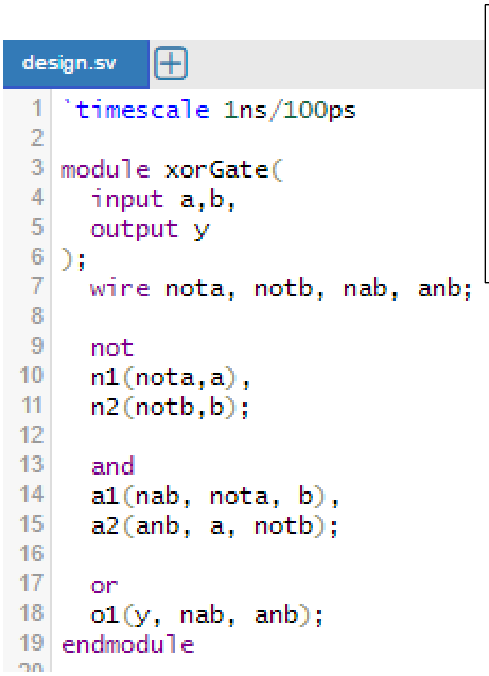
OBJECTIVE: Model a logic circuit using gate level primitives to learn port mapping for interconnecting modules.

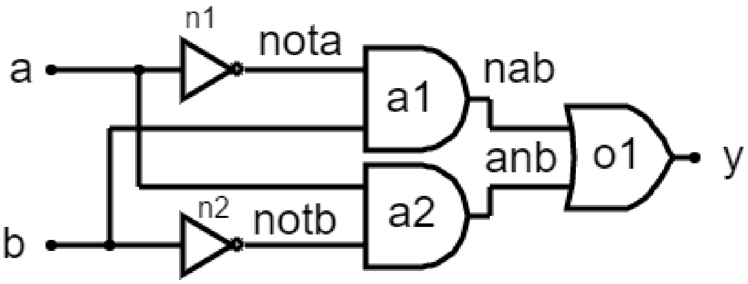
PROCEDURE: In this lab a logic circuit will be modeled using built in library module components for the and gate, the or gate, and the not gate (inverter). An example for your reference is shown below which uses gate level primitives to model an exclusive or gate along with the testbench which will write it's truth table to the simulation console:



```
`timescale 1ns/100ps

module xorGate(
  input a,b,
  output y
);
  wire nota, notb, nab, anb;

  not
  n1(nota,a),
  n2(notb,b);

  and
  a1(nab, nota, b),
  a2(anb, a, notb);

  or
  o1(y, nab, anb);
endmodule
```

Notice the relationship between the logic circuit below and the Verilog module on the left. Internal wires must be declared inside the module as local variables. Inputs and the output are considered local variables. The port list for each logic gate has parameters which follow this order:

(output, input1, input2, … inputN)

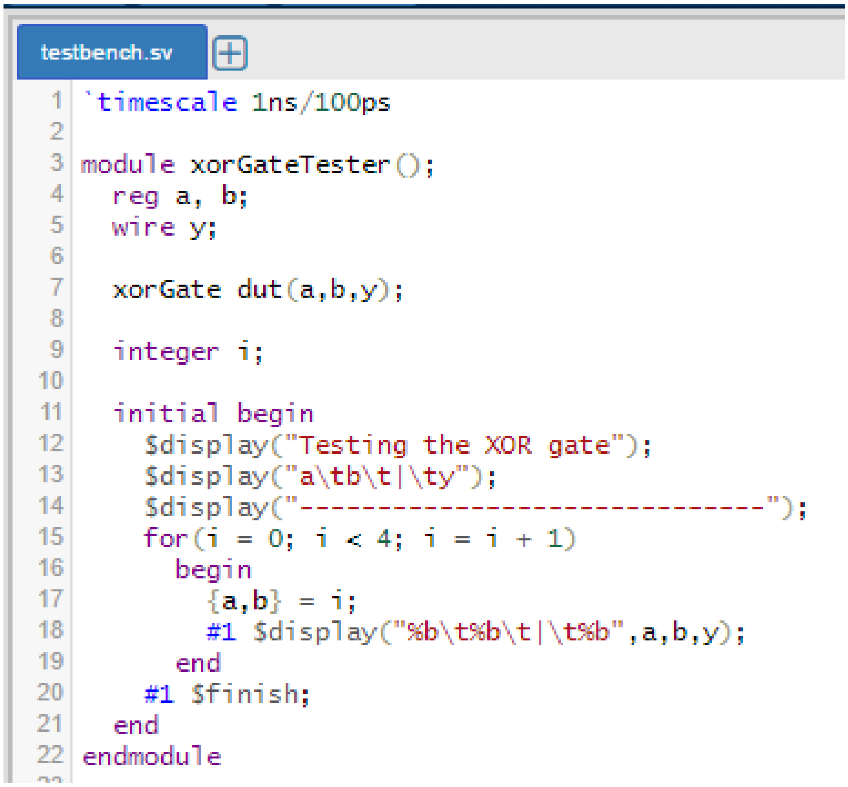


As it can be seen the Verilog module for xorGate is a list of all the connections depicted in the logic diagram above.

