Sheet 2 Verilog (Bonus Sheet)

4.3 LOGIC GATES WITH DELAY

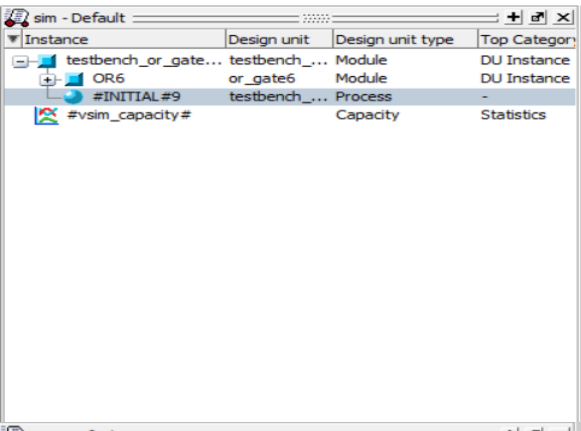
Code:

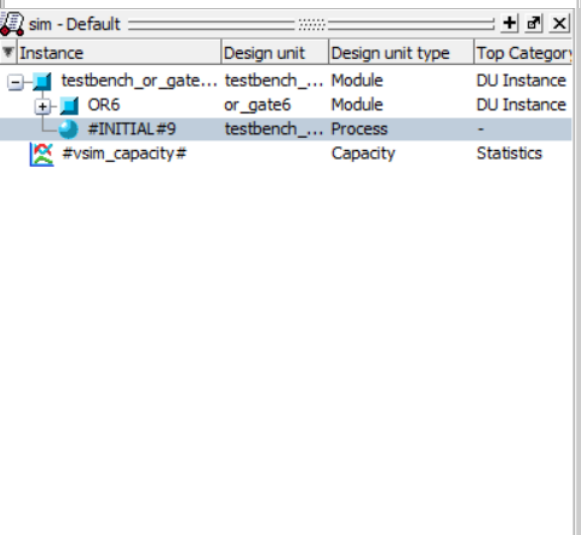
or_gate6.v

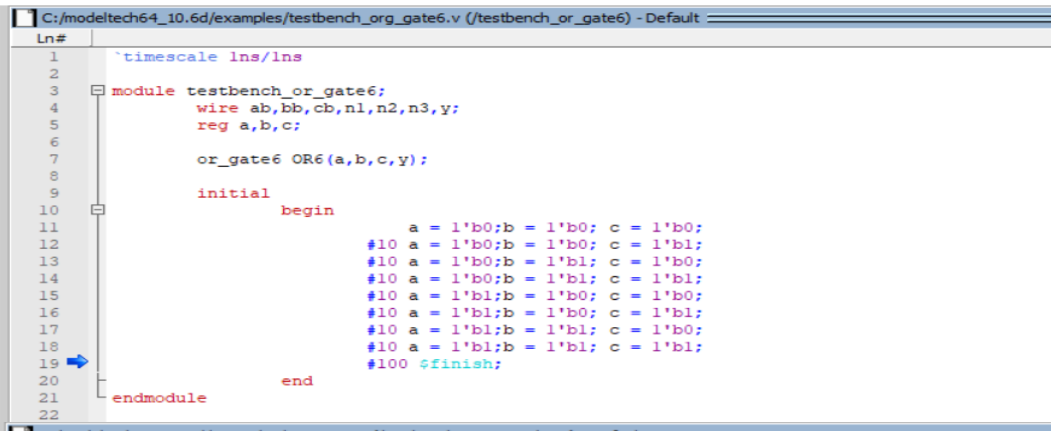
```
`timescale 1ns/1ns

module or_gate6(input a,b,c, output y);
        wire ab,bb,cb,n1,n2,n3;

        assign #1 {ab,bb,cb} = ~ {a,b,c};
        assign #2 n1 = ab& bb &cb;
        assign #2 n2 = a & bb&cb;
        assign #2 n3 = a & bb & c;
    assign #4 y = n1 | n2| n3;

endmodule
```

