

DIGITAL LOGIC AND DESIGN

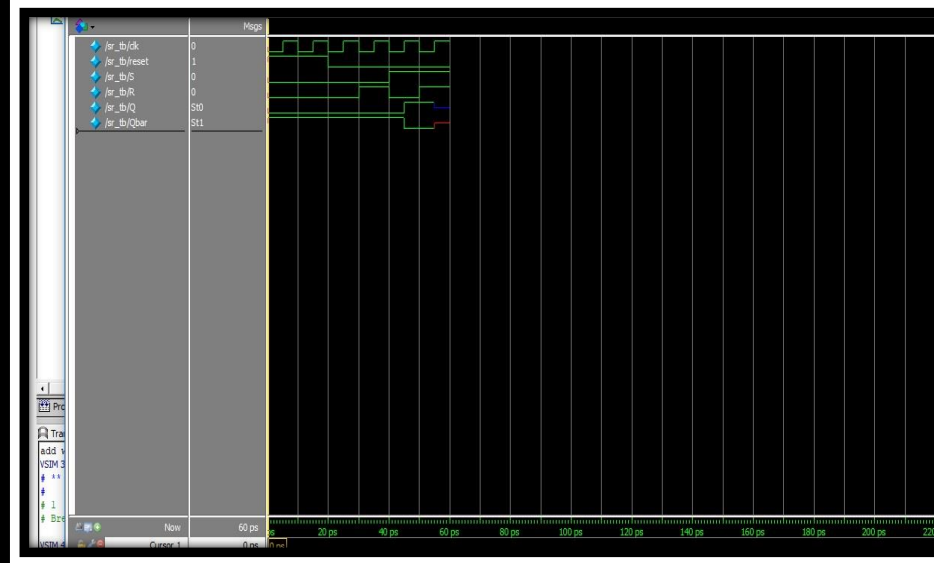
BY: SPARSH ARYA

17BEC0656

LAB-TASK 4

Aim	To write the verilog code for four types of flip flops and shift registers and verify them using testbench.
SR flip flop	
Code	<pre> module sr_ff(clk, reset,S, R, Q, Qbar); input reset, S,R, clk; output Q, Qbar; reg Q; wire Qbar; always @ (posedge clk or posedge reset) if (reset) Q<=1'b0; else if (S==1'b0 && R==1'b0) Q<=Q; else if (S==1'b0 && R==1'b1) Q<=1'b0; else if (S==1'b1 && R==1'b0) Q<=1'b1; else Q<=1'bz; assign Qbar = ~Q; endmodule </pre>
Testbench	<pre> module sr_tb(); reg clk, reset, S, R; wire Q, Qbar; sr_ff sr_inst(clk, reset, S, R, Q, Qbar); always #5 clk = ~clk; initial begin clk=1'b0; reset=1'b1; S=1'b0; R=1'b0; #20 reset = 1'b0; #10 S=1'b0; R=1'b1; #10 S=1'b1; R=1'b0; #10 S=1'b1; R=1'b1; #10 \$finish; end endmodule </pre>

