CSE369: Intro to Digital Design

L4: Finite State Machines



Instructor: Naomi Alterman

Teaching Assistants:

Isabel Froelich [Head TA]

Packard Stephenson

Mark Garcia

Ewan Kae

Violet Monserate

Avesta Rahimi

Administrivia

- Quiz 1 today, grades out Friday
  - Both the quiz and solutions will be added to the question bank on the course website
- Lab 5 – Verilog implementation of FSMs
  - Step up in difficulty from Labs 1-4 (worth 100 points)
  - Bonus points for minimal logic
    - Simplification through *design* (Verilog does the rest)

Outline

- **Sequential Logic in Verilog (wrap-up)**
- Finite State Machines
- FSMs in Verilog

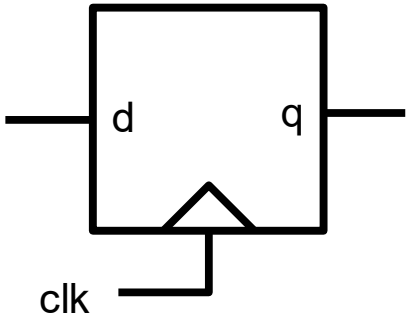
Reminder: Flip Flops

- A single bit of memory
- Copy d to q on the rising edge of the clock signal

```
module DFF (q, d, clk);
  output logic q;  // q is state-holding
  input  logic d, reset, clk;

  always_ff @(posedge clk) begin
    q <= d;
  end

endmodule
```



Verilog: Simulated Clock, three ways

- In our testbenches we need to generate a clock signal
  - Use "`forever`" block to loop testbench code for whole simulation

Explicit Edges:

```
initial begin
    clk = 0;
    forever begin
        #50  clk = 1;
        #50  clk = 0;
    end
end
```

"new thread" while (true) {

Toggle:

```
initial begin
    clk = 0;
    forever begin
        #50  clk = ~clk;
    end
end
```

compile-time constants

Parameterized clock period:

```
parameter period = 100;
initial begin
    clk = 0;
    forever begin
        #(period/2)  clk = ~clk;
    end
end
```

Verilog Testbench with Clock

```
module D_FF_testbench;
  logic CLK, reset, d;
  logic q;

  parameter PERIOD = 100;

  D_FF dut (.q, .d, .reset, .CLK); // Instantiate the D_FF

  initial CLK = 0;                 // Set up clock
  forever #(PERIOD/2) CLK = ~CLK;

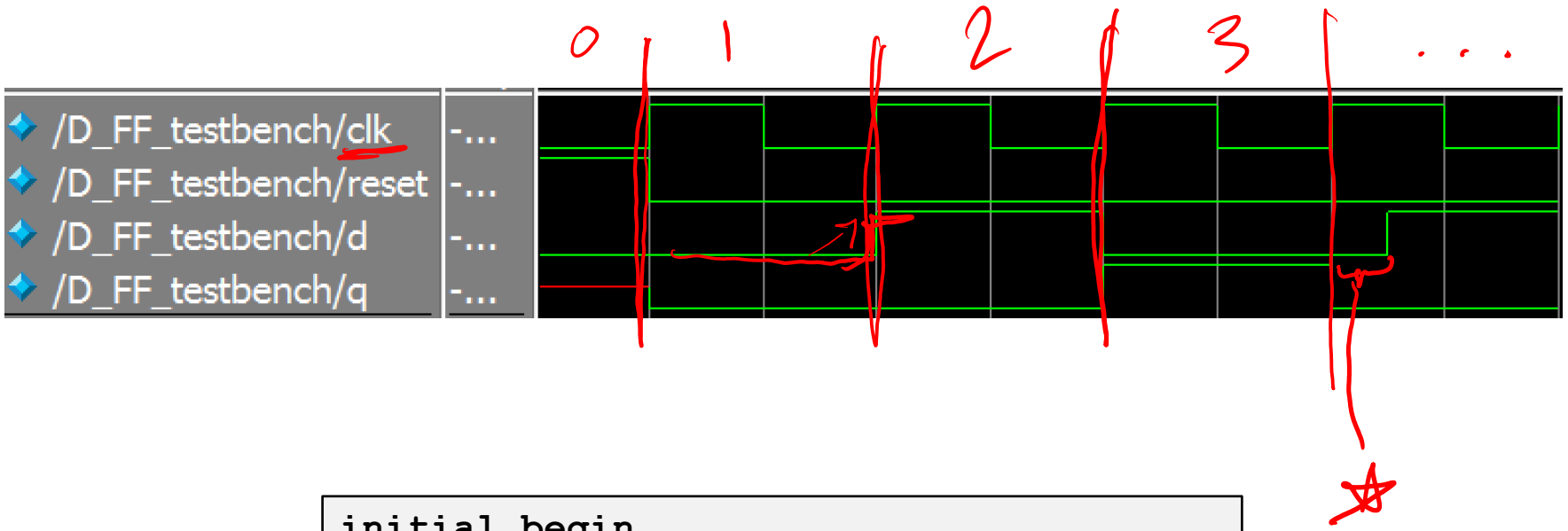
  initial begin                    // Set up signals
                    d <= 0; reset <= 1;
    @(posedge CLK);         reset <= 0;
    @(posedge CLK); d <= 1;
    @(posedge CLK); d <= 0;
    @(posedge CLK); #(PERIOD/4) d <= 1;
    @(posedge CLK);
    $stop();                      // end the simulation
  end
endmodule
```

wait for posedge

Simulation Timing Controls

- Delay: `#<time>`
  - Delays by a specific amount of simulation time
  - Can do calculations in `<time>`
  - Examples: `#(PERIOD/4),#50`
- Edge-sensitive: `@(<pos/negedge> signal)`
  - Delays next statement until specified transition on signal
  - Example: `@(posedge CLK)`
- Level-sensitive Event: `wait(<expression>)`
  - Delays next statement until `<expression>` evaluates to TRUE
  - Example: `wait(enable == 1)`

