

Laboratory 7  
Addition/Subtraction Circuits - Cadence Verilog

This experiment will allow you to become more familiar with the use of the Cadence Verilog simulation environment. You should print out this experiment before going to the lab so that you can obtain the lab instructor's initials where needed.

PART 1: In this part, we will prepare a User-Defined Primitive (UDP) description of a full-adder. Then we will test the UDP and obtain the lab instructor's initials below after demonstrating the output waveform file. To help you, I am giving you most of the file and it will be your job to fill in the rest. Consider the following file (you can get a copy directly from the website, file **fadd.v**):

```
// UDP for full-adder carry-out function
primitive coutprm (cout, a, b, cin);
    output cout;
    input  a, b, cin;
// Truth table for the carry-out function
    table
//   a b cin : cout
    0 0 0  :  0;
    0 0 1  :  0;
    0 1 0  :  0;
    0 1 1  :  1;
    1 0 0  :  0;
    1 0 1  :  1;
    1 1 0  :  1;
    1 1 1  :  1;
    endtable
endprimitive
```

```
// UDP for full-adder sum function
primitive sumprm (sum, a, b, cin);
    output sum;
    input  a, b, cin;
// Truth table for the sum function
    table
//   a b cin : sum
```

**Fill in the 8 lines of Verilog here**

```
