

VLSI & Embedded System

SEQUENTIAL LOGIC DESIGN

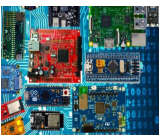
Dennis A. N. Gookyi





CONTENTS

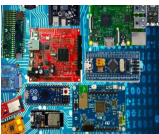
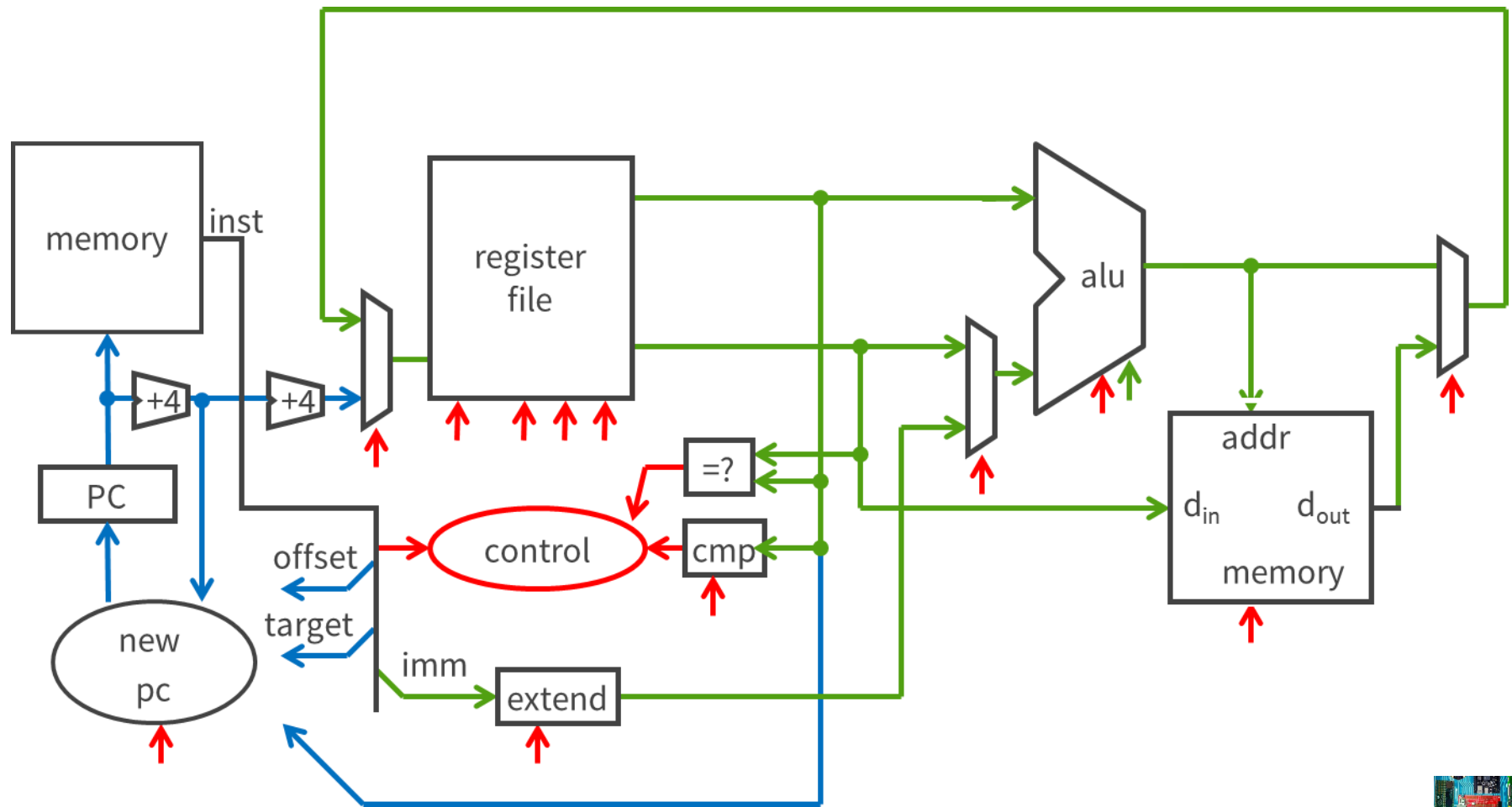
❖ Sequential Logic Design





BIG PICTURE: BUILDING A PROCESSOR

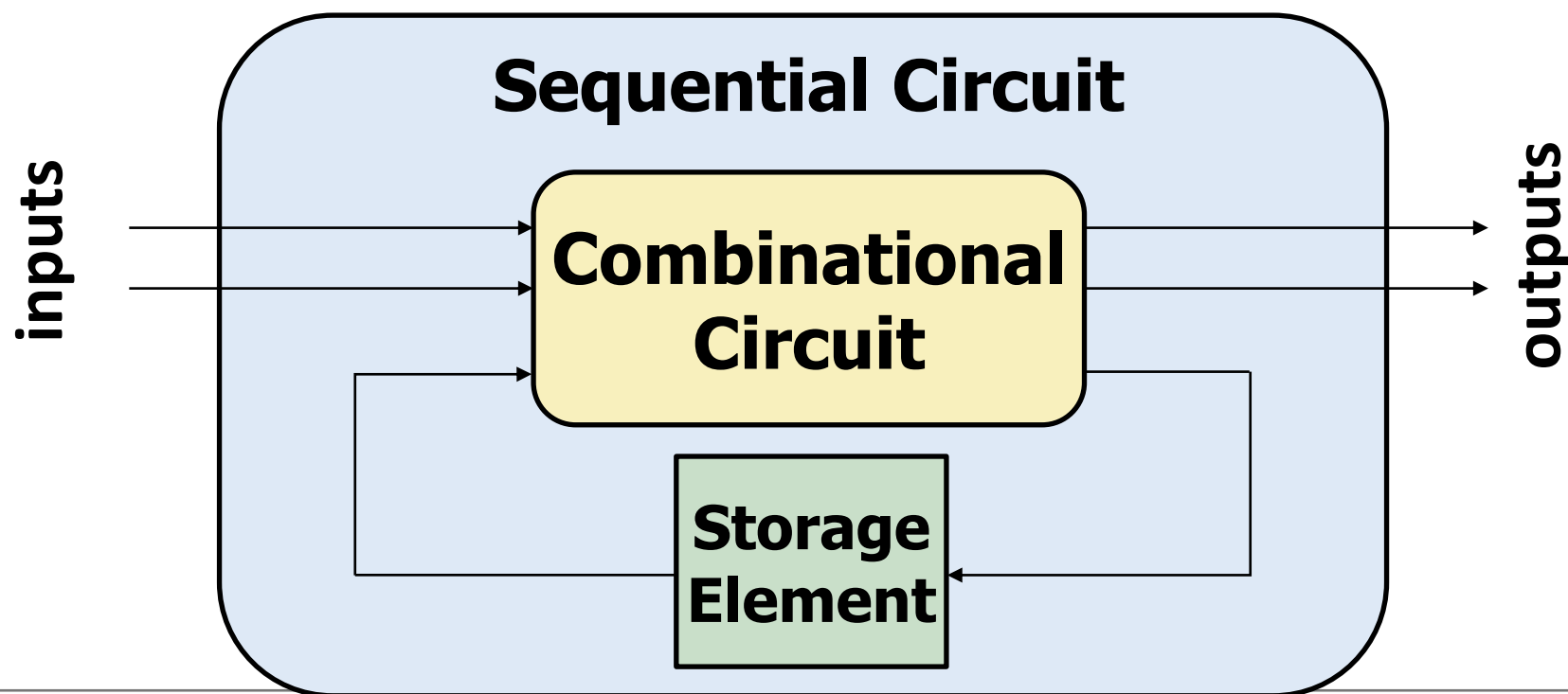
❖ Single cycle processor

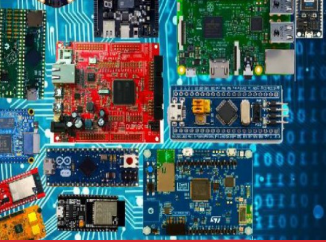




INTRODUCTION

- ❖ Combinational circuit output depends **only** on **current** input
- ❖ We want circuits that produce output depending on **current** and **past** input values – circuits with **memory**
- ❖ How can we design a circuit that **stores information**?





SEQUENTIAL LOGIC CIRCUITS

- ❖ We have examined designs of circuit elements that can store information
- ❖ Now, we will use these elements to build circuits that remember past inputs



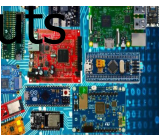
Combinational

Only depends on current inputs



Sequential

Opens depending on past inputs





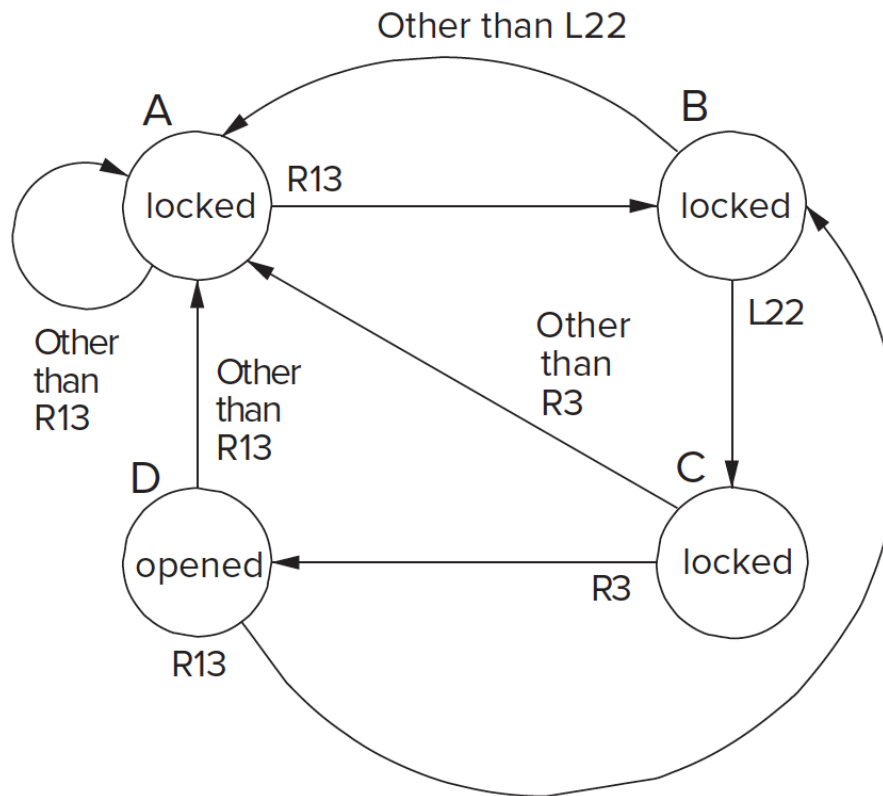
STATE

- ❖ In order for this lock to work, it has to keep track (**remember**) of the past events!
- ❖ If passcode is R13-L22-R3, sequence of **states** to unlock:
 - A. The lock is not open (locked), and no relevant operations have been performed
 - B. Locked but user has completed R13
 - C. Locked but user has completed R13-L22
 - D. Unlocked: user has completed R13-L22-R3
- ❖ The **state** of a system is a snapshot of all relevant elements of the system at the moment of the snapshot
 - To open the lock, **states A-D must be completed in order**
 - If anything else happens (e.g., L5), lock **returns** to state A



STATE DIAGRAM

- ❖ Completely describes the operation of the sequential lock



Passcode: **R13-L22-R3**

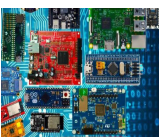
- ❖ We will understand “state diagrams” fully later today



STATE DIAGRAM

❖ Asynchronous vs. Synchronous State Changes

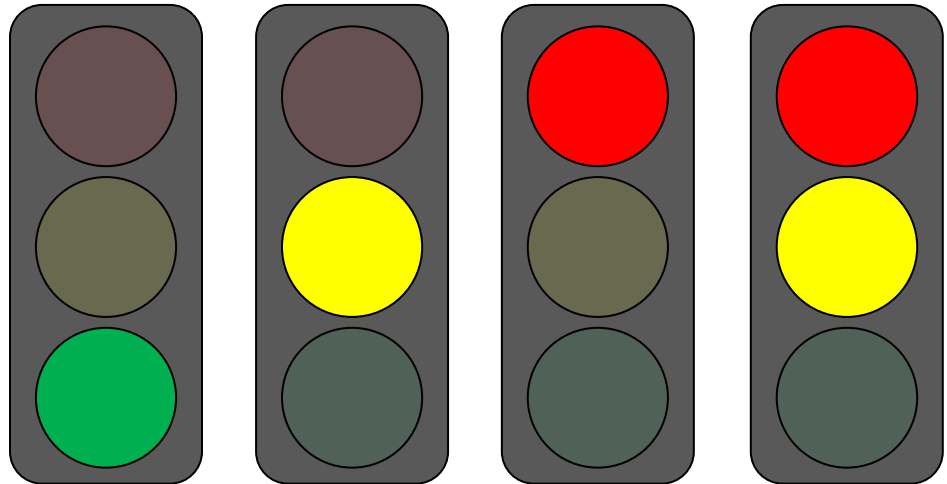
- Sequential lock we saw is an asynchronous “machine”
 - State transitions occur when they occur
 - There is nothing that synchronizes when each state transition must occur
- Most modern computers are synchronous “machines”
 - State transitions take place after fixed units of time
 - Controlled in part by a clock, as we will see soon
- These are two different design paradigms, with tradeoffs



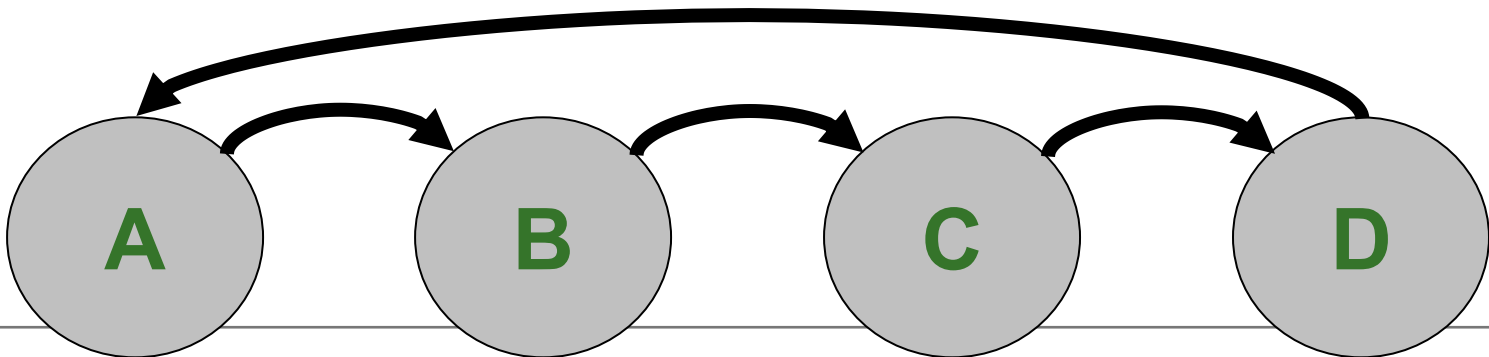
STATE DIAGRAM

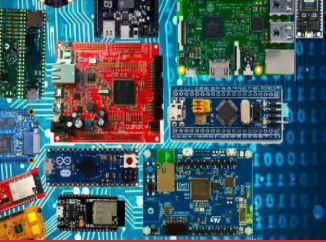
❖ A standard Swiss traffic light has 4 states

