



CSE140L: Components and Design Techniques for Digital Systems Lab

FSMs

Instructor: Mohsen Imani

Slides from Tajana Simunic Rosing

Hardware Description Languages and Sequential Logic

- Flip-flops
 - representation of clocks - timing of state changes
 - asynchronous vs. synchronous
- FSMs
 - structural view (FFs separate from combinational logic)
 - behavioral view (synthesis of sequencers – not in this course)

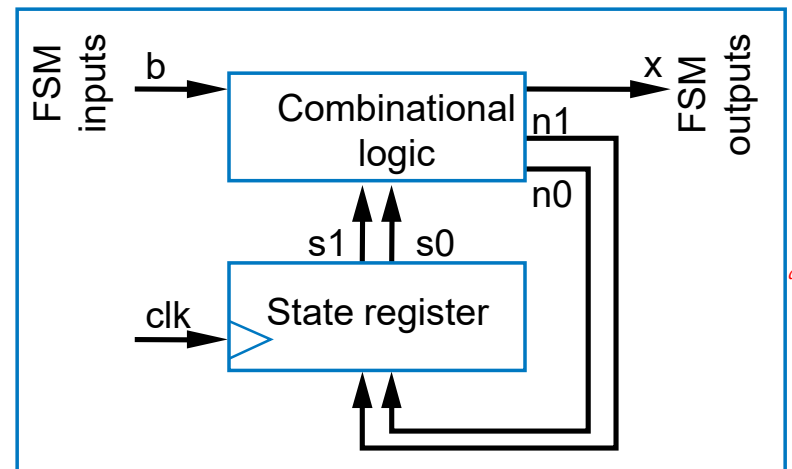
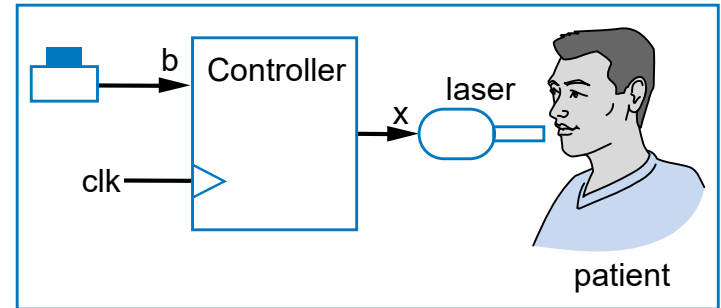
Controller Design

- Five step controller (FSM) design process

Step	Description
Step 1 <i>Capture the FSM</i>	Create an FSM that describes the desired behavior of the controller.
Step 2 <i>Create the architecture</i>	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 <i>Encode the states</i>	Assign a unique binary number to each state. Each binary number representing a state is known as an encoding . Any encoding will do as long as each state has a unique encoding.
Step 4 <i>Create the state table</i>	Create a truth table for the combinational logic such that the logic will generate the correct FSM outputs and next state signals. Ordering the inputs with state bits first makes this truth table describe the state behavior, so the table is a state table.
Step 5 <i>Implement the combinational logic</i>	Implement the combinational logic using any method.

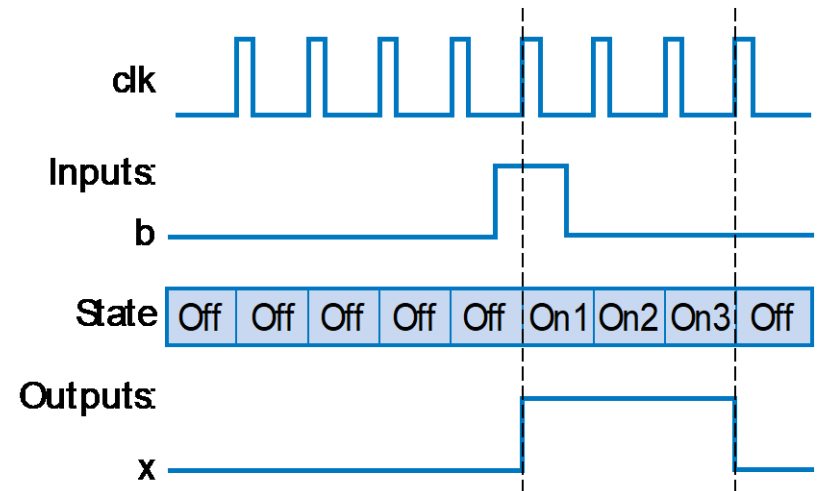
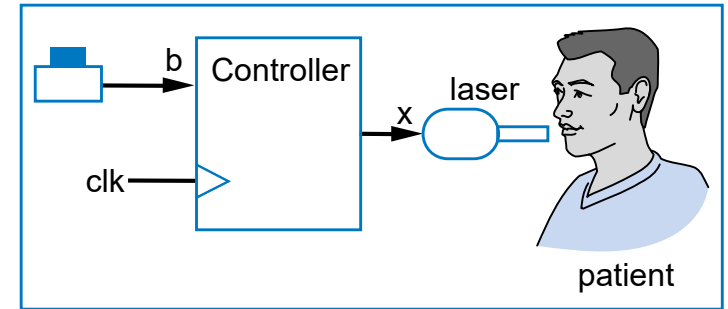
Controller Design: Laser Timer Example

