

ENGR 210 / CSCI B441
“Digital Design”

Finite State Machines I

Andrew Lukefahr

Announcements

- P8 – Elevator Controller is out
 - This one is hard.
- P9 – SPI is out
 - This one is new. Might be some changes.

Always specify
defaults for
always_comb!

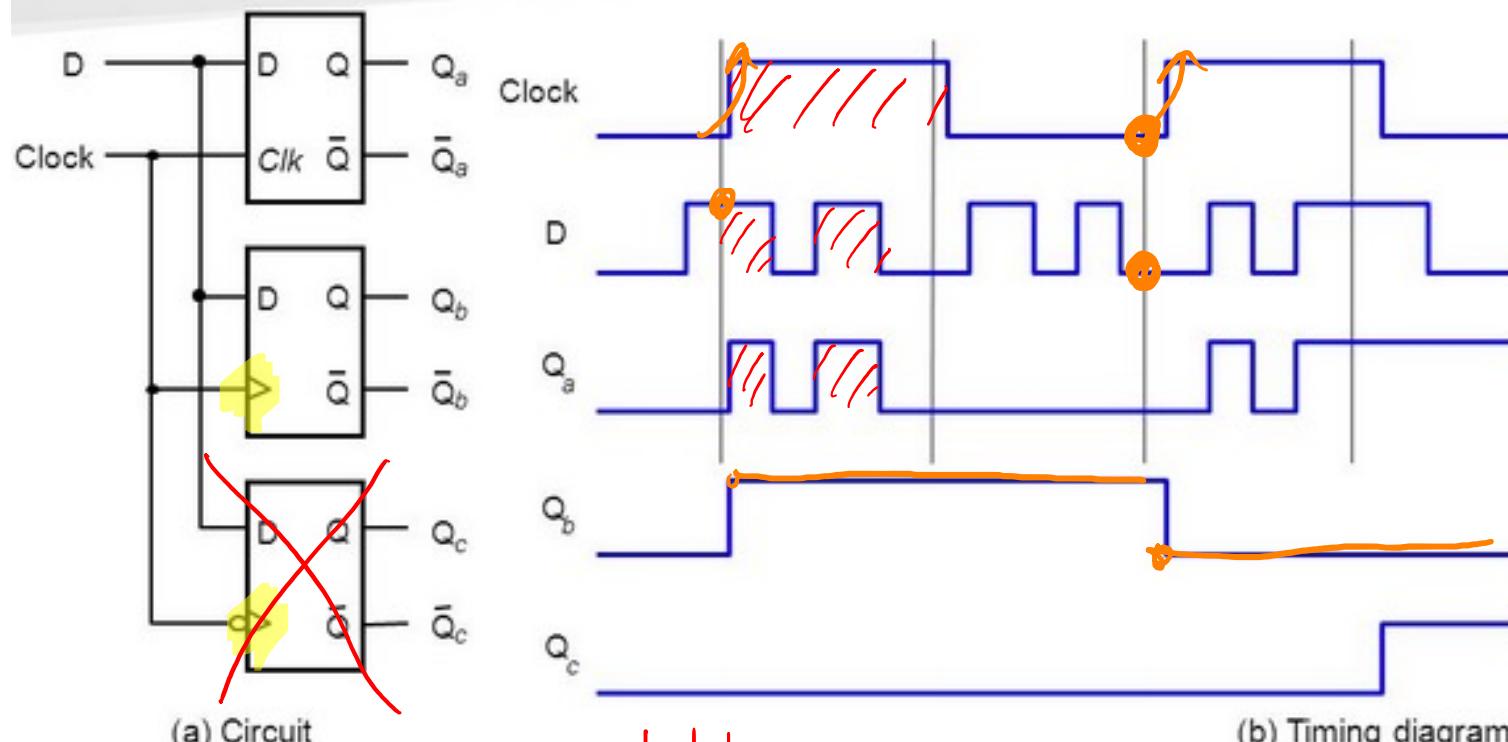
BLOCKING (=) FOR

always_comb

NON-BLOCKING (<=) for

always_ff

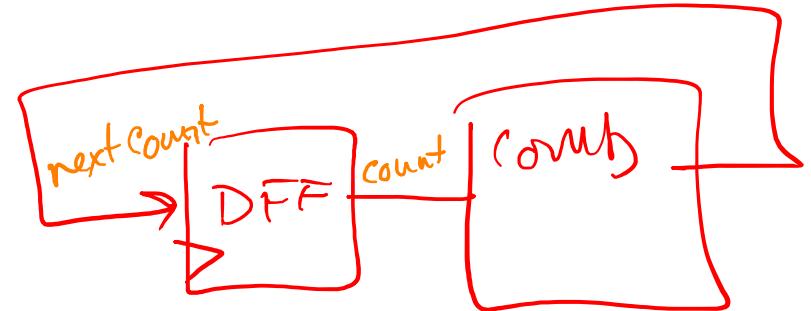
D Latch versus D Flip-Flop



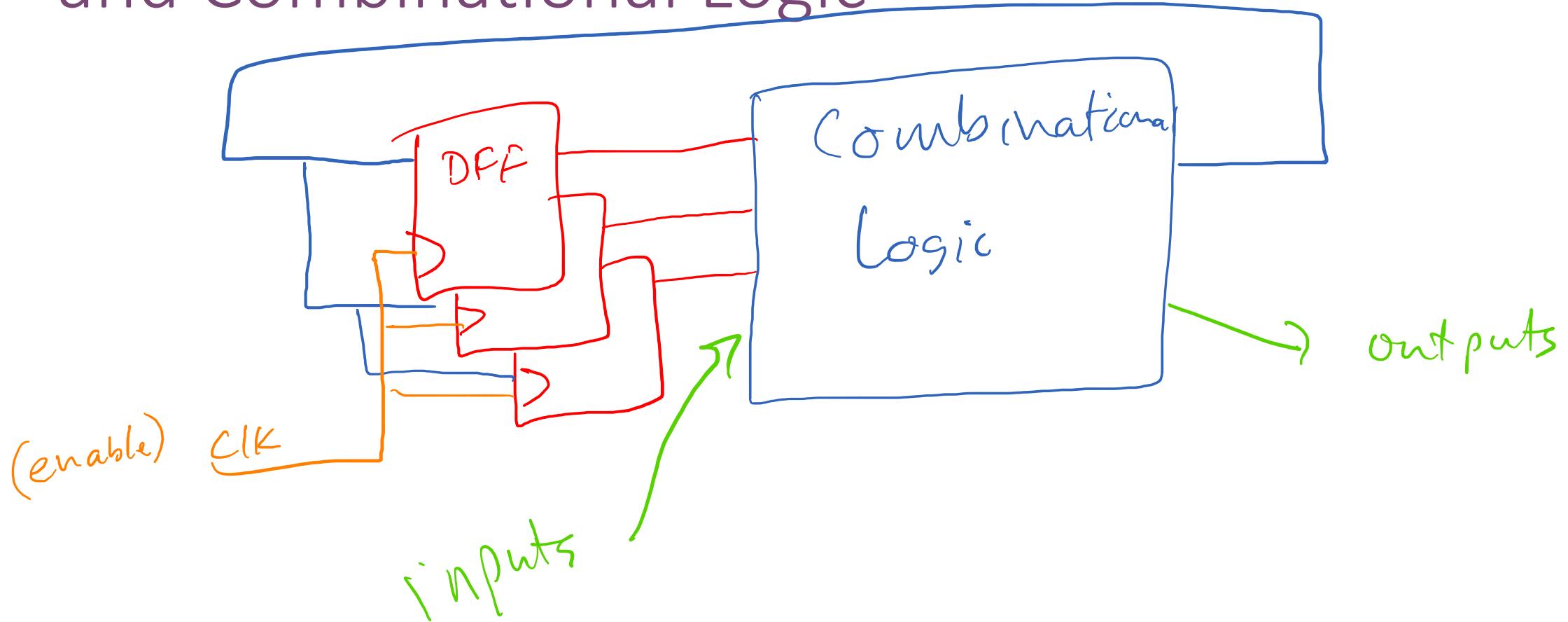
Comparison of level-sensitive and edge-triggered devices

Thing and NextThing

```
logic [1:0] thing, nextThing;  
  
always_ff @(posedge clk) begin  
    if (rst) thing <= 2'h0;  
    else      thing <= nextThing;  
end  
  
always_comb begin  
    nextThing = thing; //default  
  
    if (foo) begin  
        nextThing = thing + 1;  
    end else if (bar) begin  
        nextThing = thing - 1;  
    end  
end
```

