ENGR 210 / CSCI B441
"Digital Design"

# Finite State Machines III

+ Timing

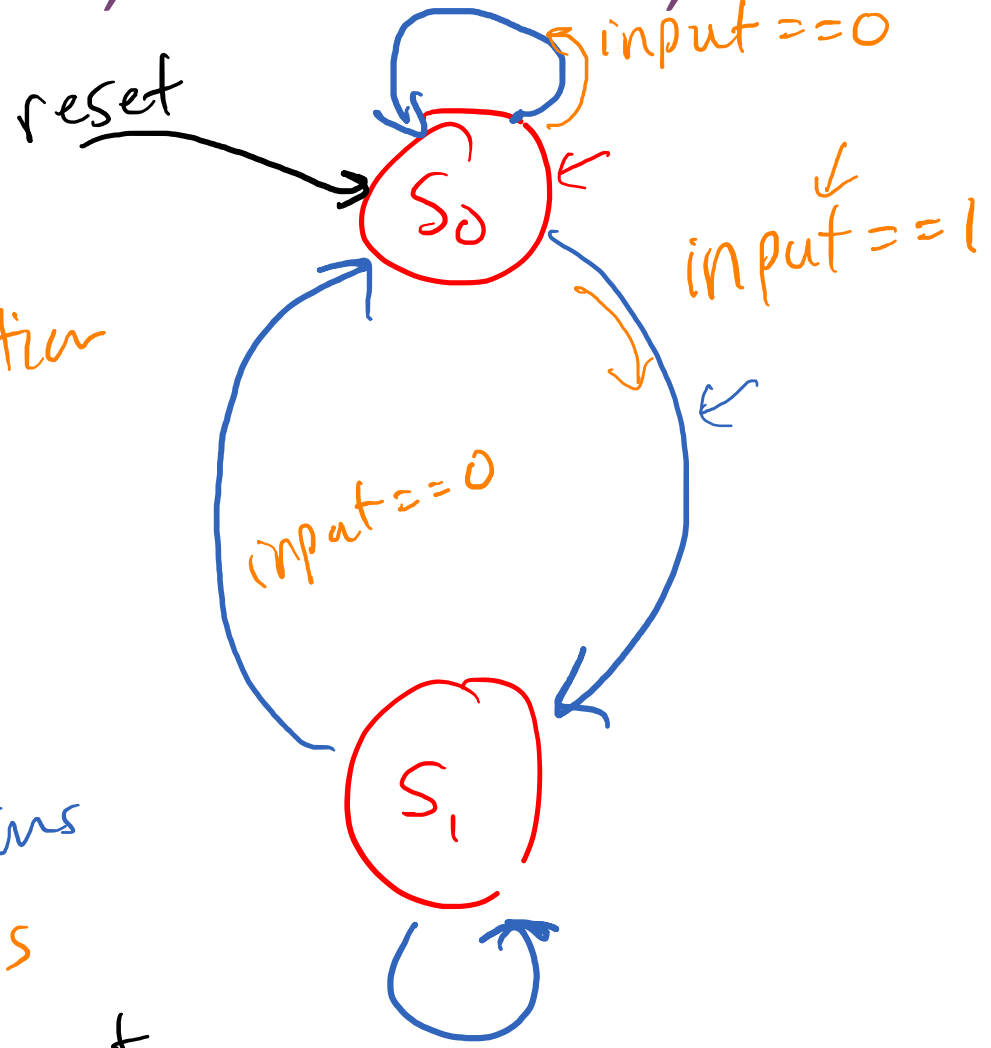
Andrew Lukefahr

# Announcements

- P3 is due Friday.
  - Adds “demo” requirement.  Will need real hardware.
  - Demo:  create video and upload to Canvas
- Upload Box Number to Canvas
  - Part of P3’s grade
- 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

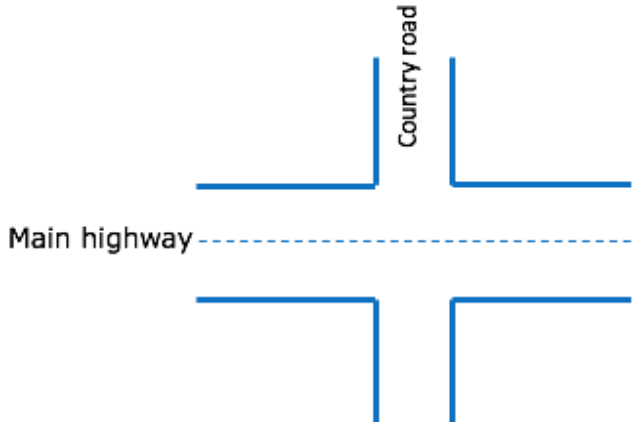
$S_0$ & $S_1$: are states

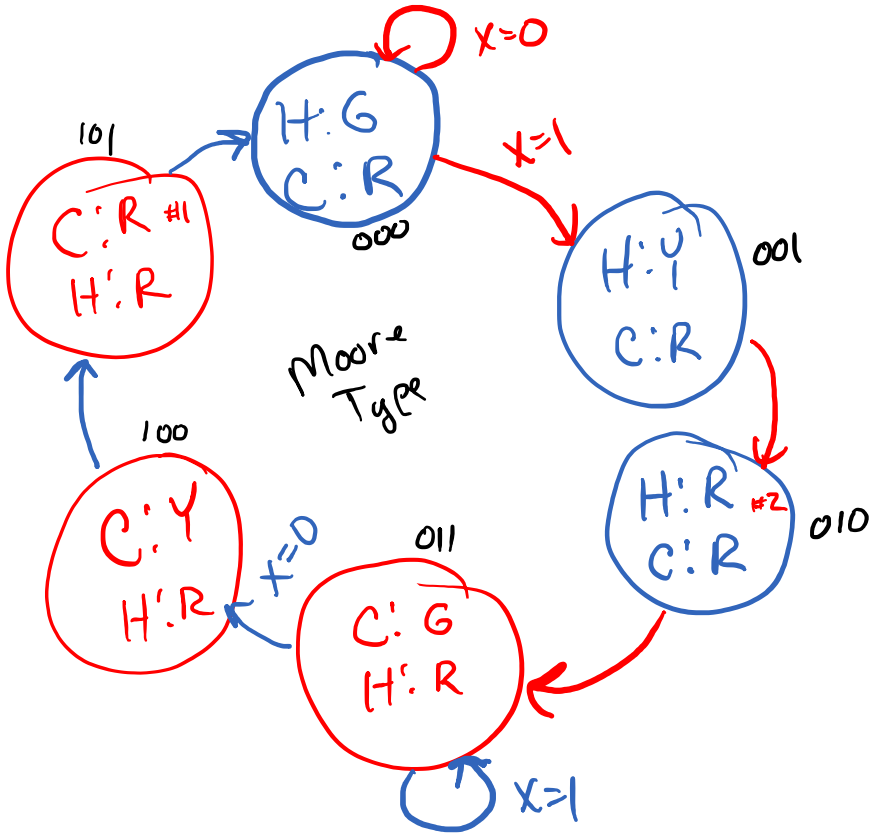
two states in this machine

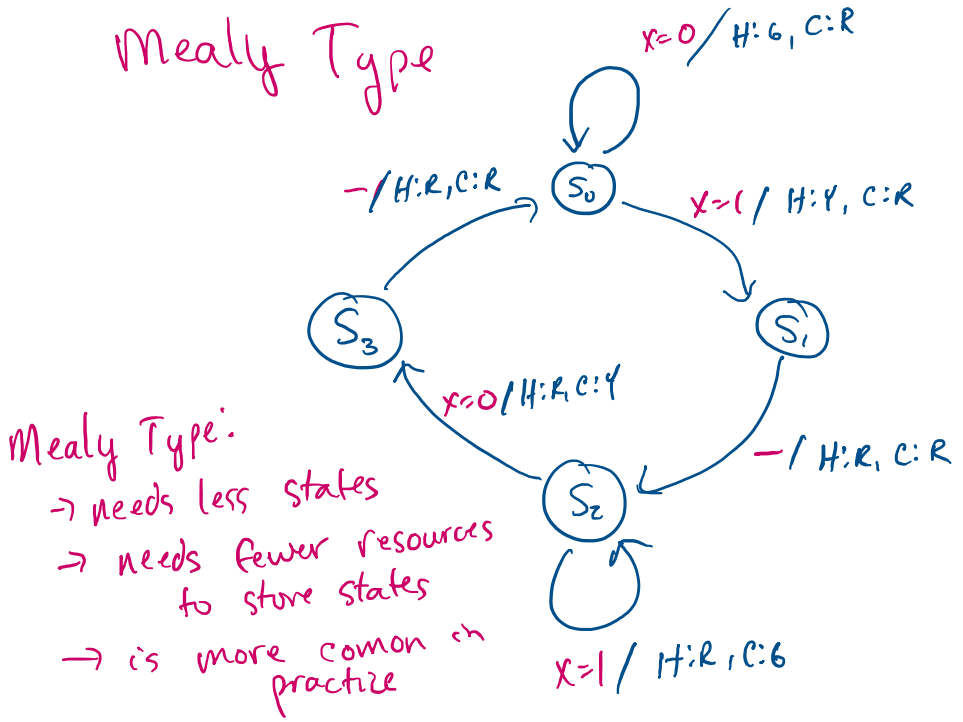
transitions
→ leaving one state & going to another
(happen @ posedge clk)

