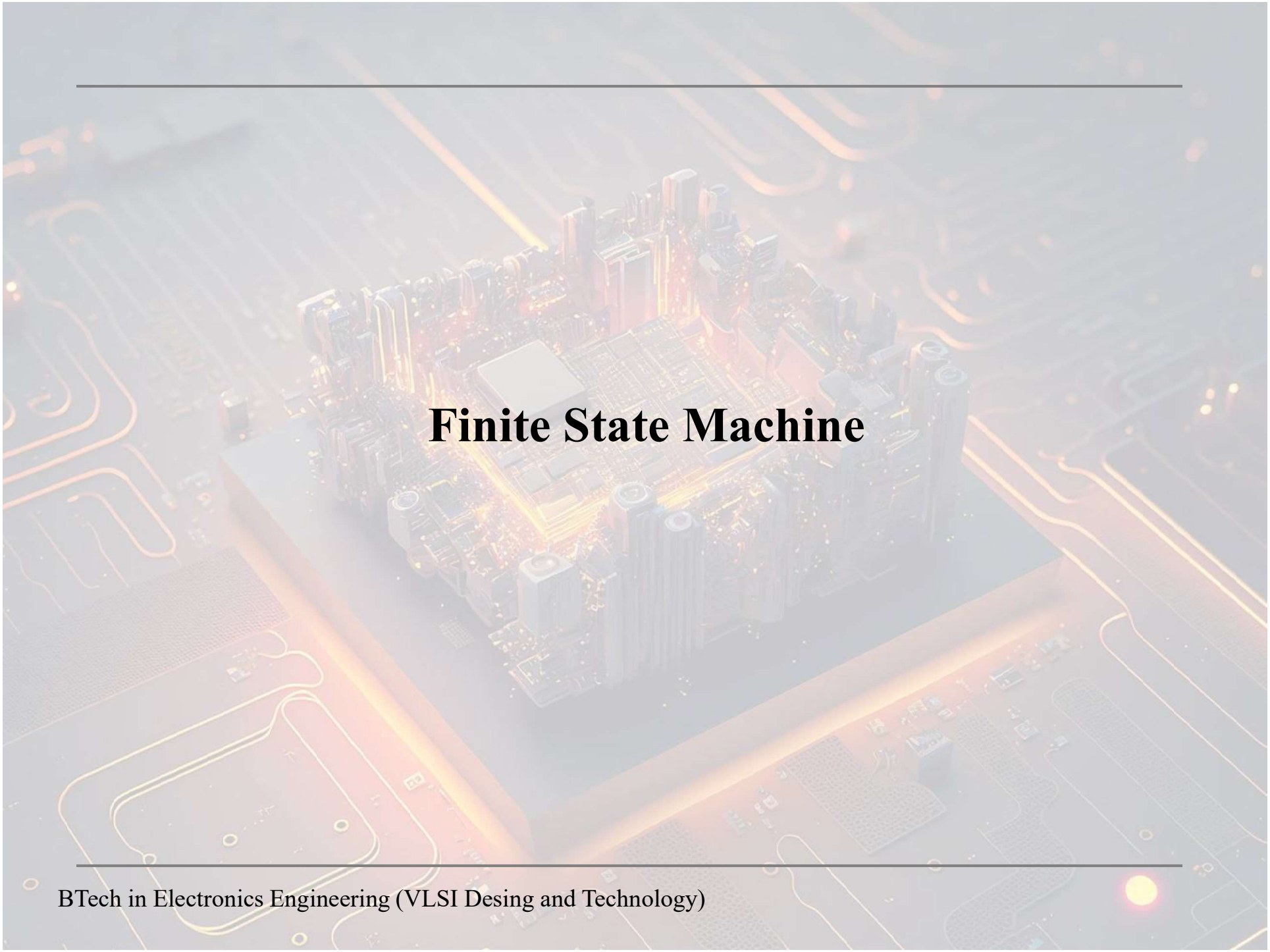
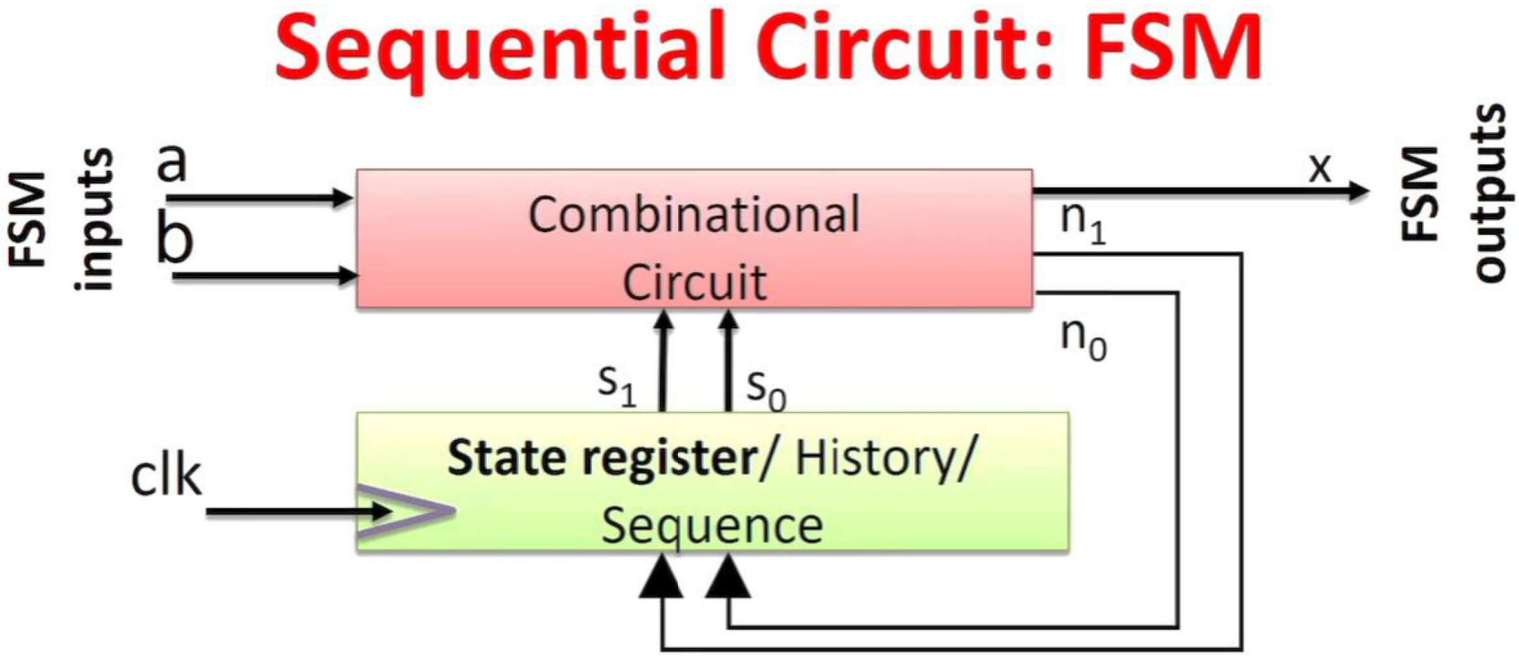


Finite State Machine

Finite State Machine



Formally Describe/mathematically Describe

Boolean Algebra: Working of Combinational Circuit

Finite State Machine: Working of Sequential Circuit

X (t) = F (a(t),b(t), H)

H is History/Sequence /State

Finite State Machine

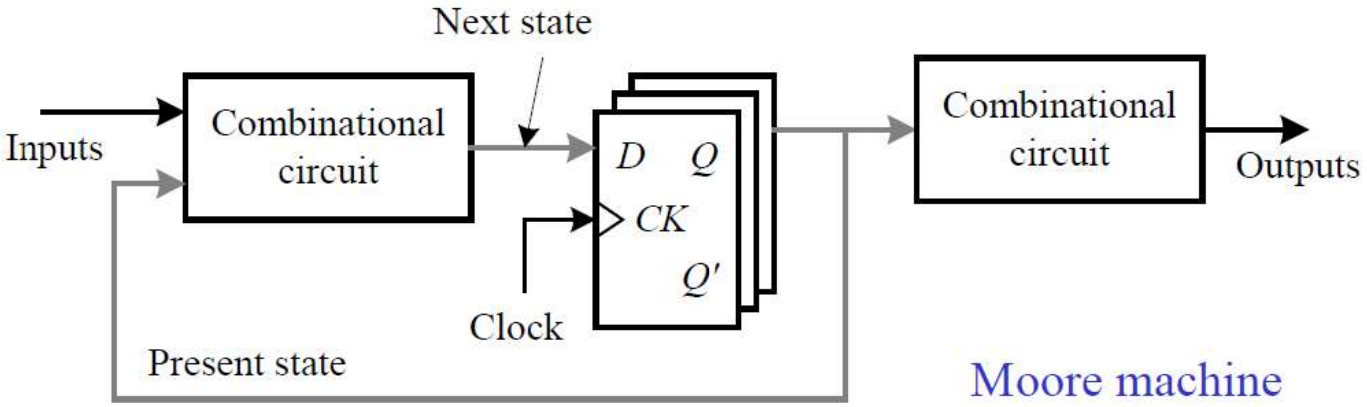
Already Designed Sequential Circuit

- Flip Flops (RS, JK, T, D)
- Register (Shift, PIPO), Memory
- Counter : Sync, Modulo Counter
- Counter : Using Shift Register

