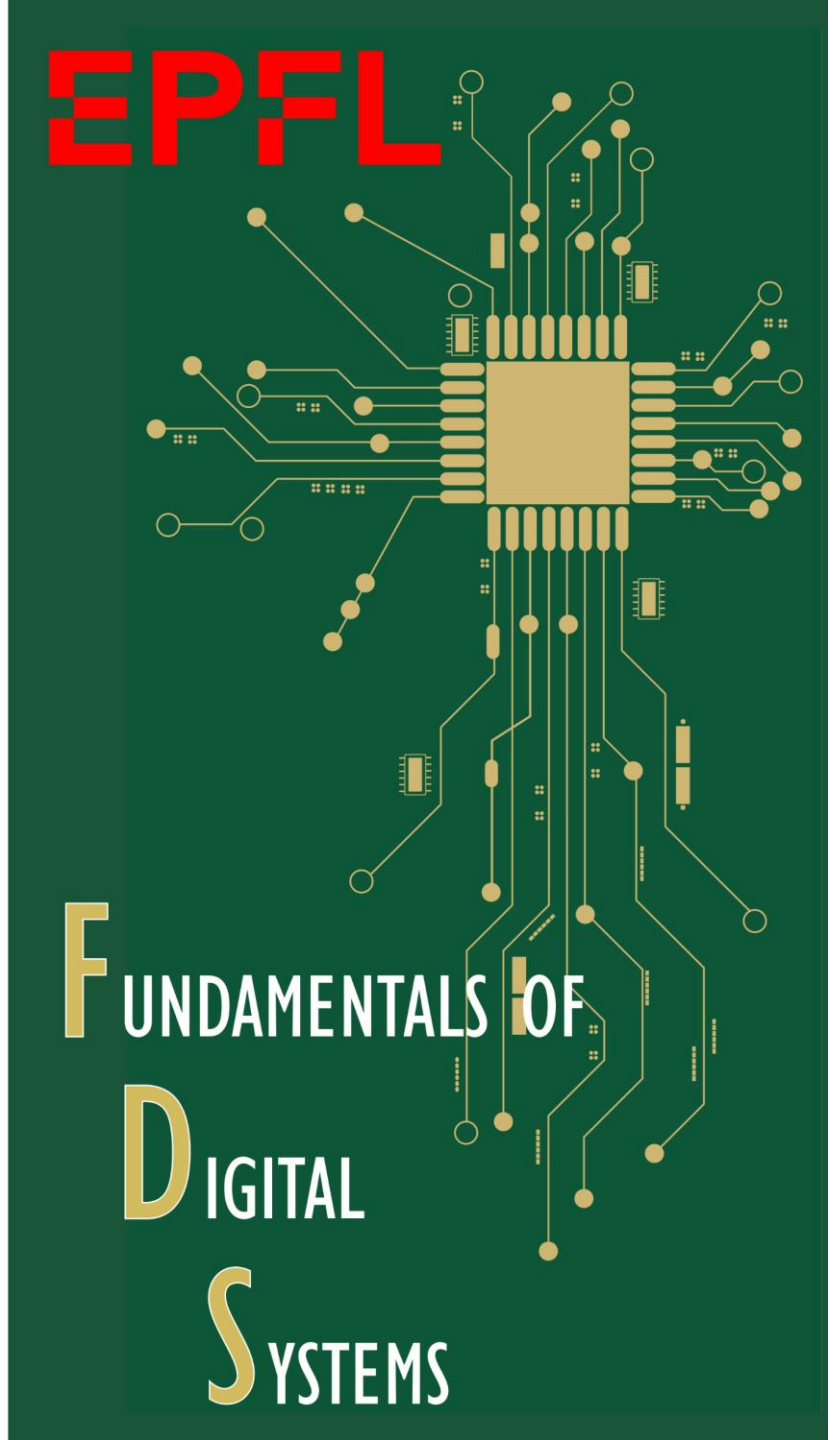


Digital Logic Circuits

Finite State Machines

CS-173 Fundamentals of Digital Systems

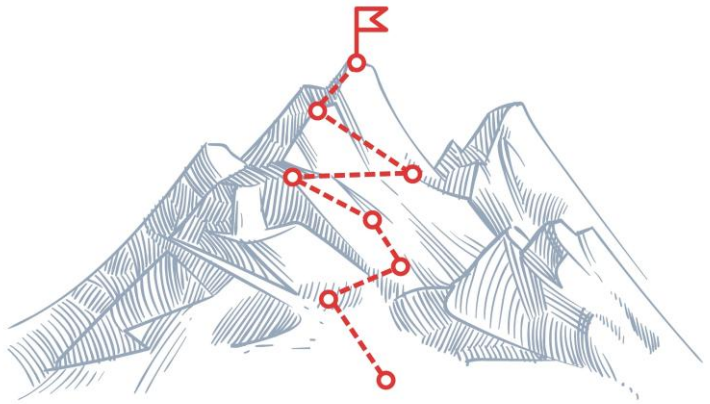
Mirjana Stojilović

Spring 2025

Previously on FDS

Memory

Previously

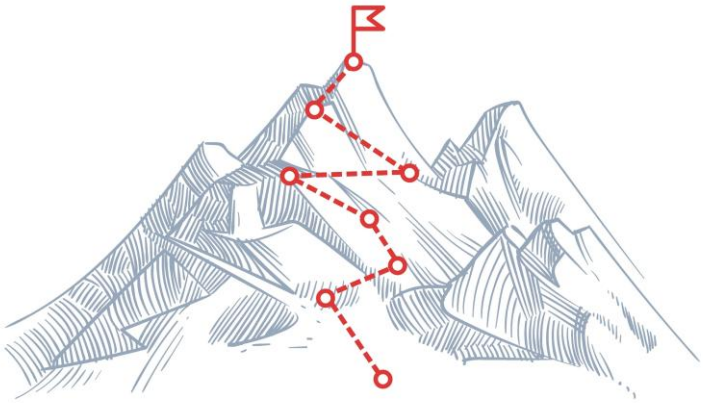


- **2-to-2^n Binary decoders**
  - Defined the functionality
  - Used Verilog **conditional operator** to model them
- Learned what **memories** are and how to build them with DFFs
  - **Two-dimensional arrays** of DFFs
  - **Address** port for indexing the row (i.e., data word)
    - Used 2-to-2^n binary decoder to activate one enable signal per row
  - **Access protocol**: when/how to read and when/how to write
  - Modeled memory as a two-dimensional array of DFFs in Verilog
    - Learned how to use **Verilog parameters** to be able to instantiate memories with various configurations (e.g., number of rows or data word width)

Let’s Talk About…

Finite state machines

Learning Outcomes



- Discover finite state machines (FSMs)
  - Mealy FSMs
  - Moore FSMs
- Analyze and design finite state machines
  - Apply step-by-step algorithms
  - Model FSMs in Verilog

Quick Outline

- State Machines
  - Mealy FSMs
  - Moore FSMs
- State Machine Analysis
  - State Diagram
  - Example 1
  - Example 2
- State Machine Design
  - Algorithm
  - Example: Traffic light controller

State Machines

Combinational vs. Sequential

Recap

- *Recall:* Logic circuits can be classified into two types
  - **Combinational**
    - Outputs depend only on the current inputs
    - No memory elements, no state
  - **Sequential**
    - Outputs depend on the current inputs and the current circuit's state
    - As the state is defined by the past inputs and past states
      we can say that the outputs depend on the **current and past inputs**
      (possibly arbitrarily far back in time)
    - Memory elements to keep (hold) states
- Practical circuits are a combination of both types

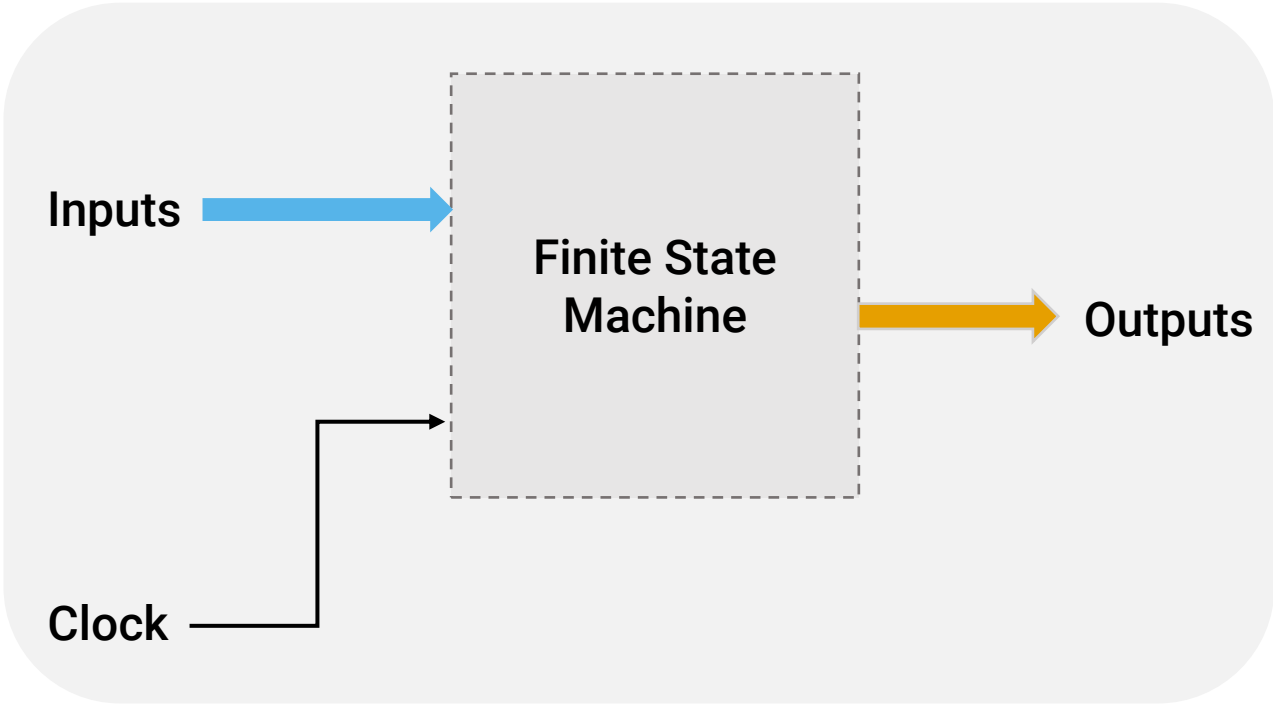
State Machines

Finite-State Machines

- Because circuits containing combinational and sequential logic have a state, we call them **state machines**
- States are represented by n-bit binary values
  (one memory element per one bit of the state)
  - Q: How many states can be represented with n bits?
  - A: $N_{\text{states}} = 2^n$
- As the number of states is limited (i.e., finite), we call such circuits **finite state machines (FSMs)**

