

University of Utah

Computer Engineering Program

Digital System Design

A comprehensive tutorial for the HW/SW platforms

Xilinx ISE 14.7 and Digilent Nexys 3

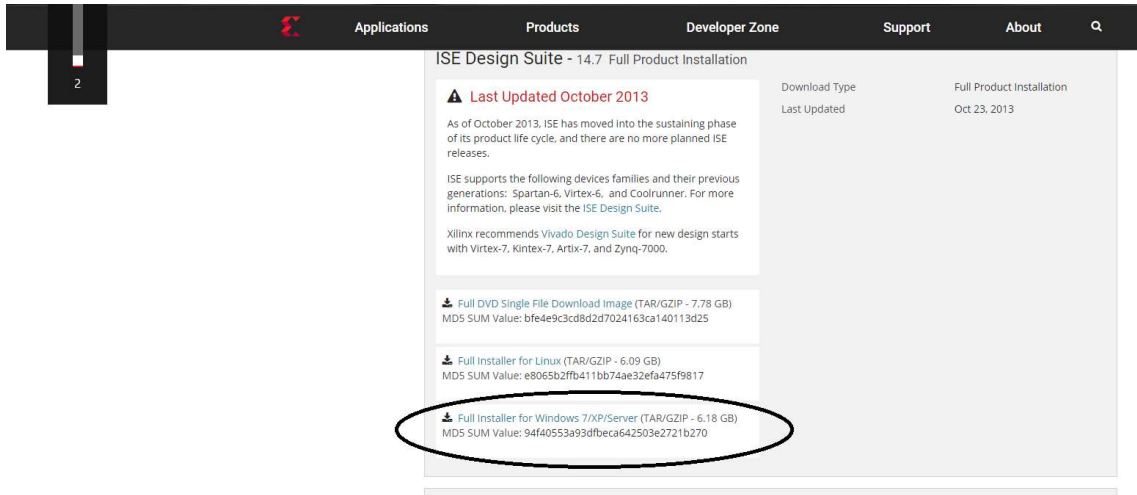
This tutorial will show you how to:

- Part 0: Download and install ISE 14.7
- Part I: Set up a new project in ISE 14.7
- Part II: Implement a function using Schematics
- Part III: Implement a function using Verilog HDL
- Part IV: Simulate the schematic/Verilog circuit using the ISim + Verilog test fixture
- Part V: Constraint, Synthesize, Implement, Generate, and Program for Nexys 3 FPGA board

Installation Notes for Xilinx 14.7 on Windows 10

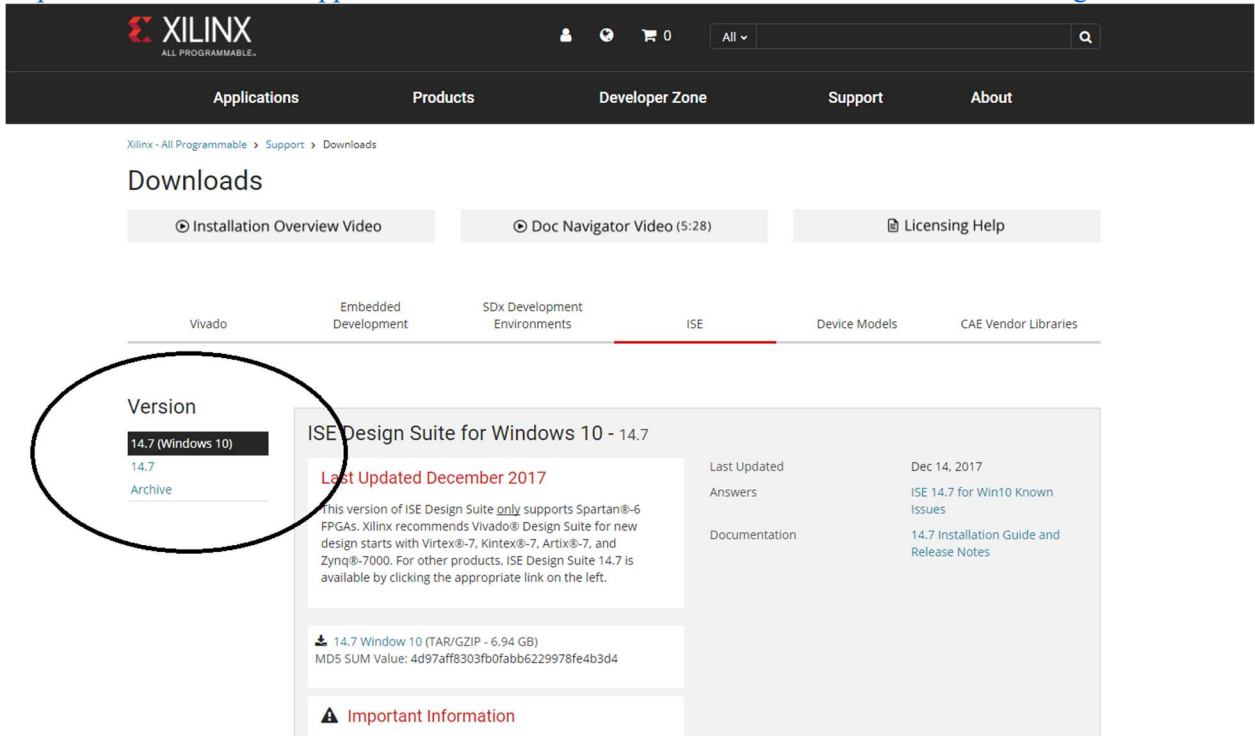
1. These instructions are for installing the Xilinx ISE version 14.7 on Windows 10 only. If you are using Windows 7/XP, then you can follow the instructions in Part 0 in the original manual written by Prof. Erik Brunvand from Page 4 onwards. In that case, please use the following link instead of the link provided in the step 1 on Page 4. The next screenshot is for the following link.

https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools/v2012_4---14_7.html



2. For using the tool on Windows 10, there are two options: Go to the following link,

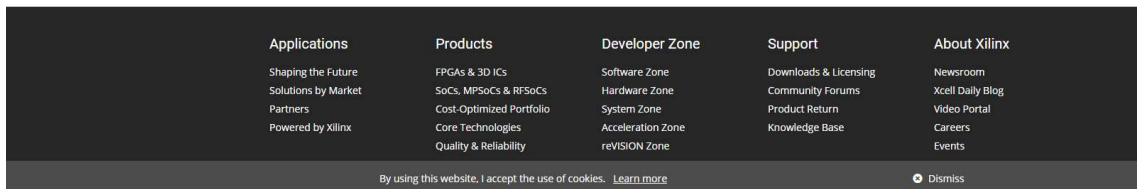
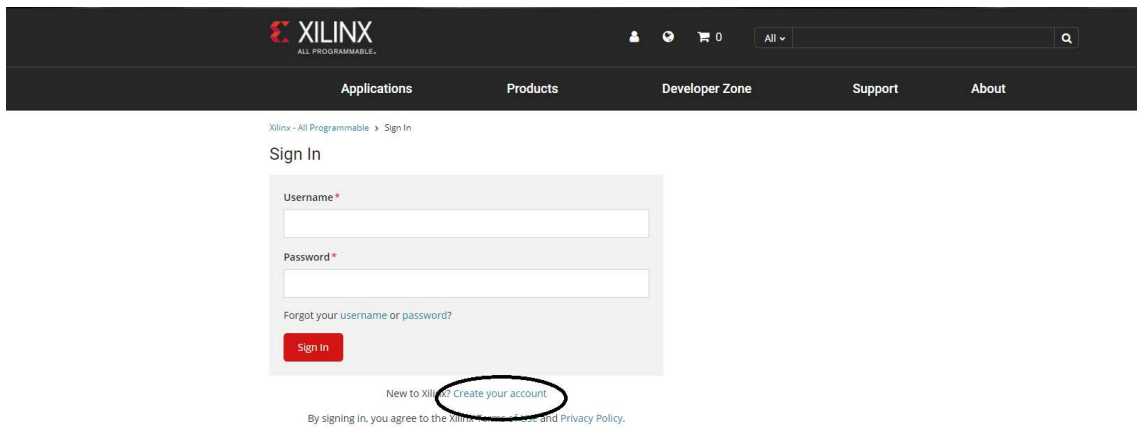
<https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html>



The first option is 14.7(Windows 10) which supports Windows 10 Pro or Windows 10 Enterprise. Since I don't have either one of those (I have Windows 10 Home), I had some errors while running it after

installation. But I can tell you that it installs a virtual machine (VM) with a Red Hat linux OS with Xilinx ISE already installed. Xilinx ISE needs to be run in that VM. If you have either Windows 10 Pro or Windows 10 Enterprise, you can try installing this version. We can help you out if there is some problem.

3. The other option (which I used) is to use the 14.7 option highlighted in the above figure. The problem with this version is that it is meant for Windows 7/XP and causes problems running the appropriate binaries after the installation. But fortunately, Rajath and I were able to come up with work arounds that would be sufficient for this course. I will go through the process in the next steps.
4. Click the 14.7 link in the figure for step 2. Then find and click the link as shown in the figure for the step 1. You will then be prompted to login to your Xilinx account.



If you don't have an account, then create one using the link in the above figure. After creating your account, you will need to fill out the name and address verification questions. Continuing through that, you will get a prompt to download the Xilinx_ISE_DS_Win_14.7_1015_1.tar file.

5. After downloading the tar file, use some file extraction software like 7zip to extract it into some directory. Next, follow the steps 3-5 on the page 5.
6. This is a large software, so it may take anywhere from 30 minutes to an hour, but keep in mind that you will be prompted a couple of times when it gets to about 90% of the installation. Accept the third-party installations offered through those prompts.
7. From this step onwards, I am assuming that you installed the software in the C:\Xilinx directory. Please make the changes accordingly if you installed in some other directory.
8. The shortcuts that you will get on the Desktop or Start menu folder (after the installation is finished) are all linked to the 64-bit binaries. We need to force all these shortcuts to run the 32-bit version instead so that they run on this OS.

9. Right click the shortcut on Desktop, click on properties, and change the Target field from *C:\Xilinx\14.7\ISE_DS\settings64.bat C:\Xilinx\14.7\ISE_DS\ISE\bin\nt64\ise.exe* to *C:\Xilinx\14.7\ISE_DS\settings64.bat C:\Xilinx\14.7\ISE_DS\ISE\bin\nt\ise.exe*
10. Then we need to add the free WebPACK license to the Xilinx ISE software. For this go to the folder *C:\Xilinx\14.7\ISE_DS\common\bin\nt* and run the executable **xlcm.exe** and follow the steps 6 and 7 on Page 5 after the window **License Configuration Manager** opens. The figure in step 7 (Page 6) tells you to select the option **Vivado Design Suite (includes ISE): WebPACK license**. But as Xilinx website has been updated, you will need to select the option **ISE WebPACK License**. You will then receive a Xilinx.lic file in your email (that you used to make the Xilinx account). Download this file and keep somewhere safe (probably in the C:\Xilinx\14.7 folder). In the **License Configuration Manager** window, go to the tab Manage Licenses, click **Load License...** and navigate to the Xilinx.lic file. This will make the ISE software use this license when you start your project.
11. This sets up your Xilinx ISE software and the license. There are two more fixes that we need to perform to run the inbuilt simulator for testing your code and to run the software PlanAhead that maps the inputs/outputs to the switches and LEDs on the FPGA.
12. Download the zip file **sim_planahead_fix.zip** from the class website. Unzip this file. You will find two files in the folder one each for the two fixes mentioned above.
13. Go to the folder *C:\Xilinx\14.7\ISE_DS\ISE\bin\nt*. There, find a file called fuse.exe. Rename this file to something like fuse_orig.exe, and then copy (in this directory) the fuse.exe file from the folder you unzipped in the last step. I renamed the original file so that we will have a copy of the original file in case we need it later. This should fix the error which occurs while using the simulator.
14. Go to the folder *C:\Xilinx\14.7\ISE_DS\PlanAhead\bin*. There, rename the file rdiArgs.bat to something like rdiArgs.bat.orig, and then copy (in this directory) the rdiArgs.bat file from the folder you unzipped in the step 12. This should fix the error while running the PlanAhead utility. These are all the errors and their fixes while trying to run the 14.7 Xilinx ISE for Windows 7/XP on Windows 10. After following through these steps go to the Part I in the original manual on Page 7 and start your first project. Start the project navigator from the Desktop shortcut (so that it runs the binary in the target *C:\Xilinx\14.7\ISE_DS\settings64.bat C:\Xilinx\14.7\ISE_DS\ISE\bin\nt\ise.exe*)

There are other fixes available too. I have mentioned the solution that I used for the installation on my system. We (TAs) will be happy to help in case you have any questions with any of the steps or if you still get some error in running the software.

Websites/Forums referred:

<https://www.eevblog.com/forum/microcontrollers/guide-getting-xilinx-ise-to-work-with-windows-8-64-bit/>

<https://forums.xilinx.com/t5/Installation-and-Licensing/pn-exe-crash-in-ISE-14-5/m-p/468650/highlight/true#M7071>

<https://forums.xilinx.com/t5/Simulation-and-Verification/ERROR-Simulator-861-Failed-to-link-the-design/m-p/355863/highlight/true#M7719>

Part 0: Download and install ISE 14.7

1. Go to the Xilinx website and to the **Downloads** option under **Support** and select the **ISE Design Tools** tab, or follow the link below:

<http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html>

Version

14.7

14.6

14.5

14.4

14.3

14.2

14.1

13.4

13.3

13.2

13.1

12.4

12.3

12.2

12.1

11.5

11.4

11.3

11.2

11.1

10.1 SP3

10.1

Archive

Multi-File Download: ISE Design - 14.7 Full Product Installation

| | | |
|---|-------------------|---|
| All Platforms - Split Installer Base Image - File 1/4 (TAR/GZIP - 1.95 GB) MD5 SUM Value: f0f8a08aba2b7110fa730c0b15087d6 | Download Includes | ISE Design Suite (All Editions) ISE WebPACK (Free) Lab Tools: Standalone Installation Platform Studio and Embedded Development Kit Software Development Kit (SDK) System Generator for DSP |
| Install Data A - File 2/4 (ZIP - 1.97 GB) MD5 SUM Value: c0962036464f9b772b20c032b2f954b | Download Type | Full Product Installation |
| Install Data B - File 3/4 (ZIP - 1.97 GB) MD5 SUM Value: e8146a7eac7c028b4b507f6b7549e4e | Last Updated | Oct 23, 2013 |
| Install Data C - File 4/4 (ZIP - 1.98 GB) MD5 SUM Value: 90943813f27a083e9829f3e742416417 | Answers | 14.7 - Release Notes ISE Design Suite 14 - Known Issues |
| Important Information Having trouble downloading? The download links above require the installation and use of a browser-based (plug-in) download manager. Your company's policy and/or firewall settings may not permit the download manager to be installed or operate properly. If you wish to bypass the use of the Xilinx download manager, please see AR# 57840 . | Enablement | License Solution Center |
| | Order DVD | ISE Design Suite DVD |

ISE Design Suite - 14.7 Full Product Installation

| | | |
|--|---------------|---------------------------|
| Full DVD Single File Download Image (TAR/GZIP - 7.78 GB) MD5 SUM Value: bf429e2e285277024433ca140113d25 | Download Type | Full Product Installation |
| Full Installer for Linux (TAR/GZIP - 6.09 GB) MD5 SUM Value: e8085b2fb411bb74ae32efa475f9817 | Last Updated | Oct 23, 2013 |
| Full Installer for Windows (TAR/GZIP - 6.18 GB) MD5 SUM Value: 94f40553a93dfbeca842503e2721b270 | | |

2. Click on the appropriate full installer link depending on your operating system, and you will be prompted to log in. If you do not have one, you need to create an account with Xilinx (free) in order to get downloads and licenses. After putting in your information, you will be asked to allow the Akamai Download manager to proceed. This is the portal in which you will receive a .tar file and you will need to extract it. For Windows you can use the WINRAR program.

Sign in Language Documentation Download Contact Us Shopping Cart ()

enter keywords Advanced Search

Products Applications Support Buy About Xilinx

Sign in to Download File

User ID

Password

Forgot your password?

Sign In

Note: Xilinx employees should Sign In using their Xilinx ID and password.

Why register?

