Sequential Logic Design

Dr. Igor Ivkovic

iivkovic@uwaterloo.ca

[with material from “Computer Organization and Design” by Patterson and Hennessy, and “Digital Design and Computer Architecture” by Harris and Harris, both published by Morgan Kaufmann]

Objectives

- Introduction to Sequential Logic Circuits
- Latches, Flip-Flops, and Registers
- Synchronous Logic Design
- Finite State Machines

Sequential Logic Circuit /1

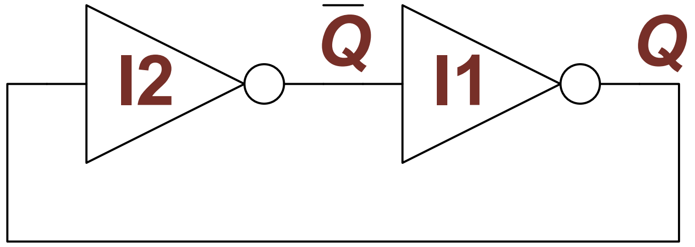
- **Sequential Logic Circuit – It Has Memory**
  - Outputs of a sequential logic circuit depend on the current and the prior input values
- **State:**
  - All the information about a circuit necessary to explain its future behavior
- **Latches and Flip-Flops:**
  - State elements that store one bit of state
- **Synchronous Sequential Circuits:**
  - Combinational logic followed by a bank of flip-flops

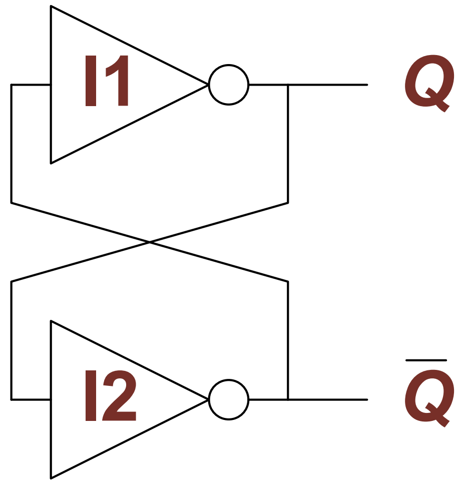
Sequential Logic Circuit /2

- **Sequential Logic Circuits:**
  - Give sequence to events
  - Have (short-term) memory
  - Use feedback from output to input to store information
- **State Elements:**
  - The state of a circuit influences its future behavior
  - State elements store the circuit state
- **State elements include:**
  - Bistable circuit
  - SR Latch
  - D Latch
  - D Flip-flop

Bistable Circuit /1

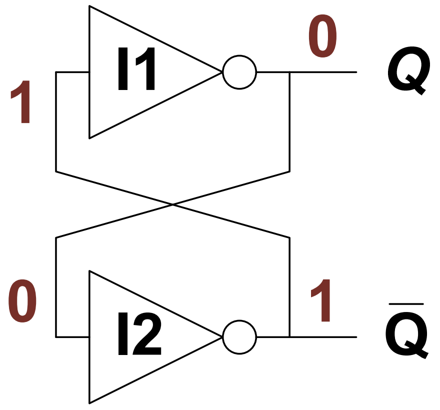
- **Bistable Circuit:**
  - Fundamental building block of other state elements
  - Two outputs: Q, $\overline{Q}$
  - **No inputs**



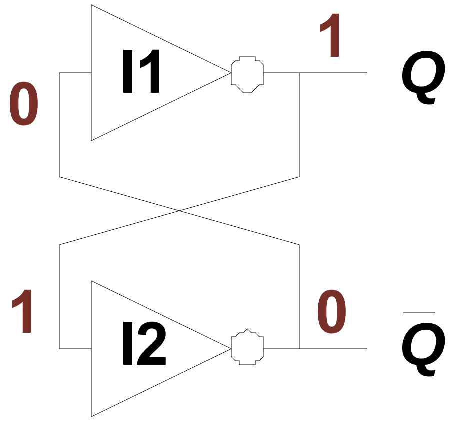


Bistable Circuit /2

- **Consider the two possible cases:**
  - $Q = 0$: then $\overline{Q} = 1$, $Q = 0$ (stable)



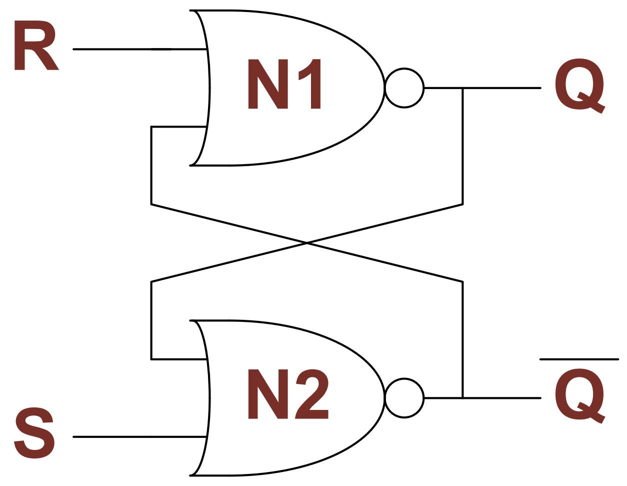
  - $Q = 1$: then $\overline{Q} = 0$, $Q = 1$ (also stable); hence, called bistable



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

SR (Set/Reset) Latch /1

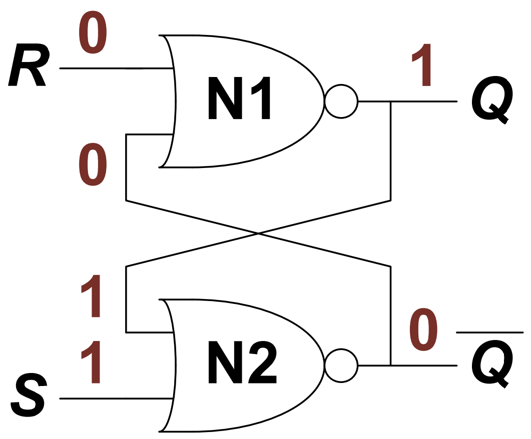
- **SR (Set/Reset) Latch:**
  - Composed of two cross-coupled NOR gates
  - It has two inputs, S and R, and two outputs, Q and $\overline{Q}$



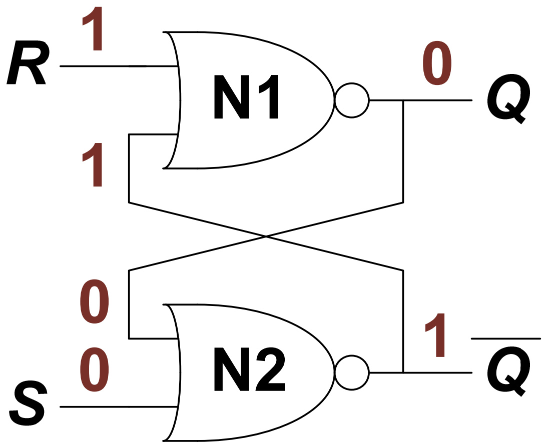
- **Consider the four possible cases:**
  - S = 1, R = 0
  - S = 0, R = 1
  - S = 0, R = 0
  - S = 1, R = 1

SR (Set/Reset) Latch /2

- **SR Latch analysis:**
  - S = 1, R = 0:
    then Q = 1 and $\overline{Q}$ = 0



  - S = 0, R = 1:
    then $\overline{Q}$ = 1 and Q = 0

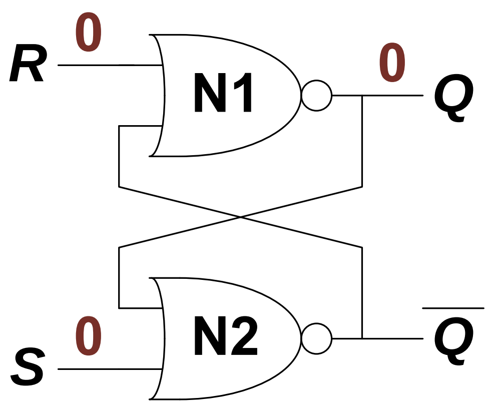


SR (Set/Reset) Latch /3

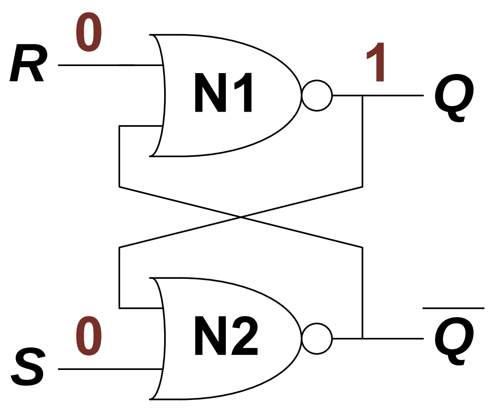
- **SR Latch**
  - S = 0, R = 0:

    then Q = Q_{prev}
  - That is, we have **MEMORY** since Q = Q_{prev}

Q_{prev} = 0

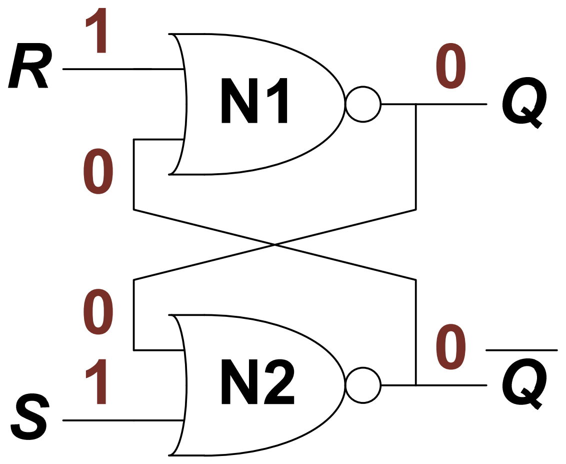


Q_{prev} = 1



  - S = 1, R = 1:

    then Q = 0, $\overline{Q}$ = 0
  - We have an **invalid state since Q ≠ NOT $\overline{Q}$**

