ENGR 210 / CSCI B441
“Digital Design”

Finite State Machines II

Andrew Lukefahr

Announcements

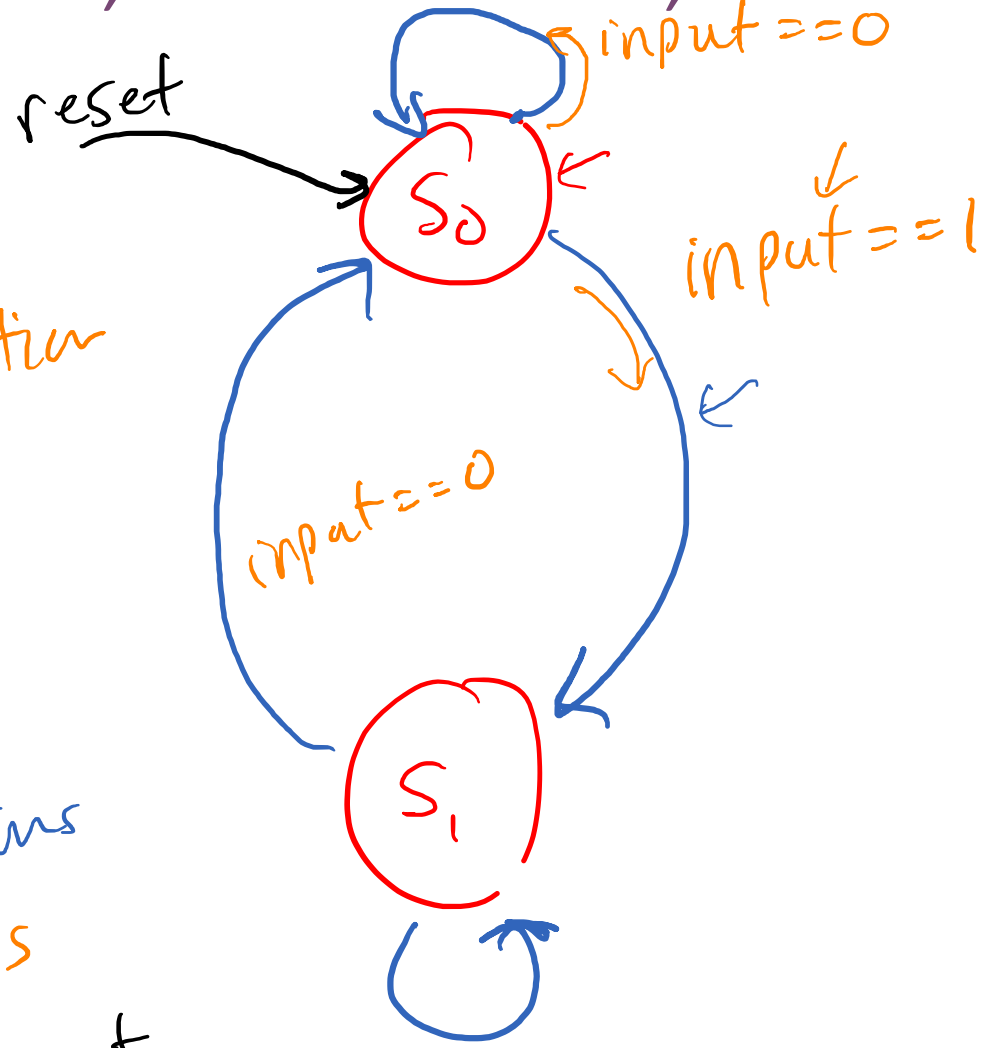
- P3 is due Friday.
  - Adds “demo” requirement. Will need real hardware.
  - Demo: create video and upload to Canvas

- Remote Students
  - Setup dedicated machines in Luddy 4111.
  - Should have received email.

- Labs/Office Hours
  - Will remain “Virtual” for now
  - You can work from home or from Luddy 4111

Always specify defaults for `always_comb`!

Review: States, Transitions, and Guards



guards: control transition which you take

S_0 & S_1: are states

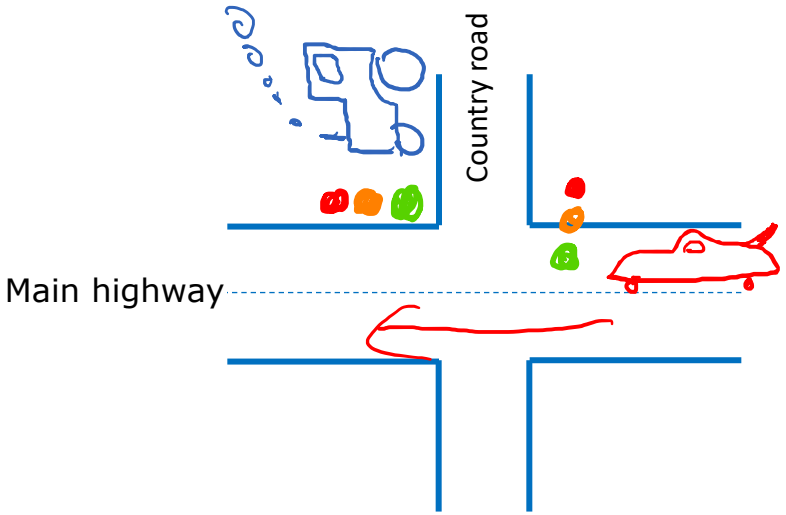
two states in this machine

transitions
→ leaving one state & going to another
(happen @ posedge clk)

S_0 & S_1 => states
~~blue~~ lines => transitions
booleans => guards
reset => reset

FSM: Traffic Signal Controller

- A controller for traffic at the intersection of a main highway and a country road.



- The main highway gets priority because it has more cars
  - The main highway signal remains green by default.

Traffic signal controller

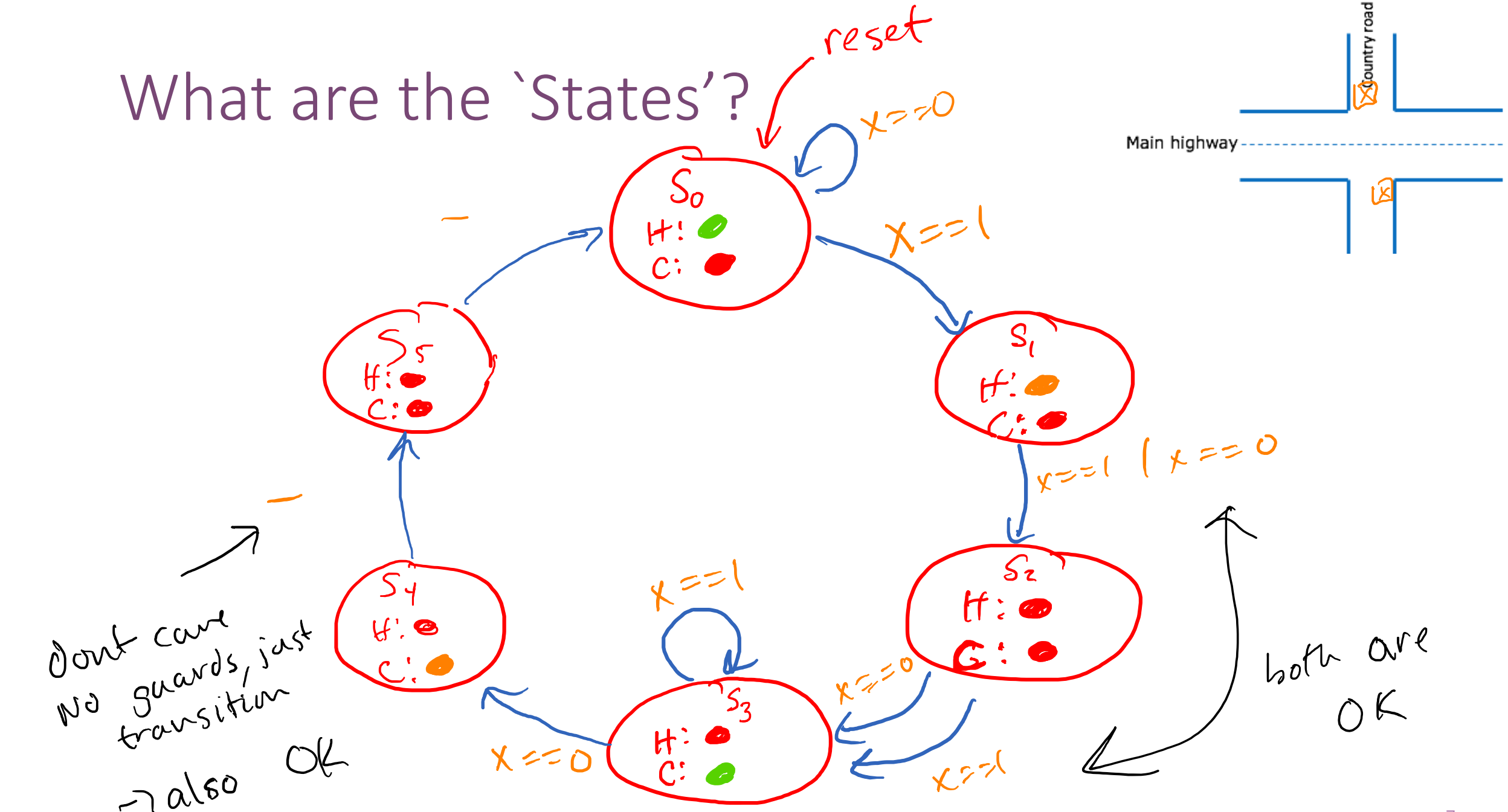
- Cars occasionally arrive from the country road. The traffic signal for the country road must turn green only long enough to let the cars on the country road go.
- When no cars are waiting on the country road, the country road traffic signal turns yellow then red and the traffic signal on the main highway turns green again.