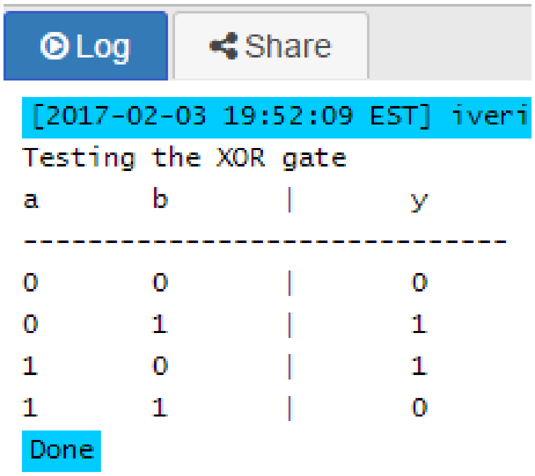
Note the way that reserved keywords are used to place the logic gates. Each logic gate also has a unique identifier which is followed by a list of connections known as a port list. Multiple instances of the same type of module (or the same type of gate) can be coded conveniently by using a comma after a port list before providing a unique identifier for the next instance of the gate/module followed by its port list and the last instance is followed by a semicolon.

Next the testbench definition will be outlined on the next page.

The code for a testbench is shown below along with the console outputs it produces:



```
1  `timescale 1ns/100ps
2
3  module xorGateTester();
4    reg a, b;
5    wire y;
6
7    xorGate dut(a,b,y);
8
9    integer i;
10
11   initial begin
12     $display("Testing the XOR gate");
13     $display("a\tb\t|\ty");
14     $display("------------------------------");
15     for(i = 0; i < 4; i = i + 1)
16       begin
17         {a,b} = i;
18         #1 $display("%b\t%b\t|\t%b",a,b,y);
19       end
20     #1 $finish;
21   end
22 endmodule
```

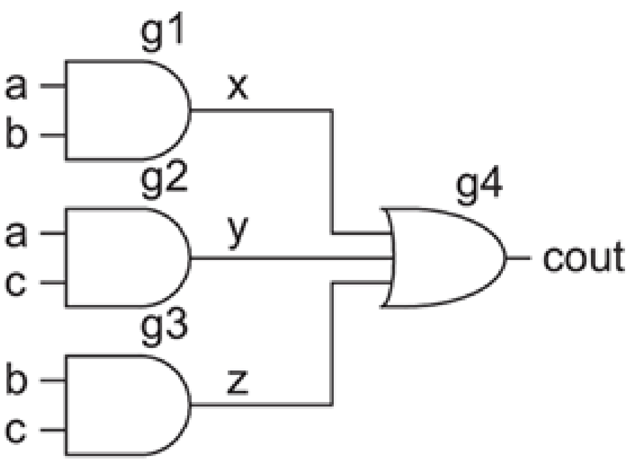


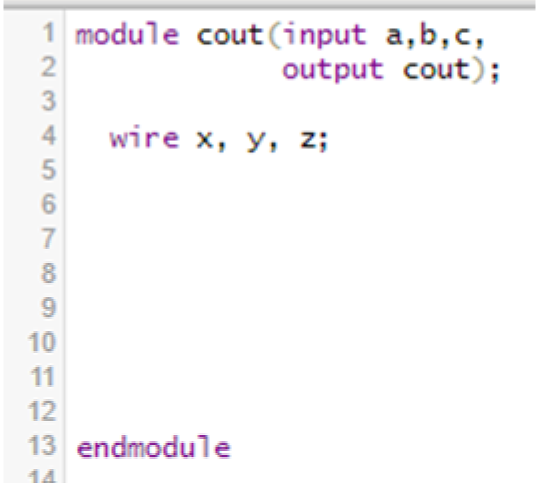
```
[2017-02-03 19:52:09 EST] iveri
Testing the XOR gate
a       b       |       y
-------------------------------
0       0       |       0
0       1       |       1
1       0       |       1
1       1       |       0
Done
```

As you can see the console output depicts the contents of the XOR truth table. In the testbench, the for loop only goes through 4 iterations since there are only 2 input variables that are each 1-bit. Each time the for loop iterates, a test case is performed.

