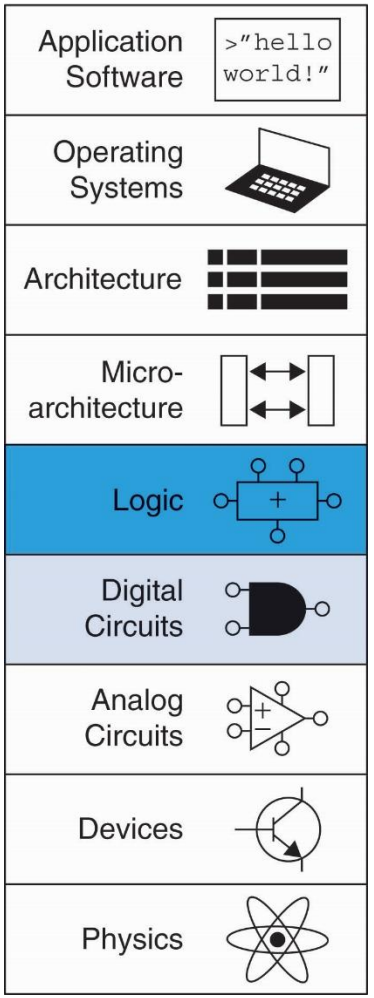
CS222: Computer Architecture

Instructors:
Dr Ahmed Shalaby http://bu.edu.eg/staff/ahmedshalaby14#

الاحترام - الادب - الاخلاق
الطالب - المعيد - الدكتور

Chapter 3 :: Topics

- **Introduction**
- **Latches and Flip-Flops**
- **Synchronous Logic Design**
- **Finite State Machines**
- **Timing of Sequential Logic**
- **Parallelism**



Introduction

- Outputs of sequential logic depend on current *and* prior input values – it has ***memory***.
- Some definitions:
  - **State:** all the information about a circuit necessary to explain its future behavior
  - **Latches and flip-flops:** state elements that store one bit of state
  - **Synchronous sequential circuits:** combinational logic followed by a bank of flip-flops

Sequential Circuits

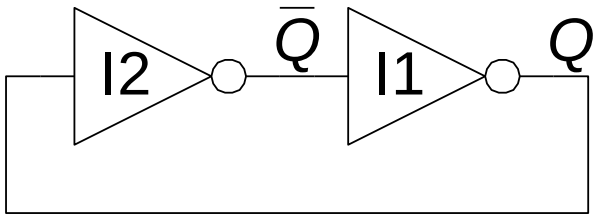
- Give sequence to events
- Have memory (short-term)
- Use feedback from output to input to store information

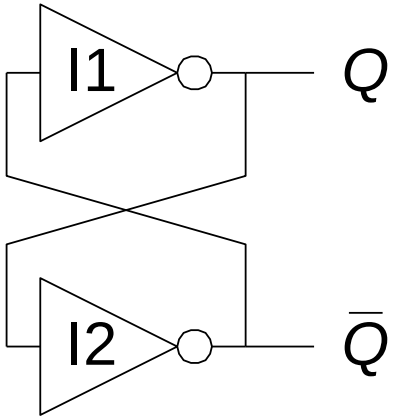
State Elements

- The state of a circuit influences its future behavior
- State elements store state
  - ~~Bistable circuit~~
  - ~~SR Latch~~
  - D Latch
  - D Flip-flop

Bistable Circuit

- Fundamental building block of other state elements
- Two outputs: Q, $\overline{Q}$
- No inputs

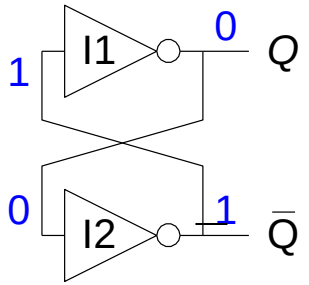




Bistable Circuit Analysis

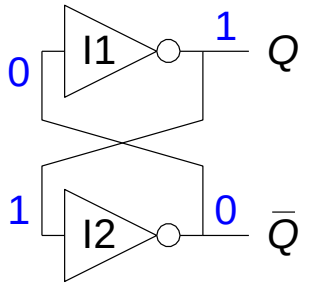
- Consider the two possible cases:
  - **$Q = 0$:**

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



  - **$Q = 1$:**

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



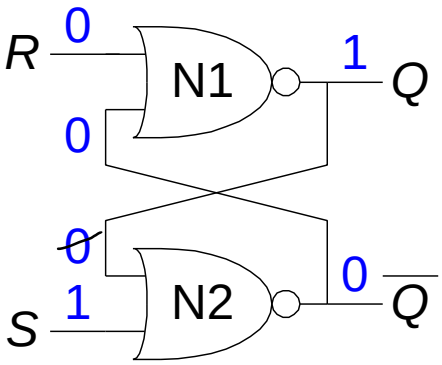
- Stores 1 bit of state in the state variable, Q (or $\bar{Q}$)
- But there are **no inputs to control the state**

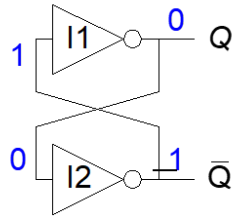
SR Latch Analysis

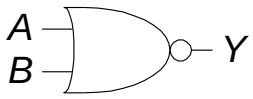
– $S = 1, R = 0$:

then $Q = 1$ and $\overline{Q} = 0$

***Set* the output**







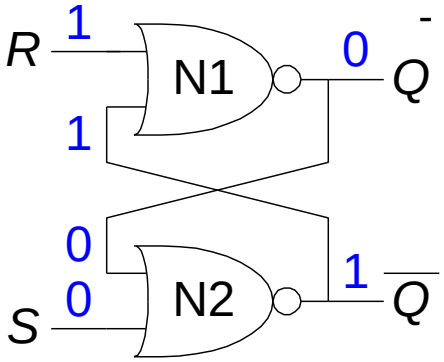
$Y = \overline{A + B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

– $S = 0, R = 1$:

then $Q = 0$ and $\overline{Q} = 1$

***Reset* the output**



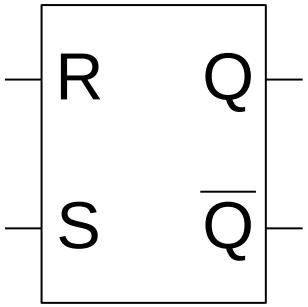
SR Latch Symbol

- 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

    ($S = 1$, $R = 0$, $Q = \mathbf{1}$)
  - **Reset:** Make the output 0

    ($S = 0$, $R = 1$, $Q = \mathbf{0}$)

SR Latch Symbol



D Latch

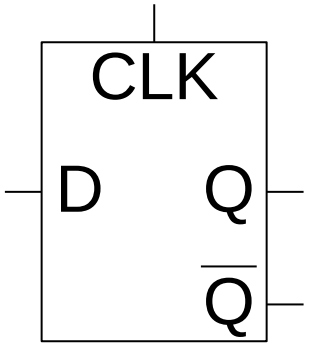
- Two inputs: *CLK*, *D*
  - ***CLK*:** controls *when* the output changes
  - ***D*** (the data input): controls *what* the output changes to
- Function
  - When ***CLK* = 1**,

    D passes through to *Q* (*transparent*)
  - When ***CLK* = 0**,

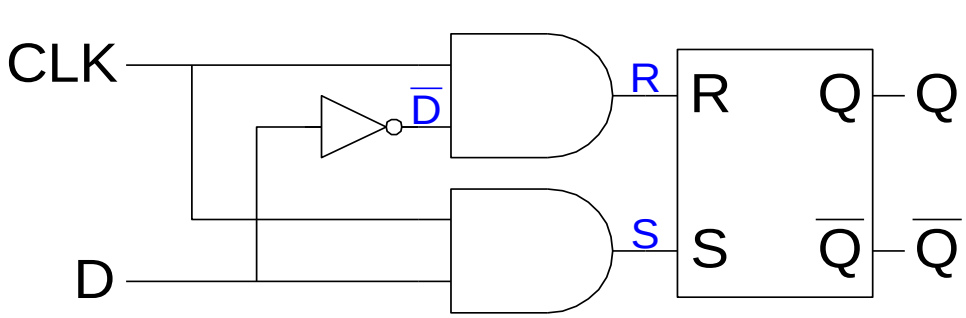
    Q holds its previous value (*opaque*)
- Avoids invalid case when

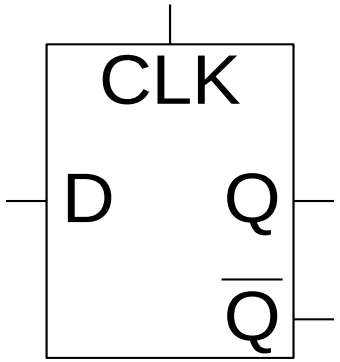
  $Q \neq \text{NOT } \overline{Q}$

D Latch Symbol



D Latch Internal Circuit



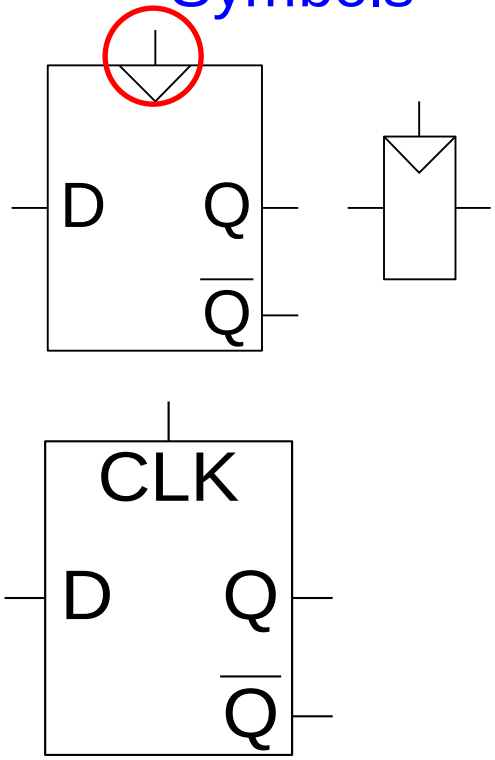


CLK	*D*	$\overline{D}$	*S*	*R*	*Q*	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

