CALIFORNIA STATE UNIVERSITY
LOS ANGELES

Department of Electrical and Computer Engineering

EE-2449 Digital Logic Lab

EXPERIMENT 5

BOOLEAN ALGEBRA AND VERILOG SIMULATION

Text: Mano and Ciletti, *Digital Design, 5th Edition,* Chapter 2

Required software: Xilinx ISE 14.7

5.1 Boolean Algebra is a mathematical approach to deriving equivalent functions. It is desirable to be able to simplify a function in order to design a circuit that uses fewer gates which can result in a more compact, lower-power design or to reduce the propagation delay of a signal through the circuit in order to speed up the circuit.

The table below shows the various identities that can be used to simplify a function.

Boolean Algebra Identities:

Name	Identity	
Involution (Double Negative)	A" = A	
Law of Intersection	A + 0 = A	A • 1 = A
Complementary Law	A + A' = 1	A • A' = 0
Idempotent Law	A + A = A	A • A = A
Null Law	A + 1 = 1	A • 0 = 0
Identity Law	A + 0 = A	A • 1 = A
Commutative Law	A + B = B + A	AB = BA
Associative Law	A + (B + C) = (A + B) + C	A(BC) = (AB)C
Distributive Law	A(B + C) = AB + AC	A + BC = (A + B)(A + C)
DeMorgan's Law	(A+B)' = A'B'	(AB)' = A' + B'
Law of Absorption	A + AB = A	A(A + B) = A

The easiest to derive form of a function is the canonical representation, either Sum of Minterms or Product of Maxterms. A minterm is a product term that contains every input variable and a maxterm is a sum term that contains every input variable. Minterm *i* will evaluate to true (1) for input combination *i*. Maxterm *i* will evaluate to false (0) for input combination *i*. The table below shows the minterms (products) and maxterms (sums) for all possible input combinations for a 3-variable function with input variables A, B, and C.

A	**B**	**C**	**minterm**	**Maxterm**
0	0	0	m0 = A'B'C'	M0 = A+B+C
0	0	1	m1 = A'B'C	M1 = A+B+C'
0	1	0	m2 = A'BC'	M2 = A+B'+C
0	1	1	m3 = A'BC	M3 = A+B'+C'
1	0	0	m4 = AB'C'	M4 = A'+B+C
1	0	1	m5 = AB'C	M**5** = A'+B+C'
1	1	0	m6 = ABC'	M6 = A'+B'+C
1	1	1	m7 = ABC	M**7** = A'+B'+C'

A function can easily be expressed using a Sum of Minterms expression by taking the logical sum (OR) of the function's minterms (ANDs); i.e., for every input combination that produces a one at the output. Likewise, a function can easily be expressed using a Product of Maxterms expression by taking the logical product (AND) of the function's Maxterms (ORs); i.e., for every input combination that produces a zero at the output.

Consider the XOR and XNOR functions shown in this table.

A	B	A ⊕ B	(A ⊕ B)'
0	**0**	**0**	**1**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**

The exclusive-or of A and B, A ⊕ B, can be expressed as the sum of minterms 1 and 2. That is,

$$A \oplus B = \Sigma m(1, 2) = A'B + AB'$$

The exclusive-nor of A and B, (A ⊕ B)' can be expressed as the sum of minterms 0 and 3. That is,

$$(A \oplus B)' = \sum m(0, 3) = A'B' + AB$$

The exclusive-or of A and B, A ⊕ B, can be expressed as the products of Maxterms 0 and 3. That is,

$$A \oplus B = \Pi M(0, 3) = (A + B)(A' + B')$$

The exclusive-nor of A and B, (A ⊕ B)' can be expressed as the product of Maxterms 1 and 2. That is,

$$(A \oplus B)' = \Pi M(1, 2) = (A + B')(A' + B)$$

In this part of the experiment you should do the following (in your lab notebook):

- Explore three different functions for implementing the majority voter, V, described in Experiment 4. The first, referred to as F1, is the function for V from Experiment 4: F1 = AB + BC + AC.
- Write the truth table for the majority voter, V, with inputs A, B, and C (this was derived in Experiment 4).
- Express V using the sum of minterms notation. Call this function F2.
- Express V using the product of maxterms notation. Call this function F3.
- Use the Boolean Algebra identities (see table, pg.1) to prove that F1 = F2. Show the steps and label each step with the appropriate identity/Law. Hint, you can use the Idempotent law to add additional redundant minterms.
- Use the Boolean Algebra identities to prove that F1 = F3. Show the steps and label each step with the appropriate identity/Law.

5.2* In this part of the experiment you will verify the Consensus Theorem (AB + A'C + BC = AB + A'C) by implementing the following two functions using the Verilog Hardware Description Language (HDL), simulating the Verilog functions using the ISim simulator, and verifying that they both produce the same simulation waveform. (That F1 = F2 can also be verified by drawing a K-map for each.)

F1 = AB +A'C + BC

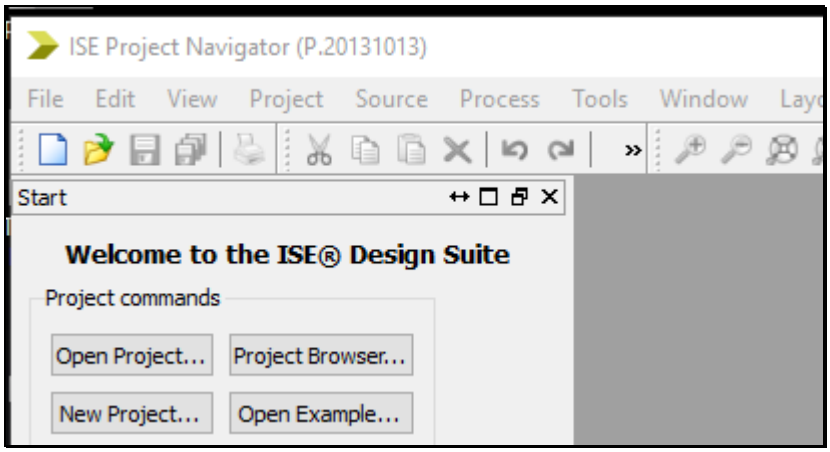
F2 = AB + A'C

The purpose of this Experiment is to help you learn the basics of designing combinational logic circuits using a hardware description language (HDL), to learn the basics of how to model a circuit using structural (gate-level) modeling in Verilog, and how to verify that a circuit works properly using a simulator.