- **Problem:** Pressing button (b) once turns on laser for 3 clock cycles
- **Step 1:** Capture the FSM
- **Step 2:** Create architecture
- **Step 3:** Encode the states
- **Step 4:** Minimize logic
- **Step 5:** Implement & test



Laser Timer

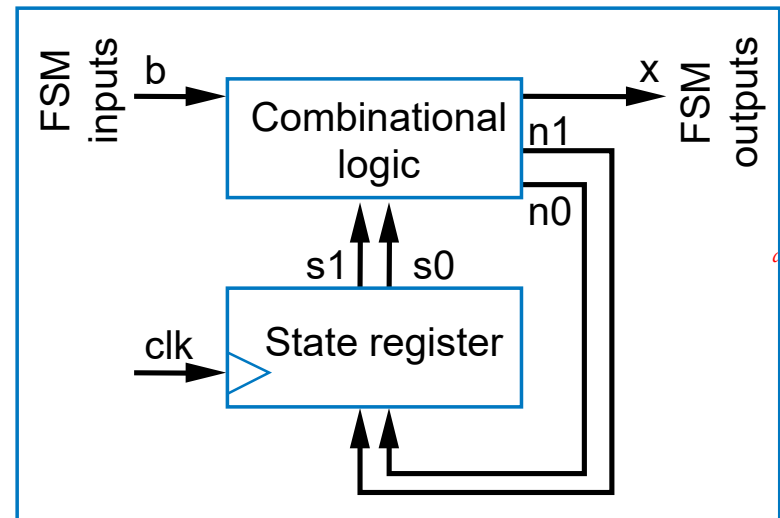
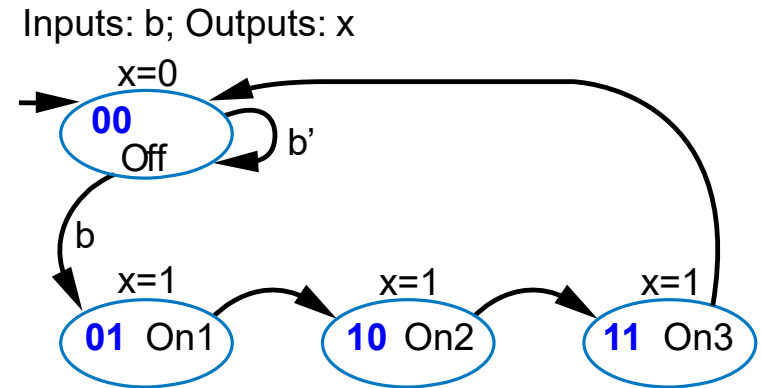
- Pressing button (b) once turns on laser for 3 clock cycles
- Step 1: Capture the FSM



Clock here is “compressed” to save space

Controller Design: Laser Timer Example

- Step 1: Capture the FSM
 - Already done
- Step 2: Create architecture
 - 2-bit state register (for 4 states)
 - Input b, output x
 - Next state signals n1, n0
- Step 3: Encode the states
 - Any encoding with each state unique will work

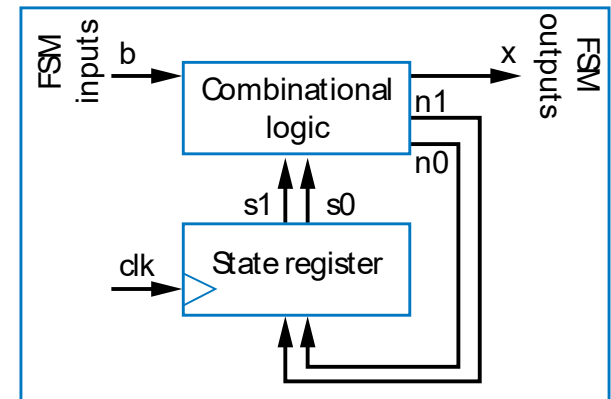
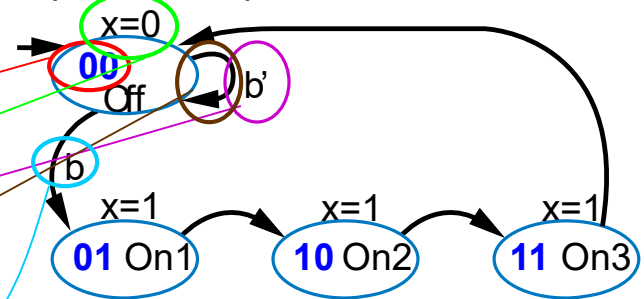


