Xilinx Vivado Tutorial

By: Larbi Boughaleb, UT ECE
Updated by: Anthony Weerasinghe, 09/10, Daniel Arulraj 09/13, Yu-Wei Lee 12/14

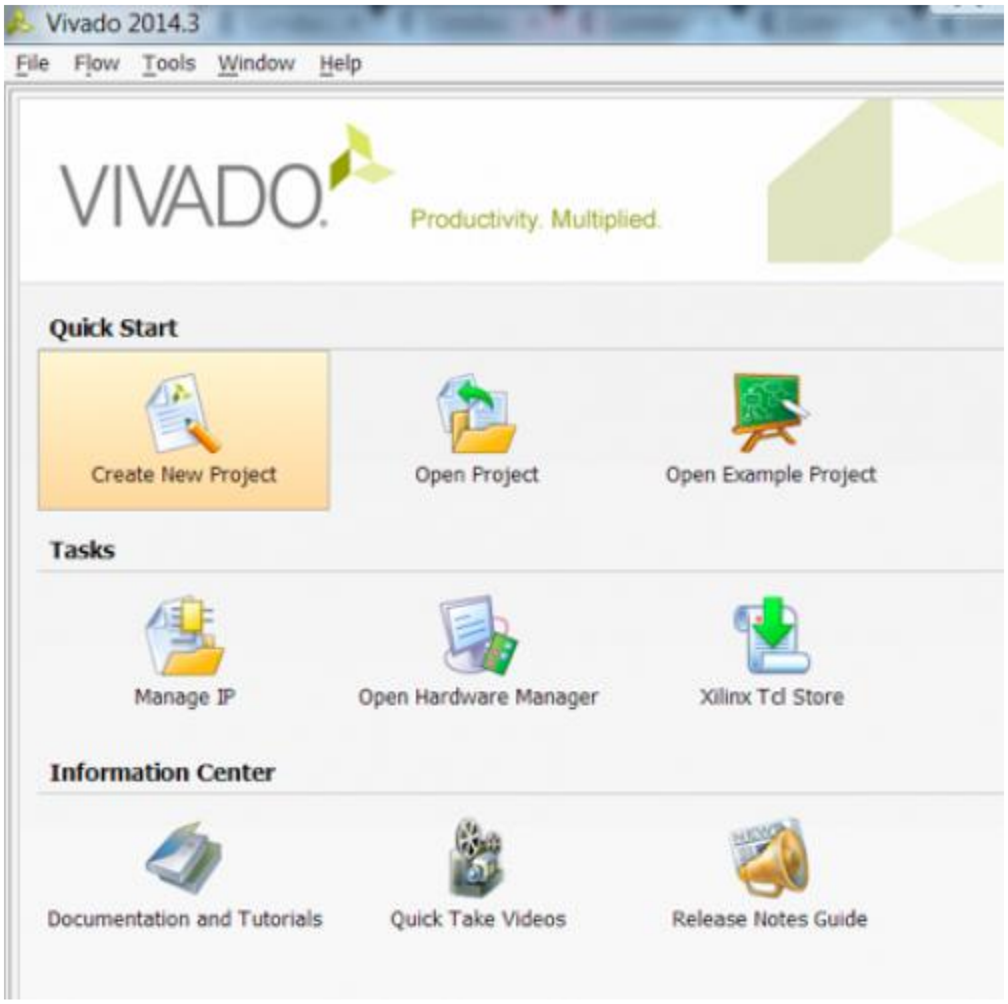
This tutorial takes you through the process of creating, synthesizing, simulating, implementing, and downloading a simple Verilog design to the Digilent Basys3 Board using Xilinx Vivado Software. Mainly, we will implement a 4-bit Adder that consists of four FullAdders that are updated when the inputs change.

Starting Vivado

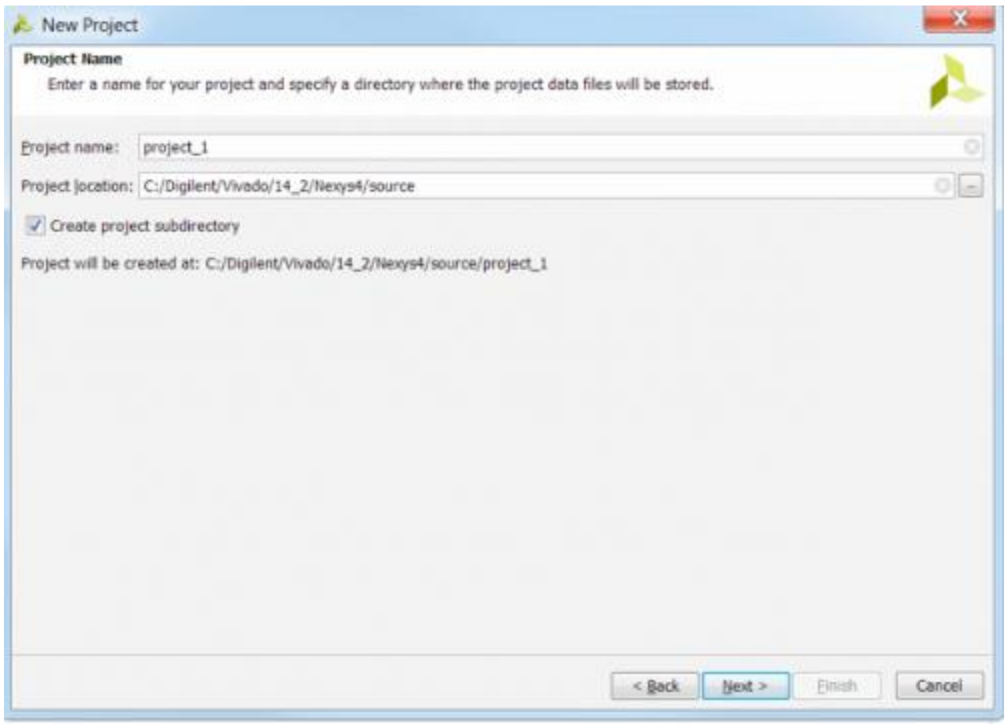
From any lab computer, open Vivado 2014.3

Creating a Project

Now that we opened Vivado, we're going to create a project.



This opens Vivado's New Project wizard. Click “Next” and you'll see this screen.



Project Name

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

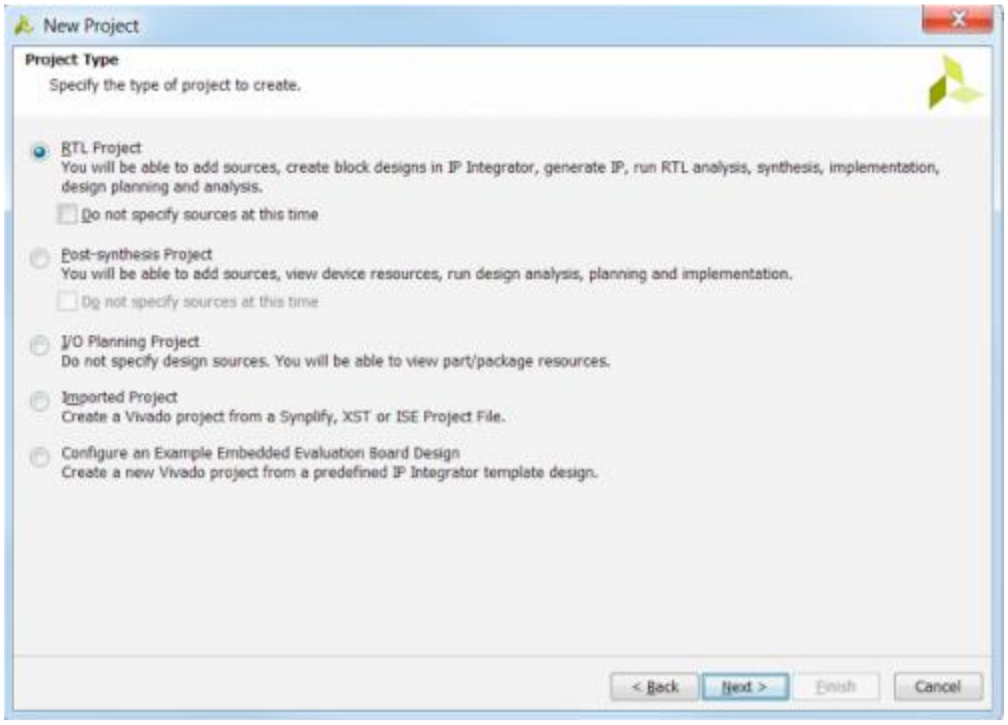
Project name: project_1

Project location: C:/Digilent/Vivado/14_2/Nexys4/source

[✓] Create project subdirectory

Project will be created at: C:/Digilent/Vivado/14_2/Nexys4/source/project_1

Name your project (no spaces!) and choose your project saving directory before clicking “Next”. You will now see this screen.



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.

