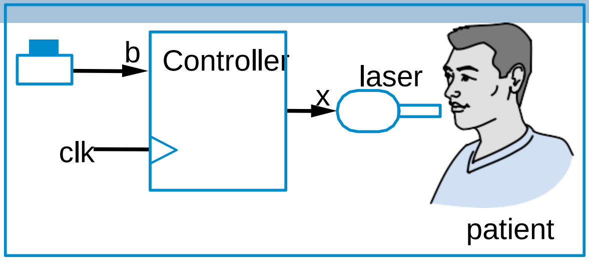
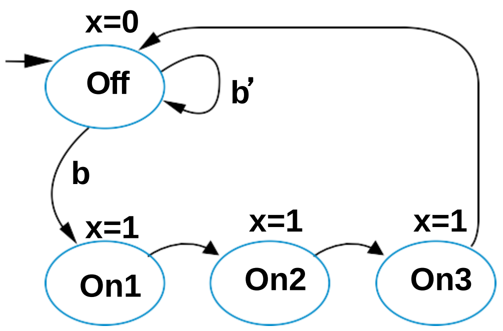
4 – FSMs and Verilog Review

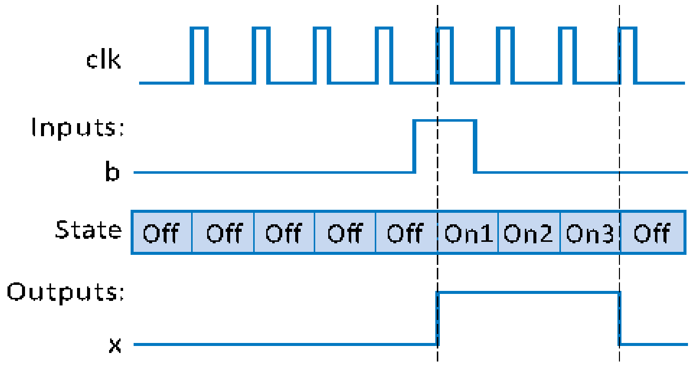
ECE 474A/574A
COMPUTER-AIDED LOGIC DESIGN

FSM Example: Three-Cycles High Laser Timer

- State Diagram or Finite-State Machine (FSM)
  - A way to describe desired behavior of sequential circuit
  - List states, and transitions among states
- Laser Timer
  - When button pressed (b=1), turn laser on (x=1) for 3 clock cycles
- Four states
  - Off state
    - Keep laser turned off
    - While b=0 (b’), we are in a wait state
    - When b=1 and rising clock edge (b • clk^), transition to On1 state
  - On1 state
    - Turns laser on (x=1)
    - On next rising clock edge (clk^) transition to On2 state
  - On2/On3 state
    - Also turns laser on (x=1)
    - Transitions on next rising clock edge

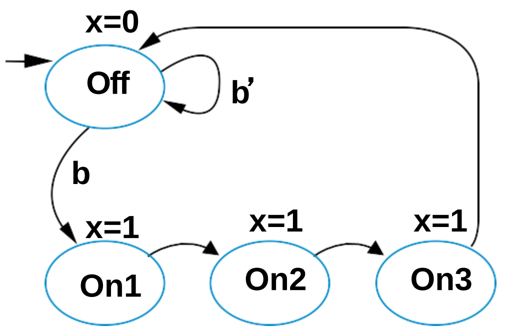




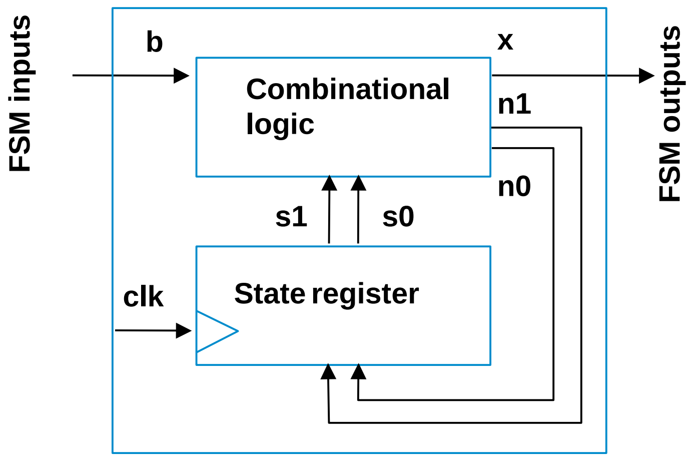


Graphical and Textual Sequential Circuit Descriptions

- FSM
  - Graphical representation
  - Formal method to describe sequential circuits
- State Table
  - Textual Representation



