3 Lab 1 – Design Entry Using Altera Quartus-II

The goal of this lab session is primarily to gain experience with the Altera Quartus Prime FPGA Design Environment. Lab1 will go through two design entry methods and some simulation steps for design testing. Later the design will be processed for the programming of an FPGA on the LogicalStep board to observe how the logic circuit actually works in hardware. Since Lab1 starts so early in the term relative to the lecture material we will be doing some basic examination of two input gate functions

3.1 Prelab

No prelab work is necessary for LAB1. However students should familiarize themselves with the ECE-124 Lab Manual Outline and FPGA technology sections.

3.2 Lab1 Outline:

Attendance will be taken. Your Team partnerships will be settled during LAB1 and Group Numbers will be assigned for each team.

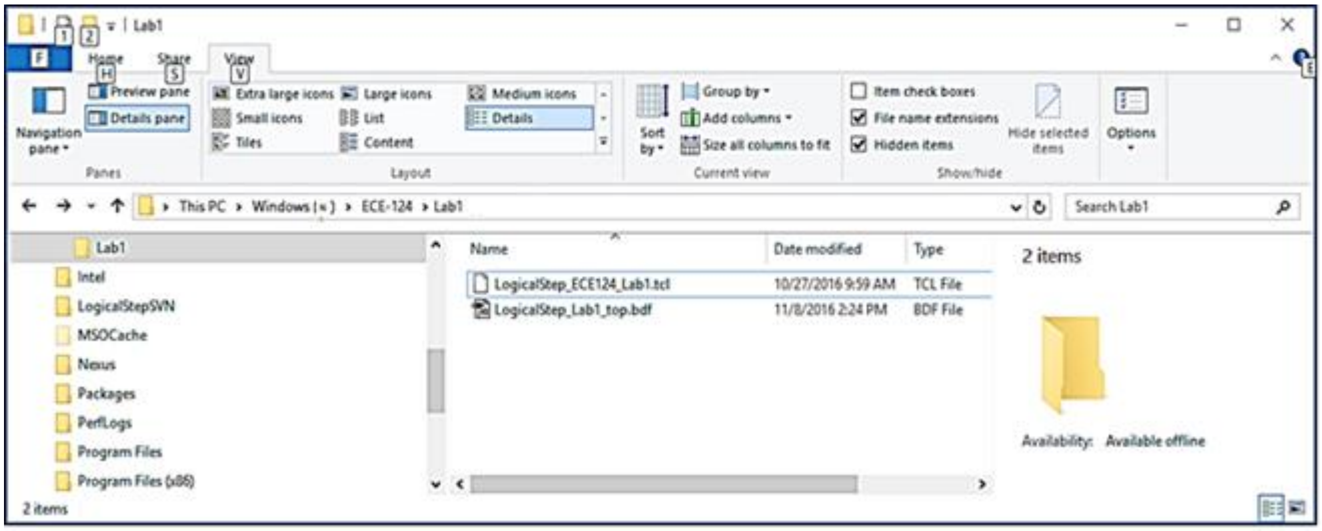
Lab 1 is composed of the following main categories:

1. Brief introduction to the laboratory, its equipment and the student conduct expected during Lab sessions.
2. Learning two Design entry methods (Schematic and VHDL) for small digital circuits.
3. Running Synthesis and Simulation processes on the circuits to check that they operate as expected.
4. Processing the FPGA designs into load files for downloading into the LogicalStep board and confirming that the hardware implementations function as expected.
5. Modifying the above circuit designs to provide new functionality and then test them.

3.3 Lab1 Activities

3.3.1 Starting Your Lab1 Project

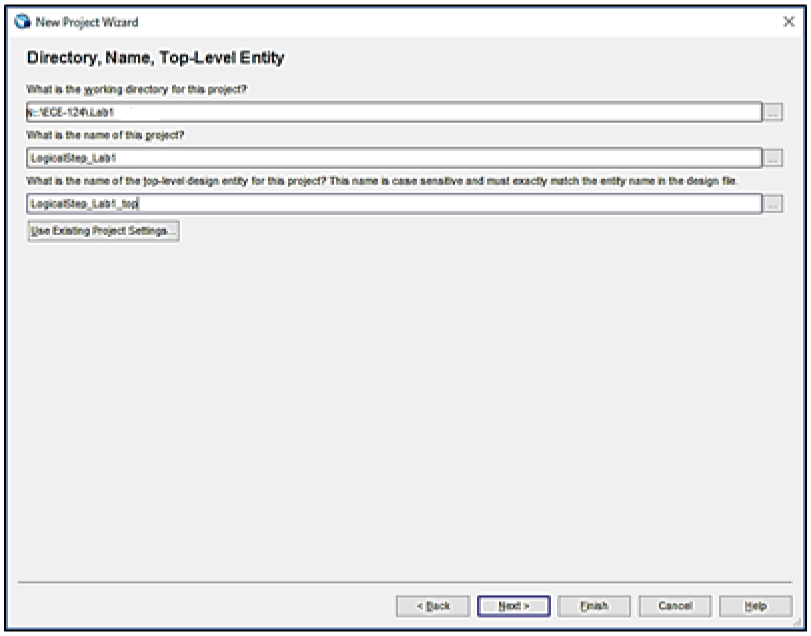
To begin your Lab1 project use the Windows10 File Explorer and browse to somewhere on a team members file space on the N: drive and create a folder called ECE124. Go to LEARN and download the Lab1 Zipped folder "Lab1" into the ECE124 folder on the N: drive. Extract the contents into a new Lab1 project folder. The new files are as follows in Figure 4:



Name	Date modified	Type
LogicalStep_ECE124_Lab1.tcl	10/27/2016 9:59 AM	TCL File
LogicalStep_Lab1_top.bdf	11/8/2016 2:24 PM	BDF File

Figure 4 Lab1: Starting Lab1 Project Folder

Start up the Altera Quartus Prime platform to begin a new project. Go to the FILE tab.


New Project Wizard
Directory, Name, Top-Level Entity
Use Existing Project Settings...
< Back
Next >
Finish
Cancel
Help

What is the working directory for this project?
N:\ECE-124\Lab1

What is the name of this project?
LogicalStep_Lab1

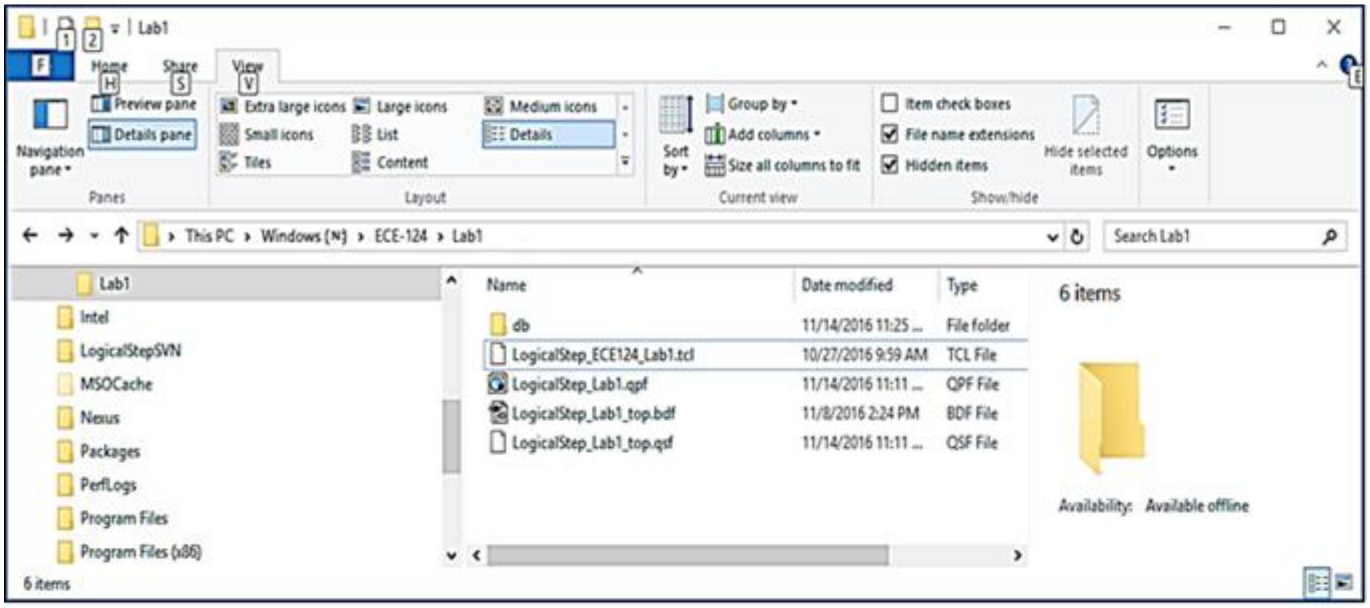
What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.
LogicalStep_Lab1_top

