ENGR 210 / CSCI B441
“Digital Design”

Finite State Machines II

Andrew Lukefahr

Announcements

- P8 – Elevator Controller is out
  - This one is hard.

- P9 – SPI is out
  - This one is new. Might be some changes.

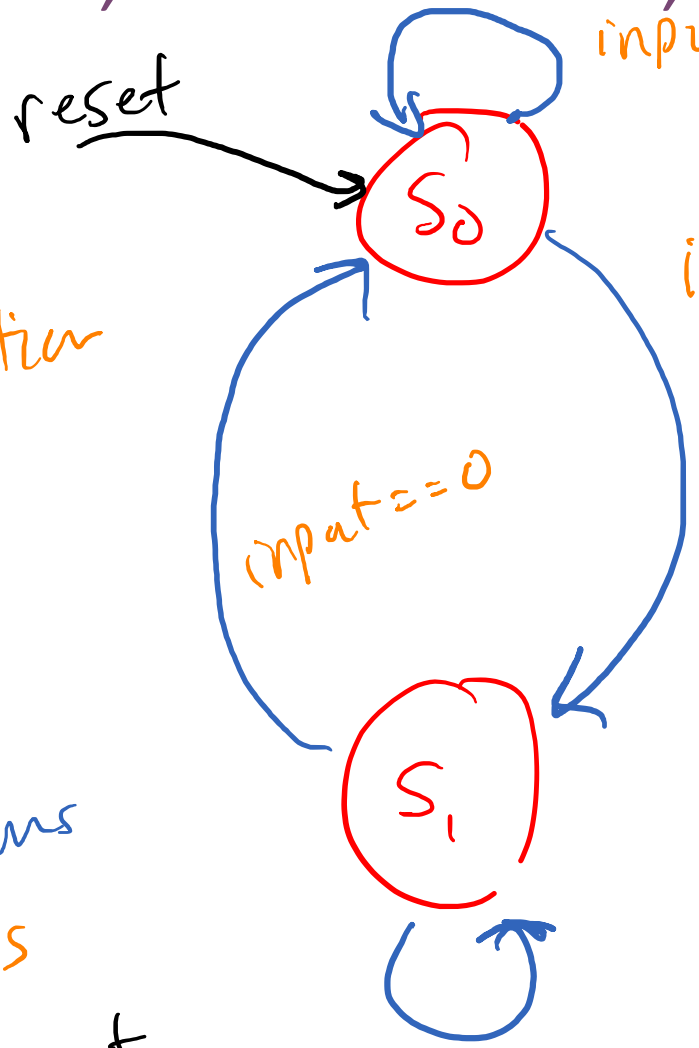
NoLatch

Always specify defaults for `always_comb`!

BLOCKING (=) FOR `always_comb`

NON-BLOCKING (<=) for `always_ff`

Review: States, Transitions, and Guards



guards: control transition which you take

S_0 & S_1: are states

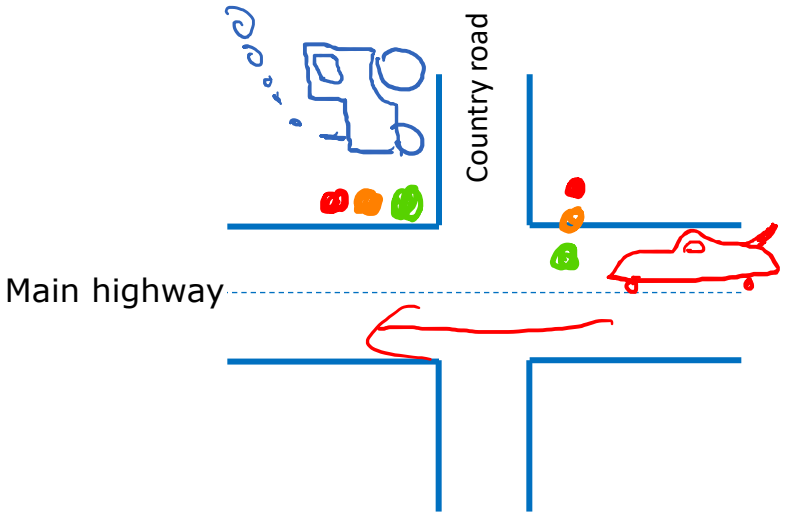
two states in this machine

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

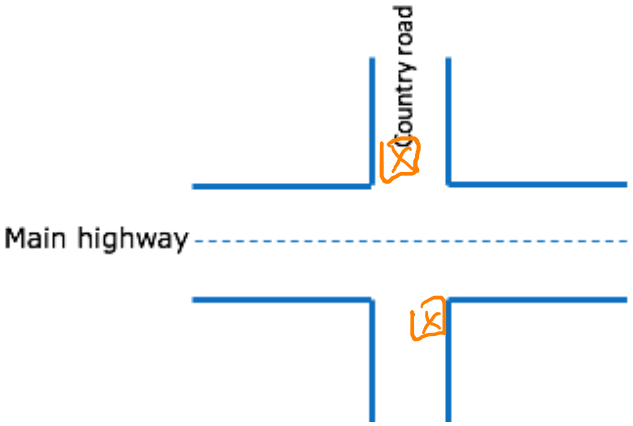
- Cars occasionally arrive from the country road. The traffic signal for the country road must turn green only long enough to let the cars on the country road go.
- When no cars are waiting on the country road, the country road traffic signal turns yellow then red and the traffic signal on the main highway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



reset

S_0 H: C:

S_1 H: C:

S_2 H: G:

S_3 H: C:

S_4 H: C:

S_5 H: C:

