

Xilinx Vivado VHDL Tutorial

This tutorial will provide instructions on how to:

- Create a Xilinx Vivado project
- Create a VHDL module
- Create a User Constraint File (UCF)
- Generate a Programming file for the Basys3

Creating a Xilinx Project

This tutorial will create a VHDL module for the logic equations:

$$X = A \cdot B + C \cdot D$$

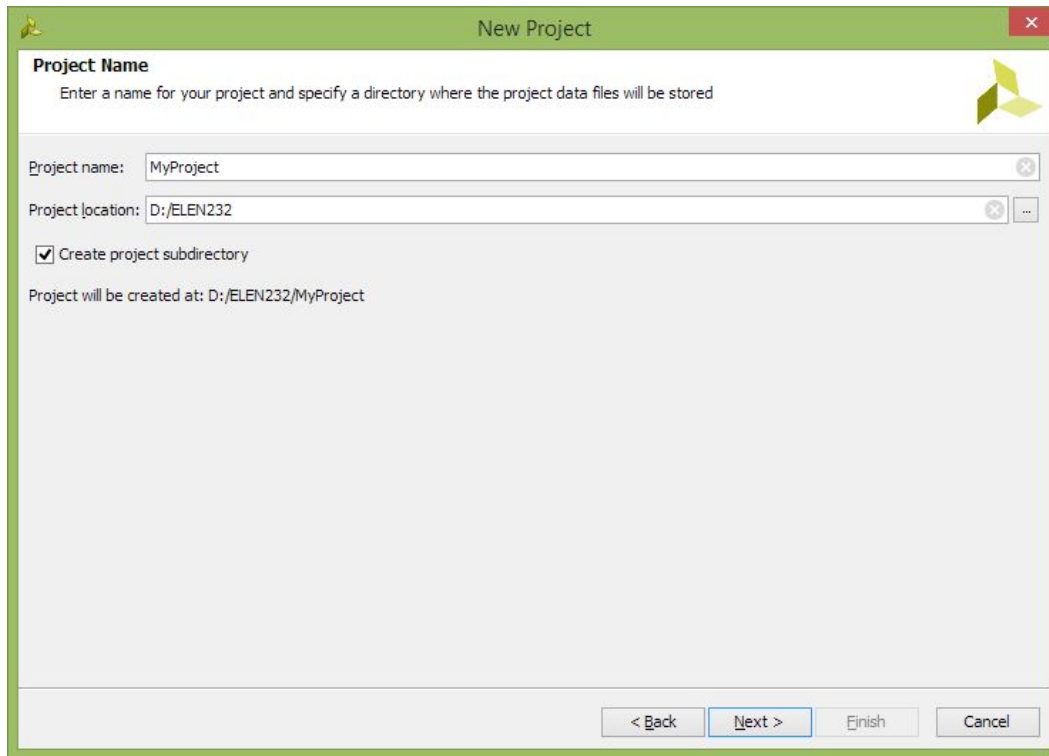
$$Y = (A + B) \cdot (C + D) \quad Z = A \oplus B \oplus C \oplus D$$

Start by running Xilinx Vivado

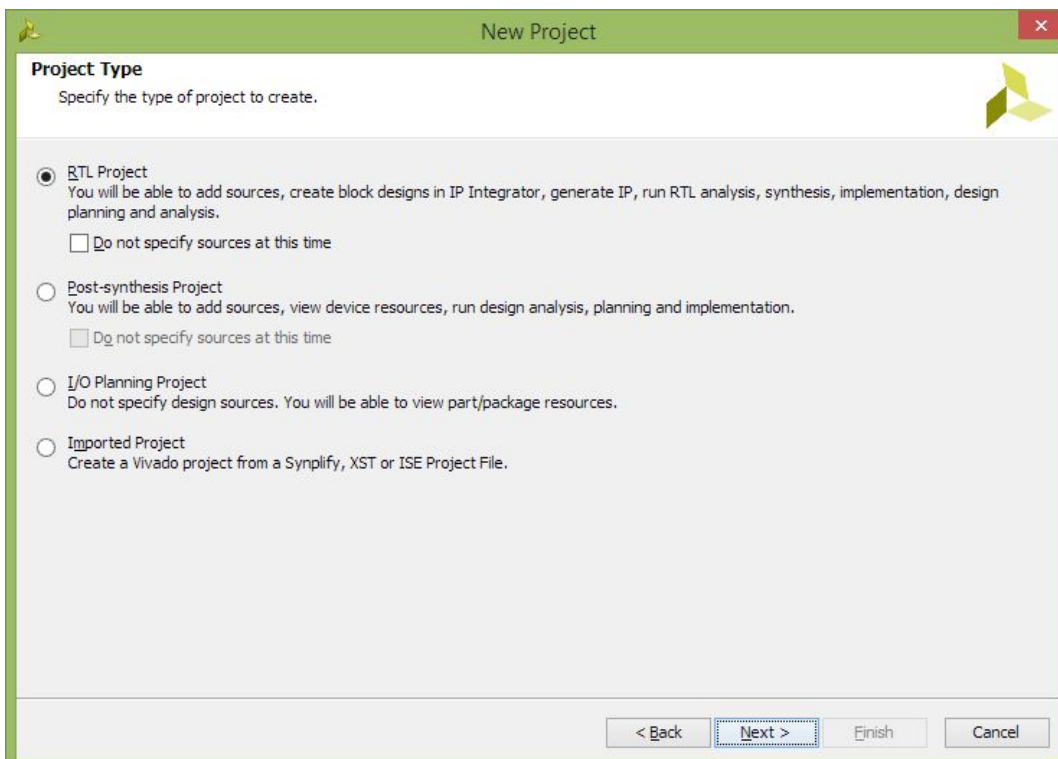


Under the Quick Start, select Create New Project

This brings up a series of dialog boxes to step you through the process of creating a project. Click Next to create a project name and save location. When picking a name and location do not use any spaces. (ie use "MyProject" or "My_Project" instead of "My Project") I would recommend selecting Create project subdirectory. That will automatically create a folder with your project name in your project location