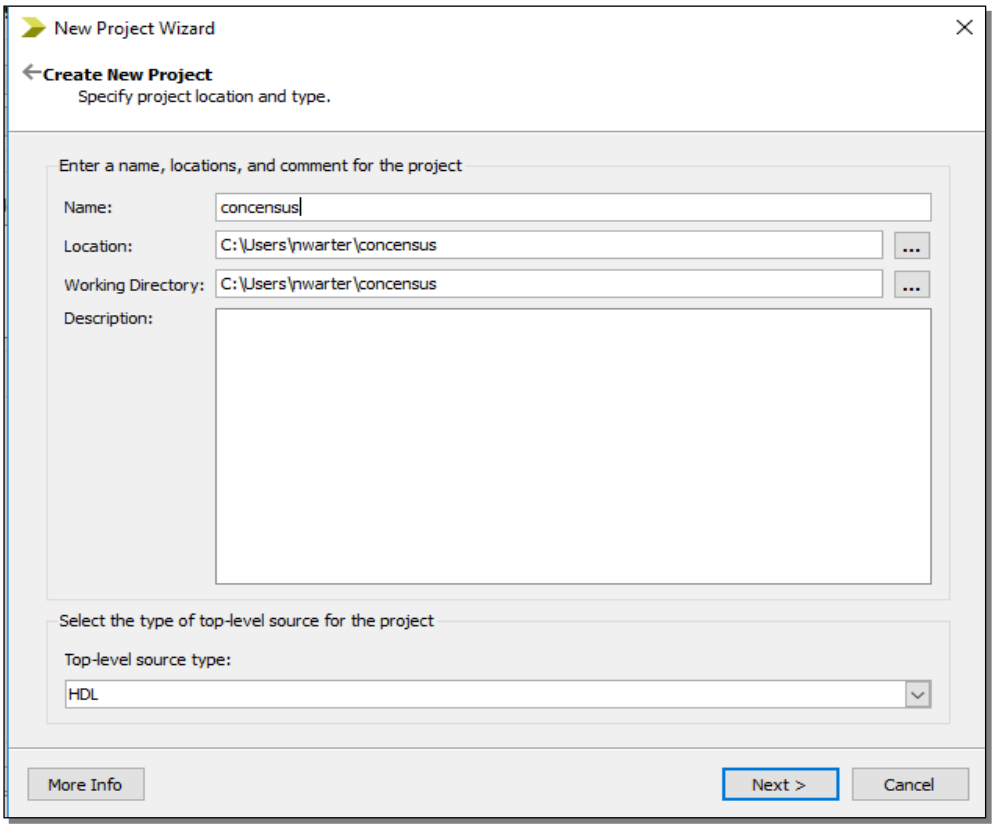
Follow along with the instructions provided to implement F1 and F2 in Verilog and to simulate the circuits to verify that F1 is equivalent to F2. Record notes and observations in your manual to assist you in the future when designing circuits in Verilog.

Turn on the lab computer. Open up the Xilinx ISE Design Suite 14.7 (located on the Desktop).

Click ok after reading the tip of the day (don't worry if you don't understand the tip). Click on **New Project**.



Add your username in the path for Location after "C:\Users" (ex: "C:\Users\nwarter). Don't click the … to navigate as there appears to be a bug in the installation and ISE will crash[1]. Then give the project a name (ex: concensus[2]).

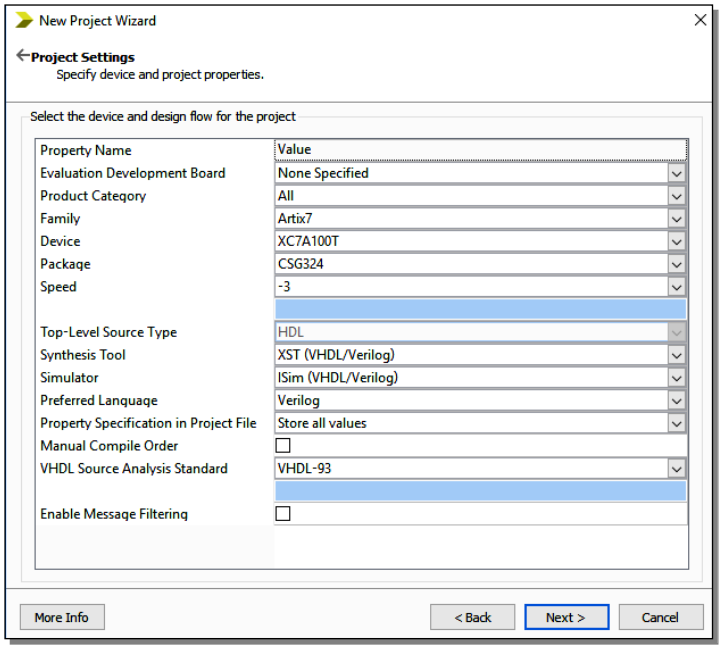


In this experiment we will be simulating the results in software. It is also possible to synthesize the code to run on hardware. However, we will not be doing that in lab this semester. Therefore, you do not need to specify the device or any of the information in the top half of the **Project settings** window.

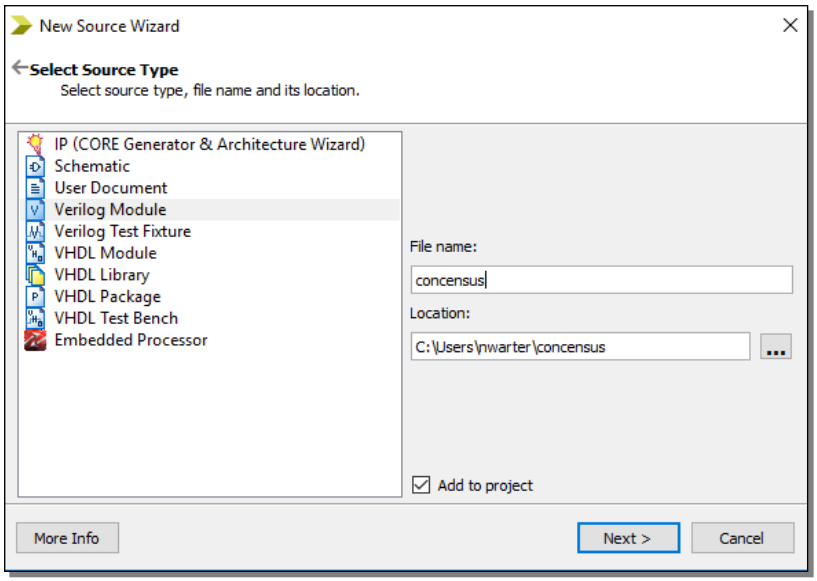
[1] Also don't use the Add Source feature in ISE at this point. This will also crash the tool. These bugs have been reported to the IT consultants.

[2] Note, in the project and filenames used in this design, consensus was misspelled as concensus.