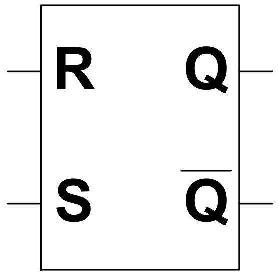


SR (Set/Reset) Latch /4

- **SR Latch Summary:**
  - Stores one bit of state (Q)
  - Control what value is being stored with S, R as inputs
  - Set: Make the output 1 (S = 1, R = 0, Q = 1)
  - Reset: Make the output 0 (S = 0, R = 1, Q = 0)
  - Must do something to avoid **invalid state (when S = R = 1)**
- **Can be implemented in different ways as long as it corresponds to the following truth table**

S	R	Q	$\overline{Q}$
0	0	Q_{prev}	$\overline{Q}_{prev}$
0	1	0	1
1	0	1	0
1	1	0	0

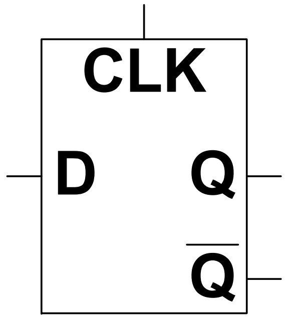
SR Latch Symbol



D Latch /1

- **D Latch:**
  - Sequential logic circuit element that separates the state data from the timing of state change
  - Two inputs: clock input signal (CLK), data input signal (D)
  - **CLK signal: controls when the output changes**
  - **D signal/bit: controls to what the output changes**
- **Operational functionality:**
  - When CLK = 1, D passes through to Q (transparent)
  - When CLK = 0, Q holds its previous value (opaque)
  - **Avoids invalid case when Q ≠ NOT $\overline{Q}$**

D Latch Symbol



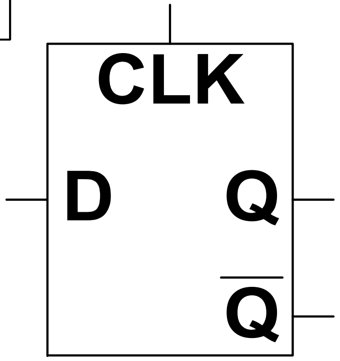
D Latch /2

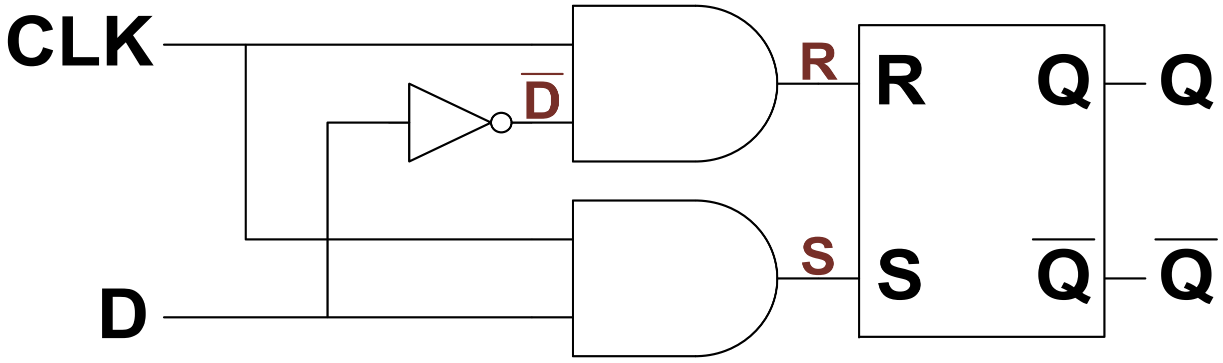
- **D Latch Internals:**

Remembers the previous state if CLK = 0; otherwise Q equals D

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

S and R cannot both be 1

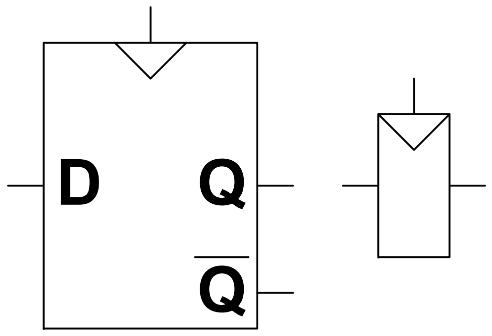




D Flip-Flop /1

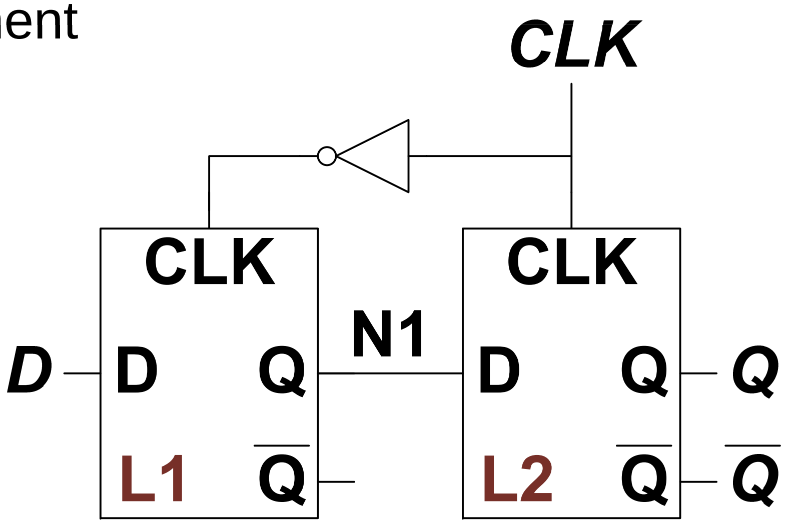
- **D Flip-Flop:**
  - Sequential logic circuit built from two D latches controlled by CLK and its complement
  - Two Inputs: CLK, D
- **Operational functionality:**
  - Samples D on the rising edge of CLK
  - When CLK rises from 0 to 1,
    D passes through to Q
  - Otherwise, Q holds its previous value
  - **Hence, Q changes only on the rising edge of CLK**
- **Also called edge-triggered**
  - Since it is activated on the clock edge

D Flip-Flop Symbols



D Flip-Flop /2

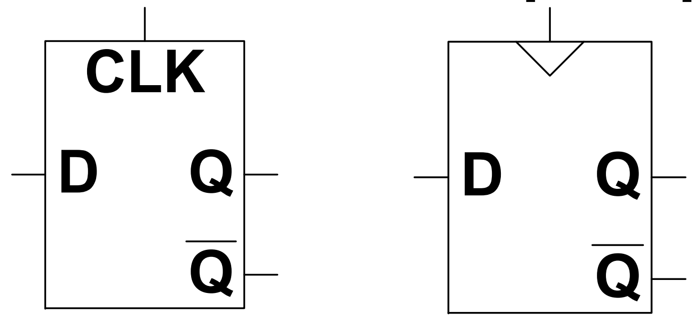
- **D Flip-Flop explained:**
  - Two back-to-back latches (L1 and L2) controlled by the clock (CLK) signal and its complement
  - **When CLK = 0**
    - L1 is transparent
    - L2 is opaque
    - D passes through to N1
  - **When CLK = 1**
    - L2 is transparent
    - L1 is opaque
    - N1 passes through to Q
  - Hence, on the edge of the clock (when CLK rises from 0 to 1)
    - **D passes through to Q**

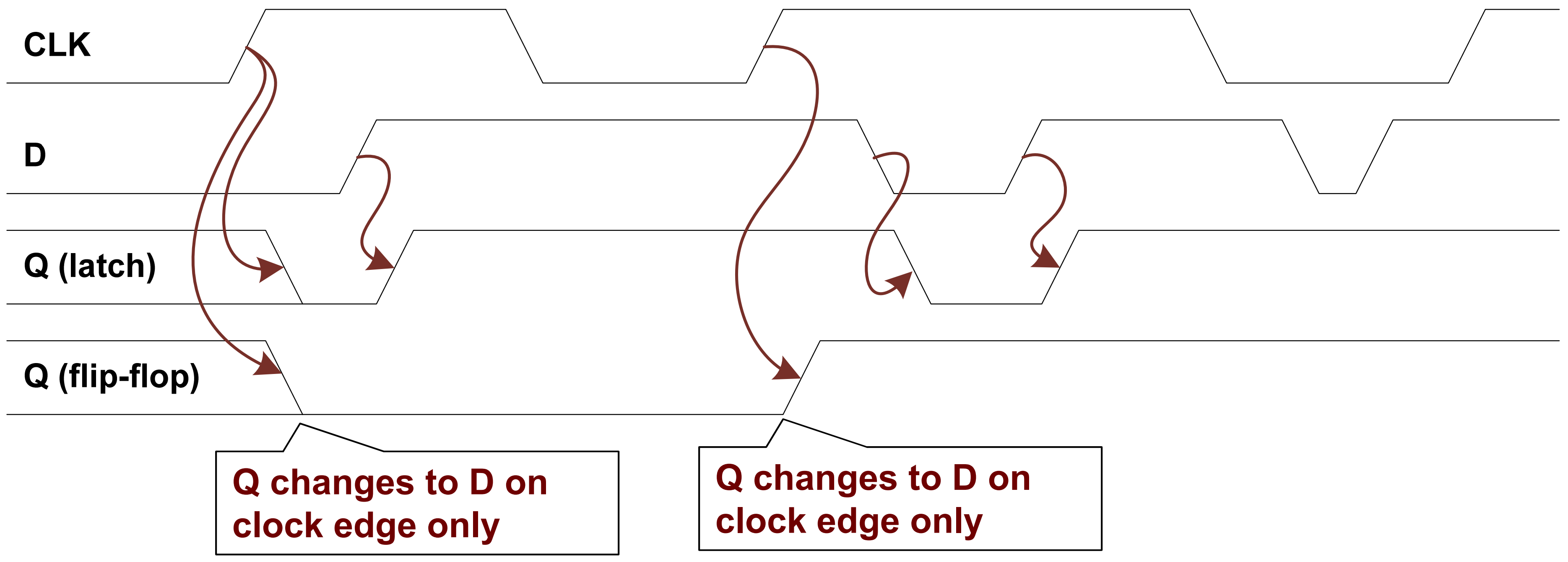


- **Fun Trivia:**
  - How many transistors are needed to build a typical D flip flop?
    - Hint: 2 transistors for NOT gate, 4 for NAND and NOR