Get easy access to all your design needs when you register on Xilinx.com:

- Sign up for product alerts and news
- Join Xilinx support forums
- Download software and tools
- Order products and tools
- Get technical help
- Gain access to special content
- Register for training and events

Create Account

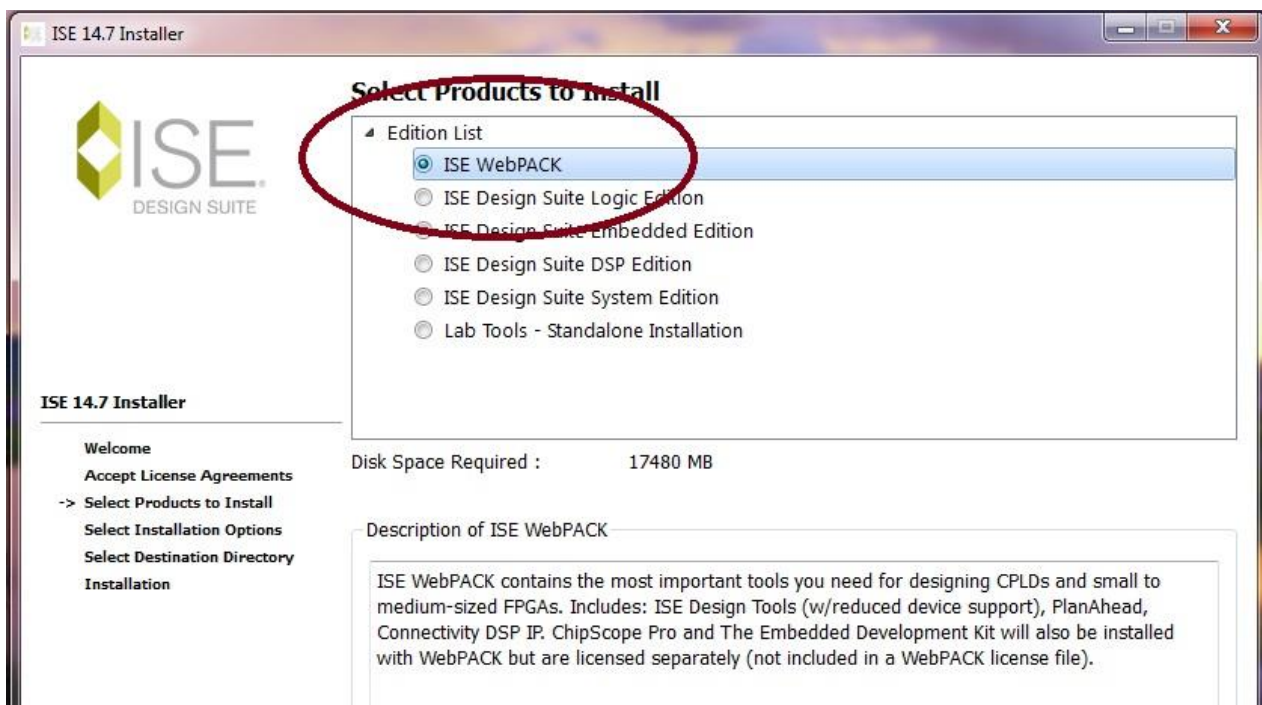
© Copyright 2013 Xilinx Inc. | Privacy | Trademarks | Legal | Sitemap | Feedback | Contact Us

About the Download Manager

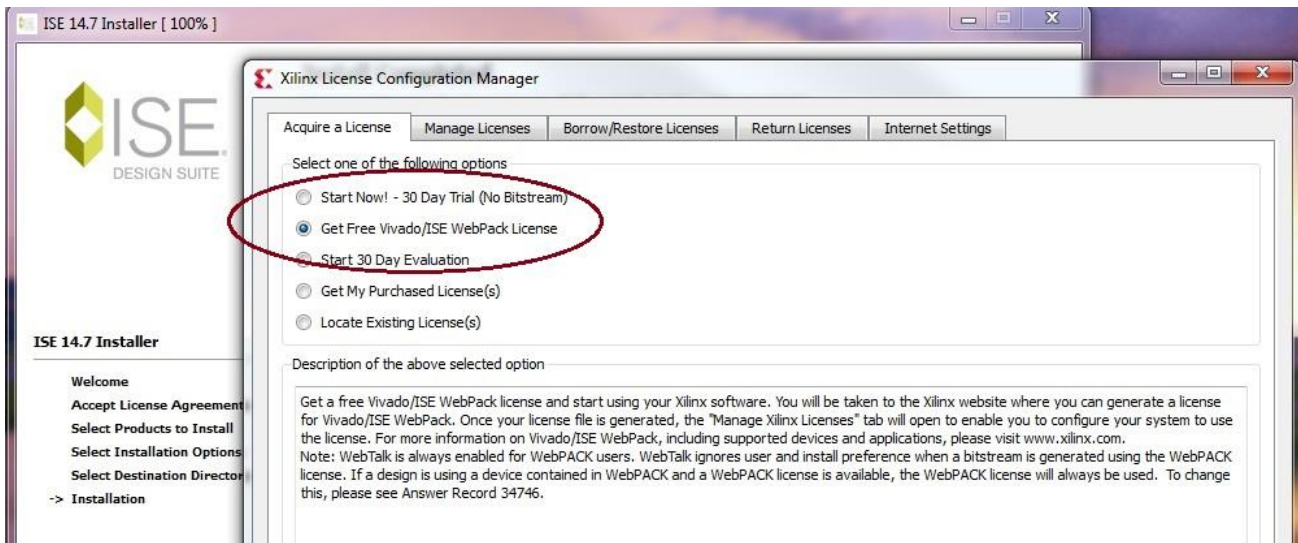
The Download Manager helps ensure that your Xilinx product download is successful. If your internet connection is temporarily lost or your computer is restarted, the Download Manager is able to continue the download process where it left off. You are being presented with a browser security warning regarding the installation of this Download Manager. By accepting the Akamai Technologies Inc. security certificate, the Download Manager will install and begin to download the requested file.



3. Go to the extracted folder and run **xsetup**, it will take a few moments for it to pop open, it is much better if you are not doing this over a network and your anti-virus is disabled,
4. After clicking **Next** on the initial page and accepting the terms and conditions (two pages) you need to select which package to install. You should download the first option **ISE WebPack** as it is the version that works with the free license, the other versions require a full license and they are about three GBs larger.



5. Do not change anything in the next page that is **Installation Options**, and leave all options checked. I recommend not changing the program path but do so if you want it installed in another specific place. Follow the summary page and click **install** when you get to it.
6. This is a large software so it may take anywhere from 30 minutes to an hour, but keep in mind that you will be prompted a couple of times when it gets to about 90% of the installation. When the installation is complete the **License Configuration Manager** will automatically pop up. Make sure to choose the second option **Get Free Vivado/ISE webpack License**, do not get the 30 day option as it will not be appropriate for our purposes. This program will not work without this free **Webpack** license.



7. Follow the instruction as you will be prompted to login to your account again and validate your information. Choose the first option **Vivado Design Suite (includes ISE): Webpack License**, which has no end date. Follow the rest of the instructions on the website and you will get node-locked with the license and ready to use ISE.

My Account | Sign Out | Language | Documentation | Downloads | Contact Us

enter keywords Advanced Search

Products | Applications | Support | Buy | About Xilinx

Home : Support : Product Licensing

Product Licensing

Create New Licenses | Manage Licenses | Legacy Licensing

Have a Voucher to Redeem? ?
 XXXXX-XXXXXX-XXXX-XXXXXXX
 enter voucher code Redeem Now

Evaluation and No Charge Cores ?
 Search the Evaluation and No Charge cores catalog and add specific cores to table below Search Now

Create a New License File
 Create a new license file by making your product selections from the table below. ?

Certificate Based Licenses

| Product | Type | License | Available Seats | Status | Subscription End Date |
|---|--------------------------|---------|-----------------|---------|-----------------------|
| <input checked="" type="checkbox"/> Vivado Design Suite (includes ISE): WebPACK License | Certificate - No Charge | Node | 1/1 | Current | None |
| <input type="checkbox"/> Vivado Design Suite (includes ISE): 30 Day Evaluation, Node-Locked License | Certificate - Evaluation | Node | 1/1 | Current | 30 days |
| <input type="checkbox"/> Petalinux SDK Evaluation License (No Support) | Certificate - Evaluation | Node | 1/1 | Current | 365 days |
| <input type="checkbox"/> Vivado HLS Evaluation License | Certificate - Evaluation | Node | 1/1 | Current | 30 days |

Part I: Set up a new project in ISE 14.7

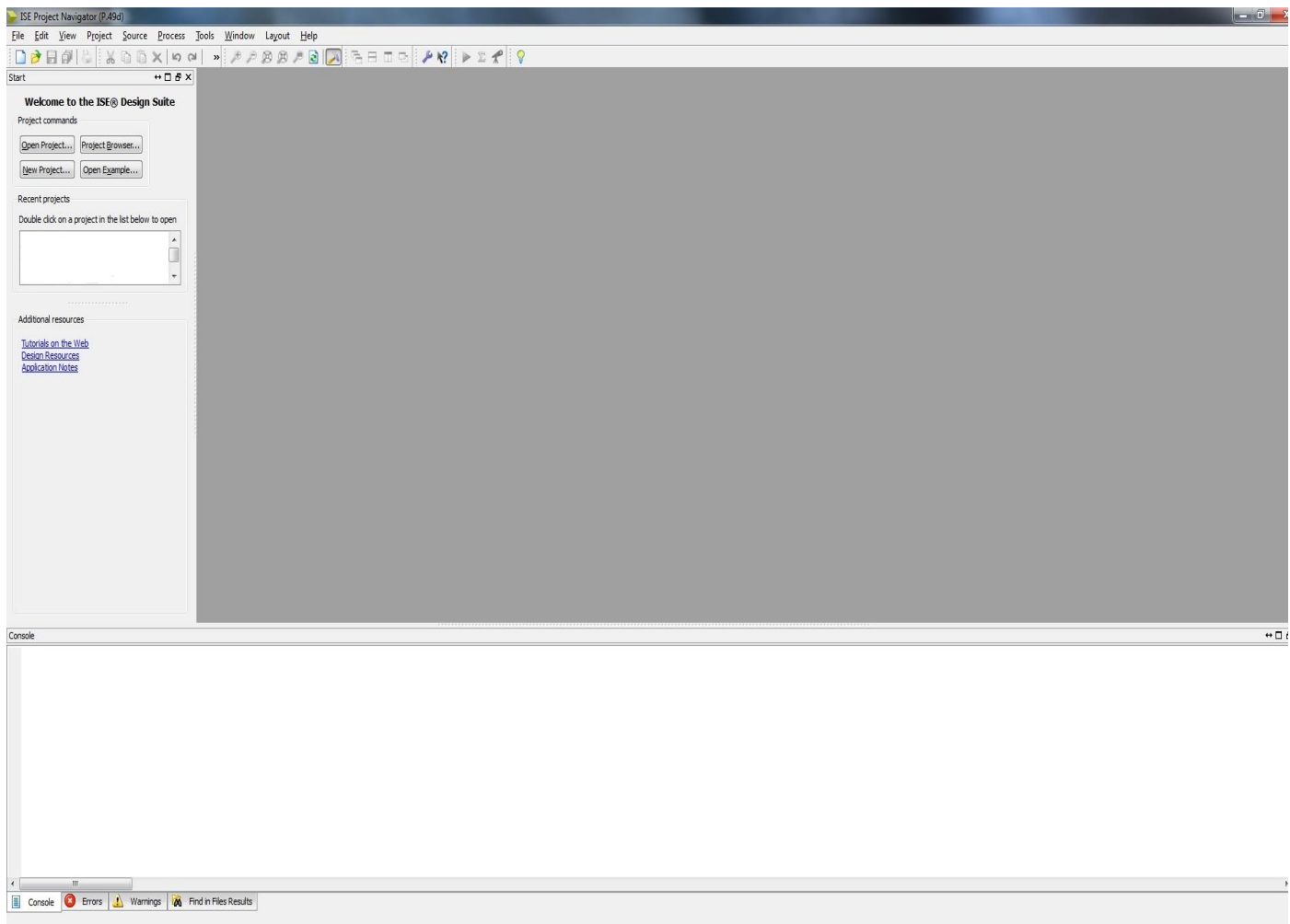
Attention: Make sure to use the appropriate version of the ISE, 64 bit navigator for a 64 bit OS, and 32 bit for 32 bit. If you don't pay attention to this, there will be unexpected behavior in the ISE software and thing may not work properly!

1. Open the **Xilinx ISE Design Suit 14.7**. You can click on the ISE icon on the desktop, or search **Start** → **All Programs** → **Xilinx ISE Design Suite 14.7** → **ISE Design Tools** → **Project Navigator**

If for some reason you cannot find the icons, you can also find the ISE project navigator by going to the following path and finding ise.exe (note nt64 for the 64bit version, go to the "nt" folder for the 32 bit version):

```
C:\Xilinx\14.7\ISE_DS\ISE\bin\nt64\ise.exe
```

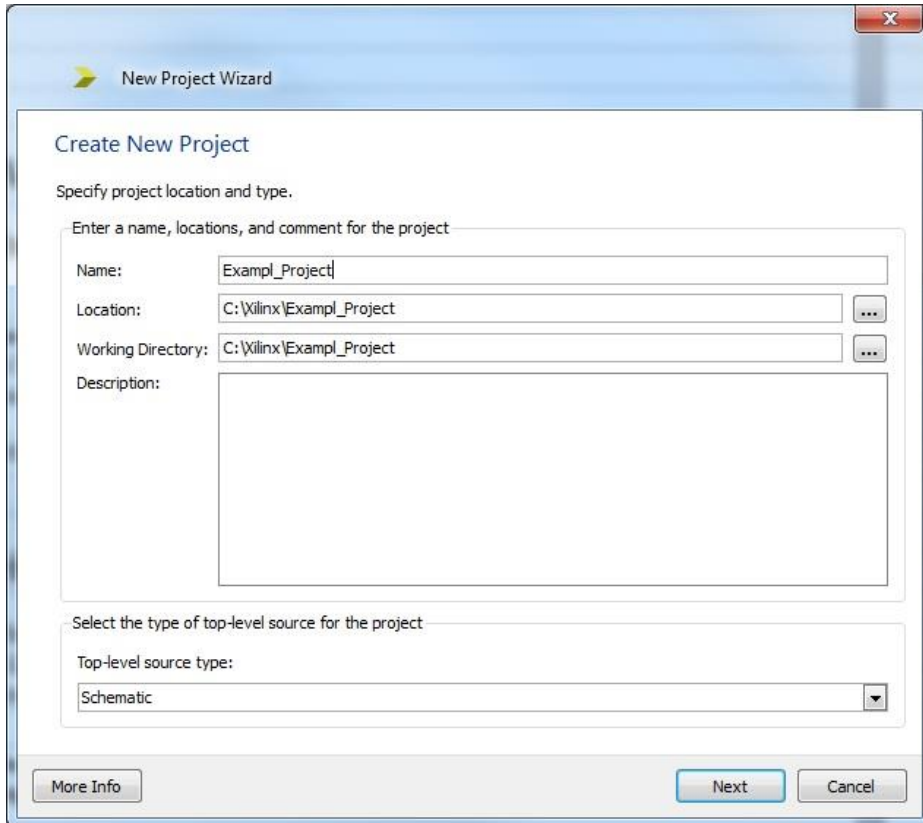
The screen should look something like the following, the ISE always defaults to the last open project unless none where open before just like the following:



- Now either press the New Project tab or select **File** → **New Project...** and change the **Name** and **Location** to whatever you like.

Attention: Xilinx does not allow spaces in path or file names! For example “C:\ECE 3700” will not work, same for the file name! Use the under_score for spaces if you need to.

The selected Top Level Source Type is **Schematic** because that’s what we’re planning on using first. This is not critical, as you can always add a new source file of any type later. The dialog box for the project wizard looks like:

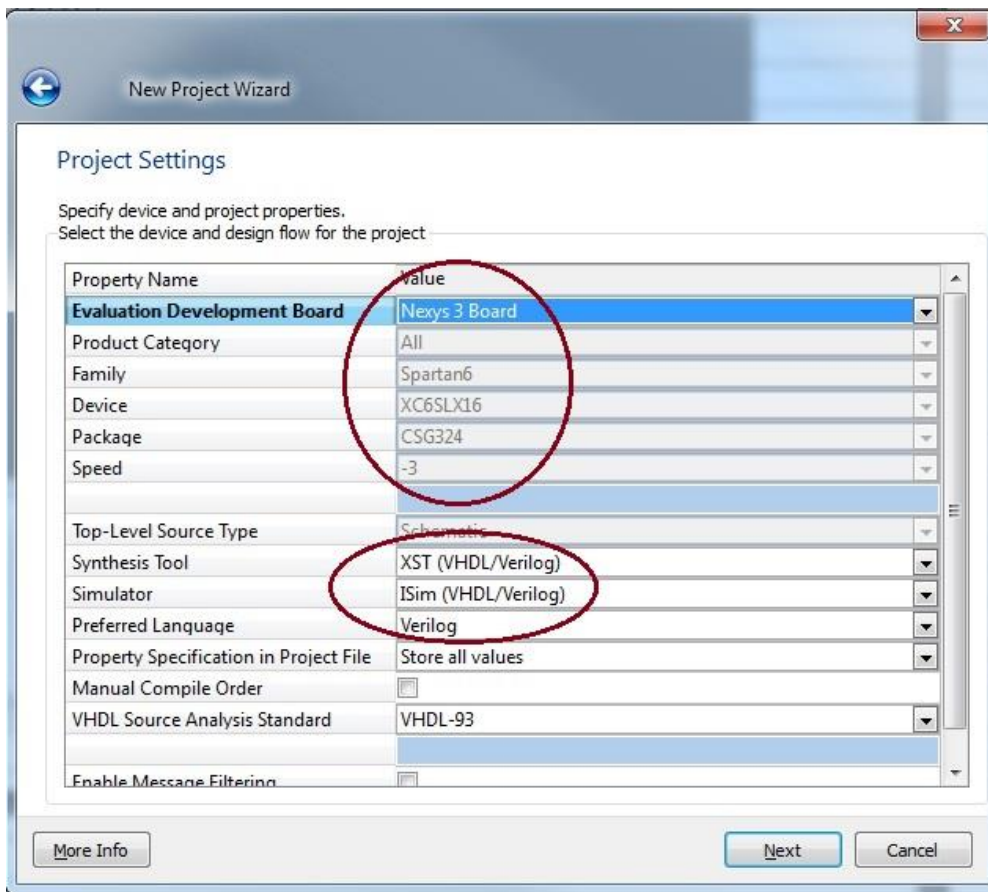


- Click NEXT and in the next dialog box you should fill in the fields as shown here. You can do this in two ways, one is to select an “**Evaluation Development Board**” from the drop down list, and in our case you should select “**Nexys 3**”. This will automatically fill out the board information in the next five sections. If the board does not exist in the list then you can set correct choices according to the following image.

