# Sequential Logic Design

Digital Design and Computer Architecture
Mohammad Sadrosadati
Frank K. Gürkaynak

http://safari.ethz.ch/ddca

# What will we learn?

- **Combinational and Sequential Circuits**
- **How can a circuit remember a value**
- **Different types of memorizing elements**
- **Finite State Machines**

# Introduction

- **Outputs of combinational circuit depend ONLY on current input values.**
- **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 == Combinational + State

- **Largest part of a sequential circuit is combinational**
  - The only additional thing we need to learn is to store the state
  - Defining flip-flops (latches), registers should do the trick
- **Sequential circuits divide the operation into time slots**
  - At every time slot inputs (if there are any) are taken
  - Present state and inputs are used to calculate the next state
  - The next state is saved in the flip-flops (registers)
- **How fast we can finish the operation?**
  - The clock signal is used to move from one state to the next state
  - I.e. a 2 GHz clock has time steps of 500ps.
  - The work within a time slot is done by a combinational circuit.

# Datapath vs Finite State Machine

# RTL design

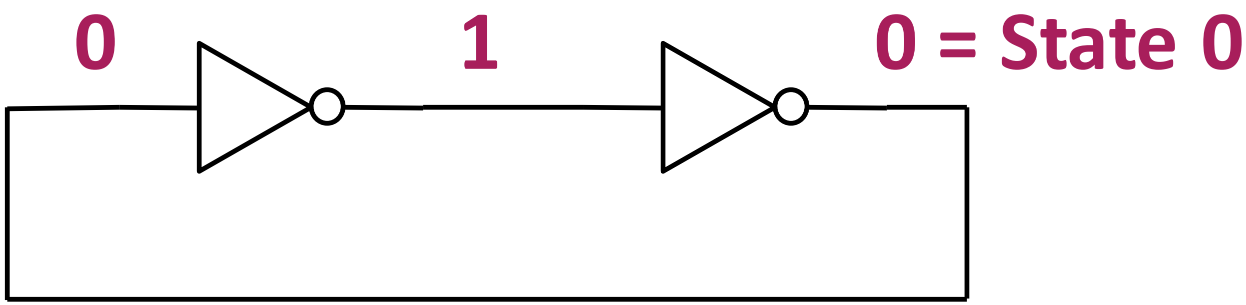
- **RTL is a generic way of defining digital circuits**
  - State is stored in registers
  - Every time slot, inputs and present state calculates the next state
  - Next state is stored in a register.
  - The clock moves the circuit from the present state to next state
- **RTL defines datapath circuits …**
  - They process data and do the main work
- **… and Finite State machines**
  - Generate control signals for the datapath
- **The distinction makes life easier**

# State Elements

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

# How can a circuit remember?

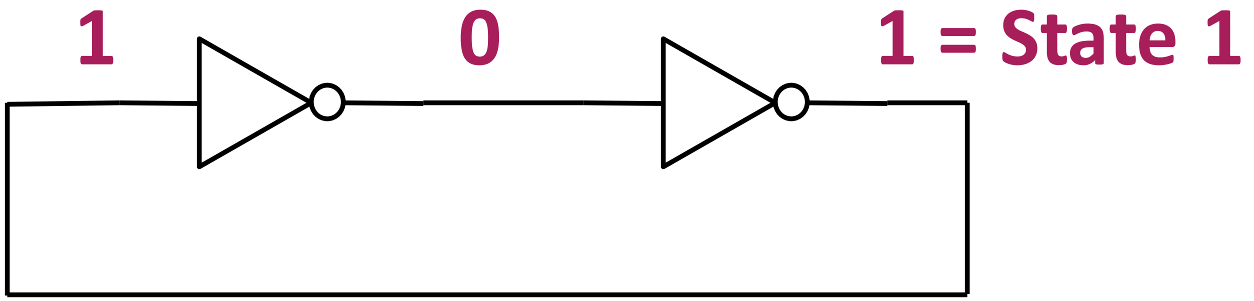
- **Bistable circuits can have two distinct states**
  - Once they are in one state, they will remain there.




- **The Loop keeps the state stable**

# How can a circuit remember?

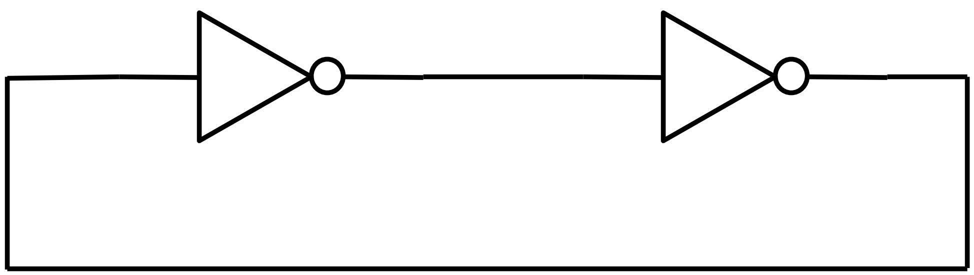
- **Bistable circuits can have two distinct states**
  - Once they are in one state, they will remain there.




- **The Loop keeps the state stable**

# How can a circuit remember?

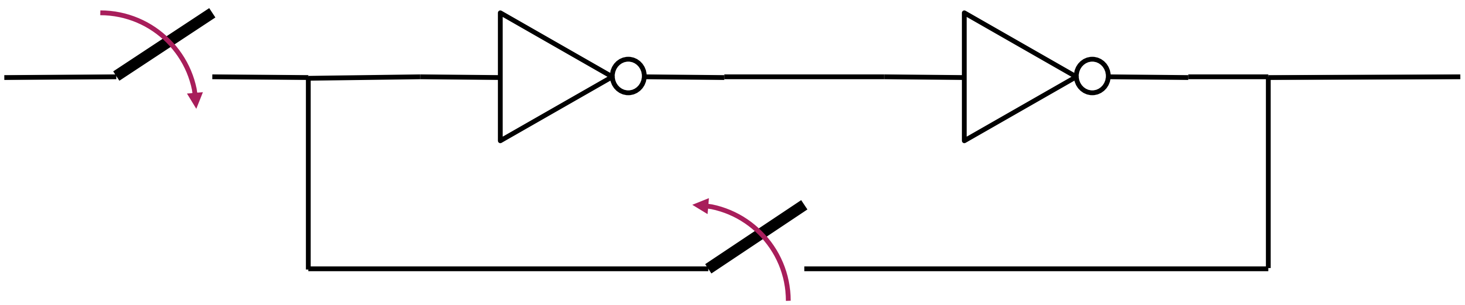
- **Bistable circuits can have two distinct states**
  - Once they are in one state, they will remain there.



- **But how can we move from one state to another?**

# How can a circuit remember?

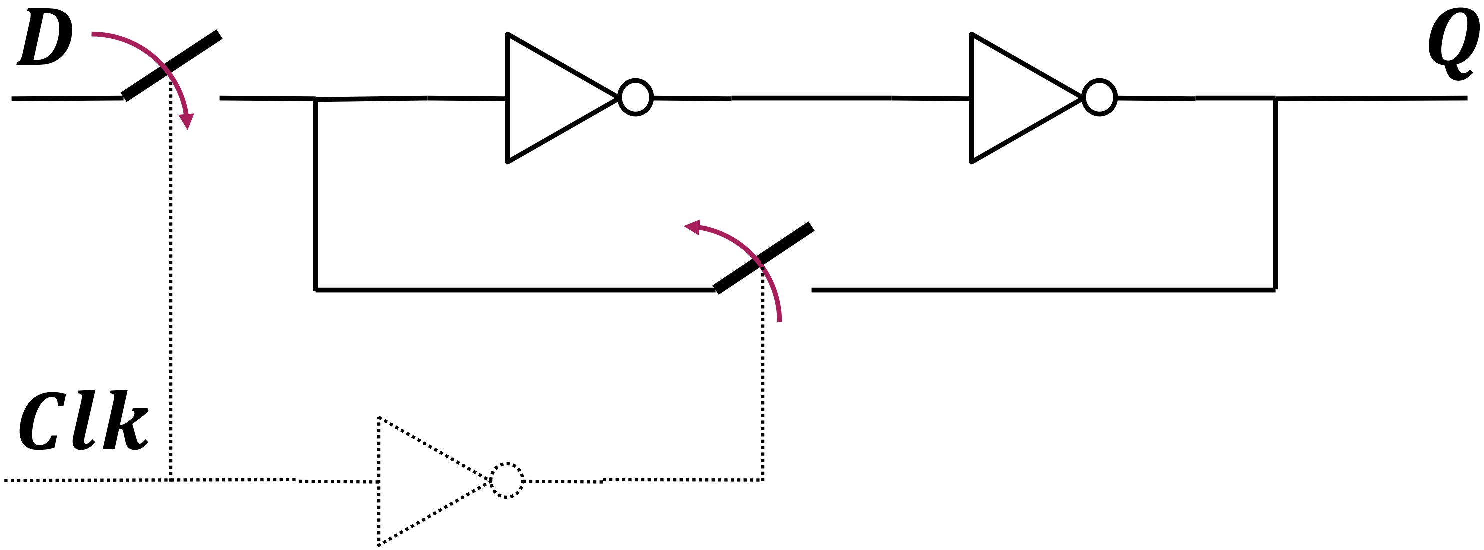
- **Bistable circuits can have two distinct states**
  - Once they are in one state, they will remain there.