Result



JK flipflop

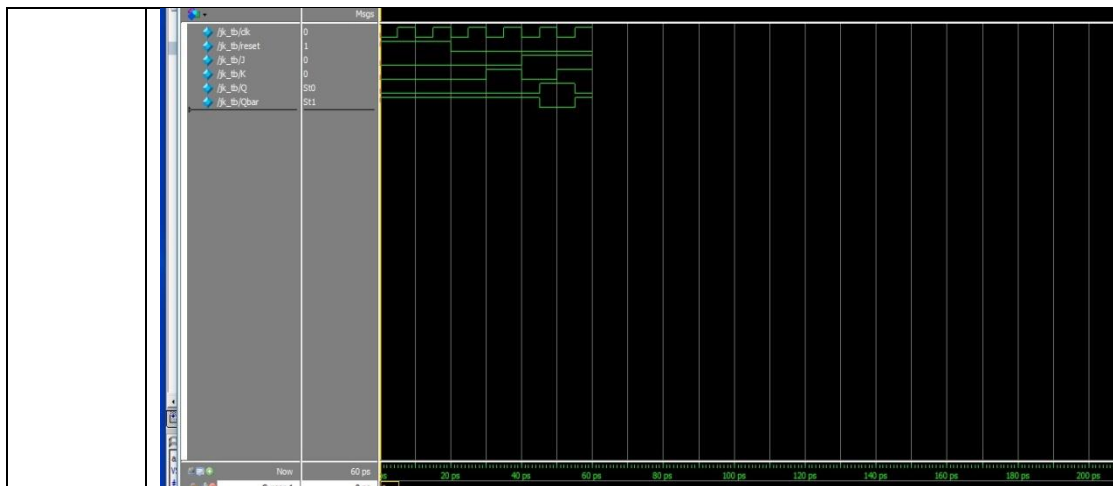
Code

```
module jk_ff(clk, reset, J, K, Q, Qbar);
    input reset, J, K, clk;
    output Q, Qbar;
    reg Q;
    wire Qbar;
    always @ (posedge clk or posedge reset)
    if (reset)
        Q<=1'b0;
    else if (J==1'b0 && K==1'b0)
        Q<=Q;
    else if (J==1'b0 && K==1'b1)
        Q<=1'b0;
    else if (J==1'b1 && K==1'b0)
        Q<=1'b1;
    else
        Q<=Qbar;

    assign Qbar = ~Q;
endmodule
```

Test bench

```
module jk_tb();
    reg clk, reset, J, K;
    wire Q, Qbar;
    jk_ff jk_inst(clk, reset, J, K, Q, Qbar);
    always
    #5 clk = ~clk;
    initial begin
        clk=1'b0; reset=1'b1; J=1'b0; K=1'b0;
        #20 reset = 1'b0;
        #10 J=1'b0; K=1'b1;
        #10 J=1'b1; K=1'b0;
        #10 J=1'b1; K=1'b1;
        #10 $finish;
    end
endmodule
```



D flipflop

Code

```

module d_ff(clk, reset, D, Q, Qbar);
    input reset, D, clk;
    output Q, Qbar;
    reg Q;
    wire Qbar;
    always @ (posedge clk or posedge reset)
        if (reset)
            Q<=1'b0;
        else
            Q<=1'bD;
    assign Qbar = ~Q;
endmodule

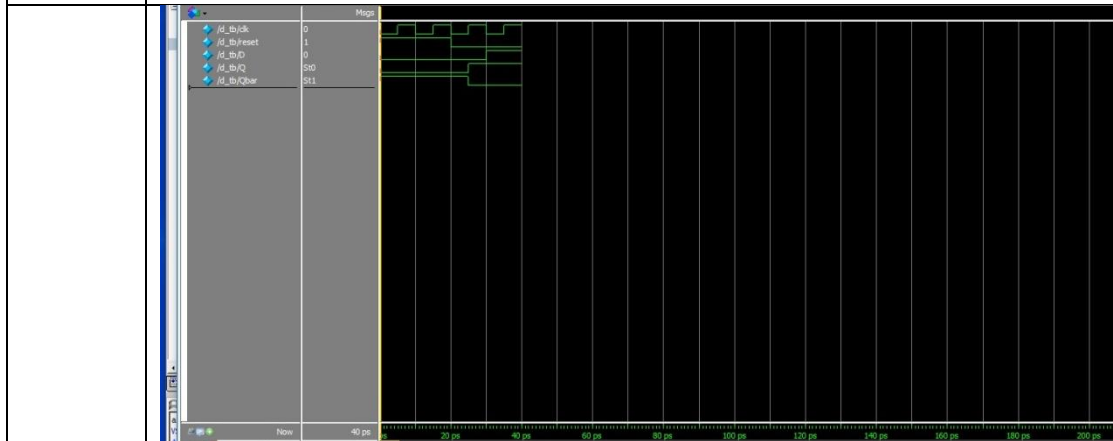
```