x==0

x==1

x==1 | x==0

x==1

x==0

x==1

x==0

Dont care
no guards, just transition

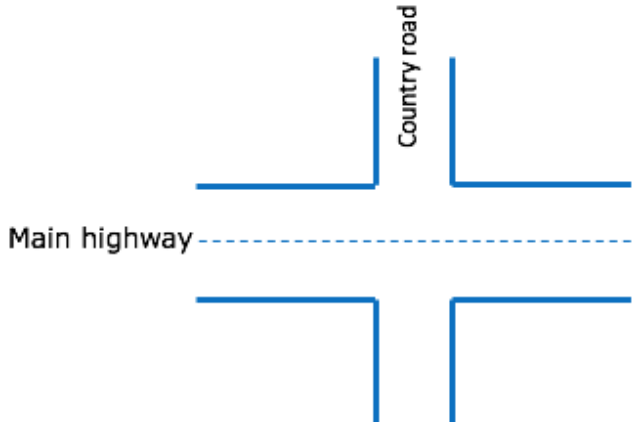
→ also OK

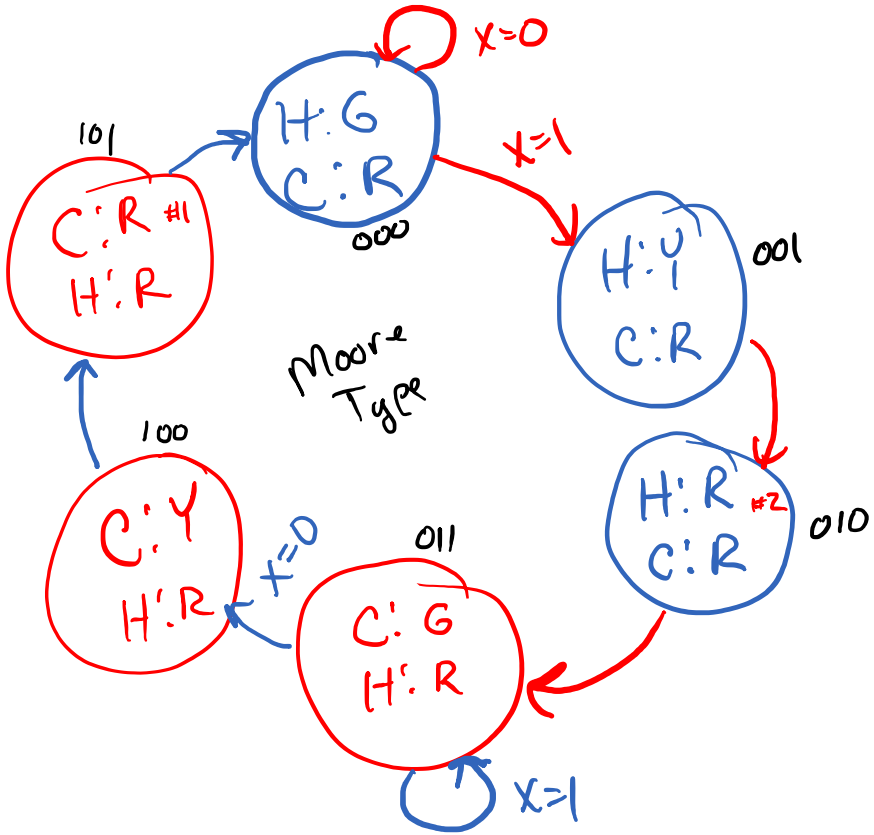
both are OK

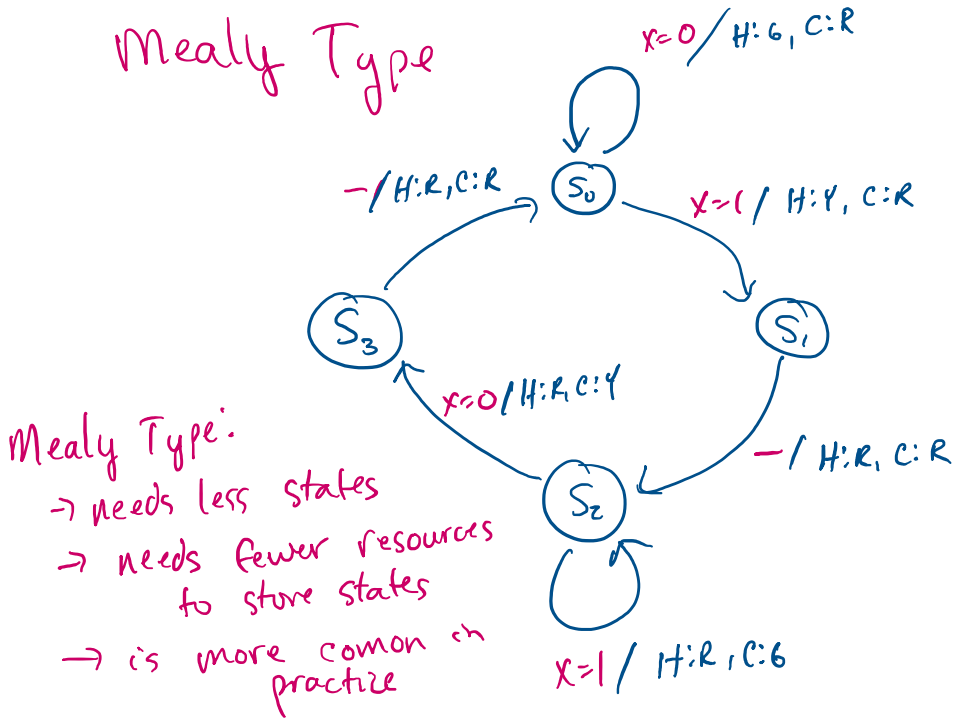
Moore vs. Mealy Type FSMs

- 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

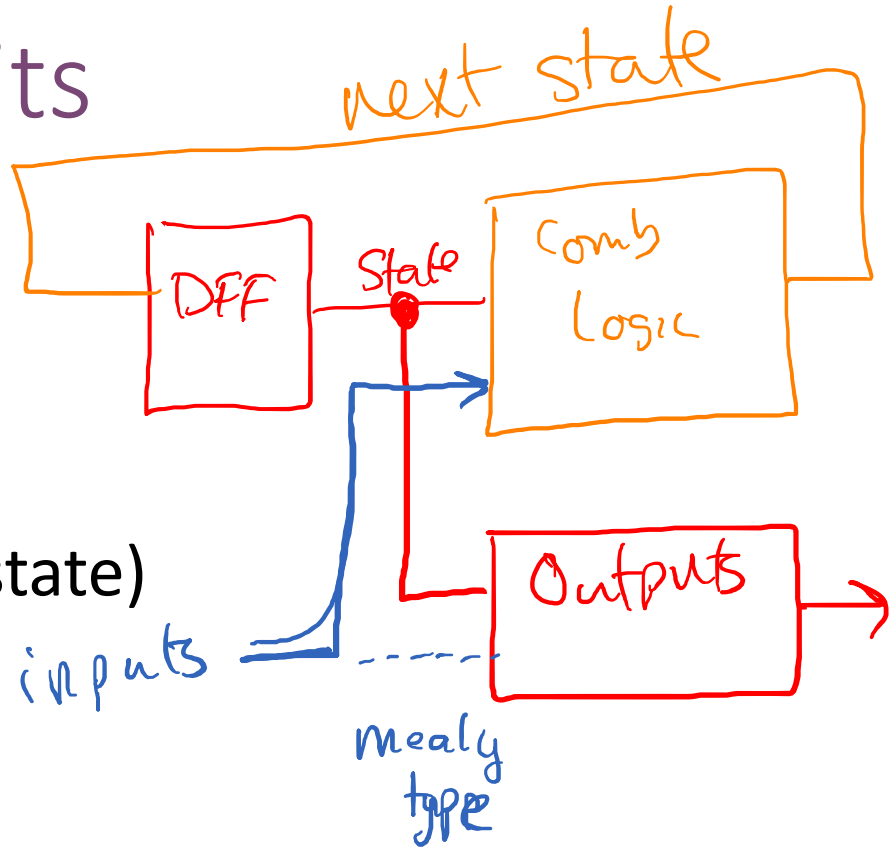






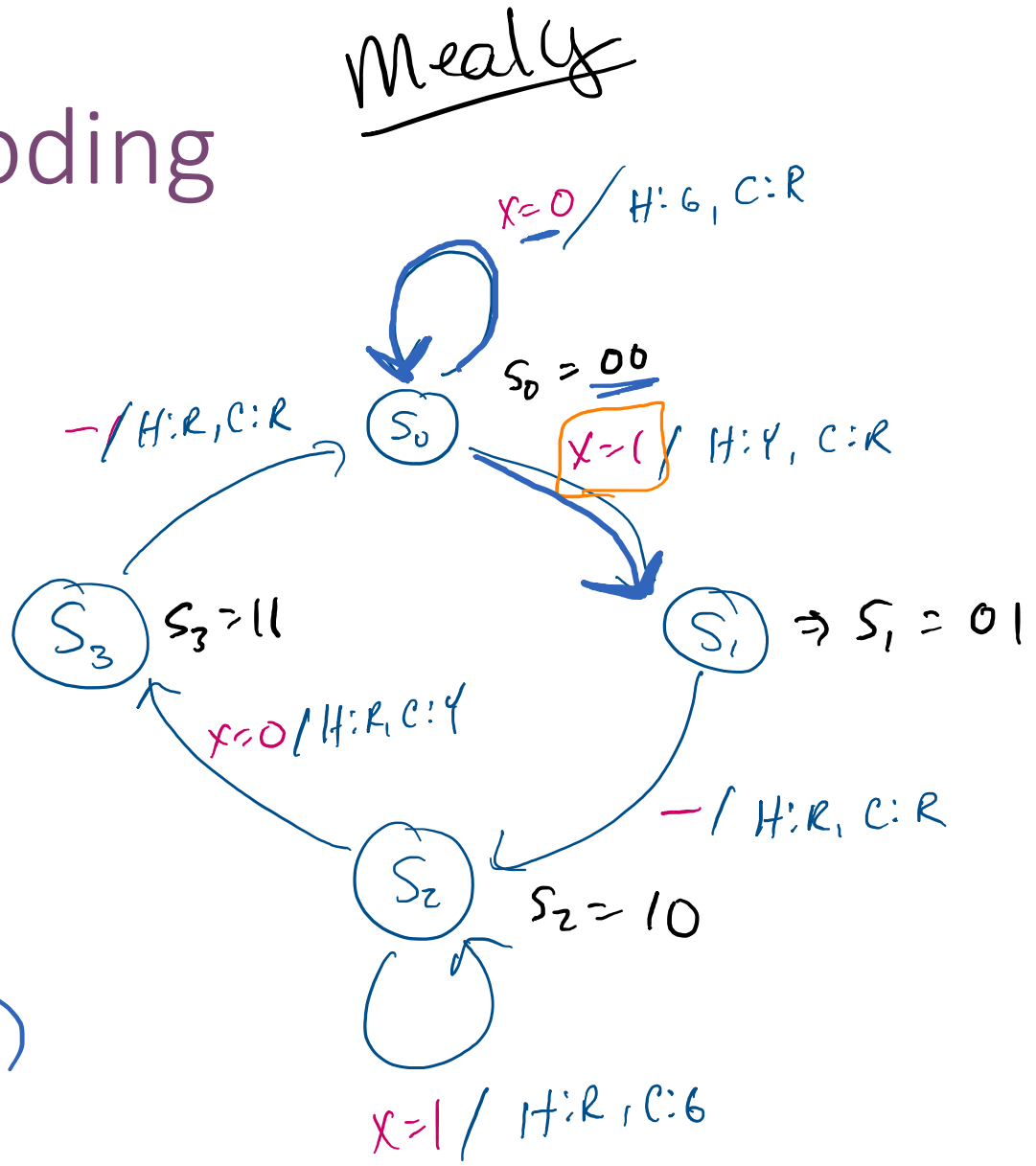
Implementing FSMs with Circuits