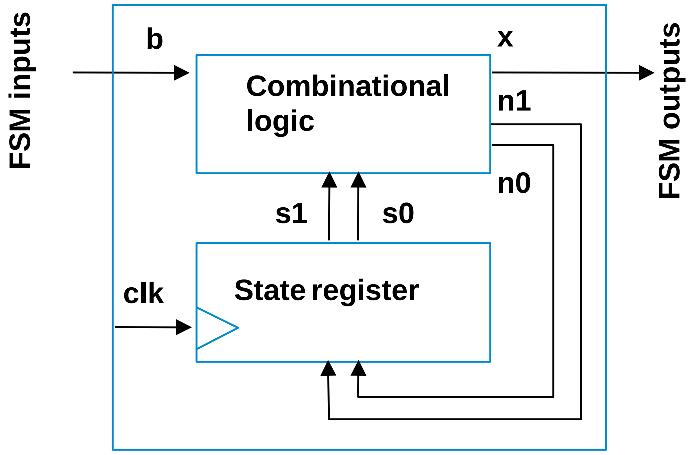
- How do we implement a sequential circuit?
  - Standard Controller architecture
    - Need to store state
      - State register (encoded state)
    - Need to determine next state
      - Current state and external input to combinational logic
    - Need to determine output
      - Current state input to combinational logic



Controller architecture for laser timer example

State Table Example: Laser Timer (cont’)

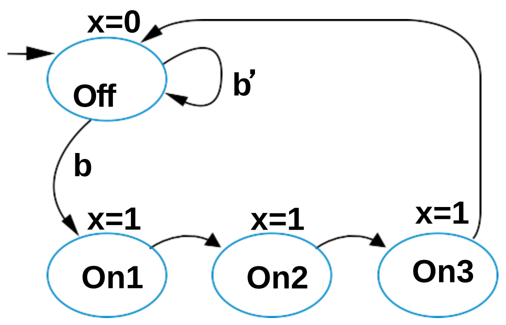
- State Table
  - Inputs
    - Current state (encoded) - Two bits s1 and s0 encode the current state
    - FSM Input - Input b indicates button press
  - Outputs
    - Next state (encoded) - Two bits n1 and n0 encode the next state
    - FSM Output - Output x controls when the laser is on/off



Inputs			Outputs		
s1	s0	b	n1	n0	x

State Table Example: Laser Timer

- State Table
  - Next state
    - Based on current state and FSM input what is the next state?
  - FSM Output
    - Output depends on current state only (Moore FSM)
    - For each state we are currently in, what is the output?



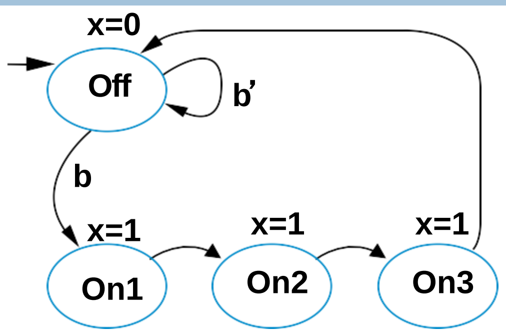
Inputs			Outputs		
s1	s0	b	n1	n0	x

(Condensed) Controller Design Process

	Step	Description
Step 1:	Capture the FSM	Create an FSM (state diagram) that describes the desired behavior of the circuit
Step 2:	Create the architecture	Create the standard architecture by using a state register of appropriate width, and combinational logic with inputs being the state register bits and the FSM inputs, and outputs being the next state bits and the FSM outputs
Step 3:	Encode the states	Assign a unique binary number to each state. Each binary number representing a state is know as an encoding. Any encoding will do as long as they are unique.
Step 4:	Create the state table	Create a truth table for the combinational logic such that the logic will generate the correct FSM output and next state signals. Ordering the inputs with state bits first make the truth table describe the state behavior, giving us a state table.
Step 5:	Implement the combinational logic	Implement the combinational logic using any method.

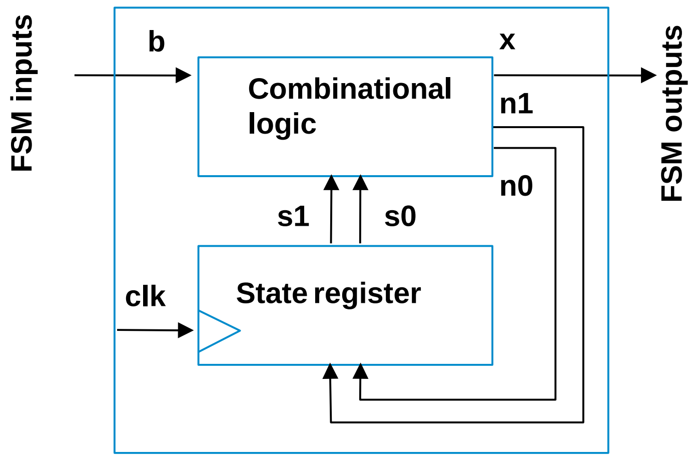
Controller Design: Laser Timer

- Example: Laser Timer



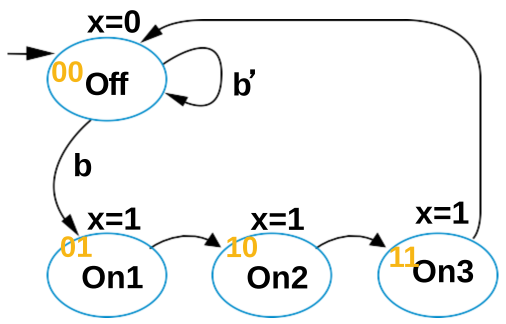
- Step 1: Capture the FSM
  - Already done

