Simulation Lab 2
Introduction to VHDL and Operation of Full Adder/Subtractor
Using Xilinx ISE Project Navigator and Nexys2 FPGA Development Board

National Science Foundation
Funded in part, by a grant from the National Science Foundation
DUE 1003736 and 1068182

Acknowledgements
Developed by Craig Kief, Alonzo Vera, Alexandria Haddad, and Quinlan Cao, at the Configurable Space Microsystems Innovations & Applications Center (COSMIAC). Based on original tutorial developed by Bassam Matar, Engineering Faculty at Chandler-Gilbert Community College, Chandler, Arizona. *Funded by the National Science Foundation (NSF).*

Introduction
A widening gap exists between what is taught in schools and the skills currently needed and utilized by the workforce. The jobs of the future are not labor intensive, they are brain intensive. Technicians and engineers no longer wire circuits or assemble a chip from scratch; they use software to run modern equipment. Traditionally, hardware digital circuits require at least five or six TTL ICs (Transistor Transistor Logic Integrated Circuits). The wiring is often very complex requiring digital test equipment to make the system functional. This technology is over thirty years old.

A FPGA (Field Programmable Gate Array) design minimizes the amount of electrical wiring and eliminates the use of complicated test equipment. This new technology provides students the ability to concentrate on digital principles instead of the electrical wiring, providing an opportunity to focus on larger and more complex projects. In general, a FPGA is an integrated circuit that can be reconfigured with any design, and it can be reconfigured multiple times. Thus, if you make a mistake in your design, you don't need to replace wires or chips, as you might have done using TTL. Instead, you simply reconfigure the FPGA with the updated design.

Lab Summary
You will become familiar with digital technology by using Xilinx ISE® and interfacing with a FPGA circuit board. Combinational Logic Circuits are used to make decisions based on a series of true statements that can be laid out in a truth table. In previous years the 74xx, family (TTL) circuits were used to design and build basic combinational logic circuits.

Lab Goals
Using the Design process discussed in Lab 1, we will:

1. Use the software application Xilinx ISE® Project Navigator, Schematic Editor to draw a schematic of a Full Adder and then program the Full Adder to the FPGA device.
2. Implement the same design using VHDL programming (VHSIC Hardware Descriptive Language).
3. Apply your understanding of VHDL to implement a 4-bit Adder/Subtractor with the FPGA.

Learning Objectives

	1. FPGA board overview.
Design	2. Design the Full Adder using the Schematic Editor.
	3. Design the Full Adder using the VHDL editor.
Design Synthesis	4. Create a Test Bench to test the results of the code.
Design Implementation	5. Develop a User Constraint File (ucf) and map I/O signals to the FPGA.
Device Programming	6. Program the Full Adder into the FPGA.
	7. Test your understanding by applying what you learned programming a 4-bit Adder/Subtractor.

Grading Criteria

Your grade is determined by your instructor.

Time Required

4 - 5 hours

Special Safety Requirements

When working with electronic components, such as the Xilinx FPGA board, there is potential of Electrostatic Discharge (ESD) hazards. Static electricity can damage the FPGA devices used in this lab. Use appropriate ESD methods to protect the devices. No serious hazards are involved in this laboratory experiment, but be careful to connect the components with the proper polarity to avoid damage.

Lab Preparation

- Review your PLD lecture from your class.
- Review Lab 1 – Introduction to Xilinx previous lab.
- Print out the laboratory experiment procedure that follows.
- Acquire required hardware components/equipment.
- Read Lab Procedure 1 and work the Full Adder schematic from Lab Procedure 2 as a pre-lab assignment.

Equipment and Materials

Students should work in teams of two or three. Each team of students will need the following supplies:

Supplies	Quantity
1. ISE® Design Suite (or WebPACK™) software from the Xilinx website, www.xilinx.com, if you don't already have it installed. Your classroom should have a full working version of Xilinx ISE® Design Suite. 2. FPGA kit including download and power cable(s). 3. Free Digilent Adept software (instructions for download and installation are included at the beginning of this lab): http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,828&Prod=ADEPT2	1

Additional References

The FPGA reference manual, data sheet, and any other supporting documents that may be of use.

ISE Design Suite User Guide :
http://www.xilinx.com/support/index.htm#nav=sd-nav-link-106173&tab=tab-dt

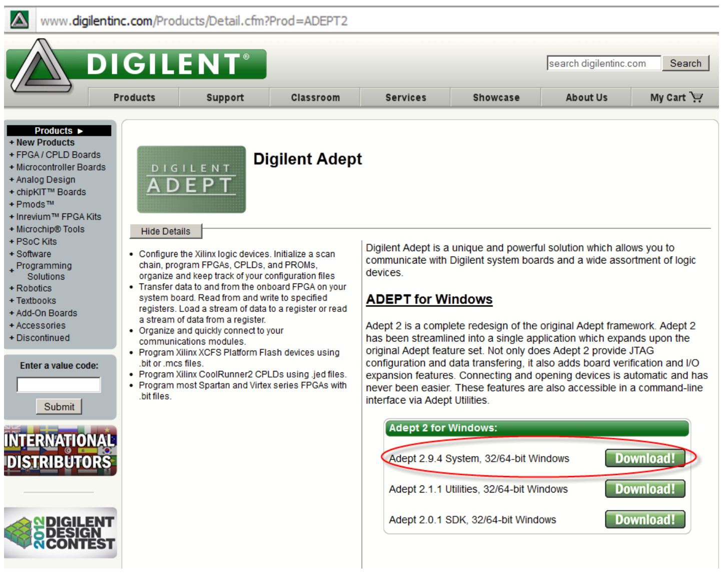
Digilent Adept User Guide:
http://www.digilentinc.com/Data/Software/Adept/Adept%20Users%20Manual.pdf

Lab Procedure (optional): Download and install Digilent Adept Software

In this lab, you download and install Digilent Adept which allows you to communicate with the Digilent FPGA boards that we use in lab. If Adept is already installed on your system, you can skip this Lab Procedure.

Step 1:

Navigate your browser to http://www.digilentinc.com/Products/Detail.cfm?Prod=ADEPT2.



Digilent Adept

- Configure the Xilinx logic devices. Initialize a scan chain, program FPGAs, CPLDs, and PROMs, organize and keep track of your configuration files
- Transfer data to and from the onboard FPGA on your system board. Read from and write to specified registers. Load a stream of data to a register or read a stream of data from a register.
- Organize and quickly connect to your communications modules.
- Program Xilinx XCFS Platform Flash devices using .bit or .mcs files.
- Program Xilinx CoolRunner2 CPLDs using .jed files.
- Program most Spartan and Virtex series FPGAs with .bit files.

Digilent Adept is a unique and powerful solution which allows you to communicate with Digilent system boards and a wide assortment of logic devices.

ADEPT for Windows

Adept 2 is a complete redesign of the original Adept framework. Adept 2 has been streamlined into a single application which expands upon the original Adept feature set. Not only does Adept 2 provide JTAG configuration and data transfering, it also adds board verification and I/O expansion features. Connecting and opening devices is automatic and has never been easier. These features are also accessible in a command-line interface via Adept Utilities.

Adept 2 for Windows:

Adept 2.9.4 System, 32/64-bit Windows

Adept 2.1.1 Utilities, 32/64-bit Windows

Adept 2.0.1 SDK, 32/64-bit Windows

Step 2:

Click the Download button for Adept 2.9.4 System, 32/64-bit Windows and click Save file when prompted. The installation .exe file will download to your computer.

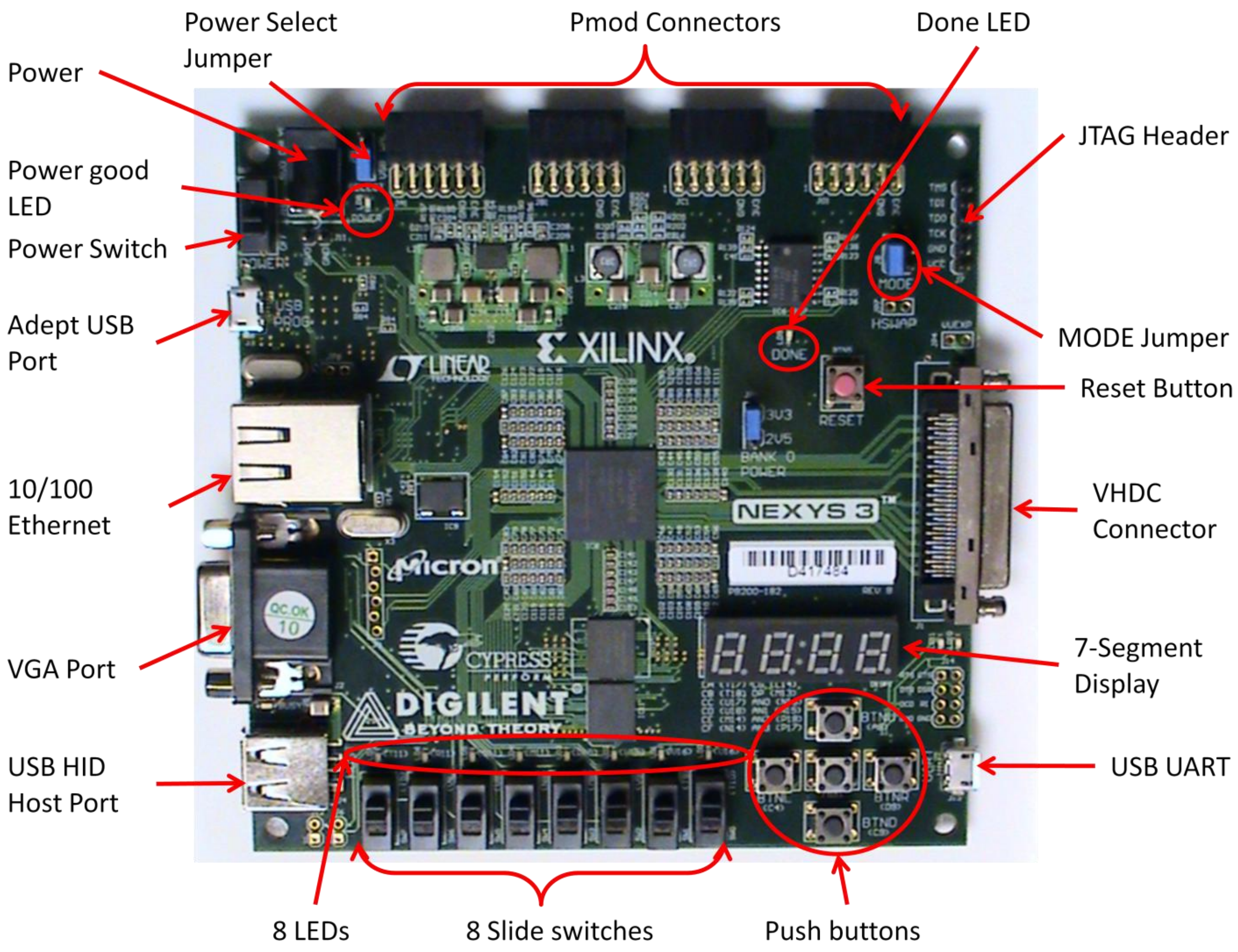
Step 3:

