Verilog Assignment

FSM Sequence detector

<u>Design code</u>

```
module fsm(input rst, clk,x, output z


   );
   parameter A=4'h1;
   parameter B=4'h2;
   parameter C = 4'h3;
   parameter D = 4'h4;
   reg [3:0] state , next_state;
   always @(posedge clk or negedge rst)
   begin
   if(!rst)
   begin
   state <= A;
   end
   else
   state <=next_state;
   end
   always @(state or x)
   begin
   case(state)
   A: if(x==1)
   next_state<=B;
   else next_state<=A;
   B : if(x==1)
   next_state<=B;
   else next_state<=C;
   C: if(x==1)
   next_state<=D;
   else next_state<=A;
   D: if(x==1)
   next_state<=B;
   else next_state<=A;
   endcase
end
assign z=(state ==D)&&(x==0)?1:0;

endmodule
```

Testbench code

```
module tb1(

   );

  reg clk, rst, x;
  wire z;

  fsm sd( rst,clk, x, z);
  initial clk = 0;
  always #2 clk = ~clk;

  initial begin
   x = 0;
   #1 rst = 0;
   #2 rst = 1;
```

```
    #3 x = 1;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 1;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #10;
    $finish;
  end

  initial begin
    // Dump waves
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end
endmodule
```

Rising edge detector

Design Code

```
module RisingEdgeCounter (
  input wire clk,
  output reg [6:0] edge_count
);

  reg [31:0] count_duration;

  always @(posedge clk) begin
    if (count_duration < 100_000_000) begin // Assuming 1 Hz clock frequency
      edge_count <= edge_count + 1;
      count_duration <= count_duration + 1;
    end else begin
      edge_count <= 0;
      count_duration <= 0;
    end
  end

endmodule
```

Testbench code

```
module tb_RisingEdgeCounter;

  reg clk;
  wire [6:0] edge_count;

  // Instantiate the RisingEdgeCounter module
  RisingEdgeCounter uut (
    .clk(clk),
    .edge_count(edge_count)
  );
```

```
 initial begin
  clk = 0;
  forever #5 clk = ~clk; // Assuming a 1 Hz clock frequency
 end

 initial begin
  #500_000;
  $display("edge_count = %0d", edge_count);

  $stop;
 end

endmodule
```

3 bit multiplier

```
module multiplier_3bit_gate_level (
  input wire A0, A1, A2,
  input wire B0, B1, B2,
  output wire P0, P1, P2, P3, P4, P5
);

  wire and0, and1, and2, and3, and4, and5;
  wire xor0, xor1, xor2, xor3, xor4, xor5, xor6, xor7;
  assign and0 = A0 & B0;
  assign and1 = A0 & B1;
  assign and2 = A0 & B2;
  assign and3 = A1 & B0;
  assign and4 = A1 & B1;
  assign and5 = A1 & B2;
```

```
    assign xor0 = and0;

    assign xor1 = and1 ^ and3;

    assign xor2 = and2 ^ and4;

    assign xor3 = and5;

    assign xor4 = xor0 ^ xor1;

    assign xor5 = xor2 ^ xor3;

    assign xor6 = xor4 ^ xor5;

    assign xor7 = xor6;



    assign {P0, P1, P2, P3, P4, P5} = {xor0, xor1, xor2, xor3, xor4, xor7};

endmodule
```

Testbench

```
module tb_multiplier_3bit_gate_level;

    reg A0, A1, A2;

    reg B0, B1, B2;

    wire P0, P1, P2, P3, P4, P5;



    multiplier_3bit_gate_level uut (

        .A0(A0),

        .A1(A1),

        .A2(A2),

        .B0(B0),

        .B1(B1),

        .B2(B2),

        .P0(P0),

        .P1(P1),

        .P2(P2),
        .P3(P3),
        .P4(P4),
        .P5(P5)
    );

    initial begin
        $display("Testing 3-bit Multiplier (Gate-Level)");

        // Test case 1
        A0 = 1; A1 = 0; A2 = 1;
        B0 = 0; B1 = 1; B2 = 0;
        #10;
        $display("Test Case 1: A = %b %b %b, B = %b %b %b, P = %b %b %b %b %b %b", A2, A1, A0, B2,
B1, B0, P5, P4, P3, P2, P1, P0);

        // Test case 2
        A0 = 1; A1 = 0; A2 = 1;
        B0 = 1; B1 = 1; B2 = 0;
        #10;
        $display("Test Case 2: A = %b %b %b, B = %b %b %b, P = %b %b %b %b %b %b", A2, A1, A0, B2,
B1, B0, P5, P4, P3, P2, P1, P0);



        $stop;
    end

endmodule
```

4 bit even parity

Design code

```
module evparity(data, out);
  input [3:0]data;
  output out;
  assign out = (data[0]^data[1]^data[2]^data[3])?1:0;
endmodule
```

Testbench

```
module tb;
  reg [3:0]data;
  wire out;
  evparity tt (data ,out);
  initial
    begin
      data = 4'b0011;
      #10
      data = 4'b1110;

  $monitor("%b",out);
    end
endmodule
```

Counter

Design code

```
module counter (rst,clk,excite,count1,count2);
  input rst;
  input clk;
  output reg excite;
  output reg [3:0] count1;
```

```
 output reg [3:0] count2;



 initial begin
   count1 <= 4'b0000;
   count2 <= 4'b1111;
   excite <= 1'b0;
 end

 always @(posedge rst or posedge clk) begin
   if (rst) begin
     count1 <= 4'b0000;
     count2 <= 4'b1111;
     excite <= 1'b0;
   end
   else begin
     count1 <= count1 + 1;
     if (count1 == 4'b1100) begin
       excite <= 1;
     end
     else begin
       excite <= 0;
     end
   end
 end

 always @(posedge excite) begin
   count2 <= count2 - 1;
 end

endmodule
```

Testbench code

```
module top;
 reg clk, rst;
 reg excite;
 reg [3:0] count1, count2;
 integer count = 0;
 integer start_time, end_time;
 real frequency;
 counter uut (.rst(rst), .clk(clk), .excite(excite), .count1(count1), .count2(count2));

 initial begin
  clk = 0;
  forever #5 clk = ~clk;
 end

 initial begin
  rst = 1;
  #10 rst = 0;
  $dumpfile("dump.vcd"); $dumpvars;
  $monitor("count1=%b, count2=%b, excite=%b frequency=%f", count1, count2, excite, frequency);
 end

 always @(posedge excite) begin
  if (!count) begin
   start_time = $time;
   count = count + 1;
  end
  else begin
   end_time = $time;
   frequency = 1 / (($time - start_time)*1e-9) ;
```

```
      count = 0;
    end
  end
  initial #500 $finish;

endmodule
```