Controller Design: Laser Timer Example (cont)

- Step 4: Create state/excitation table

	Inputs			Outputs		
	s1	s0	b	x	n1	n0
<i>Off</i>	0	0	0	0	0	0
	0	0	1	0	0	1
<i>On1</i>	0	1	0	1	1	0
	0	1	1	1	1	0
<i>On2</i>	1	0	0	1	1	1
	1	0	1	1	1	1
<i>On3</i>	1	1	0	1	0	0
	1	1	1	1	0	0

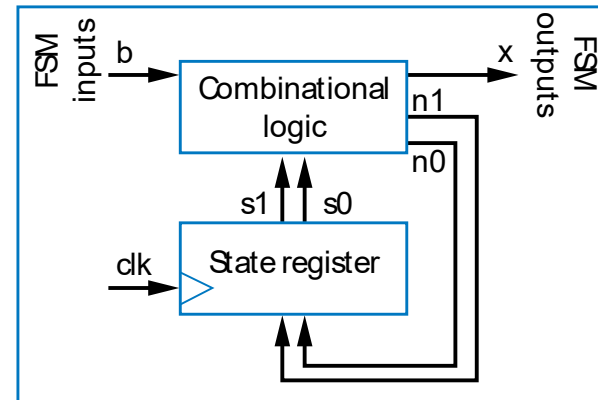
Inputs: b; Outputs: x



Controller Design: Laser Timer Example (cont)

- Step 5: Implement combinational logic

	Inputs			Outputs		
	s1	s0	b	x	n1	n0
<i>Off</i>	0	0	0	0	0	0
	0	0	1	0	0	1
<i>On1</i>	0	1	0	1	1	0
	0	1	1	1	1	0
<i>On2</i>	1	0	0	1	1	1
	1	0	1	1	1	1
<i>On3</i>	1	1	0	1	0	0
	1	1	1	1	0	0



$$x = s1 + s0$$

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

$$n1 = s1's0 + s1s0'$$

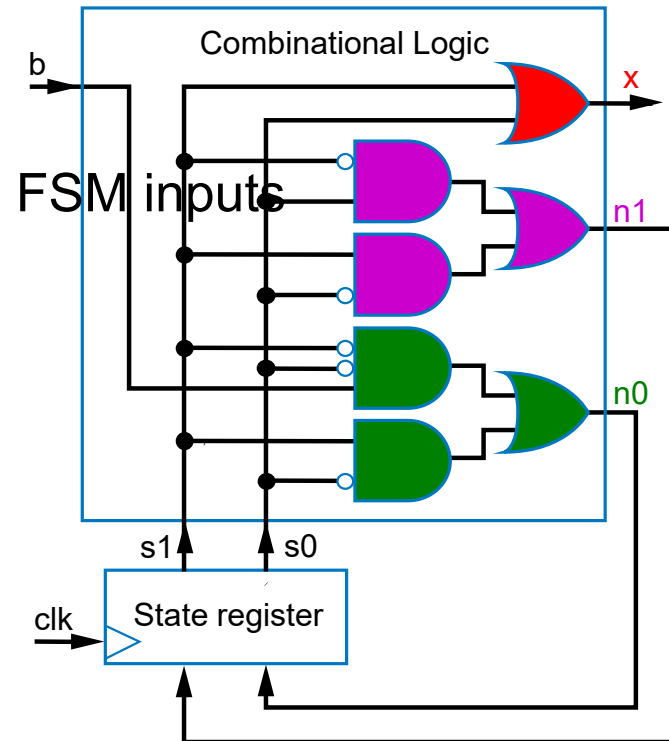
$$n0 = s1's0'b + s1s0'b' + s1s0'b$$

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

Controller Design: Laser Timer Example (cont)

- Step 5: Implement combinational logic (cont)

	Inputs			Outputs		
	s1	s0	b	x	n1	n0
<i>Off</i>	0	0	0	0	0	0
	0	0	1	0	0	1
<i>On1</i>	0	1	0	1	1	0
	0	1	1	1	1	0
<i>On2</i>	1	0	0	1	1	1
	1	0	1	1	1	1
<i>On3</i>	1	1	0	1	0	0
	1	1	1	1	0	0

