Review Problem 29

- Draw the state diagram of a machine that continuously outputs a true once at least two 0’s and at least two 1’s (in any order, not necessarily consecutively) have been seen, not including current input.
Traffic Light Controller (cont.)

- State Diagram

- Reset

- \( \overline{TS} / FR, HG \)

- \( C \rightarrow C \) / FR, HY, ST

- \( \overline{TS} / HR, FG, ST \)

- \( C + TL / FY, HR, ST \)

- \( C + TL / FY, HR, ST \)

- \( C + TL / FY, HR, ST \)

- \( C + TL / FY, HR, ST \)

- \( C + TL / FY, HR, ST \)
= vs. <=

- = ("Blocking") assign immediately
- <= ("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;
    val1 <= val0;
  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

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

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