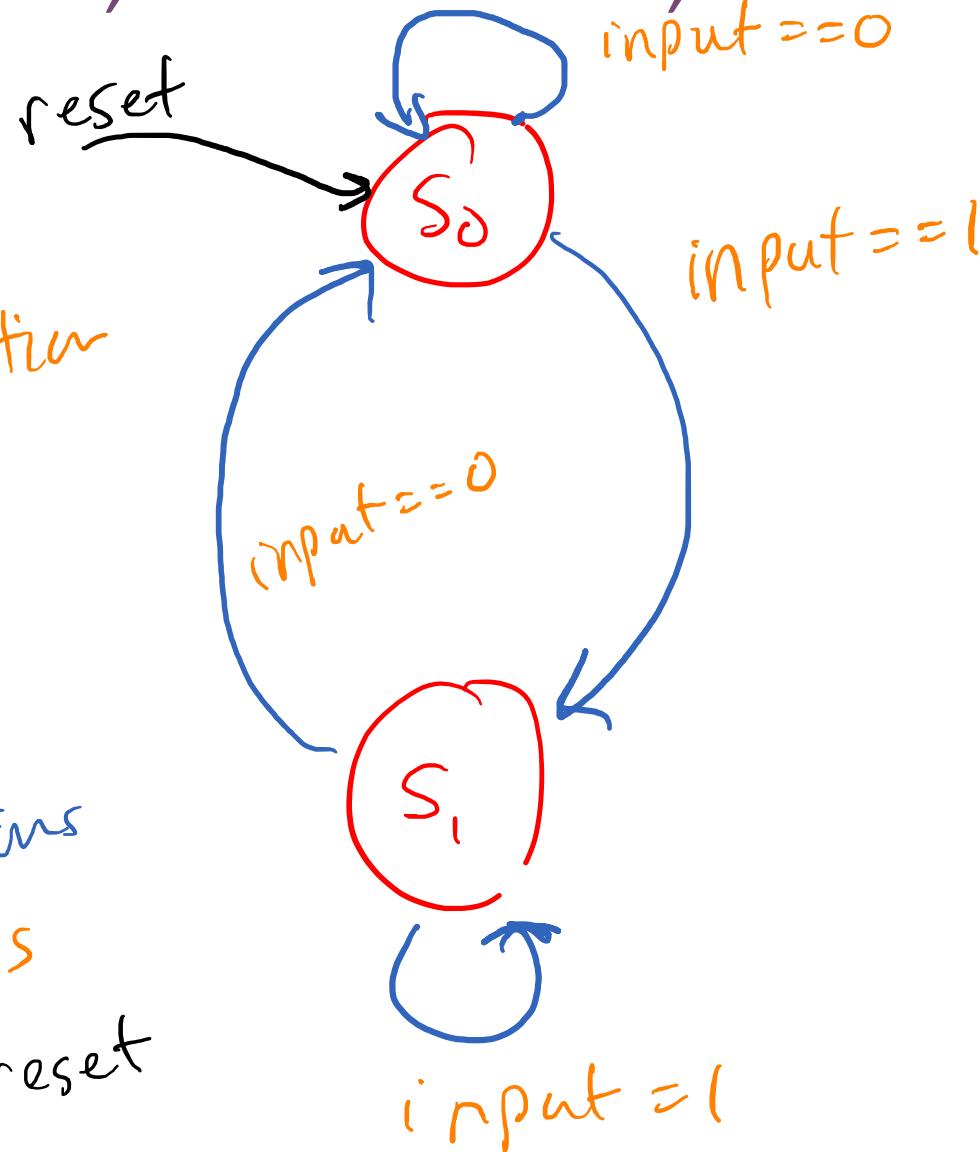


Sequential Logic uses both Flip-Flops and Combinational Logic



Review: States, Transitions, and Guards

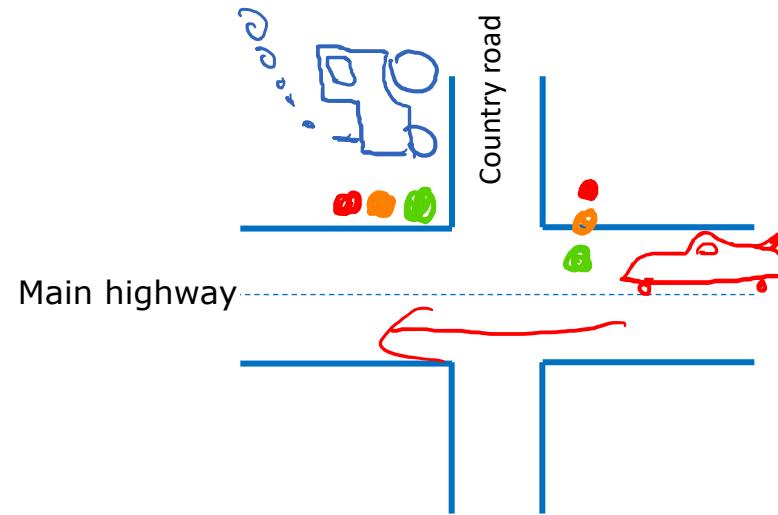
~~S₀ + S₁ = states~~
~~lines = transitions~~
booleans = guards
reset = reset



S_0 & S_1 are states
two states in this machine
guards: control which transition you take
transitions
→ leaving one state & going to another
(happen @posede clk)

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

~~tractors~~

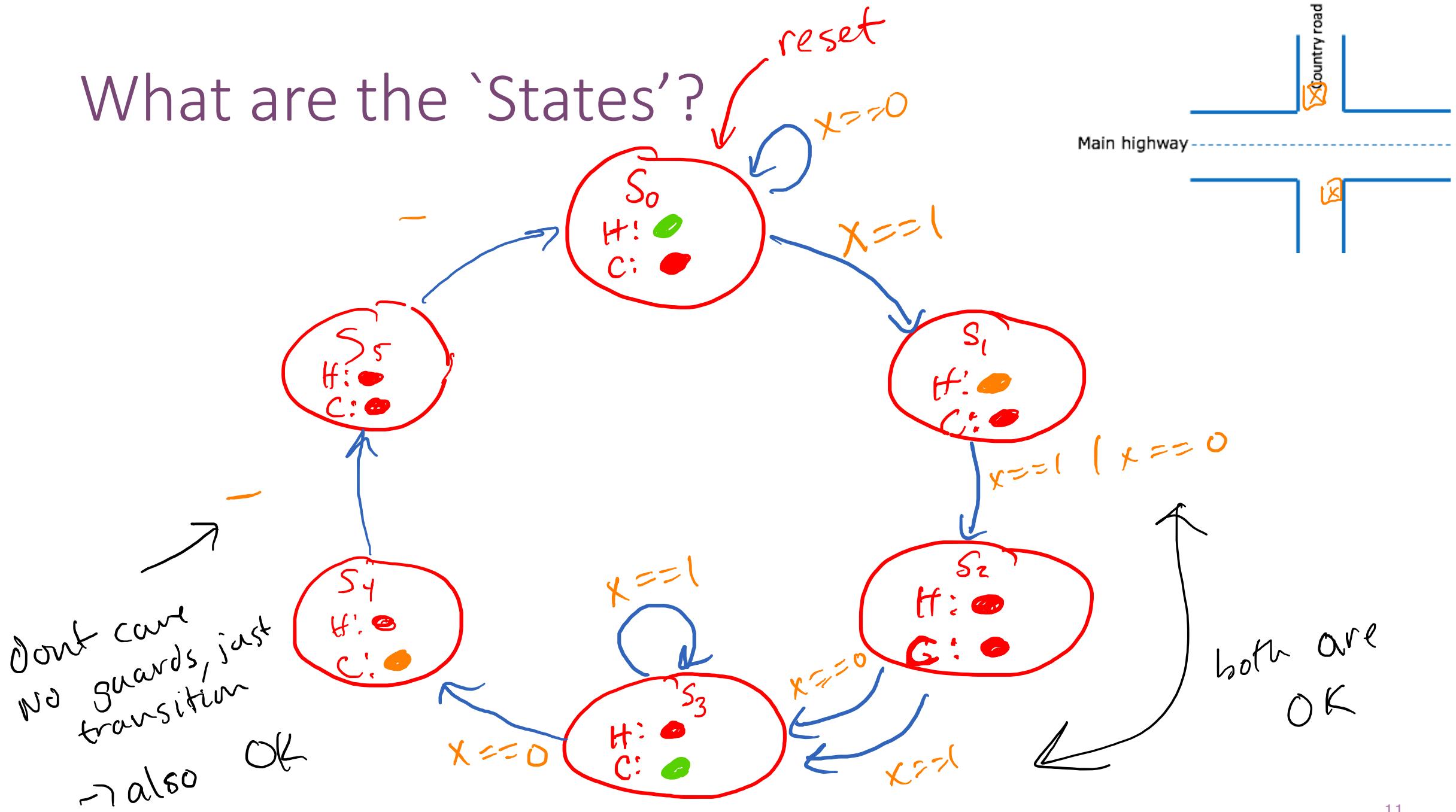
- ~~Cars~~ occasionally arrive from the country road. The traffic signal for the country road must turn **green** only long enough to let the cars on the country road go.
- When no cars are waiting on the country road, the country road traffic signal turns **yellow** then **red** and the traffic signal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

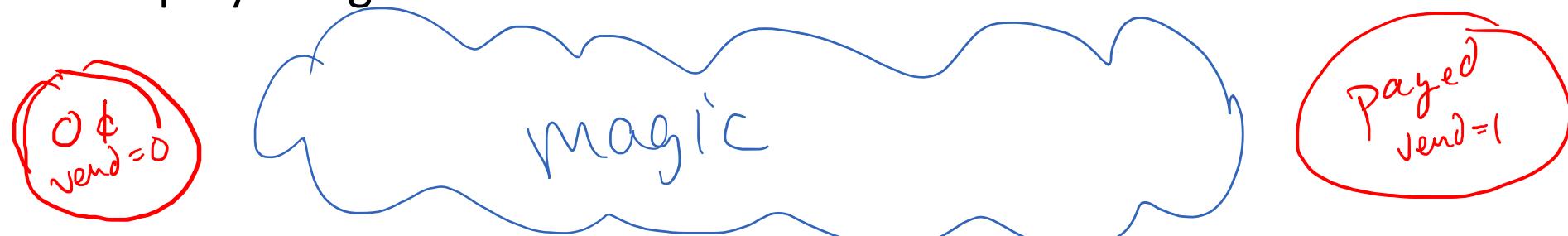
What are the 'States'?



skip till next time

FSM: Simple Vending Machine

- You are designing a Vending Machine that dispenses Widgets for \$0.25/each.
- Your machine must accept any combination of nickels (N), dimes (D), and quarters (Q) to pay for the Widget.
- When the correct payment is secured, you dispense the Widget (`vend`), and reset the payment.
- If a customer overpays, you keep the extra money. ☺
 - Just to simplify things...