- A. Green
- B. Yellow
- C. Red
- D. Red and Yellow

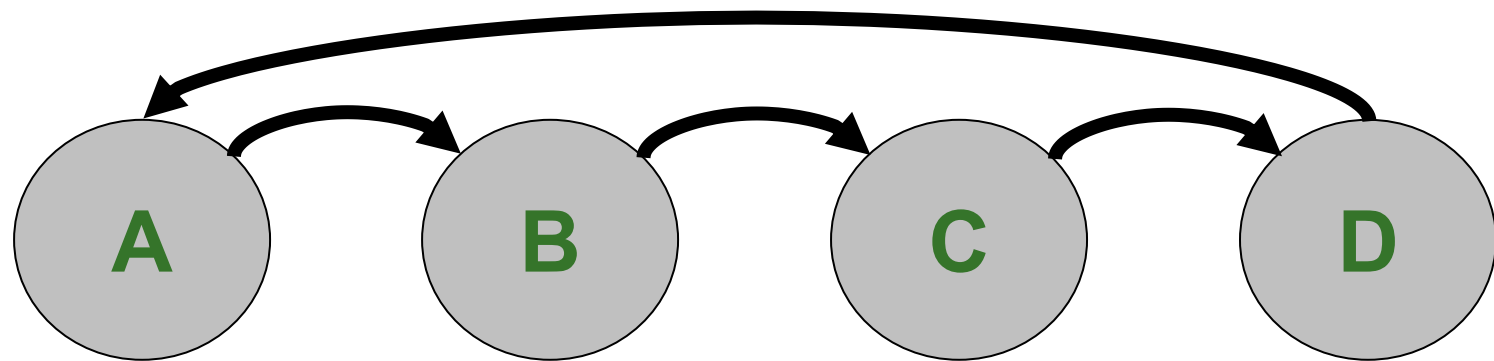


❖ The sequence of these states are always as follows





STATE DIAGRAM - CLOCK



- ❖ When should the light change from one state to another?
- ❖ We need a **clock** to dictate when to change state
 - Clock signal alternates between 0 & 1

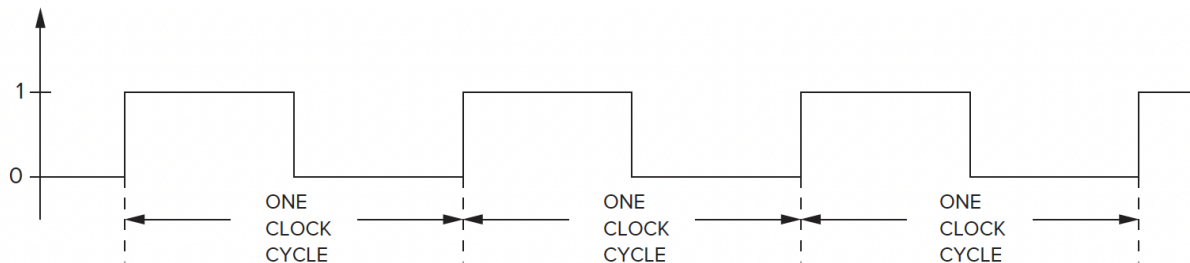
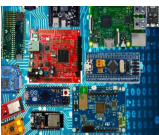
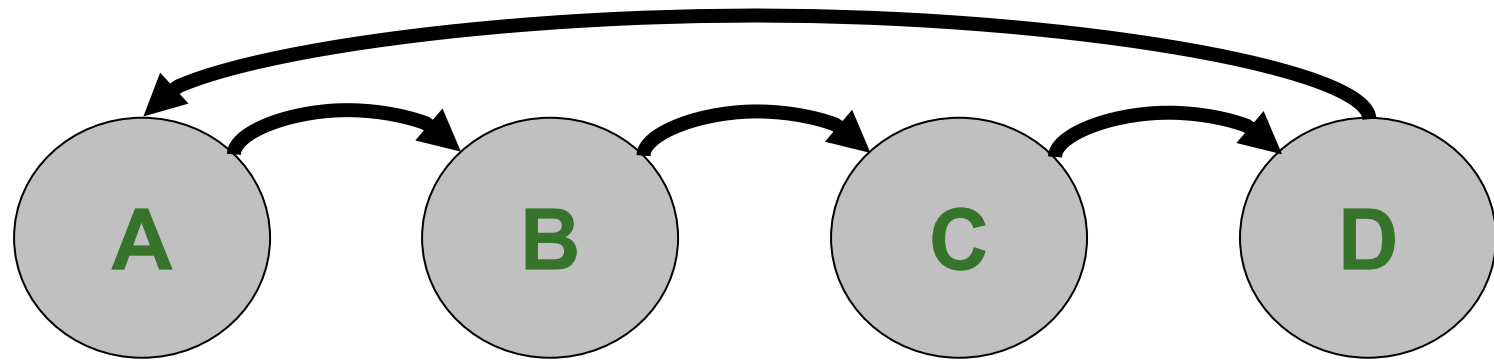


Figure 3.28 A clock signal.





STATE DIAGRAM - CLOCK

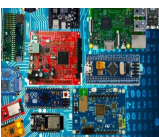


- ❖ When should the light change from one state to another?
- ❖ We need a **clock** to dictate when to change state

- ☐ Clock signal alternates between 0 & 1



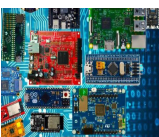
- ❖ At the start of a clock cycle (), system state changes
 - ☐ During a clock cycle, the state stays constant
 - ☐ In this traffic light example, we are assuming the traffic light stays in each state an equal amount of time





STATE DIAGRAM - CLOCK

- ❖ Clock is a general mechanism that triggers transition from one state to another in a (synchronous) sequential circuit
- ❖ Clock synchronizes state changes across many sequential circuit elements
- ❖ Combinational logic evaluates for the length of the clock cycle
- ❖ Clock cycle should be chosen to accommodate maximum combinational circuit delay

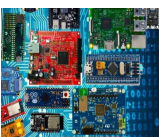




STATE DIAGRAM

❖ Asynchronous vs. Synchronous State Changes

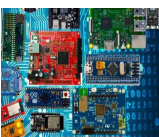
- Sequential lock we saw is an asynchronous “machine”
 - State transitions occur when they occur
 - There is nothing that synchronizes when each state transition must occur
- Most modern computers are synchronous “machines”
 - State transitions take place after fixed units of time
 - Controlled in part by a clock, as we will see soon
- These are two different design paradigms, with tradeoffs
 - Synchronous control can be easier to get correct when the system consists of many components and many states
 - Asynchronous control can be more efficient (no clock overheads)

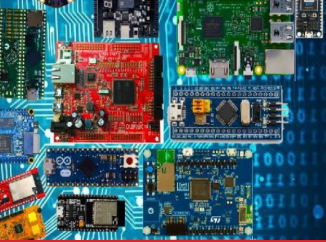




FINITE STATE MACHINE (FSM)

- ❖ What is a Finite State Machine (FSM)?
 - A discrete-time model of a stateful system
 - Each state represents a snapshot of the system at a given time
- ❖ An FSM pictorially shows
 - The set of all possible states that a system can be in
 - How the system transitions from one state to another
- ❖ An FSM can model
 - A traffic light, an elevator, fan speed, a microprocessor, etc.
- ❖ An FSM enables us to pictorially think of a stateful system using simple diagrams

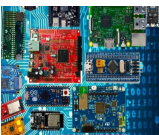




FINITE STATE MACHINE (FSM)

❖ FSM consist of five elements:

- A finite number of states
 - State: snapshot of all relevant elements of the system at the time of the snapshot
- A finite number of external inputs
- A finite number of external outputs
- An explicit specification of all state transitions
 - How to get from one state to another
- An explicit specification of what determines each external output value

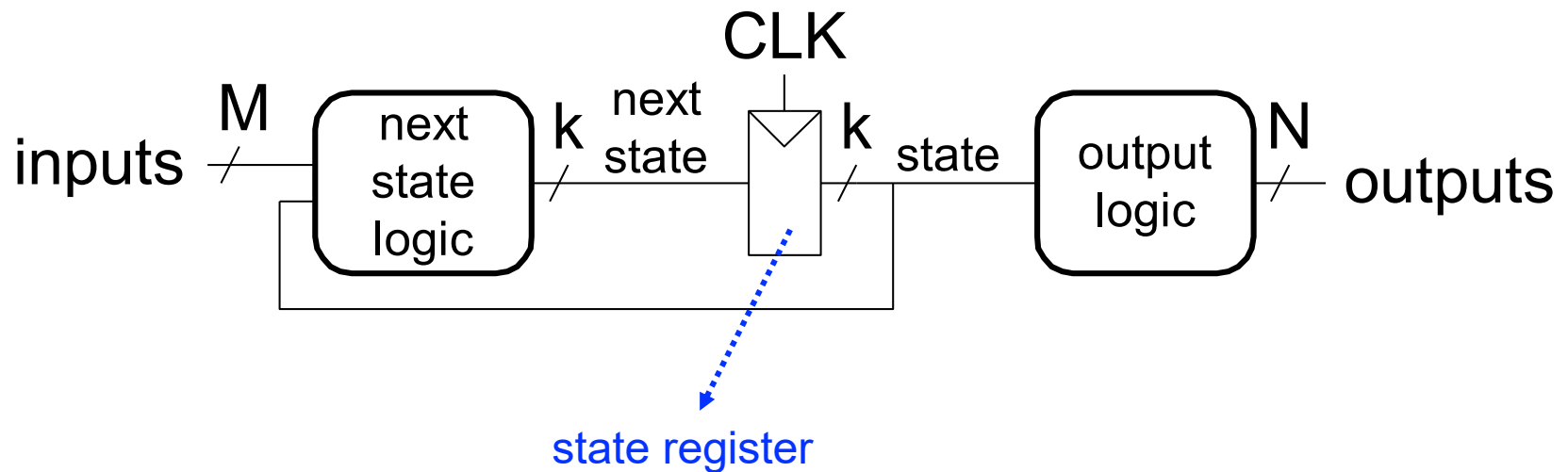




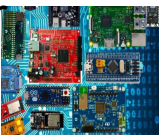
FINITE STATE MACHINE (FSM)

❖ Each FSM consists of three separate parts:

- next state logic
- state register
- output logic



At the beginning of the clock cycle, next state is latched into the state register

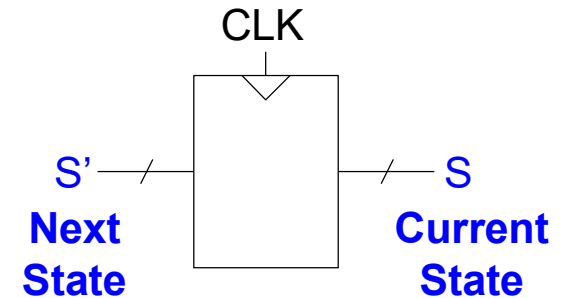




FINITE STATE MACHINE (FSM)

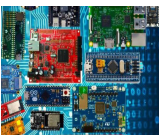
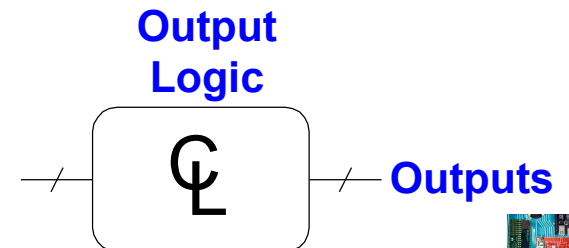
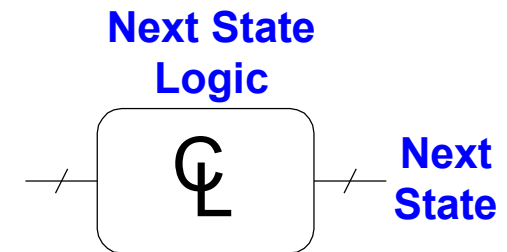
❖ Sequential Circuits

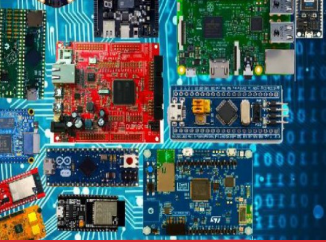
- State register(s)
 - Store the current state and
 - Load the next state at the clock edge



❖ Combinational Circuits

- Next state logic
 - Determines what the next state will be
- Output logic
 - Generates the outputs

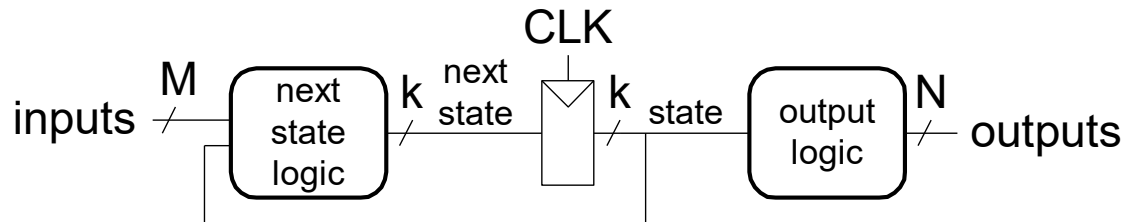




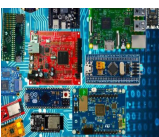
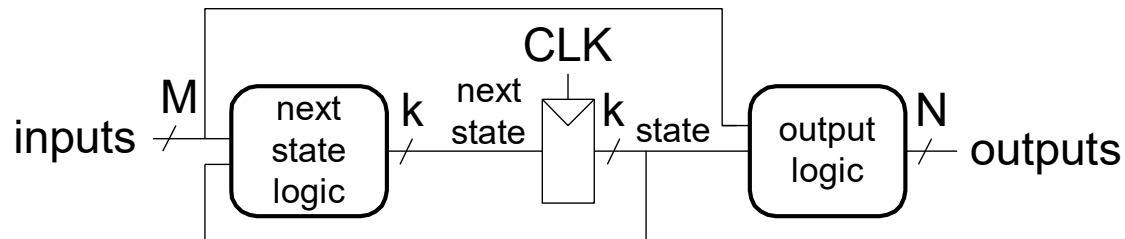
FINITE STATE MACHINE (FSM)

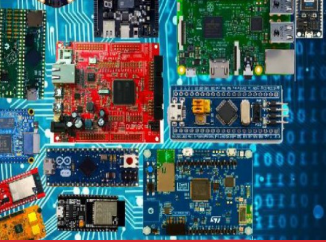
- ❖ Next state is determined by the current state and the inputs
- ❖ Two types of finite state machines differ in the output logic:
 - Moore FSM: outputs depend only on the current state
 - Mealy FSM: outputs depend on the current state and the inputs

Moore FSM



Mealy FSM

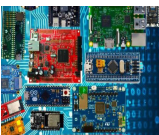
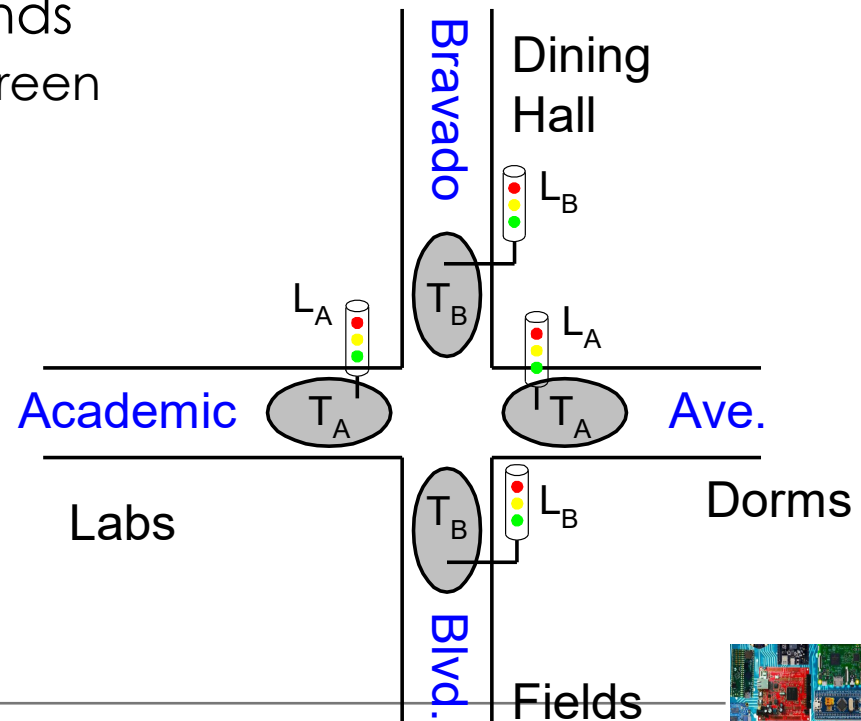




