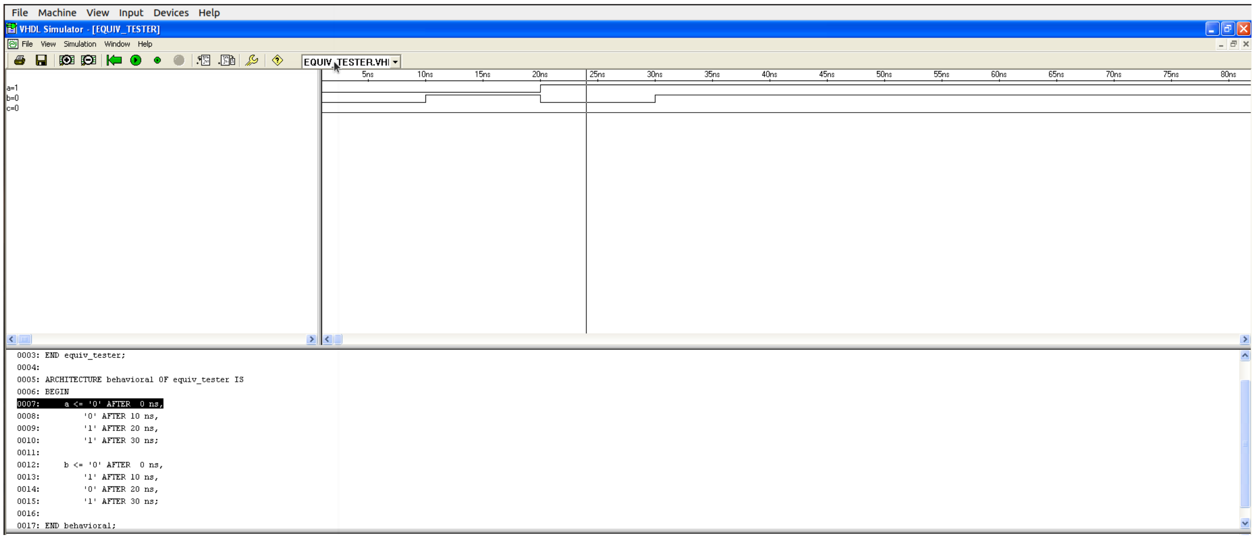
CS 247 - Project 1

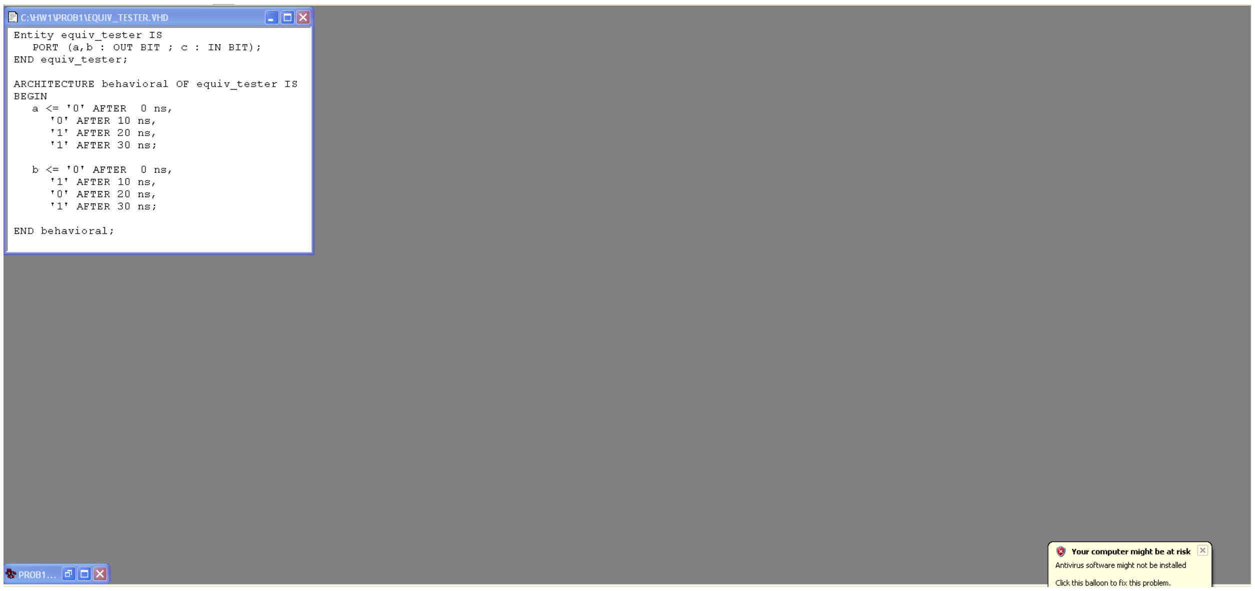
1) Enter the entity declaration and behavioral architecture specification of equiv_tester exactly as it appears (except for the line numbers) on page 22 of the class notes. Compile the entity and its behavioral specification, link, and then invoke the PeakVHDL simulator on it. Monitor the value of signals a and b to verify that their values match those as shown in the table at the bottom of page 22. [2 pts]



File Machine View Input Devices Help
VHDL Simulator - [EQUIV_TESTER]
File View Simulation Window Help
EQUIV_TESTER.VHI
5ns 10ns 15ns 20ns 25ns 30ns 35ns 40ns 45ns 50ns 55ns 60ns 65ns 70ns 75ns 80ns

```
a=1
b=0
c=0
```

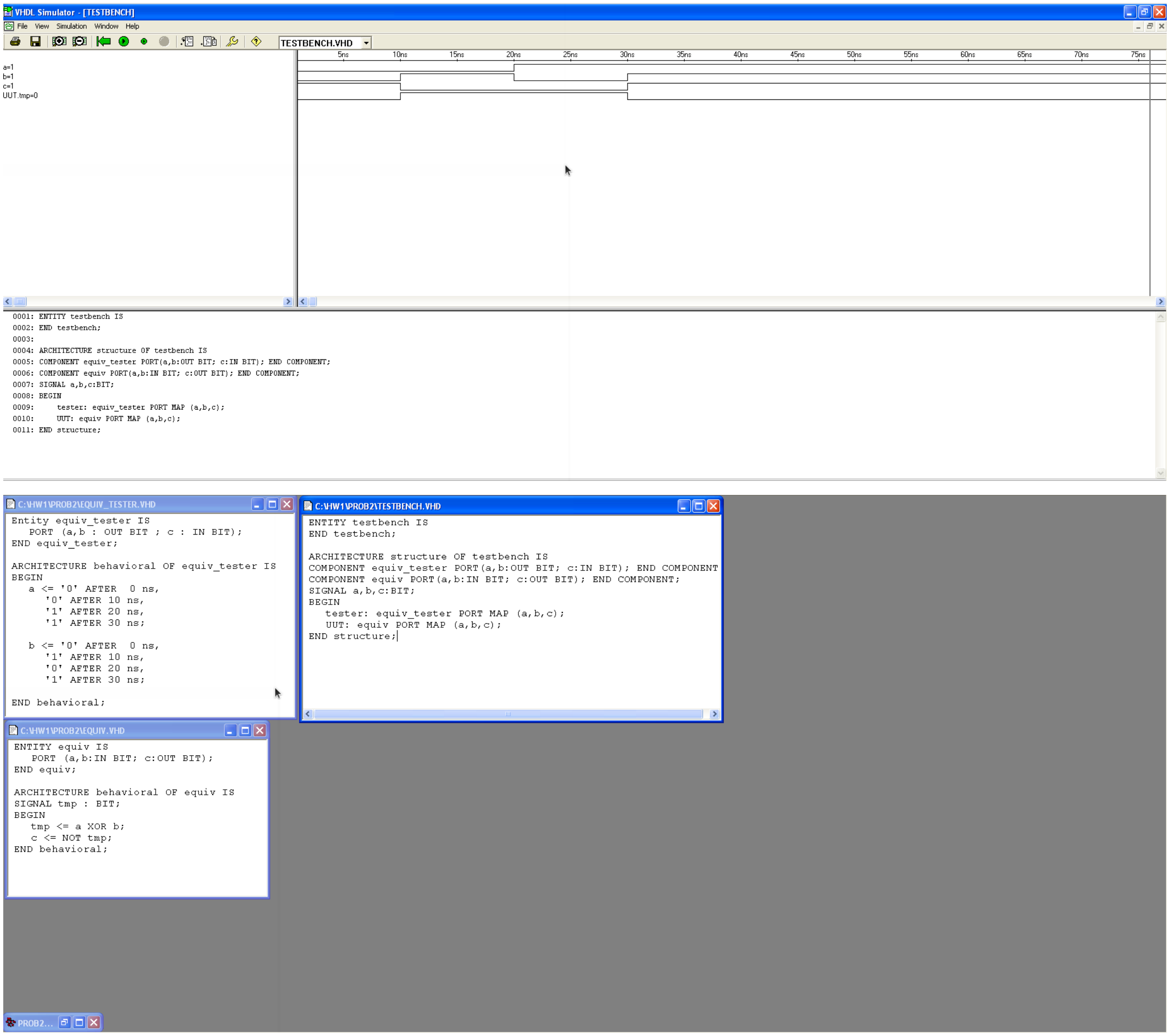
```
END equiv_tester;

ARCHITECTURE behavioral OF equiv_tester IS
BEGIN
    a <= '0' AFTER  0 ns,
         '0' AFTER 10 ns,
         '1' AFTER 20 ns,
         '1' AFTER 30 ns;

    b <= '0' AFTER  0 ns,
         '1' AFTER 10 ns,
         '0' AFTER 20 ns,
         '1' AFTER 30 ns;

END behavioral;
```



```
Entity equiv_tester IS
   PORT (a,b : OUT BIT ; c : IN BIT);
END equiv_tester;

ARCHITECTURE behavioral OF equiv_tester IS
BEGIN
   a <= '0' AFTER  0 ns,
      '0' AFTER 10 ns,
      '1' AFTER 20 ns,
      '1' AFTER 30 ns;

   b <= '0' AFTER  0 ns,
      '1' AFTER 10 ns,
      '0' AFTER 20 ns,
      '1' AFTER 30 ns;

END behavioral;
```

Your computer might be at risk
Antivirus software might not be installed
Click this balloon to fix this problem.

2) Enter the entity declaration and behavioral architecture specification of equiv exactly as it appears (except for the line numbers) on page 20 of the class notes. Enter the entity declaration and structural architecture specification of the testbench exactly as shown on page 23 of the class notes. Put these two modules into the same .acc project used for equiv_tester in problem 1. Compile all the entities and their architecture specifications, link them, and then invoke the PeakVHDL simulator. Monitor the value of all signals (including TMP). Note that there is no time delay between output signal changes and input signal changes since the behavioral architecture body of equiv does not specify any time delays. [2 pts]



```
a=1
b=1
c=1
UUT.tmp=0
```

```
ENTITY testbench IS
END testbench;

ARCHITECTURE structure OF testbench IS
COMPONENT equiv_tester PORT(a,b:OUT BIT; c:IN BIT); END COMPONENT;
COMPONENT equiv PORT(a,b:IN BIT; c:OUT BIT); END COMPONENT;
SIGNAL a,b,c:BIT;
BEGIN
    tester: equiv_tester PORT MAP (a,b,c);
    UUT: equiv PORT MAP (a,b,c);
END structure;
```

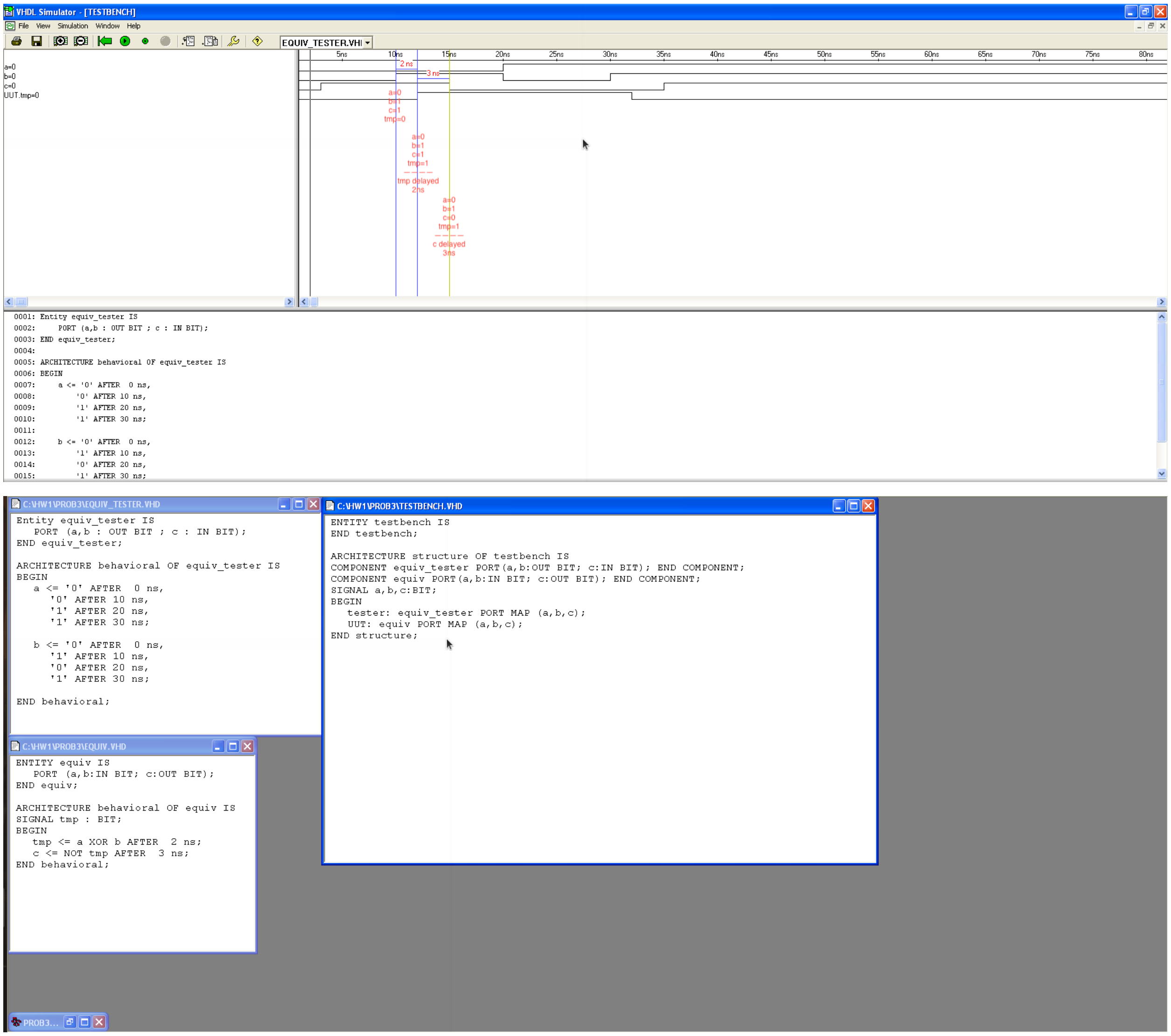
```
Entity equiv_tester IS
   PORT (a,b : OUT BIT ; c : IN BIT);
END equiv_tester;

ARCHITECTURE behavioral OF equiv_tester IS
BEGIN
   a <= '0' AFTER  0 ns,
      '0' AFTER 10 ns,
      '1' AFTER 20 ns,
      '1' AFTER 30 ns;

   b <= '0' AFTER  0 ns,
      '1' AFTER 10 ns,
      '0' AFTER 20 ns,
      '1' AFTER 30 ns;

END behavioral;
```

```
ENTITY testbench IS
END testbench;

ARCHITECTURE structure OF testbench IS
COMPONENT equiv_tester PORT(a,b:OUT BIT; c:IN BIT); END COMPONENT
COMPONENT equiv PORT(a,b:IN BIT; c:OUT BIT); END COMPONENT;
SIGNAL a,b,c:BIT;
BEGIN
   tester: equiv_tester PORT MAP (a,b,c);
   UUT: equiv PORT MAP (a,b,c);
END structure;
```

```
ENTITY equiv IS
   PORT (a,b:IN BIT; c:OUT BIT);
END equiv;

ARCHITECTURE behavioral OF equiv IS
SIGNAL tmp : BIT;
BEGIN
   tmp <= a XOR b;
   c <= NOT tmp;
END behavioral;
```