There is a sensor to detect cars waiting on the country road. The sensor sends a signal X as input to the controller:

$X = 1$, if there are cars on the country road

$X = 0$, otherwise

What are the `States'?

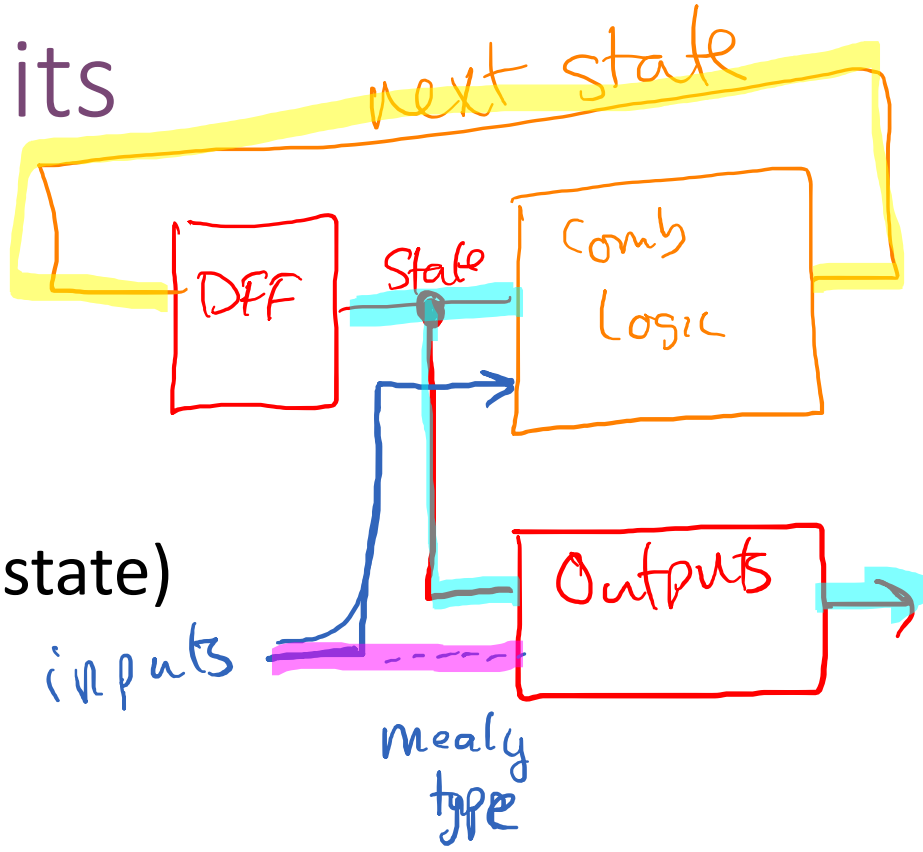


Moore vs. Mealy Type FSMs

- Thus far we've done "Moore" Type
  - Moore Type: Outputs determined by the state (circle)
- Another technique: "Mealy" Type
  - Mealy Type: Output determined by the transition (arrow)
- Moore: Easier, but more states
- Mealy: Less states, more complicated transitions

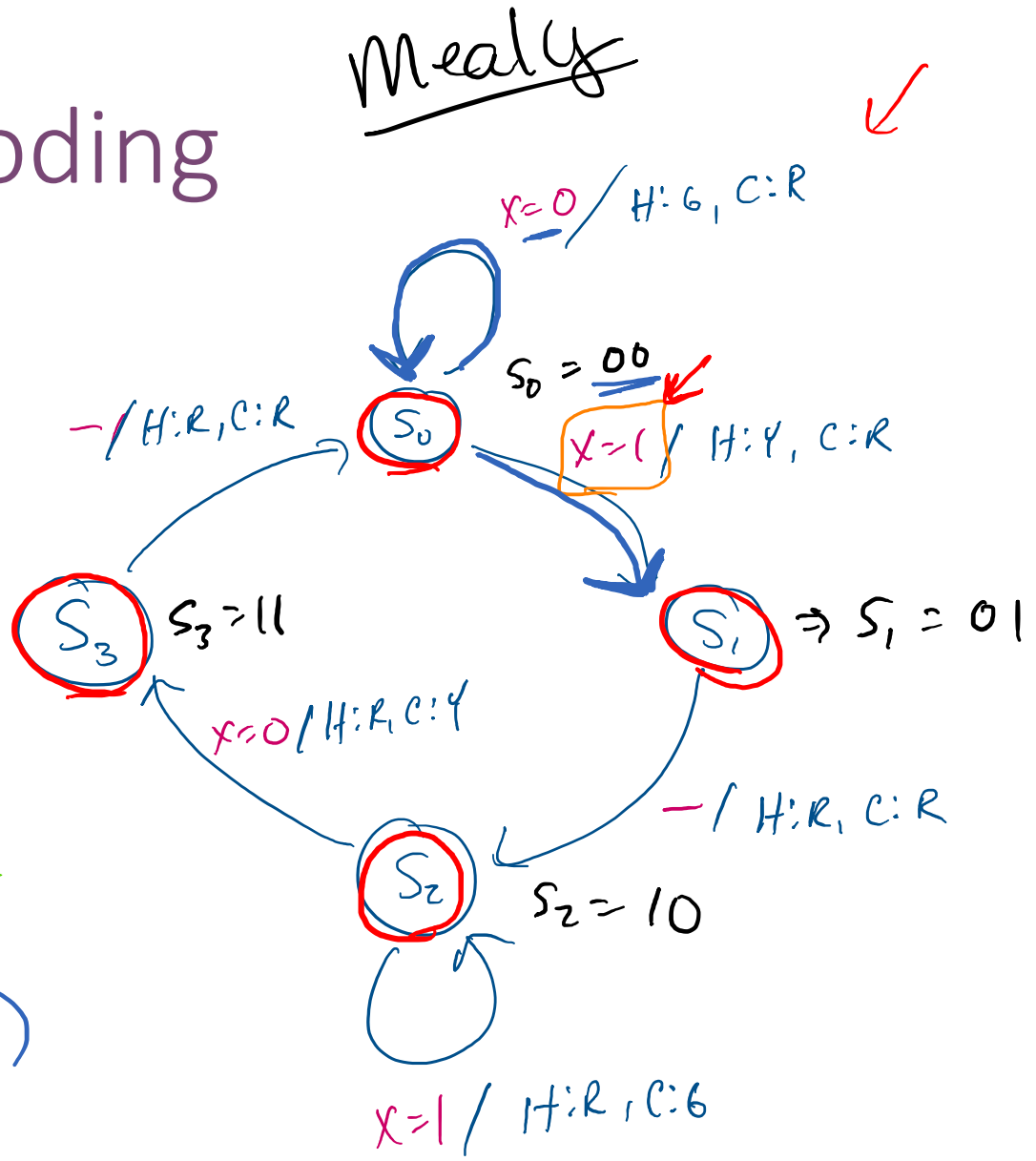
Implementing FSMs with Circuits

- Encode each state as a number
  - Store this with DFF's
- Generate state transition logic (arrow to next state)
  - Use combinational logic
- Generate output given state + inputs
  - Use combinational logic



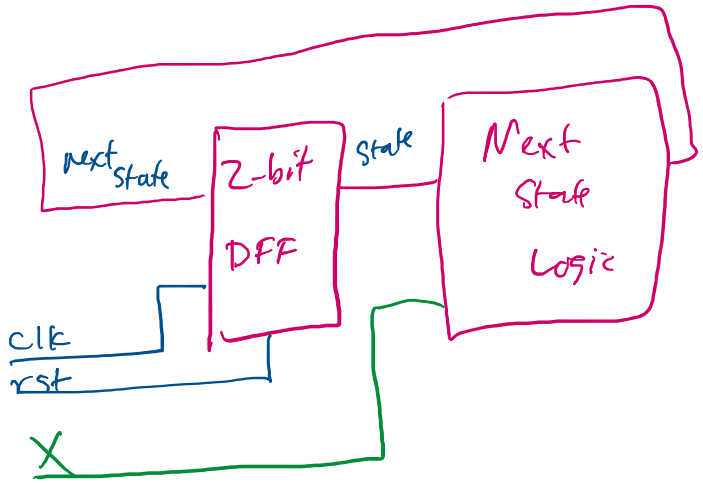
State Transition Encoding

State	X	Next State
0 0 (S_0)	0	00 (S_0)
0 0 (S_0)	1	01 (S_1)
0 1	0	10 (S_2)
0 1	1	10 (S_2)
1 0	0	11 (S_3)
1 0	1	10 (S_2)
1 1	0	00 (S_0)
1 1	1	00 (S_0)



State Machine Encoding

	State	X	Next State
0	00	0	00
1	00	1	01
2	→ 01	0	10
3	→ 01	1	00
4	→ 10	0	11 ←
5	→ 10	1	10
6	11	0	00
7	11	1	00