FINITE STATE MACHINE (FSM)

❖ Example: “Smart” traffic light controller

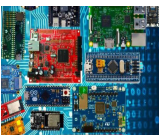
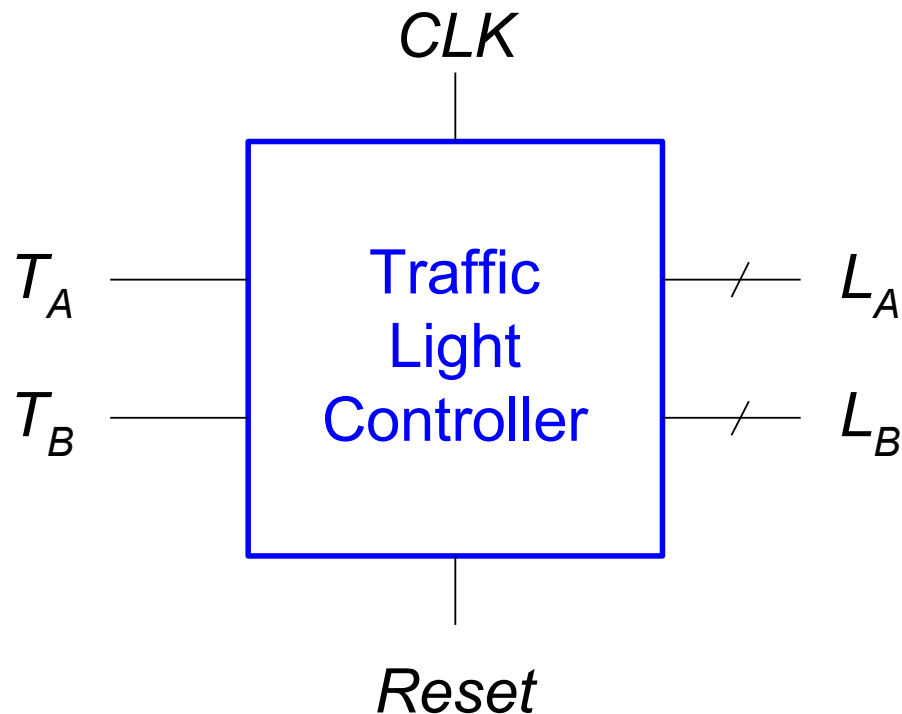
- 2 inputs:
 - Traffic sensors: T_A , T_B (TRUE when there's traffic)
- 2 outputs:
 - Lights: L_A , L_B (Red, Yellow, Green)
- State can change every 5 seconds
 - Except if green and traffic, stay green

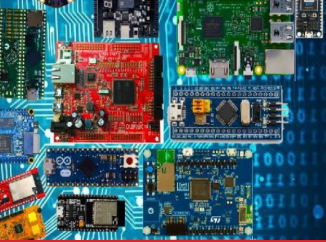




FINITE STATE MACHINE (FSM)

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

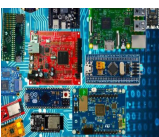
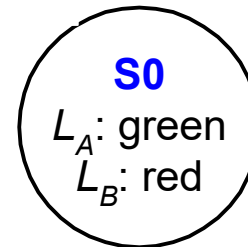
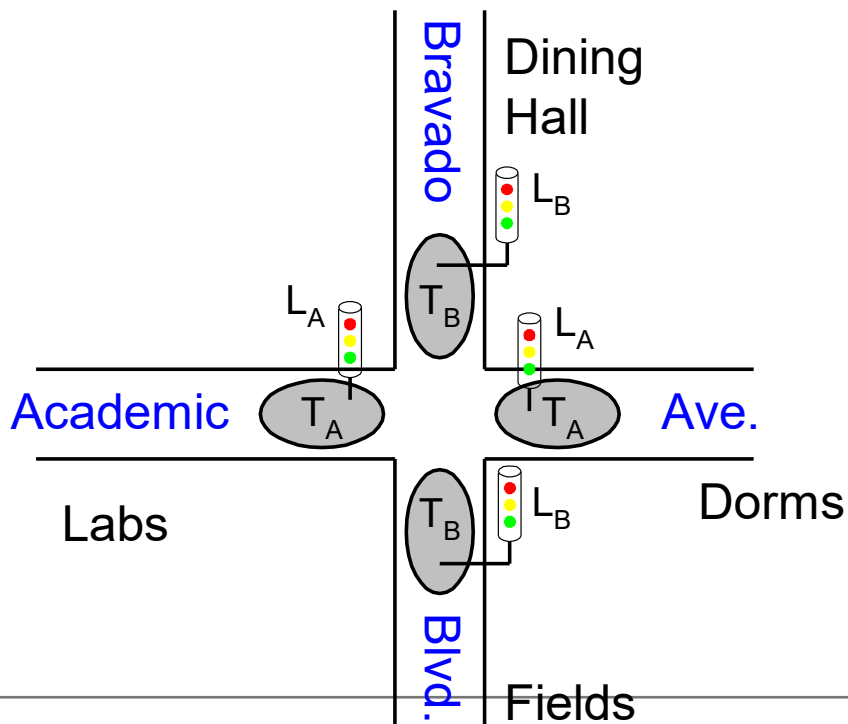




FINITE STATE MACHINE (FSM)

❖ Moore FSM: outputs labeled in each state

- States: Circles
- Transitions: Arcs

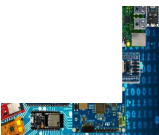
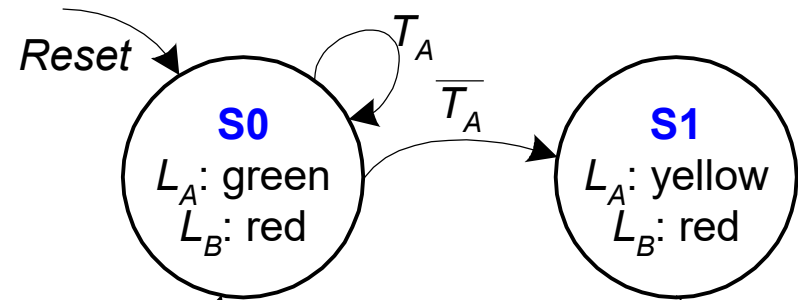
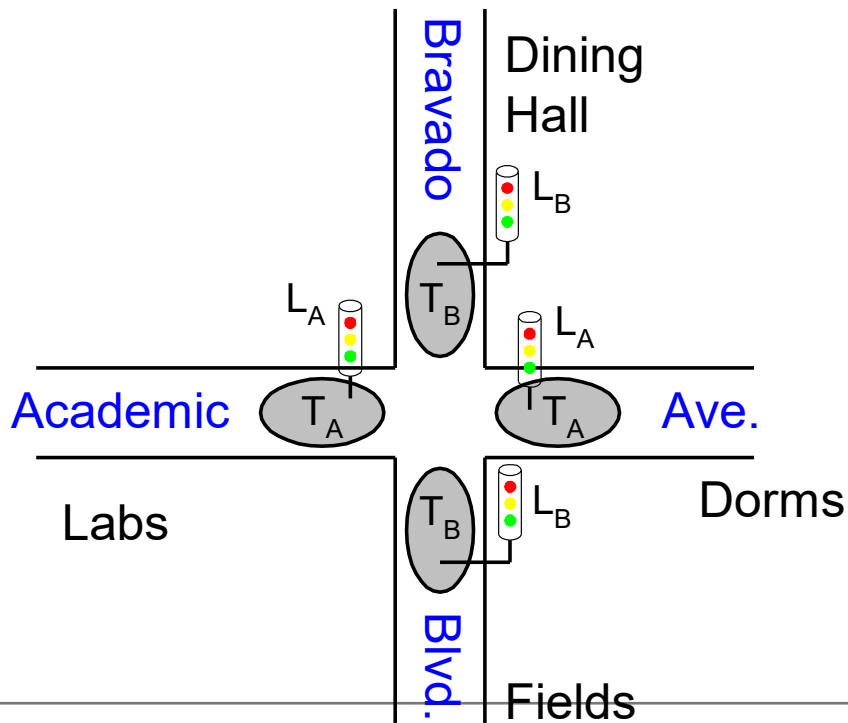




FINITE STATE MACHINE (FSM)

❖ Moore FSM: outputs labeled in each state

- States: Circles
- Transitions: Arcs

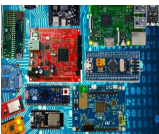
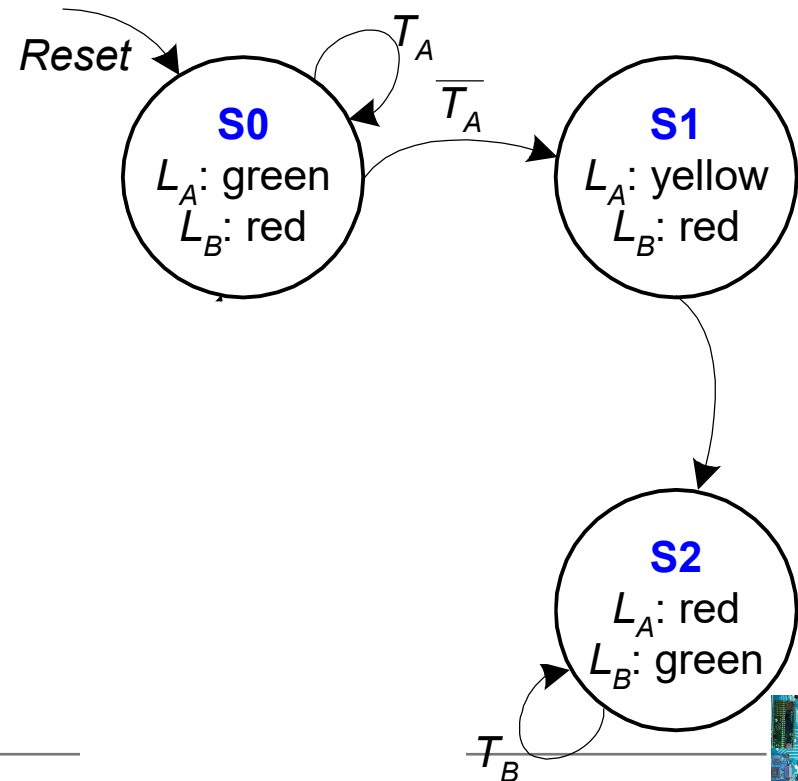
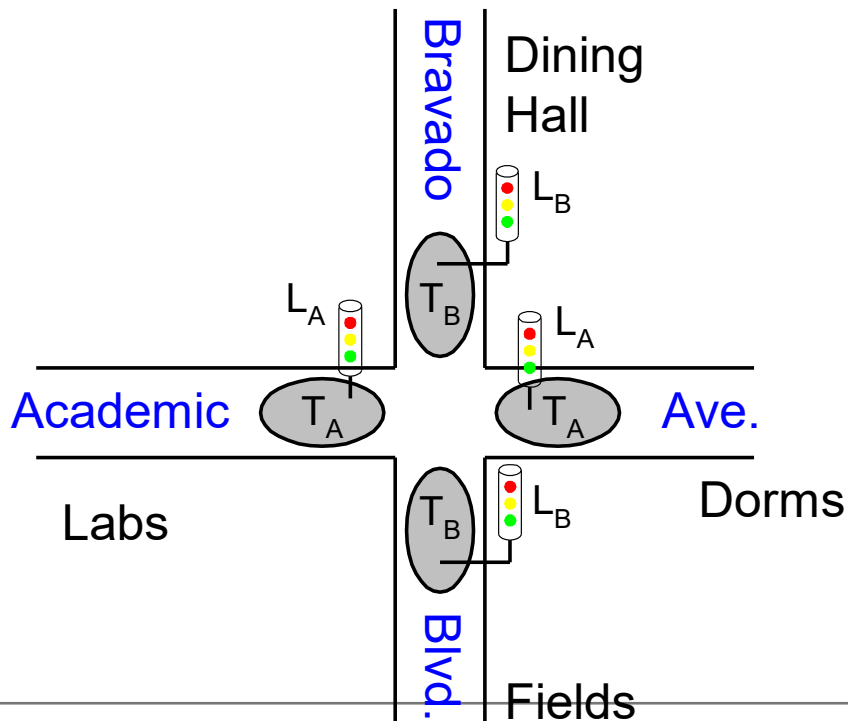


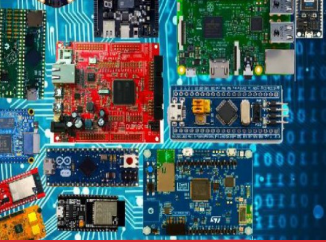


FINITE STATE MACHINE (FSM)

❖ Moore FSM: outputs labeled in each state

- States: Circles
- Transitions: Arcs

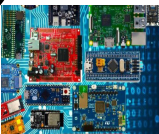
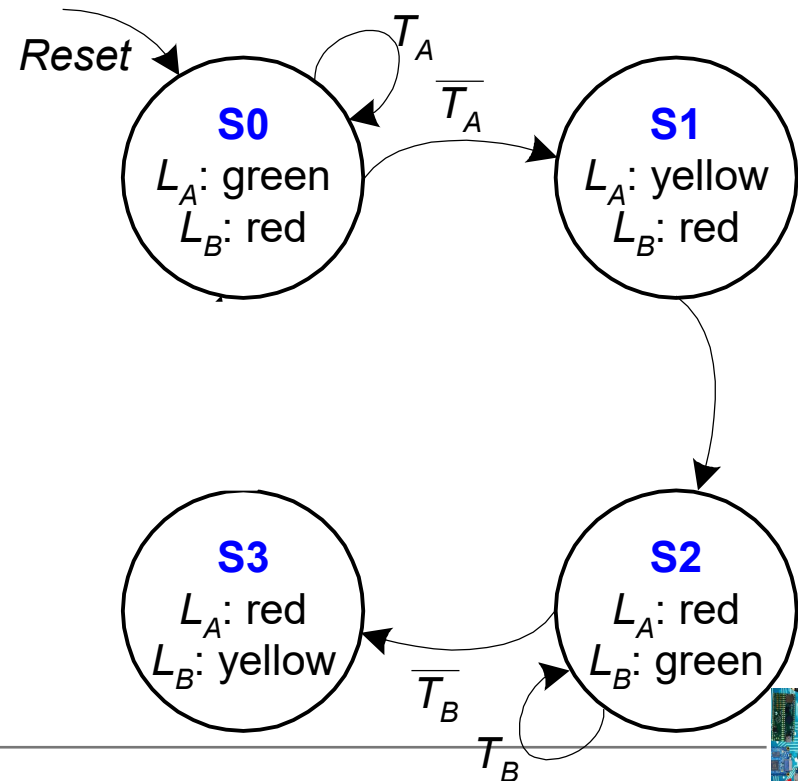
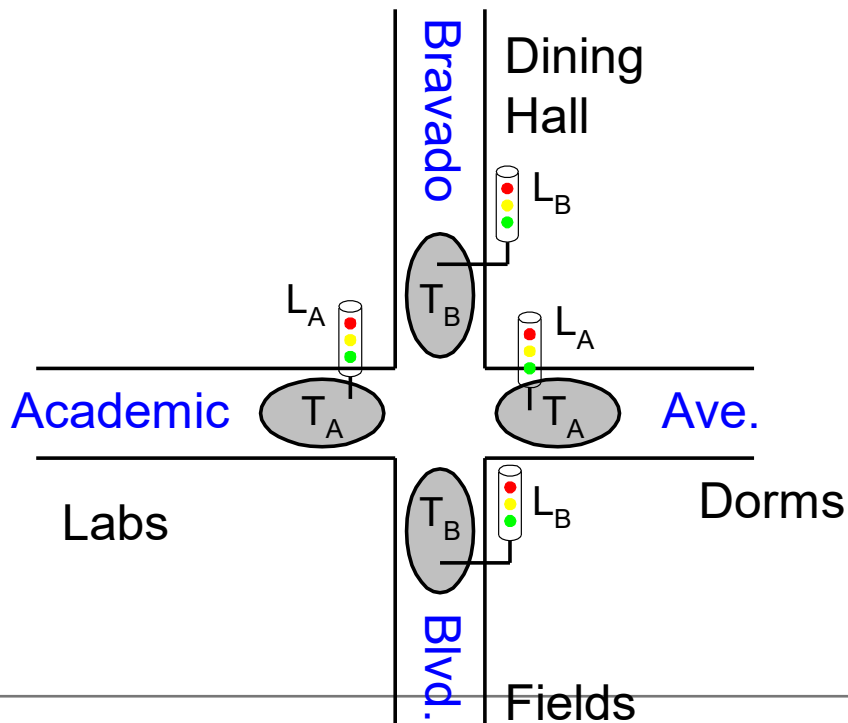