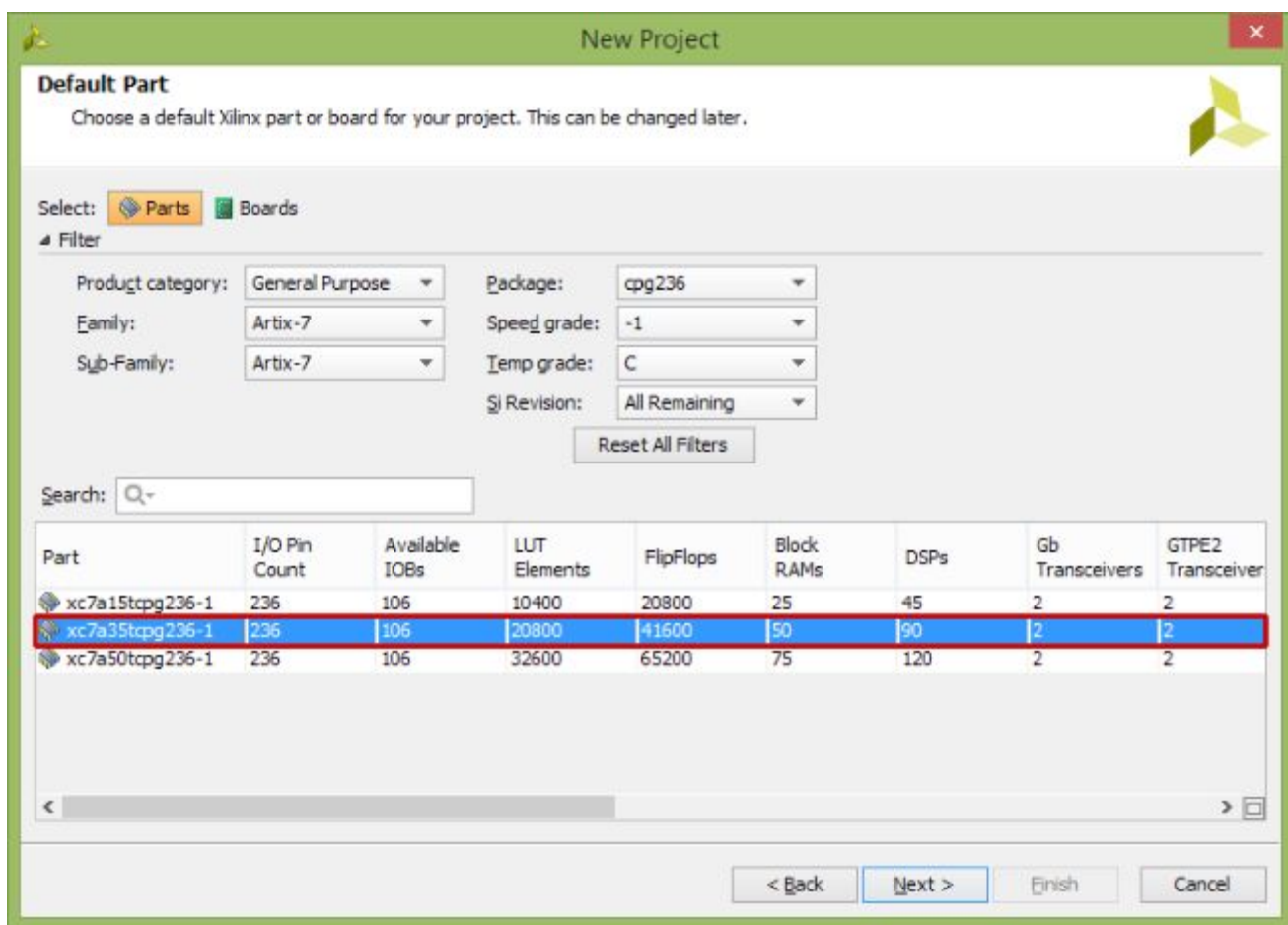
Clicking Next will bring up project type options. The projects we create for this course will be RTL Projects. Select RTL Project and click next.



The next 3 dialogue boxes are for adding Sources, Existing IP, and Constraints. We will add those source files later, so you can just click next for each of them.

The next dialogue has options to select the type of device we will be programming. For our project, we will be targeting the Atrix7 in the Basys3 board. It is important you select the following options on this dialog:

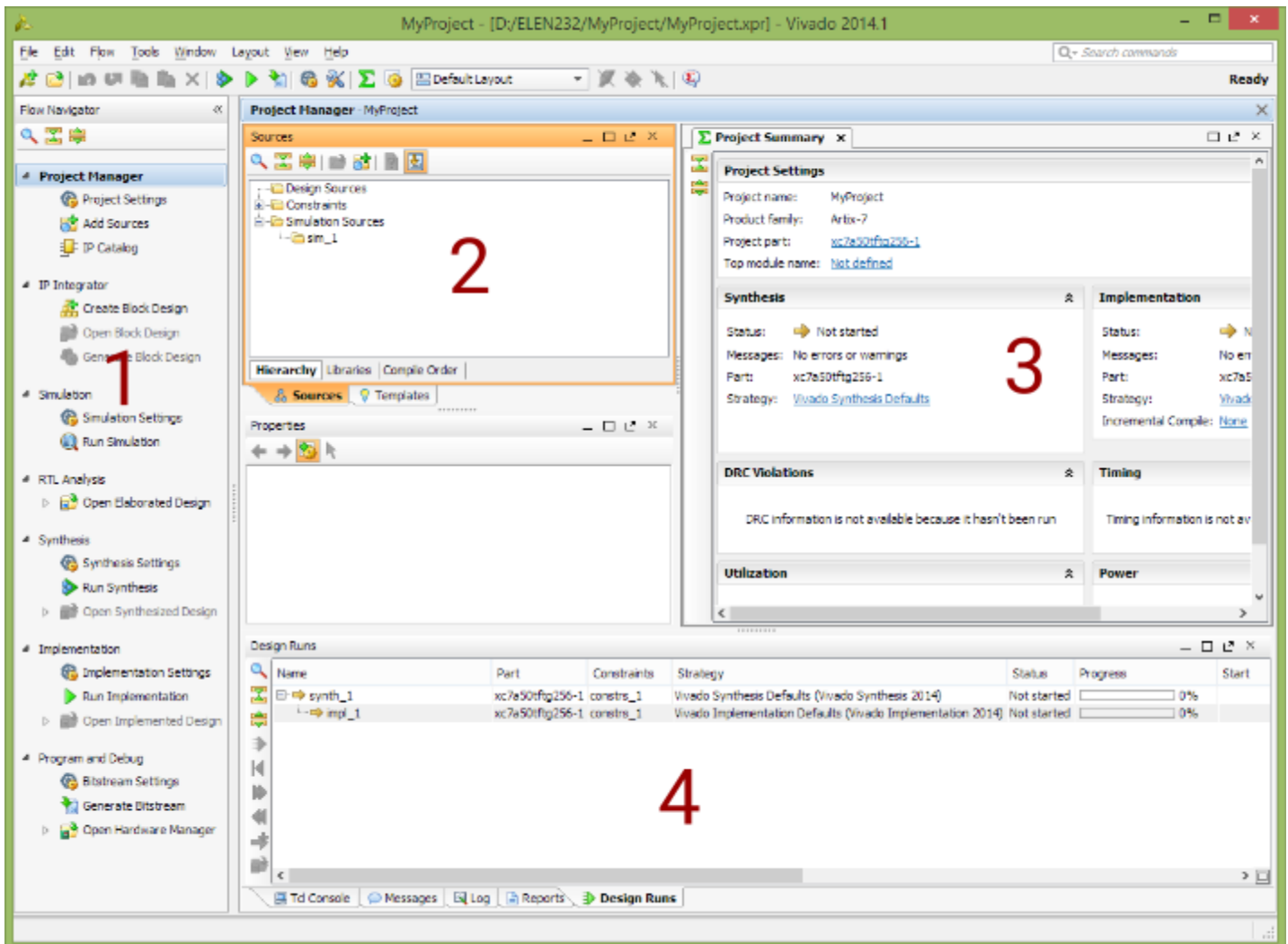
Family	Atrix-7
Package	cpg236
Speed Grade	-1
Part	xc7a35tcpg236-1