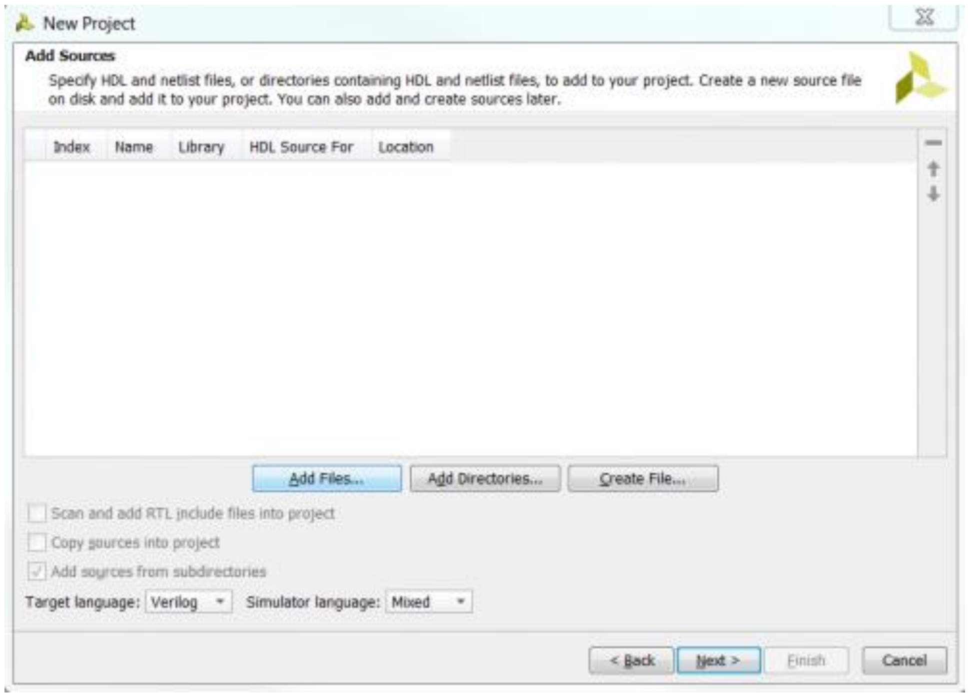
[ ] Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

[ ] Configure an Example Embedded Evaluation Board Design
Create a new Vivado project from a predefined IP Integrator template design.

Select RTL Project and click “Next”

1. Creating New Verilog files

In this window, you can select any other source files or directories that you'll want to use in your projects. We can also select which language we'll be programming in.


New Project

Add Sources

Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

Index	Name	Library	HDL Source For	Location

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

Target language: Verilog
Simulator language: Mixed

We'll be importing the pre-built Verilog files so click “Add Files” and add the code below to your files. At this point you should have two source files: Adder4.v and FullAdder.v.

Put in file Adder4.v:

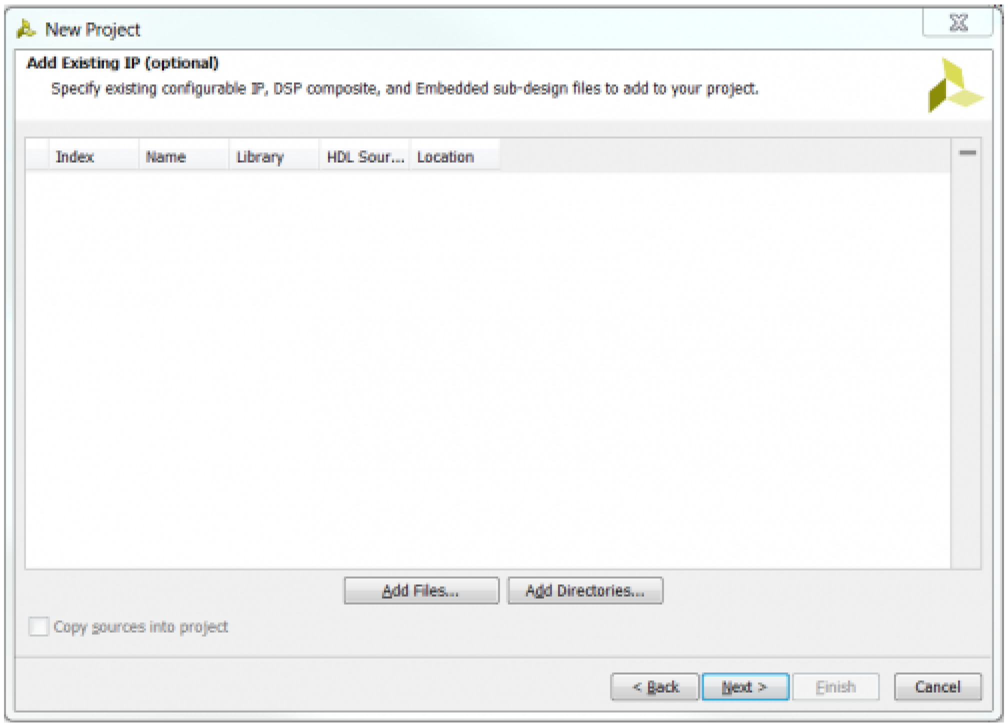
```
//Figure 2-12 Structural Description of a 4-Bit Adder
module Adder4 (S, Co, A, B, Ci);
  output [3:0] S;
output Co;
input [3:0] A, B;
  input Ci;

  wire [3:1] C; // C is an internal signal

// instantiate four copies of the FullAdder

  FullAdder a1(A[0], B[0], Ci, C[1], S[0]);
  FullAdder a2(A[1], B[1], C[1], C[2], S[1]);
  FullAdder a3(A[2], B[2], C[2], C[3], S[2]);
  FullAdder a4(A[3], B[3], C[3], Co, S[3]);

endmodule
```

Put in file FullAdder.v:

```
//Figure 2-10 Verilog Module for a Full Adder
module FullAdder (X, Y, Cin, Cout, Sum);
output Cout, Sum;
  input X, Y, Cin;

  assign #10 Sum = X ^ Y ^ Cin;
  assign #10 Cout = (X && Y) || (X && Cin) || (Y && Cin);

endmodule
```

It should be noted that if you check the “Copy sources into project” box, Vivado will create separate copies of these sources and place them within your project directory. Click “Next”.

This window lets you choose existing IPs (Intellectual Property) cores if you have them.

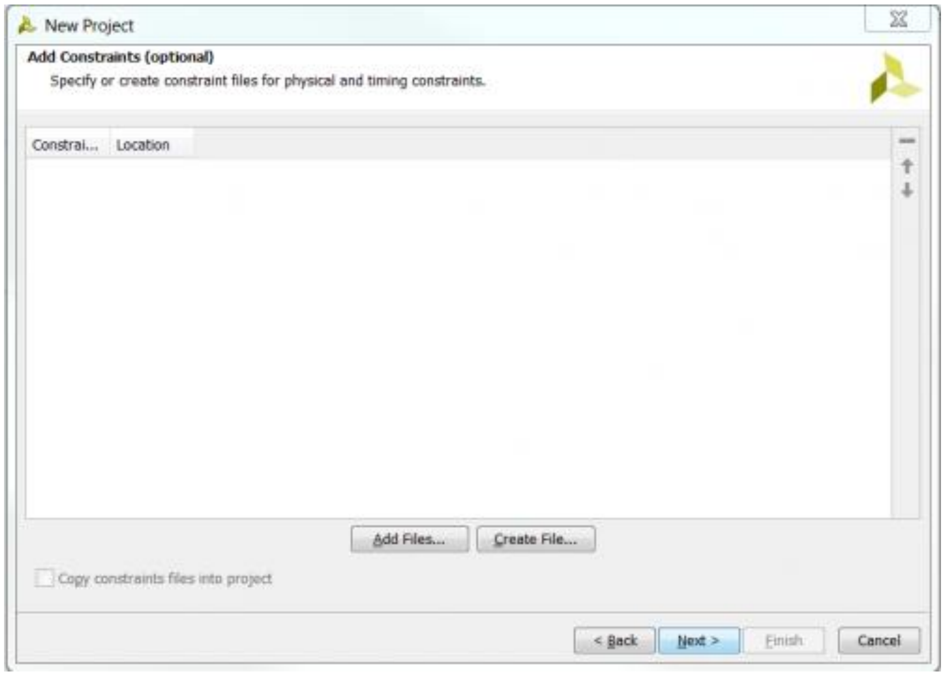

New Project

Add Existing IP (optional)

Specify existing configurable IP, DSP composite, and Embedded sub-design files to add to your project.

Index	Name	Library	HDL Sour...	Location

Click “Next” and you'll see this window.

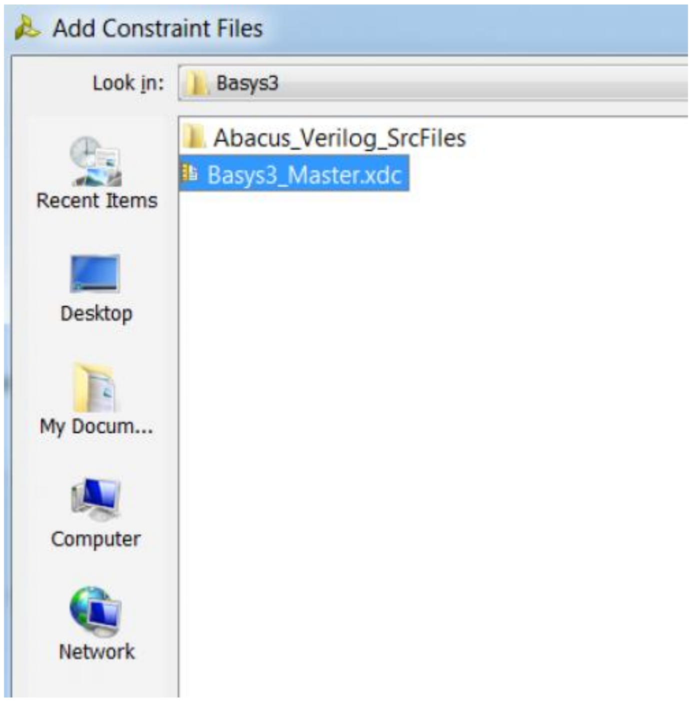

New Project

Add Constraints (optional)

Specify or create constraint files for physical and timing constraints.