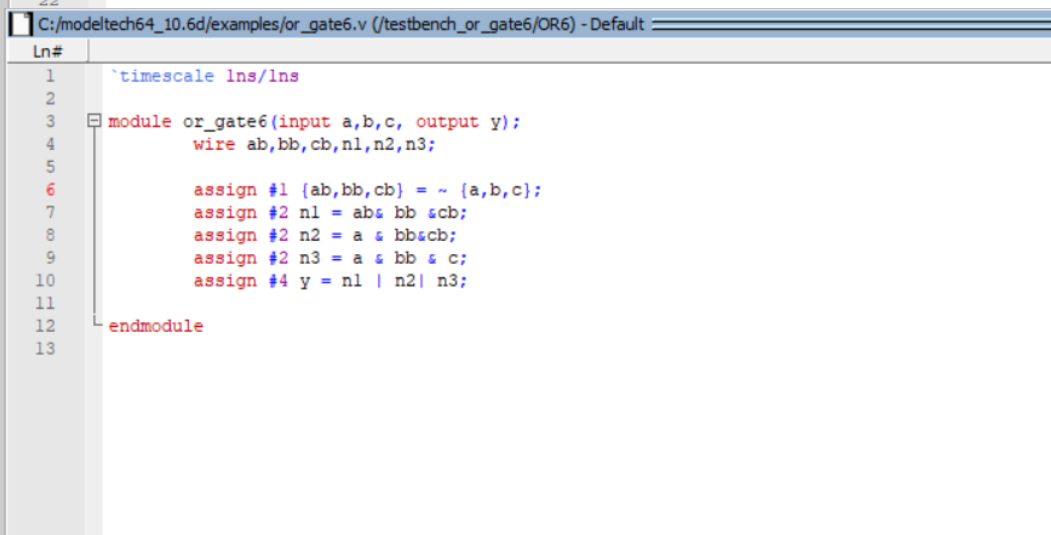
Testbench_org_gate6.v

```
`timescale 1ns/1ns

module testbench_or_gate6;
        wire ab,bb,cb,n1,n2,n3,y;
        reg a,b,c;
        or_gate6 OR6(a,b,c,y);
         initial
                begin
                        a = 1'b0;b = 1'b0; c = 1'b0;
                    #10 a = 1'b0;b = 1'b0; c = 1'b1;
                    #10 a = 1'b0;b = 1'b1; c = 1'b0;
                    #10 a = 1'b0;b = 1'b1; c = 1'b1;
                    #10 a = 1'b1;b = 1'b0; c = 1'b0;
                    #10 a = 1'b1;b = 1'b0; c = 1'b1;
                    #10 a = 1'b1;b = 1'b1; c = 1'b0;
                    #10 a = 1'b1;b = 1'b1; c = 1'b1;
                #100 $finish;
                end
endmodule
```









