Xilinx Vivado VHDL Tutorial

This tutorial will provide instructions on how to:

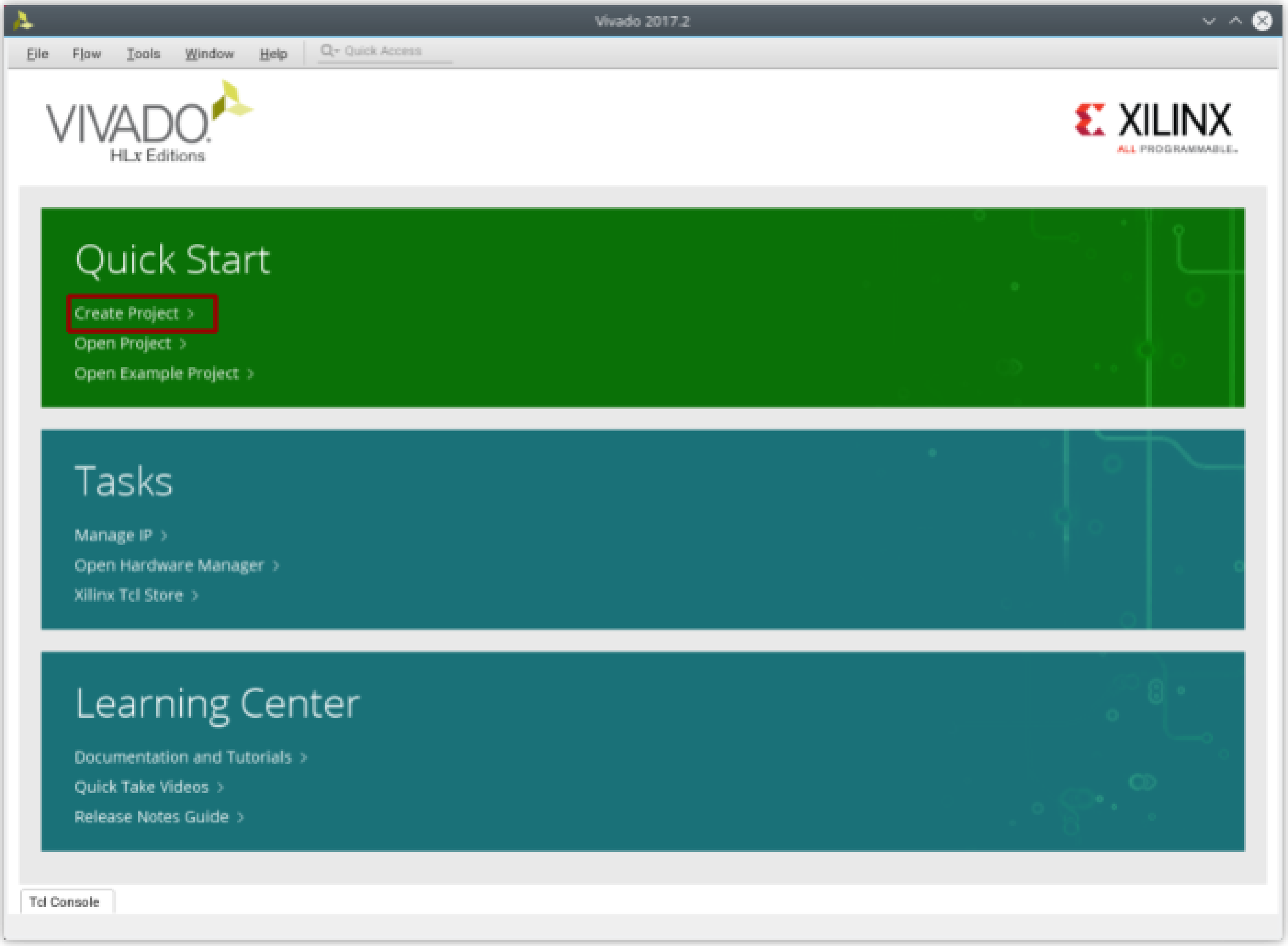
- Create a Xilinx Vivado project
- Create a VHDL module
- Create a constraint file (XDC)
- Generate a Programming file for the Basys3

Creating a Xilinx Project

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

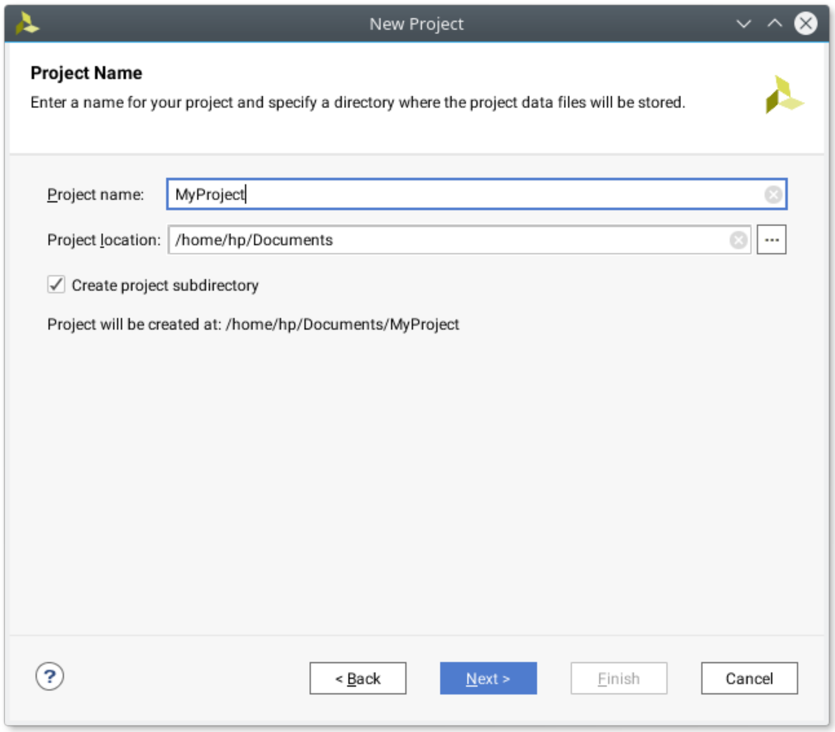
$$X = A \cdot B + C \cdot D \qquad Y = (A + B) \cdot (C + D) \qquad Z = A \oplus B \oplus C \oplus D$$

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



Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

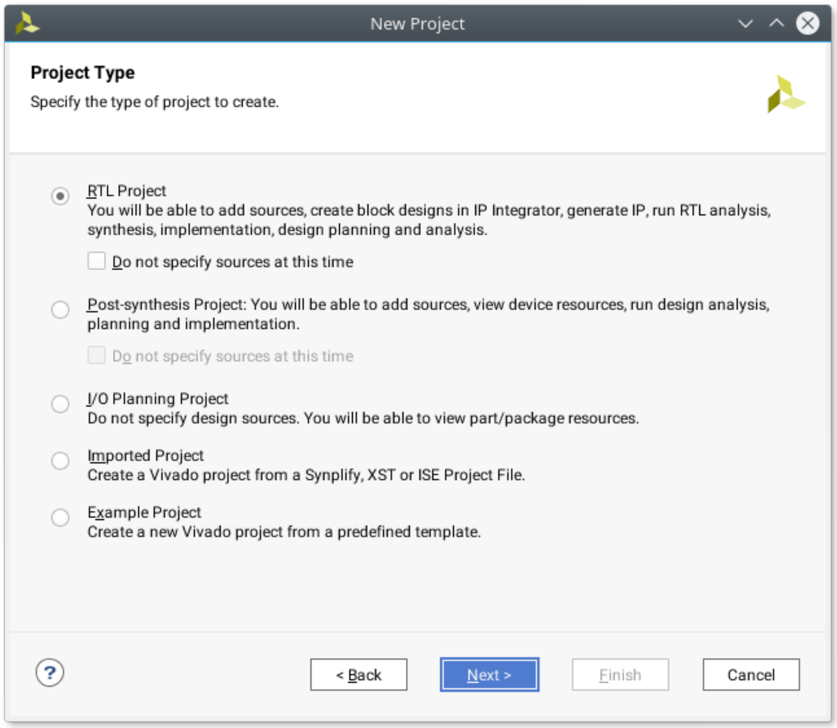
Project name: MyProject

Project location: /home/hp/Documents

[✓] Create project subdirectory

Project will be created at: /home/hp/Documents/MyProject

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.