- **But how can we move from one state to another?**
  - We add one switch to break the loop and at the same time add another switch that connects an input to the circuit

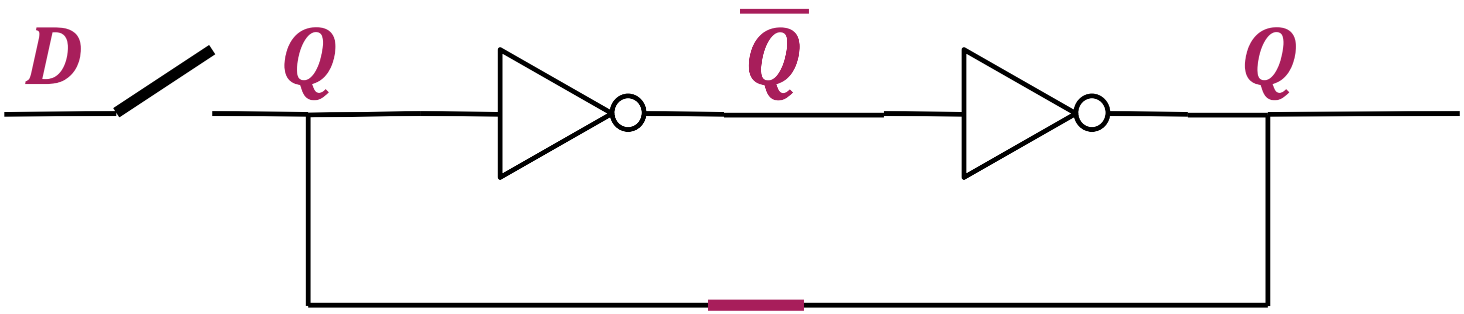
# The D Latch

- **D Latch is the basic bi-stable circuit used in modern CMOS.**
  - The clock controls the switches. **Only one is active** at a time.
  - Traditionally the input is called D (Data) and the output Q



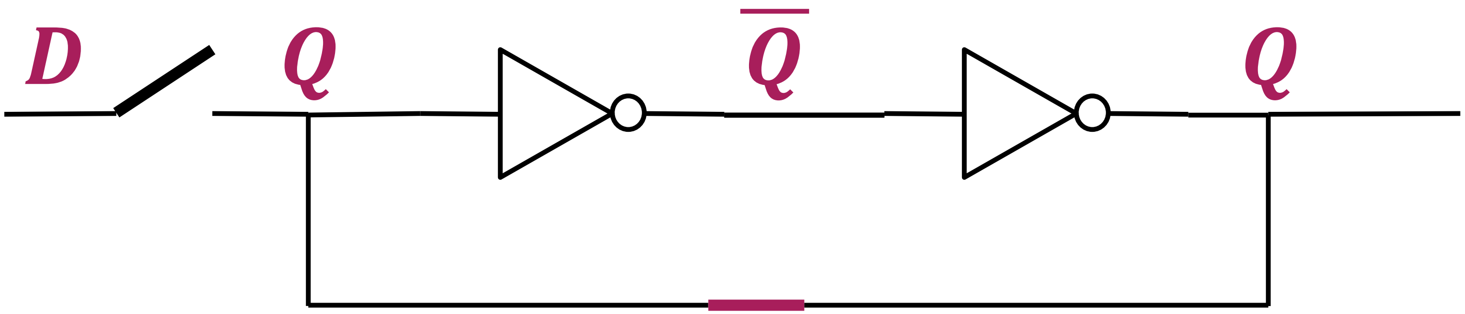
# The D Latch has two modes

- **Latch mode, loop is active, input disconnected, keeps state**

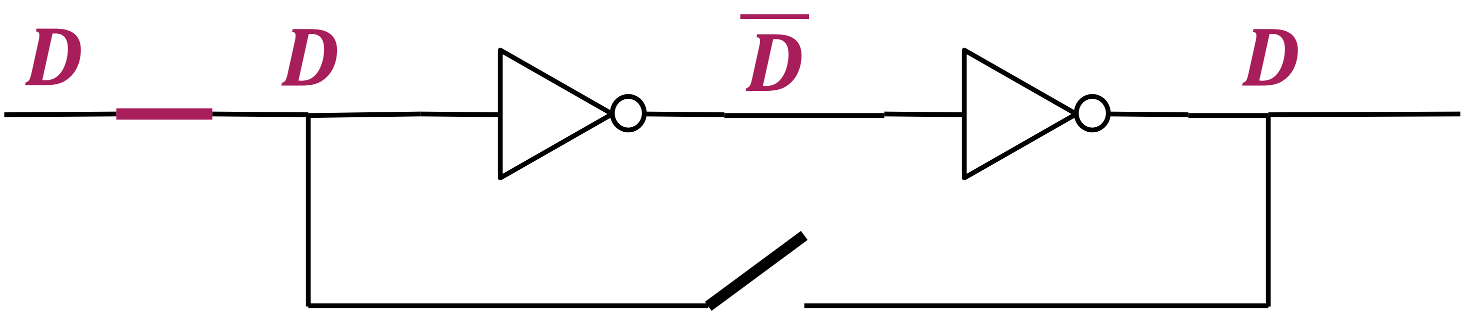


# The D Latch has two modes

- **Latch mode, loop is active, input disconnected, keeps state**




- **Transparent mode, loop is inactive, input is connected and propagates to output**



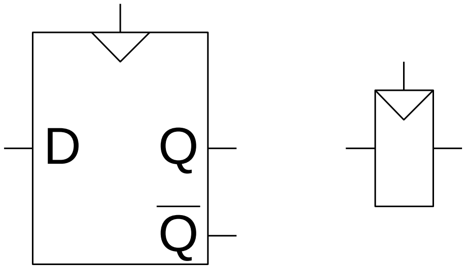
# Summary D Latch

- **Simple bi-stable circuit**
  - Can be used to store a 0 or a 1.
- **Has two modes**
  - ***Transparent mode***: input propagates to output
  - ***Latch mode***: the output is stored (also called ***opaque mode***)
- **The clock controls the modes of operation.**
  - Depending on the type, it might be latch is transparent when Clk=1 or latch is transparent when Clk=0

# Rising edge trigerred D Flip-Flop

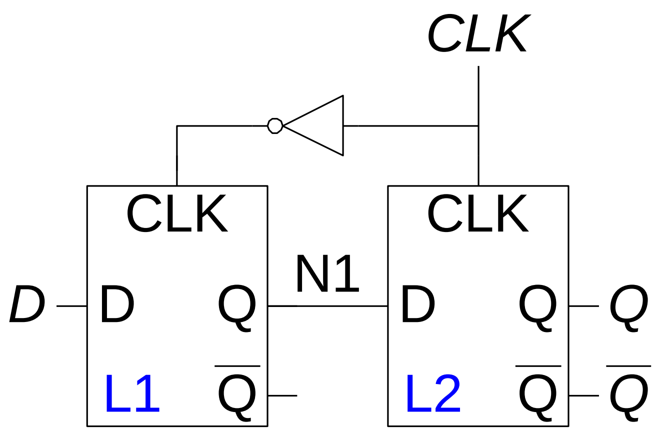
- **Two inputs: CLK, D**
- **Function**
  - The flip-flop "samples" D on the rising edge of CLK
  - When CLK rises from 0 to 1, D passes through to Q
  - Otherwise, Q holds its previous value
  - Q changes only on the rising edge of CLK
- **A flip-flop is called an edge-triggered device because it is activated on the clock edge**

D Flip-Flop Symbols



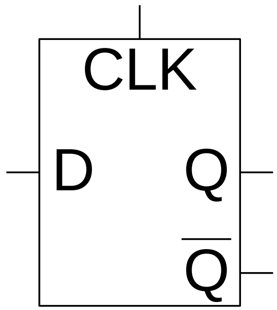
# D Flip-Flop Internal Circuit

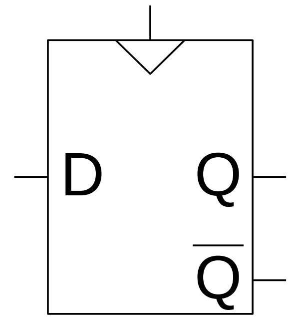
- **Two back-to-back latches (L1 and L2) controlled by complementary clocks**
  - ***When CLK = 0***
    - L1 is transparent
    - L2 is opaque
    - D passes through to N1
  - ***When CLK = 1***
    - L2 is transparent
    - L1 is opaque
    - N1 passes through to Q



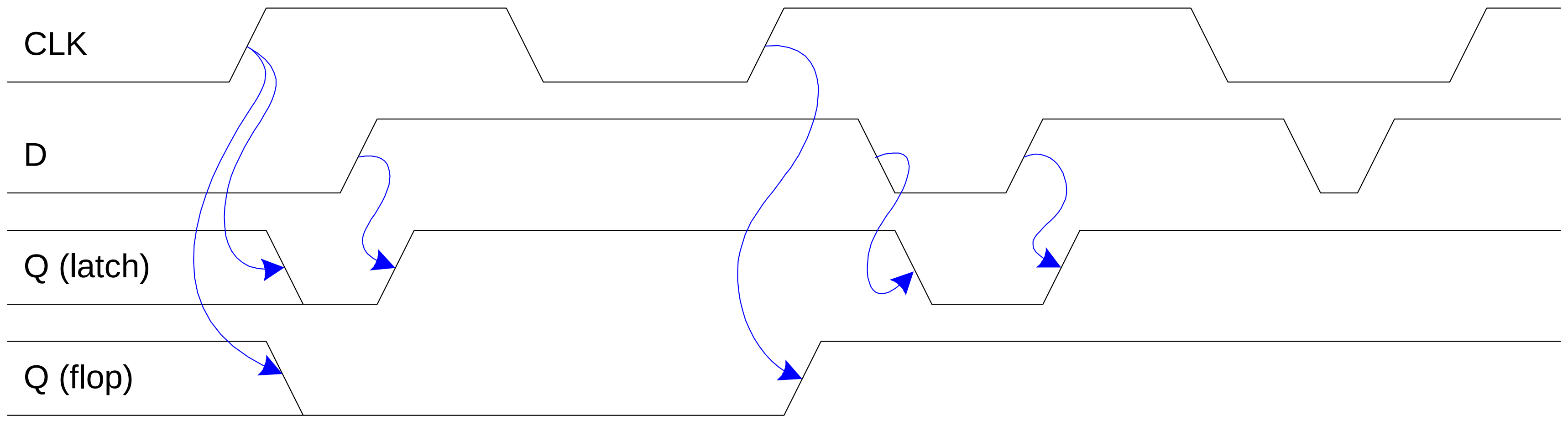

- **Thus, on the edge of the clock (when CLK rises from 0   1)**
- **D passes through to Q**