    endtable
endprimitive
```

```
// Model a full-adder
module fadd (cout, sum, a, b, cin);
    output cout, sum;
    input  a, b, cin;
    wire  a, b, cin; //Not necessary default type is wire
    wire  cout, sum; //Not necessary default type is wire
```

```

    coutprm u1 (Insert the proper nets here);
    sumprm u2 (Insert the proper nets here);
endmodule

// Testbench for full-adder
module testfadd;
    reg a, b, cin;
    wire cout, sum;
    fadd u1 (cout, sum, a, b, cin);
    initial
        begin
            $stop;
            #10;
            a=1'b0; b=1'b0; cin=1'b0;
            #10;
            a=1'b0; b=1'b0; cin=1'b1;
            #10;
            a=1'b0; b=1'b1; cin=1'b0;
            #10;
            a=1'b0; b=1'b1; cin=1'b1;
            #10;
            a=1'b1; b=1'b0; cin=1'b0;
            #10;
            a=1'b1; b=1'b0; cin=1'b1;
            #10;
            a=1'b1; b=1'b1; cin=1'b0;
            #10;
            a=1'b1; b=1'b1; cin=1'b1;
            #10;
            $stop;
        end
endmodule

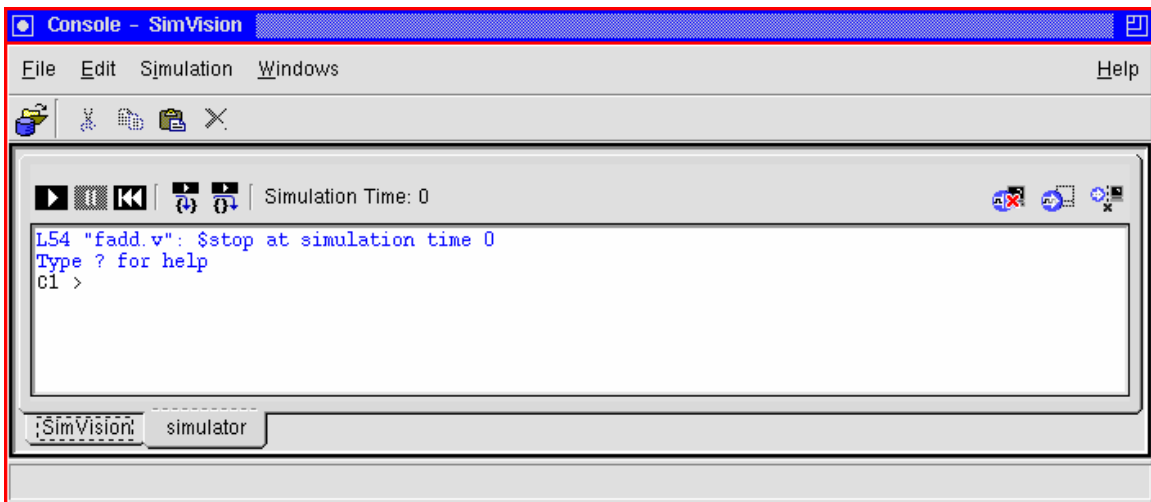
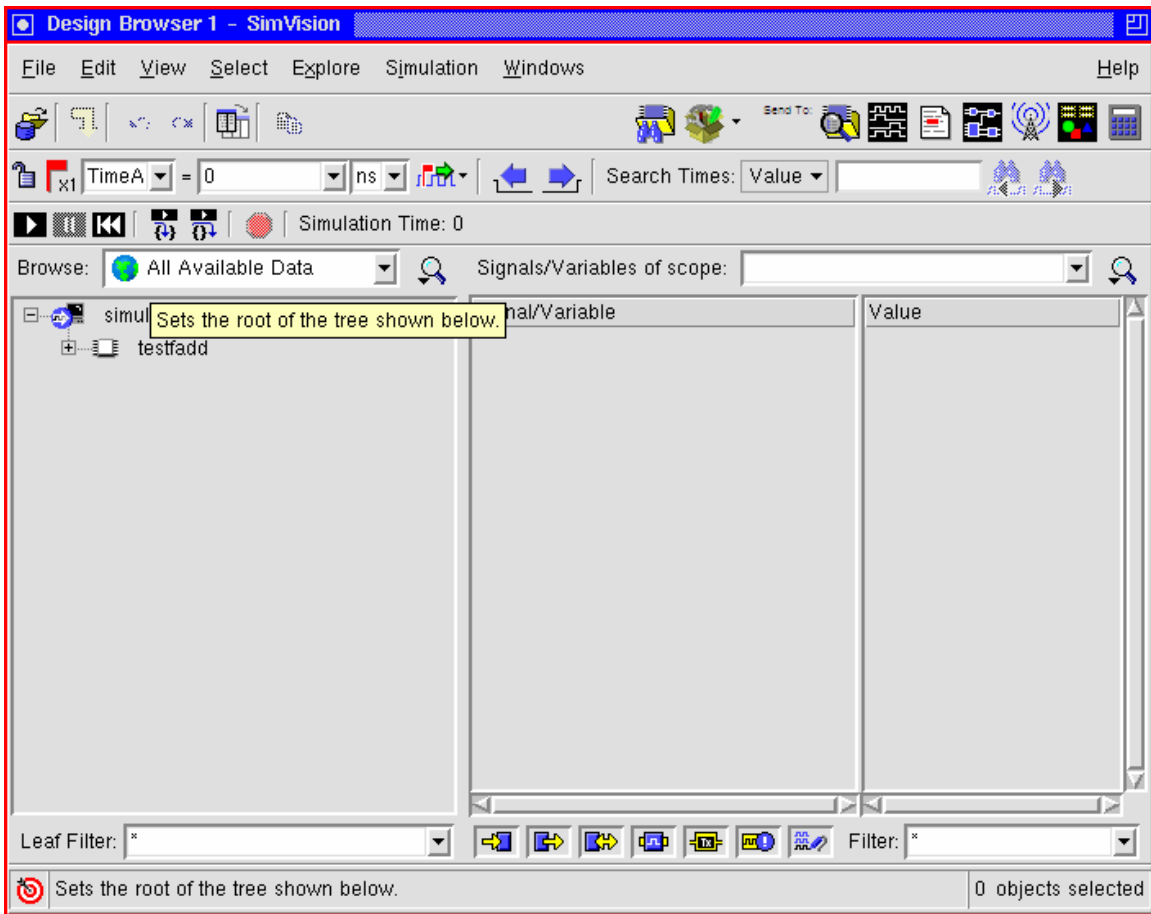
```

It is your job to complete the truth table for the UDP named `sumprm` and to put the proper nets in the two instantiations of the primitives `coutprm` and `sumprm` in the `fadd` module.