D Flip-Flop

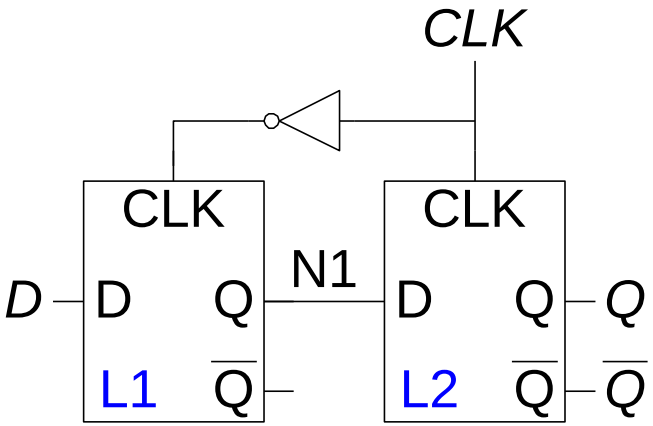
- **Inputs:** *CLK*, *D*
- **Function**
  - Samples *D* **on rising edge of *CLK***
    - When *CLK* rises from 0 to 1, *D* passes through to *Q*
    - Otherwise, *Q* holds its previous value
  - *Q* changes only on rising edge of *CLK*
- Called *edge-triggered*
- Activated on the clock edge

D Flip-Flop Symbols

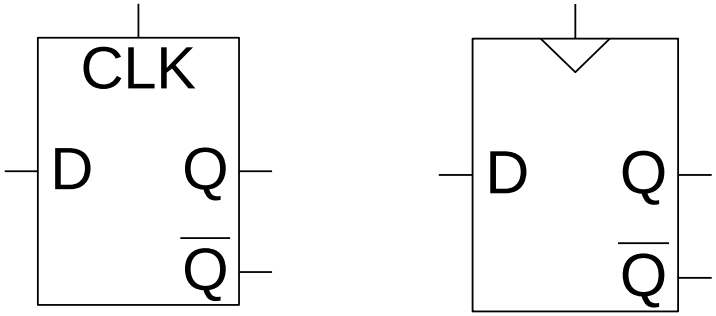


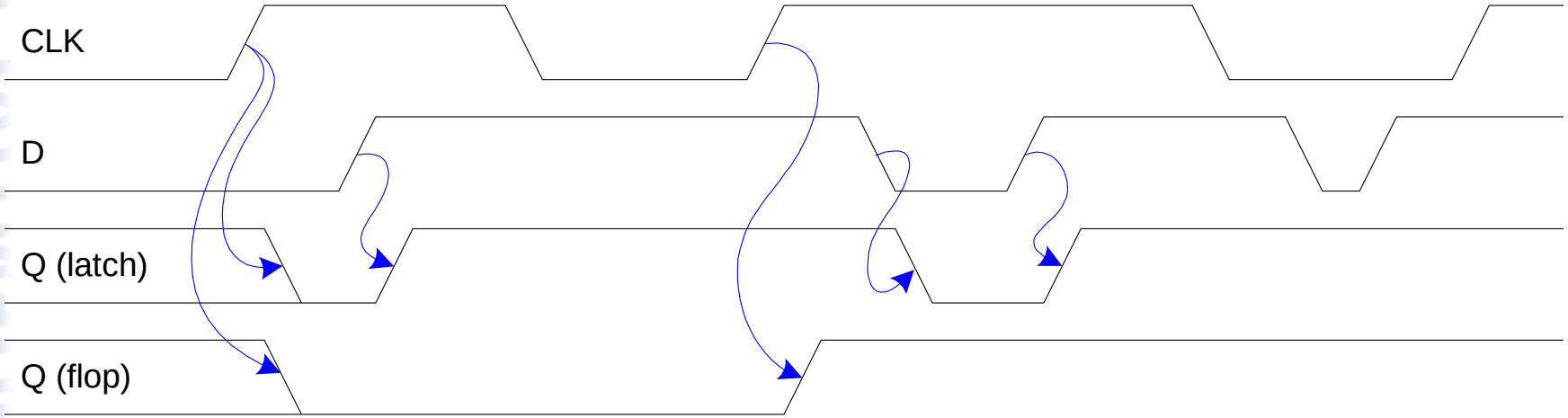
D Flip-Flop Internal Circuit

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

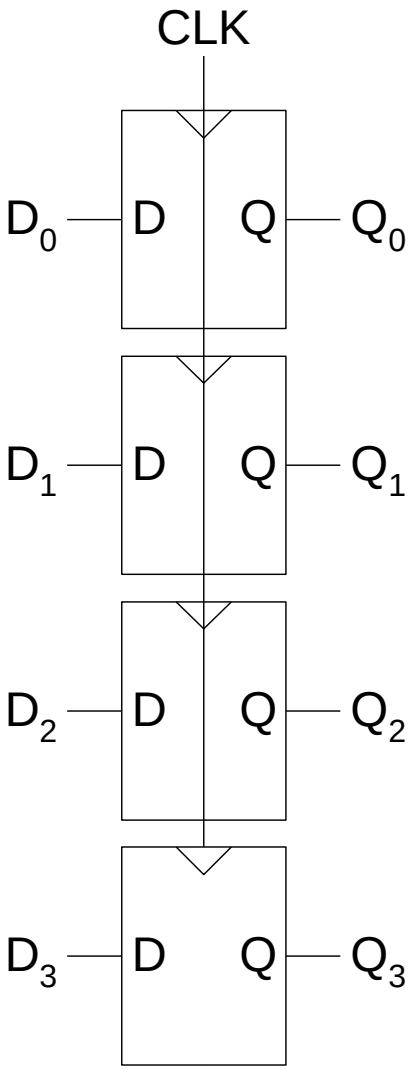


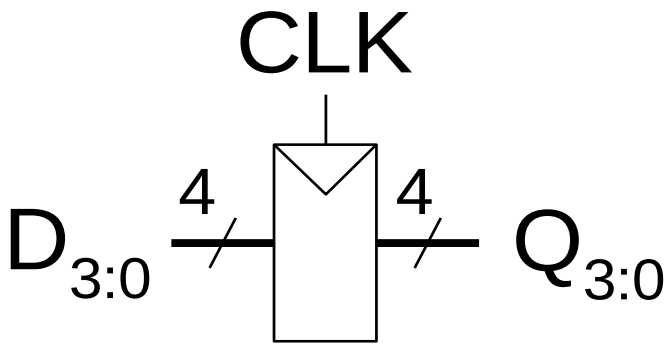
D Latch vs. D Flip-Flop





Registers

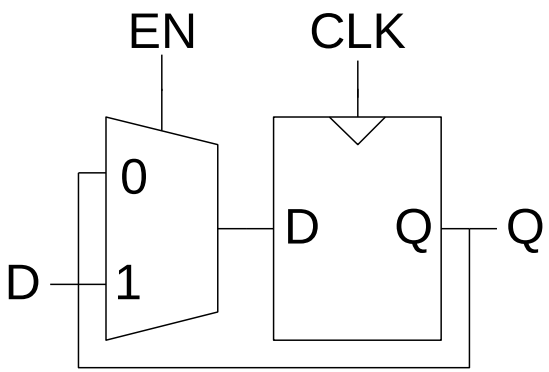




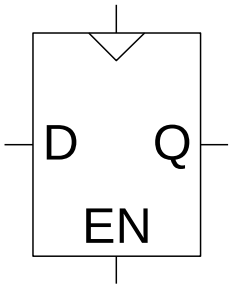
Enabled Flip-Flops

- **Inputs:** *CLK*, *D*, *EN*
  - The enable input (*EN*) controls when new data (*D*) is stored
- **Function**
  - ***EN* = 1:** *D* passes through to *Q* on the clock edge
  - ***EN* = 0:** the flip-flop retains its previous state

Internal Circuit



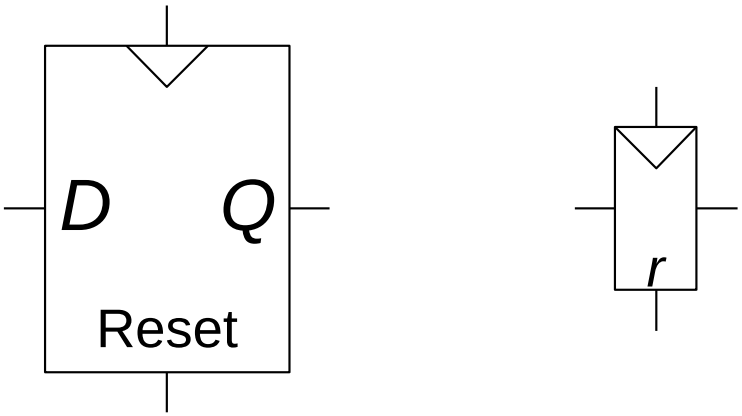
Symbol



Resettable Flip-Flops

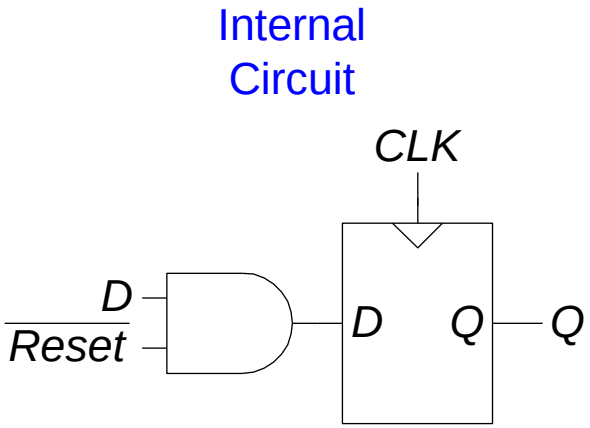
- **Inputs:** *CLK*, *D*, *Reset*
- **Function:**
  - ***Reset* = 1:** *Q* is forced to 0
  - ***Reset* = 0:** flip-flop behaves as ordinary D flip-flop