Finite State Machines

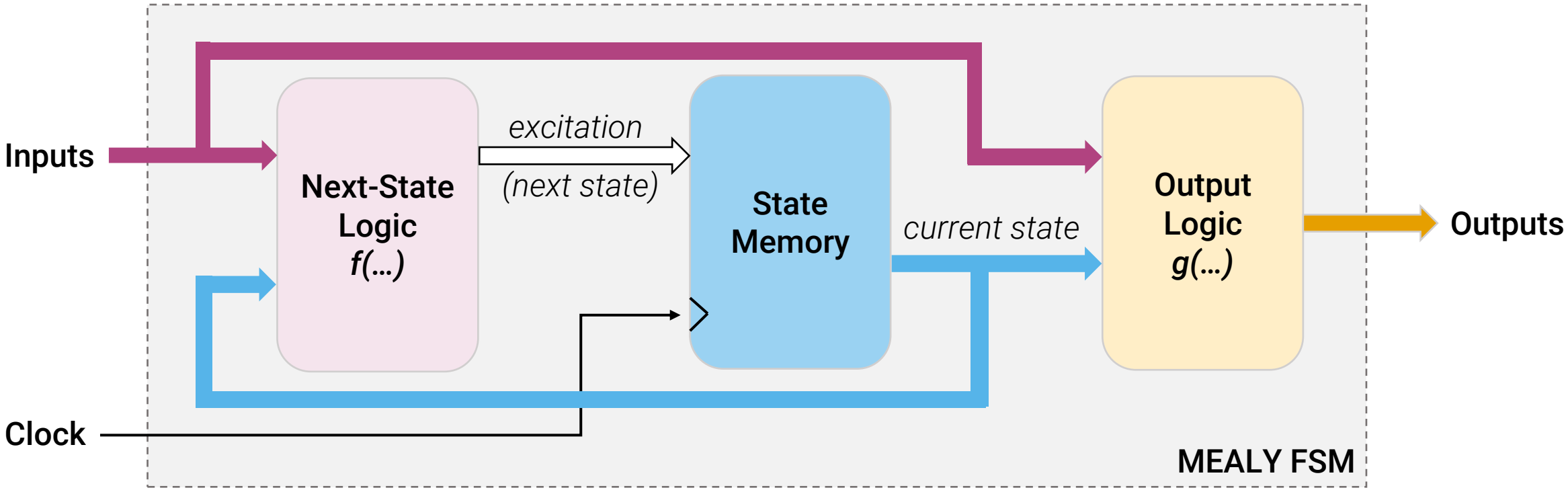
- Two types of FSMs exist
- Mealy state machines
  - Named after George H. Mealy
- Moore state machines
  - Named after Edward F. Moore



- State transitions occur in sync with the clock (e.g., rising edge)

Mealy State Machine

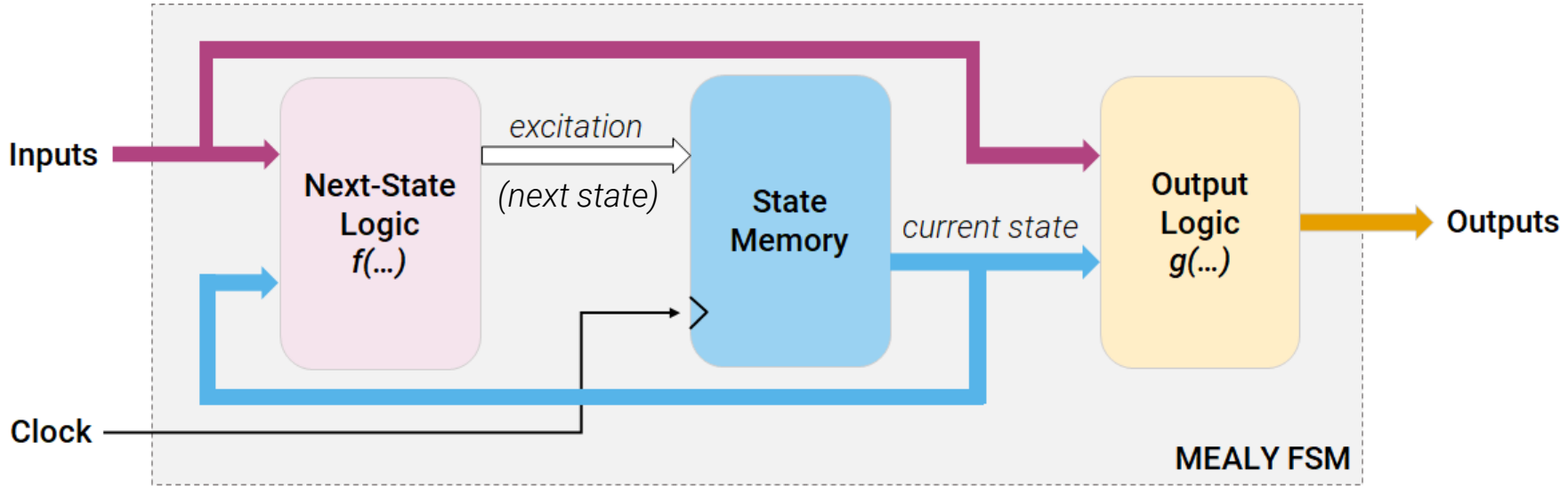
Structure



- If the outputs depend on the current state **and current inputs**, we have a **Mealy state machine**

Mealy State Machine

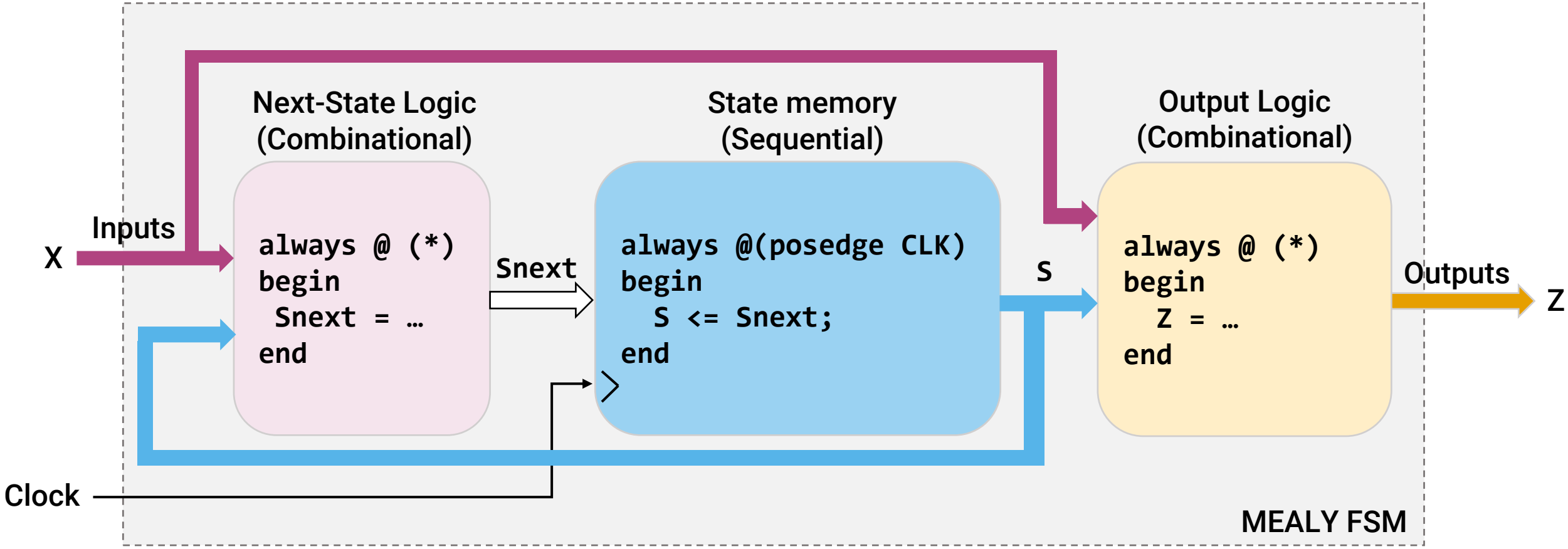
Contd.



- The **state memory** is a set of n flip-flops that store the current state
  - All connected to a common clock, causing them to update state once per clock period
- The **next state** is a function of inputs and current state
  - *Next state = f(current state, input)*
- The **output** is a function of the current state and inputs
  - *Output = g(current state, input)*

Mealy FSM Modeling

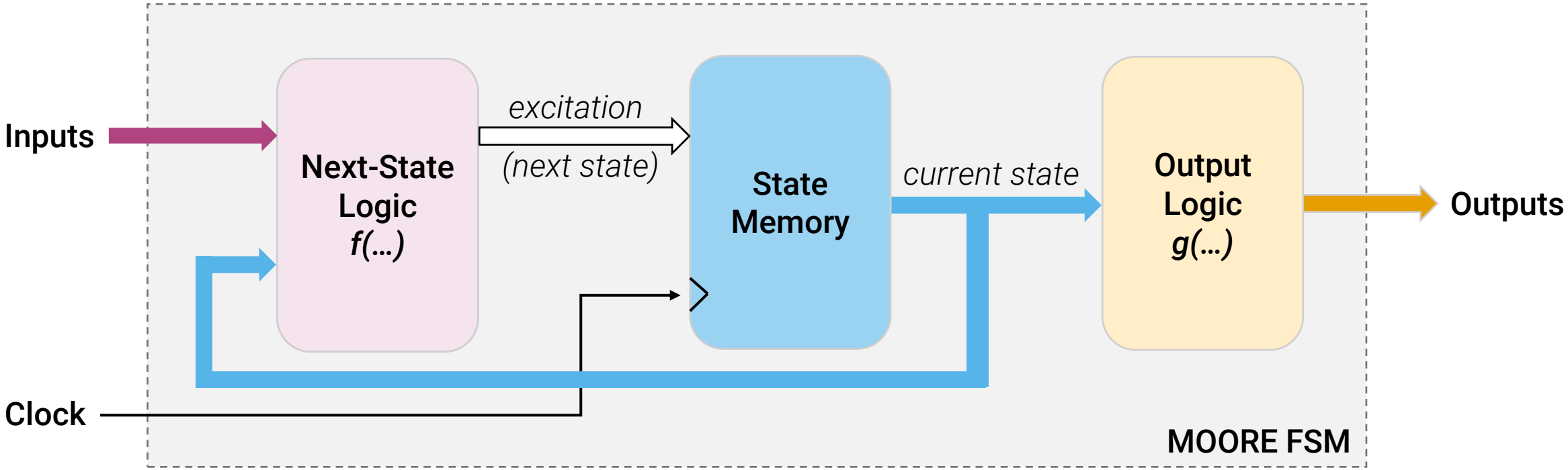
In Verilog



Note: Adjust the sensitivity list of the state memory, as required

Moore State Machine

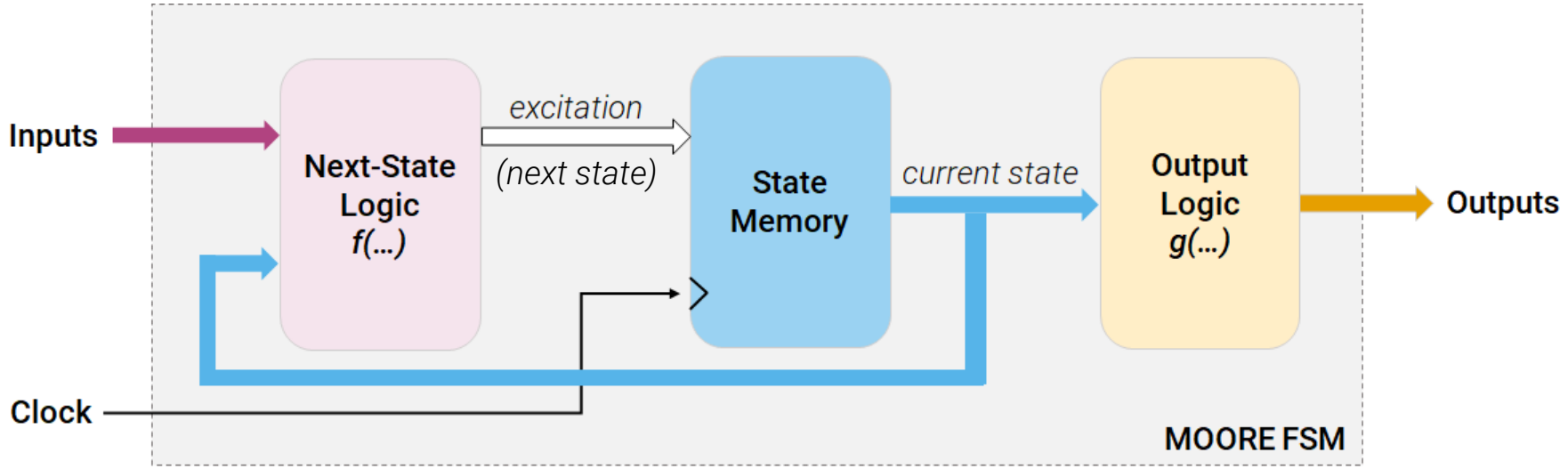
Structure



- If the outputs depend on the current state only, we have a **Moore state machine**