When the download is complete, run the .exe file and install Adept onto your computer.

Lab Procedure 1: FPGA Overview

NOTE: We used the Digilent Nexys-3 Xilinx® Spartan 6 XC6LX16-CS324 board. Your board may be a different version or from a different vendor. However, most of the components are similar. If you have a different board, review your board's documentation.

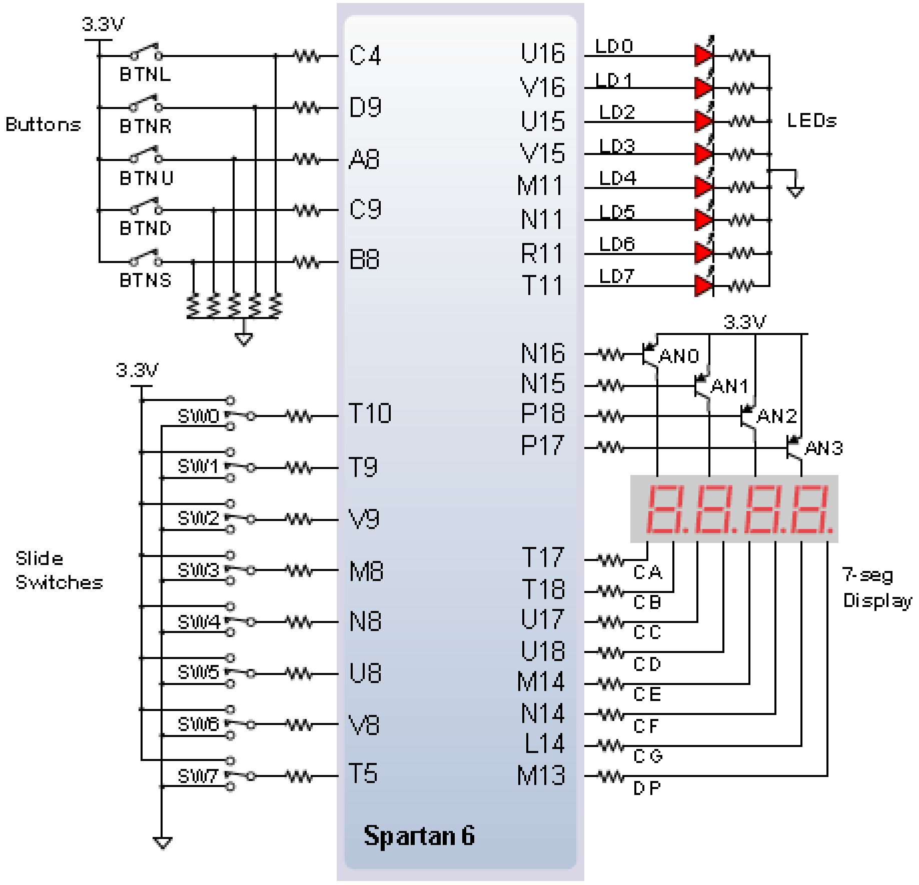
The Nexys-3 is a powerful digital system design platform built around a Xilinx Spartan 6 FPGA. The advantage of this board is that it is programmed and powered through a USB port. A list of the key features and their location on the board is listed below:



User I/O

The Nexys-3 board includes several input and output devices, and data ports allowing many implementation designs without the need for any other components. We will focus on the following inputs [slide switches, push buttons, and reset button] and outputs [LEDs, and 7-segment display].

The five pushbuttons and eight slide switches are provided for circuit inputs. There is also a reset button. Pushbutton inputs are normally low, driven high only when the pushbutton is pressed. Slide switches generate constant high or low inputs depending on their position. Pushbutton and slide switch inputs use a series resistor for protection against short circuits (a short circuit would occur if an FPGA pin assigned to a pushbutton or slide switch was inadvertently defined as an output).



Original image from the Nexys3™ Board Reference Manual

Please refer the reference manual for any additional information:

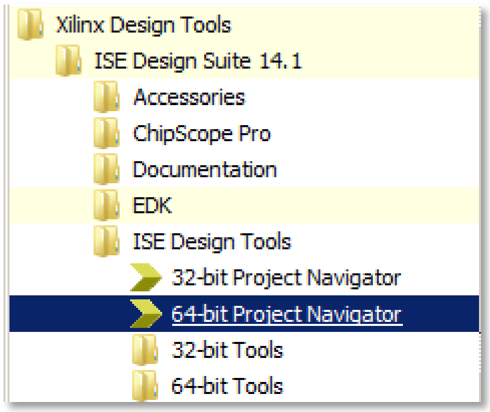
http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf

Lab Procedure 2: Design a Full Adder

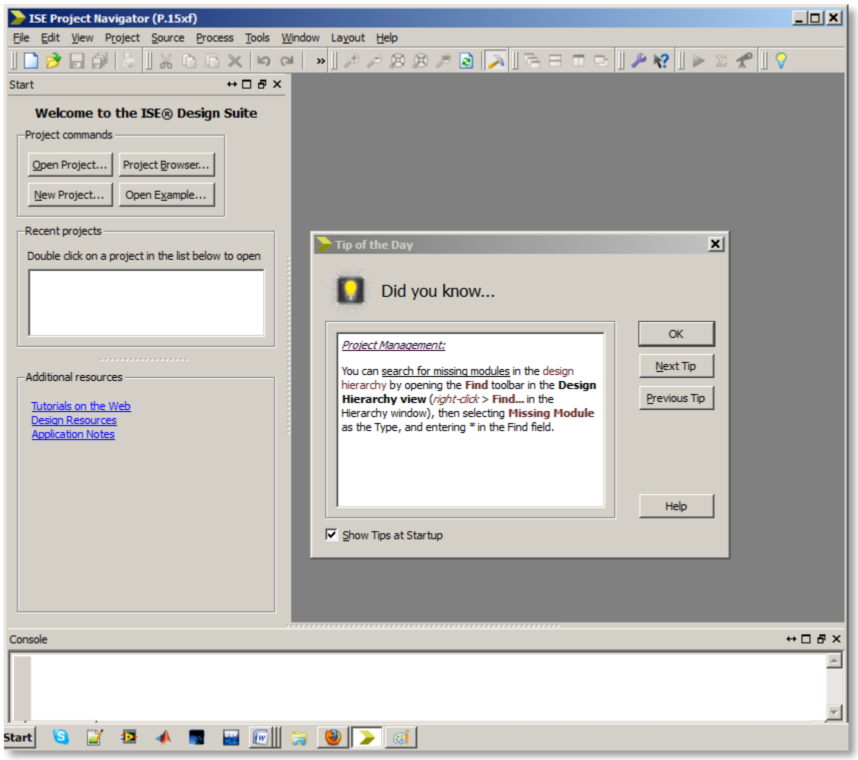
Step 1: Design Full Adder using the Schematic Editor

1. Open **Xilinx ISE Design Tool Project Navigator**.

   Your system might have slightly different Start Menu options.



2. The ISE Project Navigator window opens, with the Tip of the Day displayed. Click **OK** to close the Tip of the Day.


ISE Project Navigator (P.15xf)
File Edit View Project Source Process Tools Window Layout Help
Start
Welcome to the ISE® Design Suite
Project commands
Open Project... Project Browser...
New Project... Open Example...
Recent projects
Double click on a project in the list below to open
Additional resources
Tutorials on the Web
Design Resources
Application Notes
Tip of the Day
Did you know...
OK
Next Tip
Previous Tip
Help
Console
Start

Project Management:

You can search for missing modules in the design hierarchy by opening the **Find** toolbar in the **Design Hierarchy view** (*right-click* > **Find...** in the Hierarchy window), then selecting **Missing Module** as the Type, and entering * in the Find field.