# D Flip-Flop vs. D Latch



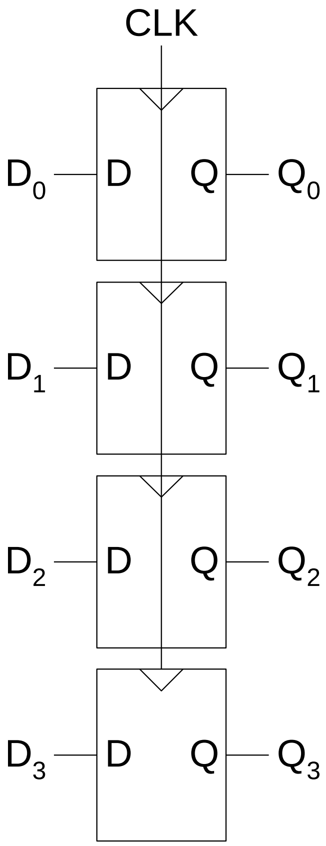



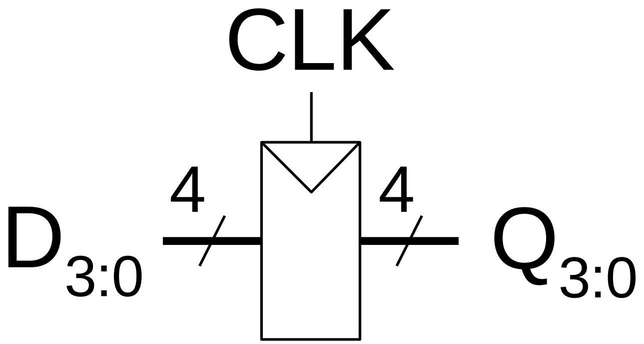
# D Flip-Flop vs. D Latch

# Registers

- **Multiple parallel flip-flops that store more than 1 bit**

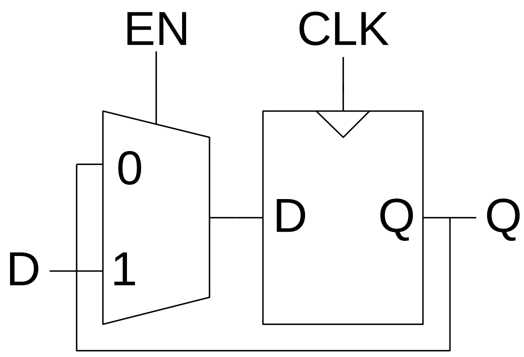


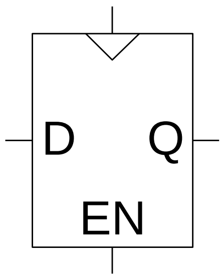
# Enabled Flip-Flops

- **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

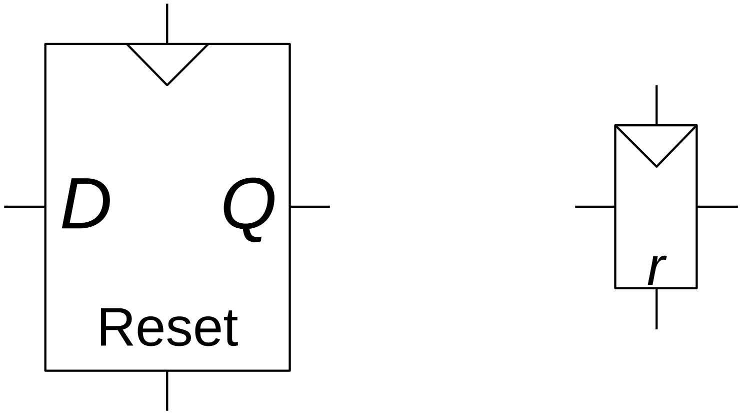


# Resettable Flip-Flops

- **Inputs: CLK, D, Reset**
  - The Reset is used to set the output to 0.

- **Function:**
  - ***Reset = 1:*** Q is forced to 0
  - ***Reset = 0:*** the flip-flop behaves like an ordinary D flip-flop

Symbols

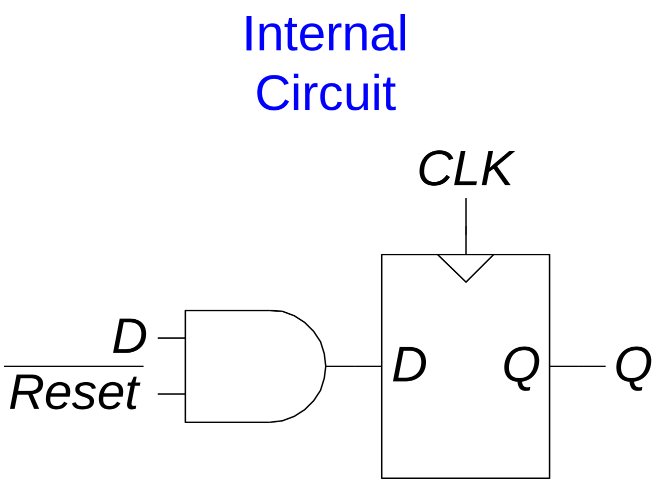


# Resettable Flip-Flops

- **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 (see Exercise 3.10)**
- **Synchronously resettable flip-flop?**

# 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 (see Exercise 3.10)**
- **Synchronously resettable flip-flop?**

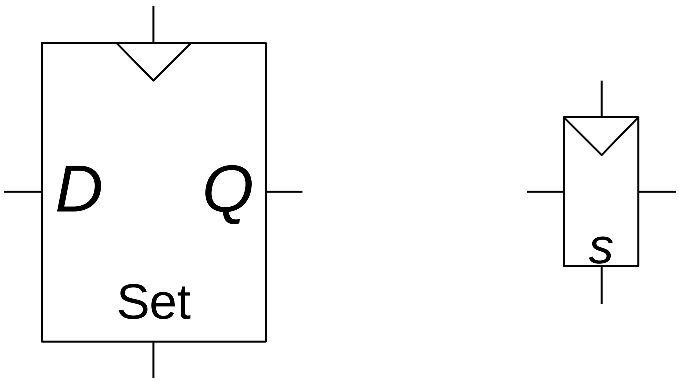


# Settable Flip-Flops

- **Inputs: CLK, D, Set**

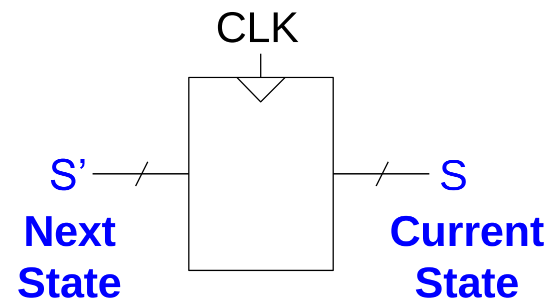
- **Function:**
  - **Set = 1**: Q is set to 1
  - **Set = 0**: the flip-flop behaves like an ordinary D flip-flop

Symbols

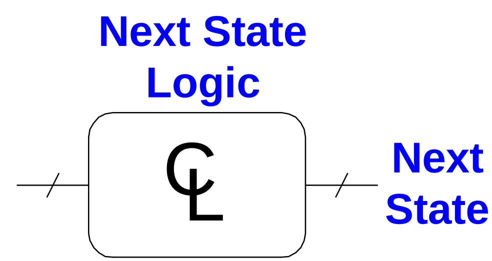


# Finite State Machine (FSM) consists of:

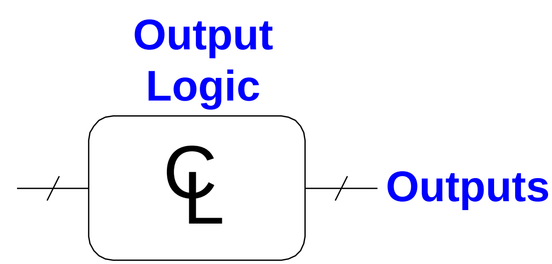
- **State register:**
  - Store the current state and
  - Load the next state at the clock edge
  - Sequential circuit




- **Next state logic**
  - Determines what the next state will be
  - Combinational circuit




- **Output logic**
  - Generates the outputs
  - Combinational Circuit



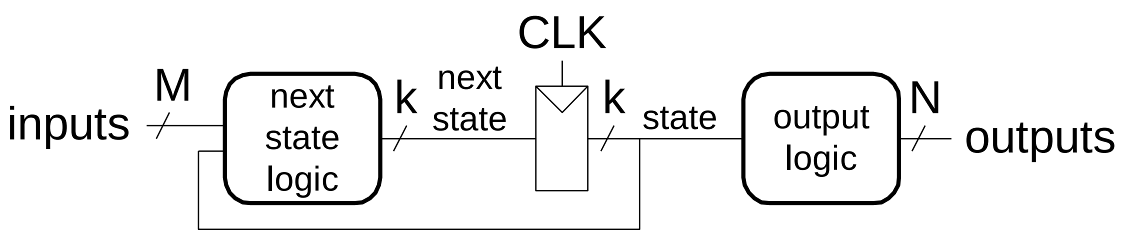
# Finite State Machine (FSM)

- FSMs get their name because a circuit with k registers can be in one of a finite number ($2^k$) of unique states.

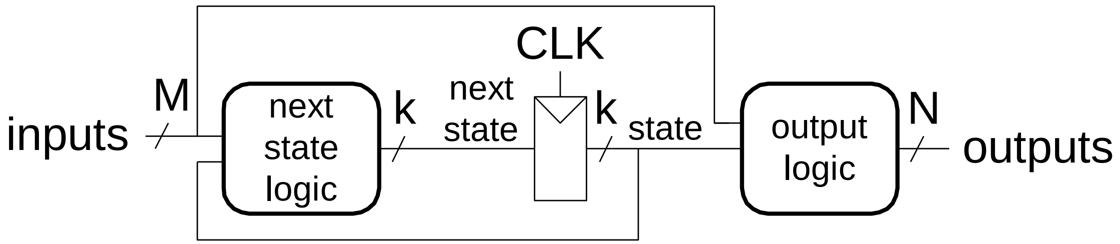
# Finite State Machines (FSMs)