D Flip-Flop /3

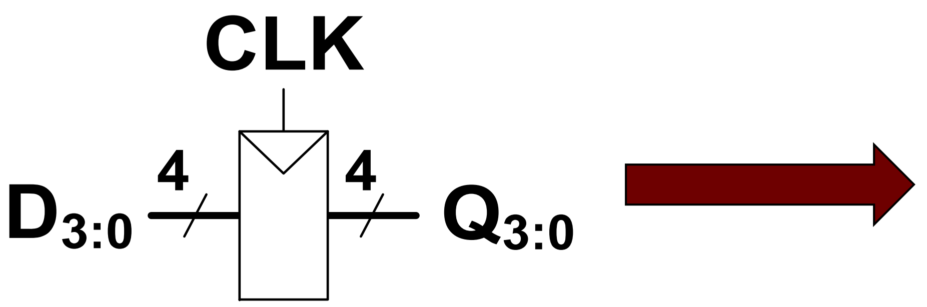
- **D Latch vs. D Flip-Flop:**

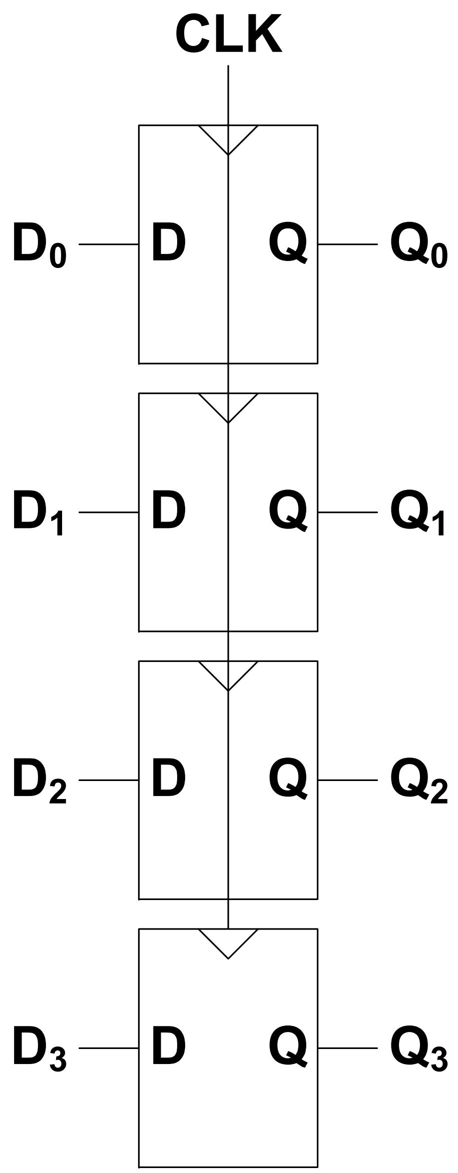



CLK
D
Q (latch)
Q (flip-flop)
Q changes to D on clock edge only
Q changes to D on clock edge only

Register Introduced

- **N-bit Register:**
  - A grouping of N flip-flops that share a common CLK input, so that all bits of the register are updated at the same time
- Example: 4-bit register

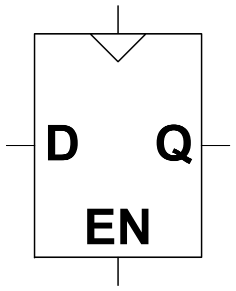


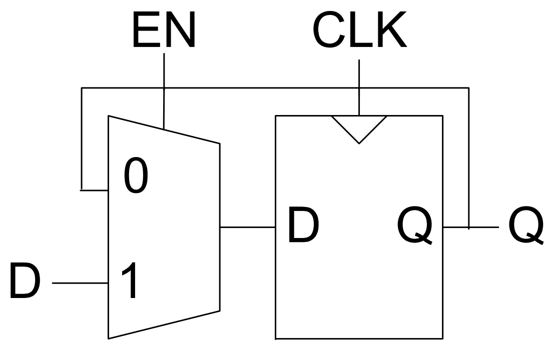


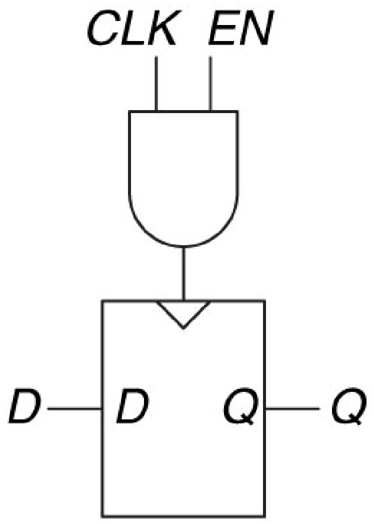
Enabled Flip-Flop

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

Symbol



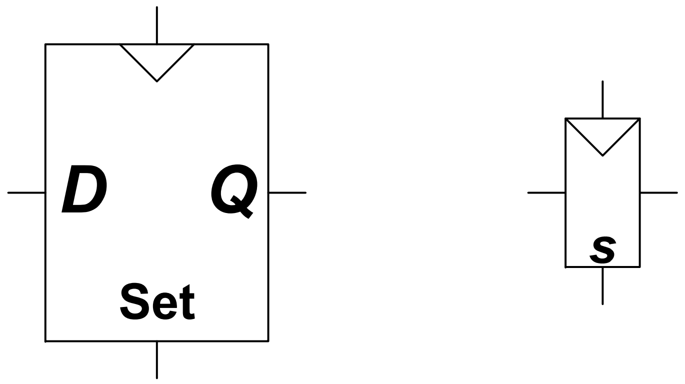




Set and Reset Flip-Flops

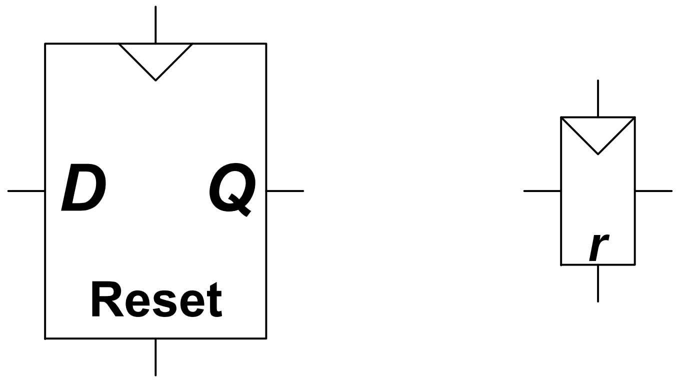
- **Set Flip-Flop:**
  - Inputs: CLK, D, Set
- **Operational functionality:**
  - Set = 1: Q is forced to 1
  - Set = 0: flip-flop behaves as an ordinary D flip-flop

Symbols



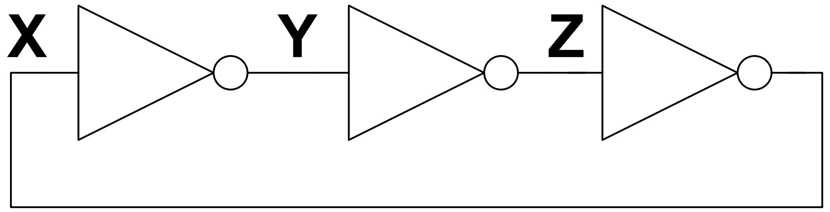
- **Reset Flip-Flop:**
  - Inputs: CLK, D, Reset
- **Operational functionality:**
  - Reset = 1: Q is forced to 0
  - Reset = 0: flip-flop behaves as an ordinary D flip-flop

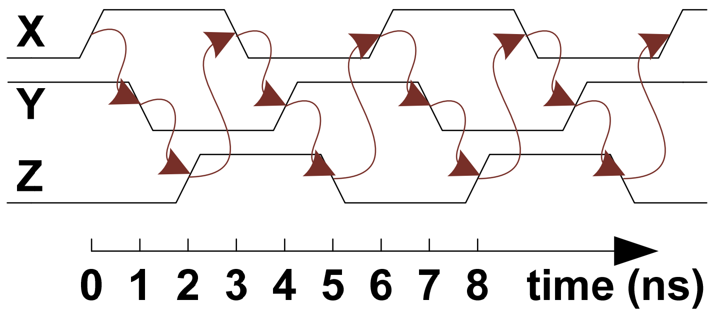
Symbols



Synchronous Logic Design /1

- **Consider the following problematic circuit:**





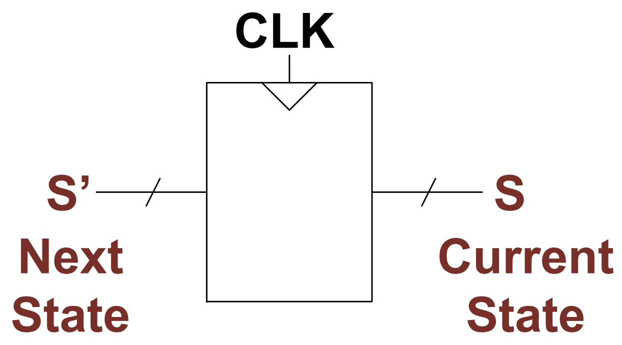
  - No inputs and 1-3 outputs
  - The circuit oscillates and it is not stable
  - It has a cyclic path: output is fed back as input

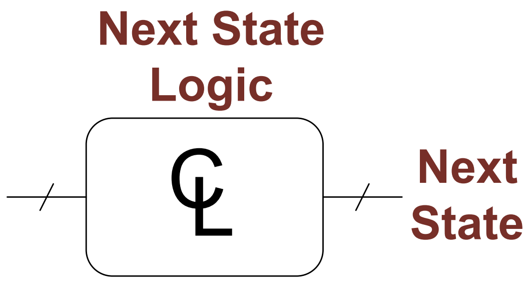
Synchronous Logic Design /2

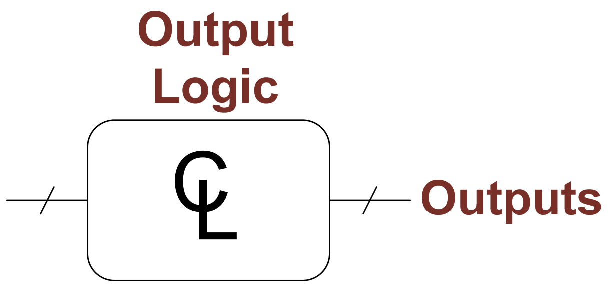
- **Breaks cyclic paths by inserting registers**
  - Registers contain state of the system
  - State changes at clock edge: system synchronized to the clock
- **Rules of synchronous sequential circuit composition:**
  - Every circuit element is either a register or a combinational logic circuit
  - At 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