[✓] Show Tips at Startup

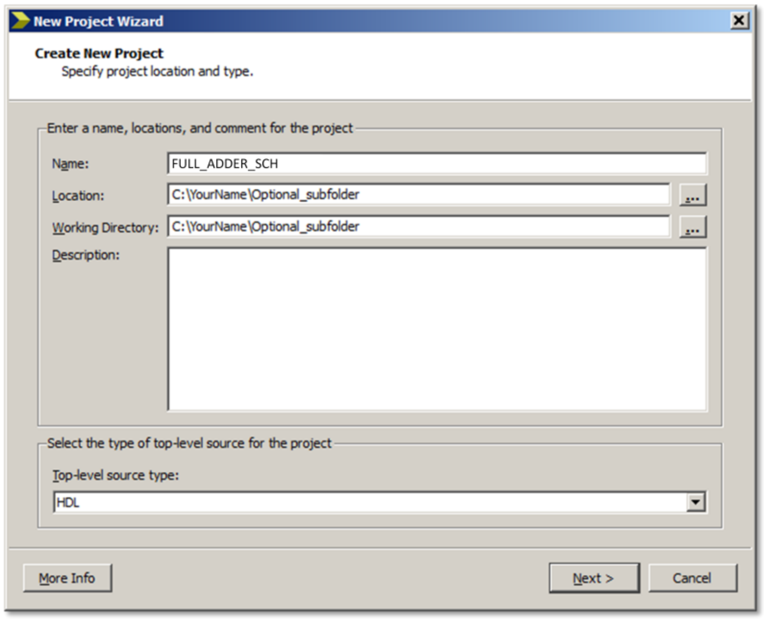
3. Start a new project by selecting **File → New Project** from the menu. The New Project Wizard starts.

   a. Type **FULL_ADDER_SCH** in the Name text box.

   b. Select a location on your computer to save your project files by clicking the ellipsis (**...**) button to the right of the **Location** text box.

   c. Under **Top-level source type**, select **HDL**.

   d. Click **Next**.


New Project Wizard

Create New Project
Specify project location and type.

Enter a name, locations, and comment for the project
Name: FULL_ADDER_SCH
Location: C:\YourName\Optional_subfolder
Working Directory: C:\YourName\Optional_subfolder
Description:

Select the type of top-level source for the project
Top-level source type: HDL

The Project Settings dialog box opens.

NOTE: File names must start with a letter. Use underscores (_) for readability. Do not use hyphens (-); although the file name will work, the entity name will not. More on this later.

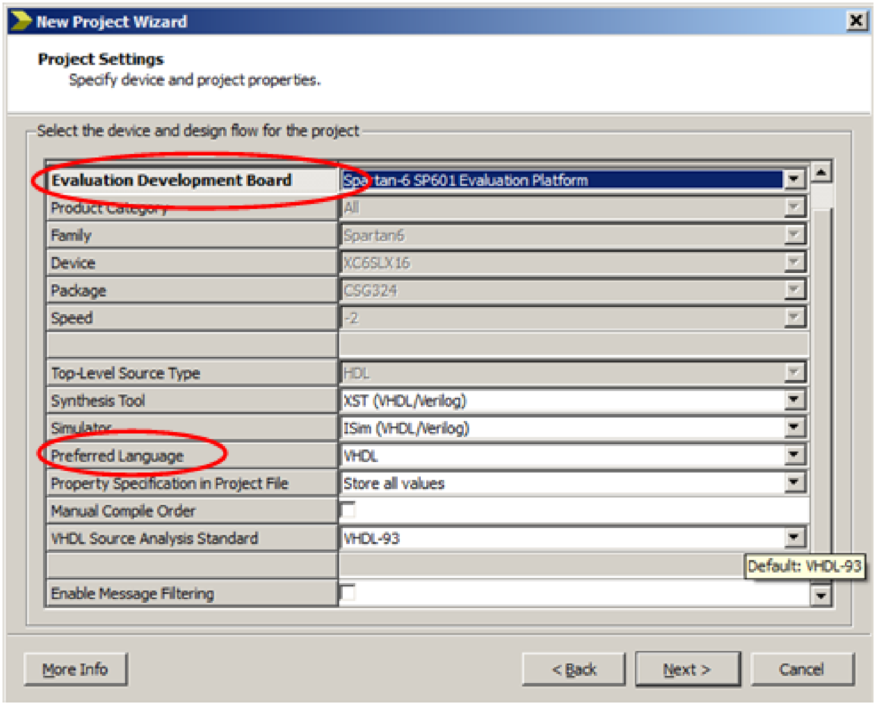
4. Select **Spartan-6 SP601 Evaluation Platform** from the **Evaluation Development Board** drop down menu.

   The **Product Category**, **Family**, **Device**, **Package**, and **Speed** should all automatically populate (top half of the screen). You will need to set some options in the lower half of the dialog box.

   Select **VHDL** from the **Preferred Language** drop down menu.

   The Project Settings should resemble the figure to the right.

5. Click **Next**.


New Project Wizard

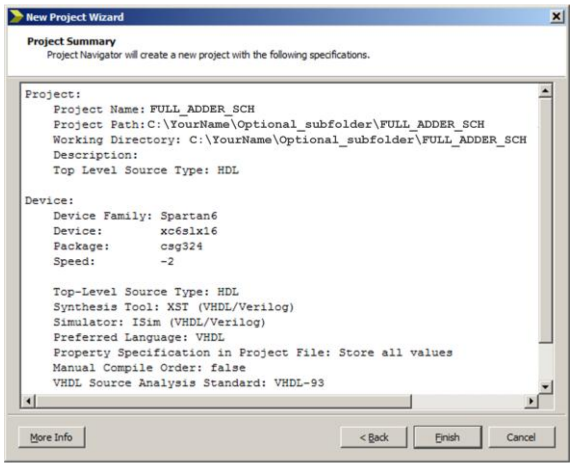
Project Settings
Specify device and project properties.

Select the device and design flow for the project

Evaluation Development Board	Spartan-6 SP601 Evaluation Platform
Product Category	All
Family	Spartan6
Device	XC6SLX16
Package	CSG324
Speed	-2
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	

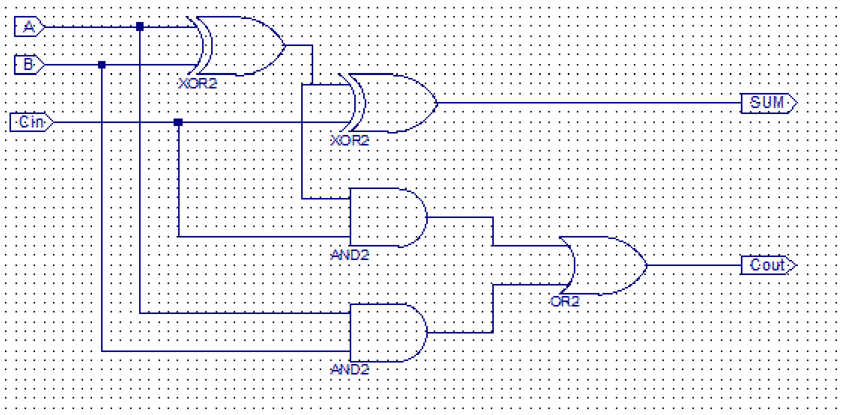
NOTE: The options specified are for the **Spartan 6 LX FPGA**. Your board might be different than the board used for these instructions. Board specifications are printed on the FPGA chip in the middle of the board. The board information is also listed on the box it came in.

6. The Project Summary displays. Verify the file type and name are correct then click **Finish** to complete the New Project creation process.


New Project Wizard
Project Summary
Project Navigator will create a new project with the following specifications.
More Info
< Back
Finish
Cancel

```
Project:
    Project Name: FULL_ADDER_SCH
    Project Path:C:\YourName\Optional_subfolder\FULL_ADDER_SCH
    Working Directory: C:\YourName\Optional_subfolder\FULL_ADDER_SCH
    Description:
    Top Level Source Type: HDL

Device:
    Device Family: Spartan6
    Device:        xc6slx16
    Package:       csg324
    Speed:         -2

    Top-Level Source Type: HDL
    Synthesis Tool: XST (VHDL/Verilog)
    Simulator: ISim (VHDL/Verilog)
    Preferred Language: VHDL
    Property Specification in Project File: Store all values
    Manual Compile Order: false
    VHDL Source Analysis Standard: VHDL-93
```

7. **Right-click** in the **Hierarchy** pane and select **New Source** from the shortcut menu (or select **Project→New Source** from the menu). The **New Source Wizard** opens.



8. Select **Schematic** from the source list, **enter a file name** for the schematic, and press **Next.** Click **Finish** to close the summary and create a blank schematic.

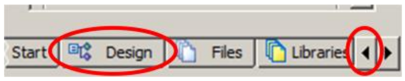
9. Create the Full Adder circuit as shown on the right. Please refer to Lab 1 for details about creating the schematic.

10. Attach input and output pins to the circuit and label them as shown.

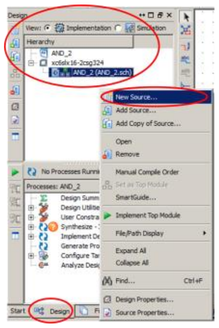
Step 2: Create Design Full Adder with VHDL Editor

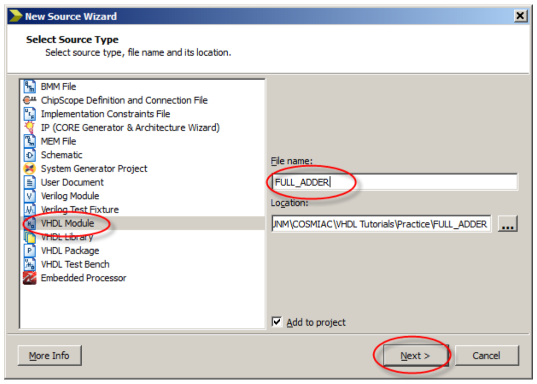
NOTE: VHDL is an acronym inside of an acronym. The 'V' stands for Very High Speed Integrated Circuit (VHSIC) and 'HDL' stands for Hardware Descriptive Language. VHDL is a powerful language with numerous language constructs that are capable of describing very complex behavior needed for today's programmable devices.

1. **Click** the panel tab's left triangle button to display the **Design** tab.