Project Type

Specify the type of project to create.

● RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

☐ Do not specify sources at this time

○ Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.

☐ Do not specify sources at this time

○ I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

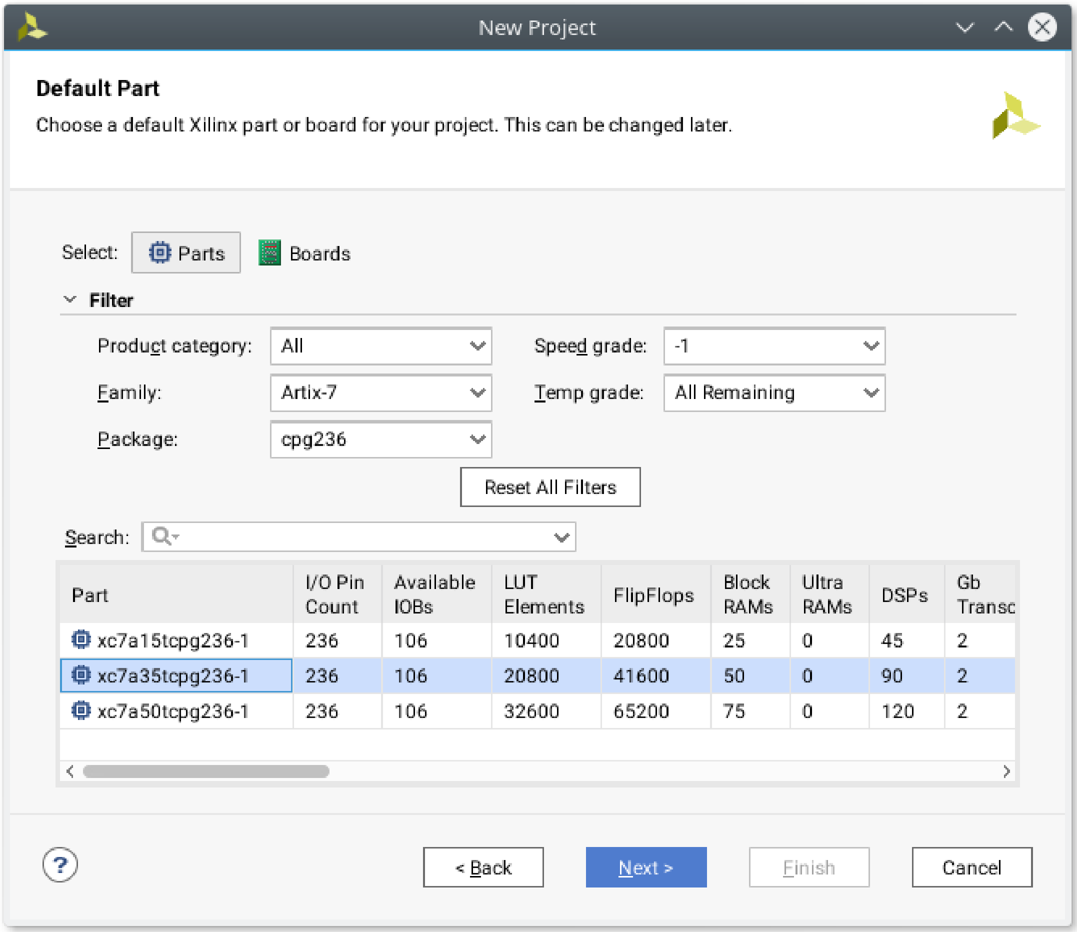
○ Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

○ Example Project
Create a new Vivado project from a predefined template.

The next 2 dialogue boxes are for adding Sources, 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



Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

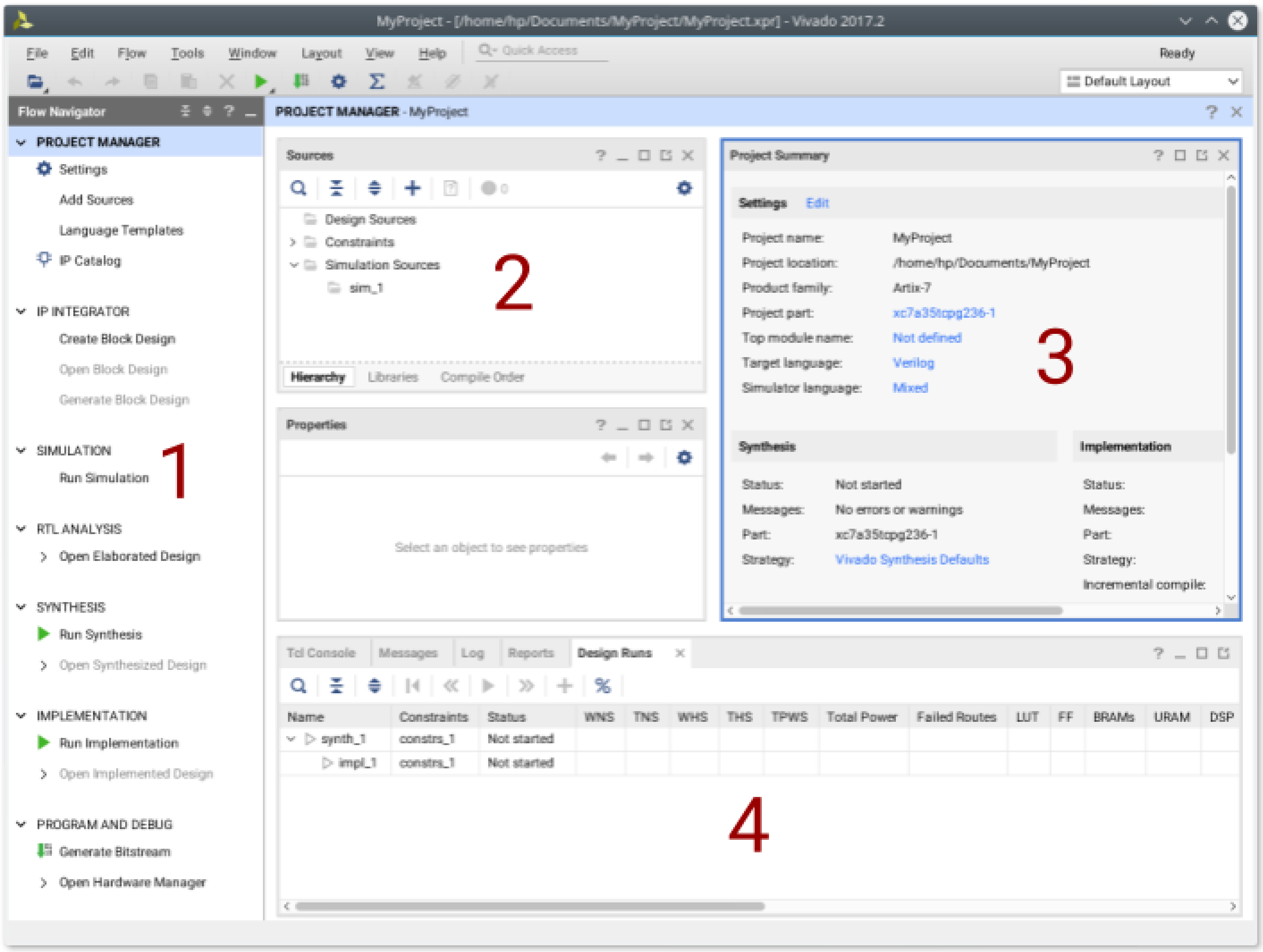
Product category: All
Family: Artix-7
Package: cpg236
Speed grade: -1
Temp grade: All Remaining
Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transc
xc7a15tcpg236-1	236	106	10400	20800	25	0	45	2
xc7a35tcpg236-1	236	106	20800	41600	50	0	90	2
xc7a50tcpg236-1	236	106	32600	65200	75	0	120	2