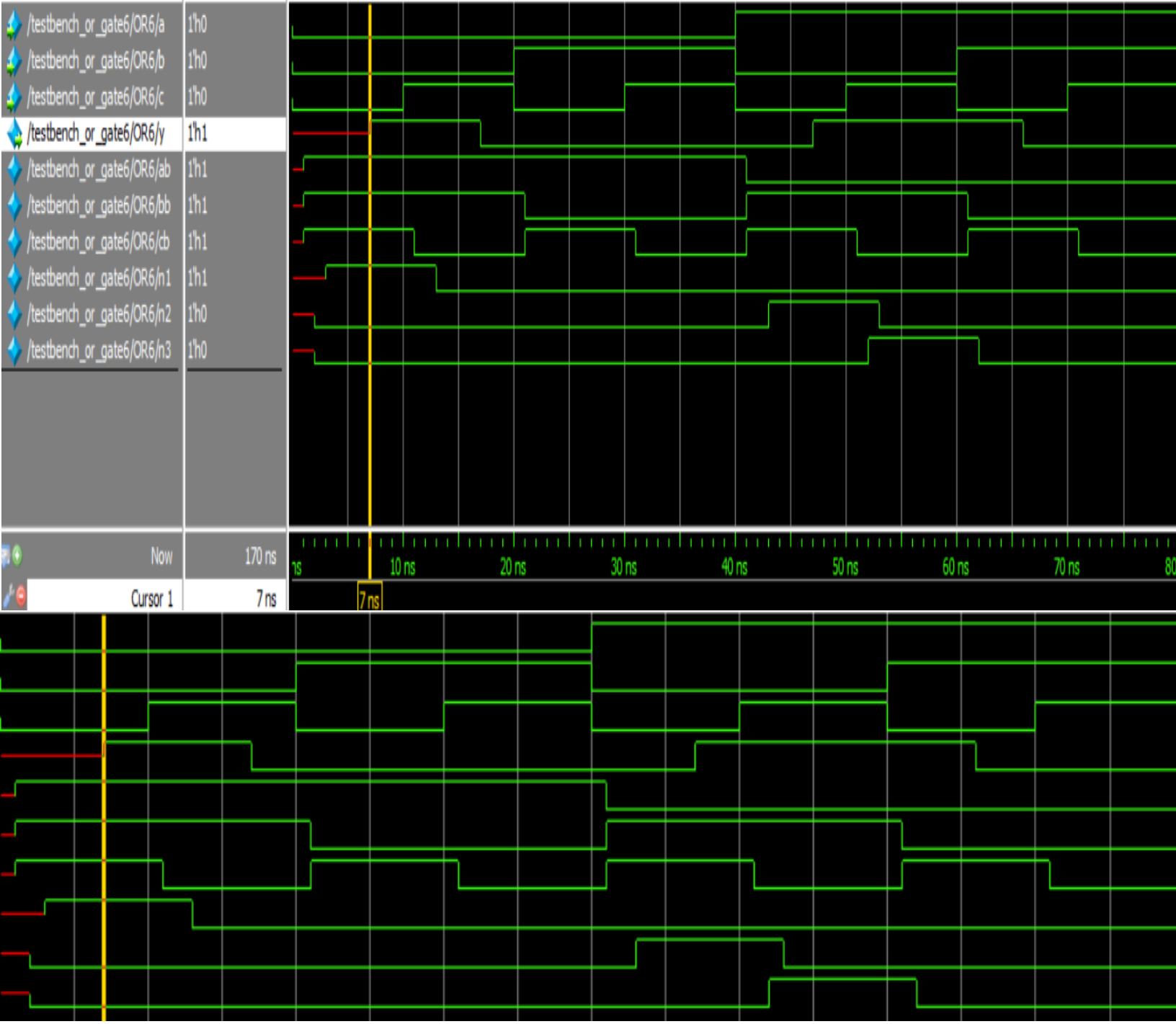


Illustration: the main purpose of this experiment to show how delay works as computing the delays in the code we found out that the total delay is 7ns as shown in the figure actually there is a 7ns delay in the y plot (output signal).

4.34 PARAMETERIZED N-BIT MULTIPLEXER

MULTIPLIXER.v

```
module mux2
# (parameter width = 8)
(input [width-1:0] d0, d1,
input s,
output [width-1:0] y);
assign y = s ? d1 : d0;
endmodule

module mux4_8 (input [7:0] d0, d1, d2, d3,
input [1:0] s,
output [7:0] y);
wire [7:0] low, hi;
mux2 lowmux (d0, d1, s[0], low);
mux2 himux (d2, d3, s[1], hi);
mux2 outmux (low, hi, s[1], y);
endmodule

module mux4_12 (input [11:0] d0, d1, d2, d3,
input [1:0] s,
output [11:0] y);
wire [11:0] low, hi;
mux2 #(12) lowmux(d0, d1, s[0], low);
mux2 #(12) himux(d2, d3, s[1], hi);
mux2 #(12) outmux(low, hi, s[1], y);
endmodule
```