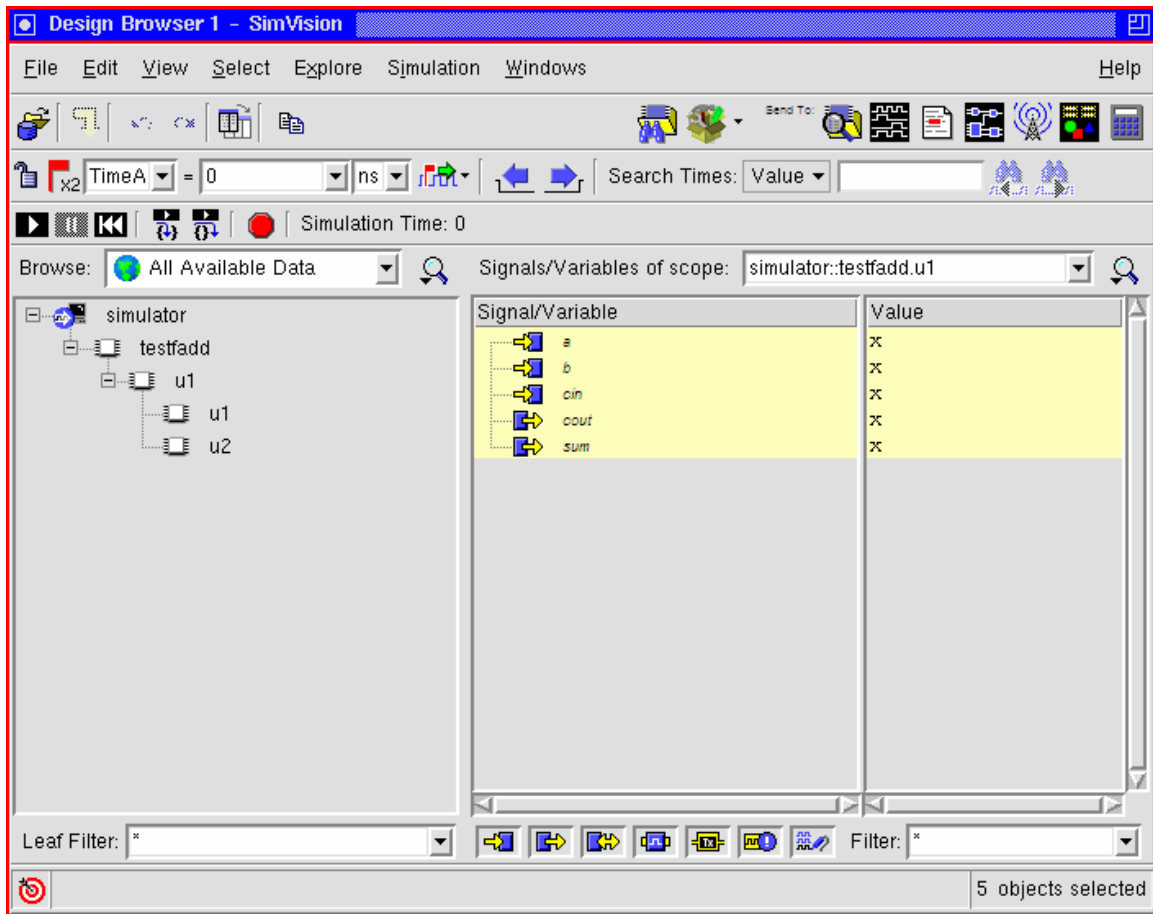
After making the changes to your Verilog code (and saving the file under the name `fadd.v`), invoke the simulator by entering:

```
verilog +gui fadd.v
```