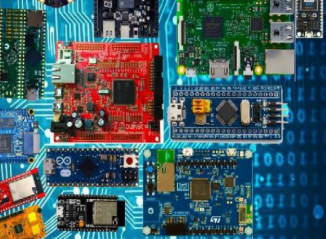


FINITE STATE MACHINE (FSM)

❖ Moore FSM: outputs labeled in each state

- States: Circles
- Transitions: Arcs

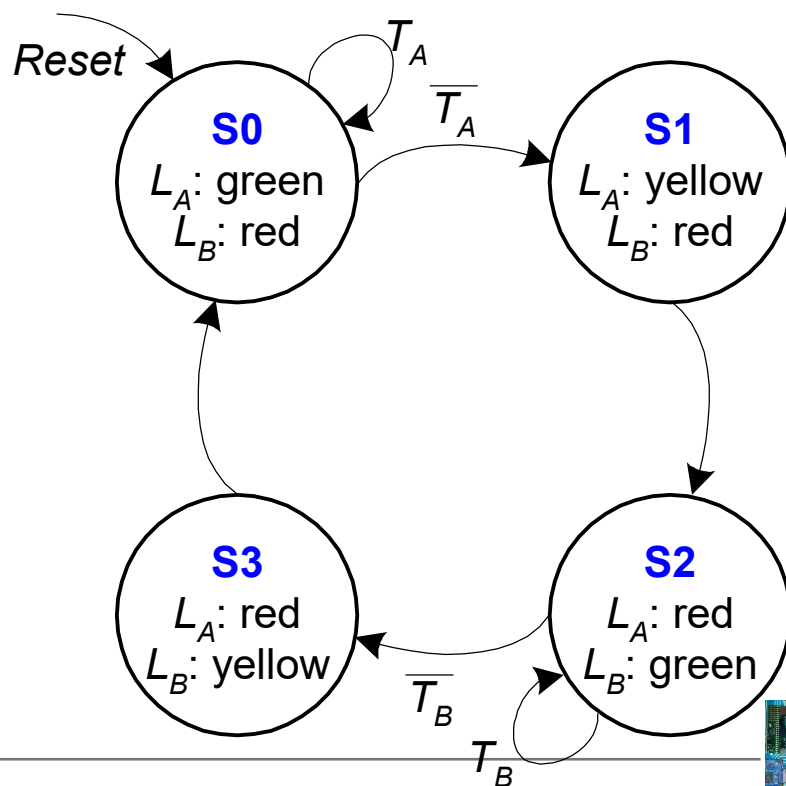
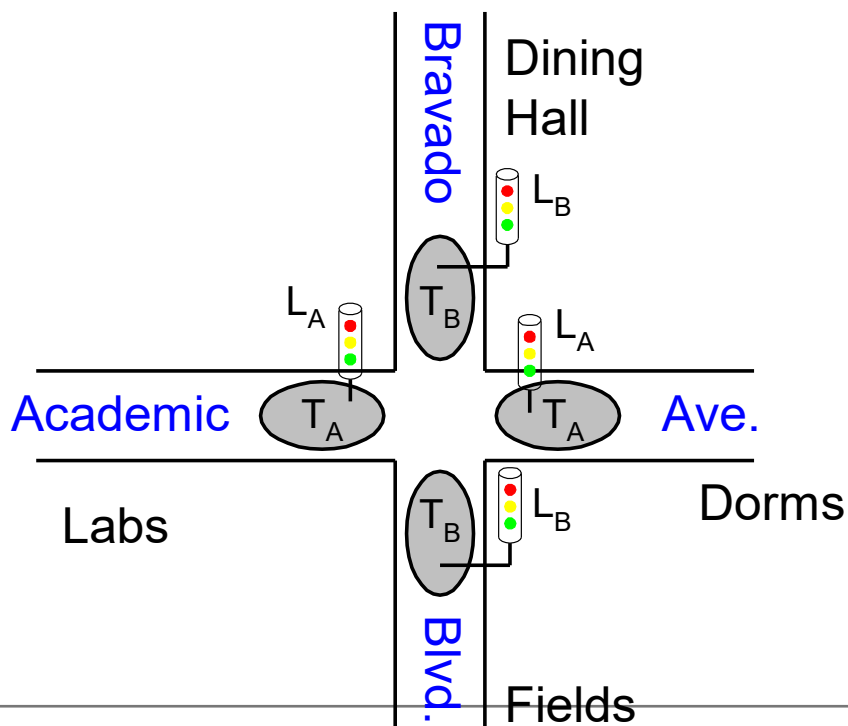




FINITE STATE MACHINE (FSM)

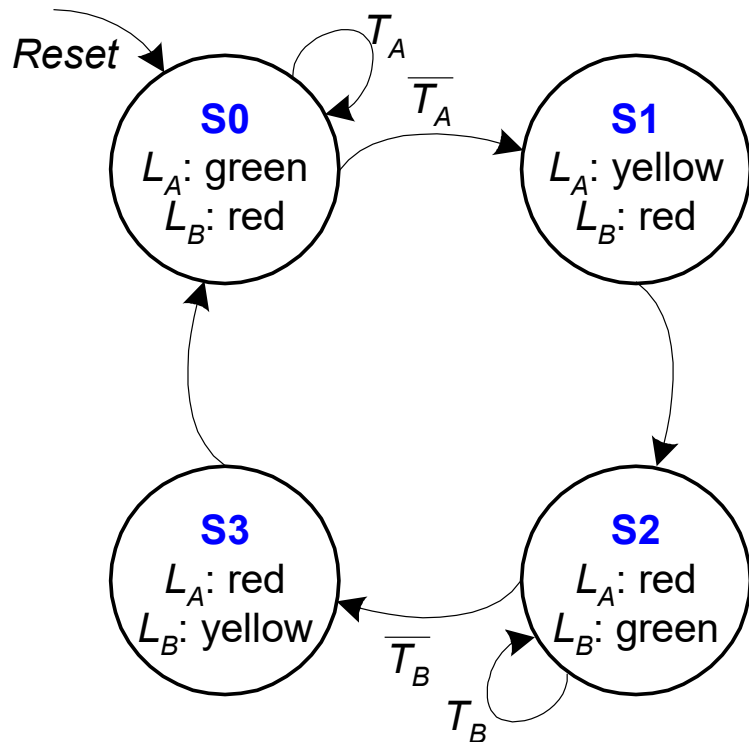
❖ Moore FSM: outputs labeled in each state

- States: Circles
- Transitions: Arcs

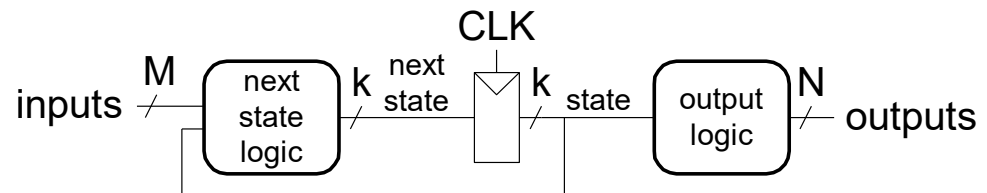


FSM – STATE TRANSITION TABLE

❖ State transition table

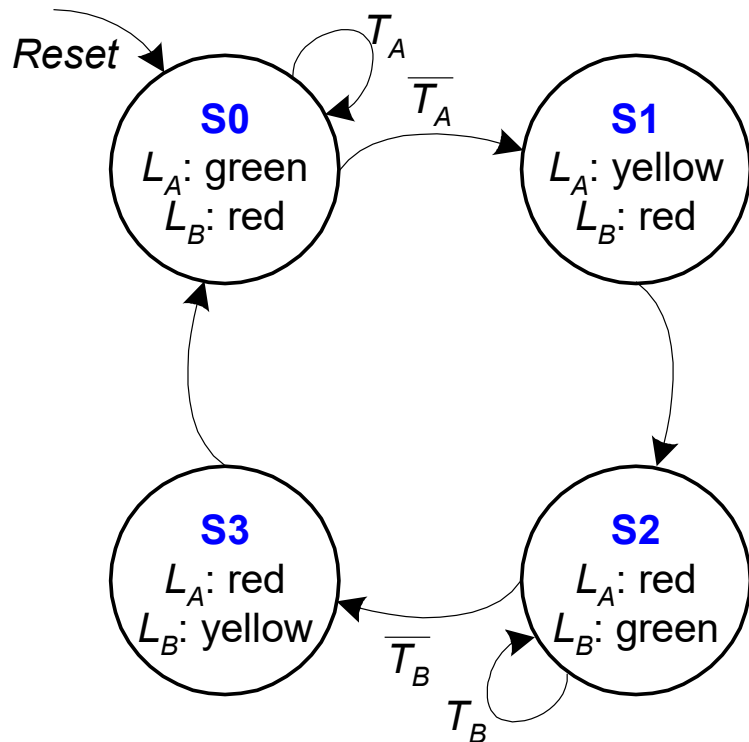


Current State	Inputs		Next State
S	T_A	T_B	S'
S0	0	X	
S0	1	X	
S1	X	X	
S2	X	0	
S2	X	1	
S3	X	X	

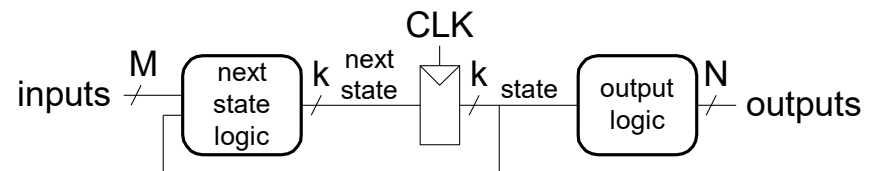


FSM – STATE TRANSITION TABLE

❖ State transition table

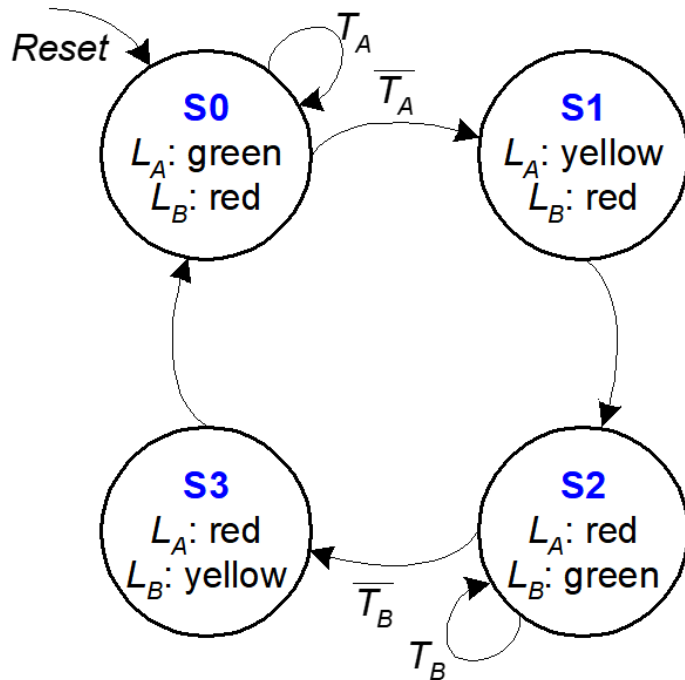


Current State	Inputs		Next State
S	T_A	T_B	S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0



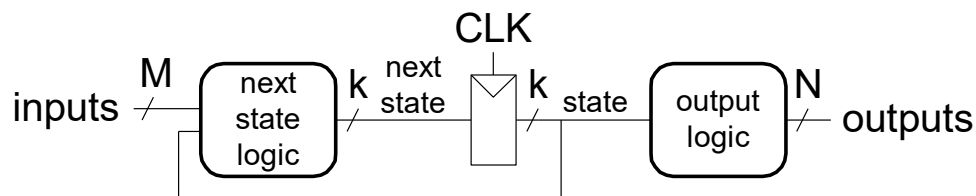
FSM – STATE TRANSITION TABLE

❖ State transition table



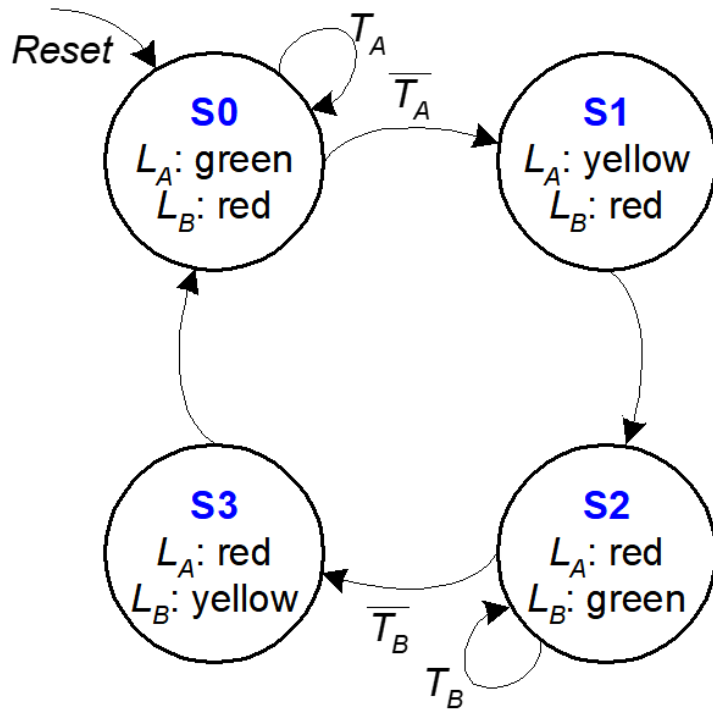
Current State	Inputs		Next State
S	T_A	T_B	S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