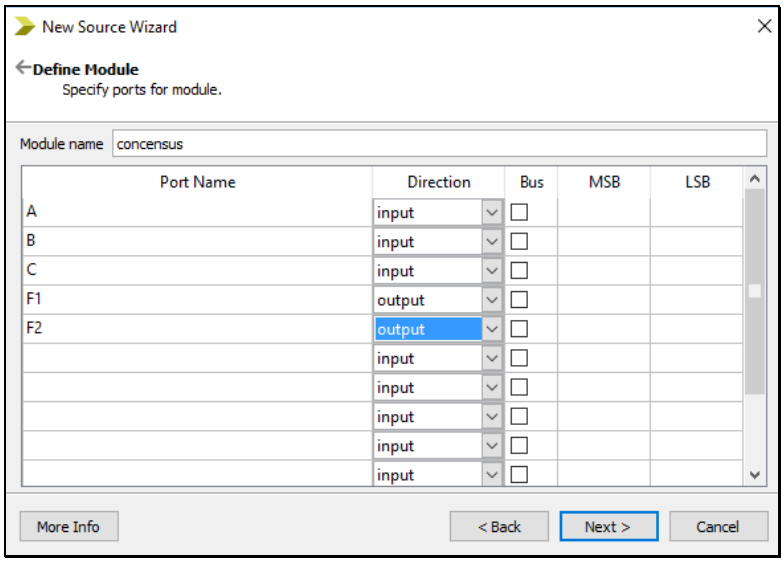
You do need to specify (if not already specified) that the **Simulator** is **ISim** and the **Preferred Language** is **Verilog**. Then click **Next** and **Finish**.



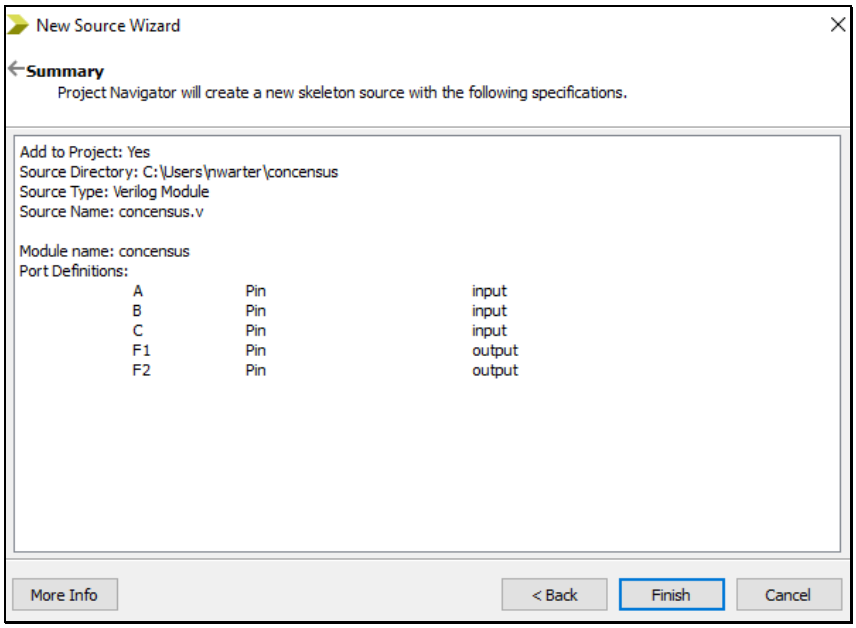
To add a source file to the project, under **Project**, select **New Source** (we'll abbreviate this instruction as **Project -> New Source**). In the **Select Source Type** window, select **Verilog Module** and specify the File name (ex: concensus). Also make sure **Add to project** is checked.



Click **Next**. Specify the names of the input and output ports. In our example, A, B, and C are inputs and F1 and F2 are outputs. Make sure to change the **Direction** of F1 and F2 to **output**.

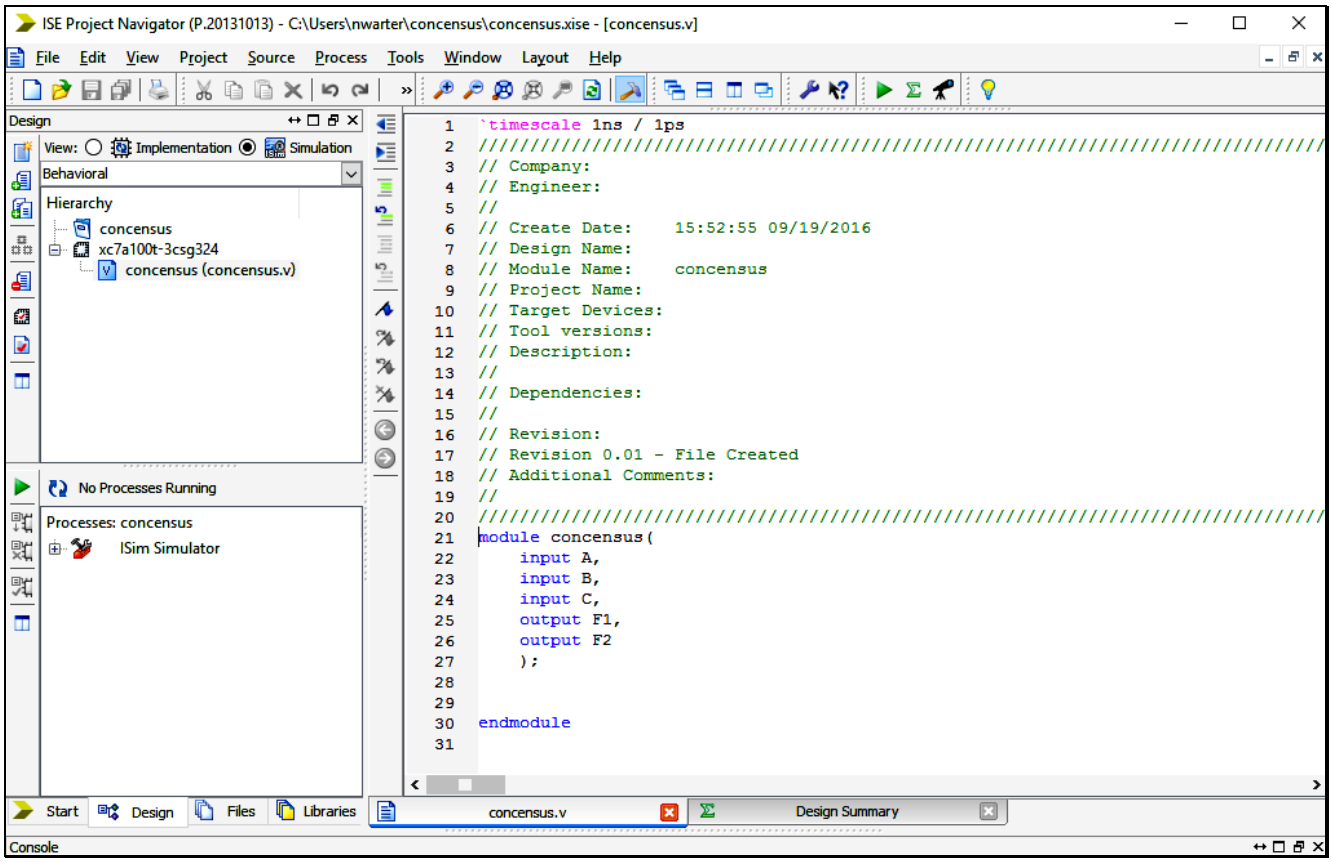


Click **Next**, review the summary to make sure it is correct and then click **Next** again. If there you notice a mistake in the summary, click **Back** and go back to correct it.