SELECT FILE>New Project Wizard. Click NEXT to go to the second slide.
The project parameters will now be entered as in Figure 5.
Project Folder: **N:/ECE-124/Lab1**
Project Name: **LogicalStep_Lab1**
Project Top Level: **LogicalStep_Lab1_top**

Click FINISH on the Wizard Dialog Window.

Figure 5 Lab1: FPGA Project Setup

After the setup has completed you should see the following in your project folder as in Figure 6:



Name	Date modified	Type
db	11/14/2016 11:25 ...	File folder
LogicalStep_ECE124_Lab1.tcl	10/27/2016 9:59 AM	TCL File
LogicalStep_Lab1.qpf	11/14/2016 11:11 ...	QPF File
LogicalStep_Lab1_top.bdf	11/8/2016 2:24 PM	BDF File
LogicalStep_Lab1_top.qsf	11/14/2016 11:11 ...	QSF File

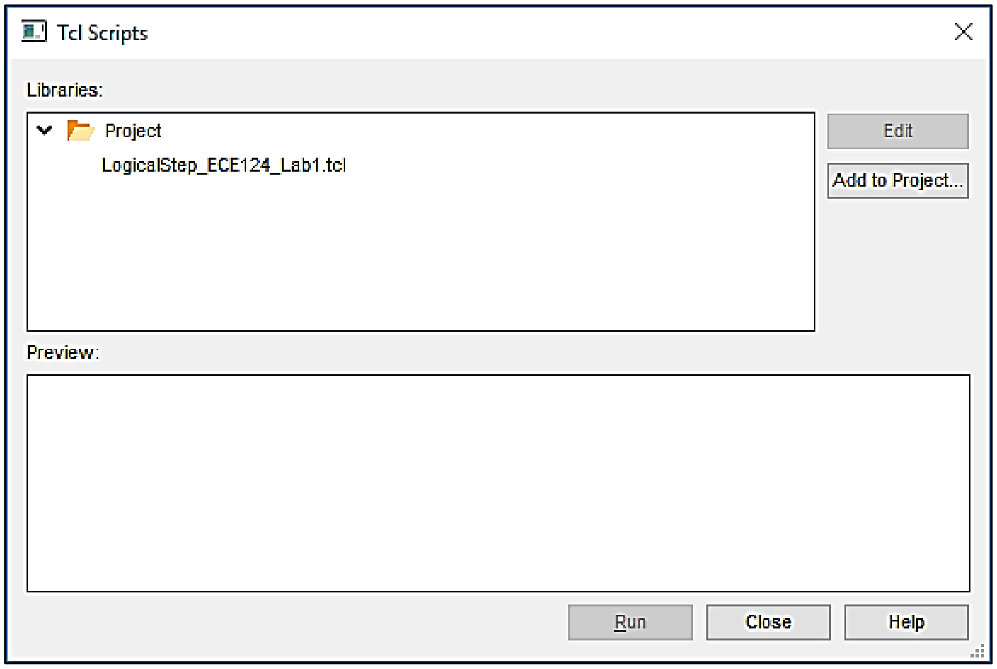
Figure 6 Lab1: Project Folder After Setup

Notice in the Project folder the file LogicalStep_Lab1.qpf (Altera Quartus Project File or QPF).

In later FPGA design work you can do one of two procedures to get back into your FPGA project to run in Quartus for Lab1.

1) You can browse to the Project folder and "double-click on the QPF file. This will launch Quartus and will load your FPGA design that you saved previously.
2) Alternatively, you can invoke the Quartus Prime v15.1 tools and then go to the FILE>Open_Project tab and then browse to the QPF file in your project folder and select the QPF file.

Next, in Quartus, the TCL script must be run to assign the FPGA device type that is being used for this lab and then pin assignments for the FPGA that are reserved for the LogicStep FPGA and finally the project LogicalStep_Lab1 is opened.

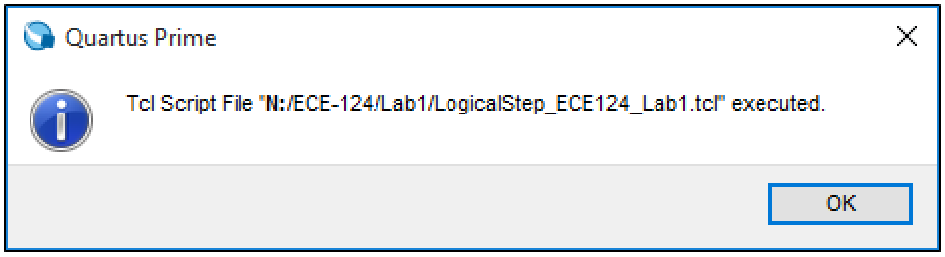


Go to the Tools TAB and SELECT Tools>Tcl Scripts. The following dialog box (Figure 7) should appear:

SELECT the TCL (pronounced as “tickle”) file and then click on the RUN button.

Figure 7 Lab1: TCL Script Invocation

The Figure 8 window should appear when it is finished.



```
Tcl Script File "N:/ECE-124/Lab1/LogicalStep_ECE124_Lab1.tcl" executed.
```

Figure 8 Lab1: TCL File Completed

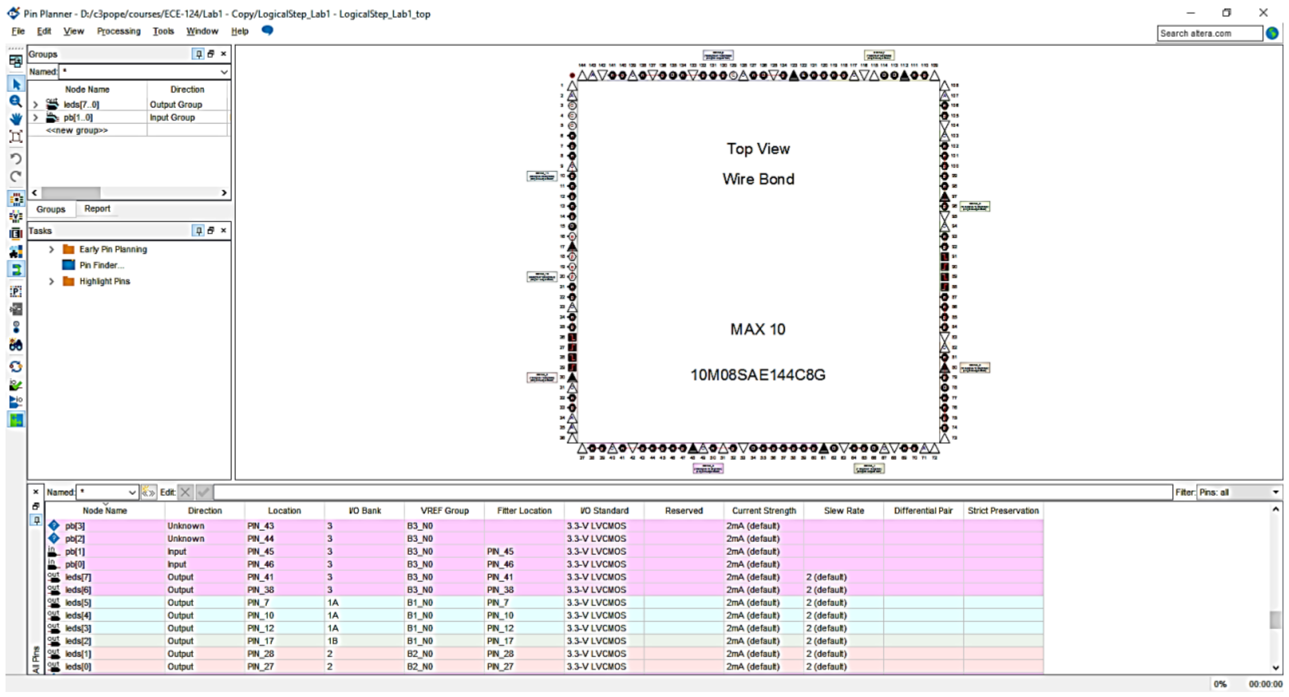
Click OK.