- Step 2: Create architecture
  - Customize generic controller architecture to our system
    - State Register
      - 2-bit state register (for 4 states)
      - s1, s0 – current state bits
      - n1, n0 – next state bits
    - FSM Input
      - Button signal b
    - FSM Output
      - Laser control x



Controller Design: Laser Timer

- Step 3: Encode the states
  - Any encoding with each state unique will work

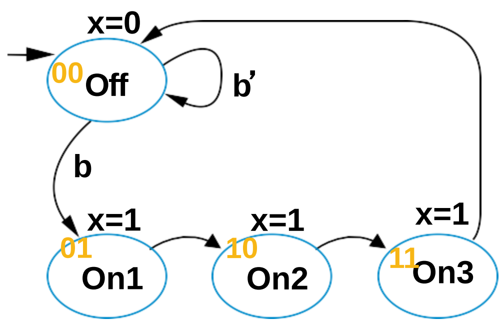


- Step 4: Create state table
  - Done this already

	Inputs			Outputs		
	s1	s0	b	n1	n0	x
Off	0	0	0	0	0	0
	0	0	1	0	1	0
On1	0	1	0	1	0	1
	0	1	1	1	0	1
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	0	0	1
	1	1	1	0	0	1

Controller Design: Laser Timer

- Step 5: Implement the combinational logic



n1 = s1' s0b' + s1' s0b + s1s0' b' + s1s0' b
n1 = s1' s0 + s1s0'

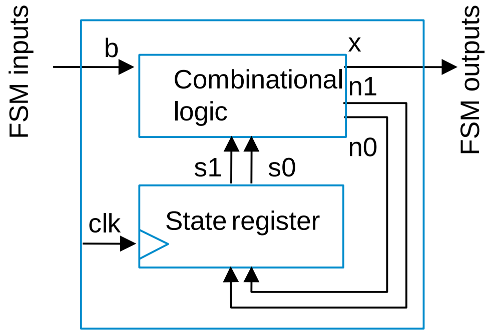
n0 = s1' s0' b + s1s0' b' + s1s0' b
n0 = s1' s0' b + s1s0'

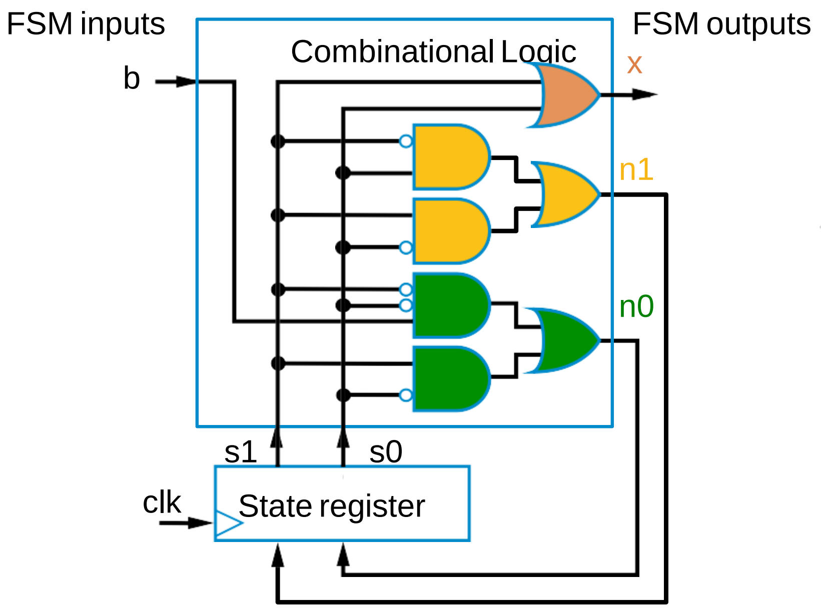
x = s1' s0' b + s1s0' b' + s1s0' b
x = s1 + s0

	Inputs			Outputs		
	s1	s0	b	n1	n0	x
Off	0	0	0	0	0	0
	0	0	1	0	1	0
On1	0	1	0	1	0	1
	0	1	1	1	0	1
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	0	0	1
	1	1	1	0	0	1

Controller Design: Laser Timer

- Step 5: Implement combinational logic (cont)





x = s1 + s0

n1 = s1' s0 + s1s0'