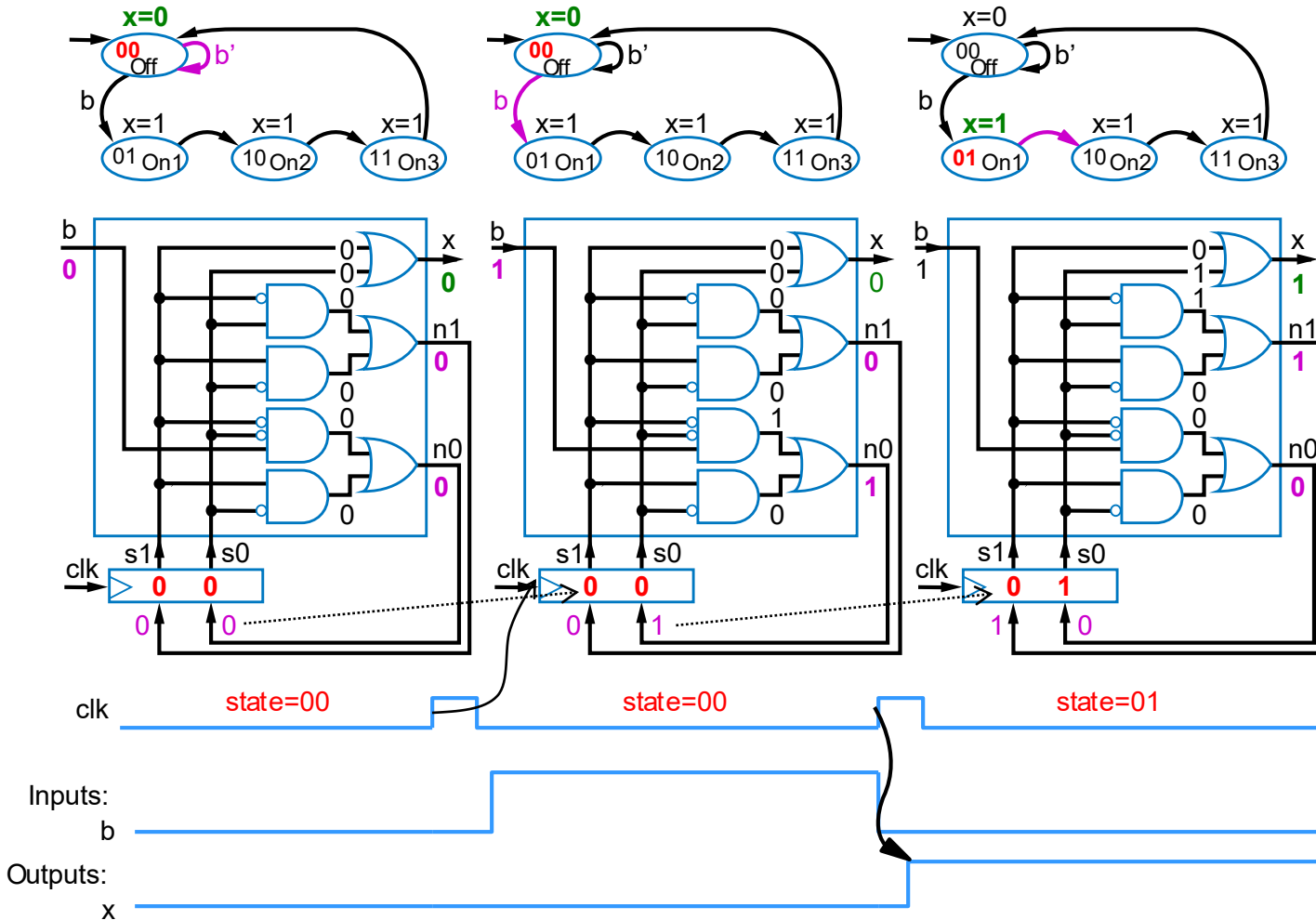


$$x = s1 + s0$$

$$n1 = s1's0 + s1s0'$$

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

Understanding the Controller's Behavior



Laser timer in Verilog

```
module LaserTimer(b, x, clk, rst);
input b, clk, rst;
output x;
reg x;

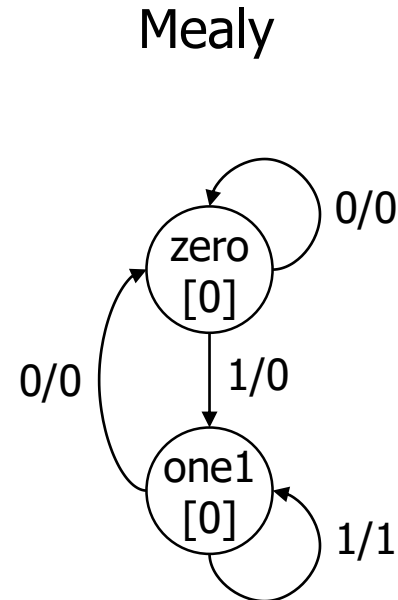
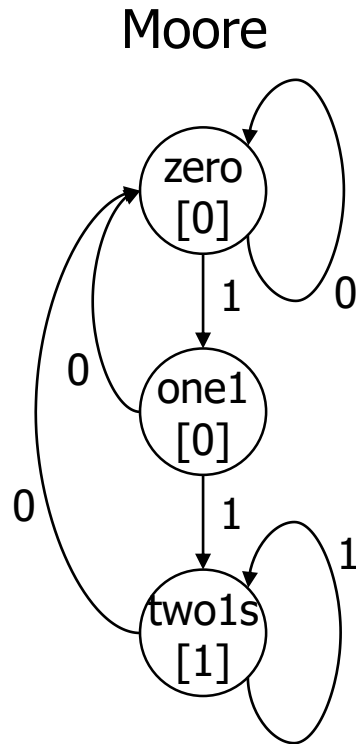
parameter S_Off = 2'b00,
          S_On1 = 2'b01,
          S_On2 = 2'b10,
          S_On3 = 2'b11;
reg [1:0] currentstate;
reg [1:0] nextstate;

// state register procedure
always @(posedge rst or posedge clk)
begin
    if (rst==1) // initial state
        currentstate <= S_Off;
    else
        currentstate <= nextstate;
end
```

```
// combinational logic procedure
always @(currentstate or b)
begin
    case (currentstate)
        S_Off: begin
            x <= 0; // laser off
            if (b==0) nextstate <= S_Off;
            else      nextstate <= S_On1;
        end
        S_On1: begin
            x <= 1; // laser on
            nextstate <= S_On2;
        end
        S_On2: begin
            x <= 1; // laser still on
            nextstate <= S_On3;
        end
        S_On3: begin
            x <= 1; // laser still on
            nextstate <= S_Off;
        end
    endcase
end
endmodule // LaserTimer
