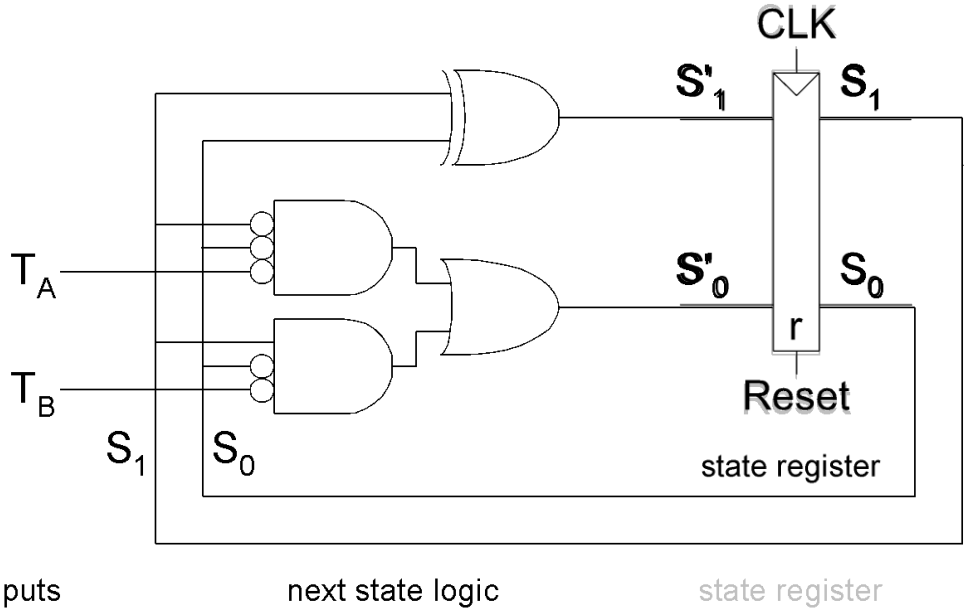
$$L_{A1} = S_1$$
$$L_{A0} = S_1 \cdot S_0$$
$$L_{B1} = S_1$$
$$L_{B0} = S_1 \cdot S_0$$