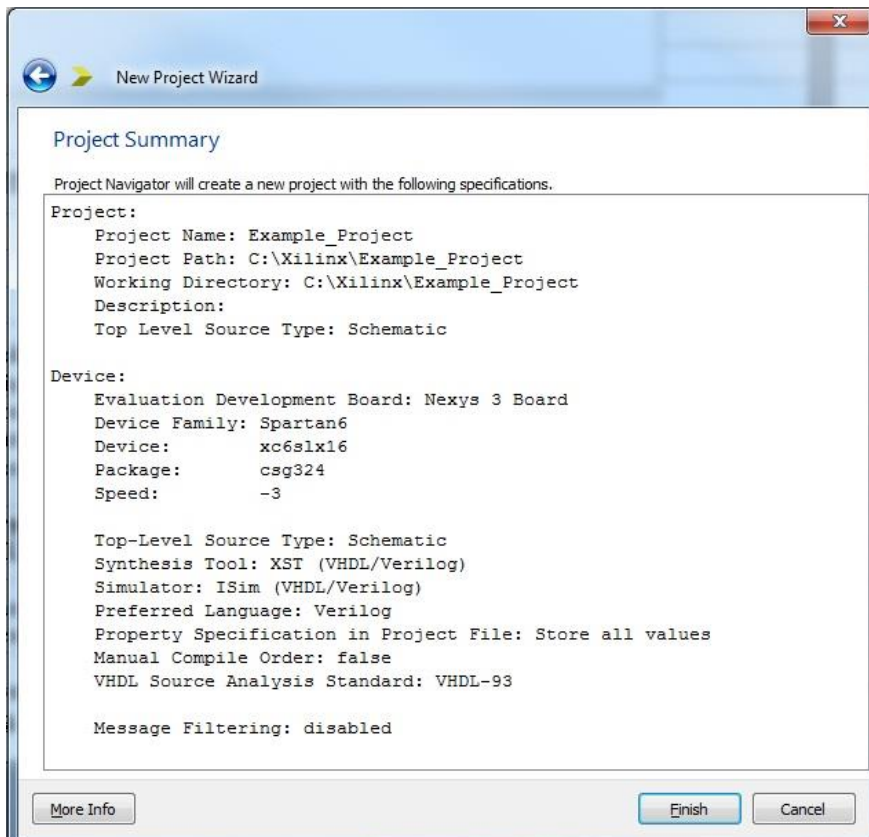
We are using a **General Purpose** product in the **Xilinx Spartan6** family. The specific chip on the Nexys 3 board is an **XC6SLX16** in a **CSG324** package and the **-3** speed grade.

Attention: If you fail to set the correct options in this part, you will not be able to implement your design and program it on the Nexys 3 board!

Please make sure that the Synthesis Tool is **XST**, the Simulator is the **ISim**, and the Preferred Language is **Verilog**. This is very important for proper operation.

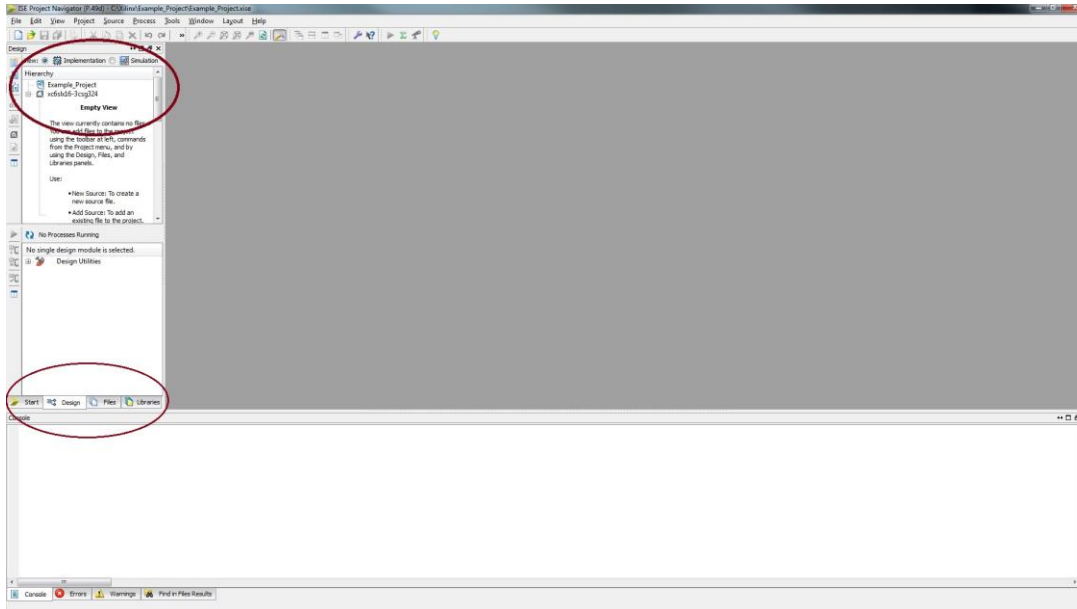


4. Click NEXT and review the project summary page and then click FINISH, it is always good to double-check the summary to prevent headaches due to the problems you can face while implementing your design if the information is incorrect.

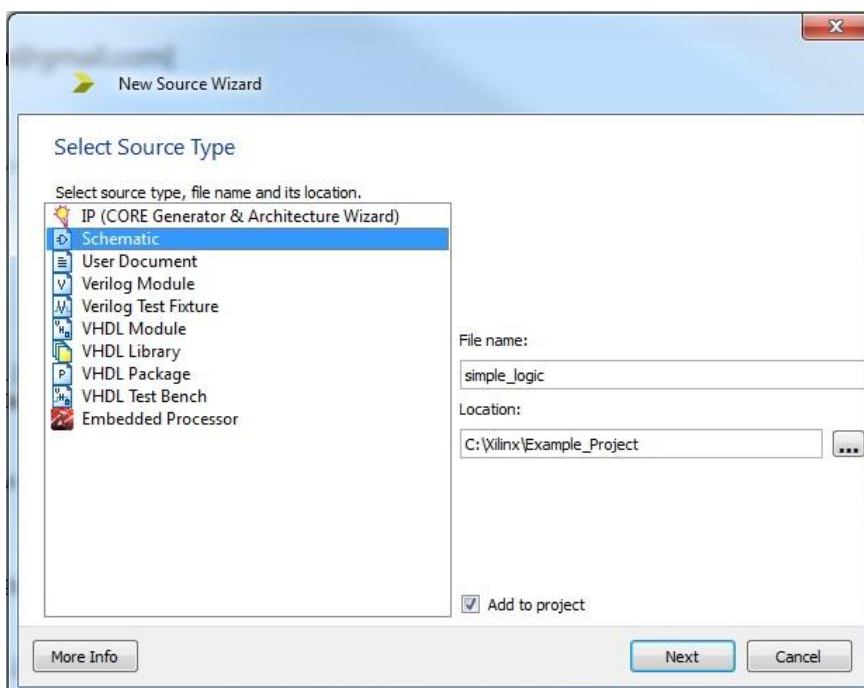


Part II: Implement a function using Schematics

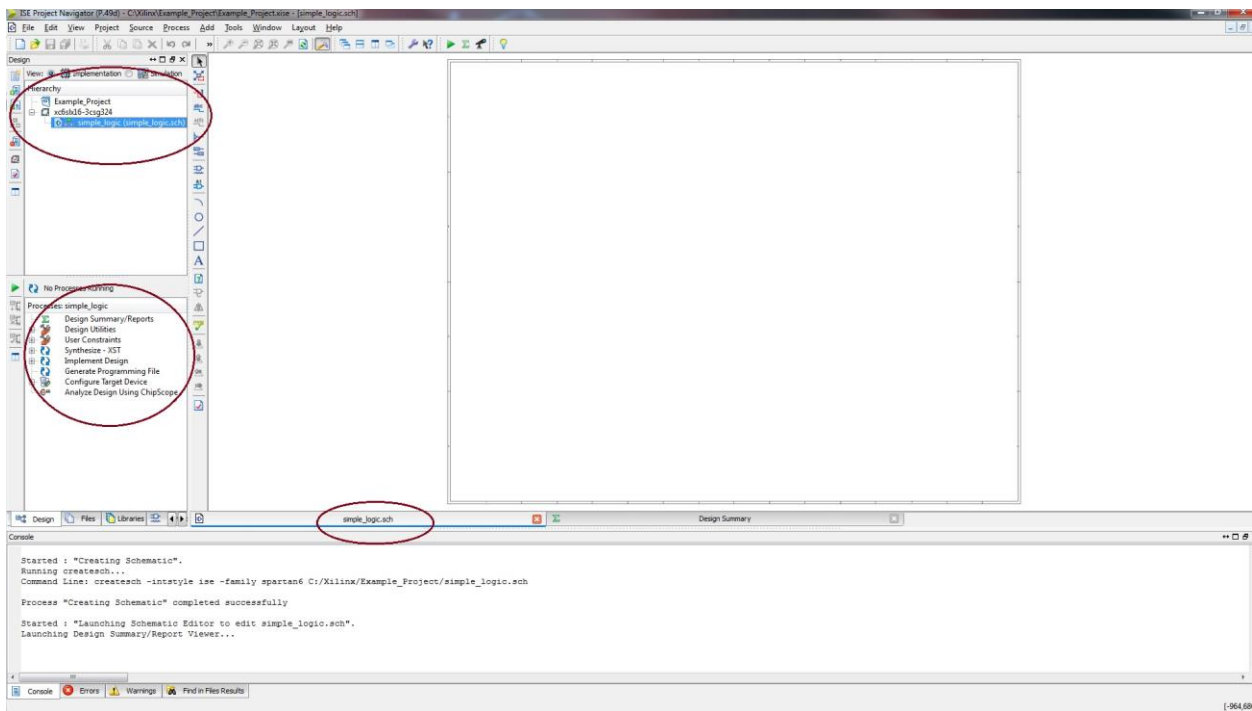
1. Now you should have a new project that targets the correct Xilinx part and other features of the ISE system. Notice the window to the left, and also notice the four tabs Start, Design, Files, ..., and in the case of an open schematic a Symbols tab will appear. Ensure that the “**Implementation**” choice is selected on the design pane.



2. Now you can create a new schematic in your project. Choose **Project** → **New Source** or right click on the Hierarchy section of the design windows to get the dialog box that adds a new source file to your project (or use the **New Source** widget on the left vertical tool bar). Source files can be of many types. Add a Schematic and name it **simple_logic** for example. Make sure to both fill in the **File Name** and select the **Schematic** type from the list on the left. I've filled in my dialog box as shown below. Now click NEXT and after observing the summary page click FINISH.



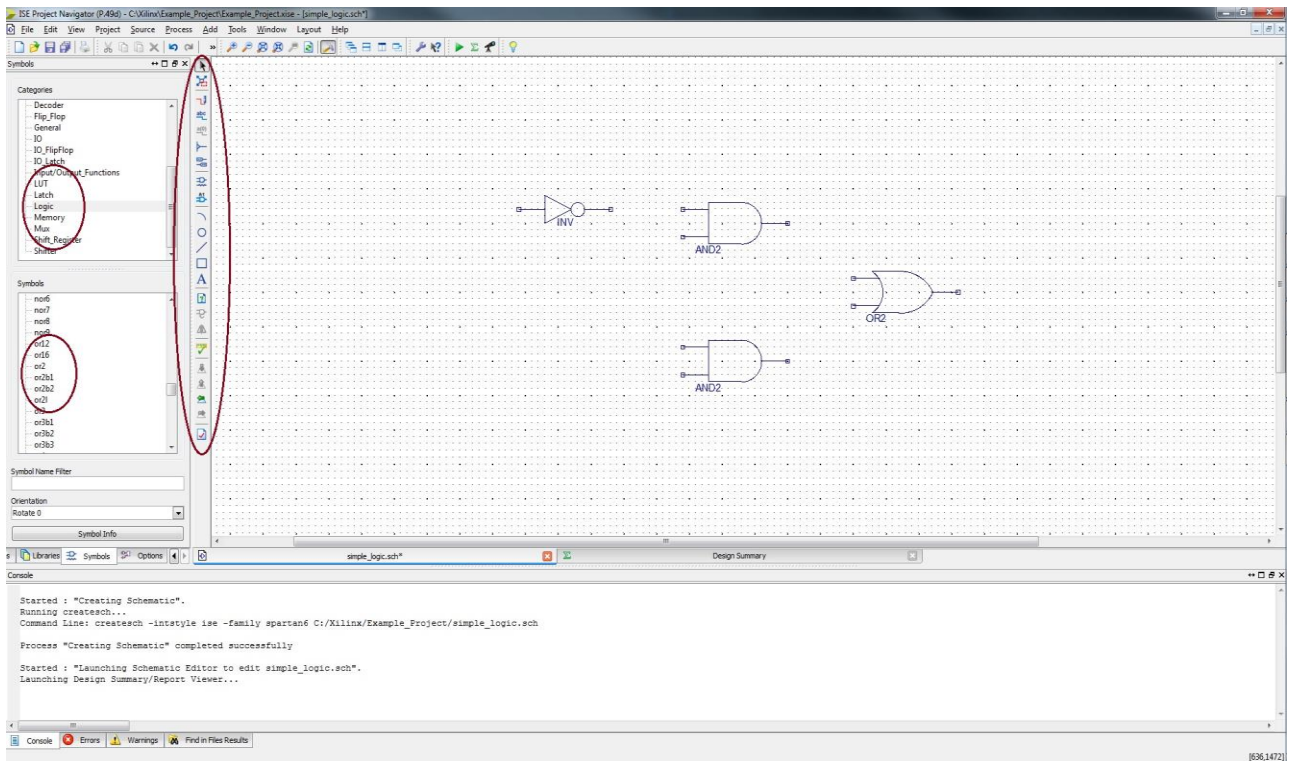
- Now you have a blank schematic view, and also a Design Summary view in the main pane of the ISE window. You can switch between windows in the main pane with the tabs along the bottom. The (empty) schematic looks like below. Also note the Hierarchy created and the number of processes such as “Synthesis”, “Implement”, “Generate”, etc. you can run on it.




- If you don't see this exact view, you may be looking at a different tab in a window. Each pane has tabs at the bottom that let you switch to look at different things. For example, for the left pane in the image above there are multiple tabs to allow different things, “Files”, “Snapshot”, “Libraries”, and “Symbols” tabs that you can click on and get different information about the project. The lower left pane is the Processes tab and Options tab that show different tools and steps. The main window in the figure above has a “simple_logic.sch” tab for the new schematic, and a “Design Summary” tab. If you open more schematics, or other types of files (like Verilog files) the main pane will have additional tabs.

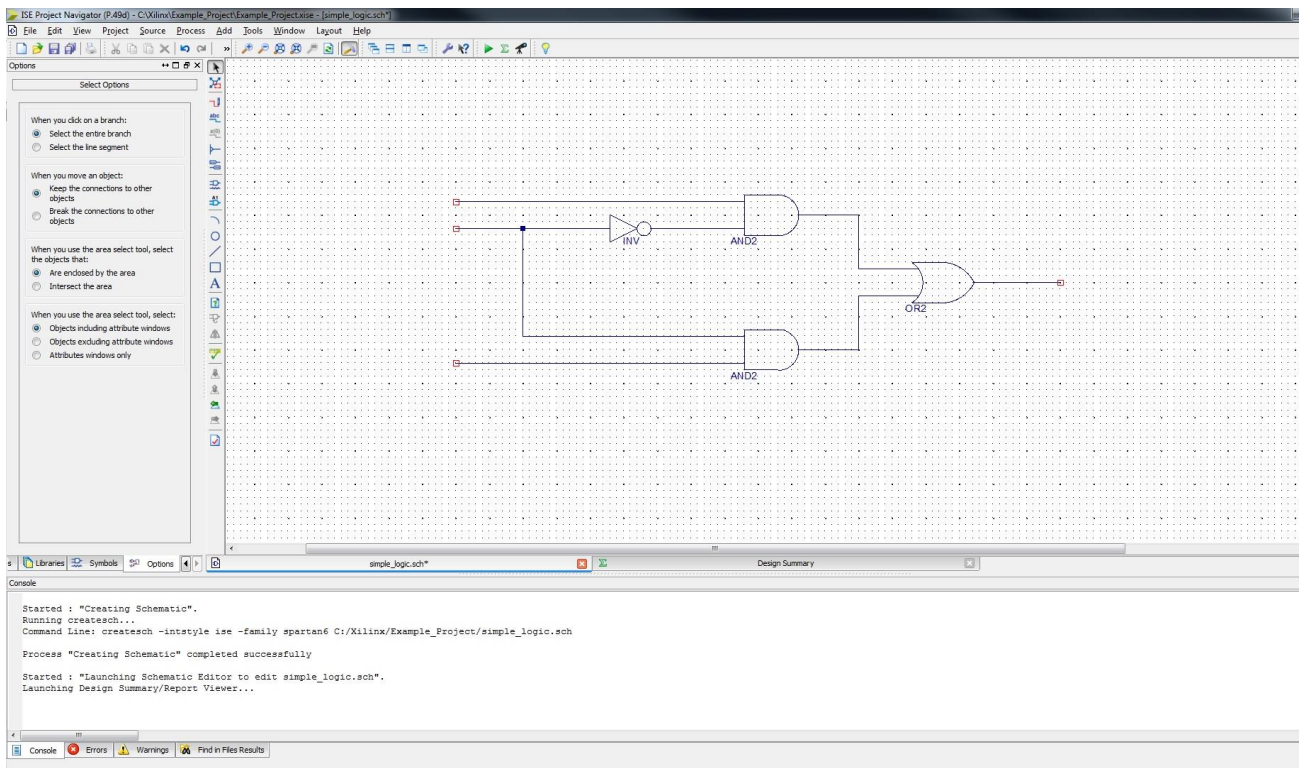
Now you can select components from the **Symbols** tab of the upper left pane and drag them to your schematic. You can narrow down your choices using the **Categories**, or by typing the first few characters of the symbol you're looking for in the **Symbol Name Filter**, or just scroll through the lists and see what's there. The important category for now is **Logic**: General logic gates. See the Lab handouts for restrictions on which logic gates you should use!