The way input variables get their values is shown on line 17 using the assignment statement {a,b} = i; This line of code takes the value of i, converts it to binary, variable a gets the most significant bit of i, variable b gets the least significant bit of i.

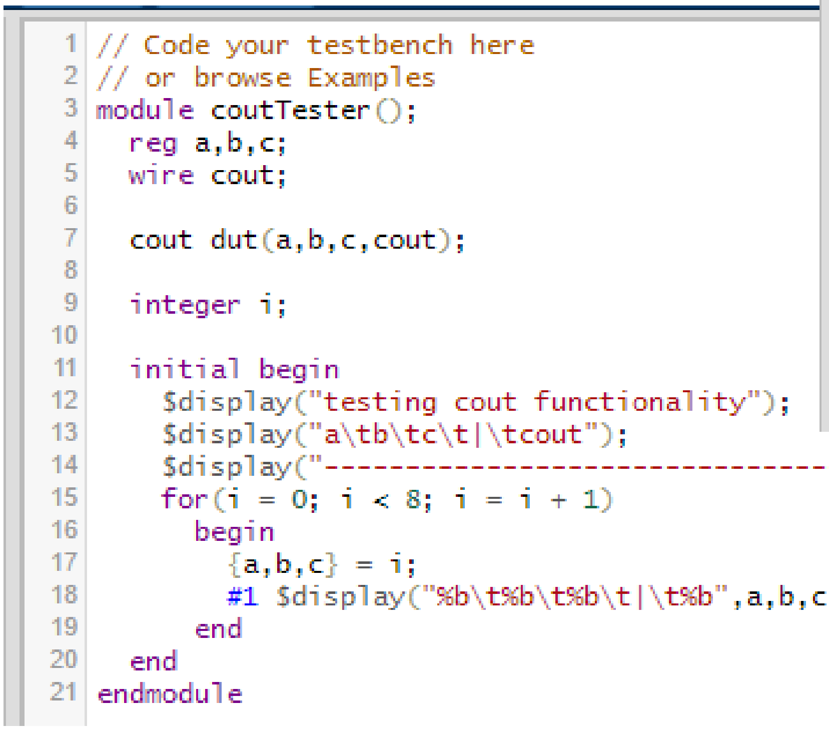
Your task is to model the following circuit using gate level primitives given the following Verilog module skeleton. Name your gates according to the logic diagram below and use wires properly:



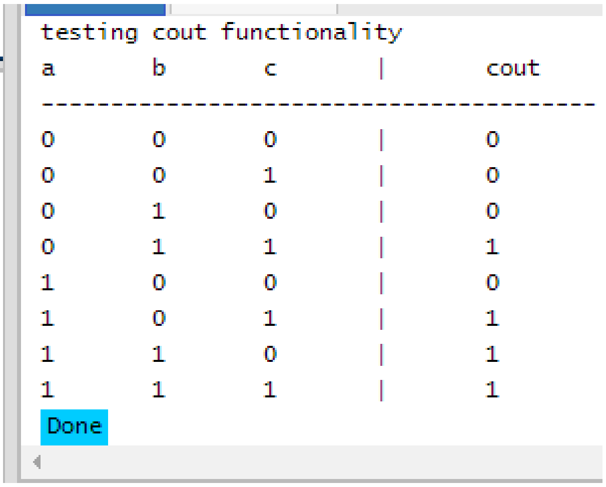


```
module cout(input a,b,c,
            output cout);

  wire x, y, z;








endmodule
```

Once you have created the cout module, use the given tester to check for correctness and the test results shown on the right should be produced:



```
// Code your testbench here
// or browse Examples
module coutTester();
  reg a,b,c;
  wire cout;

  cout dut(a,b,c,cout);

  integer i;

  initial begin
    $display("testing cout functionality");
    $display("a\tb\tc\t|\tcout");
    $display("----------------------------------------");
    for(i = 0; i < 8; i = i + 1)
      begin
        {a,b,c} = i;
        #1 $display("%b\t%b\t%b\t|\t%b",a,b,c,cout);
      end
  end
endmodule
```



```
testing cout functionality
a       b       c       |       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
Done
```

WHAT TO TURN IN: Once your cout module is working correctly:

- Copy the contents of your cout module to a file named cout.txt
- upload cout.txt to the beachboard dropbox for Lab1