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

Moore FSM

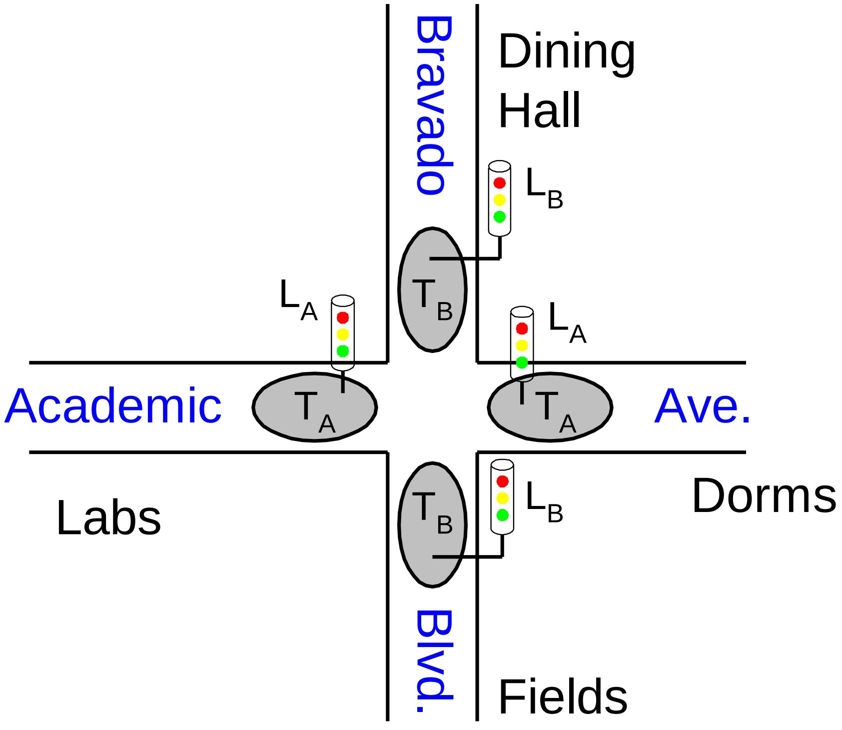



Mealy FSM



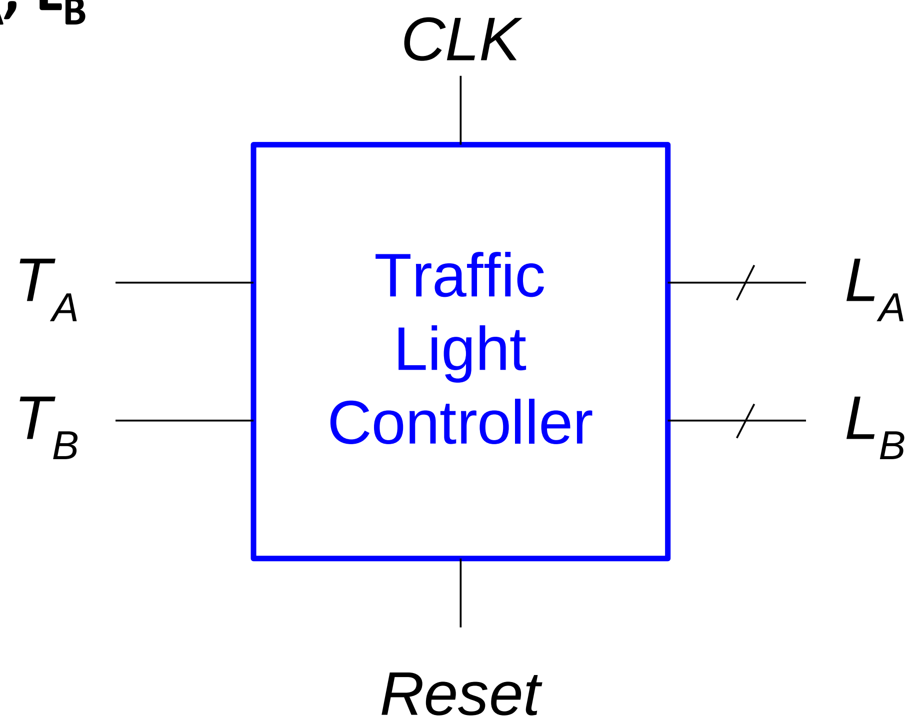
# Finite State Machine Example

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



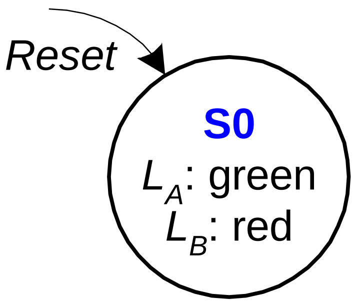
# FSM Black Box

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



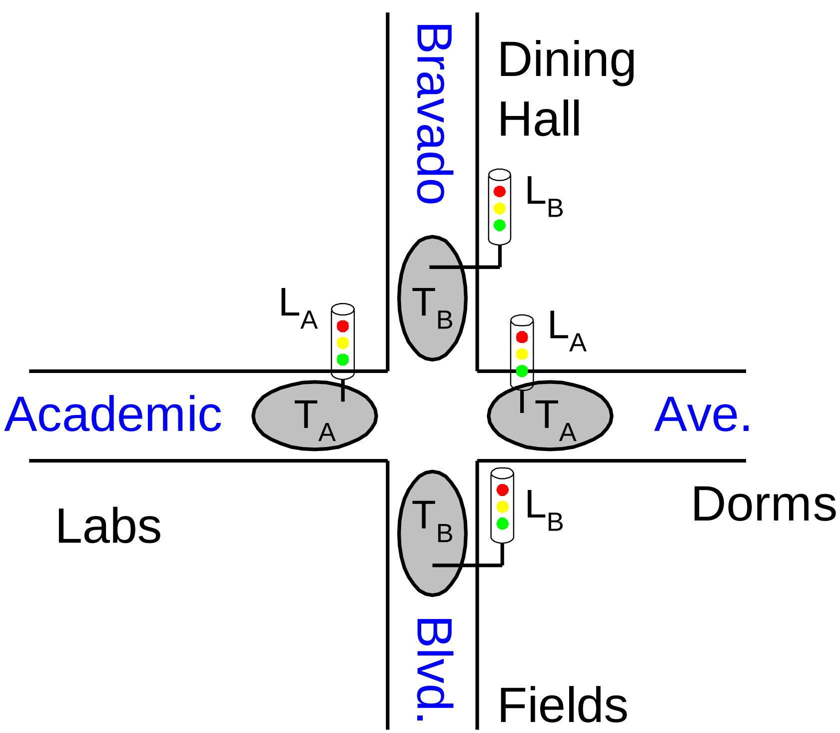
# FSM State Transition Diagram

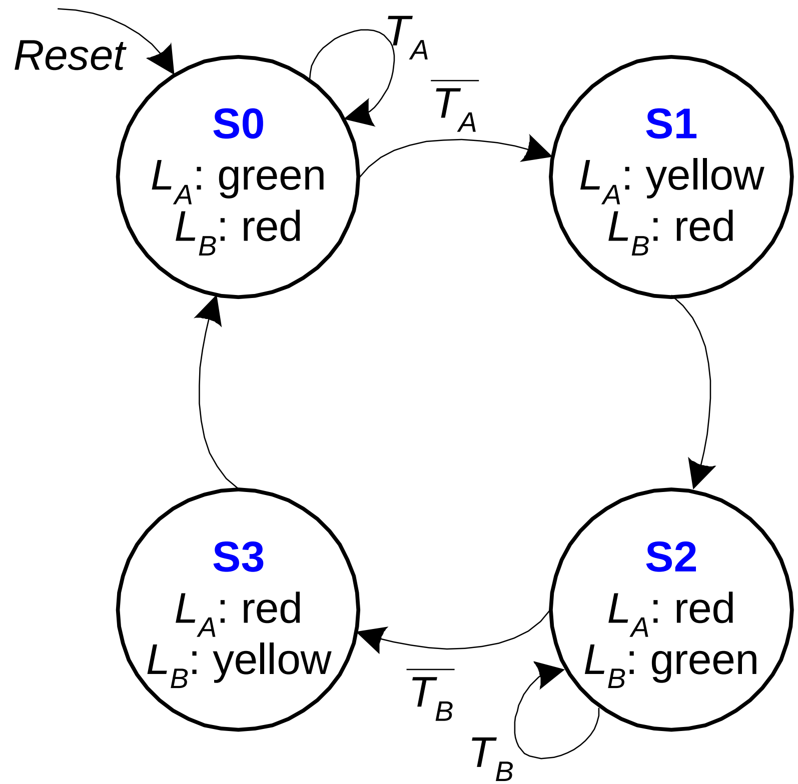
- **Moore FSM: outputs labeled in each state**
  - States: Circles
  - Transitions: Arcs



# FSM State Transition Diagram

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



# FSM State Transition Table

| Current State | 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 |

# 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 |

$$S_1' = (\overline{S_1} \cdot S_0) + (S_1 \cdot \overline{S_0} \cdot \overline{T_B}) + (S_1 \cdot \overline{S_0} \cdot T_B)$$

$$S_0' = (\overline{S_1} \cdot \overline{S_0} \cdot \overline{T_A}) + (S_1 \cdot \overline{S_0} \cdot \overline{T_B})$$

# 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 |

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

**Simplification (Inspection or K-Maps)**

$S_0' = (\overline{S_1} \cdot \overline{S_0} \cdot \overline{T_A}) + (S_1 \cdot \overline{S_0} \cdot \overline{T_B})$

# FSM Output Table

| Current State | | Outputs | |
|---|---|---|---|
| $S_1$ | $S_0$ | $L_A$ | $L_B$ |
| 0 | 0 | | |
| 0 | 1 | | |
| 1 | 0 | | |
| 1 | 1 | | |

# FSM Output Table

| Current State | | Outputs | |
|---|---|---|---|
| $S_1$ | $S_0$ | $L_A$ | $L_B$ |
| 0 | 0 | green | red |
| 0 | 1 | yellow | red |
| 1 | 0 | red | green |
| 1 | 1 | red | yellow |

| 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 |

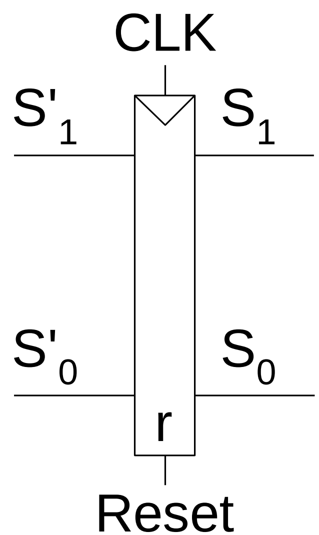
# FSM Output Table

| 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 |

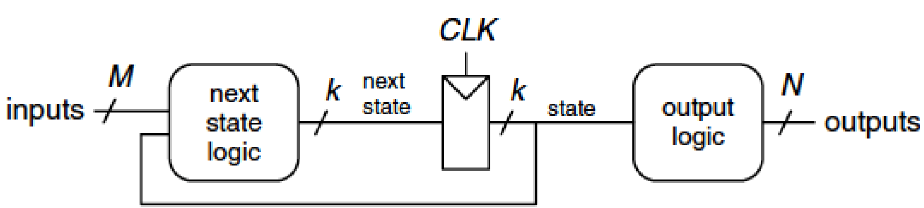
| Output | Encoding |
|---|---|
| green | 00 |
| yellow | 01 |
| red | 10 |