State	Encoding
S0	00
S1	01
S2	10
S3	11



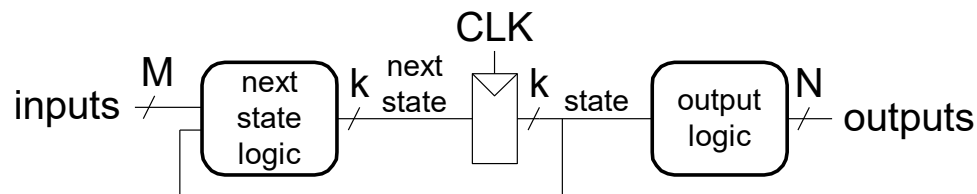
FSM – STATE TRANSITION TABLE

❖ State transition table



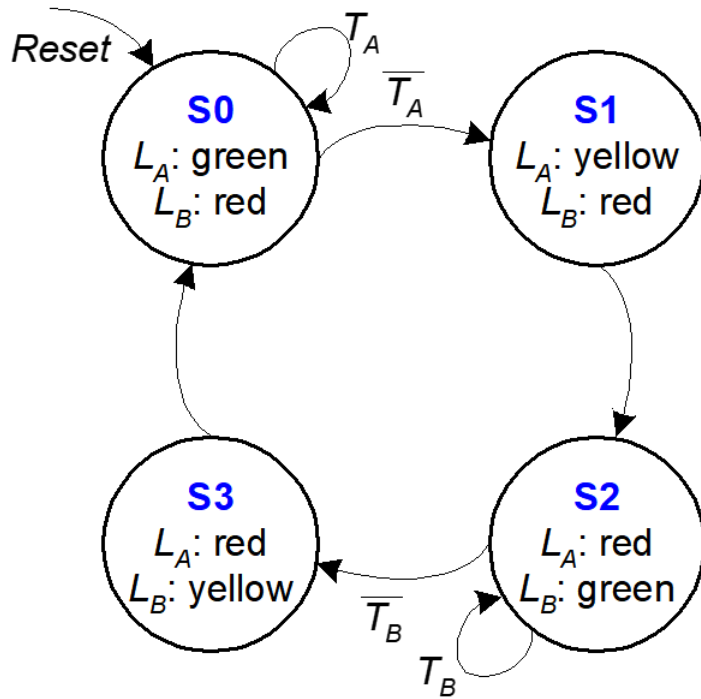
Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

State	Encoding
S0	00
S1	01
S2	10
S3	11



FSM – STATE TRANSITION TABLE

❖ State transition table



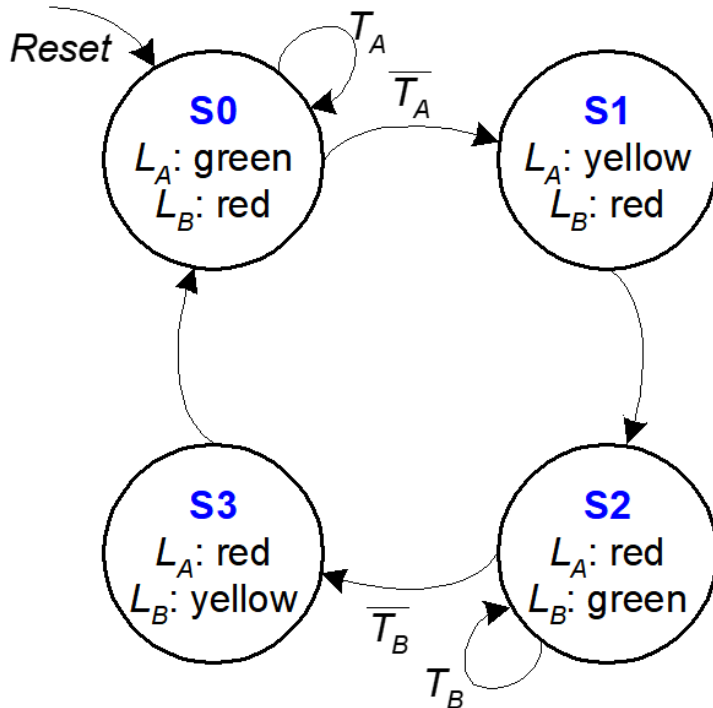
$S'_1 = ?$

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 – STATE TRANSITION TABLE

❖ State transition table



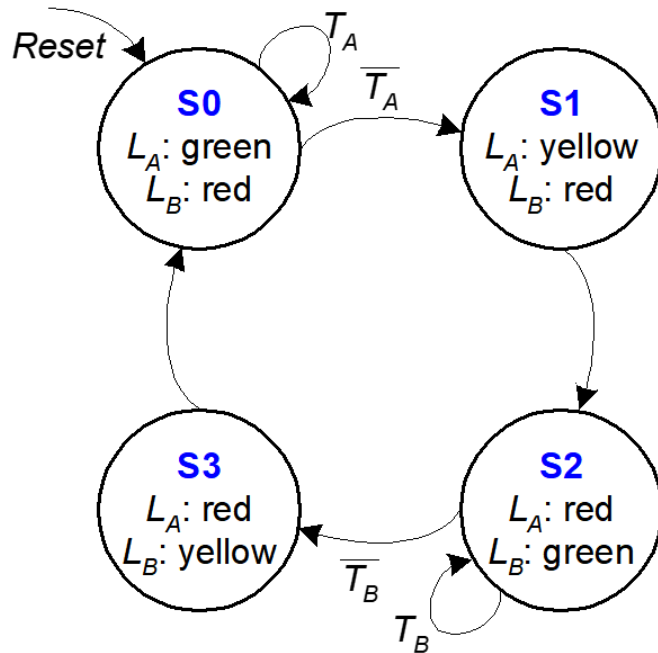
Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

State	Encoding
S0	00
S1	01
S2	10
S3	11

$$S'_1 = (\bar{S}_1 \cdot S_0) + (S_1 \cdot \bar{S}_0 \cdot \bar{T}_B) + (S_1 \cdot \bar{S}_0 \cdot T_B)$$

FSM – STATE TRANSITION TABLE

❖ 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

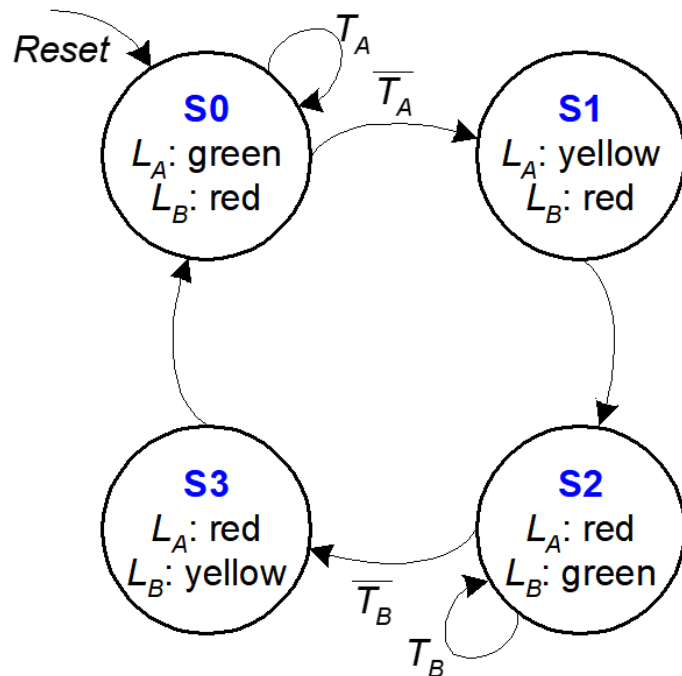
State	Encoding
S0	00
S1	01
S2	10
S3	11

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

$$S'_0 = ?$$

FSM – STATE TRANSITION TABLE

❖ 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

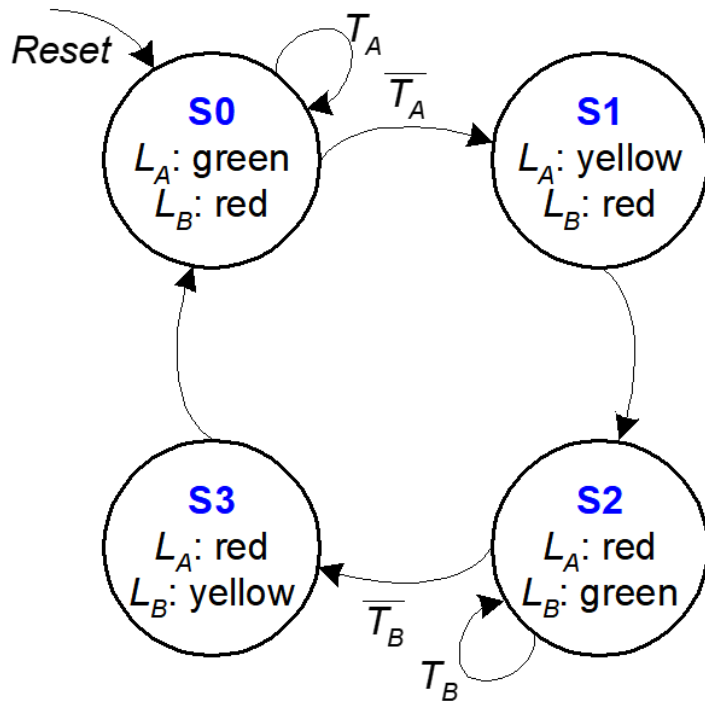
State	Encoding
S0	00
S1	01
S2	10
S3	11

$$S'_1 = (\bar{S}_1 \cdot S_0) + (S_1 \cdot \bar{S}_0 \cdot \bar{T}_B) + (S_1 \cdot \bar{S}_0 \cdot T_B)$$

$$S'_0 = (\bar{S}_1 \cdot \bar{S}_0 \cdot \bar{T}_A) + (S_1 \cdot \bar{S}_0 \cdot \bar{T}_B)$$

FSM – STATE TRANSITION TABLE

❖ 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

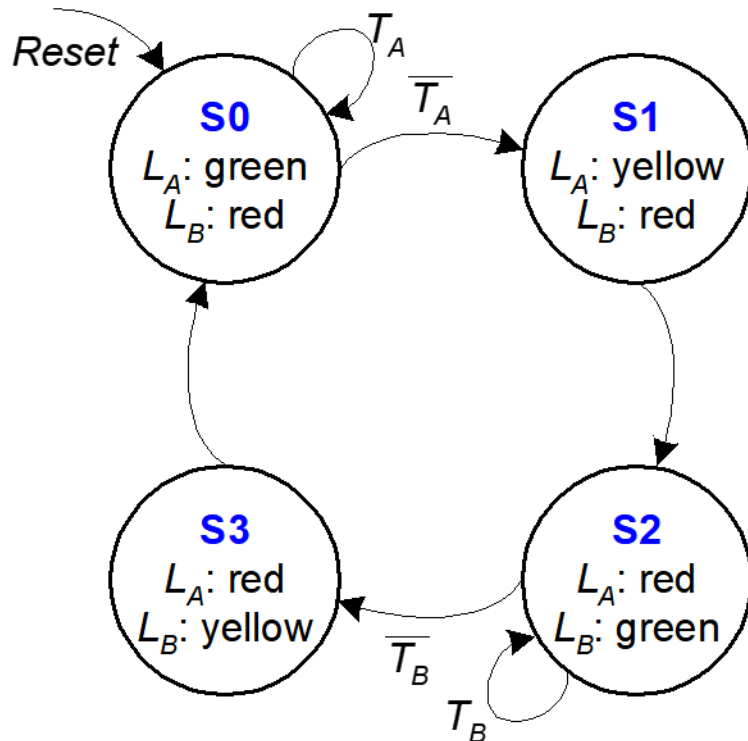
State	Encoding
S0	00
S1	01
S2	10
S3	11

$$S'_1 = S_1 \text{ xor } S_0 \quad \textbf{(Simplified)}$$

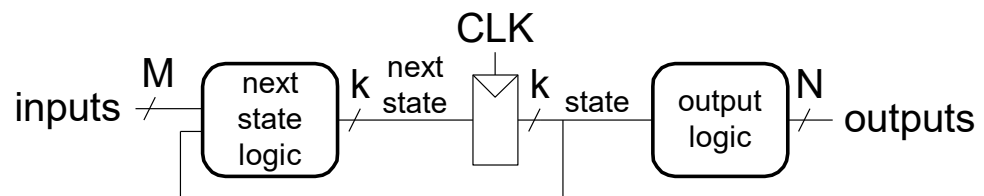
$$S'_0 = (\overline{S_1} \cdot \overline{S_0} \cdot \overline{T_A}) + (S_1 \cdot \overline{S_0} \cdot \overline{T_B})$$

FSM – OUTPUT TABLE

❖ Output table

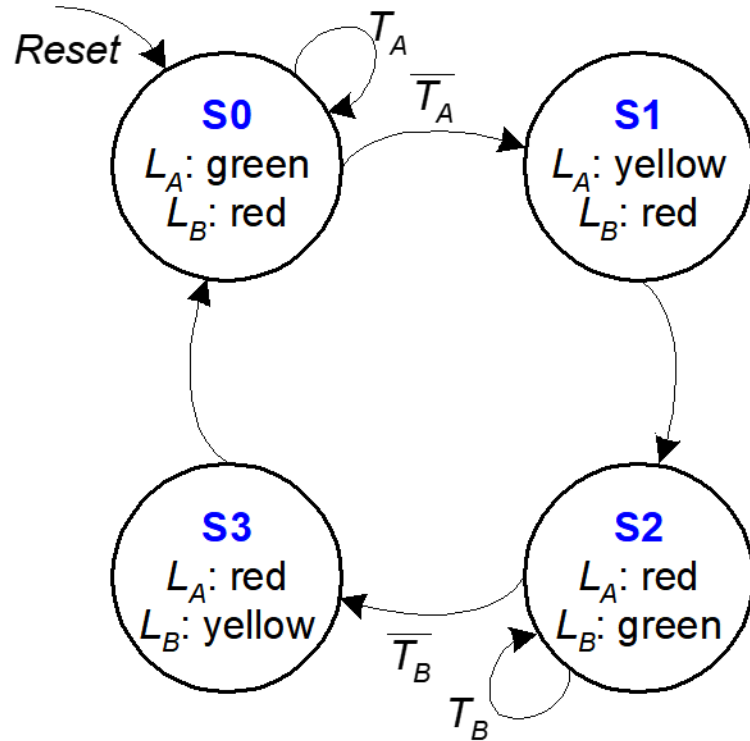


Current State		Outputs	
S_1	S_0	L_A	L_B
0	0	green	red
0	1	yellow	red
1	0	red	green
1	1	red	yellow

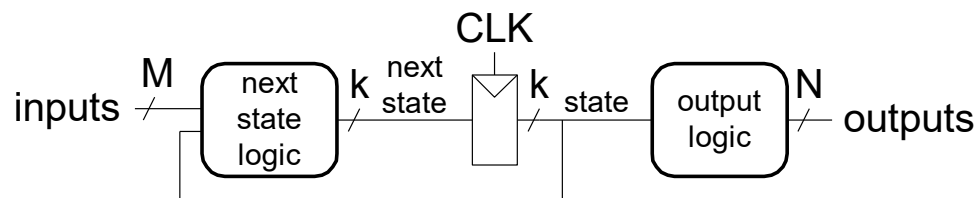


FSM – OUTPUT TABLE

❖ Output table



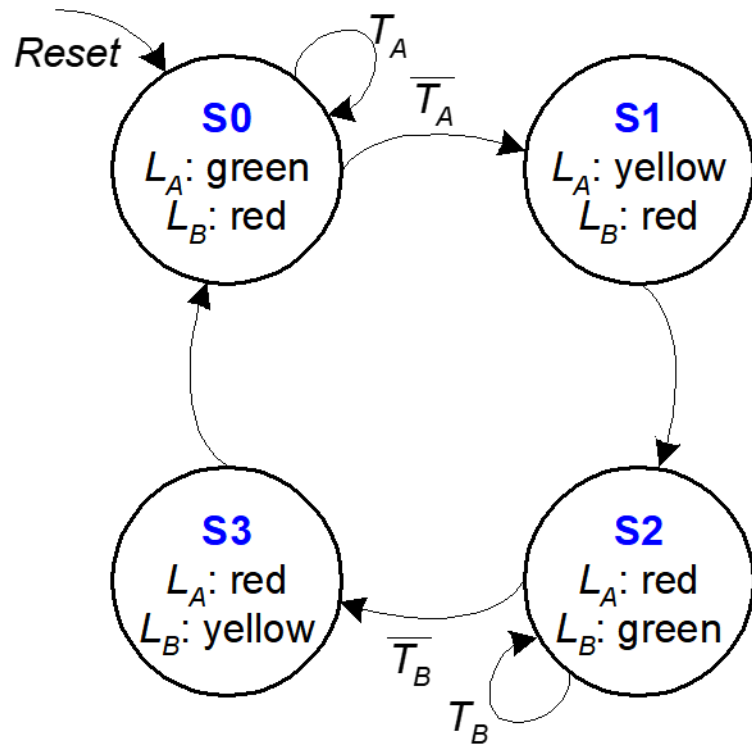
Current State		Outputs	
S_1	S_0	L_A	L_B
0	0	green	red
0	1	yellow	red
1	0	red	green
1	1	red	yellow