Till now we have not used formal Approach to design these

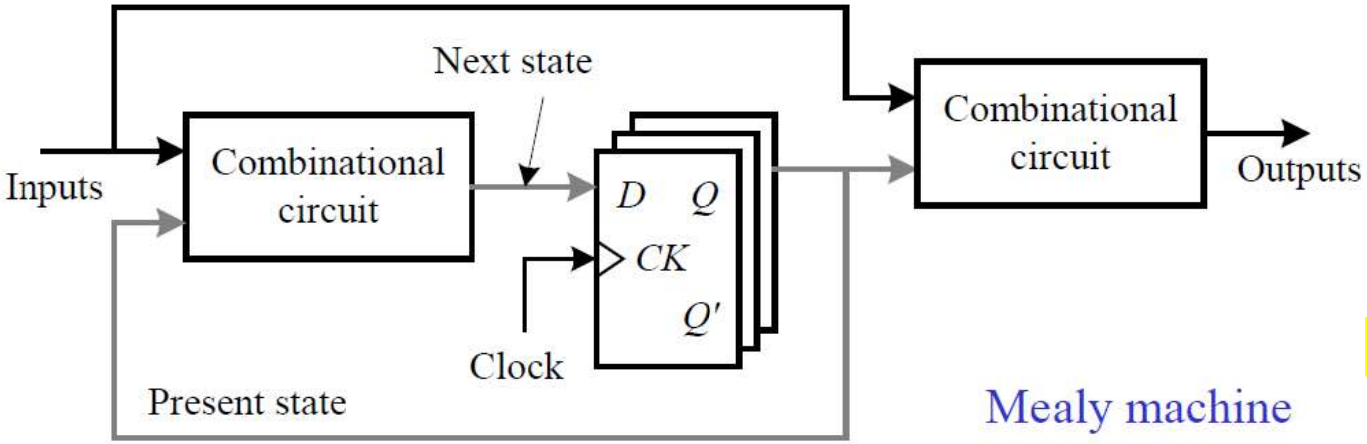
Finite State Machine

Two types of FSM



Moore machine

Output is only a function of state.



Mealy machine

Output is a function of state and input.

Finite State Machine

Moore Machine is an ordered quintuple

$$Moore = (\mathbf{S}, \mathbf{I}, \mathbf{O}, \delta, \lambda)$$

where

$$\mathbf{S} = \{s_1, s_2, \cdots, s_n\}$$

$$\mathbf{I} = \{i_1, i_2, \cdots, i_m\}$$

$$\mathbf{O} = \{o_1, o_2, \cdots, o_l\}$$

$$\delta = \mathbf{S} \times \mathbf{I} \rightarrow \mathbf{S}$$

$$\lambda = \mathbf{S} \rightarrow \mathbf{O}$$

S= finite set of state

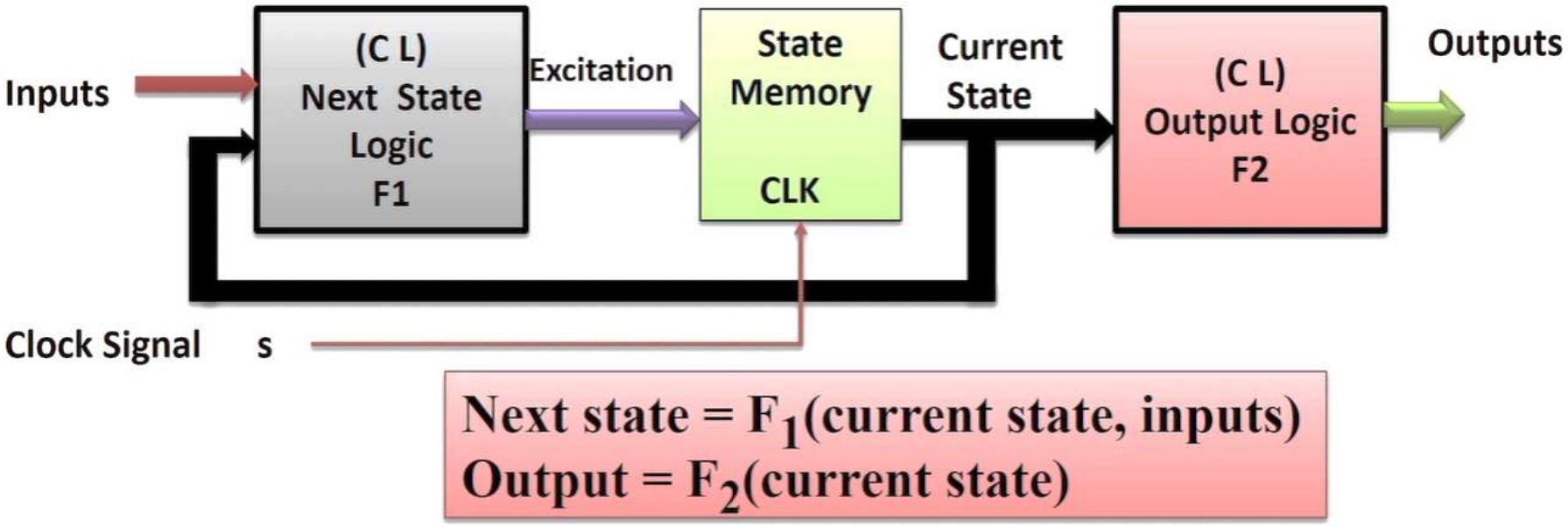
I = finite set of inputs

O=finite set of output

δ=next state function

λ= output function

Finite State Machine



Next state = F_1(current state, inputs)
Output = F_2(current state)

Moore Machine

Finite State Machine

Mealy Machine is an ordered quintuple

$$\text{Mealy} = (\mathbf{S}, \mathbf{I}, \mathbf{O}, \delta, \beta)$$

where

$$\mathbf{S} = \{s_1, s_2, \cdots, s_n\}$$

$$\mathbf{I} = \{i_1, i_2, \cdots, i_m\}$$

$$\mathbf{O} = \{o_1, o_2, \cdots, o_l\}$$

$$\delta = \mathbf{S} \times \mathbf{I} \rightarrow \mathbf{S}$$

$$\beta = \mathbf{S} \times \mathbf{I} \rightarrow \mathbf{O}$$

S= finite set of state

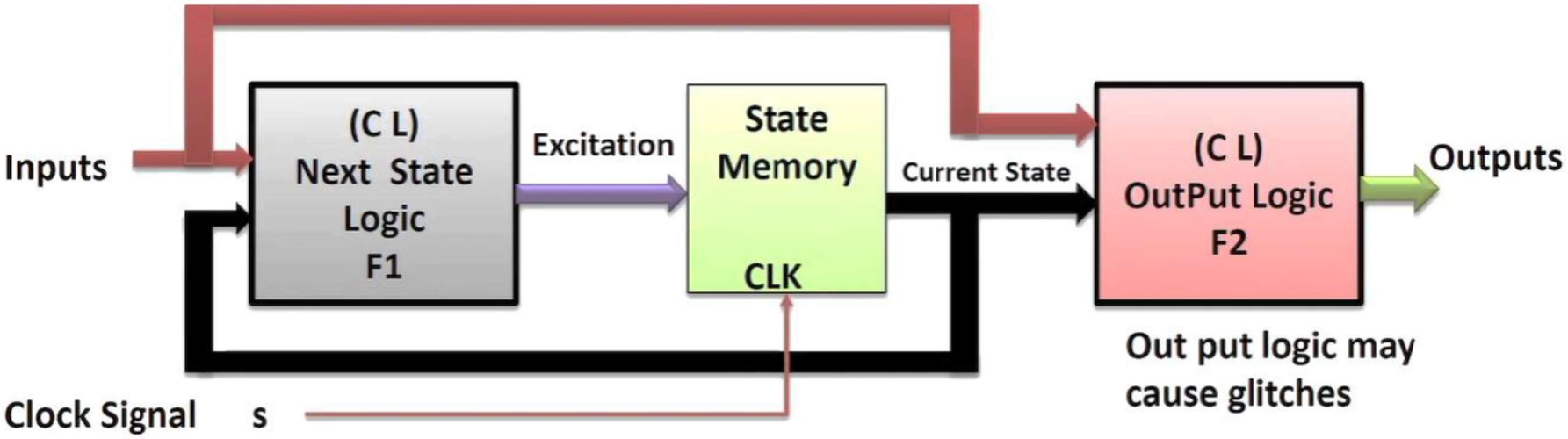
I = finite set of inputs

O=finite set of output

δ=next state function

λ= output function

Finite State Machine



State Storage= Set of n FFs
2^n State can be stored

Next state = F_1(current state, inputs)
Output = F_2(current state, inputs)