ModelSim Waveforms




```
initial begin
                        d <= 0; reset <= 1;
  @(posedge CLK);               reset <= 0;
  @(posedge CLK); d <= 1;
  @(posedge CLK); d <= 0;
  @(posedge CLK); #(PERIOD/4) d <= 1;
  @(posedge CLK);
    $stop();
end
```

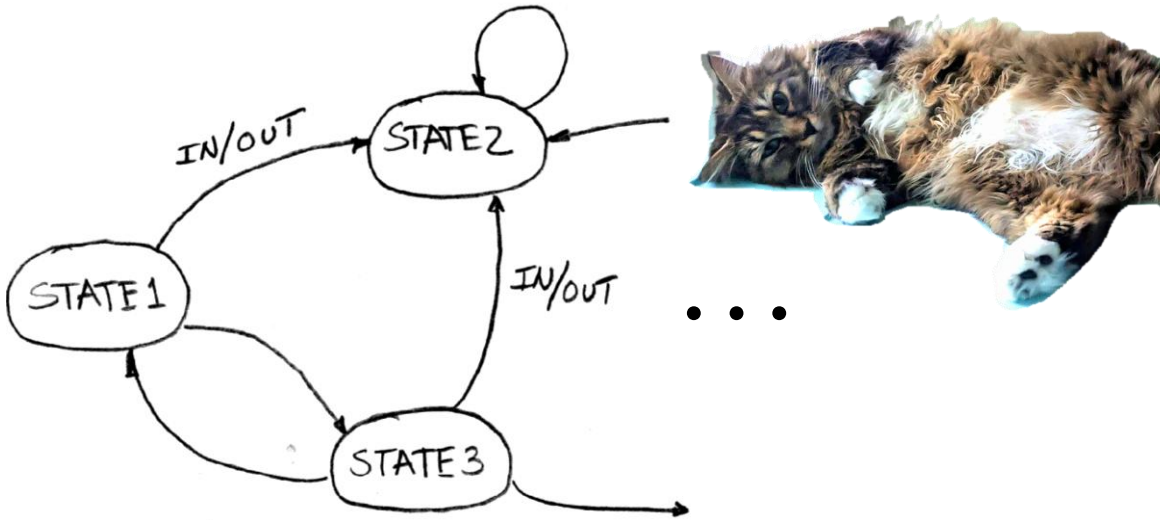
Outline

- Sequential Logic in Verilog
- **Finite State Machines**
- FSMs in Verilog

Finite State Machines (FSMs)

- A convenient way to conceptualize computation over time
  - Function can be represented with a *state transition diagram*
  - The state represents "what step we're at" in a procedure
  - The transitions represent time advancing (in units of clock cycles)
  - You've seen these before in CSE311

- **New for CSE369:** Implement FSMs in hardware as synchronous digital systems
  - Flip-flops/registers hold "state"
  - Controller (state update, I/O) implemented in combinational logic



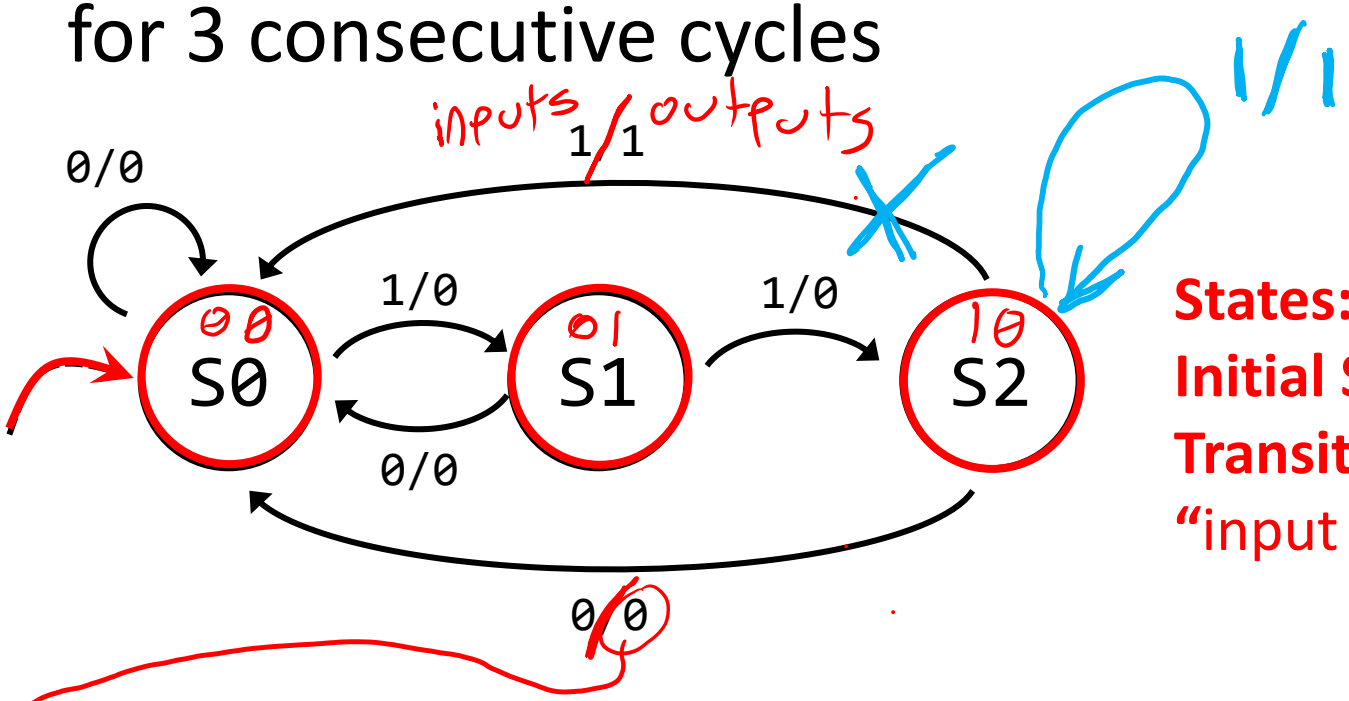
State Diagrams

- Our digital logic state diagrams are defined by:
  - A set of *states* S (circles)
  - A *transition function* that maps from the current input and current state to the output and the next state (arrows between states)
  - An *initial state* s_0 (only arrow not between states) "reset state"
- State transitions are controlled by the clock:
  - On each clock cycle the machine checks the inputs and generates a new state (could be same) and new output
- **Note:** We cover Mealy machines here; Moore machines put outputs on states, not transitions