- 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

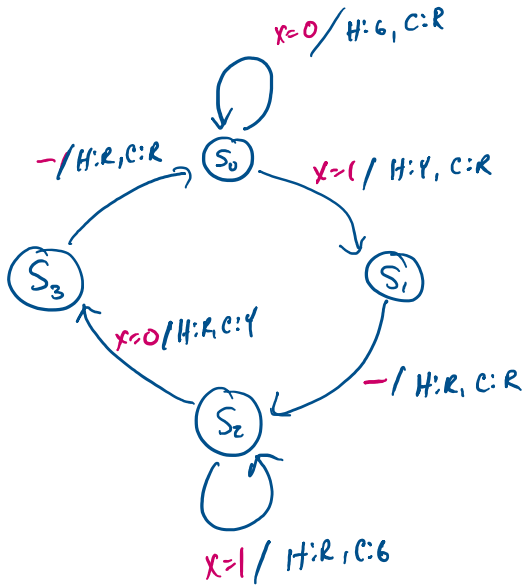


State Transition Encoding

State	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)

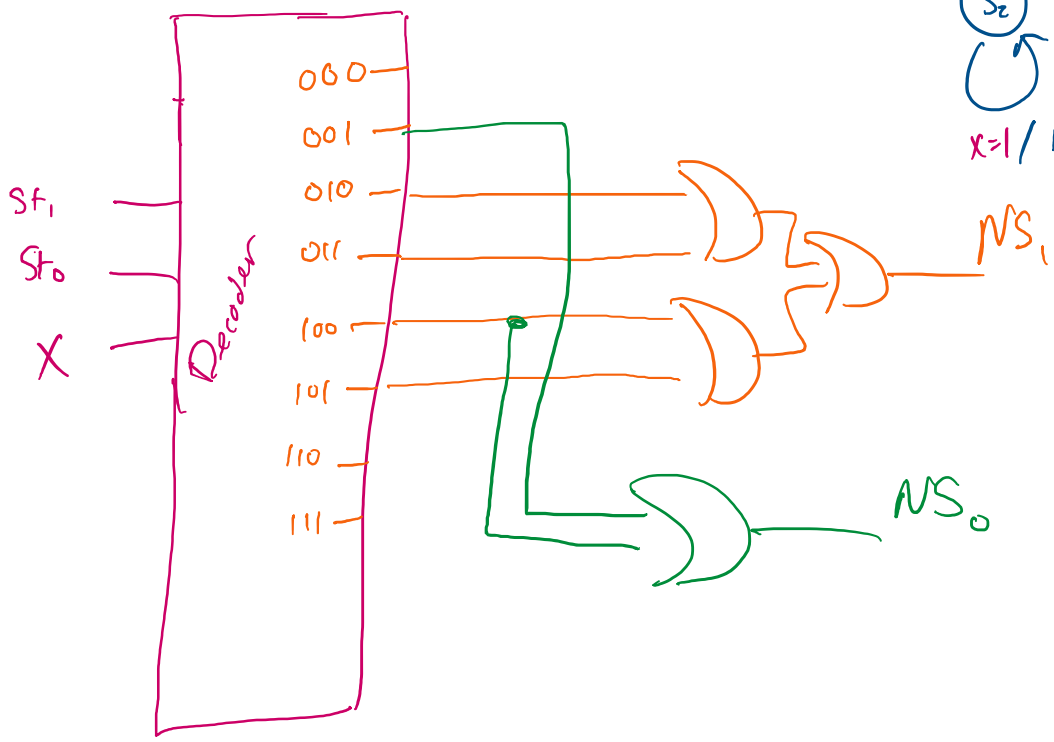


Next State Logic

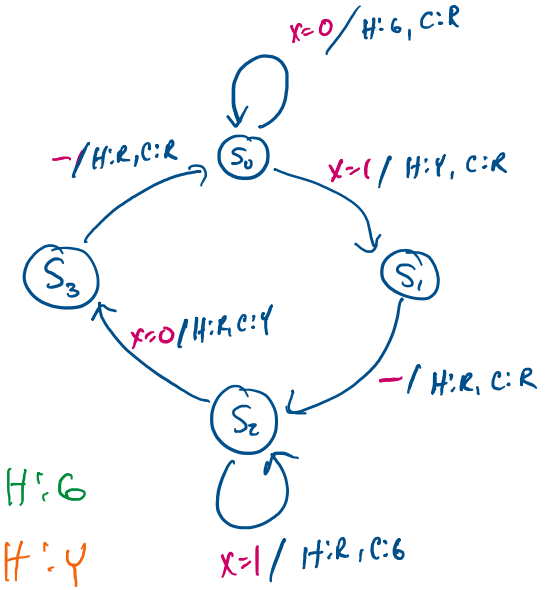


	State	X	Next State
0	00	0	00
1	00	1	01
2	01	0	10
3	01	1	10
4	10	0	11
5	10	1	10
6	11	0	00
7	11	1	00

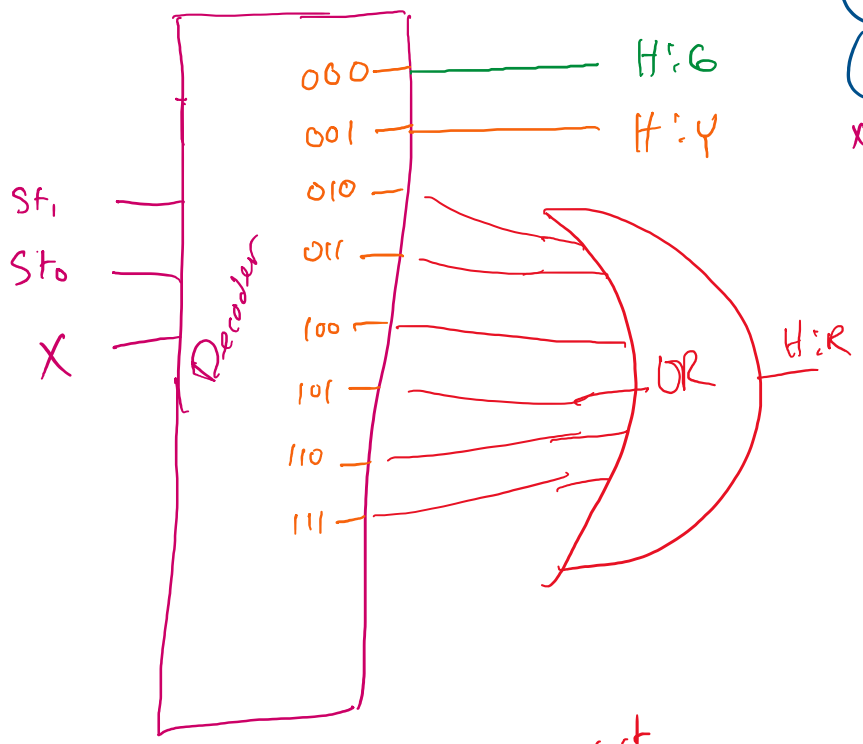
NS_1 NS_0



Output Logic (Highway)