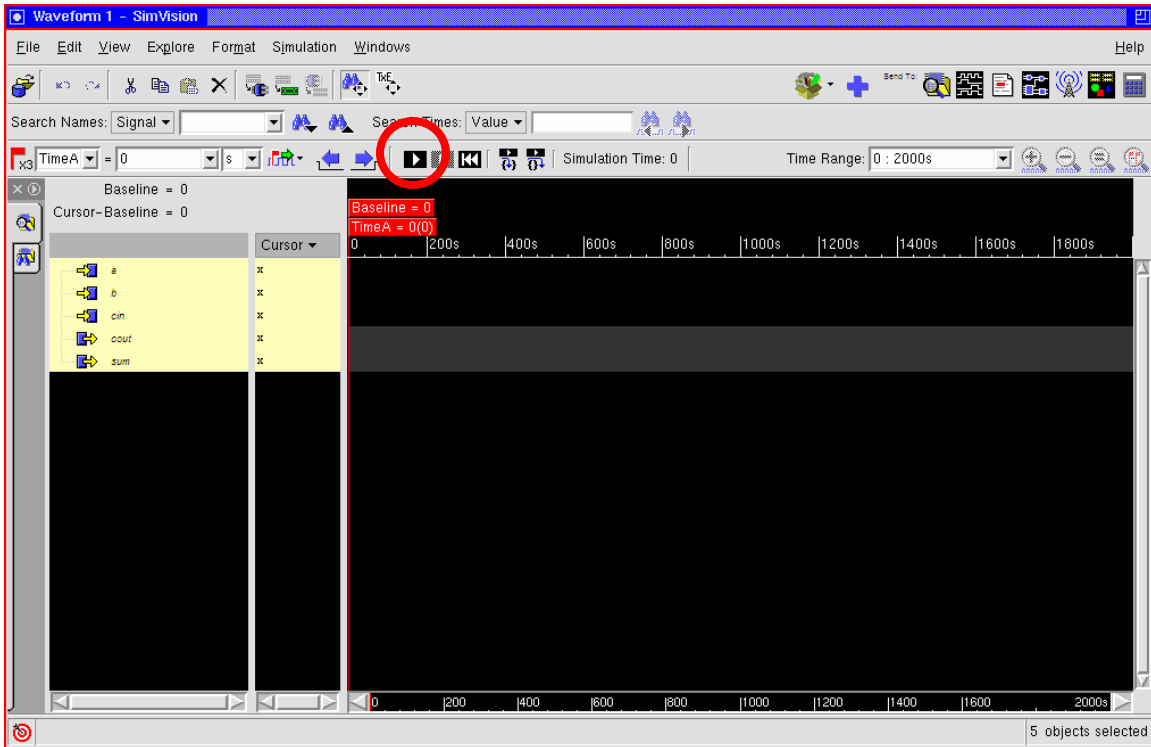
If you have no syntax errors in your code, you should see two windows that look similar to this:



Note that execution has stopped on line 54 as indicated by the message in the console window. You can now select your signals to be displayed. This can be done by using the SimVision window as you did in the previous experiment. Click on the `fadd` icon and then the `u1` icon to open up the scope. After clicking on `u1`, you should see all the signals. Then select them with the mouse. Your SimVision window should appear as:

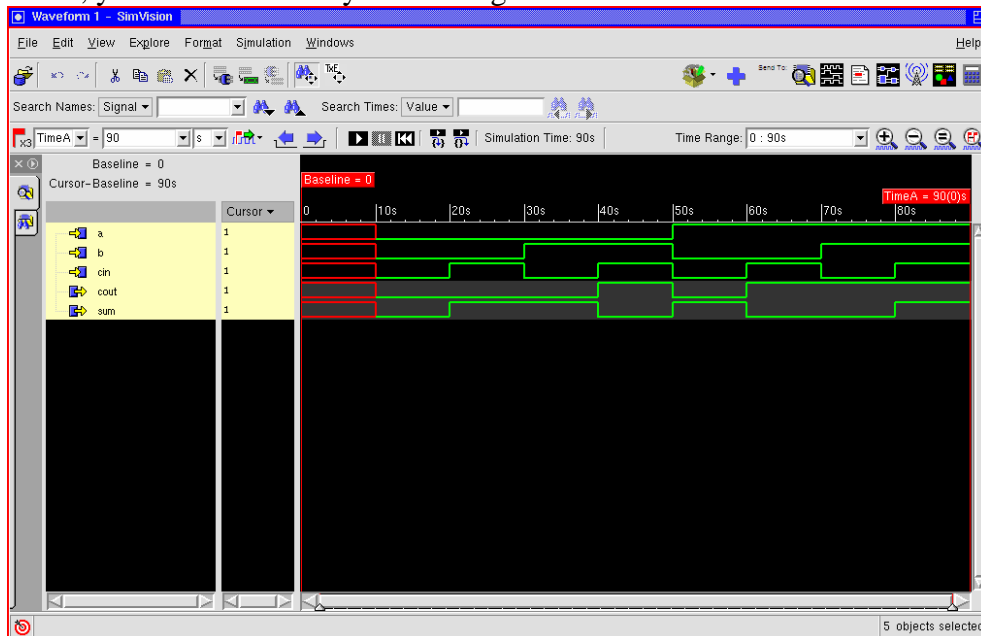


Next, open the waveform window by clicking on the waveform icon. You should get a window that looks like the following:



Next, we should start the waveform viewer. This can be done by clicking on the upper right-hand button with the little square-waves on it, or by selecting it from the “Tools” button. After starting the waveform viewer, the following window will open:

We are now ready to resume the simulation. To do this click on the small triangle button just above the two red boxes on the waveform window – it is circled in red in the screen capture above. You should see a result that is similar to the following. If you do not have this window, you should correct your Verilog code and re-simulate.