Moore State Machine

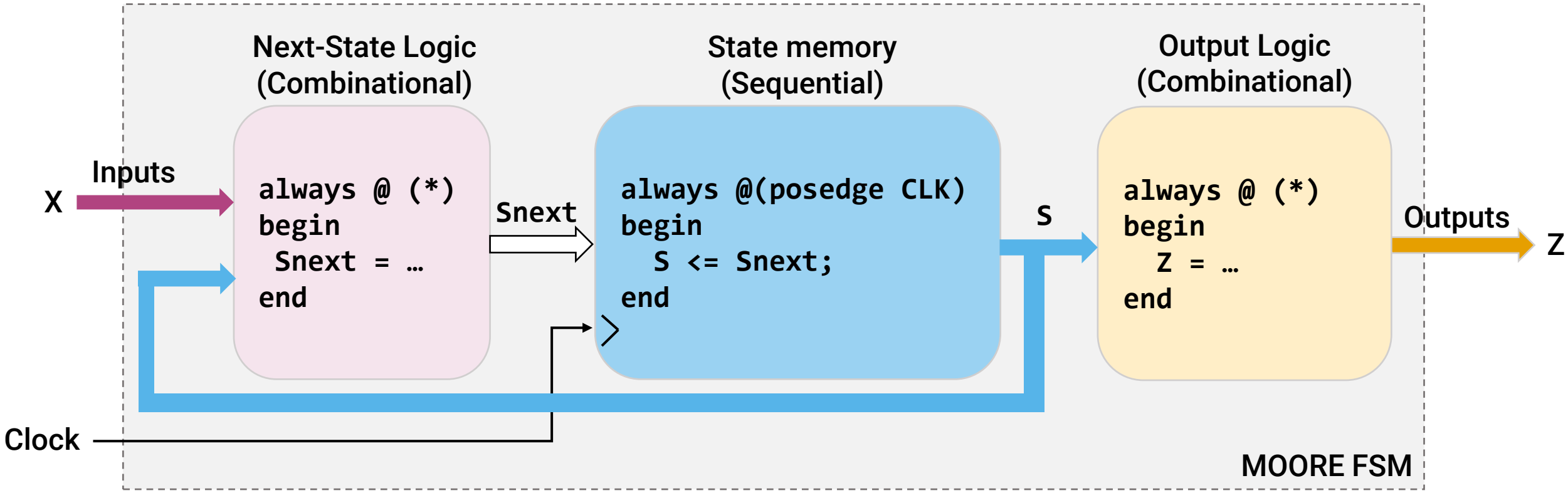
Structure



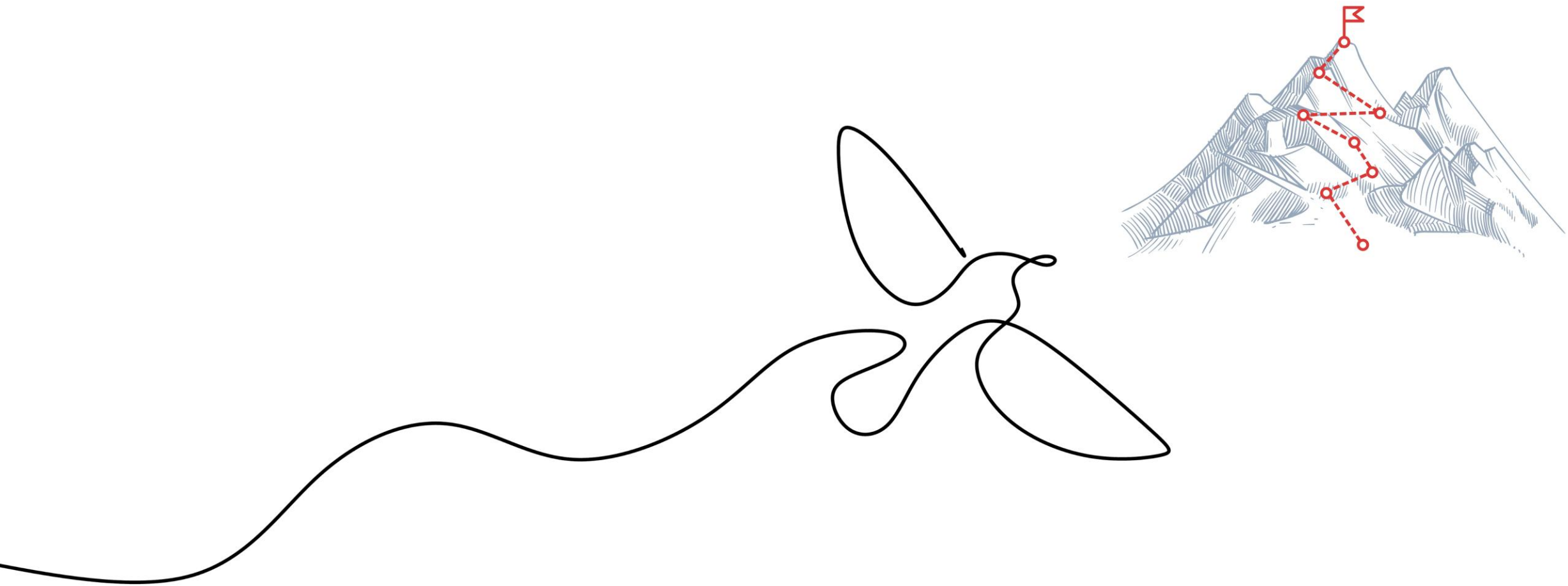
- The **next state** is a function of inputs and current state
  - *Next state = f(current state, input)*
- The **output** is a function of the current state only
  - *Output = g(current state)*
- Moore state machines are preferred (because there is no combinational path connecting inputs to outputs), whenever possible

Moore FSM Modeling

In Verilog



Note: Adjust the sensitivity list of the state memory, as required



State Machine Analysis

Analysis of a circuit containing combinational and sequential logic

State Machine Analysis

Algorithm

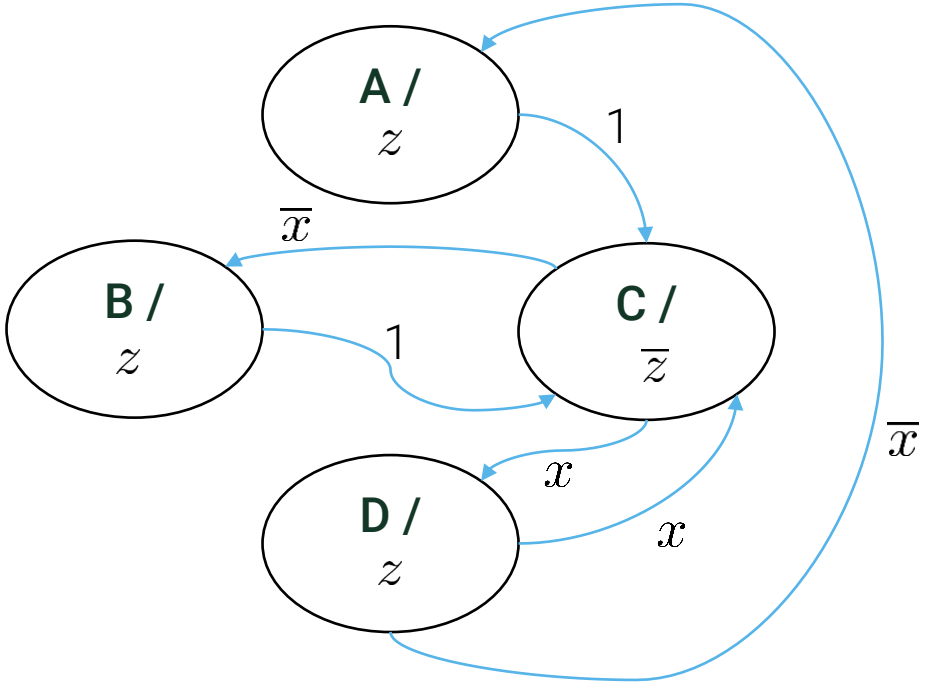
- State machine analysis involves three basic steps:
  - **Step 1:** Given a logic circuit, determine the next-state and output functions *f( )* and *g( ), respectively*
  - **Step 2:** Use *f( )* and *g( )* to construct a **state/output table** that completely specifies the next state and the output of the circuit for every possible combination of current state and inputs
  - **Step 3 (optional):** Draw a **state diagram** that presents the information from Step 2 in graphical form

- In practice, we more often design state machines than analyze them

What Are State Diagrams?

- Conceptually simplest method to describe the behavior of a sequential circuit is to draw a **state diagram**
  - A graph that depicts **states** of the circuit as nodes and **transition between states** as directed edges

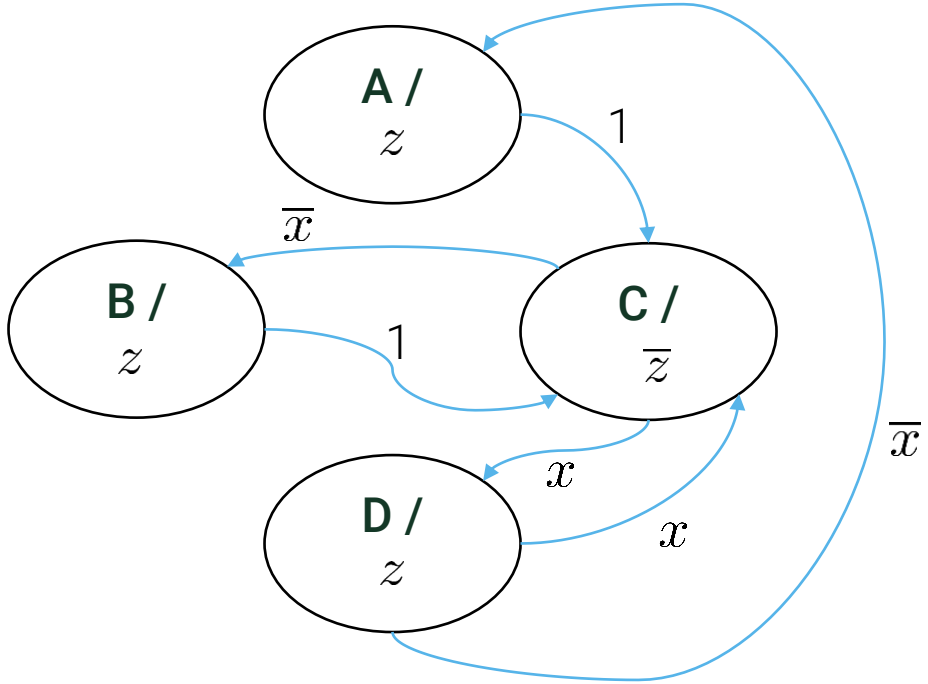
- *Example:*



State Diagrams, Contd.