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

# Traffic Light: Moore vs. Mealy

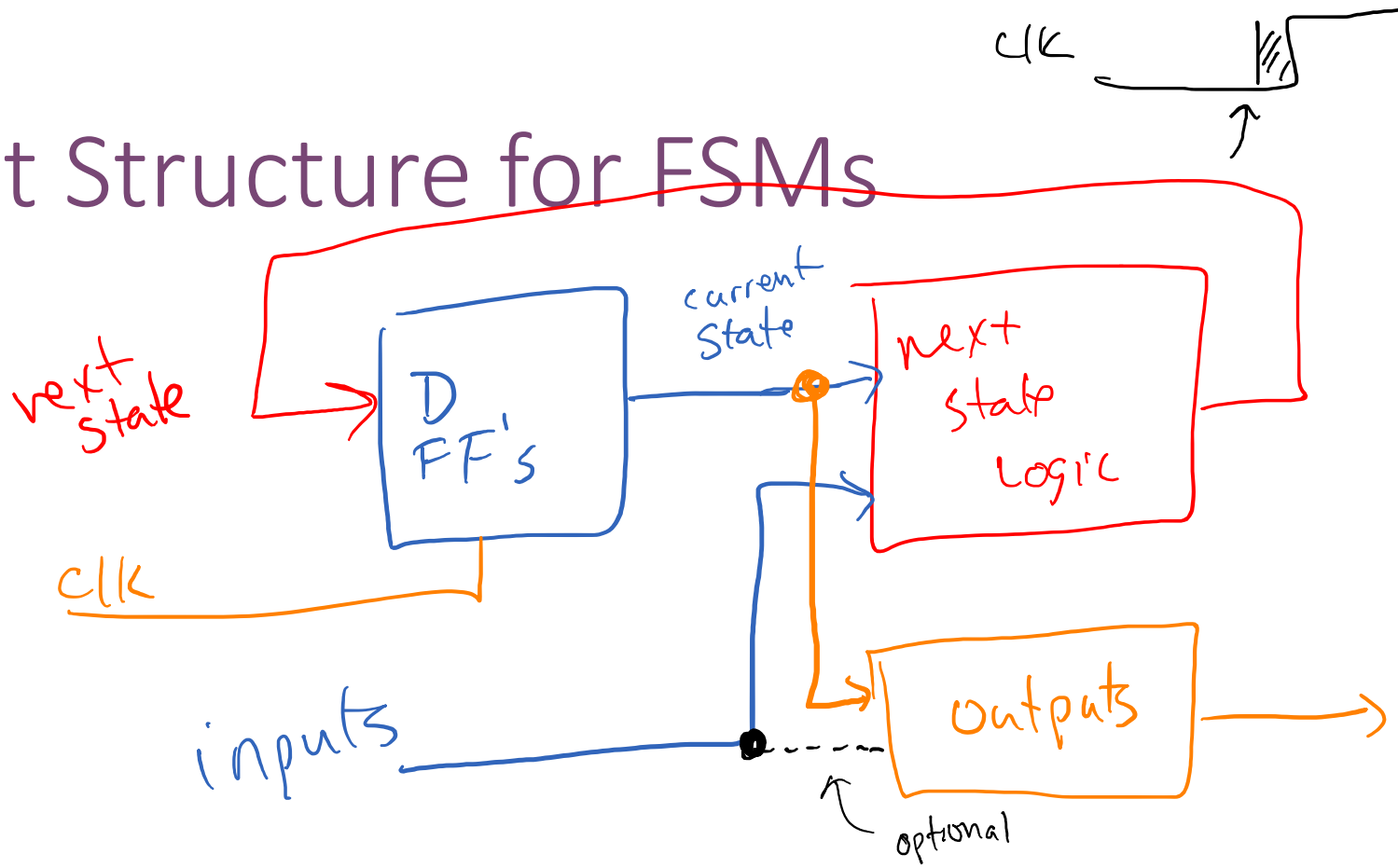



Mealy Type:
- → needs less states
- → needs fewer resources to store states
- → is more comon in practice

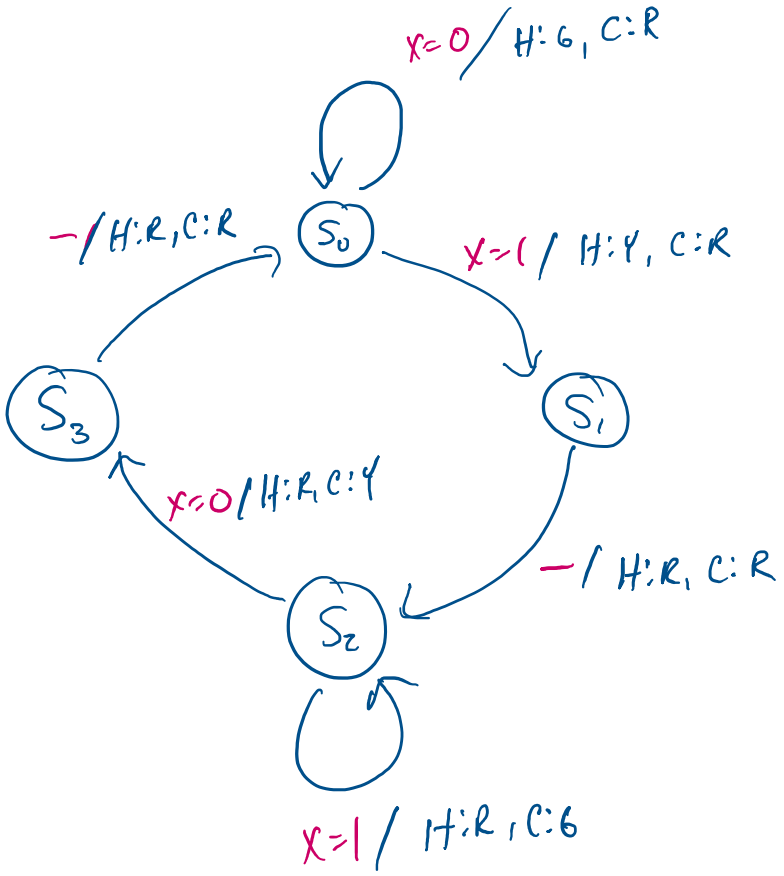
# Circuit Structure for FSMs

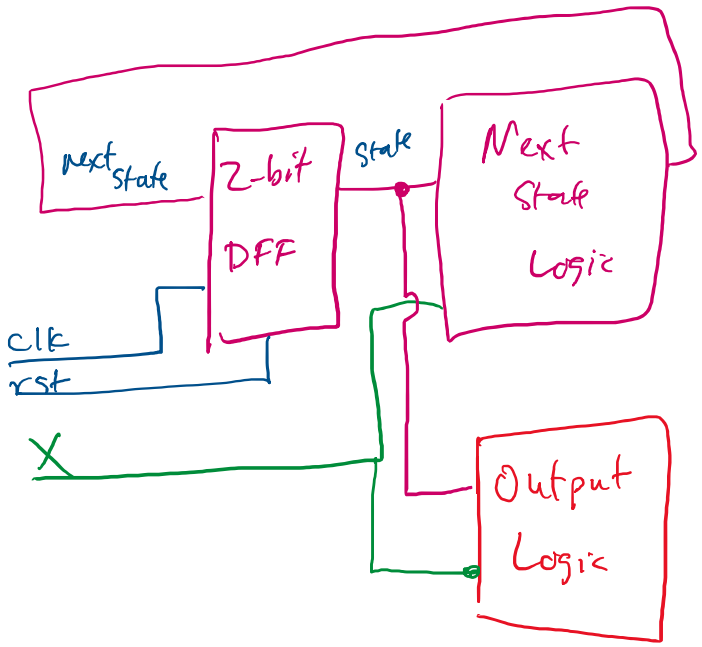



Moore Machine: outputs are a function of current state

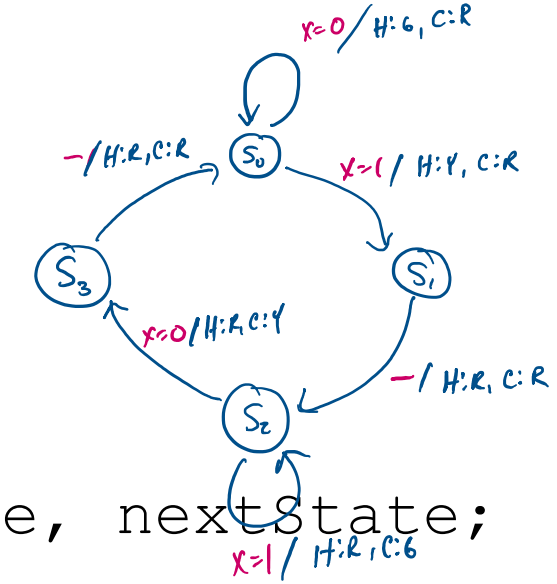
Mealy Machine: outputs are a function of current state + inputs