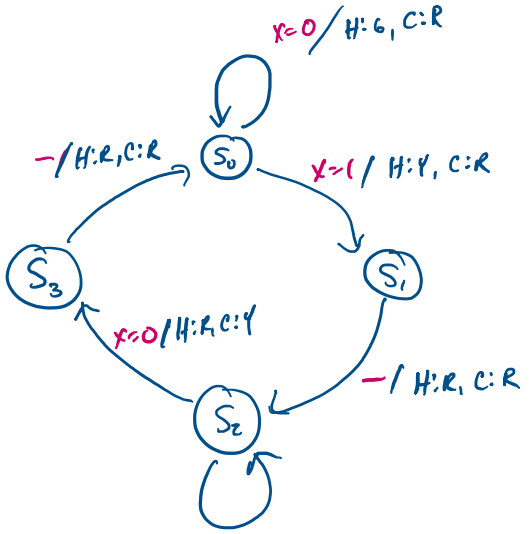


	State	X	H:R	H:Y	H:G
0	00	0	0	0	1
1	00	1	0	1	0
2	01	0	1	0	0
3	01	1	1	0	0
4	10	0	1	0	0
5	10	1	1	0	0
6	11	0	1	0	0
7	11	1	1	0	0

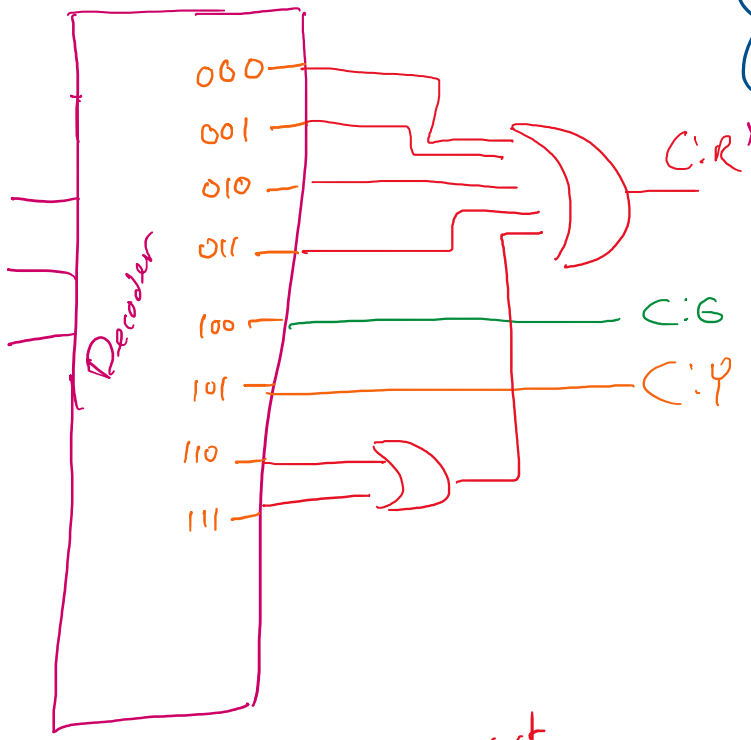


* simpler methods exist

Output Logic (Country Road)



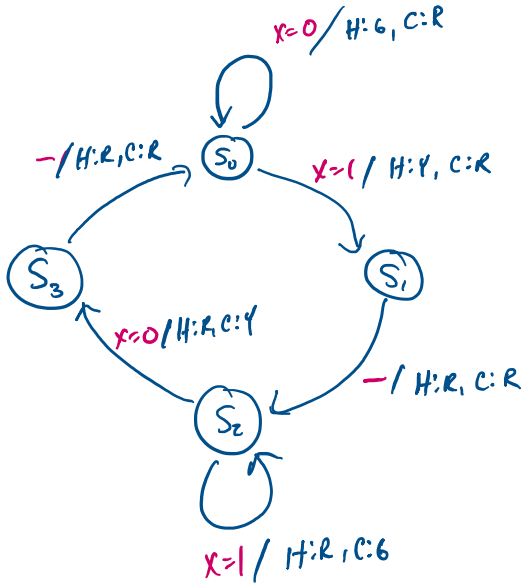
	State	X	C:R	C:Y	C:G
0	00	0	1	0	0
1	00	1	1	0	0
2	01	0	1	0	0
3	01	1	1	0	0
4	10	0	0	1	0
5	10	1	0	0	1
6	11	0	1	0	0
7	11	1	1	0	0



* simpler methods exist

State Machine to Verilog

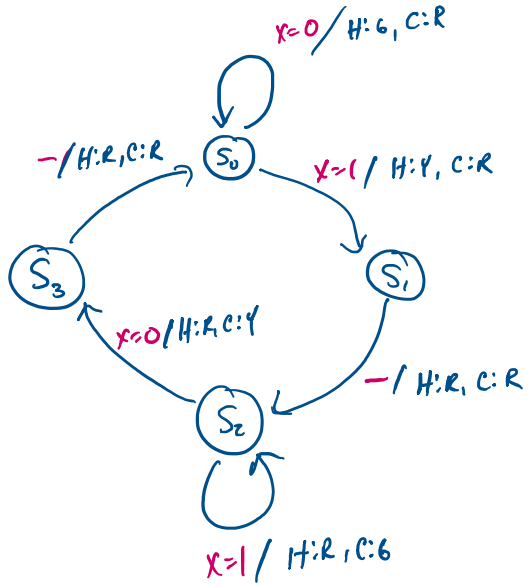
- Define states?




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

State Machine to Verilog

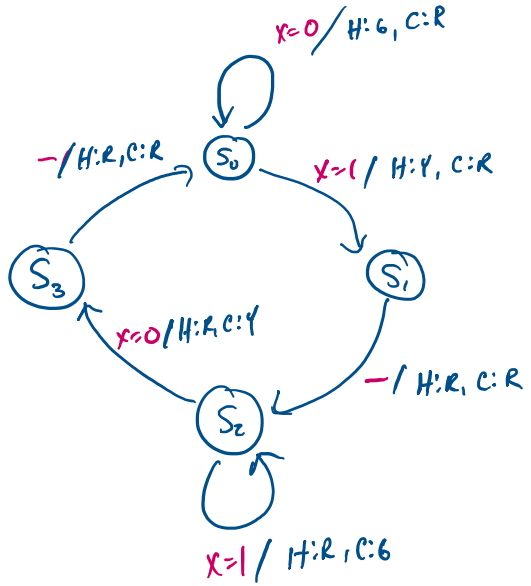
```
always_ff @(posedge clk) begin
    if (rst) state <= ST_0;
    else state <= nextState;
end

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?

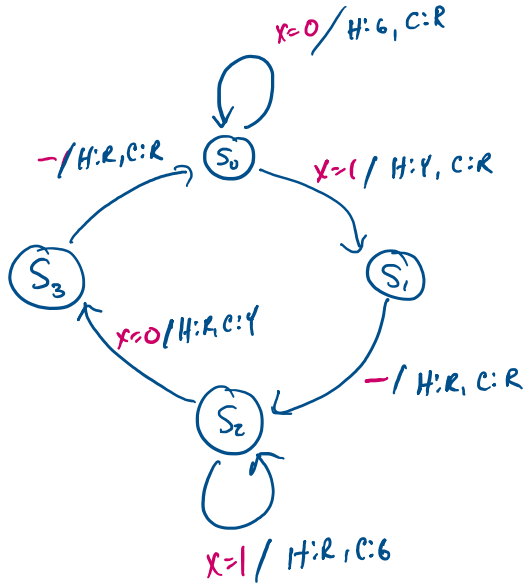
```
always_comb begin
        nextState = state; //default
        case(state)
                ST_0:
                        if (X) nextState = ST_1;
                ST_1:
                        nextState = ST_2;
                ST_2:
                        if (~X) nextState = ST_3;
              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



