Review Problem 23

- The following two flip-flops are subtly different, but both useful. The difference in code is shown in bold. What is the difference in their behavior?

module D_FF1 (q, d, reset, clk);
  output logic q;
  input logic d, reset, clk;
  always_ff @(posedge clk) begin
    if (reset)
      q <= 0;
    else
      q <= d;
  end
endmodule

module D_FF2 (q, d, reset, clk);
  output logic q;
  input logic d, reset, clk;
  always_ff @(posedge clk or posedge reset) begin
    if (reset)
      q <= 0;
    else
      q <= d;
  end
endmodule

Synchronous (clocked) Reset

Asynchronous Reset
Example: Odd Parity Checker

Assert output whenever have previously seen an odd # of 1's (i.e. how many have you seen NOT INCLUDING the current one)

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Output</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Output = PS
Next State = PS ⊕ In

Even: State = 0, Odd: State = 1
Finite State Machine Example (cont.)

\[ \text{NS} = \text{PS} \text{ xor Input}; \quad \text{OUT} = \text{PS} \]
State Diagrams

- Graphical diagram of FSM behavior
- States represented by circles
- Transitions (actions) represented by arrows connecting states
- Labels on Transitions give 
  `<triggering input pattern> / <outputs>`
  - Note: We cover Mealy machines here; Moore machines put outputs on states, not transitions
- Finite State Machine: State Diagram with finite number of states
State Diagram Example

- Circuit that is true every 4th cycle.
State Table

- “Truth table” for sequential circuits

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Output</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
State Table Example

- State Table for 4\textsuperscript{th} cycle circuit

<table>
<thead>
<tr>
<th>$P_{S_1}$</th>
<th>$P_{S_0}$</th>
<th>$\text{out}$</th>
<th>$N_{S_1}$</th>
<th>$N_{S_0}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$\text{out} = P_{S_1} \times P_{S_0}$

$N_{S_1} = P_{S_1} \oplus P_{S_0}$

$P_{S_0} = \overline{P_{S_0}}$
FSM Design Process

- 1. Understand the problem
- 2. Draw the state diagram
- 3. Use state diagram to produce state table
- 4. Implement the combinational control logic
Vending Machine Example

- Vending Machine:
  - Deliver package of gum after $\geq 10$ cents deposited
  - Single coin slot for dimes, nickels
  - No change returned

- State Diagram:

FSM Rules
For each state, for each legal input pattern:
1. must be a matching edge.
2. must not be more than one matching edge.