FSM Schematic: State Register



state register

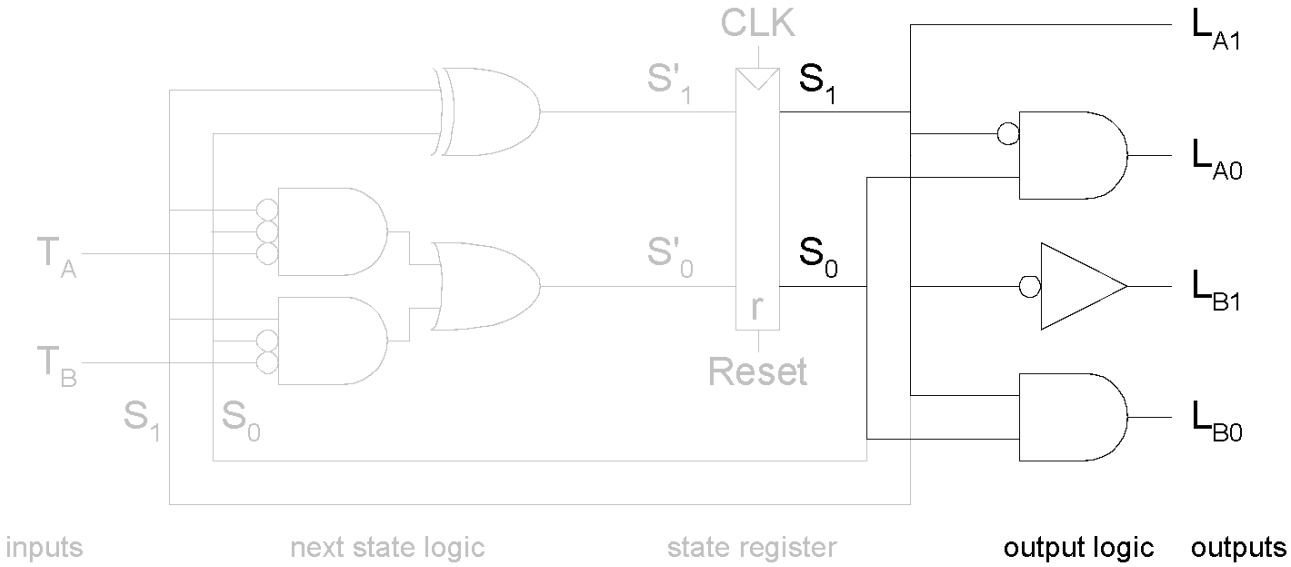
FSM Schematic: Next State Logic



$$S'_1 = S_1 \text{ xor } S_0$$

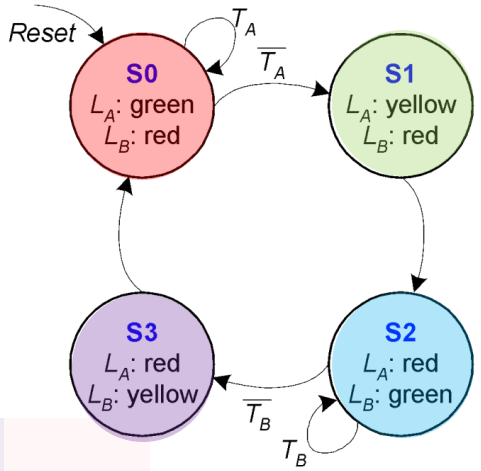
$$S'_0 = (S_1 \cdot S_0 \cdot T_A) + (S_1 \cdot S_0 \cdot T_B)$$

