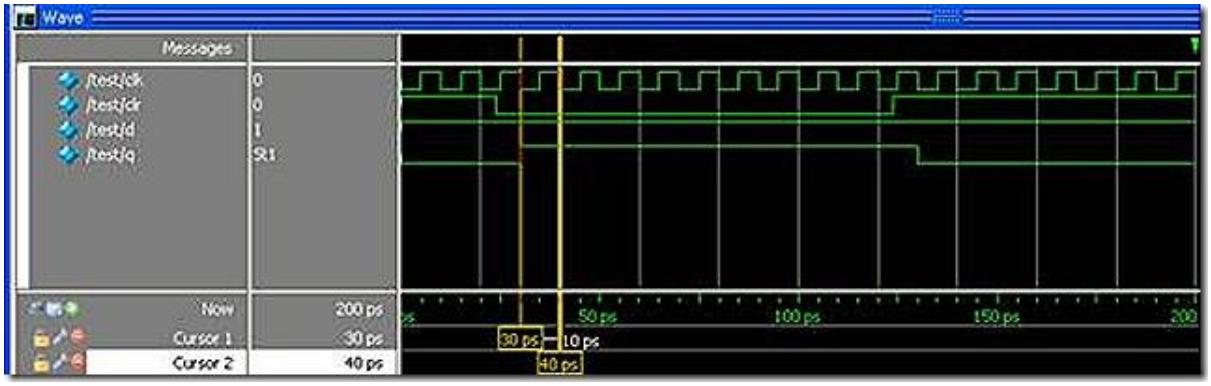
Chapter 7: (Part 2)

7. Design a negative edge-triggered D-flipflop(D_FF) with synchronous clear, active high (D_FF clears only at a negative edge of clock when clear is high). Use behavioral statements only. (Hint: Output q of D_FF must be declared as reg). Design a clock with a period of 10 units and test the D_FF.

My answer:

```
1   //ex7_7 d_ff
2   module d_ff(q,clk,clr,d);
3   output reg q;
4   input clk,clr,d;
5
6   always @(negedge clk)
7     if(clr)
8       q<=1'b0;
9     else
10      q<=d;
11
12  endmodule
```

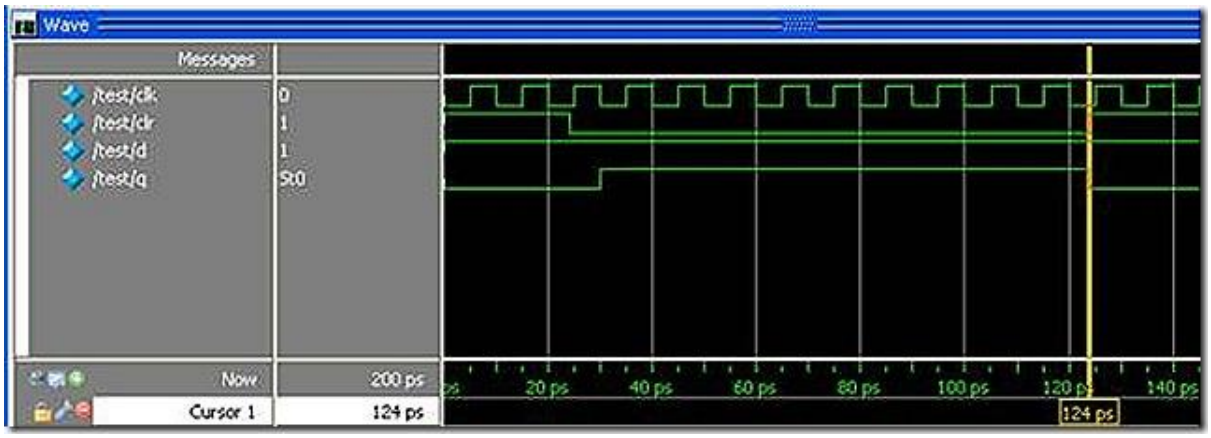
```
1   //stimulus
2   module test;
3   reg clk,clr,d;
4   wire q;
5
6   d_ff d1(.clk(clk),.clr(clr),.d(d),.q(q));
7
8   initial
9   begin
10    clk=1'b0;
11    forever #5 clk=~clk;
12  end
13
14  initial
15  begin
16    clr=1'b1; d=1'b1;
17    #24 clr=1'b0;
18    #100 clr=1'b1;
19    #120 clr=1'b0;
20  end
21
22  initial
23    #200 $finish;
24
25  endmodule
```