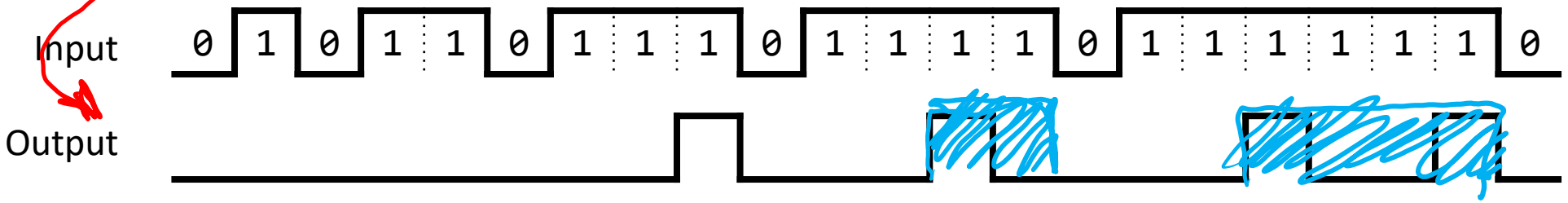
Example: Long-Press FSM



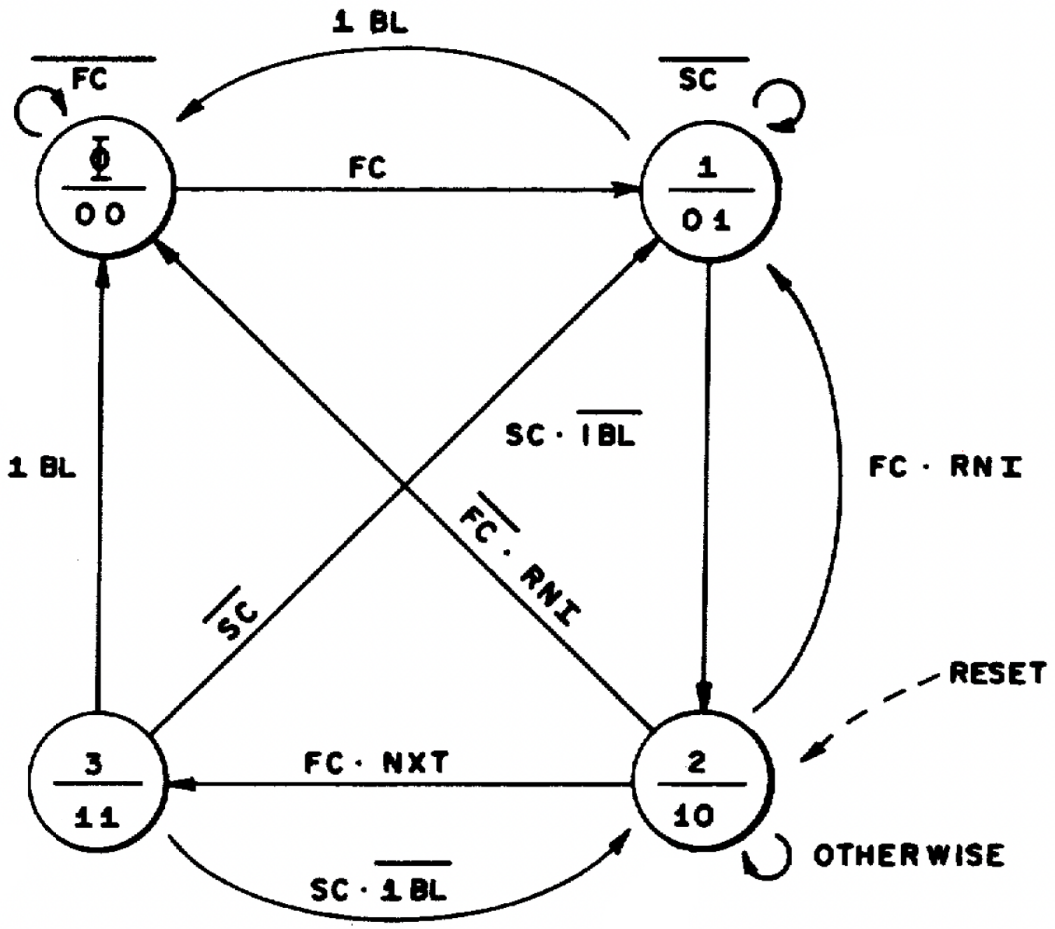
- FSM to detect if we've held a button down for 3 consecutive cycles



States: S0, S1, S2
Initial State: S0
Transition label format:
"input bits/output bits"



State diagrams, historical example!