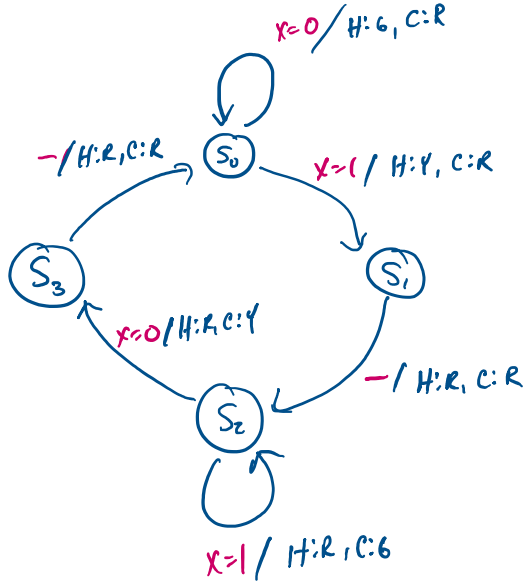


$$Next\ State[1] = (\overline{state[1]} \,\&\, state[0] \,\&\, \overline{X}) \mid (\overline{state[1]} \,\&\, state[0] \,\&\, X) \mid (state[1] \,\&\, \overline{state[0]} \,\&\, \overline{X}) \mid (state[1] \,\&\, \overline{state[0]} \,\&\, X)$$

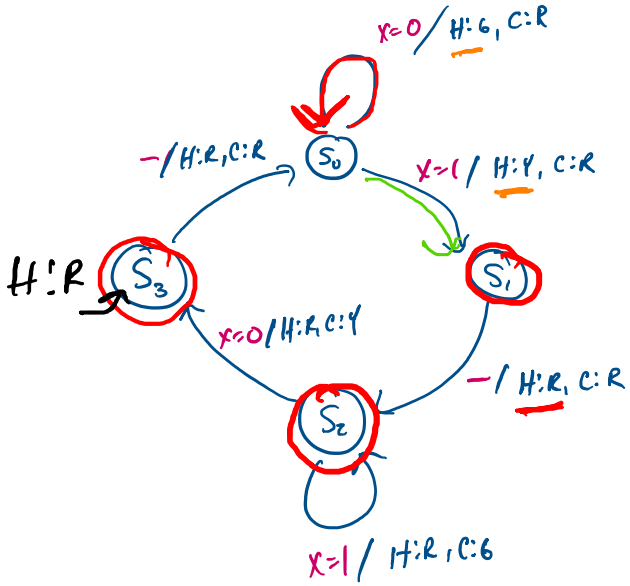
$$Next\ State[0] = (\overline{state[1]} \,\&\, \overline{state[0]} \,\&\, X) \mid (state[1] \,\&\, state[0] \,\&\, \overline{X})$$

Next State Logic

State	X	Next State
00	0	00
00	1	01
01	0	10
01	1	10
10	0	11
10	1	10
11	0	00
11	1	00

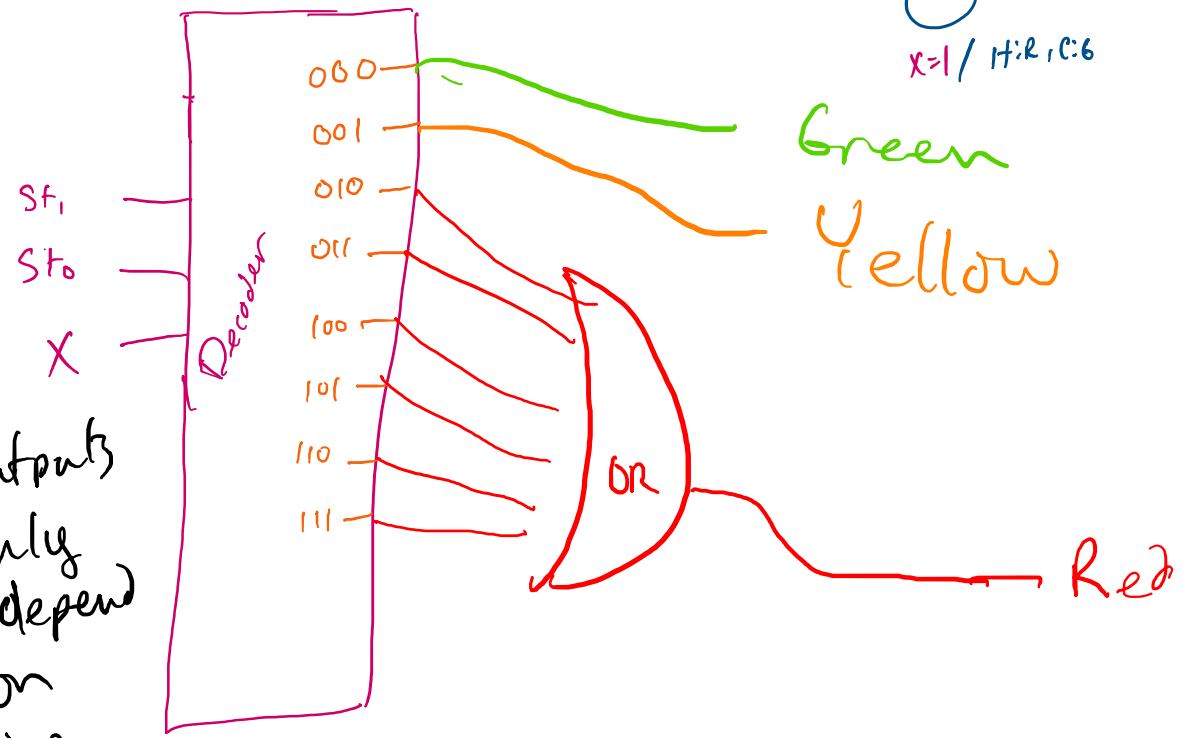


Output Logic (Highway)

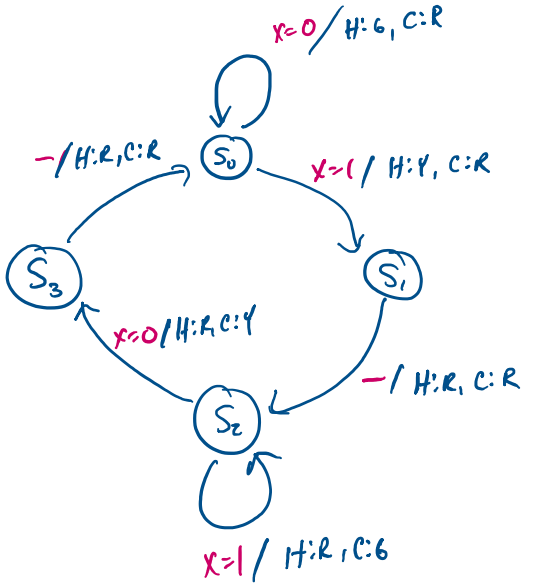


	State	X	R	Y	G
0	00	0	0	0	1
1	00	1	0	1	0
2	01	0	1	0	0
3	01	1	1	0	0
4	10	0	1	0	0
5	10	1	1	0	0
6	11	0	1	0	0
7	11	1	1	0	0

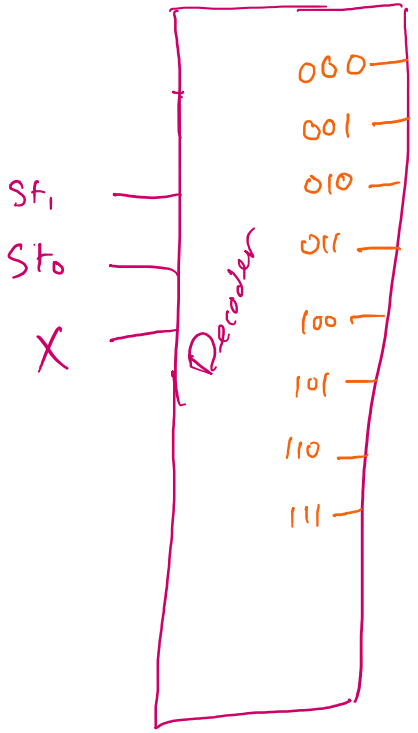
outputs only depend on ~~input~~ State



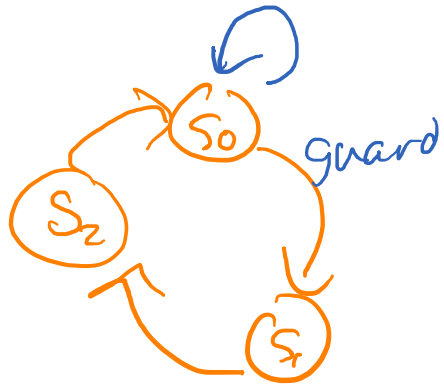
Output Logic (Country Rd)