```

FSM design example – Moore vs. Mealy

- Remove one 1 from every string of 1s on the input

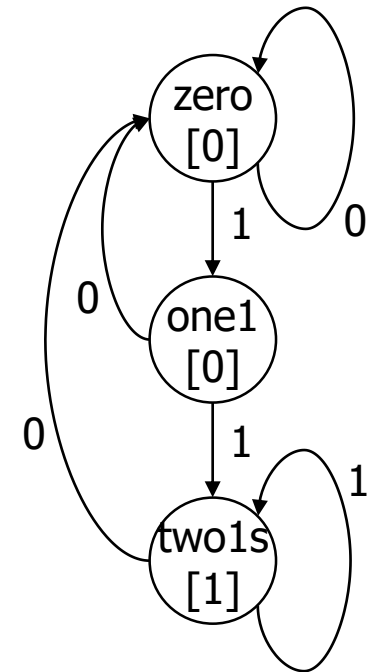


Verilog FSM - Reduce 1s example

- Moore machine

```
module reduce (clk, reset, in, out);  
  input clk, reset, in;  
  output out;  
  
  parameter zero    = 2'b00;  
  parameter one1    = 2'b01;  
  parameter two1s   = 2'b10;  
  
  reg out;  
  reg [2:1] state;      // state variables  
  reg [2:1] next_state;  
  
  always @(posedge clk)  
    if (reset) state = zero;  
    else      state = next_state;
```

state assignment
(easy to change,
if in one place)



Moore Verilog FSM (cont'd)

```
always @(in or state)
```

crucial to include
all signals that are
input to state determination

```
case (state)
  zero:
    // last input was a zero
    begin
      if (in) next_state = one1;
      else   next_state = zero;
    end
  one1:
    // we've seen one 1
    begin
      if (in) next_state = twos;
      else   next_state = zero;
    end
  twos:
    // we've seen at least 2 ones
    begin
      if (in) next_state = twos;
      else   next_state = zero;
    end
endcase
```

note that output
depends only on state

```
always @(state)
  case (state)
    zero: out = 0;
    one1: out = 0;
    twos: out = 1;
  endcase
```