2. **Right-click** in the **Hierarchy Panel** and Click **New Source...** from the shortcut menu (or select **Project→New Source** from the menu). The New Source Wizard displays.
3. Select **VHDL Module** from the source type list.
4. Type **AND_2** as the file name.
5. Click the **Next** button. The Define Module dialog displays.




New Source Wizard

Select Source Type
Select source type, file name and its location.

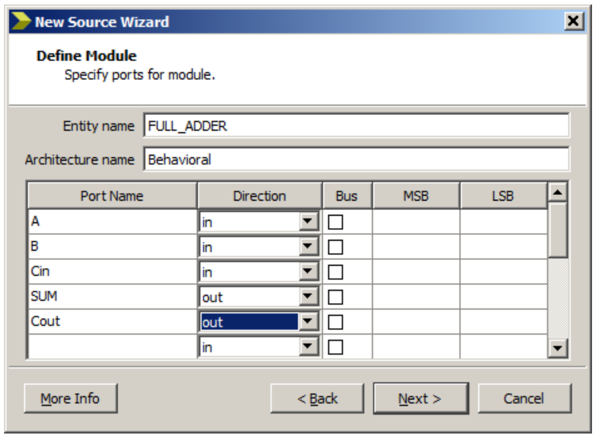
- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- System Generator Project
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: FULL_ADDER

Location: \UNM\COSMIAC\VHDL Tutorials\Practice\FULL_ADDER

☑ Add to project

6. Type the **port names** and select the **direction** of each port as shown in the figure to the right.
7. Click the **Next** button. The Summary dialog displays.
8. Click the **Finish** button to accept the New Source Wizards options.


New Source Wizard

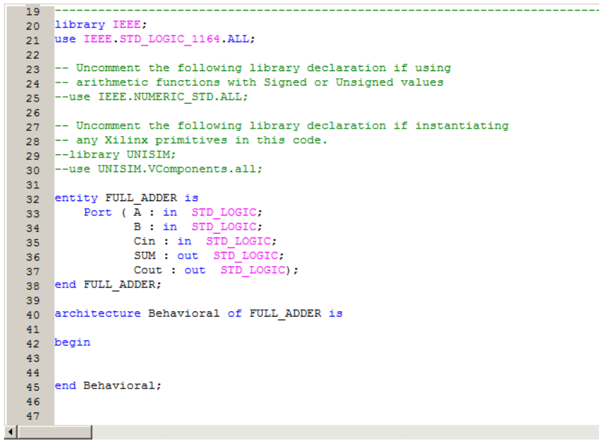
Define Module
Specify ports for module.

Entity name: FULL_ADDER

Architecture name: Behavioral

Port Name	Direction	Bus	MSB	LSB
A	in	☐		
B	in	☐		
Cin	in	☐		
SUM	out	☐		
Cout	out	☐		
	in	☐		

9. ISE will generate source code for you and it will display in the **workspace** area.



```
-------------------------------------------------------------------------------
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 primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity FULL_ADDER is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           Cin : in  STD_LOGIC;
           SUM : out  STD_LOGIC;
           Cout : out  STD_LOGIC);
end FULL_ADDER;

architecture Behavioral of FULL_ADDER is

begin


end Behavioral;


```

10. Based on our previous schematic design, the logic expression for the output Sum and Cout are:

    SUM = A⊕B⊕Cin

    Cout = ((A⊕B) Cin) + AB

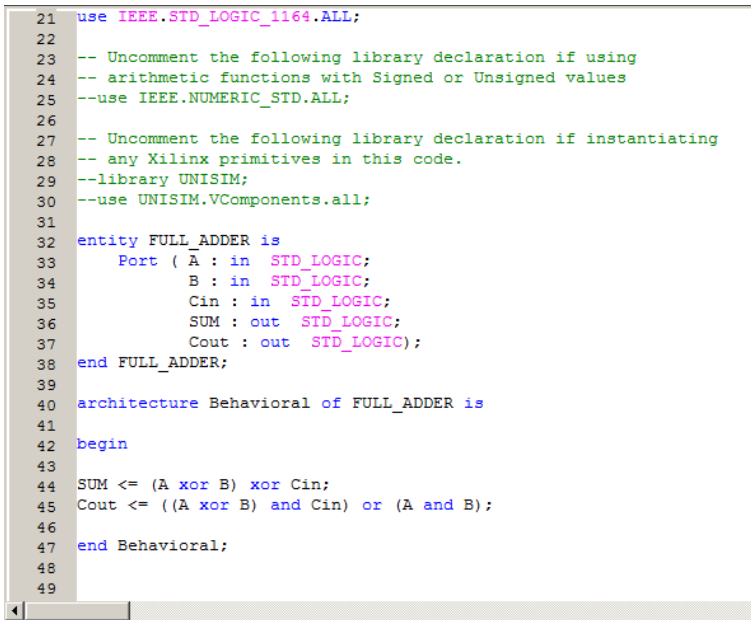
    Scroll down to the **architecture** section of the code and modify the code as highlighted:

begin

SUM <= (A xor B) xor Cin;

Cout <= ((A xor B) and Cin) or (A and B);

end Behavioral;



```
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 primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity FULL_ADDER is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           Cin : in  STD_LOGIC;
           SUM : out  STD_LOGIC;
           Cout : out  STD_LOGIC);
end FULL_ADDER;

architecture Behavioral of FULL_ADDER is

begin

SUM <= (A xor B) xor Cin;
Cout <= ((A xor B) and Cin) or (A and B);

end Behavioral;


```

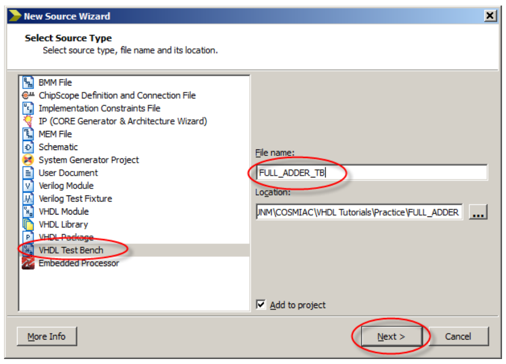
11. Save your files by clicking on the **Save All** button.

Step 3: Create a test bench for the Full Adder

This simulation will be a little more complex than the one we created for Lab 1. However, it will give you a good feel for how to read timing diagrams. The logic table for the circuit is shown at right.

inputs			outputs	
A	B	Cin	SUM	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

1. Create a test bench for your file. **Right-click** in the Hierarchy Pane and select **New Source** from the short cut menu.
2. Select **VHDL Test Bench** as the source type.
3. Enter **FULL_ADDER_TB** as the file name.

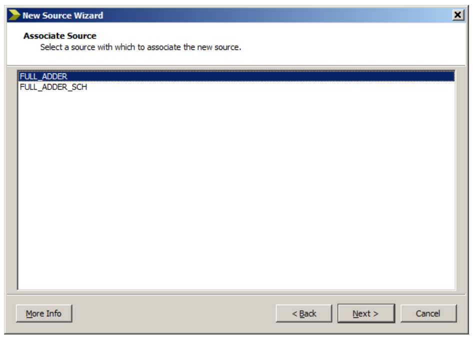

New Source Wizard

Select Source Type
Select source type, file name and its location.

- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- System Generator Project
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: FULL_ADDER_TB
Location: JNM\COSMIAC\VHDL Tutorials\Practice\FULL_ADDER
☑ Add to project

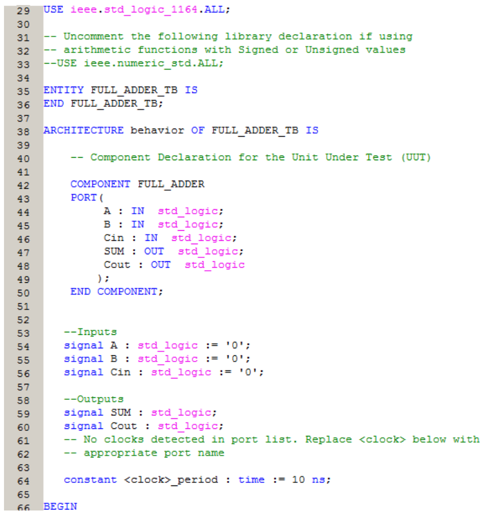
4. Click the **Next** button. The Associate Source dialog box opens. Select **FULL_ADDER** and click the Next Button.
5. The New Source Summary dialog box displays. Click the **Finish** button.


New Source Wizard

Associate Source
Select a source with which to associate the new source.

FULL_ADDER
FULL_ADDER_SCH

6. The test bench file will display in the workspace.



```
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;

ENTITY FULL_ADDER_TB IS
END FULL_ADDER_TB;

ARCHITECTURE behavior OF FULL_ADDER_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT FULL_ADDER
    PORT(
         A : IN  std_logic;
         B : IN  std_logic;
         Cin : IN  std_logic;
         SUM : OUT  std_logic;
         Cout : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal A : std_logic := '0';
   signal B : std_logic := '0';
   signal Cin : std_logic := '0';

   --Outputs
   signal SUM : std_logic;
   signal Cout : std_logic;
   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name

   constant <clock>_period : time := 10 ns;

BEGIN
```

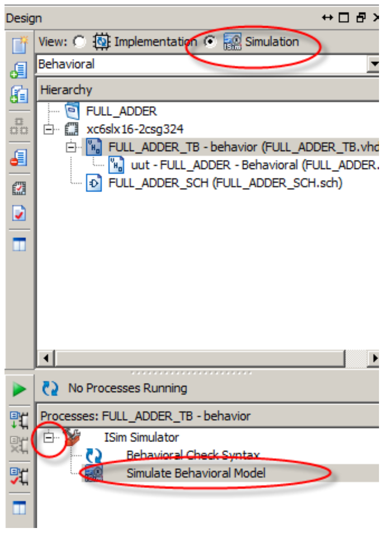
7. Starting around line 71, make the changes highlighted on the right. Delete strikeout text and add highlighted on the right.