Output	Encoding
green	00
yellow	01
red	10

FSM – OUTPUT TABLE

❖ Output table



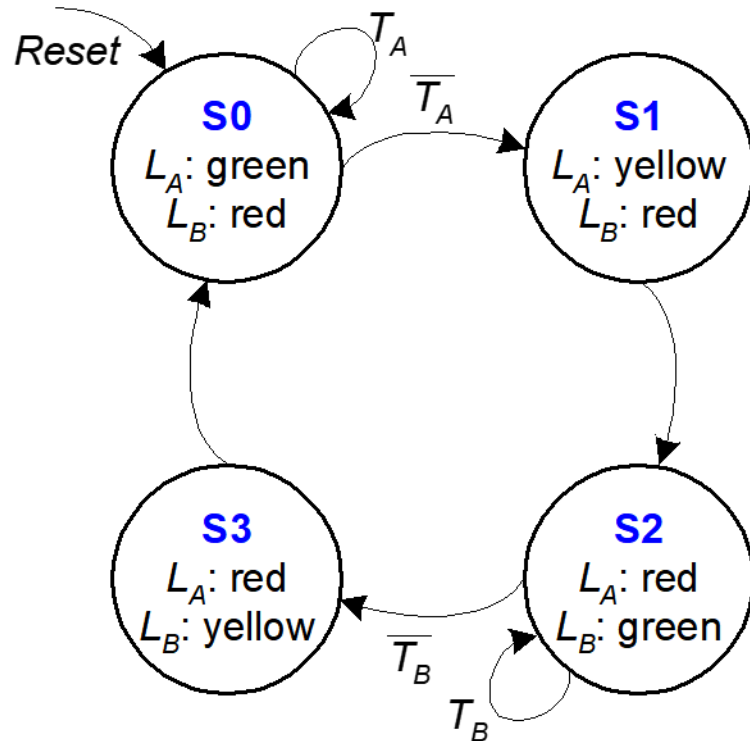
$$L_{A1} = S_1$$

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

❖ Output table



$$L_{A1} = S_1$$

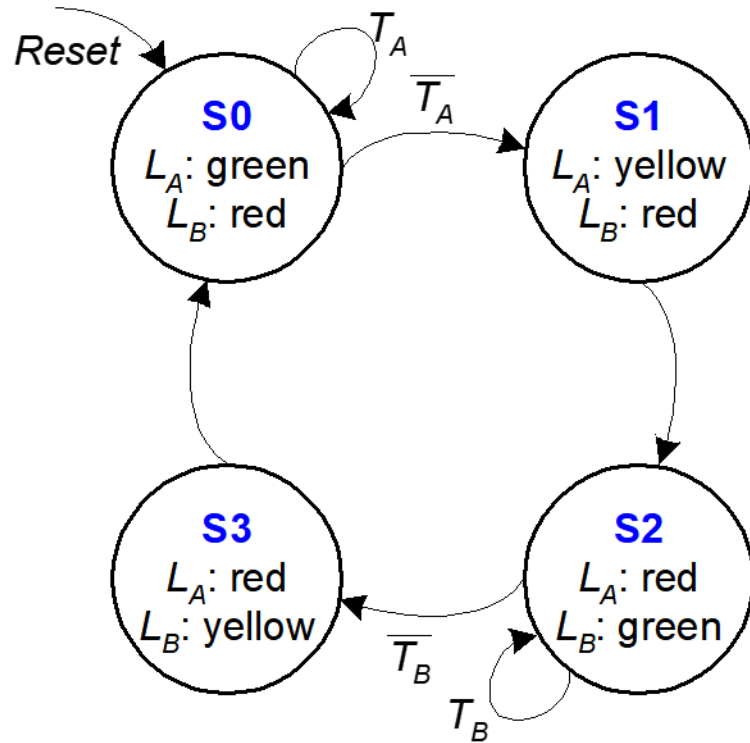
$$L_{A0} = \overline{S_1} \cdot S_0$$

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

❖ Output table



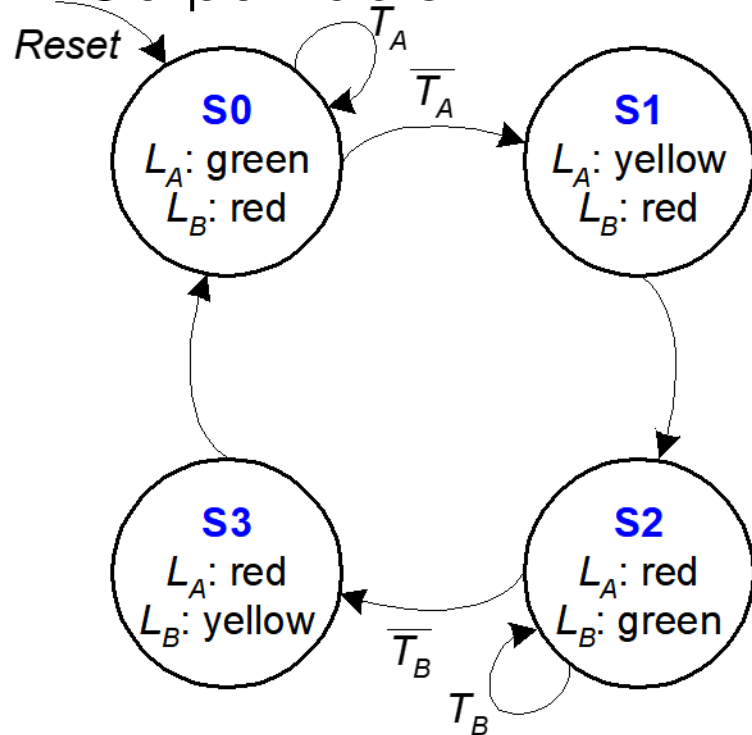
Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

Output	Encoding
green	00
yellow	01
red	10

$$\begin{aligned}
 L_{A1} &= \overline{S_1} \\
 L_{A0} &= \overline{S_1} \cdot S_0 \\
 L_{B1} &= S_1
 \end{aligned}$$

FSM – OUTPUT TABLE

❖ 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

$$L_{A1} = S_1$$

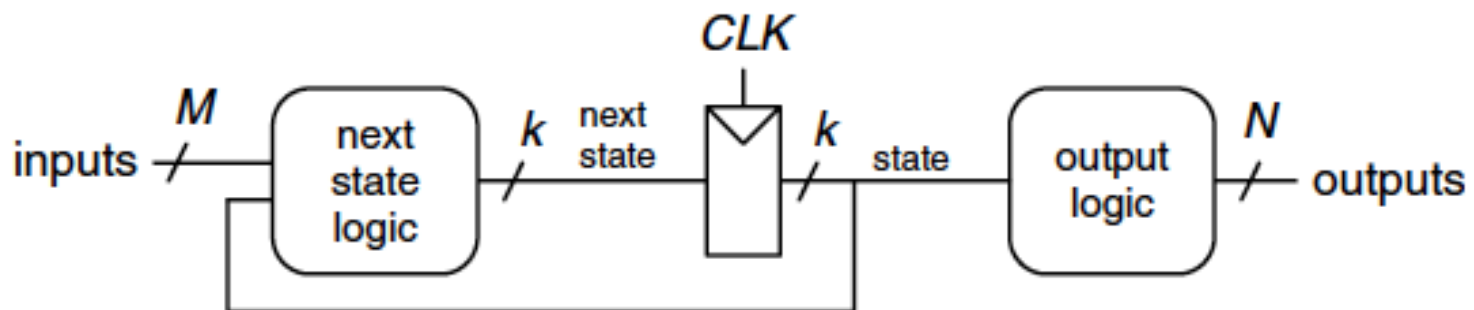
$$L_{A0} = \overline{S_1} \cdot S_0$$

$$L_{B1} = \overline{S_1}$$

$$L_{B0} = S_1 \cdot S_0$$

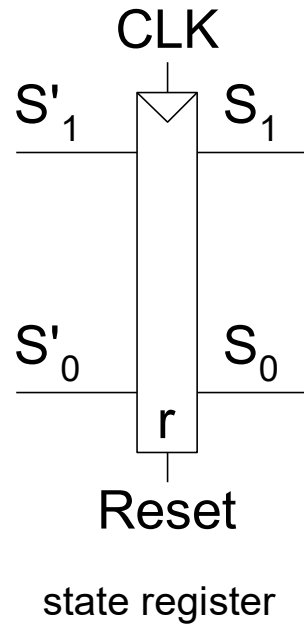
FSM – SCHEMATIC

❖ Overview



FSM – SCHEMATIC

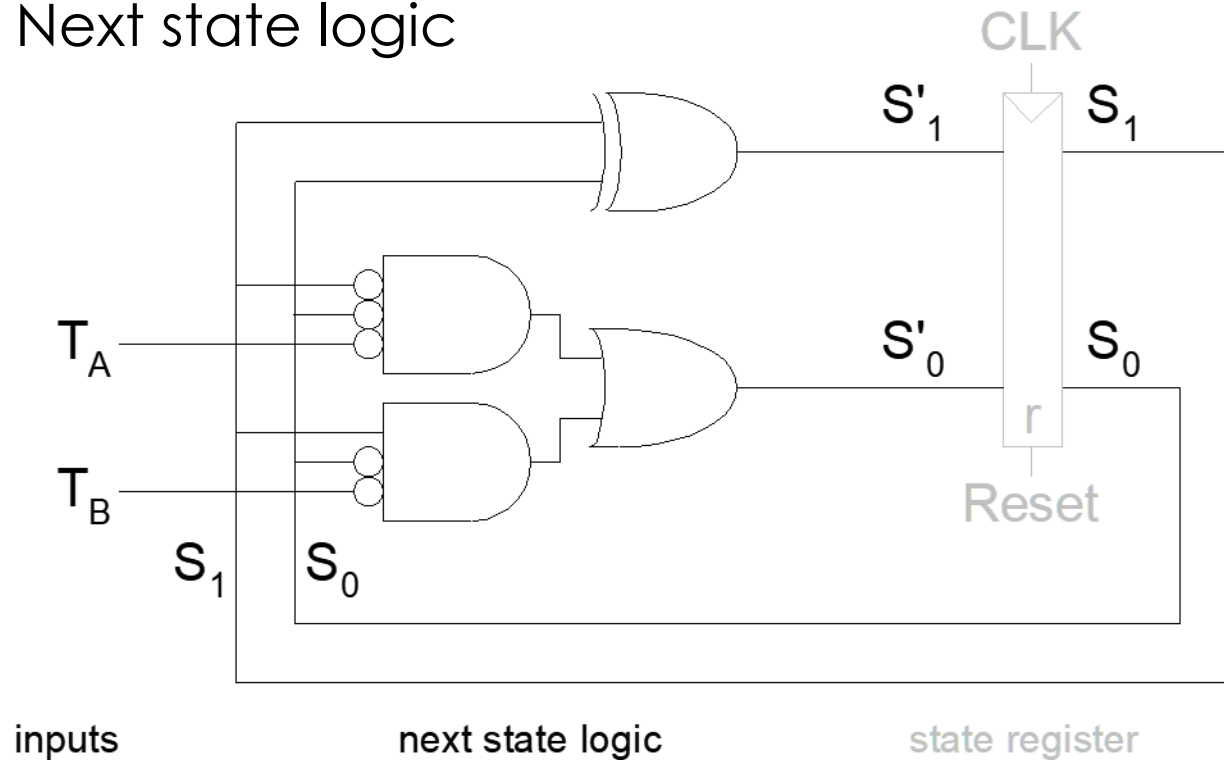
❖ State register





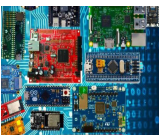
FSM – SCHEMATIC

❖ Next state logic



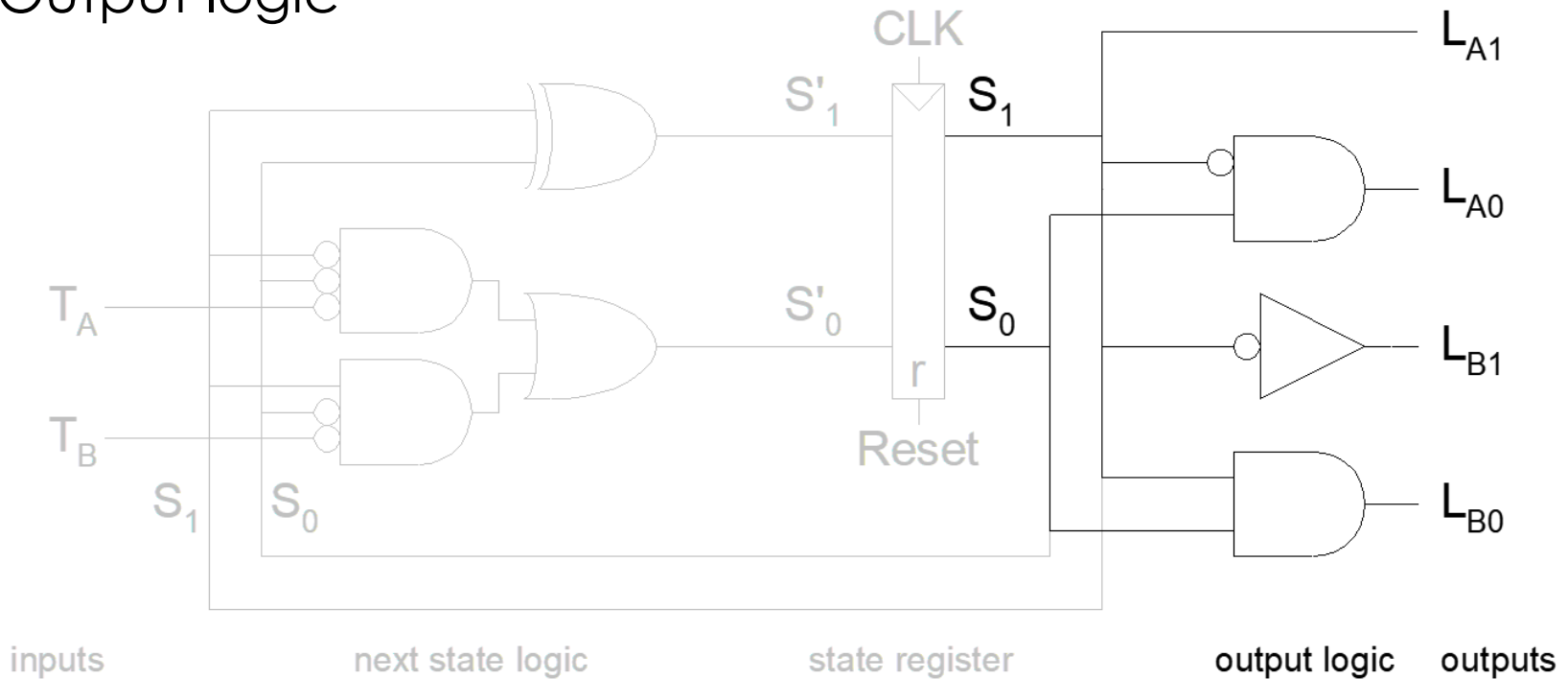
$$S'_1 = S_1 \text{ xor } S_0$$

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

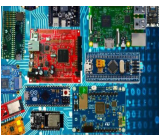


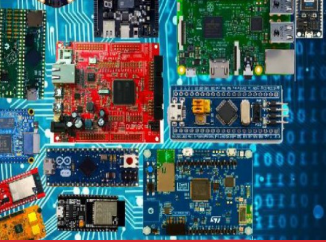
FSM – SCHEMATIC

❖ Output logic



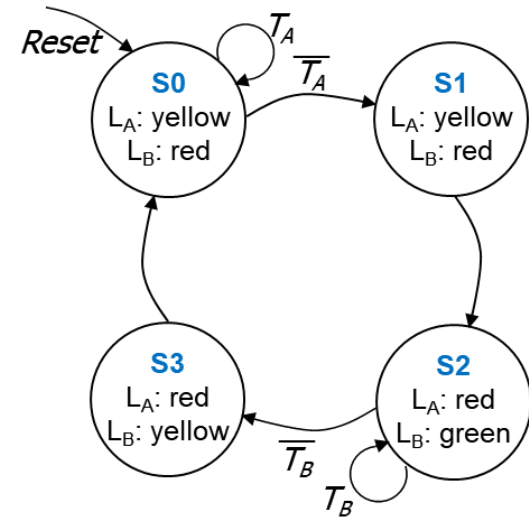
$$\begin{aligned} L_{A1} &= S_1 \\ L_{A0} &= \overline{S_1} \cdot S_0 \\ L_{B1} &= \overline{S_1} \\ L_{B0} &= S_1 \cdot S_0 \end{aligned}$$