I'll grab some components from those Categories to make a very simple schematic to implement $(A \& (!B)) | (B \& C)$ and drop them into the schematic page. Now we need to add wires and I/O markers by using the tools on the vertical bar between the schematic page and the side pane. Also pay attention to the **Options** tab at the bottom of the left pane. This section gives you some options on the schematic such as “selecting the entire wire branch” or “selecting line segments”. This is good to know if you wanted to only remove a piece of wire and not everything that it is attached to.

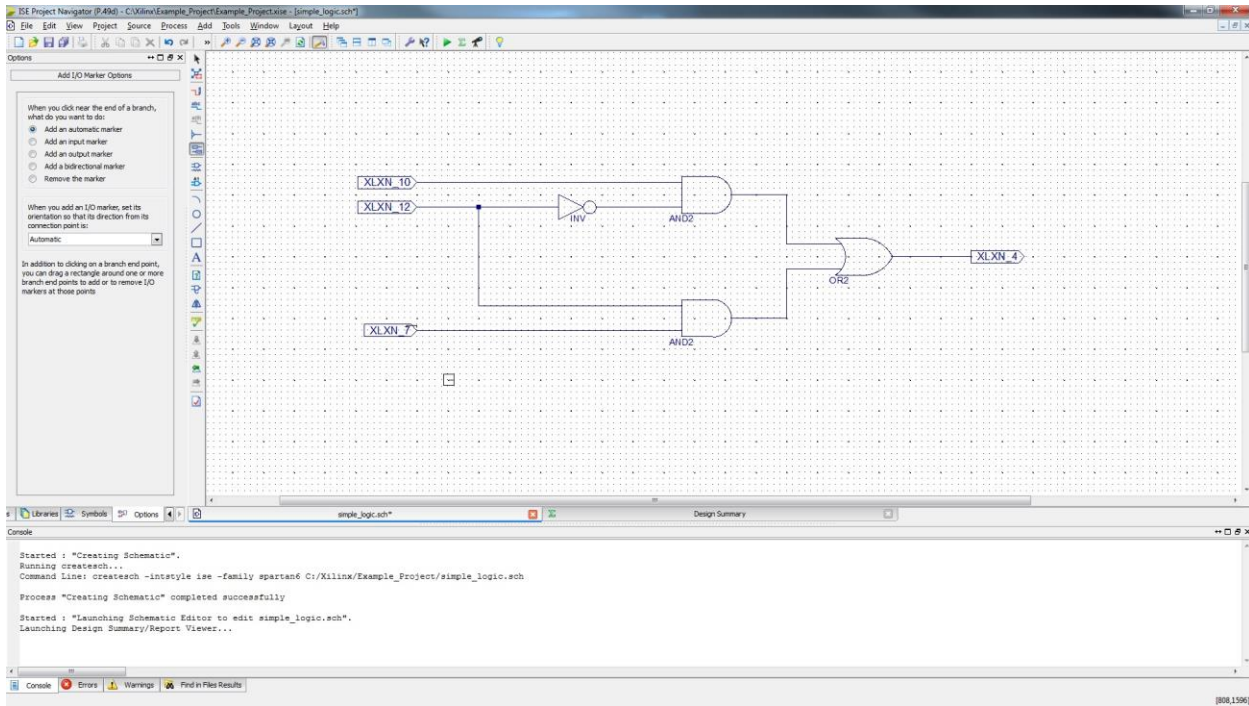


5. Use the wiring tool  to wire up the components. It is in the tool bar and looks like a red line and a pencil. You could also use **Add** → **Wire** from the menu. I'm using the following components (from the **Logic** category) in this example:

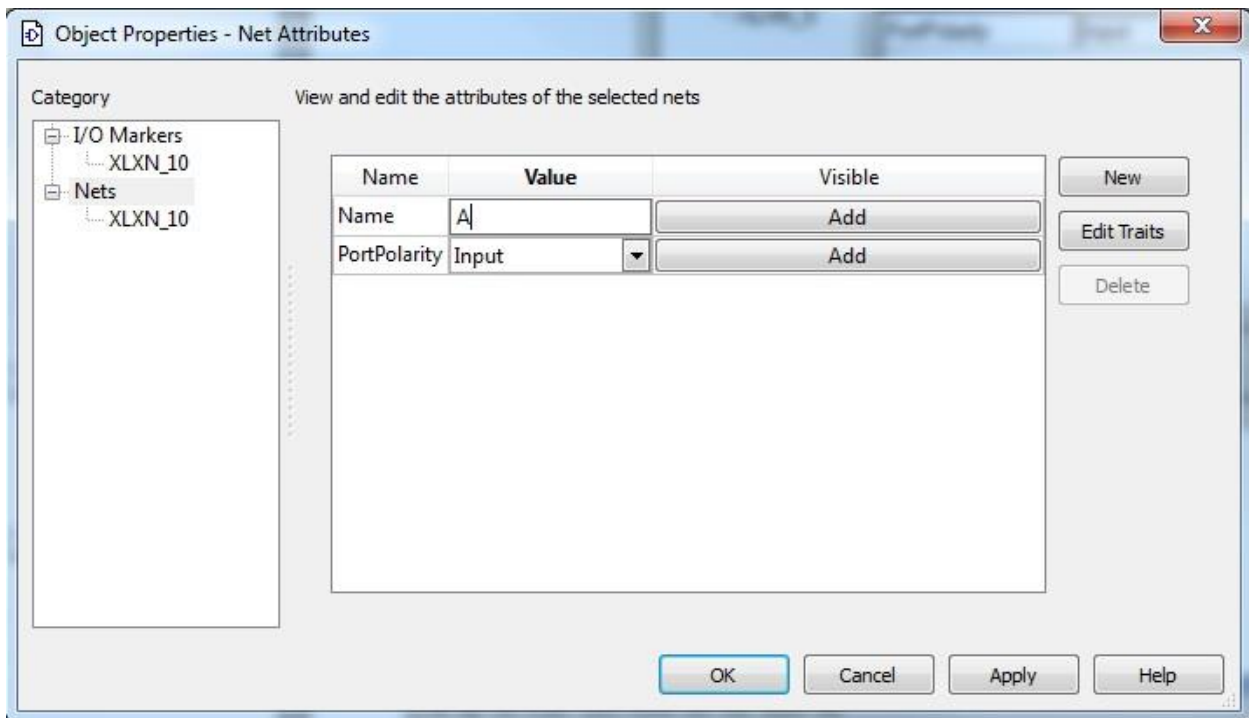
- a. **and2**: a two-input AND gate
- b. **or2**: a two-input OR gate
- c. **inv**: an inverter

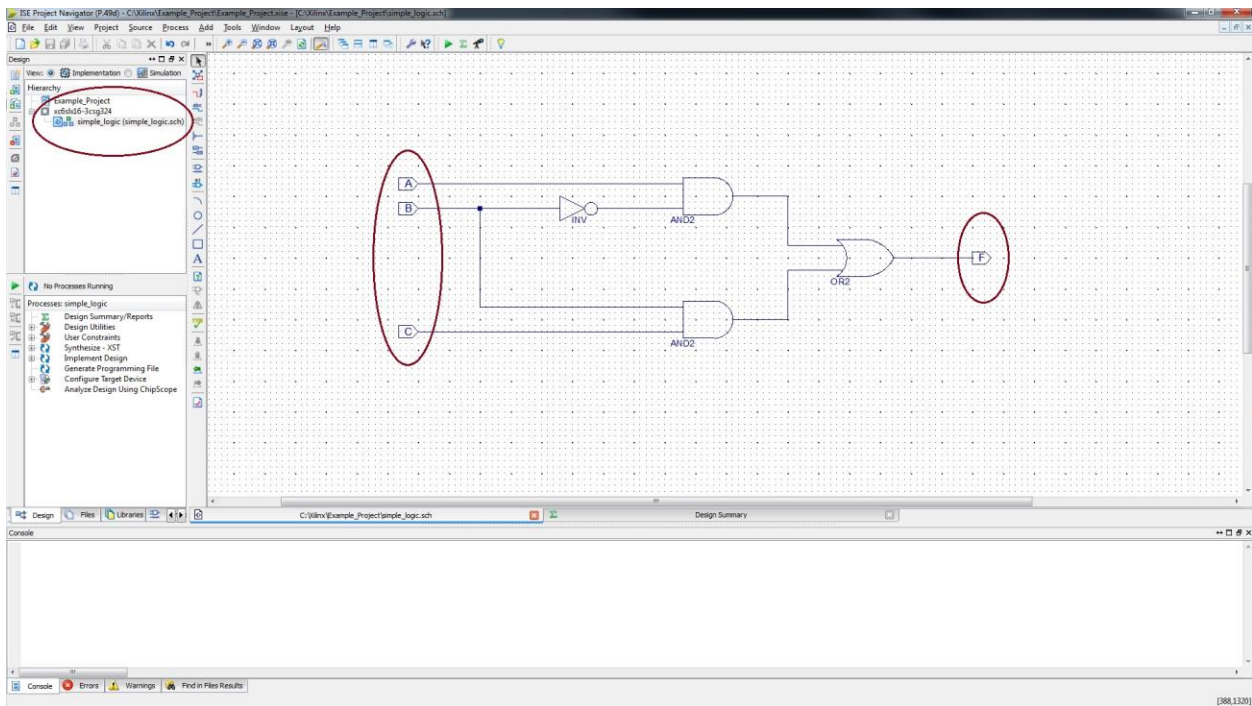


6. Now place **I/O Markers** to the inputs and outputs. I've used the **I/O Marker** widget that looks like two little labels. You can also use the **Add** → **I/O Marker** command from the menu. Click on the endpoints of the wire to add the marker.



7. You should always change the name of the marker to whatever you want but choose a good identifier. You should double click the marker, or select the marker and right click to get a menu and choose **Edit** → **Properties**. Then click on “Nets” and then edit the “Name”, also observe the Port Polarity, then click OK. I'm calling the inputs **A**, **B** and **C** and the output **F**. An I/O Marker dialog box looks like:





- Now the resulting schematic looks like the following and it is ready for simulation or synthesis. Once your schematic is saved it should show up in the **Sources** pane in the upper left of the ISE screen.

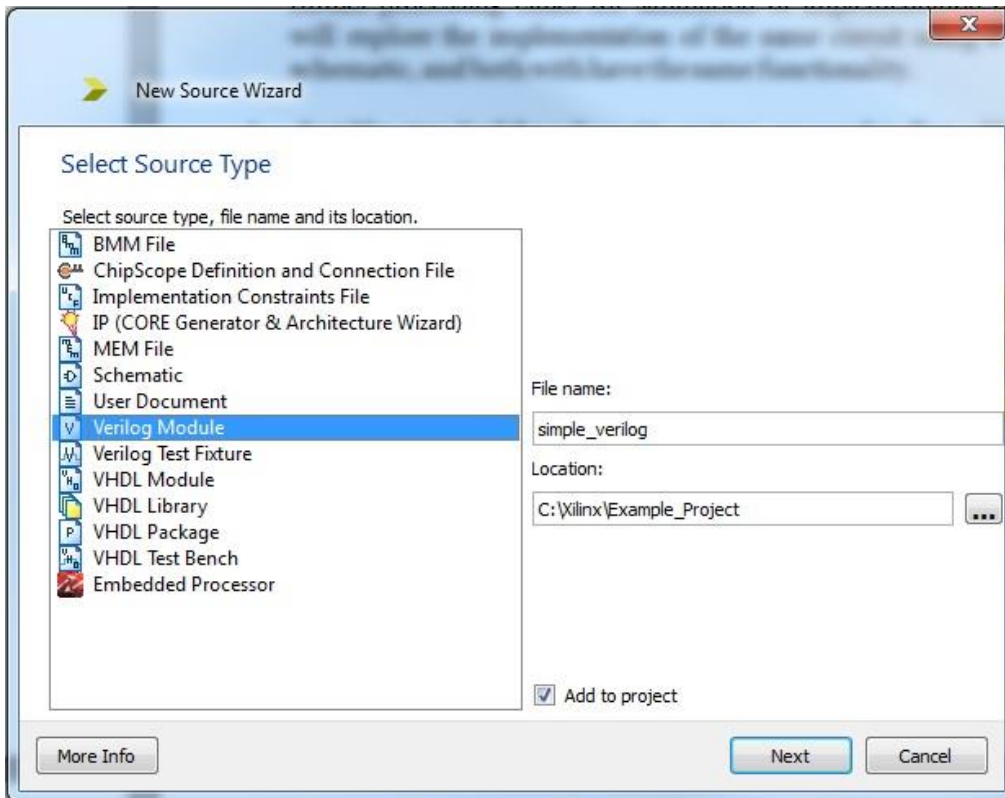
When you are happy with your schematic (and your schematic will likely include other logic gates from the Logic category that I haven't used in this example!), save it. Hopefully you don't have any errors or warnings. If you do, you need to fix them.

Part III: Implement a function using Verilog

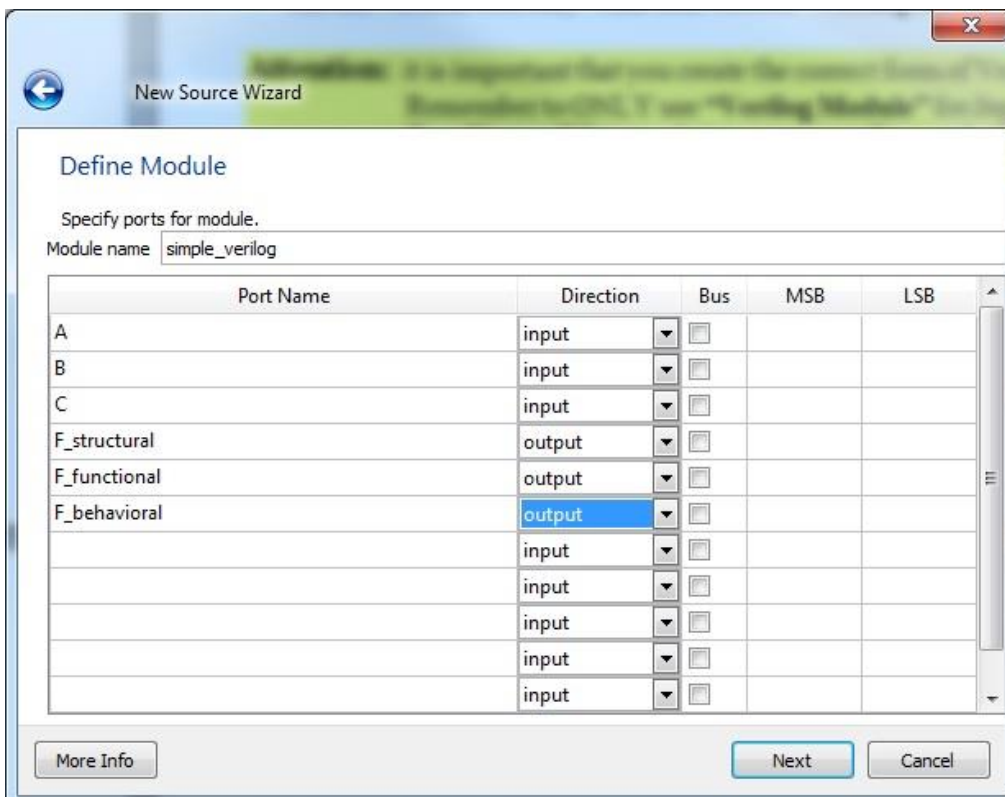
This schematic representation of the function $F = (A \& (!B)) \mid (B \& C)$ is now ready for further processing either for simulation or implementation on the Nexys 3 board. Now we will explore the implementation of the same circuit using a “**Verilog**” module instead of a schematic, and both will have the same functionality.

- Just like step 2 of the schematic capture, we need to first add a new source. So again choose **Project** → **New Source** or right click on the Hierarchy section of the design windows to get the dialog box that adds a new source file to your project (or use the **New Source** widget on the left vertical tool bar). This time choose “**Verilog Module**” and give it a file name.

Attention: it is important that you create the correct form of Verilog file for the specific use. Remember to **ONLY** use “**Verilog Module**” for *Implementation* and “**Verilog Test Fixture**” for *simulation* purposes. Do not do this the other way or things will not work right



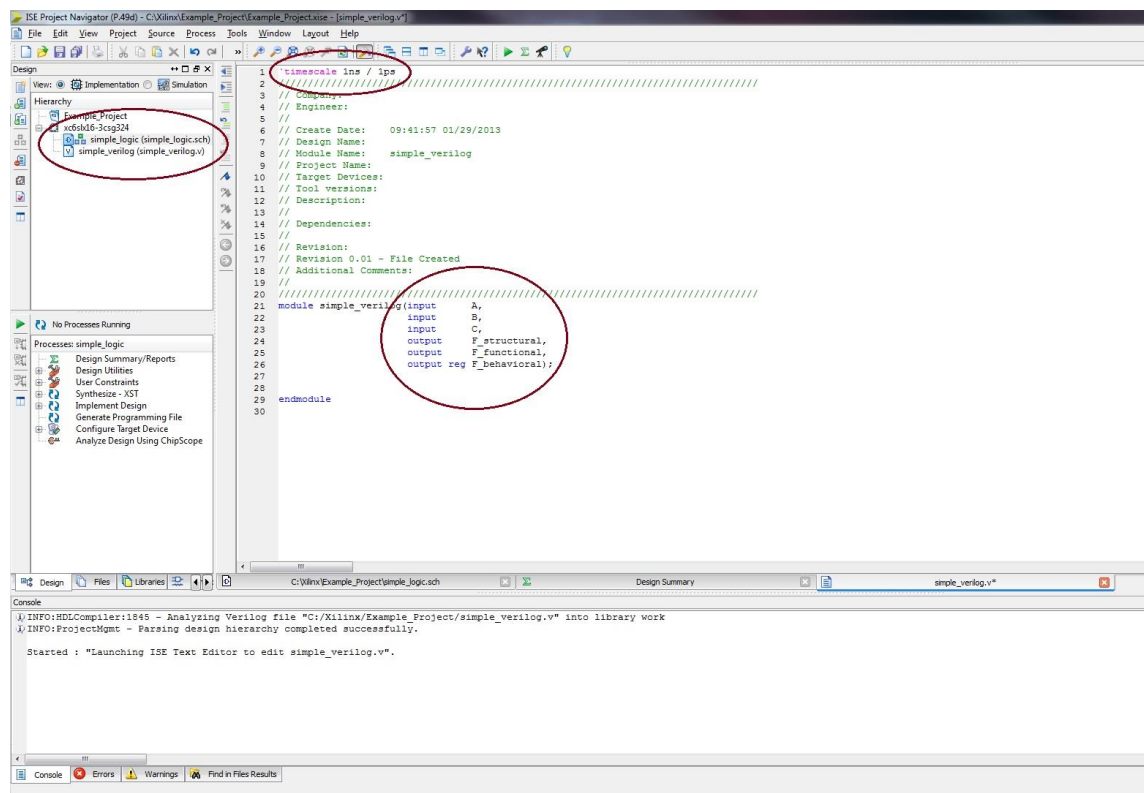
2. Click NEXT and you should see the module definition box. Here you can setup I/O names with correct polarity and a choice for buses and the width which we will be using in the future labs. Note that you do not have to add anything here right away and you can always add the I/O definitions to the module's header when it is created. After you're done click NEXT and then observe the summary page for a quick review of your I/O list.