3) Add a delay time of 2ns and 3ns to lines 4 and 5 of equiv respectively. Recompile, relink, and run the simulator to observe the output as a function of time. How long does it take for output c to react to a change on either a or b? How long does it take for the signal tmp to react to a change on either a or b? Are these time delays as expected? [2 pts]

- how long does it take for output c to react to a change on either a or b? 5 ns
- How long does it take for the signal tmp to react to a change on either a or b? 2 ns
- Are these time delays as expected? Yes



```
Entity equiv_tester IS
    PORT (a,b : OUT BIT ; c : IN BIT);
END equiv_tester;

ARCHITECTURE behavioral OF equiv_tester IS
BEGIN
    a <= '0' AFTER  0 ns,
         '0' AFTER 10 ns,
         '1' AFTER 20 ns,
         '1' AFTER 30 ns;

    b <= '0' AFTER  0 ns,
         '1' AFTER 10 ns,
         '0' AFTER 20 ns,
         '1' AFTER 30 ns;
```

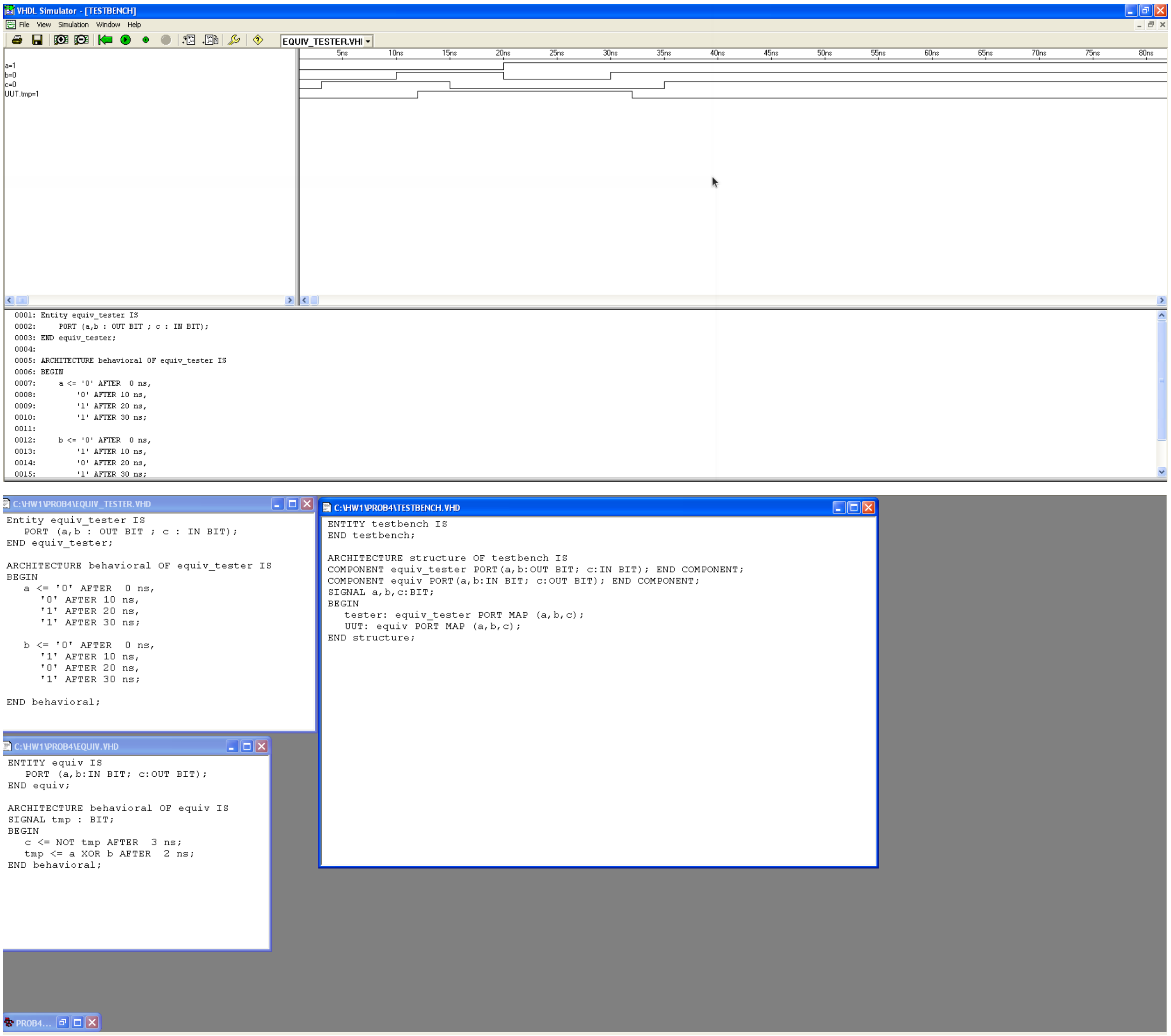
```
Entity equiv_tester IS
   PORT (a,b : OUT BIT ; c : IN BIT);
END equiv_tester;

ARCHITECTURE behavioral OF equiv_tester IS
BEGIN
   a <= '0' AFTER  0 ns,
      '0' AFTER 10 ns,
      '1' AFTER 20 ns,
      '1' AFTER 30 ns;

   b <= '0' AFTER  0 ns,
      '1' AFTER 10 ns,
      '0' AFTER 20 ns,
      '1' AFTER 30 ns;

END behavioral;
```

```
ENTITY testbench IS
END testbench;

ARCHITECTURE structure OF testbench IS
COMPONENT equiv_tester PORT(a,b:OUT BIT; c:IN BIT); END COMPONENT;
COMPONENT equiv PORT(a,b:IN BIT; c:OUT BIT); END COMPONENT;
SIGNAL a,b,c:BIT;
BEGIN
   tester: equiv_tester PORT MAP (a,b,c);
   UUT: equiv PORT MAP (a,b,c);
END structure;
```

```
ENTITY equiv IS
   PORT (a,b:IN BIT; c:OUT BIT);
END equiv;

ARCHITECTURE behavioral OF equiv IS
SIGNAL tmp : BIT;
BEGIN
  tmp <= a XOR b AFTER  2 ns;
   c <= NOT tmp AFTER  3 ns;
END behavioral;
```

4) Reverse lines 4 and 5 of the behavioral model of equiv (the one with delay times) to verify that the order of statements is not important in event-driven simulation. What would the result of this statement order reversal be if VHDL behaved as a typical programming language ? [2 pts]

- verify that the order of statements is not important in event-driven simulation. waveform shows nothing changed compared to PROB3
- What would the result of this statement order reversal be if VHDL behaved as a typical programming language? "tmp" will be exactly the same as PROB3, "c" will delay 3 ns when simulation starts, and delay 5 ns after "a" or "b" changed.


VHDL Simulator - [TESTBENCH]
File View Simulation Window Help
EQUIV_TESTER.VHI
5ns 10ns 15ns 20ns 25ns 30ns 35ns 40ns 45ns 50ns 55ns 60ns 65ns 70ns 75ns 80ns
a=1
b=0
c=0
UUT.tmp=1
C:\HW1\PROB4\EQUIV_TESTER.VHD
C:\HW1\PROB4\TESTBENCH.VHD
C:\HW1\PROB4\EQUIV.VHD
PROB4...

```
Entity equiv_tester IS
    PORT (a,b : OUT BIT ; c : IN BIT);
END equiv_tester;

ARCHITECTURE behavioral OF equiv_tester IS
BEGIN
    a <= '0' AFTER  0 ns,
        '0' AFTER 10 ns,
        '1' AFTER 20 ns,
        '1' AFTER 30 ns;

    b <= '0' AFTER  0 ns,
        '1' AFTER 10 ns,
        '0' AFTER 20 ns,
        '1' AFTER 30 ns;
```

C:\HW1\PROB4\EQUIV_TESTER.VHD

```
Entity equiv_tester IS
   PORT (a,b : OUT BIT ; c : IN BIT);
END equiv_tester;

ARCHITECTURE behavioral OF equiv_tester IS
BEGIN
   a <= '0' AFTER  0 ns,
      '0' AFTER 10 ns,
      '1' AFTER 20 ns,
      '1' AFTER 30 ns;

   b <= '0' AFTER  0 ns,
      '1' AFTER 10 ns,
      '0' AFTER 20 ns,
      '1' AFTER 30 ns;

END behavioral;
```

C:\HW1\PROB4\TESTBENCH.VHD

```
ENTITY testbench IS
END testbench;

ARCHITECTURE structure OF testbench IS
COMPONENT equiv_tester PORT(a,b:OUT BIT; c:IN BIT); END COMPONENT;
COMPONENT equiv PORT(a,b:IN BIT; c:OUT BIT); END COMPONENT;
SIGNAL a,b,c:BIT;
BEGIN
   tester: equiv_tester PORT MAP (a,b,c);
   UUT: equiv PORT MAP (a,b,c);
END structure;
```

C:\HW1\PROB4\EQUIV.VHD

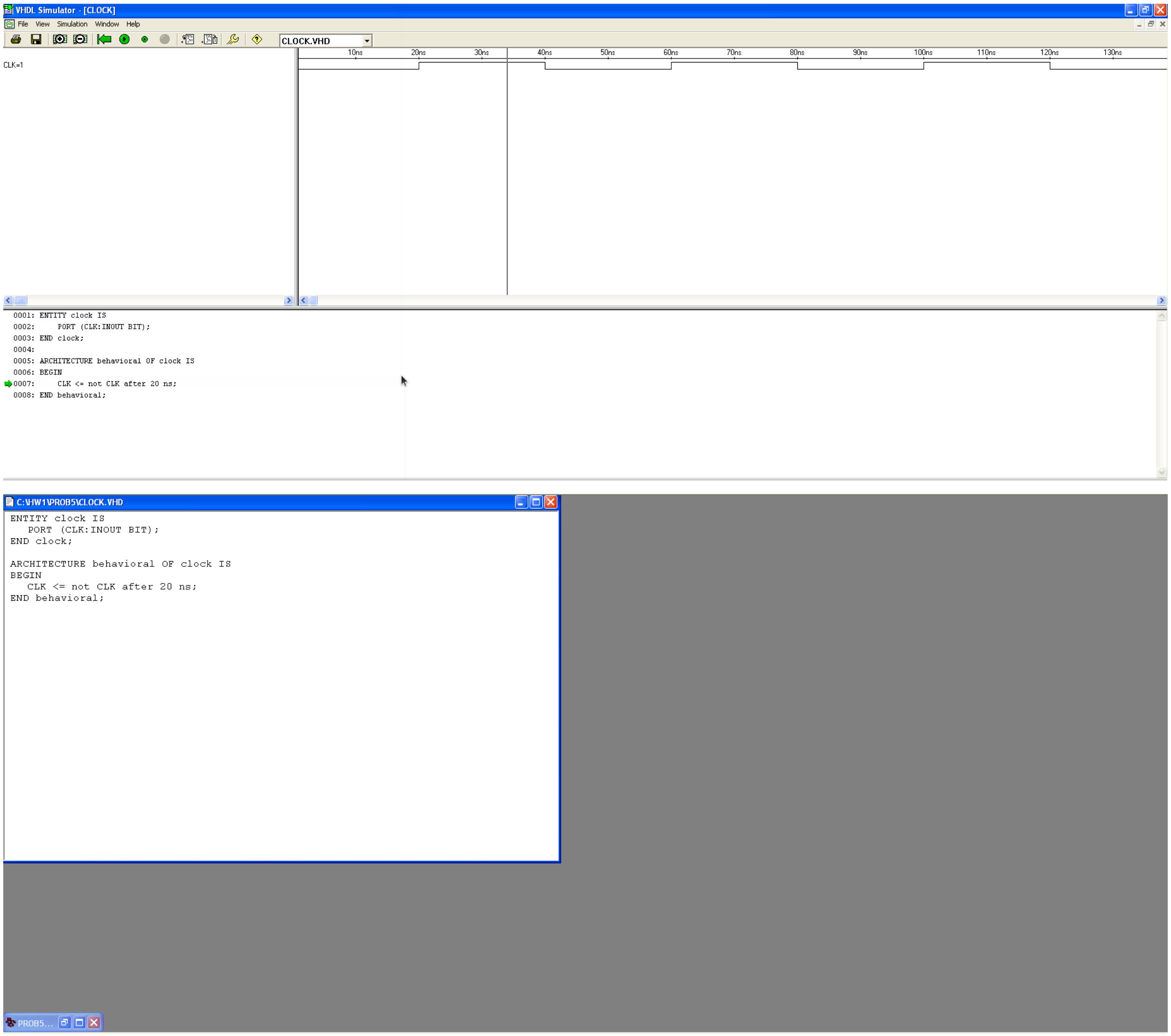
```
ENTITY equiv IS
   PORT (a,b:IN BIT; c:OUT BIT);
END equiv;

ARCHITECTURE behavioral OF equiv IS
SIGNAL tmp : BIT;
BEGIN
   c <= NOT tmp AFTER  3 ns;
   tmp <= a XOR b AFTER  2 ns;
END behavioral;
```