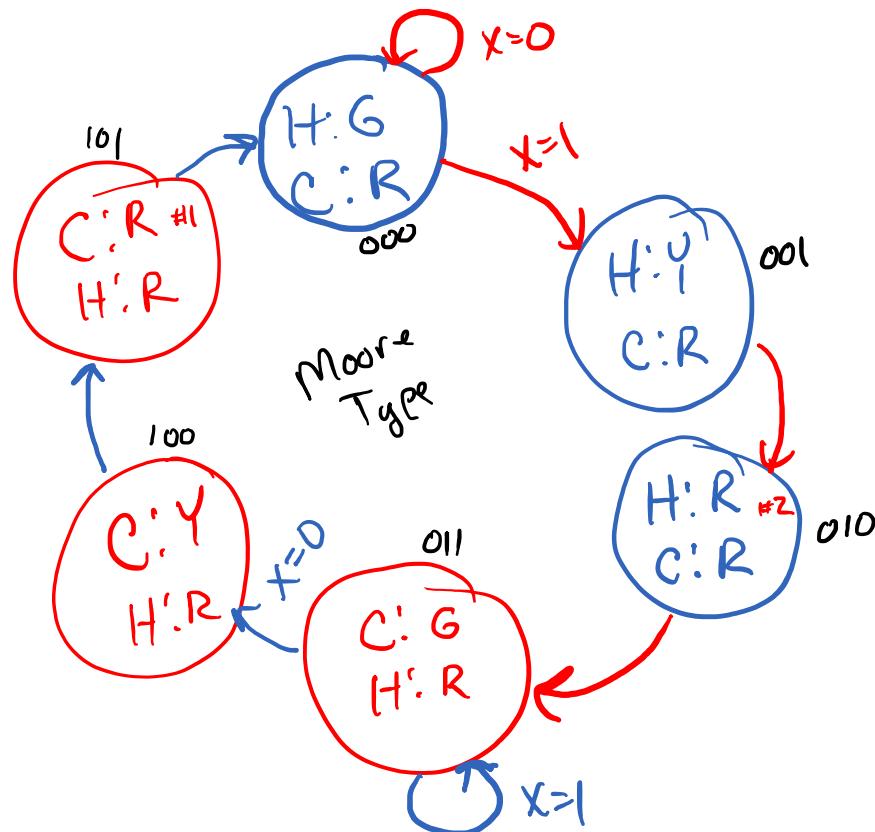
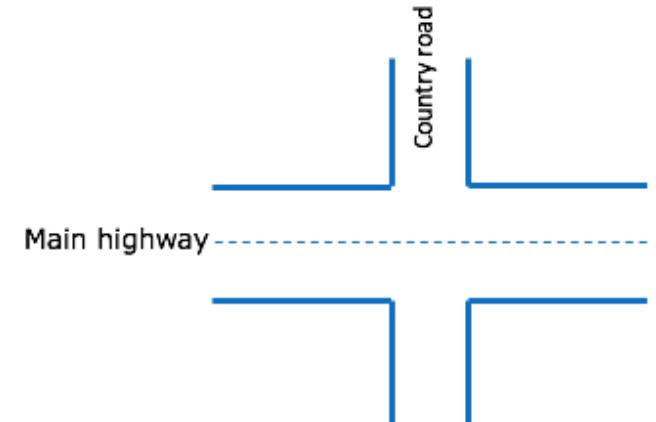


FSM: Vending Machine

Moore vs. Mealy Type FSMs

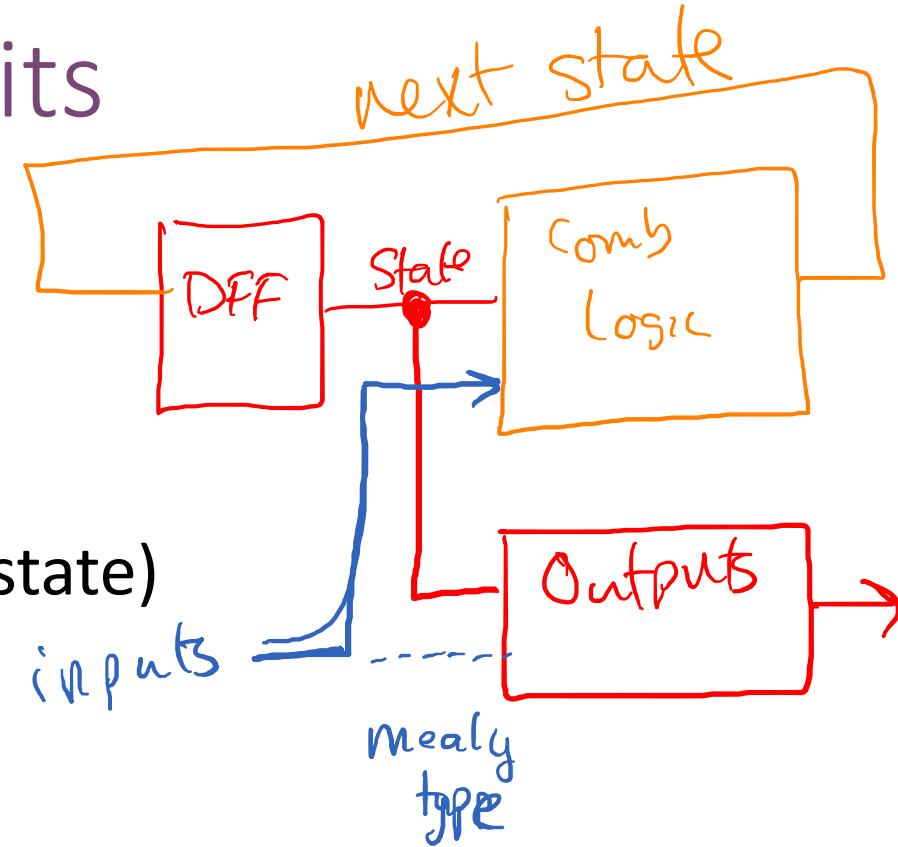
- Thus far we've done "Moore" Type
 - Moore Type: Outputs determined by the state (circle)
- Another technique: "Mealy" Type
 - Mealy Type: Output determined by the transition (arrow)
- Moore: Easier, but more states
- Mealy: Less states, more complicated transitions