- Now you should have a tab for the Verilog file opened in the ISE main pane. There is a line reading “**timescale 1ns / 1ps**”. Please leave line alone and never delete it. It just lets the ISE know that each unit of time should be 1ns and have the resolution up to 1ps for any timing purposes (more on this later).

Next is the module header that contains the module name and the list of I/O as its parameters. If you notice I have declared multiple outputs to implement the same circuit using different HDL forms when writing in Verilog. These will all implement the same function **F** represented by the schematic in Part I.

Also you should notice that the Verilog file is added to the Hierarchy next to the schematic as a part of this project. Now we’re ready to implement the Verilog description of the function **F** in the three intended forms.



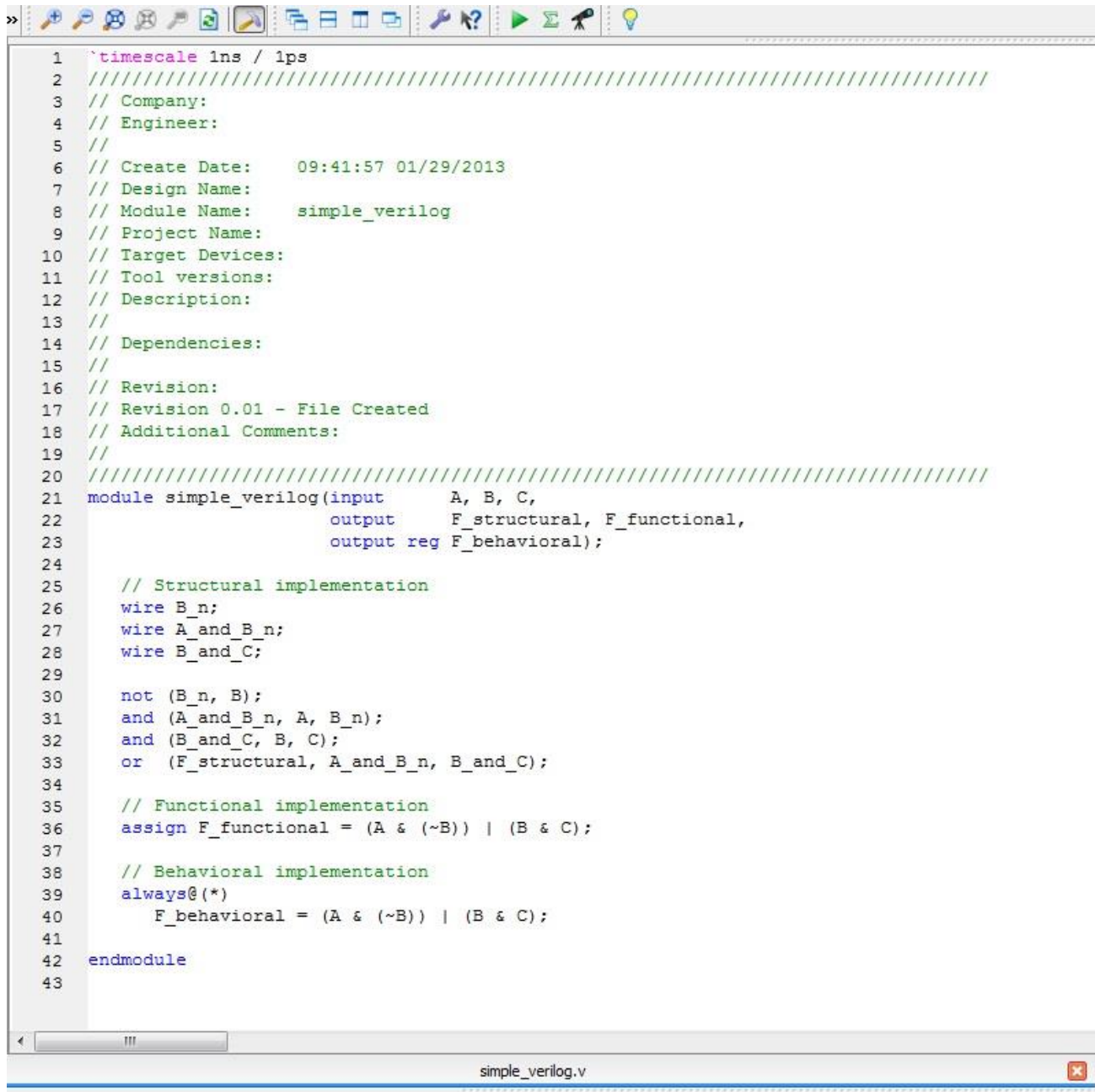
- For the implementation the three forms we have different formats. Sometimes we want to declare internal wires to connect things easily and in a more organized fashion, but if you didn’t declare these wires, ISE will assume a single bit wire for the use.

The Structural version: This is done using gate primitives that are automatically taken from the Xilinx libraries (just as you would with a schematic) by calling their name and passing parameters, so for a two input gate we have the format “gate (output, input1, input2)” and this is just like doing a schematic in words.

The Functional version: Using the “**assign**” keyword to assign the results of the function expression to the output. The expression of the function looks a lot like how you would write it down on paper.

The Behavioral version: Using a synthesis directive called an “**Always Block**” we can implement the same function. The difference is that it only wakes up and assign the output

when any of the inputs are changes, hence the (*) which means “any change”. We will be using this format extensively in the future labs. One thing to notice is that any output being assigned inside of an always block needs to be declared as a “reg” for synthesis purposes, and that you cannot use the “assign” keyword inside of such block.



```
1 `timescale 1ns / 1ps
2 ///////////////////////////////////////////////////////////////////
3 // Company:
4 // Engineer:
5 //
6 // Create Date:    09:41:57 01/29/2013
7 // Design Name:
8 // Module Name:    simple_verilog
9 // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21 module simple_verilog(input    A, B, C,
22                       output   F_structural, F_functional,
23                       output reg F_behavioral);
24
25     // Structural implementation
26     wire B_n;
27     wire A_and_B_n;
28     wire B_and_C;
29
30     not (B_n, B);
31     and (A_and_B_n, A, B_n);
32     and (B_and_C, B, C);
33     or  (F_structural, A_and_B_n, B_and_C);
34
35     // Functional implementation
36     assign F_functional = (A & (~B)) | (B & C);
37
38     // Behavioral implementation
39     always@(*)
40         F_behavioral = (A & (~B)) | (B & C);
41
42 endmodule
43
```

5. You should always save your work multiple times as there always a slight chance that the tool may crash and you will lose your work. It is important to understand that hardware CAD tools are massive and complex so there is always a chance for unexpected or faulty behavior.

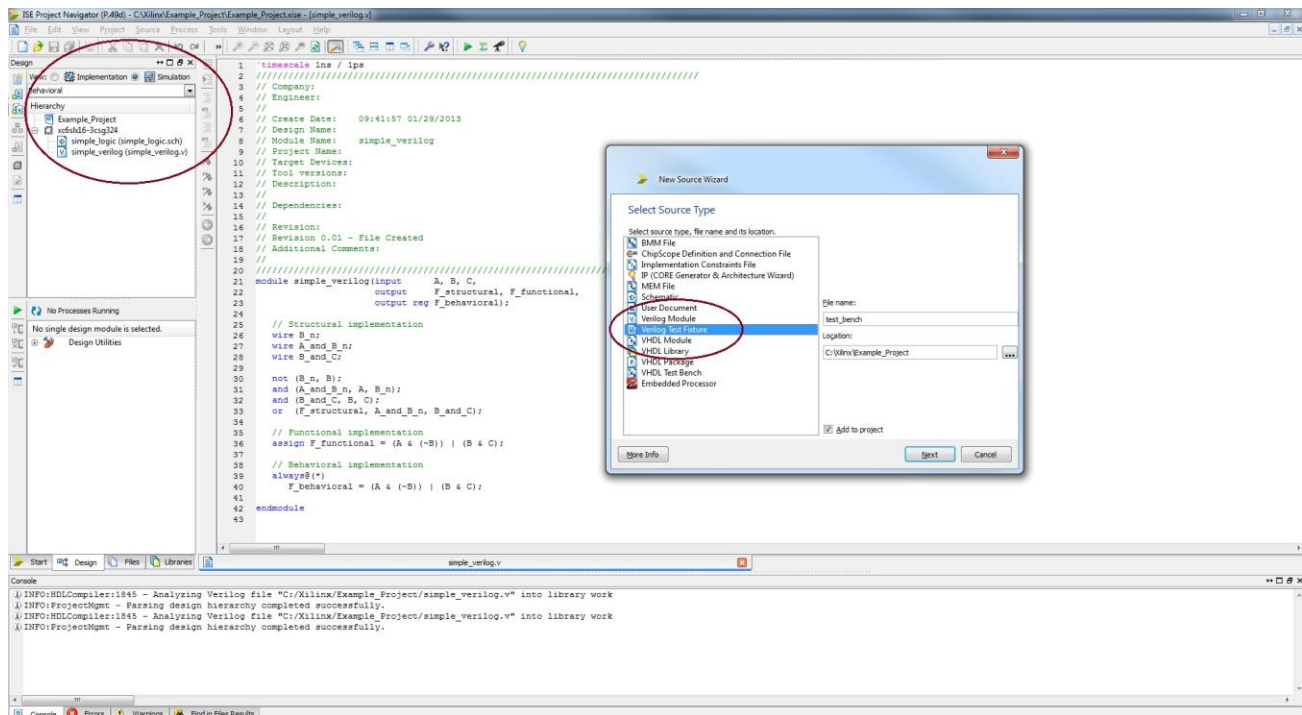
Now the circuit is ready for simulation or implementation on the board. It is important to notice that most processes are the same for both the schematic and the Verilog version of your design.

Part IV: Simulate the schematic/Verilog circuit using the ISIM and a Verilog test fixture

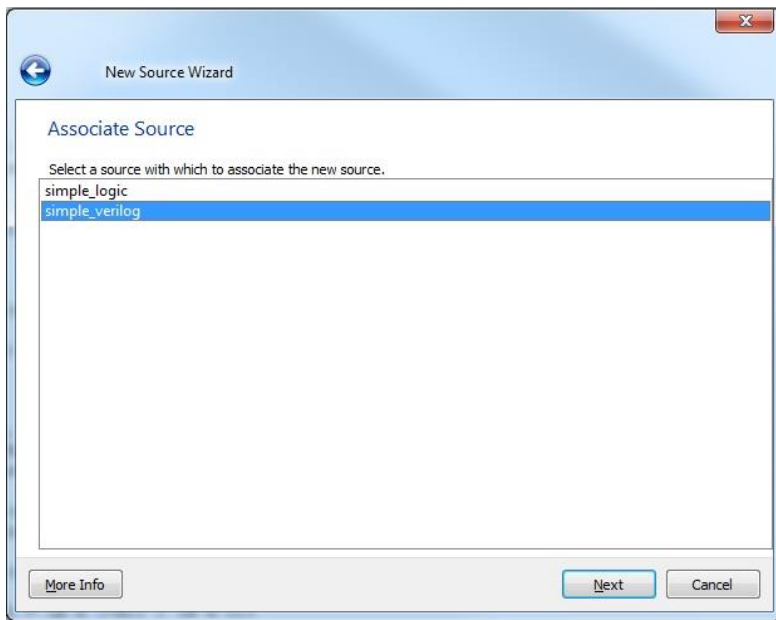
Now that you have a saved schematic, you need to simulate its behavior. The simulator we'll use is the **ISE** built in simulator, which is essentially a Verilog simulator. The schematic that you just drew will be saved as a piece of Verilog behind your back if you choose to simulate it. In order to simulate the circuit you need:

Testbench: is a file that becomes a top module to your design and applies inputs to your circuit, and potentially checks that the outputs are correct. This will be another Verilog file written slightly differently than circuit implementation. The testbench will instantiate one copy of your circuit, and call it **UUT** for "Unit Under Test". You will then write the Verilog statements that set the inputs to your circuit (the UUT), and looks at the outputs produced by your circuit. You need to know only very basic Verilog syntax to do this.

1. First you need to ensure that the ISE more is changed to "**Simulation**" from implementation. Go to the top left pane and change the "**View**" field to simulation. The design window will then change slightly with different options. Referring back to the same step in creating a "**New Source**" create a "**Verilog Test Fixture**" to create a Verilog file that will contain the test code.



2. Click NEXT and choose which design you want to associate the test bench with. This is very important as you will have multiple modules or schematics in the future and you need to be sure which design will be going under test using the test bench. In this case I will just choose the "simple_verilog" module to be tested. The procedure for testing the schematic version is exactly the same, you just have to choose the appropriate source to be associated with the test bench.



3. Click NEXT and after observing the summary click FINISH. Now you'll get a new piece of Verilog code generated for you. This Verilog code instantiates the “**simple_verilog**” module as the **UUT**, and includes some other stuff related to how the **UUT** is connected to the testbench. It looks like this:

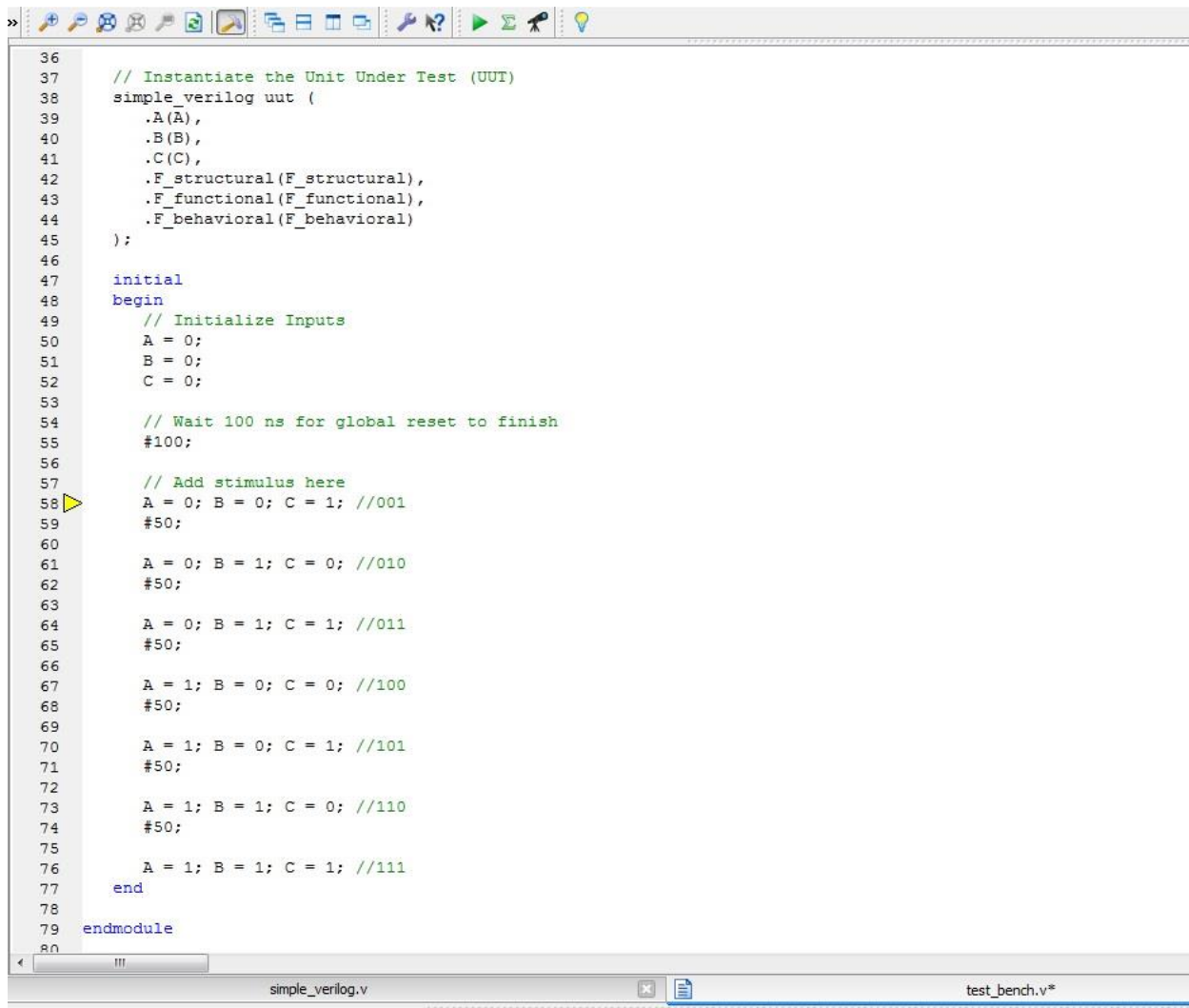
```

19 // Revision:
20 // Revision 0.01 - File Created
21 // Additional Comments:
22 //
23 ///////////////////////////////////////////////////////////////////
24
25 module test_bench;
26
27     // Inputs
28     reg A;
29     reg B;
30     reg C;
31
32     // Outputs
33     wire F_structural;
34     wire F_functional;
35     wire F_behavioral;
36
37     // Instantiate the Unit Under Test (UUT)
38     simple_verilog uut (
39         .A(A),
40         .B(B),
41         .C(C), |
42         .F_structural(F_structural),
43         .F_functional(F_functional),
44         .F_behavioral(F_behavioral)
45     );
46
47     initial begin
48         // Initialize Inputs
49         A = 0;
50         B = 0;
51         C = 0;
52
53         // Wait 100 ns for global reset to finish
54         #100;
55
56         // Add stimulus here
57
58     end
59 endmodule
60
61
62

```

4. You can now write your test bench code as an **initial** block right before the **endmodule**. Basically you set the values of your inputs, and tell the simulator how long to wait between each change on the inputs. The results will eventually be plotted on a waveform for you. Verilog syntax for setting a variable is very simple, and the **#50** notation just means for the simulation to wait for 50 ticks of the simulation clock before moving on to the next statement. A very simple test bench for this circuit looks like the following. I added the lines between **initial** and **end** to drive the inputs with different values so that we can see what the circuit result is. Note that each statement in Verilog ends with a semicolon, and you can put multiple statements on a line if you like.