# State Machine to Logic



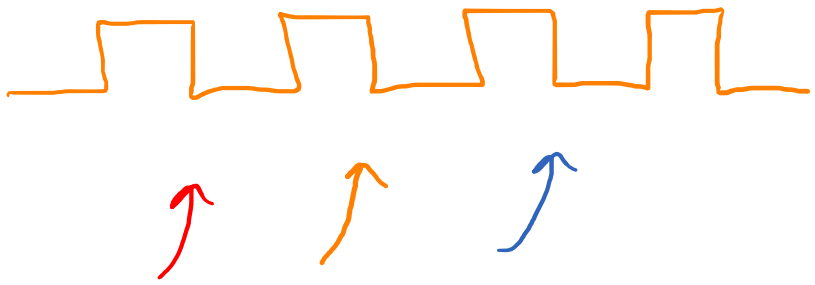
# State Machine to Verilog




- Build State Machine?

```
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
```



@(negedge clk);

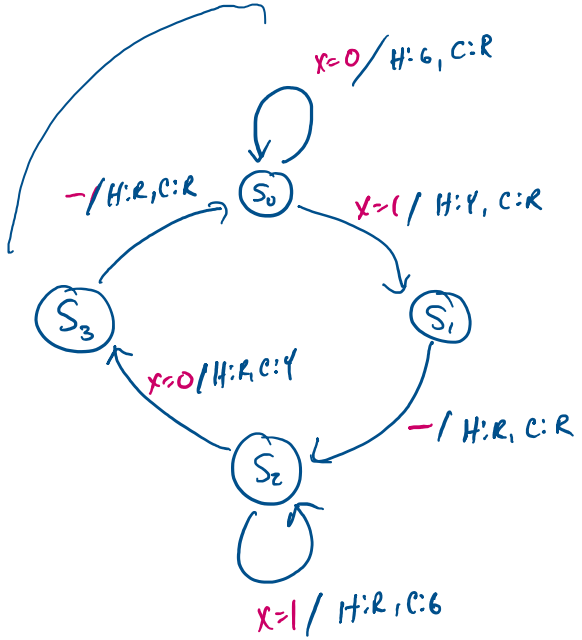
@(negedge clk); // advance simulation

@(negedge clk);

- What is `nextState`?

# State Machine to Verilog

```
always_comb begin
        nextState = state; //default
        Hryg = 3'b001; Cryg=3'b100;
        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);
    @(negedge clk);

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

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

    $finish;
end
endmodule
```

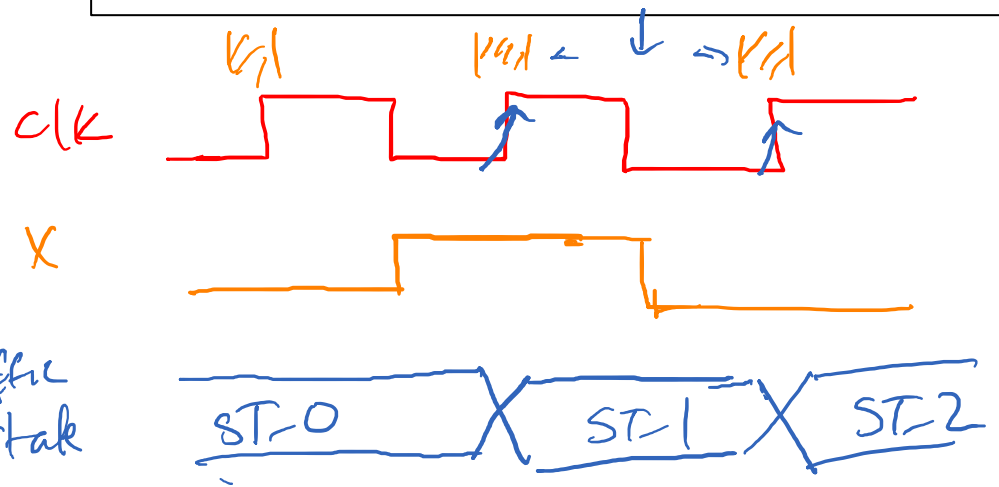
outputs

2 cycles

reset off

testbench: negedge

FPGA code: posedge



# 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 the `E` key is pressed. Ignore all other keys while unlocked.

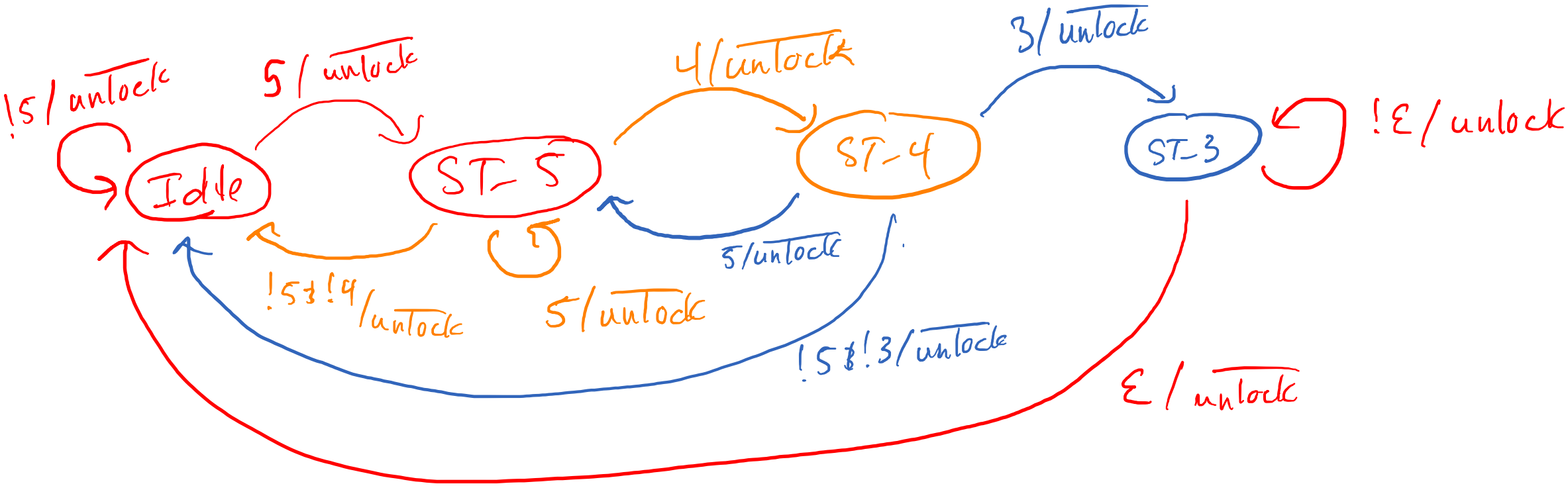
Should unlock: 5-5-4-3, 5-4-5-4-3

- Draw the state machine!

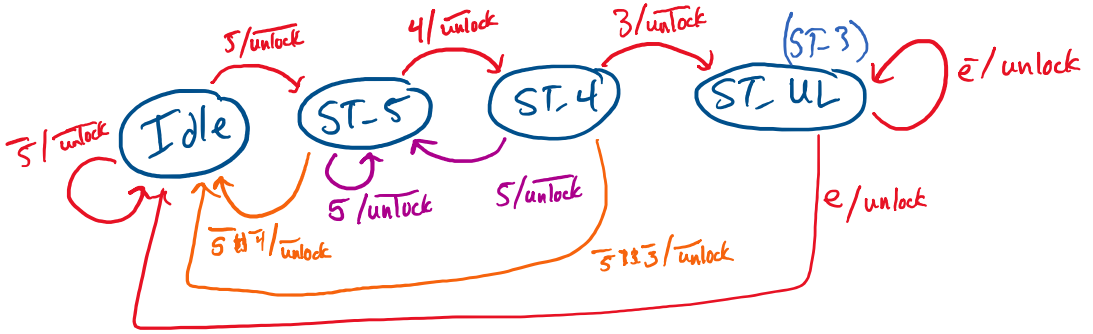
5 5 4 4 3


# 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
);
```

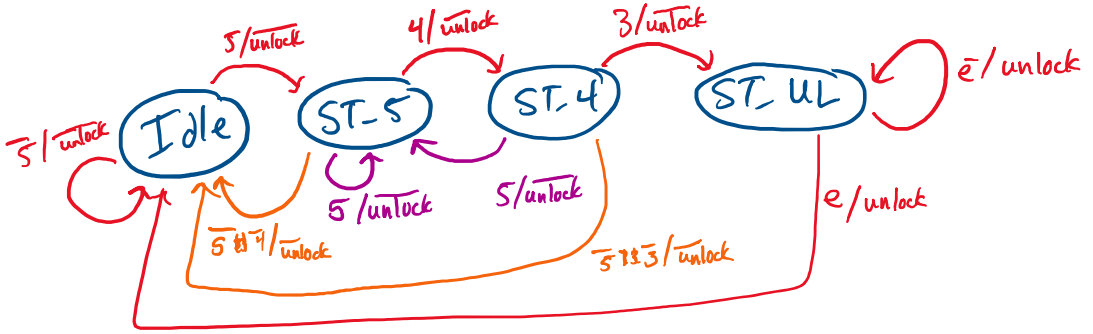
enum {Idle, ST_5, ST_4, ST_3} state, nextState;

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

0 1 2 3 4 5 6 7 8 9

num 0 0 0 0 0 1 0 0 0 0

# 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
```

→ e | (num);

=> e OR

num[0] OR

num[1] ...