State Machine to Verilog

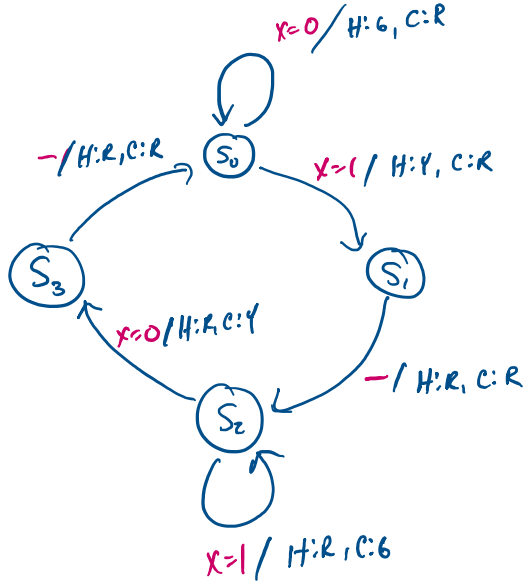
- What <u>else</u> is this missing?

```
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
```

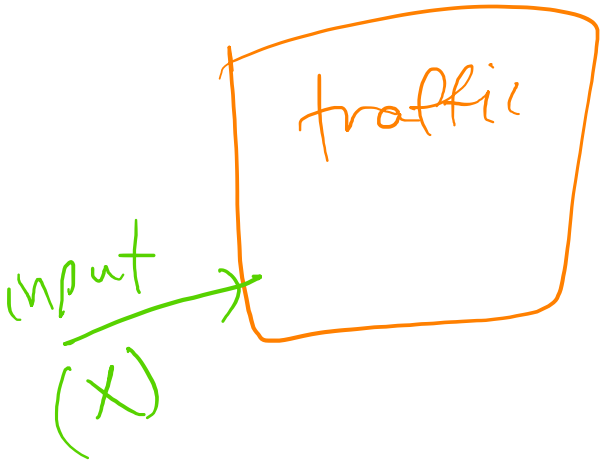


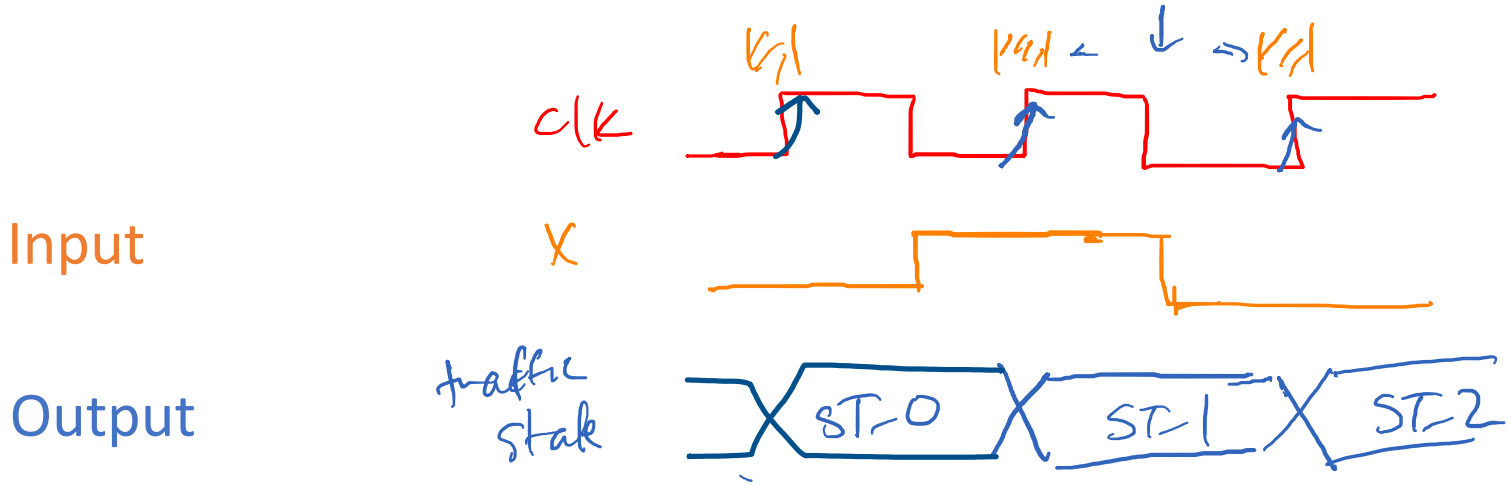
```
module traffic(
    input clk,
    input rst,
    input x,
    output logic [2:0] Hryg, //red-yellow-green
    output logic [2:0] Cryg  //red-yellow-green
    );

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
```

```
 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
```

Testbenches for State Machines





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

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.

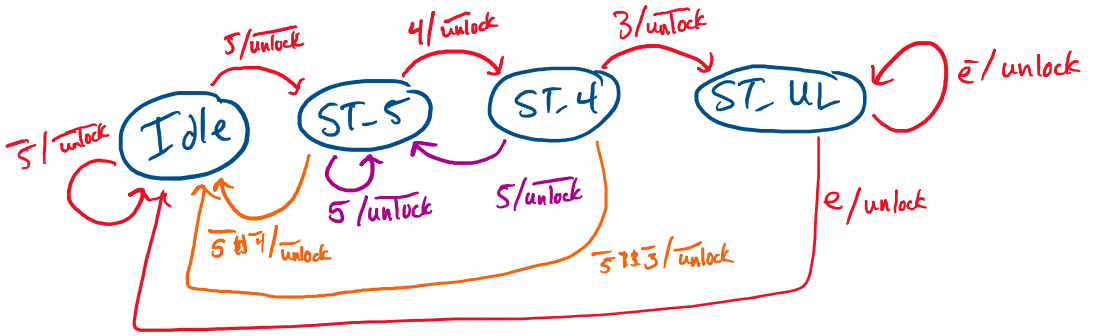
Should unlock: 5-5-4-3, 5-4-5-4-3

- Draw the state machine!