A template for your Verilog design should appear. Since we are simulating the circuit, click on the **Simulation** radial button under **Design** next to **View**. If the Verilog code is not showing in the large window, select the Verilog file (concensus.v) in the **Hierarchy** window.

Notice in the template below for concensus.v, there is a module named *concensus.* The module has three input ports, A, B, and C, and two output ports F1 and F2.



Implement the functions F1 and F2 in the module concensus using Verilog HDL. The following logic primitives are available to be used:

not, and, or, nand, nor, xor, xnor

These primitives are the same as the basic gates you tested in Experiment 3. Notice that they are all lower case (case matters in Verilog) and that they turn blue when you type them, indicating that they are reserved keywords. When using a primitive, after the name of the primitive there is a set of parentheses **( )** and between the parentheses are the ports. The ports are the inputs and outputs to the function. When using a primitive, the output port is always first because the basic logic functions (gates) only have one output. However, they can have multiple inputs. For example, we have seen both a two input NAND and a three input NAND. There is no need to indicate the number of inputs in Verilog, instead all ports after the output port are assumed to be inputs).

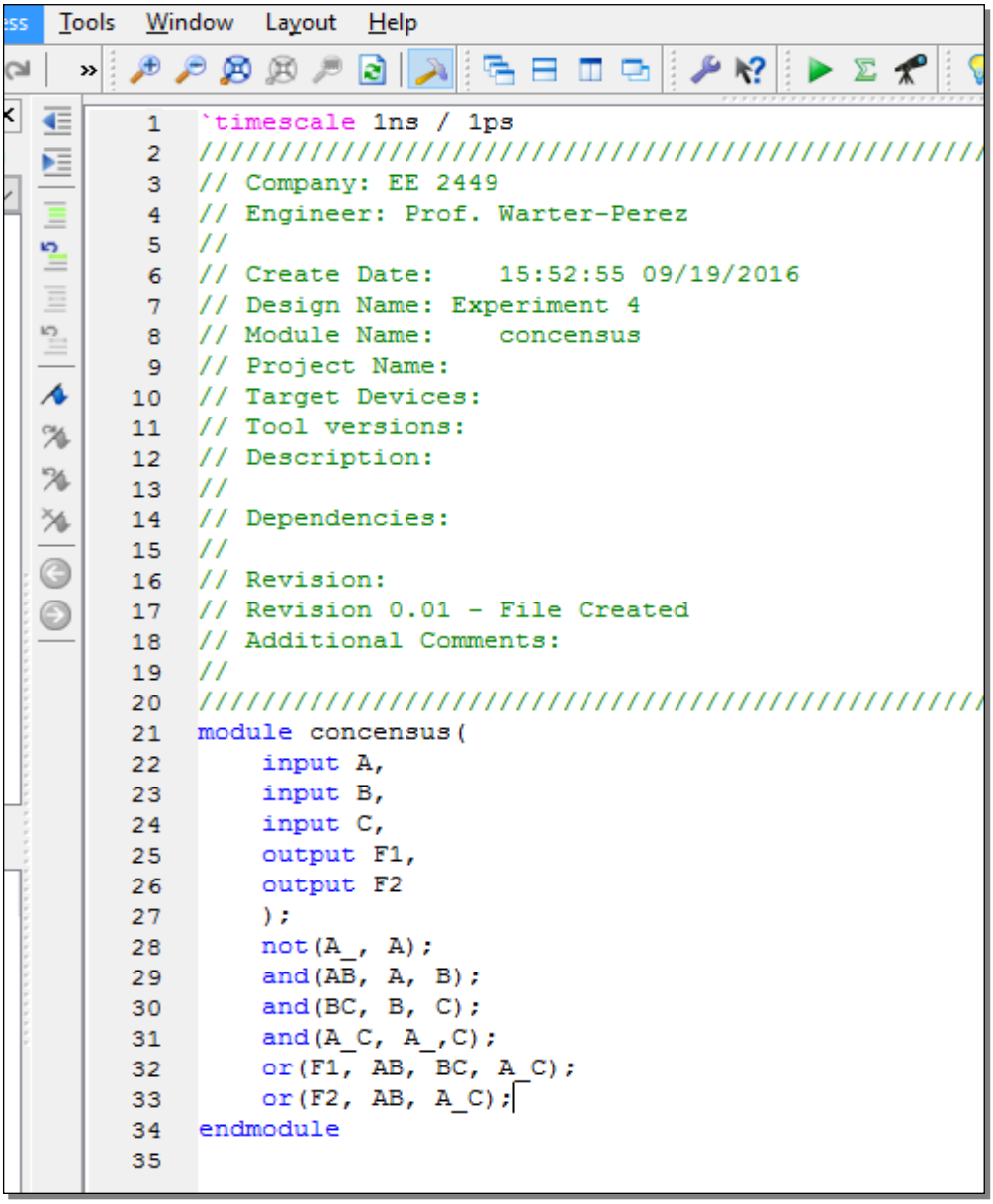
Consider the Verilog statement: **and (AB, A, B);**

Notice that the output of the **and** primitive (gate) is given the name **AB** and that its inputs are **A** and **B**. Since the Verilog language is case sensitive, input **A** would be different from **a**. An input or output can have any name provided it consists only of letters, numbers and underscores. Also, it can start only with a letter or underscore and it cannot be a reserved keyword like input or and, etc..

AB was the output name chosen here because it signifies the product term **AB**. Notice that the statement ends with a semicolon (;). Before proceeding, refer to the following Verilog file and convince yourself that the code actually implements the functions: F1 = AB + BC + A'C and F2 = AB + A'C.

