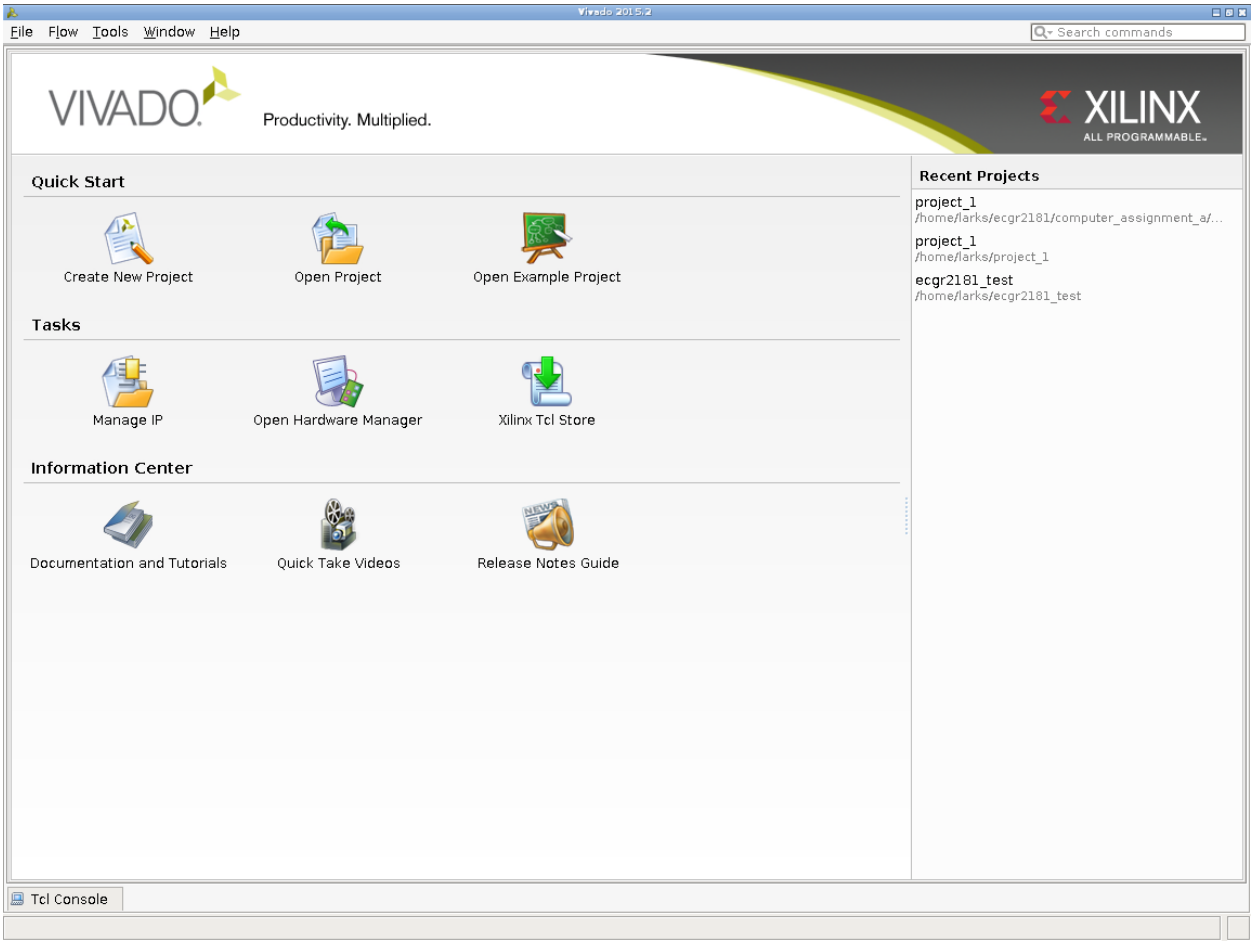
Vivado Walkthrough
ECGR 2181 - Fall 2015

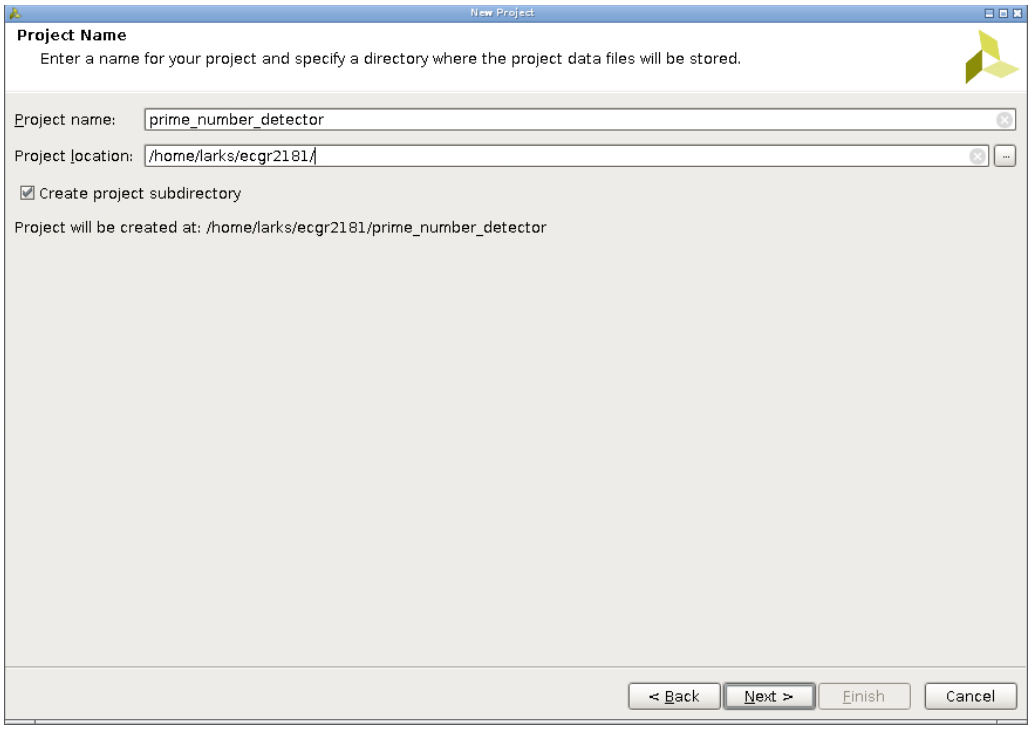
Intro
In this walkthrough we're going to go through the process of creating a project, adding sources, writing vhdl, simulating the design, and creating a bitsteam (as if you were going to put it on an FPGA) of a prime number detector. Vivado can be pretty complex if you just stick with the basics, it's pretty simple.

Creating a Project
First open Vivado. You should have something like this in front of you.



Select the "Create New Project" button.