5) The following is a simple, commonly used model to generate a perpetual clock signal in VHDL. Write an entity declaration and behavioral model for a one pin device (call that pin CLK). The behavioral model will contain just one signal assignment statement of the form:

CLK <= not CLK after 20 ns;

Assuming that CLK was needed to drive the input of another device, what direction (in, out, or inout) of signal does it need to be and why ? [2 pts]

- what direction (in, out, or inout) of signal does it need to be and why ? Direction is inout, because input and output signal is same in the behavioral model



```
ENTITY clock IS
    PORT (CLK:INOUT BIT);
END clock;

ARCHITECTURE behavioral OF clock IS
BEGIN
    CLK <= not CLK after 20 ns;
END behavioral;
```

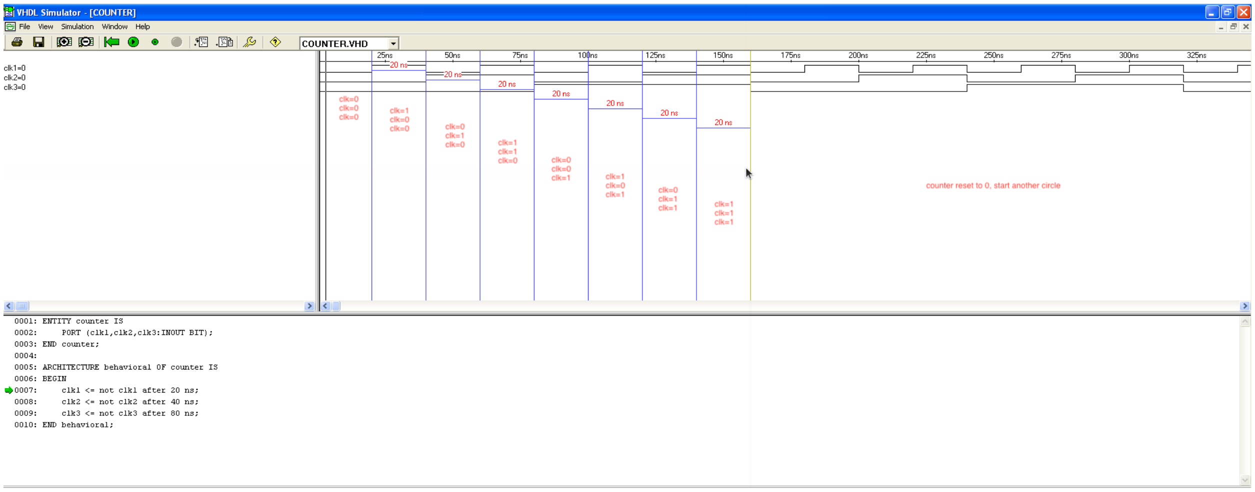
```
ENTITY clock IS
   PORT (CLK:INOUT BIT);
END clock;

ARCHITECTURE behavioral OF clock IS
BEGIN
   CLK <= not CLK after 20 ns;
END behavioral;
```

6) Use the same type of model as problem 5 to develop an entity called counter that has three clocks (clk1, clk2, clk3) to generate a truth table consisting of all eight exhaustive binary values for the three signals. That is, assume clk1 is the LSB and clk3 is the MSB of a truth table. Describe your technique in words. In particular, what would be a simple rule to employ for the delays between different bit positions assuming you want to count binary values at fixed time intervals, say every t ns ? [3 pts]

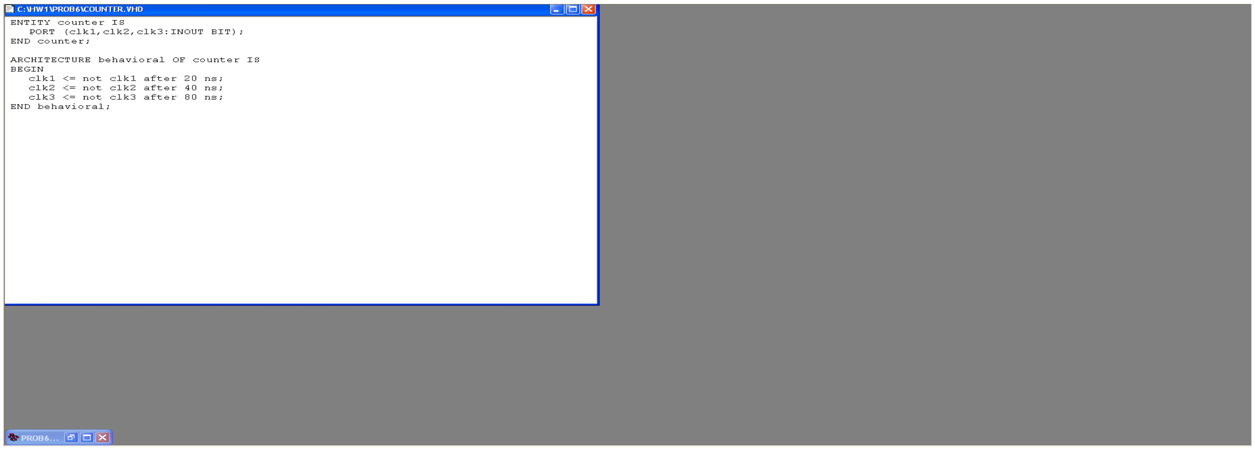
- Describe your technique in words:

clk1	clk2	clk3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

The truth tables indicate, assuming 1 time unit is 20ns. clk1 flip every 20ns, clk1 flip every 40ns, clk1 flip every 80ns.

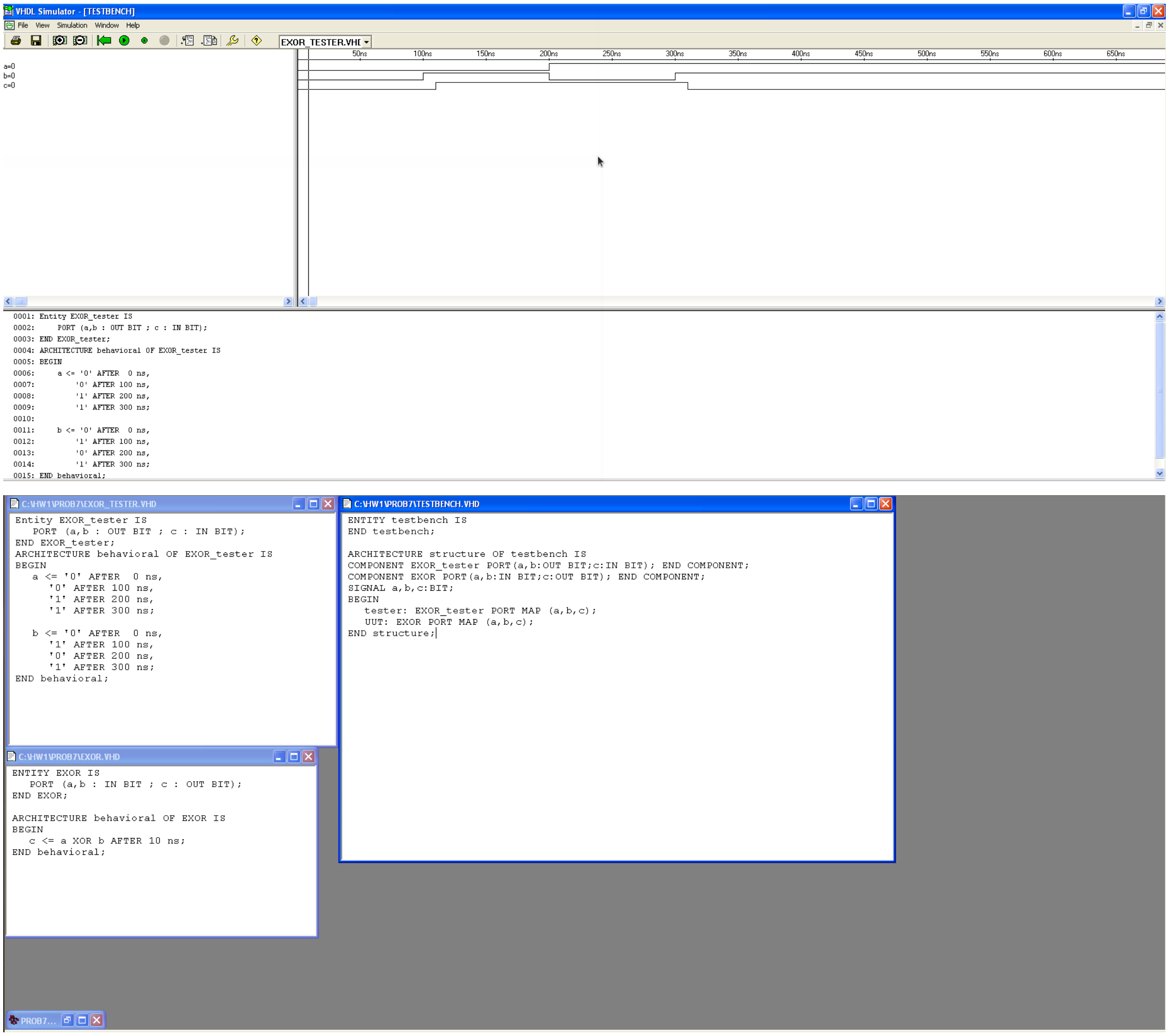


```
ENTITY counter IS
    PORT (clk1,clk2,clk3:INOUT BIT);
END counter;

ARCHITECTURE behavioral OF counter IS
BEGIN
    clk1 <= not clk1 after 20 ns;
    clk2 <= not clk2 after 40 ns;
    clk3 <= not clk3 after 80 ns;
END behavioral;
```



```
ENTITY counter IS
   PORT (clk1,clk2,clk3:INOUT BIT);
END counter;

ARCHITECTURE behavioral OF counter IS
BEGIN
   clk1 <= not clk1 after 20 ns;
   clk2 <= not clk2 after 40 ns;
   clk3 <= not clk3 after 80 ns;
END behavioral;
```

7) Write an entity declaration and behavioral architecture specification for a two input EXOR function. Assume that this EXOR function responds to changes in its inputs after 10 ns of inertial delay. Compile the EXOR entity and invoke the VHDL simulator on it. Monitor the value of all signals. Verify proper operation by setting the inputs to all four possible input combinations and observing the output signal's response as a function of simulation time. Hint: reuse as much of equiv_tester and testbench from problems 1 and 2. Note that this EXOR "function" only specifies "stimulus-response" behavior. This type of model is often used by engineers to obtain a high-level functional verification for the "building blocks" that they intend on using in a design. In problem 9, you will refine the design of the EXOR function to the gate level of detail by specifying a structural implementation for it. [2 pts]


VHDL Simulator - [TESTBENCH]
File View Simulation Window Help
EXOR_TESTER.VHI
50ns 100ns 150ns 200ns 250ns 300ns 350ns 400ns 450ns 500ns 550ns 600ns 650ns
a=0
b=0
c=0
C:\HW1\PROB7\EXOR_TESTER.VHD
C:\HW1\PROB7\TESTBENCH.VHD
C:\HW1\PROB7\EXOR.VHD
PROB7...

```
Entity EXOR_tester IS
    PORT (a,b : OUT BIT ; c : IN BIT);
END EXOR_tester;
ARCHITECTURE behavioral OF EXOR_tester IS
BEGIN
    a <= '0' AFTER  0 ns,
         '0' AFTER 100 ns,
         '1' AFTER 200 ns,
         '1' AFTER 300 ns;

    b <= '0' AFTER  0 ns,
         '1' AFTER 100 ns,
         '0' AFTER 200 ns,
         '1' AFTER 300 ns;
END behavioral;
```

C:\HW1\PROB7\EXOR_TESTER.VHD

```
Entity EXOR_tester IS
   PORT (a,b : OUT BIT ; c : IN BIT);
END EXOR_tester;
ARCHITECTURE behavioral OF EXOR_tester IS
BEGIN
   a <= '0' AFTER  0 ns,
      '0' AFTER 100 ns,
      '1' AFTER 200 ns,
      '1' AFTER 300 ns;

   b <= '0' AFTER  0 ns,
      '1' AFTER 100 ns,
      '0' AFTER 200 ns,
      '1' AFTER 300 ns;
END behavioral;
```