Then SELECT the CLOSE button on the TCL Script Dialog window.

NOTE that this TCL file will NOT have to be run again for the entire Lab1 project since the pin and FPGA Device assignments are established.

DEEP DIVE :

With the assignments made with the TCL file you can observe the signal pins that are used on the FPGA by calling up the Pin Planner utility. Go to the ASSIGNMENTS Tab and select the Pin Planner option. You should see something like the following:



Node Name	Direction
leds[7..0]	Output Group
pb[1..0]	Input Group
<<new group>>	

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
pb[3]	Unknown	PIN_43	3	B3_N0		3.3-V LVCMOS		2mA (default)			
pb[2]	Unknown	PIN_44	3	B3_N0		3.3-V LVCMOS		2mA (default)			
pb[1]	Input	PIN_45	3	B3_N0	PIN_45	3.3-V LVCMOS		2mA (default)			
pb[0]	Input	PIN_46	3	B3_N0	PIN_46	3.3-V LVCMOS		2mA (default)			
leds[7]	Output	PIN_41	3	B3_N0	PIN_41	3.3-V LVCMOS		2mA (default)	2 (default)		
leds[6]	Output	PIN_38	3	B3_N0	PIN_38	3.3-V LVCMOS		2mA (default)	2 (default)		
leds[5]	Output	PIN_7	1A	B1_N0	PIN_7	3.3-V LVCMOS		2mA (default)	2 (default)		
leds[4]	Output	PIN_10	1A	B1_N0	PIN_10	3.3-V LVCMOS		2mA (default)	2 (default)		
leds[3]	Output	PIN_12	1A	B1_N0	PIN_12	3.3-V LVCMOS		2mA (default)	2 (default)		
leds[2]	Output	PIN_17	1B	B1_N0	PIN_17	3.3-V LVCMOS		2mA (default)	2 (default)		
leds[1]	Output	PIN_28	2	B2_N0	PIN_28	3.3-V LVCMOS		2mA (default)	2 (default)		
leds[0]	Output	PIN_27	2	B2_N0	PIN_27	3.3-V LVCMOS		2mA (default)	2 (default)		

Figure 9 Lab1: FPGA PIN PLANNER

Earlier the top level file for this lab was downloaded from LEARN into the Lab1 project folder. The top level file is in schematic form (see Figure 10). Schematic entry methods for a simple set of gate-level functions will be the first part of this lab. Go to FILE Tab and SELECT File>Open and then browse to the LogicalStep_Lab1_top.bdf file (see Figure 6).

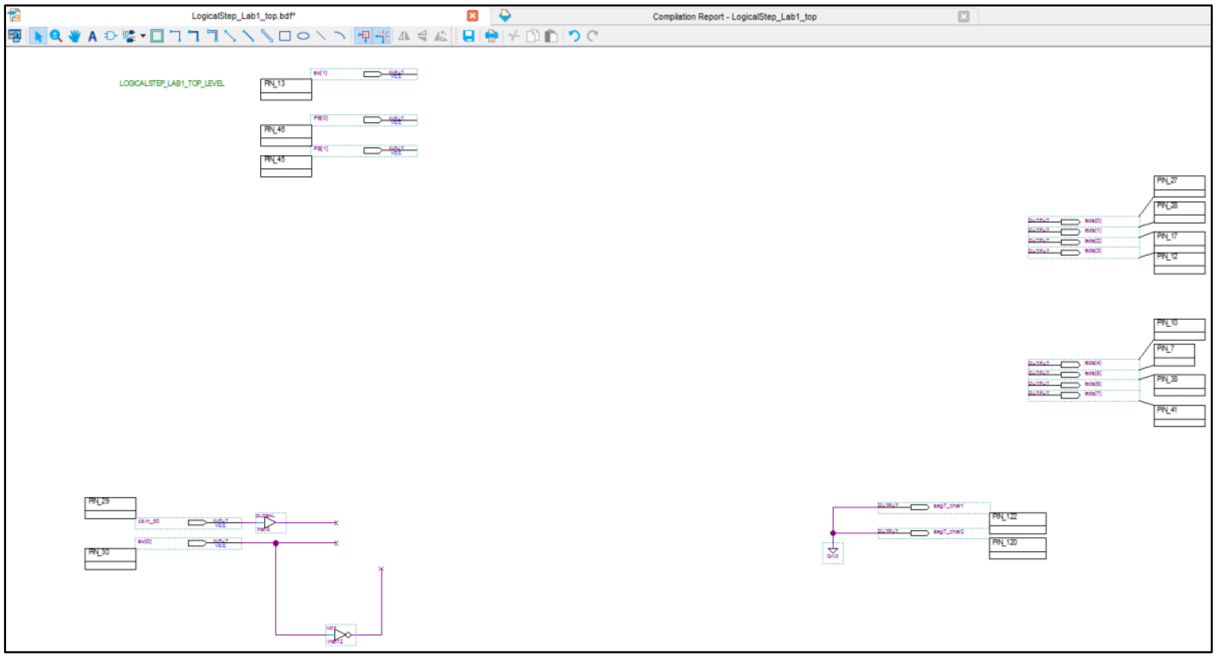

LogicalStep_Lab1_top.bdf*
Compilation Report - LogicalStep_Lab1_top
LOGICALSTEP_LAB1_TOP_LEVEL

Figure 10 Lab1: Starting Top Level Schematic

There are two pins for Push-Button Key Inputs, two pins for Switch inputs, eight pins for LED Outputs and a pin for a Clock Input provided in the Lab1 design. The Clock Input will be used later in the lab session.

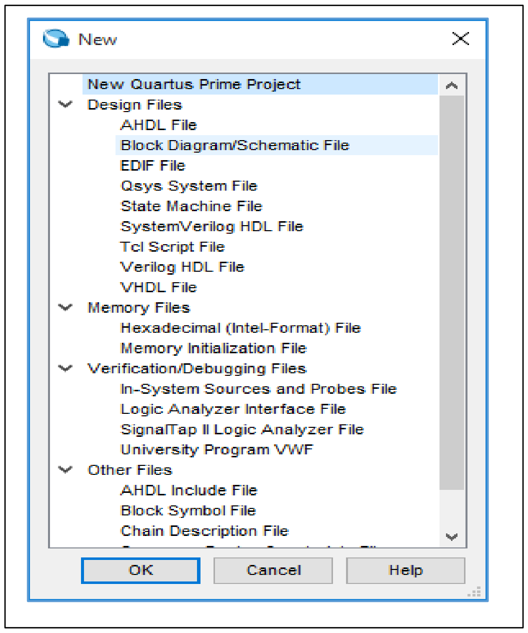


Figure 11 Lab1: New Schematic Creation for schem_gates Block

Now we can start adding some design hierarchy by creating functional blocks and then installing them at the top level design later. The first block that will be created will be of a schematic entry format. Go to the FILE Tab and select File>New. The dialog box as shown in Figure 11 will open. SELECT the Block Diagram/Schematic File. A blank schematic window will then open in Quartus. After it opens save the schematic file as “schem_gates.bdf” by going to the FILE Tab and selecting the File>Save As option.