Synchronous Logic Design /3

- **Finite State Machine (FSM) consists of:**
  - State register
  - Combinational logic
- **State register:**
  - Stores the current state
  - Loads the next state at clock edge
- **Combinational (combinatorial) logic:**
  - Computes the next state
  - Computes the outputs



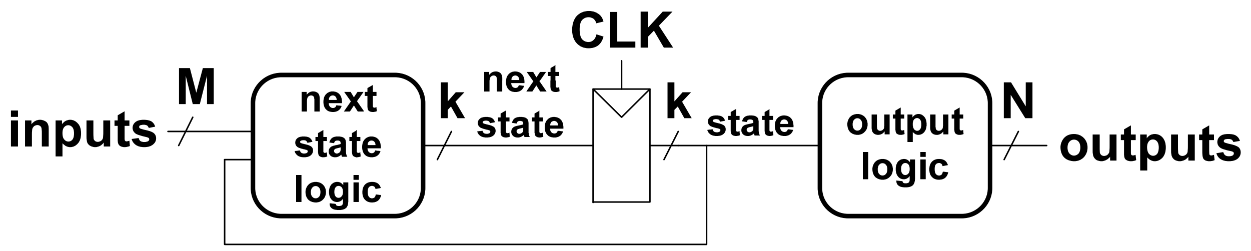




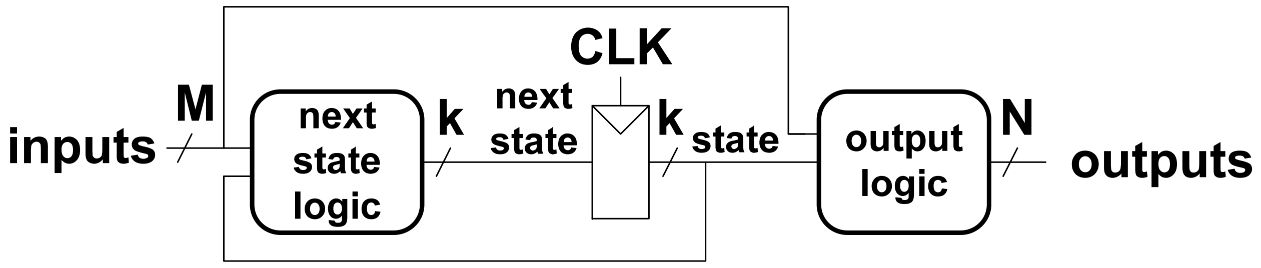
Finite State Machines /1

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

Moore FSM

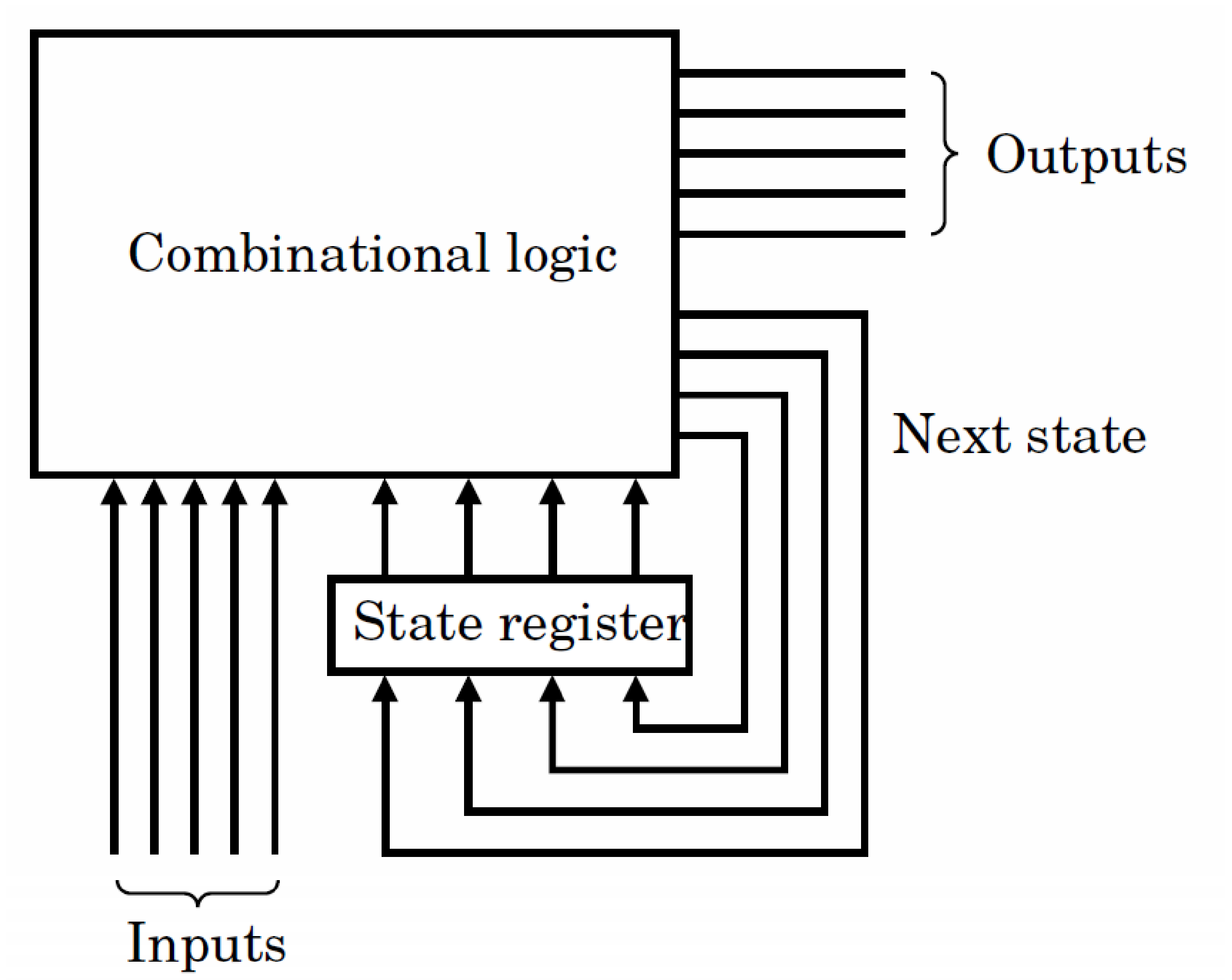


Mealy FSM



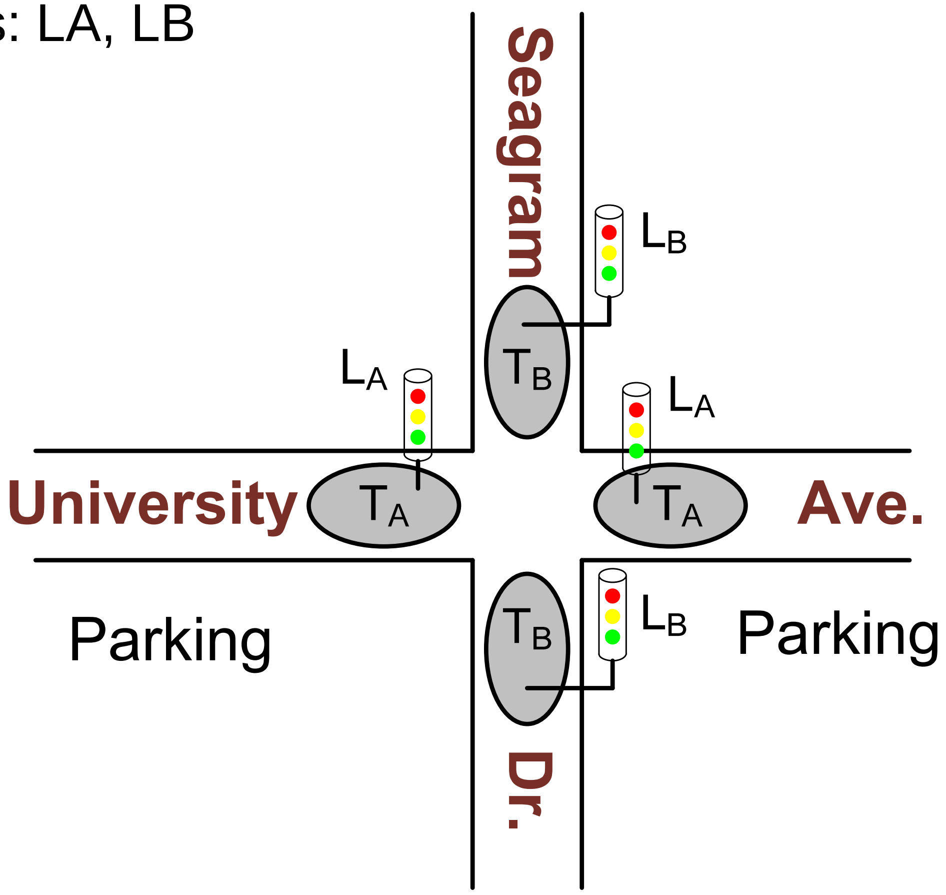
Finite State Machines /2

- **Finite-State Digital Controller:**



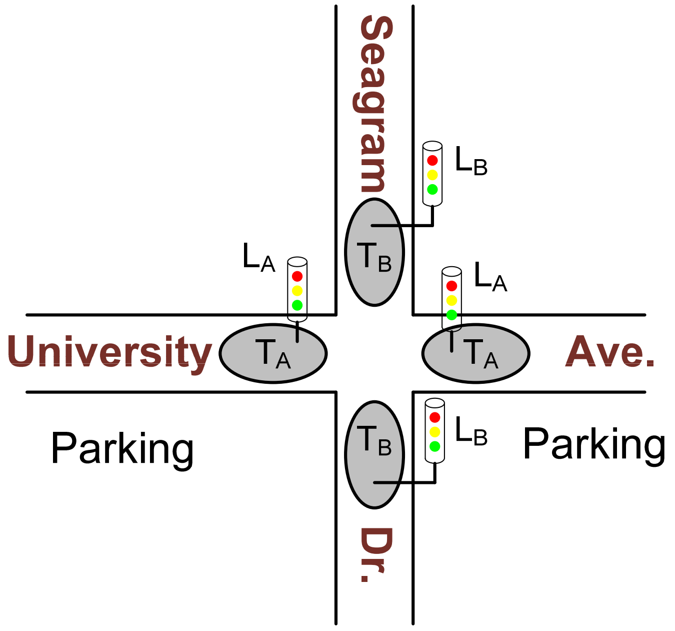
Finite State Machines /3

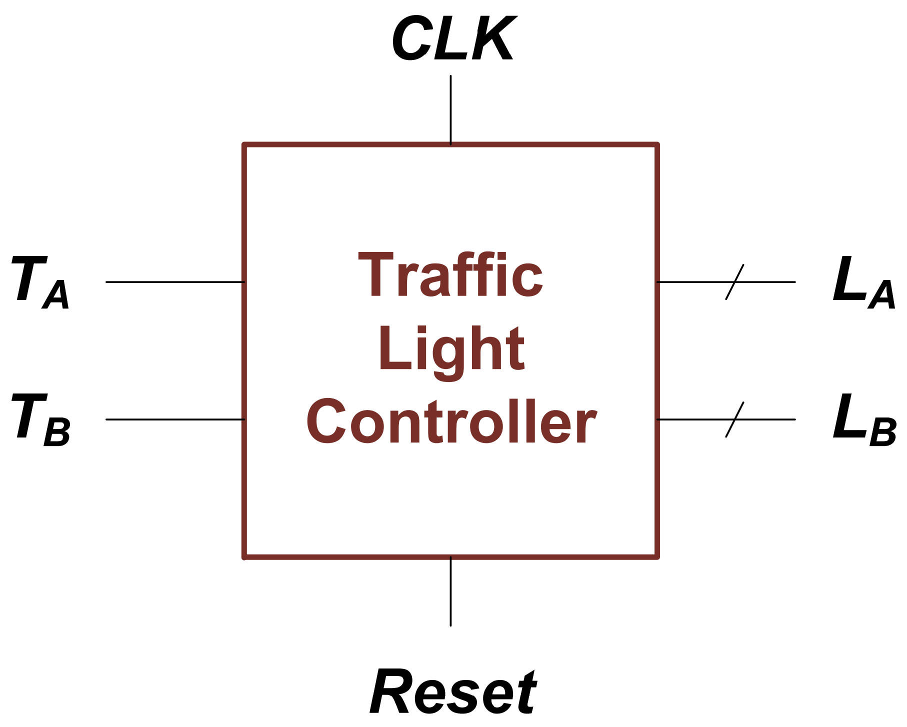
- **Traffic light problem:**
  - Traffic sensors: TA, TB (TRUE when there's traffic)
  - Lights: LA, LB



Finite State Machines /4