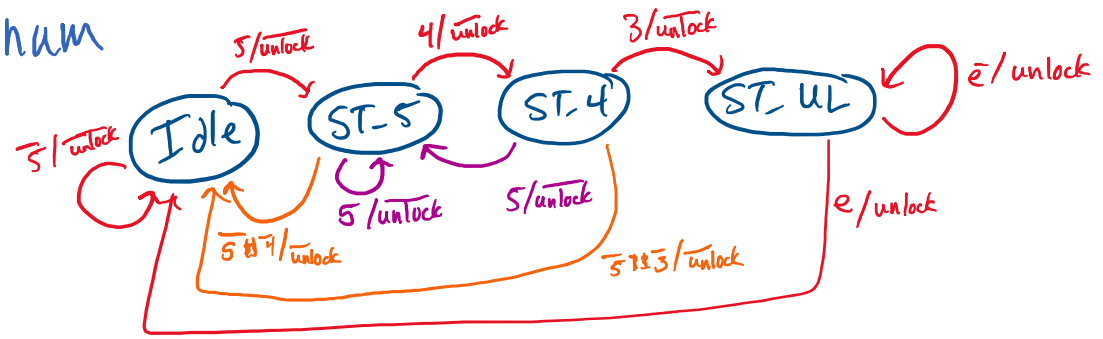
OR

num[9]

→ (num == 5)

num[0] | num[1] | num[2] ... | num[9] = | num

# State Machine in Verilog

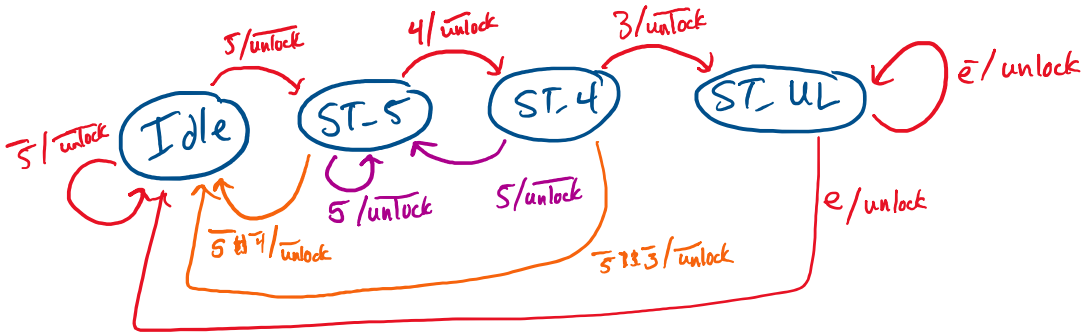



```
//comb logic block
always_comb begin
    nextState = state;
    unlock = 'b0;

    case (state)
        Idle: if (num[5] == 'b1) nextState = ST_5;
        ST_5: begin
                if (num[4] == 'b1) nextState = ST_4;
              else if (num[5] == 'b1) nextState = ST_5;
              else if (|num)          nextState = Idle;
```

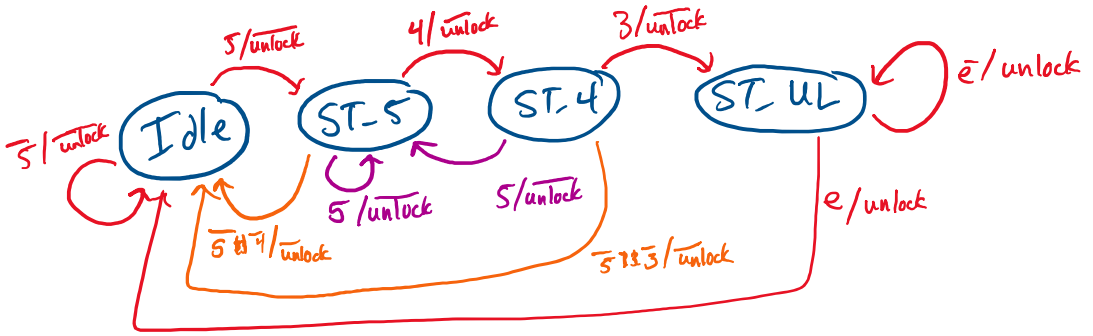
end

# State Machine in Verilog



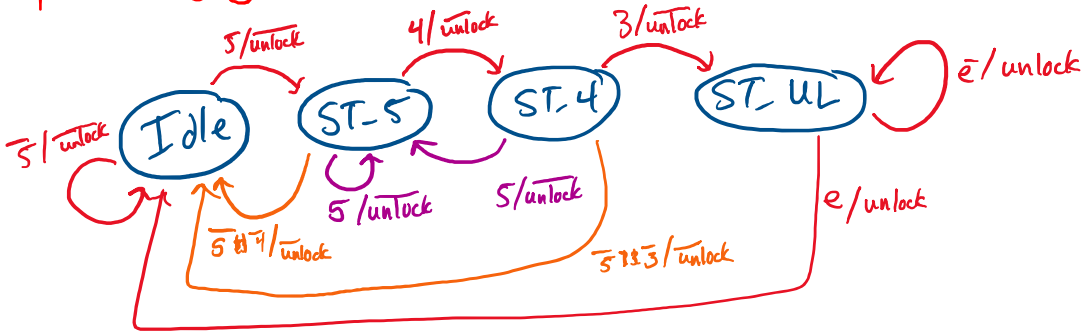

```
//comb logic block
always_comb begin
        next_state = state; //default
        unlock = 1'h0; //default
        case (state)
                ST_IDLE:
                        if (num[5])   next_state = ST_5;
                ST_5:
                        if (num[4])   next_state = ST_4;
                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;
else if (num[5])
      next_state = ST_5;
    else (!num)
      next_state = Idle;
  end
 ST_4: begin
  if (num[3])
      next_state = ST_UL;
  end
  ST_UL: begin
    unlock = 'b1;
    if (e)
      next_state = ST_IDLE;
  end
endcase
```