- **Nodes** in a state diagram
  - Annotated with the state name
  - If Moore FSM, nodes are also annotated with the output value

- *Example:*

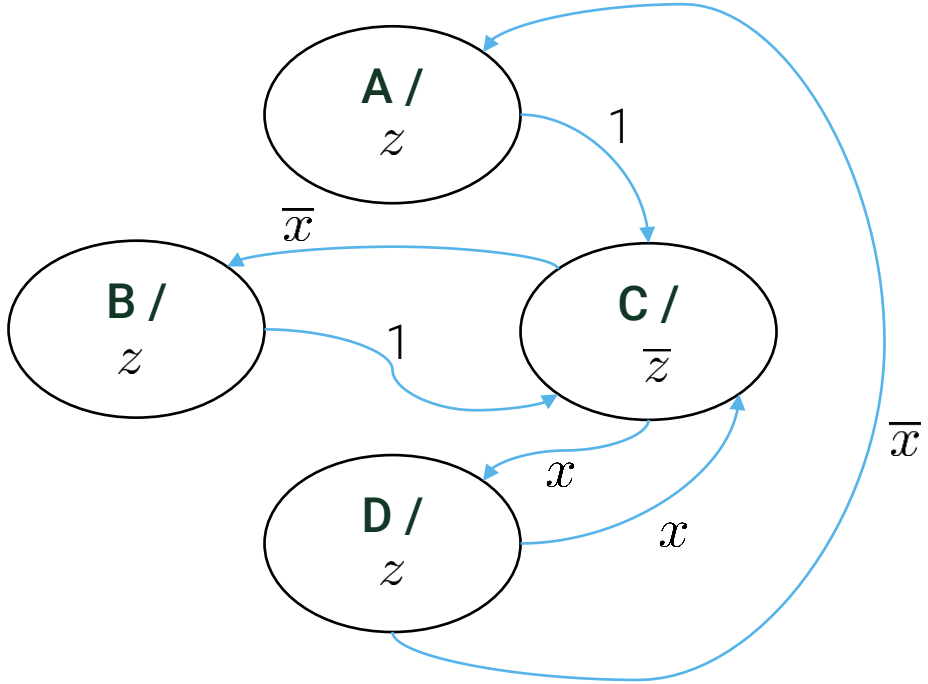


- *State names: A, B, C, D*
- *Output values: z (=1) or $\overline{z}$ (=0)*
  - */ acts as a separator*

State Diagrams, Contd.

- **Directed edges**
  in a state diagram
  - Annotated with the transition (signals) causing the change of state
  - State changes occur only in sync with clock
  - If Mealy FSM, edges are also annotated with the output value

- *Example:*

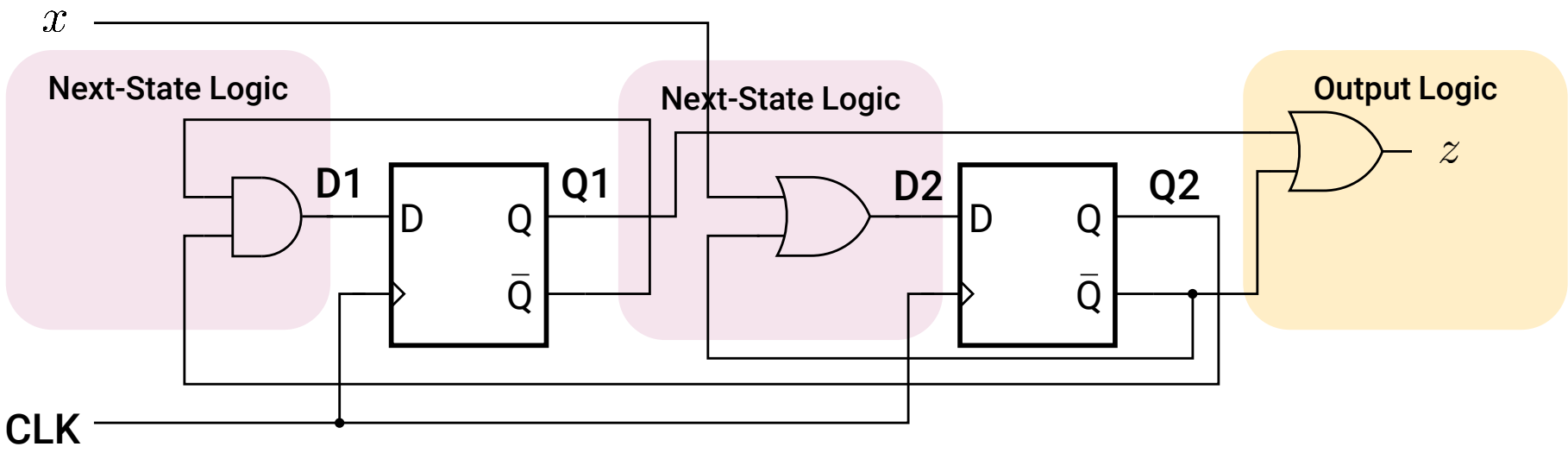


- *State names: A, B, C, D*
- *Transitions: 1 (unconditionally) or as a function of input x*

Example 1

State Machine Analysis

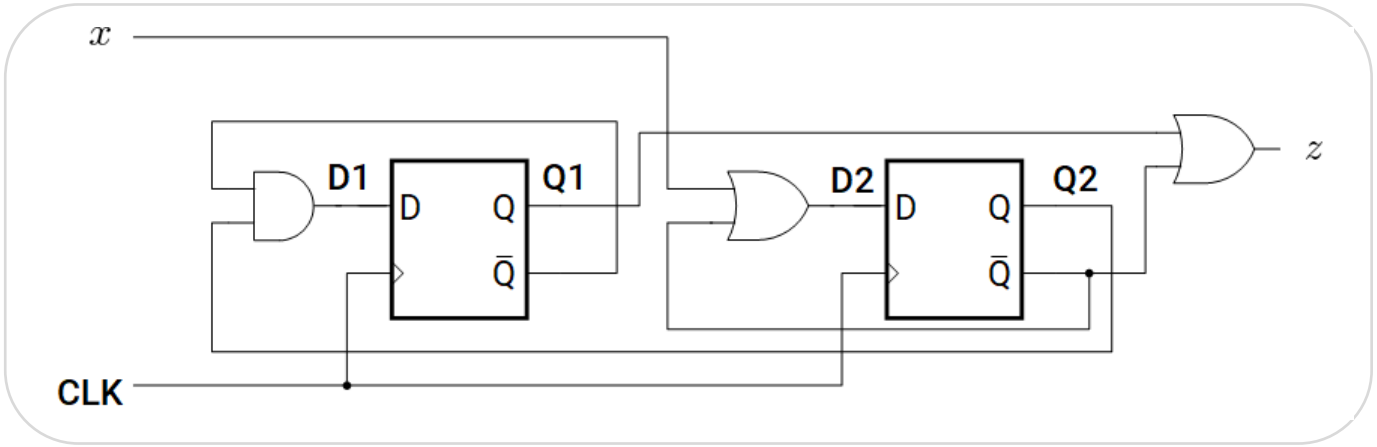
- Analyze the state machine implemented by the circuit below



- Which type of FSM is this?
  - This is a Moore state machine; outputs depend only on the state

Example 1

State Machine Analysis, Contd.



- **Step1**: Given a logic circuit, determine the next-state and output functions *f( )* and *g( )*

$$Q_1^* = D_1 = f_1(x, Q_1, Q_2) = \overline{Q_1} \cdot Q_2$$

$$Q_2^* = D_2 = f_2(x, Q_1, Q_2) = x + \overline{Q_2}$$

$$z = g(x, Q_1, Q_2) = Q_1 + \overline{Q_2}$$

** denotes next value of the state variable (next state)*

Example 1

State Machine Analysis, Contd.

$$Q_1^* = D_1 = f_1(x, Q_1, Q_2) = \overline{Q_1} \cdot Q_2$$

$$Q_2^* = D_2 = f_2(x, Q_1, Q_2) = x + \overline{Q_2}$$

$$z = g(x, Q_1, Q_2) = Q_1 + \overline{Q_2}$$

State variables	Input	Next state S* (Excitation)	Outputs
Q2 Q1	x	D2 D1	z
0 0	0	1 0	1
0 1	0	1 0	1
1 0	0	0 1	0
1 1	0	0 0	1
0 0	1	1 0	1
0 1	1	1 0	1
1 0	1	1 1	0
1 1	1	1 0	1

** denotes next value of the state variable (next state)*

Hey, we make our table more compact here!

State variables	Input	Next state (Excitation)	Outputs
Q2 Q1	x	D2 D1	z
0 0	0	1 0	1
0 1	0	1 0	1
1 0	0	0 1	0
1 1	0	0 0	1
0 0	1	1 0	1
0 1	1	1 0	1
1 0	1	1 1	0
1 1	1	1 0	1

Same sequence repeating

State variables	Next state S* (Excitation) D2 D1		Outputs
Q2 Q1	when x is		z
	0	1	
0 0	1 0	1 0	1
0 1	1 0	1 0	1
1 0	0 1	1 1	0
1 1	0 0	1 0	1

Note: As output does not depend on x, we can compact this table to avoid repetitions

Back to our Example…

State Machine Analysis, Contd.

- **Step 2:** Construct a state/output table
- Let us give the states some names (e.g., A, B, C, …)
  - 00 → A, 01 → B, 10 → C, 11 → D

State variables	Next state S* (Excitation) D2 D1		Outputs
Q2 Q1	when x is		z
	0	1	
0 0	1 0	1 0	1
0 1	1 0	1 0	1
1 0	0 1	1 1	0
1 1	0 0	1 0	1

State/output table

Current state, S	Next state, S*		Outputs
S	x		z
	0	1	
A	C	C	1
B	C	C	1
C	B	D	0
D	A	C	1

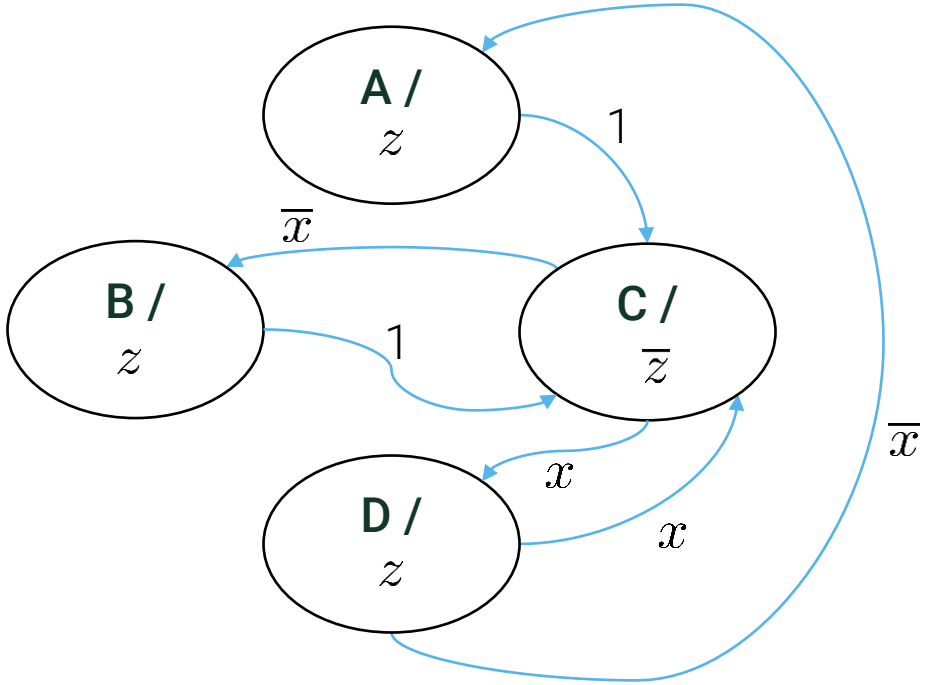
Example 1

State Machine Analysis, Contd.

