Review Problem 21

- For the buggy majority circuit below, the expected and the measured results are shown in the table. What gate is broken in this circuit?

<table>
<thead>
<tr>
<th>Signal</th>
<th>Expected</th>
<th>Measured</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
D Flip Flop w/Synchronous Reset

// D flip-flop w/synchronous reset

module D_FF (q, d, reset, clk);
    output logic q;
    input logic d, reset, clk;

    always_ff @(posedge clk) begin
        if (reset)
            q <= 0;  // On reset, set to 0
        else
            q <= d;  // Otherwise out = d
    end

endmodule
module stimulus;
  logic clk, reset, d, q;

  parameter ClockDelay = 100;

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

  initial begin // Set up the clock
    clk <= 0;
    forever #(ClockDelay/2) clk <= ~clk;
  end

  initial begin // Set up the reset signal
    d <= 0; reset <= 1; @(posedge clk);
    reset <= 0; @(posedge clk);
    d <= 1; @(posedge clk);
    d <= 0; @(posedge clk);
    @(posedge clk);
    $stop(); // end the simulation
  end

endmodule
Finite State Machines

- **Readings: 6-6.4.7**
- Need to implement circuits that remember history
  - Traffic Light controller, Sequence Lock, ...
- History will be held in flip flops
- Sequential Logic needs more complex design steps
  - State Diagram to describe behavior
  - State Table to specify functions (like Truth Table)
  - Implementation of combinational logic as controller
Finite State Machine Example

**Example: Odd Parity Checker**

Assert output whenever you 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>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>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Output = PS**

Even: State = 0, Odd: State = 1

\[
NS = PS \oplus \text{Input}
\]
Finite State Machine Example (cont.)

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

![Diagram of a finite state machine with signal waveforms and logic gates.](image)
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
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?

```verilog
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
```
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>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p_{51}) (p_{50})</td>
<td>(q_{5}) (q_{50})</td>
</tr>
<tr>
<td>0 0</td>
<td>1 1</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
</tr>
</tbody>
</table>

- \(q_{out} = p_{51}, p_{50}\)
- \(q_{51} = p_{51} \oplus p_{50}\)
- \(q_{50} = \overline{p_{50}}\)
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 >= 10 cents deposited
  - Single coin slot for dimes, nickels
  - No change returned

- State Diagram:

  FSM Rules
  - For each state, search legal input pattern
    - Pattern must match an edge
    - Pattern must not match more than one edge.