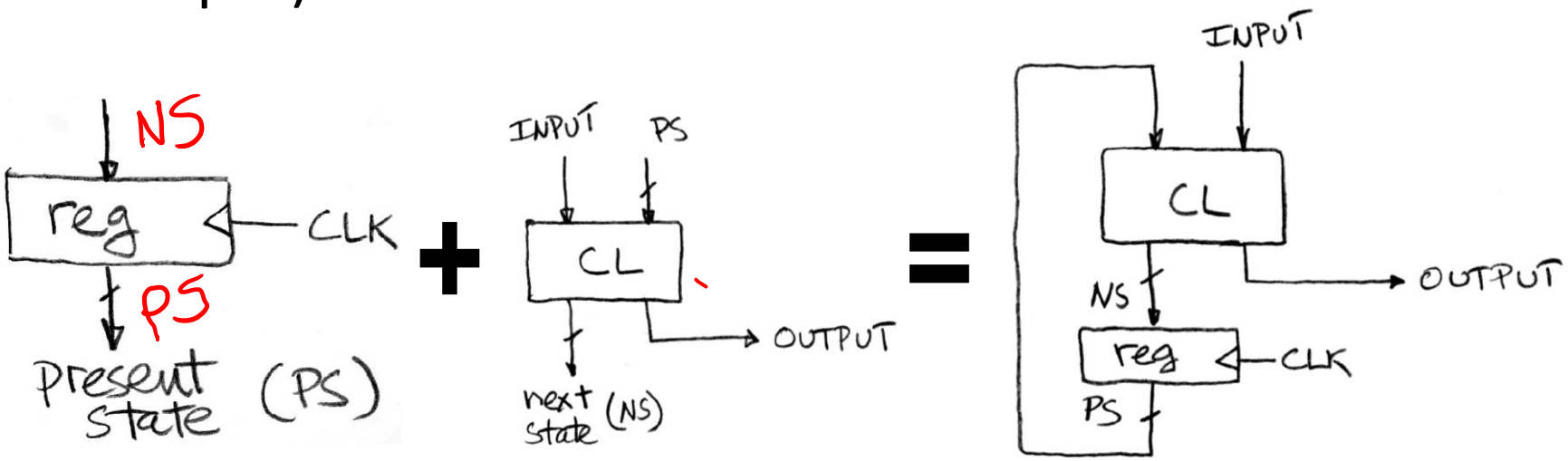


- The first x86 processor (the Intel 8086) used this state machine to know when to fetch instruction bytes from memory
- This diagram was submitted as a part of Intel's patent
- Note, this is a Moore machine (outputs on states)

https://www.righto.com/2023/01/the-8086-processors-microcode-pipeline.html

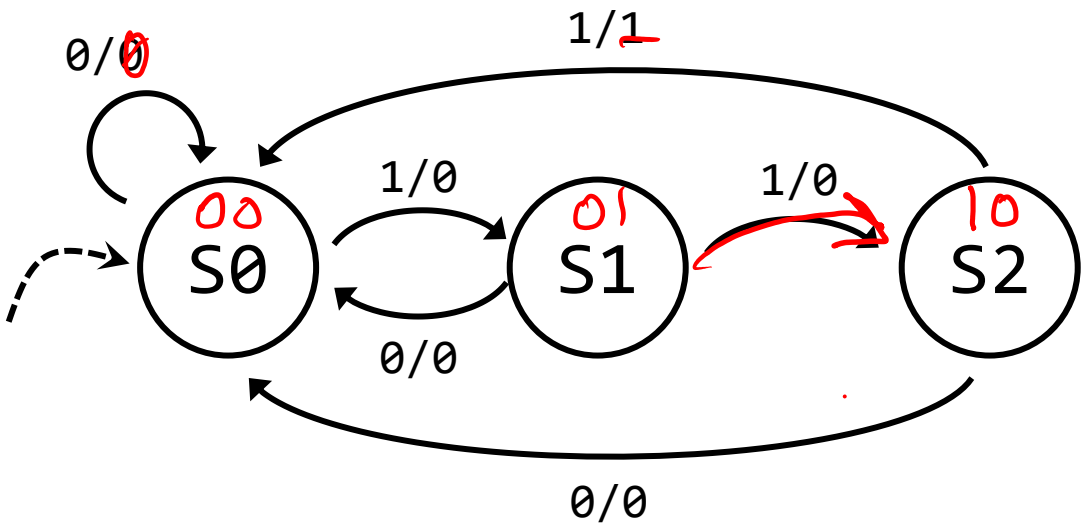
Hardware Implementation of FSM

- Register holds a representation of the FSM's state
  - Must assign a *unique* bit pattern for each state
  - Output is *present/current state* (PS/CS)
  - Input is *next state* (NS)
- Combinational Logic implements transition function (state transitions + output)



FSM: Combinational Logic

- Read off transitions into Truth Table!
  - **Inputs:** Present State (PS) and Input (In)
  - **Outputs:** Next State (NS) and Output (Out)