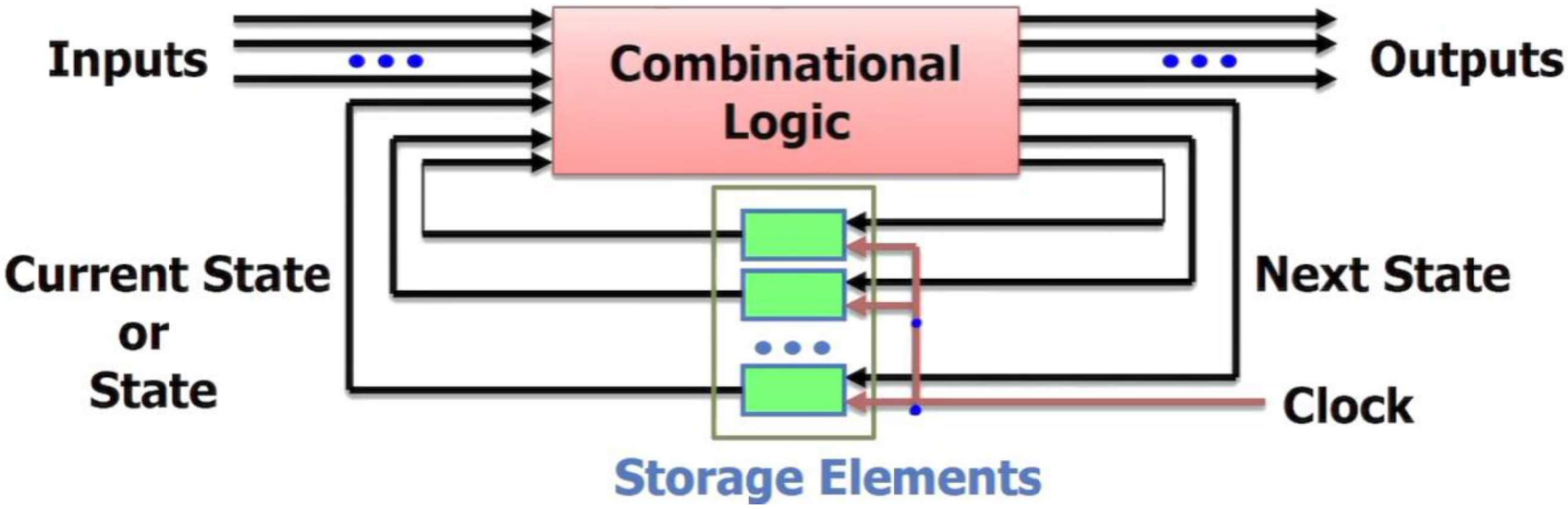
Melay Machine

Clocked FSM

- **Clocked**
  - All storage elements employ a clock input
  - (i.e. all storage elements are flip-flops)
- **Synchronous**
  - All of the flip flops use the same clock signal
- **FSM**
  - State machine is simply another name for sequential circuits.
  - Finite refers to the fact that the number of states the circuit can assume if finite

Clocked FSM

- **States**: Determined by possible values in sequential storage elements
- **Transitions**: Change of state
- **Clock**: Controls when state can change by controlling storage



Finite State Machine - Design

The steps for designing a state machine are:

1. **Understand the problem**: Understand the requirements of the system you are designing.

Minimization

2. **Identify states**: Determine the different states the system can be in.

3. **Identify inputs and outputs**: Determine what the system can receive and produce.

State Encoding

4. **Assign values**: Assign values to each input and output, and to each state.

5. **Create a state transition diagram or table**: Show how the system transitions between states.

Depending on FF

6. **Design the next-state logic**: Determine how the system will transition from one state to the next.

7. **Implement the state machine**: Put the design into action.

Finite State Machine - Modeling Styles

- Style 1: one state register
  - part 1: initialize and update the state register

```
always @(posedge clk or negedge reset_n)
    if   (!reset_n) state <= A;
    else  state <= next_state (state, x) ;
```

  - part 2: determine next state using function

```
function next_state (input present_state, x)
    case (present_state ) …
endcase endfunction
```

  - part 3: determine output function

```
always @(state or x) case (state) … or
    assignment statements
```

Finite State Machine - Modeling Styles

- Style 2: two state registers
  - part 1: initialize and update the state register

```
always @(posedge clk or negedge reset_n)
    if (!reset_n) present_state <= A;
    else  present_state <= next_state;
```

  - part 2: determine next state

```
always @(present_state or x) case (present_state ) … or
assignment statements
```

  - part 3: determine output function

```
always @(present_state or x) case (present_state ) … or
assignment statements
```

Finite State Machine - Example

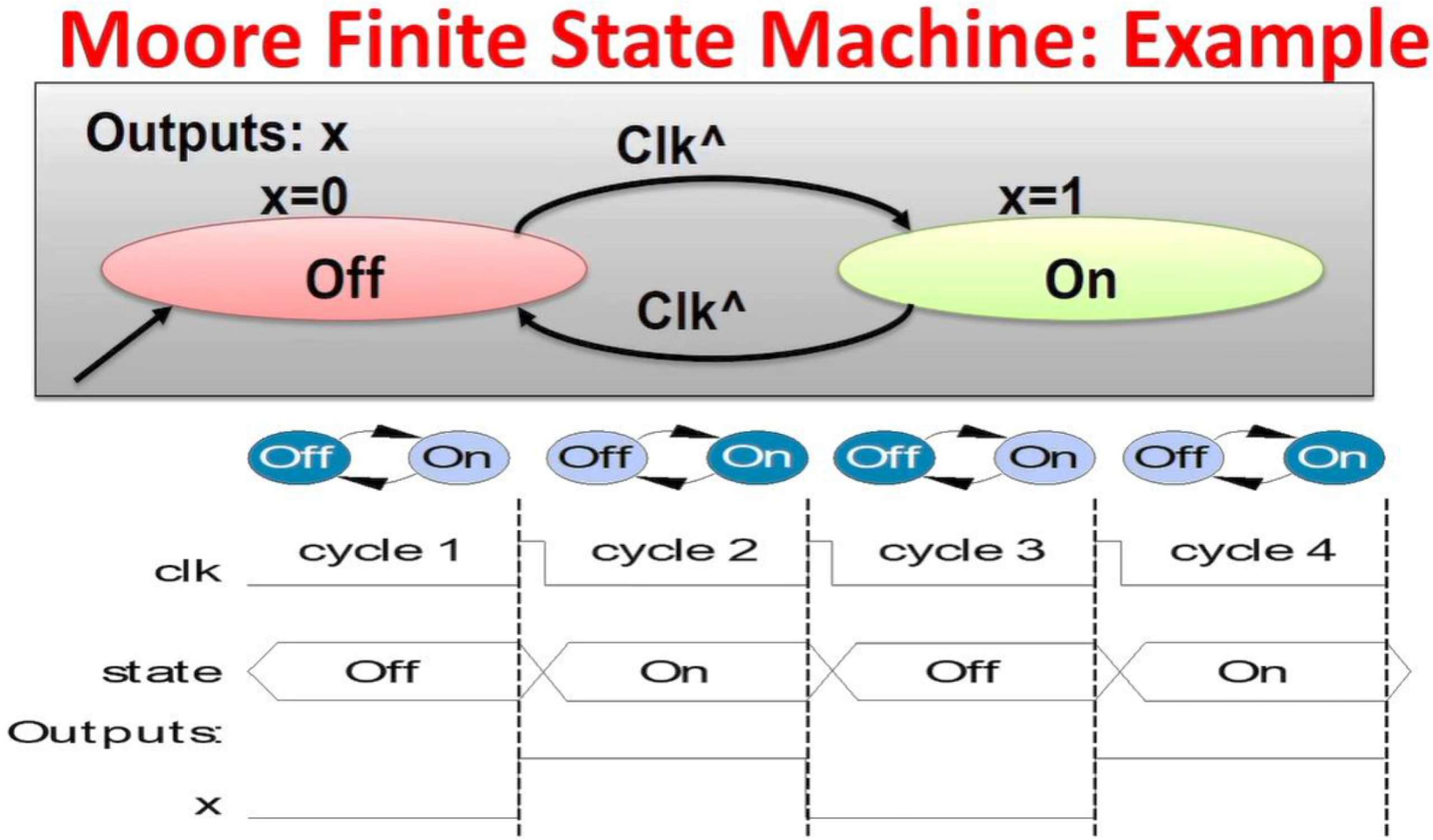
Design a ON/OFF circuits

Design a Circuit that changes its output at every clock plus.