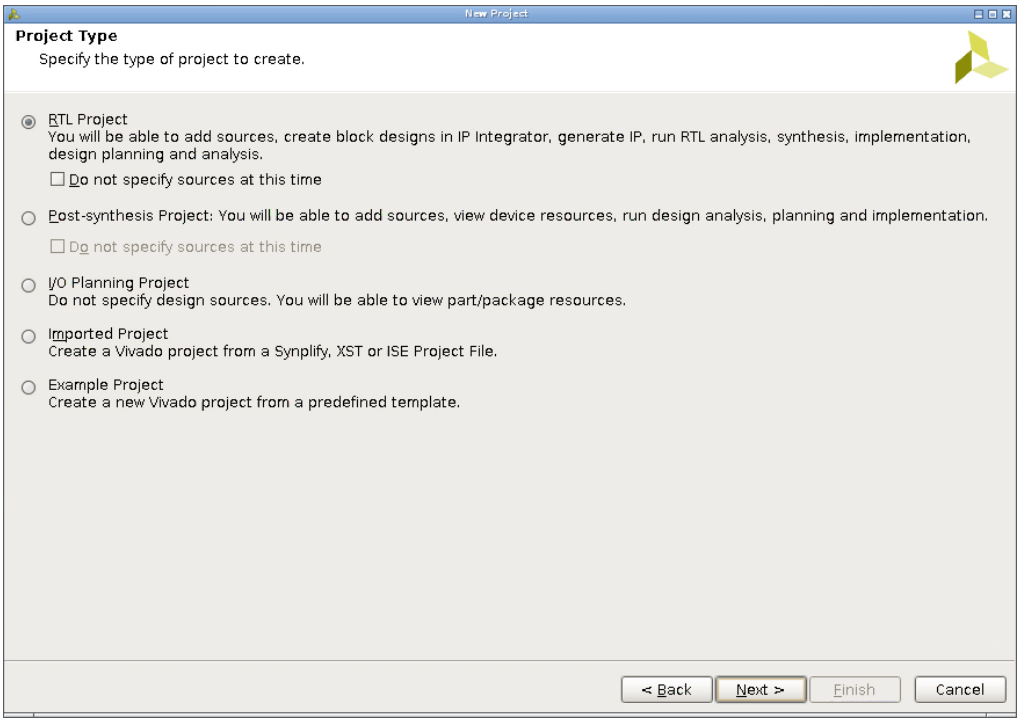
The first window just says that it's a wizard etc and you can press next. The next window you want to input your project name and the path to the project directory.



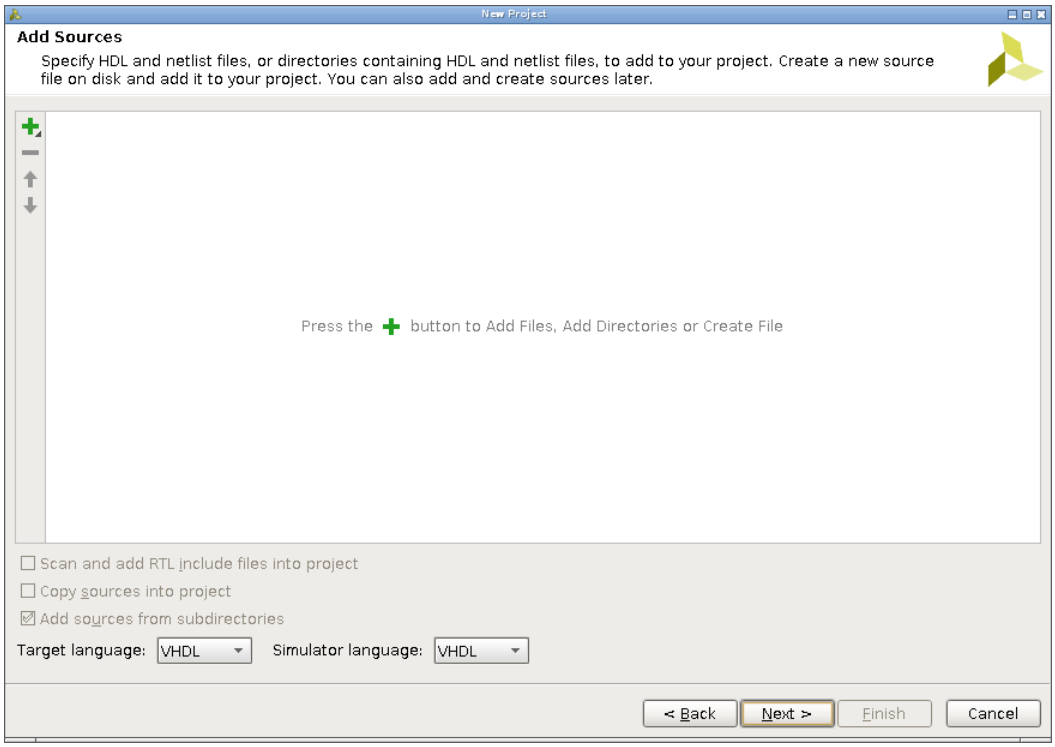
I have Linux directory paths but your's should look something like this. **Name your project "prime_number_detector" and make sure you've enable "Create project subdirectory."** Remember to always give your projects clear meaningful names and make sure that they're placed somewhere meaningful. As a side note, it's generally best to not have white space in directory or file names. In the past spaces in file names or directories has caused errors with Xilinx tools. It's a good idea to have a ecgr2181 directory and then subdirectories in it for each project/lab/etc like:

ecgr2181\cad1\
ecgr2181\cad2\
ecgr2181\project1\
etc…

Once you've filled out your project name and clicked next, you'll see a screen asking you about about Project Type. **You'll want to select "RTL Project."**