Later in upcoming labs we will explore different options that we can write the sequencing of the inputs to prevent having to hardcode lots of input changes, you can see that if we had two more inputs then we had to write out 32 different possibilities for 5 inputs. We will see how using a “For” loop in the test bench will make things a lot and shorter.

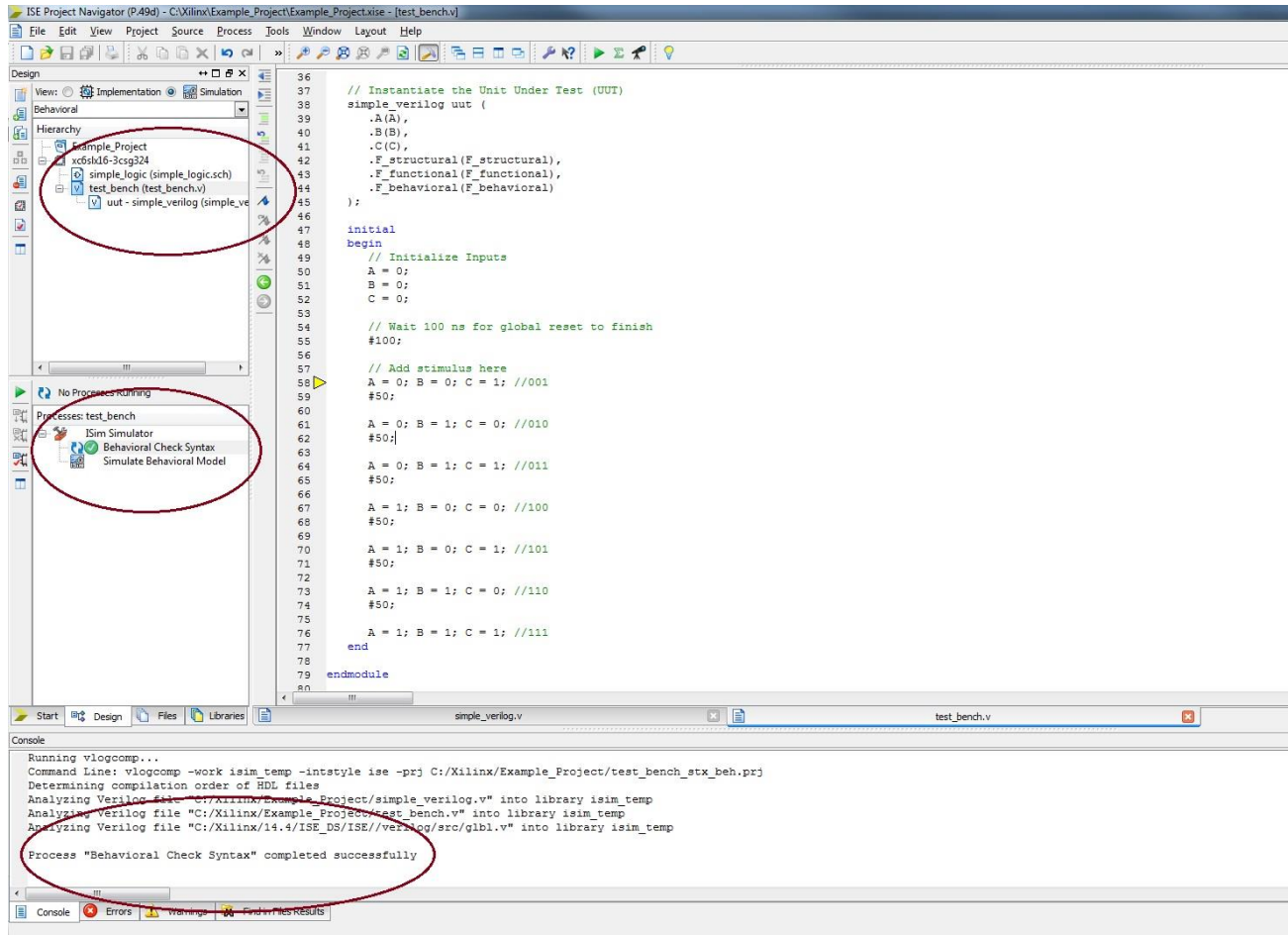


```
36
37 // Instantiate the Unit Under Test (UUT)
38 simple_verilog uut (
39     .A(A),
40     .B(B),
41     .C(C),
42     .F_structural(F_structural),
43     .F_functional(F_functional),
44     .F_behavioral(F_behavioral)
45 );
46
47 initial
48 begin
49     // Initialize Inputs
50     A = 0;
51     B = 0;
52     C = 0;
53
54     // Wait 100 ns for global reset to finish
55     #100;
56
57     // Add stimulus here
58     A = 0; B = 0; C = 1; //001
59     #50;
60
61     A = 0; B = 1; C = 0; //010
62     #50;
63
64     A = 0; B = 1; C = 1; //011
65     #50;
66
67     A = 1; B = 0; C = 0; //100
68     #50;
69
70     A = 1; B = 0; C = 1; //101
71     #50;
72
73     A = 1; B = 1; C = 0; //110
74     #50;
75
76     A = 1; B = 1; C = 1; //111
77 end
78
79 endmodule
80
```

5. We usually want to test all possible inputs to be able to draw a better conclusion on whether the circuit is functioning correctly. After you’re satisfied with the input setting of your test bench make sure to save. Always observe the console window to look out for errors after saving.

Now you are ready to simulate your Verilog circuit. Observe that the test bench Verilog file is now the top module to your “simple_verilog” module in the simulation design

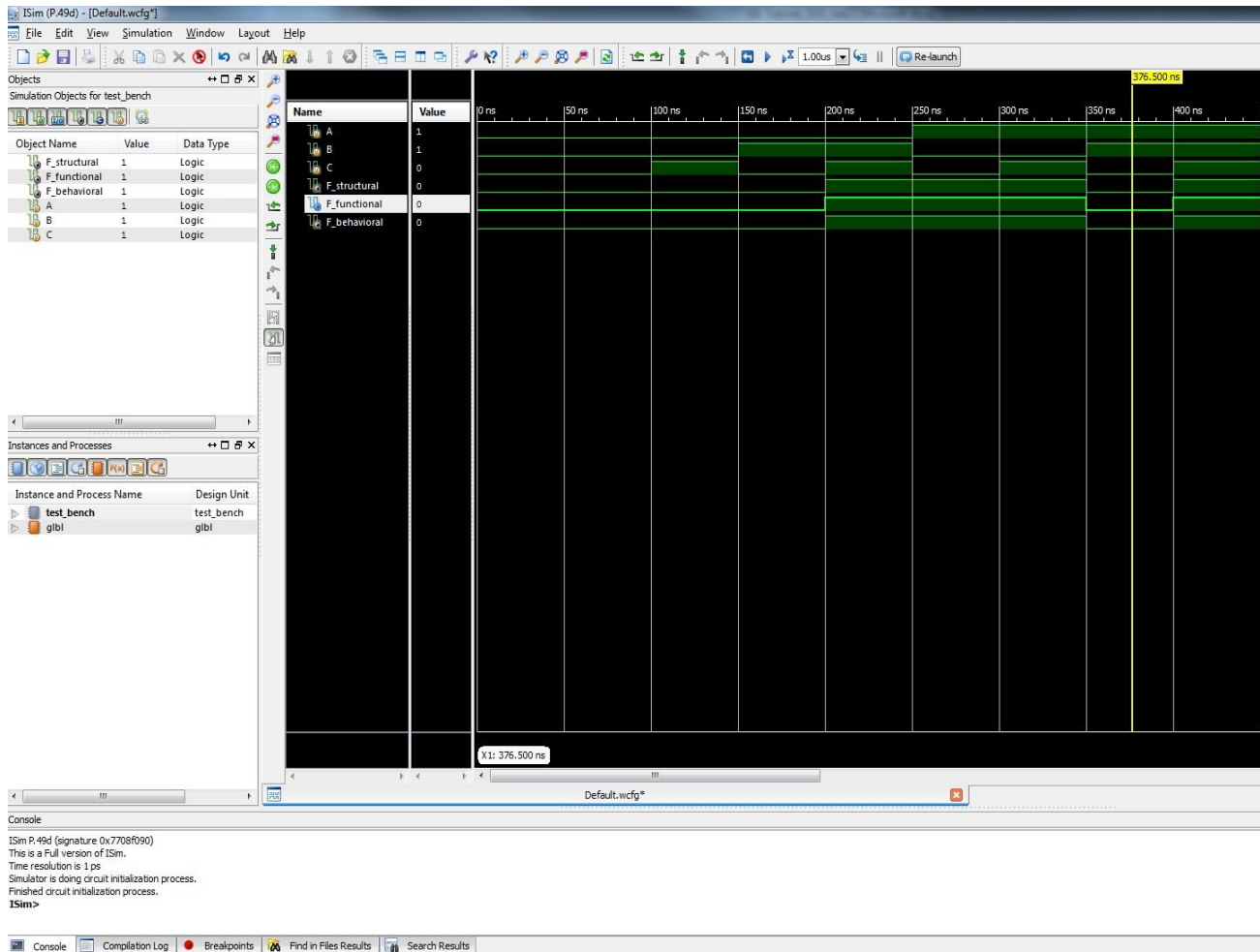
view. It is very important to have the test bench file selected for simulation or things will go wrong. After selecting and highlighting the test bench file in the design windows, you can check your test bench for syntactic errors by clicking the **“Behavioral Check Syntax”** in the processes windows right below. If your code is correct syntactically then you’ll get a green check mark. This is not crucial as if you run the simulation without it then the syntax will be automatically checked by the ISim and will not run if there are errors in your test bench. Now you can double-click the **“Simulate Behavioral Model”** to see the waveform generated by the ISim.



7. Double-clicking the **Simulate Behavioral Model** will fire up the simulator on your testbench file. Because your testbench includes an instance of your schematic (the **UUT**) and some commands to drive signals into your schematic, this will result in simulating your schematic. The output will be displayed as waveforms as shown. Note that the simulator is by default set up to simulate for 1000ns, so all the stuff I did is bunched up at the beginning of the simulation (the first 0ns). I had to zoom out a little to see this view. The values reported for A, B, S, and F are the values seen at the blue bar. You can pick up (with the mouse) and move the blue bar to see the values at different points in the simulation.

By looking at the waveform we can see that all three different forms of expressing the function in Verilog (structural, functional, and behavioral) are all holding the same behavior throughout the simulation. You can click on the waveform in different places (the yellow line is where in the range of time in the waveform it was clicked) and you can see values quickly for all I/O in the **“Name”** and **“Value”** sections to the left of the waveform.

The simulation is now done. You can always go back to the test bench and make a more complicated set of tests and re-simulate the circuit.



It is good to observe the other capabilities of your simulation windows. The console is where all of the simulator messages will be printed. Look for errors or warning about your design.

The left panes will allow you to dig into your design hierarchy to grab signals that are deep in the design and may not be set in the top module, recall that the test bench is only created for one module, so if you have a multiple level hierarchy then you need to use these windows to navigate to the desired signal and drag-and-drop into the simulation windows.

Sometimes we want the values of the test bench results for the Verilog module's output to be printed into the console of the simulator windows. We can then add the Verilog print statement called **\$display** in between our test bench code. This will allow us to monitor signals in large designs and long waveforms with many signals being looked at. We can also format the printed statement to what we want it to show on the console for better readability of the results. This is called a **Self-Checking** testbench.

Below are some \$display statements added to the current test bench to show the effects it will have. Never the less, you should always put a \$display statement at the beginning and the end of your test code to indicate the starting and actual finishing of the simulations.

```

47  initial
48  begin
49      $display("The simple_verilog module simulation is starting ...");
50
51      // Initialize Inputs
52      A = 0;
53      B = 0;
54      C = 0;
55
56      // Wait 100 ns for global reset to finish
57      #100;
58      $display("ABC = %b%b%b, F_structural = %b, F_functional = %b, F_behavioral = %b", A, B, C, F_structural, F_functional, F_behavioral);
59
60      // Add stimulus here
61      A = 0; B = 0; C = 1; //001
62      #50;
63      $display("ABC = %b%b%b, F_structural = %b, F_functional = %b, F_behavioral = %b", A, B, C, F_structural, F_functional, F_behavioral);
64
65      A = 0; B = 1; C = 0; //010
66      #50;
67      $display("ABC = %b%b%b, F_structural = %b, F_functional = %b, F_behavioral = %b", A, B, C, F_structural, F_functional, F_behavioral);
68      |
69      A = 0; B = 1; C = 1; //011
70      #50;
71      $display("ABC = %b%b%b, F_structural = %b, F_functional = %b, F_behavioral = %b", A, B, C, F_structural, F_functional, F_behavioral);
72
73      A = 1; B = 0; C = 0; //100
74      #50;
75      $display("ABC = %b%b%b, F_structural = %b, F_functional = %b, F_behavioral = %b", A, B, C, F_structural, F_functional, F_behavioral);
76
77      A = 1; B = 0; C = 1; //101
78      #50;
79      $display("ABC = %b%b%b, F_structural = %b, F_functional = %b, F_behavioral = %b", A, B, C, F_structural, F_functional, F_behavioral);
80
81      A = 1; B = 1; C = 0; //110
82      #50;
83      $display("ABC = %b%b%b, F_structural = %b, F_functional = %b, F_behavioral = %b", A, B, C, F_structural, F_functional, F_behavioral);
84
85      A = 1; B = 1; C = 1; //111
86      #50;
87      $display("ABC = %b%b%b, F_structural = %b, F_functional = %b, F_behavioral = %b", A, B, C, F_structural, F_functional, F_behavioral);
88
89      $display("The simple_verilog module simulation has ended ...");
90  end
91

```

And the resulting console output for the above test bench code is the following:

```

This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
The simple_verilog module simulation is starting ...
Finished circuit initialization process.
ABC = 000, F_structural = 0, F_functional = 0, F_behavioral = 0
ABC = 001, F_structural = 0, F_functional = 0, F_behavioral = 0
ABC = 010, F_structural = 0, F_functional = 0, F_behavioral = 0
ABC = 011, F_structural = 1, F_functional = 1, F_behavioral = 1
ABC = 100, F_structural = 1, F_functional = 1, F_behavioral = 1
ABC = 101, F_structural = 1, F_functional = 1, F_behavioral = 1
ABC = 110, F_structural = 0, F_functional = 0, F_behavioral = 0
ABC = 111, F_structural = 1, F_functional = 1, F_behavioral = 1
The simple_verilog module simulation has ended ...
ISim> |

```


We can also get a bit fancier and use loops to assign values to the circuit inputs. With larger bus widths we will need to use them to compose a more elegant and comprehensive test fixture.

```

43 integer i,j,k;
44
45 initial
46 begin
47     $display("Simulations Begins...");
48
49     // Initialize Inputs
50     A = 0;
51     B = 0;
52     C = 0;
53
54     // Wait 100 ns for global reset to finish
55     #100;
56
57     // Add stimulus here
58     for (i = 0; i < 2; i = i + 1)
59         for(j = 0; j < 2; j = j + 1)
60             for(k = 0; k < 2; k = k + 1)
61                 begin
62                     C = i;
63                     B = j;
64                     A = k;
65
66                     #10;
67
68                     $display("A = %b, B = %b, C = %b, F = %b", A, B, C, F);
69                 end
70
71     $display("... Simulations Ends");
72 end
73
74 endmodule