Symbols



Resettable Flip-Flops

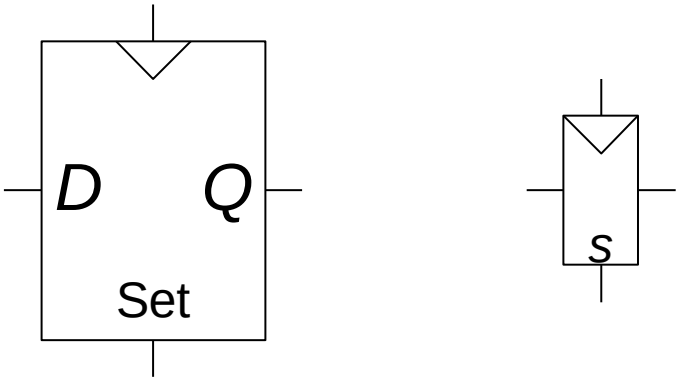
- Two types:
  - **Synchronous:** resets at the clock edge only
  - **Asynchronous:** resets immediately when *Reset* = 1
- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop
- Synchronously resettable flip-flop?



Settable Flip-Flops

- **Inputs:** *CLK*, *D*, *Set*
- **Function:**
  - ***Set* = 1:** *Q* is set to 1
  - ***Set* = 0:** the flip-flop behaves as ordinary D flip-flop

Symbols

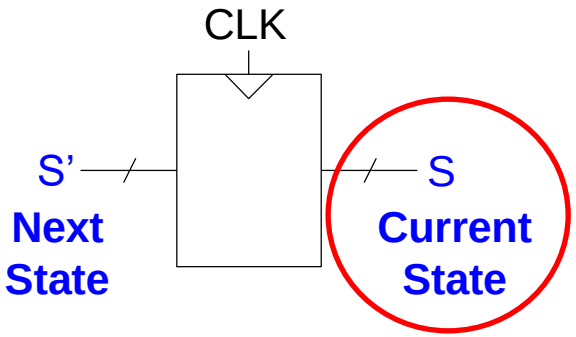


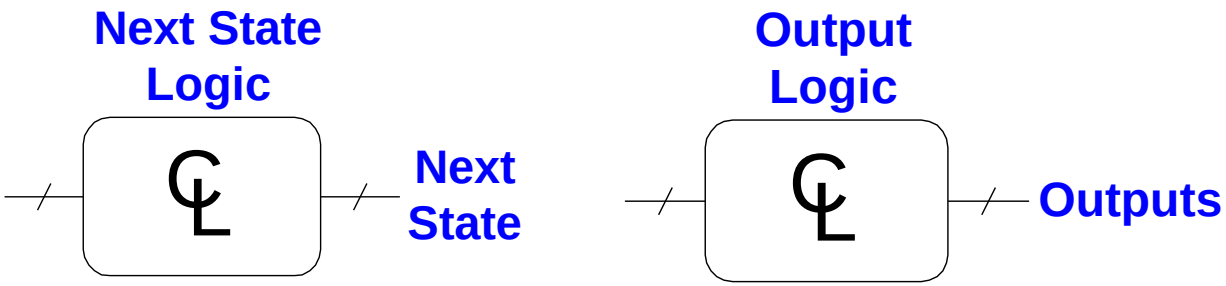
Synchronous Sequential Logic Design

- Breaks cyclic paths by **inserting registers**
- Registers contain **state** of the system
- State changes at clock edge: system **synchronized** to the clock
- **Rules** of synchronous sequential circuit composition:
  - Every circuit element is either a register or a combinational circuit
  - At least one circuit element is a register
  - All registers receive the same clock signal
  - Every cyclic path contains at least one register
- Two common synchronous sequential circuits
  - **Finite State Machines (FSMs)**
  - **Pipelines**

Finite State Machine (FSM)

- Consists of:
  - **State register**
    - Stores current state
    - Loads next state at clock edge
  - **Combinational logic**
    - Computes the next state
    - Computes the outputs

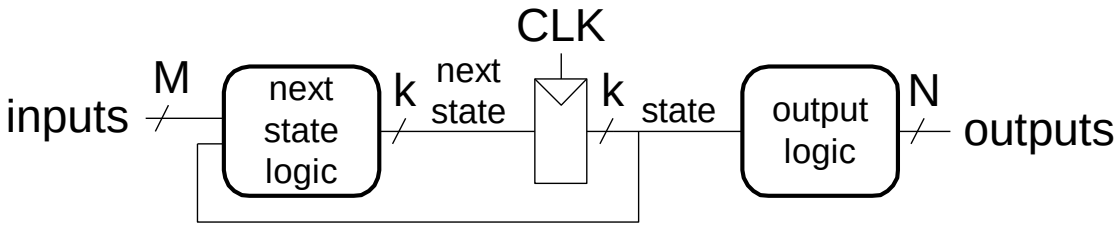




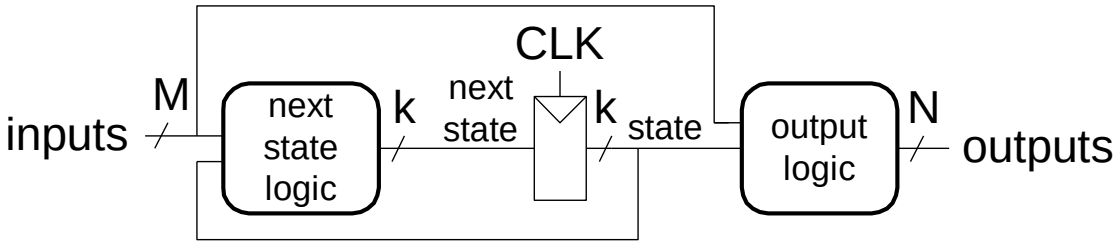
Finite State Machines (FSMs)

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

Moore FSM

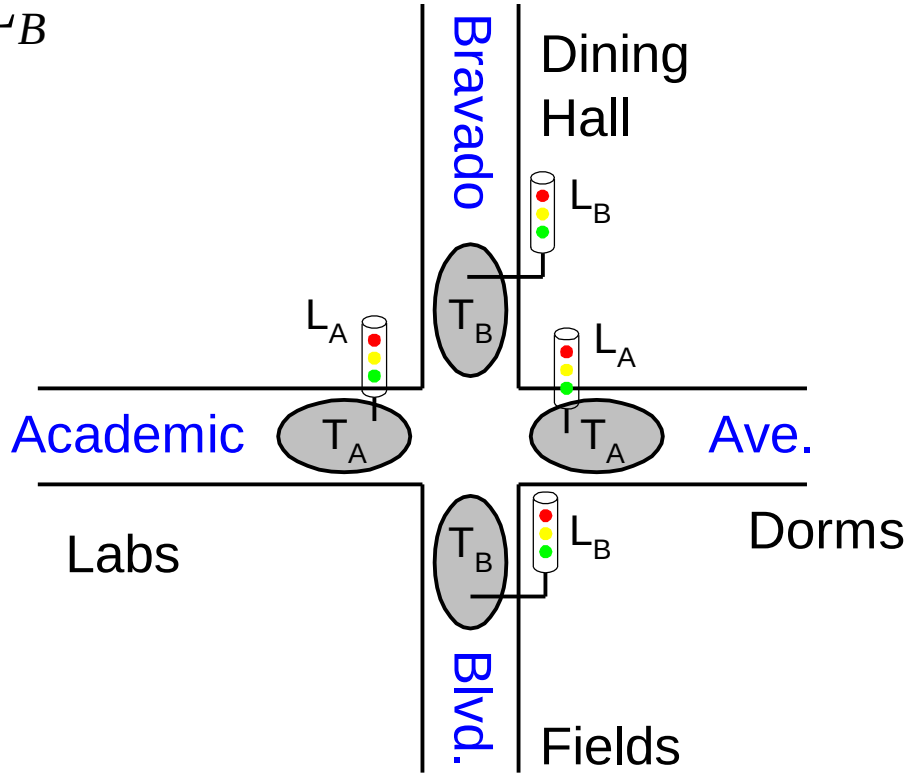


Mealy FSM



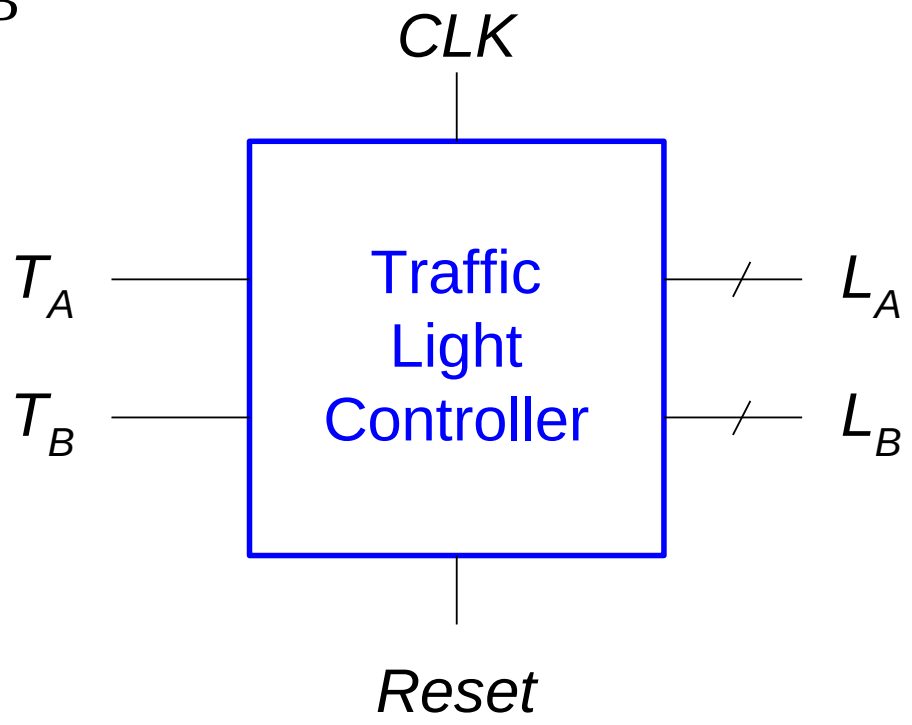
FSM Example

- Traffic light controller
  - Traffic sensors: T_A, T_B (TRUE when there's traffic)
  - Lights: L_A, L_B