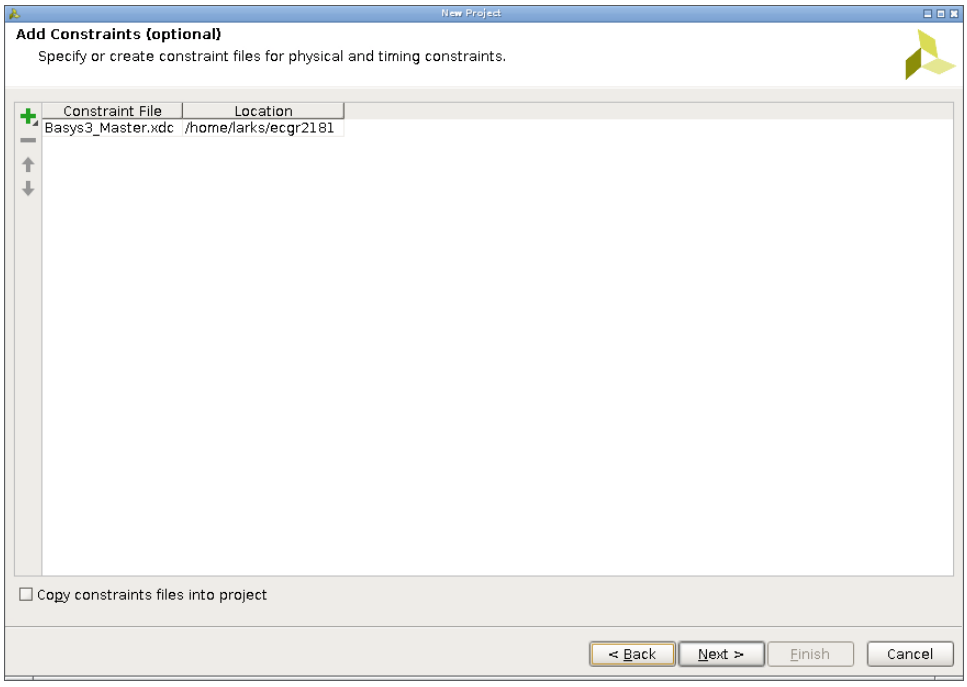


After clicking selecting “RTL Project” and then next, you’ll be at the “Adding Sources” screen. At this point in time, we won’t be adding sources, but **you need to make sure that “Target language” and “Simulator language” are both “VHDL.”**

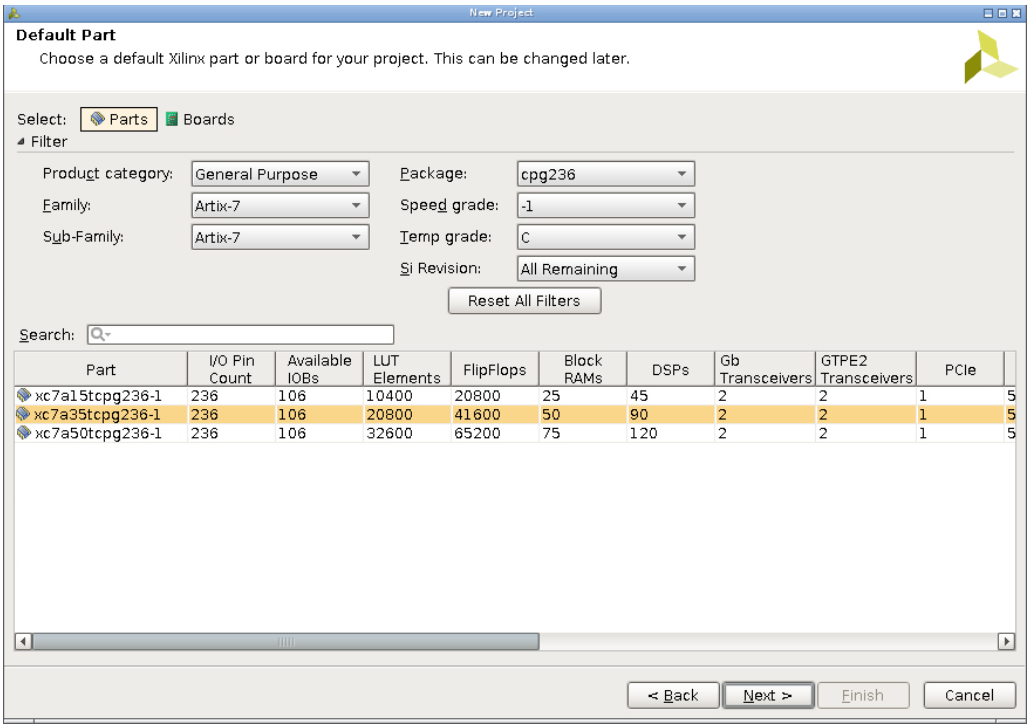


The next screen should be the "Adding Existing IP (optional)" screen which we can just click next.

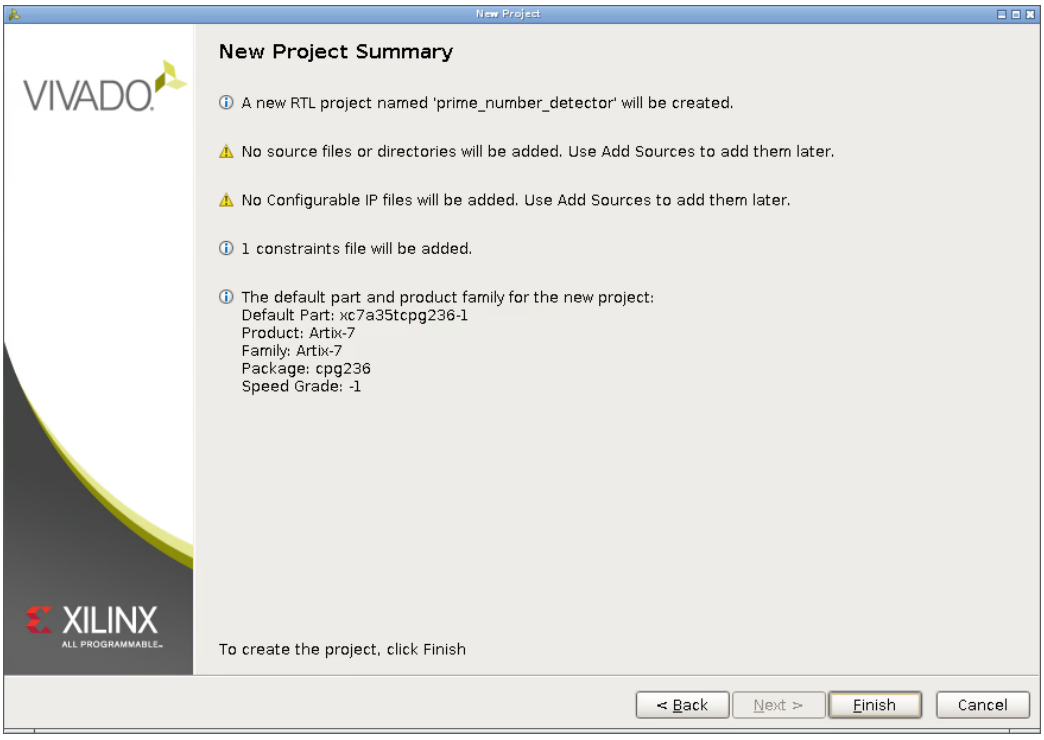
The screen after this this is important if you're going to be putting the design on an FPGA (which we assume we will in this example). **For this, you'll need to go to Digilent's website and get the xdc file for the Basys3.** It'll be in a zip file and you'll need to unzip it. I suggest putting it in the project directory or you can also use the "Copy constraints files into project" checkbox, either one works.



Next you need to select the (FPGA) part the tools will be targeting. The Basys3 uses a XC7A35T-1CPG236C FPGA.