```

Please note that the loop limits match the values each input can take, for a single input $2^1=2$ here for each input. We can also use concatenation {} to reduce the loops even further to only a single loop for all inputs. This way the loop limit is $2^3=8$ since the A, B, and C combined width is three. This way all possibilities will be assigned to the inputs and nothing missed.

```

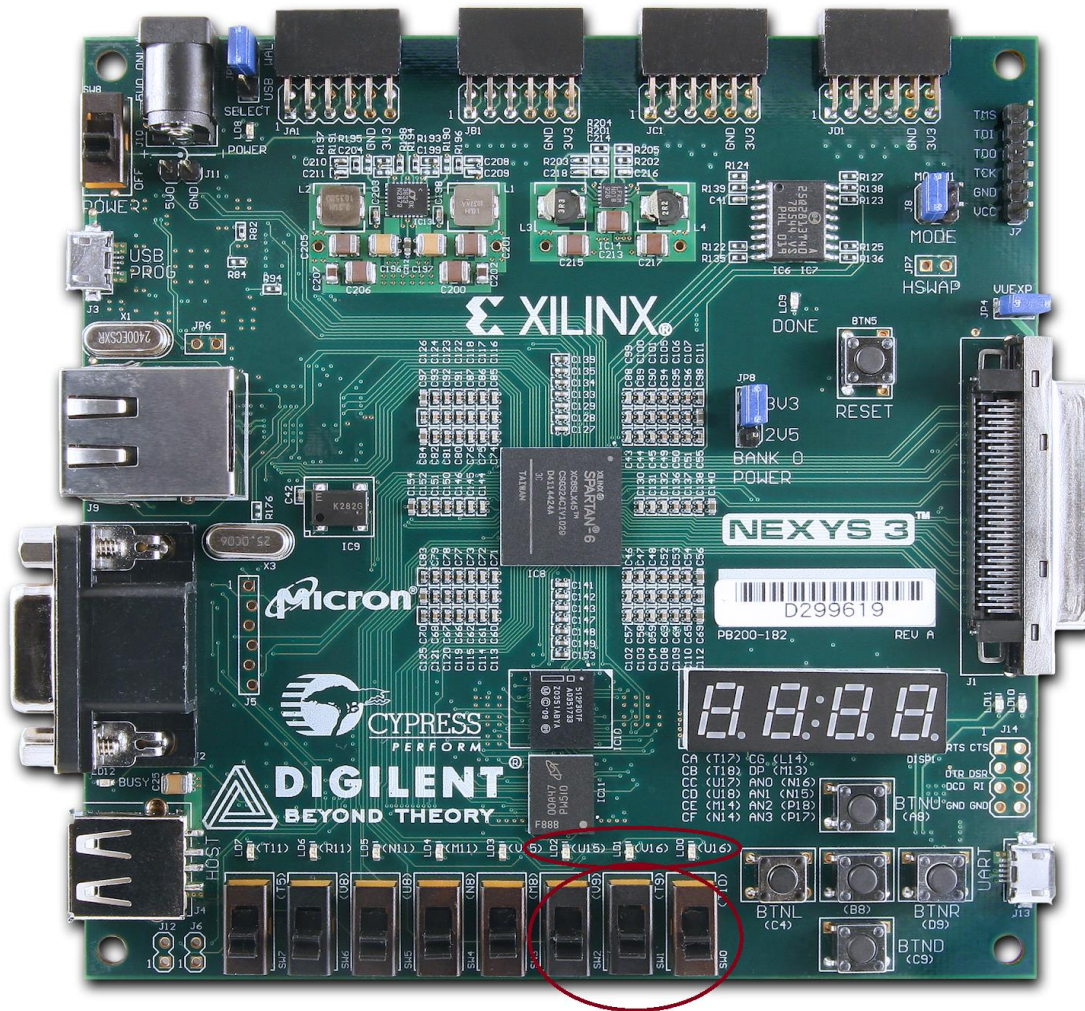
43 integer i;
44
45 initial
46 begin
47     $display("Simulations Begins...");
48
49     // Initialize Inputs
50     A = 0;
51     B = 0;
52     C = 0;
53
54     // Wait 100 ns for global reset to finish
55     #100;
56
57     // Add stimulus here
58
59     for (i = 0; i < 8; i = i + 1)
60         begin
61             {C, B, A} = i;
62
63             #10;
64
65             $display("A = %b, B = %b, I = %b, F = %b", A, B, I, F);
66         end
67
68     $display("... Simulations Ends");
69 end
70
71 endmodule

```

Part V: Constraint, Synthesize, Implement, Generate Bitstream, and Program the Nexys 3 FPGA board

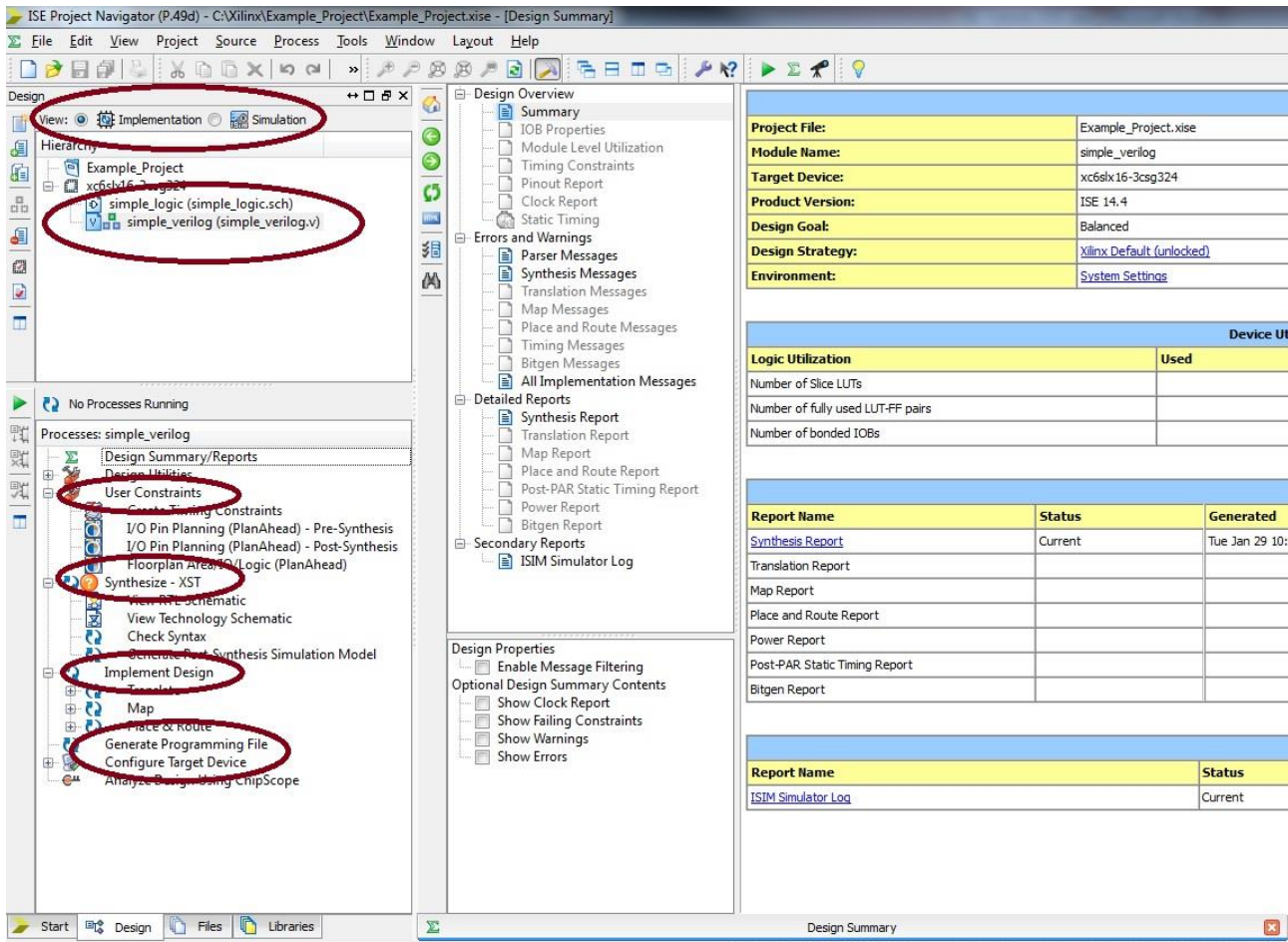
Overview: Now that you have a correctly simulating Verilog module, you will have the ISE tools synthesize your Verilog or schematic to something that can be mapped to the Xilinx FPGA. That is, the Verilog code will be converted by ISE to some gates that are on the FPGA. To be even more specific, ISE will convert the Verilog description into a set of configuration bits that are used to program the Xilinx part to behave just like the Verilog code. Those configuration bits are in a **.bit** file and are downloaded to the Xilinx part in this next section of the tutorial.

For the purposes of this tutorial I will choose to put the Verilog version on the Nexys board, but the process is exactly the same for any other design form i.e. schematics. I will use the first three toggle switches on the board for **A, B, and C**, and the first three LEDs for **F**.

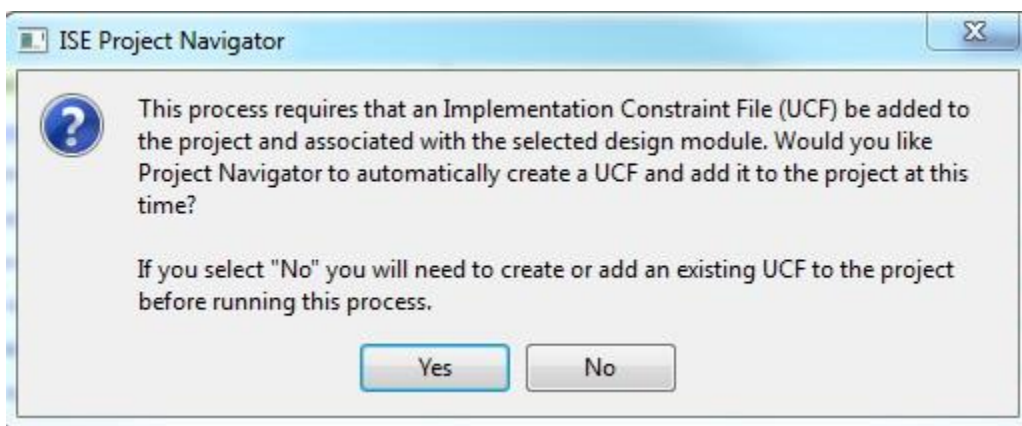


UCF (User Constraints File): Because we're headed towards putting this on the Xilinx FPGA on the Nexys 3 board, we need to set some constraints. In particular, we need to tell ISE which pins on the Xilinx chip we want **A, B, C**, and **F** assigned to so that we can access those from switches and LEDs on the Nexys 3 board. For that we need a "User Constraints File".

1. First, you need to ensure that you're in the Implementation view and that the module you're trying to implement on the board is set as the top module in the top left pane in order to get the options to synthesize, implement, and generate the design. Now if you look at the bottom left pane you can see a number of processes you can run on this Verilog top module.



2. **Constraint:** Now it starts with creating a floor plan by setting the UCF file. To do this take a look at the **User Constraints** drop down option in the bottom left pane. We can set the pins in two different ways. Double click on the **I/O Pin Planning (PlanAhead) – Pre-Synthesis** since we want to set our pins before the synthesis process so they are included in it. This should bring up a message box for adding a new UCF file to your design, so click yes and this will kick start another Xilinx tool called **Plan Ahead**. This program allows you to set all constraints on all I/O pins in the design. Please follow these steps carefully.



3. **Pin Assignment:** We need to edit the initial UCF for the details of connections between the ports and pins in the design, so as **Plan Ahead** opens (it looks a lot like ISE), take a look at the horizontal bottom pane. After expanding the **Scalar Ports** drop-down you should see all of your I/O pins. The first thing to do is to set all of the pins to **LVC MOS33** right away, if you don't do this, things might not work correctly on the board. Click on the name of the pin (A, B, C, ...) and then look at the properties table line listed in front of it. The only two properties that we want to change are the **I/O Std** and **Site**. Remember, the I/O standard is always and should be set to LVC MOS33 (3.3 V Low-Voltage-Complimentary-Metal-Oxide-Semiconductor) for the Nexys 3 board. The Site is the Pin# (FPGA board Pin Identifier). You can find all sorts of the information including the module schematics and the Pin #s for all of the Nexys 3 Peripherals by studying the Nexys 3 manual found at http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf.

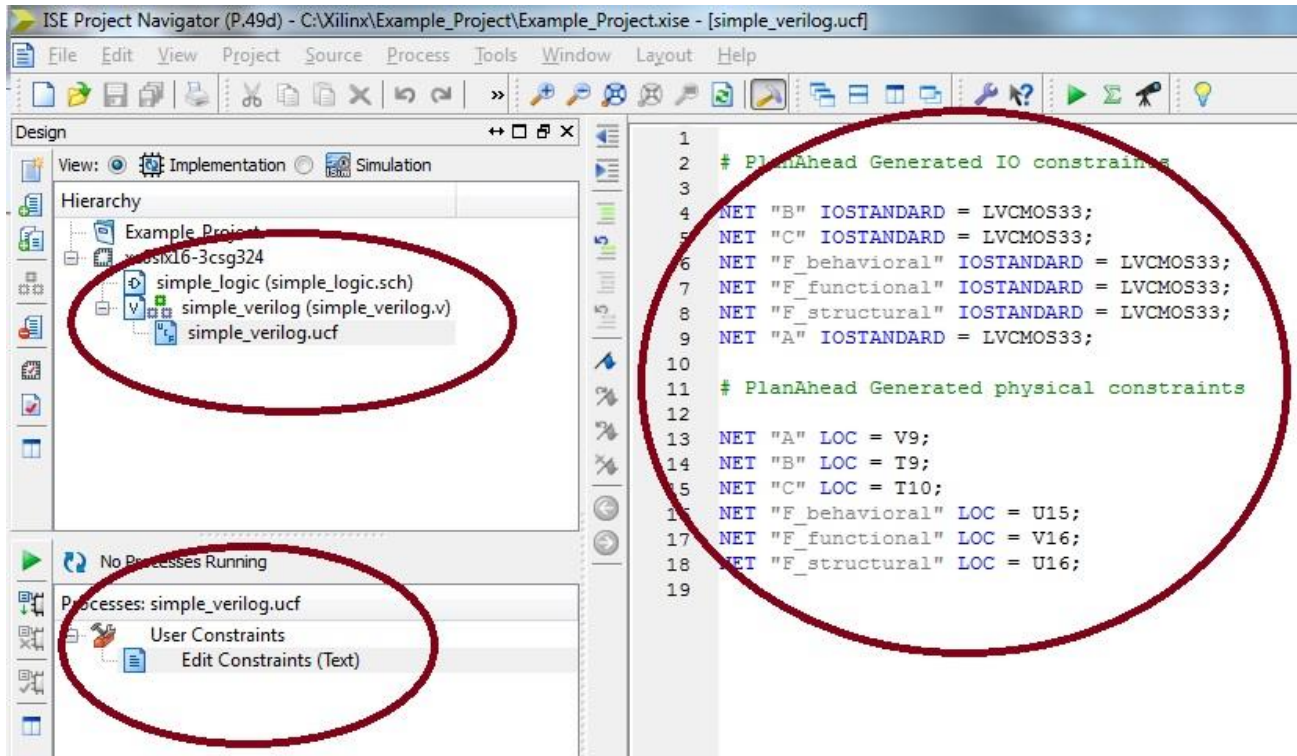
In order to change a property for any port, click on the rectangular space in the correct column and the correct row for the respective port. This click causes a drop-down menu to appear and then you can select the right choice. You can either type or select the correct choice. If there is a choice already selected then clicking on the text will allow you to change it, the hit enter. I looked up the three switches and the three LEDs, you can also look closely at the actual switch or LED on the board and you will see an identifier in parentheses (i.e. T10 for SW0 referring to Pin #T10 connecting to Switch 0, the first toggle switch from the right). Set the correct identifier to the correct port in your design in Plan Ahead as follows:

The screenshot shows the PlanAhead software interface. The 'I/O Ports' window is open, displaying a table of I/O ports. The 'I/O Std' column is highlighted in blue, and a dropdown menu is open for the 'F_structural' port, showing 'LVC MOS33' selected. Red circles highlight the 'Name' column, the 'Site' column, and the 'I/O Std' column.

| Name | Direction | Neg Diff Pair | Site | Fixed | Bank | I/O Std | Vcco | Vref | Drive Stre... | Slew Type | Pull Type | Off-Chip T... | IN_TERM | OUT_TERM |
|--------------|-----------|---------------|------|-------------------------------------|------|--------------|-------|------|---------------|-----------|-----------|---------------|---------|----------|
| A | Input | | V9 | <input checked="" type="checkbox"/> | | 2 LVC MOS33* | | | | | NONE | NONE | NONE | |
| B | Input | | T9 | <input checked="" type="checkbox"/> | | 2 LVC MOS33* | | | | | NONE | NONE | NONE | |
| C | Input | | T10 | <input checked="" type="checkbox"/> | | 2 LVC MOS33* | | | | | NONE | NONE | NONE | |
| F_behavioral | Output | | U15 | <input checked="" type="checkbox"/> | | 2 LVC MOS33* | 3.300 | | 12 SLOW | | NONE | FP_VTT_50 | NONE | |
| F_functional | Output | | V16 | <input checked="" type="checkbox"/> | | 2 LVC MOS33* | 3.300 | | 12 SLOW | | NONE | FP_VTT_50 | NONE | |
| F_structural | Output | | U16 | <input checked="" type="checkbox"/> | | 2 LVC MOS33* | 3.300 | | 12 SLOW | | NONE | FP_VTT_50 | NONE | |

Attention: It is important that you never use Plan Ahead to edit an existing UCF file as it will not override old values but concatenate the new values. Always go to the project folder and edit the .UCF file with a text editor. You can also do this in ISE but don't double click on the UCF file that is now in the hierarchy of your design in the upper left pane of ISE, single click it and then choose **Edit Constraints (Text)** from the User Constraints expander in the lower left pane. Make sure to hit save after you're done.