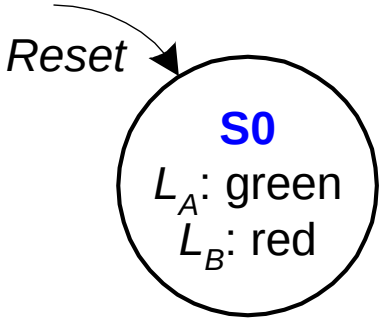
FSM Black Box

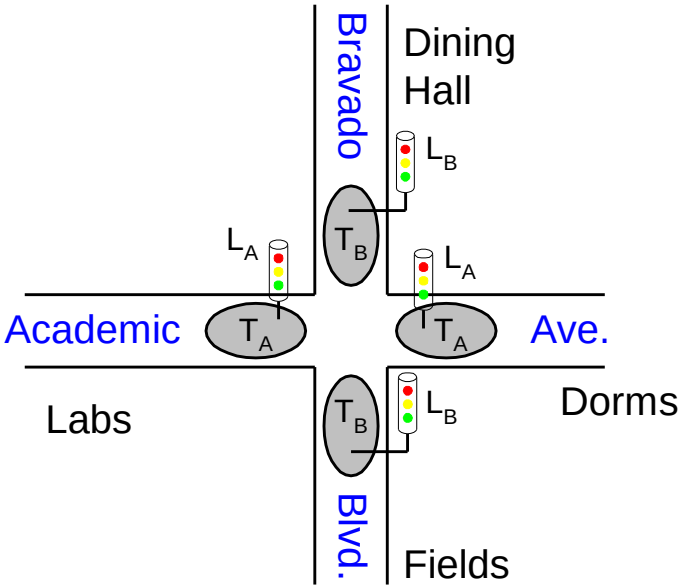
- Inputs: *CLK*, *Reset*, T_A, T_B
- Outputs: L_A, L_B



FSM State Transition Diagram

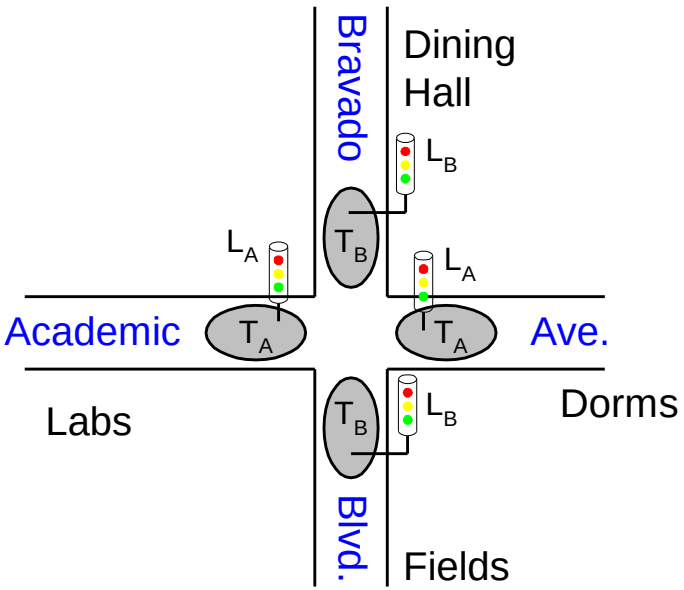
- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs

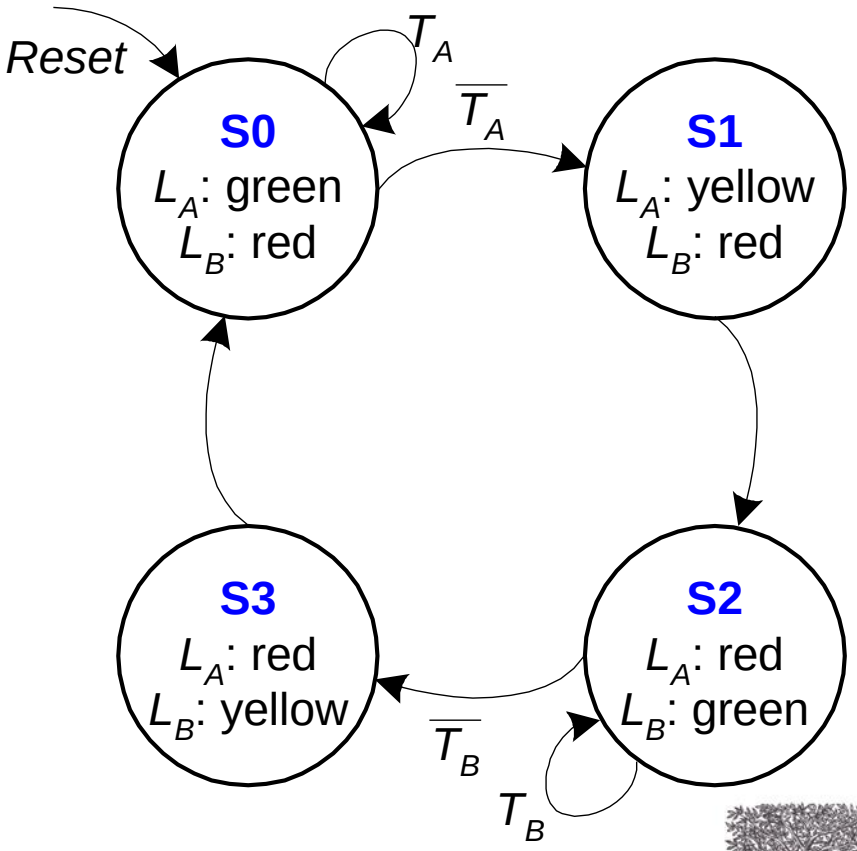




FSM State Transition Diagram

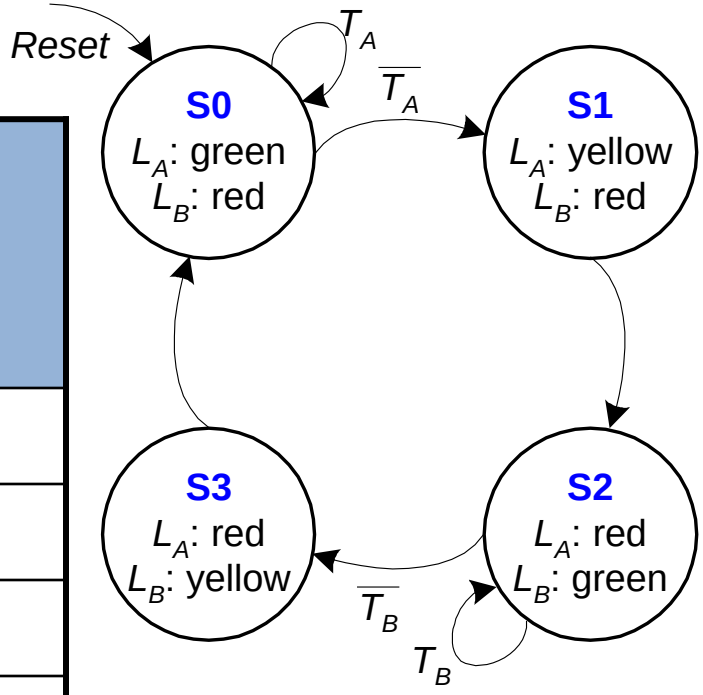
- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs





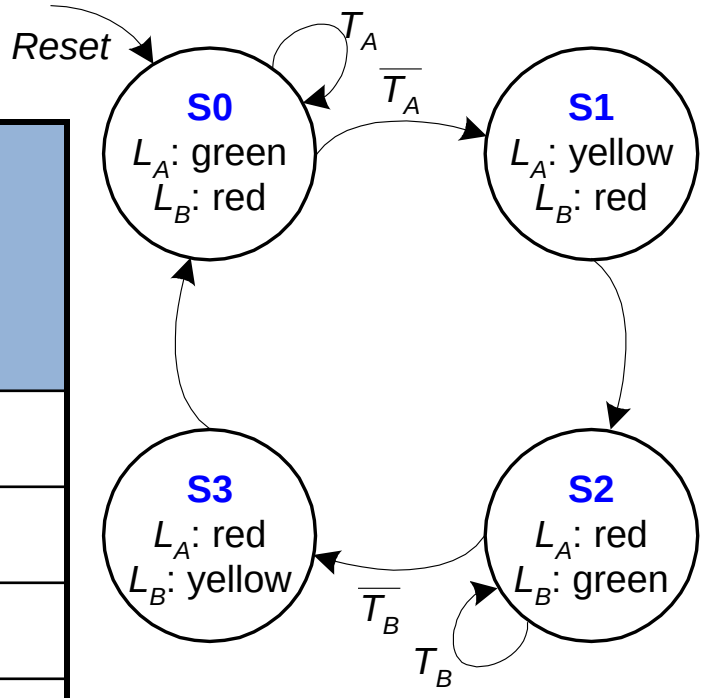
FSM State Transition Table

Current State S	Inputs T_A	Inputs T_B	Next State 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 S	Inputs T_A	Inputs T_B	Next State 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