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

# FSM Schematic: State Register

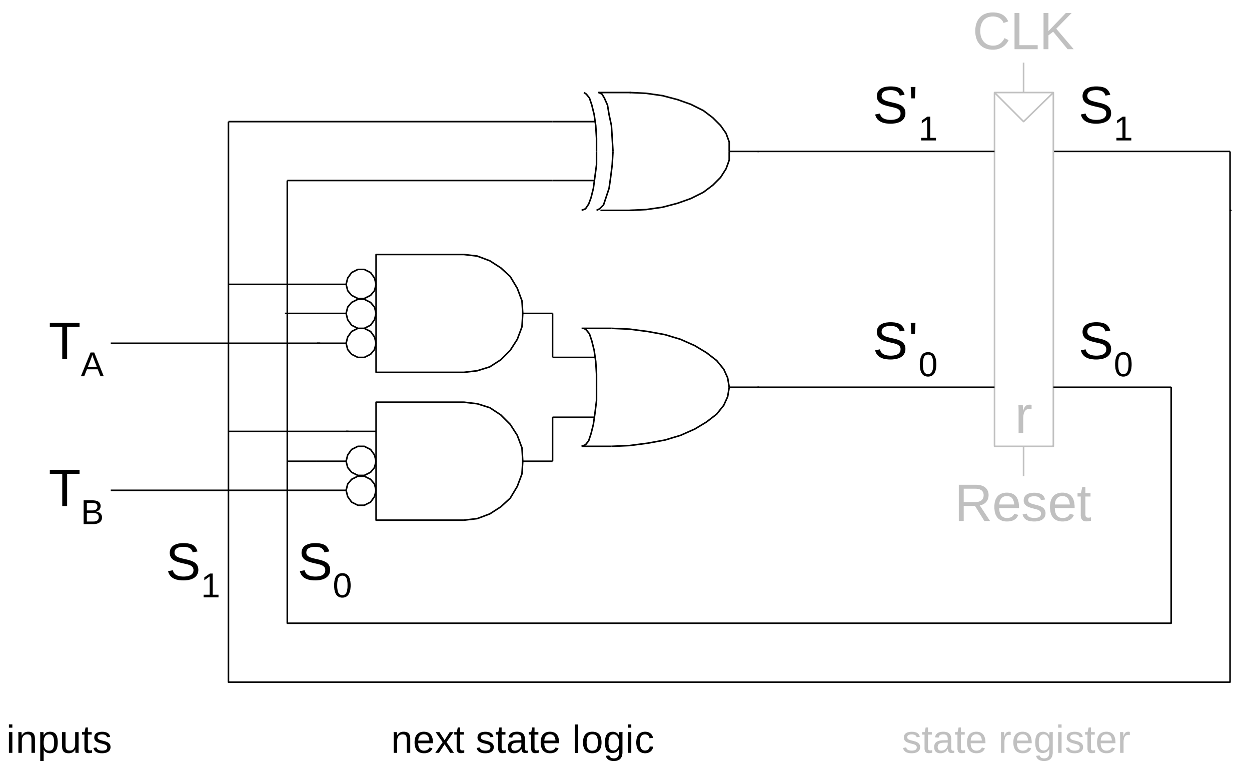


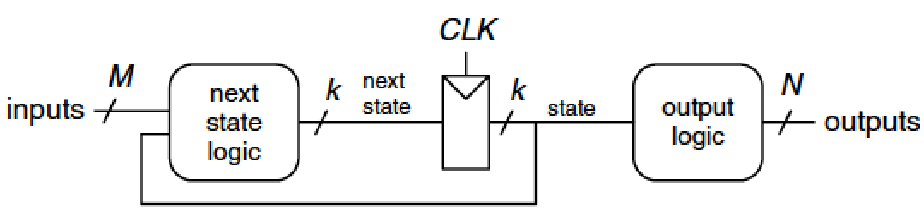

state register




(a)

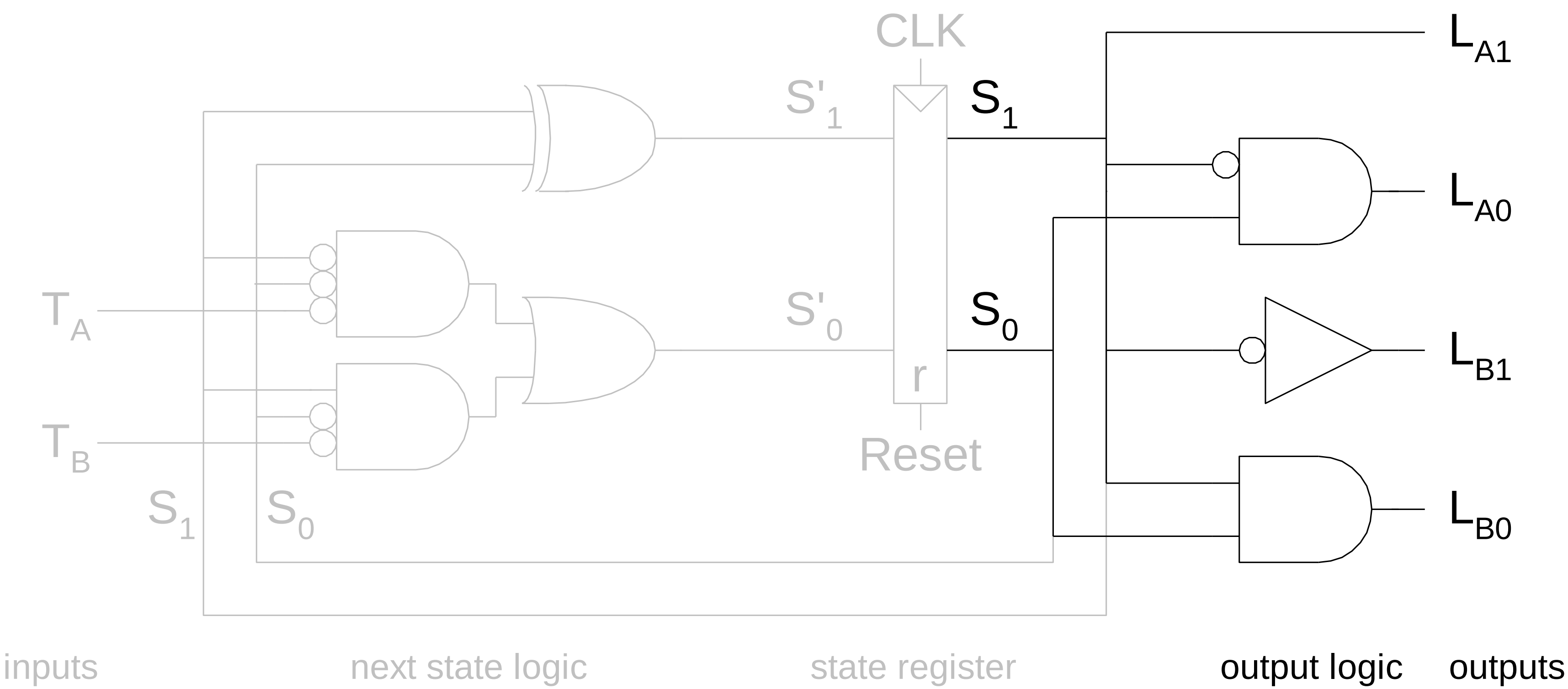
# FSM Schematic: Next State Logic

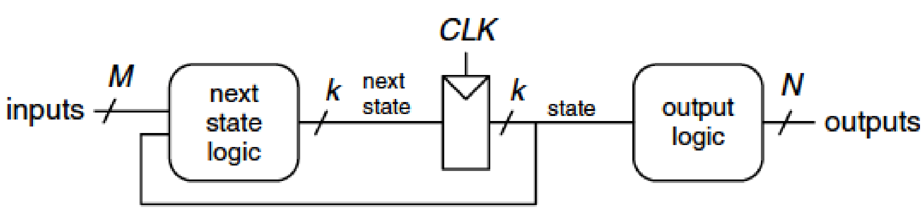



(a)

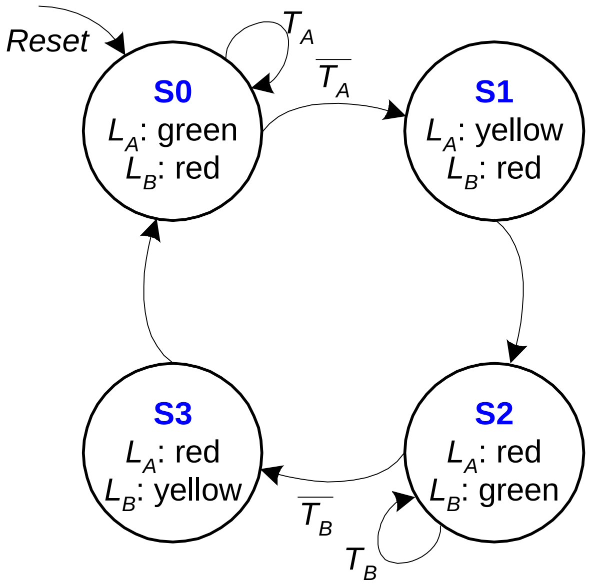
# FSM Schematic: Output Logic

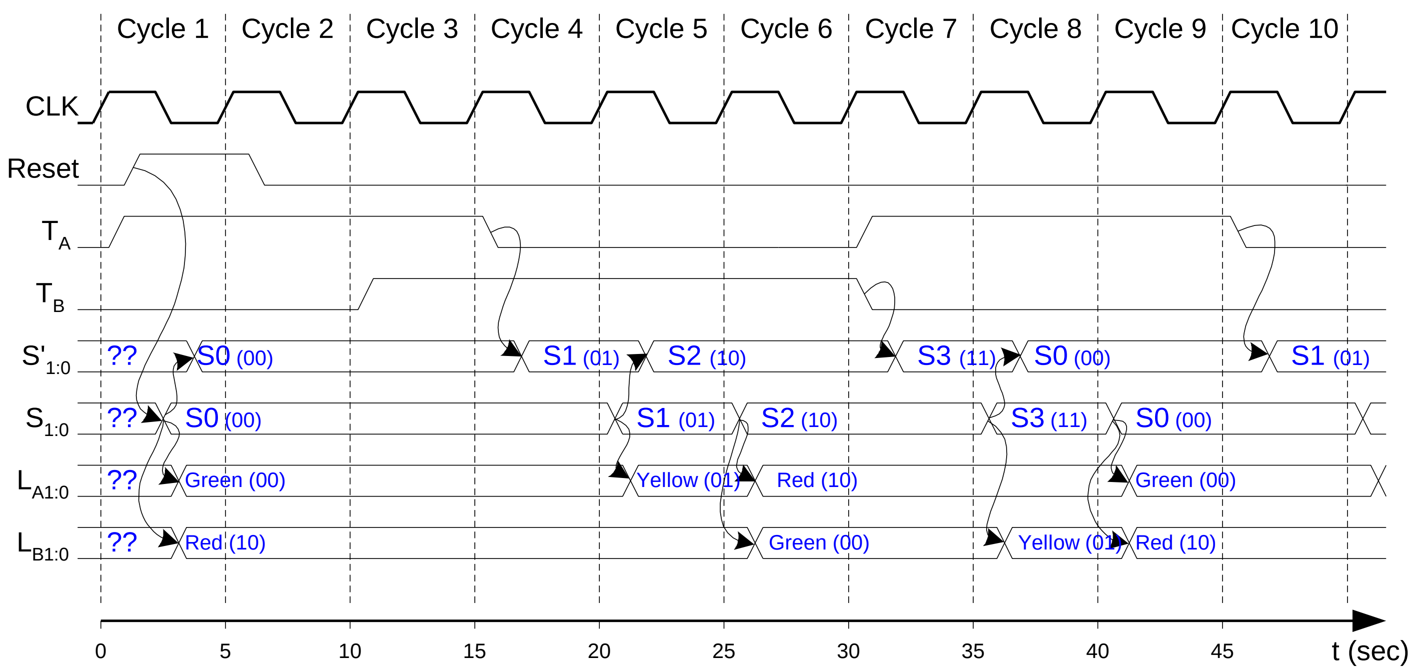



(a)