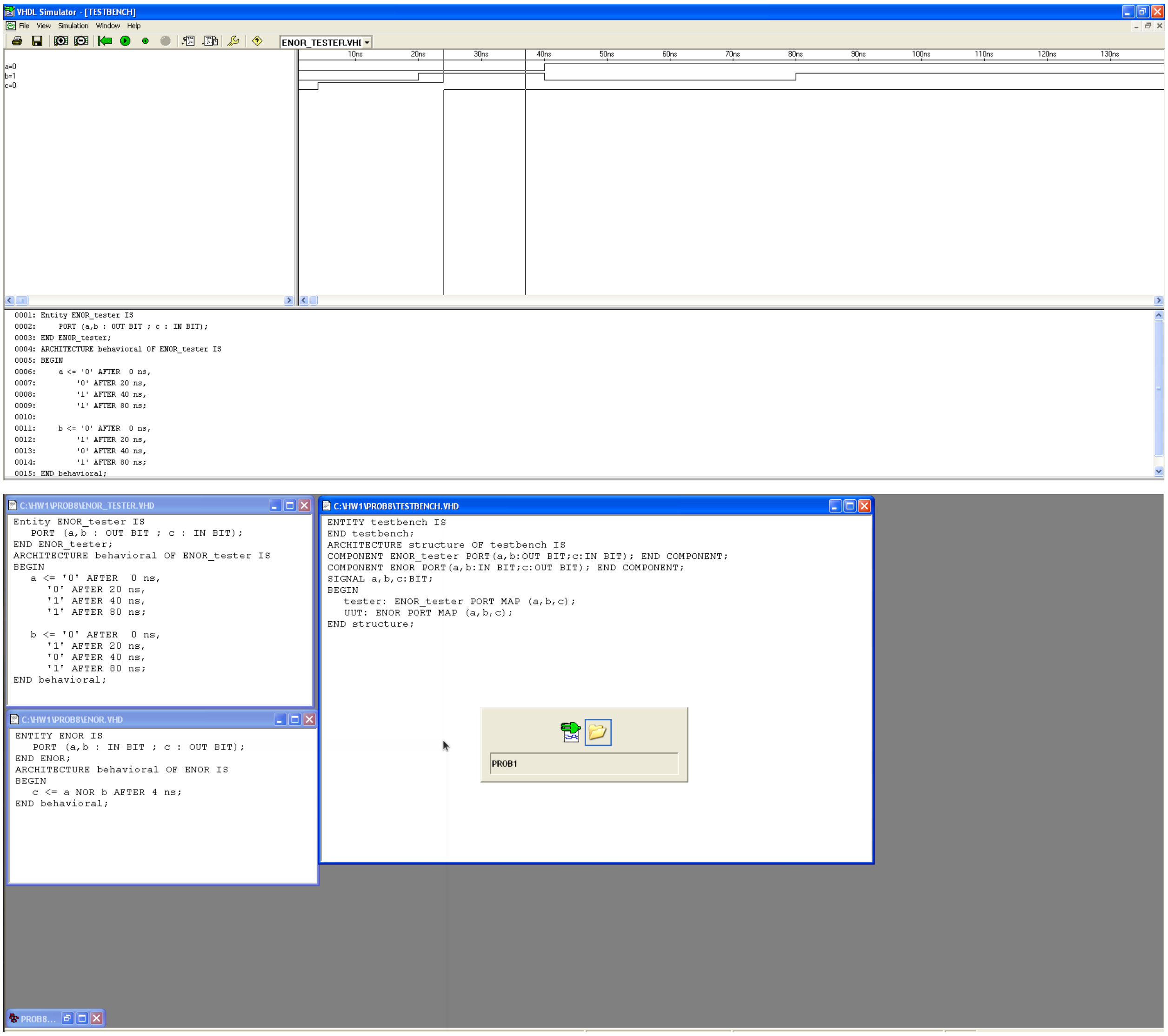
C:\HW1\PROB7\TESTBENCH.VHD

```
ENTITY testbench IS
END testbench;

ARCHITECTURE structure OF testbench IS
COMPONENT EXOR_tester PORT(a,b:OUT BIT;c:IN BIT); END COMPONENT;
COMPONENT EXOR PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
SIGNAL a,b,c:BIT;
BEGIN
   tester: EXOR_tester PORT MAP (a,b,c);
   UUT: EXOR PORT MAP (a,b,c);
END structure;
```

C:\HW1\PROB7\EXOR.VHD

```
ENTITY EXOR IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END EXOR;

ARCHITECTURE behavioral OF EXOR IS
BEGIN
   c <= a XOR b AFTER 10 ns;
END behavioral;
```

8) Write an entity declaration and behavioral architecture specification for a two-input NOR gate. Assume that this NOR gate responds to changes in its inputs after 4ns of inertial delay. Verify its proper operation. Again, reuse as many entities as possible from previous problems. [2 pts]


VHDL Simulator - [TESTBENCH]
File View Simulation Window Help
ENOR_TESTER.VHI
10ns
20ns
30ns
40ns
50ns
60ns
70ns
80ns
90ns
100ns
110ns
120ns
130ns
a=0
b=1
c=0
C:\HW1\PROB8\ENOR_TESTER.VHD
C:\HW1\PROB8\TESTBENCH.VHD
C:\HW1\PROB8\ENOR.VHD
PROB1
PROB8...

```
Entity ENOR_tester IS
     PORT (a,b : OUT BIT ; c : IN BIT);
END ENOR_tester;
ARCHITECTURE behavioral OF ENOR_tester IS
BEGIN
     a <= '0' AFTER  0 ns,
         '0' AFTER 20 ns,
         '1' AFTER 40 ns,
         '1' AFTER 80 ns;

     b <= '0' AFTER  0 ns,
         '1' AFTER 20 ns,
         '0' AFTER 40 ns,
         '1' AFTER 80 ns;
END behavioral;
```

C:\HW1\PROB8\ENOR_TESTER.VHD

```
Entity ENOR_tester IS
   PORT (a,b : OUT BIT ; c : IN BIT);
END ENOR_tester;
ARCHITECTURE behavioral OF ENOR_tester IS
BEGIN
   a <= '0' AFTER  0 ns,
      '0' AFTER 20 ns,
      '1' AFTER 40 ns,
      '1' AFTER 80 ns;

   b <= '0' AFTER  0 ns,
      '1' AFTER 20 ns,
      '0' AFTER 40 ns,
      '1' AFTER 80 ns;
END behavioral;
```

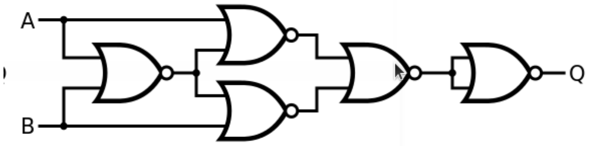
C:\HW1\PROB8\TESTBENCH.VHD

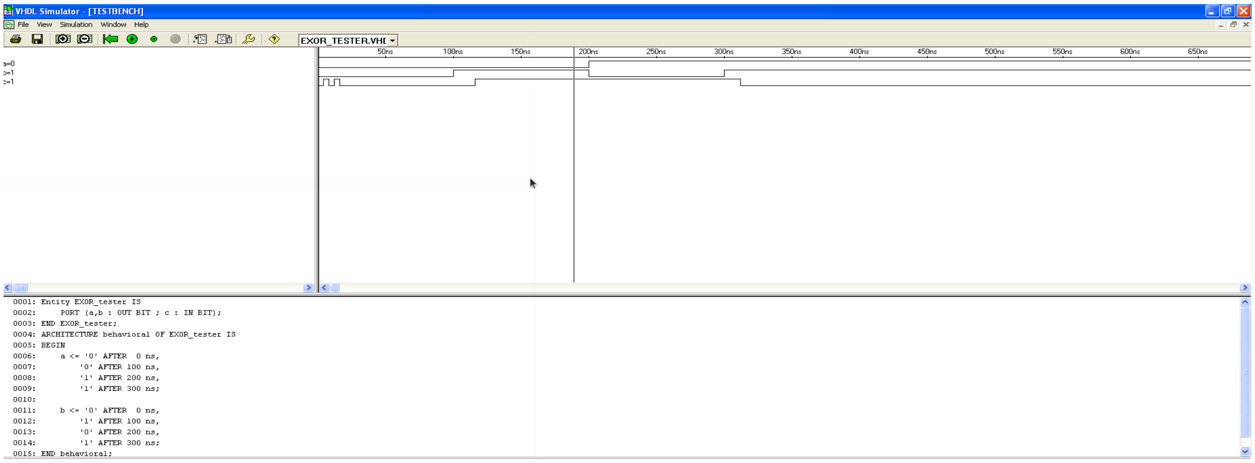
```
ENTITY testbench IS
END testbench;
ARCHITECTURE structure OF testbench IS
COMPONENT ENOR_tester PORT(a,b:OUT BIT;c:IN BIT); END COMPONENT;
COMPONENT ENOR PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
SIGNAL a,b,c:BIT;
BEGIN
   tester: ENOR_tester PORT MAP (a,b,c);
   UUT: ENOR PORT MAP (a,b,c);
END structure;
```

C:\HW1\PROB8\ENOR.VHD

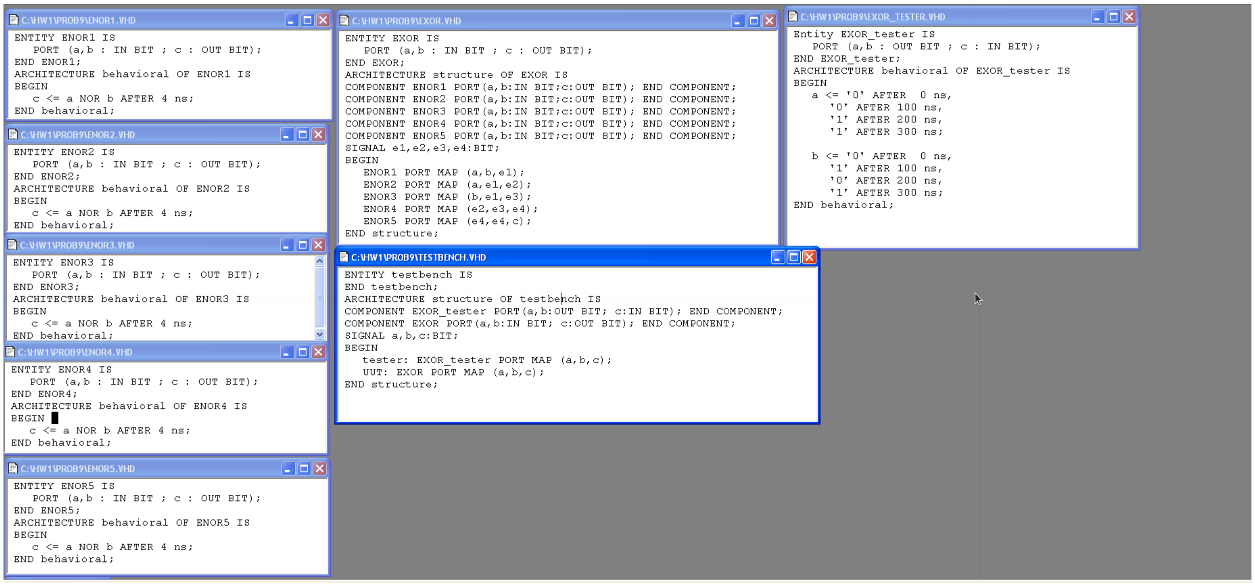
```
ENTITY ENOR IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR;
ARCHITECTURE behavioral OF ENOR IS
BEGIN
   c <= a NOR b AFTER 4 ns;
END behavioral;
```

9) Assume that the only parts you have for building various functions are NOR gates. Using only the NOR gates defined in problem 8 with 4ns delay as components, design a 2-input EXOR function. Draw a schematic of the circuit and develop a VHDL structural model of the circuit based upon your schematic. Label your schematic to show correspondence with your structural VHDL model. Apply the four input test vectors needed to completely test the 2-input part to verify proper performance. The responses obtained here should match those of the behavioral model from problem 7 (except for timing differences). Note that you can use the same entity declaration for EXOR as that used for problem 7; however, if you do so, be sure to specify that you want the simulator to use the structure model (instead of the behavior model) as the design unit. [6 pts]





```
Entity EXOR_tester IS
    PORT (a,b : OUT BIT ; c : IN BIT);
END EXOR_tester;
ARCHITECTURE behavioral OF EXOR_tester IS
BEGIN
    a <= '0' AFTER  0 ns,
         '0' AFTER 100 ns,
         '1' AFTER 200 ns,
         '1' AFTER 300 ns;

    b <= '0' AFTER  0 ns,
         '1' AFTER 100 ns,
         '0' AFTER 200 ns,
         '1' AFTER 300 ns;
END behavioral;
```



```
ENTITY ENOR1 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR1;
ARCHITECTURE behavioral OF ENOR1 IS
BEGIN
   c <= a NOR b AFTER 4 ns;
END behavioral;
```

```
ENTITY ENOR2 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR2;
ARCHITECTURE behavioral OF ENOR2 IS
BEGIN
   c <= a NOR b AFTER 4 ns;
END behavioral;
```

```
ENTITY ENOR3 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR3;
ARCHITECTURE behavioral OF ENOR3 IS
BEGIN
   c <= a NOR b AFTER 4 ns;
END behavioral;
```

```
ENTITY ENOR4 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR4;
ARCHITECTURE behavioral OF ENOR4 IS
BEGIN
   c <= a NOR b AFTER 4 ns;
END behavioral;
```

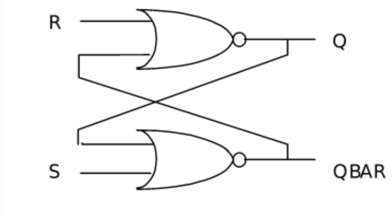
```
ENTITY ENOR5 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR5;
ARCHITECTURE behavioral OF ENOR5 IS
BEGIN
   c <= a NOR b AFTER 4 ns;
END behavioral;
```

```
ENTITY EXOR IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END EXOR;
ARCHITECTURE structure OF EXOR IS
COMPONENT ENOR1 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
COMPONENT ENOR2 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
COMPONENT ENOR3 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
COMPONENT ENOR4 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
COMPONENT ENOR5 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
SIGNAL e1,e2,e3,e4:BIT;
BEGIN
   ENOR1 PORT MAP (a,b,e1);
   ENOR2 PORT MAP (a,e1,e2);
   ENOR3 PORT MAP (b,e1,e3);
   ENOR4 PORT MAP (e2,e3,e4);
   ENOR5 PORT MAP (e4,e4,c);
END structure;
```

```
Entity EXOR_tester IS
   PORT (a,b : OUT BIT ; c : IN BIT);
END EXOR_tester;
ARCHITECTURE behavioral OF EXOR_tester IS
BEGIN
   a <= '0' AFTER  0 ns,
      '0' AFTER 100 ns,
      '1' AFTER 200 ns,
      '1' AFTER 300 ns;

   b <= '0' AFTER  0 ns,
      '1' AFTER 100 ns,
      '0' AFTER 200 ns,
      '1' AFTER 300 ns;
END behavioral;
```