Clicking Next will give a Project Summary of the options selected. If any of the options need to be corrected, you can click Back and make any of the necessary changes. Once you verify all of the options are corrected, click Finish

Vivador Interface

The new project will be opened in Vivado. Only 1 project can be open at a time, so opening a different project will automatically close the current one. The Vivado interface is shown below.

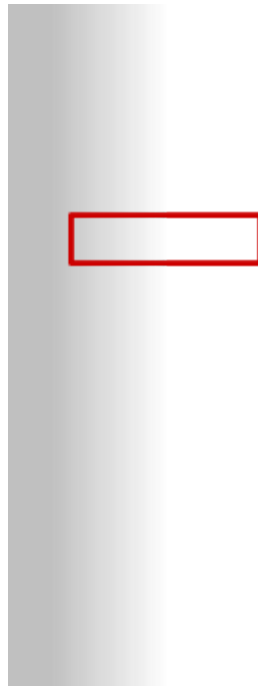


There are 4 main sections to the Project Navigator interface:

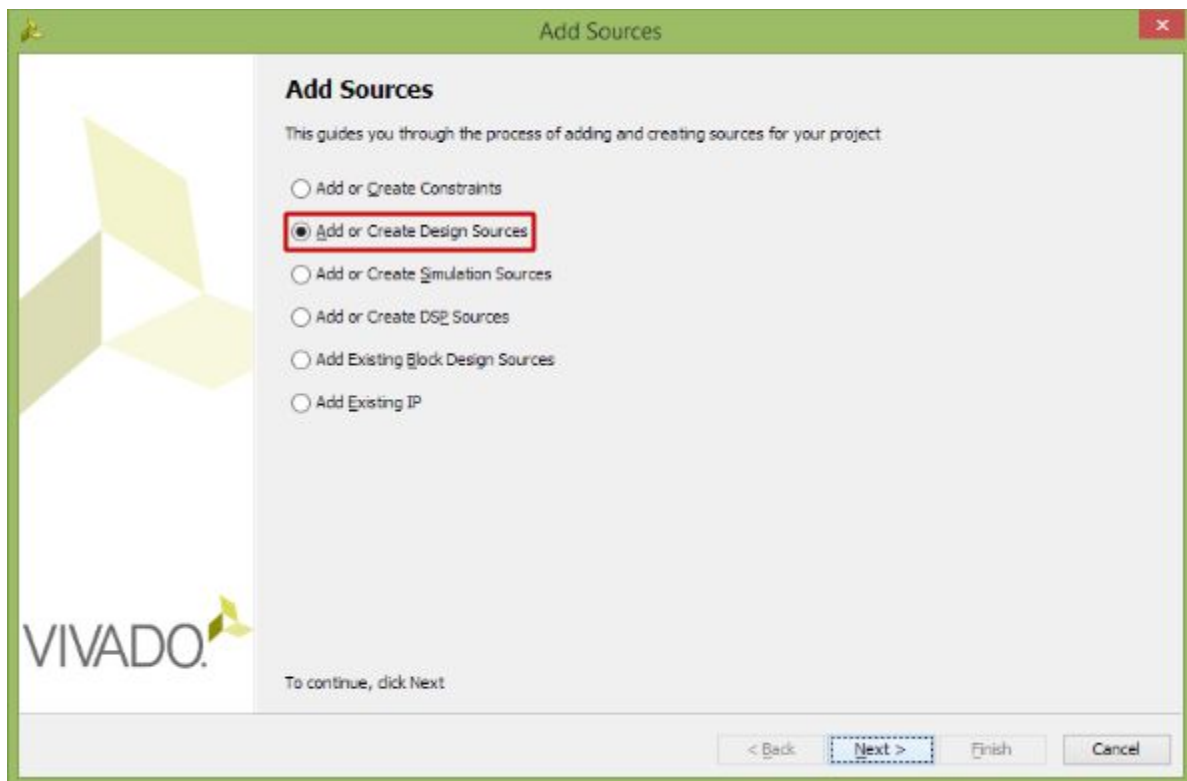
1. Flow Navigator: List of all available processes that can be run in the current project and their current state if running.
2. Sources Window: This is where all of the source files for the project are listed.
3. Editor Window: Display of the current file being edited. If no source file is selected, it defaults to a project summary page
4. Console Panel: Display of all status messages. This is an important window to look at for warning and error messages