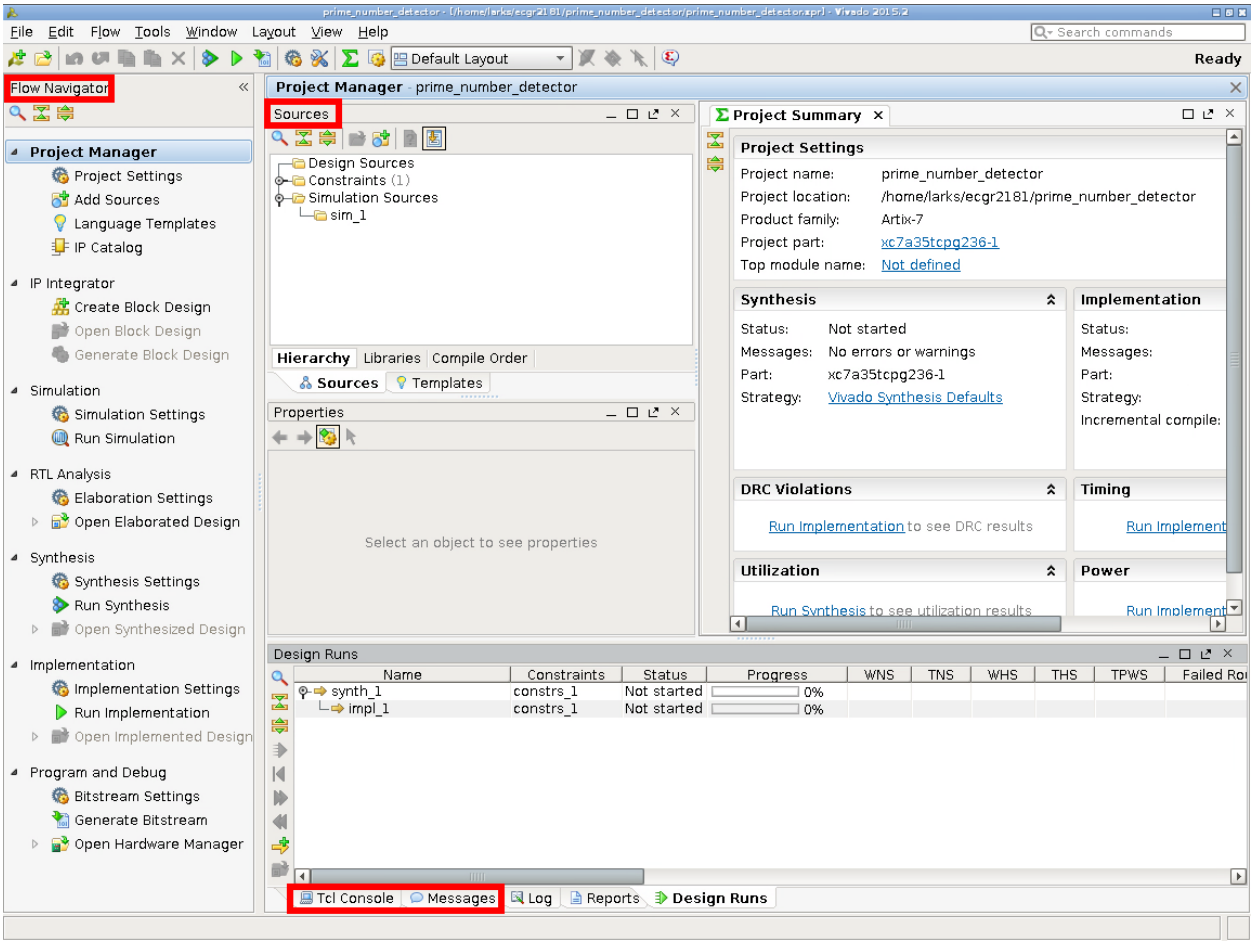
After selecting the right part, click next and that should bring you to a summary window.



Then you're done setting up your new project! Now for the fun stuff.

Vivado Interface Overview

So now that the project is set up, let's look at the Vivado interface some.



There's a couple of key things we need to note:

- Flow Navigator
- Sources
- Tcl Console and Messages
-

In the Flow Navigator, we'll mostly be using: Add Sources, Simulation and Generate Bitstream. There's a lot of intermediate steps, but those are the main ones that we'll be using. I would like to note that Synthesizing a design can help catch errors that aren't syntax related (like libraries missing etc).

In the sources section, I don't think you'll ever need to switch away from the sources tab or the hierarchy sub tab. It's a pretty good way to look at the code and see how multiple source files are related.

The Tcl Console and Messages are good for figuring out what went wrong. Depending on the error I've found useful information in either/or.

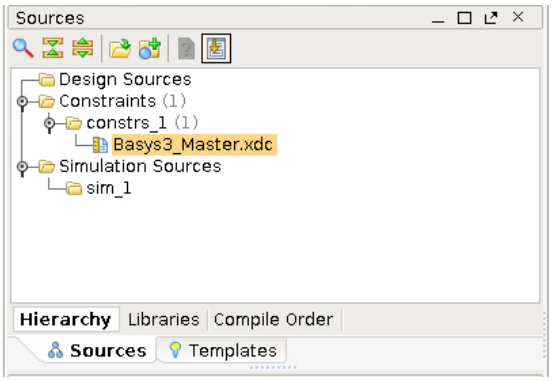
As a note, if there's something you'd prefer to change with Xilinx's built in editor you can access that via "tools" -> "options" in the "general" left vertical tab. There's the "Shortcuts" tab, which you can view/edit shorts for the editor and various other parts of Vivado. (Just an FYI)

We'll learn more about some of the interfaces as we use them in our project.

Writing VHDL/Implementation

Now that we've had a short intro the interface, let's start using it.

First we need to figure which FPGA pins we're using for the top level VHDL file. **The constraint file will be in the Source section under "Constraints (1)" -> "constrs_1 (1)". Double click on it to open the file.**

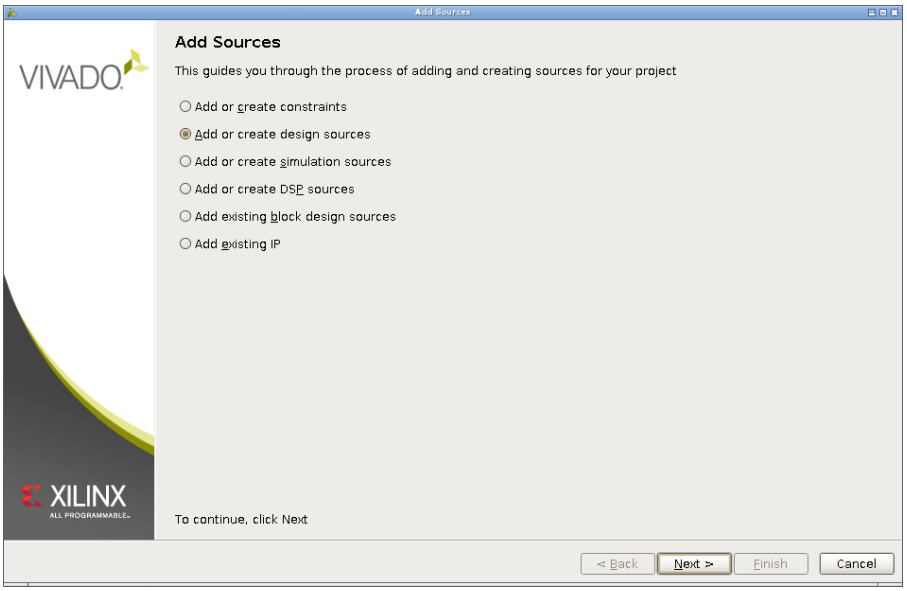


In the xdc file, there's a section for the clock, switches, leds, etc. **We want to uncomment the first 4 switches (0 through 3) and the first led (0).** Afterwards the file should look like:

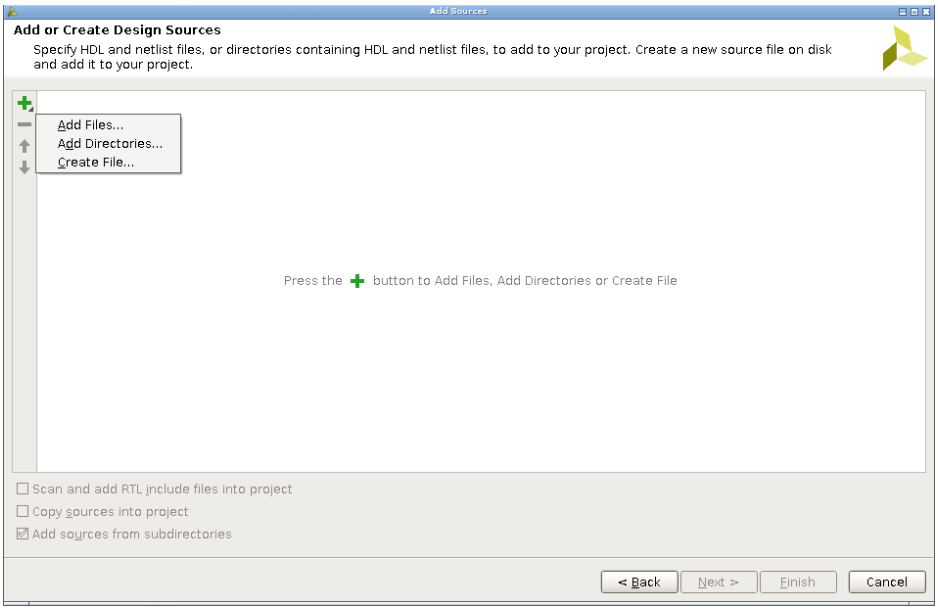
```
10
11 ## Switches
12 set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
13         set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
14 set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
15         set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
16 set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
17         set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
18 set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
19         set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
20 #set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
21         #set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
22 #set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
23         #set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
24 #set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
25         #set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
26 #set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
27         #set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
28 #set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
29         #set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
30 #set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
31         #set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
32 #set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
33         #set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
34 #set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
35         #set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]
36 #set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
37         #set_property IOSTANDARD LVCMOS33 [get_ports {sw[12]}]
38 #set_property PACKAGE_PIN U1 [get_ports {sw[13]}]
39         #set_property IOSTANDARD LVCMOS33 [get_ports {sw[13]}]
40 #set_property PACKAGE_PIN T1 [get_ports {sw[14]}]
41         #set_property IOSTANDARD LVCMOS33 [get_ports {sw[14]}]
42 #set_property PACKAGE_PIN R2 [get_ports {sw[15]}]
43         #set_property IOSTANDARD LVCMOS33 [get_ports {sw[15]}]
44
45
46 ## LEDs
47 set_property PACKAGE_PIN U16 [get_ports {led[0]}]
48         set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
49 #set_property PACKAGE_PIN E19 [get_ports {led[1]}]
50         #set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
```

We could change the names from sw and led but they suit our purposes. In the top level VHDL file they'll be "sw: in std_logic_vector(3 downto 0);" and "led: out std_logic_vector(0 downto 0)". Since we know our ports, let's create a VHDL file. Normally you don't use a vector/bus for a 1 bit value, but since led was part of a bus in this cause, we're just using one bit of that bus. In your own designs just use "std_logic" rather that "std_logic_vector (0 downto 0)".

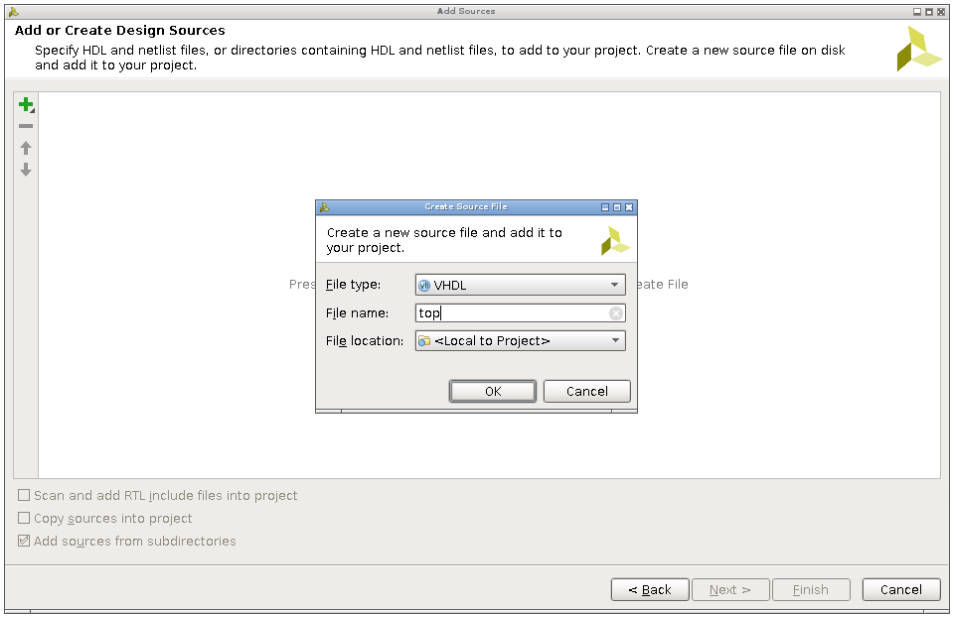
In the "Flow Navigator" Panel, click the "Add Sources" button. You'll want to select "Add or create design sources" in the "Add Sources" window**.**