Traffic Light: Moore vs. Mealy

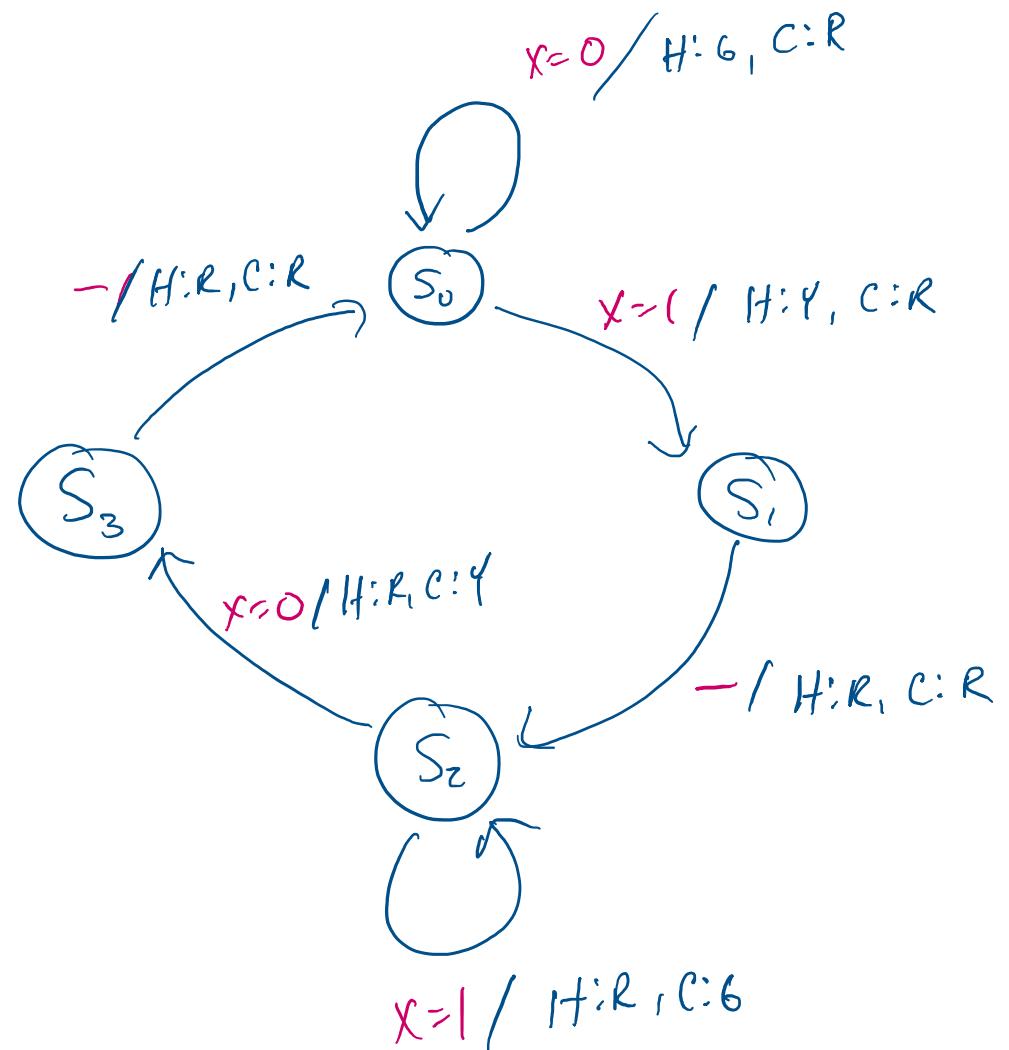


Implementing FSMs with Circuits

- 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



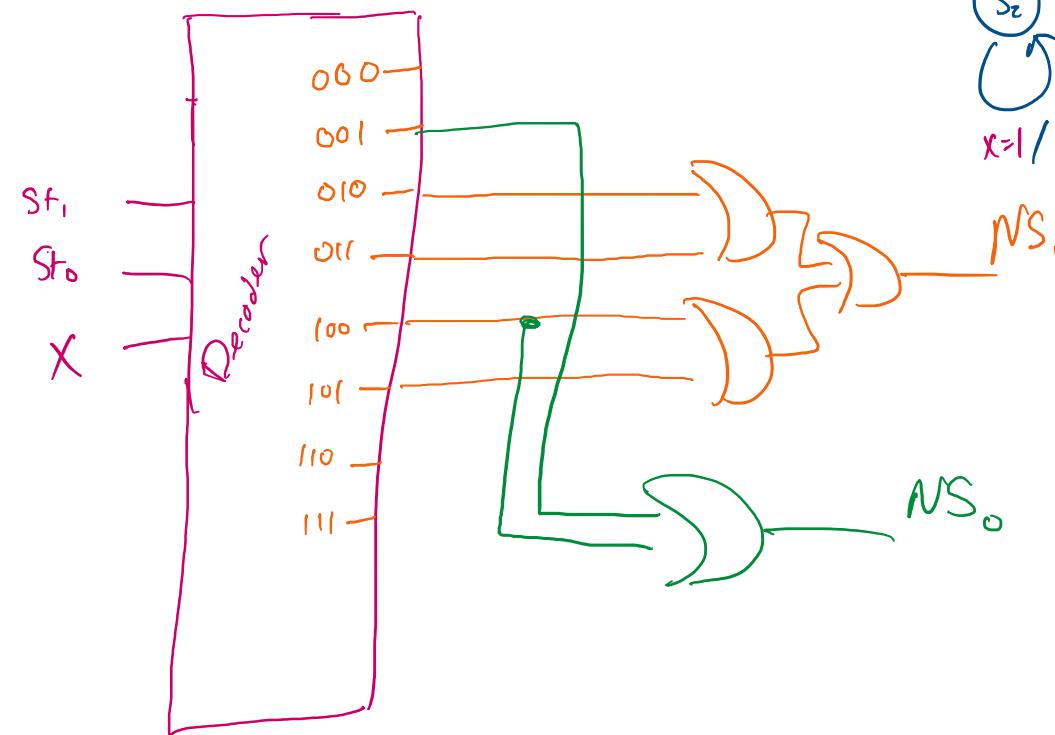
State Machine Encoding

	<u>State</u>	<u>X</u>	<u>Next State</u>
0	00	0	00
1	00	1	01
2	01	0	10
3	01	1	10
4	10	0	11
5	10	1	10
6	11	0	00
7	11	1	00

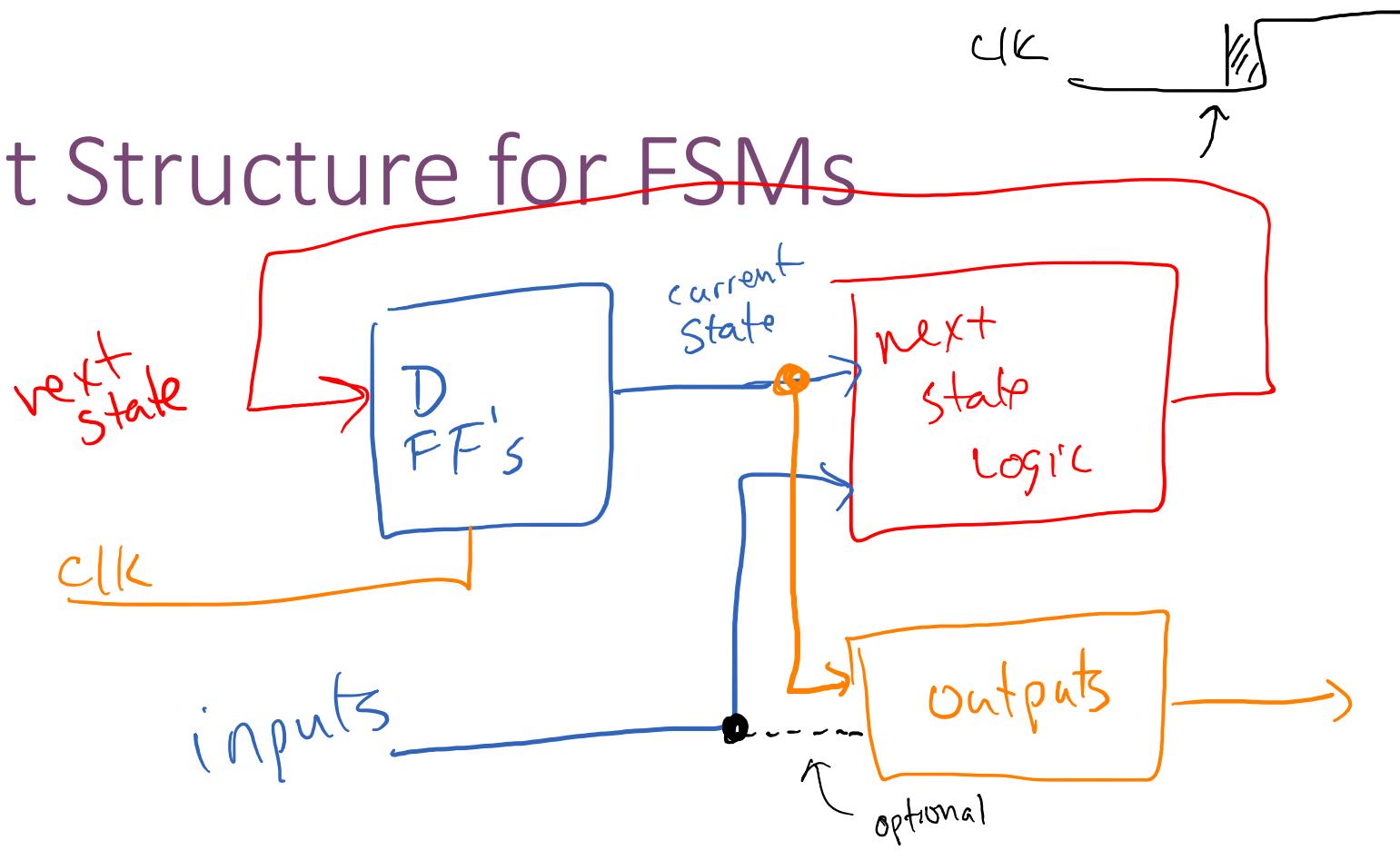
Next State Logic

	<u>State</u>	<u>X</u>	<u>Next State</u>
0	00	0	00
1	00	1	01
2	01	0	10
3	01	1	10
4	10	0	11
5	10	1	10
6	11	0	00
7	11	1	00

↑↑ NS₁ NS₀



Circuit Structure for FSMs

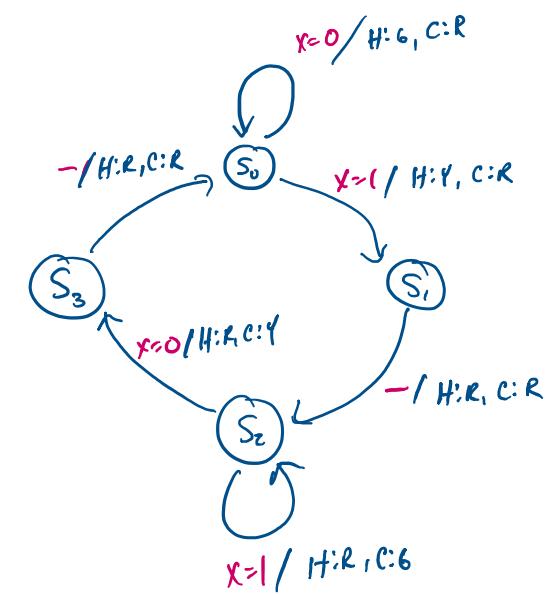
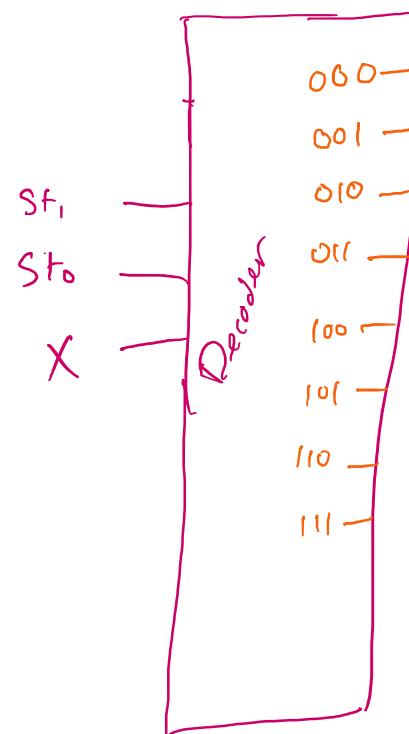