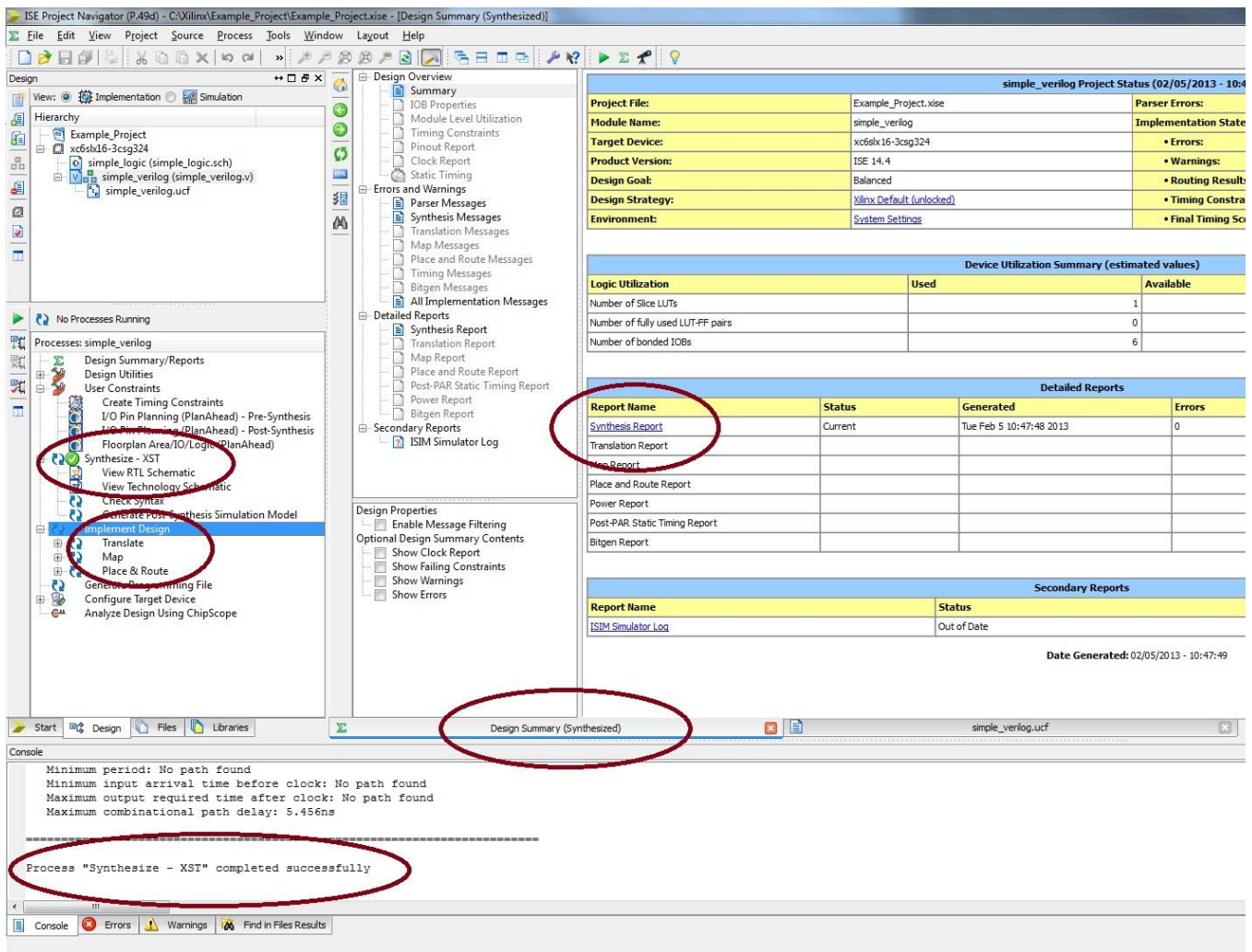
And your resulting UCF file should look something like the following:



After you're done changing every port to LVCMOS33 and putting the right Pin # for all the ports in the design, click the **SAVE** button from the top menu in the page and close down Plan Ahead.

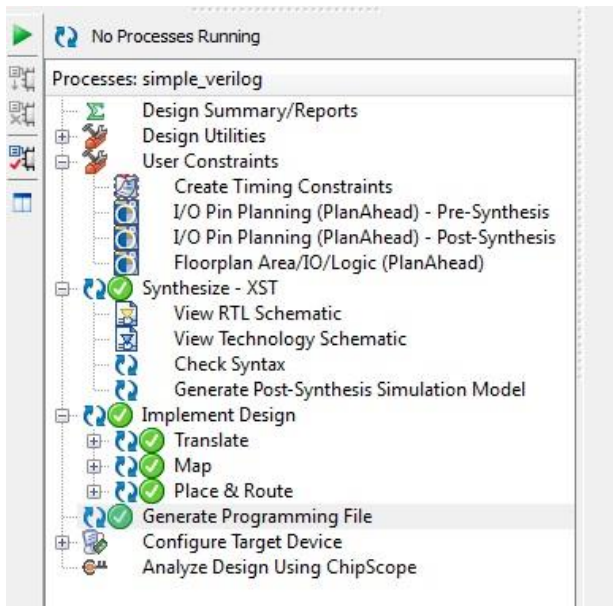
4. **Synthesize:** Now the design is ready for more processing and the next in line is to synthesize. This process will create a structural representation of the design (similar to compiling C code into assembly code). Do this by first ensuring that the top module is selected and highlighted in the top left pane of ISE and then simply double clicking **Synthesize – XST** in the lower left pane. After the process is done, you will either get a green check mark (everything is peachy), a yellow attention mark (there are warnings!), or a red x mark (there are errors!), and orange question mark just means “out of date”. In the case of errors you need to investigate them by looking at the Errors tab on the bottom pane and fix, and then rerun synthesis. In the case or warnings make sure to review them and validate they are safe, or in the case of green you're ready for the next step.

The synthesis process also creates a couple of more useful things that you should explore and study. One is the synthesis report full of information about timing, resource usage details, and etc. The other is generated RTL schematic; sometimes it is very useful to see what the XST made out of your Verilog description, or even your version of schematic. You can view these (highly recommended) by expanding **Synthesize – XST** for the schematic and the Design Summary page in the main pane for the report.



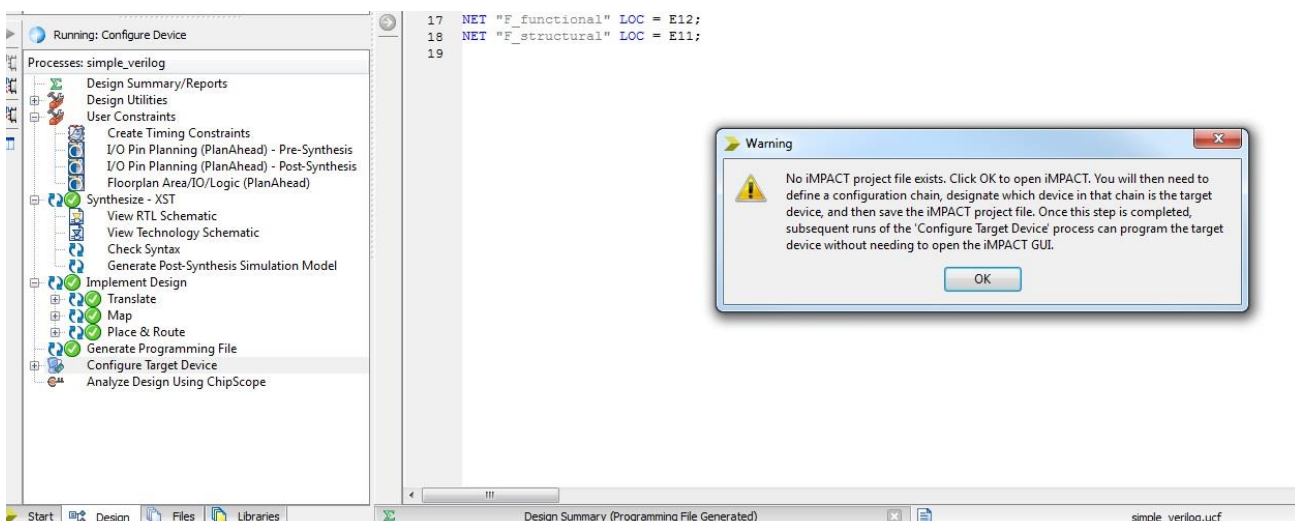
5. **Implement:** Next step is to define the hardware configuration. With your top module source file selected (simple_verilog.v in this case), double click the **Implement Design** process in the **Processes** tab. This will translate the design to something that can physically be mapped to the particular FPGA that's on our board (the xc6slx16-3csg324). You should see a green check mark if this step finishes without issues. If there are issues, you need to read them for clues about what went wrong and what you should look at to fix things. If you expand this **Implement Design** tab (which is not necessary) you will see that the **Implement Design** process actually consists of three parts:
 - a. **Translate:** Translate is the first step in the implementation process. The Translate process merges all of the input netlists and design constraint information and outputs a Xilinx NGD (Native Generic Database) file. The output NGD file can then be mapped to the targeted FPGA device.
 - b. **Map:** Mapping is the process of assigning a design's logic elements to the specific physical elements that actually implement logic functions in a device. The Map process creates an NCD (Native Circuit Description) file. The NCD file will be used by the PAR process.
 - c. **Place and Route (PAR):** PAR uses the NCD file created by the Map process to place and route your design. PAR outputs an NCD file that is used by the bitstream generator (BitGen) to create a (.bit) file. The Bit file (see the next step) is what's used to actually program the FPGA. In this part the actual transistor configuration and wire routing is decided.

6. **Generate:** In this step all information resulting from the previous steps are gathered and put into a Bit format that the USB programmer on the FPGA board (in our case Nexys3) understands. Simply double-click the **Generate Programming File** to generate a .bit file which will be used in the final step to program the board. Now the design is ready to be put on the board so we can physically see its functionality.

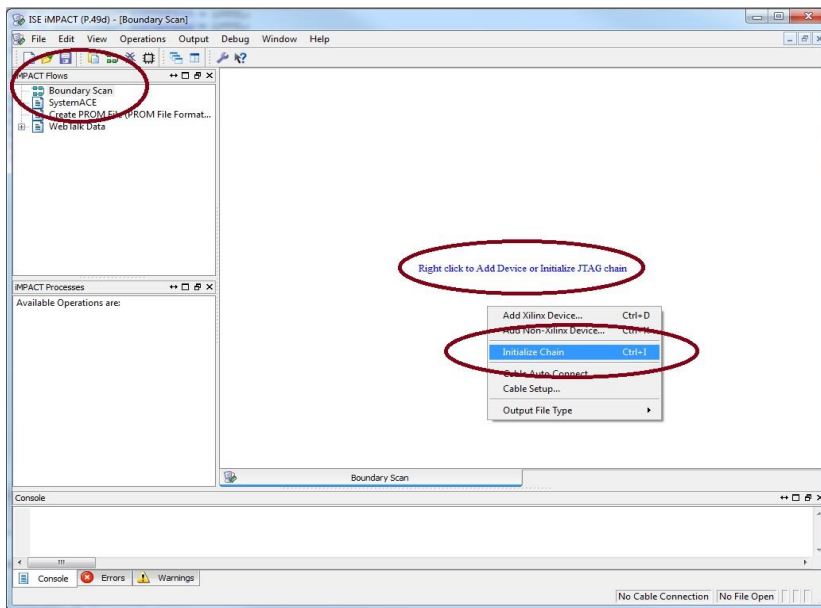


Attention: Before proceeding to the next step connect the board via the USB cable to the PC you're using, and turn on the power, the next step ensure proper connection to the board. Also note that you do not have to do steps 4, 5, and 6 individually. These processes are dependent on each other, if one needs a preceding process to be updated then it will automatically run that process before it runs itself. So you can just proceed to step 7 and watch ISE do everything.

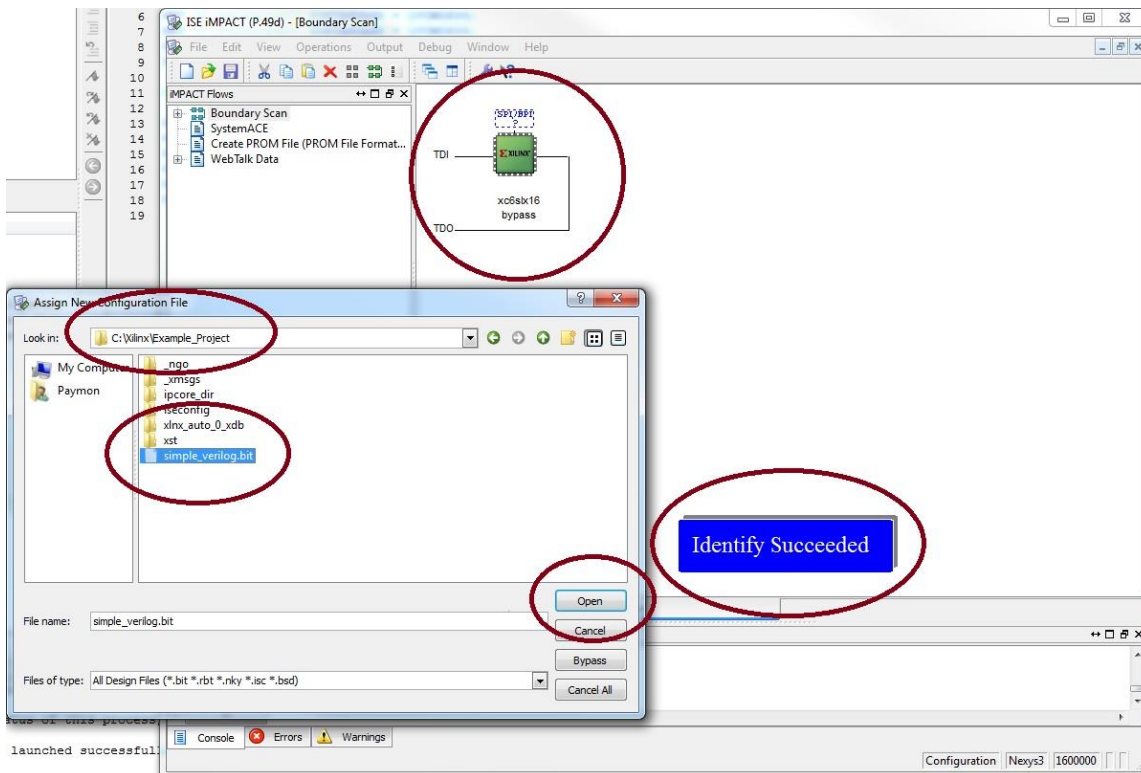
7. **Program:** You can start the programming process by double clicking **Configure Target Device** and ISE will launch yet another Xilinx tool called **iMPact**. A warning box appears complaining about "No iMPact project file exists...", so just click OK to launch iMPact as it will automatically read your existing project.



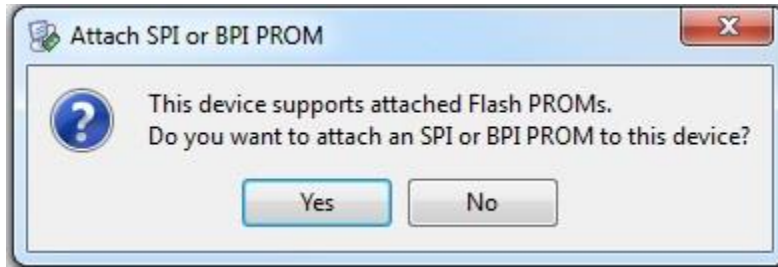
- a. In the ISE iMPact window which again looks a lot like ISE, double click **Boundary Scan** in the top left pane. In the boundary scan windows in the main pane, where it says “Right click to Add Device or Initialize chain” right-click in the middle of the page and select **Initialize Chain** or just press Ctrl + I. This will ensure there is a good connection to your board and it can communicate with it.



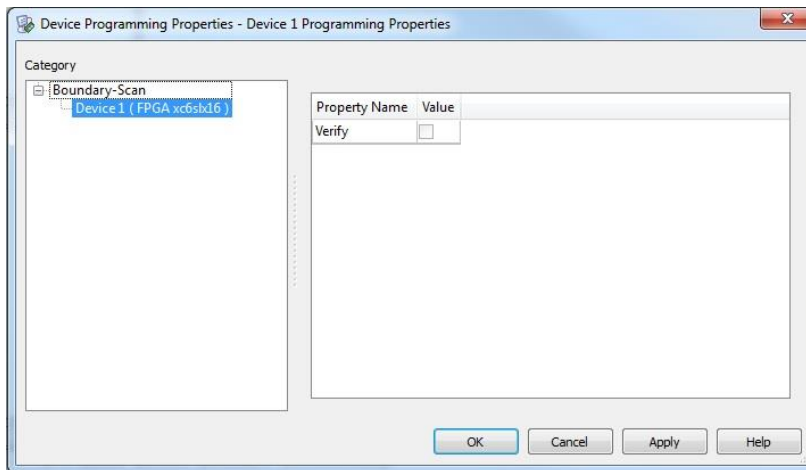
- b. After iMPact verifies that the cable is connected it will prompt you to load your **.bit** file that you generated in step 6. Note that this file selection window doesn't always default to your existing project so you may need to navigate to your project folder and locate the **.bit** file. This file is always named to your top module so in our case it is **simple_verilog.bit**. Double-click or select the bit file and click open, again make sure it is the right file.



- c. After the bit file is read in, iMPact prompts you to attach a PROM controller, just click NO to skip this step since we're not putting anything in the Flash memory.



- d. In the next dialog box you would be verifying which device on the board you're targeting but in our case we only have the FPGA chip to program, so click Ok and the preparation for programming the board is complete



- e. All that is left is to right click on the green chip icon with the Xilinx logo in the main pane and click **Program**. After the communication bar finishes, your design is programmed to the Nexys 3 FPGA board.