Once you have this window, get a screen capture of it (alt-prtscr in Windows) and save it to a file for your final report. Also, have the lab instructor look at it and initial below.

LAB INSTRUCTOR'S INITIALS: \_\_\_\_\_

PART 2: You will design a 4-bit ripple adder in Verilog and simulate it. As before, I am giving you a portion of the code including the testbench. This code is as follows (and can be downloaded from the class webpage, the file named **ripple4.v**).

```
// UDP for full-adder carry-out function
primitive coutprm (cout, a, b, cin);
    output cout;
    input  a, b, cin;
// Truth table for the carry-out function
    table
//   a b cin  :  cout
      0 0  0  :   0;
      0 0  1  :   0;
      0 1  0  :   0;
      0 1  1  :   1;
      1 0  0  :   0;
      1 0  1  :   1;
      1 1  0  :   1;
      1 1  1  :   1;
    endtable
endprimitive

// UDP for full-adder sum function
primitive sumprm (sum, a, b, cin);
    output sum;
    input  a, b, cin;
// Truth table for the sum function
    table
//   a b cin  :  sum

    endtable
endprimitive

// Model a full-adder
module fadd (cout, sum, a, b, cin);
    output cout, sum;
    input  a, b, cin;
    wire  a, b, cin; //Not necessary default type is wire
    wire  cout, sum; //Not necessary default type is wire
    coutprm u1 (Your code from part 1 is here);
    sumprm  u2 (Your code from part 1 is here);
endmodule

// Model a 4-bit ripple adder
module ripple4(cout, sum, a, b, cin);
    output [3:0] sum;
    output cout;
```