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

<u>Vivador Interface</u>

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.



Project name:	MyProject
Project location:	/home/hp/Documents/MyProject
Product family:	Artix-7
Project part:	xc7a35tcpg236-1
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

Status:	Not started	Status:	
Messages:	No errors or warnings	Messages:	
Part:	xc7a35tcpg236-1	Part:	
Strategy:	Vivado Synthesis Defaults	Strategy:	
		Incremental compile:	

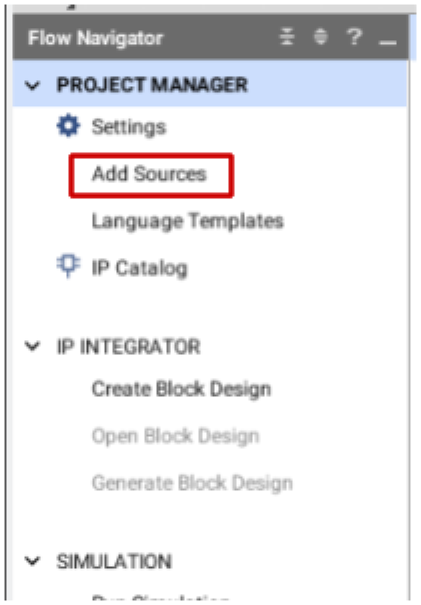
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	Not started												
impl_1	constrs_1	Not started												

There are 4 main sections to the Project Navigator interface:

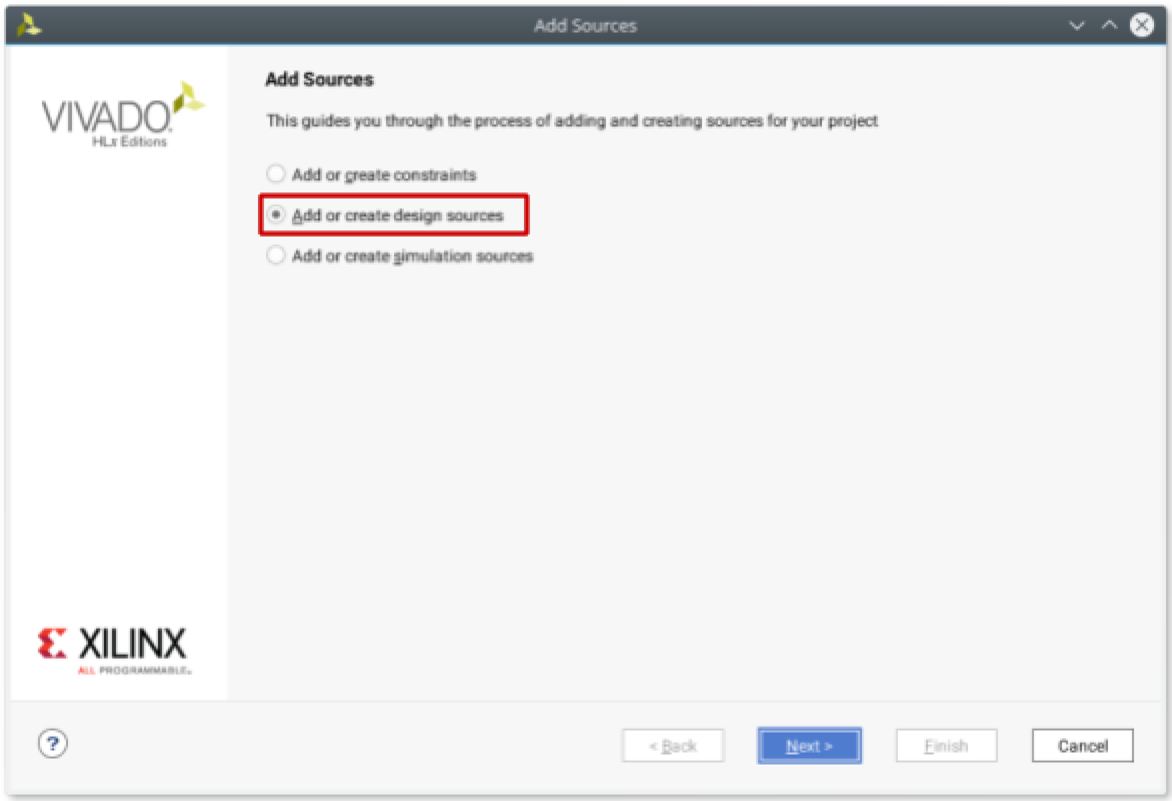
1. Flow Navigator: List of all available processes that can be run in the current project and their current state if running.
2. Sources Window: This is where all of the source files for the project are listed.
3. Editor Window: Display of the current file being edited. If no source file is selected, 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.