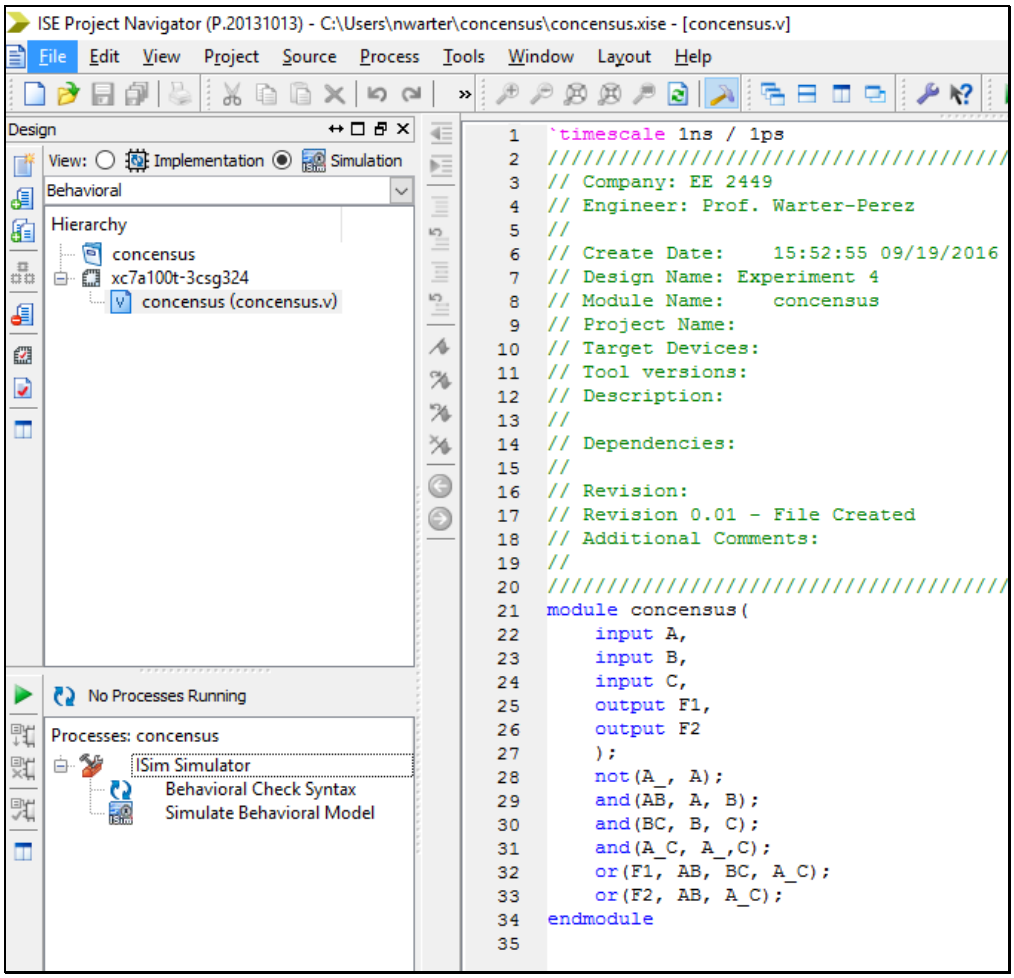
Notice that the underscore (_) is often used to indicate "not"; e.g., **A_** is **A not** or **not A**. Also notice that the code for the design is indented (tabbed over) so that it is easy to see what is contained in the module. Extra white space is ignored by the HDL Compiler but you should use tab to make your code more readable.

Type the code for F1 and F2 into your source file (ex: concensus.v). Also modify the comments as shown but include your and your partners names under Engineer.

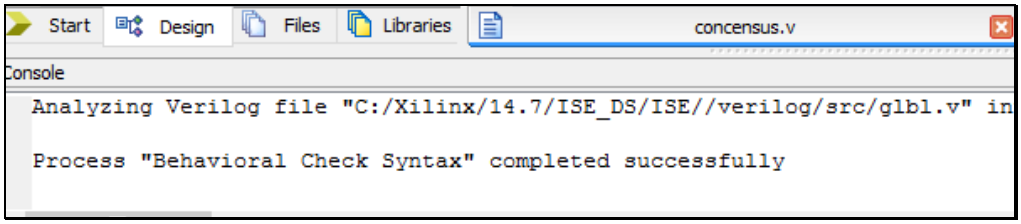


After you finish entering the design, save the file. If there are any syntax errors indicated in the console window, fix them. For example, delete the semicolon (;) after the **not** primitive and click save. You'll notice that there is an error message indicating that there is a syntax error (Syntax error near "and"). When the HDL Compiler tries to analyze the code it encounters the **and** primitives and it notices that there is an error because there should have been a semicolon **(;)** encountered before the **and**. Replace the semicolon after the **not** statement and save the file again. The syntax error message should go away.

Under Processes, expand the ISim Simulator dropdown and click on Behavioral Check Syntax. Again, if there are any mistakes, fix them, save the file, and run Behavioral Check Syntax again (click on it again).

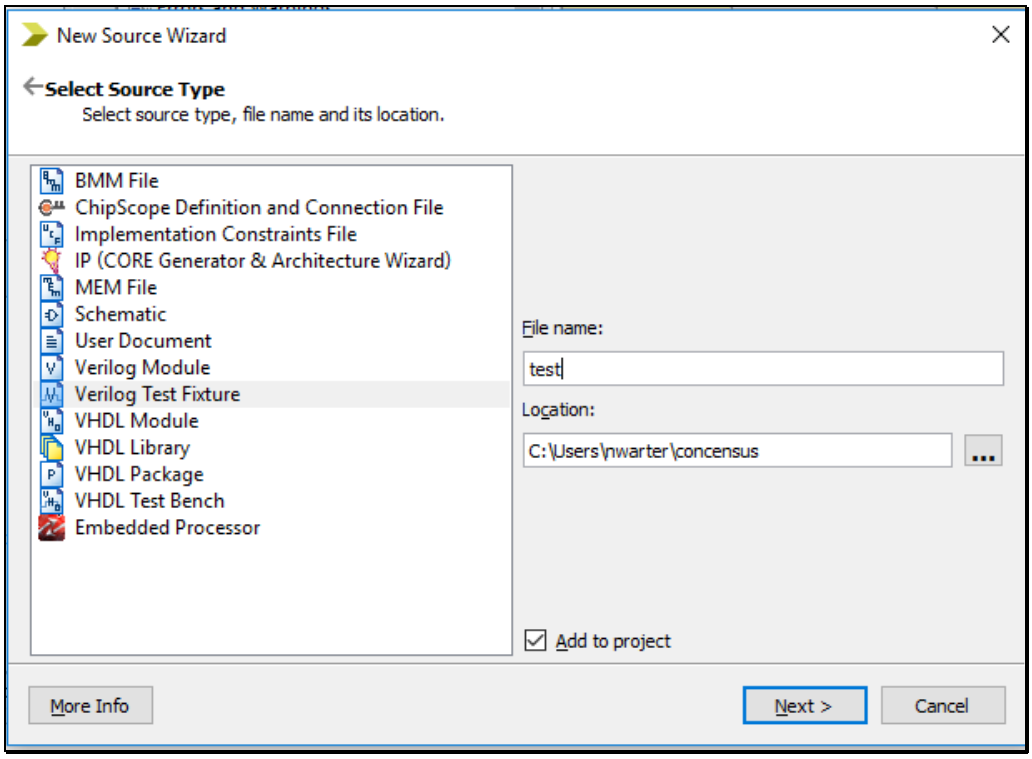


In the console window you should see the following message if there are no errors.