- Step1: English like statement
  - Make x change toggle (0 to 1, or 1 to 0) every clock cycle
- Step2: Model the problem using FSM
  - Two states: “Off” (x=0), and “On” (x=1)
    - Transition on rising clock edge
  - from Off to On, or On to Off
  - Arrow with no starting state points to
    - initial state (when circuit first starts)

Finite State Machine - Example

Design a ON/OFF circuits

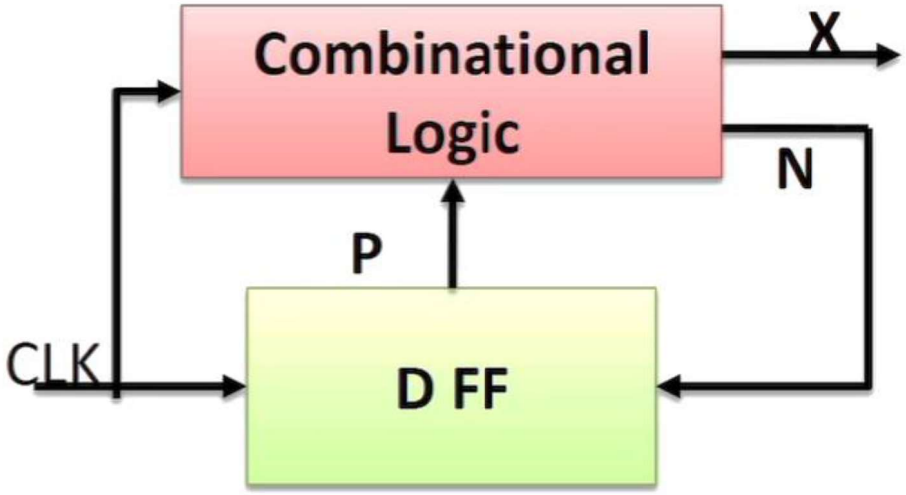


Finite State Machine - Example

Design a ON/OFF circuits

Step 3: Represent the FSM in Tabular Form

Input		Output	
CLK	PS	X	NS
RE 1	0	1	1
RE 1	1	0	0

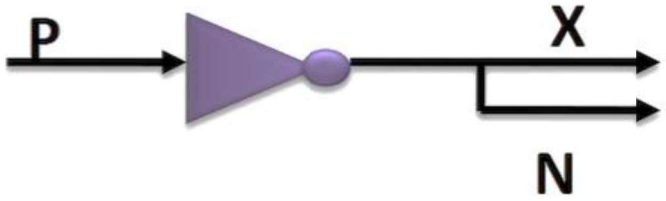


Step 4: Derive number of FF require

Two States: 1 FF

Step 5: State Encoding

State 0: 0 State 1: 1



Rising Edge: Clock implicit

Finite State Machine - Example

Design a ON/OFF circuits

Input		Output	
CLK	P	X	N
RE 1	0	1	1
RE 1	1	0	0

Step 6: Derivation of

Next State : from Present State and Input

Output : from Present State

Step 7: Derive combination function for NS logic, OP logic

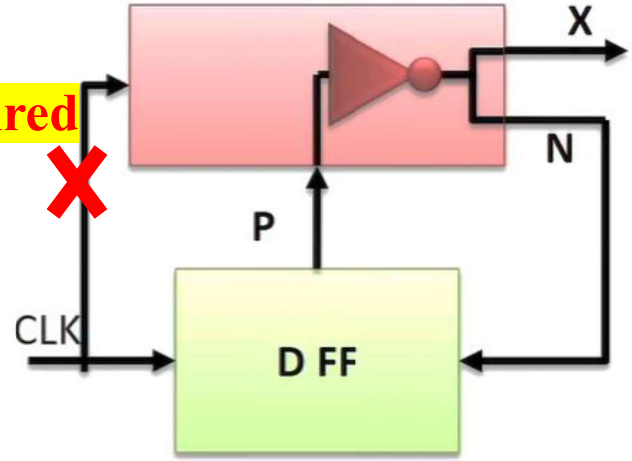
NS = not P; X=not P;

Rising Edge: Clock implicit

Step 8: Realize FSM

by Connecting the NS, OP logic, State Reg and Clock

Not Required



Finite State Machine - Example

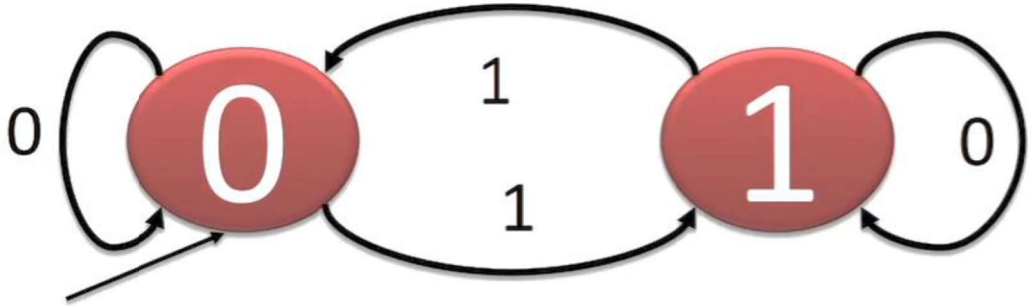
Design a T-FF

- Step1: English like statement
  - When NS toggle every clock when input T is one, else stay with same present state
- Step2: Model the problem using FSM
  - Two states: “Off” (S=0), and “On” (S=1)
    - Transition on rising clock edge
  - Arrow with no starting state points to
    - initial state (when circuit first starts)

Finite State Machine - Example

Design a T-FF

Step2: Model the problem using FSM



State: 0, 1
Input : 0, 1
Output: 0,1

Step 3: Represent the FSM in Tabular Form

PS= Q(t)	Input	NS =Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

Step 4: Derive number of FF require
Two States: 1 FF : Obvious

Step 5: State Encoding
State 0: 0 State 1: 1

Finite State Machine - Example

Design a T-FF

Step 6: Derivation of Next State

PS= Q(t)	Input	NS =Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

$Q^{+}=QT'+Q'T$

Step 7: Derive for NS logic, OP logic

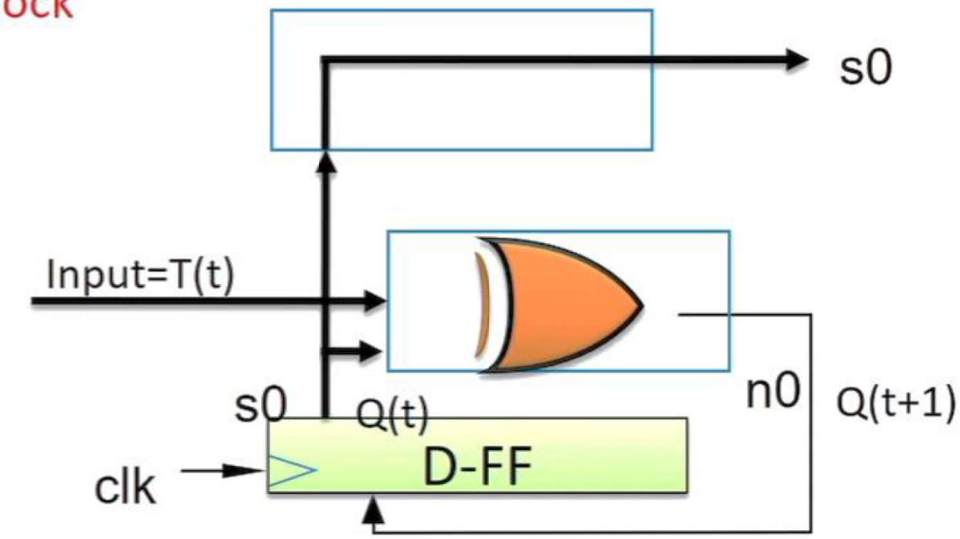
NS = = Q(t)'T(t)+Q(t)T'(t)

=Q(t) xor T(t)

X=Q;

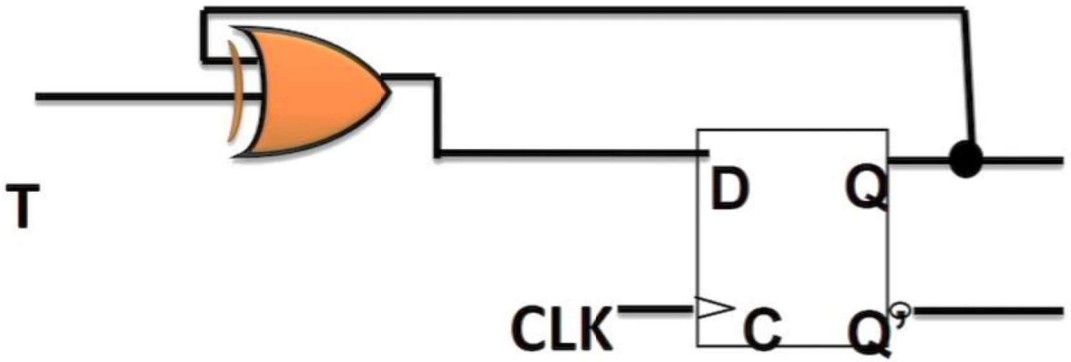
Step 8: Realize FSM

by Connecting the NS, OP logic, State Reg and Clock



Finite State Machine - Example

Design a T-FF from D-FF



T	D	TQ'+T'Q
0	Q	0Q'+1Q
0	Q'	1Q'+0Q

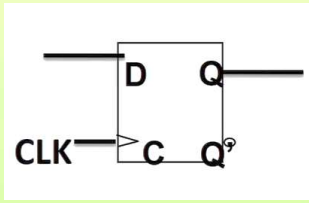
$$D = T\,Q' + T'\,Q$$

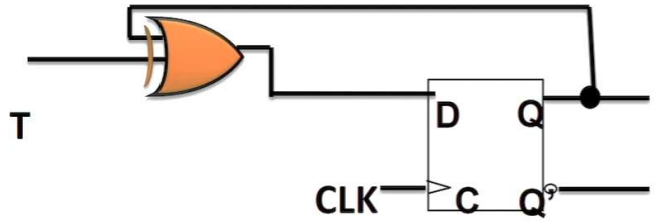
Finite State Machine - Example

Verilog Implementation

Design a T-FF from D-FF

```
//Code for Data Flipflop; will be used in FSM
implementation
module DFF( input wire D, input wire clk, input rst,
                        output reg Q)
   always @ (posedge rst) begin
      if (rst) Q=0;
   end
    always @(posedge clk) begin
      if (!rst) Q <= D;
   end
endmodule
```