Constrai...	Location

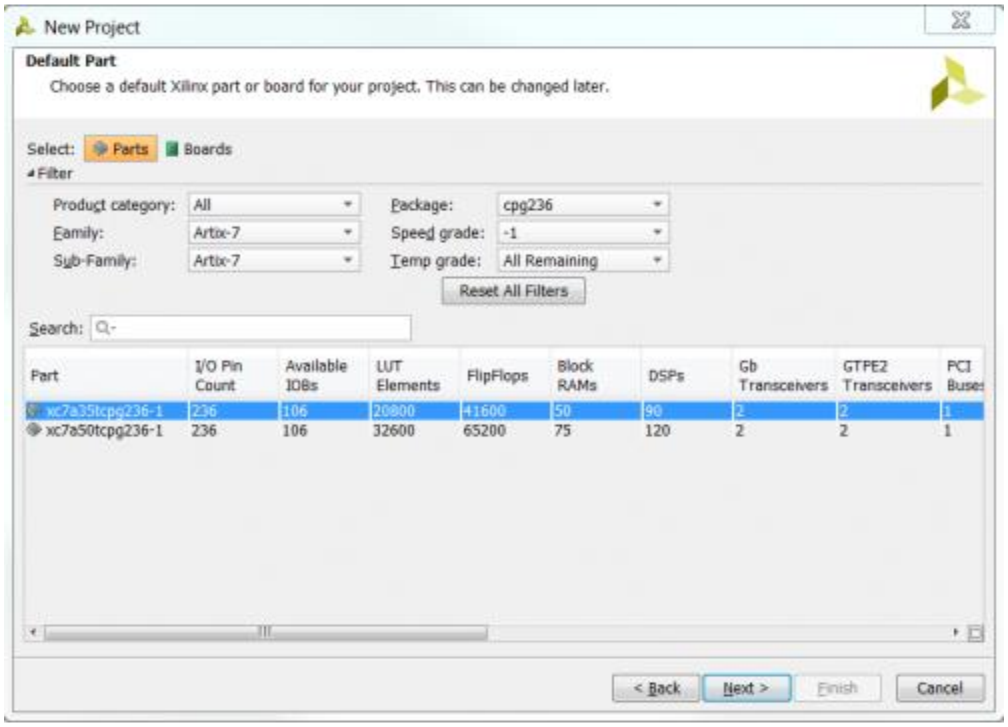
This is where we'll import our Xlilinx Design Constraints file (XDC) to map the HDL signals to the Artix-7 pins. You can get the Basys3 Master XDC file from Canvas under the lab files. Click on “Add Files”, navigate to where you saved your Basys3_Master.xdc file, select it, and click “Next”.


Add Constraint Files

Look in: Basys3

Abacus_Verilog_SrcFiles
Basys3_Master.xdc

At this point you'll see the part selection screen.


New Project

Default Part

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

Select: Parts Boards

Filter

Product category: All

Family: Artix-7

Sub-Family: Artix-7

Package: cpg236

Speed grade: -1

Temp grade: All Remaining

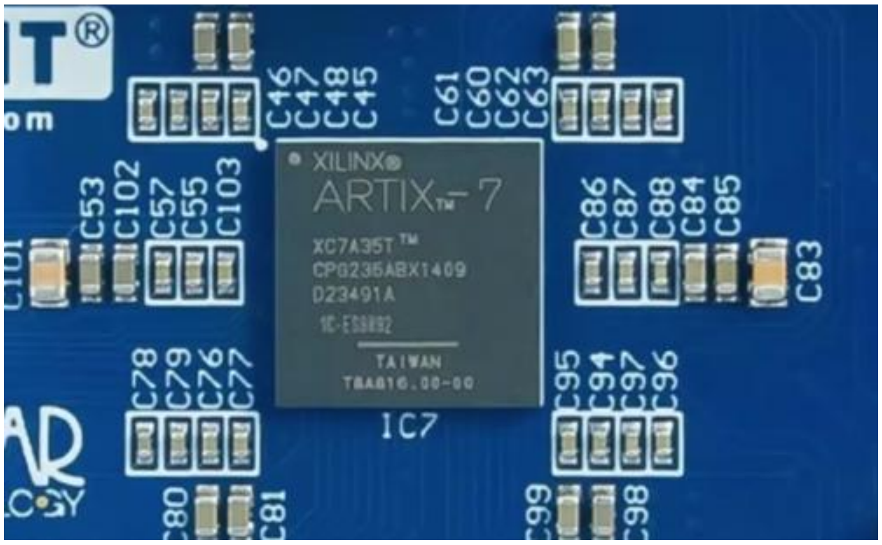
Reset All Filters

Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceivers	GTPE2 Transceivers	PCI Buses
xc7a35tcpg236-1	236	106	20800	41600	50	90	2	2	1
xc7a50tcpg236-1	236	106	32600	65200	75	120	2	2	1

< Back | Next > | Finish | Cancel

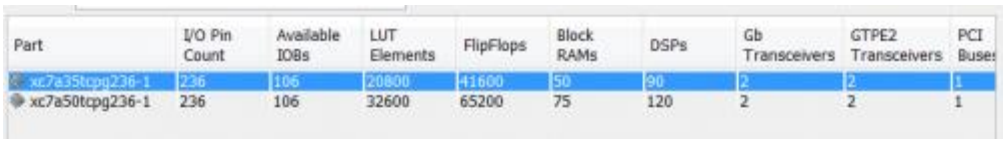
You'll find all of the information you need on the Artix-7 chip on your board.



To find our board set the following filters

Family: Artix-7
Sub-Family: Artix-7
Package: cpg236
Speed grade: -1

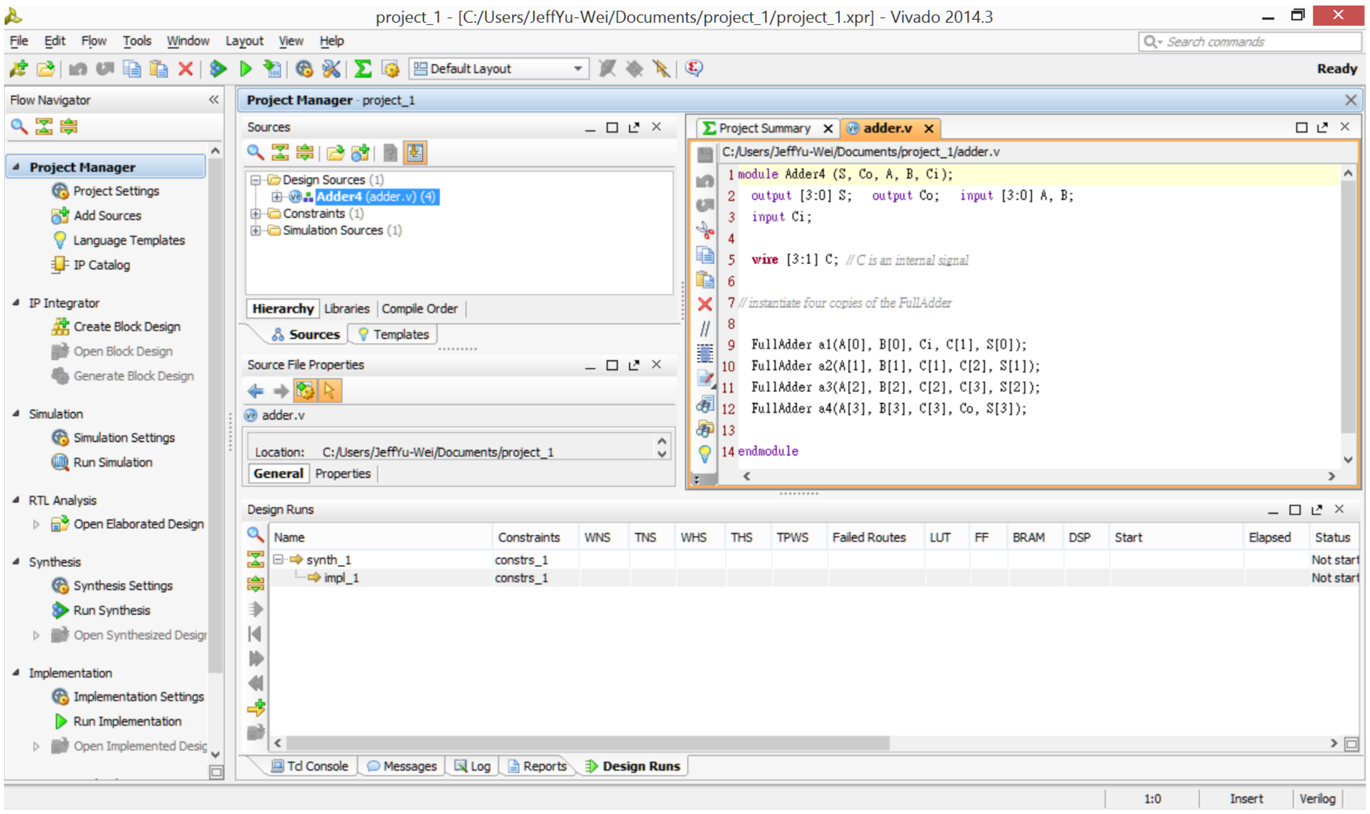
Then choose the top part labeled “xc7a35tcpg236-1” and click “Next” and then “Finish”.



Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceivers	GTPE2 Transceivers	PCI Buses
xc7a35tcpg236-1	236	106	20800	41600	50	90	2	2	1
xc7a50tcpg236-1	236	106	32600	65200	75	120	2	2	1

This will create your project and bring you to the Vivado project home page.

On the home page, you will see the files that you imported earlier in the “Sources” box.



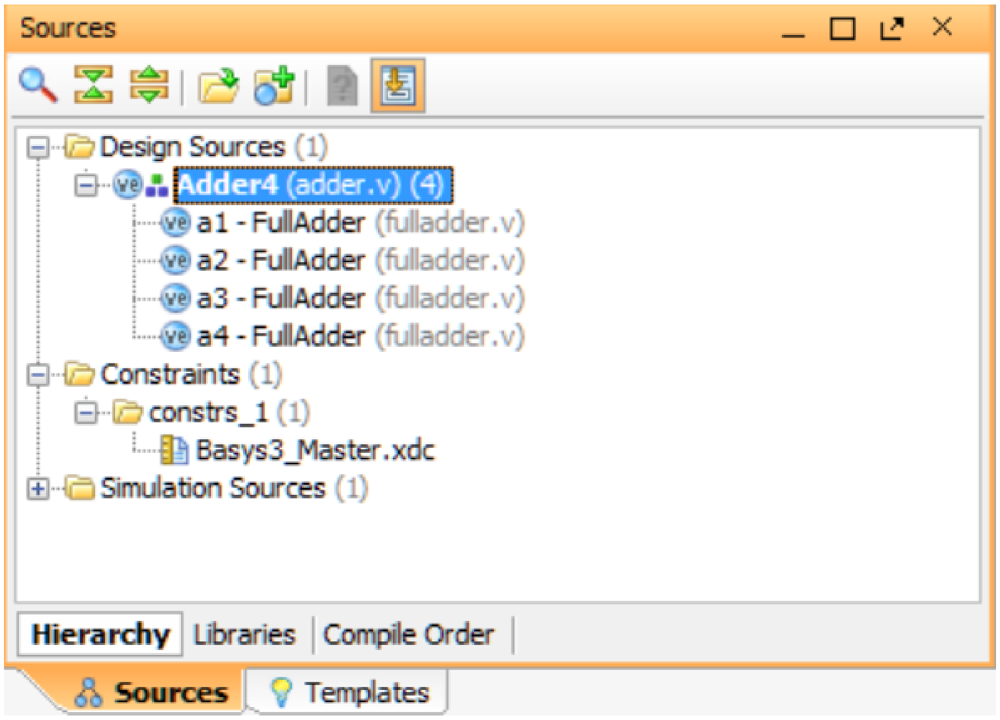
project_1 - [C:/Users/JeffYu-Wei/Documents/project_1/project_1.xpr] - Vivado 2014.3

C:/Users/JeffYu-Wei/Documents/project_1/adder.v

```
module Adder4 (S, Co, A, B, Ci);
  output [3:0] S;   output Co;   input [3:0] A, B;
  input Ci;

  wire [3:1] C;  // C is an internal signal

// instantiate four copies of the FullAdder

  FullAdder a1(A[0], B[0], Ci, C[1], S[0]);
  FullAdder a2(A[1], B[1], C[1], C[2], S[1]);
  FullAdder a3(A[2], B[2], C[2], C[3], S[2]);
  FullAdder a4(A[3], B[3], C[3], Co, S[3]);

endmodule
```

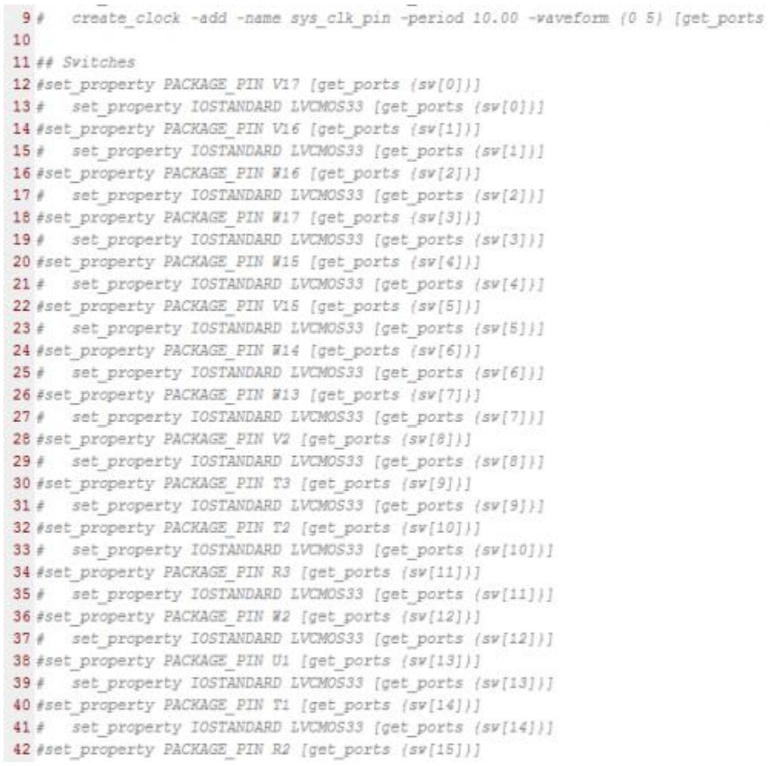
Name	Constraints	Status
synth_1	constrs_1	Not start
impl_1	constrs_1	Not start

Double clicking a file will open it in the window to the right. “Adder4” is the top module for this tutorial we will be running. Clicking the [+] button will reveal the lower level modules used in it.



2. Assigning I/O Pins

Before we run our program, we must first map the signals to pins using the Basys3_Master.xdc file we imported. To do this, we will open Basys3_Master.xdc. Inside this file, we will see how Vivado maps signals to pins. Each line should be commented out at this point (with #), so it should look something like this.



```
#    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports

## Switches
#set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
#set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
#set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
#set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
#set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
#set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
#set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
#set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
#set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
#set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
#set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
#set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]
#set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[12]}]
#set_property PACKAGE_PIN U1 [get_ports {sw[13]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[13]}]
#set_property PACKAGE_PIN T1 [get_ports {sw[14]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {sw[14]}]
#set_property PACKAGE_PIN R2 [get_ports {sw[15]}]
```

First, we want to make sure our signal names match the ones in the .xdc file. This can be confirmed by comparing the signal name in the .xdc file with the signal name in the top module. ***You may need to modify the signal name in the xdc file to match your code. Also, note these are case sensitive***.

- For this tutorial, we will use switches SW7-SW0 to feed our 4-bit A and B inputs, LEDs LD3-LD0 to display the Sum output, BTNU to feed the Carry In, LED4 will display the Carry Out signal. Note that the actual pin names as well as the components are listed directly on the Basys3 Board. They are also available in the Basys3 Board User Manual.

- Now that we know the I/O devices that we want to use to test our design, we need assign them to FPGA pins. The pins assignment for all pins in this tutorial is shown below. Note again that the pin assignment may be different for other future lab pin assignments.

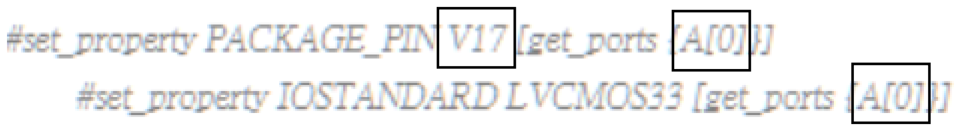
- Although we are not using a CLK in this design, other labs using will use the onboard 100MHz clock located at pin W5 on the board.