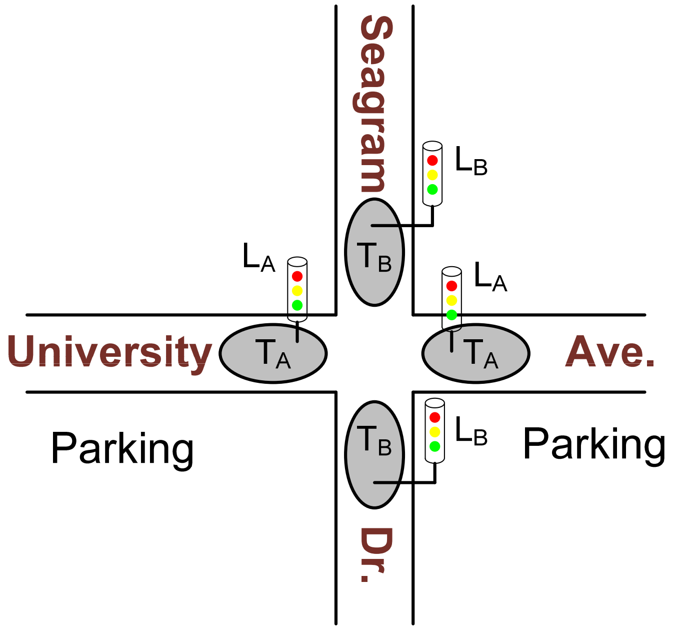
- **Traffic light controller:**
  - Inputs: CLK, Reset, TA, TB
  - Outputs: LA, LB

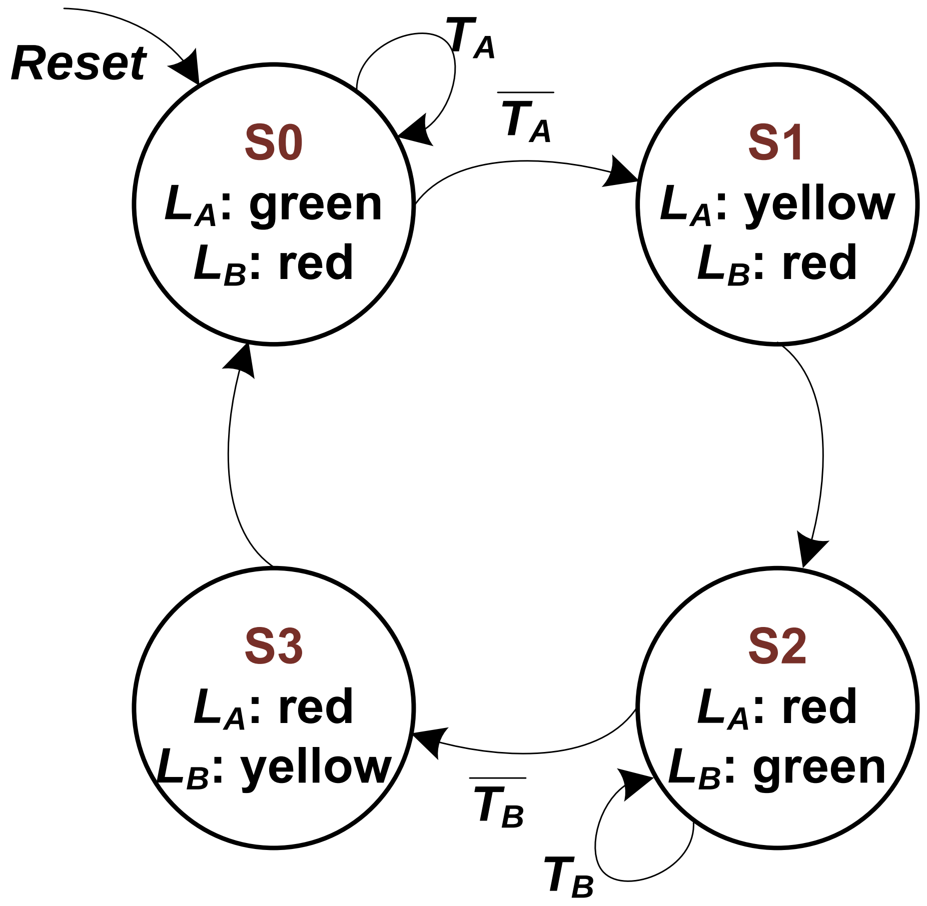




Finite State Machines /5

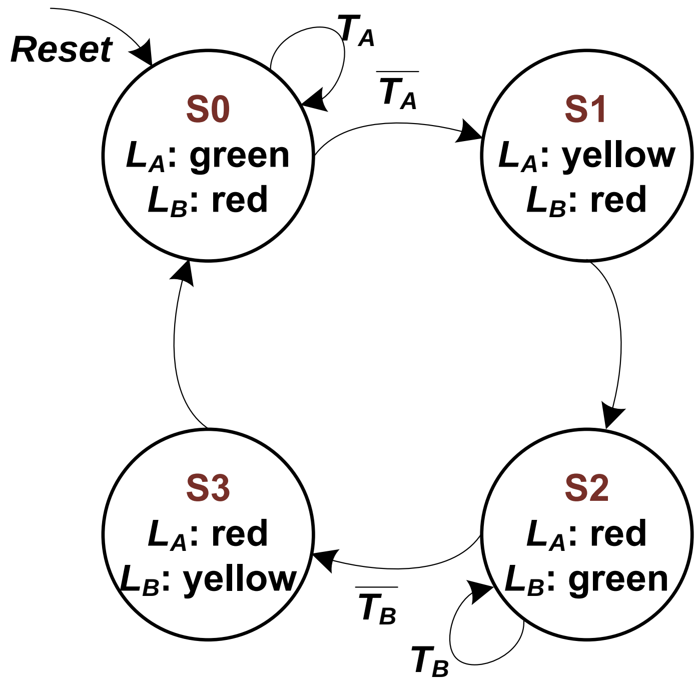
- **Traffic light FSM:**
  - Moore FSM: outputs labeled in each state
  - States: Circles
  - Transitions: Arcs





Finite State Machines /6

- **Traffic light FSM:**



Current State S	Inputs T_A	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

Finite State Machines /7

- **FSM state encoding and state transition table:**

State	Encoding
S0	00
S1	01
S2	10
S3	11

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

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

Finite State Machines /8

- **FSM output encoding and output table:**

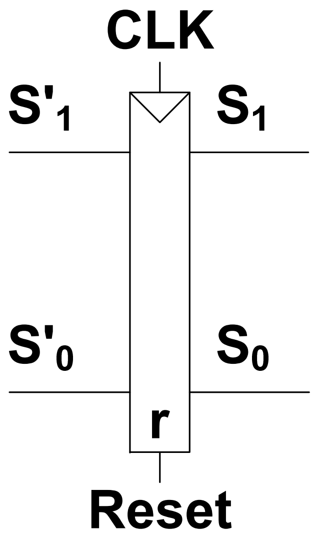
Output	Encoding
green	00
yellow	01
red	10