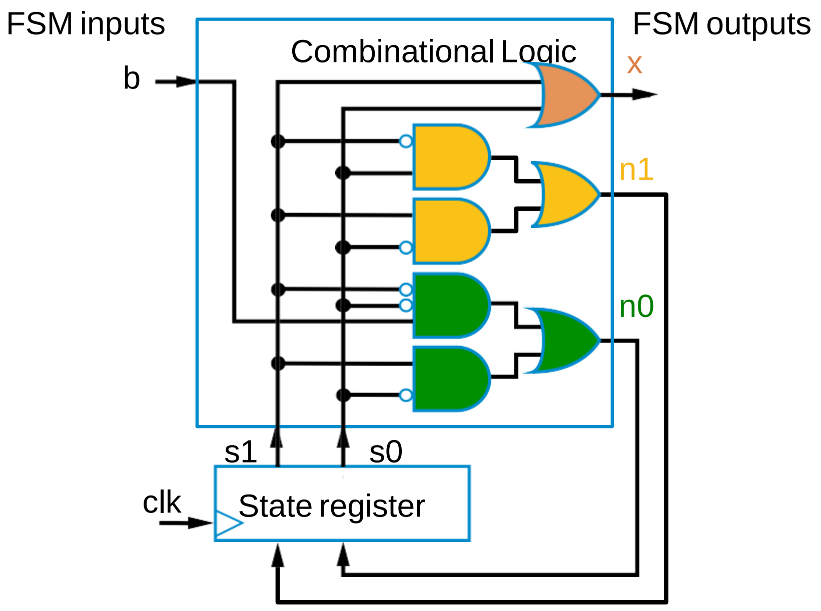
n0 = s1' s0' b + s1s0'

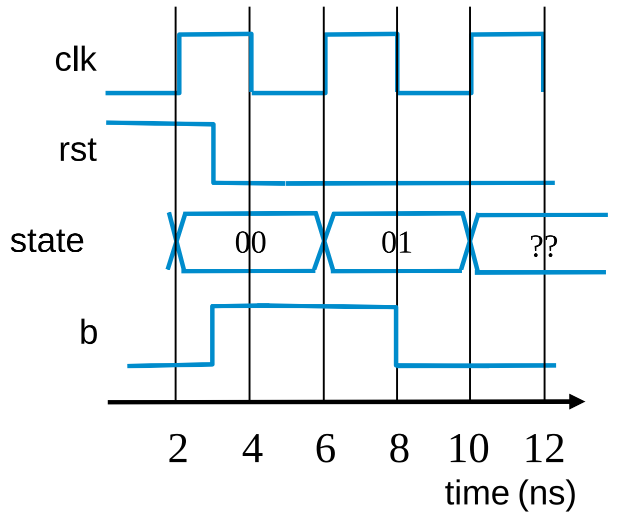
Exercise

Using the Controller Design Process, design a circuit that outputs Y = 1 when it detects 3 or more 1’s in an input bit string.

Critical Path

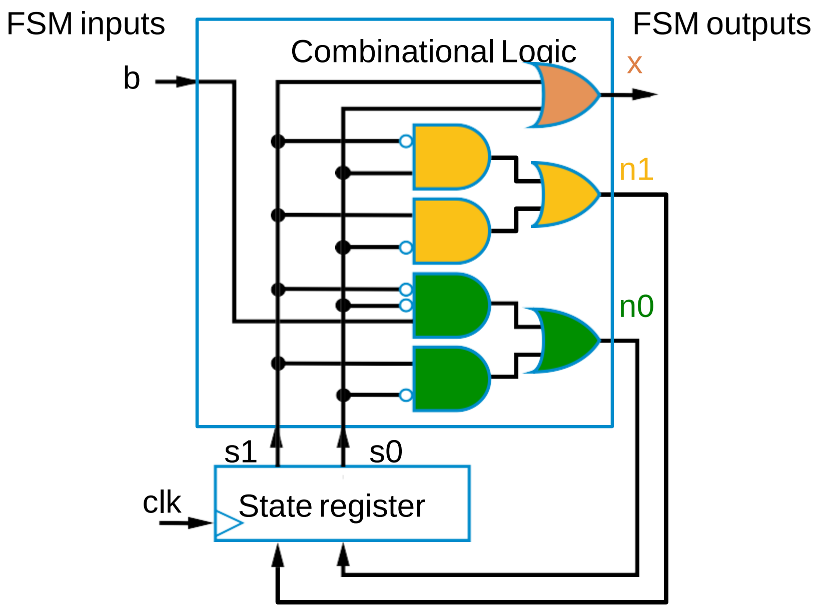
1. What should the next state (n1n0) be at the indicated time?
2. Assume each gate has a 2 ns delay, inverters have a 1 ns delay, a register setup time of 0.5 ns, and a clock period of 4 ns. What will be stored in the state register at the indicated time?