I. Tutorial Pin Assignment

Signal Name	Pin Assignment
Ci	T18 (BTNU)
Co	W18 (LD4)
S(3)	V19 (LD3)
S(2)	U19 (LD2)
S(1)	E19 (LD1)
S(0)	U16 (LD0)
A(3)	W17 (SW3)
A(2)	W16 (SW2)
A(1)	V16 (SW1)
A(0)	V17 (SW0)
B(3)	W13 (SW7)
B(2)	W14 (SW6)
B(1)	V15(SW5)
B(0)	W15 (SW4)

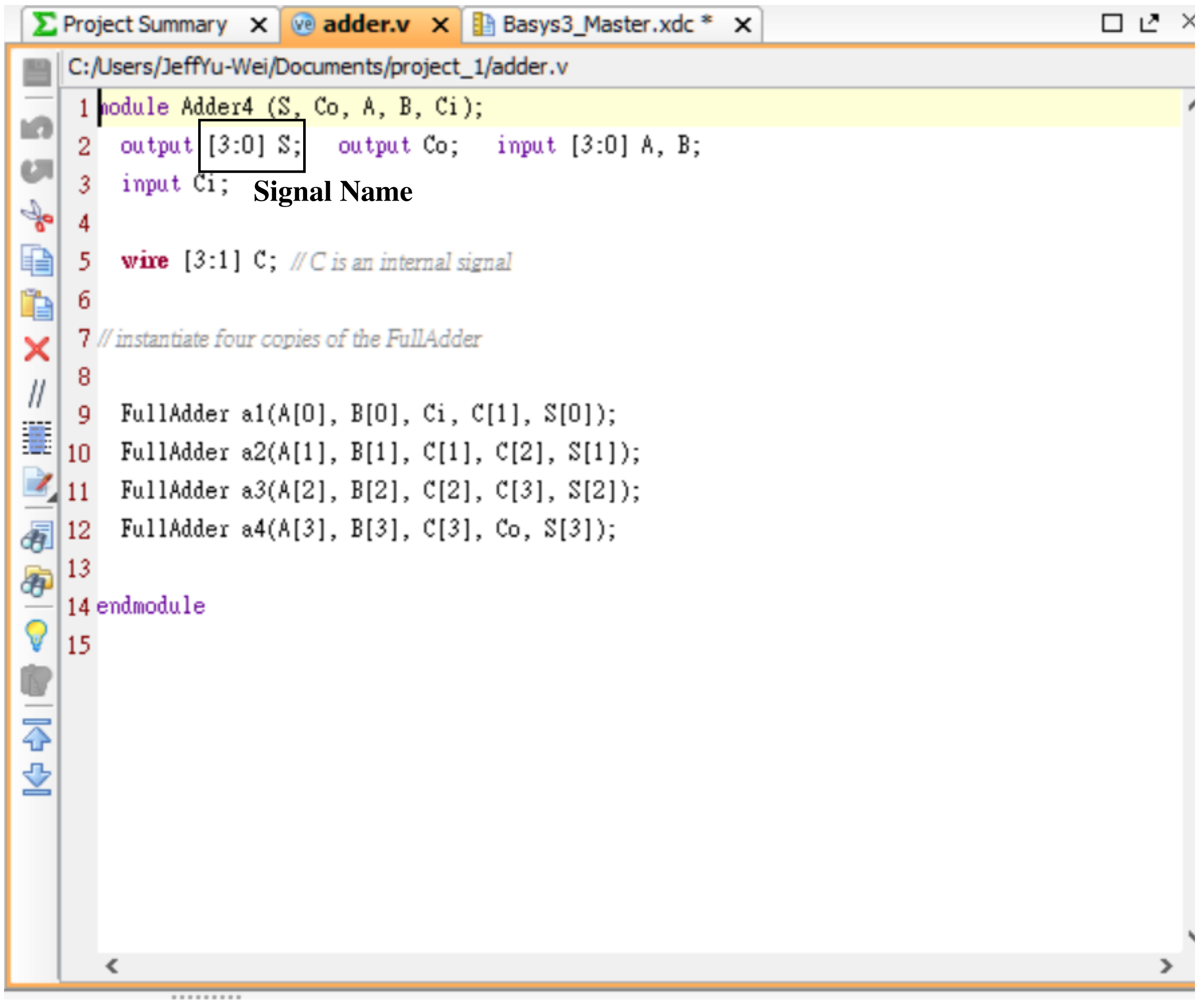
II. .xdc file

Pin name on the board **Signal name in code**



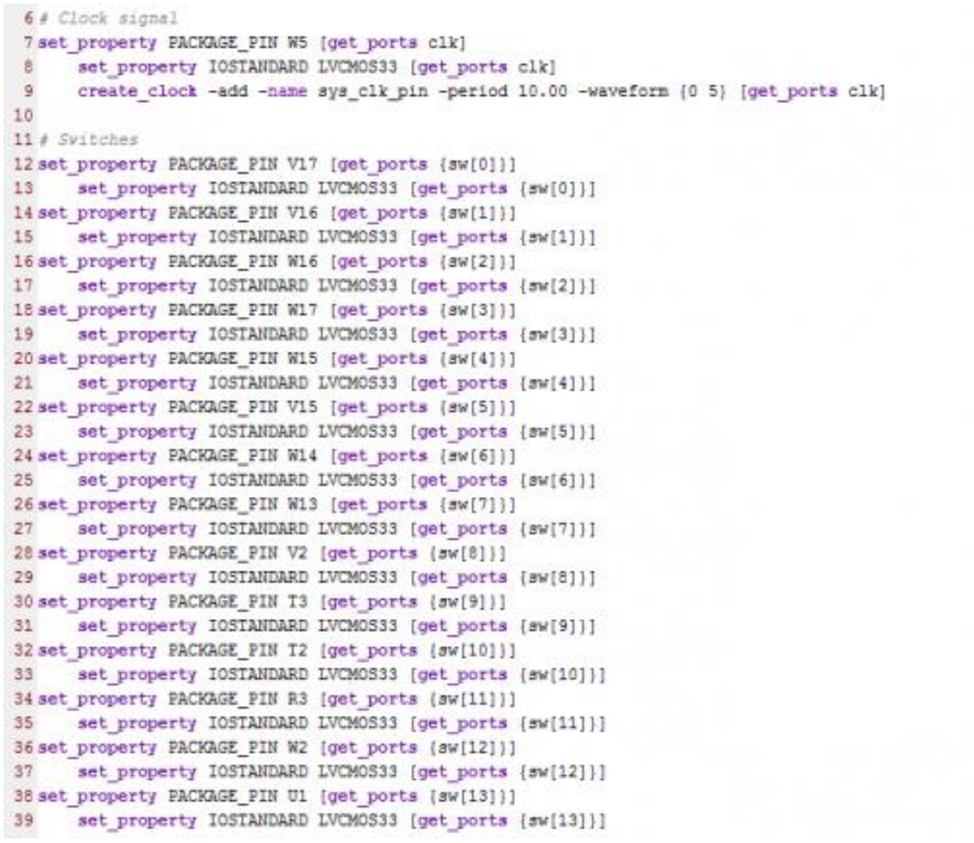
```
#set_property PACKAGE_PIN V17 [get_ports {A[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {A[0]}]
```

III. Top module file



```
module Adder4 (S, Co, A, B, Ci);
  output [3:0] S;   output Co;   input [3:0] A, B;
  input Ci;

  wire [3:1] C; // C is an internal signal

// instantiate four copies of the FullAdder

  FullAdder a1(A[0], B[0], Ci, C[1], S[0]);
  FullAdder a2(A[1], B[1], C[1], C[2], S[1]);
  FullAdder a3(A[2], B[2], C[2], C[3], S[2]);
  FullAdder a4(A[3], B[3], C[3], Co, S[3]);

endmodule
```

Once these are confirmed, we will uncomment whichever constraints we are actually using in the .xdc file. These constraints can be uncommented by selecting the lines of the signals we are using and un-toggling the comments (Ctrl+/). In this case, we are using clk, btnU, sw[0] through sw[7], led[0] through led[7]. Go through the xdc file and uncomment the lines corresponding to these signals.



```
# Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
    set_property IOSTANDARD LVCMOS33 [get_ports clk]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

# Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]
set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[12]}]
set_property PACKAGE_PIN U1 [get_ports {sw[13]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[13]}]
```

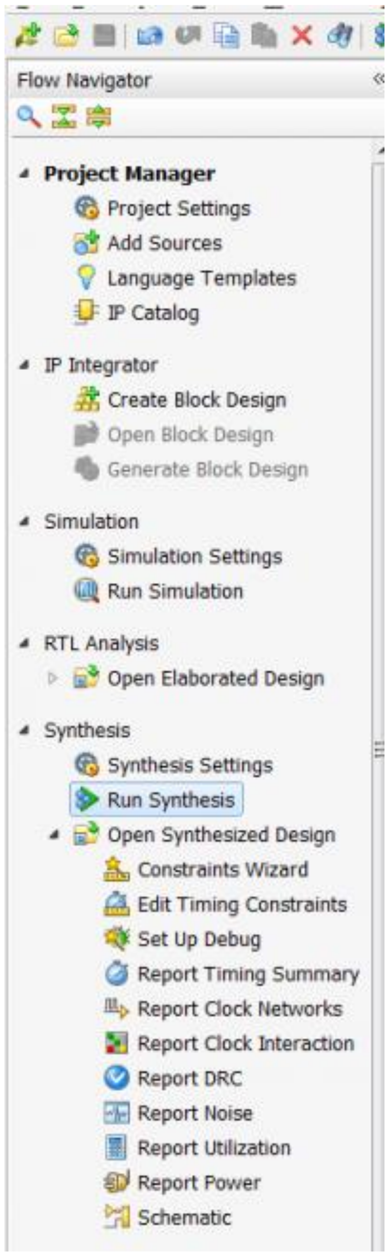
After uncommenting the xdc file, save it and we can start programming your Basys3.