8. Click the **Save All** button on the toolbar.

```
~~-- No clocks detected in port list. Replace <clock> below with~~
~~   -- appropriate port name~~
~~   constant <clock>_period : time := 10 ns;~~
BEGIN
         -- Instantiate the Unit Under Test (UUT)
   uut: FULL_ADDER PORT MAP (
          A => A,
          B => B,
          Cin => Cin,
          SUM => SUM,
          Cout => Cout
        );
~~-- Clock process definitions~~
~~   <clock>_process :process~~
~~   begin~~
~~                <clock> <= '0';~~
~~                wait for <clock>_period/2;~~
~~                <clock> <= '1';~~
~~                wait for <clock>_period/2;~~
~~   end process;~~
   -- Stimulus process
stim_proc: process
~~   begin~~
~~      -- hold reset state for 100 ns.~~
~~      wait for 100 ns;~~
~~      wait for <clock>_period*10;~~
~~      -- insert stimulus here~~
~~      wait;~~
~~   end process;~~
```

```
          A_proc: process
          begin
                    A <= not A;
                    wait for 25 ns;
          end process;

          B_proc: process
                    begin
                    B <= not B;
                    wait for 50 ns;
          end process;

          Cin_proc: process
                    begin
                    Cin <= not Cin;
                    wait for 100 ns;
          end process;
END;
```

9. On the **View Pane**, select **Simulation**. The Simulation Hierarchy and Process options display.
10. **Expand** the ISim Simulator tool if it isn't already expanded.
11. **Double-click** the **Simulate Behavioral Model** tool. Click **Yes** to save any changes if prompted.


Design
View: ○ Implementation ● Simulation
Behavioral
Hierarchy
FULL_ADDER
xc6slx16-2csg324
FULL_ADDER_TB - behavior (FULL_ADDER_TB.vhd
uut - FULL_ADDER - Behavioral (FULL_ADDER.
FULL_ADDER_SCH (FULL_ADDER_SCH.sch)
No Processes Running
Processes: FULL_ADDER_TB - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model

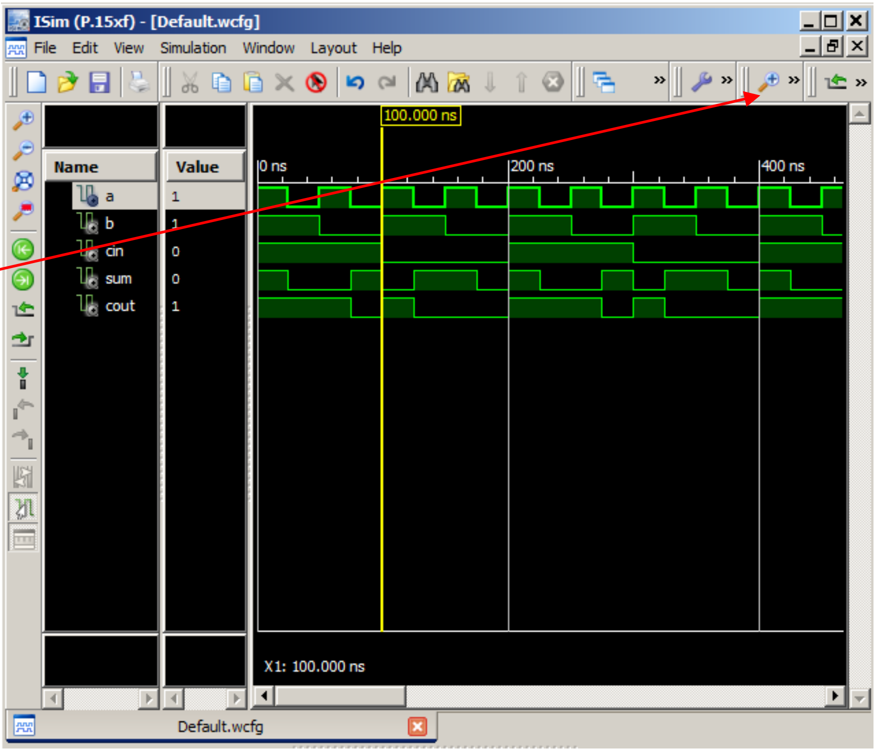
12. The Simulation Elaboration will begin to run. When it is finished the **ISim** window will open displaying the simulation.

    Use the **Zoom to Full View** and the **Zoom In** buttons to display the simulation waveform clearer.

    Review the waveform to ensure the .vhd file is working as designed.

    When you are finished **close** the ISim application and return to ISE.

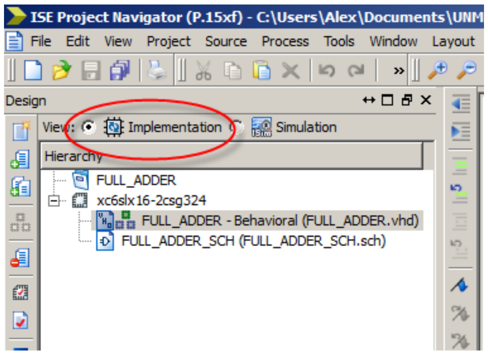
13. When you have returned to the **ISE Project Manager** window, click the **Save all** button to save your work.



Step 4: Create the User Constraint File (UCF) and map I/O to the FPGA board

The last step before we actually program our code into the FPGA is to create a UCF file. This file assigns the I/O pins of the FPGA to the I/O ports of our code.

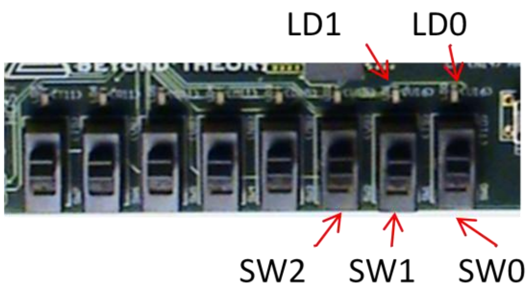
1. Switch to **Implementation View** by clicking the **Implementation** option in the **View Panel**.


ISE Project Navigator (P.15xf) - C:\Users\Alex\Documents\UNM
File Edit View Project Source Process Tools Window Layout
Design
View: Implementation Simulation
Hierarchy
FULL_ADDER
xc6slx16-2csg324
FULL_ADDER - Behavioral (FULL_ADDER.vhd)
FULL_ADDER_SCH (FULL_ADDER_SCH.sch)

2. To write the UCF code you need to know what pins you are using. This tutorial was written for the Digilent® Nexys 3™ Spartan 6 XC6LX16-CS324 board.

   Refer to the previous section, **Lab Procedure 1: FPGA Overview, User I/O** for pin label identification for the Spartan 6.

   If you are using a different board, either look at the board or read the board's reference manual. Make a note of the pin labels for SW0, SW1, SW2, LED0, and LED1.



NOTE: The Digilent® Spartan 3 boards are available in two speeds, 500k and 1200k.

Both boards are labeled the same, however **the 1200k board has different pin assignments for LD4-7** as follows:

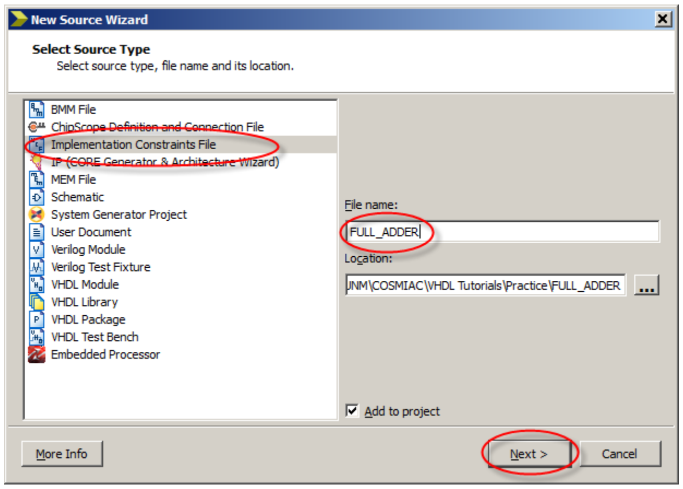
LD4	E16
LD5	P16
LD	E4
LD7	P4

3. **Right-click** in the Hierarchy Pane and select **New Source** from the short cut menu.

4. Select **Implementation Constraints File** as the source type.

5. Type **FULL_ADDER** as the file name.

6. Click the **Next** button to view the Summary screen, and then click the **Finish** button.



File name: FULL_ADDER

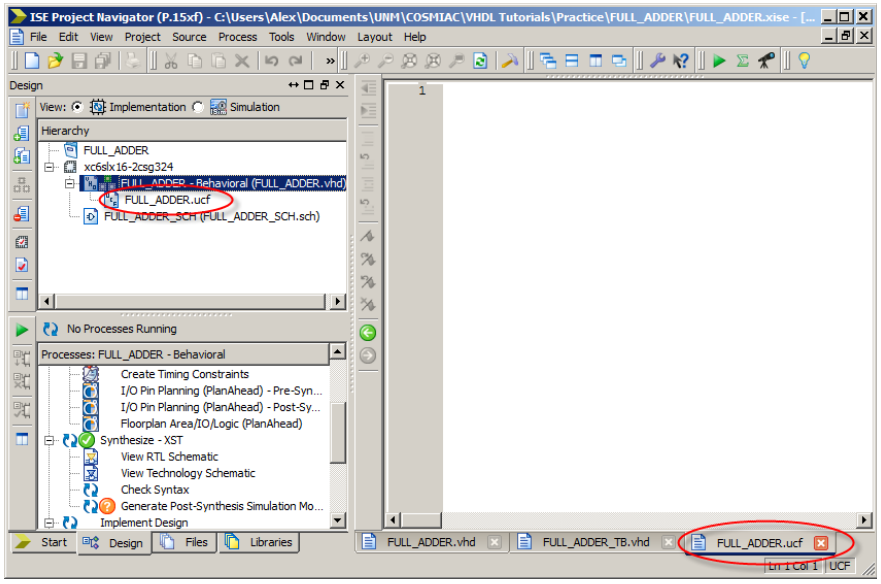
Location: JNM\COSMIAC\VHDL Tutorials\Practice\FULL_ADDER

[✓] Add to project

7. A new, blank file is added to your project.

NOTE: Notice the hierarchy of the files in your project. The project FULL_ADDER is at the top. The .vhd and .sch file are sub-files (children) of the project. Under the .vhd file is another sub-file, **FULL_ADDER.ucf**, which is the file we just created.