- **Step 3:** Draw a state diagram
  - Present the information from the state/output table in a graphical way
  - One circle (node) for each state and an arrow (directed arc) for each state transition

State/output table

Current state, S	Next state, S*		Outputs
S	x		z
	0	1	
A	C	C	1
B	C	C	1
C	B	D	0
D	A	C	1



Example 1

State Machine in Verilog

$$Q_1^* = D_1 = f_1(x, Q_1, Q_2) = \overline{Q_1} \cdot Q_2$$
$$Q_2^* = D_2 = f_2(x, Q_1, Q_2) = x + \overline{Q_2}$$

$$z = g(x, Q_1, Q_2) = Q_1 + \overline{Q_2}$$

```
module fsm_example1 (
  input x,
  input CLK,
  output reg z
  );

  reg [2:1] D, Q;

  // Next-state logic
  always @ (*) begin
    D[1] = ~Q[1] & Q[2];
    D[2] = x | ~Q[2];
  end

  // State memory
  always @ (posedge CLK) begin
      Q <= D;
  end

  // Output logic
  always @ (*) begin
    z = Q[1] | ~Q[2];
  end
endmodule
```

Example 2

State Machine Analysis

- Analyze the state machine implemented by the circuit
- **Step 1:** Determine next-state transition and output functions

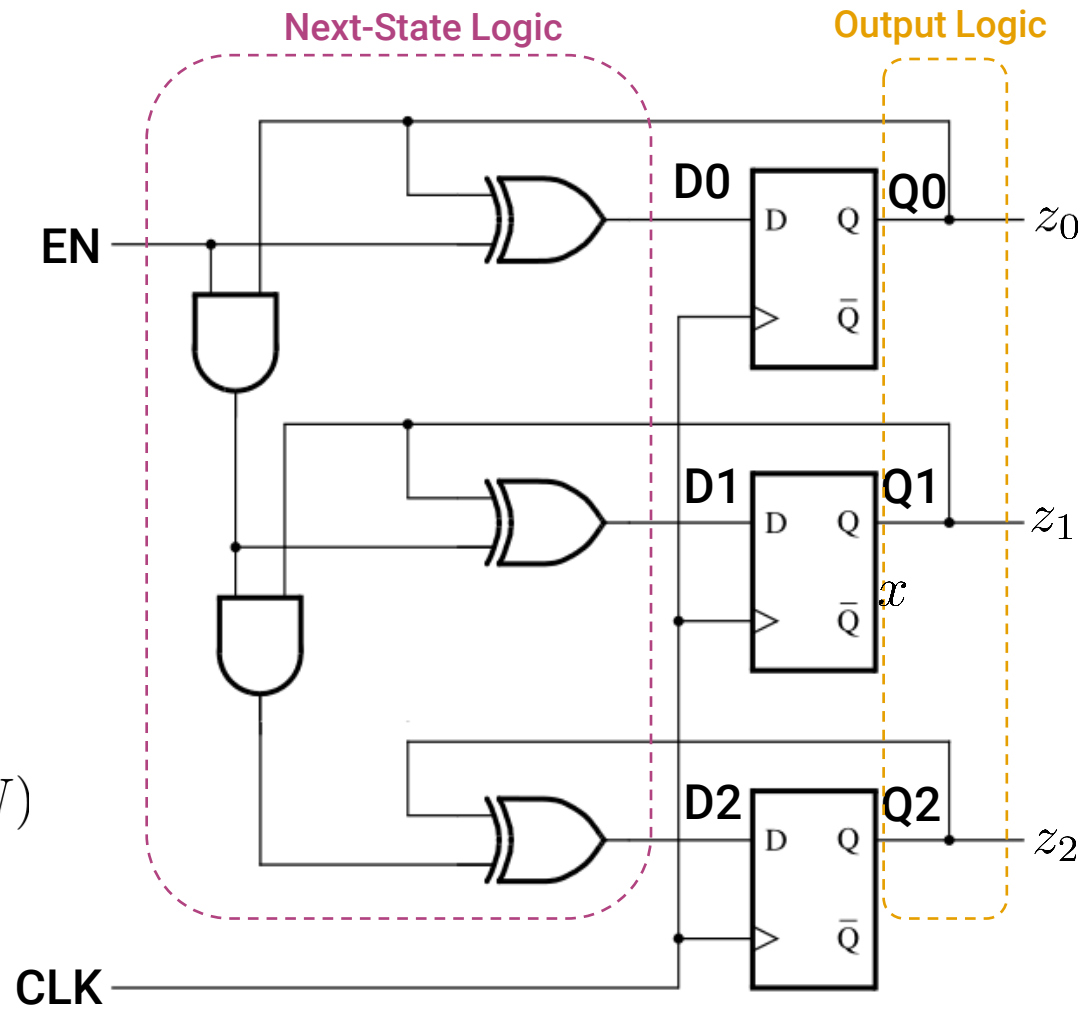
$$Q_0^* = D_0 = f_0(EN, Q_0, Q_1, Q_2) = Q_0 \oplus EN$$

$$Q_1^* = D_1 = f_1(EN, Q_0, Q_1, Q_2) = Q_1 \oplus (Q_0 EN)$$

$$Q_2^* = D_2 = f_2(EN, Q_0, Q_1, Q_2) = Q_2 \oplus (Q_0 Q_1 EN)$$

$$z_i = g_i(Q_i) = Q_i \ , \ 0 \leq i \leq 2$$

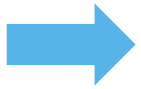
** denotes next value of the state variable*



Example 2

State Machine Analysis, Step 2

$$Q_0^* = D_0 = Q_0 \oplus EN$$
$$Q_1^* = D_1 = Q_1 \oplus (Q_0 EN)$$
$$Q_2^* = D_2 = Q_2 \oplus (Q_0 Q_1 EN)$$
$$z_i = Q_i \ , \ 0 \leq i \leq 2$$



State variables			Excitation D2 D1 D0					
			EN					
Q2	Q1	Q0	0			1		
0	0	0	0	0	0	0	0	1
0	0	1	0	0	1	0	1	0
0	1	0	0	1	0	0	1	1
0	1	1	0	1	1	1	0	0
1	0	0	1	0	0	1	0	1
1	0	1	1	0	1	1	1	0
1	1	0	1	1	0	1	1	1
1	1	1	1	1	1	0	0	0

Example 2

State Machine Analysis, Step 2

State variables			Excitation D2 D1 D0					
			EN					
Q2	Q1	Q0	0			1		
0	0	0	0	0	0	0	0	1
0	0	1	0	0	1	0	1	0
0	1	0	0	1	0	0	1	1
0	1	1	0	1	1	1	0	0
1	0	0	1	0	0	1	0	1
1	0	1	1	0	1	1	1	0
1	1	0	1	1	0	1	1	1
1	1	1	1	1	1	0	0	0

000→A
001→B
...
111→H

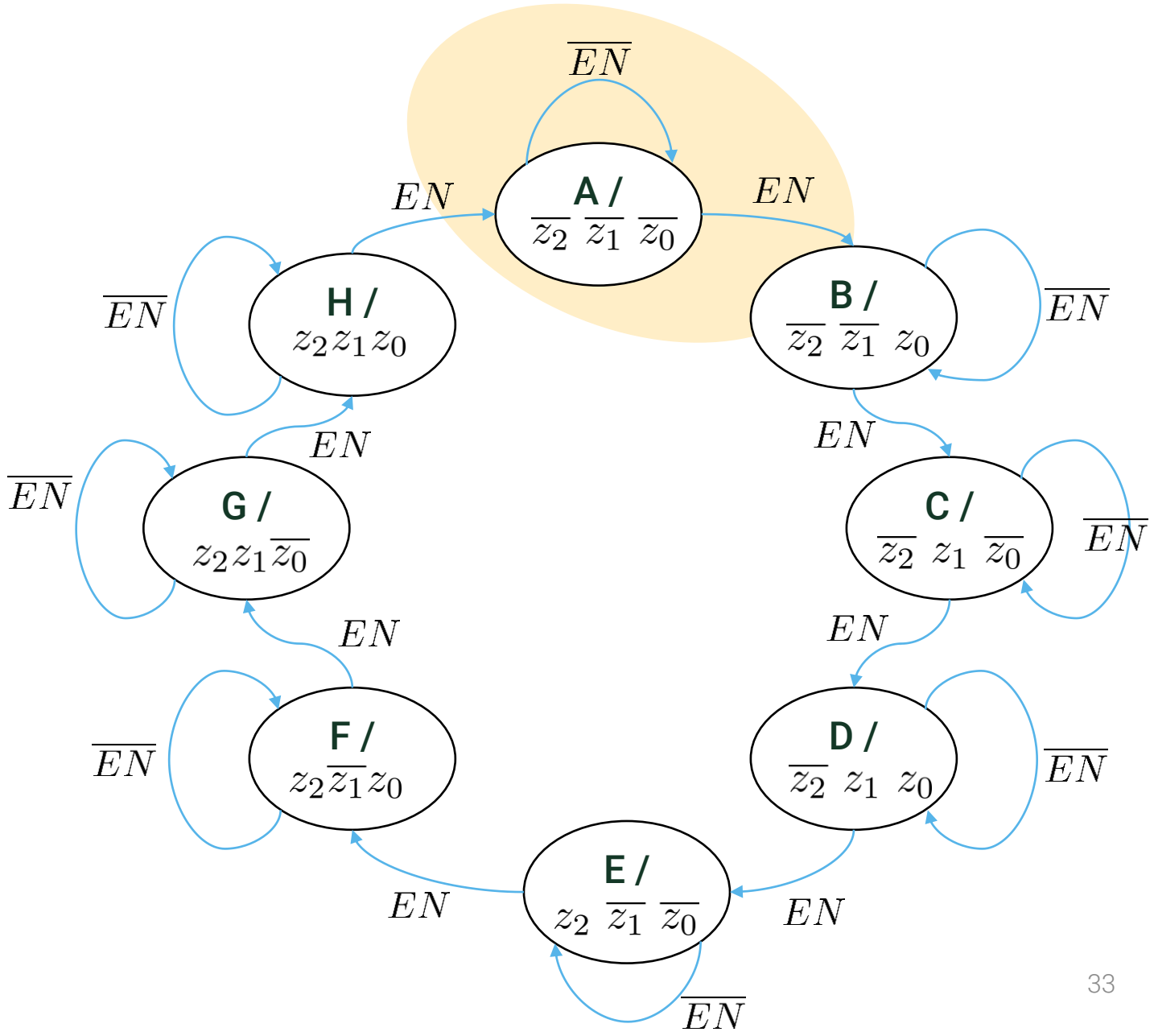
State/output table

Current state, S	Next state, S*		Outputs		
	EN				
S	0	1	z2	z1	z0
A	A	B	0	0	0
B	B	C	0	0	1
C	C	D	0	1	0
D	D	E	0	1	1
E	E	F	1	0	0
F	F	G	1	0	1
G	G	H	1	1	0
H	H	A	1	1	1

Example 2

State Machine Analysis, Step 3

Current state, S	Next state, S*		Outputs		
S	EN		z2	z1	z0
	0	1			
A	A	B	0	0	0
B	B	C	0	0	1
C	C	D	0	1	0
D	D	E	0	1	1
E	E	F	1	0	0
F	F	G	1	0	1
G	G	H	1	1	0
H	H	A	1	1	1



State Machine Analysis

- What can you tell about the circuit?

- A: The circuit is a 3-bit counter
  - If enabled (EN = 1), it counts up in steps of one. Once the max count is reached (i.e., 111), it returns to the initial value (i.e., 000)
  - Otherwise, it keeps the last computed value

  It is a Moore state machine (outputs depend only on the state).
  As there are three FFs, the FSM has 2^3 states.