Programming the Basys3

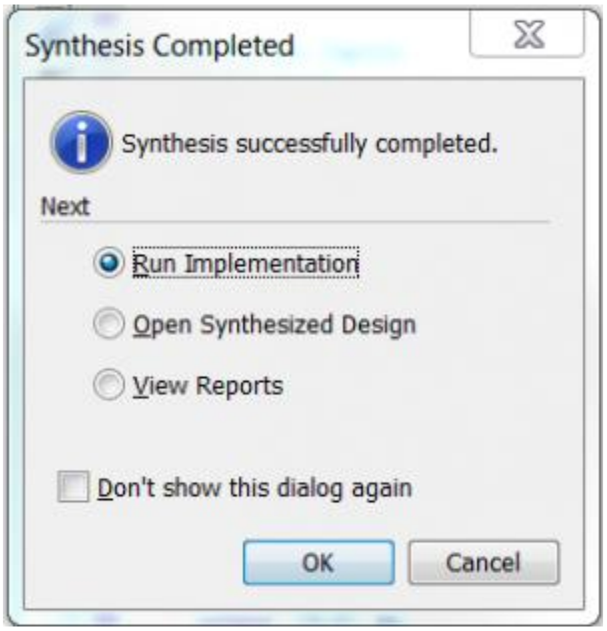
1. Synthesizing

Note that only a subset of the Verilog language is synthesizable, so a design that simulates properly is not necessarily synthesizable. For some synthesis tips, you can check the Tips for Writing Synthesizable Code document, which is available on Canvas.

To begin, we will run the synthesis by clicking "Run Synthesis" beneath Synthesis in the Flow Navigator on the left side of Vivado.

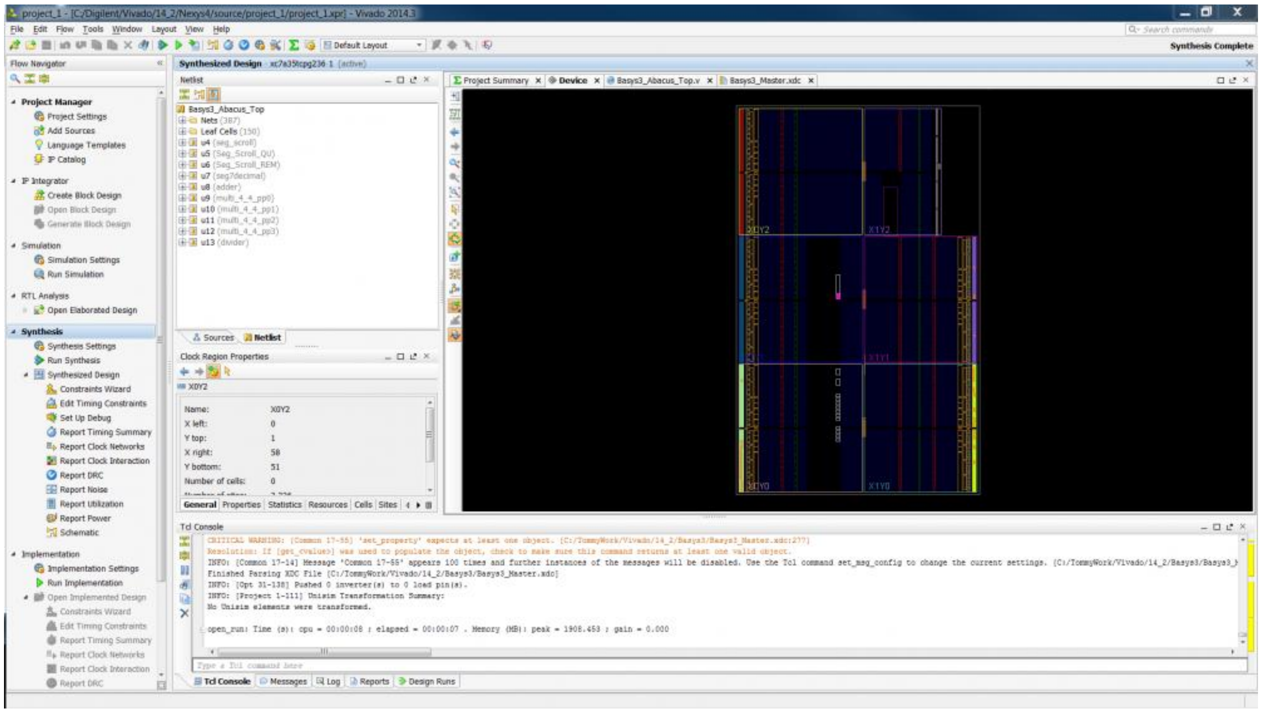


When it is done synthesizing your project, you will see the Synthesis Completed window.



Click “Open Synthesized Design” and then press Ok.

You should now see this window



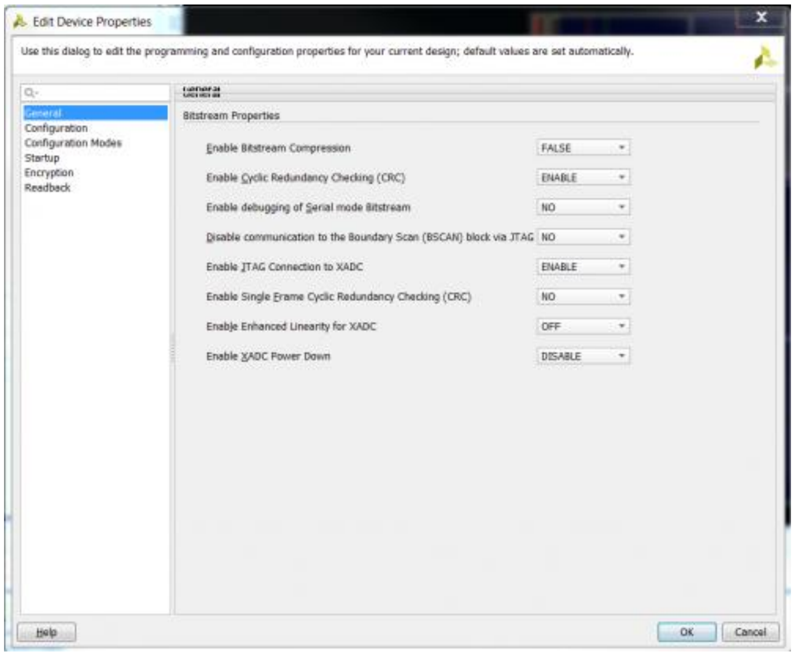
```
CRITICAL WARNING: [Common 17-55] 'set_property' expects at least one object. [C:/TommyWork/Vivado/14_2/Basys3/Basys3_Master.xdc:277]
Resolution: If [get_<value>] was used to populate the object, check to make sure this command returns at least one valid object.
INFO: [Common 17-14] Message 'Common 17-55' appears 100 times and further instances of the messages will be disabled. Use the Tcl command set_msg_config to change the current settings. [C:/TommyWork/Vivado/14_2/Basys3/Basys3_
Finished Parsing XDC File [C:/TommyWork/Vivado/14_2/Basys3/Basys3_Master.xdc]
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
INFO: [Project 1-111] Unisim Transformation Summary:
No Unisim elements were transformed.

open_run: Time (s): cpu = 00:00:08 ; elapsed = 00:00:07 . Memory (MB): peak = 1908.453 ; gain = 0.000
```

You may check the RTL schematic to verify that the hardware generated matches your expectations from the code. To see the schematic, click “Synthesized Design” under Synthesis in the Flow Navigator and select “Schematic”.

2. (Optional) Setting the device properties

To improve programming speed, in the main toolbar select Tools>Edit Device Properties…

Under General, set Enable Bitsream Compression to “TRUE”.



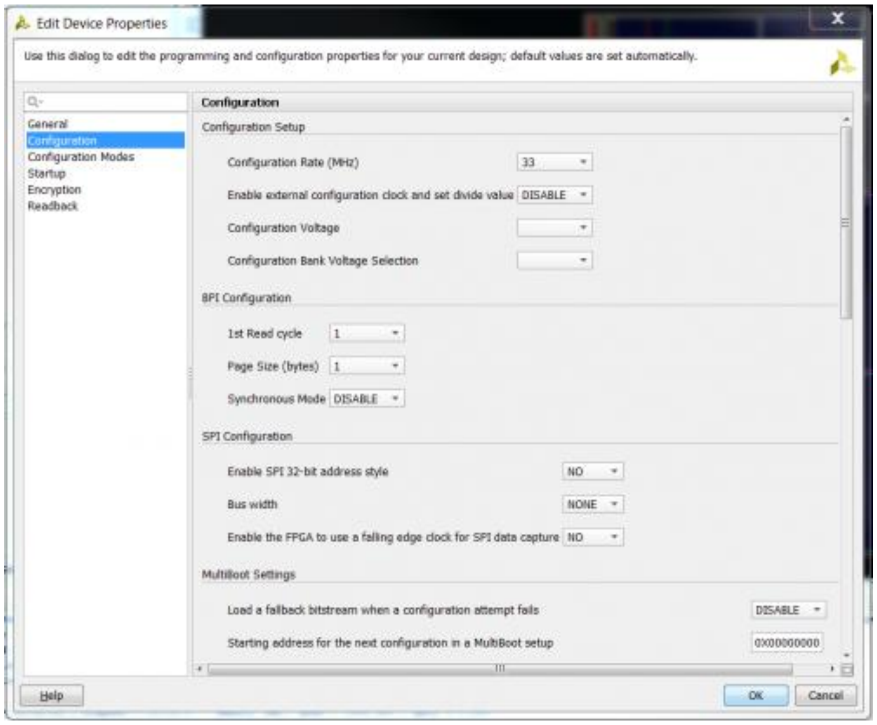
Use this dialog to edit the programming and configuration properties for your current design; default values are set automatically.