Moore Machine: outputs are a function of current state

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

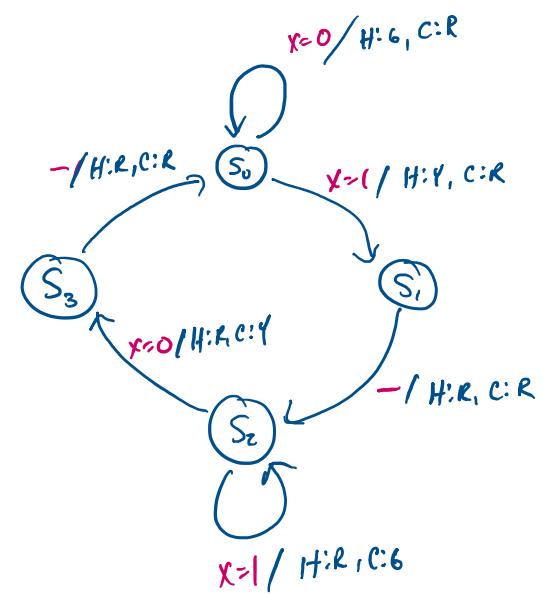
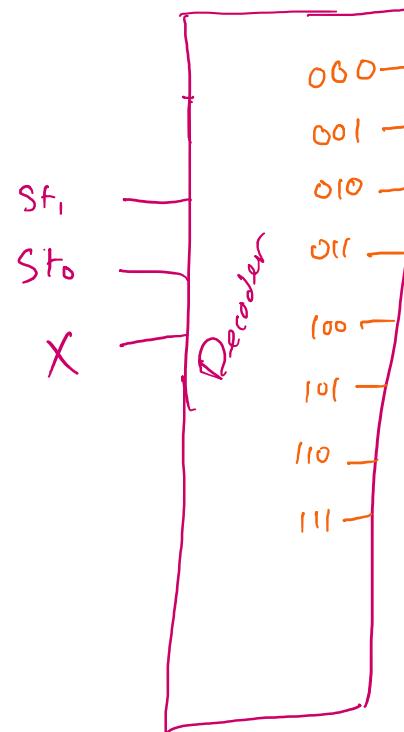
Output Logic (Highway)

	<u>State</u>	<u>X</u>
0	00	0
1	00	1
2	01	0
3	01	1
4	10	0
5	10	1
6	11	0
7	11	1

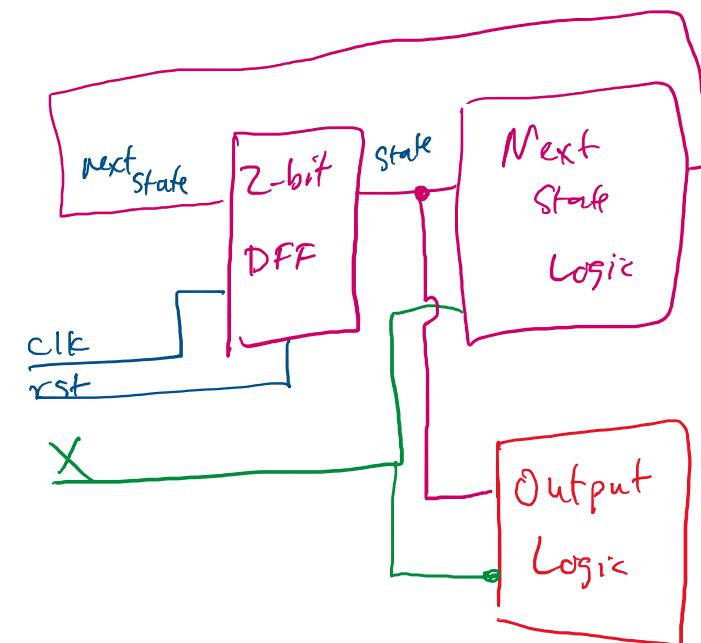
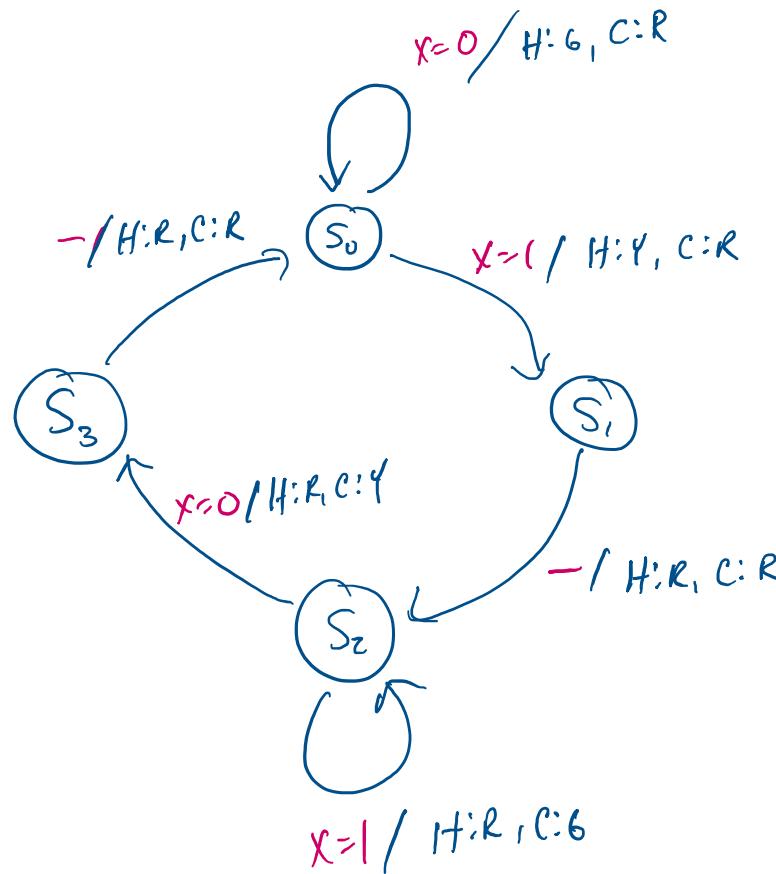


Output Logic (Country Rd)

	<u>State</u>	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

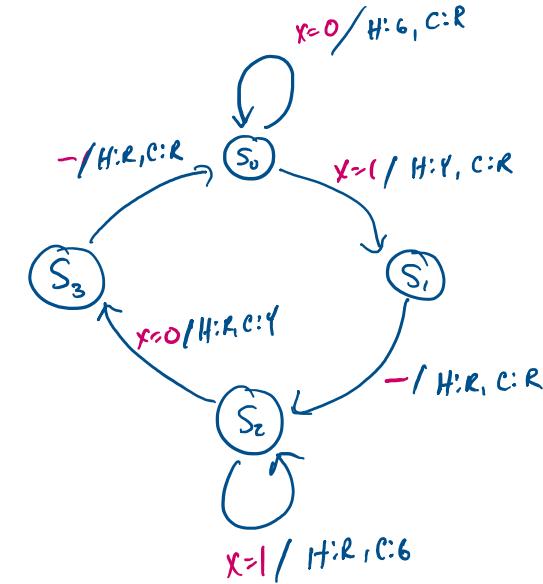


State Machine to Logic



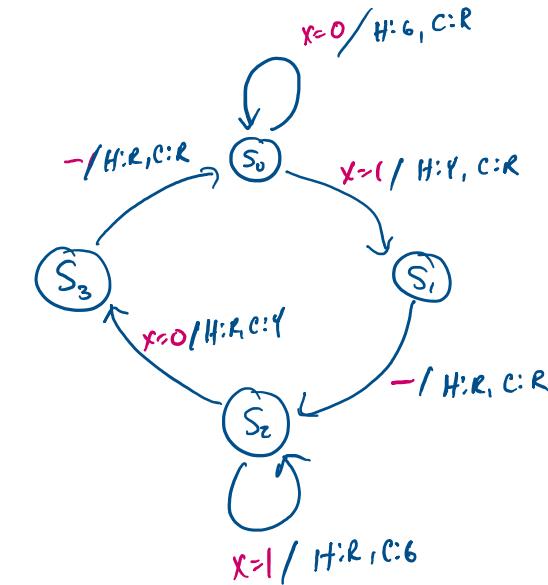
State Machine to Verilog

- Define states?



State Machine to Verilog

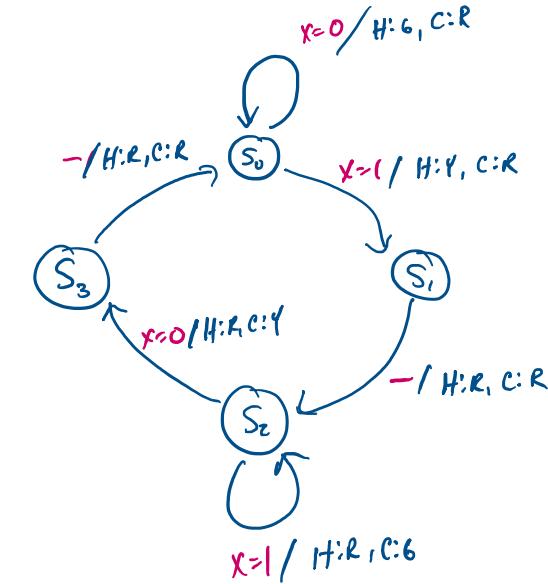
- Define states?



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

State Machine to Verilog

- Build State Machine?

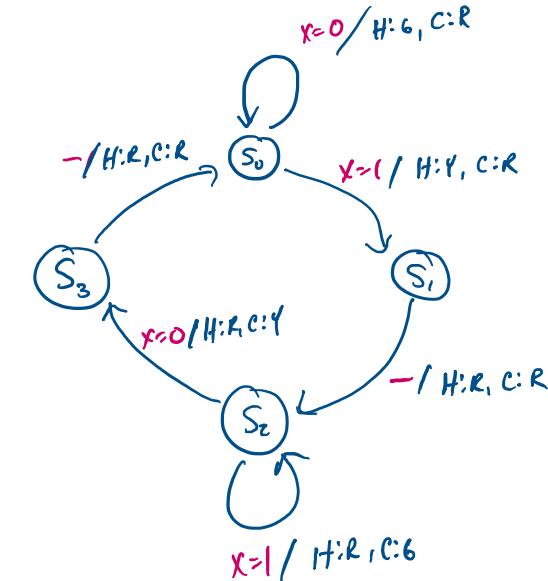


State Machine to Verilog

- Build State Machine?