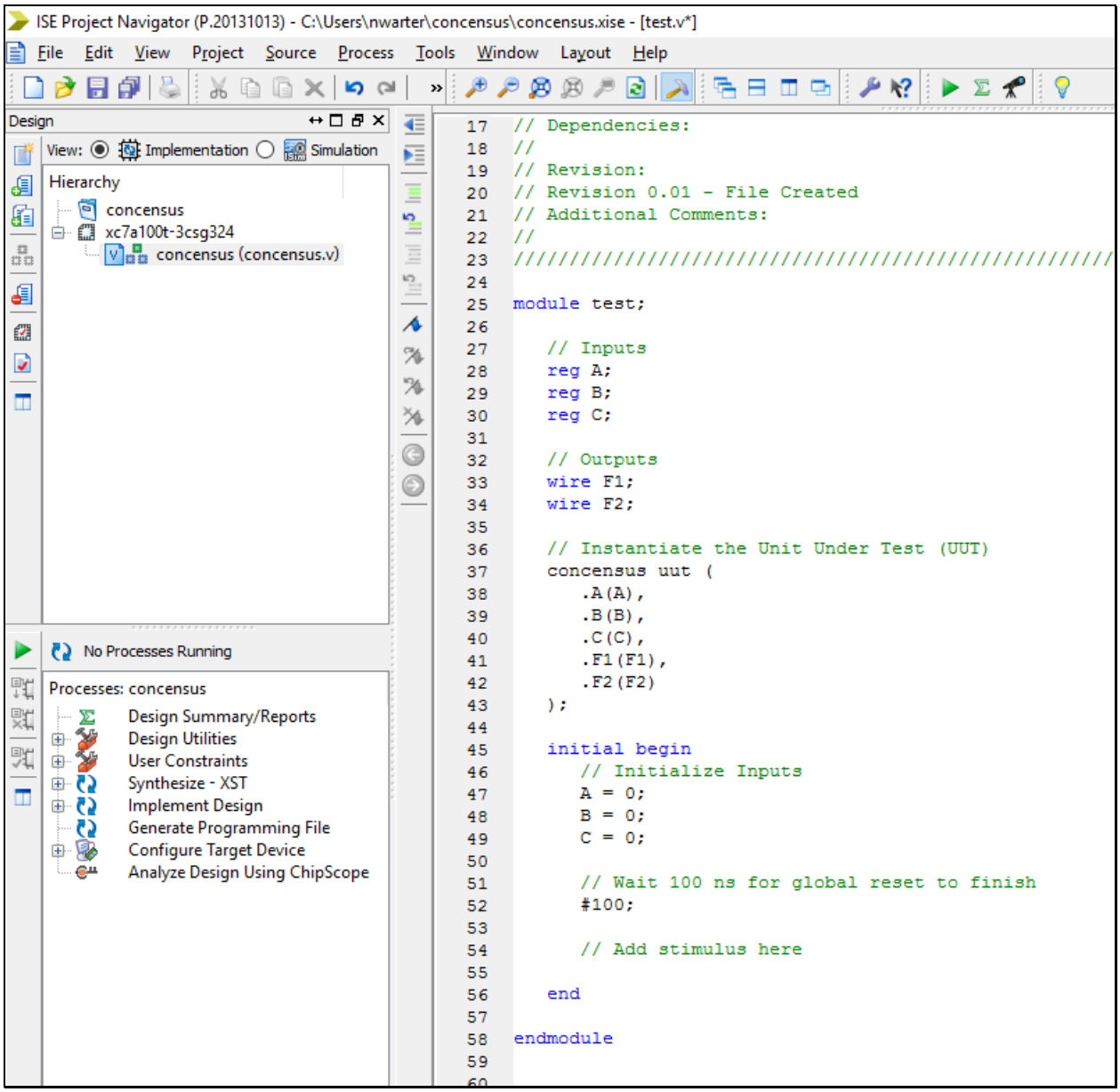


To simulate the circuit, we need to provide test inputs just as we use switches or the counter output to provide test inputs to our circuits in Experiments 3 and 4.

