**Q1a) D flip flop with asynchronous preset and clear with Top level entity: -** d\_flip\_flop\_with\_async\_preset\_clear

module d\_flip\_flop\_with\_async\_preset\_clear (input d, pre, clr, clk, output reg q, q\_);

always @ (\*)

begin

if(pre == 0 && clr == 0 && clk != 0)

q <= d;

else if(pre == 1 && clr == 1)

q <= 1'bx;

else if(pre == 1)

q <= 1'b1;

else if(clr == 1)

q <= 1'b0;

end

always @ (\*)

q\_ <= ~q;

endmodule

**Test bench for D flip flop with asynchronous preset and clear: -**

module d\_flip\_flop\_With\_async\_preset\_clear\_test;

reg d, pre, clr, clk;

wire q, q\_;

integer i;

always #10 clk = ~clk;

initial begin

clk = 0;

end

d\_flip\_flop\_With\_async\_preset\_clear gate (d, pre, clr, clk, q, q\_);

initial

begin

for(i = 0; i < 8; i = i + 1)

begin

{d, pre, clr} = i;

#10;

end

$finish;

end

endmodule

**Q1b) Clocked D latch (Level triggering) with asynchronous preset and clear with Top level entity: -** clocked\_d\_latch\_with\_async\_preset\_clear

module clocked\_d\_latch\_with\_async\_preset\_clear(input d, pre, clr, clk, output reg q, q\_);

always @ (clk)

begin

if(pre == 0 && clr == 0)

q <= d;

else if(pre == 1 && clr == 1)

q <= 1'bx;

else if(pre == 1)

q <= 1'b1;

else if(clr == 1)

q <= 1'b0;

end

always @ (\*)

q\_ <= ~q;

endmodule

**Test bench for Clocked D latch (Level triggering) with asynchronous preset and clear: -**

module clocked\_d\_latch\_with\_async\_preset\_clear\_test;

reg d, pre, clr, clk;

wire q, q\_;

integer i;

clocked\_d\_latch\_with\_async\_preset\_clear gate (d, pre, clr, clk, q, q\_);

initial begin

for(i = 0; i < 16; i = i + 1)

begin

{d, pre, clr, clk} = i;

#10;

end

end

endmodule

**Q1c) 8 bit Shift Register with Top level entity: -** eight\_bit\_shift\_reg

module d\_ff (input d, clk, output reg q);

always @ (posedge clk)

begin

q <= d;

end

endmodule

module eight\_bit\_shift\_reg (input a, input clk, output [7:0] q);

wire [7:0] b;

d\_ff d0 (a, clk, b[0]);

d\_ff d1 (b[0], clk, b[1]);

d\_ff d2 (b[1], clk, b[2]);

d\_ff d3 (b[2], clk, b[3]);

d\_ff d4 (b[3], clk, b[4]);

d\_ff d5 (b[4], clk, b[5]);

d\_ff d6 (b[5], clk, b[6]);

d\_ff d7 (b[6], clk, b[7]);

assign q = b;

endmodule

**Test bench for 8 bit Shift Register: -**

module eight\_bit\_shift\_reg\_test;

reg [7:0] a;

reg in, clk;

wire [7:0] q;

always #10 clk = ~clk;

eight\_bit\_shift\_reg instance1 (in, clk, q);

initial begin

clk = 0;

a = 0;

end

integer i;

initial begin

a = 8'b10000000;

for(i = 0; i < 8; i = i + 1)

begin

in = a[i];

#10;

end

$finish;

end

endmodule

**Q1d) 8 bit Register with Top level entity: -** eight\_bit\_register

module d\_ff (input d, clk, output reg q);

always @ (posedge clk)

begin

q <= d;

end

endmodule

module eight\_bit\_register(input [7:0] a, input clk, output [7:0] q);

wire [7:0] b;

d\_ff d0 (a[0], clk, b[0]);

d\_ff d1 (a[1], clk, b[1]);

d\_ff d2 (a[2], clk, b[2]);

d\_ff d3 (a[3], clk, b[3]);

d\_ff d4 (a[4], clk, b[4]);

d\_ff d5 (a[5], clk, b[5]);

d\_ff d6 (a[6], clk, b[6]);

d\_ff d7 (a[7], clk, b[7]);

assign q = b;

endmodule

**Test bench for 8 bit Register: -**

module eight\_bit\_register\_test;

reg [7:0] a;

reg clk;

wire [7:0] q;

eight\_bit\_register instance1 (a, clk, q);

integer i;

initial

begin

for(i = 0; i < 512; i = i + 1)

begin

clk = i;

a = i;

#10;

end

end

endmodule

**Q1e) Multiplier (4×4) using full adders with Top level entity: -** four\_by\_four\_multiplier

module four\_by\_four\_multiplier (input [3:0] a, b, output reg [7:0] p);

reg [7:0] input\_a, input\_b;

integer i;

always @ (\*)

begin

input\_a = {1'b0, 1'b0, 1'b0, 1'b0, a};

input\_b = {1'b0, 1'b0, 1'b0, 1'b0, b};

for(i = 1; i < input\_b; i = i + 1)

begin

input\_a = input\_a + input\_a;

end

p = input\_a;

end

endmodule

**Test bench for Multiplier (4×4) using full adders: -**

module four\_by\_four\_multiplier\_test;

reg [3:0] a, b;

wire [7:0] p;

four\_by\_four\_multiplier instance0 (a, b, p);

integer i;

initial begin

for(i = 0; i < 5000; i = i + 1)

begin

a = $random() \* i;

b = $random() \* i;

#10;

end

end

endmodule

**Q2) The given state machine is a Mealy machine with the Top level entity: -** mealy\_machine

module mealy\_machine(input wire in,rst,clk, output reg out);

reg[1:0] p;

always @(posedge clk) begin

if (rst) begin

out = 0;

p = 2'b01;

end

else if (in == 0) begin

case (p)

2'b01: begin

out = 0;

p = 2'b10;

end

2'b10:begin

out = 0;

p = 2'b10;

end

2'b11:begin

out = 1;

p = 2'b10;

end

endcase

end

else if (in == 1) begin

case (p)

2'b01:begin

out = 0;

p = 2'b11;

end

2'b10:begin

out = 1;

p = 2'b11;

end

2'b11:begin

out = 0;

p = 2'b11;

end

endcase

end

end

endmodule

**Test bench for the mealy machine: -**

module mealy\_machine\_test;

reg in,rst,clk;

wire out;

mealy\_machine ml(in,rst,clk, out);

initial begin

clk = 0;

forever #10 clk = ~clk;

end

initial begin

rst = 1'b1;

in = 1'b0;

#50

rst = 1'b0;

in = 1'b0;

#50

in = 1'b1;

#50

in = 1'b0;

$finish;

end

endmodule