Example 2

State Machine in Verilog

$$Q_0^* = D_0 = Q_0 \oplus EN$$
$$Q_1^* = D_1 = Q_1 \oplus (Q_0 EN)$$
$$Q_2^* = D_2 = Q_2 \oplus (Q_0 Q_1 EN)$$

Verilog operators:
& - bitwise AND
| - bitwise OR
~ - bitwise NOT
^ - bitwise XOR

$$z_i = Q_i \ , \ 0 \leq i \leq 2$$

```
// equivalent to
// output [2:0] z
assign z = Q;
```

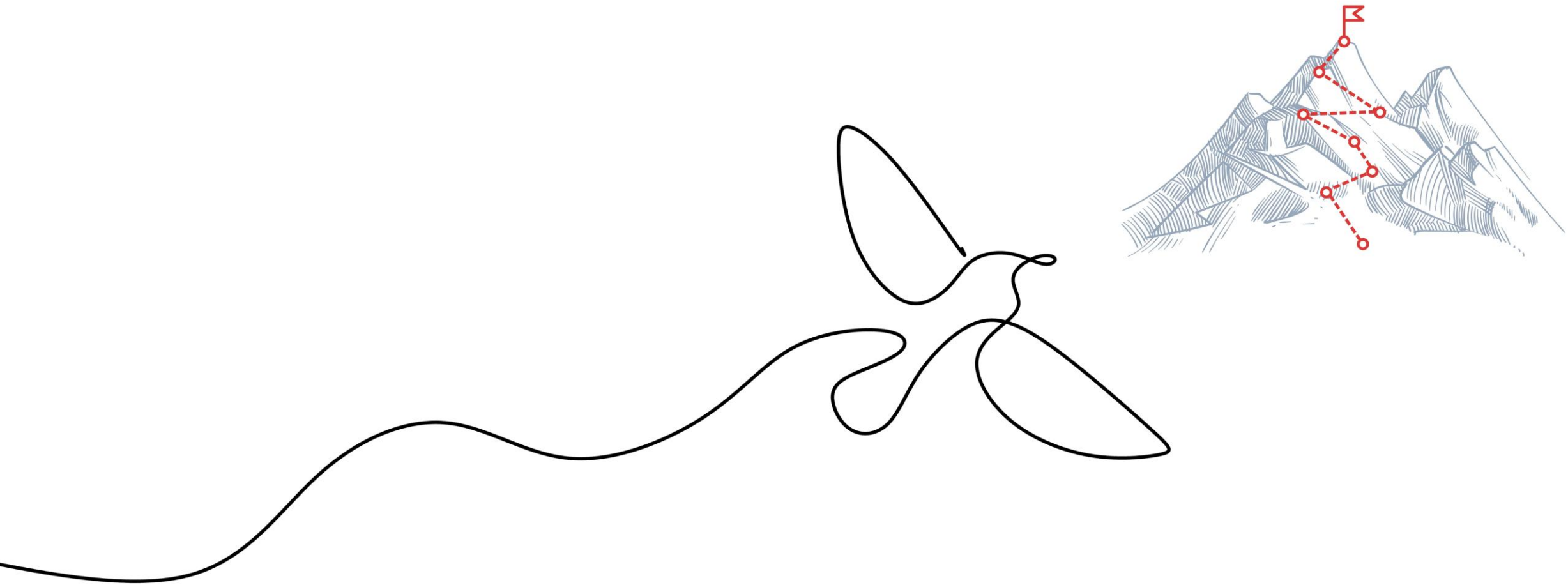
```
module fsm_example2 (
  input EN,
  input CLK,
  output reg [2:0] z
  );

  reg [2:0] D, Q;

  // Next-state logic
  always @ (*) begin
    D[0] = Q[0] ^ EN;
    D[1] = Q[1] ^ (Q[0] & EN);
    D[2] = Q[2] ^ (Q[1] & Q[0] & EN);
  end

  // State memory
  always @ (posedge CLK) begin
      Q <= D;
  end

  // Output logic
  always @ (*) begin
    z = Q;
  end
endmodule
```

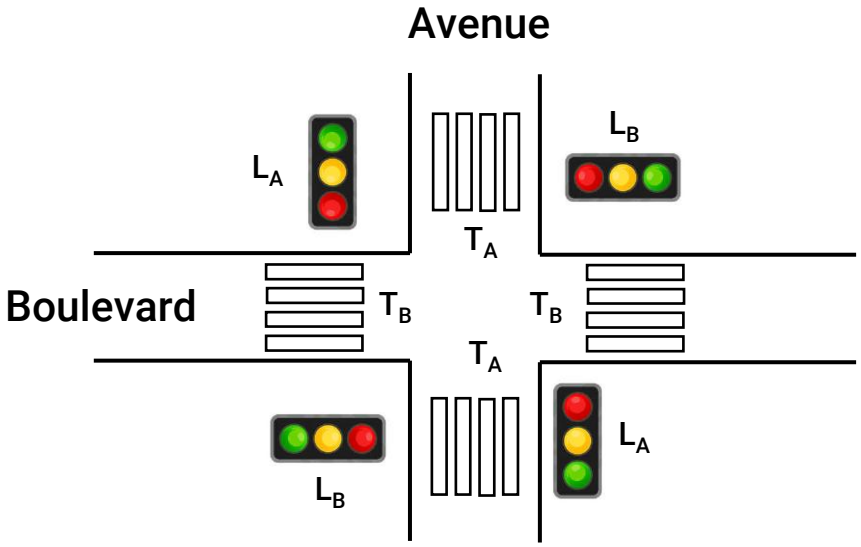


State Machine Synthesis

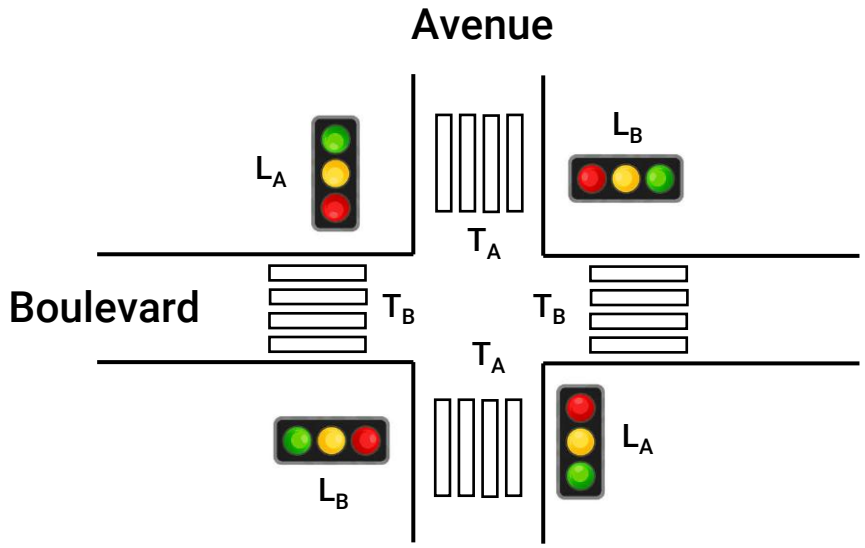
Much more common scenario in practice

FSM Design Using an Example

- Consider that we need to design a traffic light controller
  - T_A, T_B : Signals from the traffic sensors; active when there is traffic
  - L_A, L_B: Signals for controlling traffic lights



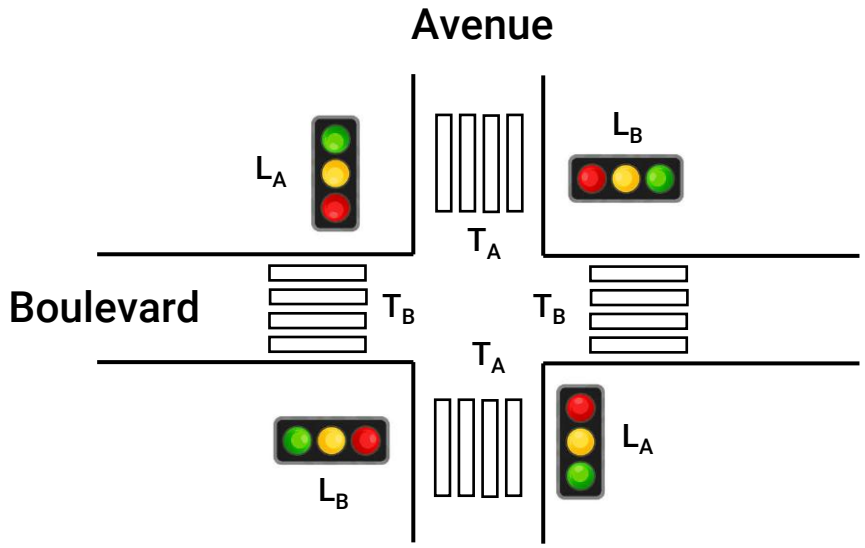
FSM Design Algorithm



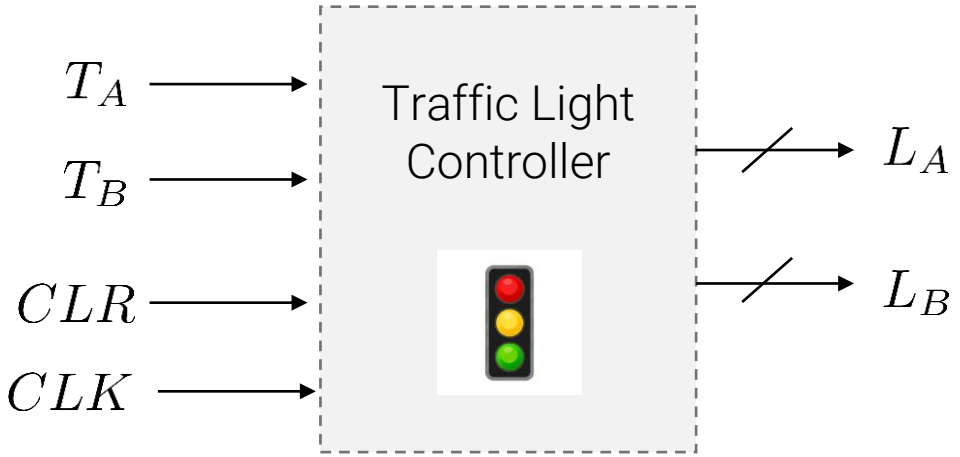
- Identify FSM inputs and outputs
- Draw state transition diagram
- Construct state table
  - Select state encodings (binary vectors)
  - Rewrite state table with state encodings
  - Write logic expressions for next state
- Construct output table
  - Select output encodings (binary vectors)
  - Write logic expressions for output
- Construct the equivalent logic circuit (drawing, Verilog)