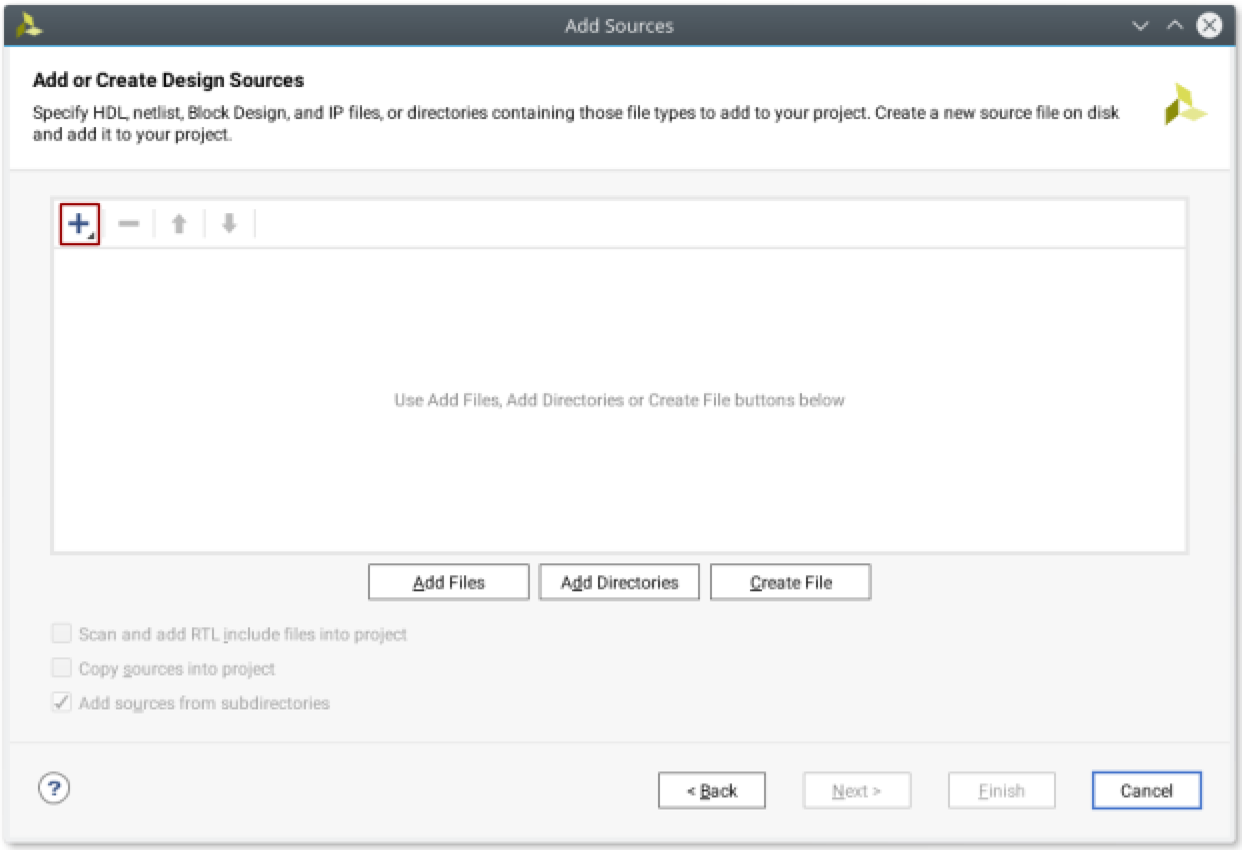


Add Sources

This guides you through the process of adding and creating sources for your project

- Add or create constraints
- [✓ Add or create design sources]
- Add or create simulation 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.


Add Sources

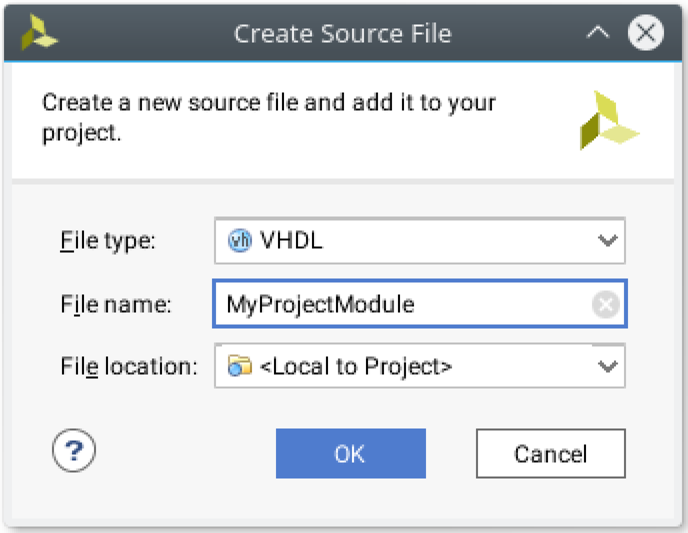
Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

Use Add Files, Add Directories or Create File buttons below

Add Files
Add Directories
Create File
Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
< Back
Next >
Finish
Cancel

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


Create Source File

Create a new source file and add it to your project.

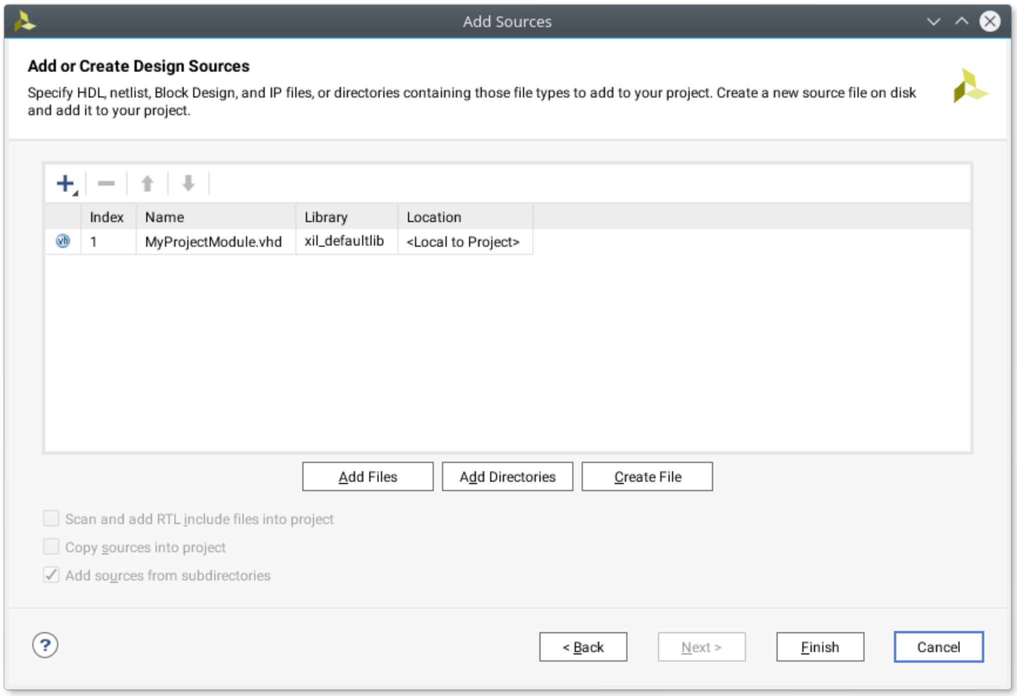
File type: VHDL

File name: MyProjectModule

File location: <Local to Project>

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.



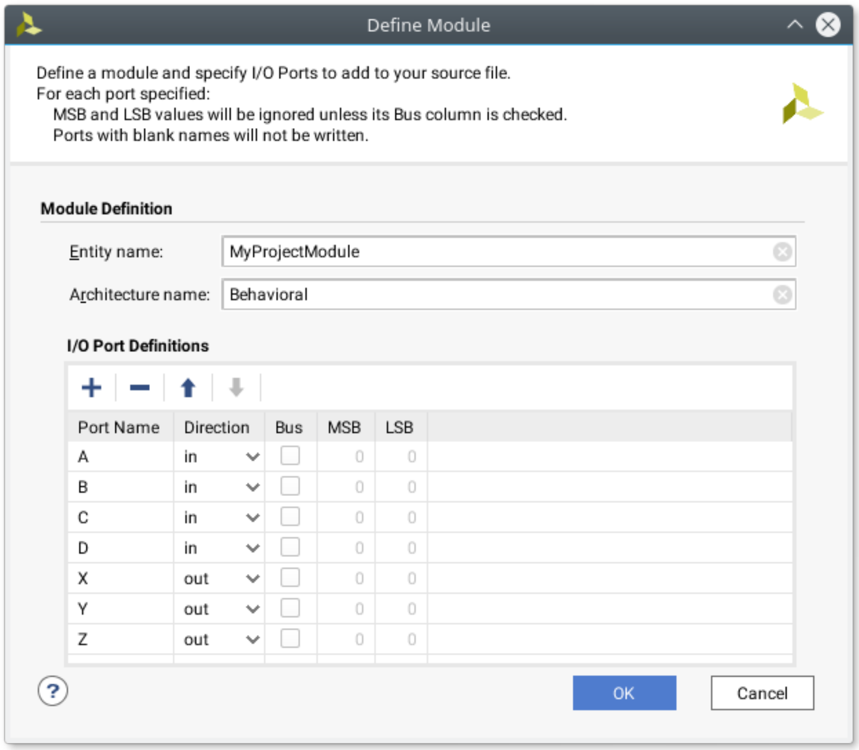
Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

	Index	Name	Library	Location
	1	MyProjectModule.vhd	xil_defaultlib	<Local to Project>

- [ ] Scan and add RTL include files into project
- [ ] Copy sources into project
- [✓] Add sources from subdirectories

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).



Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Entity name: MyProjectModule
Architecture name: Behavioral

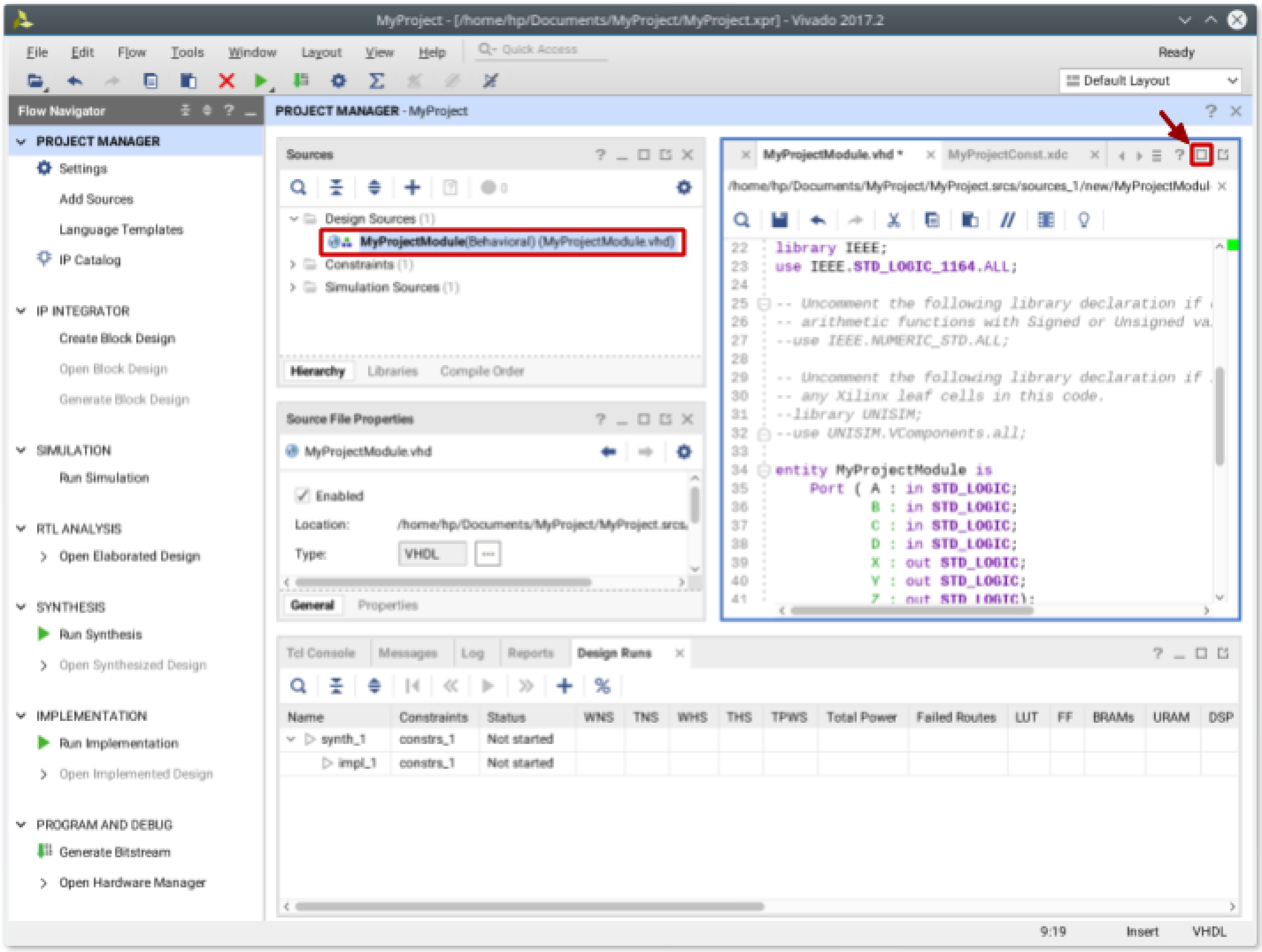
I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
A	in	☐	0	0
B	in	☐	0	0
C	in	☐	0	0
D	in	☐	0	0
X	out	☐	0	0
Y	out	☐	0	0
Z	out	☐	0	0

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.