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

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

Finite State Machines /9

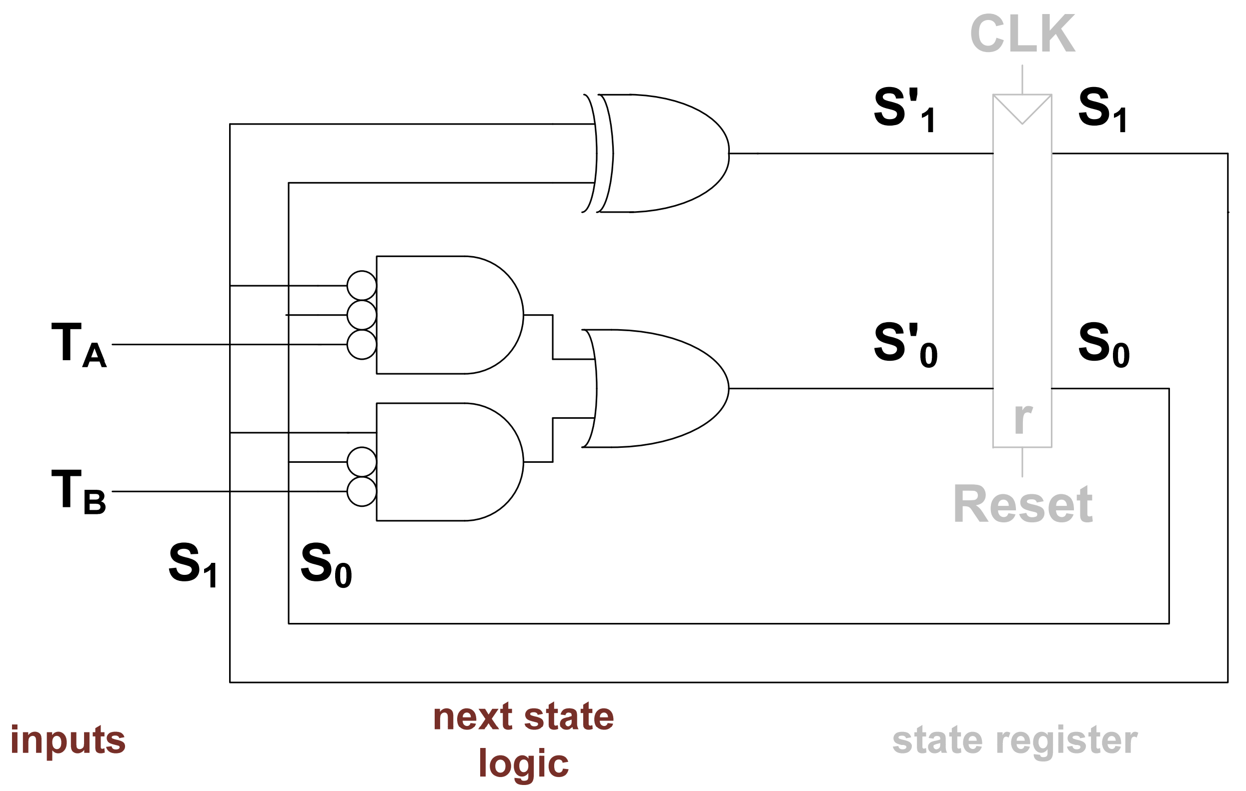
- **FSM Schematic: State Register**



state register

Finite State Machines /10

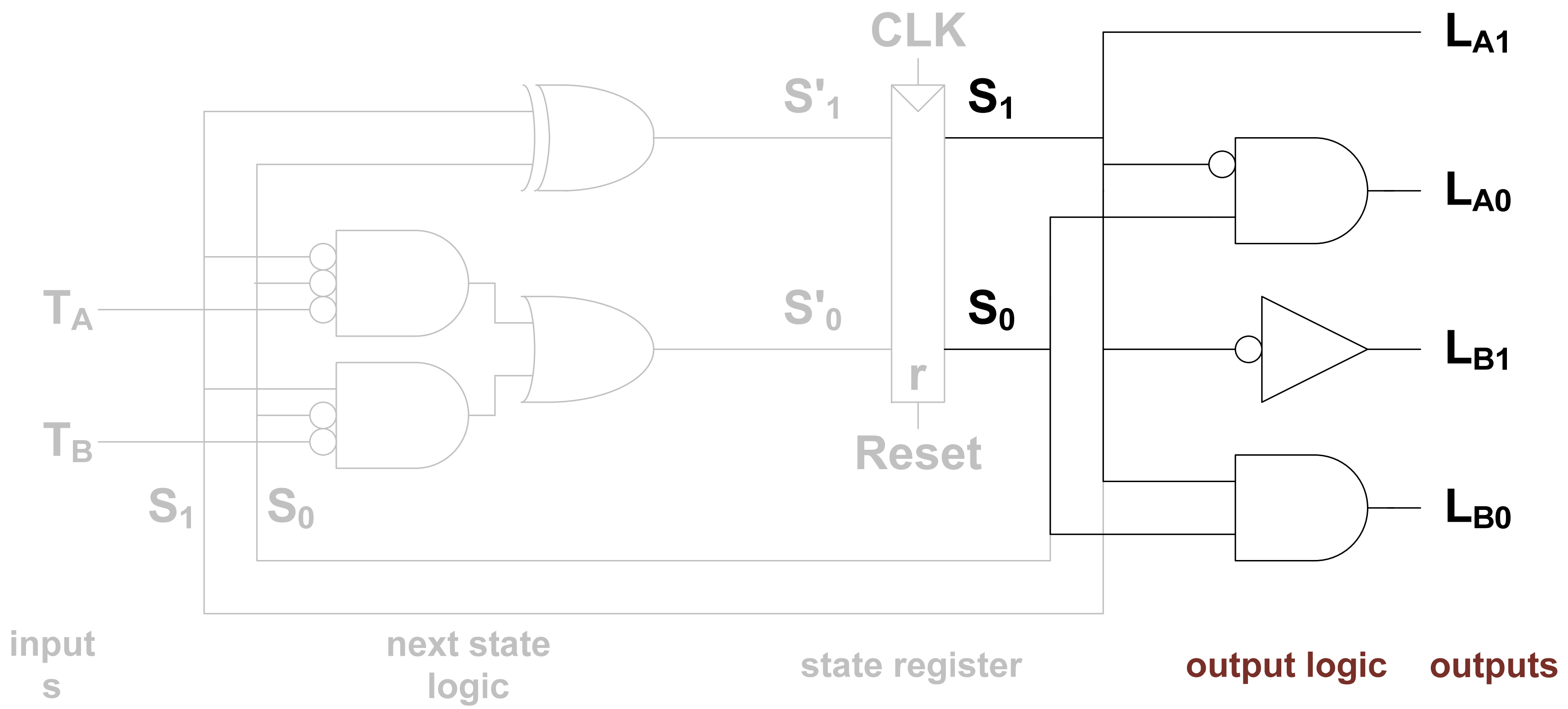
- **FSM Schematic: Next State Logic**



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

Finite State Machines /11

- **FSM Schematic: Output Logic**



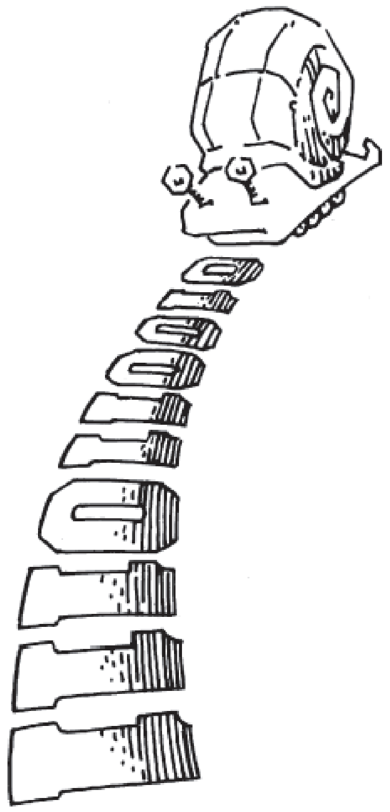
$$L_{A1} = S_1 \qquad L_{B1} = \overline{S_1}$$
$$L_{A0} = \overline{S_1}S_0 \qquad L_{B0} = S_1S_0$$

Finite State Machines /12

- **FSM binary state encoding:**
  - For four states: 00, 01, 10, 11
- **FSM one-hot encoding:**
  - One state bit per state
  - Only one state bit is true at one time
  - For four states: 0001, 0010, 0100, 1000
  - One-hot encoding can run at higher clock rate but it requires more circuitry (i.e., a flip-flop for each bit)

More on Finite State Machines /1

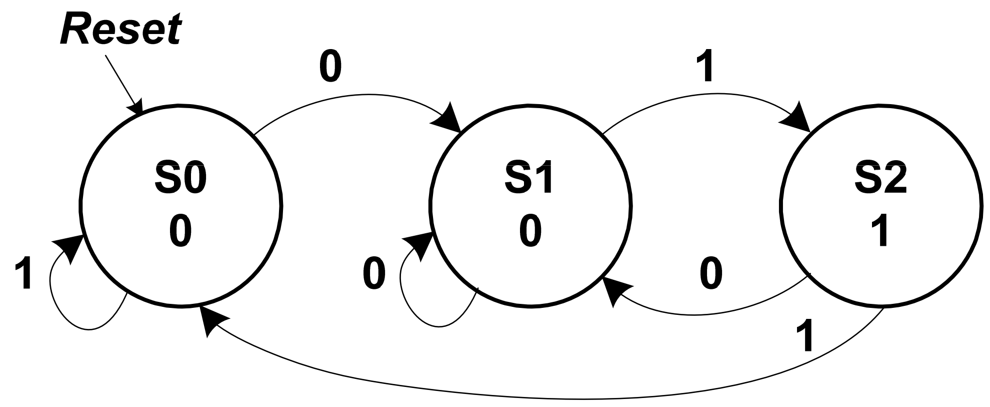
- **Snail's Brain Example:**
  - Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it
  - The snail smiles whenever the last two digits it has crawled over are 01
  - Design Moore FSM of the snail's brain