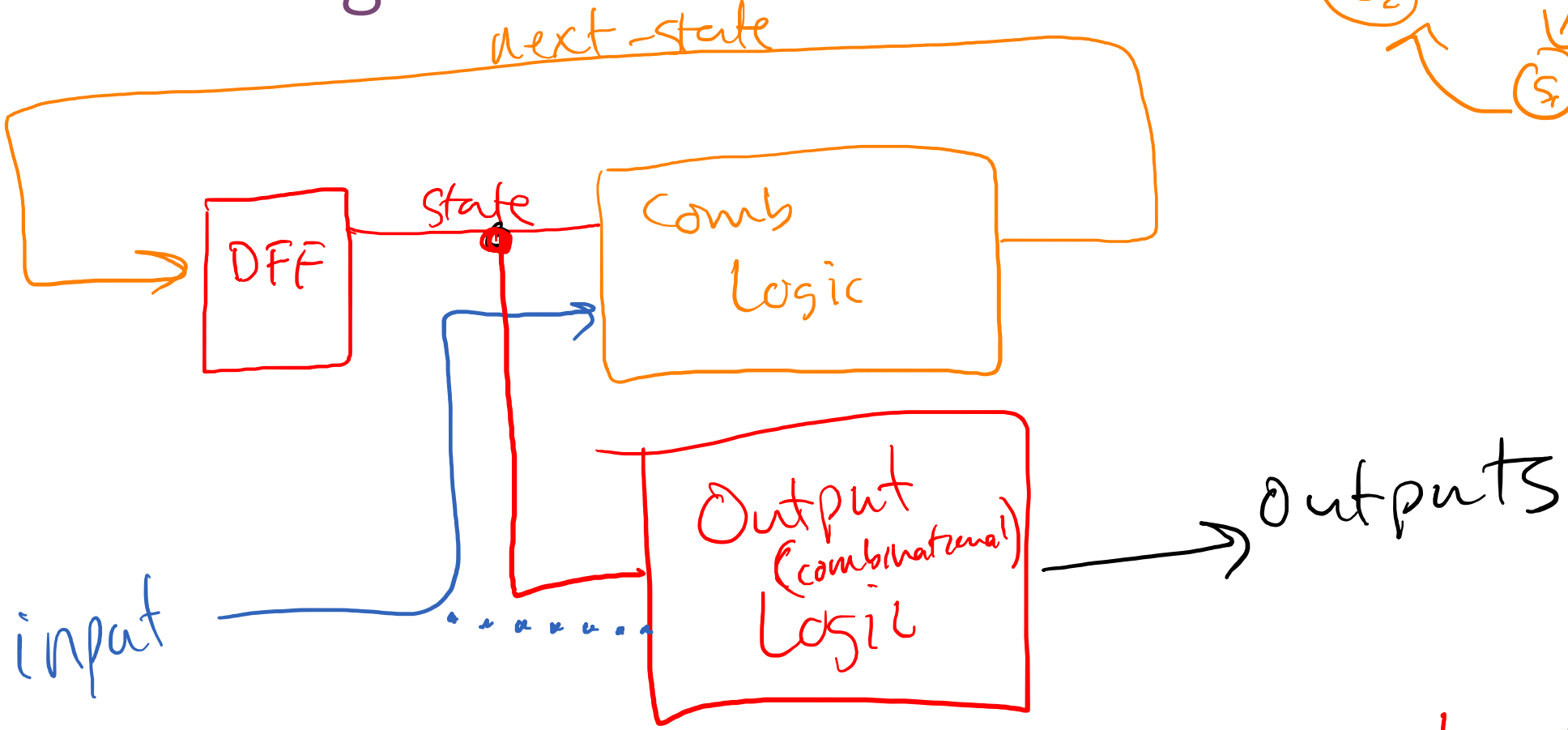


	State	X
0	00	0
1	00	1
2	01	0
3	01	1
4	10	0
5	10	1
6	11	0
7	11	1



Implementing FSMs with Circuits

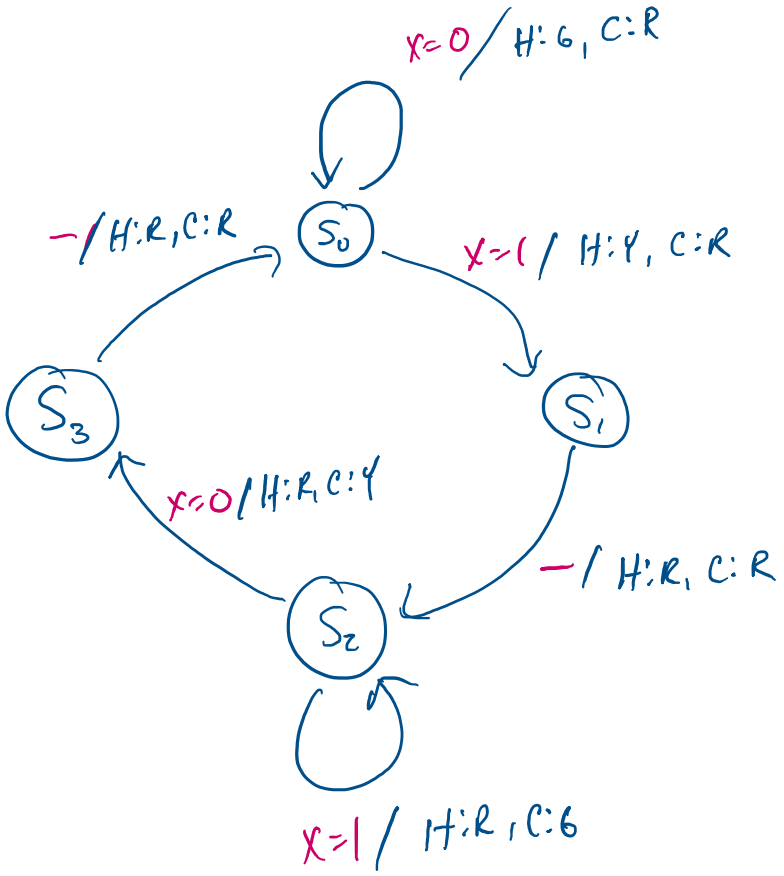


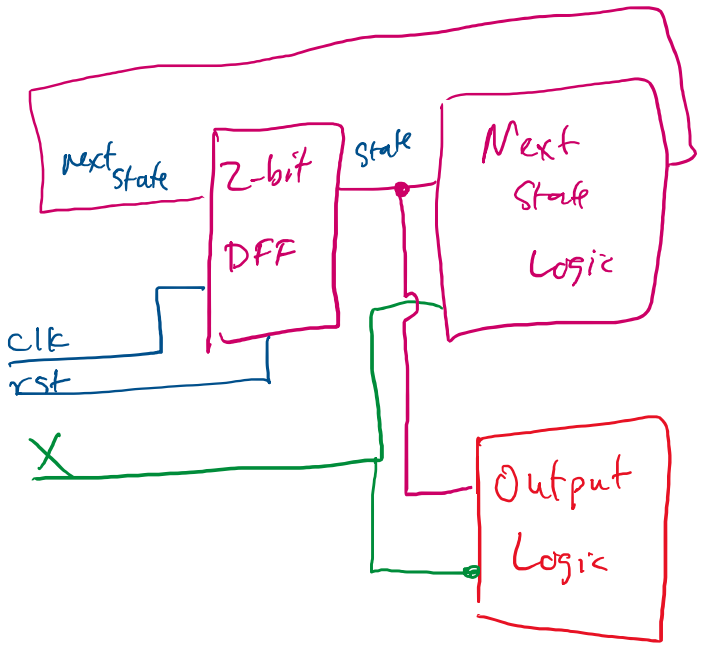


Moore Type: outputs depend on State

Mealy Type: outputs depend on State & inputs

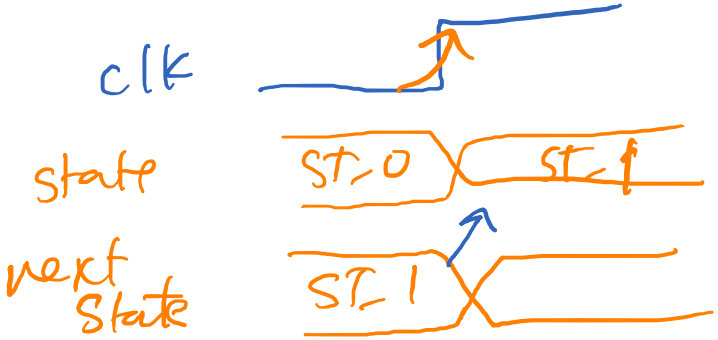
State Machine to Logic

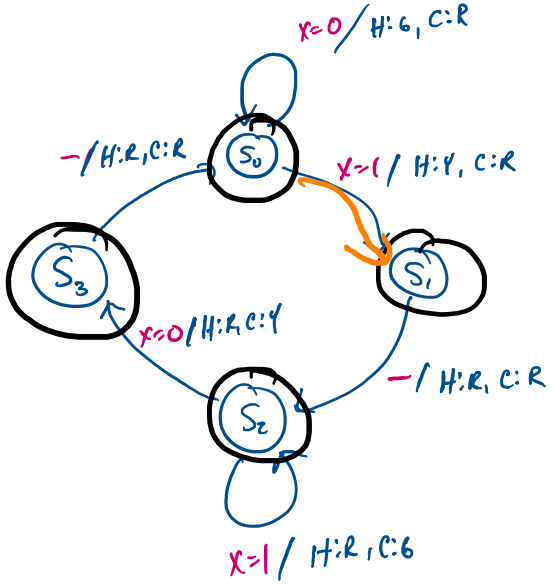




State Machine to Verilog

- Define states?

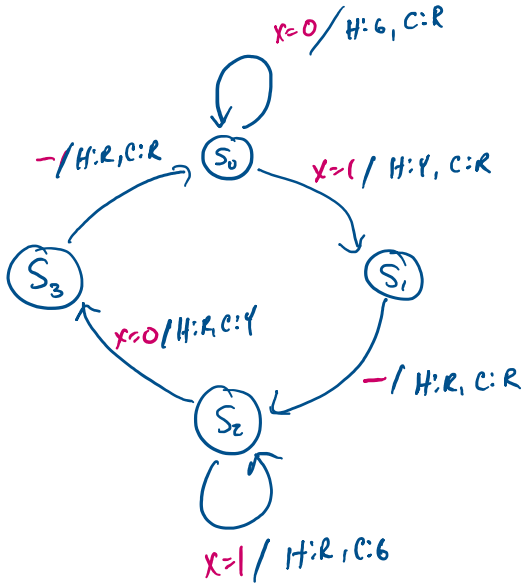





```
enum { ST_0, ST_1, ST_2, ST_3 } state, next_state;

always_ff @ (posedge clk) begin
    if (rst)  state <= ST_0;
    else      state <= next_state;
end
```