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

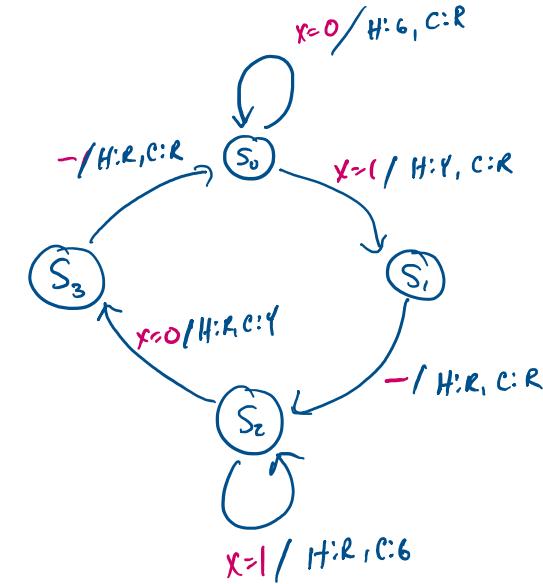
- What is nextState?



State Machine to Verilog

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

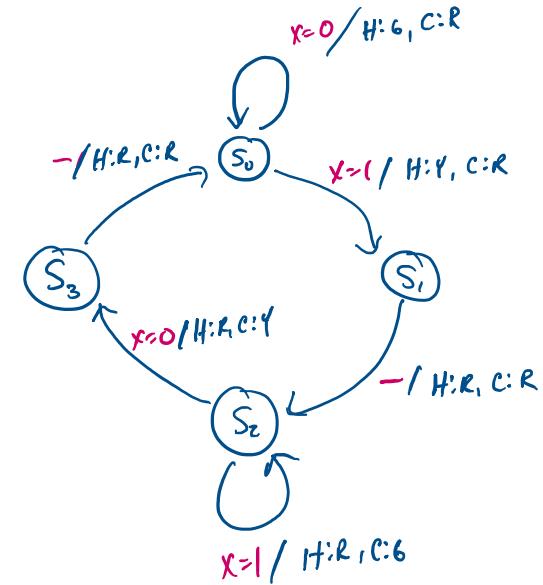
- What is nextState?



State Machine to Verilog

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

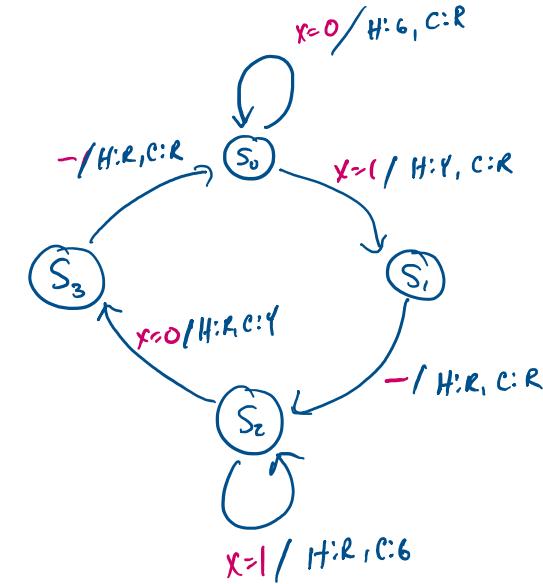
always_comb begin
    nextState = state; //default
    case(state)
        ST_0: nextState = ST_1; //goto state 1
        ST_1: nextState = ST_2;
        ST_2: nextState = ST_3;
        ST_3: nextState = ST_0; //loop
        default: nextState = ST_0; //just in case
    endcase
end
```



State Machine to Verilog

- What is this missing?

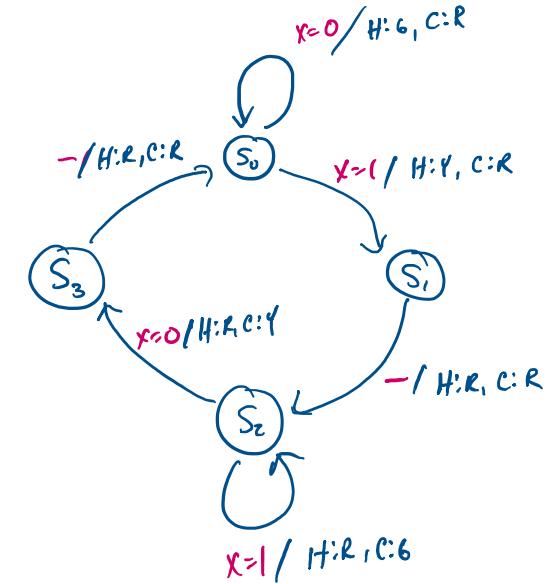
```
always_comb begin
    nextState = state; //default
    case(state)
        ST_0: nextState = ST_1; //goto state 1
        ST_1: nextState = ST_2;
        ST_2: nextState = ST_3;
        ST_3: nextState = ST_0; //loop
        default: nextState = ST_0; //just in case
    endcase
end
```



State Machine to Verilog

- What is this missing?

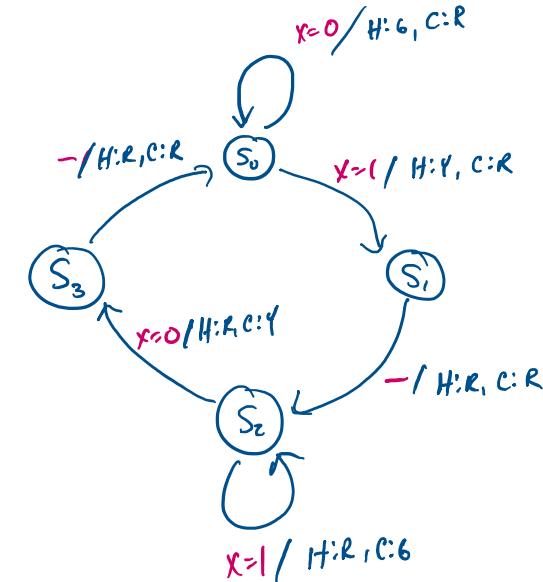
```
always_comb begin
    nextState = state; //default
    case(state)
        ST_0:
            nextState = ST_1;
        ST_1:
            nextState = ST_2;
        ST_2:
            nextState = ST_3;
        // ST_3 and default cases      endcase
    end
```



State Machine to Verilog

- What is this missing?

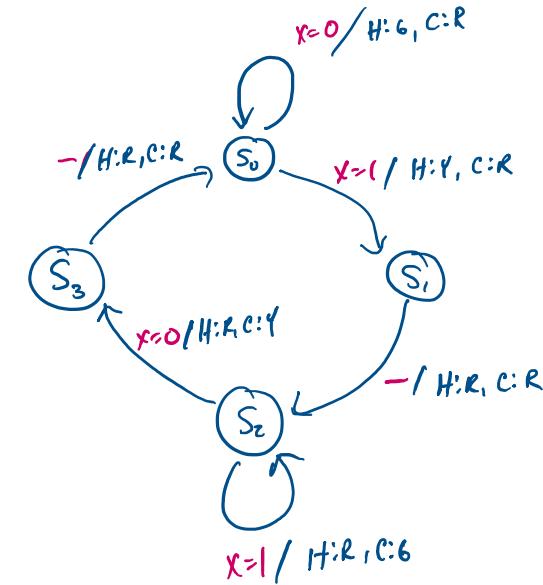
```
always_comb begin
    nextState = state; //default
    case(state)
        ST_0:
            if (X) nextState = ST_1;
        ST_1:
            nextState = ST_2;
        ST_2:
            if (~X) nextState = ST_3;
        // ST_3 and default cases    endcase
    end
```



State Machine to Verilog

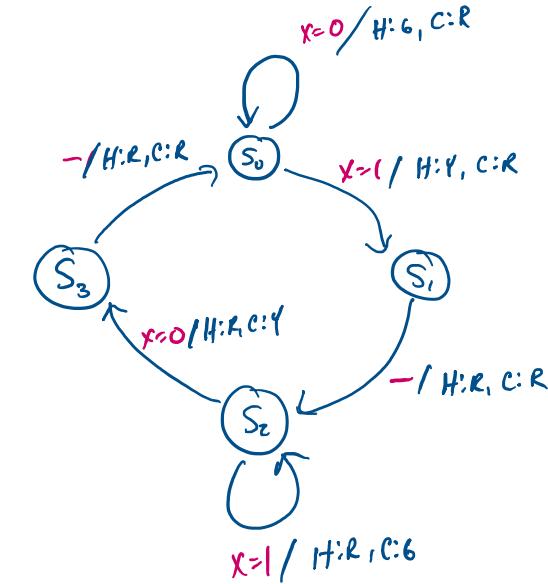
- What else is this missing?