More on Finite State Machines /2

- **Snail's Brain Example:**
  - 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

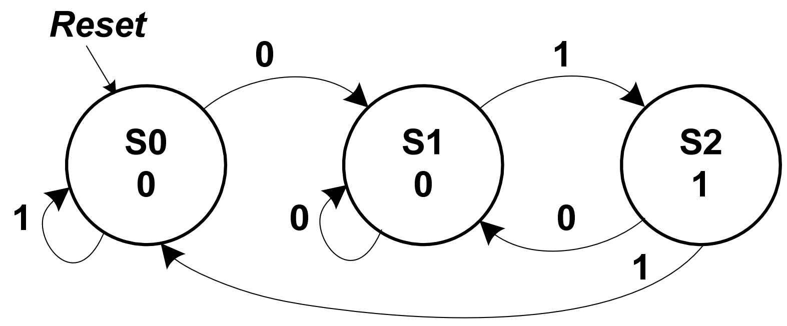
Moore FSM



More on Finite State Machines /3

- **Snail's Brain Example:**

Moore FSM



State	Encoding
S0	00
S1	01
S2	10

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

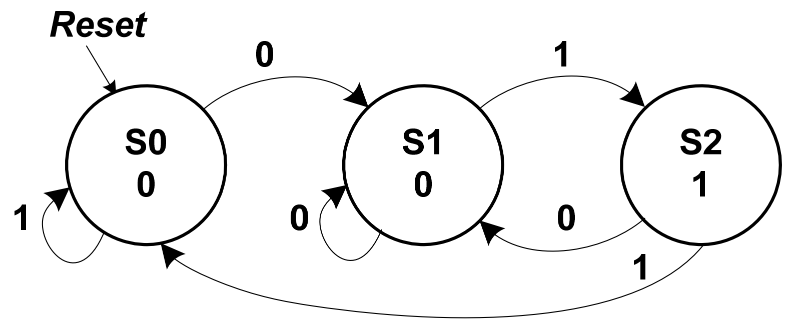
Not always true; but why here?

$$S_1' = S_0 A$$
$$S_0' = \overline{A}\,(\overline{S_1} + \overline{S_0}) = \overline{A}$$

More on Finite State Machines /4

- **Snail's Brain Example:**

Moore FSM



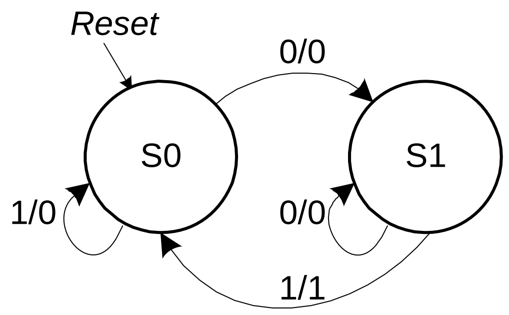
Current State		Output
S_1	S_0	Y
0	0	0
0	1	0
1	0	1

$$Y = S_1$$

More on Finite State Machines /5

- **Snail's Brain Example:**
  - Mealy FSM: arcs indicate input/output

Mealy FSM



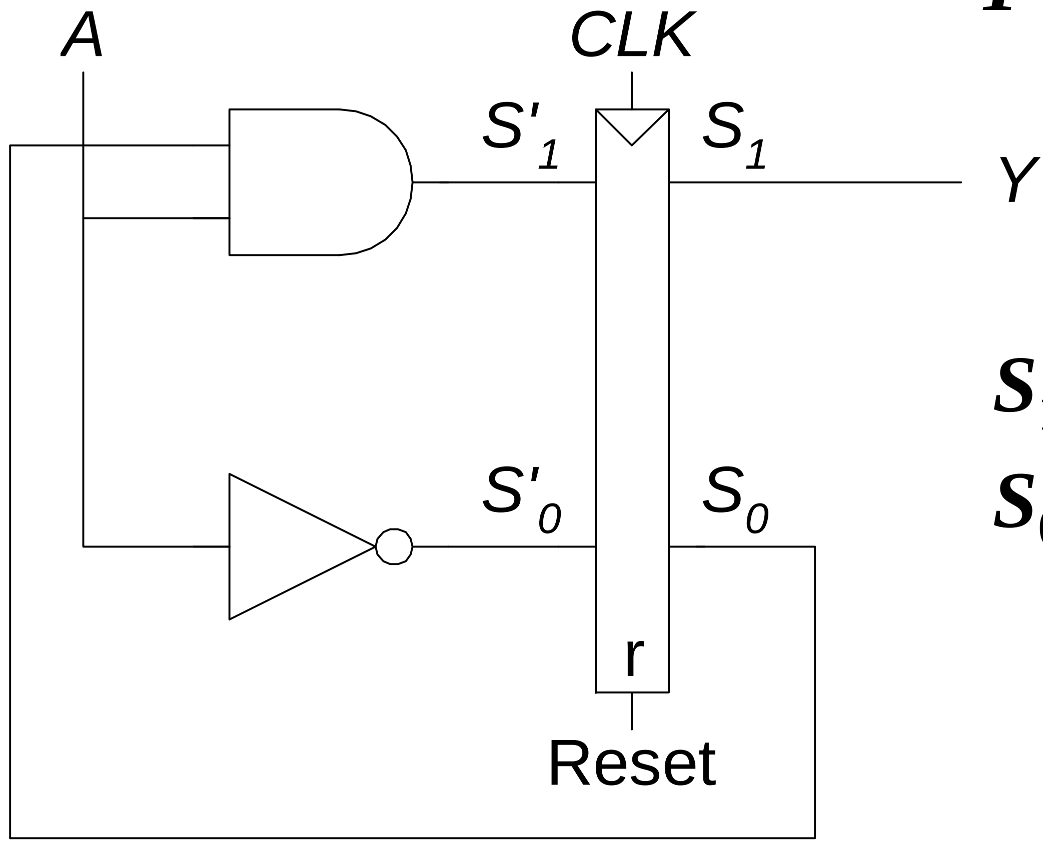
Current State	Input	Next State	Output
S_0	A	S'_0	Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

State	Encoding
S0	00
S1	01

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

More on Finite State Machines /6

- **Moore FSM Schematic:**



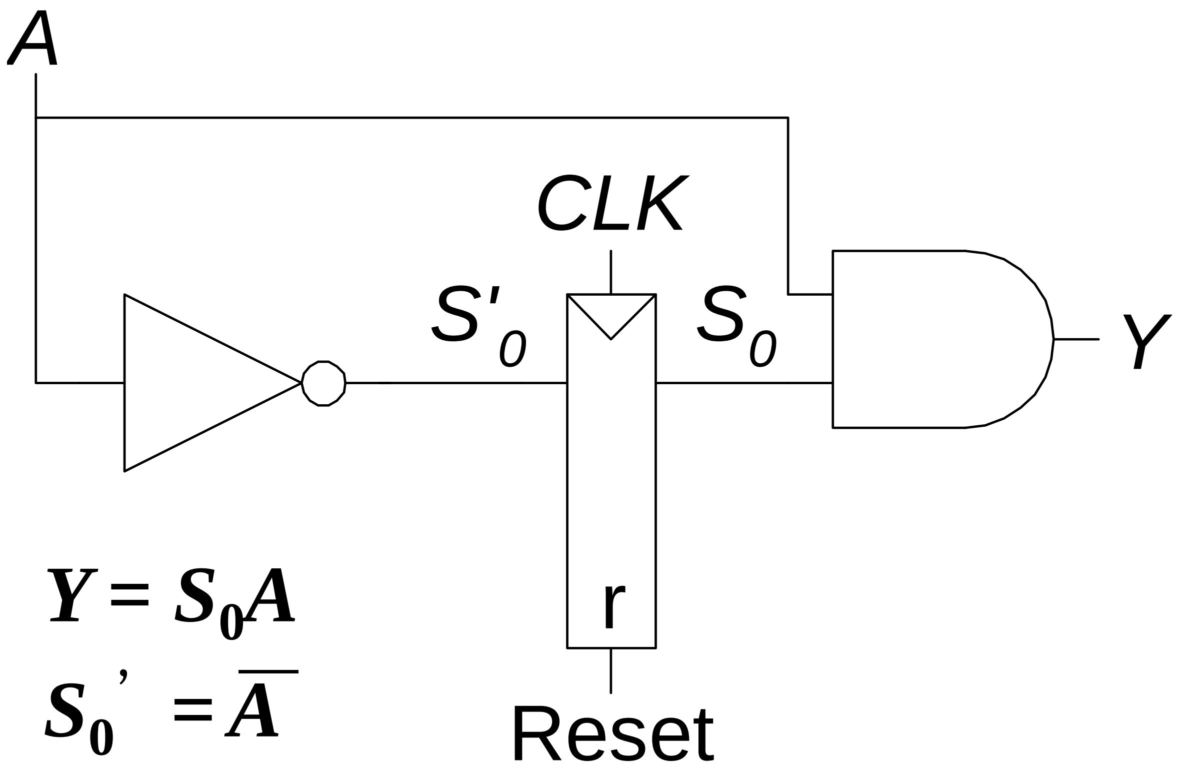
$$Y = S_1$$

$$S_1' = S_0 A$$

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

More on Finite State Machines /7

- **Mealy FSM Schematic:**



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

More on Finite State Machines /8

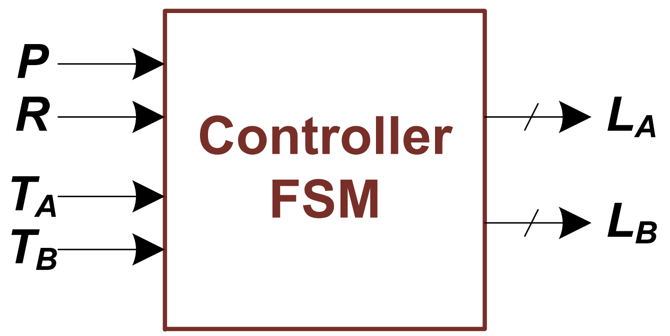
- **Designing FSMs Summary:**
  1. Identify inputs and outputs
  2. Sketch a state transition diagram
  3. Write state transition table
  4. Select state encodings
  5. Rewrite state transition table with state encodings
  6. Write output table
  7. Sketch the circuit schematic