Adding A Source File

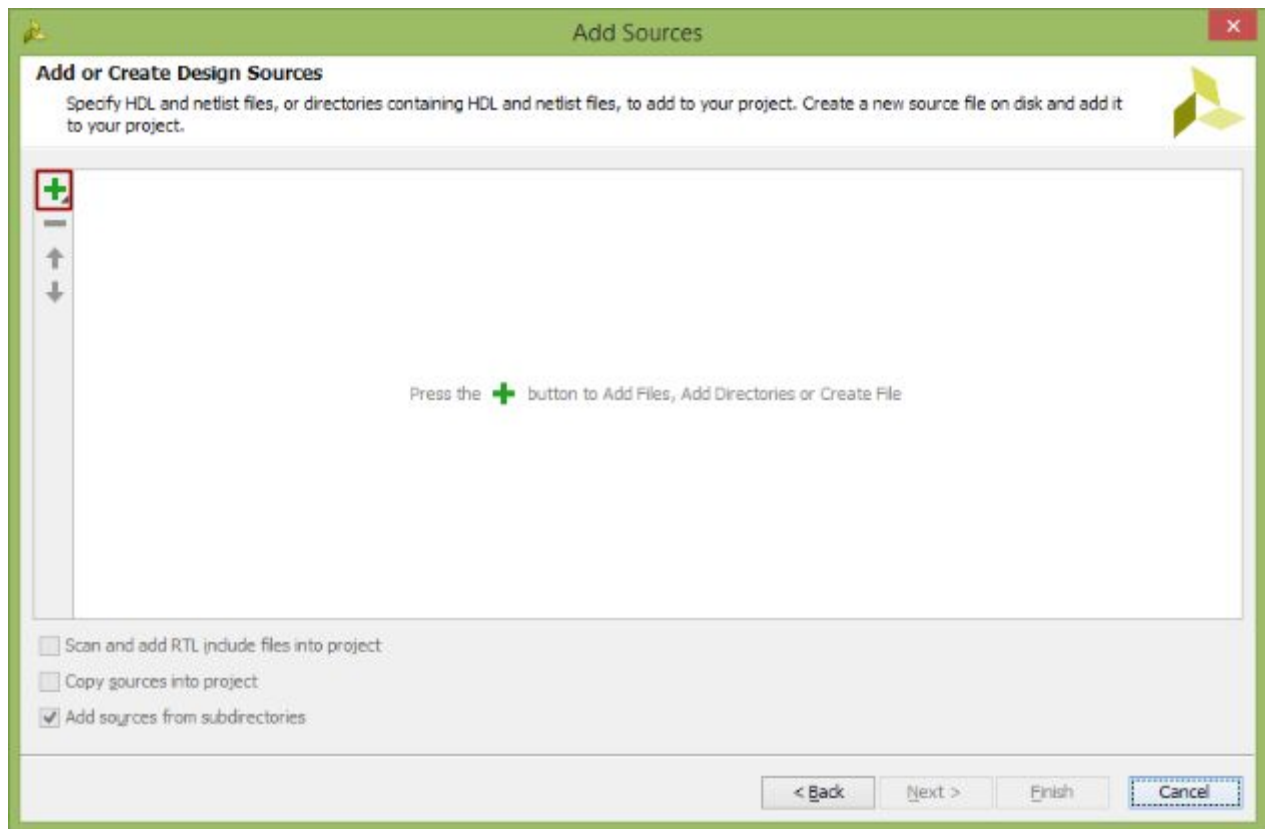
To add a source file to the project, select Add Sources in the Flow Navigator.



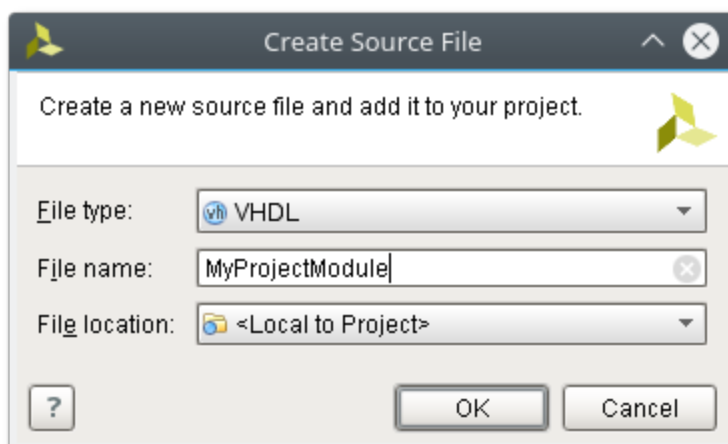
This will open an add source dialogue box. To add a VHDL Module, select Add or Create Design Sources.



Clicking next will bring up a similar Add or Create Design Sources that we skipped over when creating the project. A source file could have been created when first creating the project using the same process as detailed below.