General

Bitstream Properties

Enable Bitstream Compression	FALSE
Enable Cyclic Redundancy Checking (CRC)	ENABLE
Enable debugging of Serial mode Bitstream	NO
Disable communication to the Boundary Scan (BSCAN) block via JTAG	NO
Enable JTAG Connection to XADC	ENABLE
Enable Single Frame Cyclic Redundancy Checking (CRC)	NO
Enable Enhanced Linearity for XADC	OFF
Enable XADC Power Down	DISABLE

Under Configuration, set Configuration Rate (Mhz) to 33.



Use this dialog to edit the programming and configuration properties for your current design; default values are set automatically.

Configuration

Configuration Setup

Configuration Rate (MHz)	33
Enable external configuration clock and set divide value	DISABLE
Configuration Voltage	
Configuration Bank Voltage Selection	

BPI Configuration

1st Read cycle	1
Page Size (bytes)	1
Synchronous Mode	DISABLE

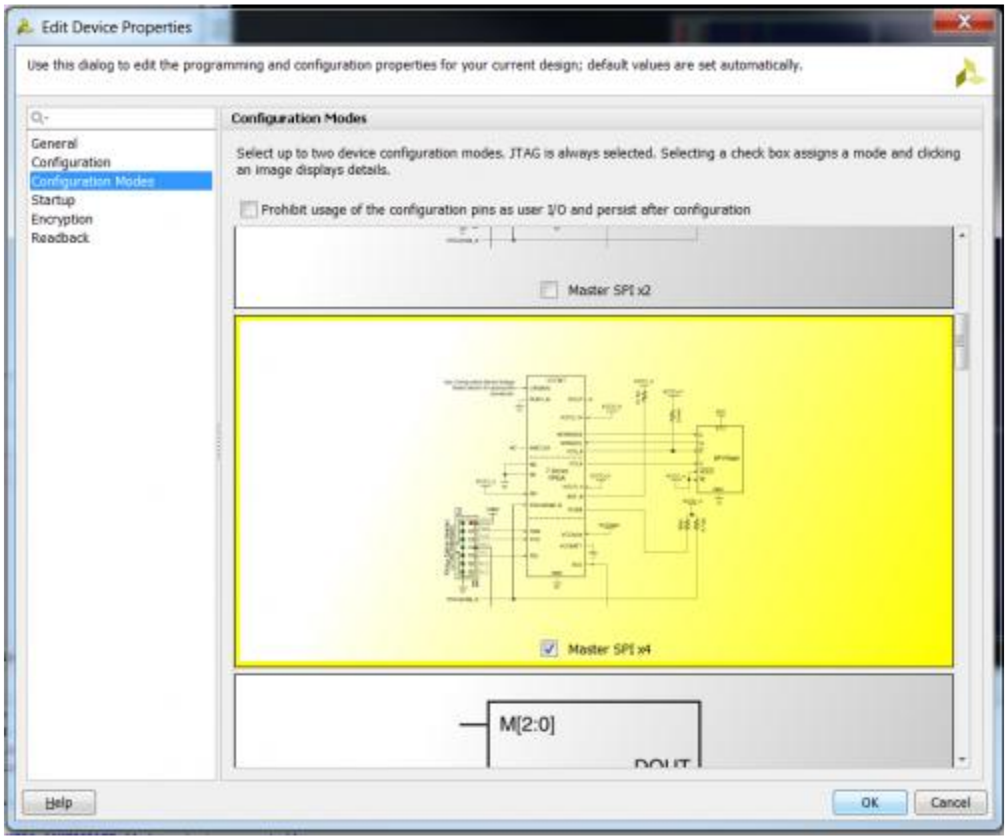
SPI Configuration

Enable SPI 32-bit address style	NO
Bus width	NONE
Enable the FPGA to use a falling edge clock for SPI data capture	NO

Multiboot Settings

Load a fallback bitstream when a configuration attempt fails	DISABLE
Starting address for the next configuration in a MultiBoot setup	0X00000000

Under Configuration Modes, select Master SPI x4



Use this dialog to edit the programming and configuration properties for your current design; default values are set automatically.

Configuration Modes

Select up to two device configuration modes. JTAG is always selected. Selecting a check box assigns a mode and clicking an image displays details.

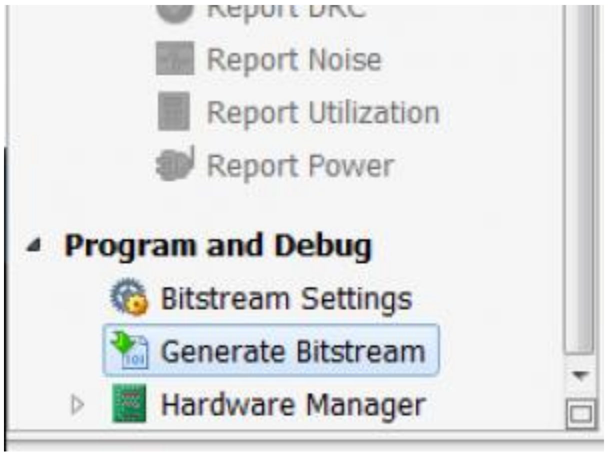
☐ Prohibit usage of the configuration pins as user I/O and persist after configuration

☑ Master SPI x4

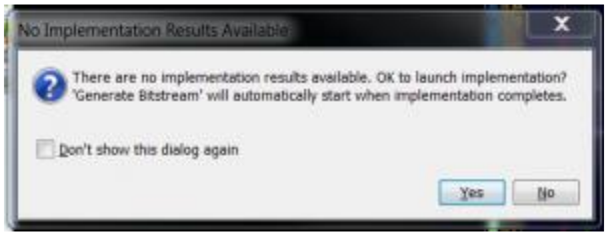
Press Ok, save your synthesized design (Ctrl+S).

3. Generating Bitstream (.bit) file

Click “Generate Bitstream” in the Flow Navigator on the left side.



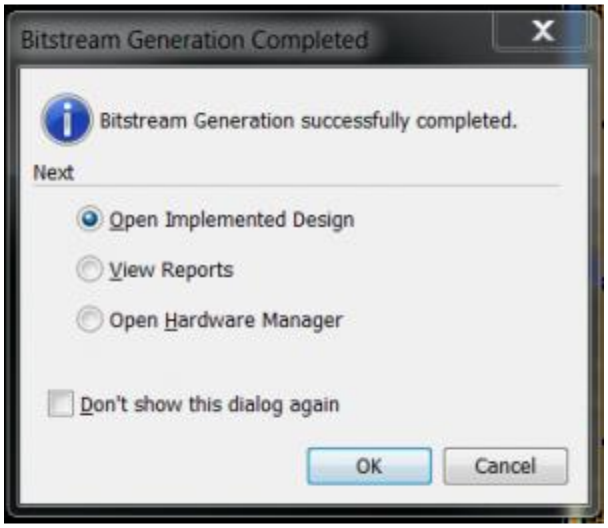
This will open a box stating that you have not implemented your design.


No Implementation Results Available

There are no implementation results available. OK to launch implementation?
'Generate Bitstream' will automatically start when implementation completes.

Don't show this dialog again
Yes
No

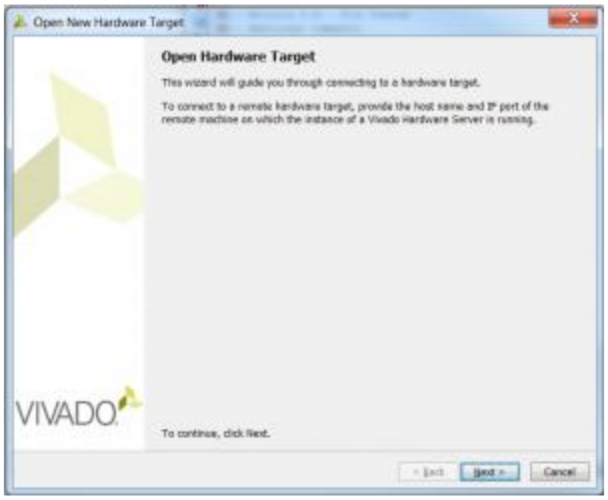
Click “Ok” and Vivado will generate your .bit file and will show you this box.



Bitstream Generation successfully completed.

Click Open Hardware Manager and click OK.

You will see this screen.