Factoring State Machines /1

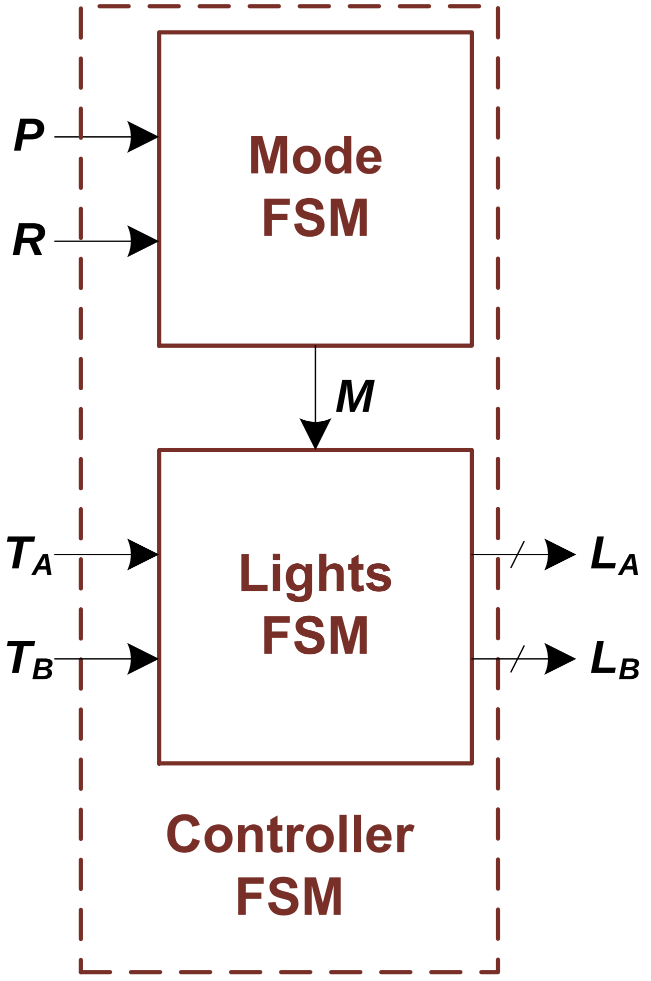
- **Factoring a FSM:**
  - Break a complex FSMs into smaller interacting FSMs
- **Example:**
  - Modify traffic light controller to add Special Mode
  - Two more inputs: P, R
  - When P = 1, enter the Special Mode and Seagram Dr. light stays green (e.g., under construction, special University events)
  - When R = 1, leave the Special Mode

Factoring State Machines /2

- **Unfactored FSM:**

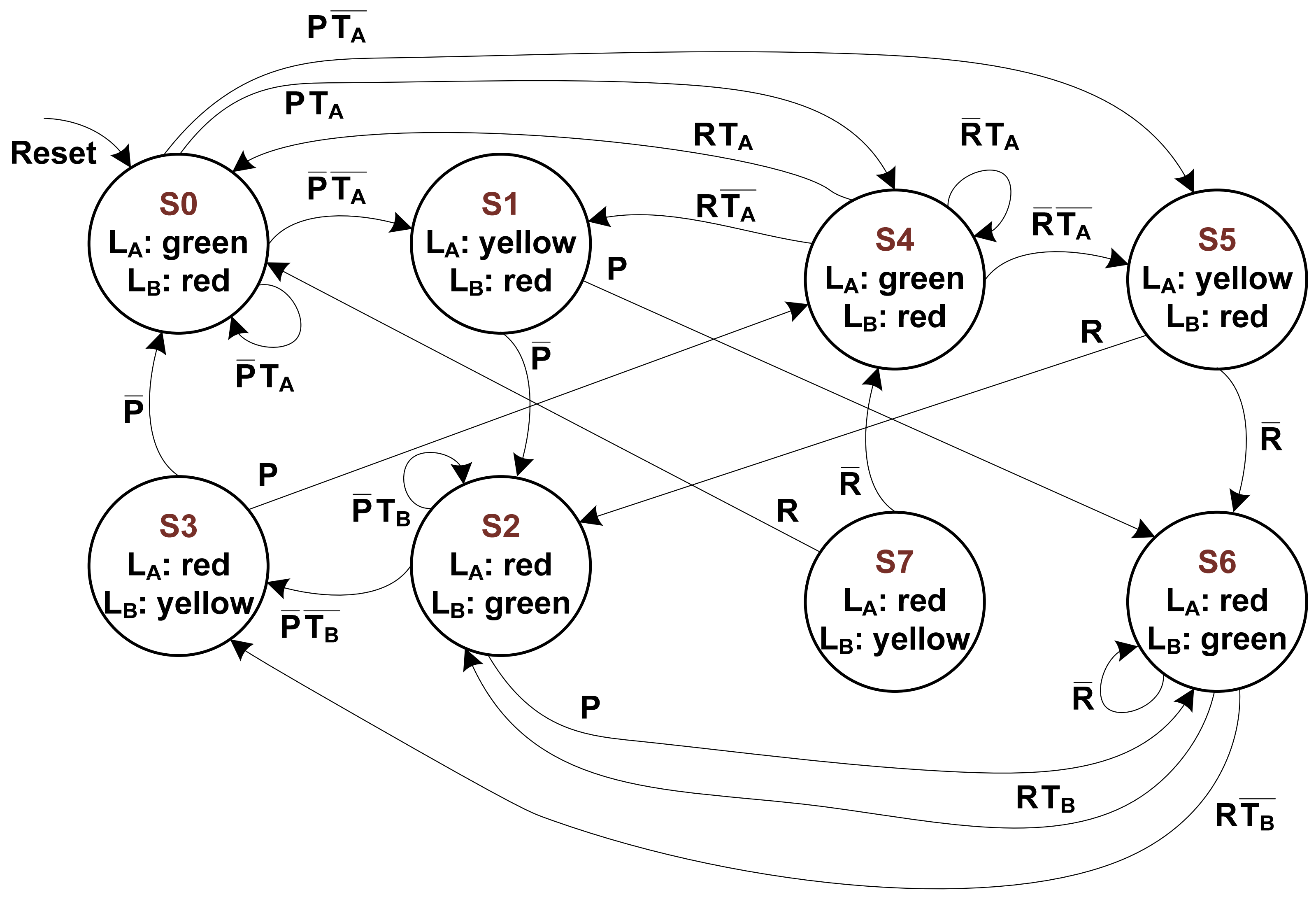


- **Factored FSM:**



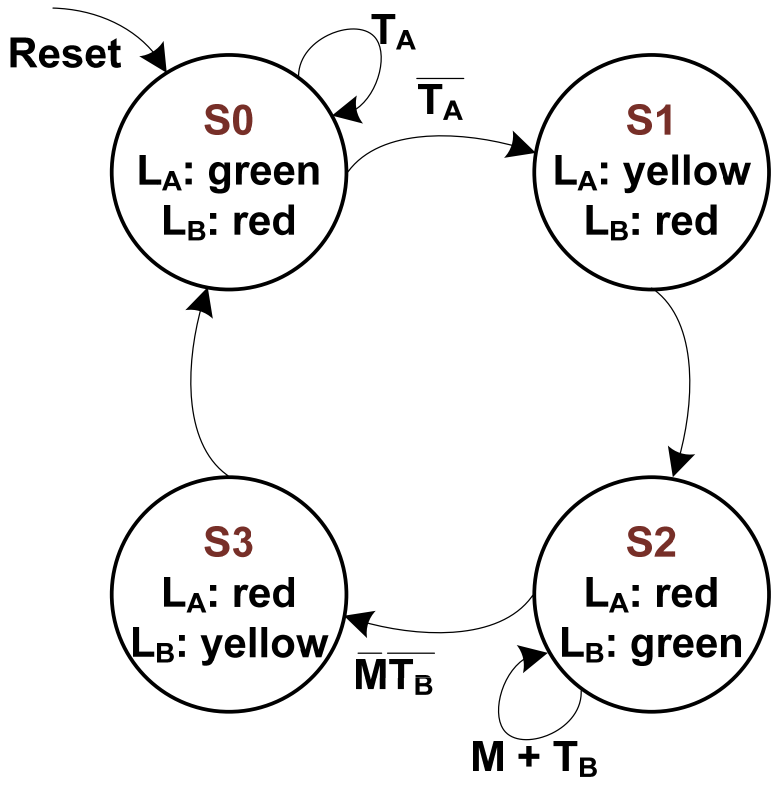
Factoring State Machines /3

- **Unfactored FSM:**

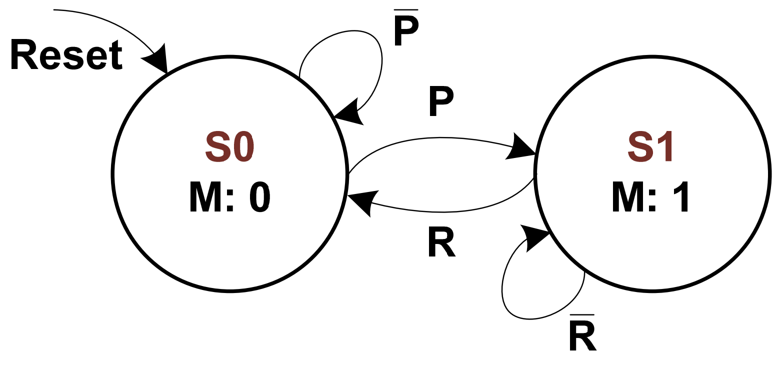


Factoring State Machines /4

- **Factored FSM:**



Lights FSM



Mode FSM

Food for Thought

- **Download and Read Assignment #1 Specifications:**
  - Assignment #1 is intended as an introduction to the combinatorial logic design
- **Read:**
  - Appendix C and Appendix D of the course textbook
    - Review the material discussed in the lecture notes in more detail
  - (Optional) Chapter 3 of the Harris and Harris textbook