To insert schematic symbols on the schem_gates sheet RIGHT-CLICK anywhere on the schematic sheet (as in Figure 12) and a dialog box should appear. SELECT the Insert>Symbol option.

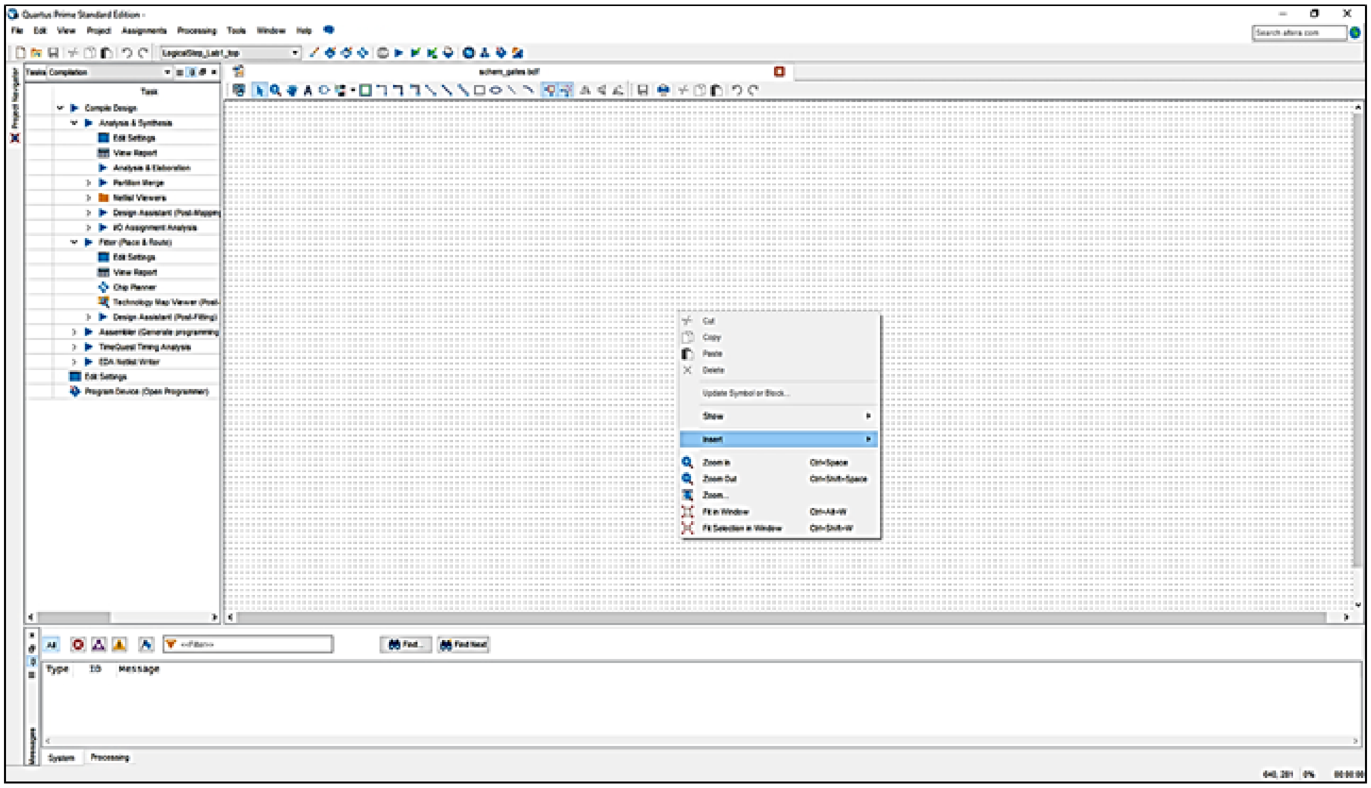
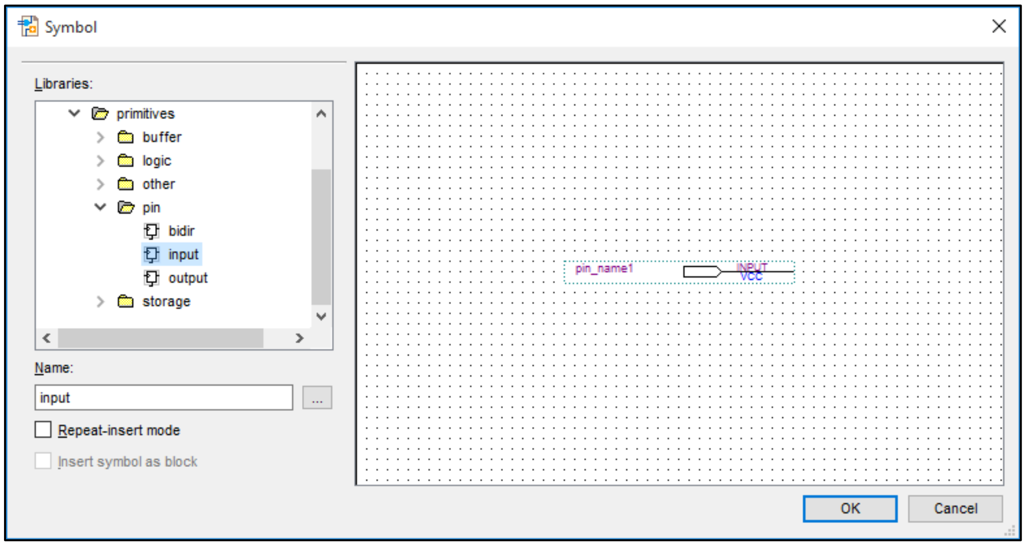


Figure 12 Lab1: Insertion of Library Symbols into schem_gates Schematic

To this schematic we will add INPUT pins on the left and OUTPUT pins on the right (typical for schematic convention for readability). So within the Symbols Dialog box that will appear browse to the altera/quartus libraries and then SELECT the Primitives/pin folder. Here you can select the pins as required and place them on the schematic sheet.