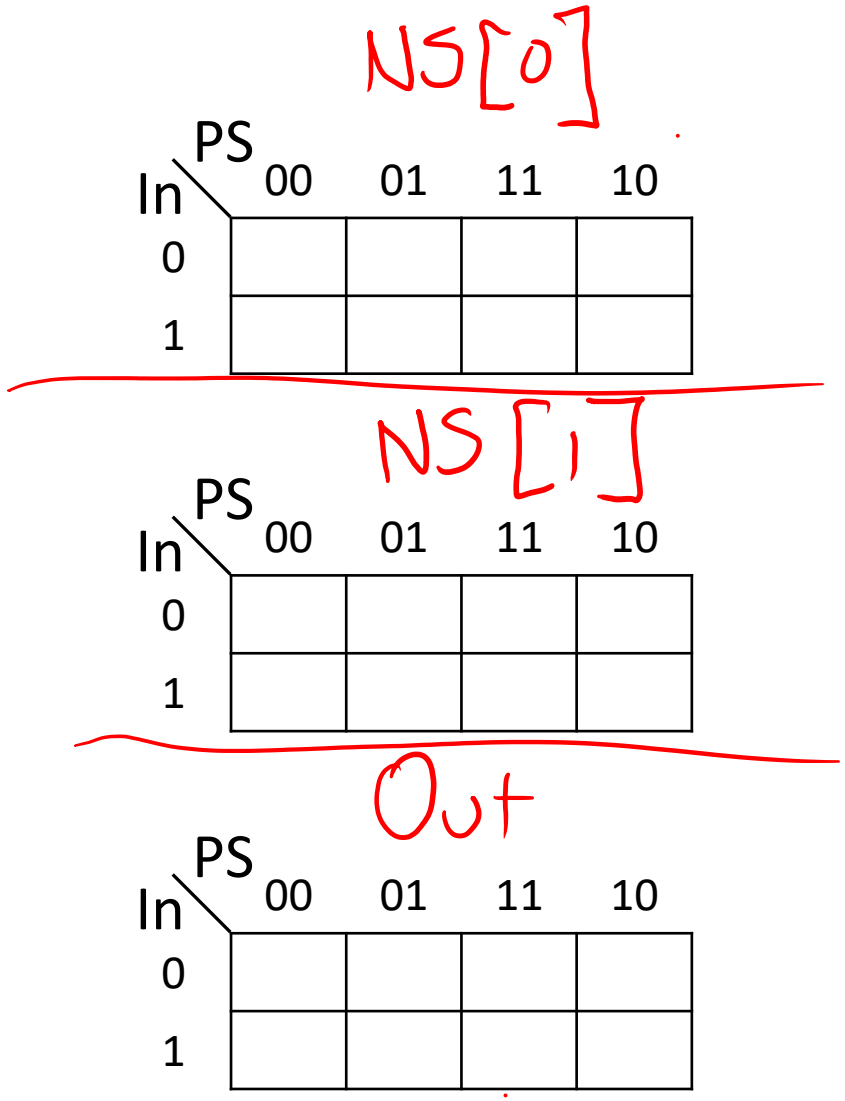


PS	In	NS	Out
0 0	0	0 0	0
0 0	1	0 1	0
0 1	0	0 0	0
0 1	1	1 0	0
1 0	0	0 0	0
1 0	1	0 0	1

- Implement logic for *EACH* output (2 for NS, 1 for Out)

FSM: Logic Simplification

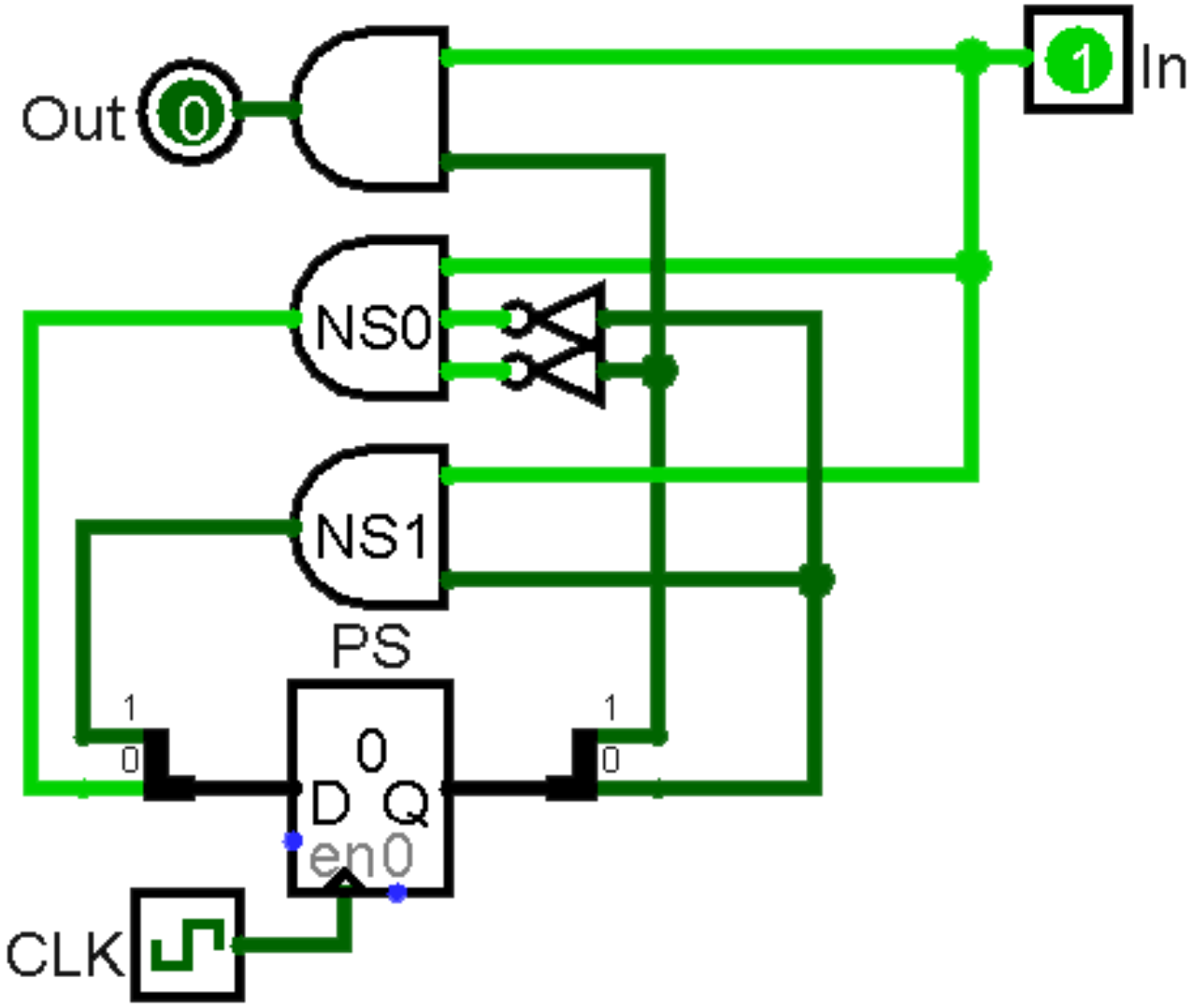
PS	In	NS	Out
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1
11	0	XX	X
11	1	XX	X



FSM: Implementation

- $NS_1 = PS_0 \cdot In$
- $NS_0 = \overline{PS_1} \cdot \overline{PS_0} \cdot In$
- $Out = PS_1 \cdot In$

- How do we test the FSM?
  - "Take" every *transition* that we care about!



State Diagram Properties

- For S states, how many state bits do we need?

Ceil(log2(S))

- For I inputs, what is the *maximum* number of transition arrows on the state diagram?