# FSM Timing Diagram



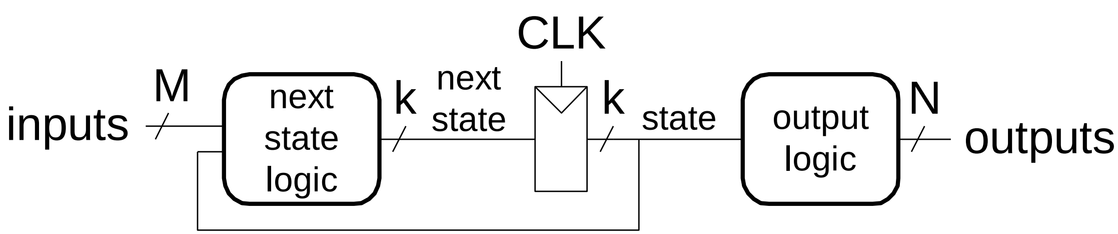
# FSM State Encoding

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

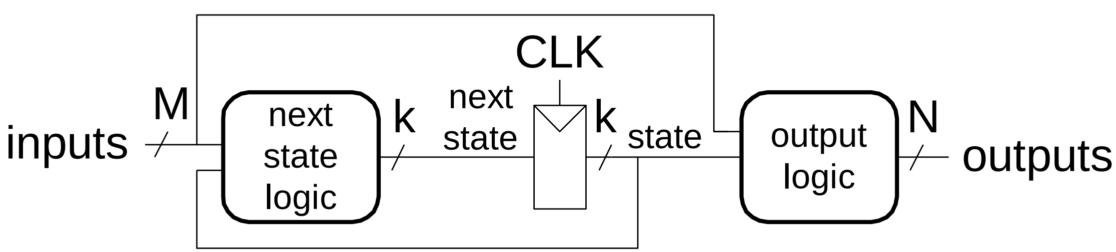
# Moore vs. Mealy FSM

- **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 four digits it has crawled over are 1101. Design Moore and Mealy FSMs of the snail's brain.**

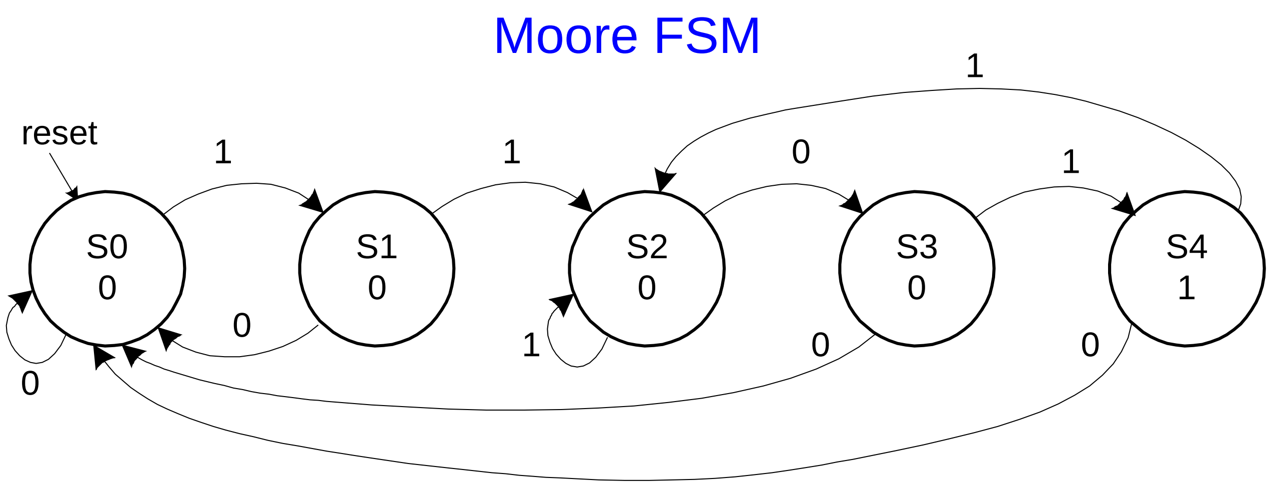
Moore FSM



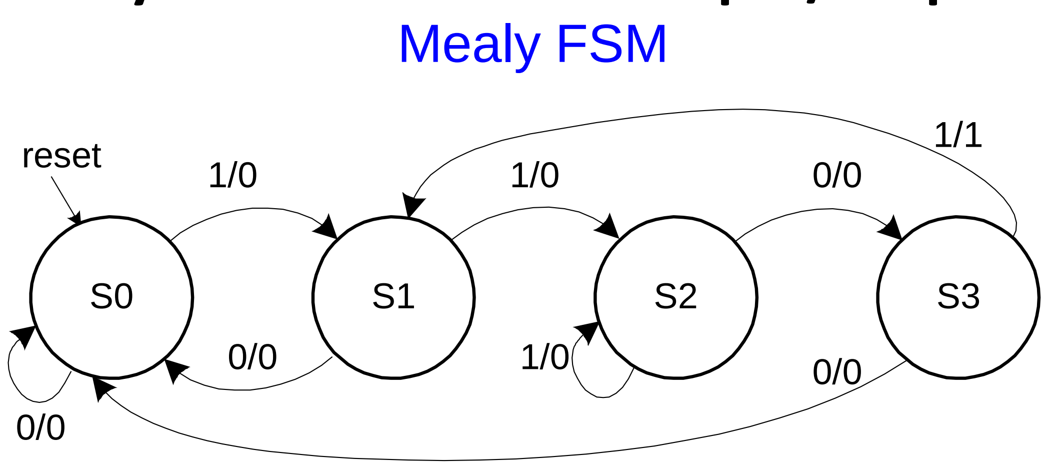

Mealy FSM




# State Transition Diagrams (snail - 1101)




- **Mealy FSM: arcs indicate input/output**



# Moore FSM State Transition Table

| Current State | | | Inputs | Next State | | |
|---|---|---|---|---|---|---|
| $S_2$ | $S_1$ | $S_0$ | A | $S'_2$ | $S'_1$ | $S'_0$ |
| 0 | 0 | 0 | 0 | | | |
| 0 | 0 | 0 | 1 | | | |
| 0 | 0 | 1 | 0 | | | |
| 0 | 0 | 1 | 1 | | | |
| 0 | 1 | 0 | 0 | | | |
| 0 | 1 | 0 | 1 | | | |
| 0 | 1 | 1 | 0 | | | |
| 0 | 1 | 1 | 1 | | | |
| 1 | 0 | 0 | 0 | | | |
| 1 | 0 | 0 | 1 | | | |

| State | Encoding |
|---|---|
| S0 | 000 |
| S1 | 001 |
| S2 | 010 |
| S3 | 011 |
| S4 | 100 |

# Moore FSM State Transition Table

| Current State | | | Inputs | Next State | | |
|---|---|---|---|---|---|---|
| $S_2$ | $S_1$ | $S_0$ | A | $S'_2$ | $S'_1$ | $S'_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |

| State | Encoding |
|---|---|
| S0 | 000 |
| S1 | 001 |
| S2 | 010 |
| S3 | 011 |
| S4 | 100 |

# Moore FSM Output Table

| Current State | | | Output |
|---|---|---|---|
| $S_2$ | $S_1$ | $S_0$ | $Y$ |
| 0 | 0 | 0 | |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |

# Moore FSM Output Table

| Current State | | | Output |
|---|---|---|---|
| $S_2$ | $S_1$ | $S_0$ | $Y$ |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |

$\mathbf{Y = S_2}$

# Mealy FSM State Transition and Output

| Current State | | Input | Next State | | Output |
|---|---|---|---|---|---|
| $S_1$ | $S_0$ | A | $S'_1$ | $S'_0$ | Y |
| 0 | 0 | 0 | | | |
| 0 | 0 | 1 | | | |
| 0 | 1 | 0 | | | |
| 0 | 1 | 1 | | | |
| 1 | 0 | 0 | | | |
| 1 | 0 | 1 | | | |
| 1 | 1 | 0 | | | |
| 1 | 1 | 1 | | | |