Critical Path

- Assume each gate has a 2 ns delay, inverters have a 1 ns delay, a register setup time of 0.5 ns, and a clock period of 4 ns. What's the minimum required clock period?

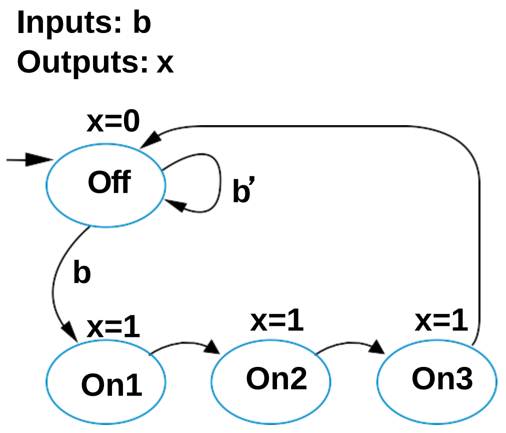


Sequential Logic Design Process

- Create an FSM for a timing verification component that has three inputs CStart, CEnd, and ErrorRst, and one output Error. In a correctly functioning system, whenever the CStart input becomes 1, two cycles later the CEnd signal should become 1 for exactly 1 cycle. The timing verification controller should assert the output Error to 1 whenever this behavior is NOT observed. The controller will continue to assert the output Error to 1 until the ErrorRst input becomes 1, after which the controller should again monitor the CStart and CEnd inputs.

FSM Formal Definition

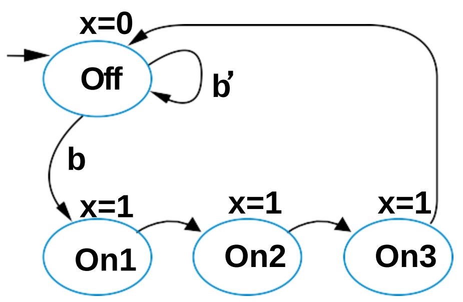
- FSM defined as follows:
  - M = (Σ, Γ, S, δ, λ, s_o)
  - Σ is the input alphabet
  - Γ is the output alphabet
  - S is a finite set of states
  - δ is the transition function, δ: X x S→S
    - Given an input and state, what is the next state
  - λ is the output function, λ: S → Y
    - Mealy FSM, λ: X x S → Y
  - s_o is the initial state



FSM Formal Definition

- Formally specify the Laser Timer FSM
  - M = (Σ, Γ, S, δ, λ, s_o)

Inputs: b
Outputs: x



LaserTimer = (Σ, Γ, S, δ, λ, qo), where

Σ = {0, 1}

Γ = {0, 1}

S = {Off, On1, On2, On3}

δ(Off, 0) = Off,	δ(Off, 1) = On1
δ(On1, 0) = On2,	δ(On1, 1) = On2
δ(On2, 0) = On3,	δ(On2, 1) =On3
δ(On3, 0) = Off,	δ(On3, 1) = Off

λ(Off) = 0, λ(On1) = 1, λ(On2) = 1, λ(On3) = 1

s_o = Off

Σ is the input alphabet

Y is the output alphabet

S is a finite set of states

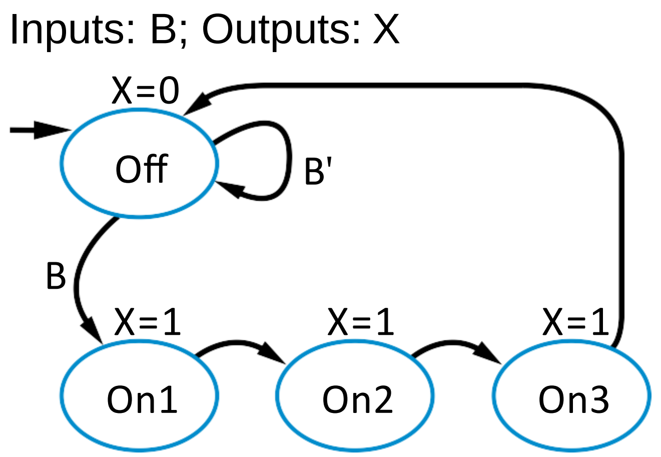
δ is the transition function, δ: X x S→S
Given an input and state, what is the next state