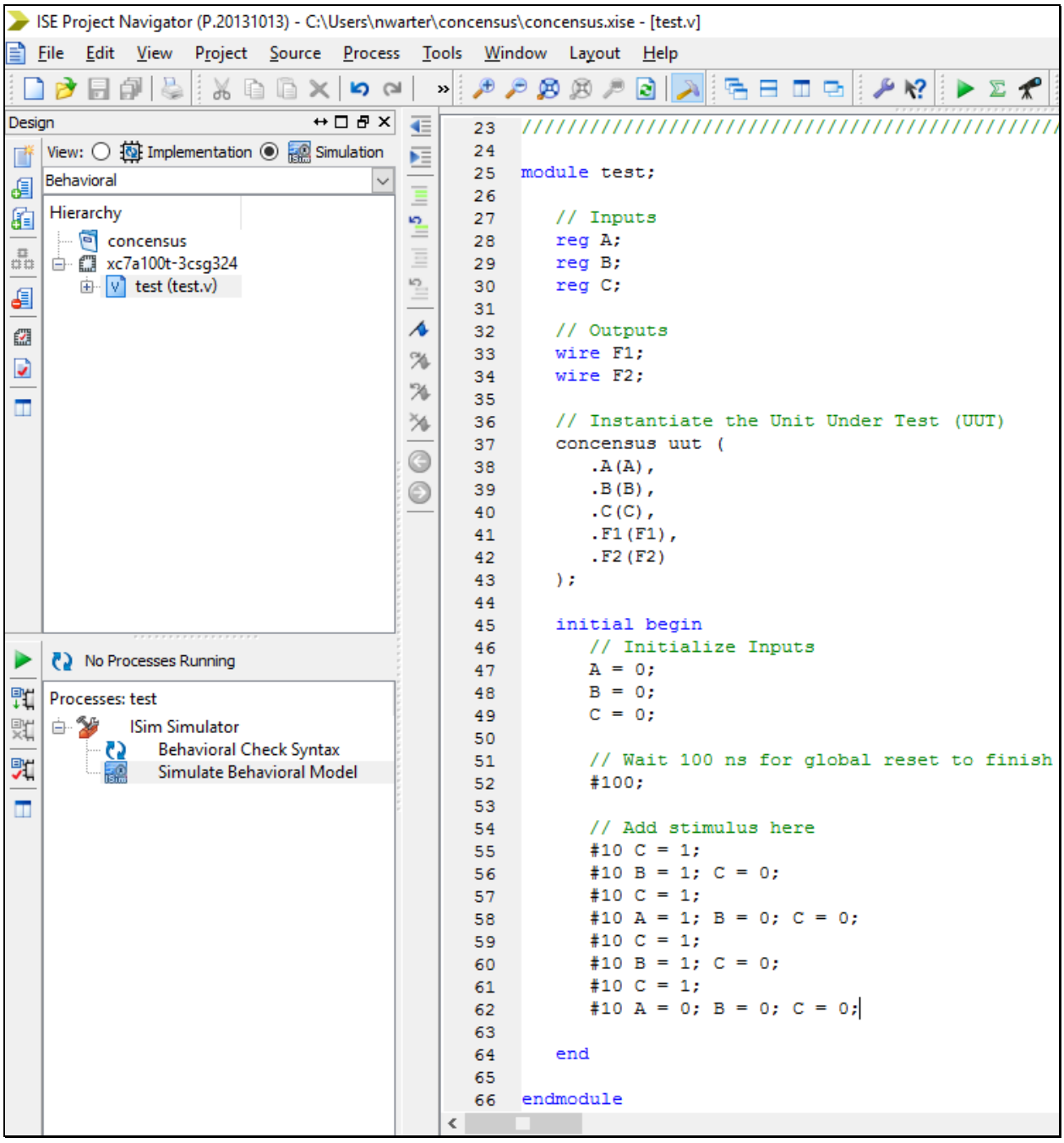
Under **Project**, select **New Source** (or right click in the **Hierarchy** window and select **New Source**). Select **Verilog Test Fixture** and name the file (ex: test). Make sure **Add to project** is selected. Click **Next**, **Next**, and **Finish**. Note, that you do want it associated with concensus.v since it will be providing the inputs for that module.



Notice in the following screen shot that the test fixture provides a template for you with the inputs for the "uut" (uut means unit-under-test; otherwise use "dut" for device-under-test); these are already inserted (ex: A, B, C). To test the circuits, values have to be loaded into registers (components that hold values) A, B, and C. This is similar to what we have done in hardware on the breadboard where we have driven the inputs from the counter (which can hold values) or from the switches which can be mechanically/physically set to a particular value. The outputs will be driven on the wires F1 and F2. Under the comment **Add stimulus here**, you should assign various test input combinations to A, B, and C. Notice they have already been initialized to 0. In between each test case, insert a delay using the statement #10 (the value of the delay doesn't matter in this case because we are only simulating for behavior and not for timing).

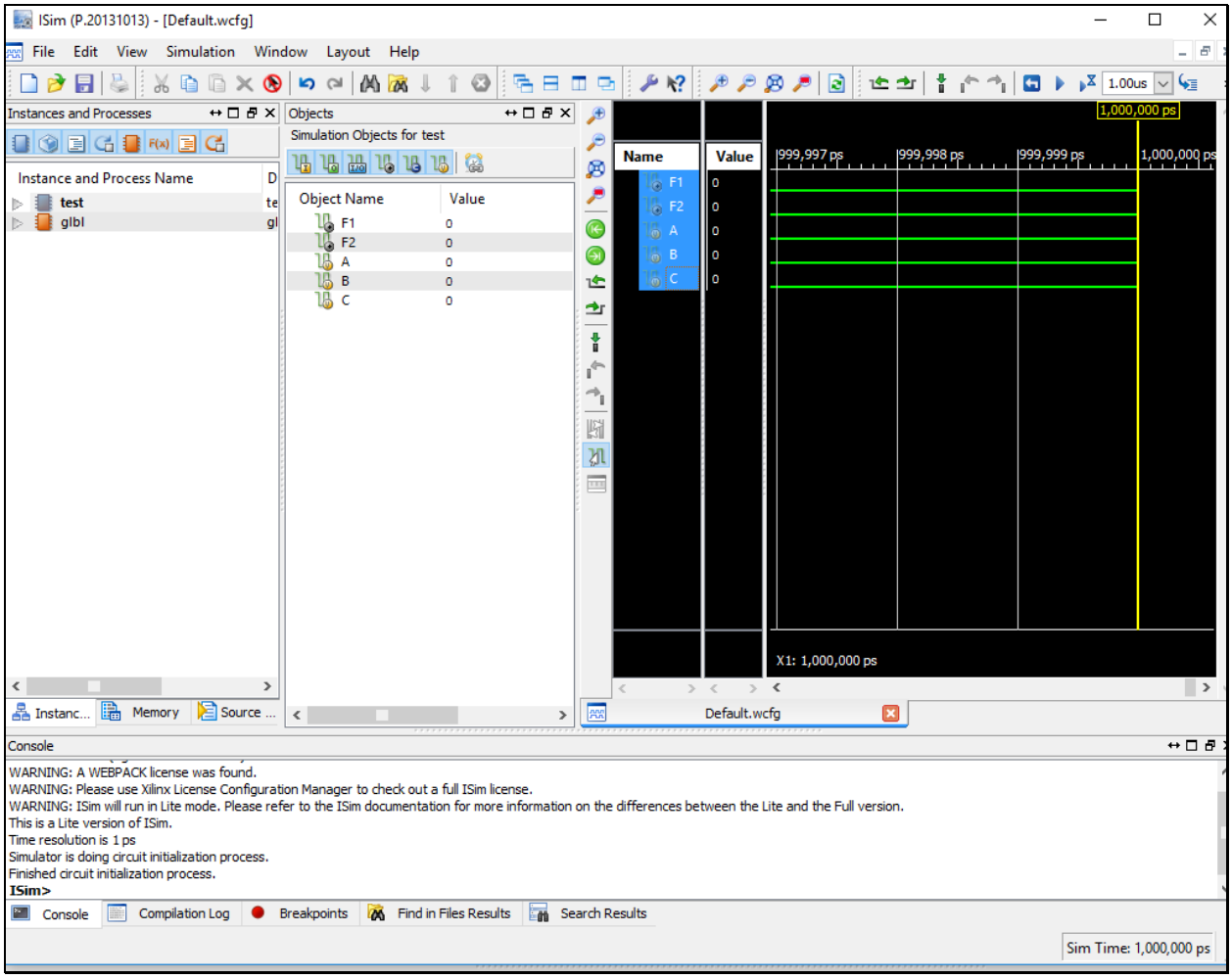


The various test cases have been inserted into the code below. Note, that this will test every combination from ABC = 000 to ABC = 111. The last statements sets the inputs back to 000 (similar to the counter rolling over). The last test case (ABC = 000) is redundant and was inserted so that you can easily see the test cases in the simulation waveform. Our test cases will start after a delay of 100 (indicated by #100). Note that one unit of delay has been set to 1 ns by default.