For 1 state: 2^I For S states: $S\,2^I$

  - Can sometimes combine transition arrows
  - Can sometimes omit transitions (don't cares) ← state default behavior! ("stay in state")

- For s state bits and I inputs, how many rows is the truth table?

2^{I+S}

Outline

- Sequential Logic in Verilog (wrap-up)
- Finite State Machines
- **FSM Design Example**

FSM Design Process

1) Understand the problem

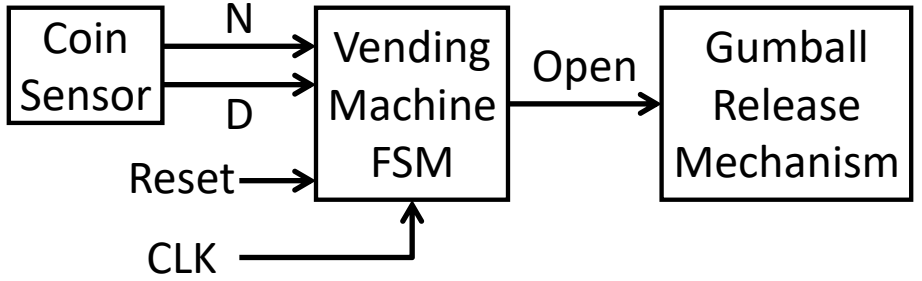
2) Draw the state diagram and a block diagram for any dependent combinational logic

3) Use state diagram to produce truth table

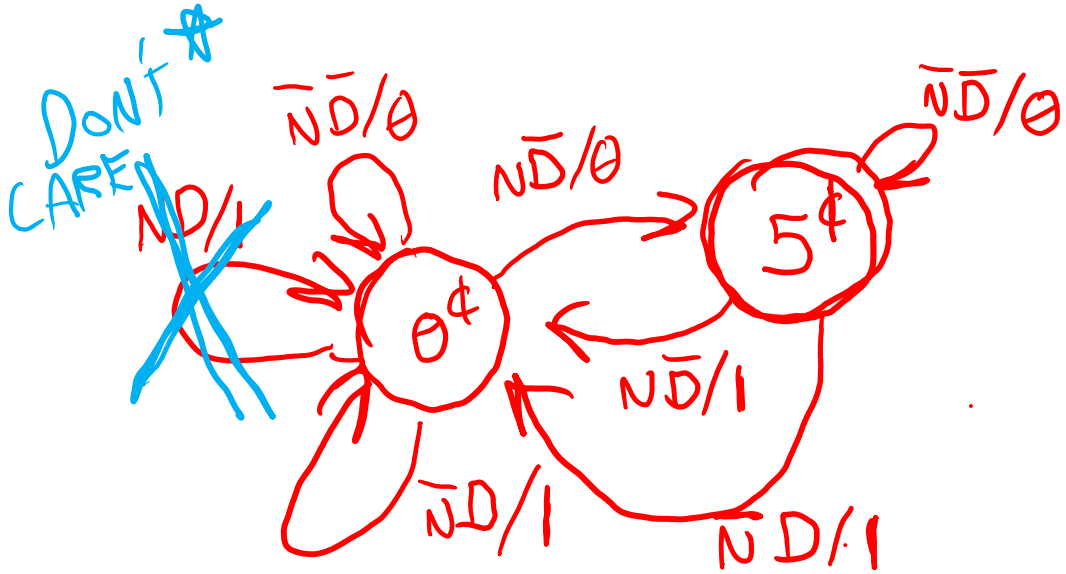
4) Use truth table to implement combinational logic

Vending Machine Example

- Vending machine description/behavior:
  - Single coin slot for dimes and nickels
  - Releases gumball after ≥ 10 cents deposited
  - Gives no change



- **State Diagram:**



ASSUMPTION: N & D never both true

Vending Machine State Table

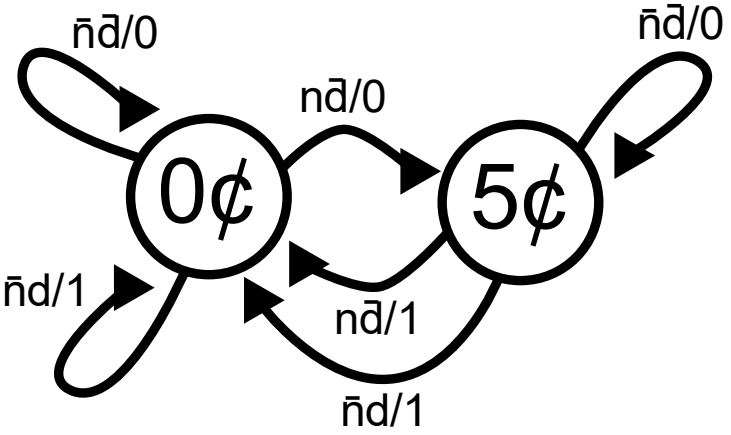
PS	N	D	NS	Open
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