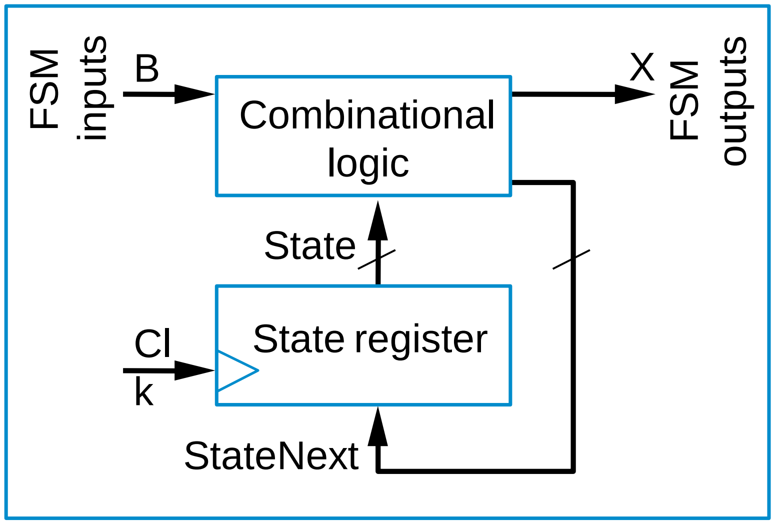
λ is the output funciton, λ: S → Y

s_o is the initial state

Finite-State Machines (FSMs)—Sequential Behavior

- Finite-state machine (FSM) is a common model of sequential behavior
  - Example: If B=1, hold X=1 for 3 clock cycles
    - Note: Transitions implicitly ANDed with rising clock edge
  - Implementation model has two parts:
    - State register
    - Combinational logic
  - HDL model will reflect those two parts

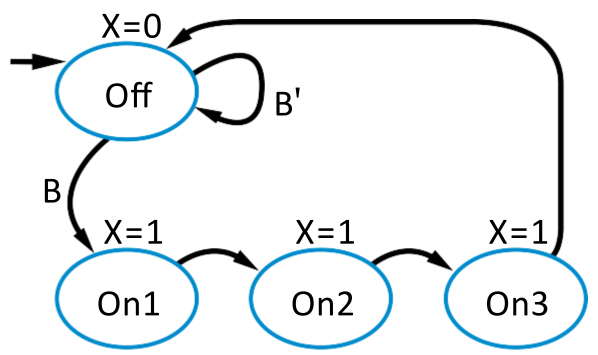


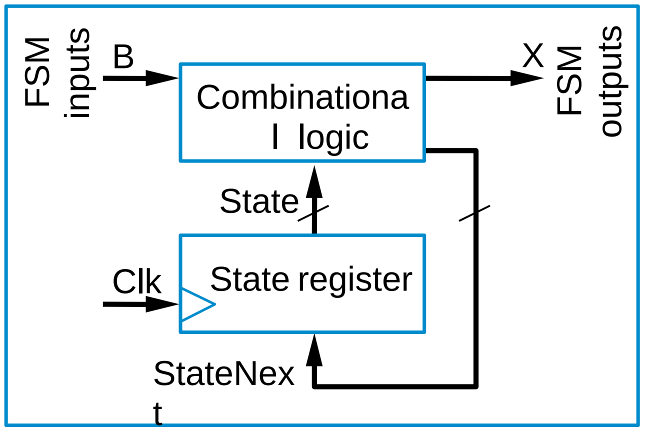


Finite-State Machines (FSMs)—Sequential Behavior

Modules with Multiple Procedures and Shared Variables

Inputs: B; Outputs: X



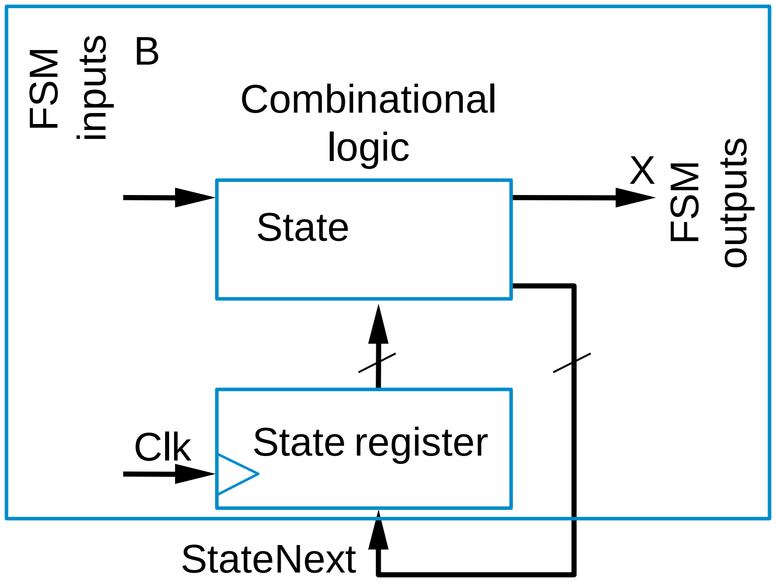


```
`timescale 1 ns/1 ns

module LaserTimer(B, X, Clk, Rst);

   input B;
   output reg X;
   input Clk, Rst;

   parameter S_Off = 0, S_On1 = 1,
             S_On2 = 2, S_On3 = 3;

   reg [1:0] State, StateNext;

   // CombLogic
   always @(State, B) begin
      case (State)
         S_Off: begin
            X <= 0;
            if (B == 0)
               StateNext <= S_Off;
            else
               StateNext <= S_On1;
         end
         ...
```

```
         ...
         S_On1: begin
            X <= 1;
            StateNext <= S_On2;
         end
         S_On2: begin
            X <= 1;
            StateNext <= S_On3;
         end
         S_On3: begin
            X <= 1;
            StateNext <= S_Off;
         end
      endcase
   end

   // StateReg
   always @(posedge Clk) begin
      if (Rst == 1 )
         State <= S_Off;
      else
         State <= StateNext;
   end
endmodule
```

