Assignment-3 (Y22)

Implement the following circuits in Verilog:

1. Four-Bit-Binary Counter

Problem Statement: Build a 4-bit binary counter that counts from 0 through 15, inclusive, with a period of 16. The reset input is synchronous, and should reset the counter to 0.

2. Four-Bit- Decimal Counter

Problem Statement : Build a 4-digit BCD (binary-coded decimal) counter. Each decimal digit is encoded using 4 bits: q[3:0] is the ones digit, q[7:4] is the tens digit, etc. For digits [3:1], also output an enable signal indicating when each of the upper three digits should be incremented.

3. Left/Right Rotator

Problem Statement: Build a 100-bit left/right rotator, with synchronous load and left/right enable. A rotator shifts-in the shifted-out bit from the other end of the register, unlike a shifter that discards the shifted-out bit and shifts in a zero. If enabled, a rotator rotates the bits around and does not modify/discard them.

- `load`: Loads shift register with `data[99:0]` instead of rotating.
- `ena[1:0]`: Chooses whether and which direction to rotate.
    - `2'b01` rotates right by one bit
    - `2'b10` rotates left by one bit
    - `2'b00` and `2'b11` do not rotate.
- q: The contents of the rotator.

☐ Finite State Machines

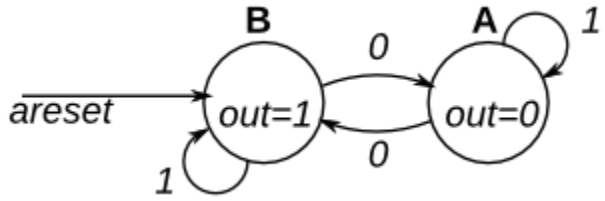
Watch the below 2 lectures on Verilog modeling of Finite State Machines and attempt the following example for better understanding and practice.

https://youtu.be/Ln7dAZtRnFY?si=g4jcrJFLP7vWIa2F

https://youtu.be/mEm4I5wNm1A?si=W80--E2XKNCWq1pq

Write a Verilog code to design Moore State Machine:

Moore state machine with two states, one input, and one output. Implement this state machine. Notice that the reset state is B.



Roll.No.220280

1)
```
module four_bit_binary_counter(
input clk,
input reset,
output reg[3:0] z
);
always @(posedge clk or posedge reset) begin
if (reset) begin
z <= 4'b0000;
end
else begin
if (z == 4'b1111) begin
z <= 4'b0000;
end
else begin
z <= z + 1;
end
end
end
endmodule
```

2)
```
module bcd_counter(
  input clk,
  input reset,
  output reg [15:0] q,
  output reg en1,
  output reg en2,
  output reg en3
);

  always@(posedge clk or posedge reset) begin
    en1<=0; en2<=0; en3=0;
    if(reset) q<=16'd0;

    if(q[3:0] == 4'd9) begin
      q[3:0] <= 4'd0;
      en1 <= 1;

      if(q[7:4] == 4'd9) begin
        q[7:4] <= 4'd0;
        en2 <= 1;

        if(q[11:8] == 4'd9) begin
          q[11:8] <= 4'd0;
          en3 <=1;
          q[15:12] = q[15:12] + 1;
        end

        else q[11:8] = q[11:8] + 1;
      end

      else q[7:4] = q[7:4] + 1;
    end

    else q[3:0] = q[3:0] + 1;
  end

endmodule
```

3)
```
module rotator (clk,reset,data,load,enable,q);
input clk,reset,load;
input [99:0] data;
input [1:0] enable;
output reg [99:0] q;

initial begin q<=100'd0;
end

always@(posedge clk or posedge reset) begin
if(reset) q<=100'd0;
else if(load) q<=data;
else begin
case(enable)
2'b01 : q<={q[0],q[99:1]};
2'b10 : q<={q[98:0],q[99]};
default : q<=q; endcase
end
end
endmodule
```

4)
```
module moore (
  input clk,
  input x,
  output reg z,
  output reg ab
);
  parameter A=0,B=1;
  always @(posedge clk)
    case (ab)
      B: begin
        z <= x ? 1 : 0;
        ab <= x ? B : A;
      end
      A: begin
        z <= x ? 1 : 0;
        ab <= x ? A : B;
      end
      default: ab <= B;
    endcase
endmodule
```