8. Design the D-flipflop in exercise 7 with asynchronous clear (D_FF clears whenever clear goes high. It does not wait for next negative edge). Test the D_FF.

My answer:

```
1   //ex7_8 d_ff
2   module d_ff(q,clk,clr,d);
3   output reg q;
4   input clk,clr,d;
5
6   always @(negedge clk or posedge clr)
7     if(clr)
8       q<=1'b0;
9     else
10      q<=d;
11
12  endmodule
```

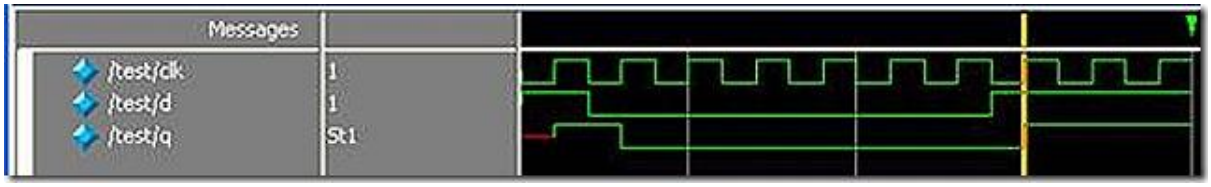


9. Using the wait statement, design a level-sensitive latch that takes clock and d as inputs and q as output. q=d whenever clock=1.

My answer:

```
1   //ex7_9 wait statement
2   module d_latch(q,d,clk);
3   output reg q;
4   input d,clk;
5
6   /*always
7     wait (clk) q=d;*/
8
9   always @(clk or d)
10    if(clk)
11      q=d;
12
13
14  endmodule
```

```
1   //stimulus
2   module test;
3   reg clk,d;
4   wire q;
5
6   d_latch d1(.q(q),.d(d),.clk(clk));
7
8   initial
9   begin
10    clk=1'b0;
11    d=1'b1;
12    #20 d=1'b0;
13    #120 d=1'b1;
14  end
15
16  always
17    #10 clk=~clk;
18
19  initial
20    #200 $finish;
21
22  endmodule
```