Name: input

☐ Repeat-insert mode

☐ Insert symbol as block

Figure 13 Lab1: Locating Pins in Symbol Library for schem_gates Block

Add Inputs and Outputs to the schematic sheet as shown below in Figure 14:

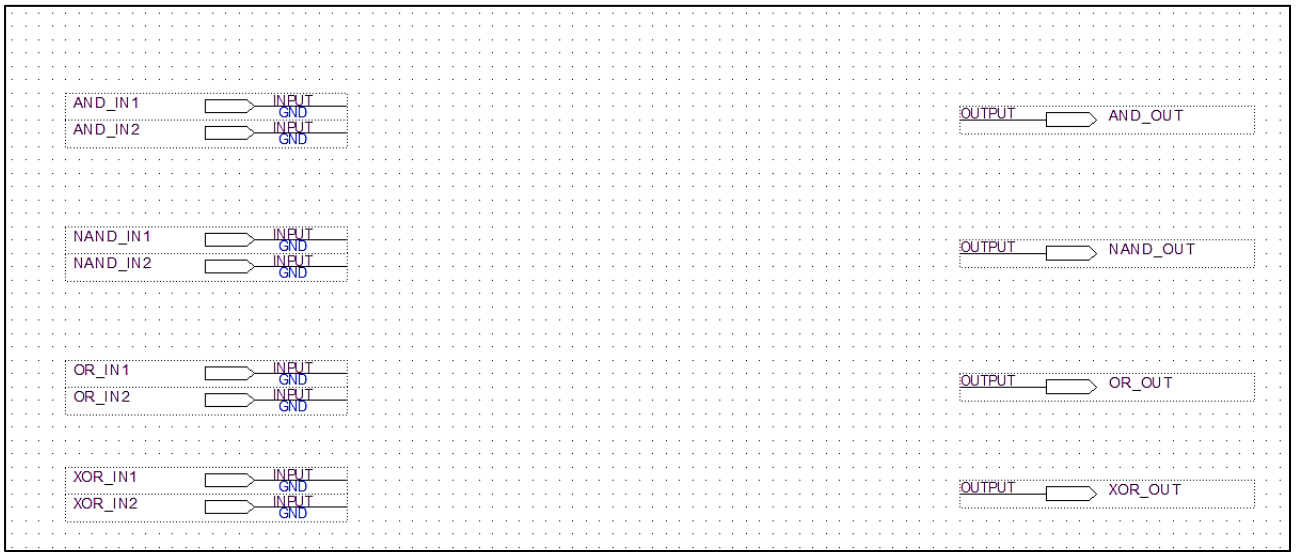


Figure 14 Lab1: Insertion of I/O Pins into schem_gates Schematic

List of inputs:
AND_IN1, AND_IN2, NAND_IN1, NAND_IN2, OR_IN1, OR_IN2, XOR_IN1, XOR_IN2
List of outputs :
AND_OUT, NAND_OUT, OR_OUT, XOR_OUT
Name each of the pins as in the lists above. (Double-click each pin and modify its Name property).

After placing and naming the pins on the schematic sheet return again to the symbol Libraries for logic gates in the Primitives/Logic folder.

We will only be using 2 input gates for this lab. Below is a truth table for the gates that are to be entered. Also notice how in the INPUTS that bit1 changes at half the rate of bit0.

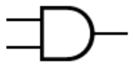
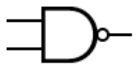
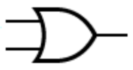
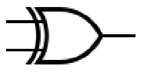
IN 1	IN 0	AND	NAND	OR	XOR
0	0	0	1	0	0
0	1	0	1	1	1
1	0	0	1	1	1
1	1	1	0	1	0
GATE SYMBOL →					

Figure 15 Lab1: Look-up Table for Gate Logic Functions

You must locate the basic 2 input gate functions (AND, NAND, OR, XOR) from the altera/quartus libraries and insert them into this schematic. Below in Figure 16 is an example of the two input AND gate.

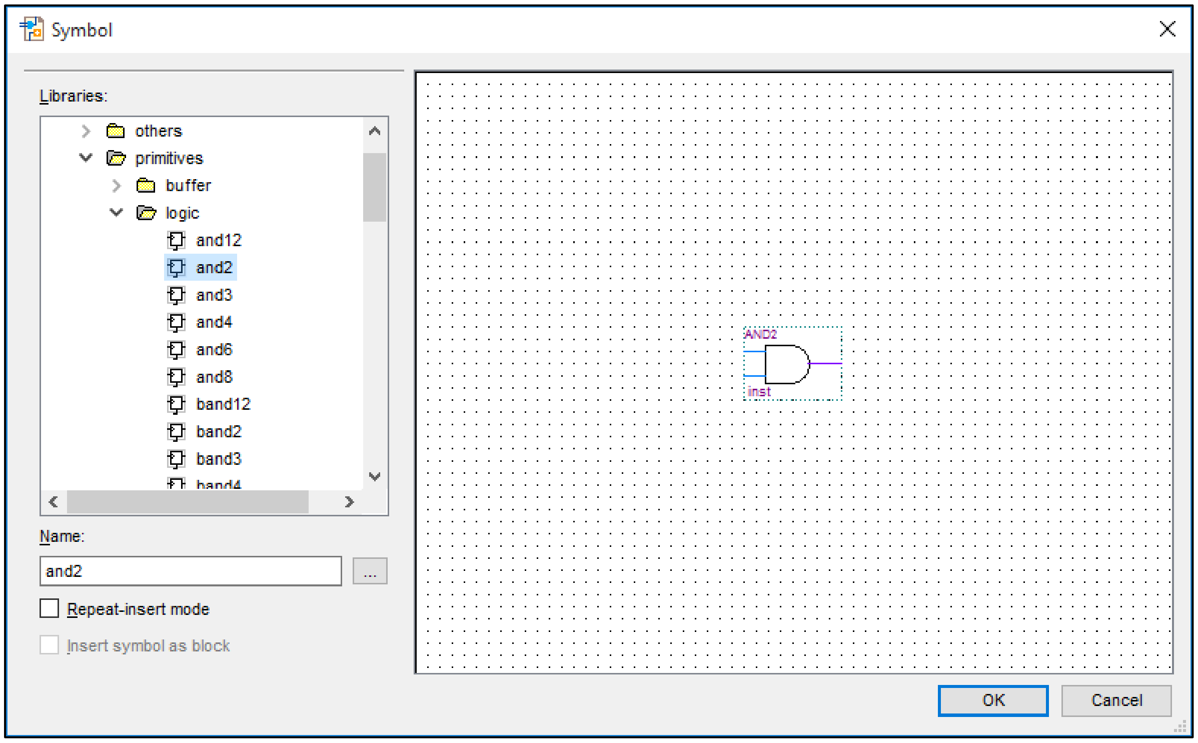


Figure 16 Lab1: Locating Gates in Symbol Library for schem_gates Block

Connect the input pins to the gate inputs and the output pins to the gate outputs as shown below in Figure 17. Use the Orthogonal Node Tool (highlighted below)). After the schematic is drawn save the file. Go to the FILE Tab and select File>Save.