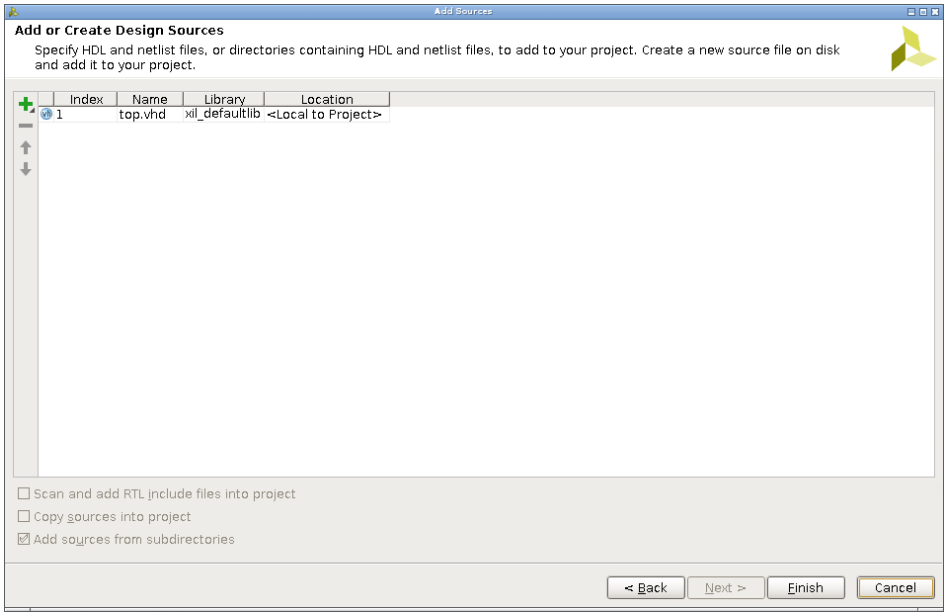
After that you'll see the "Add or Create Design Source" window. There's a green plus symbol which will create a pop up and then **select "Create File..."**



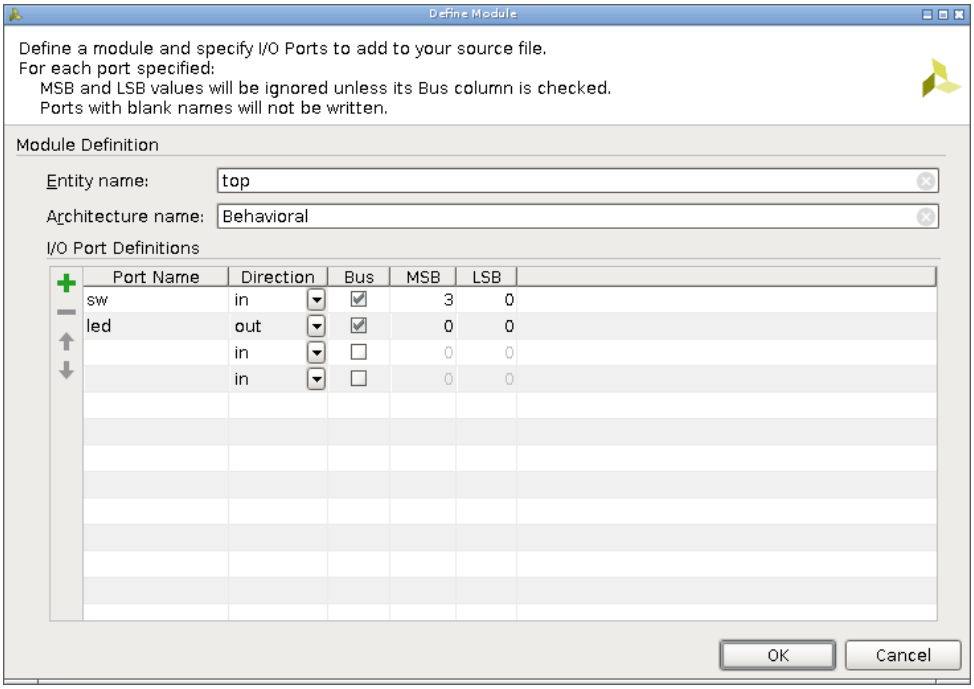
Input your VHDL file name and click ok.



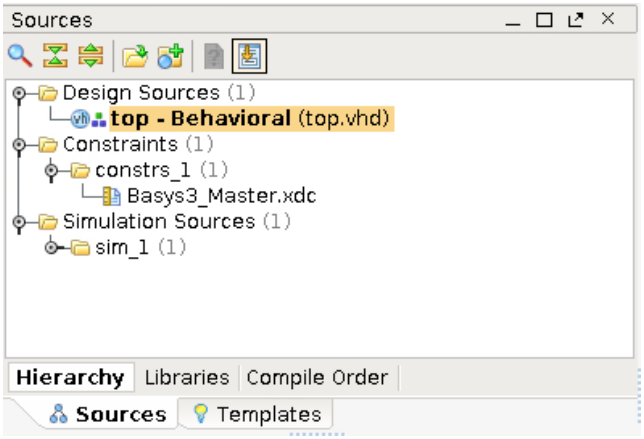
It should look like:



Since we know that **sw will be a 4 bit wide bus and led will be a 0 bit wide bus**, we put that into the VHDL “Module Definition” that is next.



Click ok and you’ll have a vhdl source file in your “Sources” panel.



Now we need to write some VHDL. Since we have a 4 bit input, that means our input range is 0 to 15. The prime numbers in that range are: 2, 3, 5, 7, 9, 11, 13. So the output (led(0)) needs to be higher when the input is equal to those.

The VHDL file from the entity down (excluding the libraries) is:

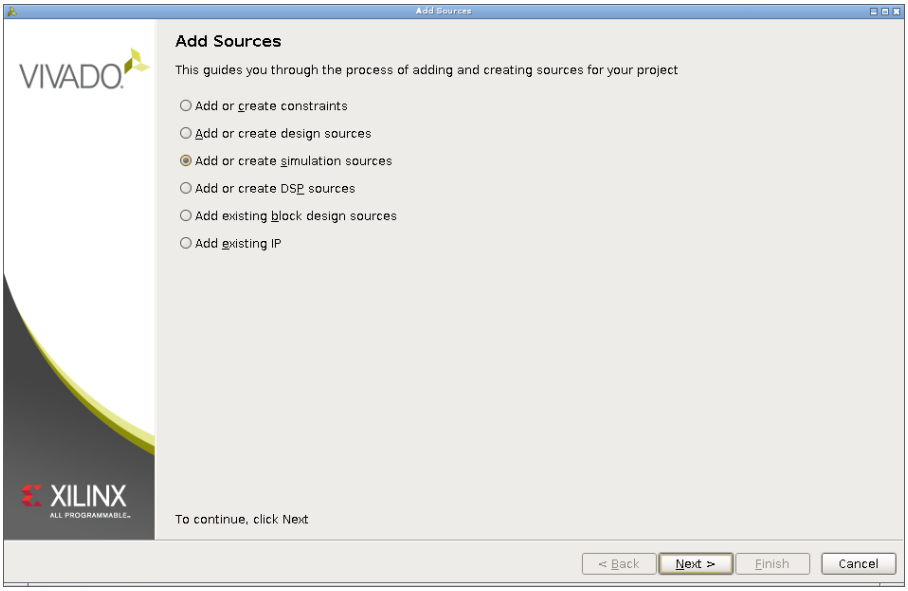
```
34 entity top is
35     Port ( sw : in STD_LOGIC_VECTOR (3 downto 0);
36            led : out STD_LOGIC_VECTOR (0 downto 0));
37 end top;
38
39 architecture Behavioral of top is
40
41 begin
42
43         led(0) <=
44                 '1' when (sw = "0010") else      -- 2
45                 '1' when (sw = "0011") else      -- 3
46                 '1' when (sw = "0101") else      -- 5
47                 '1' when (sw = "0111") else      -- 7
48                 '1' when (sw = "1001") else      -- 9
49                 '1' when (sw = "1011") else      -- 11
50                 '1' when (sw = "1101") else      -- 13
51                 '0';
52
53 end Behavioral;
```