PS,N

D \	00	01	11	10
0				
1				

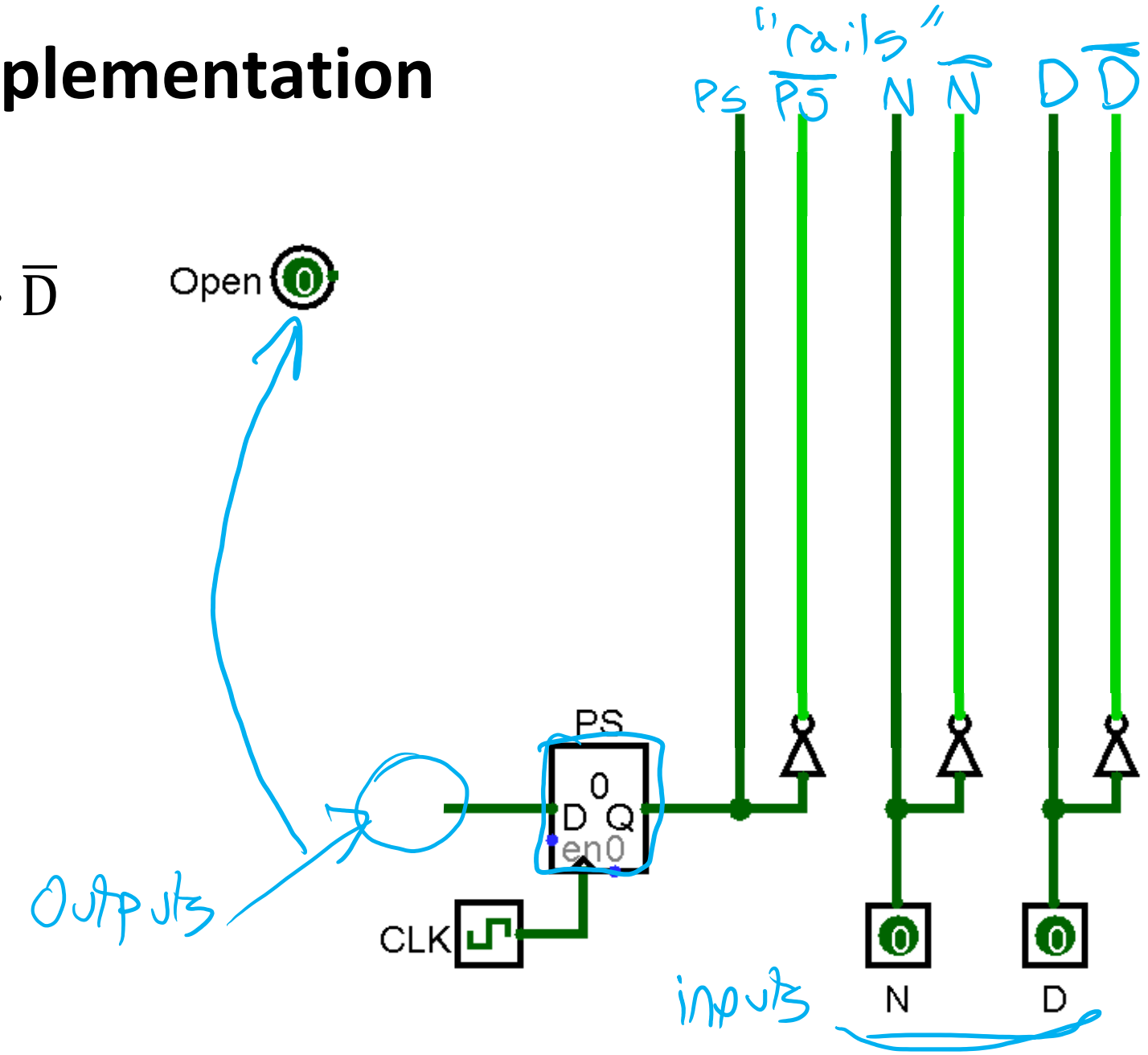
PS,N

D \	00	01	11	10
0				
1				



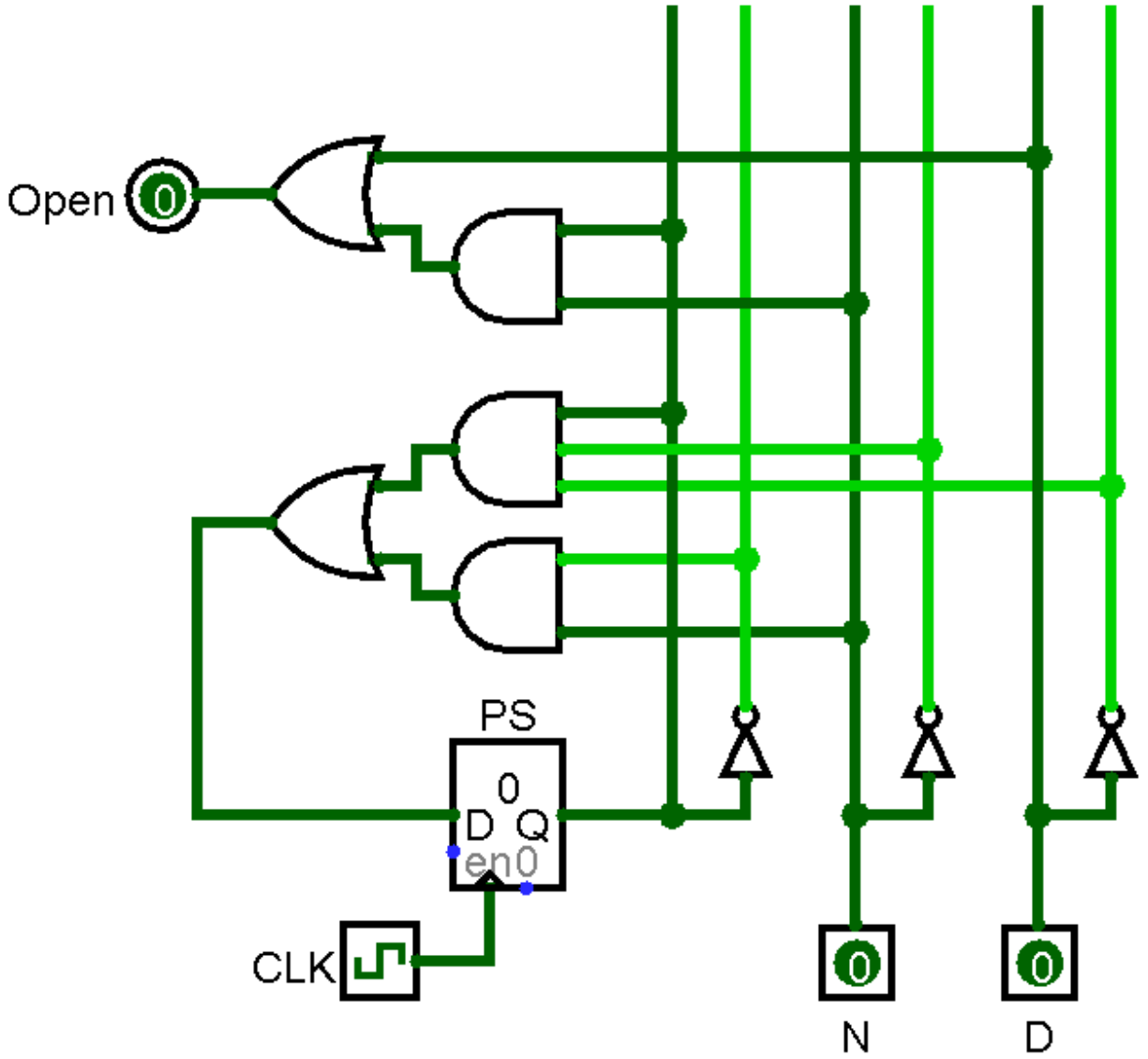
Vending Machine Implementation

- $\text{Open} = \text{D} + \text{PS} \cdot \text{N}$
- $\text{NS} = \overline{\text{PS}} \cdot \text{N} + \text{PS} \cdot \overline{\text{N}} \cdot \overline{\text{D}}$



Vending Machine Implementation

- $\text{Open} = \text{D} + \text{PS} \cdot \text{N}$
- $\text{NS} = \overline{\text{PS}} \cdot \text{N} + \text{PS} \cdot \overline{\text{N}} \cdot \overline{\text{D}}$



FSMs in Verilog (1/3) : Declarations

- Let's examine the components of the Verilog FSM example module on the next few slides

```
module vendingMachineFSM (clk, reset, n, d, open);
  input  logic clk, reset, n, d;
  output logic open;

  // State Encodings and variables
  // ps = Present State, ns = Next State
  enum logic {C0 = 1'b0, C5 = 1'b1} ps, ns;

  ...
```

FSMs in Verilog (2/3) : Combinational Logic

```
...

// Next State Logic
always_comb
  case (ps)
    C0:  if (n & ~d) ns = C5;
         else        ns = C0;
    C5:  if (n |  d) ns = C0;
         else        ns = C5;
endcase

// Output Logic – could have been in "always" block
// or part of Next State Logic.
assign open = ((ps == C0) & d) | ((ps == C5) & (n | d)) ;

...
```