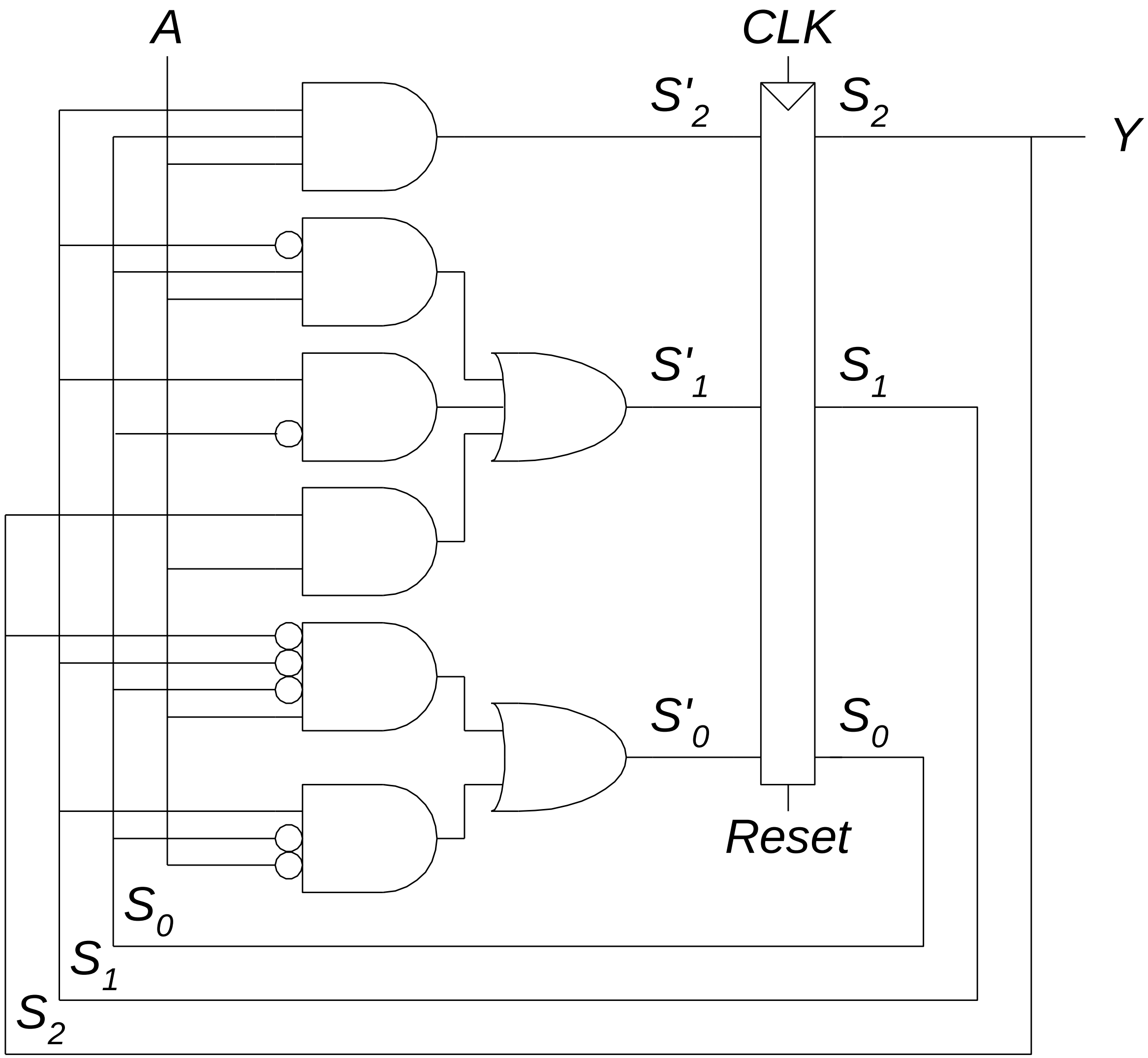
| State | Encoding |
|---|---|
| S0 | 00 |
| S1 | 01 |
| S2 | 10 |
| S3 | 11 |

# Mealy FSM State Transition and Output

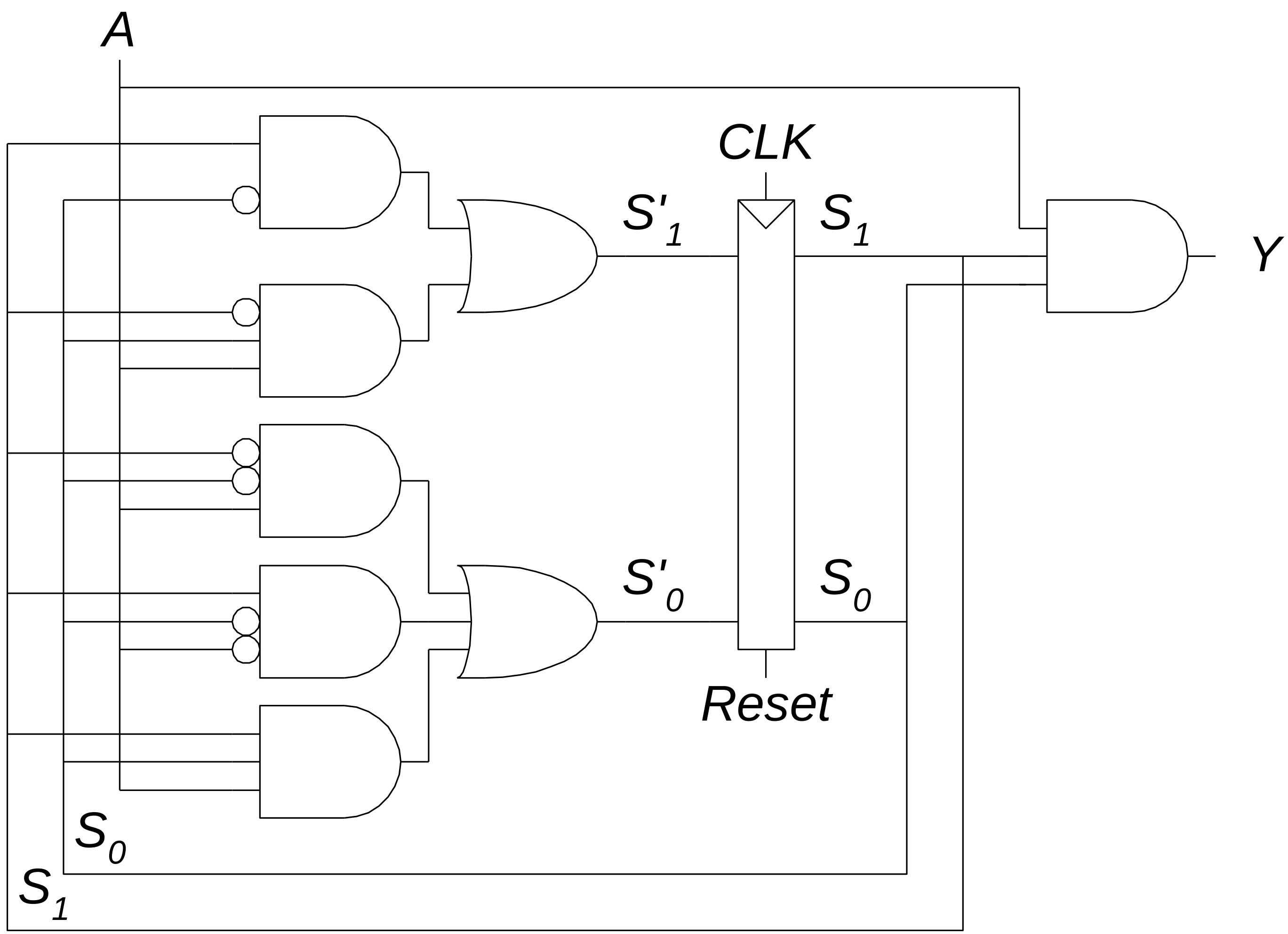
| Current State | | Input | Next State | | Output |
|---|---|---|---|---|---|
| $S_1$ | $S_0$ | A | $S'_1$ | $S'_0$ | Y |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 |

| State | Encoding |
|---|---|
| S0 | 00 |
| S1 | 01 |
| S2 | 10 |
| S3 | 11 |

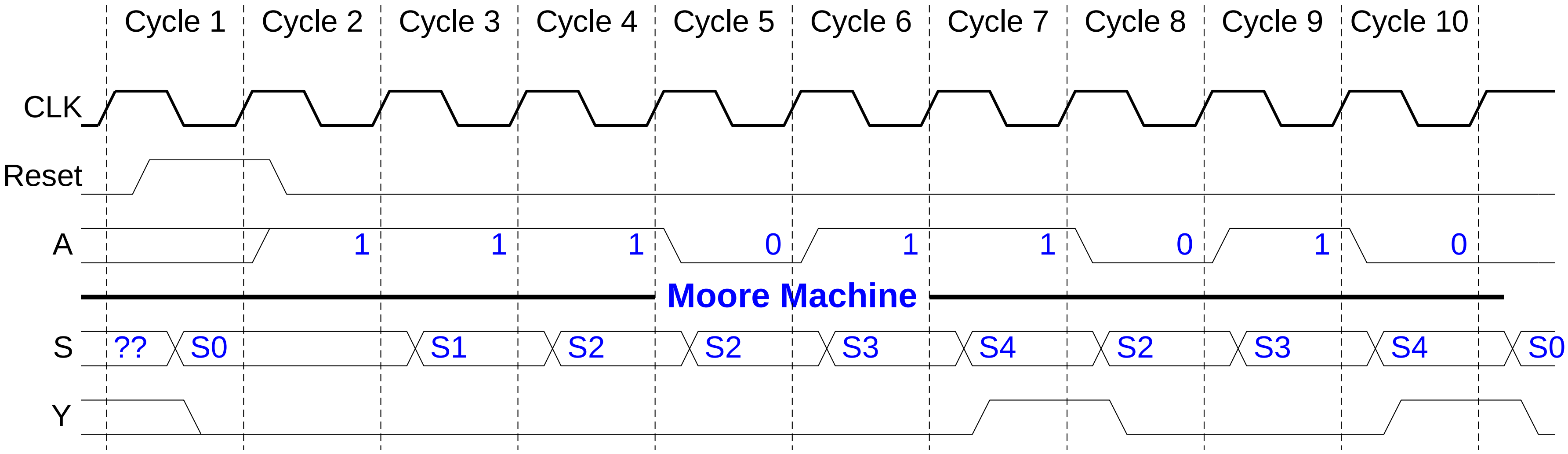
# Moore FSM Schematic



# Mealy FSM Schematic

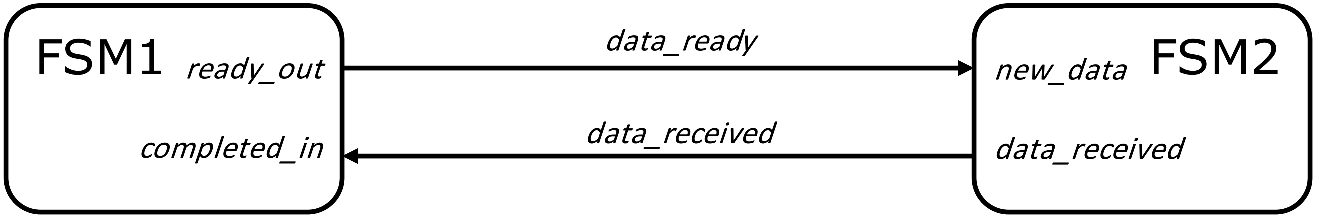


# Moore and Mealy Timing Diagram



# Why do we care if FSM is Moore/Mealy?

- **Is it to ask questions in an exam?**
  - Traditionally, Frank always asks a Mealy/Moore question
- **Remember combinational circuits**
  - You are not supposed to have loops
- **If two FSMs are connected, there will be NO loop if**
  - At least one of them is MOORE
- **There CAN BE a combinational loop if**
  - Both FSMs are MEALY type.