The hierarchy of the project will become very important when you are creating bigger and more complex projects.


ISE Project Navigator (P.15xf) - C:\Users\Alex\Documents\UNM\COSMIAC\VHDL Tutorials\Practice\FULL_ADDER\FULL_ADDER.xise - [...
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
FULL_ADDER
xc6slx16-2csg324
FULL_ADDER - Behavioral (FULL_ADDER.vhd)
FULL_ADDER.ucf
FULL_ADDER_SCH (FULL_ADDER_SCH.sch)
No Processes Running
Processes: FULL_ADDER - Behavioral
Create Timing Constraints
I/O Pin Planning (PlanAhead) - Pre-Syn...
I/O Pin Planning (PlanAhead) - Post-Sy...
Floorplan Area/IO/Logic (PlanAhead)
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Simulation Mo...
Implement Design
Start
Design
Files
Libraries
FULL_ADDER.vhd
FULL_ADDER_TB.vhd
FULL_ADDER.ucf
Ln 1 Col 1
UCF

8. Type the pin assignments as shown on the right.

NOTE: UCF files use the pound sign (#) to comment compared to VHDL files which use dash dash (--) for comments.

9. Click the **Save All** button to save your work.

```
#Inputs

NET "A" LOC = "T10";       # Switch 0 (SW0)
NET "B" LOC = "T9";        # Switch 1 (SW1)
NET "Cin" LOC = "V9";      # Switch 2 (SW2)


# Outputs

NET "SUM" LOC = "U16";   # LED 0 for SUM
NET "COUT" LOC = "V16";  # LED 1 for Cout
```

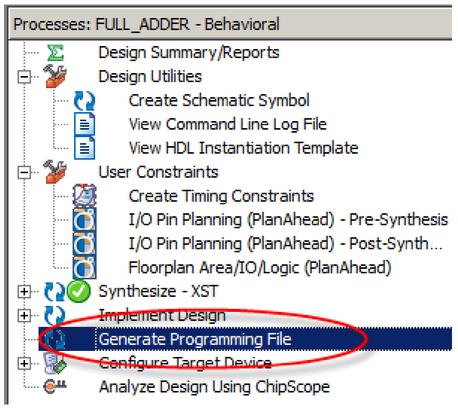
Step 5: Program Full Adder to the FPGA board

Task A: Synthesize, Implement, Map, Place and Route, Generate Bitstream

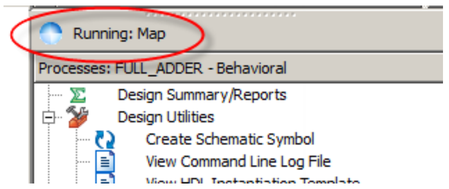
1. Insert the small end of the USB cable into the Adept USB Port on the FPGA board. Insert the USB end into your computer.

2. Turn on the FPGA board. The display should alternate flash PASS and 128.

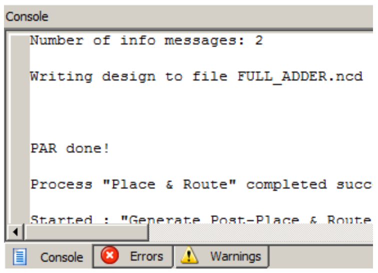
3. In the Processes Panel, double-click Generate Programming File.


Processes: FULL_ADDER - Behavioral
Design Summary/Reports
Design Utilities
Create Schematic Symbol
View Command Line Log File
View HDL Instantiation Template
User Constraints
Create Timing Constraints
I/O Pin Planning (PlanAhead) - Pre-Synthesis
I/O Pin Planning (PlanAhead) - Post-Synth...
Floorplan Area/IO/Logic (PlanAhead)
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope

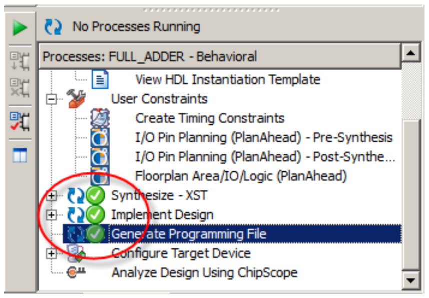
As the program is going through the compile process the compiling the process status icon spins and displays the current process that is running.


Running: Map
Processes: FULL_ADDER - Behavioral
Design Summary/Reports
Design Utilities
Create Schematic Symbol
View Command Line Log File

The Console Panel also displays textually what is happening.


Console
```
Number of info messages: 2

Writing design to file FULL_ADDER.ncd



PAR done!

Process "Place & Route" completed succ

Started : "Generate Post-Place & Route
```

4. Once the process has stopped running, and there are no errors or warnings, there will be three green checkmarks next to the Synthesize, Implement Design, and Generate Programming File processes.


No Processes Running
Processes: FULL_ADDER - Behavioral
View HDL Instantiation Template
User Constraints
Create Timing Constraints
I/O Pin Planning (PlanAhead) - Pre-Synthesis
I/O Pin Planning (PlanAhead) - Post-Synthe...
Floorplan Area/IO/Logic (PlanAhead)
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope

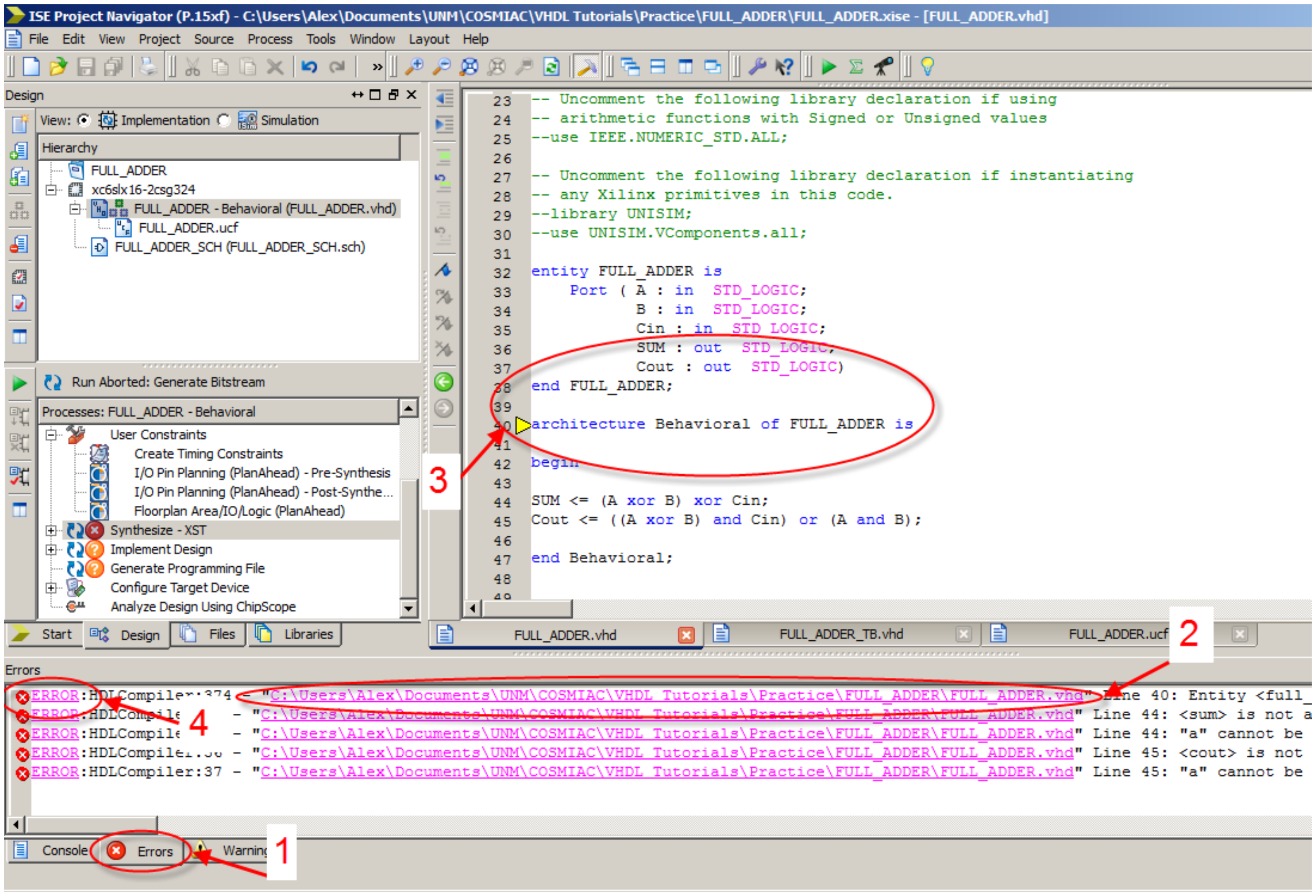
NOTE: If you have errors or warnings you will need to find and fix the errors. You may or many not need to fix warnings. After fixing any errors, re-generate the programming file.

Click the Errors (and/or Warnings) Panel tab to display a list of all errors (and/or warnings) (1).