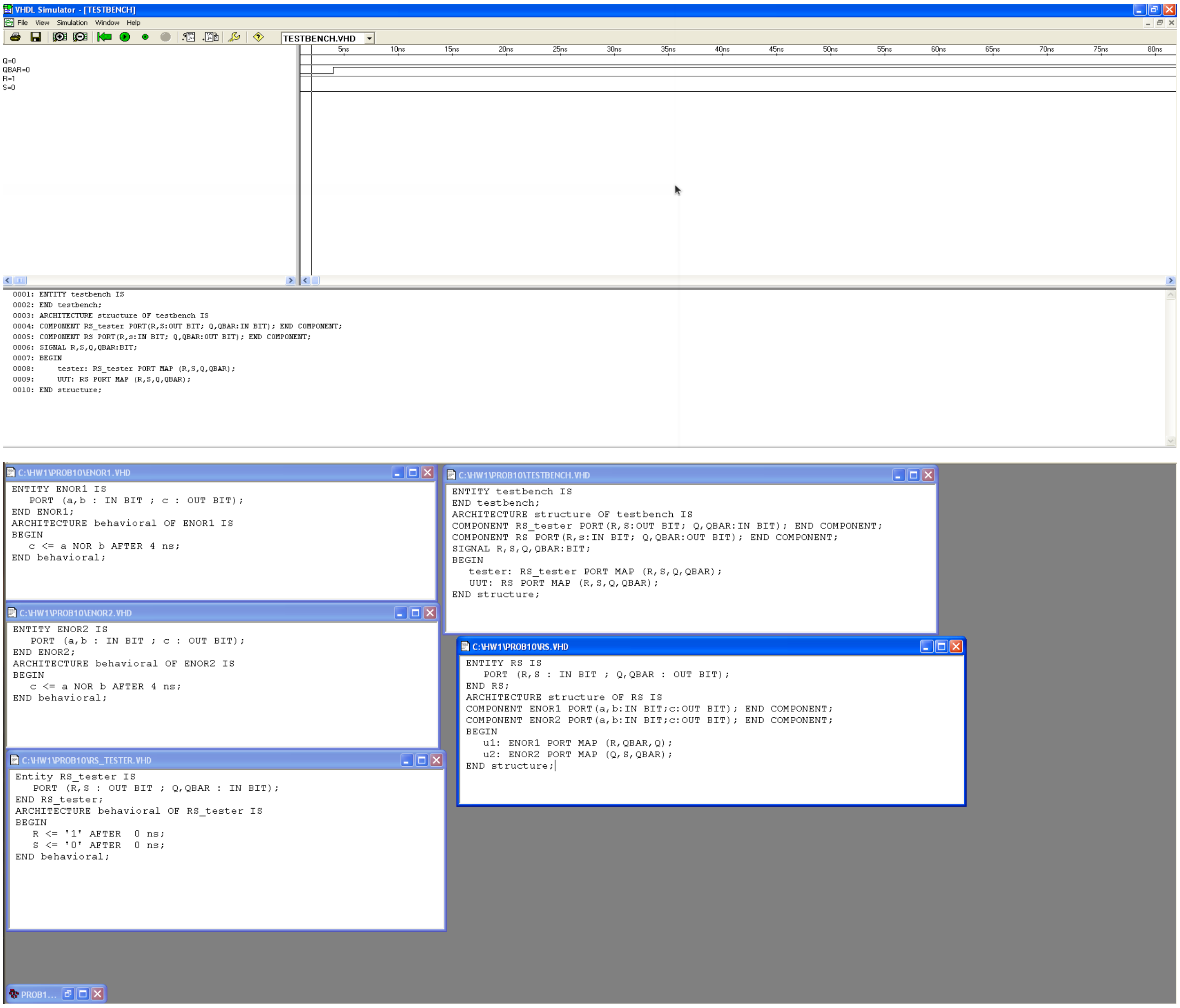
```
ENTITY testbench IS
END testbench;
ARCHITECTURE structure OF testbench IS
COMPONENT EXOR_tester PORT(a,b:OUT BIT; c:IN BIT); END COMPONENT;
COMPONENT EXOR PORT(a,b:IN BIT; c:OUT BIT); END COMPONENT;
SIGNAL a,b,c:BIT;
BEGIN
   tester: EXOR_tester PORT MAP (a,b,c);
   UUT: EXOR PORT MAP (a,b,c);
END structure;
```

10) Using the NOR gates with a 4ns delay as the fundamental component of an RS flip flop, write the entity declaration and structural architecture model which corresponds to the gate-level schematic below. Do not put any initial values on any signals within the model itself.



Write a simple tester and testbench which just initializes R to '1' at simulation time = 0. Do not do anything else to R or S yet. Compile the RS flip flop entity and start the simulator. Verify that the Q output goes to 0 and QBAR goes to 1 after the proper gate delays. [4 pts]

- After 4ns, QBAR turns to 1. Q remain to 0.



```
Q=0
QBAR=0
R=1
S=0
```

```
ENTITY testbench IS
END testbench;
ARCHITECTURE structure OF testbench IS
COMPONENT RS_tester PORT(R,S:OUT BIT; Q,QBAR:IN BIT); END COMPONENT;
COMPONENT RS PORT(R,s:IN BIT; Q,QBAR:OUT BIT); END COMPONENT;
SIGNAL R,S,Q,QBAR:BIT;
BEGIN
    tester: RS_tester PORT MAP (R,S,Q,QBAR);
    UUT: RS PORT MAP (R,S,Q,QBAR);
END structure;
```

C:\HW1\PROB10\ENOR1.VHD

```
ENTITY ENOR1 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR1;
ARCHITECTURE behavioral OF ENOR1 IS
BEGIN
   c <= a NOR b AFTER 4 ns;
END behavioral;
```

C:\HW1\PROB10\TESTBENCH.VHD

```
ENTITY testbench IS
END testbench;
ARCHITECTURE structure OF testbench IS
COMPONENT RS_tester PORT(R,S:OUT BIT; Q,QBAR:IN BIT); END COMPONENT;
COMPONENT RS PORT(R,s:IN BIT; Q,QBAR:OUT BIT); END COMPONENT;
SIGNAL R,S,Q,QBAR:BIT;
BEGIN
   tester: RS_tester PORT MAP (R,S,Q,QBAR);
   UUT: RS PORT MAP (R,S,Q,QBAR);
END structure;
```

C:\HW1\PROB10\ENOR2.VHD

```
ENTITY ENOR2 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR2;
ARCHITECTURE behavioral OF ENOR2 IS
BEGIN
   c <= a NOR b AFTER 4 ns;
END behavioral;
```

C:\HW1\PROB10\RS.VHD

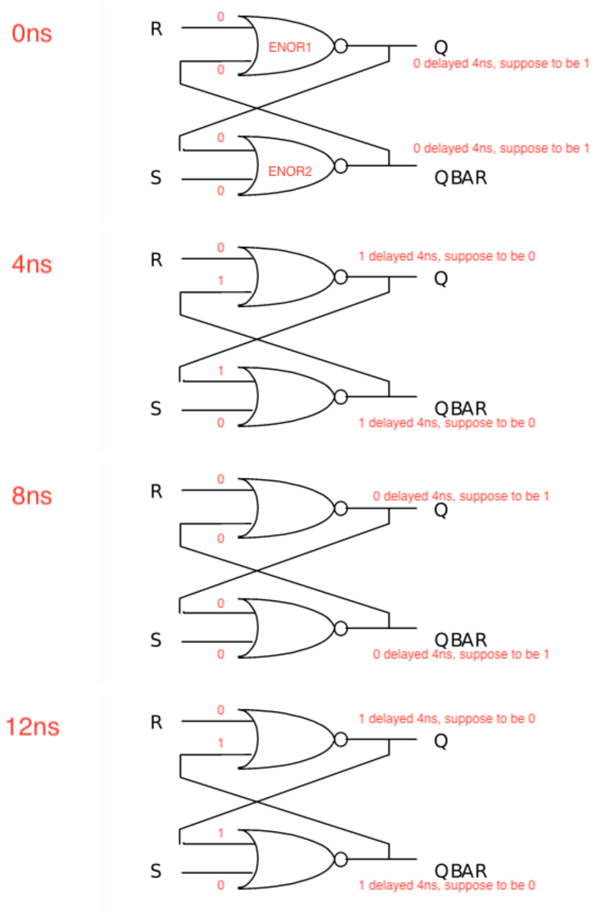
```
ENTITY RS IS
   PORT (R,S : IN BIT ; Q,QBAR : OUT BIT);
END RS;
ARCHITECTURE structure OF RS IS
COMPONENT ENOR1 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
COMPONENT ENOR2 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
BEGIN
   u1: ENOR1 PORT MAP (R,QBAR,Q);
   u2: ENOR2 PORT MAP (Q,S,QBAR);
END structure;
```

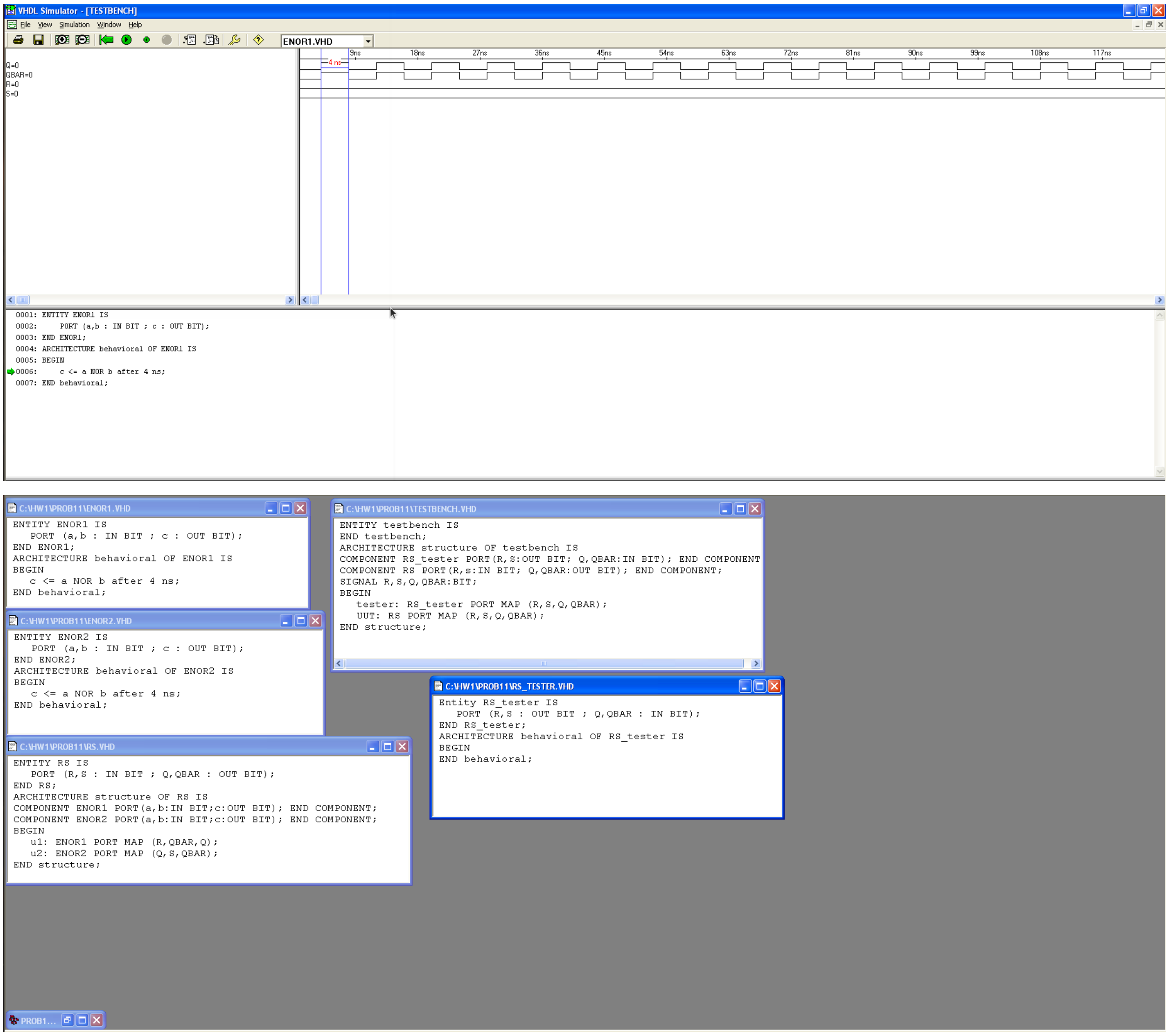
C:\HW1\PROB10\RS_TESTER.VHD

```
Entity RS_tester IS
   PORT (R,S : OUT BIT ; Q,QBAR : IN BIT);
END RS_tester;
ARCHITECTURE behavioral OF RS_tester IS
BEGIN
   R <= '1' AFTER  0 ns;
   S <= '0' AFTER  0 ns;
END behavioral;
```

11) Modify the tester file for problem 10 so that R is not initialized, and neither is S. Without initializing R (or any other signals) to '1' at simulation time = 0, rerun the simulation for problem 10 above so that the simulator is in the default initial startup state (i.e. current sim time = 0 and, for PeakVHDL, all signals = '0'). Explain what happens. Could such a phenomena actually occur in real life using real physical components ? Explain why or why not. [Hint: Think about how a flip-flop works. Consider what simulation is and how it models the world. Then consider what a physical flip-flop would do in the first few moments after power-up, which is what is being modeled by the simulator at sim time = (0 + some very small delta) with all signals initialized to '0'.] [12 pts, based primarily upon the written explanations you provide.]