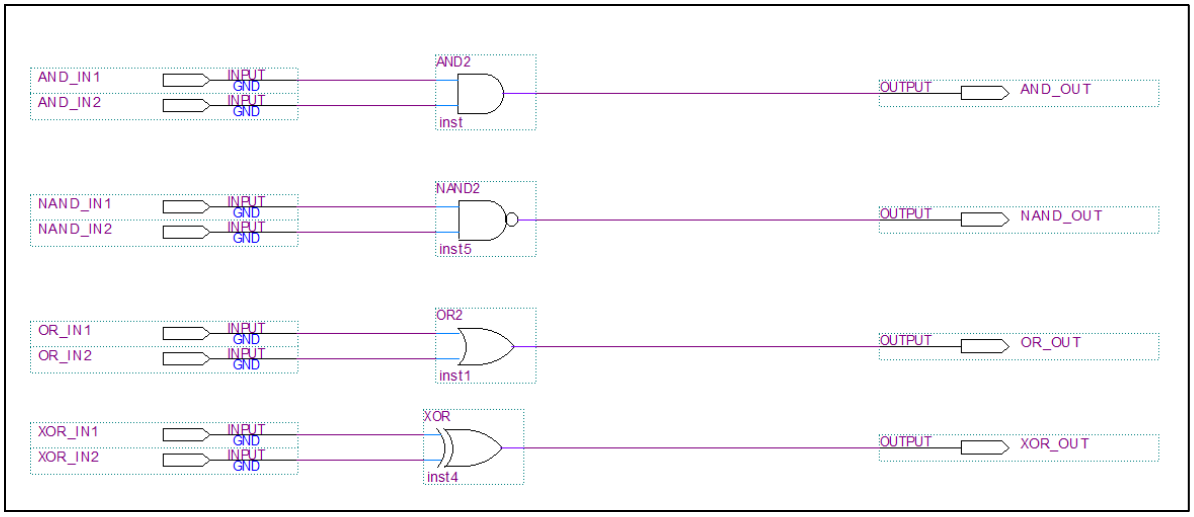
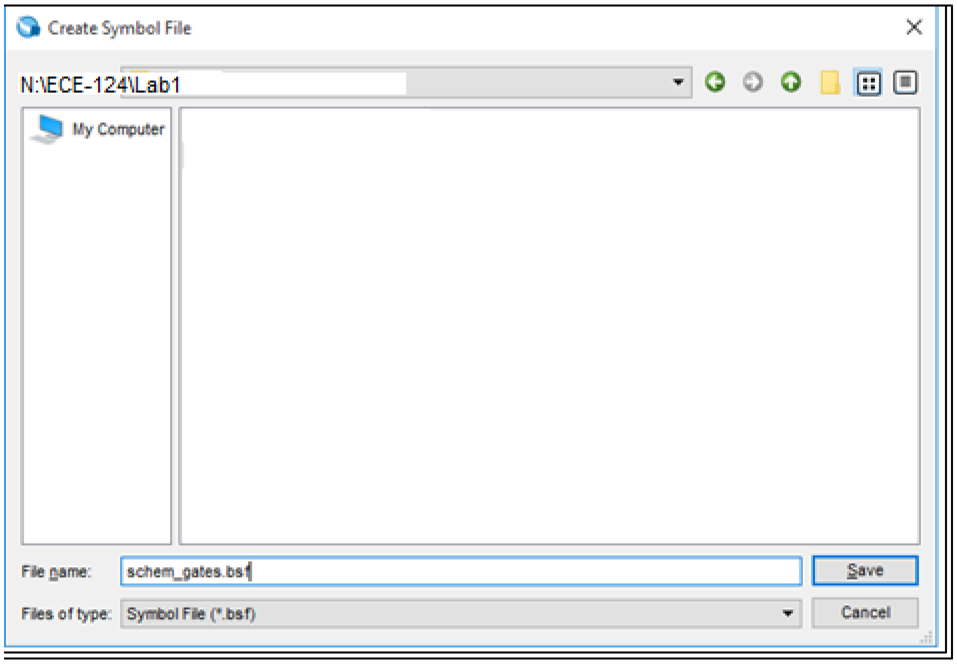


Figure 17 Lab1: Connecting Gates in schem_gates Block

Now that a schematic design file has been created we must make a symbol for it so that we can add its symbol to the top level schematic (LogicalStep_Lab1_top). Go to the FILE Tab.

SELECT the File>Create / Update> Create Symbol Files for Current File option. A Window like the one in Figure 18 appears. The symbol filename option for “schem_gates.bsf” should be visible.



Click Save.

Close the schem_gates design file. Return to the LocialStep_Lab1_top schematic as in Figure 19.

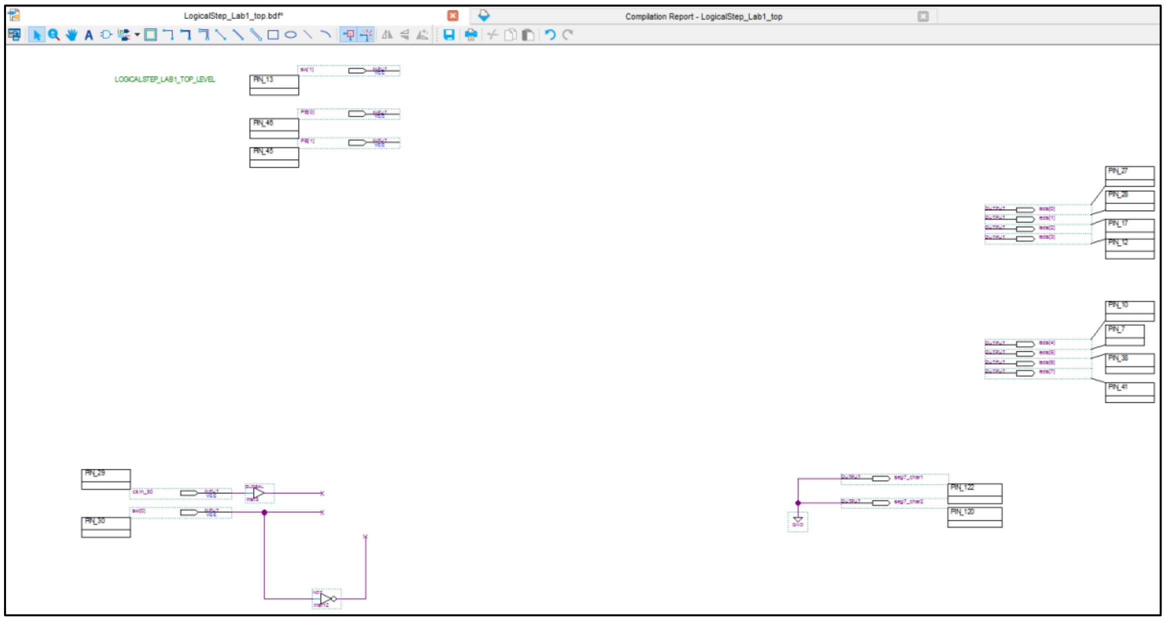

LogicalStep_Lab1_top.bdf*
Compilation Report - LogicalStep_Lab1_top
LOGICALSTEP_LAB1_TOP_LEVEL

Figure 19 Lab1: Top Level Schematic Before Adding Symbols

On the LogicalStep_Lab1_top schematic design the new schem_gates.bsf symbol (block that was just created in the previous step) will be inserted. To insert a symbol RIGHT-CLICK anywhere on the top level schematic. SELECT the Insert>Symbol option.

The Symbol Dialog box appears (see Figure 20). Expand the Project folder and browse to the schem_gates file.

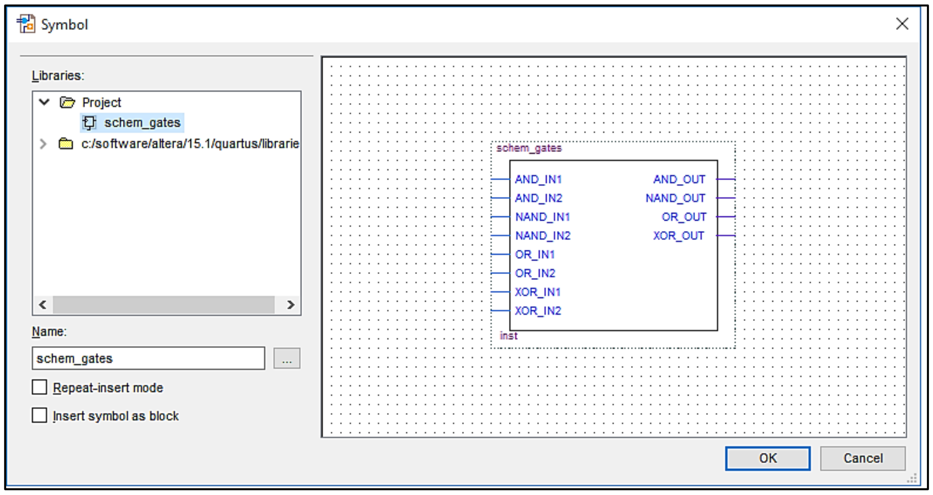


Figure 20 Lab1: Selecting the schem_gates Symbol for Insertion

Click the OK Button.

Place the symbol on the schematic.

After it is added to the top level schematic sheet connect its symbol pins to the Push-Button port pins and to the output pins that drive LED's on the LogicalStep board. Use the ORTHOGONAL NODE TOOL as before.