Current State S	Inputs T_A	Inputs T_B	Next State 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

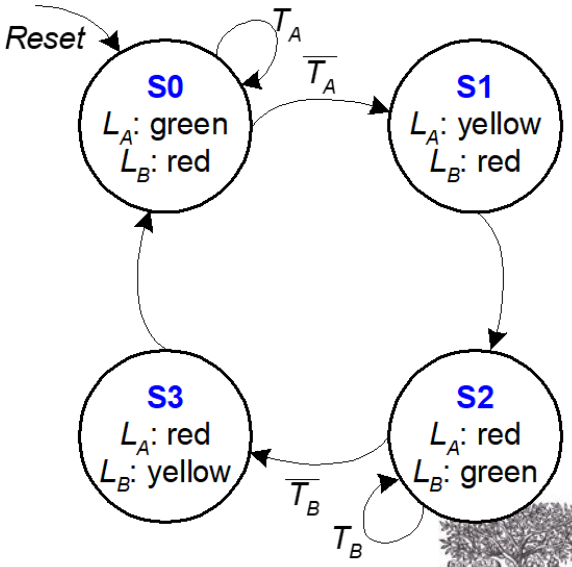
$$S'_1 = S_1 \oplus S_0$$

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

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

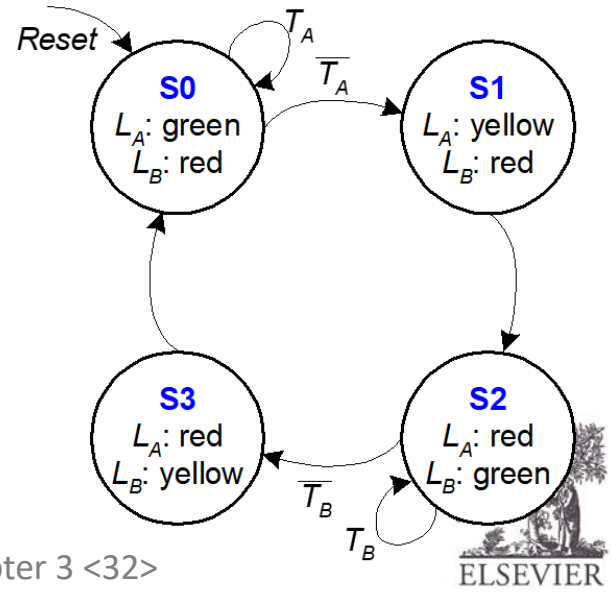


FSM Output Table

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

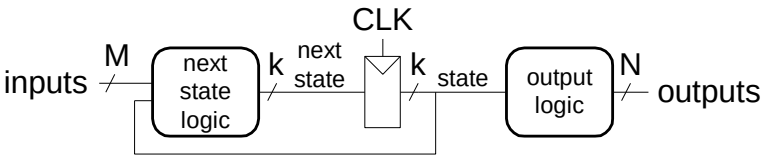
Output	Encoding
green	00
yellow	01
red	10

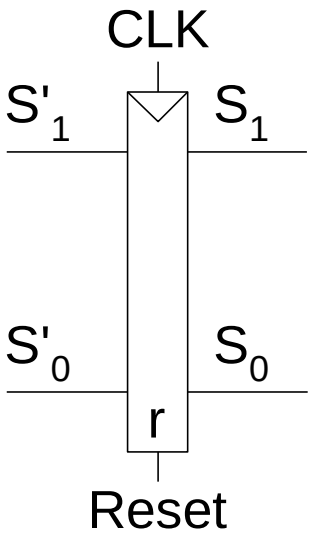
$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1} S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1 S_0$$



FSM Schematic: State Register

Moore FSM



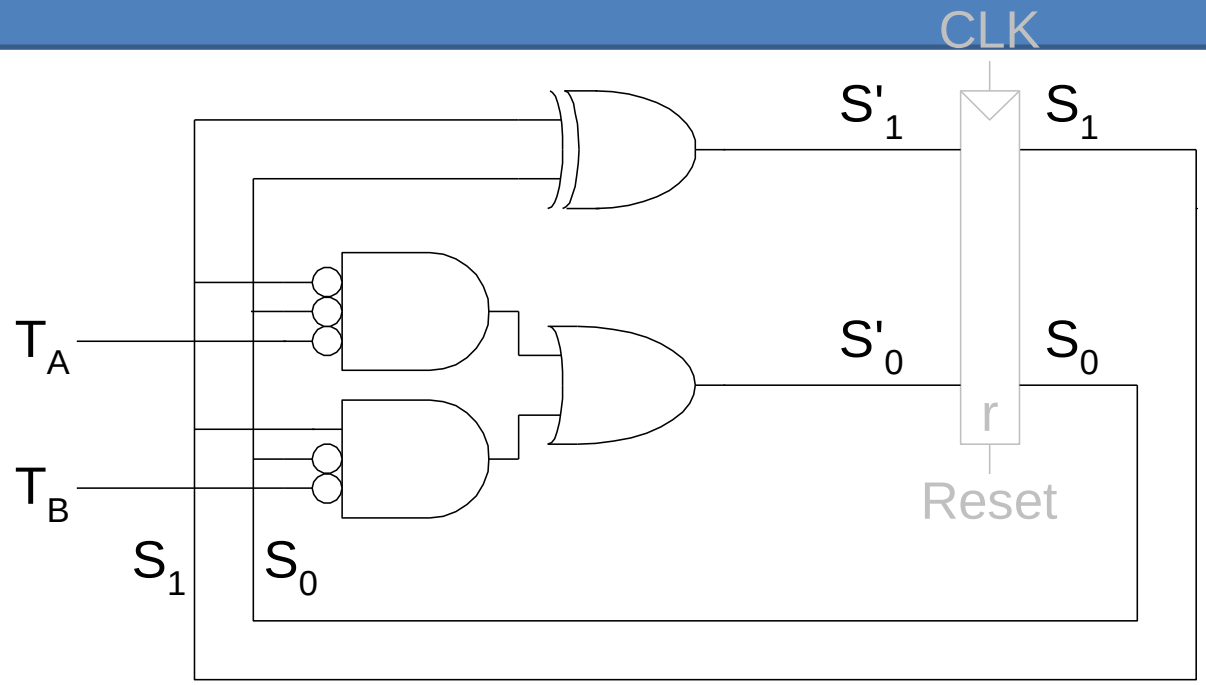


state register

$$S'_1 = S_1 \oplus S_0$$
$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1}S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1S_0$$