- The waveform shows there are continuous glitches during the entire simulation.
- The duration of each glitch is 4ns which is exactly equal to 1 NOR gate delay.
- This will happen in real physical components.


0ns
R
0
ENOR1
0
Q
0 delayed 4ns, suppose to be 1
0
0 delayed 4ns, suppose to be 1
ENOR2
S
0
QBAR
4ns
R
0
1
1 delayed 4ns, suppose to be 0
Q
1
S
0
QBAR
1 delayed 4ns, suppose to be 0
8ns
R
0
0 delayed 4ns, suppose to be 1
Q
0
0
S
0
QBAR
0 delayed 4ns, suppose to be 1
12ns
R
0
1
1 delayed 4ns, suppose to be 0
Q
1
S
0
QBAR
1 delayed 4ns, suppose to be 0



```
Q=0
QBAR=0
R=0
S=0
```

```
ENTITY ENOR1 IS
    PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR1;
ARCHITECTURE behavioral OF ENOR1 IS
BEGIN
    c <= a NOR b after 4 ns;
END behavioral;
```

C:\HW1\PROB11\ENOR1.VHD

```
ENTITY ENOR1 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR1;
ARCHITECTURE behavioral OF ENOR1 IS
BEGIN
   c <= a NOR b after 4 ns;
END behavioral;
```

C:\HW1\PROB11\TESTBENCH.VHD

```
ENTITY testbench IS
END testbench;
ARCHITECTURE structure OF testbench IS
COMPONENT RS_tester PORT(R,S:OUT BIT; Q,QBAR:IN BIT); END COMPONENT
COMPONENT RS PORT(R,s:IN BIT; Q,QBAR:OUT BIT); END COMPONENT;
SIGNAL R,S,Q,QBAR:BIT;
BEGIN
   tester: RS_tester PORT MAP (R,S,Q,QBAR);
   UUT: RS PORT MAP (R,S,Q,QBAR);
END structure;
```

C:\HW1\PROB11\ENOR2.VHD

```
ENTITY ENOR2 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR2;
ARCHITECTURE behavioral OF ENOR2 IS
BEGIN
   c <= a NOR b after 4 ns;
END behavioral;
```

C:\HW1\PROB11\RS_TESTER.VHD

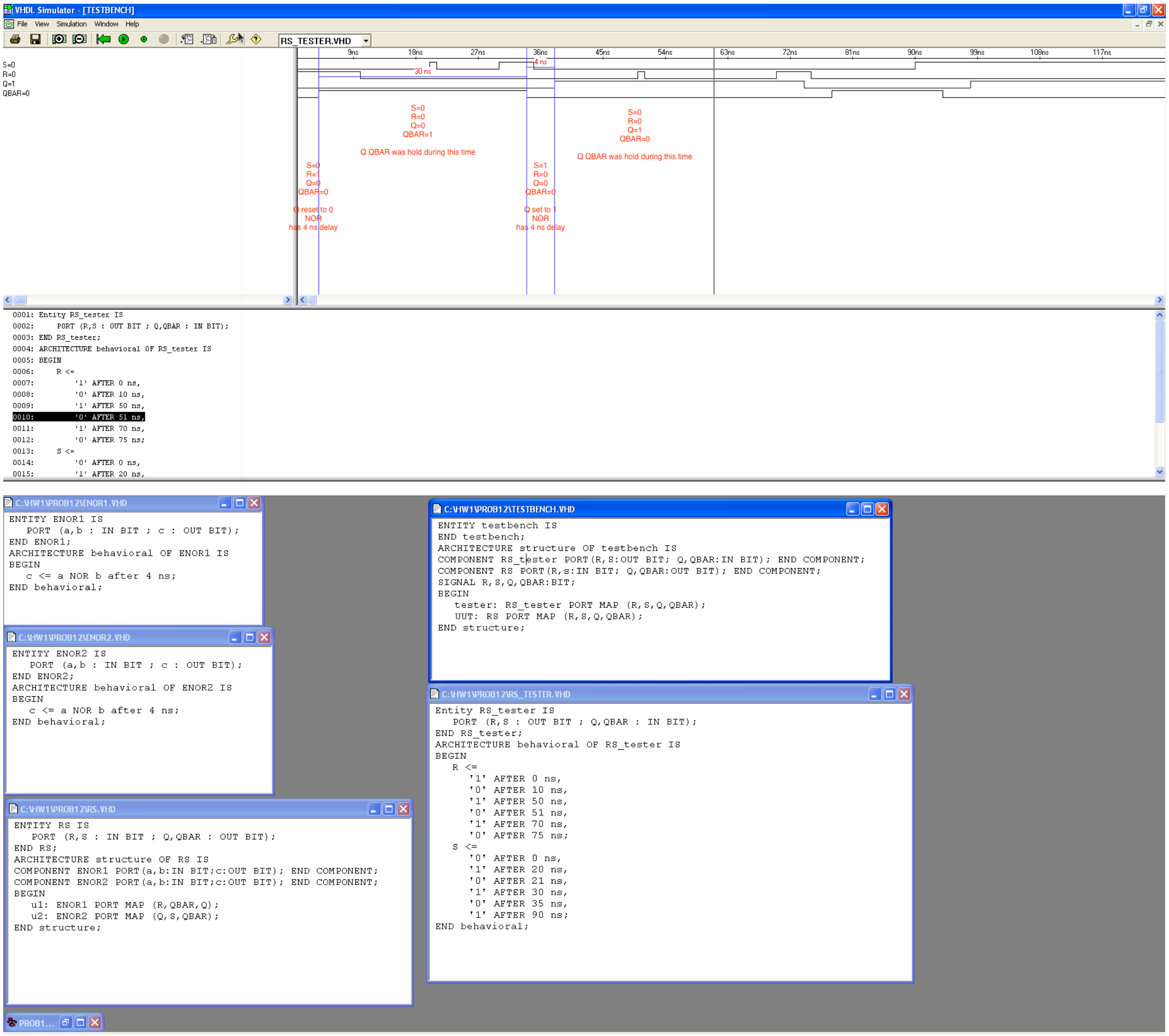
```
Entity RS_tester IS
   PORT (R,S : OUT BIT ; Q,QBAR : IN BIT);
END RS_tester;
ARCHITECTURE behavioral OF RS_tester IS
BEGIN
END behavioral;
```

C:\HW1\PROB11\RS.VHD

```
ENTITY RS IS
   PORT (R,S : IN BIT ; Q,QBAR : OUT BIT);
END RS;
ARCHITECTURE structure OF RS IS
COMPONENT ENOR1 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
COMPONENT ENOR2 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
BEGIN
   u1: ENOR1 PORT MAP (R,QBAR,Q);
   u2: ENOR2 PORT MAP (Q,S,QBAR);
END structure;
```

12) Write an entity declaration and behavioral specification for a test generation circuit appropriate for use on the RS flip flop of problem 10 that will generate the following waveform. Verify that the RS flip flop can be set and reset by R and S. As with any flip-flop, note that at any given time, only either R or S is '1'. List all input and output signals of the UUT to verify proper operation. Explain the UUT's response when either the set or reset inputs are 1 ns pulses. [5 pts]

- The waveform shows both Q and QBAR are not changed during that 1 ns pulses.
- The NOR gate has an inertial delay of 4ns, because R and S inputs only hold for 1 ns, the NOR gate will not change.
- Value on inputs must persist for a given time before output responds.



```
Entity RS_tester IS
     PORT (R,S : OUT BIT ; Q,QBAR : IN BIT);
END RS_tester;
ARCHITECTURE behavioral OF RS_tester IS
BEGIN
     R <=
          '1' AFTER 0 ns,
          '0' AFTER 10 ns,
          '1' AFTER 50 ns,
          '0' AFTER 51 ns,
          '1' AFTER 70 ns,
          '0' AFTER 75 ns;
     S <=
          '0' AFTER 0 ns,
          '1' AFTER 20 ns,
```

```
ENTITY ENOR1 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR1;
ARCHITECTURE behavioral OF ENOR1 IS
BEGIN
   c <= a NOR b after 4 ns;
END behavioral;
```

```
ENTITY ENOR2 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR2;
ARCHITECTURE behavioral OF ENOR2 IS
BEGIN
   c <= a NOR b after 4 ns;
END behavioral;
```

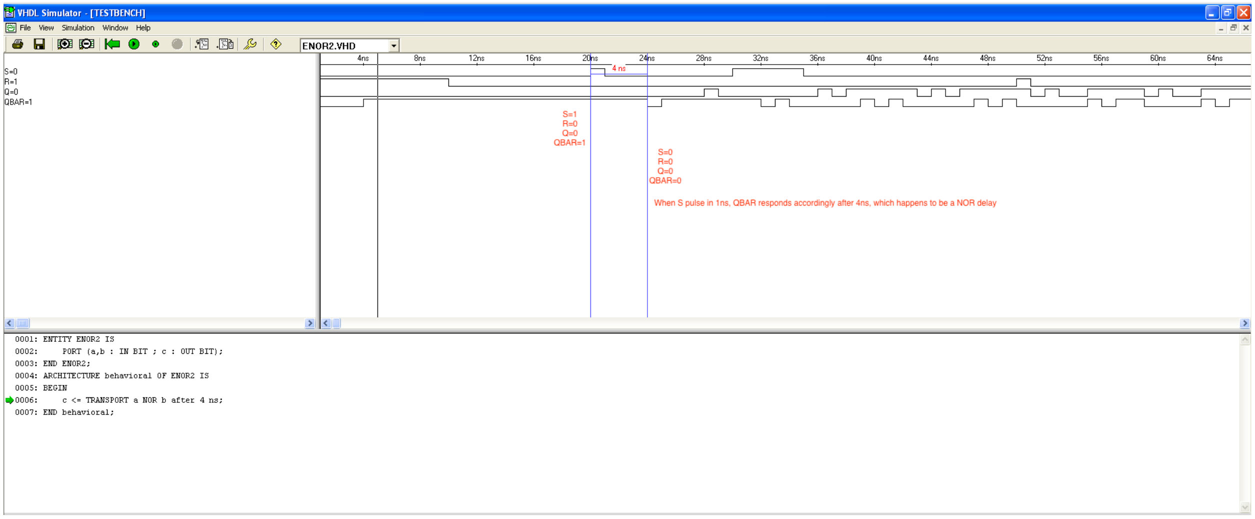
```
ENTITY RS IS
   PORT (R,S : IN BIT ; Q,QBAR : OUT BIT);
END RS;
ARCHITECTURE structure OF RS IS
COMPONENT ENOR1 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
COMPONENT ENOR2 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
BEGIN
   u1: ENOR1 PORT MAP (R,QBAR,Q);
   u2: ENOR2 PORT MAP (Q,S,QBAR);
END structure;
```

```
ENTITY testbench IS
END testbench;
ARCHITECTURE structure OF testbench IS
COMPONENT RS_tester PORT(R,S:OUT BIT; Q,QBAR:IN BIT); END COMPONENT;
COMPONENT RS PORT(R,s:IN BIT; Q,QBAR:OUT BIT); END COMPONENT;
SIGNAL R,S,Q,QBAR:BIT;
BEGIN
   tester: RS_tester PORT MAP (R,S,Q,QBAR);
   UUT: RS PORT MAP (R,S,Q,QBAR);
END structure;
```

```
Entity RS_tester IS
   PORT (R,S : OUT BIT ; Q,QBAR : IN BIT);
END RS_tester;
ARCHITECTURE behavioral OF RS_tester IS
BEGIN
   R <=
      '1' AFTER 0 ns,
      '0' AFTER 10 ns,
      '1' AFTER 50 ns,
      '0' AFTER 51 ns,
      '1' AFTER 70 ns,
      '0' AFTER 75 ns;
   S <=
      '0' AFTER 0 ns,
      '1' AFTER 20 ns,
      '0' AFTER 21 ns,
      '1' AFTER 30 ns,
      '0' AFTER 35 ns,
      '1' AFTER 90 ns;
END behavioral;
```

13) Repeat the simulation performed as part of problem 12 using TRANSPORT delay (instead of the default inertial delay) on the NOR gates of the RS flip flop. Explain how and why the UUT's response is different from the inertial delay model. [5 pts]

- Tester has 1 ns pulse, totally 16 ns simulation time.
- The waveform shows both Q and QBAR are responded as expected in the time period of 4 to 20 ns.
- The NOR gate has a transport delay of 4ns, So both Q and QBAR signal waves will shift to the right to 4 ns.
- Output always changes regardless of the time duration of the input signal.