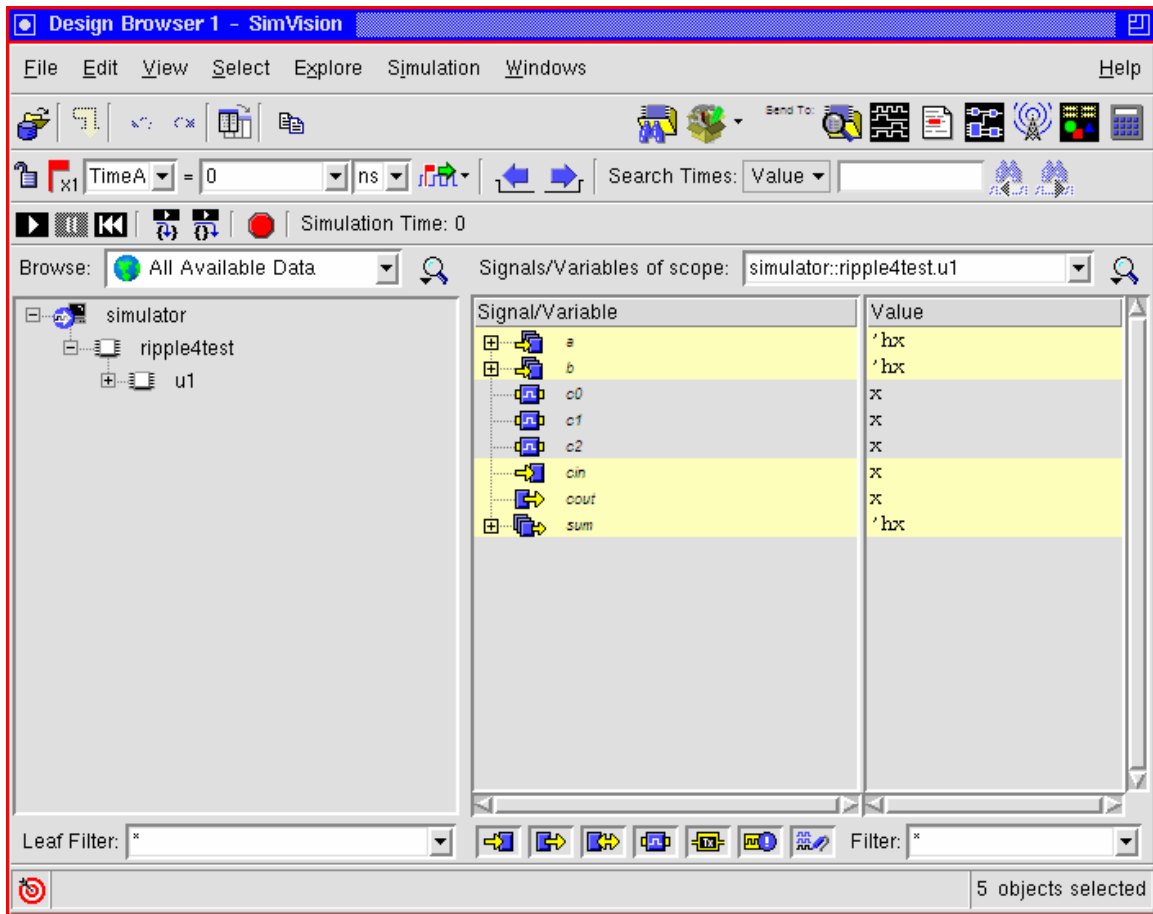
```
input [3:0] a, b;
input cin;
```

**Your code goes here. It is four instantiations of the module fadd.**

```
endmodule

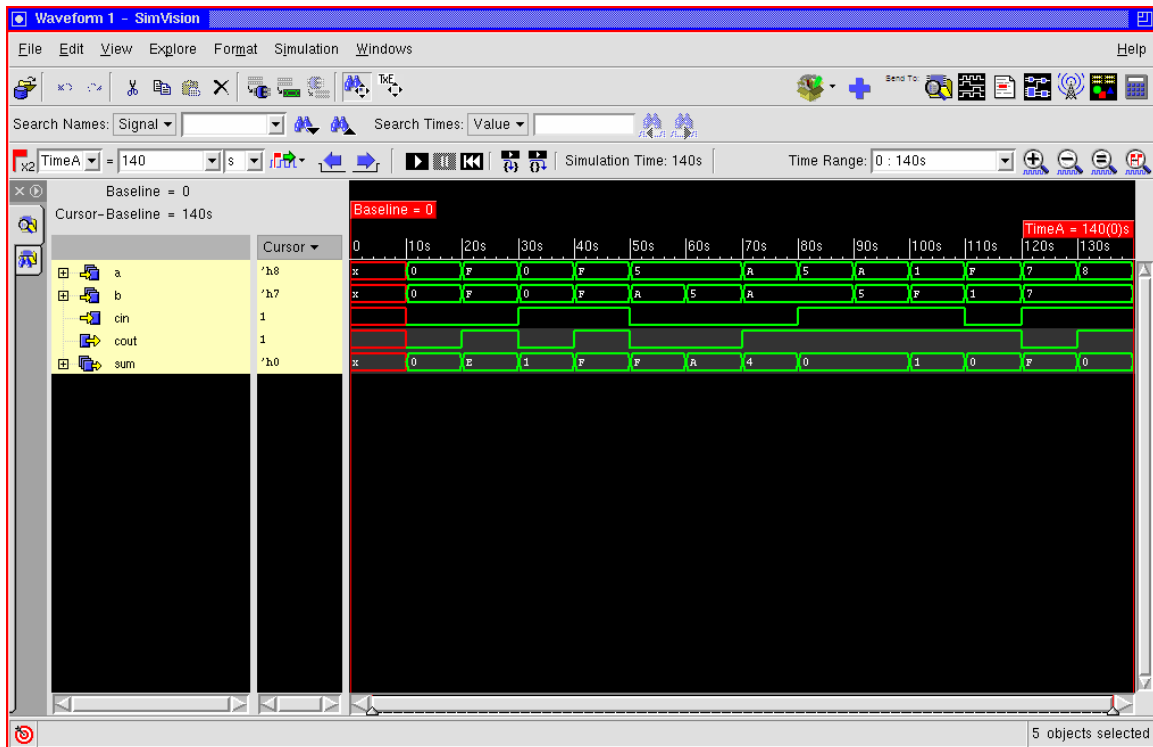
// Testbench for 4-bit ripple adder
module ripple4test;
  reg [3:0] a, b;
  reg cin0;
  wire [3:0] sum;
  wire cout3;
  ripple4 u1 (cout3, sum, a, b, cin0);
  initial
    begin
      $stop;
      #10;
      a=4'h0; b=4'h0; cin0=1'b0;
      #10;
      a=4'hf; b=4'hf; cin0=1'b0;
      #10;
      a=4'h0; b=4'h0; cin0=1'b1;
      #10;
      a=4'hf; b=4'hf; cin0=1'b1;
      #10;
      a=4'h5; b=4'ha; cin0=1'b0;
      #10;
      a=4'h5; b=4'h5; cin0=1'b0;
      #10;
      a=4'ha; b=4'ha; cin0=1'b0;
      #10;
      a=4'h5; b=4'ha; cin0=1'b1;
      #10;
      a=4'ha; b=4'h5; cin0=1'b1;
      #10;
      a=4'h1; b=4'hf; cin0=1'b1;
      #10;
      a=4'hf; b=4'h1; cin0=1'b0;
      #10;
      a=4'h7; b=4'h7; cin0=1'b1;
      #10;
      a=4'h8; b=4'h7; cin0=1'b1;
      #10;
      $stop;
    end
endmodule
```