Location: /home/hp/Documents/MyProject/MyProject.srcs

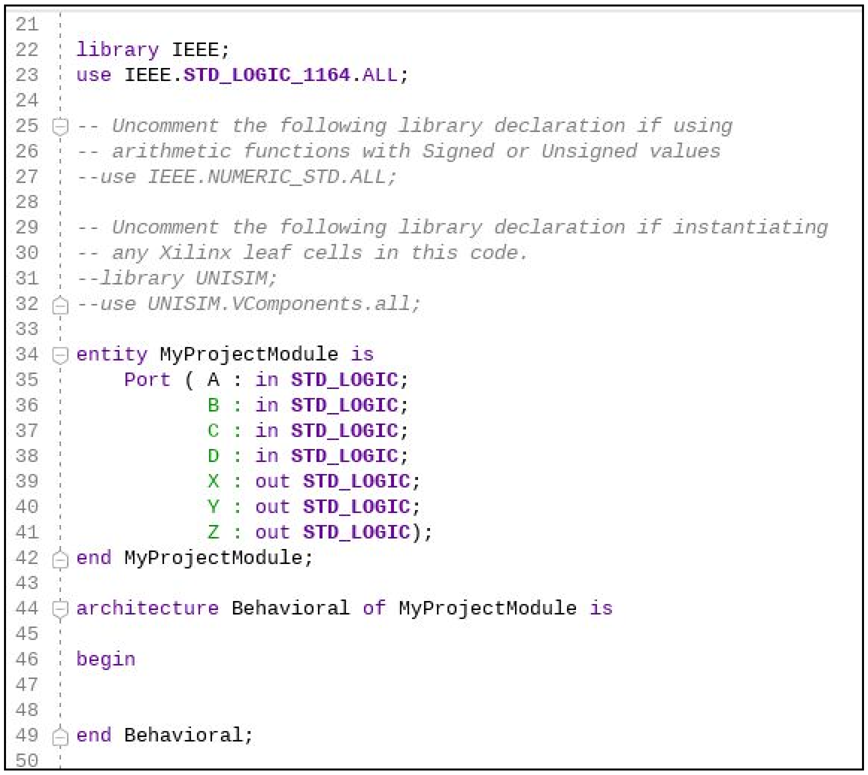
Type: VHDL

/home/hp/Documents/MyProject/MyProject.srcs/sources_1/new/MyProjectModul

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if
-- arithmetic functions with Signed or Unsigned va
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity MyProjectModule is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : in STD_LOGIC;
           D : in STD_LOGIC;
           X : out STD_LOGIC;
           Y : out STD_LOGIC;
           Z : out STD_LOGIC);
```

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	Not started												
impl_1	constrs_1	Not started												

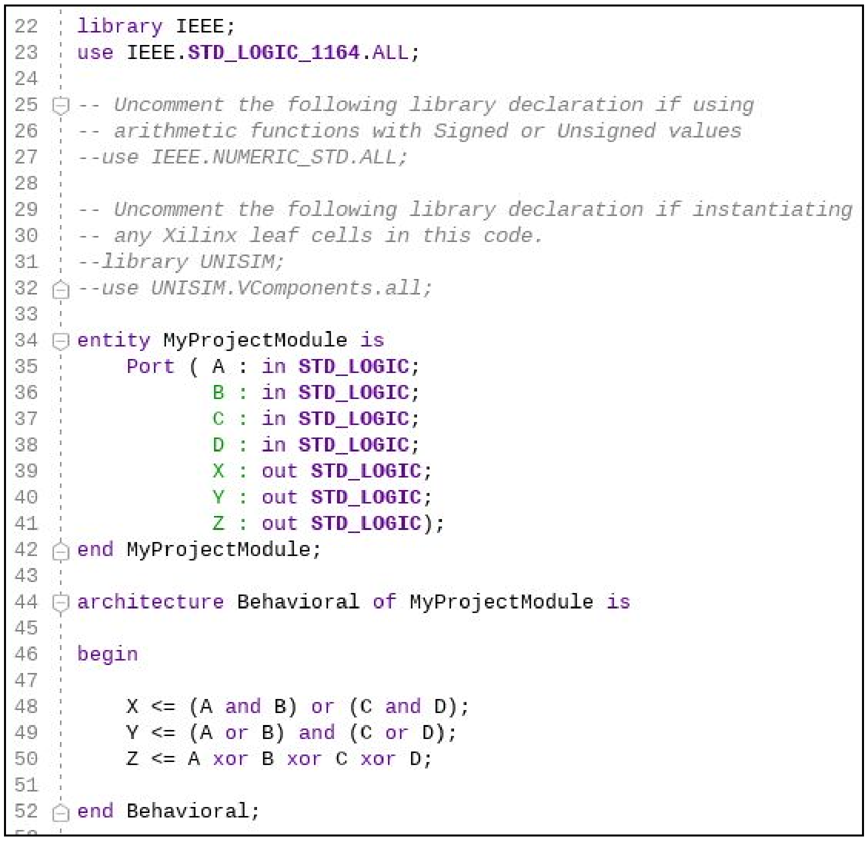
Looking at the editor window, the module has been created with all of the inputs and outputs defined



```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity MyProjectModule is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : in STD_LOGIC;
           D : in STD_LOGIC;
           X : out STD_LOGIC;
           Y : out STD_LOGIC;
           Z : out STD_LOGIC);
end MyProjectModule;

architecture Behavioral of MyProjectModule is

begin


end Behavioral;

```

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

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



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity MyProjectModule is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : in STD_LOGIC;
           D : in STD_LOGIC;
           X : out STD_LOGIC;
           Y : out STD_LOGIC;
           Z : out STD_LOGIC);
end MyProjectModule;

architecture Behavioral of MyProjectModule is

begin

    X <= (A and B) or (C and D);
    Y <= (A or B) and (C or D);
    Z <= A xor B xor C xor D;

end Behavioral;
```

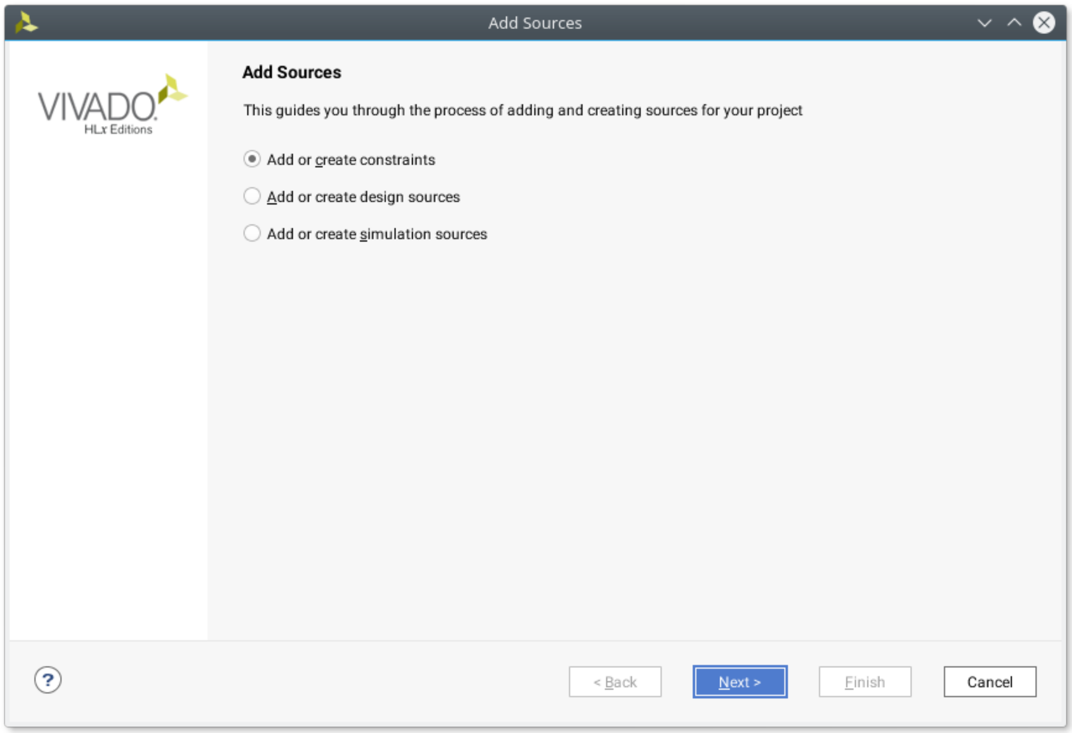
The design can now be synthesized and simulated (*refer to the Vivado Simulation tutorial*). To create a programming (.bit) file to be loaded onto the FPGA, we will need to add another source file.