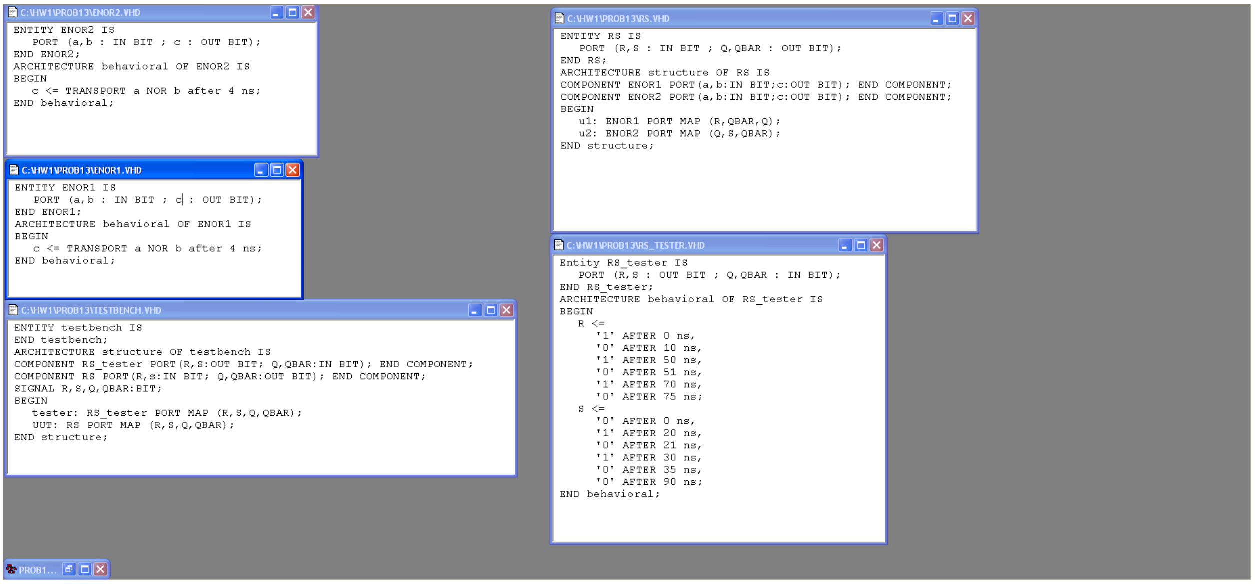


S=1
R=0
Q=0
QBAR=1

S=0
R=0
Q=0
QBAR=0

When S pulse in 1ns, QBAR responds accordingly after 4ns, which happens to be a NOR delay

```
ENTITY ENOR2 IS
    PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR2;
ARCHITECTURE behavioral OF ENOR2 IS
BEGIN
    c <= TRANSPORT a NOR b after 4 ns;
END behavioral;
```



```
ENTITY ENOR2 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR2;
ARCHITECTURE behavioral OF ENOR2 IS
BEGIN
   c <= TRANSPORT a NOR b after 4 ns;
END behavioral;
```

```
ENTITY ENOR1 IS
   PORT (a,b : IN BIT ; c : OUT BIT);
END ENOR1;
ARCHITECTURE behavioral OF ENOR1 IS
BEGIN
   c <= TRANSPORT a NOR b after 4 ns;
END behavioral;
```

```
ENTITY testbench IS
END testbench;
ARCHITECTURE structure OF testbench IS
COMPONENT RS_tester PORT(R,S:OUT BIT; Q,QBAR:IN BIT); END COMPONENT;
COMPONENT RS PORT(R,s:IN BIT; Q,QBAR:OUT BIT); END COMPONENT;
SIGNAL R,S,Q,QBAR:BIT;
BEGIN
   tester: RS_tester PORT MAP (R,S,Q,QBAR);
   UUT: RS PORT MAP (R,S,Q,QBAR);
END structure;
```

```
ENTITY RS IS
   PORT (R,S : IN BIT ; Q,QBAR : OUT BIT);
END RS;
ARCHITECTURE structure OF RS IS
COMPONENT ENOR1 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
COMPONENT ENOR2 PORT(a,b:IN BIT;c:OUT BIT); END COMPONENT;
BEGIN
   u1: ENOR1 PORT MAP (R,QBAR,Q);
   u2: ENOR2 PORT MAP (Q,S,QBAR);
END structure;
```

```
Entity RS_tester IS
   PORT (R,S : OUT BIT ; Q,QBAR : IN BIT);
END RS_tester;
ARCHITECTURE behavioral OF RS_tester IS
BEGIN
   R <=
      '1' AFTER 0 ns,
      '0' AFTER 10 ns,
      '1' AFTER 50 ns,
      '0' AFTER 51 ns,
      '1' AFTER 70 ns,
      '0' AFTER 75 ns;
   S <=
      '0' AFTER 0 ns,
      '1' AFTER 20 ns,
      '0' AFTER 21 ns,
      '1' AFTER 30 ns,
      '0' AFTER 35 ns,
      '0' AFTER 90 ns;
END behavioral;
```

14) Perform the following 4-bit binary additions. The numbers are in 2's complement form. Which ones produce an overflow, and how should the answer be interpreted knowing that an overflow occurred. [4 pts]

0101	0101	1110	0011
<u>0100</u>	<u>1100</u>	<u>1100</u>	<u>0100</u>
1001	0001	1010	0111
overflow			

If two numbers are added, and they are both positive or both negative, then overflow occurs if and only if the result has the opposite sign.

15) Compute the decimal value which corresponds to the following binary numbers using the various representation schemes listed. Assume all of the numbers are positive (no sign bit is used). Recall that Modified Booth's is based on a radix four recoding. [6 pts]

Representation Scheme	Binary value	Decimal value
Floating Point:	1011.01	11.25
A four bit, base 2, biased 8 exponent:	.100 E 1001	**2**
Normalized notation and Phantom 1:	.100	0.75
Booth's Notation:	+1 0 -1	3
Modified Booth's Notation:	+1 -1	3
Modified Booth's Notation:	+2 -1	7

16) Give the base 10 value of the following 4-bit binary numbers using the three number representation schemes. Distinguish between +0 and -0. [2 pts]

Binary	Signed Magnitude	1's Complement	2's Complement
1111	-7	-0	-1
1000	-0	-7	-8
0000	+0	+0	0
1011	-3	-4	-5
0111	+7	7	7

17) Assume the following two numbers are normalized. They are shown below with their mantissa values in binary and their exponent values in decimal. Perform a floating point addition

showing the pre-addition denormalization step and the post-addition renormalization step. [2 pts]   X = .10011 E 2 Y = .11101 E 6

```
          X = .10011 E 2
          X = .00001 E 6
          Y = .11101 E 6
+         ----------------------
          S = .11110 E 6
```

18) Determine an algorithm that could be used for detecting an overflow by comparing the sign of the sum with the signs of the augend and addend. Assume the numbers are in signed-2's complement representation. Explain in words how and why the algorithm works. Next, formulate a hardware implementation of the scheme by drawing a simple schematic. [4 pts]

If two numbers are added, and they are both positive or both negative, then overflow occurs if and only if the result has the opposite sign.

The first condition is that the augend and addend has to be the same positive or negative.

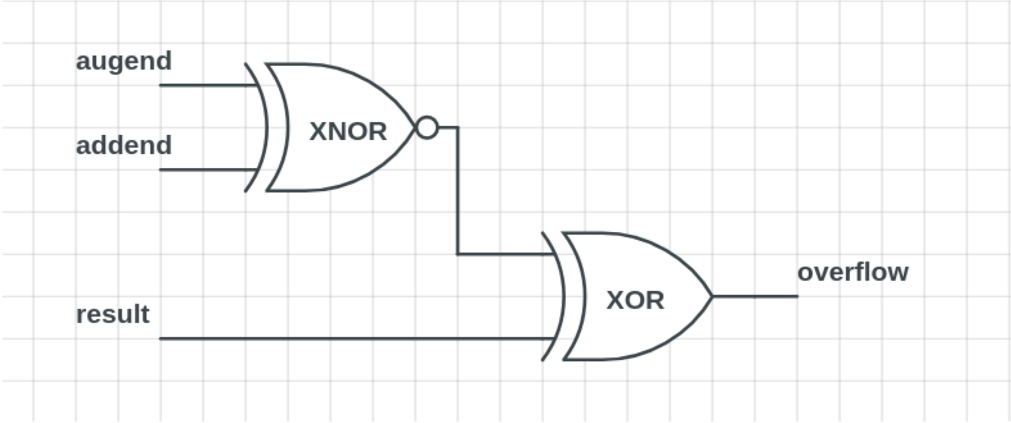
if positive value stand for 0 and negative value stand for 1, the truth table would be:

augend	**addend**	**condition**
1	**1**	**1**
1	**0**	**0**
0	**1**	**0**
0	**0**	**1**

It is a xnor gate. The second condition is that the result has the opposite sign. the truth table would be:

condition1	**result**	**overflow**
1	**1**	**0**
1	**0**	**1**
0	**1**	**1**
0	**0**	**0**

It is a xor gate. combining xnor and xor, if overflow is 1, then it is overflow.



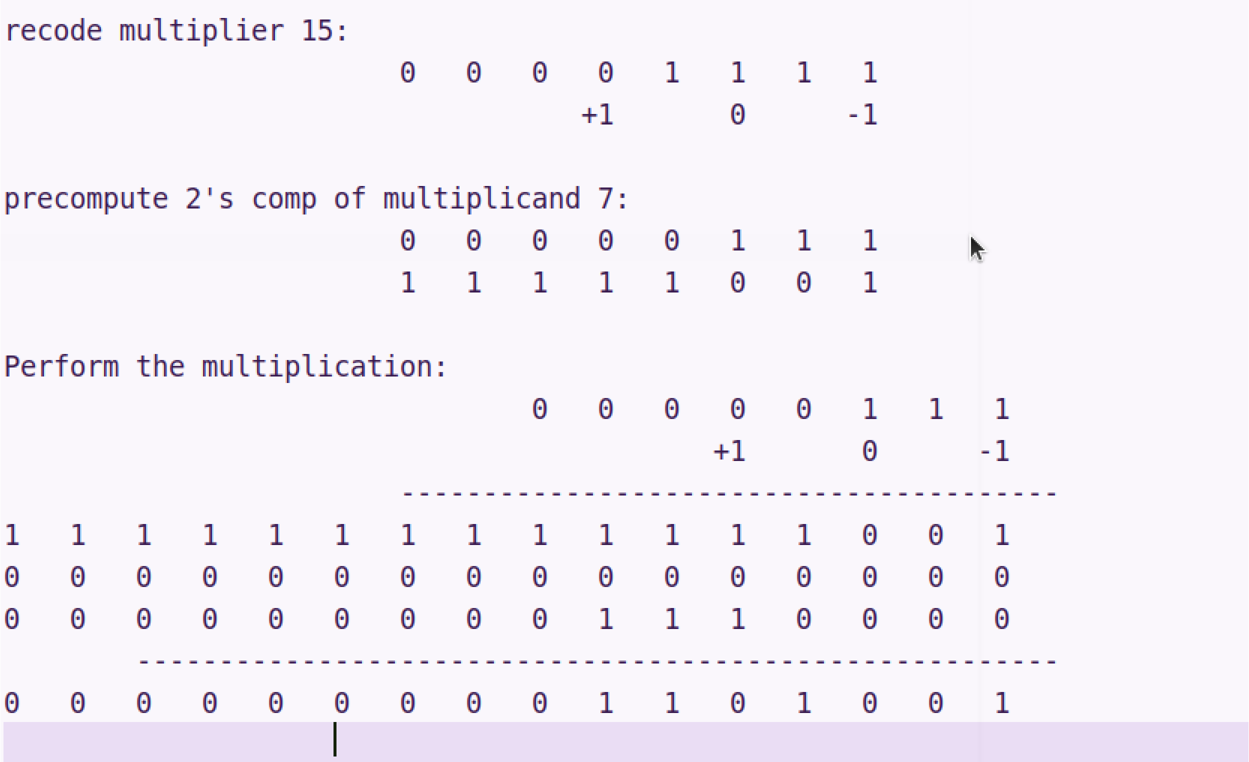
19) Perform the multiplication 7 x 15 where you are given 7 as the multiplicand and 15 as the

multiplier. Both original operands are 8 bit numbers. Use unsigned, ordinary binary notation and the ordinary multiplication algorithm similar to that shown in the class notes. [4 pts]

```
                        0   0   0   0   0   1   1   1
                        0   0   0   0   1   1   1   1
---------------------------------------------------------------
                                        0   1   1   1
                                    0   1   1   1
                                0   1   1   1
                            0   1   1   1
---------------------------------------------------------------
                        0   1   1   0   1   0   0   1
```

The answer is 01101001.

20) Perform the multiplication 7 x 15 where you are given 7 as the multiplicand and 15 as the multiplier. Both original operands are 8 bit numbers. Use Modified Booth's Algorithm to recode the multiplier. Show all partial products that are generated along with any sign extension which needs to be done in order to produce a product that will fit into a 16 bit register. Use 2's complement representation and math. [5 pts]

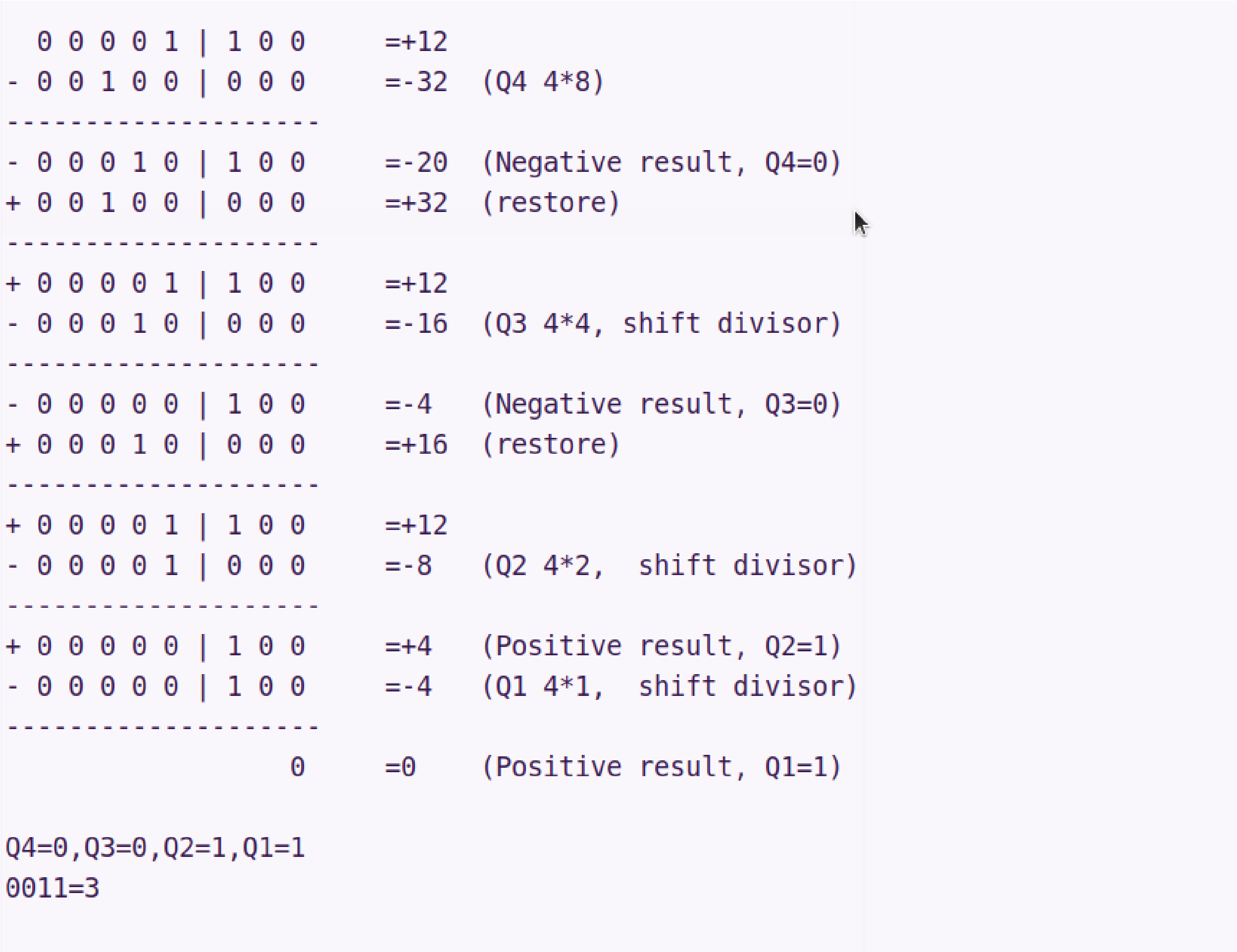


```
recode multiplier 15:
                        0   0   0   0   1   1   1   1
                                   +1       0      -1

precompute 2's comp of multiplicand 7:
                        0   0   0   0   0   1   1   1
                        1   1   1   1   1   0   0   1

Perform the multiplication:
                                0   0   0   0   0   1   1   1
                                           +1       0      -1
                        ----------------------------------------
1   1   1   1   1   1   1   1   1   1   1   1   1   0   0   1
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   1   1   1   0   0   0   0
        --------------------------------------------------------
0   0   0   0   0   0   0   0   0   1   1   0   1   0   0   1
```

because it is 16 bits 2's comp, so the answer is 00000000 01101001.