State Machine to Verilog

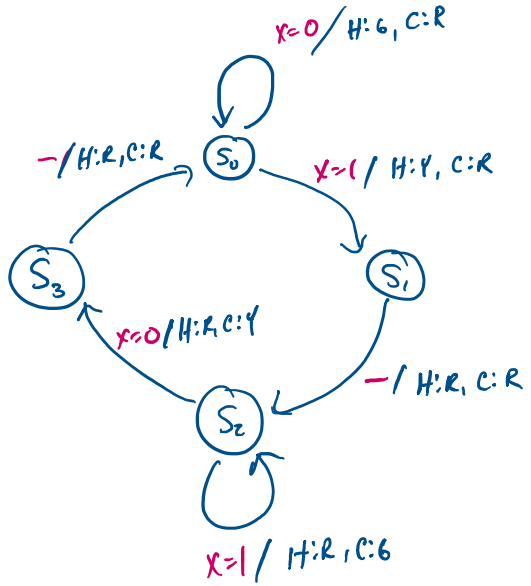
- Define states?




```
enum { ST_0, ST_1, ST_2, ST_3} state, nextState;
```

State Machine to Verilog

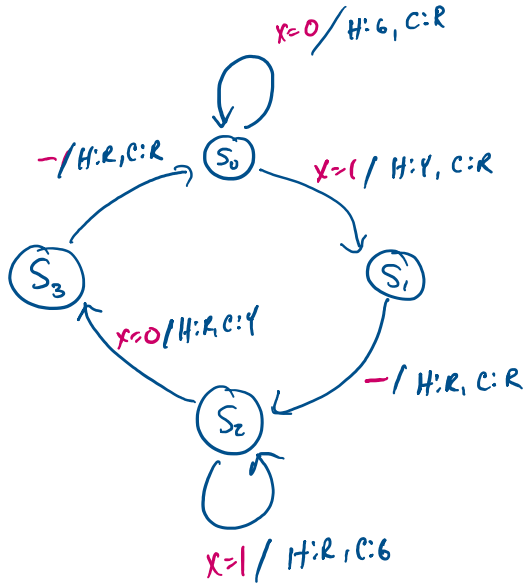
- Build State Machine?



State Machine to Verilog

- Build State Machine?

```
always_ff @(posedge clk) begin
    if (rst) state <= ST_0;
    else state <= nextState;
end
```



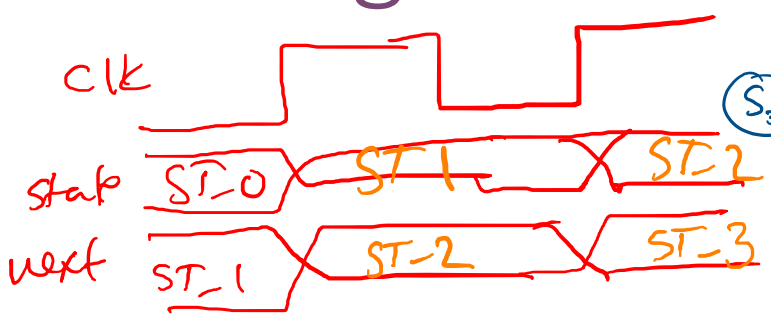
- What is `nextState`?

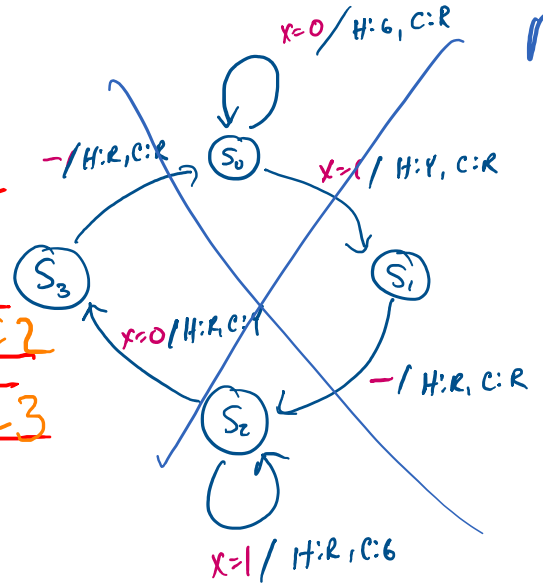
State Machine to Verilog

enum ...

```
always_ff @(posedge clk) begin
    if (rst) state <= ST_0;
    else state <= nextState;
end
```

= ST_1

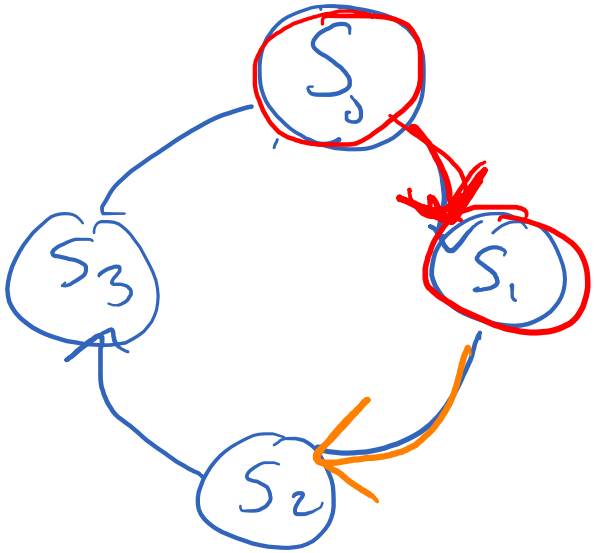




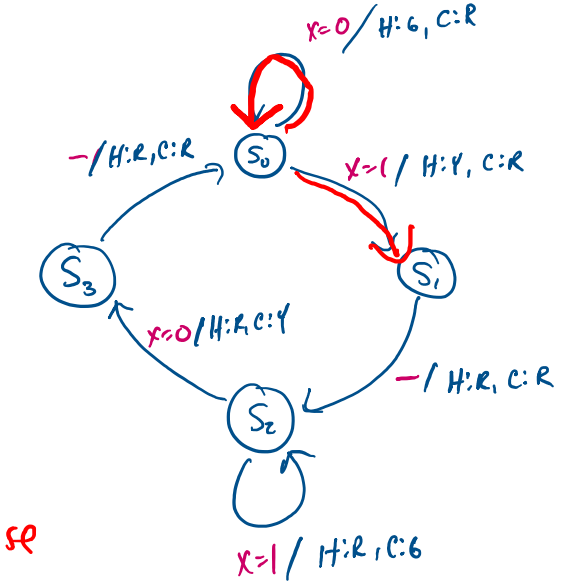
NOT Yet!

```
always_comb begin
    nextState = state; //default
    case(state)
        ST_0: nextState = ST_1; //goto state 1
        ST_1: nextState = ST_2;
        ST_2: nextState = ST_3;
        ST_3: nextState = ST_0; //loop
        default: nextState = ST_0; //just in case
    endcase
end
```

ST_1



State Machine to Verilog



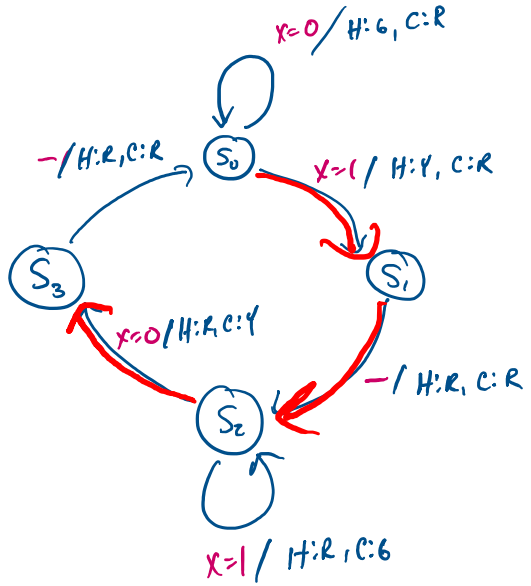
- What is this missing?

```
always_comb begin
    nextState = state; //default
    case(state)
        ST_0: nextState = ST_1; //goto state 1
        ST_1: nextState = ST_2;
        ST_2: nextState = ST_3;
        ST_3: nextState = ST_0; //loop
        default: nextState = ST_0; //just in case
    endcase
end
```

State Machine to Verilog

- What is this missing?

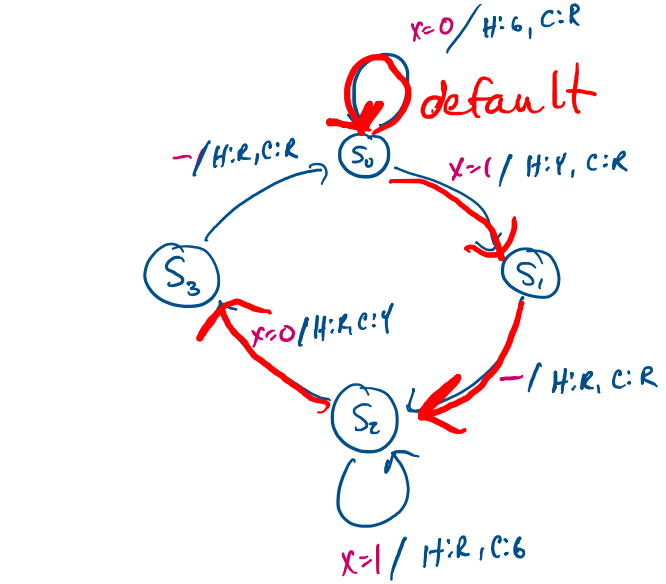
```
always_comb begin
        nextState = state; //default
        case(state)
                ST_0:   if(x)
                        nextState = ST_1;
                ST_1:
                        nextState = ST_2;
                ST_2:   if (~x)
                        nextState = ST_3;
                // ST_3 and default cases   endcase
end
```