Clicking the hyperlink (2) of a particular Error (or Warning) will take you to the file and line (3) of the Error (or Warning). Note in the example the semicolon is missing at the end of line 37.

Clicking the hyperlink word Error (4) (or Warning) will display context sensitive information about the particular error/warning.

To re-generate the programming file, right-click on **Generate Programming File** and select **Rerun All**.


ISE Project Navigator (P.15xf) - C:\Users\Alex\Documents\UNM\COSMIAC\VHDL Tutorials\Practice\FULL_ADDER\FULL_ADDER.xise - [FULL_ADDER.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
FULL_ADDER
xc6slx16-2csg324
FULL_ADDER - Behavioral (FULL_ADDER.vhd)
FULL_ADDER.ucf
FULL_ADDER_SCH (FULL_ADDER_SCH.sch)
Run Aborted: Generate Bitstream
Processes: FULL_ADDER - Behavioral
User Constraints
Create Timing Constraints
I/O Pin Planning (PlanAhead) - Pre-Synthesis
I/O Pin Planning (PlanAhead) - Post-Synthe...
Floorplan Area/IO/Logic (PlanAhead)
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
Start
Design
Files
Libraries
FULL_ADDER.vhd
FULL_ADDER_TB.vhd
FULL_ADDER.ucf
Errors
Console
Errors
Warning
1
2
3
4

```
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity FULL_ADDER is
33       Port ( A : in  STD_LOGIC;
34              B : in  STD_LOGIC;
35              Cin : in  STD_LOGIC;
36              SUM : out  STD_LOGIC;
37              Cout : out  STD_LOGIC)
38  end FULL_ADDER;
39
40  architecture Behavioral of FULL_ADDER is
41
42  begin
43
44  SUM <= (A xor B) xor Cin;
45  Cout <= ((A xor B) and Cin) or (A and B);
46
47  end Behavioral;
48
```

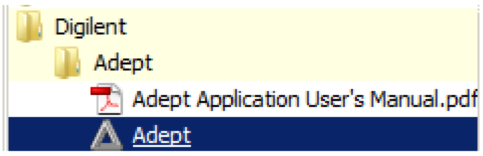
```
ERROR:HDLCompiler:374 - "C:\Users\Alex\Documents\UNM\COSMIAC\VHDL Tutorials\Practice\FULL_ADDER\FULL_ADDER.vhd" Line 40: Entity <full_
ERROR:HDLCompile      - "C:\Users\Alex\Documents\UNM\COSMIAC\VHDL Tutorials\Practice\FULL_ADDER\FULL_ADDER.vhd" Line 44: <sum> is not a
ERROR:HDLCompile      - "C:\Users\Alex\Documents\UNM\COSMIAC\VHDL Tutorials\Practice\FULL_ADDER\FULL_ADDER.vhd" Line 44: "a" cannot be
ERROR:HDLCompiler:    - "C:\Users\Alex\Documents\UNM\COSMIAC\VHDL Tutorials\Practice\FULL_ADDER\FULL_ADDER.vhd" Line 45: <cout> is not
ERROR:HDLCompiler:37  - "C:\Users\Alex\Documents\UNM\COSMIAC\VHDL Tutorials\Practice\FULL_ADDER\FULL_ADDER.vhd" Line 45: "a" cannot be
```

Task B: Implement design to FPGA board

Because the FPGAs we are working with use a single USB cable for power and PC connection we need to use Digilent Adept to implement the .vhd code to the board.

1. Open **Digilent Adept** by selecting Start → All Programs→ Digilent → Adept → Adept

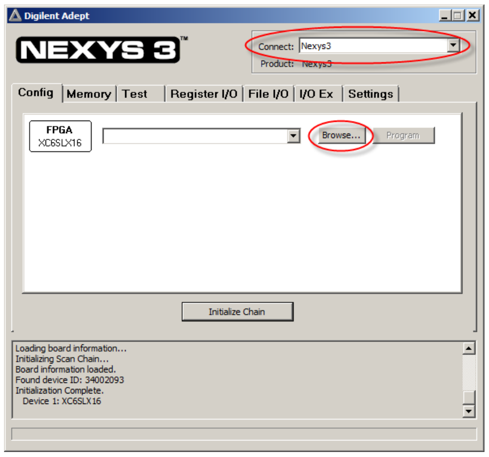
   Your system might have slightly different Start Menu options.



2. Choose **Nexys3** from the **Connect Product** drop down menu.

NOTE: If you do not see Nexys3 as an option, check the USB connection to ensure the board is connected to the computer.

3. Click the Browse button. The Open dialog box displays.

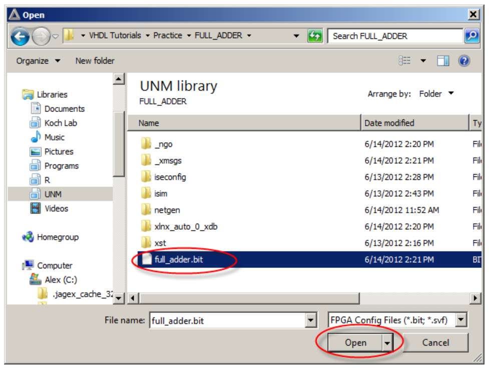


Connect: Nexys3
Product: Nexys3

```
Loading board information...
Initializing Scan Chain...
Board information loaded.
Found device ID: 34002093
Initialization Complete.
  Device 1: XC6SLX16
```

4. Navigate to the folder of the Full_Adder project and select the full_adder.bit file.

5. Click the Open button. The file is displayed in the dropdown list next to the FPGA icon.

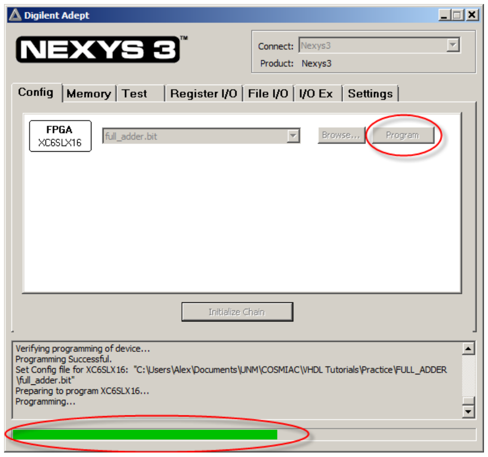


Name	Date modified	Ty
_ngo	6/14/2012 2:20 PM	Fil
_xmsgs	6/14/2012 2:21 PM	Fil
iseconfig	6/13/2012 2:28 PM	Fil
isim	6/13/2012 2:43 PM	Fil
netgen	6/14/2012 11:52 AM	Fil
xlnx_auto_0_xdb	6/14/2012 2:20 PM	Fil
xst	6/13/2012 2:16 PM	Fil
full_adder.bit	6/14/2012 2:21 PM	BI

File name: full_adder.bit
FPGA Config Files (*.bit; *.svf)

6. Click the Program button. The bit file is programmed to the FPGA board. The green status bar at the bottom of the Adept screen displays the status as the chip is programmed.

7. The program gets downloaded to the FPGA board.

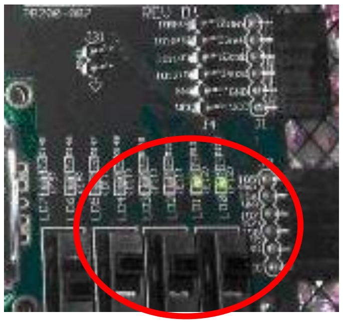


```
Verifying programming of device...
Programming Successful.
Set Config file for XC6SLX16:  "C:\Users\Alex\Documents\UNM\COSMIAC\VHDL Tutorials\Practice\FULL_ADDER
\full_adder.bit"
Preparing to program XC6SLX16...
Programming...
```

8. Test the FPGA by sliding the programmed switches ON (up toward the middle of the board) and OFF (down toward the edge of the board). The LED lights 0 and 1 should light according to the truth table.

SW(0) (A) G18	SW(1) (B) H18	SW(2) (Cin) K18	LD0 (SUM) J14	LD1 (Cout) J15
0	0	0	OFF	OFF
0	0	1	ON	OFF
0	1	0	ON	OFF
0	1	1	OFF	ON
1	0	0	ON	OFF
1	0	1	OFF	ON
1	1	0	OFF	ON
1	1	1	ON	ON

9. When all three Switches are ON, both outputs are High (SUM and Cout).



10. Close Adept and close the
    FULL_ADDER project.

As you can see with this approach, you do not need to replace wires and ICs if you make mistake as you might have done in Labs 1 and 2 with TTL. Instead, you will have to reconfigure the FPGA with the updated design.

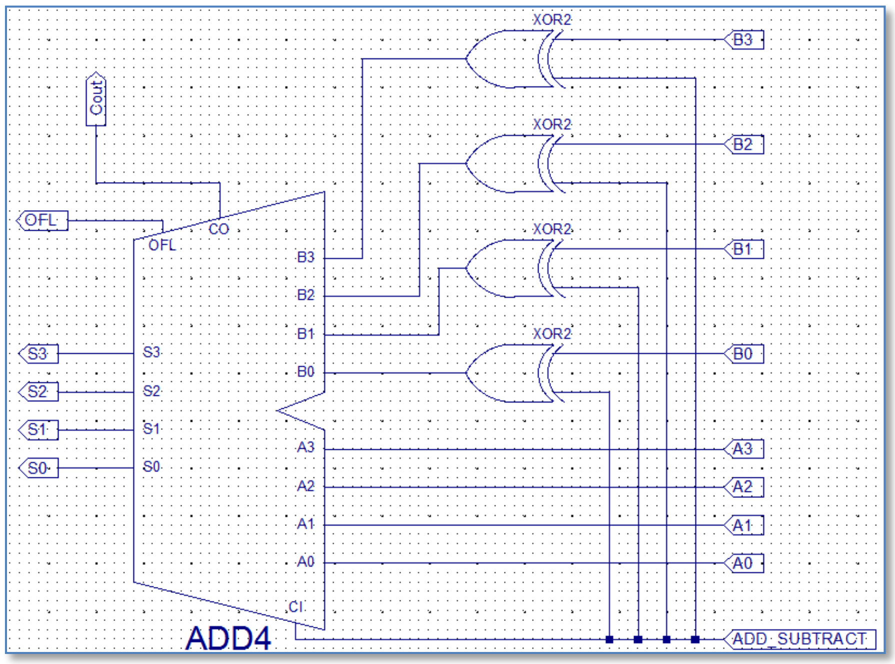
This type of technology allows you to concentrate on digital principles and not on the electrical wiring. Larger and more complex projects can be undertaken now that the tedious manual procedures are automated. Lab Procedure 3 of this lab demonstrates the design of a large design.