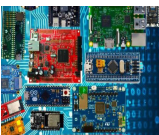


FSM – TIMING DIAGRAM

❖ Timing diagram



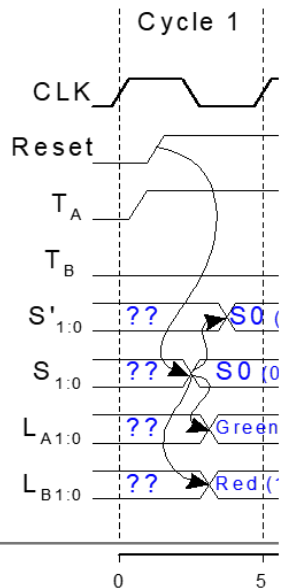
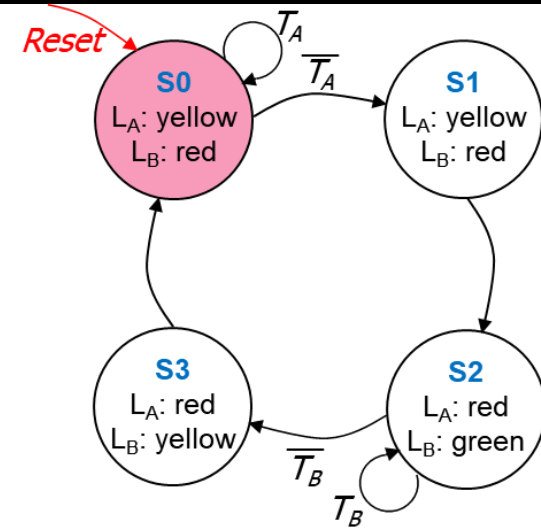
CLK_
Reset_
 T_A _
 T_B _
 $S'_{1:0}$ _
 $S_{1:0}$ _
 $L_{A1:0}$ _
 $L_{B1:0}$ _

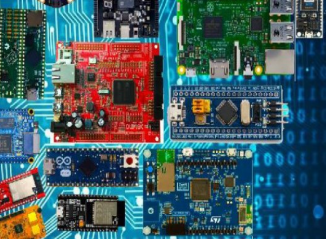




FSM – TIMING DIAGRAM

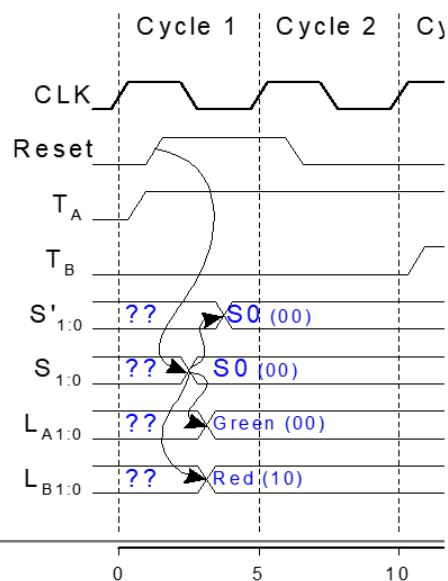
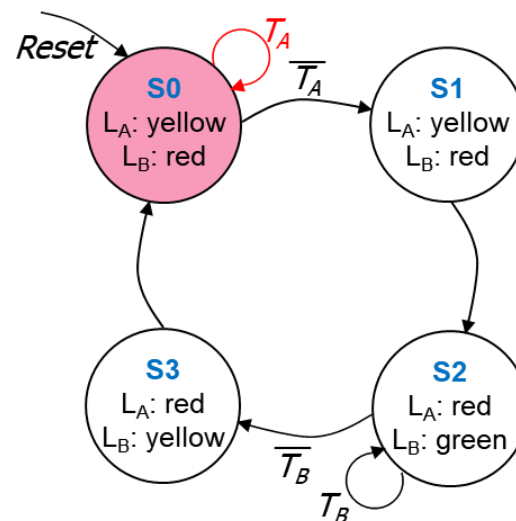
❖ Timing diagram

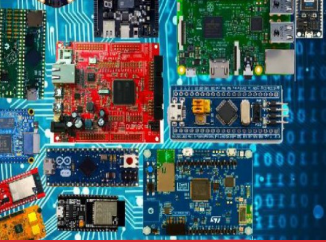




FSM – TIMING DIAGRAM

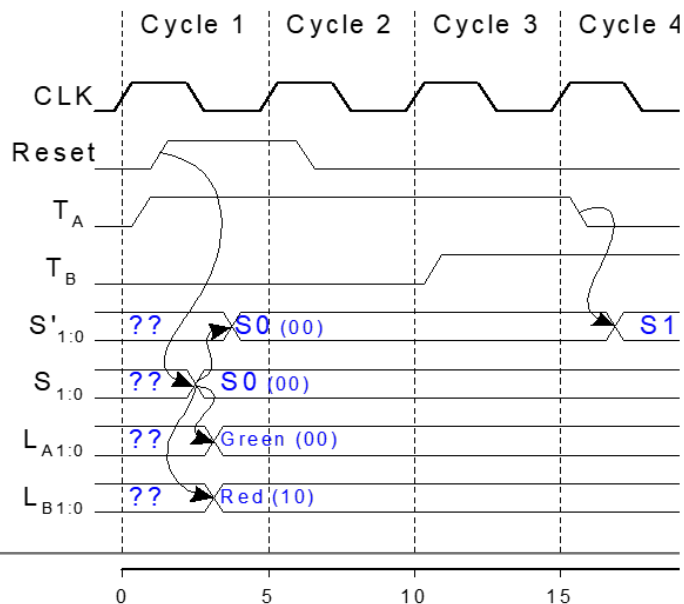
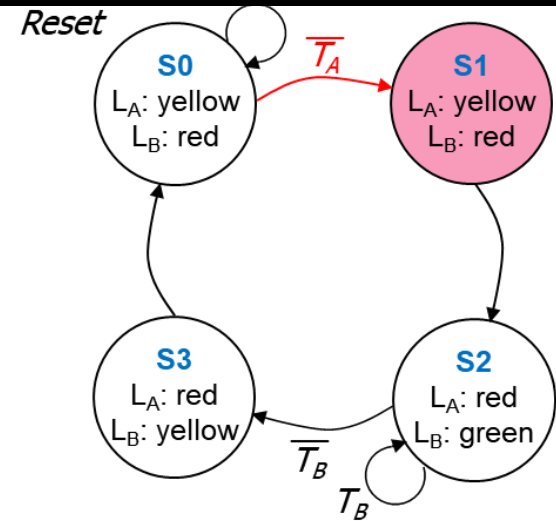
❖ Timing diagram

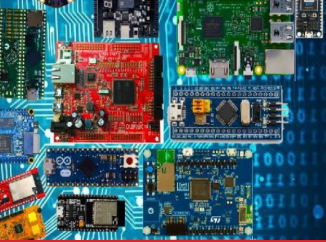




FSM – TIMING DIAGRAM

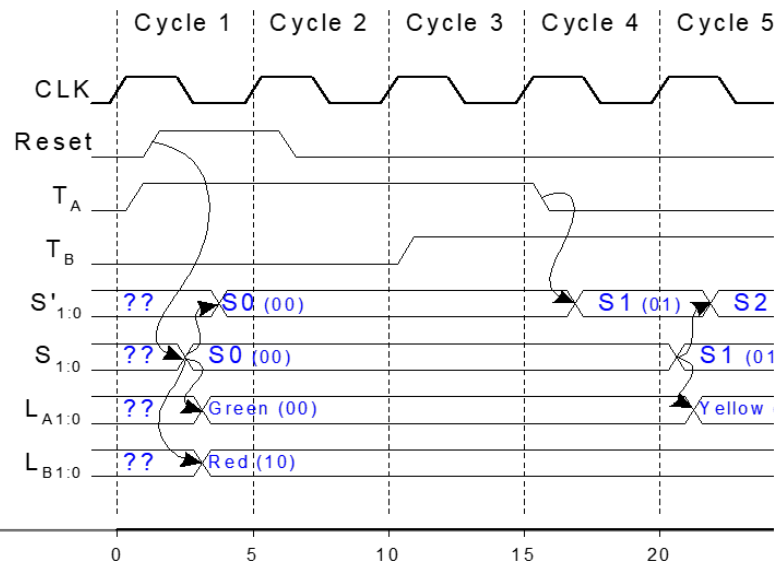
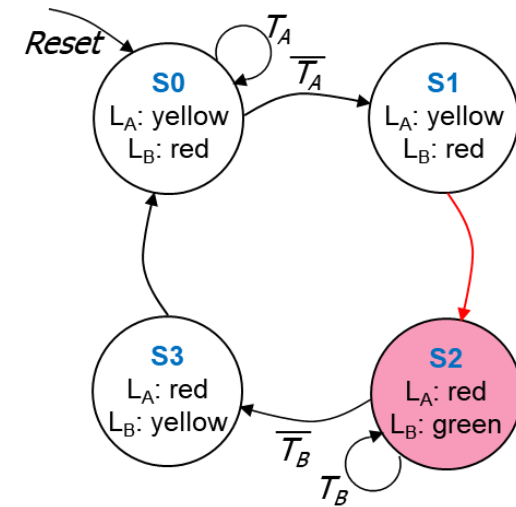
❖ Timing diagram





FSM – TIMING DIAGRAM

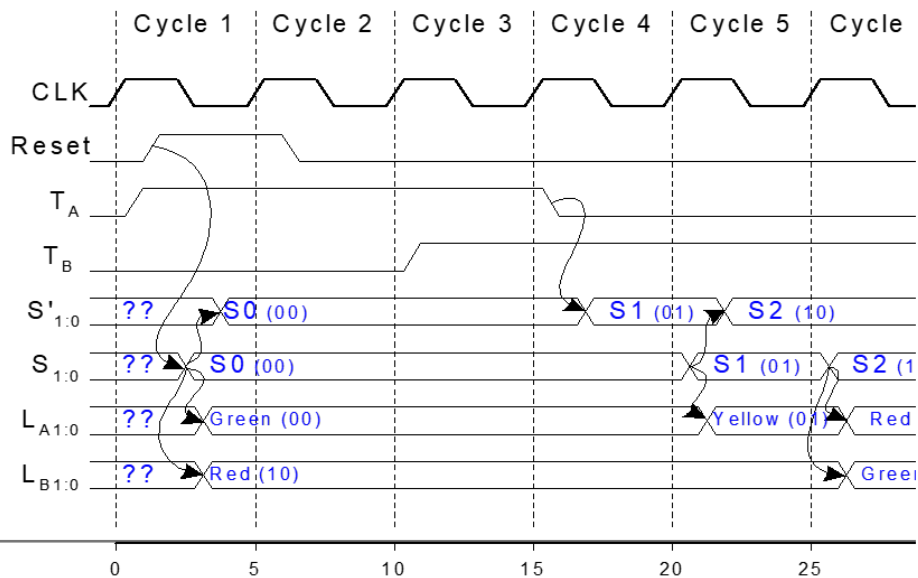
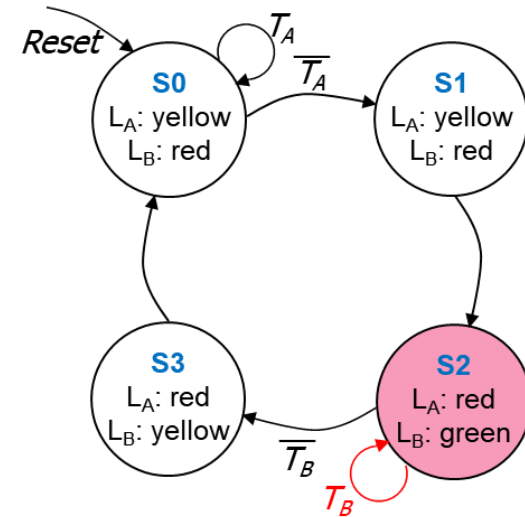
❖ Timing diagram





FSM – TIMING DIAGRAM

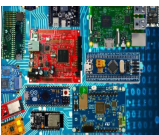
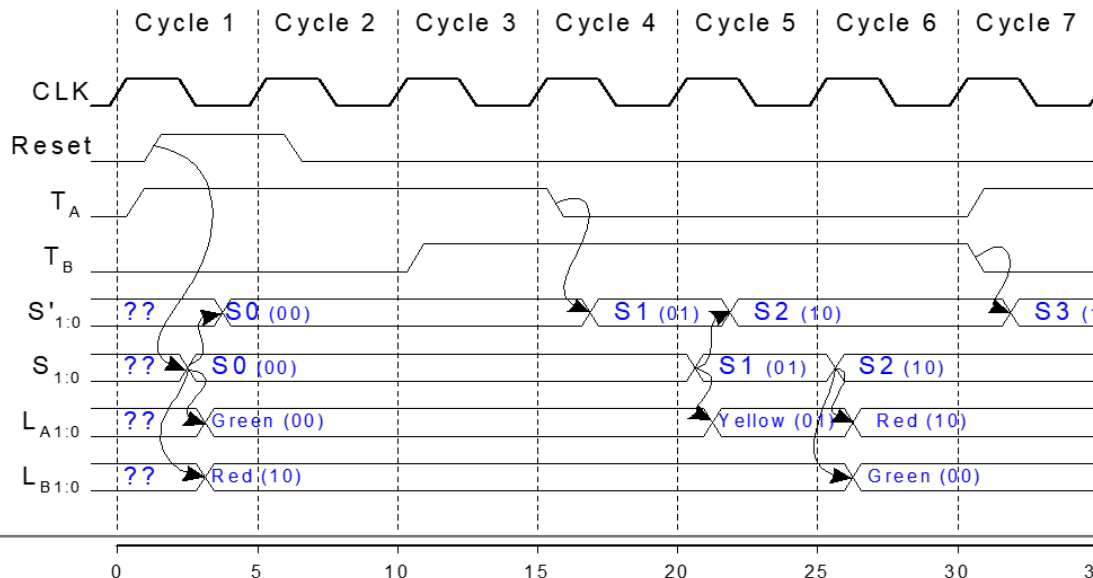
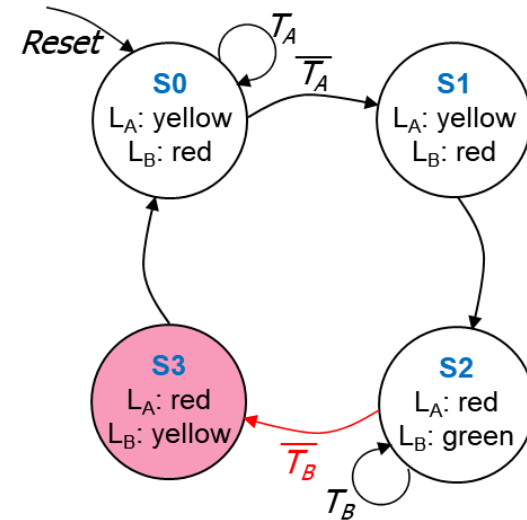
❖ Timing diagram

