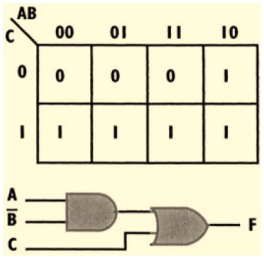
Sheet 1

Exercise 1-1:

Write a structural Verilog description of the logic diagram shown in the figure below. Use primitive gates having delays of 2, 2, and 1 for AND, OR, and NOT gates respectively.



Exercise 1-2:

Test the description of Exercise 1-1 by adding an initial block that will provide several different inputs to the gates and display **all values** in the circuit (using an appropriate **`$monitor`** statement). Modify the module accordingly and get rid of all ports since they are not needed here. The ordered series of inputs that we will put into our design will be (a, b, c): 100,110, 010, 011. Assume a delay of 2 before changing each input.

What is the exact output displayed by the **`$monitor`** statement?

Why is this testing method not recommended?

Exercise 1-3:

What would the exact output be changed if the delay between different inputs values is removed?

Exercise 1-4:

Test the description of Exercise 1-1 using the same input sequence as in Exercise 1-2, but this time do the testing by instantiating the DUT on a testbench that generates the inputs and monitors the output.

Exercise 1-5:

Test the description of Exercise 1-1 using the same input sequence as in Exercise 1-2, but this time do the testing by designing a special testing module that generates the inputs and monitors the output. A testbench should also be designed to instantiate both the DUT and the testing module and connect them.

Exercise 1-6:

Modify the description of Exercise 1-1 to use NAND gates (delay 1) only. How does that affect the output assuming the same input sequence as Exercise 1-2.

Exercise 1-7:

Draw the logic diagram of the circuit described by the following Verilog description:

```
module halfadder (cOut, sum, a, b);
   output cOut, sum;
   input a, b;
   xor #1 (sum, a, b);
   and #2 (cOut, a, b);
endmodule
```

Exercise 1-8:

Write a testing module and a testbench module to test all possible inputs to the circuit description in Exercise 1-7.

Exercise 1-9:

Write a structural Verilog description of a full-adder using the half adder module defined in Exercise 1-7. Test your full-adder using a testbench module that generates all possible input combinations.