FSM Schematic: Next State Logic

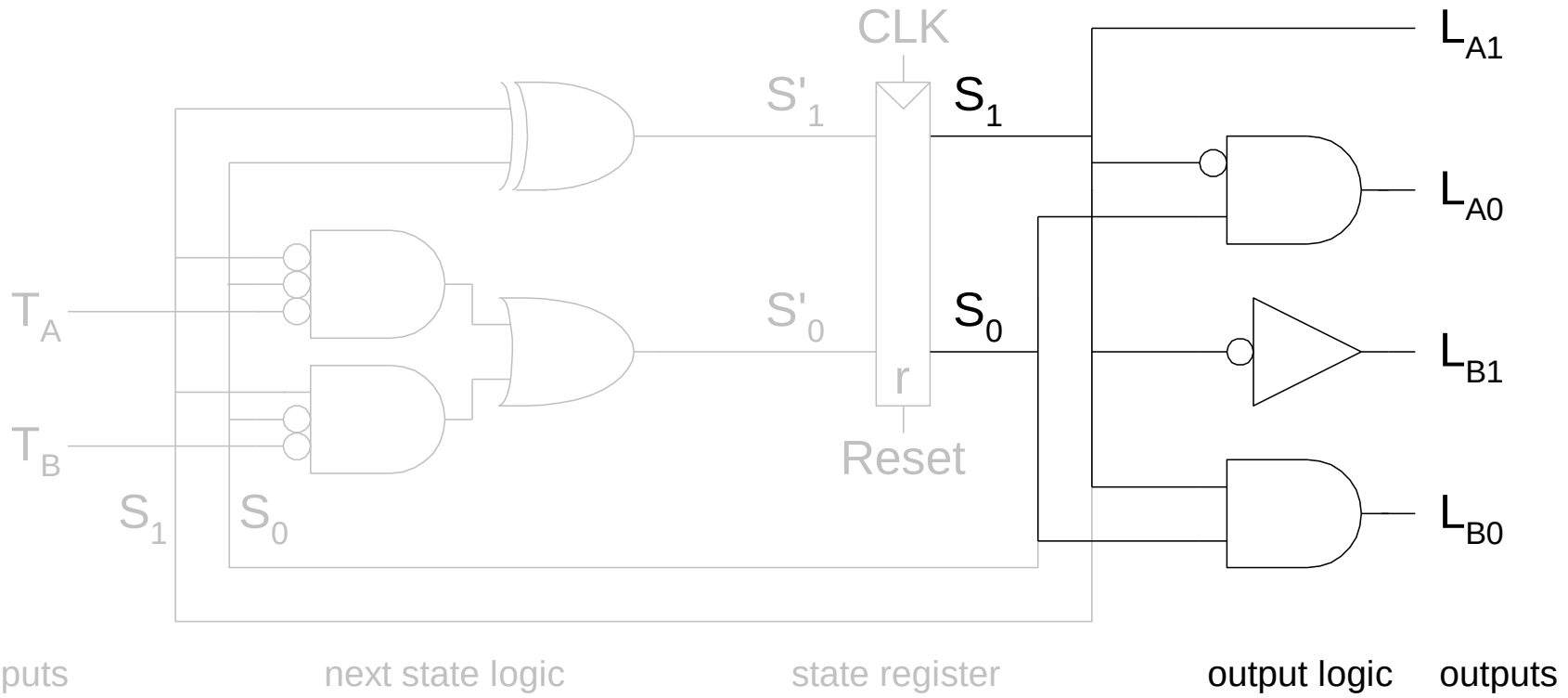


inputs | next state logic | state register

$$S'_1 = S_1 \oplus S_0$$
$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

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

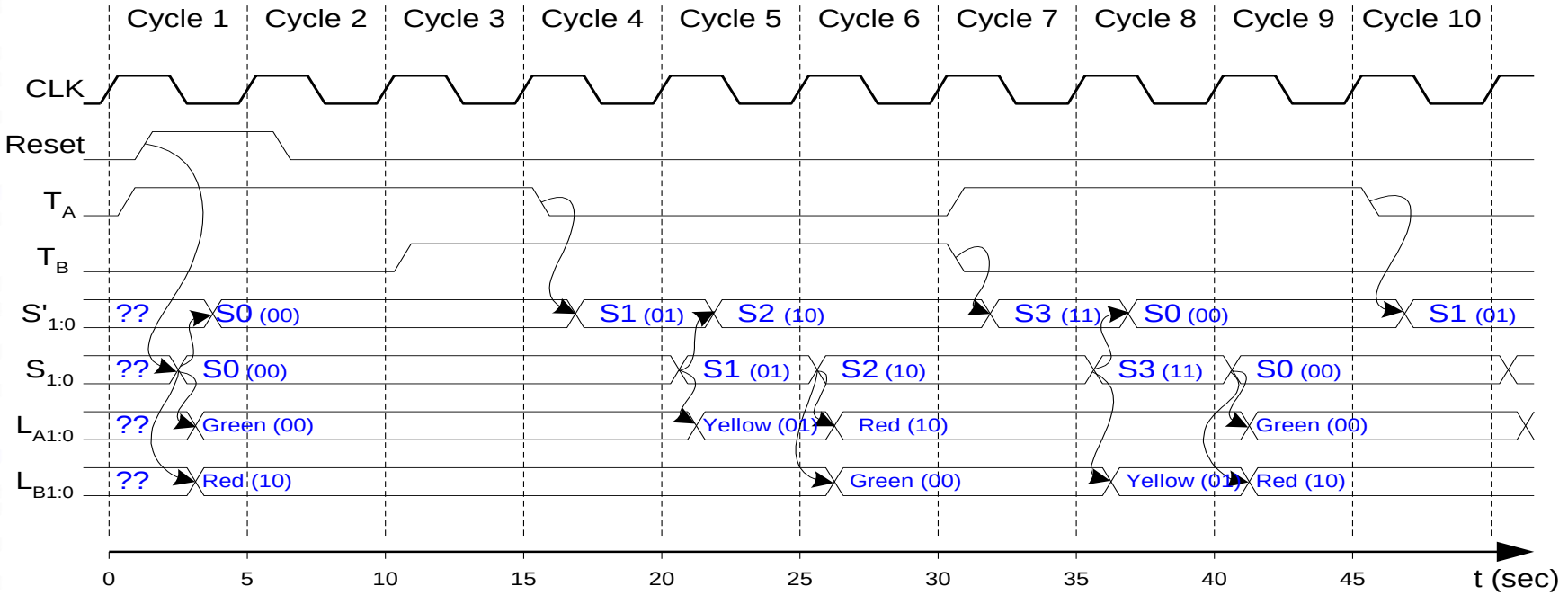
FSM Schematic: Output Logic

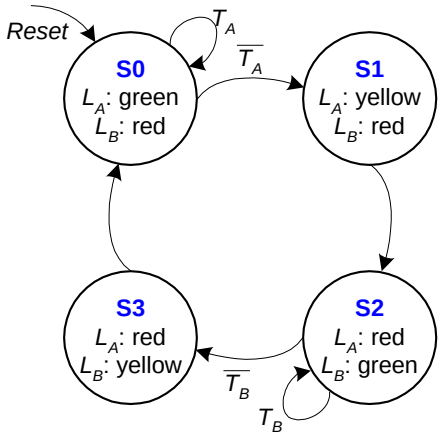


$$S'_1 = S_1 \oplus S_0$$
$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1}S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1S_0$$

FSM Timing Diagram



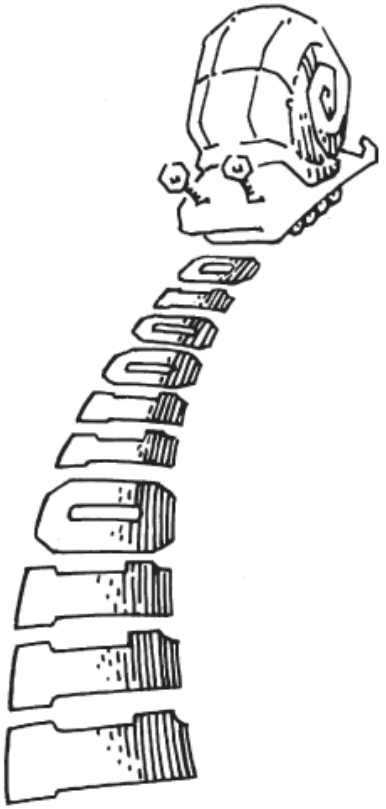


FSM State Encoding

- **Binary** encoding:
  - i.e., for four states, 00, 01, 10, 11
- **One-hot** encoding
  - One state bit per state
  - Only one state bit HIGH at once
  - i.e., for 4 states, 0001, 0010, 0100, 1000
  - Requires more flip-flops
  - Often next state and output logic is simpler

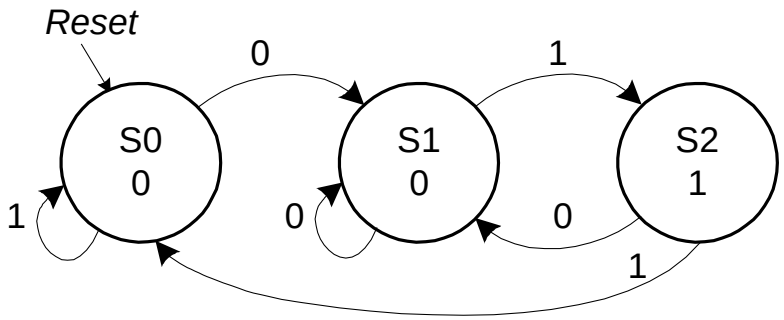
Moore vs. Mealy FSM

- Layla 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.



State Transition Diagrams

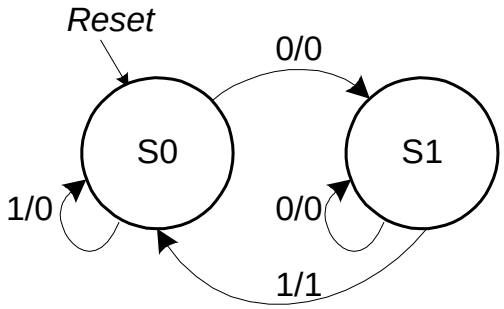
Moore FSM



outputs depend only on the current state

smiles whenever the last two digits it has crawled over are 01.

Mealy FSM



outputs depend on current state *and* inputs

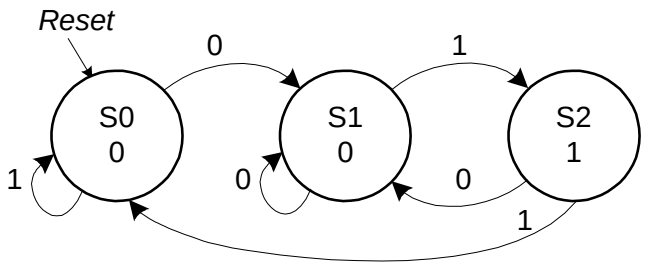
Mealy FSM: arcs indicate input/output

Moore FSM State Transition Table

Current State		Inputs	Next State	
S_1	S_0	A	S'_1	S'_0
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		

State	Encoding
S0	00
S1	01
S2	10

Moore FSM

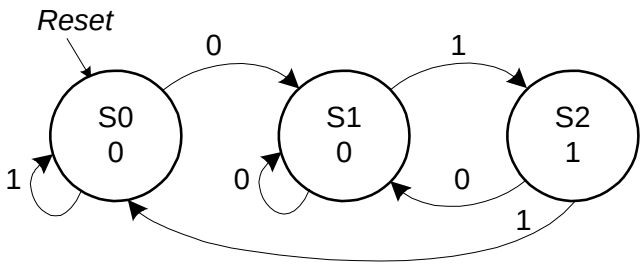


Moore FSM State Transition Table

Current State		Inputs	Next State	
S_1	S_0	A	S'_1	S'_0
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

State	Encoding
S0	00
S1	01
S2	10

Moore FSM



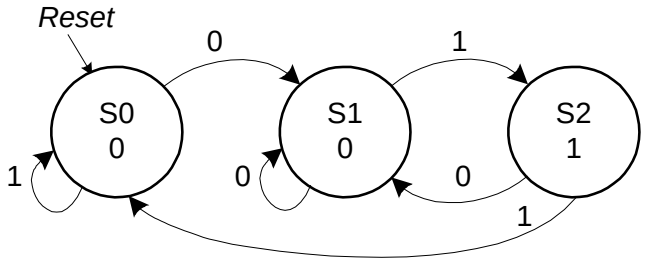
$$S_1' = S_0 A$$
$$S_0' = \overline{A}$$