# State Machine in Verilog

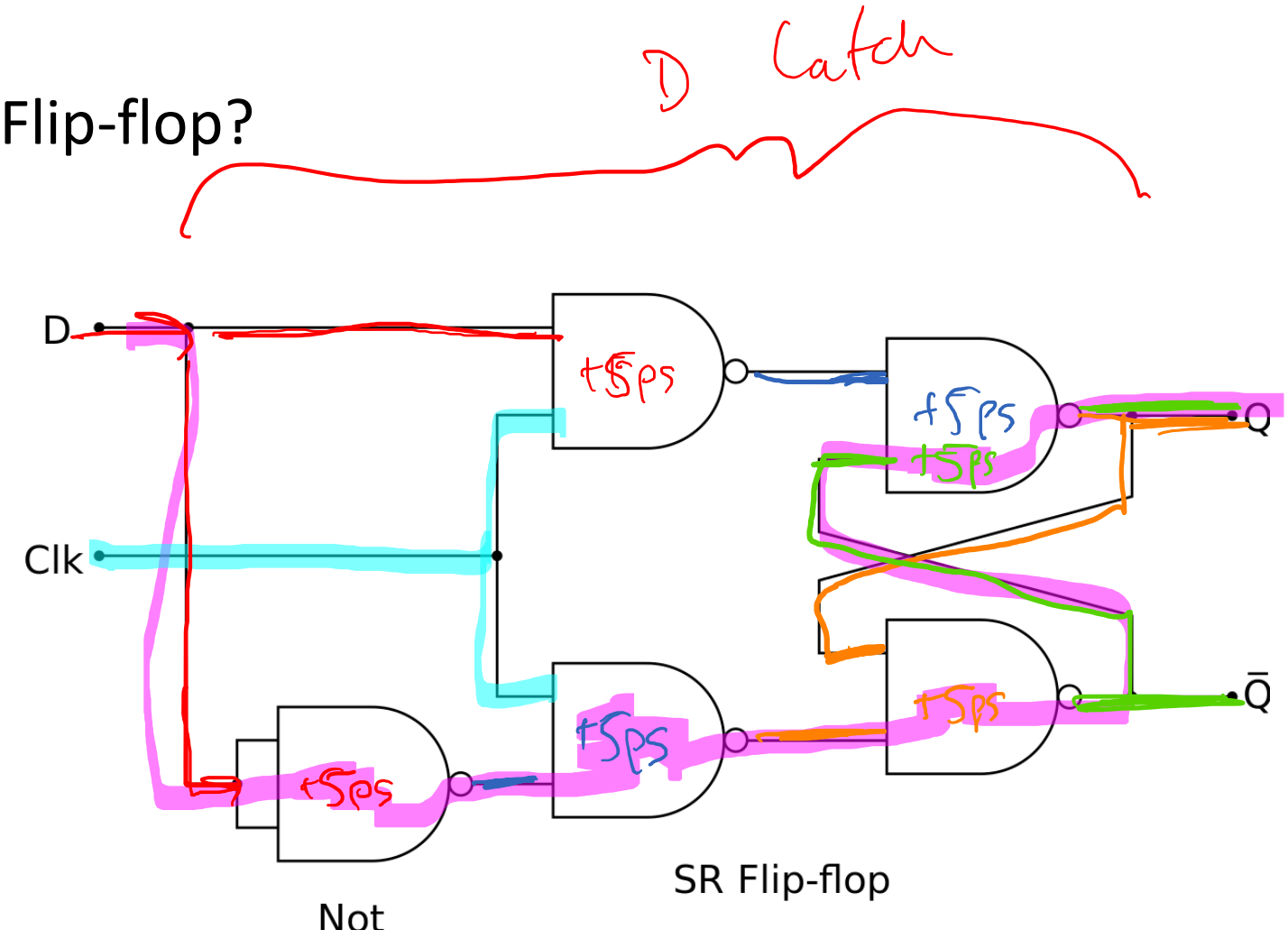



```
case (state)
  ST_IDLE:
    if (num[5])
      next_state = ST_5;
  ST_5: begin
    if (num[4])
      next_state = ST_4;
    else if (num[5])
      next_state = ST_5;
    else if ( (|num) | e ) //other btns
      next_state = ST_IDLE;
  end
 ST_4: begin
    if (num[3])
      next_state = ST_UL;
 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

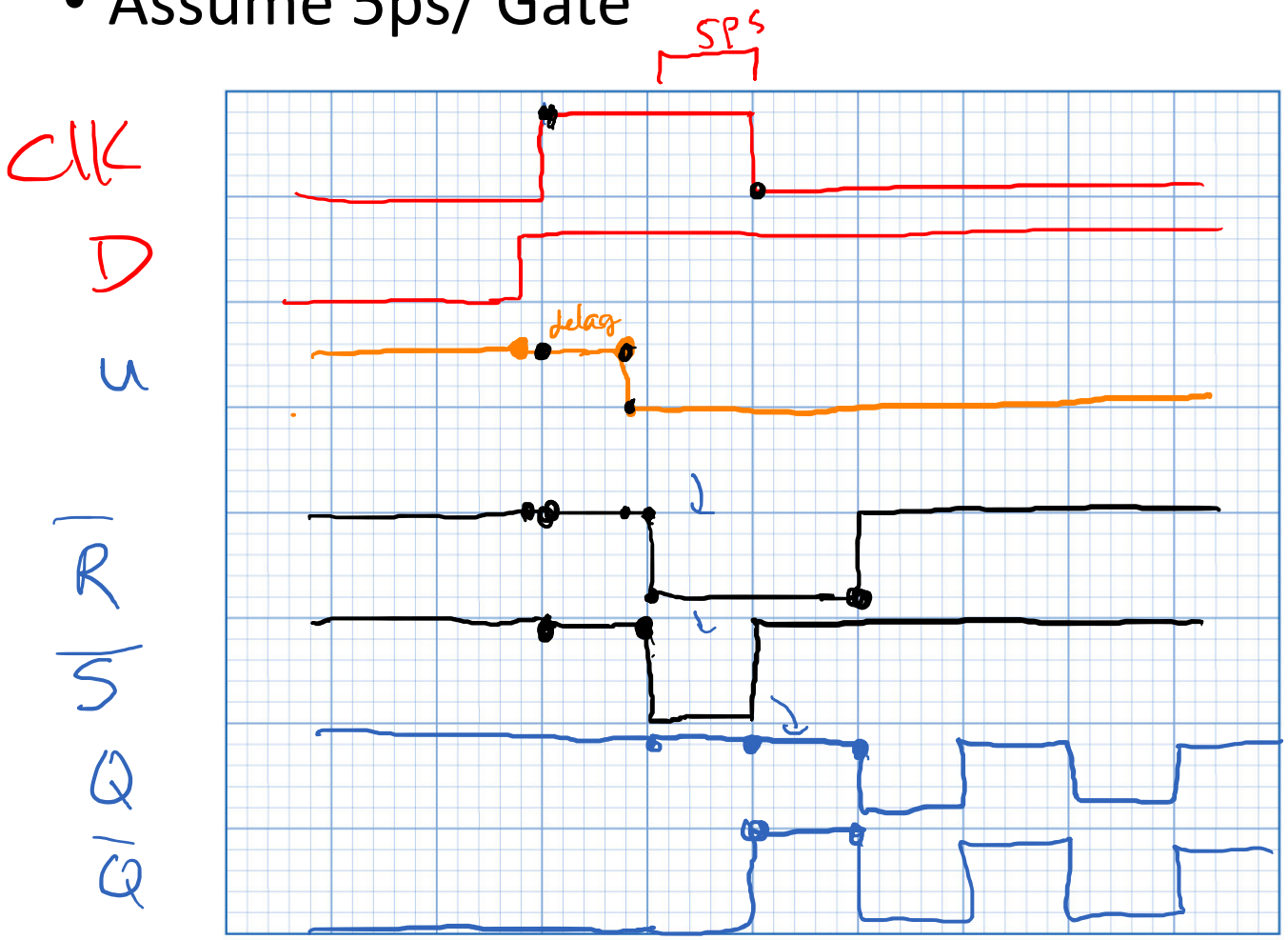
- What's the delay for this Flip-flop?
- Assume 5ps/ gate