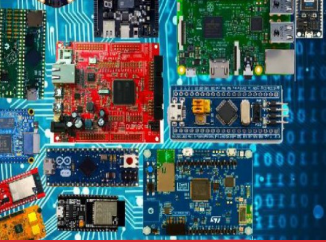




FSM – TIMING DIAGRAM

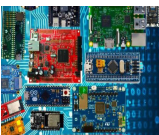
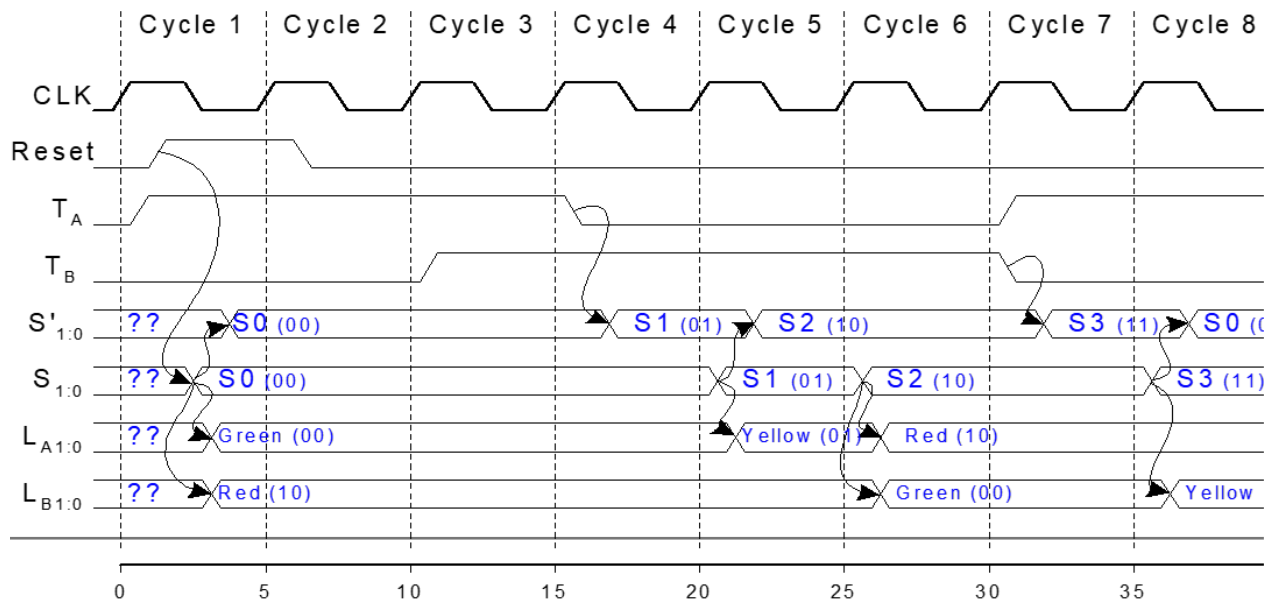
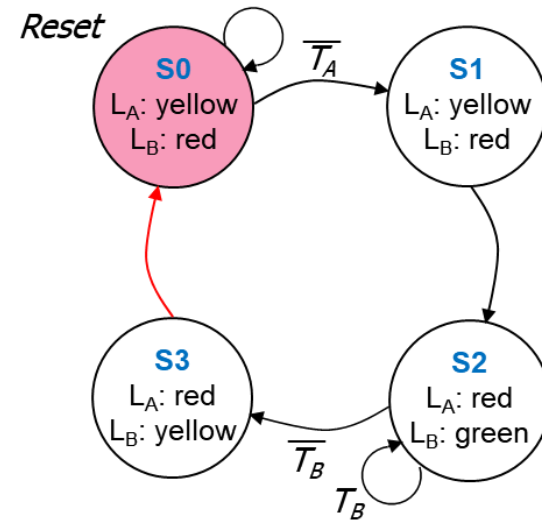
❖ Timing diagram

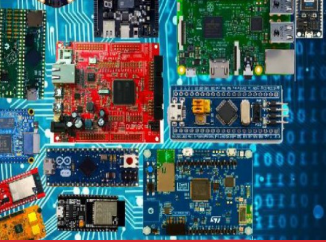




FSM – TIMING DIAGRAM

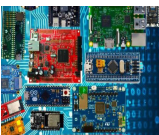
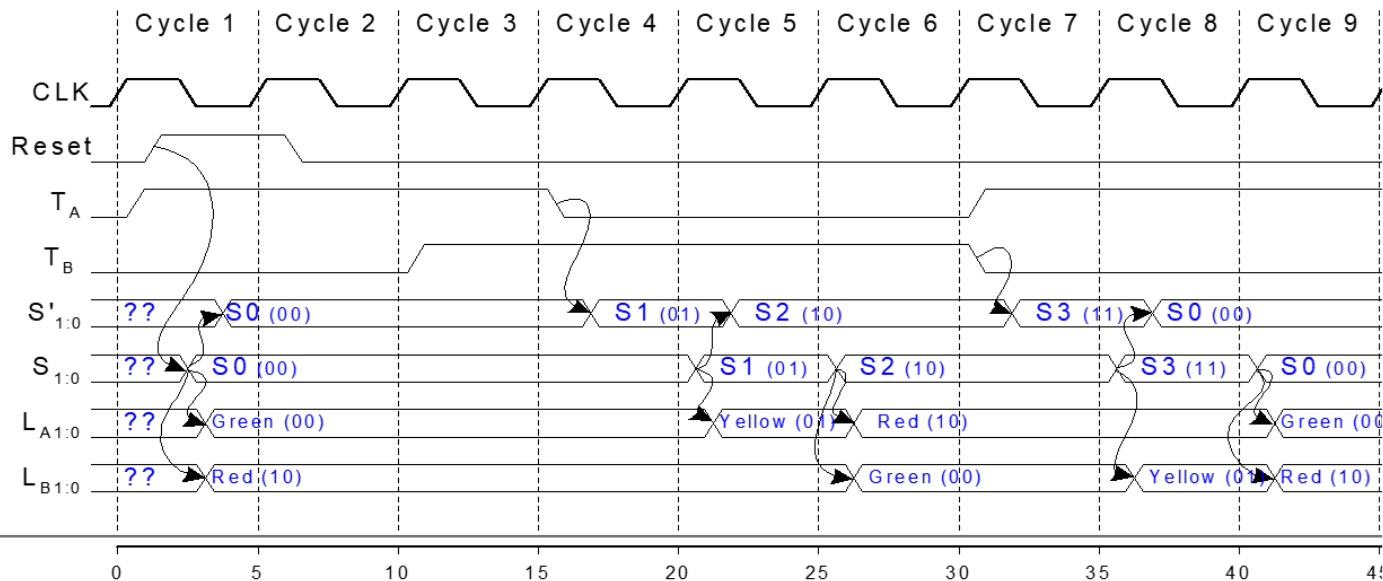
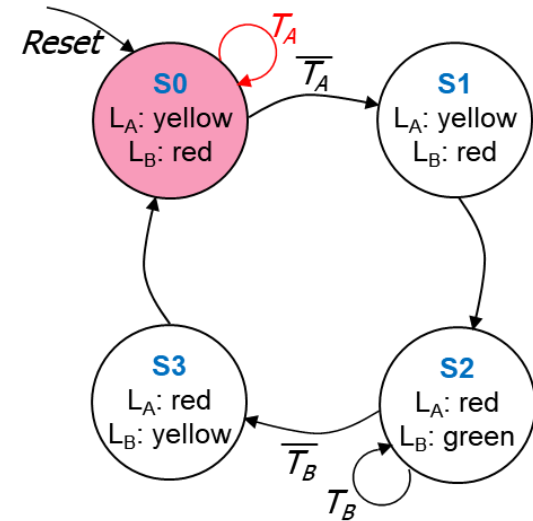
❖ Timing diagram





FSM – TIMING DIAGRAM

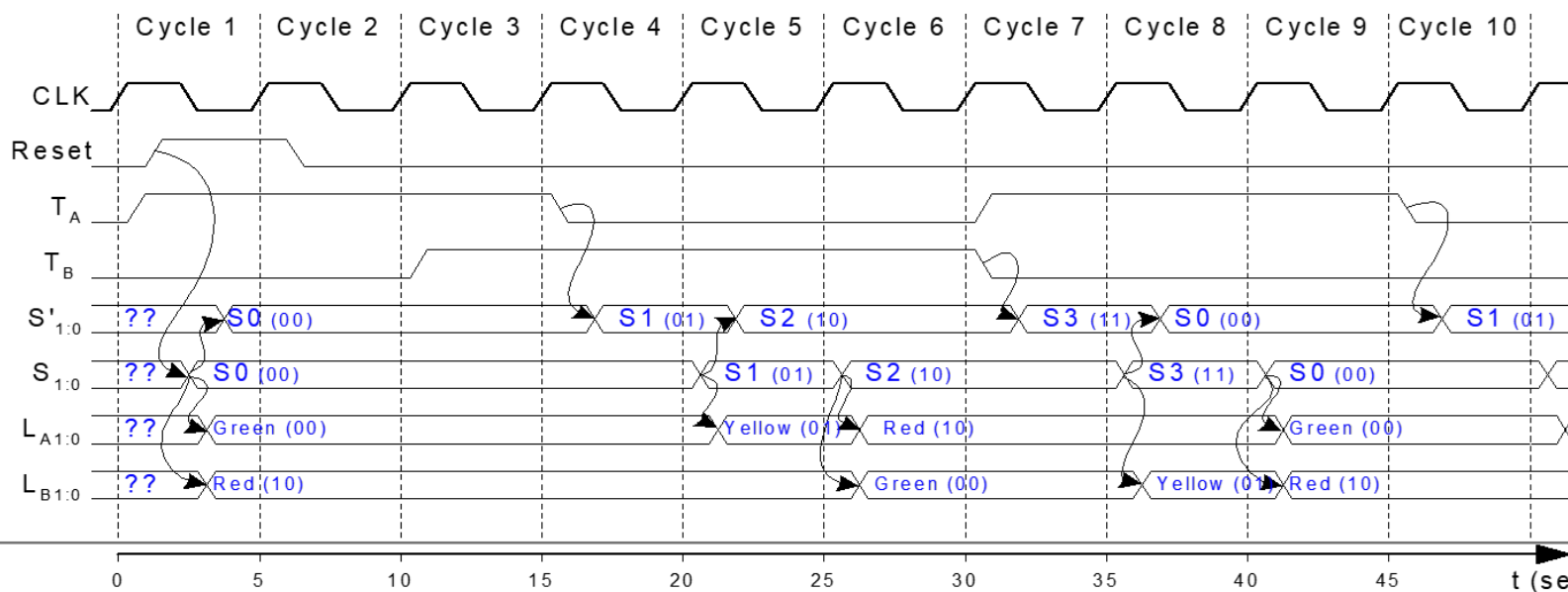
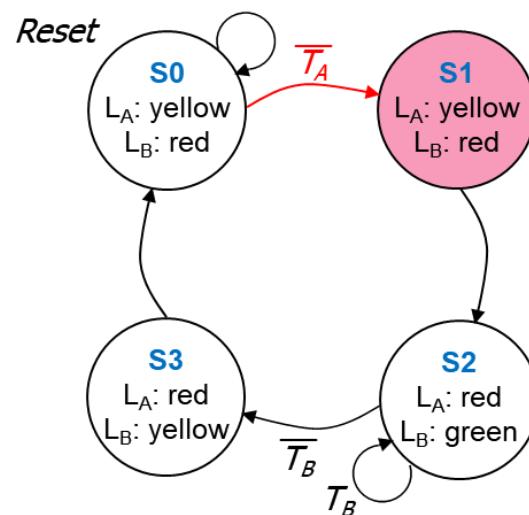
❖ Timing diagram





FSM – TIMING DIAGRAM

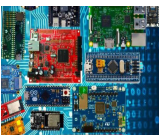
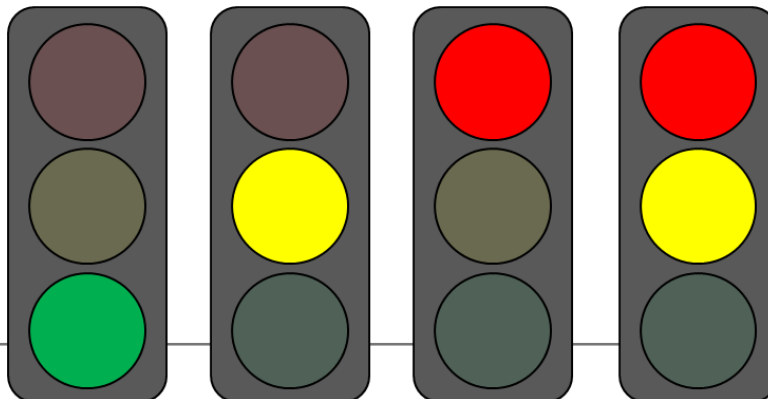
❖ Timing diagram





FSM – STATE ENCODING

- ❖ How do we encode the state bits?
- ❖ Three common state binary encodings with different tradeoffs
 - Fully Encoded
 - 1-Hot Encoded
 - Output Encoded
- ❖ Let's see an example Swiss traffic light with 4 states
 - Green, Yellow, Red, Yellow+Red

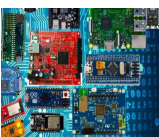




FSM – STATE ENCODING

❖ Binary Encoding (Full Encoding):

- Use the minimum possible number of bits
 - Use $\log_2(\text{num_states})$ bits to represent the states
- Example state encodings: 00, 01, 10, 11
- Minimizes # flip-flops, but not necessarily output logic or next state logic

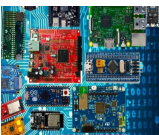




FSM – STATE ENCODING

❖ 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 state encodings: 0001, 0010, 0100, 1000
- Simplest design process – very automatable
- Maximizes # flip-flops, minimizes next state logic

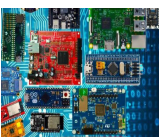




FSM – STATE ENCODING

❖ Output Encoding:

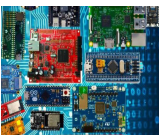
- Outputs are directly accessible in the state encoding
- For example, since we have 3 outputs (light color), encode state with 3 bits, where each bit represents a color
- Example states: 001, 010, 100, 110
 - Bit0 encodes green light output,
 - Bit1 encodes yellow light output
 - Bit2 encodes red light output
- Minimizes output logic
- Only works for Moore Machines (output function of state)





FSM – STATE ENCODING

- ❖ The designer must carefully choose an encoding scheme to optimize the design under given constraints





State Transition Table

Current State	Inputs		Next State
	T_A	T_B	
S			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

State	Encoding
S0	0001
S1	0010
S2	0100
S3	1000

Output Table

Current State	LA
S0	green
S1	yellow
S2	red
S3	red

Output	Encoding
green	0001
yellow	0010
red	0100