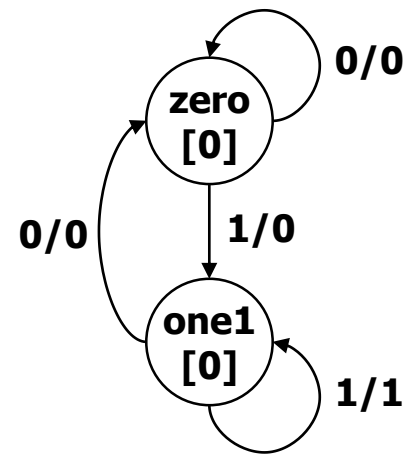
```
endmodule
```

Mealy Verilog FSM

```
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg state; // state variables
  reg next_state;

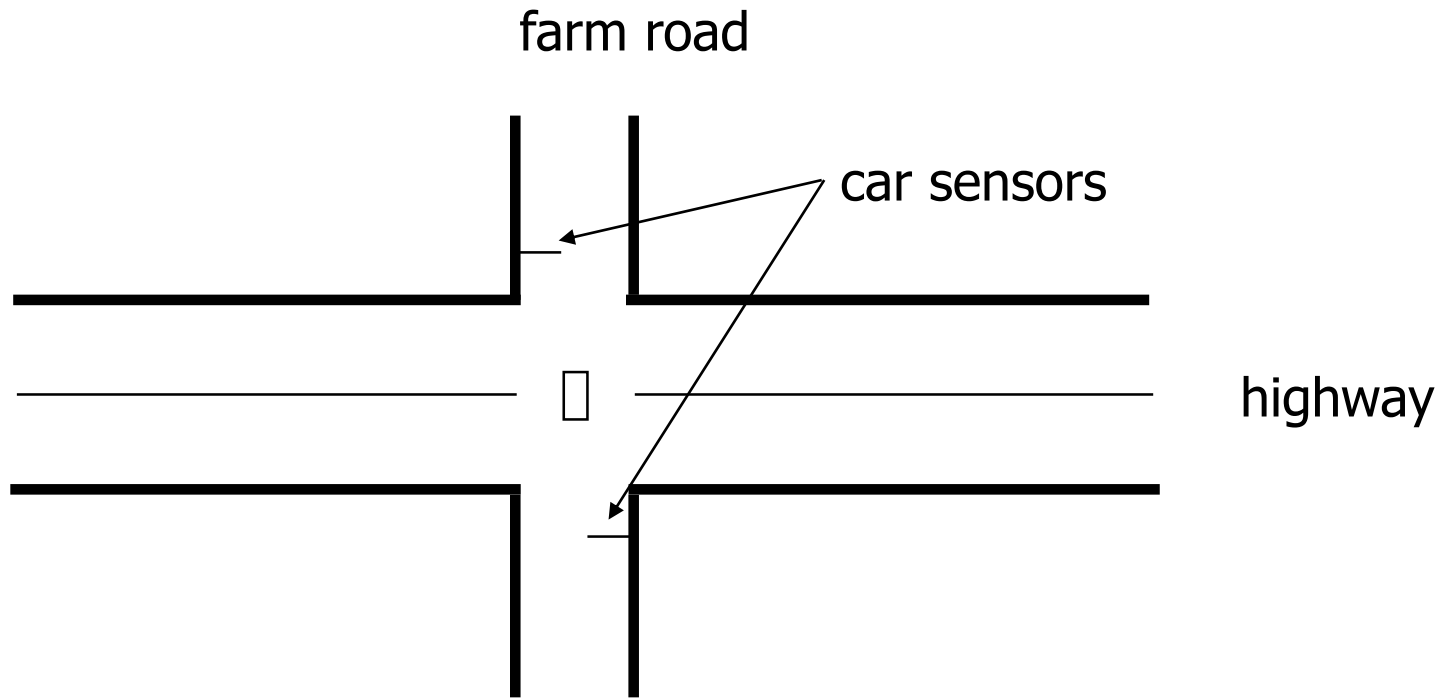
  always @(posedge clk)
    if (reset) state = zero;
    else      state = next_state;

  always @(in or state)
    case (state)
      zero:           // last input was a zero
        begin
          out = 0;
          if (in) next_state = one;
          else   next_state = zero;
        end
      one:            // we've seen one 1
        if (in) begin
          next_state = one; out = 1;
        end else begin
          next_state = zero; out = 0;
        end
    endcase
endmodule
```