Finite-State Machines (FSMs)—Sequential Behavior

- Module has two procedures
  - One procedure for combinational logic
  - One procedure for state register
  - But it's still a behavioral description



```
`timescale 1 ns/1 ns

module LaserTimer(B, X, Clk, Rst);

   input B;
   output reg X;
   input Clk, Rst;

   parameter S_Off = 0, S_On1 = 1,
             S_On2 = 2, S_On3 = 3;

   reg [1:0] State, StateNext;

   // CombLogic
   always @(State, B) begin
      ...
   end

   // StateReg
   always @(posedge Clk) begin
      ...
   end
endmodule
```

Finite-State Machines (FSMs)—Sequential Behavior

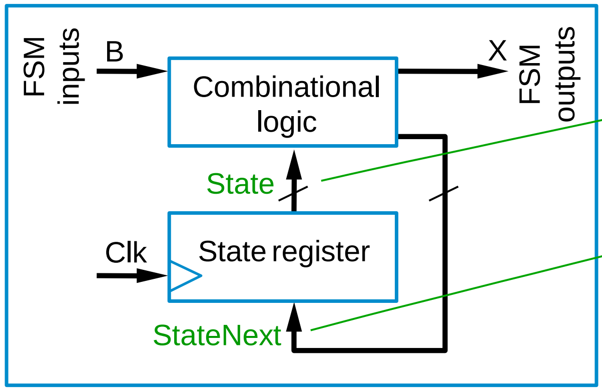
Parameters

- parameter declaration
  - Not a variable or net, but rather a *constant*
  - A constant is a value that must be initialized, and that cannot be changed within the module's definition
  - Four parameters defined
    - *S_Off, S_On1, S_On2, S_On3*
    - Correspond to FSM's states
      - Should be initialized to unique values

```
`timescale 1 ns/1 ns

module LaserTimer(B, X, Clk, Rst);

   input B;
   output reg X;
   input Clk, Rst;

   parameter S_Off = 0, S_On1 = 1,
             S_On2 = 2, S_On3 = 3;

   reg [1:0] State, StateNext;

   // CombLogic
   always @(State, B) begin
      ...
   end

   // StateReg
   always @(posedge Clk) begin
      ...
   end
endmodule
```

Finite-State Machines (FSMs)—Sequential Behavior

- Module declares two reg variables
  - *State, StateNext*
  - Each is 2-bit vector (need two bits to represent four unique state values 0 to 3)
  - Variables are shared between CombLogic and StateReg procedures
- CombLogic procedure
  - Event control sensitive to *State* and input *B*
  - Will output *StateNext* and *X*
- StateReg procedure
  - Sensitive to *Clk* input
  - Will output *State*, which it stores



```
`timescale 1 ns/1 ns

module LaserTimer(B, X, Clk, Rst);

   input B;
   output reg X;
   input Clk, Rst;

   parameter S_Off = 0, S_On1 = 1,
             S_On2 = 2, S_On3 = 3;

   reg [1:0] State, StateNext;

   // CombLogic
   always @(State, B) begin
      ...
   end

   // StateReg
   always @(posedge Clk) begin
      ...
   end
endmodule
```

Finite-State Machines (FSMs)—Sequential Behavior

Procedures with Case Statements

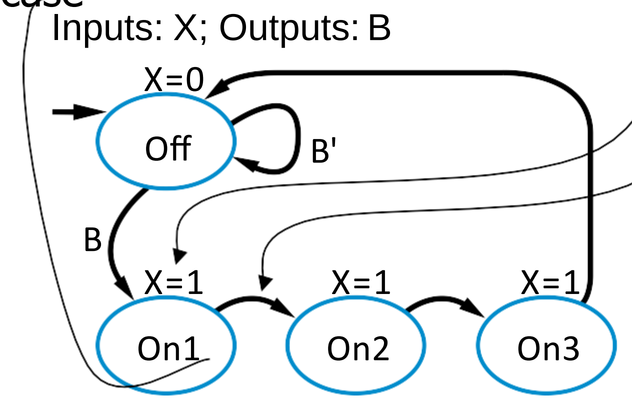
- Procedure may use case statement
  - Preferred over if-else-if when just one expression determines which statement to execute
  - case (expression*)*
    - Execute statement whose case item expression value matches case expression
      - *case item expression : statement*
      - *statement* is commonly a begin-end block, as in example
  - First case item expression that matches executes; remaining case items ignored
  - If no item matches, nothing executes
  - Last item may be "*default : statement*"
    - Statement executes if none of the previous items matched

```
// CombLogic
always @(State, B) begin
   case (State)
      S_Off: begin
         X <= 0;
         if (B == 0)
            StateNext <= S_Off;
         else
            StateNext <= S_On1;
      end
      S_On1: begin
         X <= 1;
         StateNext <= S_On2;
      end
      S_On2: begin
         X <= 1;
         StateNext <= S_On3;
      end
      S_On3: begin
         X <= 1;
         StateNext <= S_Off;
      end
   endcase
end
```