testbench_MULTIPLIXER.v

```
module testbench_or_gate6;
  reg  [11:0] d0, d1, d2, d3;
  reg   [1:0] s;
  wire  [11:0] y;
  wire [11:0] low, hi;
  mux4_12  fa0 (  .d0 (d0),
                  .d1 (d1),
                  .d2 (d2),
                  .d3 (d3),
                  .s (s),
                  .y (y));

  initial begin
                    d0 <= 0;d1 <= 0;d2 <= 0;d3 <= 0;s <=  00;
                #10 d0 <= 0;d1 <= 0;d2 <= 0;d3 <= 1;s <=  01;
                #10 d0 <= 0;d1 <= 0;d2 <= 1;d3 <= 0;s <=  10;
                #10 d0 <= 0;d1 <= 0;d2 <= 1;d3 <= 1;s <=  11;
                #10 d0 <= 0;d1 <= 1;d2 <= 0;d3 <= 0;s <=  10;
                #10 d0 <= 0;d1 <= 1;d2 <= 0;d3 <= 1;s <=  00;
                #10 d0 <= 0;d1 <= 1;d2 <= 1;d3 <= 0;s <=  01;
                #10 d0 <= 0;d1 <= 1;d2 <= 1;d3 <= 1;s <=  11;
    end
endmodule
```

4.34 RESETTABLE REGISTER

flopr.v

```
module flopr (input clk,reset, input [3:0] d,
output reg [3:0] q);
// asynchronous reset
   always @ (posedge clk, posedge reset)
          if (reset) q <=4'b0;
      else q <= d;
endmodule

module flopr (input clk, reset, input [3:0] d,
output reg [3:0] q);
// synchronous reset
 always @ (posedge clk)
   if (reset) q <= 4'b0;
   else q <= d;
endmodule
```

testbench_flopr.v

```
module testbench_flopr;
  reg  clk,reset;
  reg   [3:0] d;
  wire   [3:0] q;
  flopr  fa0 (  .clk (clk),
                .reset (reset),
                .d (d),
                .q (q));

  initial begin
                d <= 5;reset <= 1;clk <= 1;
   end
endmodule
```

4.34 RESETTABLE REGISTER

flopr.v

```
module tristate (input [3:0] d,input rs ,
output reg[3:0]q);
   always @ (*)
          assign q=(rs)?d:4'bz;

 endmodule
module MUX (input [3:0] d0, d1,input s,
output [3:0] y);
tristate t0 (d0, ~s, y);
tristate t1 (d1, s, y);
endmodule
```

testbench_flopr.v

```
module testbench_or_gate6;
  reg  [3:0] d0, d1;
  reg   s;
  wire  [3:0] y;
  MUX  fa0 (    .d0 (d0),
               .d1 (d1),
                .s (s),
                .y (y));

  initial begin
                d0 <= 5;d1 <= 1;s <= 0;

  end
endmodule
```

4.29 FuLL ADDER NON-BLOCKING

full_adder.v

```
module full_adder(input a, b, cin, output reg s, cout);
        reg p, g;
        always @ (*)
          begin
                p = a ^ b;
                g = a & b;

                s = p ^ cin;
                cout = g | (p & cin);
          end
endmodule
```

```
module testbench_or_gate6;
  reg  a;
  reg  b;
  reg cin;
  wire  s;
  wire  cout;
  or_gate6  fa0 ( .a (a),
            .b (b),
            .cin (cin),
            .s (s),
            .cout (cout));
  initial begin
              a <= 0;b <= 0;cin <=  0;
          #10 a <= 0;b <= 0;cin <=  1;
          #10 a <= 0;b <= 1;cin <= 0;
          #10 a <= 0;b <= 1;cin <= 1;
          #10 a <= 1;b <= 0;cin <= 0;
          #10 a <= 1;b <= 0;cin <= 1;
          #10 a <= 1;b <= 1;cin <= 0;
          #10 a <= 1;b <= 1;cin <= 1;
      end
```