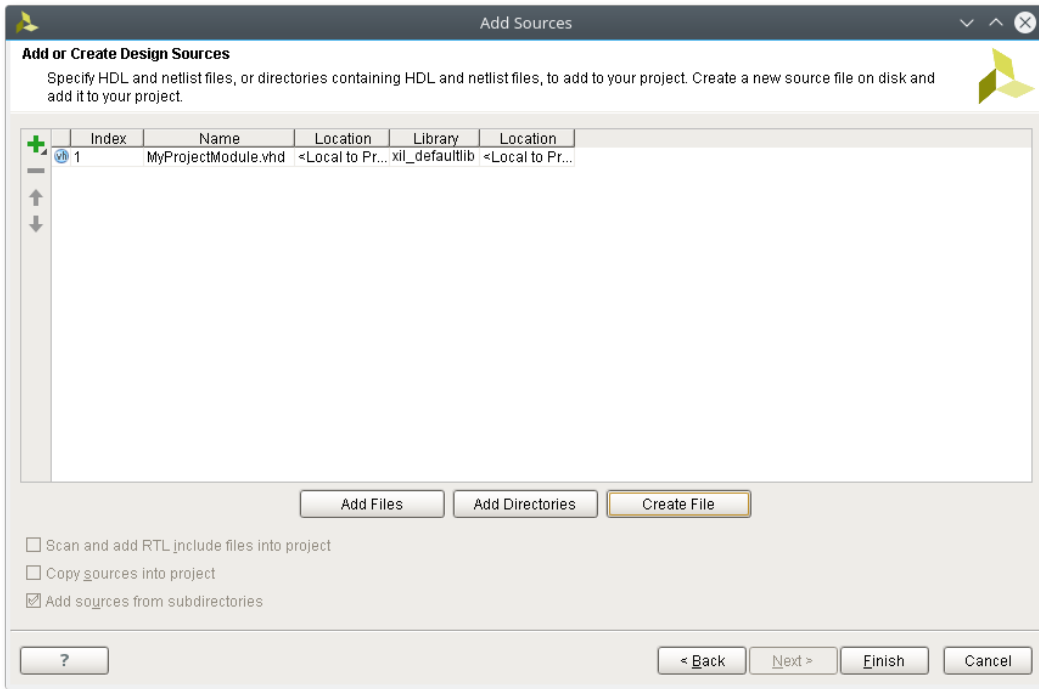


Click on the green + icon. A previously created VHDL modules could be added to the project by clicking Add Files. To create a new module, click Create File

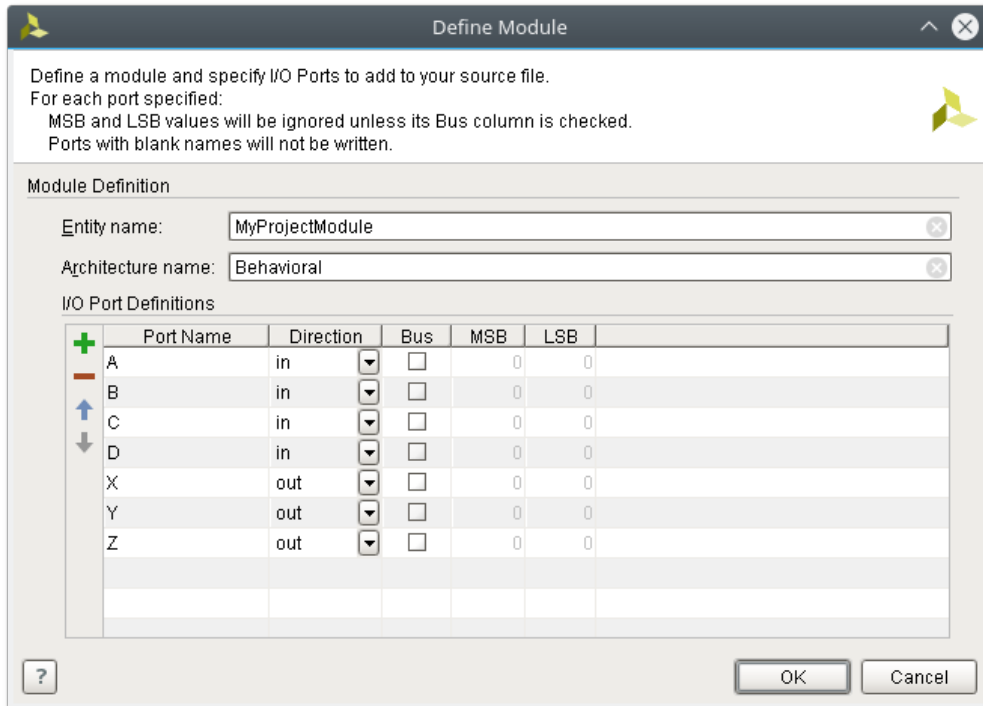


Select File type as VHDL. You can enter any File name for the module, but it is recommended to not use any spaces. Click OK

This returns to the Add or Create Design Sources so multiple modules can be added at the same time.



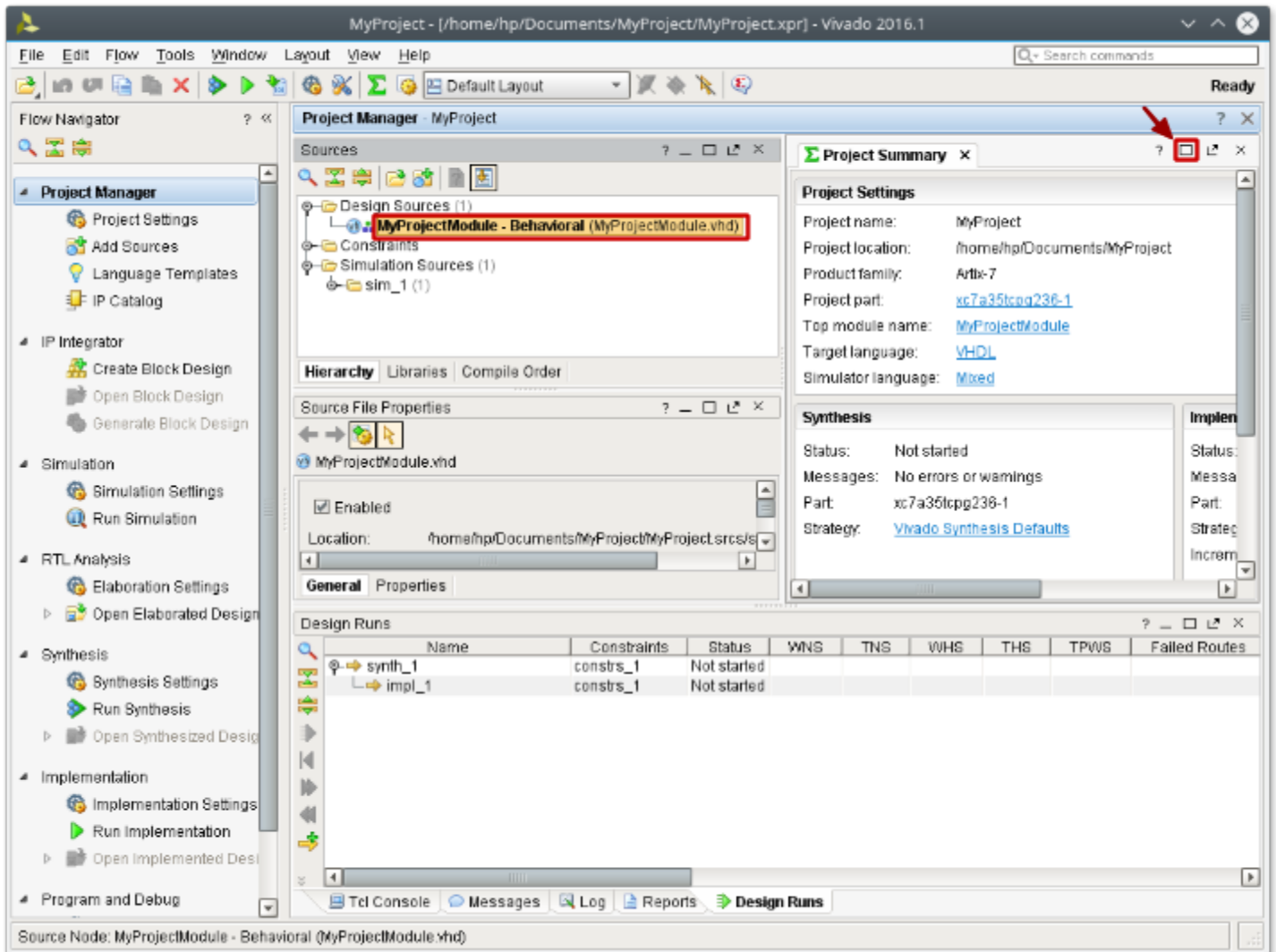
After all of the wanted sources have been added, click Finish. This will bring up a Define Module dialogue box to create a framework for each of the VHDL modules. With this dialogue, each module can have the inputs and outputs defined. For this project, A, B, C, and D are inputs (in) and X, Y, and Z are outputs (out).