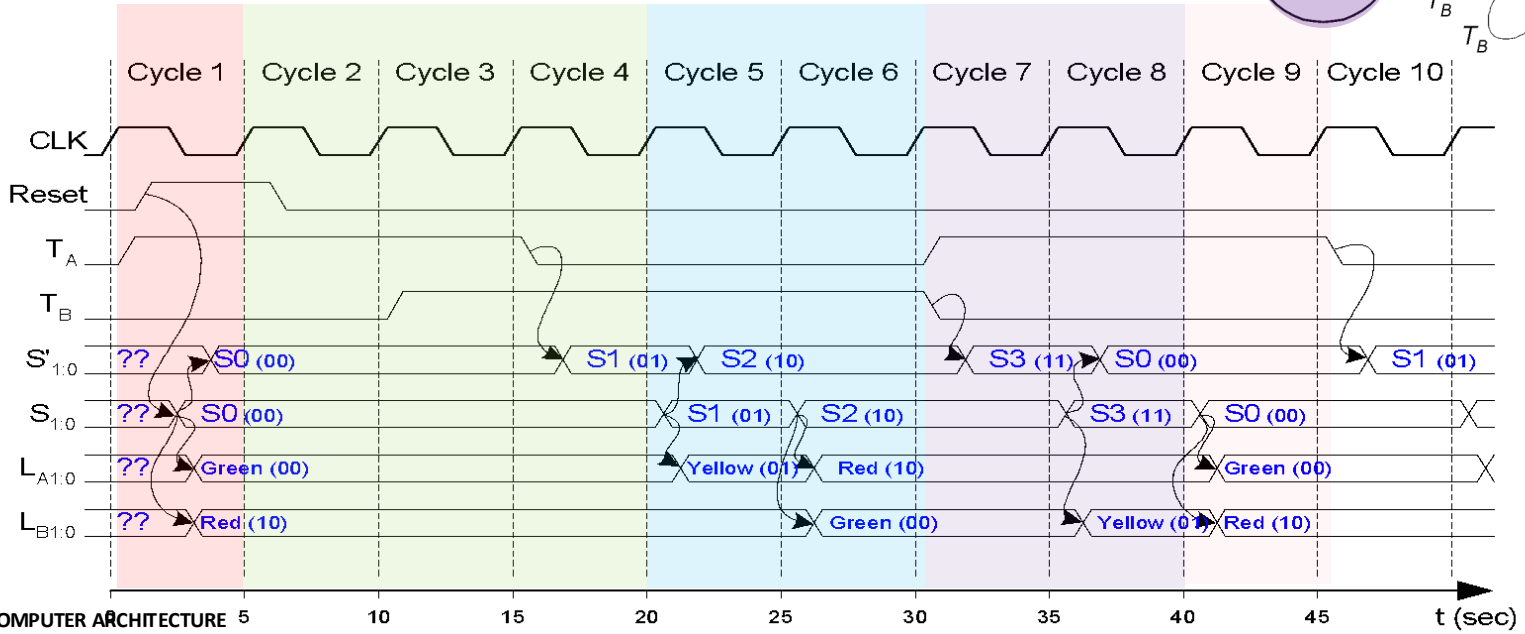
FSM Schematic: Output Logic



$$L_{A1} = S_1$$
$$L_{A0} = S_1 \cdot S_0$$
$$L_{B1} = S_1$$
$$L_{B0} = S_1 \cdot S_0$$

FSM Timing Diagram

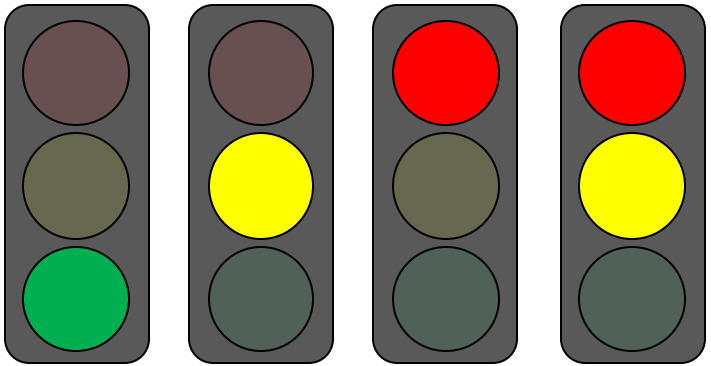




FSM State Encoding

- How do we encode the state bits?
  - **Three common state binary encodings** with **different tradeoffs**
    1. **Fully Encoded**
    2. **1-Hot Encoded**
    3. **Output Encoded**

- Let's see an example **Swiss** Semaphore with 4 states
  - Green, Yellow, Red, Yellow+Red



FSM State Encoding Types

1. **Binary Encoding (Full Encoding):**
   - Use the minimum number of bits used to encode all states
     - Use $log_2(num\_states)$ bits to represent the states
   - *Example states:* 00, 01, 10, 11
   - **Minimizes** # flip-flops, but not necessarily output logic or next state logic

2. **One-Hot Encoding:**
   - Each bit encodes a different state
     - Uses *num_states* bits to represent the states
     - Exactly 1 bit is "hot" for a given state
   - *Example states:* 0001, 0010, 0100, 1000
   - **Simplest design process** – very automatable
   - **Maximizes** # flip-flops, **minimizes** next state logic

FSM State Encoding Types

3. Output Encoding:

- Outputs are **directly accessible** in the state encoding

- For example, since we have **3 outputs** (light color), encode state with **3 bits**, where each bit represents a color
- *Example states:* 001, 010, 100, 110
  - Bit_0 encodes **green** light output,
  - Bit_1 encodes **yellow** light output
  - Bit_2 encodes **red** light output

- **Minimizes** output logic
- Only works for Moore Machines (output function of state)

Which encoding to use?

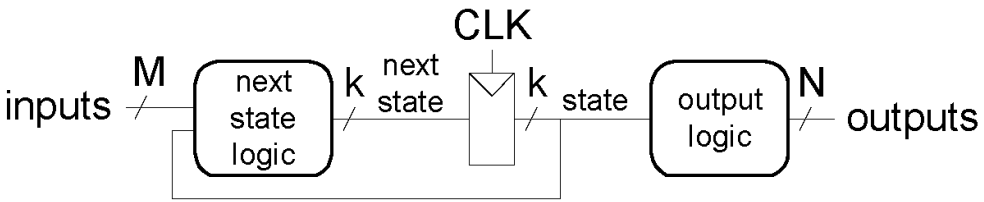
Encoding:

Depends on the designer critera to optimize or follow constraints.