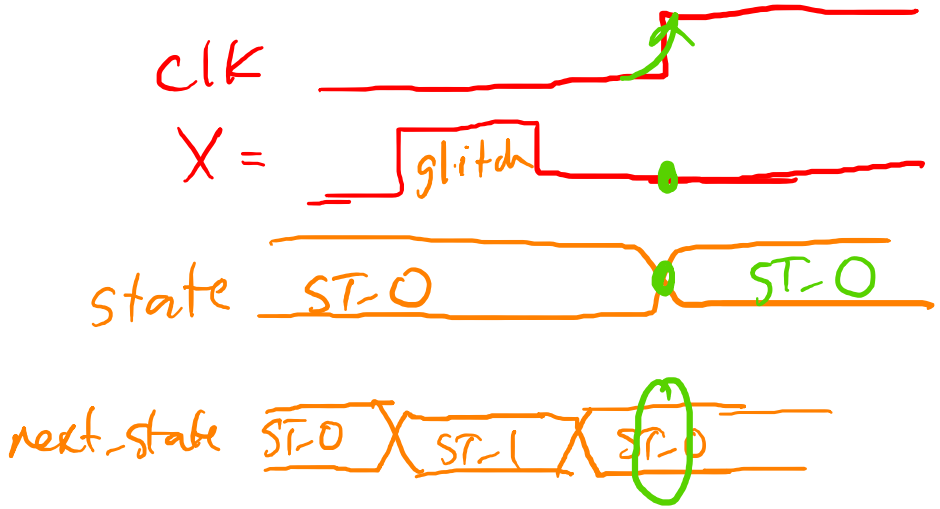


State Machine to Verilog

- What is this missing?

```
always_comb begin
    nextState = state; //default
    case(state)
        ST_0:
            if (X) nextState = ST_1;
        ST_1:
            nextState = ST_2;
        ST_2:
            if (~X) nextState = ST_3;
        // ST_3 and default cases
    endcase
end
```

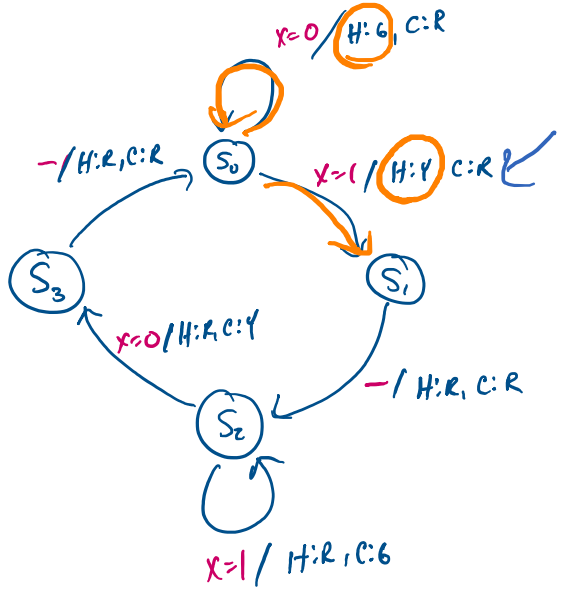




State Machine to Verilog

logic [2:0] Hryg;

- What else is this missing?

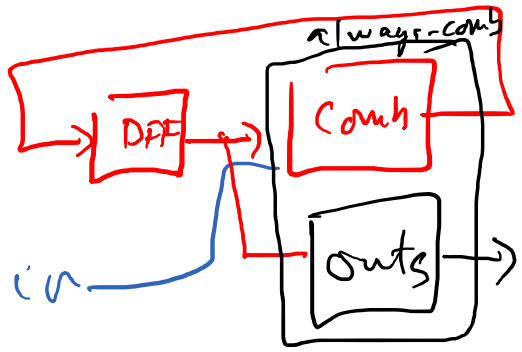



```
always_comb begin
        nextState = state; //default
        case(state)
            ST_0:
                if (X) nextState = ST_1;

            // ST_1-3 and default cases
        endcase
end
```

Hryg = 3'h001; //default

if (X) Hryg = 3'b010;

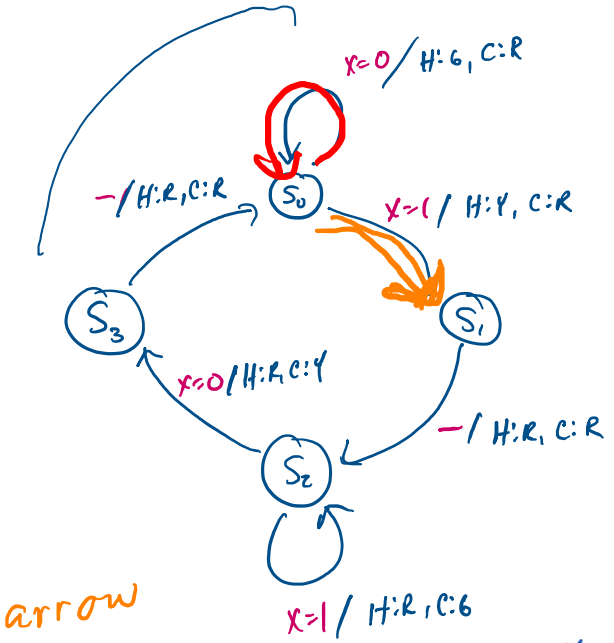
// other case covered by default



else Hryg = 3'b001; (optional)

State Machine to Verilog

```
always_comb begin
        nextState = state; //default
        Hryg = {0,0,1}; Cryg={1,0,0}; // default
        case(state)
            ST_0: begin
                if (X) begin
                    nextState = ST_1;
                    Hryg = {0,1,0}; // outputs
                    Cryg = {1,0,0); //optional
                end else begin // loop back case
                    nextState = ST_0; //optional
                    Hryg = {0,0,1}; //optional
                    Cryg = {1,0,0); //optional
                end
            end
            // ST_1-3 and default cases
        endcase
end
```



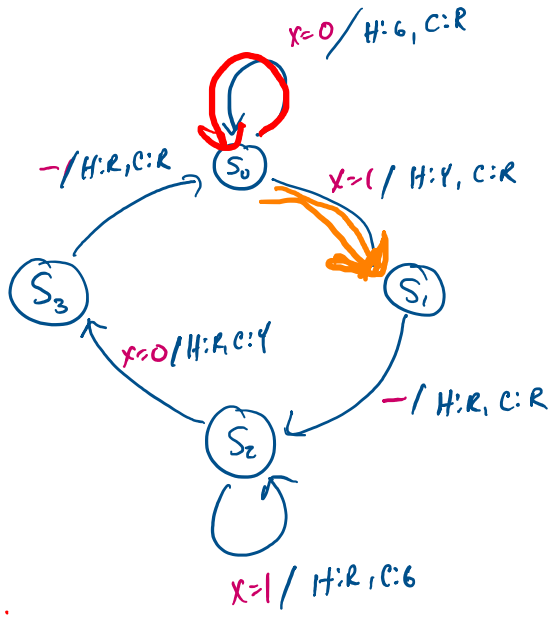
goto ST_1

arrow

highway = yellow

cntry rd = red

```
module Traffic (
    input clk,
    input rst,
    input x,
    output logic [2:0] Hrgg,
    output logic [2:0] Crgg
    );

enum {ST_0, ST_1, ST_2, ST_3} state, nextState;

always_ff @(posedge clk) begin
    if(rst)   state <= ST_0;
    else      state <= nextState;
end

always_comb
    nextState = state // default
    Hrgg = 3'b001; Crgg = 3'b100;
```




```
case (state)
    ST_0: if (x) begin
            nextState = ST_1;
            Hrgg = 3'b010;
            Crgg = 3'b100; //optional
          end //else optimal
    ST_1: nextState = ST_2;
          H: 3'b100; C: 3'b100;
```

~~ST_2~~

(next slide)

```
ST_2:  if (x) begin
           &nextState = ST_2; // optional
           Hrgg = 3'b100;  Crgg = 3'b001
       end else begin
           nextState = ST_3; // not optional
           Hrgg = 3'b100;  Crgg = 3'b010
       end

ST_3:  nextState = ST_0;
       Hrgg = 3'b100;  Crgg = 3'b100;

endcase

endmodule
```

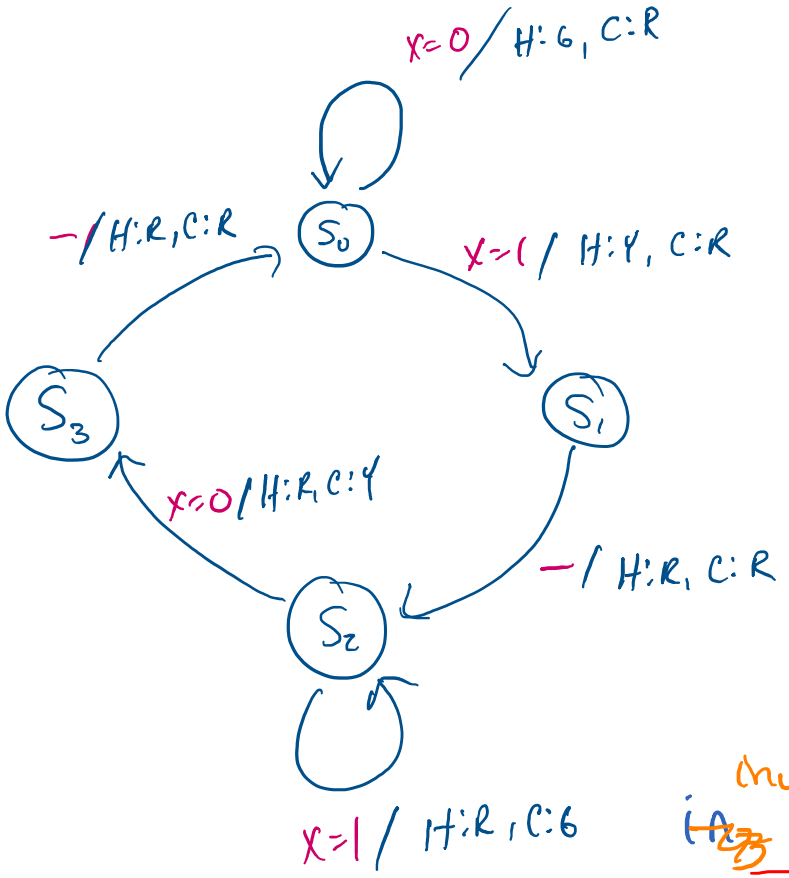
```
module traffic(
    input clk,
    input rst,
    input x,
    output logic [2:0] Hryg, //red-yellow-green
    output logic [2:0] Cryg  //red-yellow-green
    );

enum { ST_0, ST_1, ST_2, ST_3 } state, nextState;

always_ff @(posedge clk) begin
    if (rst) state <= ST_0;
    else     state <= nextState;
end


always_comb begin
    nextState = state; //default
    Hryg = 3'b001; Cryg = 3'b100;

    case (state)

        ST_0: begin
            if (x) begin
                nextState = ST_1;
                Hryg = 3'b010;
                Cryg = 3'b100; //opt
            end else begin //opt
                nextState = ST_0; //opt
                Hryg = 3'b001; //opt
                Cryg = 3'b100; //opt
            end
        end
```

