Review Problem 27

- Draw the circuit diagram for this code

```verilog
module foo (clk, reset, in, out);
    input logic clk, reset, in;
    output logic out;
    enum { A=0, B=1 } ps, ns;

    always_comb begin
        case (ps)
            A: ns = B;
            B: if (in) ns = B;
            else ns = A;
        endcase
        out = ~ps;
    end

    always_ff @(posedge clk) begin
        if (reset) ps <= A;
        else ps <= ns;
    end
endmodule
```
Subdividing FSMs

- Some problems best solved with multiple pieces

- Psychic Tester:
  - Machine generates pattern of 4 values (on or off)
  - If user guesses 8 patterns in a row, they’re psychic

- States?

\[ 2^4 = 16 \text{ patterns} \]

\[ 0.7 = \text{guesses} \]

\[ 128 \text{ states} + 1 \text{ security} \]
Subdividing FSMs (cont.)

- Pieces?

- Count
  - Guesses (0...8)
  - 9 states

- Pick A Pattern
  - (16 state FSM)

- Check the Guess
  - Correct?

- User input
  - S3, S2, S1, S0

- Go back to Call Helicopter.
= vs. \(\leq\)

- = ("Blocking") assign immediately
- \(\leq\) ("Non-Blocking") first eval all righthand sides, then do all assignments simultaneously.

```verilog
module swap1();
    ...
    logic [3:0] val0, val1;
    always_ff @(posedge clk) begin
        if (swap) begin
            val0 = val1;
            val1 = val0;
        end
        out = val1;
    end
endmodule

module swap2();
    ...
    logic [3:0] val0, val1;
    always_ff @(posedge clk) begin
        if (swap) begin
            val0 <= val1; \(\text{Note: set } val0 \text{ to } B\)
            val1 <= val0; \(\text{Note: set } val1 \text{ to } A\)
        end
        out <= val1;
    end
endmodule
```
= vs. <= in practice

- = in combinational logic: always_comb, assign
- <= in sequential: always_ff @(posedge clk)
- NEVER mix in one always block!
- Each variable written in only one always block

```verilog
// Output logic
always_comb begin
  out = (ps == A);
end

// Sequential Logic
always_ff @(posedge clk) begin
  assign out = (ps == A);
  if (reset)
    ps <= A;
  else
    ps <= ns;
end

// Next State Logic
always_comb begin
  case (ps)
    A: if (w) ns = B;
    else ns = A;
    B: if (w) ns = C;
    else ns = A;
    C: if (w) ns = C;
    else ns = A;
  endcase
end
```
Flipflop Realities 1: Gating the Clock

D flipflop

Clock Enable

Clock Enable not synchronous

glomerous period of the clock short

NEVER put a logic gate between the clock and DFF’s CLK input.