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?

```vhdl
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
```

```vhdl
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 reset**

Reset is synchronized to the clock.

**Asynchronous reset**

Reset has an effect at any time.
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
Verilog Testbench

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 // Set up the reset signal
begin
    d <= 0; reset <= 1; @(posedge clk); #2
    reset <= 0; @(posedge clk);
    d <= 1;
    @(posedge clk); #3
    d <= 0; @(posedge clk); #4
    @(posedge clk); #5
    $stop(); // end the simulation
end

endmodule
Testbench Waveforms

clk

d
reset

q
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
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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Out = PS
NS = PS ⊕ Input

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

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

![Diagram of a finite state machine with input, output, and state transitions](image-url)
State Diagrams

- Graphical diagram of FSM behavior
- States represented by circles
- Transitions (actions) represented by arrows connecting states
- Lables 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\(^{th}\) cycle circuit

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>NS</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>00</td>
</tr>
</tbody>
</table>