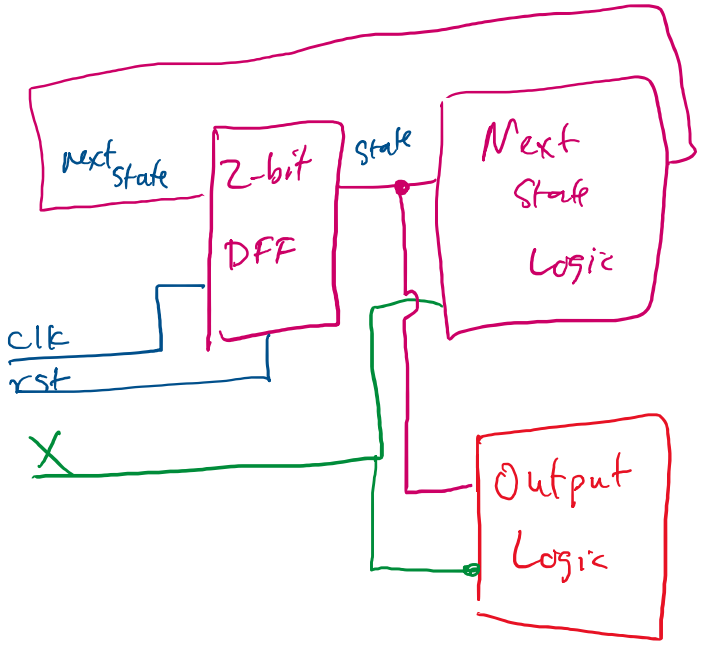
```
 ST_1: begin
            nextState = ST_2;
            Hryg = 3'b100;
            Cryg = 3'b100; //opt
        end

        ST_2: begin
            if (x) begin
                nextState = ST_2;
                Hryg = 3'b100;
                Cryg = 3'b001;
            end else begin
                nextState = ST_3;
                Hryg = 3'b100;
                Cryg = 3'b010;
            end
        end

        ST_3: begin
            nextState = ST_0;
            Hryg = 3'b100;
            Cryg = 3'b100; //opt
        end

    endcase
end

endmodule
```

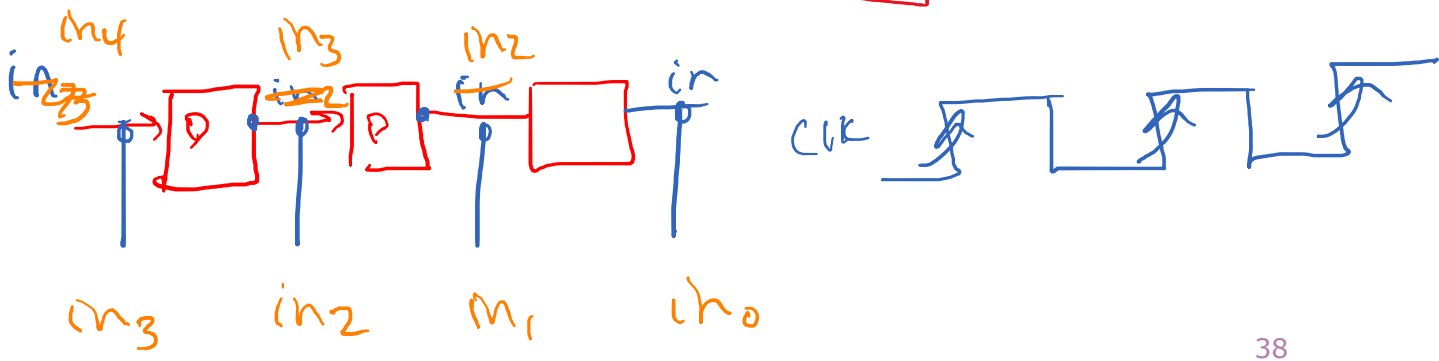
```
`timescale 1ns / 1ps

module traffic_tb();

logic clk;
logic rst;
logic x;
wire [2:0] Hryg;
wire [2:0] Cryg;

traffic t0( .clk, .rst, .x, .Hryg, .Cryg);

always #10 clk = ~clk;

initial begin
    clk = 0; rst = 1; x=0;
    @(negedge clk);
    @(negedge clk);
    rst = 0;

    @(negedge clk);
    @(negedge clk);

    x = 1;
    @(negedge clk);
    @(negedge clk);
    @(negedge clk);

    x = 0;
    @(negedge clk);
    @(negedge clk);
    @(negedge clk);

    $finish;
end
endmodule
```

Stopped here?

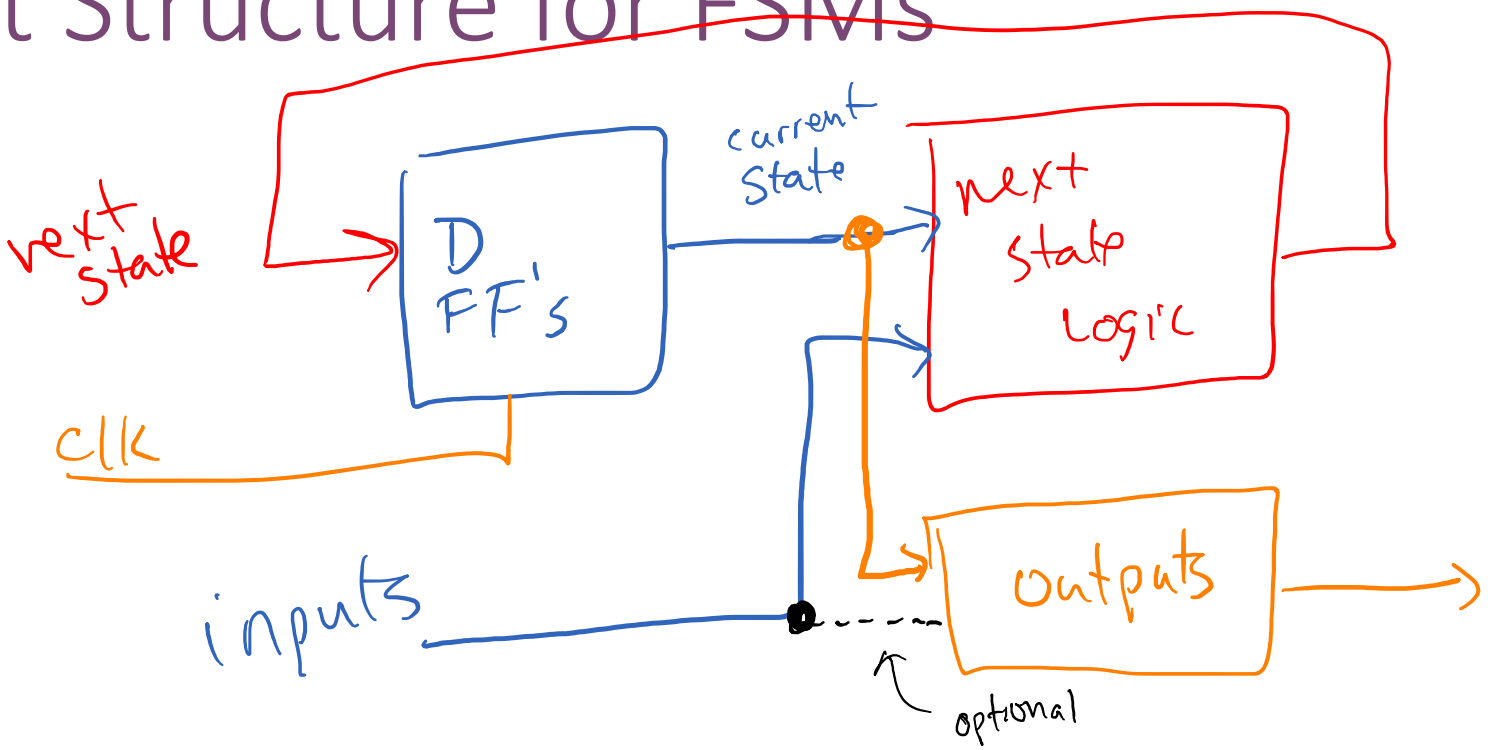
State Machine in Logic







Circuit Structure for FSMs



Moore Machine: outputs are a function of current state

Mealy Machine: outputs are a function of current state + inputs

Your Turn

- Build a digital safe / keypad lock
- The user must enter the digits `5 – 4 – 3` in that order to unlock the door. Any other inputs result in a locked door.
- Once unlocked, the door remains unlocked until `E` key pressed. Ignore all other keys while unlocked.