```
always_comb begin
    nextState = state; //default
    case(state)
        ST_0:
            if (X) nextState = ST_1;
            // ST_1-3 and default cases
    endcase
end
```



State Machine to Verilog

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



```

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
    end

```

```

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

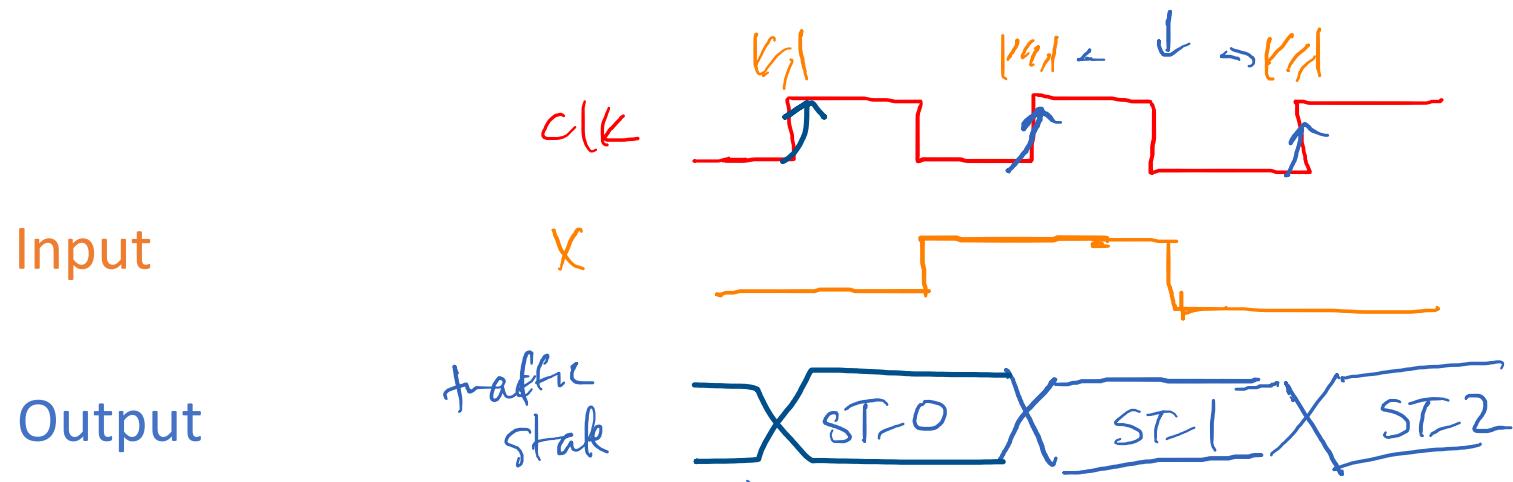
        ST_2: begin
            if (x) begin
                nextState = ST_2;
                Hryg = 3'b100;
                Cryg = 3'b001;
            end else begin
                nextState = ST_3;
                Hryg = 3'b100;
                Cryg = 3'b010;
            end
        end

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

endmodule

```

Testbenches for State Machines



- Outputs change @posedge clk
- Change Inputs @negedge clk

```

`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; //auto-update clock

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

```

```

`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); ↗ 2 cycles
    rst = 0; ↗ reset off

```

testbench: negedge
FPGA code: posedge

```

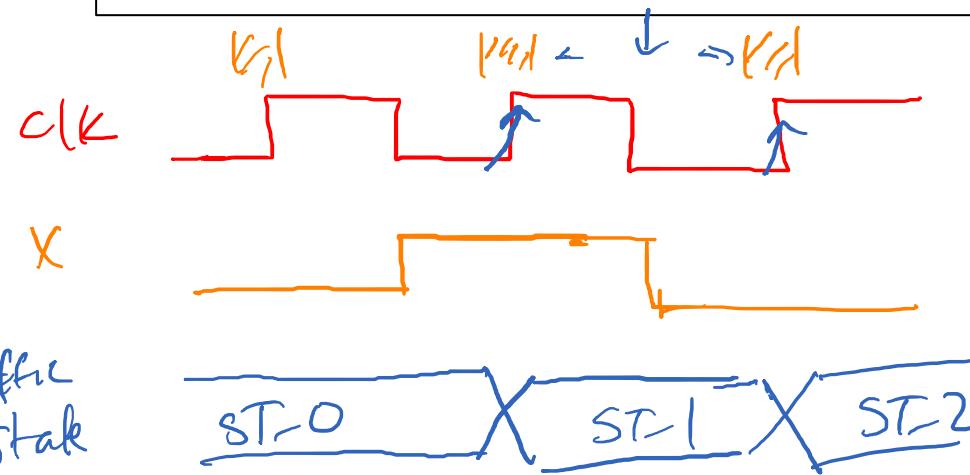
@ (negedge clk);
@ (negedge clk);

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

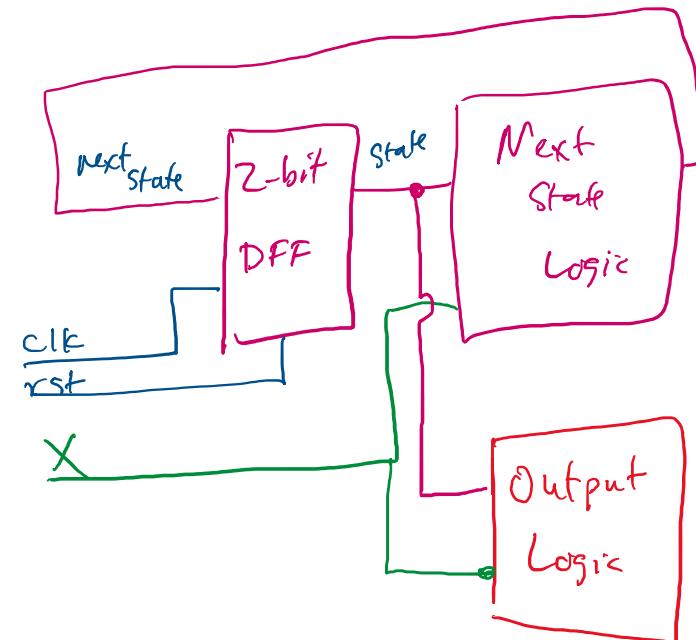
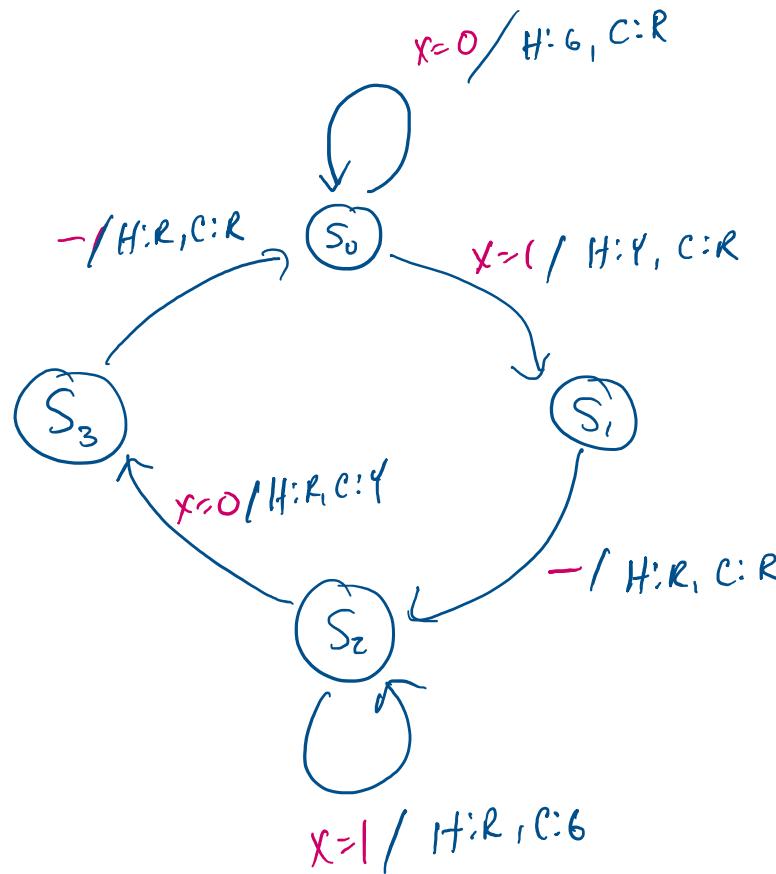
x = 0; ↗
@ (negedge clk);
@ (negedge clk);
@ (negedge clk);