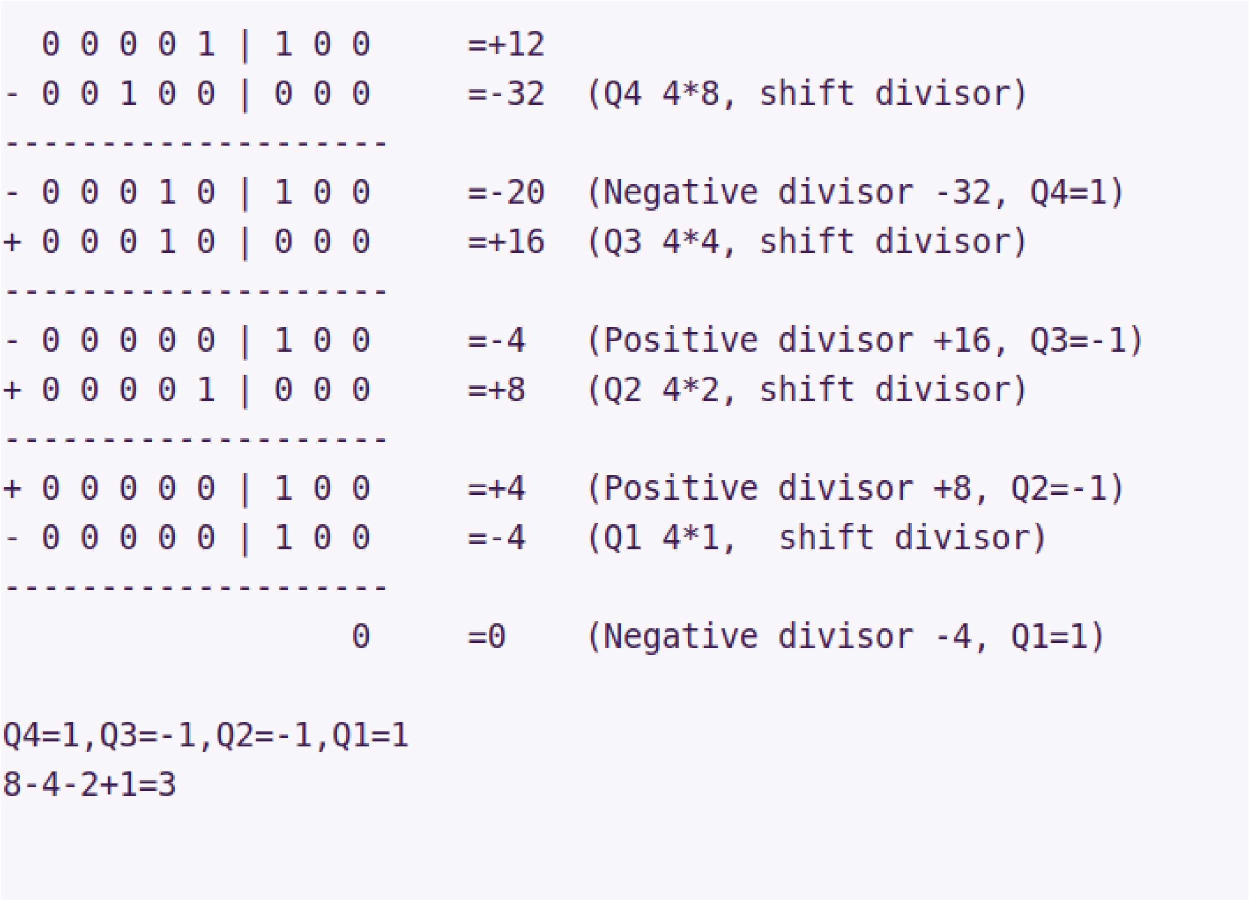
21) Perform 12 divided by 4 using the restoring division algorithm in binary. Show and explain all

steps involved by using a running commentary in decimal similar to that used in the class notes. Compute a 4 bit answer. [5 pts]



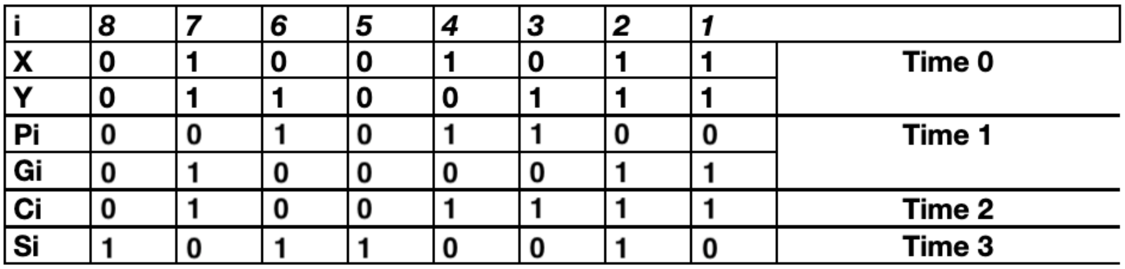
```
  0 0 0 0 1 | 1 0 0     =+12
- 0 0 1 0 0 | 0 0 0     =-32  (Q4 4*8)
-------------------
- 0 0 0 1 0 | 1 0 0     =-20  (Negative result, Q4=0)
+ 0 0 1 0 0 | 0 0 0     =+32  (restore)
-------------------
+ 0 0 0 0 1 | 1 0 0     =+12
- 0 0 0 1 0 | 0 0 0     =-16  (Q3 4*4, shift divisor)
-------------------
- 0 0 0 0 0 | 1 0 0     =-4   (Negative result, Q3=0)
+ 0 0 0 1 0 | 0 0 0     =+16  (restore)
-------------------
+ 0 0 0 0 1 | 1 0 0     =+12
- 0 0 0 0 1 | 0 0 0     =-8   (Q2 4*2,  shift divisor)
-------------------
+ 0 0 0 0 0 | 1 0 0     =+4   (Positive result, Q2=1)
- 0 0 0 0 0 | 1 0 0     =-4   (Q1 4*1,  shift divisor)
-------------------
                  0     =0    (Positive result, Q1=1)

Q4=0,Q3=0,Q2=1,Q1=1
0011=3
```

22) Perform 12 divided by 4 using the non-restoring division algorithm in binary. Show and explain all steps involved by using a running commentary in decimal similar to that used in the class notes. Compute a 4 bit answer. [5 pts]



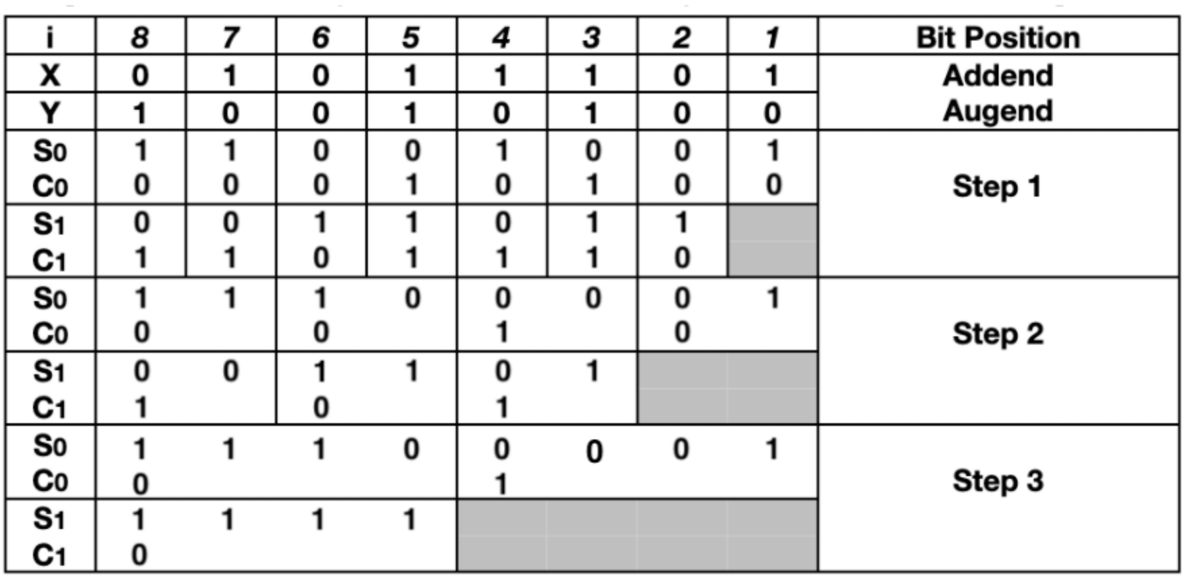
```
  0 0 0 0 1 | 1 0 0     =+12
- 0 0 1 0 0 | 0 0 0     =-32  (Q4 4*8, shift divisor)
--------------------
- 0 0 0 1 0 | 1 0 0     =-20  (Negative divisor -32, Q4=1)
+ 0 0 0 1 0 | 0 0 0     =+16  (Q3 4*4, shift divisor)
--------------------
- 0 0 0 0 0 | 1 0 0     =-4   (Positive divisor +16, Q3=-1)
+ 0 0 0 0 1 | 0 0 0     =+8   (Q2 4*2, shift divisor)
--------------------
+ 0 0 0 0 0 | 1 0 0     =+4   (Positive divisor +8, Q2=-1)
- 0 0 0 0 0 | 1 0 0     =-4   (Q1 4*1,  shift divisor)
--------------------
                  0     =0    (Negative divisor -4, Q1=1)

Q4=1,Q3=-1,Q2=-1,Q1=1
8-4-2+1=3
```

23) Compute the proper entries for the table below which shows a carry lookahead adder implementation to compute the sum of X and Y as given. Assume that the carry into the LSB is zero. [4 pts]



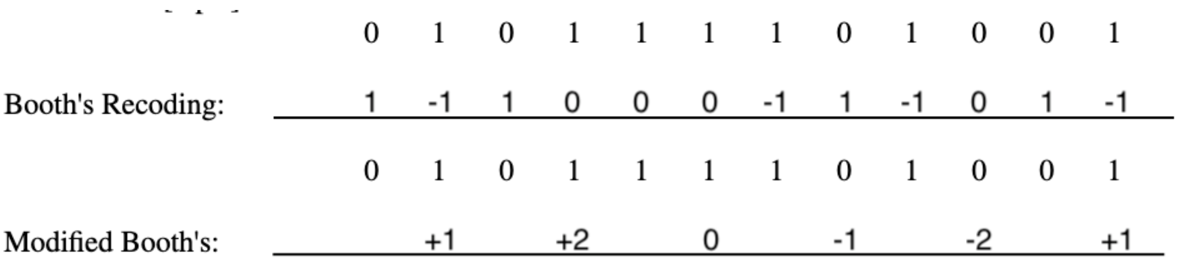
i	8	7	6	5	4	3	2	1	
X	0	1	0	0	1	0	1	1	Time 0
Y	0	1	1	0	0	1	1	1	
Pi	0	0	1	0	1	1	0	0	Time 1
Gi	0	1	0	0	0	0	1	1	
Ci	0	1	0	0	1	1	1	1	Time 2
Si	1	0	1	1	0	0	1	0	Time 3

24) Compute the proper entries for the table below which shows a carry select adder implementation to compute the sum of two eight bit numbers, X and Y as given. Assume that C0 = 0. [8 pts]



i	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	**Bit Position**
X	0	1	0	1	1	1	0	1	**Addend**
Y	1	0	0	1	0	1	0	0	**Augend**
S0	1	1	0	0	1	0	0	1	**Step 1**
C0	0	0	0	1	0	1	0	0	
S1	0	0	1	1	0	1	1		
C1	1	1	0	1	1	1	0		
S0	1	1	1	0	0	0	0	1	**Step 2**
C0	0		0		1		0		
S1	0	0	1	1	0	1			
C1	1		0		1				
S0	1	1	1	0	0	0	0	1	**Step 3**
C0	0				1				
S1	1	1	1	1					
C1	0								

25) Using the reference tables for Booth's and Modified Booth's representation, recode the two binary numbers below. [2 pts]



0 1 0 1 1 1 1 0 1 0 0 1

Booth's Recoding: 1 -1 1 0 0 0 -1 1 -1 0 1 -1

0 1 0 1 1 1 1 0 1 0 0 1

Modified Booth's: +1 +2 0 -1 -2 +1