After, inserting all the code, invoke the simulator. This time open up the SimVision window to module u2 and only select the signals as shown in the following:



You can do this by holding the shift key while selecting the signal names. You may see signals other than `c0`, `c1`, `c2` as you may have used different wire names when you implemented your circuit.

Now, invoke the waveform viewer and run the simulation. You should see a window that looks like the following:



Notice we are using hexadecimal to make the display more readable. You should verify that these additions are correct by doing a few by hand. We have not tested all possible combinations of inputs here.

Once you have this window, get a screen capture of it (alt-prtscr in Windows) and save it to a file for your final report. Also, have the lab instructor look at it and initial below.

LAB INSTRUCTOR'S INITIALS: \_\_\_\_\_

QUESTION 1: In your final report calculate the total number of needed input combinations that would be required to test all combinations of inputs.

PART 3: For the final portion of this lab, you will design and simulate a 4-bit carry-lookahead adder similar to the diagram on page 126 of the text (Figure 4-12). You will design a separate module for the large CLA generator box in the middle of the figure. I suggest you do this by forming Boolean equations for each output. Note that the equation for output C4 is not given in the text. You are to derive this equation as part of your Pre-lab exercise.

Again, most of the code will be given to you. It is listed below and is available for download on the class website (file is named **cla4.v**).

```
// Model a four-bit CLA generator
module clagen (C, P, G, cin);
    output [4:1] C;
    input  [3:0] P, G;
```

```
input cin;
```

**Your code goes here**

```
endmodule
```

```
// Model a four-bit CLA adder
module cla4 (cout, Sum, A, B, cin);
    output [3:0] Sum;
    output cout;
    input [3:0] A, B;
    input cin;
```