Finite-State Machines (FSMs)—Sequential Behavior

Procedures with Case Statements

- FSM’s CombLogic procedure
  - Case statement describes states
  - case (State)
    - Executes corresponding statement (often a begin-end block) based on State's current value
  - A state's statements consist of
    - Actions of the state
    - Setting of next state (transitions)
- Ex: State is S_On1
  - Executes statements for state On1, jumps to endcase



Suppose State is S_On1

```
reg [1:0] State, StateNext;

// CombLogic
always @(State, B) begin
  case (State)
    S_Off: begin
       X <= 0;
       if (B == 0)
          StateNext <= S_Off;
       else
          StateNext <= S_On1;
    end
    S_On1: begin
       X <= 1;
       StateNext <= S_On2;
    end
    S_On2: begin
       X <= 1;
       StateNext <= S_On3;
    end
    S_On3: begin
       X <= 1;
       StateNext <= S_Off;
    end
  endcase
end
```

Finite-State Machines (FSMs)—Sequential Behavior

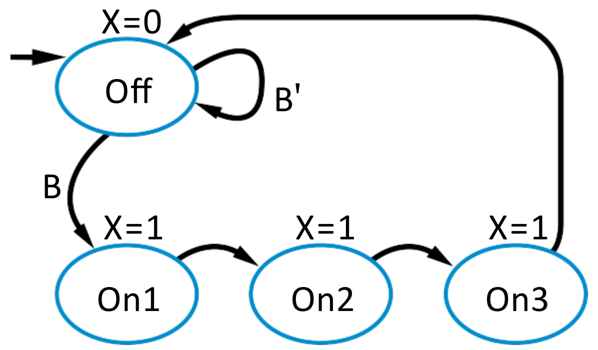
- FSM StateReg Procedure
  - Similar to 4-bit register
    - Register for State is 2-bit vector reg variable
  - Procedure has synchronous reset
    - Resets State to FSM's initial state, S_Off

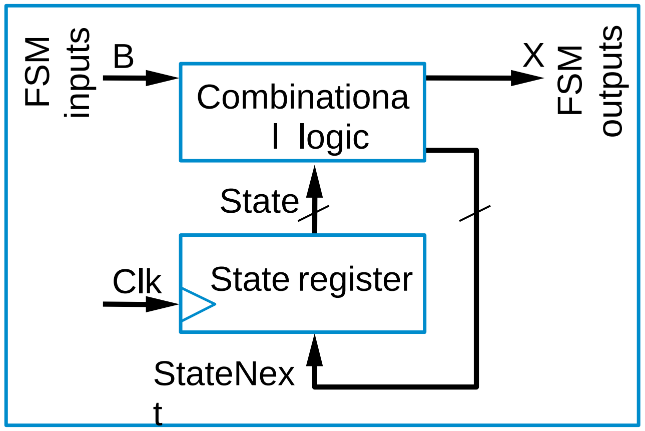
```
...
   parameter S_Off = 0, S_On1 = 1,
             S_On2 = 2, S_On3 = 3;

   reg [1:0] State, StateNext;

...

   // StateReg
   always @(posedge Clk) begin
      if (Rst == 1 )
         State <= S_Off;
      else
         State <= StateNext;
   end
...
```

Finite-State Machines (FSMs)—Sequential Behavior

Modules with Multiple Procedures and Shared Variables

Inputs: B; Outputs: X





```
`timescale 1 ns/1 ns

module LaserTimer(B, X, Clk, Rst);

   input B;
   output reg X;
   input Clk, Rst;

   parameter S_Off = 0, S_On1 = 1,
             S_On2 = 2, S_On3 = 3;

   reg [1:0] State, StateNext;

   // CombLogic
   always @(State, B) begin
      case (State)
         S_Off: begin
            X <= 0;
            if (B == 0)
               StateNext <= S_Off;
            else
               StateNext <= S_On1;
         end
         ...
         S_On1: begin
            X <= 1;
            StateNext <= S_On2;
         end
         S_On2: begin
            X <= 1;
            StateNext <= S_On3;
         end
         S_On3: begin
            X <= 1;
            StateNext <= S_Off;
         end
      endcase
   end

   // StateReg
   always @(posedge Clk) begin
      if (Rst == 1 )
         State <= S_Off;
      else
         State <= StateNext;
   end
endmodule
```