# Factoring State Machines

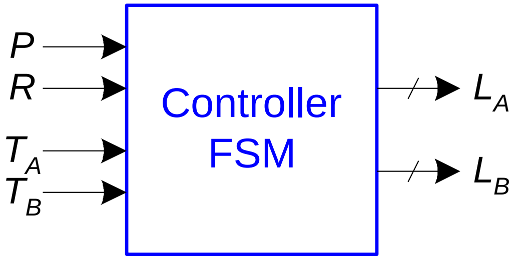
- **Break complex FSMs into smaller interacting FSMs**

- **Example: Modify the traffic light controller to have a Parade Mode.**
  - The FSM receives two more inputs: *P*, *R*
  - When *P* = 1, it enters Parade Mode and the Bravado Blvd. light stays green.
  - When *R* = 1, it leaves Parade Mode

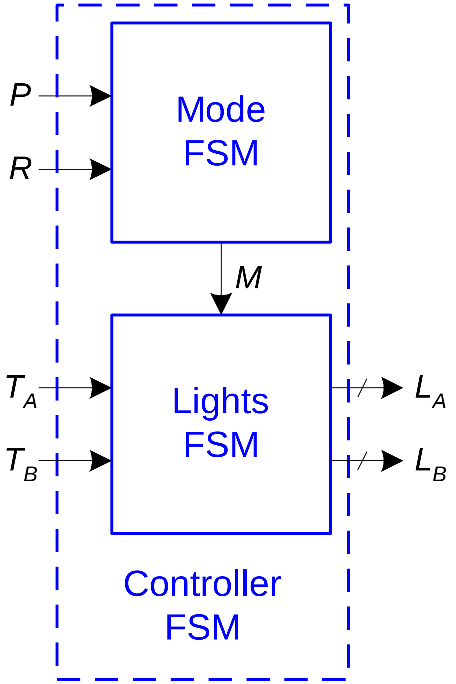
- **Designing complex FSMs is often easier if they can be broken down into multiple interacting simpler state machines such that the output of some machines is the input of others. This application of hierarchy and modularity is called factoring of state machines.**

# Parade FSM

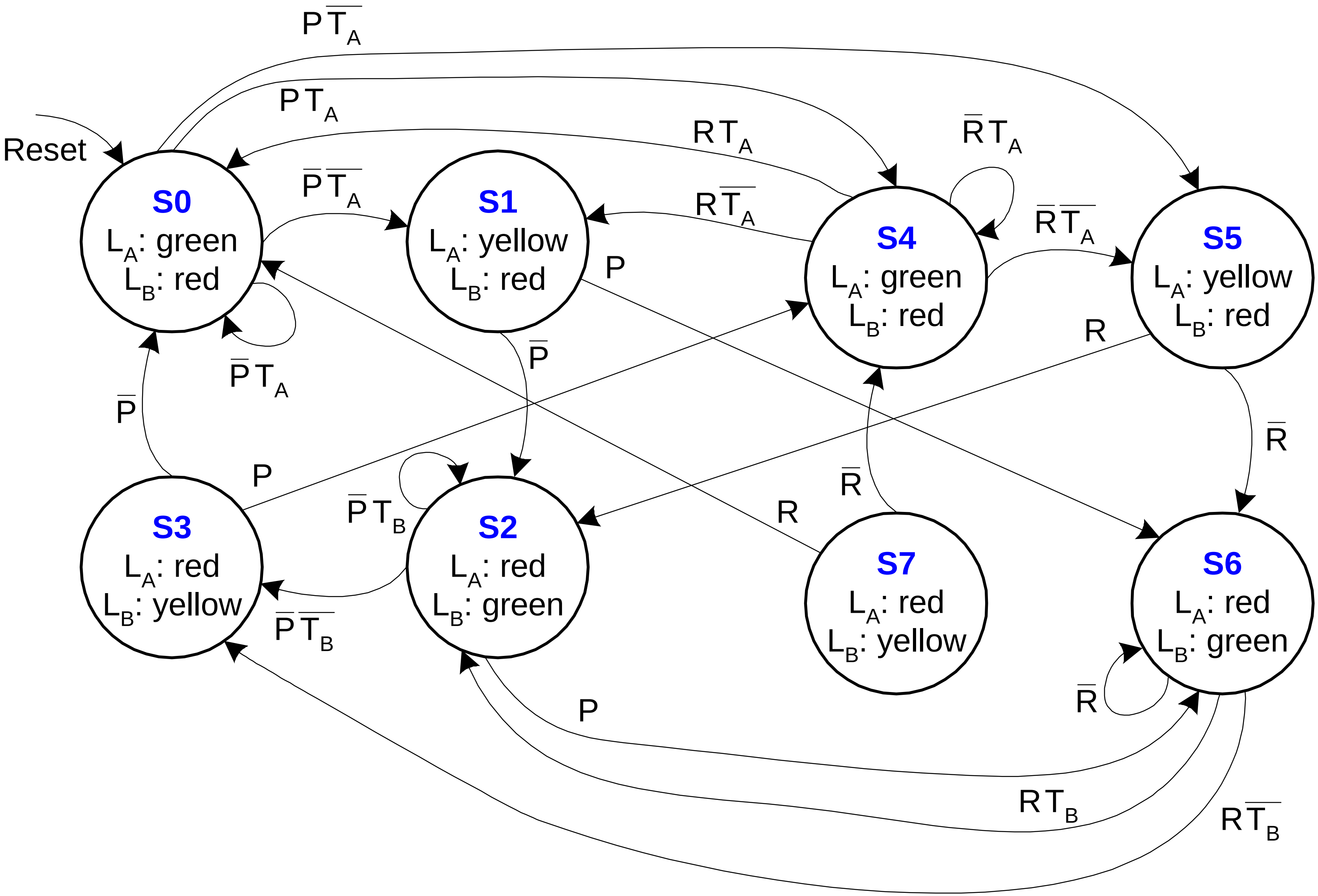
## Unfactored FSM



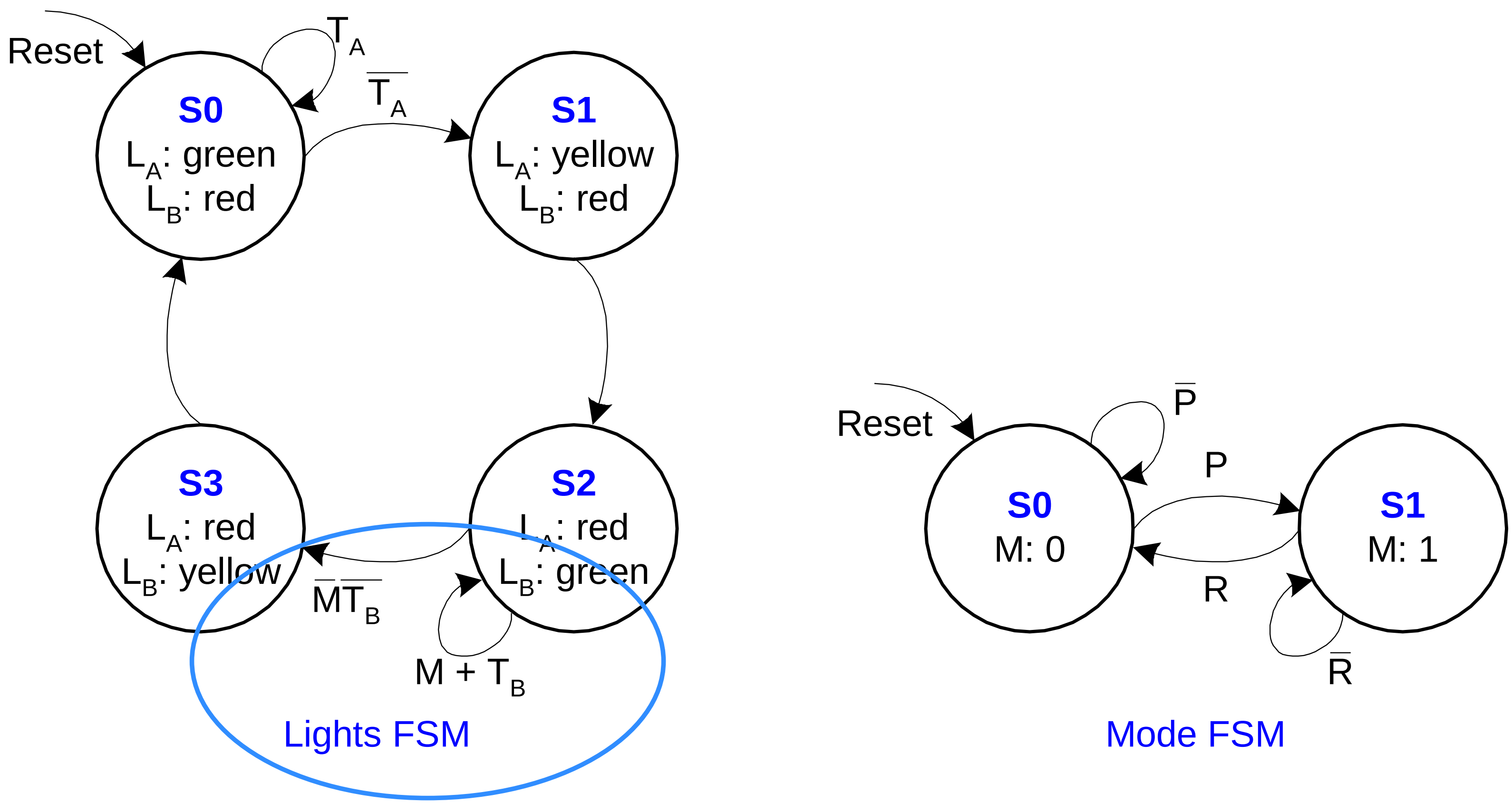

## Factored FSM



# Unfactored FSM State Transition Diagram



# Factored FSM State Transition Diagram



# FSM Design Procedure

- **Prepare**
  - Identify the inputs and outputs
  - Sketch a state transition diagram
  - Write a state transition table
  - Select state encodings
- **For a *Moore* machine**:
  - Rewrite the state transition table with the selected state encodings
  - Write the output table
- **For a *Mealy* machine:**
  - Rewrite the combined state transition and output table with the selected state encodings
- **Write Boolean equations for the next state and output logic**
- **Sketch the circuit schematic**

# What Did We Learn?

- **D Latch is the basic memorizing element**
  - Transparent mode, copies input to output
  - Latch mode, keeps content
- **(Rising) Edge Triggered Flip-Flops are more practical**
  - Input is copied to output when the clock rises from 0 to 1
- **Finite State Machines**
  - ***Moore***, output depends on **only** the **current state**
  - ***Mealy***, output depends on **current state and the inputs**.
- **Three Aspects of an FSM**
  - Holds the present state
  - Calculate the next state
  - Determine the outputs