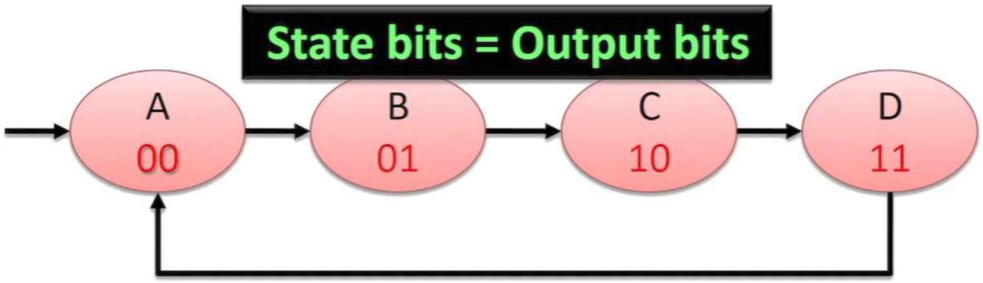




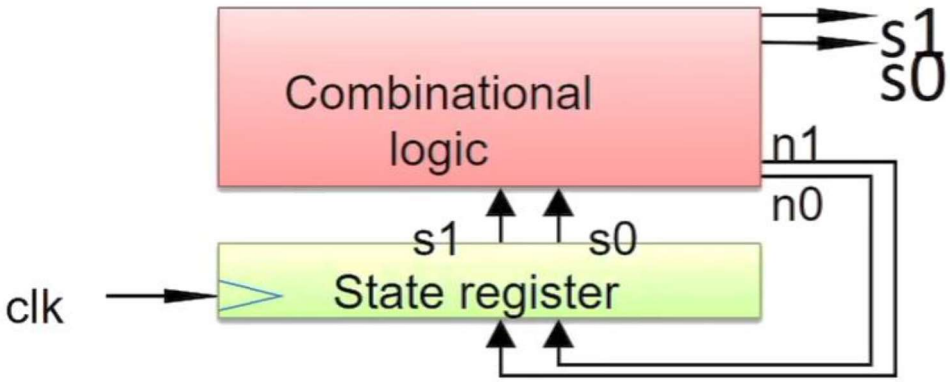

```
module tff_fsm (input clk, input t, input rst, output q);
        wire NS, S;

        DFF D0(NS, clk, rst, S);
        assign NS = t ^ S; //NS logic
        assign q = S;  //output logic

endmodule
```

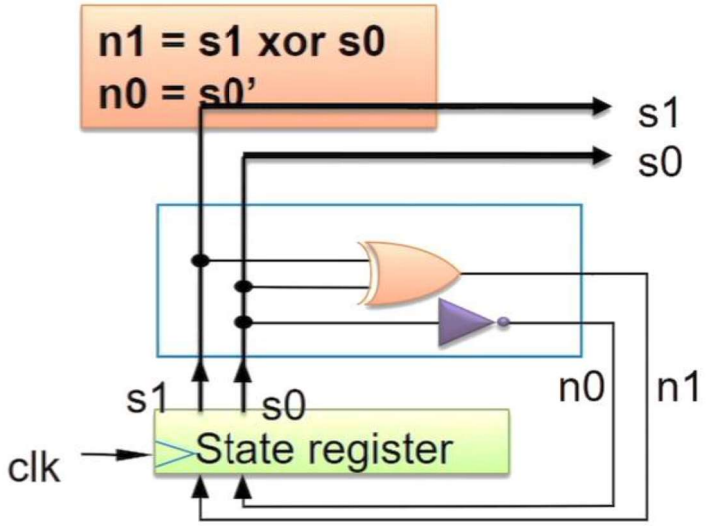
Finite State Machine - Example

Design a 2-bit counter



	I/P		O/P	
	s1	s0	n0	n1
A	0	0	0	1
B	0	1	1	0
C	1	0	1	1
B	1	1	0	0





Finite State Machine – State Encoding

- Common FSM encoding options:
  - One-hot code
  - Binary code
  - Gray code
  - Random code

State	Binary	Gray	One hot
A	00	00	1000
B	01	01	0100
C	10	11	0010
D	11	10	0001

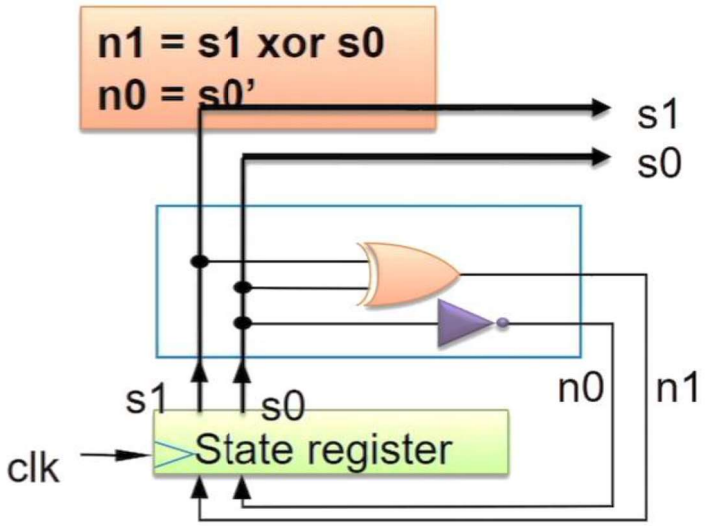
One-hot encoding is usually used in FPGA-based design, because there are a lot of registers in theses devices.