Lock State Machine

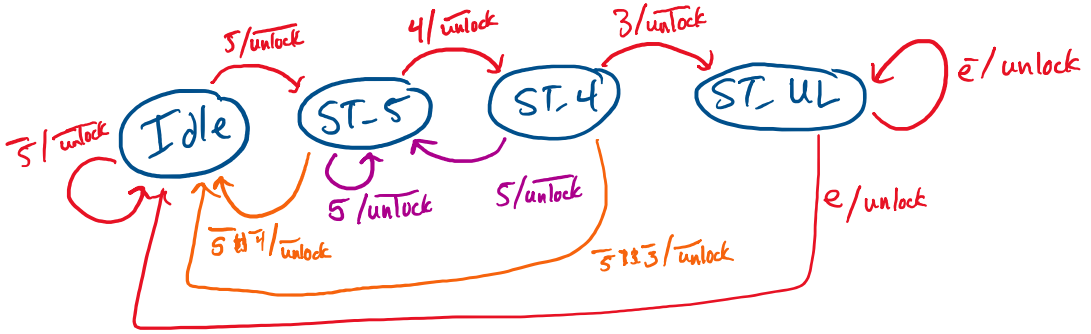
- `Recall: 5 - 4 - 3`
- `E: relock`

State Machine in Verilog




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

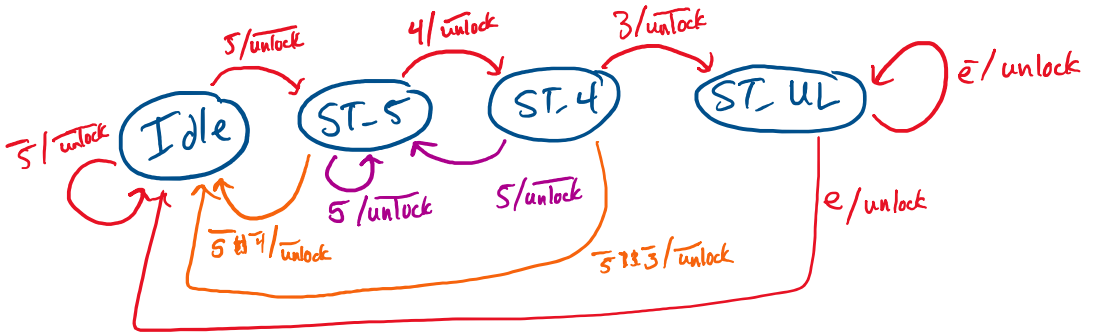
State Machine in Verilog



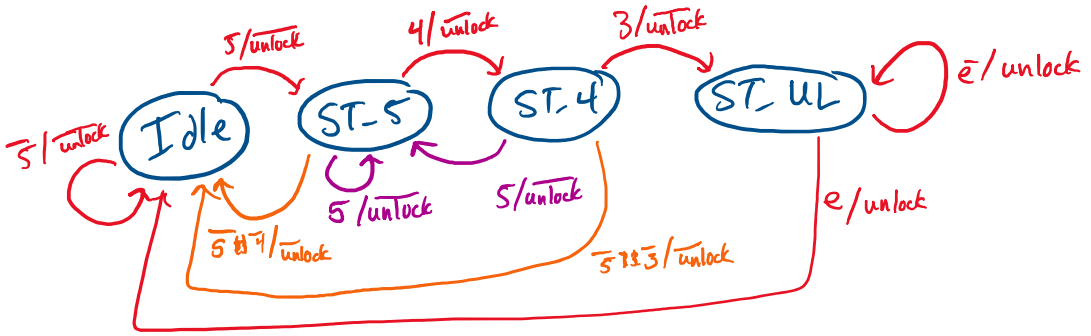

```
module Lock(
    input clk, rst,
    input [9:0] num,
    input e, //relock
    output unlock
);

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
```



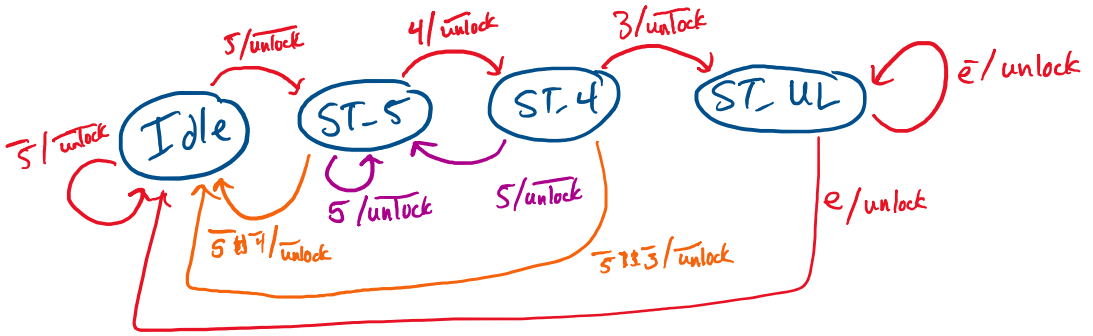
State Machine in Verilog




```
//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
```

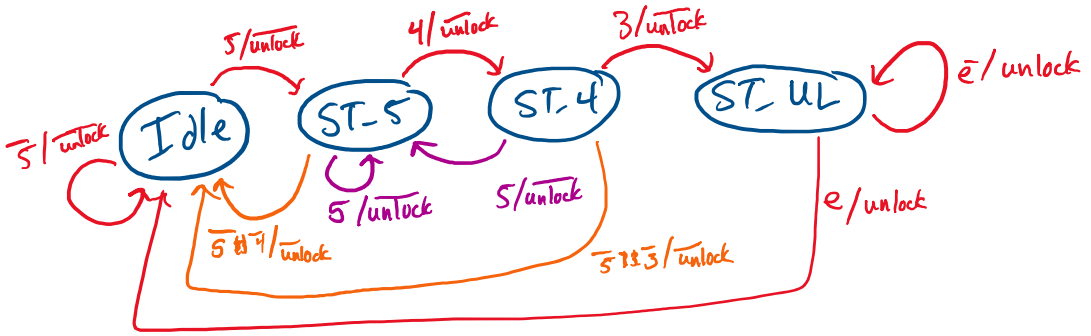
State Machine in Verilog

```
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

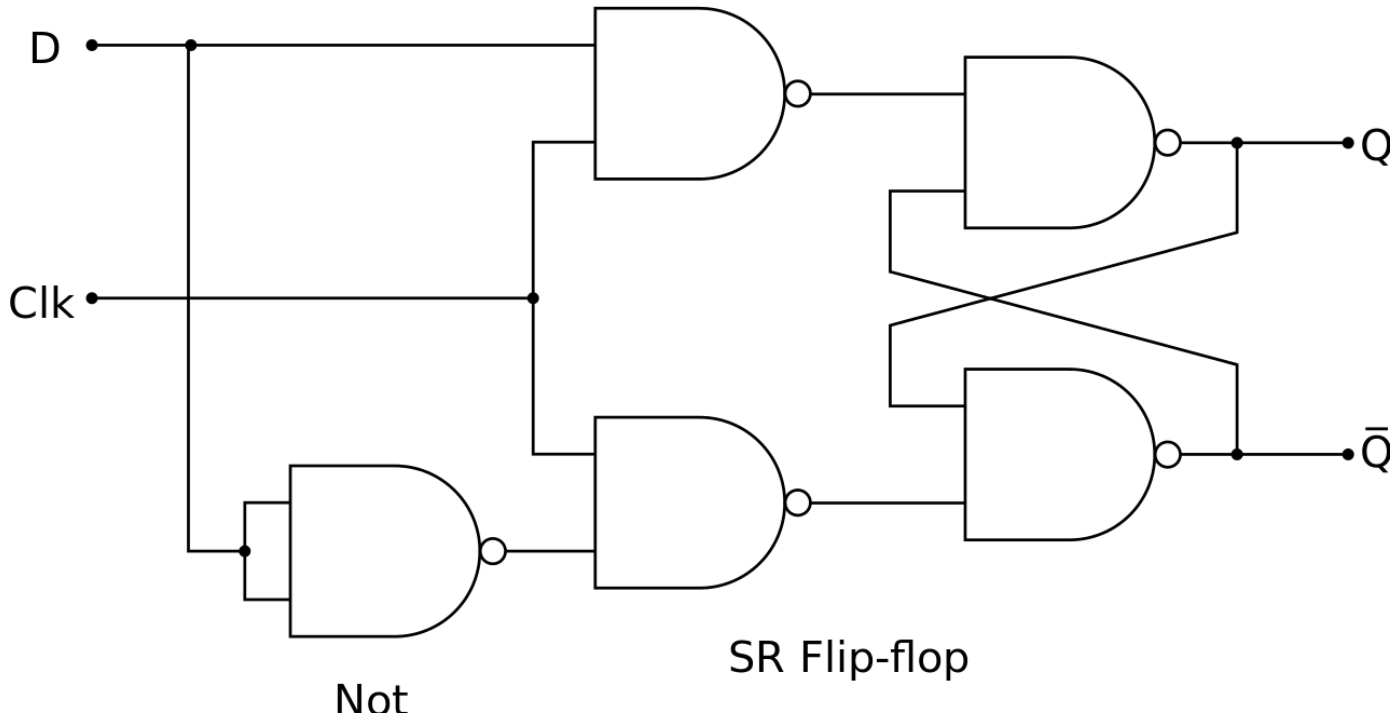