Test bench

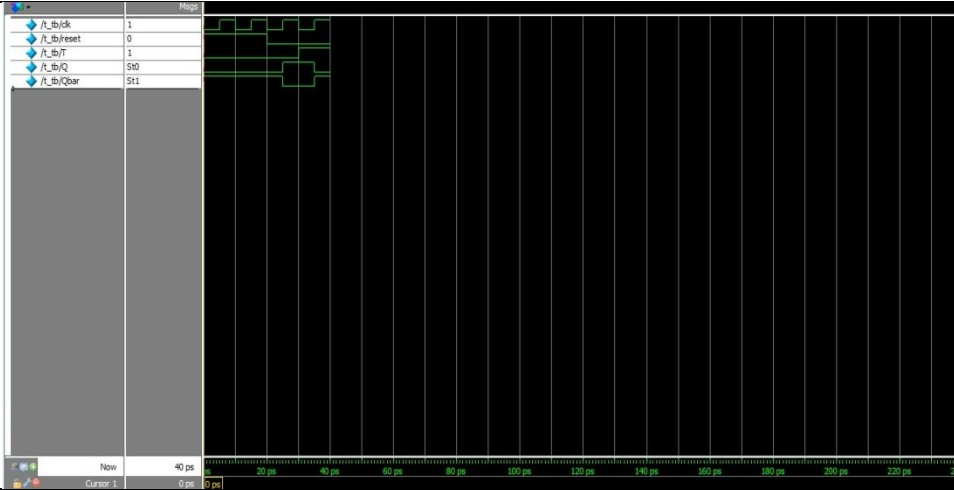
```

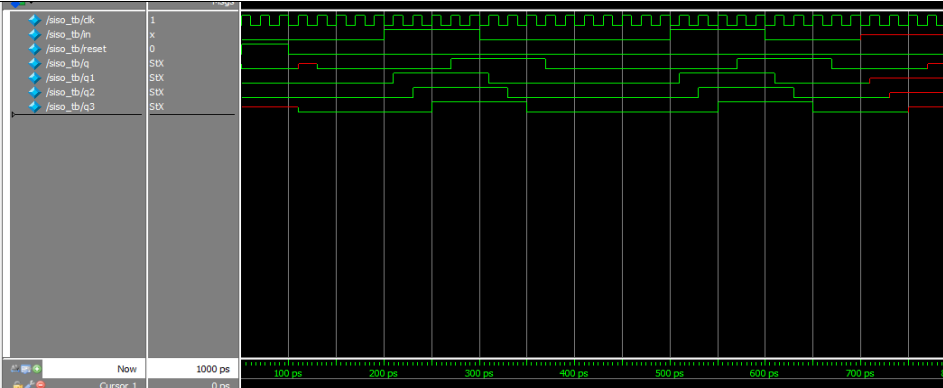
module d_tb();
    reg clk, reset, D;
    wire Q, Qbar;
    d_ff d_inst(clk, reset, D, Q, Qbar);
    always
        #5 clk = ~clk;
    initial begin
        clk=1'b0; reset=1'b1; D=1'b0;
        #20 reset = 1'b0;
        #10 D=1'b1;
        #10 $finish;
    end
endmodule