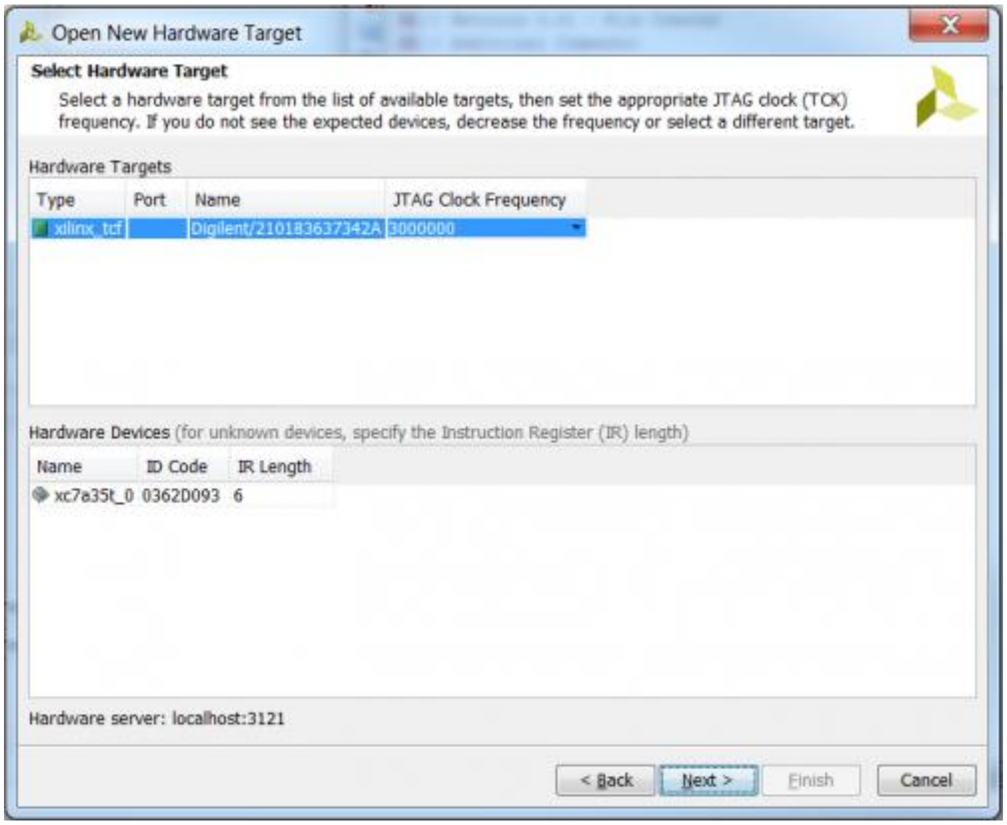
Open Hardware Target

This wizard will guide you through connecting to a hardware target.

To connect to a remote hardware target, provide the host name and IP port of the remote machine on which the instance of a Vivado Hardware Server is running.

To continue, click Next.

At this point, make sure your Basys 3 is plugged in via USB and turned on. Now click Next twice and you will see this screen.


Open New Hardware Target

Select Hardware Target

Select a hardware target from the list of available targets, then set the appropriate JTAG clock (TCK) frequency. If you do not see the expected devices, decrease the frequency or select a different target.

Hardware Targets

Type	Port	Name	JTAG Clock Frequency
xilinx_tcf		Digilent/210183637342A	3000000

Hardware Devices (for unknown devices, specify the Instruction Register (IR) length)

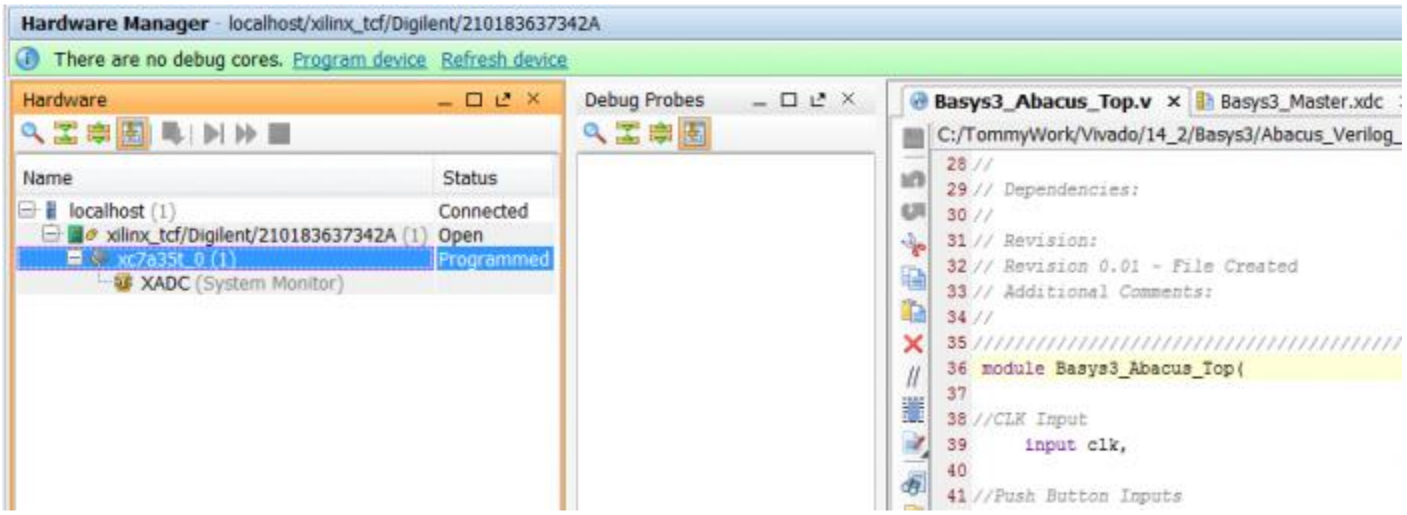
Name	ID Code	IR Length
xc7a35t_0	0362D093	6

Hardware server: localhost:3121

Set the JTAG Clock Frequency to 30000000, select the device, and click “Next” followed by “Finish”.

4. Programming the Basys3 board using a .bit file

First, make sure that the jumper JP1 is in the JTAG position. You should see something like this.


Hardware Manager - localhost/xilinx_tcf/Digilent/210183637342A

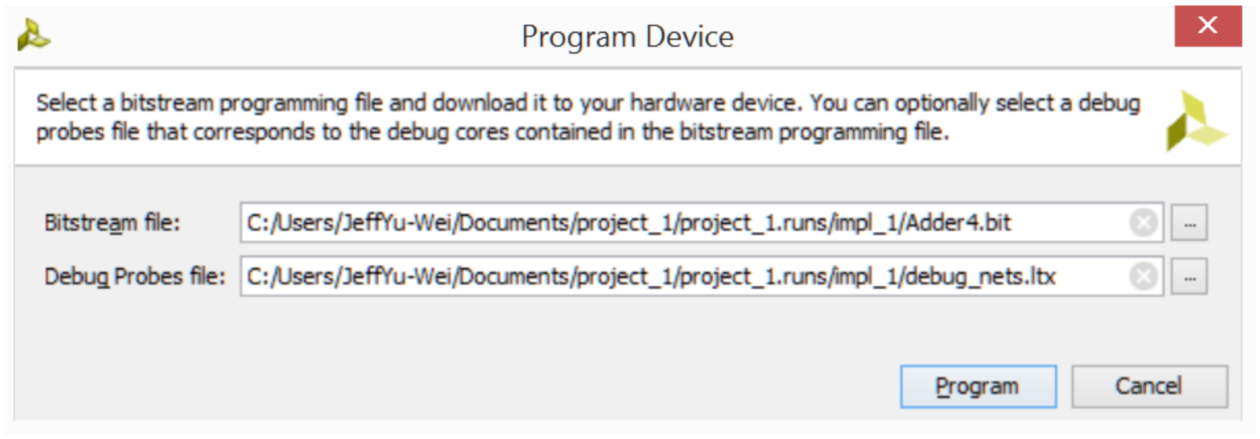
There are no debug cores. Program device Refresh device

Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/210183637342A (1)	Open
xc7a35t_0 (1)	Programmed
XADC (System Monitor)	

C:/TommyWork/Vivado/14_2/Basys3/Abacus_Verilog_

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////
module Basys3_Abacus_Top(

//CLK Input
    input clk,

//Push Button Inputs
```

Click “Program device” (in the green bar) then xc7a35t_0, select your .bit file in the bitstream file box, and click Program.



Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: C:/Users/JeffYu-Wei/Documents/project_1/project_1.runs/impl_1/Adder4.bit

Debug Probes file: C:/Users/JeffYu-Wei/Documents/project_1/project_1.runs/impl_1/debug_nets.ltx

This will program your Basys3 through the JTAG connector (USB connection).

Vivado will now erase the old configuration file, and reprogram your Basys3 with the demo file. From now on, when you power cycle the Basys3, the demo will load at startup.

5. Testing the Example Design

Now that you have downloaded your design to the board, you need to make sure that it works properly by testing it for correct functionality. Don't turn the power off or reset the board. We can test the circuit for this tutorial by changing inputs A (switches 0 to 3) and B (switches 4 to 7); and pushing Cin (btnU). Observe the output of the Adder on the LEDs.