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

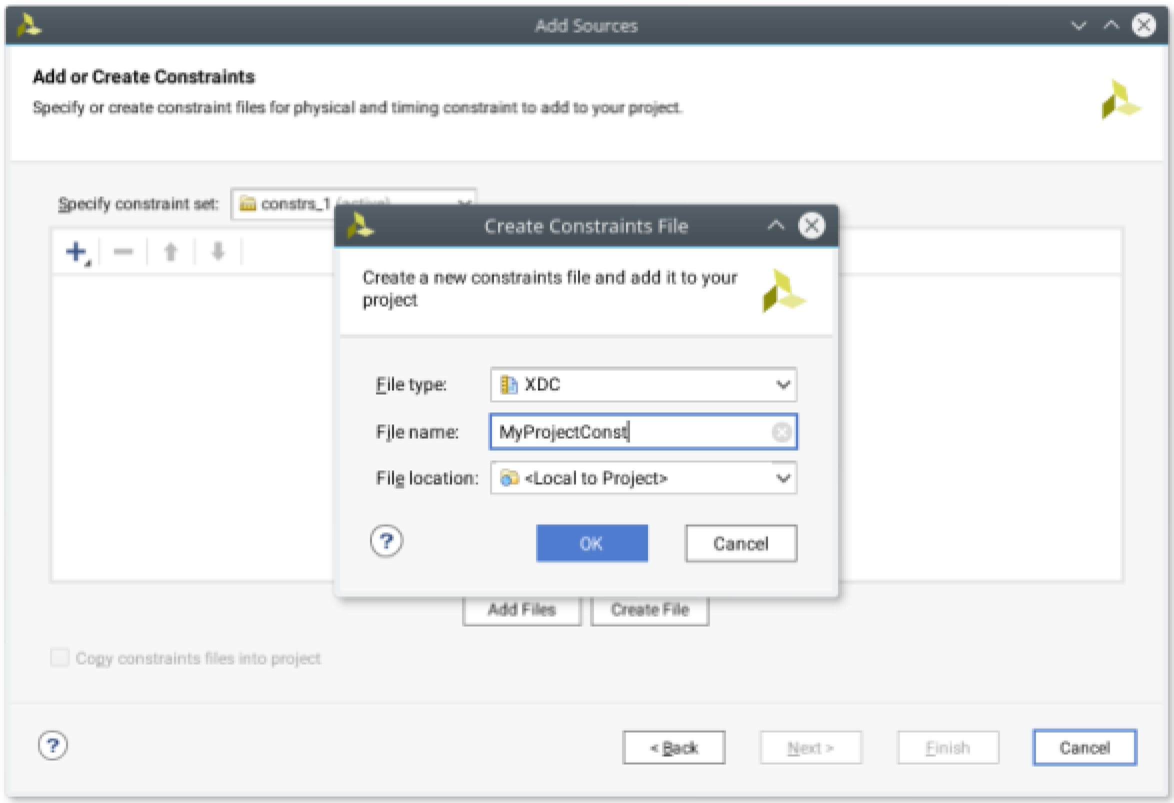

Add Sources

This guides you through the process of adding and creating sources for your project

(●) Add or create constraints
( ) Add or create design sources
( ) Add or create simulation sources

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


Add Sources

Add or Create Constraints

Specify or create constraint files for physical and timing constraint to add to your project.

Specify constraint set: constrs_1 (active)

Create Constraints File

Create a new constraints file and add it to your project

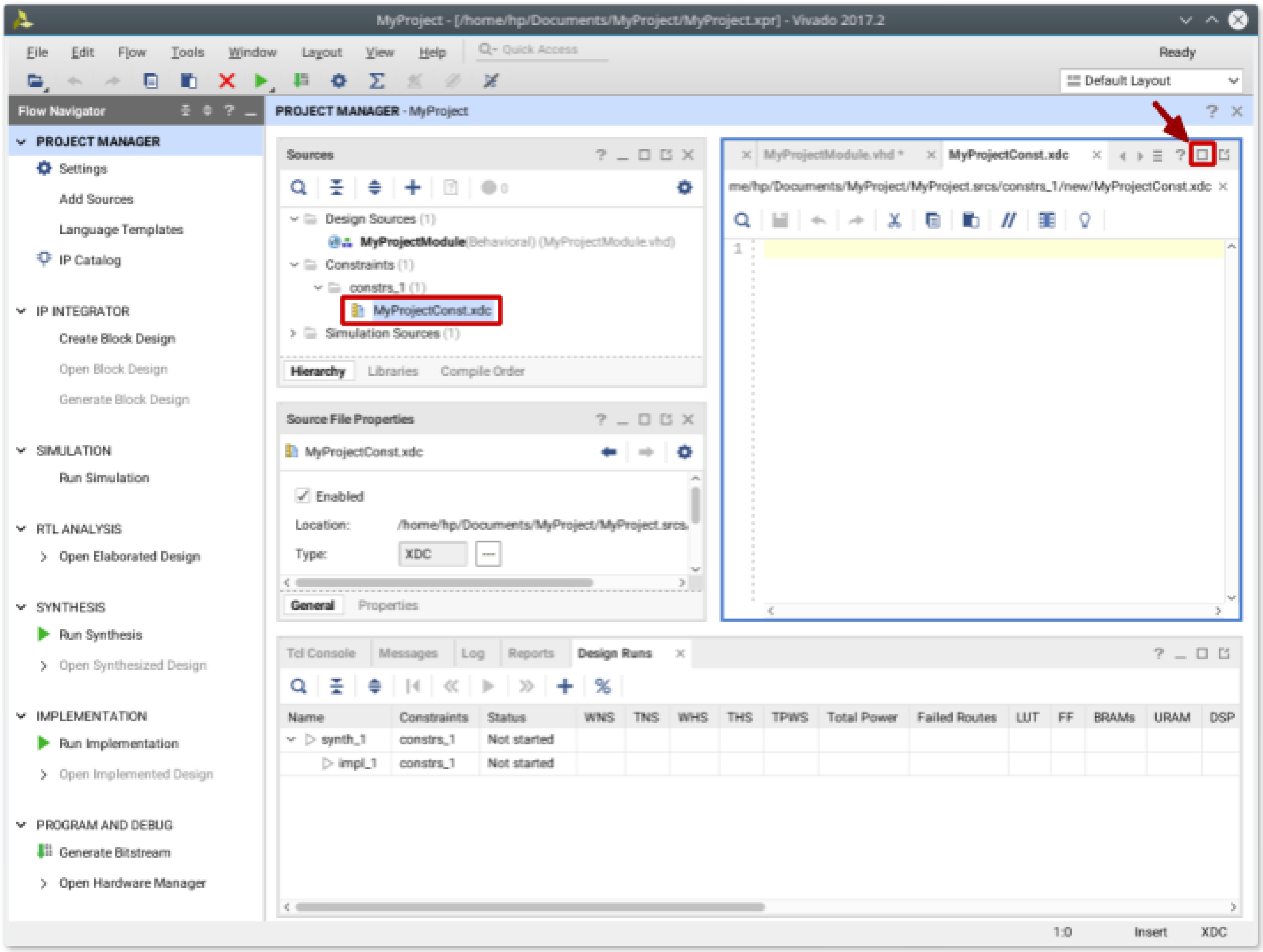
File type: XDC

File name: MyProjectConst

File location: <Local to Project>

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.