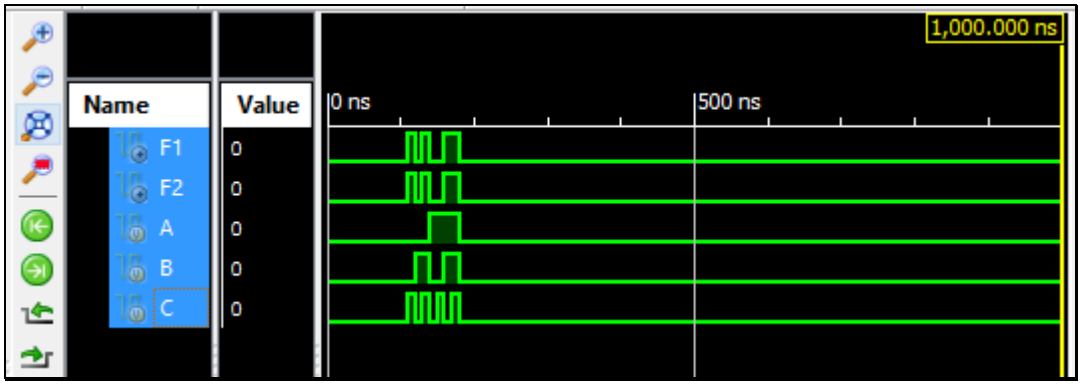


Save the file and make sure that there are no syntax errors. Make sure that your Design window still has the Simulation View clicked. Next, click on test.v in the hierarchy and then click on **Simulate Behavioral Model** under the **ISim Simulator** drop down in the **Processes** window.

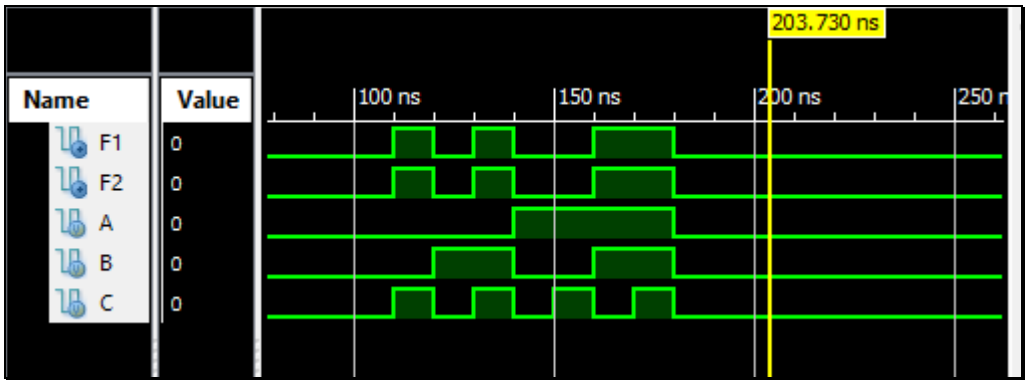
The simulator window will pop up.



Click on the Zoom to full view button  . You should now see the following waveform. If you do not, you need to go back and check your code to see if there are any mistakes.



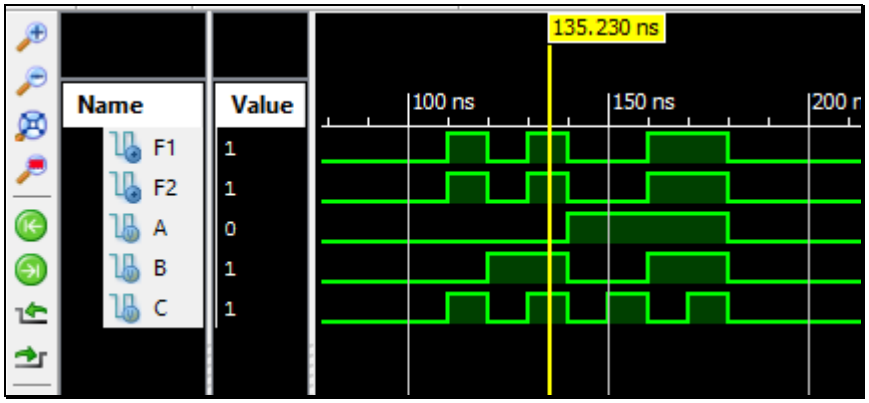
To expand you can click on the + magnifier and move the slider at the bottom over (this is similar to adjusting the settings on the oscilloscope).



In the waveform, note that the starting at 100 ns, the values of ABC count from 000 to 111 and at 180 ns go back to 000. Reading the simulation waveform is similar to reading the oscilloscope waveforms except in this case we can see all inputs and outputs together. This is a very useful function and in fact, there are logical analyzers that are similar to oscilloscopes that capture and display multiple inputs and outputs in a digital hardware design.

Note that F1 and F2 have the same waveform which proves that they are the same function. If you do not obtain the correct results, there must be an error in your design that you need to fix. In this example, if your results do not show that F1 and F2 are the same, there is an error in your design in the consensus module. Go back and fix it and repeat the steps to simulate the code.

The yellow bar can be moved over the waveforms to check the values of the inputs and outputs at different times during the simulation. For example, at time 135.23 ns, ABC are 011 and F1 and F2 are 1.



Show your instructor your code and simulation results. Also, print out and paste the code and simulation results into your lab notebook.

Answer the following question in your lab notebook.
Question: Describe intuitively why the term **BC** is redundant in **F1**. Hint: consider the other two terms **AB** and **A'C** and the possible values of **A**. (Example: if A=1, then F1 = BC + B. Is BC needed?).

5.3* Use Xilinx ISE to implement the three functions for the majority voter in Verilog and run a behavioral simulation to verify that the three functions are the same.

F1 = AB + BC + AC

F2 = Sum of Minterms (determined in part 5.1)

F3 = Product of Maxterms (determined in part 5.1).

Follow the process outlined in section 5.2 to design your circuit in Verilog using structural modeling and to simulate your design. Show your instructor your code and simulation results. Also, print out and paste the code and simulation results into your lab notebook.

Answer the following question in your lab notebook.
What is the benefit to designing (and simulating) a circuit in a hardware description language (HDL) such as Verilog before building it in hardware?