```
case (state)
  ST_IDLE:
    if (num[5])
      next_state = ST_5;
  ST_5: begin
    if (num[4])
      next_state = ST_4;
    else if (num[5])
      next_state = ST_5;
    else if ( (|num) | e ) //other btns
      next_state = ST_IDLE;
  end
  ST_4: begin
    if (num[3])
      next_state = ST_UL;
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

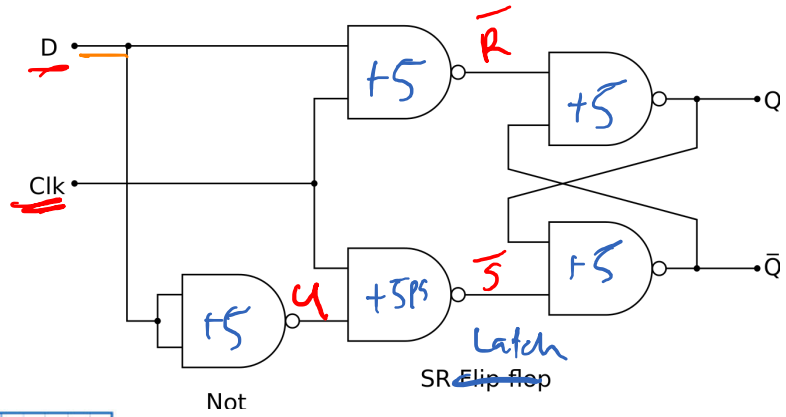
Timing

Flip-Flop Timing

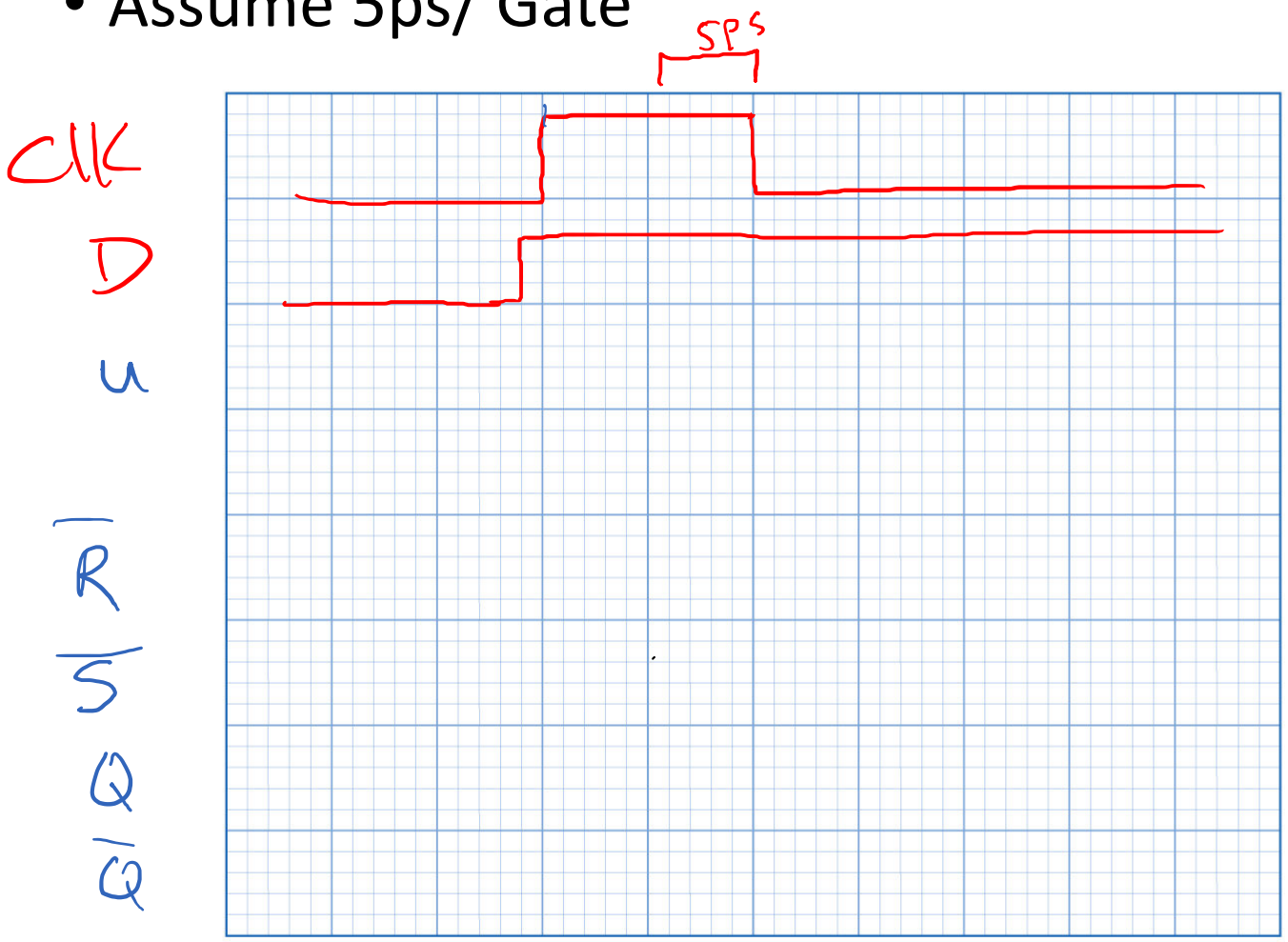
- What's the delay for this Latch?
- Assume 5ps/ gate
- What about a Flip-Flop?