Moore FSM Output Table

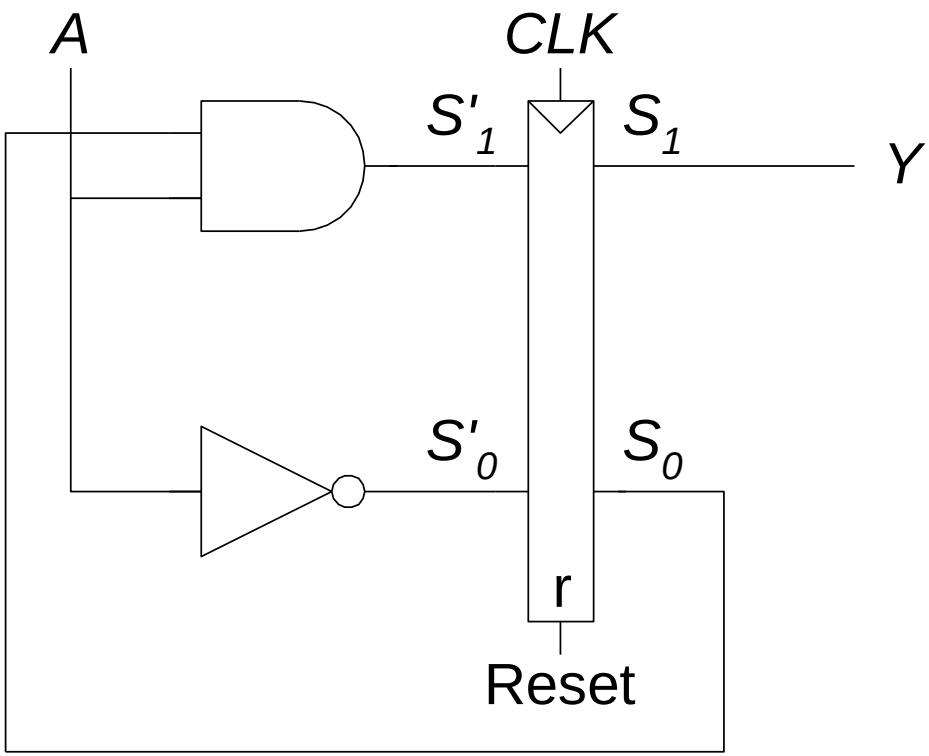
Current State S_1	Current State S_0	Output Y
0	0	0
0	1	0
1	0	1

$$Y = S_1$$

Moore FSM



Moore FSM Schematic



Current State		Inputs	Next State	
S_1	S_0	A	S'_1	S'_0
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

Current State		Output
S_1	S_0	Y
0	0	0
0	1	0
1	0	1

$$S_1' = S_0 A$$
$$S_0' = \overline{A}$$

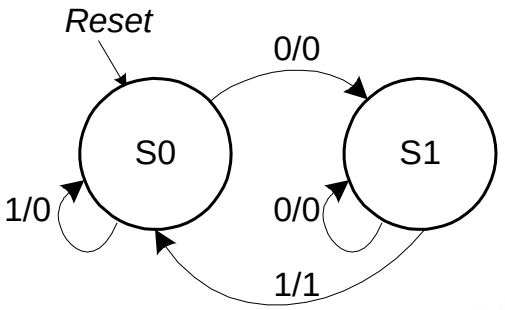
$$Y = S_1$$

Mealy FSM State Transition & Output Table

Current State	Input	Next State	Output
S_0	A	S'_0	Y
0	0		
0	1		
1	0		
1	1		

State	Encoding
S0	0
S1	1

Mealy FSM



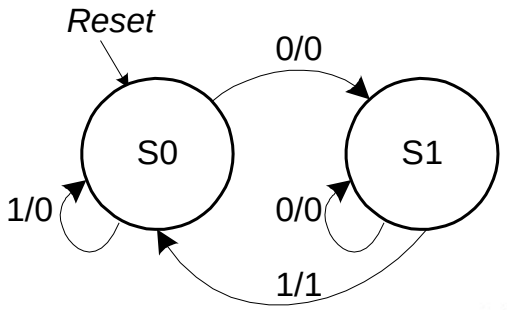
Mealy FSM State Transition & Output Table

Current State	Input	Next State	Output
S_0	A	S'_0	Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

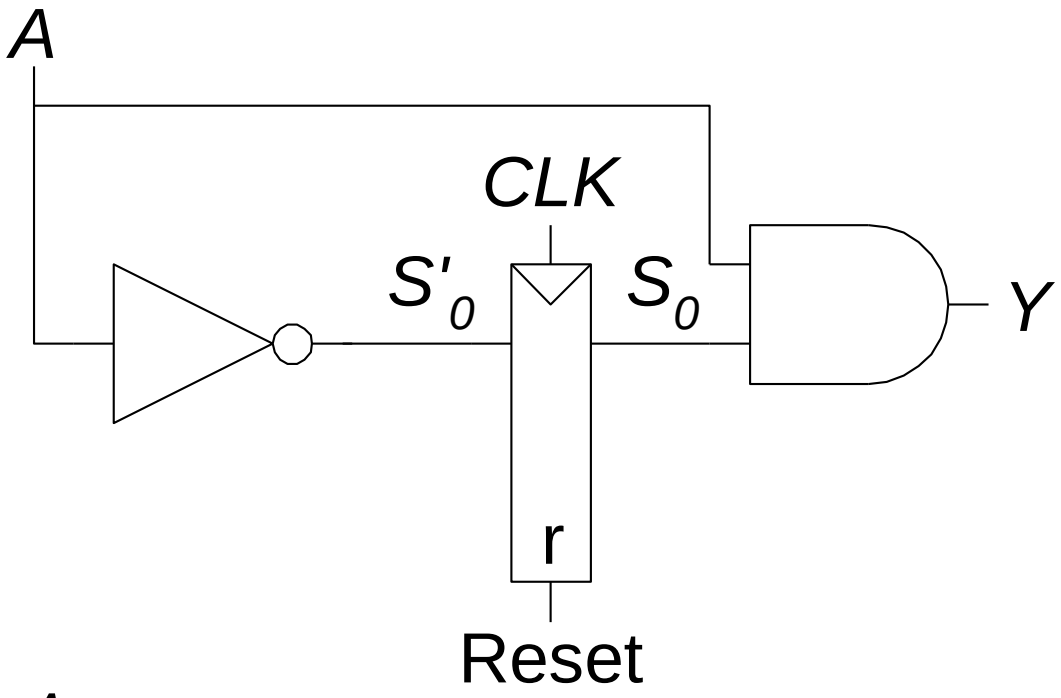
State	Encoding
S0	0
S1	1

$$Y = S_0 A$$
$$S_0' = \overline{A}$$

Mealy FSM



Mealy FSM Schematic

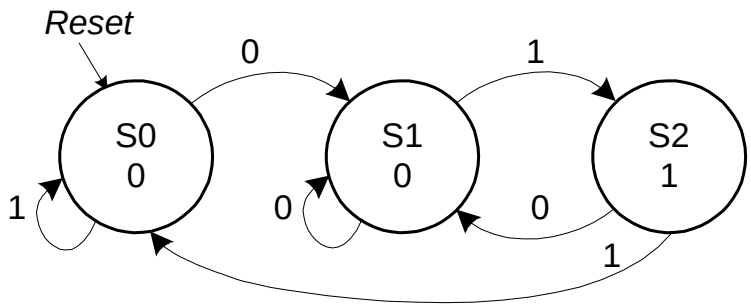


$$Y = S_0 A$$

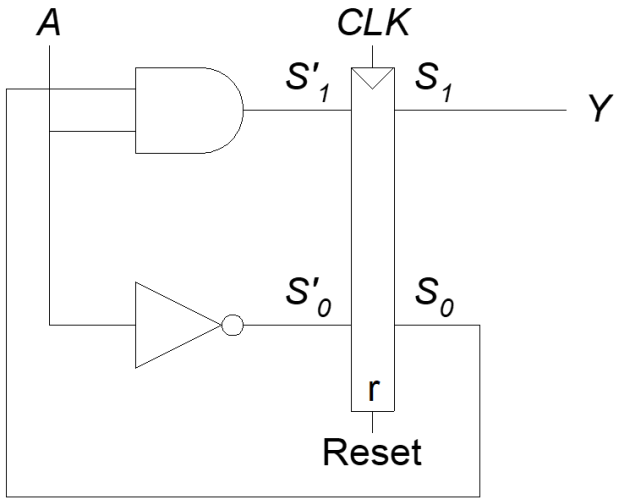
$$S_0' = \overline{A}$$

Moore Vs. Mealy FSM Schematic

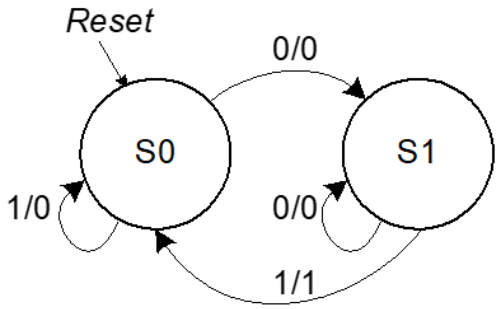
Moore FSM



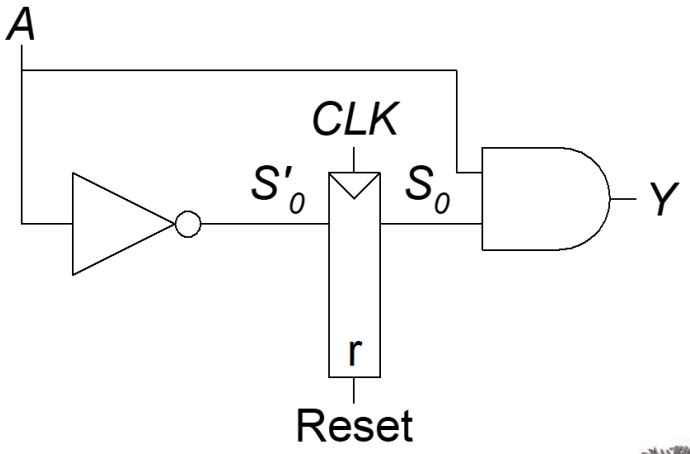
outputs depend only on the current state