Example: Traffic light controller

- Highway/farm road intersection

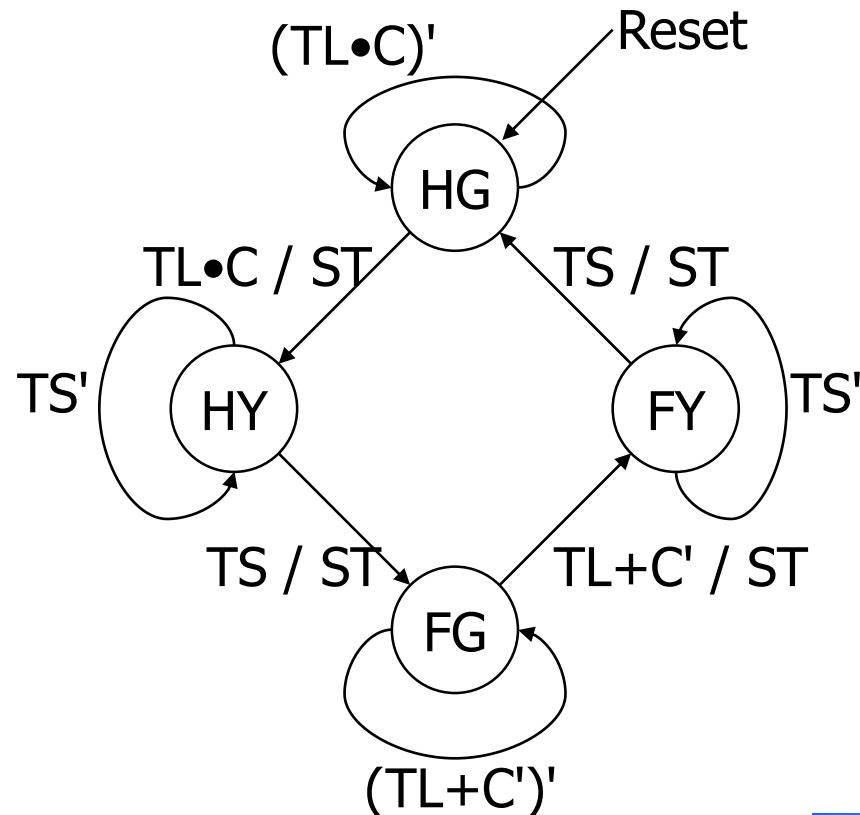


Traffic light controller (cont.)

- Detectors C sense the presence of cars waiting on the farm road
 - with no car on farm road, light remain green in highway direction
 - if vehicle on farm road, highway lights go from Green to Yellow to Red, allowing the farm road lights to become green
 - these stay green only as long as a farm road car is detected but never longer than a set interval; after the interval expires, farm lights transition from Green to Yellow to Red, allowing highway to return to green
 - even if farm road vehicles are waiting, highway gets at least a set interval of green
- Assume you have an interval timer that generates:
 - a short time pulse (TS) and
 - a long time pulse (TL),
 - in response to a set (ST) signal.
 - TS is to be used for timing yellow lights and TL for green lights

Traffic light controller (cont.)

- | inputs | description | outputs | description |
|--------|---------------------------------|------------|--|
| reset | place FSM in initial state | HG, HY, HR | assert green/yellow/red highway lights |
| C | detect vehicle on the farm road | FG, FY, FR | assert green/yellow/red highway lights |
| TS | short time interval expired | ST | start timing a short or long interval |
| TL | long time interval expired | | |
- | state | description |
|-------|--------------------------------|
| HG | highway green (farm road red) |
| HY | highway yellow (farm road red) |
| FG | farm road green (highway red) |
| FY | farm road yellow (highway red) |



Traffic light controller (cont.)

- Generate state table with symbolic states
- Consider state assignments

output encoding – similar problem to state assignment
(Green = 00, Yellow = 01, Red = 10)

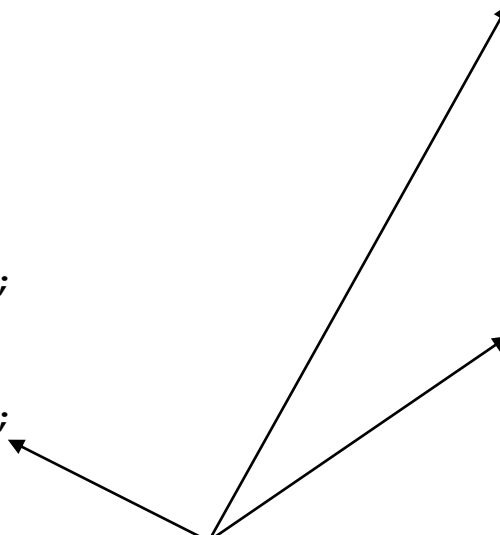
Inputs			Present State	Next State	Outputs		
C	TL	TS			ST	H	F
0	–	–	HG	HG	0	Green	Red
–	0	–	HG	HG	0	Green	Red
1	1	–	HG	HY	1	Green	Red
–	–	0	HY	HY	0	Yellow	Red
–	–	1	HY	FG	1	Yellow	Red
1	0	–	FG	FG	0	Red	Green
0	–	–	FG	FY	1	Red	Green
–	1	–	FG	FY	1	Red	Green
–	–	0	FY	FY	0	Red	Yellow
–	–	1	FY	HG	1	Red	Yellow