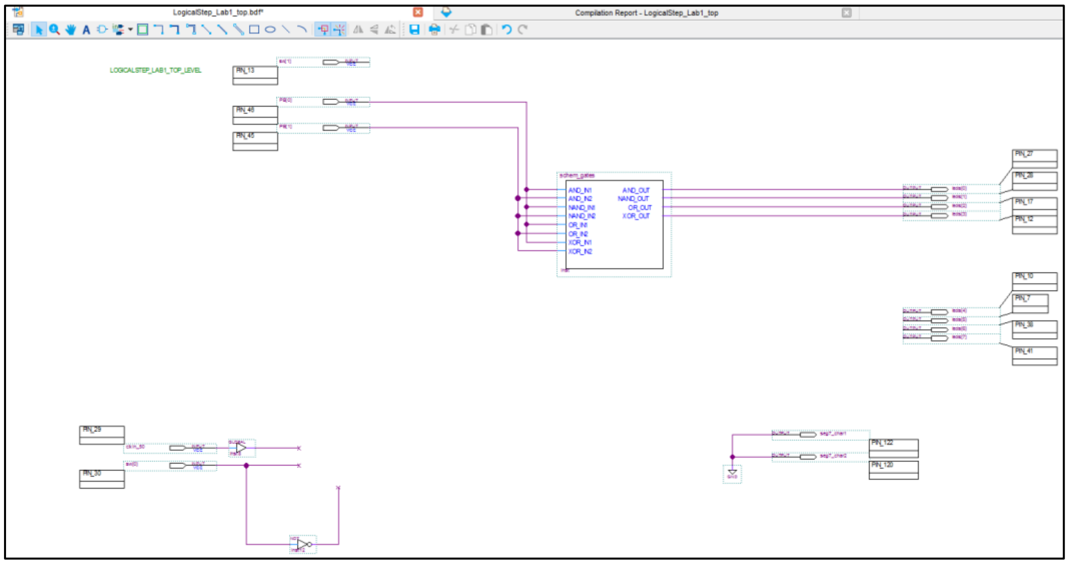


Figure 21 Lab1: Hooking Up Pins to schem_gates Block

SELECT File>Save to save the top schematic design as in Figure 21.

Now we are ready to do some functional testing by simulation.

3.3.2 Functional Simulations

3.3.2.1 Preparation for Functional Simulation – Analysis and Synthesis

Within Quartus go to the PROCESSING Tab. SELECT Start>Start Analysis and Synthesis" process to process the design into a gate level logic file for simulation purposes.

3.3.2.2 Opening a Simulation Window

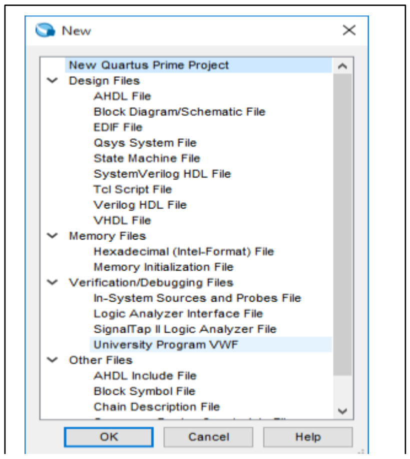


Figure 22 Lab1: Starting a New Simulation

Now under the FILE Tab SELECT the FILE>New and then SELECT the University Program VWF utility to open a Functional Simulation window (see Figure 22). This will be used to illustrate the gate-level functionality in a visual manner. Nodes (nets) will be inserted from the design (see below) for control and observation.

Click OK.

A new window will open like the one shown in Figure 23.

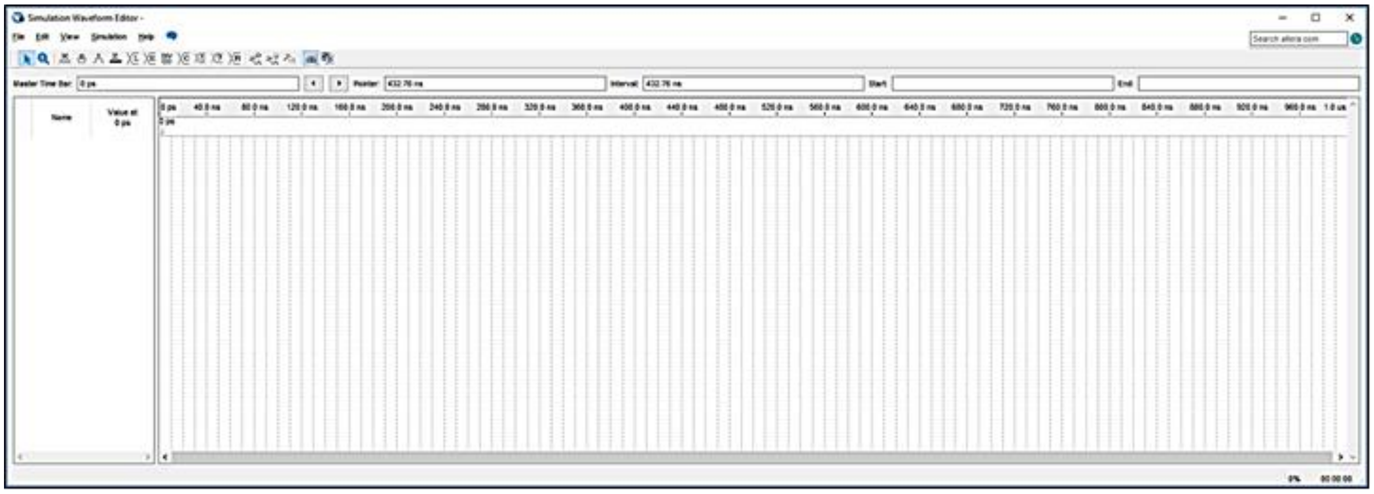
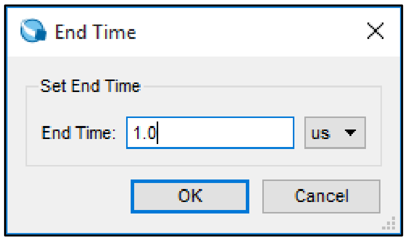


Figure 23 Lab1: Simulation Window

Set the Time scale by going to the Simulator Window EDIT Tab and SELECT the Edit >End Time option.

Then a window like the one shown in Figure 24 will appear.



End Time: 1.0 us

Set it to 1 usec. Click the OK button.

Figure 24 Lab1: Setting Simulation End Time

3.3.2.3 *Adding Nodes to the Simulation Window*

For the simulation only the two Push-Button inputs and the first four LED outputs will be inserted into the simulator.