Traffic Light Controller

Example



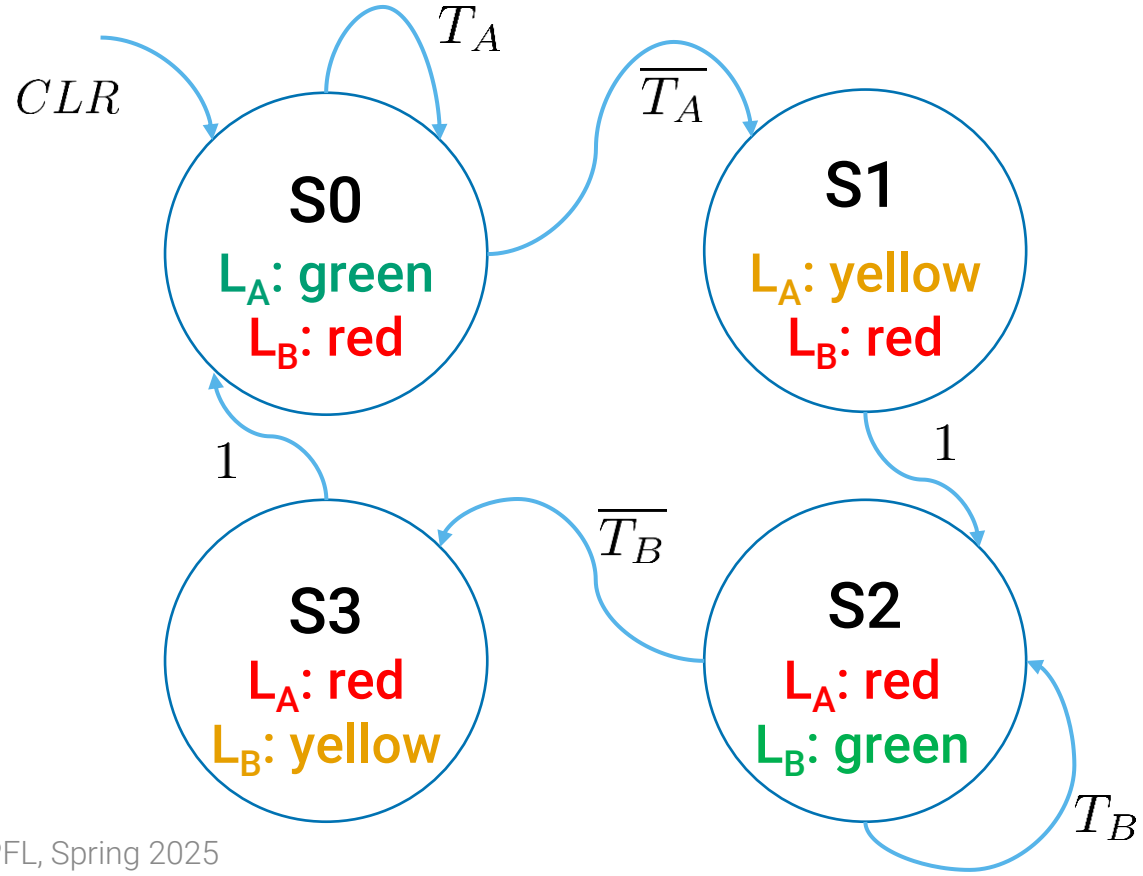
- Identify FSM inputs and outputs
  - Two one-bit inputs from the traffic sensors
  - CLK, system clock
  - CLR, synchronous reset for clearing the state memory
  - Multiple bits for the outputs (there are three lights: red, yellow, green)



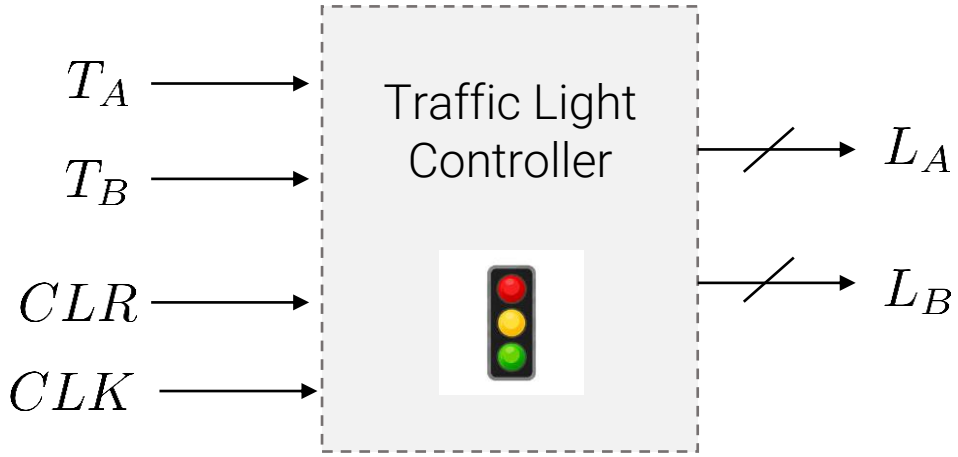
Traffic Light Controller

Example, Contd.

- Draw state transition diagram



Recall…



Traffic Light Controller

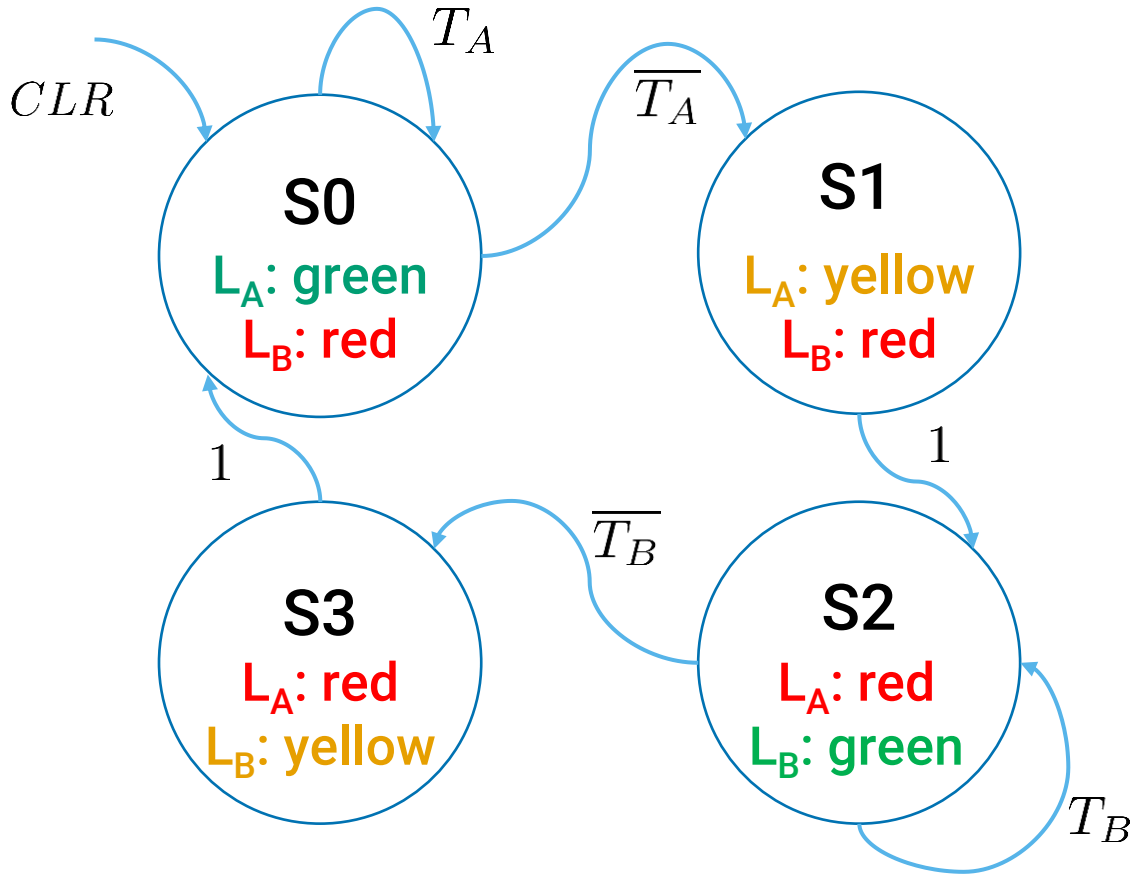
Example, Contd.

- Construct state table

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

X stands for 0/1 (i.e., both options)

Recall…



Traffic Light Controller

Example, Contd.

- Select state encodings
  - Four states; hence, two bits suffice to encode all states
  - Example state encoding:

State	Encoding
S0	00
S1	01
S2	10
S3	11

Note: Choice of encoding impacts implementation; in practice, we prefer to let tools infer best encoding

Recall…

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

X stands for 0/1 (i.e., both options)

Traffic Light Controller

Example, Contd.

- Rewrite state table

Current State, S	Inputs		Next State, S*
Q1 Q0	T_A	T_B	D1 D0
0 0	1	X	0 0
0 0	0	X	0 1
0 1	X	X	1 0
1 0	X	1	1 0
1 0	X	0	1 1
1 1	X	X	0 0

X stands for 0/1 (i.e., both options)

Next-State Logic

$$Q_1^* = D_1 = \overline{Q_1}\ Q_0 + Q_1\ \overline{Q_0} = Q_0 \oplus Q_1$$

$$Q_0^* = D_0 = \overline{Q_1}\ \overline{Q_0}\ \overline{T_A} + Q_1\ \overline{Q_0}\ \overline{T_B}$$

Recall…

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

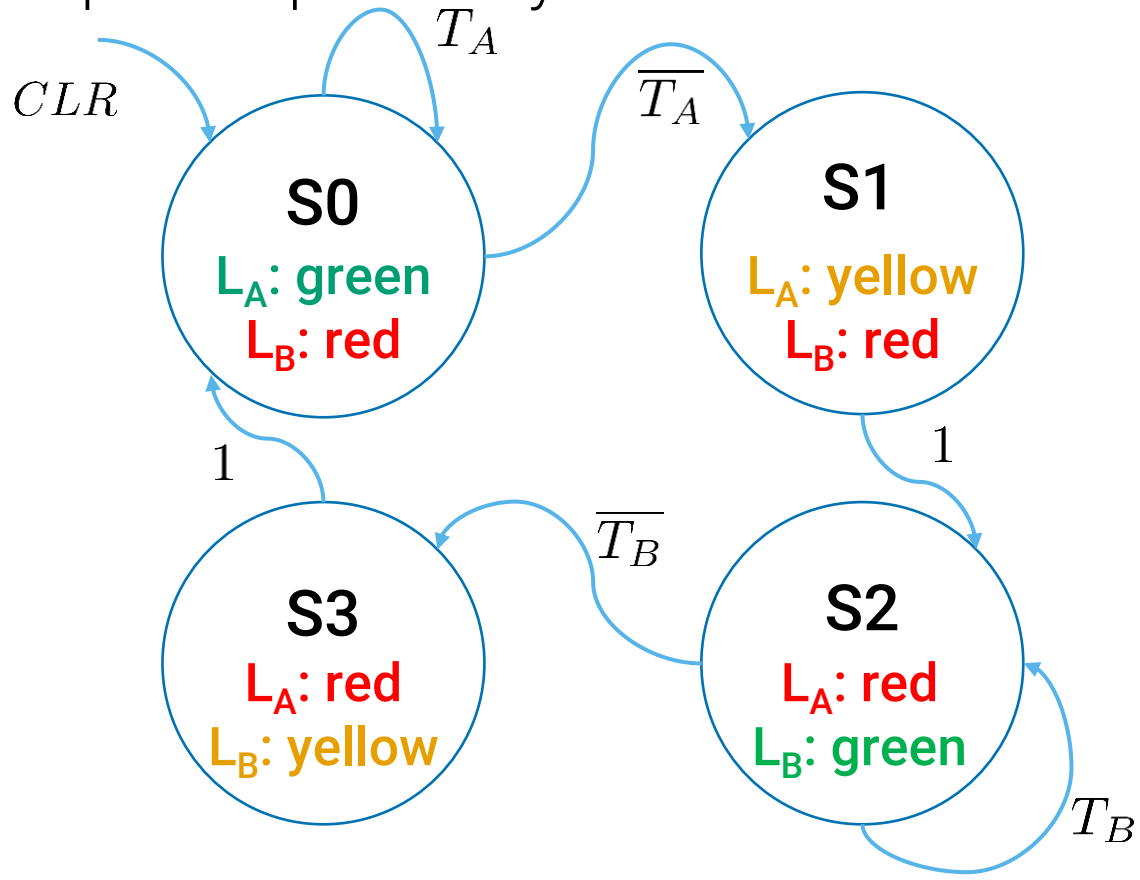
X stands for 0/1 (i.e., both options)

State	Encoding
S0	00
S1	01
S2	10
S3	11

Traffic Light Controller

Example, Contd.

- Construct output table; select output encodings
  - Moore FSM: outputs depend only on the state



Traffic Light Controller

Example, Contd.