Recall: Types of FSM

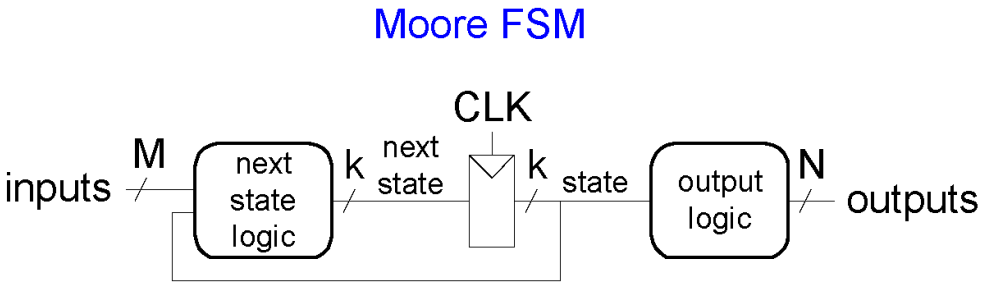
- **Next state** determined by current state and inputs
- Two types of finite state machines differ in **output logic**:
  1. Moore FSM: **outputs depend only on current state**

Moore FSM

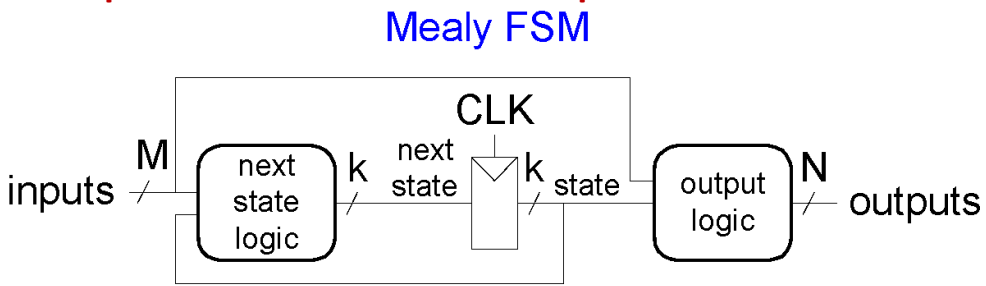


Recall: Types of FSM

- **Next state** determined by current state and inputs
- Two types of finite state machines differ in **output logic**:
  1. Moore FSM: **outputs depend only on current state**