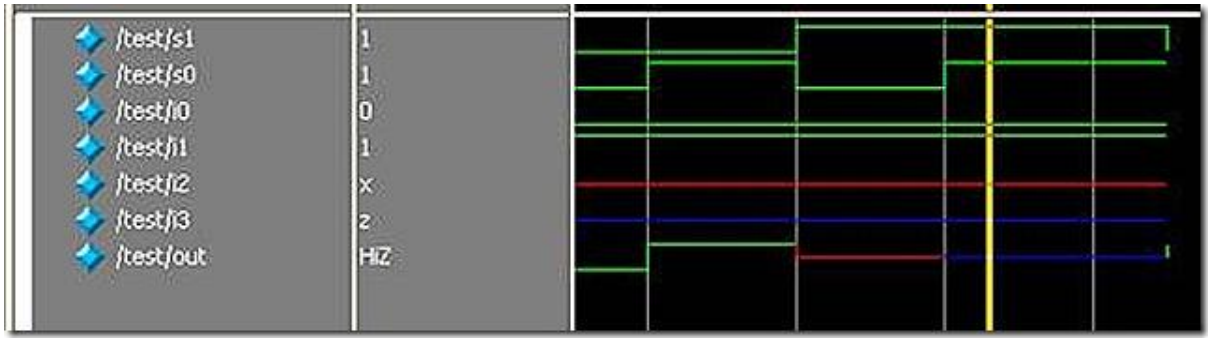


10. Design the 4-to-1 multiplexer in eg 7-19 by using if and else statements. The port interface must remain the same.

My answer:

```
1   //ex7_10 mux4-to-1
2   module mux4_to_1(out,i0,i1,i2,i3,s1,s0);
3
4   output reg out;
5   input i0,i1,i2,i3;
6   input s1,s0;
7
8   always @(*)
9   begin
10    if({s1,s0}==2'd0)
11      out=i0;
12    else if({s1,s0}==2'd1)
13      out=i1;
14    else if({s1,s0}==2'd2)
15      out=i2;
16    else
17      out=i3;
18  end
19
20  endmodule
```

```
1   //stimulus
2   module test;
3   reg s1,s0,i0,i1,i2,i3;
4   wire out;
5
6   mux4_to_1 m1(.out(out),.s1(s1),.s0(s0),.i0(i0),.i1(i1),
7                .i2(i2),.i3(i3));
8
9   initial
10  begin
11    {s1,s0}=2'd0;
12    i0=1'b0;
13    i1=1'b1;
14    i2=1'bx;
15    i3=1'bz;
16    #20 {s1,s0}=2'd1;
17    #20 {s1,s0}=2'd2;
18    #20 {s1,s0}=2'd3;
19    #30 {s1,s0}=2'd1;
20  end
21
22  endmodule
```



11. Design the traffic signal controller discussed in this chapter by using if and else statements.

My answer:

```
`define TRUE 1'b1;

`define FALSE 1'b0;

//Delays

`define Y2RDELAY 3 //Yellow to red delay

`define R2GDELAY 2 //Red to green delay

module sig_control

(hwy,cntry,X,clock,clear);

//I/O ports

output [1:0] hwy, cntry;

//2-bit output for 3 states of signal

//GREEN, YELLOW, RED

reg [1:0] hwy,cntry;

//declared output signals are registers

input X;
```

```
//if TRUE, indicates that there is car on

//the country road, otherwise FALSE

input clock,clear;

parameter RED=2'd0,

YELLOW=2'd1,

GREEN=2'd2;

//State definition HWY CONTRY

parameter S0=3'd0, //GREEN RED

S1=3'd1, //YELLOW RED

S2=3'd2, //RED RED

S3=3'd3, //RED GREEN

S4=3'd4; //RED YELLOW

//Internal state variables

reg [2:0] state;

reg [2:0] next_state;

//state changes only at postive edge of clock

always @(posedge clock)

if(clear)

state<=S0; //Controller starts in S0 state
```

```
else

state<=next_state; //State change

//Compute values of main signal and country signal

always @(state)

begin

//case(state)

//S0: ; //No change, use default
```

```
if(state==S1)

hwy=YELLOW;

else if(state==S2)

hwy=RED;

else if(state==S3)

begin

hwy=RED;

cntry=GREEN;

end

else if(state==S4)
```

```
begin

hwy=RED;

cntry=YELLOW;

end

else

begin

hwy=GREEN; //Default Light Assignment for Highway light

cntry=RED; //Default light Assignment for Country light
```

```
end

end

//State machine using case statements

always @(state or X)

begin

if(state==S0)

if(X)

next_state=S1;

else

next_state=S0;

else if(state== S1)

begin //delay some positive edges of clock

repeat(`Y2RDELAY) @(posedge clock);

next_state=S2;

end

else if(state== S2)

begin //delay some positive edges of clock

repeat(`R2GDELAY) @(posedge clock);

next_state=S3;

end
```

```
else if(state== S3)

if(X)

next_state=S3;

else

next_state=S4;

else

begin //delay some positive edges of clock

repeat(`Y2RDELAY) @(posedge clock);

next_state=S0;

end

//default:next_state=S0;

//endcase

end

endmodule
```