- Construct output table; select output encodings
  - Moore FSM: outputs depend only on the state
  - Three different outputs, hence two bits required to represent them
    - L_A: Two bits (L_{A1}, L_{A0})
    - L_B: Two bits (L_{B1}, L_{B0})

- For the encoding given in the table:
  - If L_A is green: $L_{A1} = 0$, $L_{A0} = 0$;
  - If L_B is yellow: $L_{B1} = 0$, $L_{B0} = 1$;
  - etc.

Output	Encoding
GREEN	00
YELLOW	01
RED	10

Note: Choice of encoding impacts implementation; in practice, we prefer to let tools infer best encoding

Traffic Light Controller

Example, Contd.

- Construct output table; write logic expressions

Current State, S		Outputs			
Q1	Q0	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

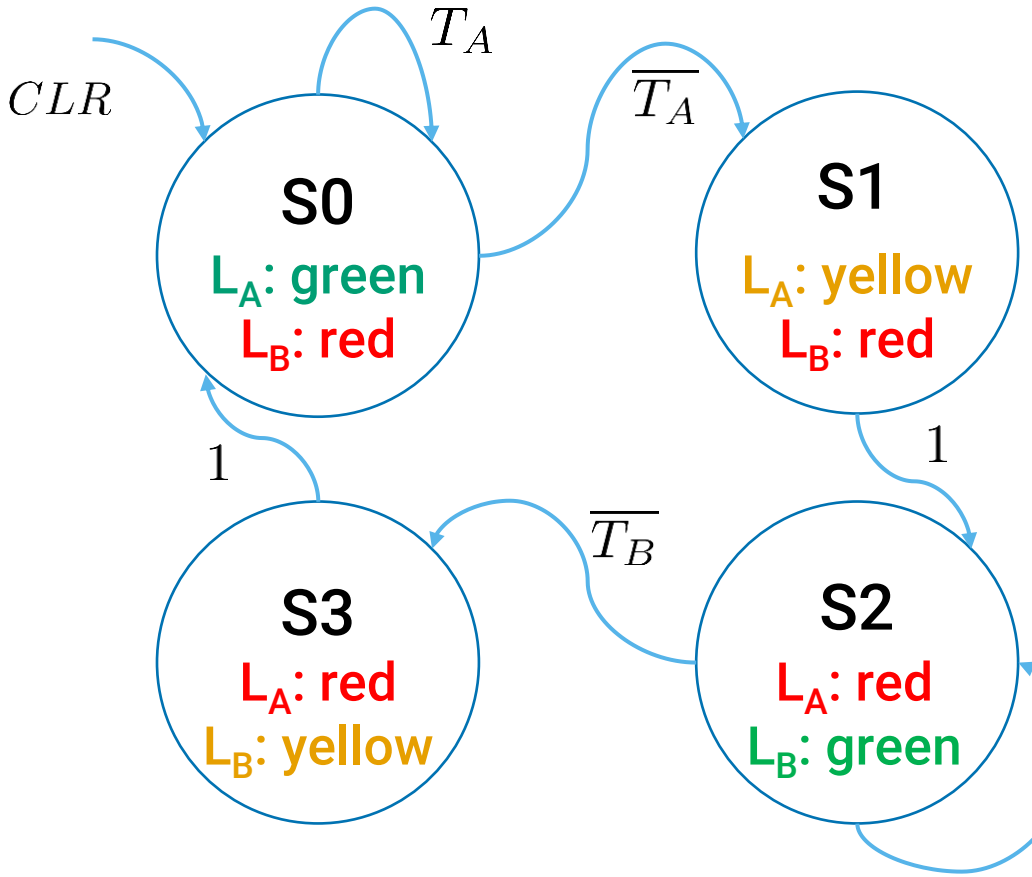
Output Logic

$$L_{A1} = Q_1 \qquad L_{B1} = \overline{Q_1}$$
$$L_{A0} = \overline{Q_1}\, Q_0 \qquad L_{B0} = Q_1 Q_0$$

Recall…

State	Encoding
S0	00
S1	01
S2	10
S3	11

Output	Encoding
GREEN	00
YELLOW	01
RED	10



Traffic Light Controller

Example, Contd.

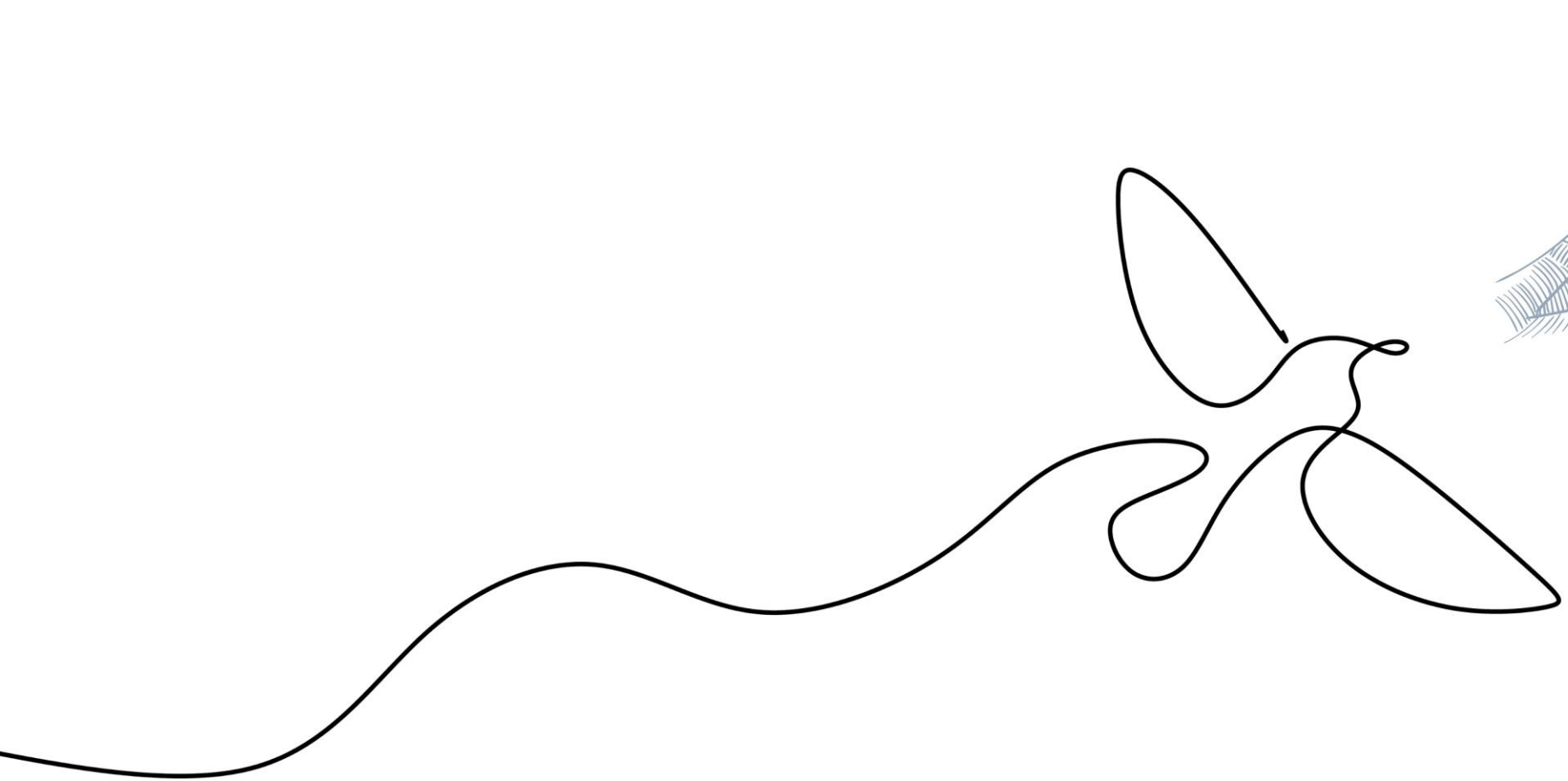
Next-State Logic

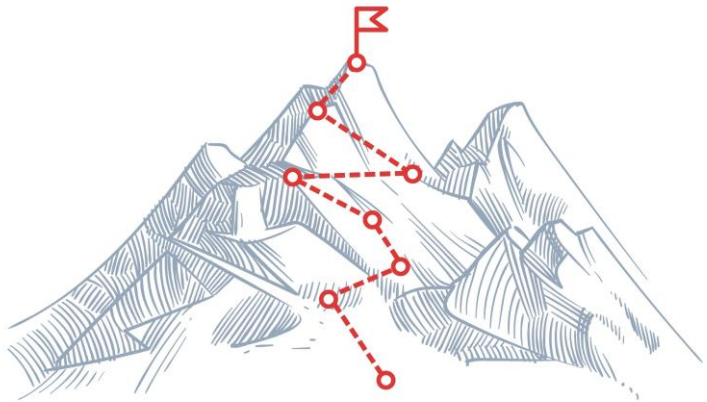
$$Q_1^* = D_1 = \overline{Q_1}\ Q_0 + Q_1\ \overline{Q_0} = Q_0 \oplus Q_1$$
$$Q_0^* = D_0 = \overline{Q_1}\ \overline{Q_0}\ \overline{T_A} + Q_1\ \overline{Q_0}\ \overline{T_B}$$

Output Logic

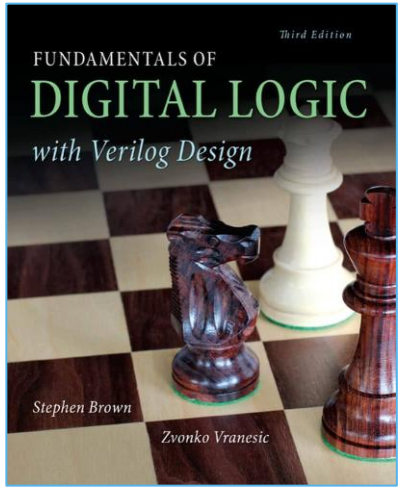
$$L_{A1} = Q_1 \qquad L_{B1} = \overline{Q_1}$$
$$L_{A0} = \overline{Q_1}\ Q_0 \qquad L_{B0} = Q_1 Q_0$$

```
module traffic_light_ctrl (
  input TA, TB, CLR, CLK,
  output reg [1:0] LA, LB
  );
  reg [1:0] D, Q;
  // Next-state logic
  always @ (*) begin
    D[1] = Q[0] ^ Q[1];
    D[0] = (~Q[1] & ~Q[0] & ~TA)
           | (Q[1] & ~Q[0] & ~TB);
  end
  // State memory
  always @ (posedge CLK)
  begin
    if (CLR == 1) Q <= 0;
    else Q <= D;
  end
  // Output logic
  always @ (*) begin
    LA[1] = Q[1]; LA[0] = ~Q[1] & Q[0];
    LB[1] = ~Q[1]; LB[0] = Q[1] & Q[0];
  end
endmodule
```

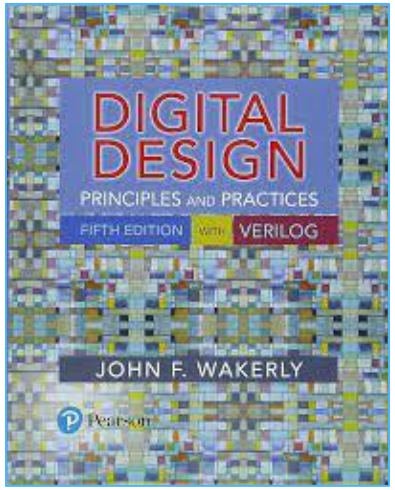




Literature



- Chapter 6: Synchronous Sequential Circuits
  - 6.1.1, 6.1.2, 6.3. 6.4.1



- Chapter 9: State Machines
  - 9.1-9.3