$finish;
end
endmodule

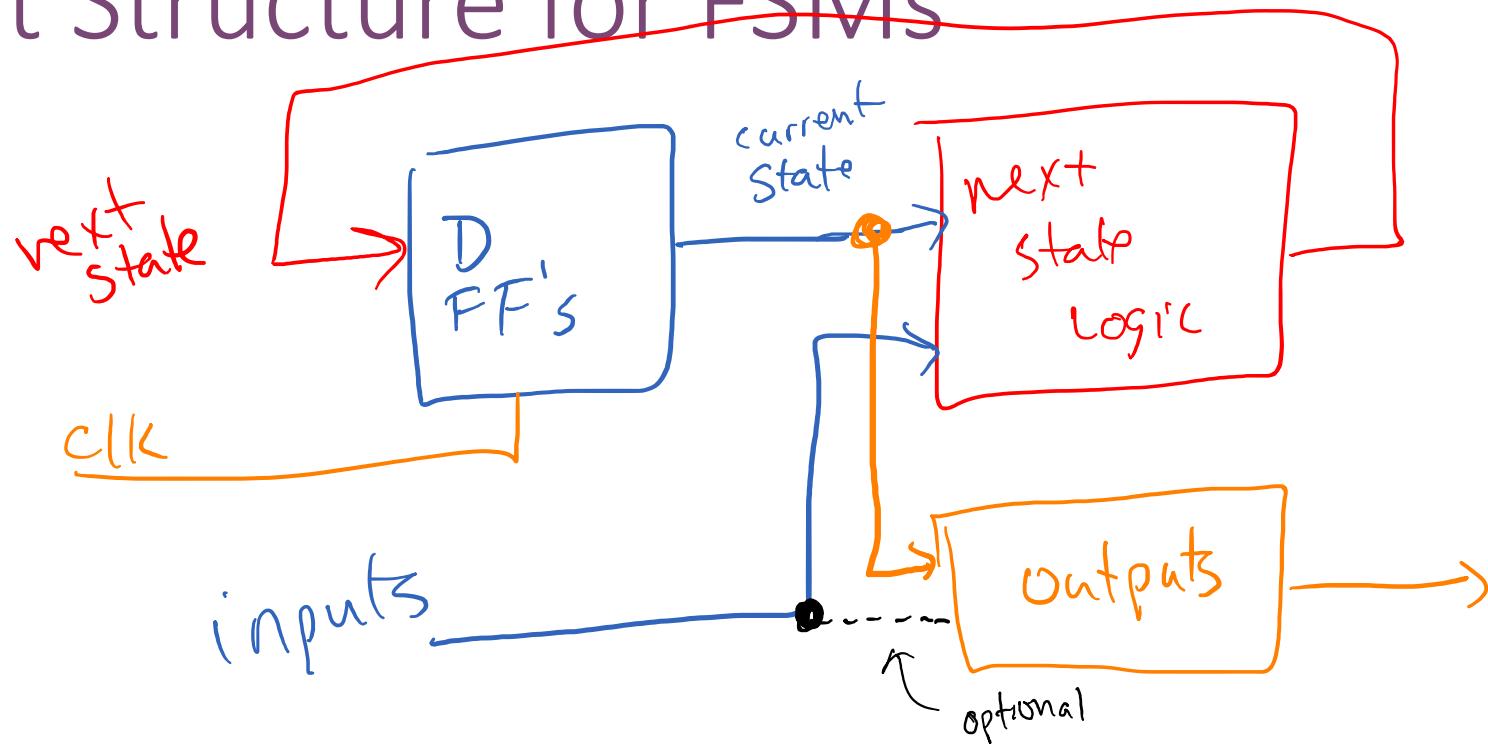
```



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!

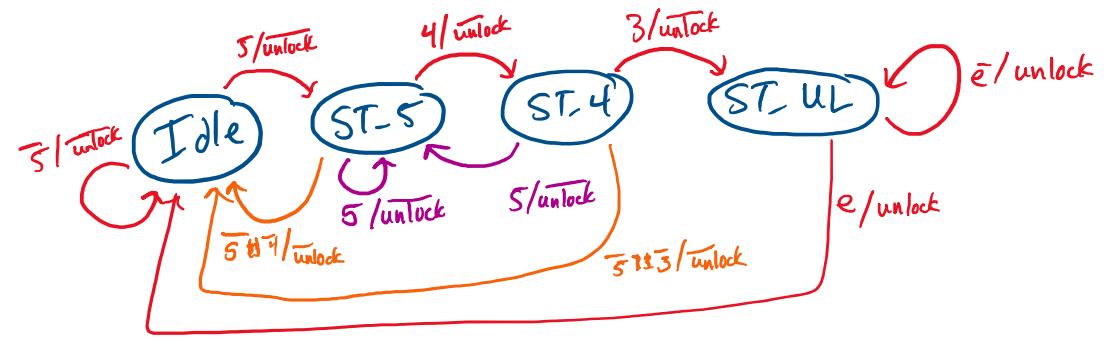
Lock State Machine

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



State Machine in Verilog

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

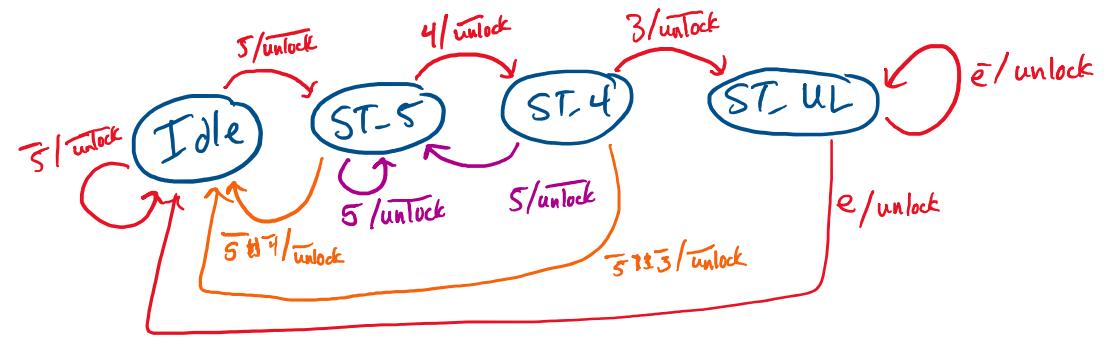


State Machine in Verilog

```
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;

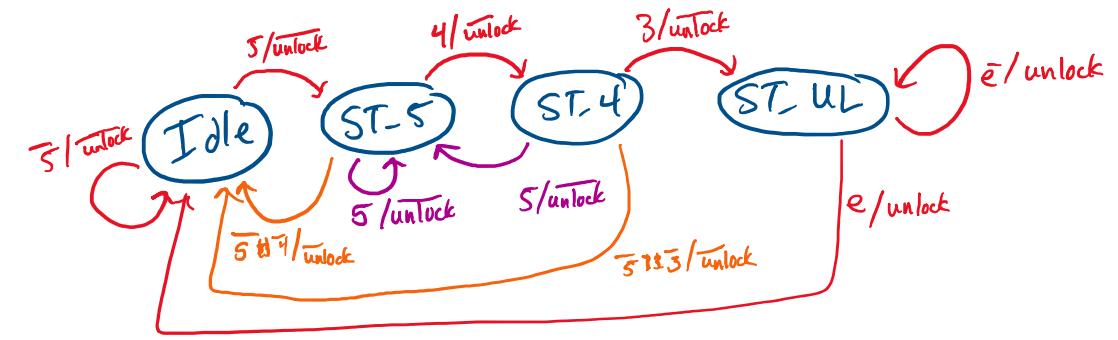
//seq logic
always_ff @(posedge clk) begin
    if (rst) state <= ST_IDLE;
    else      state <= next_state;
end
```



State Machine in Verilog

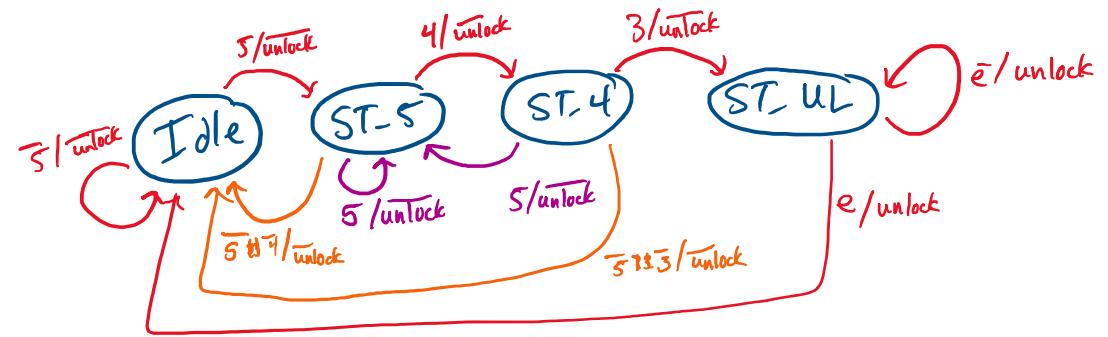
```
//comb logic block  
always_comb begin
```

```
end
```



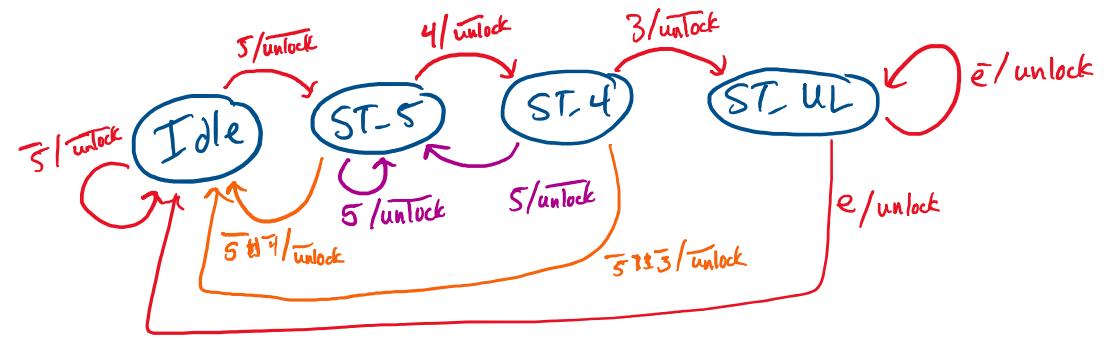
State Machine in Verilog

```
//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

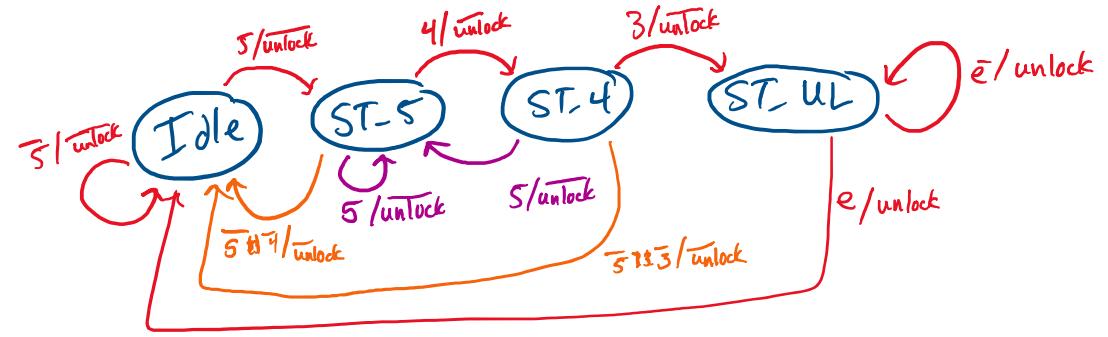
```
case (state)
    ST_IDLE:
        if (num[5])
            next_state = ST_5;
    ST_5: begin
        if (num[4])
            next_state = ST_4;
    end
    ST_4: begin
        if (num[3])
            next_state = ST_UL;
```



```
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
```