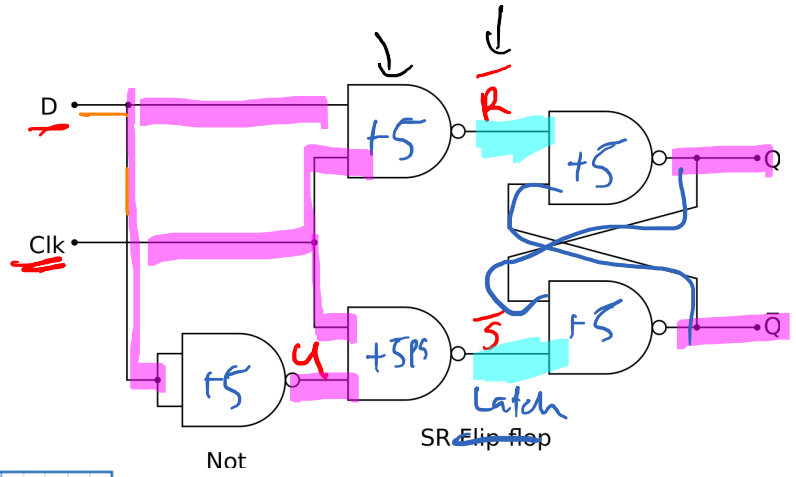



Worst-case delay = 20ps

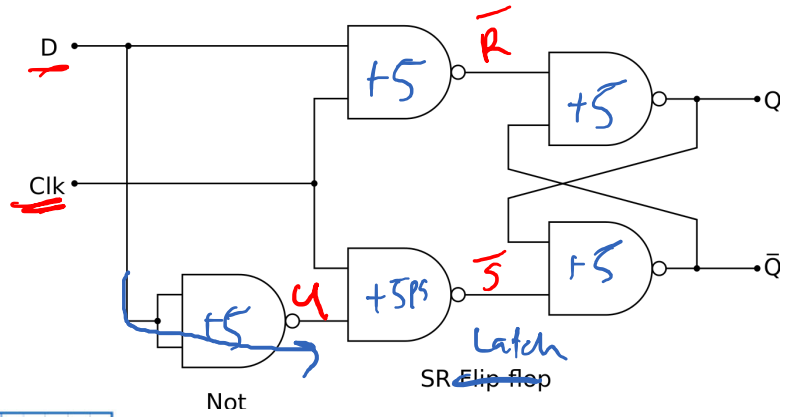
# Flip-Flop Timing

- Assume 5ps/ Gate

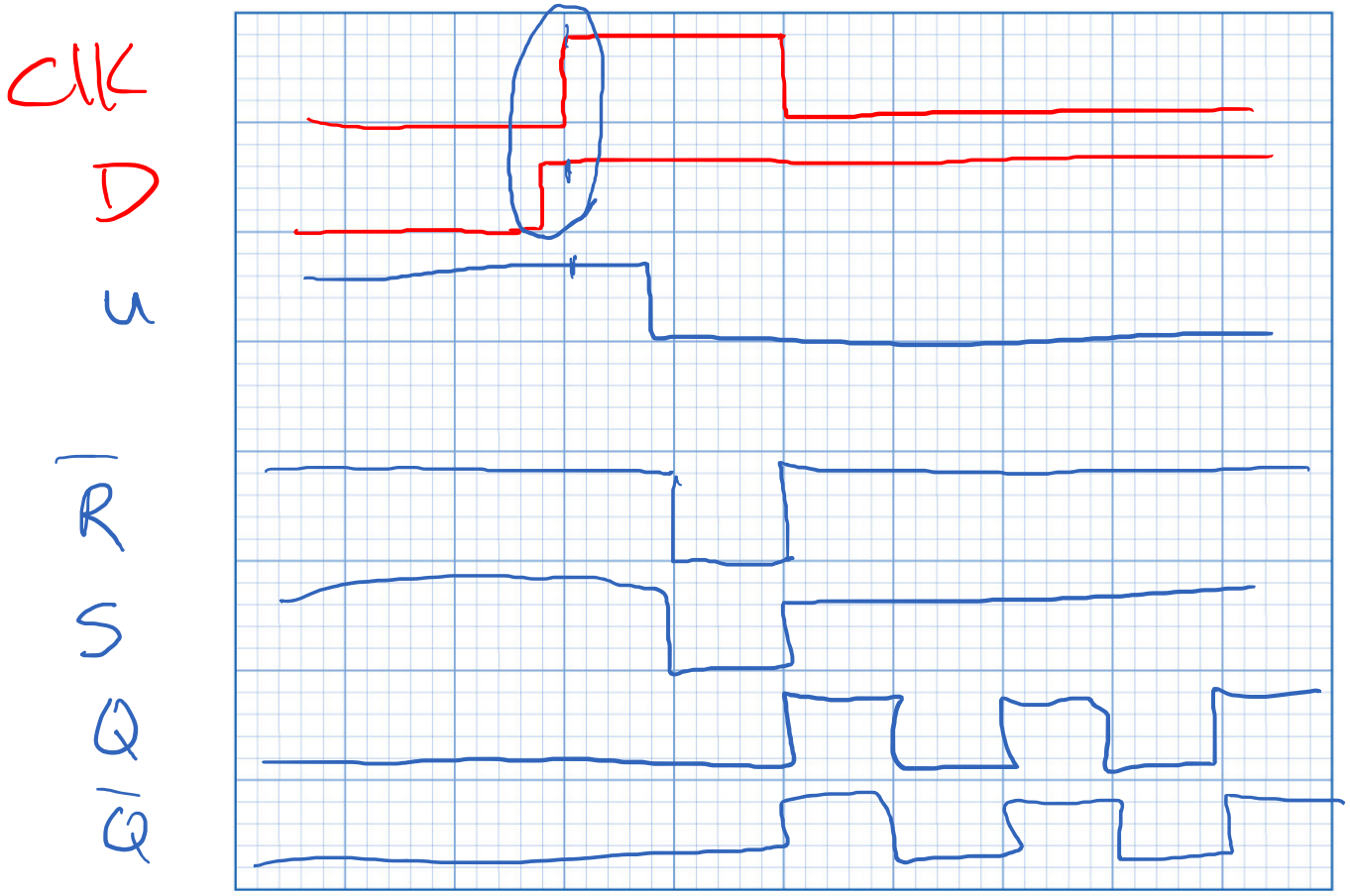


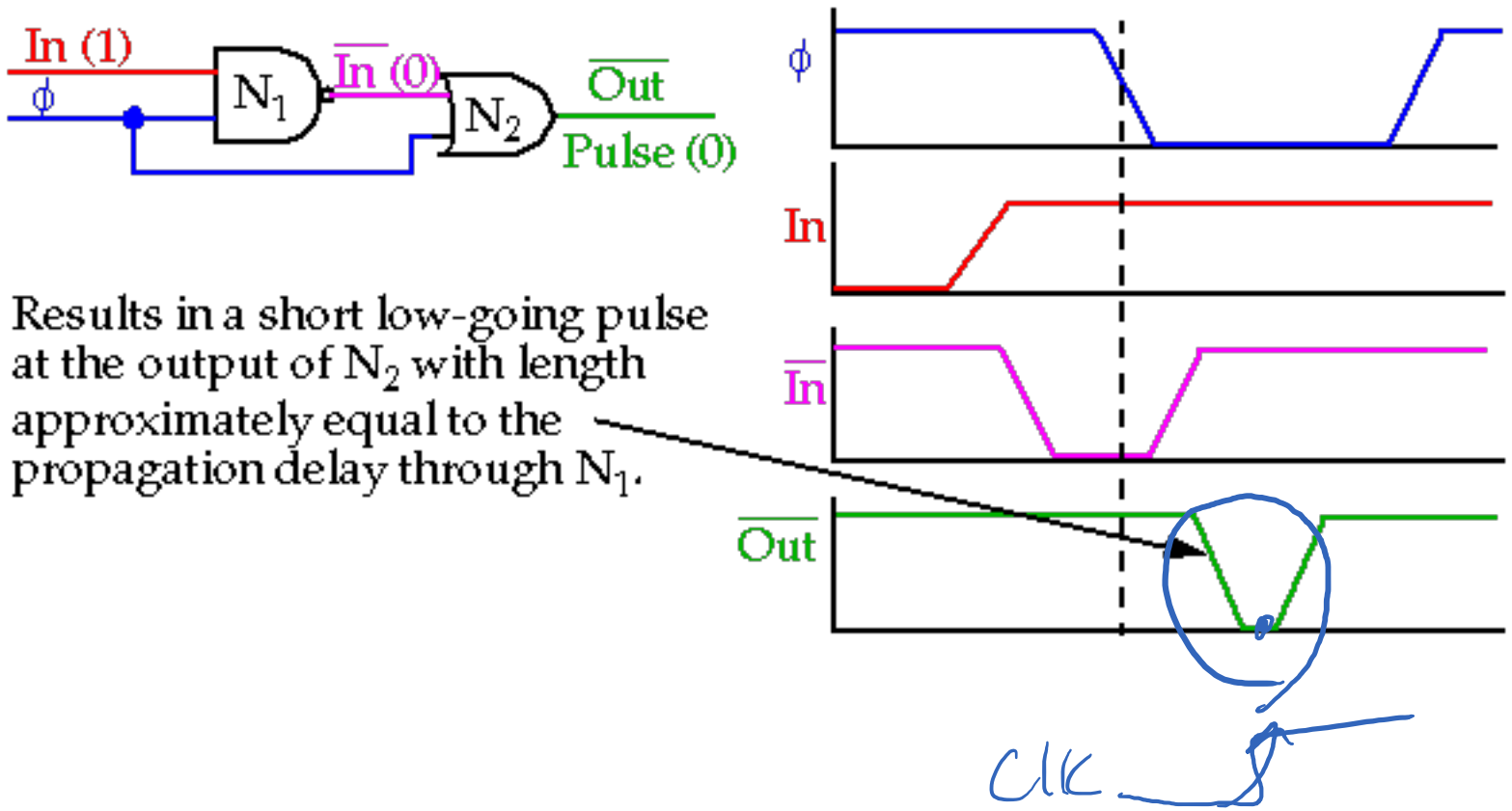
# Flip-Flop Timing




- Assume 5ps/ Gate



# Glitch

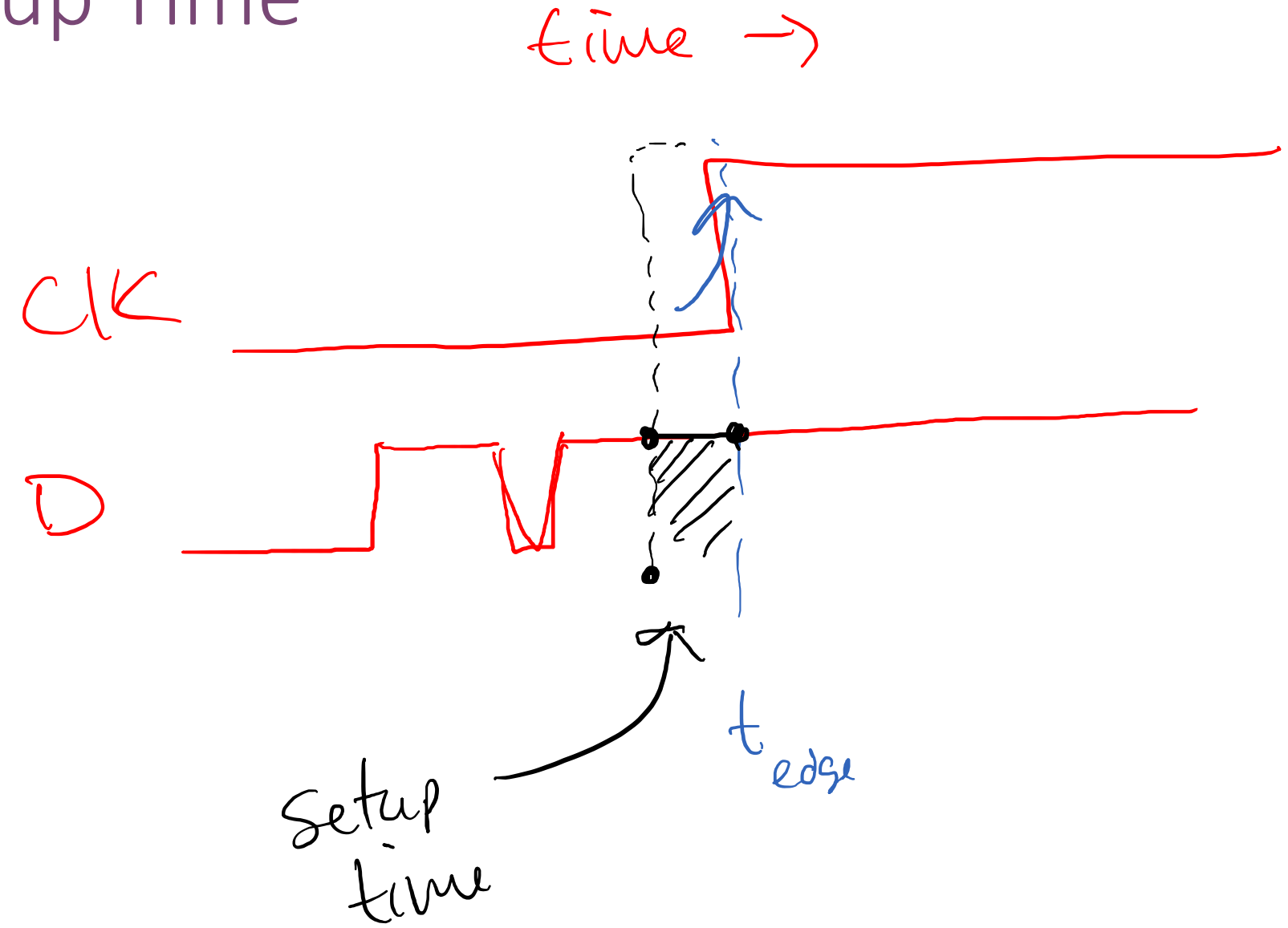

In (1)
ϕ
N1
In (0)
N2
Out
Pulse (0)
ϕ
In
In
Out
Results in a short low-going pulse at the output of N2 with length approximately equal to the propagation delay through N1.
CLK