I run the synthesis tool (in the “Flow Navigator” as “Run Synthesis”) after this to make sure there weren’t any errors in the code.

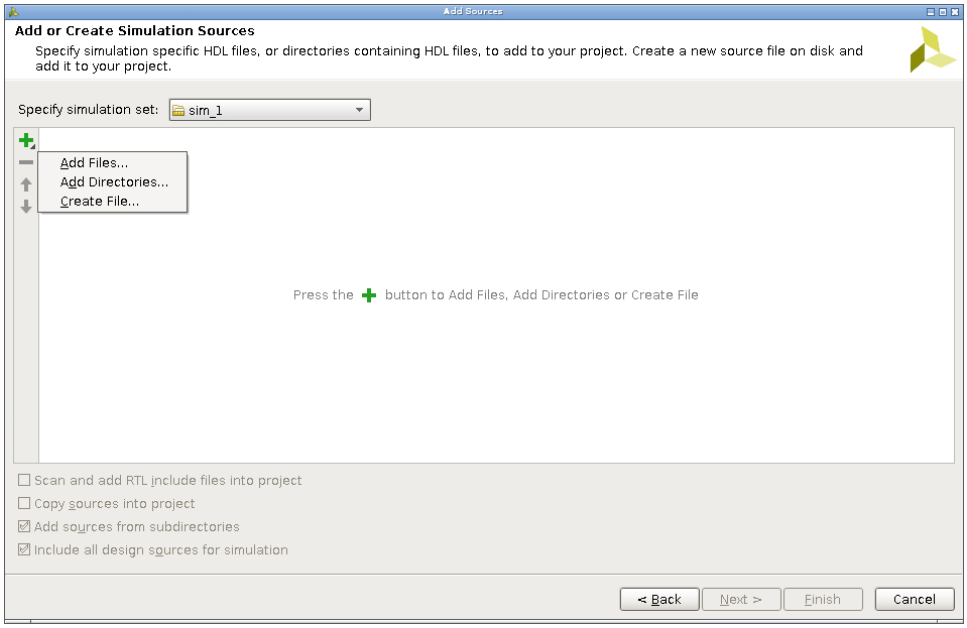
Creating a Testbench

Test benches are extremely using for debugging circuit logic. Hardware isn’t like software where you can easily run a debugger while it runs, so logic that isn’t depending on complex I/O, simulation can useful for testing. Although if you don’t use the idioms and try to be “creative” the simulation results can differ from implementation results (aka what gets loaded to the FPGA). Just a note that simulation isn’t the final test, running it on the FPGA is.

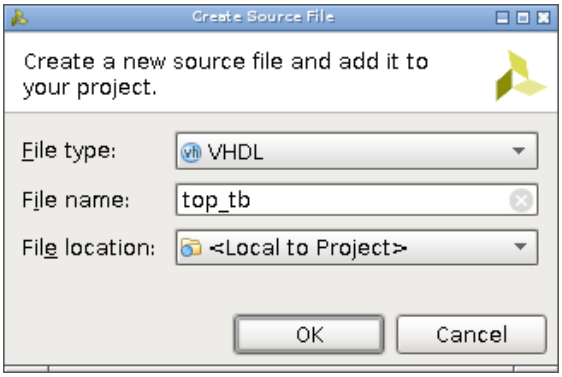
First click on “Add sources” and select “Add or create simulation source.”



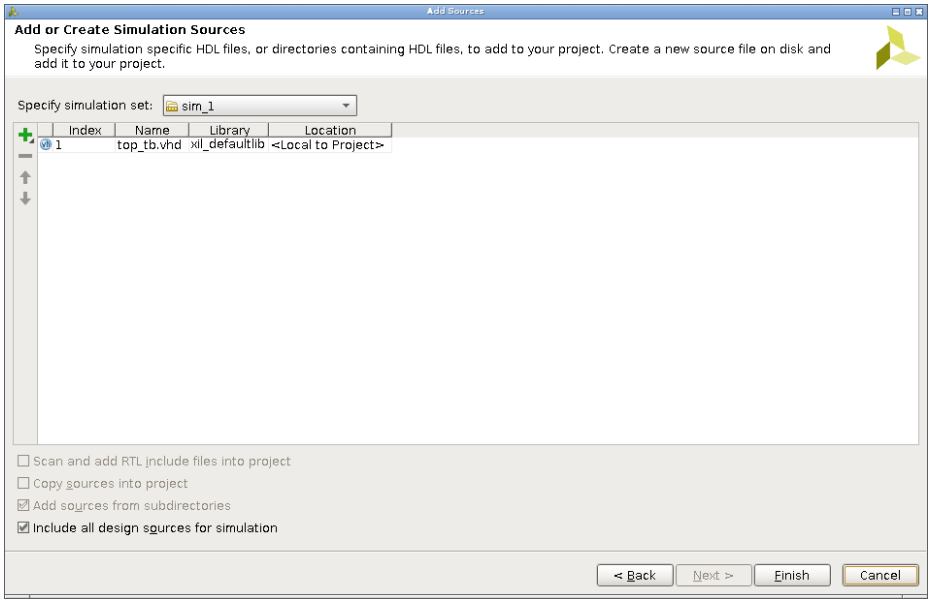
Just like adding a source file, **you need to click the green plus symbol and select “Create File…”** on the “Create Simulation Sources” window.