Lab Procedure 3: Test your Knowledge

Implement the following 4-bit adder/subtractor schematic using Xilinx ISE tools for Nexys3 FPGA board. Also, implement the circuit with VHDL.

1. Create below schematic by choosing ADD4 symbol from Xilinx library of devices

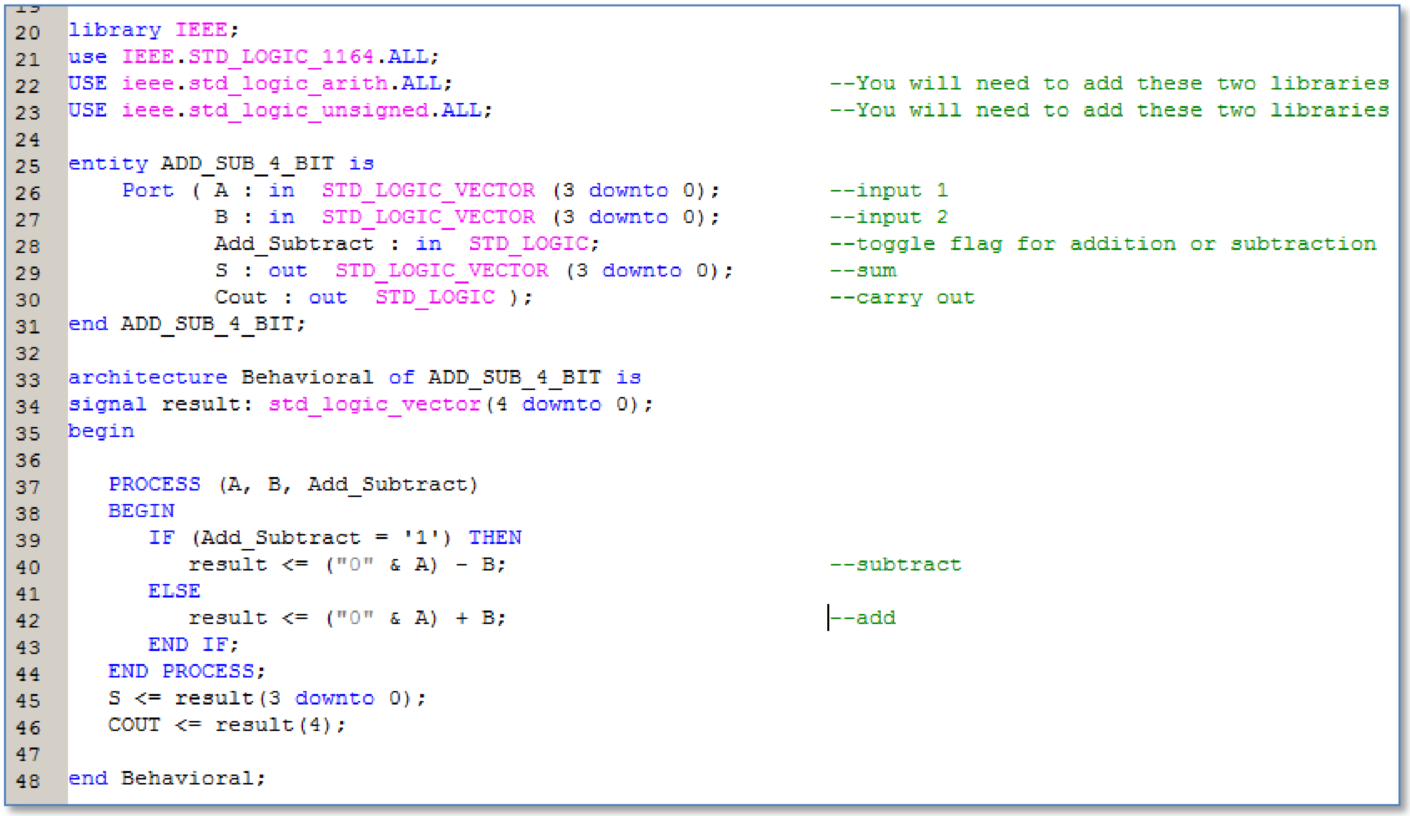
NOTE: You can rotate a symbol by selecting it and pressing CTRL+R, or select the symbol, click on the right key mouse→select symbol→rotate.



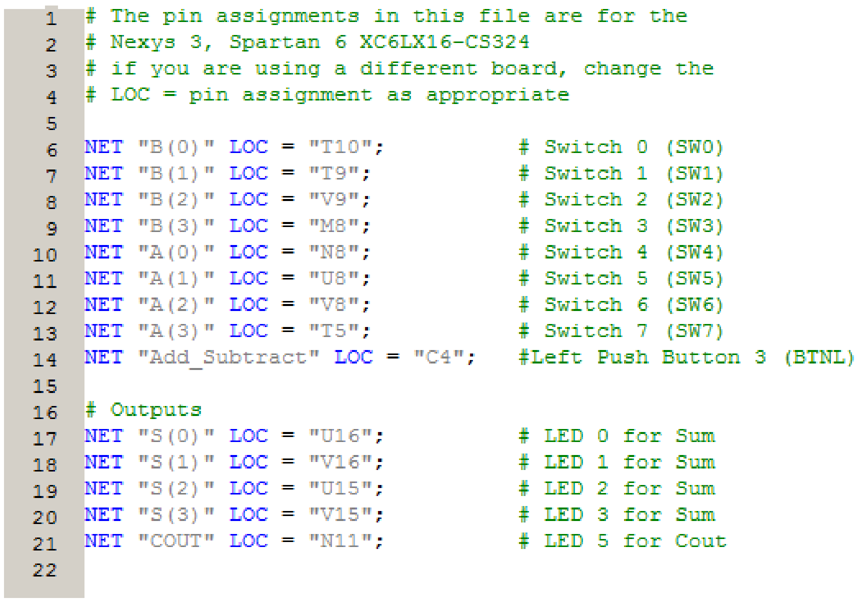
2. Add inputs and output ports
3. Simulate your design.
4. Develop a UCF that maps the input (A3..0, B3..0 and ADD/SUB) and output signals (S3..0, Cout and OFL) to your FPGA.
5. Program your design to your FPGA.
6. Demo your design to your instructor and complete the following truth table:

ADD/SUBTRACT	**B3 B2 B1 B0**	**A3 A2 A1 A0**	**S3 S2 S1 S0**	**OFL (yes/no)**	**Cout**
1	0 1 0 1	1 0 0 1			
0	0 1 0 1	1 0 0 1			

Here is one version of the VHDL and UCF files:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.ALL;                             --You will need to add these two libraries
USE ieee.std_logic_unsigned.ALL;                          --You will need to add these two libraries

entity ADD_SUB_4_BIT is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);         --input 1
           B : in  STD_LOGIC_VECTOR (3 downto 0);         --input 2
           Add_Subtract : in  STD_LOGIC;                  --toggle flag for addition or subtraction
           S : out  STD_LOGIC_VECTOR (3 downto 0);        --sum
           Cout : out  STD_LOGIC );                       --carry out
end ADD_SUB_4_BIT;

architecture Behavioral of ADD_SUB_4_BIT is
signal result: std_logic_vector(4 downto 0);
begin

   PROCESS (A, B, Add_Subtract)
   BEGIN
      IF (Add_Subtract = '1') THEN
         result <= ("0" & A) - B;                         --subtract
      ELSE
         result <= ("0" & A) + B;                         --add
      END IF;
   END PROCESS;
   S <= result(3 downto 0);
   COUT <= result(4);

end Behavioral;
```



```
# The pin assignments in this file are for the
# Nexys 3, Spartan 6 XC6LX16-CS324
# if you are using a different board, change the
# LOC = pin assignment as appropriate

NET "B(0)" LOC = "T10";          # Switch 0 (SW0)
NET "B(1)" LOC = "T9";           # Switch 1 (SW1)
NET "B(2)" LOC = "V9";           # Switch 2 (SW2)
NET "B(3)" LOC = "M8";           # Switch 3 (SW3)
NET "A(0)" LOC = "N8";           # Switch 4 (SW4)
NET "A(1)" LOC = "U8";           # Switch 5 (SW5)
NET "A(2)" LOC = "V8";           # Switch 6 (SW6)
NET "A(3)" LOC = "T5";           # Switch 7 (SW7)
NET "Add_Subtract" LOC = "C4";   #Left Push Button 3 (BTNL)

# Outputs
NET "S(0)" LOC = "U16";          # LED 0 for Sum
NET "S(1)" LOC = "V16";          # LED 1 for Sum
NET "S(2)" LOC = "U15";          # LED 2 for Sum
NET "S(3)" LOC = "V15";          # LED 3 for Sum
NET "COUT" LOC = "N11";          # LED 5 for Cout
```

NOTE: Since inputs A and B are declared as vectors in the VHDL code, you need to have the location of each input (0-3) in parenthesis as shown above. This is not necessary for bit vectors such as inputs A and B in the FULL_ADDER program used previously.