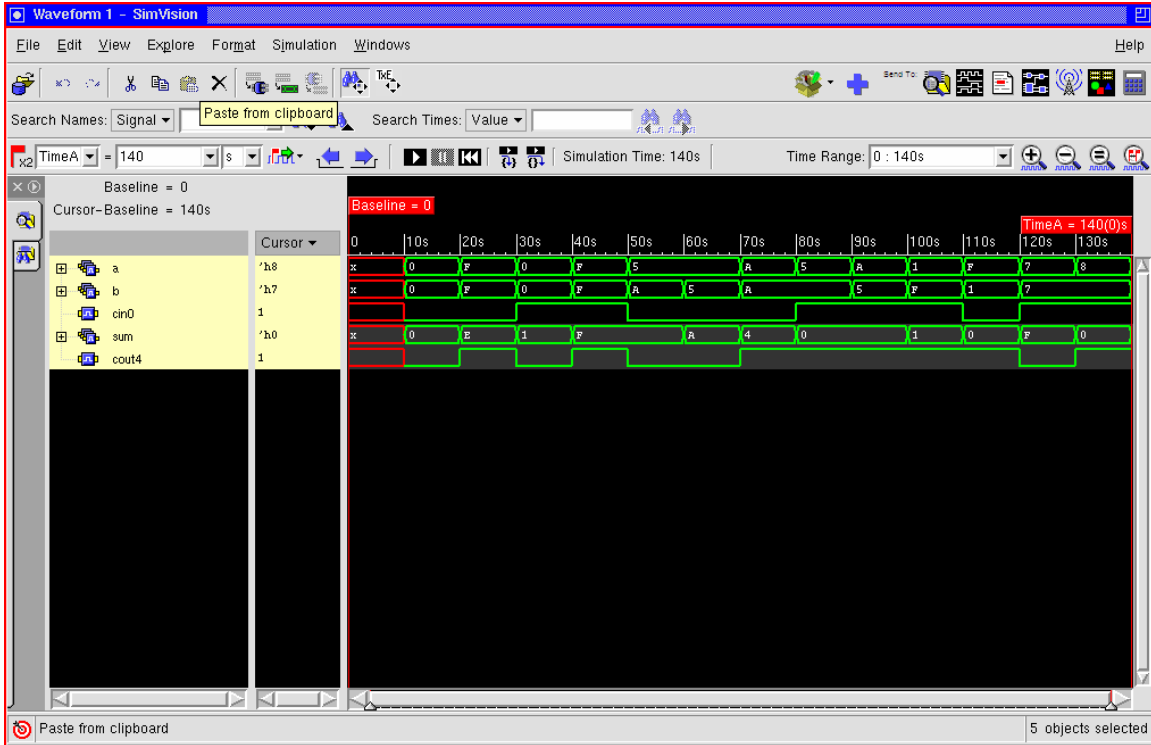
**Your code goes here**

```
endmodule
```

```
// Testbench for 4-bit CLA adder
module clatest;
    reg [3:0] a, b;
    reg cin0;
    wire [3:0] sum;
    wire cout4;
    cla4 u1 (cout4, sum, a, b, cin0);
    initial
        begin
            $stop;
            #10;
            a=4'h0; b=4'h0; cin0=1'b0;
            #10;
            a=4'hf; b=4'hf; cin0=1'b0;
            #10;
            a=4'h0; b=4'h0; cin0=1'b1;
            #10;
            a=4'hf; b=4'hf; cin0=1'b1;
            #10;
            a=4'h5; b=4'ha; cin0=1'b0;
            #10;
            a=4'h5; b=4'h5; cin0=1'b0;
            #10;
            a=4'ha; b=4'hf; cin0=1'b0;
            #10;
            a=4'hb; b=4'he; cin0=1'b1;
            #10;
            a=4'hc; b=4'hd; cin0=1'b1;
            #10;
            a=4'hd; b=4'hc; cin0=1'b1;
            #10;
            a=4'he; b=4'hb; cin0=1'b0;
            #10;
            a=4'hf; b=4'ha; cin0=1'b1;
            #10;
            a=4'h8; b=4'h7; cin0=1'b1;
            #10;
            $stop;
        end
end
```

endmodule

If your circuit simulates correctly, you should see the following output:



Once you have this window, get a screen capture of it (alt-prtscr in Windows) and save it to a file for your final report. Also, have the lab instructor look at it and initial below.

LAB INSTRUCTOR'S INITIALS: \_\_\_\_\_

PRELAB:

You are to generate the expression for  $C_4$  in terms of  $P_i$ ,  $G_i$ , and  $C_0$  as shown in Figure 4-12 of your text. You should show this expression to your lab instructor at the beginning of class and get his initials below.

LAB INSTRUCTOR'S INITIALS: \_\_\_\_\_

**FINAL REPORT:** Your final report is due the following class period and should include a discussion of any problems you had with the lab and how they were resolved. It should also contain the three waveform screen captures described above, a listing of your Verilog source code, and your design information (i.e. the maps, logic equations, circuits if drawn, etc.). It should also include the original hardcopy of this sheet with the lab instructors' initials. Finally, your report should contain the answer (and the associated analysis) of QUESTION 1 given above.