SA1:	HG = 00	HY = 01	FG = 11	FY = 10	
SA2:	HG = 00	HY = 10	FG = 01	FY = 11	
SA3:	HG = 0001	HY = 0010	FG = 0100	FY = 1000	(one-hot)

Traffic light controller FSM

- Specification of inputs, outputs, and state elements

```
module FSM(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, Clk);  
    output    HR;  
    output    HY;  
    output    HG;  
    output    FR;  
    output    FY;  
    output    FG;  
    output    ST;  
    input     TS;  
    input     TL;  
    input     C;  
    input     reset;  
    input     Clk;  
  
    reg [6:1] state;  
    reg      ST;  
  
    parameter highwaygreen  = 6'b001100;  
    parameter highwayyellow = 6'b010100;  
    parameter farmroadgreen  = 6'b100001;  
    parameter farmroadyellow = 6'b100010;  
  
    assign HR = state[6];  
    assign HY = state[5];  
    assign HG = state[4];  
    assign FR = state[3];  
    assign FY = state[2];  
    assign FG = state[1];
```



The diagram shows three arrows originating from the 'state' variable declaration 'reg [6:1] state;' and pointing to the assignment lines for HR, HY, and HG. This illustrates that the state variable is used to determine the values of these outputs.

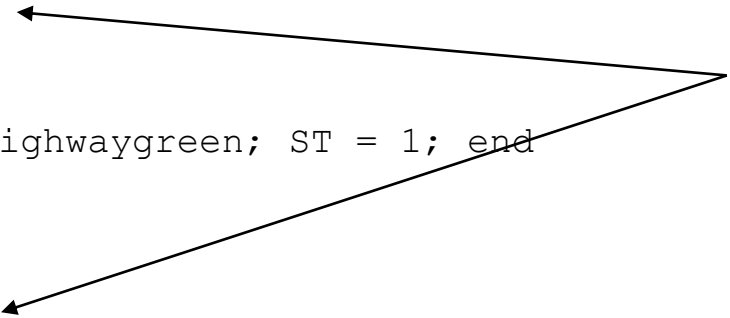
specify state bits and codes
for each state as well as
connections to outputs

Traffic light controller FSM

```
initial begin state = highwaygreen; ST = 0; end

always @(posedge Clk)
begin
    if (reset)
        begin state = highwaygreen; ST = 1; end
    else
        begin
            ST = 0;
            case (state)
                highwaygreen:
                    if (TL & C) begin state = highwayyellow; ST = 1; end
                highwayyellow:
                    if (TS) begin state = farmroadgreen; ST = 1; end
                farmroadgreen:
                    if (TL | !C) begin state = farmroadyellow; ST = 1; end
                farmroadyellow:
                    if (TS) begin state = highwaygreen; ST = 1; end
            endcase
        end
    end
end
endmodule
```

case statement triggered by clock edge



Timer FSM for traffic light controller

```
module Timer(TS, TL, ST, Clk);
    output TS;
    output TL;
    input    ST;
    input    Clk;
    integer  value;

    assign TS = (value >= 4); // 5 cycles after reset
    assign TL = (value >= 14); // 15 cycles after reset

    always @(posedge ST) value = 0; // async reset

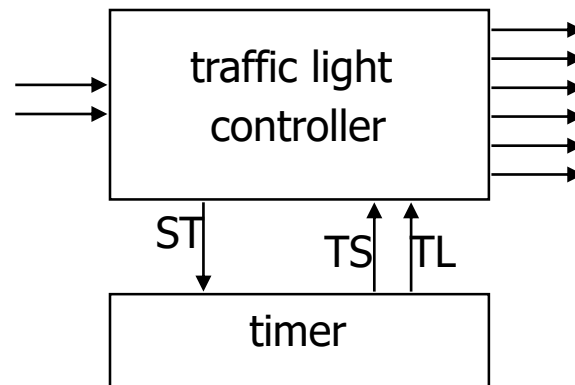
    always @(posedge Clk) value = value + 1;

endmodule
```

Complete traffic light controller

- Tying it all together (FSM + timer) with structural Verilog (same as a schematic drawing)

```
module main(HR, HY, HG, FR, FY, FG, reset, C, Clk);  
  output HR, HY, HG, FR, FY, FG;  
  input  reset, C, Clk;  
  
  Timer part1(TS, TL, ST, Clk);  
  FSM   part2(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, Clk);  
endmodule
```



Finite state machines summary

- Models for representing sequential circuits
 - abstraction of sequential elements
 - finite state machines and their state diagrams
 - inputs/outputs
 - Mealy, Moore, and synchronous Mealy machines
- Finite state machine design procedure
 - deriving state diagram
 - deriving state transition table
 - determining next state and output functions
 - implementing combinational logic
- Hardware description languages
 - Use good coding style
 - Communicating FSMs