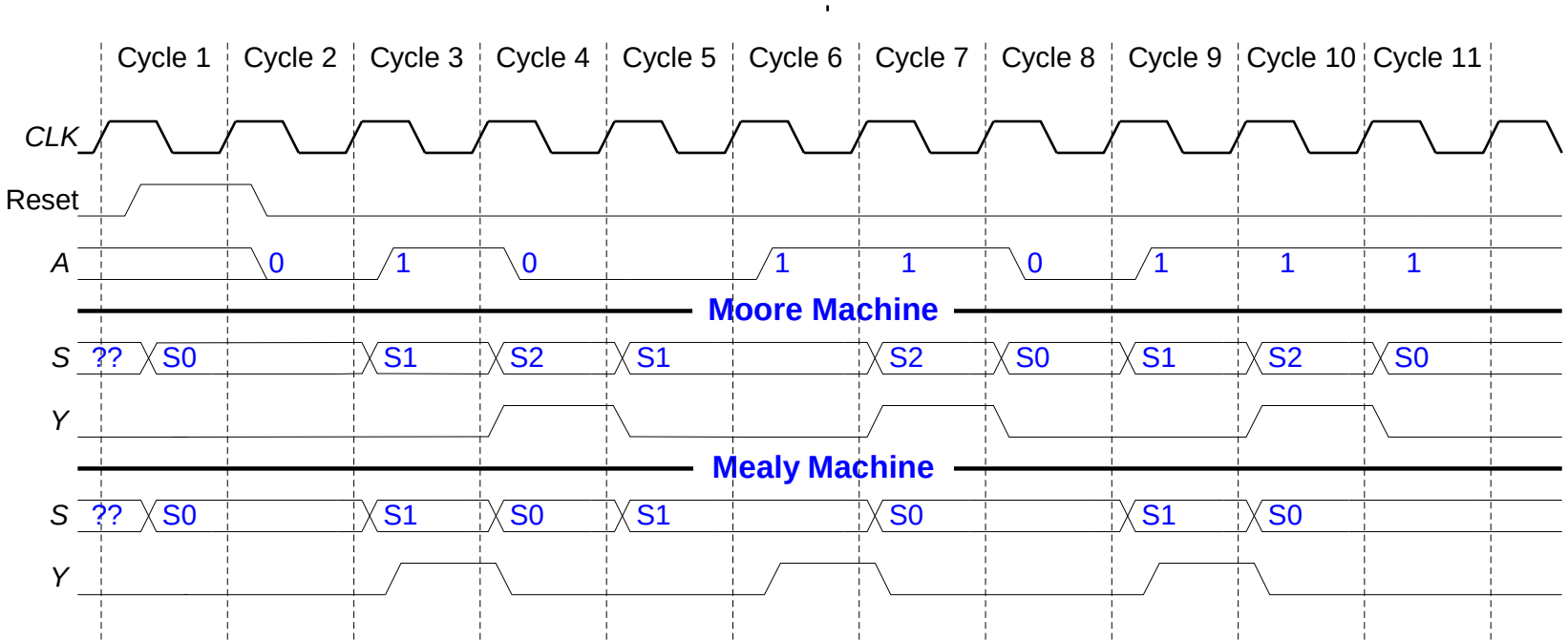
Mealy FSM



outputs depend on current state *and* inputs



Moore & Mealy Timing Diagram



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

Parallelism

- **Two types of parallelism:**
  - **Spatial parallelism**
    - duplicate hardware performs multiple tasks at once
  - **Temporal parallelism**
    - task is broken into multiple stages
    - also called pipelining
    - for example, an assembly line

Parallelism Definitions

- **Token:** Group of inputs processed to produce group of outputs
- **Latency:** Time for one token to pass from start to end
- **Throughput:** Number of tokens produced per unit time

Parallelism increases throughput

Parallelism Example

- Ben bakes cookies to celebrate traffic light controller installation
- 5 minutes to roll cookies
- 15 minutes to bake
- What is the latency and throughput without parallelism?

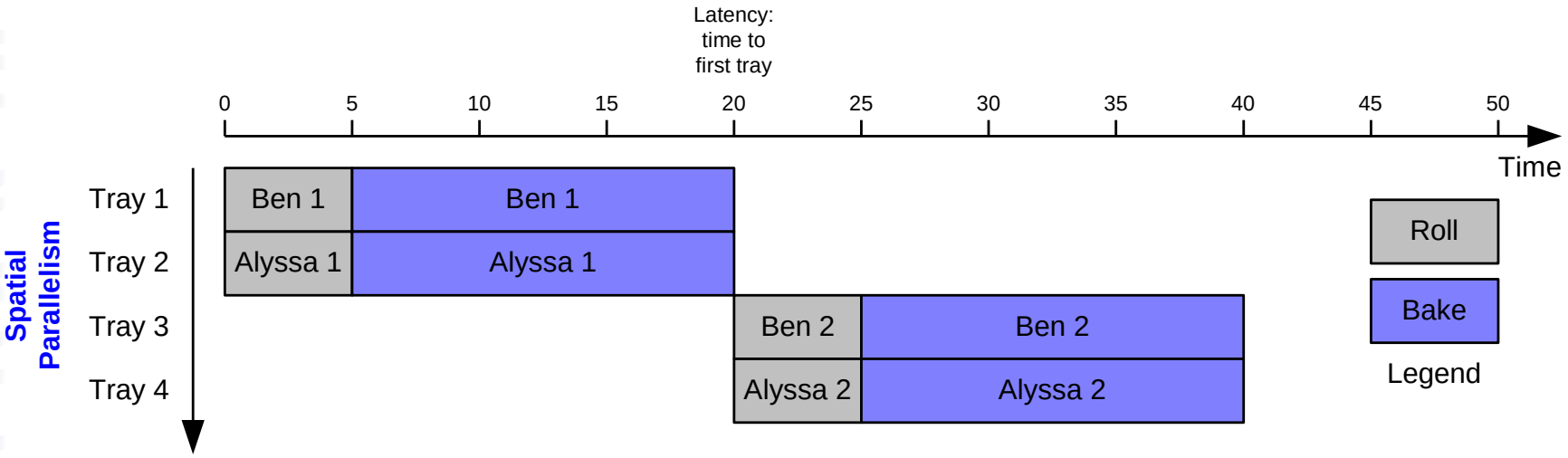
Latency = 5 + 15 = 20 minutes = **1/3 hour**

Throughput = 1 tray/ 1/3 hour = **3 trays/hour**

Parallelism Example

- What is the latency and throughput if Ben uses parallelism?
  - **Spatial parallelism:** Ben asks Allysa to help, using her own oven
  - **Temporal parallelism:**
    - two stages: rolling and baking
    - He uses two trays
    - While first batch is baking, he rolls the second batch, etc.

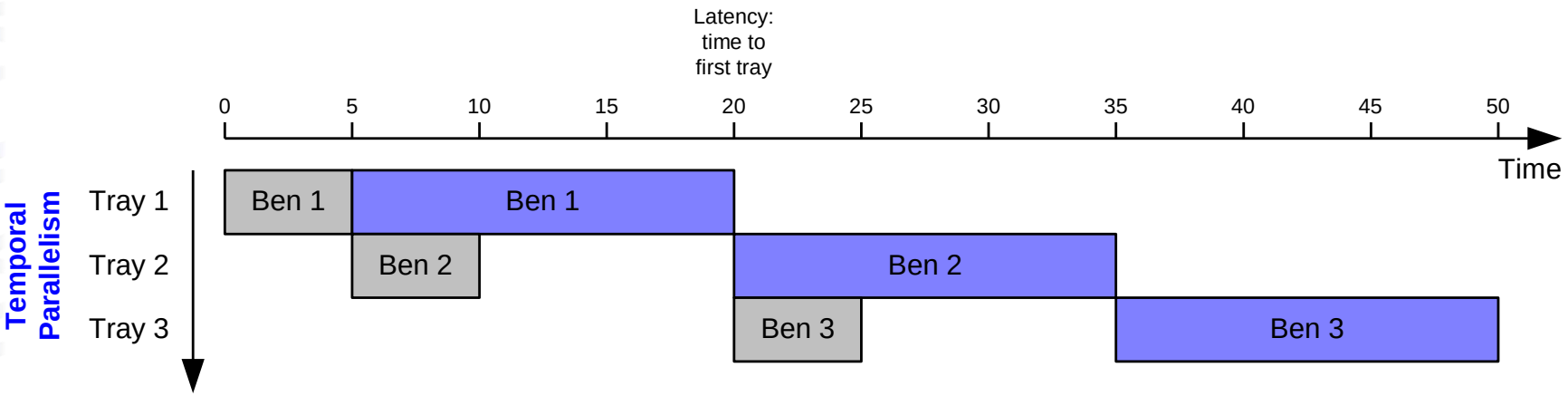
Spatial Parallelism



Latency = 5 + 15 = 20 minutes = **1/3 hour**

Throughput = 2 trays/ 1/3 hour = **6 trays/hour**

Temporal Parallelism



Latency = 5 + 15 = 20 minutes = **1/3 hour**

Throughput = 1 trays/ 1/4 hour = **4 trays/hour**

Using both techniques, the throughput would be **8 trays/hour**