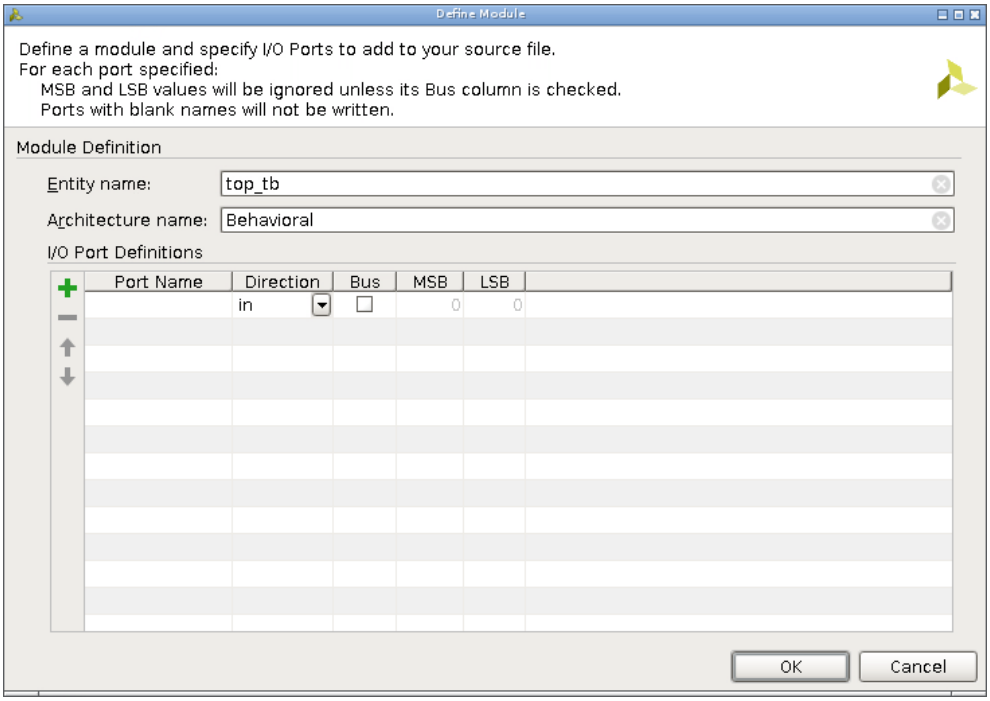
Then input “top_tb”, make sure the file type is VHDL and then hit ok.



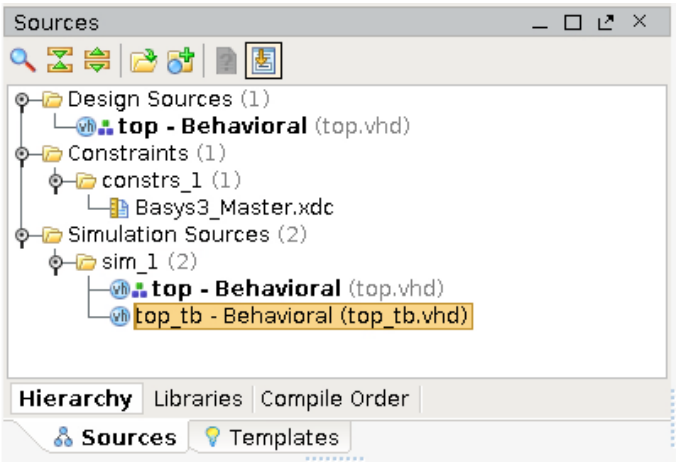
After creating the VHDL file it should look like:



Select Finish and then after that the “Module Definition” window will pop up. You don’t want to add anything here. With test benches, the goal is to exercise a component and doesn’t have anything to do with signals going in or out of the test bench entity. If a test bench had input ports you need another test bench to drive that test bench… **So leave these blank and hit Ok.**



Afterwards your source windows should have the top.vhd and the top_tb.vhd.

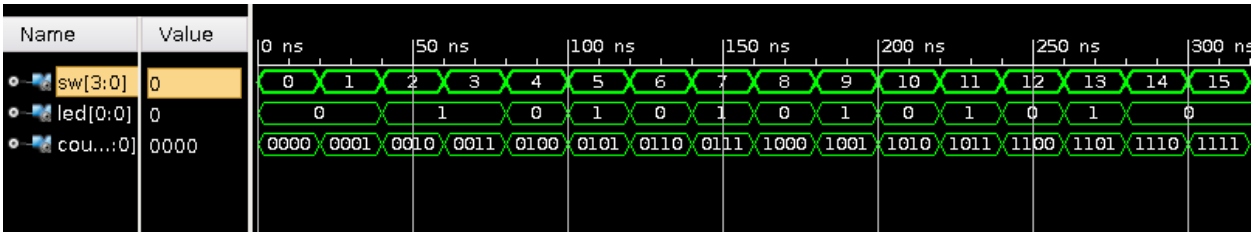


Open the top_tb VHDL file. It should be pretty barren. You'll want to add a component definition for top and add signals internal to the test bench. The internal signals would be sw and led, but also a counter signal. The reason for this is that a counter for n bits will go through all 2^n iterations of the n bits exercising all possibilities of n bits. Since we're using unsigned signals we needs to uncomment line 27 ("use IEEE.NUMERIC_STD.ALL;"). Then after tying the signals together, the architecture should look like:

```
38 architecture Behavioral of top_tb is
39
40         component top
41                 Port (
42                         sw : in STD_LOGIC_VECTOR (3 downto 0);
43                         led : out STD_LOGIC_VECTOR (0 downto 0)
44                 );
45         end component;
46
47         --signals internal to the test bench
48         signal sw:       std_logic_vector(3 downto 0);
49         signal led:      std_logic_vector(0 downto 0);
50
51         --counter used to provide input to the swtiches
52         signal counter: unsigned(3 downto 0):="0000";
53
54 begin
55
56         uut: top port map (
57                 sw  => sw,
58                 led => led
59         );
60
61         --since VHDL is strongly typed we need to convert
62         --unsigned to std_logic_vecter
63         sw <= std_logic_vector(counter);
64
65         --counter process
66         --warning: will NOT work in hardware
67         tb: process
68         begin
69                 wait for 20ns;
70                 counter <= counter + 1;
71         end process tb;
72
73 end Behavioral;
```

Running The Testbench

Now for the easy part, running it all. **All you need to do is on the "Flow Navigator" panel select "Run Simulation" and when the menu pops up select "Run Behavioral Simulation."** That should start up the simulator. Initially it should be really zoomed in so you'll need to zoom out a fair bit to see something reasonable. Another thing that might be helpful is to change the radix of the sw bus. All you have to do is right click on the signal where it has "Name" and "Value", (see picture to get a better idea) and then select "Radix" and then "unsigned decimal." It should look something like:



As you can see, the circuit works as aspected. When the switches form a prime number the LED goes high. Sadly Vivado doesn't have a way to print waveforms so you have to take a screenshot of it to include it in a report/document/etc.

If there's a more complex circuit or multiple VHDL files, you can go into the scope panel and add those signals to the waveform.

Creating the Bitstream

To create the bitstream simple go to the "Flow Navigator" panel and select "Generate Bitstream" under the "Program and Debug" section.

Programming the FPGA

I haven't yet gotten a Basys3 so I can't say for certain if you need to DL the Digilent Adept software/drivers to program the FPGA but once I acquire one this section will be updated.