After all of the inputs and outputs are defined, click OK to return to Vivado

VHDL Editor

The added VHDL module will show up in the Source window. Double clicking on the module name in the Source window will open the file in the editor window. To make editing easier, the editor maximize button will cause the editor window to fill the Vivado application.



Looking at the editor window, the module has been created with all of the inputs and outputs defined

```
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx leaf cells in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity MyProjectModule is
35     Port ( A : in STD_LOGIC;
36           B : in STD_LOGIC;
37           C : in STD_LOGIC;
38           D : in STD_LOGIC;
39           X : out STD_LOGIC;
40           Y : out STD_LOGIC;
41           Z : out STD_LOGIC);
42 end MyProjectModule;
43
44 architecture Behavioral of MyProjectModule is
45
46     begin
47
48
49 end Behavioral;
50
```

Add the necessary VHDL code for the logic circuits wanted. The added code will go after the **begin** and before the **end Behavioral;**

$$X = A \cdot B + C \cdot D$$

$$Y = (A + B) \cdot (C + D) \quad Z = A \oplus B \oplus C \oplus D$$

```
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx leaf cells in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity MyProjectModule is
35     Port ( A : in STD_LOGIC;
36           B : in STD_LOGIC;
37           C : in STD_LOGIC;
38           D : in STD_LOGIC;
39           X : out STD_LOGIC;
40           Y : out STD_LOGIC;
41           Z : out STD_LOGIC);
42 end MyProjectModule;
43
44 architecture Behavioral of MyProjectModule is
45
46     begin
47
48         X <= (A and B) or (C and D);
49         Y <= (A or B) and (C or D);
50         Z <= A xor B xor C xor D;
51
52
53 end Behavioral;
```

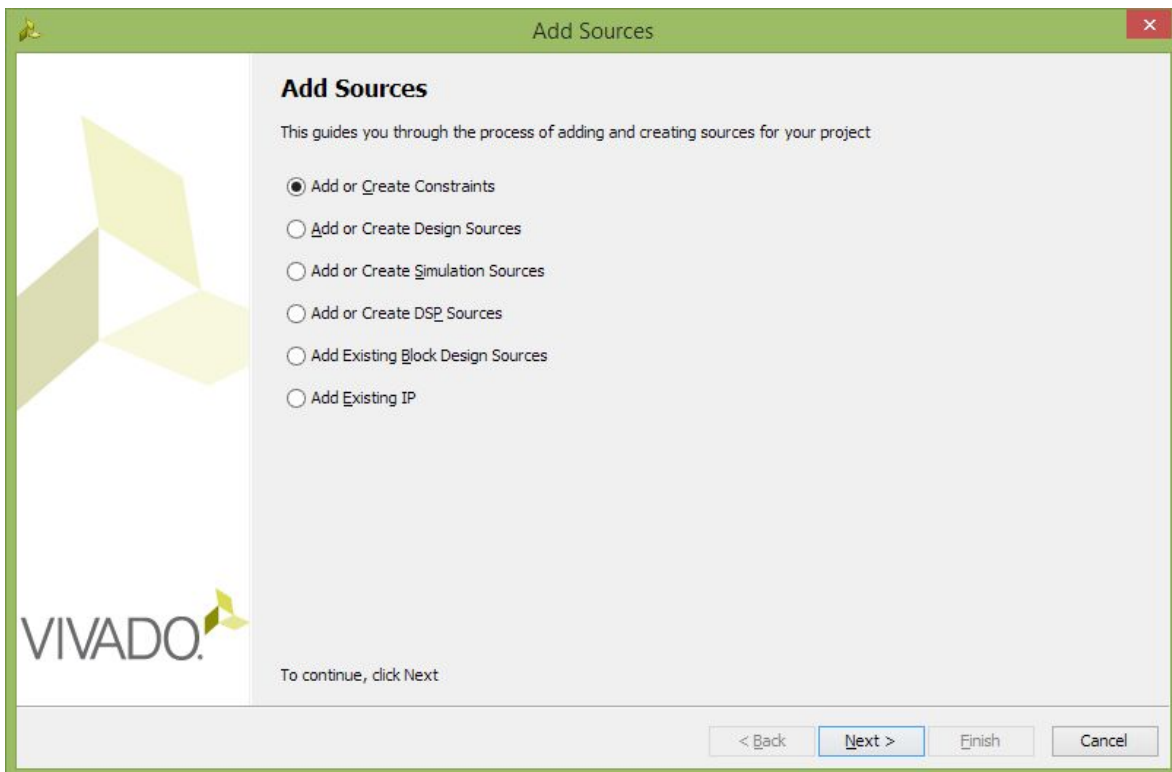
The design can now be synthesized and simulated (refer to the ISim tutorial). To create a programming file to be loaded onto the FPGA, we will need to add another source file.