# Setup and Hold Time

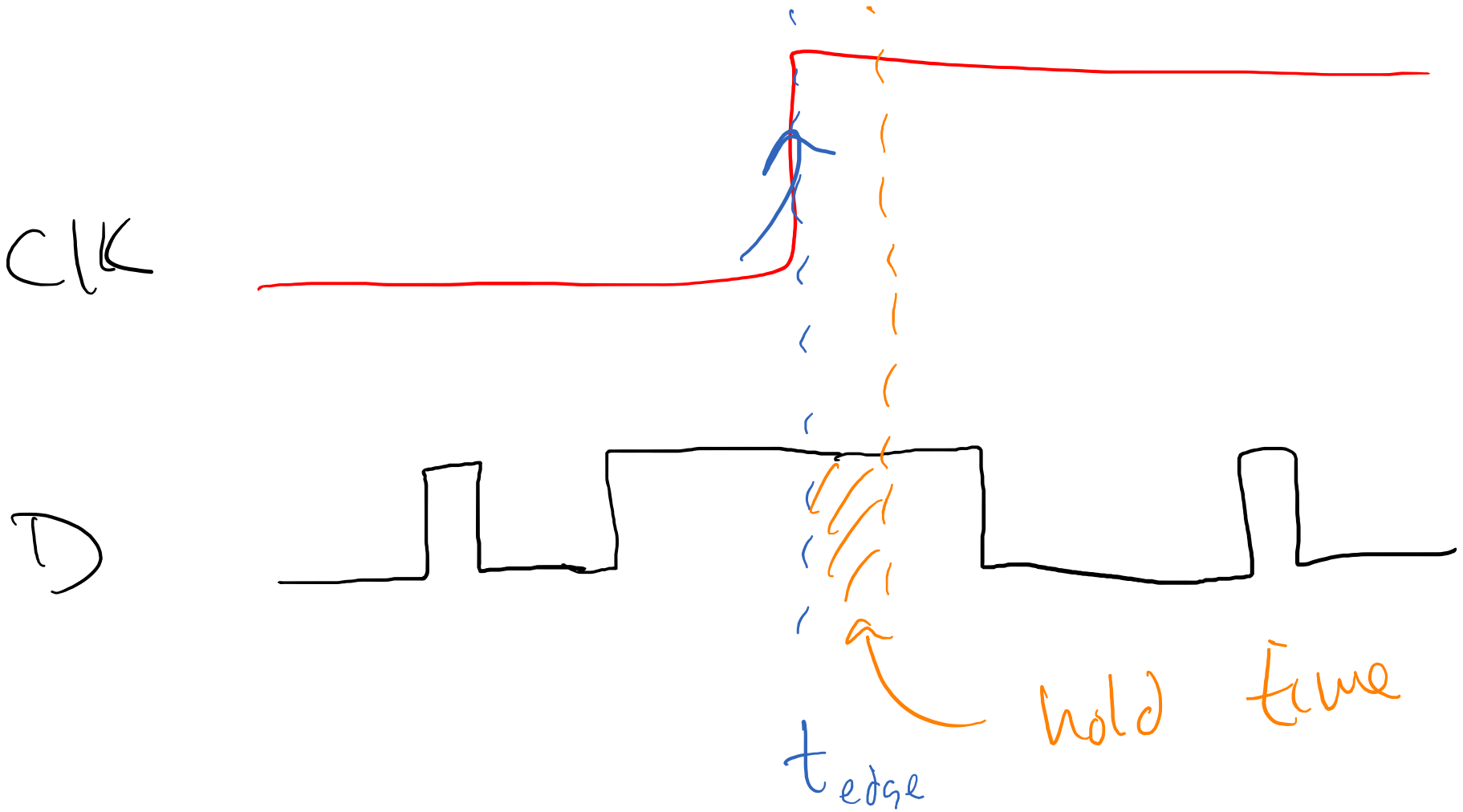
- **Setup Time**: minimum time the inputs to a flip-flop **must** be stable **before** the clock edge

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

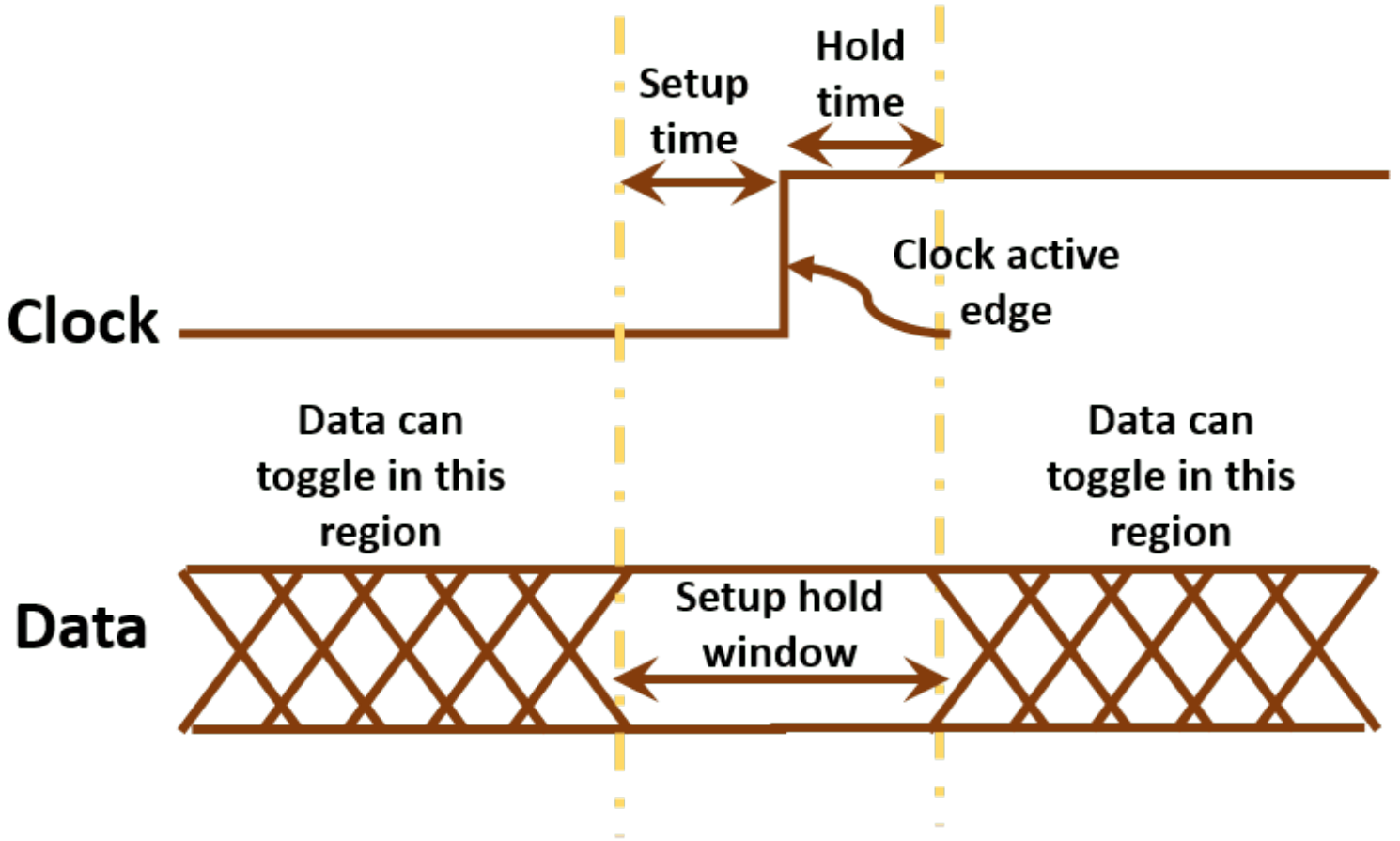
# Setup Time



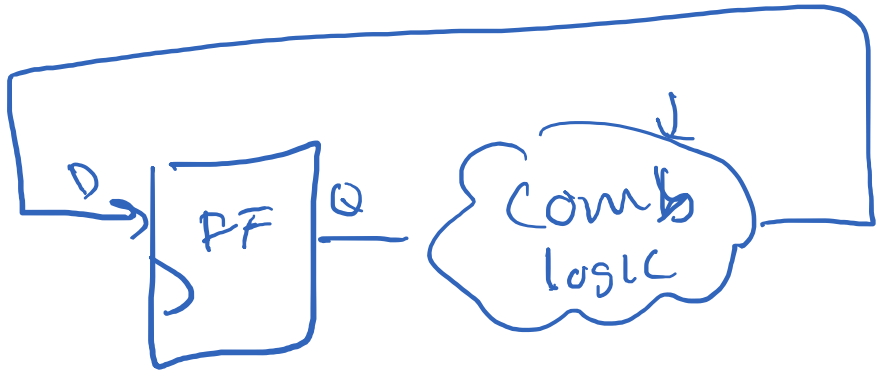
# Hold Time

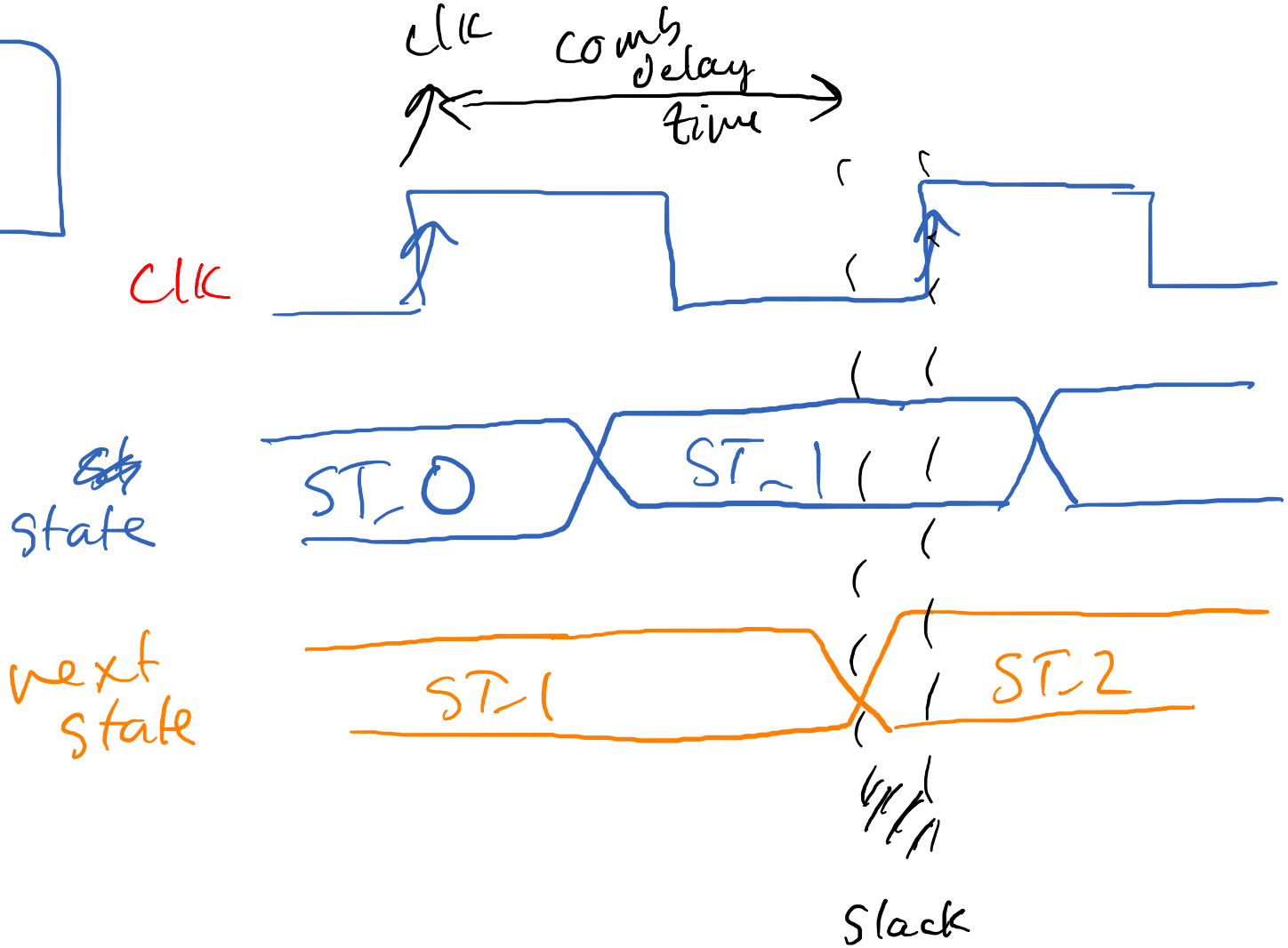


# Setup/Hold Time



# Inter Flip-Flop Timing

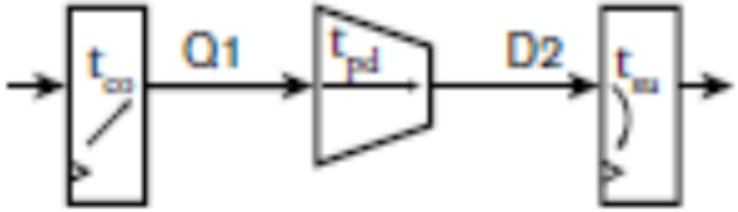


# Inter Flip-Flop Timing

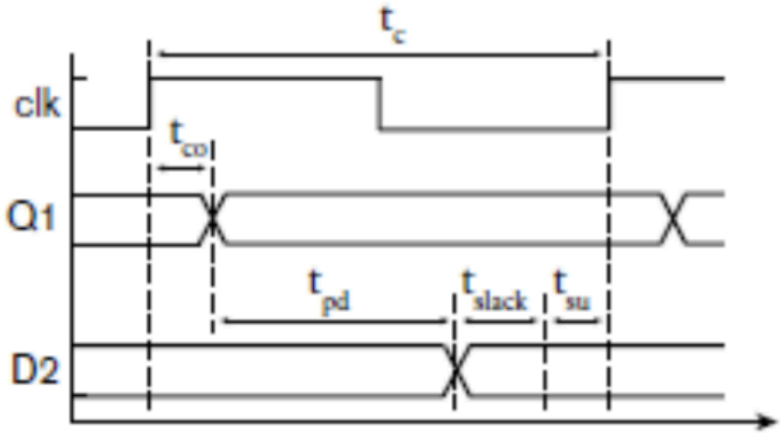