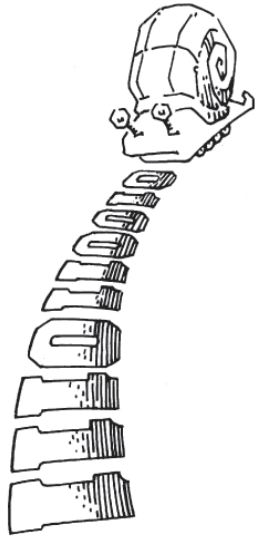


  1. Mealy FSM: **outputs depend on current state *and* inputs**

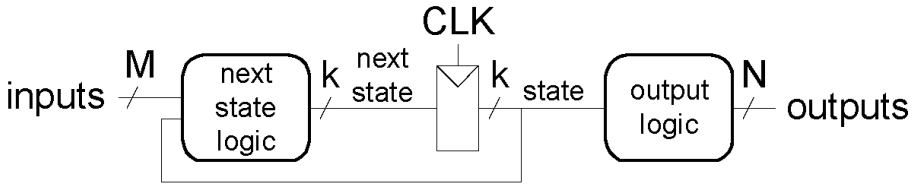


FSM: Moore vs Mealy

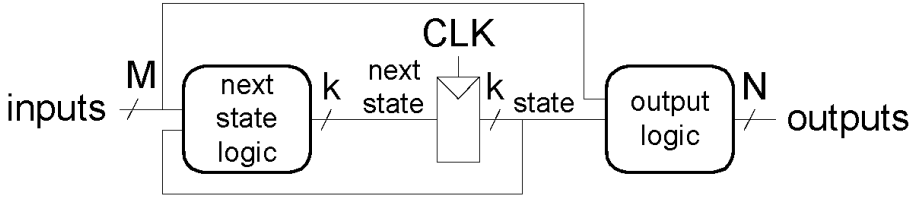
- Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it.
  - **The snail smiles whenever the last two digits** it has crawled over are 01.
- Design **Moore and Mealy FSMs** of the snail's brain.



Moore FSM

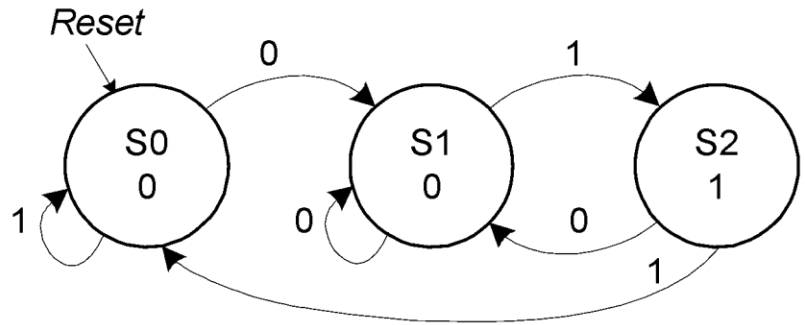


Mealy FSM

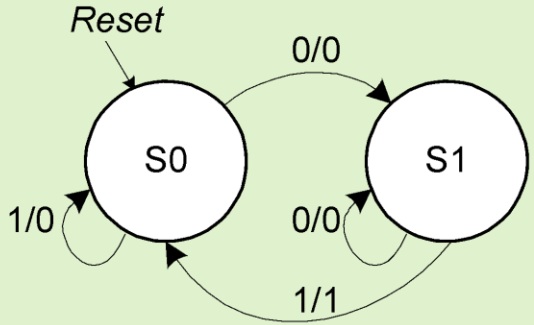


Example Moore vs Mealy

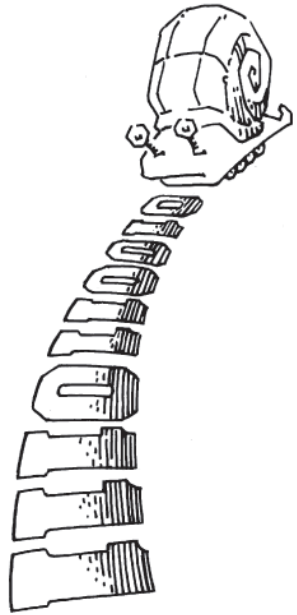
Moore FSM



Mealy FSM



Mealy FSM: arcs indicate input/output



FSM Design Procedure

1. Identify **inputs and outputs**
2. **Sketch state transition diagram**
3. **Write state transition table**
4. **Select state encodings**
5. **For Moore** machine:
   a. Rewrite state transition table with state encodings
   b. Write output table

5. **For a Mealy** machine:
   Rewrite combined state transition and output table with state encodings
6. **Write Boolean equations for next state and output logic**
7. **Sketch the circuit schematic**

Homework

- **Read the FSM Moore to Mealy conversion** (and backwards) in Piazza>resources>lecture notes.
- **Finish Chapter 3** from HH book.
- **Try solving the suggested exercises (check piazza).**

Outline

Introduction

Definitions

Finite State Machines

Conclusion

Conclusions

- **We introduced fundamentals concepts** finite state machines
- **We detailed the design of** finite state machines.
- **A sequential circuit with combinational circuit allows to implement a finite state machine.**
- We conclude that **complex design can be implemented as a digital circuit following design guidelines and defining state transitions.**

Finite State Machines

Computer Architecture

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CS3501 – 2025I
PROF.: JGONZALEZ@UTEC.EDU.PE