FSMs in Verilog (3/3) : State

```
  ...

  // Sequential Logic (DFFs)
  always_ff @(posedge clk)
    if (reset)
      ps <= C0;
    else
      ps <= ns;

endmodule
```

Minimal amount of state in always_ff

FSM Testbench (1/2)

```
module vendingMachineFSM_tb();
  logic clk, reset, n, d;
  logic open;

  vendingMachineFSM dut (.clk, .reset, .n, .d, .open);

  // Set up the clock
  parameter CLOCK_PERIOD=100;

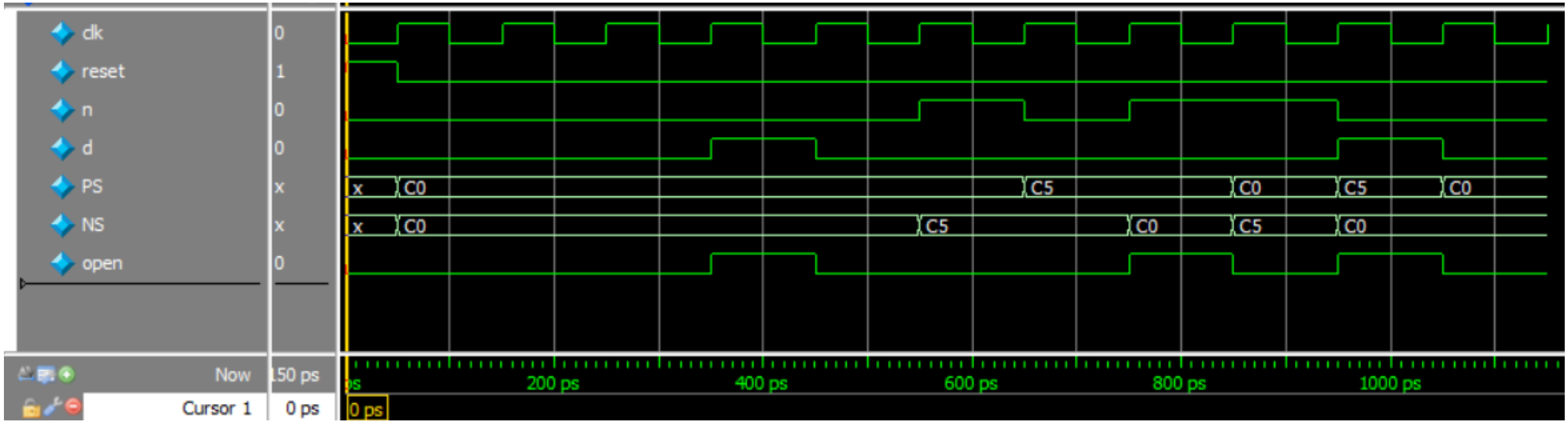
  initial begin
     clk <= 0;
    forever #(CLOCK_PERIOD/2) clk <= ~clk;
  end

  ...
```

FSM Testbench (2/2)

```
  // Set up the inputs to the design (each line is a clock cycle)
initial begin
    reset <= 1; n <= 0; d <= 0; @(posedge clk);
                  reset <= 0; @(posedge clk);
                              @(posedge clk);
                              @(posedge clk);
                      d <= 1; @(posedge clk);
                      d <= 0; @(posedge clk);
                      n <= 1; @(posedge clk);
                      n <= 0; @(posedge clk);
                      n <= 1; @(posedge clk);
                              @(posedge clk);
                n <= 0; d<=1; @(posedge clk);
                        d<=0; @(posedge clk);
    $stop;  // End the simulation
end
```

Testbench Waveforms



- What is the min # of clock cycles to *completely* test this FSM? 😳