Register to register timing:

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




Delays:

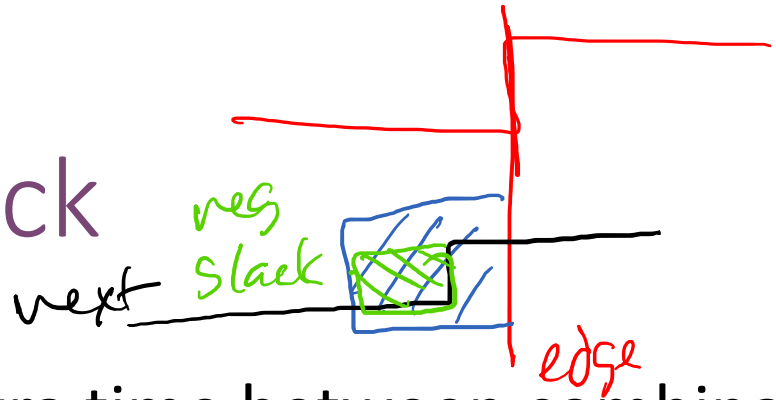
- $t_{co}$ , clock to output delay,
- $t_{pd}$ , propagation delay in combinational circuit
- $t_c$ , clock period

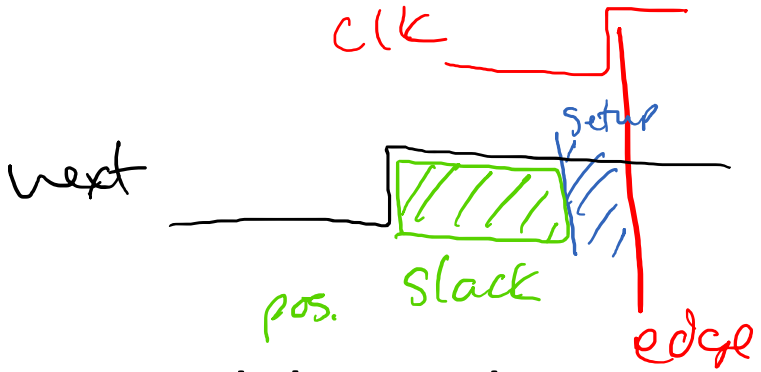



Timing requirement:

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

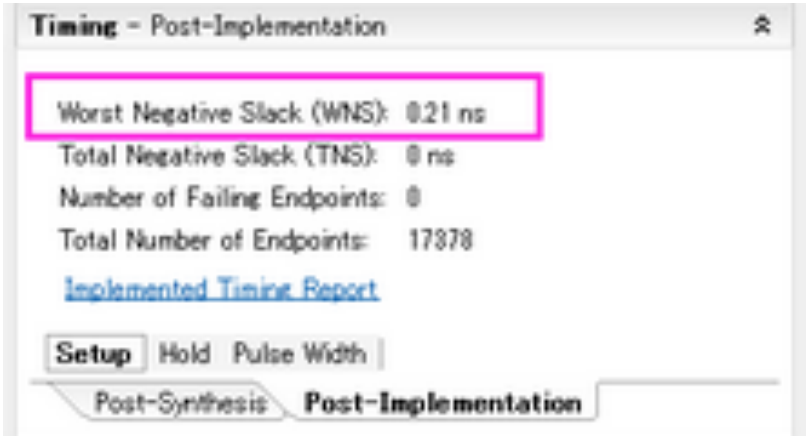
# Slack




- 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

P3 -> separate bitstream failed

# Next Time

- Memory