Finite State Machine - Example

Verilog Implementation

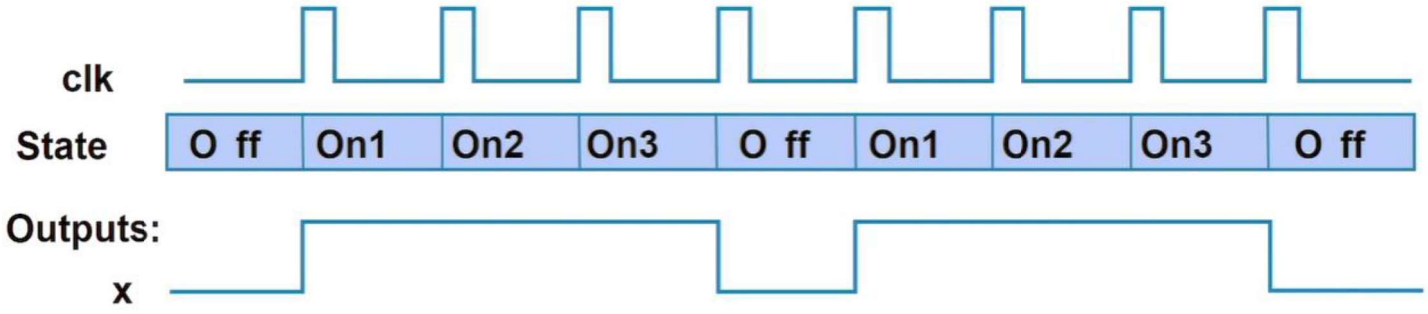
Design a 2-bit counter

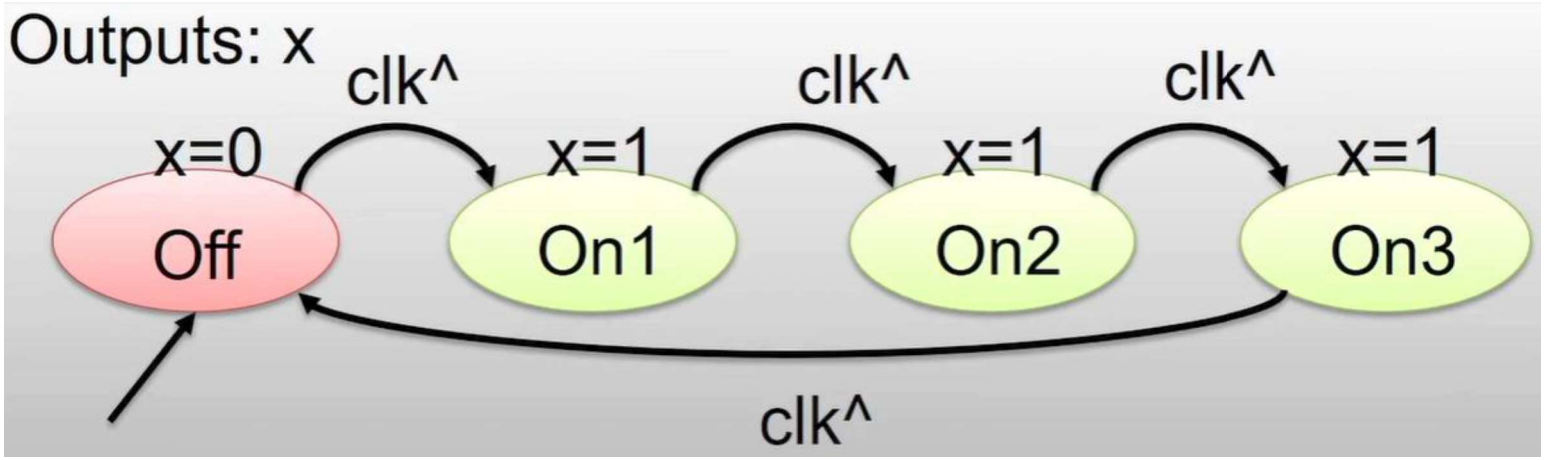
```
module  ctr2bit_fsm (input clk, input rst, output q1, output q0);
        wire NS0, NS1, NS2, S0, S1, S2;

        DFF D0(NS0, clk, rst, S0); //Instantiation
        DFF D1(NS1, clk, rst, S1); //Instantiation

        assign NS1 = S1 ^ S0;  //NS logic
        assign NS0 = ~S0;      //NS logic
        assign q1 = S1;        //output logic
        assign q0 = S0;        //output logic
endmodule
```



Finite State Machine - Example

Design Sequence Generator 0 1 1 1 0 1 1 1 …



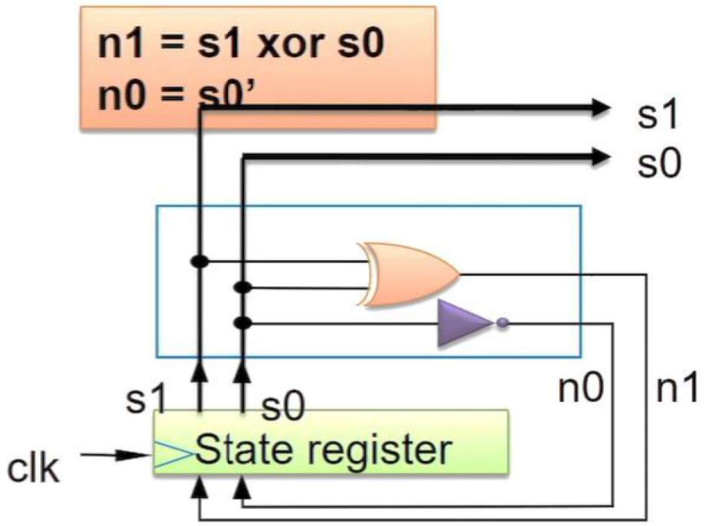


Finite State Machine - Example

Verilog Implementation

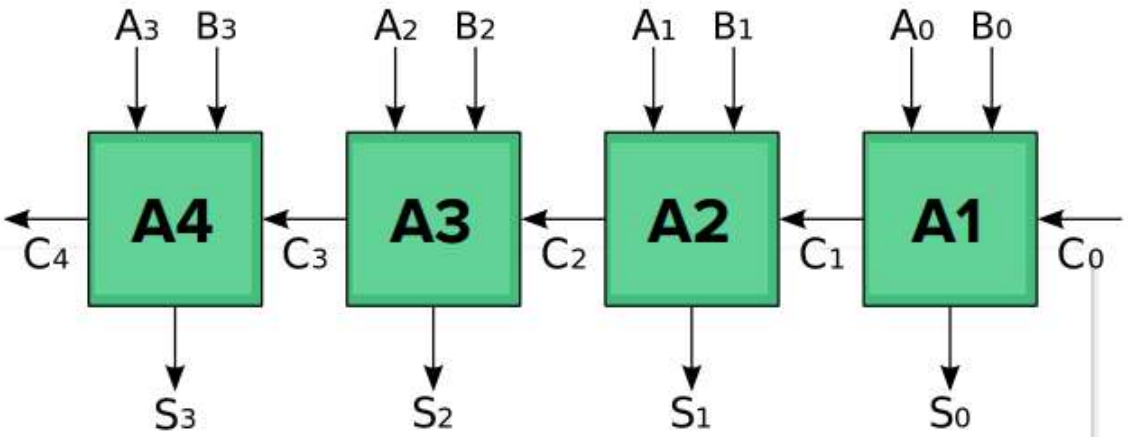
Design a 2-bit counter

```
module  ctr2bit_fsm (input clk, input rst, output q1, output q0);
        wire NS0, NS1, NS2, S0, S1, S2;

        DFF D0(NS0, clk, rst, S0); //Instantiation
        DFF D1(NS1, clk, rst, S1); //Instantiation

        assign NS1 = S1 ^ S0;  //NS logic
        assign NS0 = ~S0;      //NS logic
        assign q1 = S1;        //output logic
        assign q0 = S0;        //output logic
endmodule
```