```



T flipflop

Code	<pre> module t_ff(clk, reset, T, Q, Qbar); input reset, T, clk; output Q, Qbar; reg Q; wire Qbar; always @ (posedge clk or posedge reset) if (reset) Q<=1'b0; else Q<= ~Q; assign Qbar = ~Q; endmodule </pre>
Test bench	<pre> module t_tb(); reg clk, reset, T; wire Q, Qbar; t_ff t_inst(clk, reset, T, Q, Qbar); always #5 clk = ~clk; initial begin clk=1'b0; reset=1'b1; T=1'b0; #20 reset = 1'b0; #10 T=1'b1; #10 \$finish; end endmodule </pre>
	 <p>The timing diagram illustrates the behavior of the toggle flip-flop. The clock (clk) starts at 0, goes high at 0 ps, and remains high until 20 ps. The reset signal (reset) is high from 0 to 20 ps. The T signal (T) is high from 0 to 10 ps. The output Q starts at 0, goes high at 0 ps, and then toggles at each clock edge after the reset period. The output Qbar is the complement of Q.</p>
Serial In Serial Out	
Code	<pre> module siso(clk,reset,in,q,q1,q2,q3); input in; input clk,reset; output q,q1,q2,q3; reg q,q1,q2,q3; always@(posedge clk,posedge reset) begin if(reset==1'b1) q<=1'b0; else begin q1<=in; q2<=q1; q3<=q2; q<=q3; end end </pre>

	<pre> end endmodule </pre>
Test bench	<pre> module siso_tb(); reg clk; reg in; reg reset; wire q,q1,q2,q3; siso S(.clk(clk),.reset(reset),.in(in),.q(q),.q1(q1),.q2(q2),.q3(q3)); always #10 clk=~clk; initial begin clk=0; reset=0; in=0; #50 reset = 1'b1; #50 reset = 1'b0; #100 in=1'b1; #100 in=1'b0; #100 in=1'b0; #100 in=1'b1; #100 in=1'b0; #100 in=1'bx; end initial #1000 \$stop; endmodule </pre>
	
Serial In Parallel Out	
Code	<pre> module sipo(din ,clk ,reset ,dout); output [3:0] dout ; wire [3:0] dout ; input din ; wire din ; input clk ; wire clk ; input reset ; wire reset ; reg [3:0]s; always @ (posedge (clk)) begin if (reset) s <= 0; else begin s[3] <= din; s[2] <= s[3]; s[1] <= s[2]; </pre>

	<pre> s[0] <= s[1]; end end assign dout = s; endmodule </pre>
Test bench	<pre> module sipo_tb(); reg clk, reset, din; wire [3:0]dout; initial clk=1'b0; always #10 clk=~clk; initial begin reset=1'b1; din=1'b1; #500 reset=1'b0; #100 din=1'b0; #100 din=1'b1; #100 din=1'b0; #100 din=1'b0; #100 din=1'b1; #100 din=1'b0; end initial #1300 \$stop; endmodule </pre>
Parallel In Serial Out	
Code	<pre> module piso(din ,clk ,rst ,load ,dout); output dout ; input [3:0] din ; input clk ; input rst ; input load ; reg [3:0]a; always @ (posedge (clk)) begin if (rst) a <= 1; else if (load) a <= din; </pre>

	<pre> else begin dout<=a[3]; a <= {a[2:0],1'b0}; end end endmodule </pre>
Test bench	<pre> module piso_tb(); reg clk, rst; reg [3:0] din; wire [3:0] a; wire dout; piso P(.clk(clk), .rst(rst), .din(din), .dout(dout)); initial clk=1'b1; always #10 clk=~clk; initial begin rst=1'b1; a=4'b1101; #300 rst=1'b0; #200 rst=1'b1; #200 rst=1'b0; end initial #1000 \$stop; endmodule </pre>
Parallel In Parallel Out	
Code	<pre> module pipo(clk,rst,a,q); input clk,rst; input [3:0] a; output [3:0] q; reg [3:0] q; always@(posedge clk,posedge rst) begin if (rst==1'b1) q<=4'b0000; else q<=a; end endmodule </pre>
Test bench	<pre> module pipo_tb(); reg clk; reg [3:0] in; reg reset; wire [3:0] q; pipo P(.in(in), .clk(clk), .reset(reset), .q(q)); initial clk='b1; always #10 clk=~clk; initial begin in=4'b1101; reset=1'b1; #100 reset=1'b0; #100 in=4'b1000; #100 reset=1'b1; #100 reset=1'b0; </pre>