Enabled

Location: /home/hp/Documents/MyProject/MyProject.srcs

Type: XDC

me/hp/Documents/MyProject/MyProject.srcs/constrs_1/new/MyProjectConst.xdc

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	Not started												
impl_1	constrs_1	Not started												

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 {}

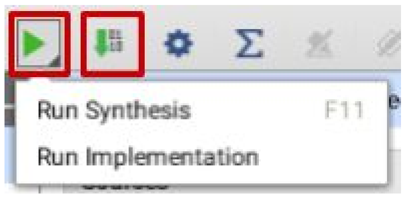
DO NOT copy and paste the above from this pdf, some non-printing characters will get copied and cause errors in the XDC file during bitstream generation

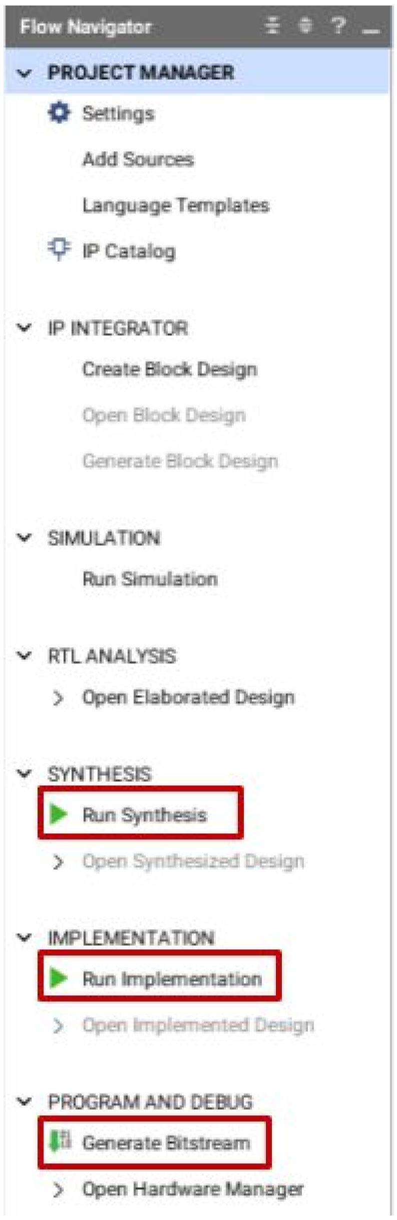
Programming File Generation

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 programing file.

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

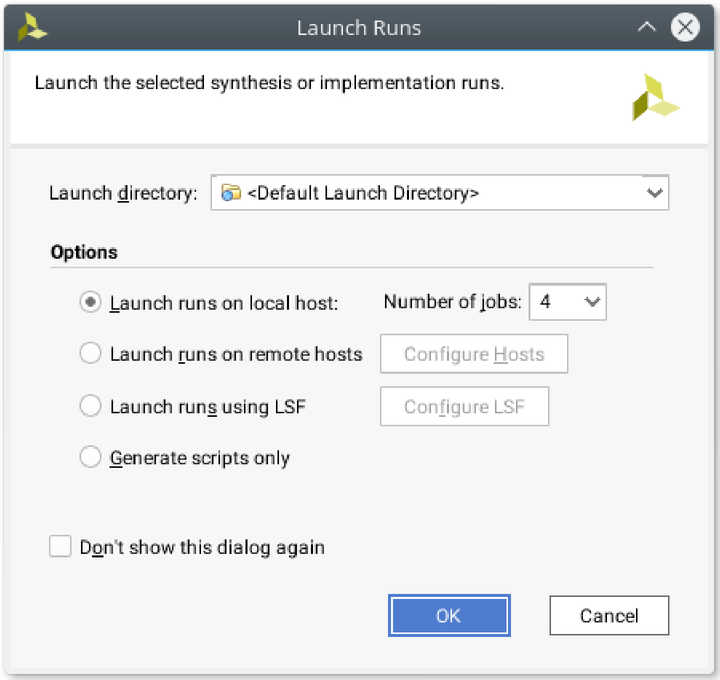
Alternatively you can use the toolbar buttons





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.

When running the synthesis and implementation, a pop up will offer running the process on remote servers. This can be useful when making large designs in commercial use cases. Our designs will be small and simple enough to be synthesizable in a reasonable amount of time on your local computer. The local number of jobs can be adjustable for those of you will multithreaded, multicore processors. The default value of 4 is suggested, but if you want to devote more or less cores to processing you can adjust accordingly. Click OK to begin the processing.


Launch Runs

Launch the selected synthesis or implementation runs.

Launch directory: <Default Launch Directory>

Options

[✓ Launch runs on local host:] Number of jobs: 4

Launch runs on remote hosts — Configure Hosts

Launch runs using LSF — Configure LSF

Generate scripts only

Don't show this dialog again

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 the Hardware Manager. If there are any errors in the process, they will show up in the Messages Tab of the Console Panel.

<u>The created bit file is saved in the project folder under ProjectName.runs/impl_#/ModuleName.bit</u>

Programming the FPGA (Basys3)

We can now program the FPGA with our generated bit file. There are two methods to do this, using Vivado's Hardware Manager or using Digilent's Adept software.

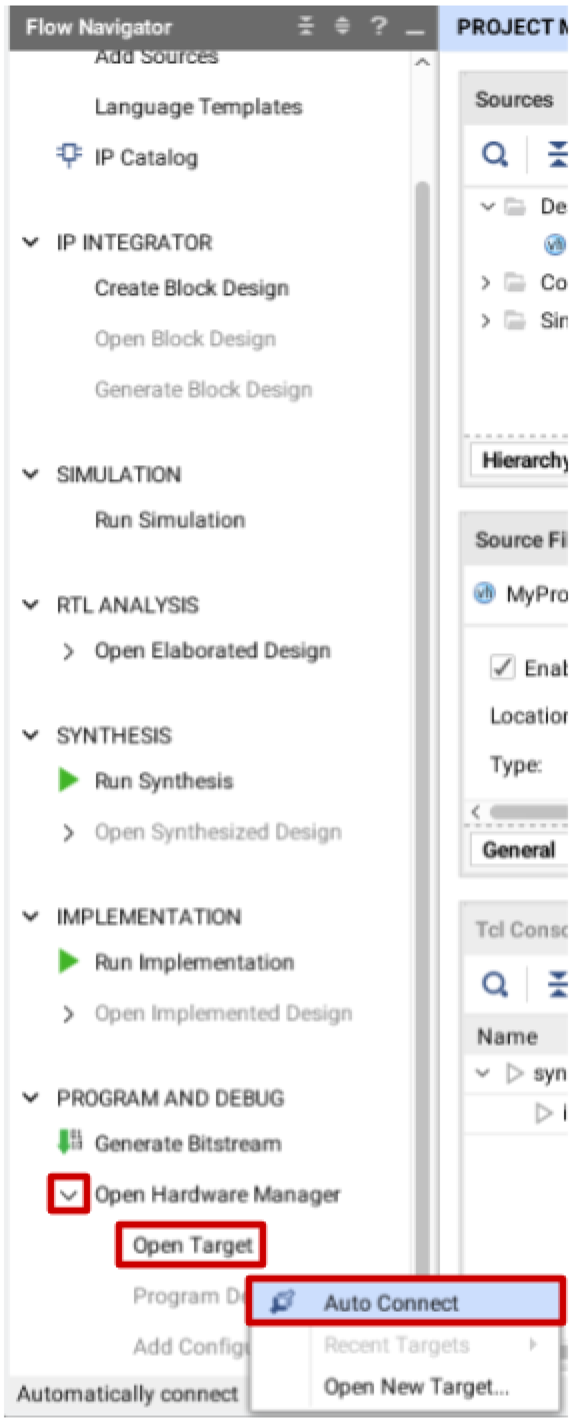
Before trying to connect to the Basys3, make sure it is plugged in and powered on!

Vivado's Hardware Manager Method

Expand "Open Hardware Manager" on the Flow Navigator panel by clicking on the triangle next to "Open Hardware Manager".

Click on the "Open Target" and select "Auto Connect"

If you get a windows firewall warning, click allow to give Vivado access.



Digilent Adept Method

Run Digilent Adept. Under Connect, select the Basys3

Click select and browse to the directory you saved the project. From there you can find the generated bit file in ***ProjectName/ProjectName.runs/impl_1/ModuleName.bit***

Click Program.