Constraint File

The Constraints file will tell the FPGA where to connect the inputs and outputs of your logic circuit to the development board. In this example the inputs A, B, C, and D will be connected to switches on the Basys3, SW0, SW1, SW2, and SW3 respectively. The outputs X, Y, and Z will be connected to the LEDs LD0, LD1, and LD2.

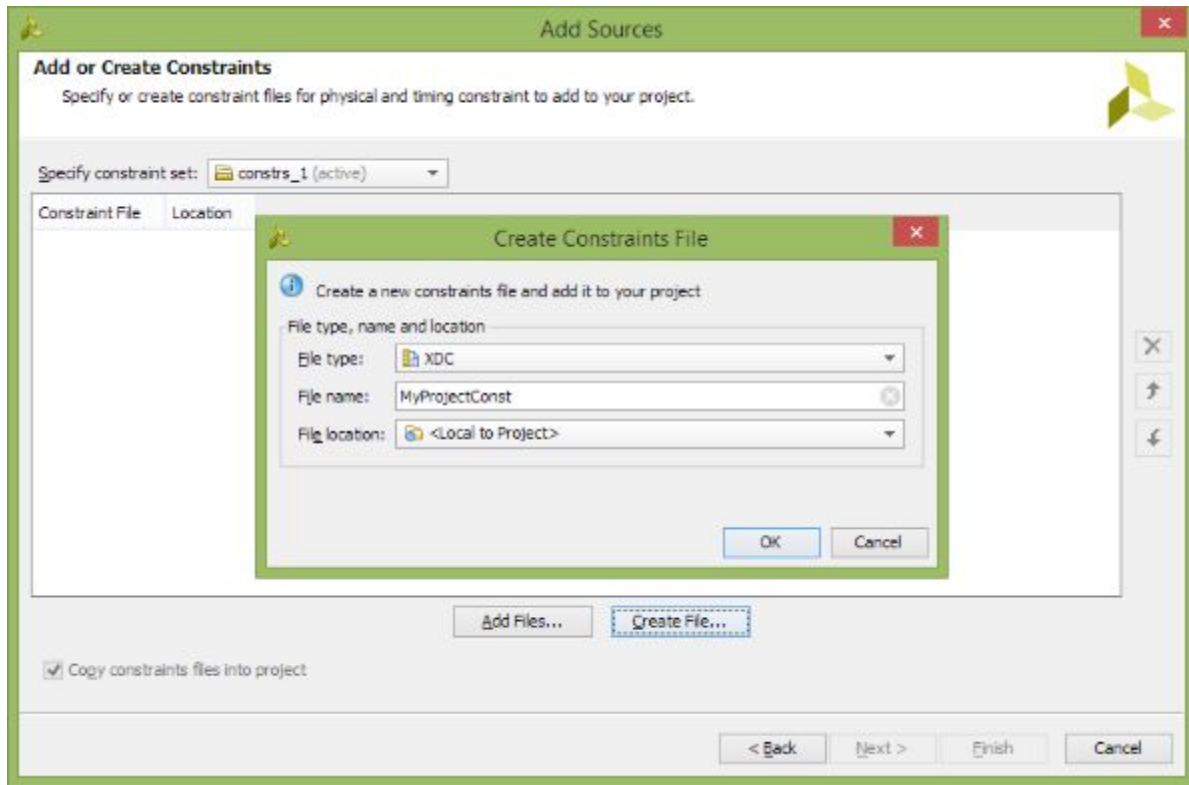
Any circuit input can be connected to any input on the FPGA development board, although multiple inputs cannot be connected to the same component. In the same way, multiple outputs cannot be connected to the same device on the development board.

Add a new source file as before, selecting "Add or Create Constraints File" from the Add Sources dialogue this time and click Next



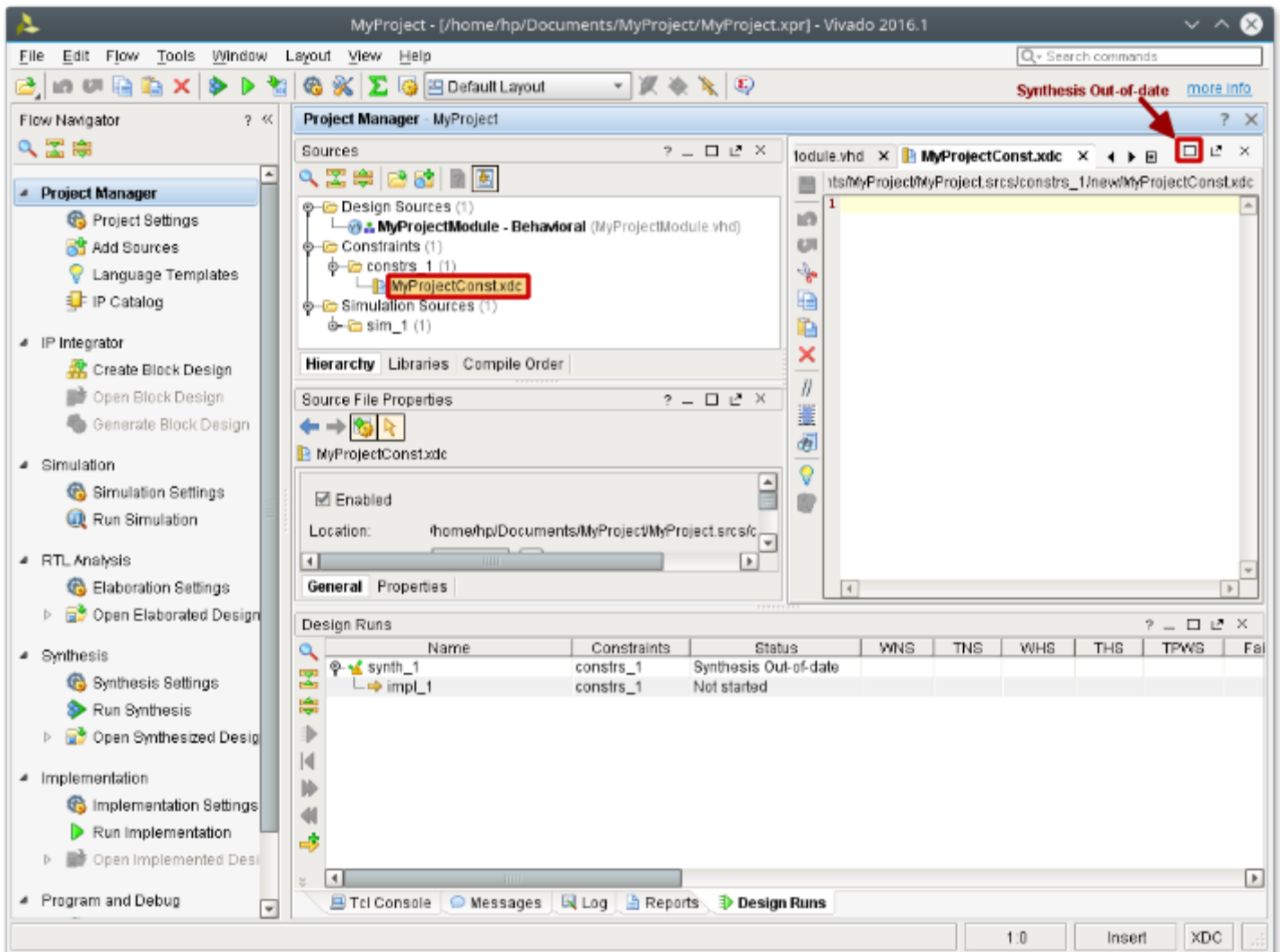
Just as before with the VHDL module, a previously created constraints file could be added to the project by clicking Add Files. To create a new file, click Create File