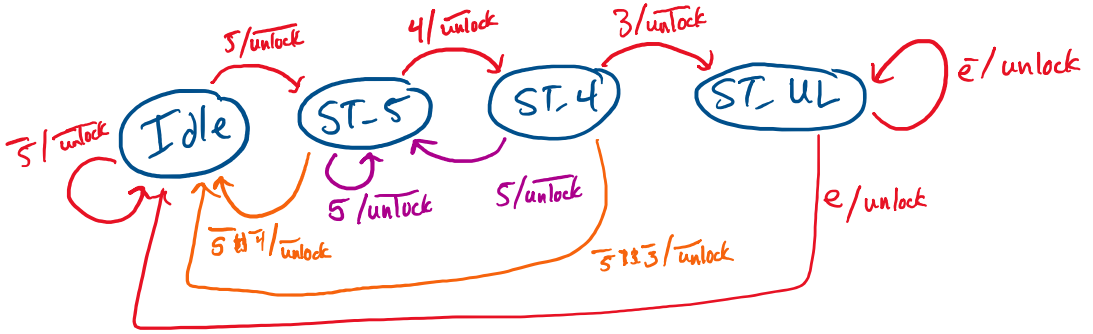
Should unlock: 5-5-4-3, 5-4-5-4-3

- Draw the state machine!

Lock State Machine

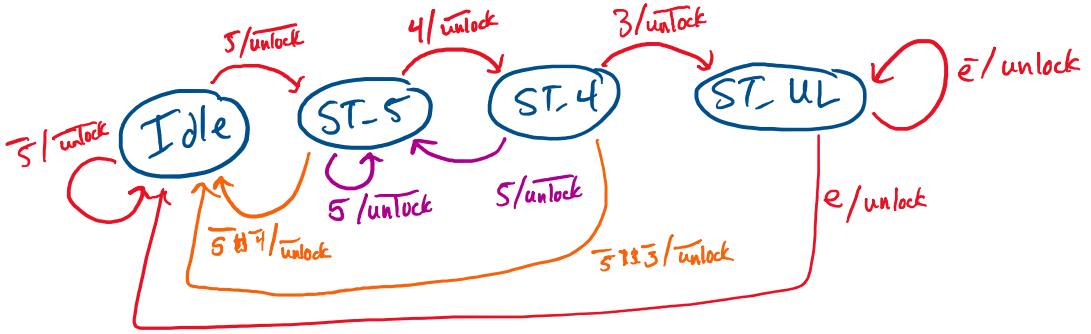
- `Recall: 5 - 4 - 3`
- `E: relock`

State Machine in Verilog




```
module Lock(
     input clk, rst,
     input [9:0] num,
     input e, //relock
     output unlock
);
```

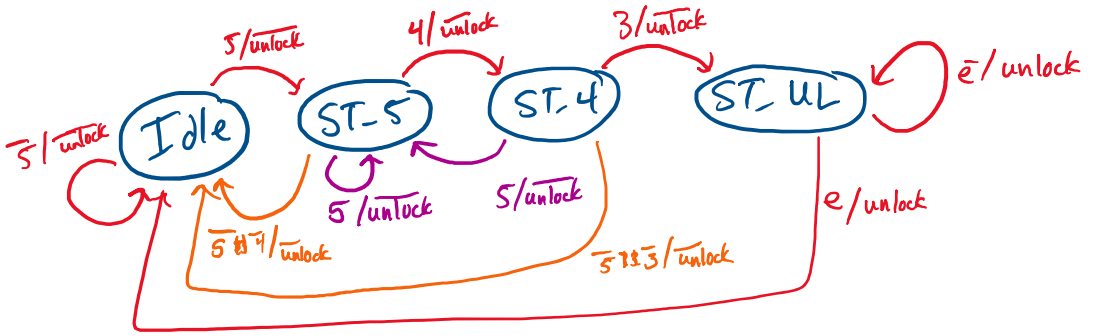
State Machine in Verilog



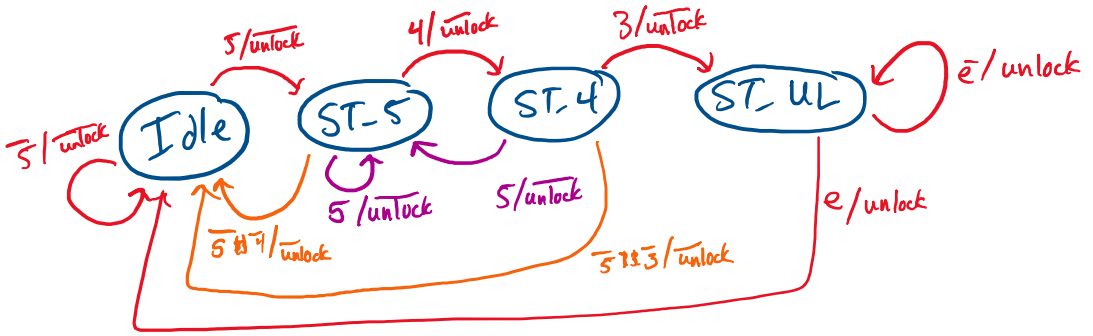

```
module Lock(
    input clk, rst,
    input [9:0] num,
    input e, //relock
    output unlock
);

enum {ST_IDLE, ST_5, ST_4, ST_UL } state, next_state;

//seq logic
always_ff @(posedge clk) begin
       if (rst) state <= ST_IDLE;
       else     state <= next_state;
end
```

State Machine in Verilog

```
//comb logic block
always_comb begin




end
```



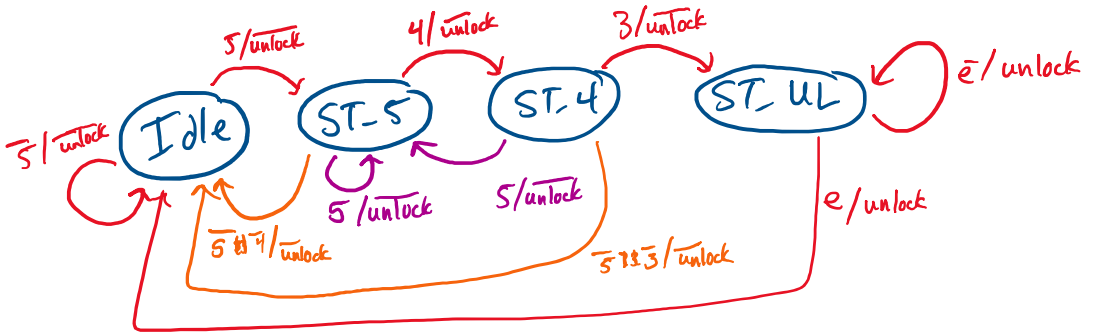
State Machine in Verilog




```
//comb logic block
always_comb begin
        next_state = state; //default
        unlock = 1'h0; //default
        case (state)
                ST_IDLE:
                        if (num[5])   next_state = ST_5;
                ST_5:
                        if (num[4])   next_state = ST_4;
                ST_4:
                        if (num[3])   next_state = ST_UL;
                ST_UL: if (e)         next_state = ST_IDLE;
        endcase
end
```

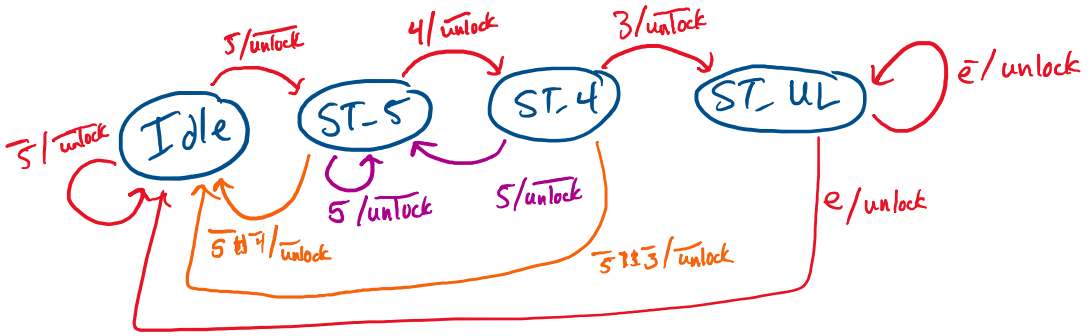
State Machine in Verilog

```
case (state)
  ST_IDLE:
    if (num[5])
      next_state = ST_5;
  ST_5: begin
    if (num[4])
      next_state = ST_4;




  end
 ST_4: begin
    if (num[3])
      next_state = ST_UL;
```




```
  end
  ST_UL: begin


    if (e)
      next_state = ST_IDLE;
  end
endcase
```

State Machine in Verilog




```
case (state)
  ST_IDLE:
    if (num[5])
      next_state = ST_5;
  ST_5: begin
    if (num[4])
      next_state = ST_4;
    else if (num[5])
      next_state = ST_5;
    else if ( (|num) | e ) //other btns
      next_state = ST_IDLE;
  end
 ST_4: begin
    if (num[3])
      next_state = ST_UL;
    else if (num[5])
      next_state = ST_5;
    else if ( (|num) | e) // other btns
      next_state = ST_IDLE;
  end
  ST_UL: begin
    unlock = 1'h1;
    if (e)
      next_state = ST_IDLE;
  end
endcase
```

Next Time

- Memory