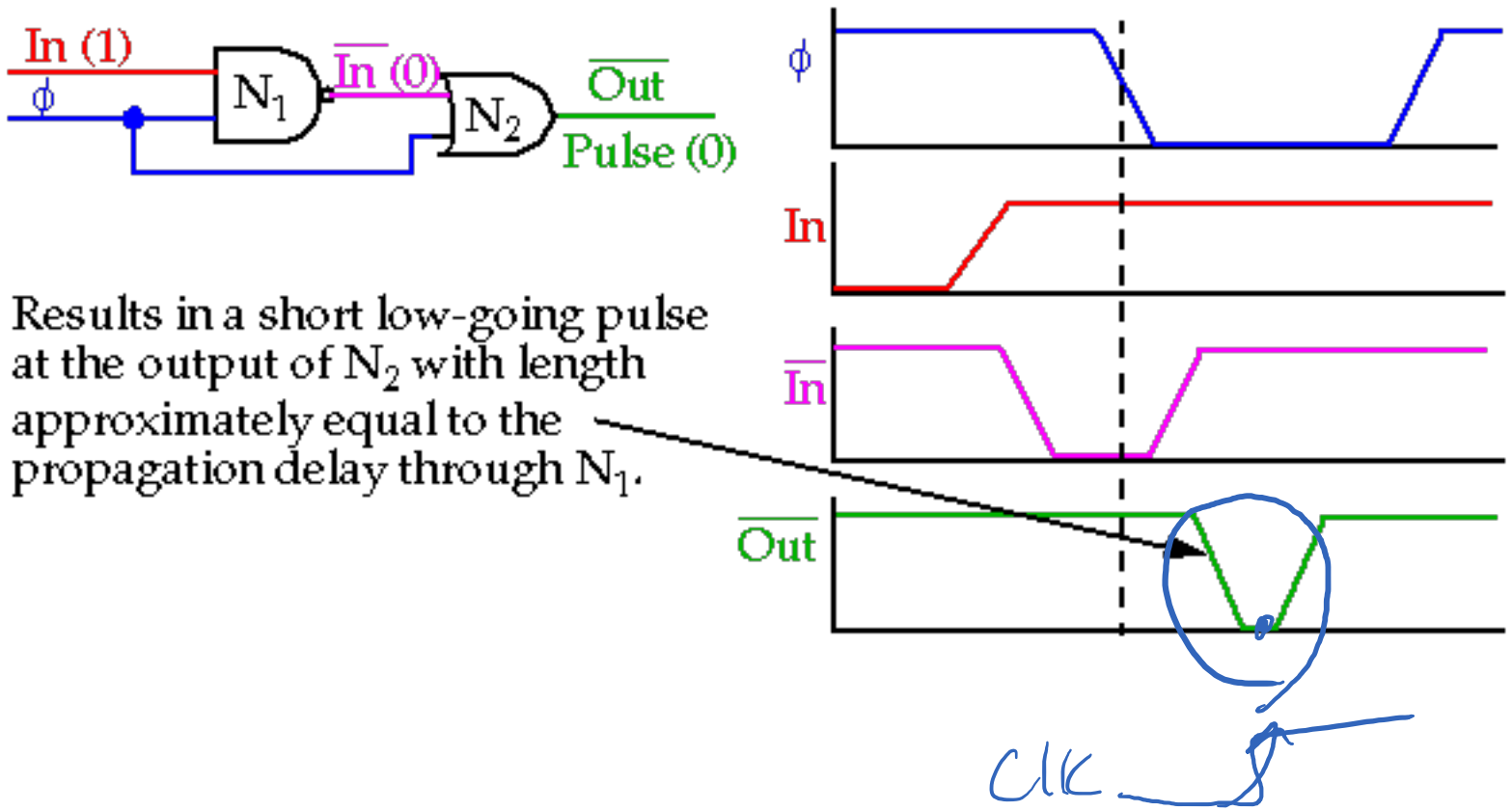
Flip-Flop Timing



- Assume 5ps/ Gate



Glitch


In (1)
ϕ
N1
In (0)
N2
Out
Pulse (0)
Results in a short low-going pulse at the output of N2 with length approximately equal to the propagation delay through N1.
ϕ
In
In
Out
CLK

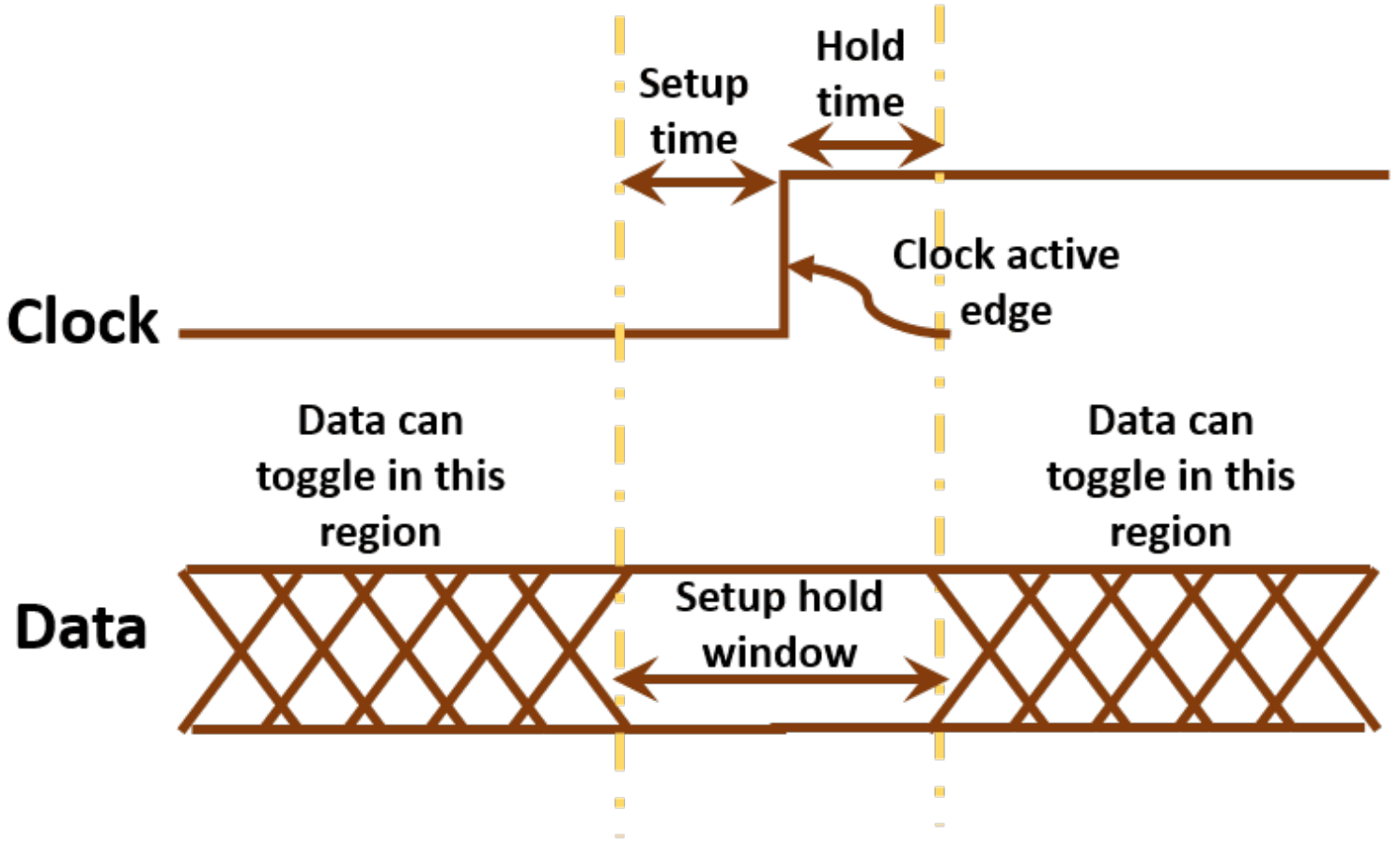
Setup and Hold Time

- **Setup Time**: minimum time the inputs to a flip-flop **must** be stable **before** the clock edge

- **Hold Time:** minimum time the inputs to a flip-flop **must** be stable **after** the clock edge

Setup Time

Setup/Hold Time



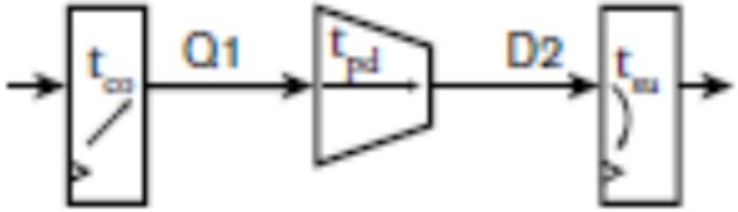
Inter Flip-Flop Timing

Intra Flip-Flop Timing

Intra Flip-Flop Timing

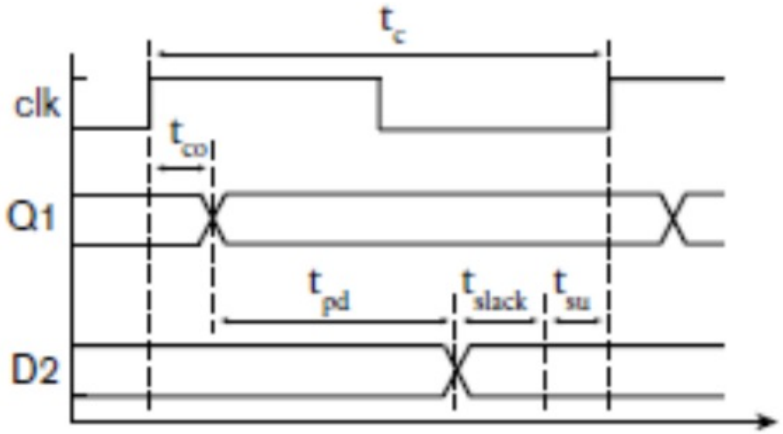
Register to register timing:

- output of a register *Q1*
- some combinational circuit
- input to the next register *D2*



Delays:

- t_{co} , clock to output delay,
- t_{pd} , propagation delay in combinational circuit
- t_c , clock period



Timing requirement:

$$t_{co} + t_{pd} + t_{su} < t_c$$

Slack

- Extra time between combinational propagation delay and setup time
- Time between stable input to Flip-Flop and next clock edge

- Vivado:
  - WNS: Worst-case Negative Slack

- If this number is <0, your circuit will (probably) not work

Next Time

- Communication