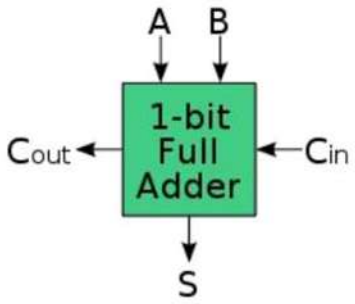


Adder

Implement a 4-Bit Ripple Carry Adder

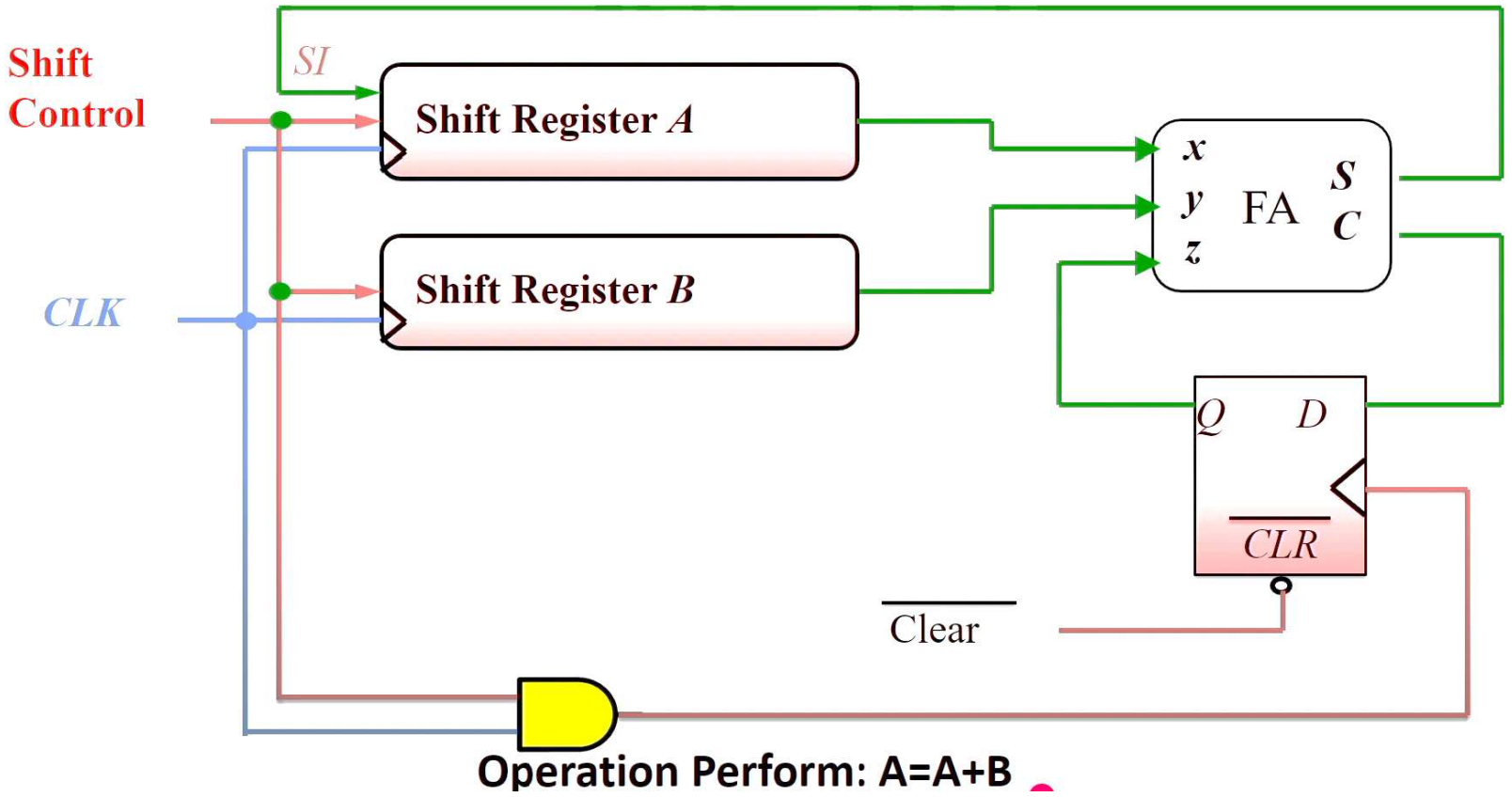


A	B	C	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



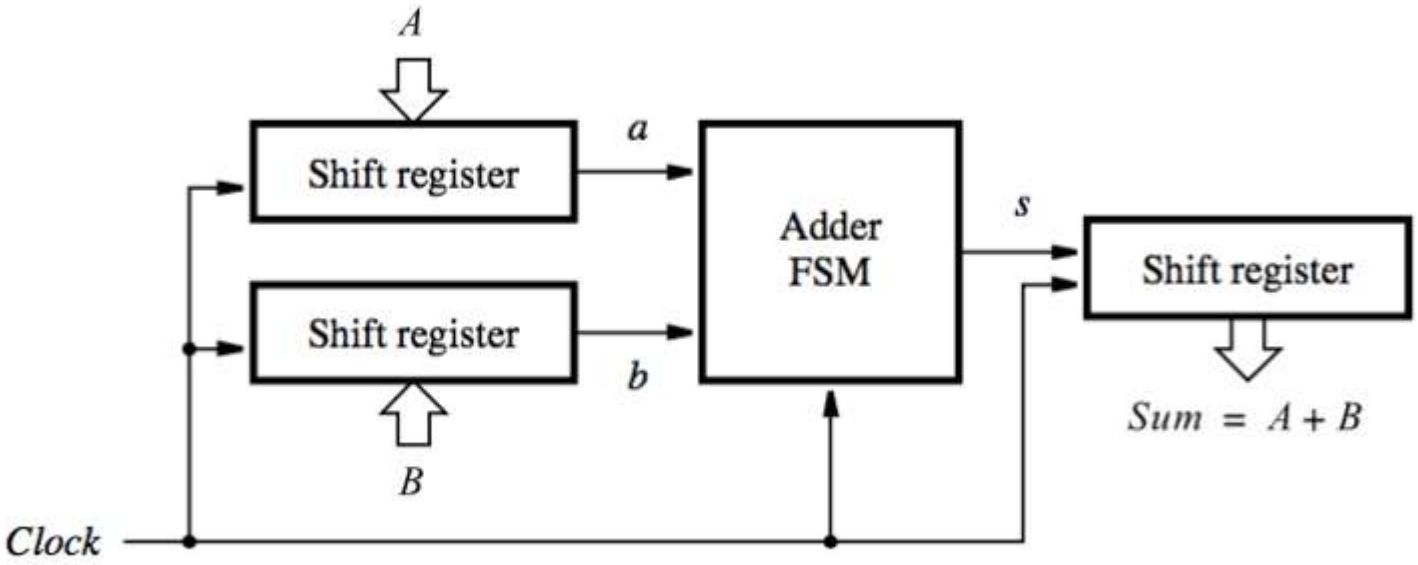

```
sum = (A^B)^C
Cout = AB+BC+AC
```

Sequential Circuit Design



Serial Adder

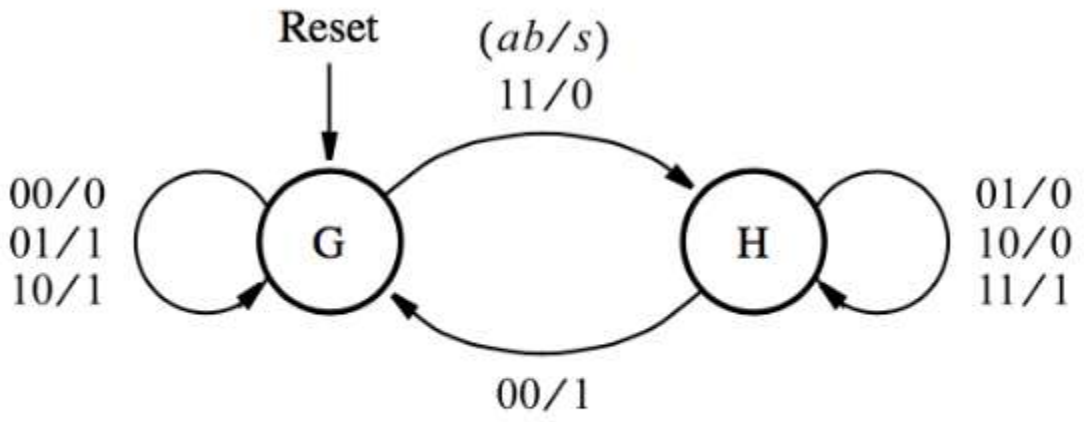
Serial Adder



Let **G** and **H** denote the states where the carry-in-values are **0** and **1**. Output value ***s*** depends on both the state and the present value of inputs ***a*** and ***b***.

Melay Serial Adder

State Diagram



G: carry-in = 0
H: carry-in = 1

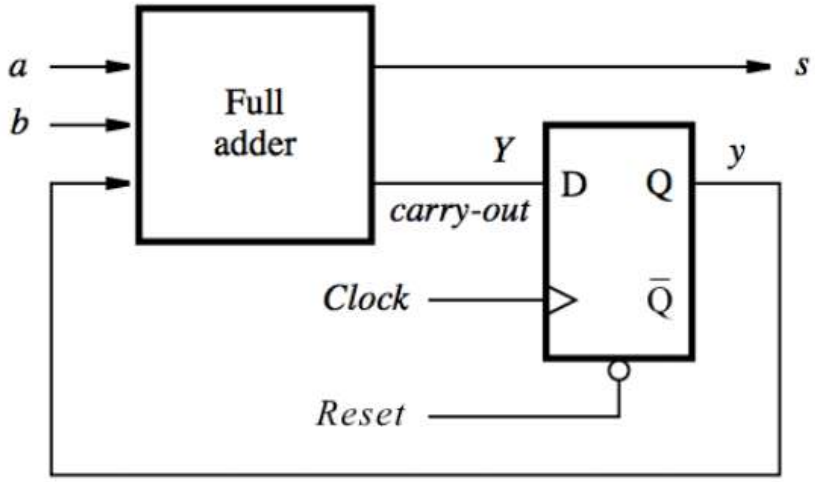
State Table

Present state	Next state				Output *s*			
	ab = 00	01	10	11	00	01	10	11
G	G	G	G	H	0	1	1	0
H	G	H	H	H	1	0	0	1

Melay Serial Adder

State Assignment

Present state	Next state				Output			
	$ab = 00$	01	10	11	00	01	10	11
y	Y				s			
0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	0	0	1