Select a file type of XDC. You can enter any File name for the module, but it is recommended to not use any spaces. Notice the File location for saving the new file is in the current project folder. Click OK



This returns to the Add or Create Constraints so multiple files can be added at the same time. Only 1 file is needed for this example, so click Finish

The added constraints file will show up in the Source window. Double clicking on the constraints file name in the Source window will open the file in the editor window. To make editing easier, the editor maximize button will cause the editor window to fill the Vivado screen. The constraints file will show up as a blank file.



Each input and output being connected requires 2 lines in the XDC file. The format the XDC file is:

```
set_property PACKAGE_PIN Port_Number [get_ports {Net_Label}]  
set_property IOSTANDARD LVCMOS33 [get_ports {Net_Label}]
```

Where **Net_Label** is the label given for the input or output in the VHDL module and **Port_Number** is the port address connected to the development board device. **These port addresses can all be found in the reference manual for the development board.** In this example, we will use the following connections on a Basys3 development board.

I/O	Device	Port Number
A	Switch 0 (SW0)	V17
B	Switch 1 (SW1)	V16
C	Switch 2 (SW2)	W16
D	Switch 3 (SW3)	W17
X	LED 0 (LD0)	U16
Y	LED 1 (LD1)	E19
Z	LED 2 (LD2)	U19

The finished XDC file would look like:

```
set_property PACKAGE_PIN V17 [get_ports {A}]  
set_property IOSTANDARD LVCMOS33 [get_ports {A}]
```

```
set_property PACKAGE_PIN V16 [get_ports {B}]  
set_property IOSTANDARD LVCMOS33 [get_ports {B}]
```

```
set_property PACKAGE_PIN W16 [get_ports {C}]  
set_property IOSTANDARD LVCMOS33 [get_ports {C}]
```

```
set_property PACKAGE_PIN W17 [get_ports {D}]  
set_property IOSTANDARD LVCMOS33 [get_ports {D}]
```

```
set_property PACKAGE_PIN U16 [get_ports {X}]  
set_property IOSTANDARD LVCMOS33 [get_ports {X}]
```

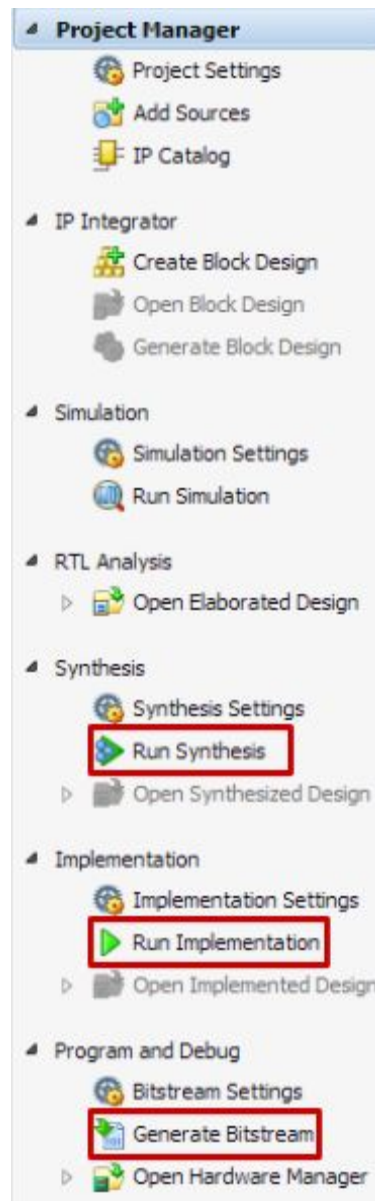
```
set_property PACKAGE_PIN E19 [get_ports {Y}]  
set_property IOSTANDARD LVCMOS33 [get_ports {Y}]
```

```
set_property PACKAGE_PIN U19 [get_ports {Z}]  
set_property IOSTANDARD LVCMOS33 [get_ports {Z}]
```

Spacing is important! Make sure to use a single space between get_ports and {} and no space inside {}
Programming File Generation

We can now create a programming file to load on our Basys3 development board. The Flow Navigator panel lists the processes in order that need to be run to create a programming file.

1. Run Synthesis
2. Run Implementation
3. Generate Bitstream



Each process can be run individually by double clicking on them or right clicking and selecting Launch. If you double click on a process that requires a previous process that has not already been run, the prerequisite processes will be run automatically.

If the Generate Programming File process finishes without error, it will create a .bit file you can use to program the development board. Programming the board can be accomplished with Digilent Adept. If there are any errors in the process, they will show up in the Messages Tab of the Console Panel.

The created bit file is saved in the project folder under ProjectName.runs/impl_1/ModuleName.bit