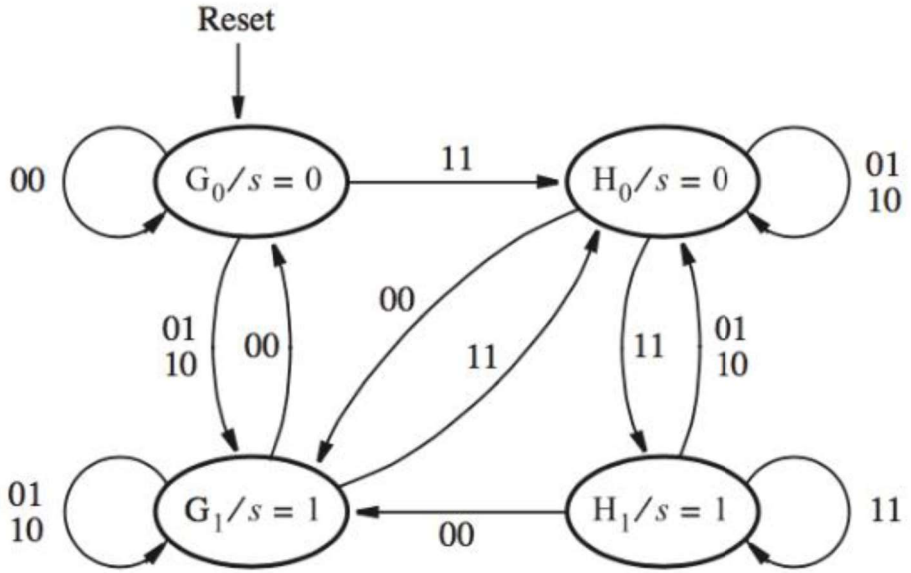


$$Y = ab + ay + by$$

$$s = a \oplus b \oplus y$$

Moore Serial Adder

State Diagram



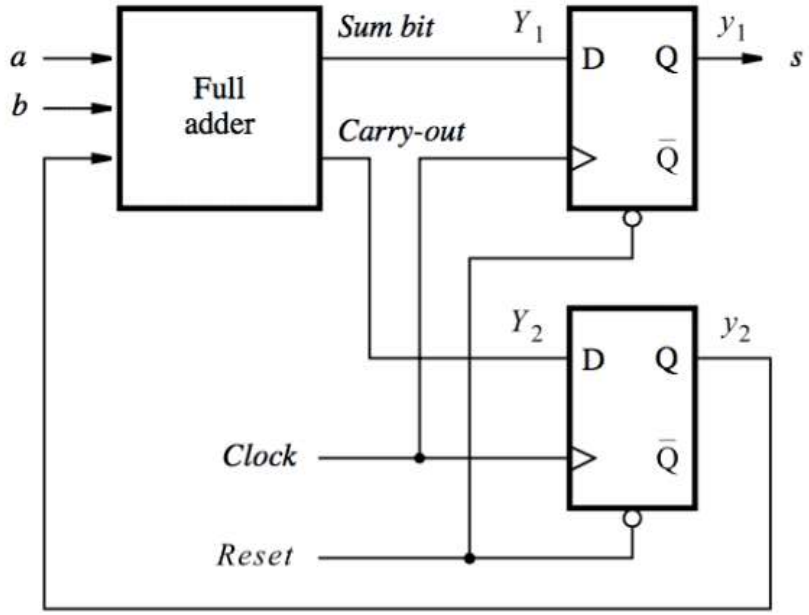
State Table

Present state	Next state				Output
	$ab = 00$	01	10	11	s
G_0	G_0	G_1	G_1	H_0	0
G_1	G_0	G_1	G_1	H_0	1
H_0	G_1	H_0	H_0	H_1	0
H_1	G_1	H_0	H_0	H_1	1

Moore Serial Adder

State Assignment

Present state y_2y_1	Next state Y_2Y_1				Output s
	$ab = 00$	01	10	11	
0 0	0 0	0 1	0 1	1 0	0
0 1	0 0	0 1	0 1	1 0	1
1 0	0 1	1 0	1 0	1 1	0
1 1	0 1	1 0	1 0	1 1	1



$$Y_1 = a \oplus b \oplus y_2$$

$$Y_2 = ab + by_2 + by_2$$

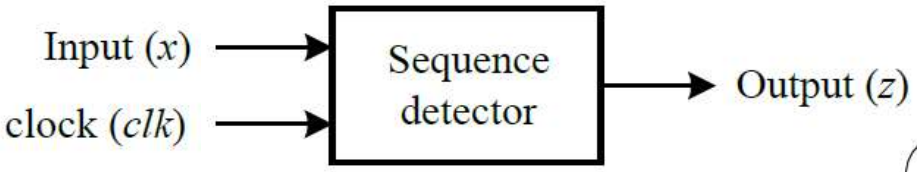
$$s = y_1$$

Finite State Machine - Example

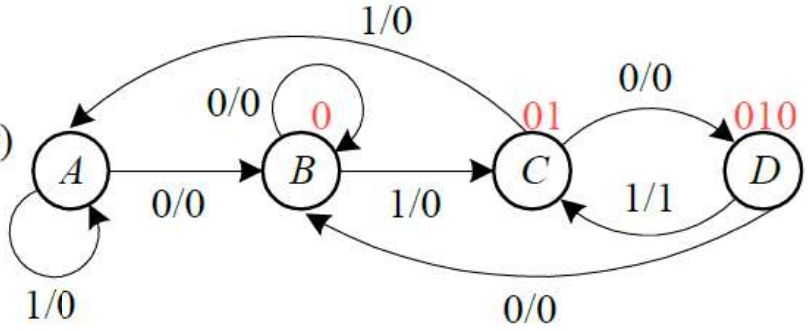
Verilog Implementation

An FSM Example ---A 0101 Sequence Detector

- Design a finite-state machine to detect the pattern 0101 in the input sequence x. Assume that overlapping pattern is allowed.

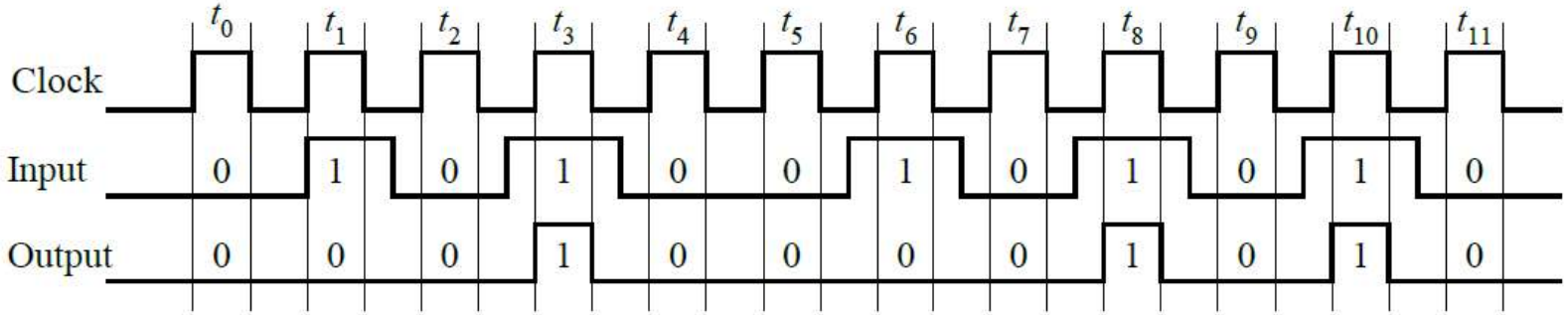


Block diagram



State diagram

Timing chart



Finite State Machine - Example

0101 Sequence

Mealy machine example ---Using only state register and one always block.

```
module sequence_detector_mealy (clk, reset_n, x, z);
input  x, clk, reset_n;
output wire z;
reg  [1:0] state;
parameter A = 2'b00, B = 2'b01, C = 2'b10, D = 2'b11;
// the body of sequence detector --- Using only one always block.
always @(posedge clk or negedge reset_n) begin
   if (!reset_n) state <= A; else state <= fsm_next_state(state, x); end
function [1:0] fsm_next_state (input [1:0] present_state, input x);
reg [1:0] next_state;
begin case (present_state)
        A: if (x) next_state = A;  else next_state = B;
        B: if (x) next_state = C;  else next_state = B;
        C: if (x) next_state = A;  else next_state = D;
        D: if (x) next_state = C;  else next_state = B;
   endcase
   fsm_next_state = next_state;
end endfunction
assign z = (x == 1 && state == D);
endmodule
```

Finite State Machine - Example

Verilog Implementation

```
// Mealy machine example --- Using only one state register and two always blocks.
module sequence_detector_mealy (clk, reset_n, x, z);
input  x, clk, reset_n;
output reg z;
// Local declaration
reg  [1:0] state;
parameter A = 2'b00, B = 2'b01, C = 2'b10, D = 2'b11;
// the body of sequence detector --- Using two always blocks.
always @(posedge clk or negedge reset_n) begin
  if (!reset_n) state <= A; else state <= fsm_next_state(state, x); end
function [1:0] fsm_next_state (input [1:0] present_state, input x);
  …. Same as that of style 1a
end endfunction
// evaluate output function z
always @(state or x) case (state)
       A: if (x) z = 1'b0; else z = 1'b0;  B: if (x) z = 1'b0; else z = 1'b0;
       C: if (x) z = 1'b0; else z = 1'b0;  D: if (x) z = 1'b1; else z = 1'b0;
   endcase
endmodule
```

Finite State Machine - Example

Verilog Implementation

```
// Mealy machine example --- using two state registers and three always blocks.
module sequence_detector_mealy (clk, reset_n, x, z);
input  x, clk, reset_n;
output z;
// local declaration
reg [1:0] present_state, next_state; // present state and next state
reg    z;
parameter A = 2'b00, B = 2'b01, C = 2'b10, D = 2'b11;
// initialize to state A and update state register
always @(posedge clk or negedge reset_n)
    if    (!reset_n) present_state <= A;
    else  present_state <= next_state;          // update present state
```

```
// determine next state
always @(present_state or x)
   case (present_state)
        A: if (x) next_state = A; else next_state = B;
        B: if (x) next_state = C; else next_state = B;
        C: if (x) next_state = A; else next_state = D;
        D: if (x) next_state = C; else next_state = B;
   endcase
```

```
// evaluate output function z
always @(present_state or x)
   case (present_state)
      A: if (x) z = 1'b0; else z = 1'b0;
      B: if (x) z = 1'b0; else z = 1'b0;
      C: if (x) z = 1'b0; else z = 1'b0;
      D: if (x) z = 1'b1; else z = 1'b0;
   endcase
endmodule
```