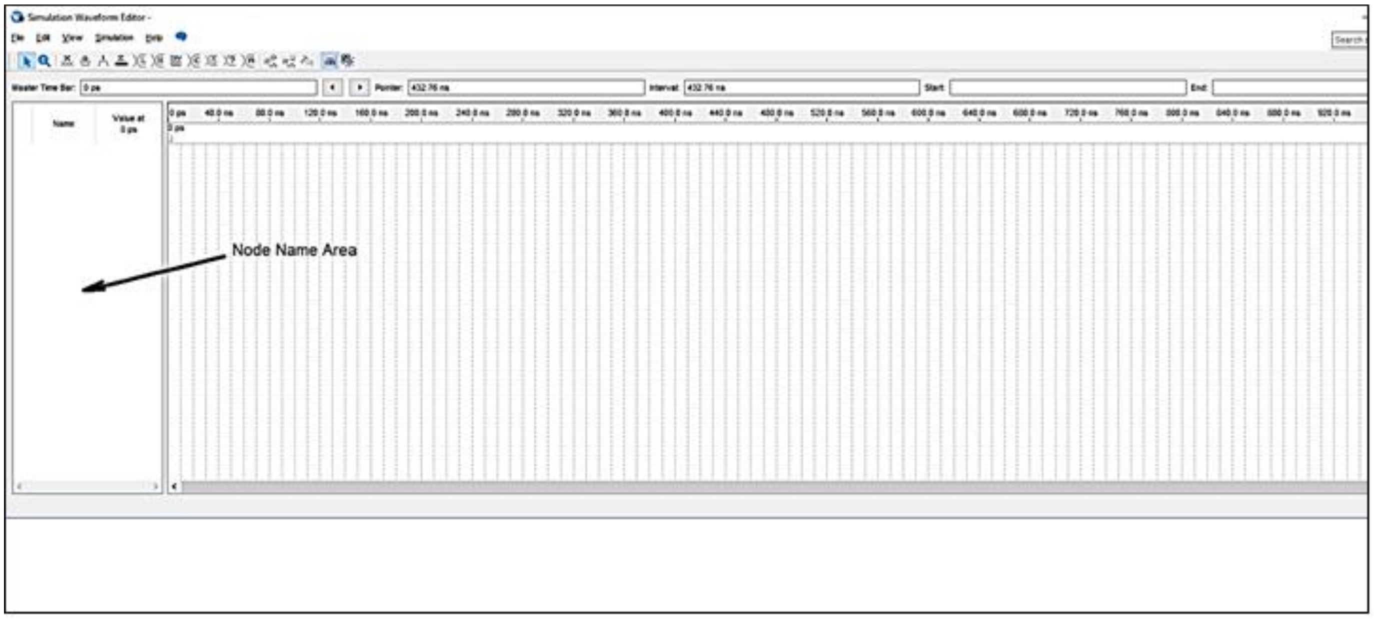
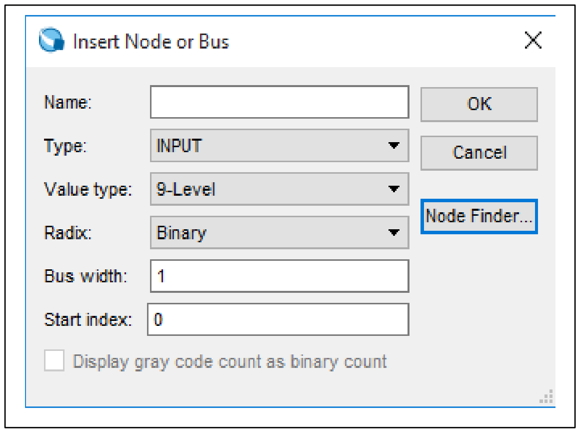


Figure 25 Lab1: Adding Nodes to Simulator Window

Double-Click the Node NAME area of the Simulator Window and the following Dialog window will appear as in Figure 26 below:

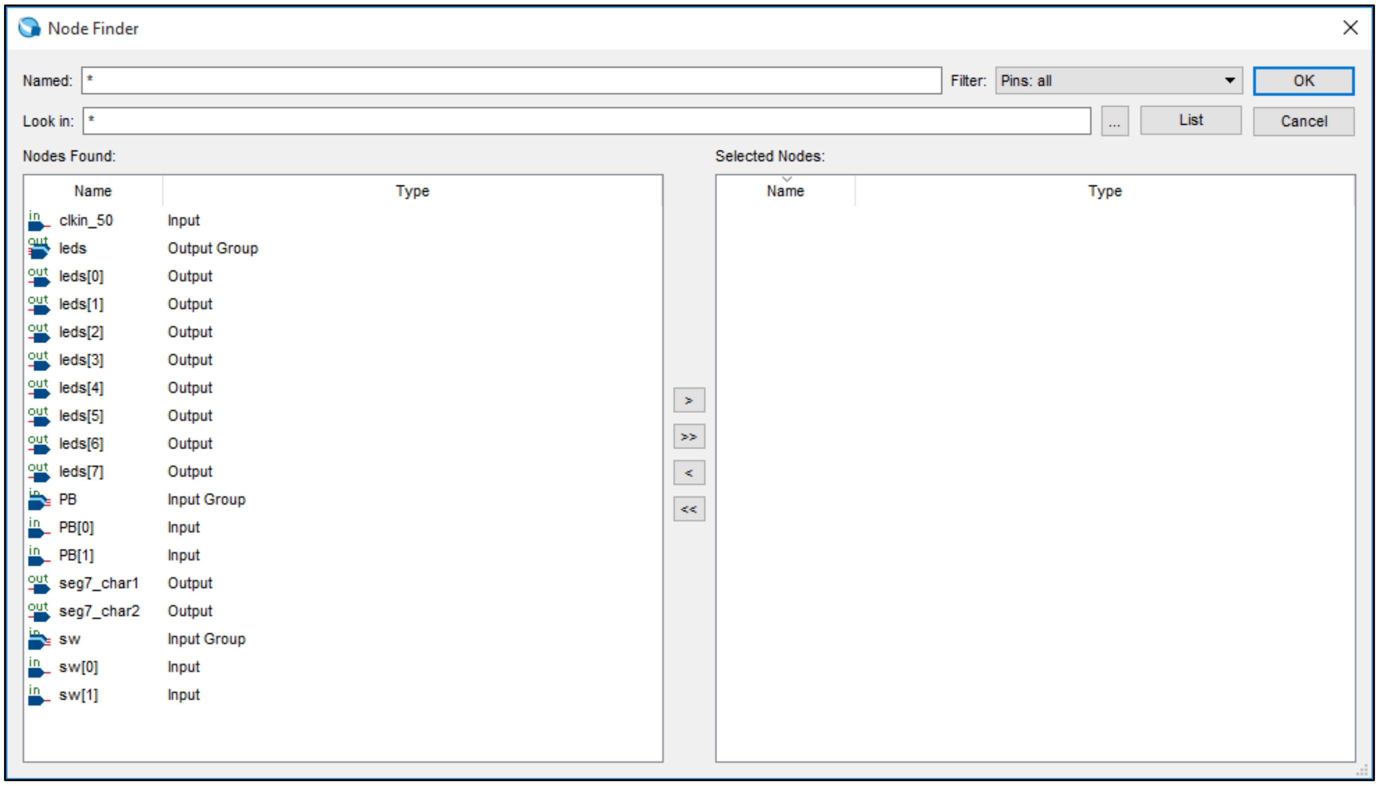


Name:
Type: INPUT
Value type: 9-Level
Radix: Binary
Bus width: 1
Start index: 0
Display gray code count as binary count

Click on the Node Finder Button for faster node identification and insertion. The Node Finder Dialog Box will appear. This will allow you to browse the synthesized design for nodes (nets) to probe for the Functional Simulation.

Figure 26 Lab1: Calling up Simulator Node Finder

With the node FILTER setting set to "Pins all" Click on the LIST button as shown in Figure 27. The list of pins from the design will appear as shown.



Named: *
Filter: Pins: all
Look in: *

Nodes Found:

Name	Type
clkin_50	Input
leds	Output Group
leds[0]	Output
leds[1]	Output
leds[2]	Output
leds[3]	Output
leds[4]	Output
leds[5]	Output
leds[6]	Output
leds[7]	Output
PB	Input Group
PB[0]	Input
PB[1]	Input
seg7_char1	Output
seg7_char2	Output
sw	Input Group
sw[0]	Input
sw[1]	Input

Figure 27 Lab1: Listing Pins with Node Finder

SELECT the following pins **<u>in the order</u>** specified: PB[0], PB[1], leds[0], leds[1], leds[2], leds[3], leds[4], leds[5], leds[6], leds[7]. After selection of the group click on the '>' button to copy them to the Selected Nodes window. Then click on the OK button and again Click on the OK button on the Node_Finder Dialog Box (Figure 26).

3.3.2.4 Adding Stimulus to the Input Nodes

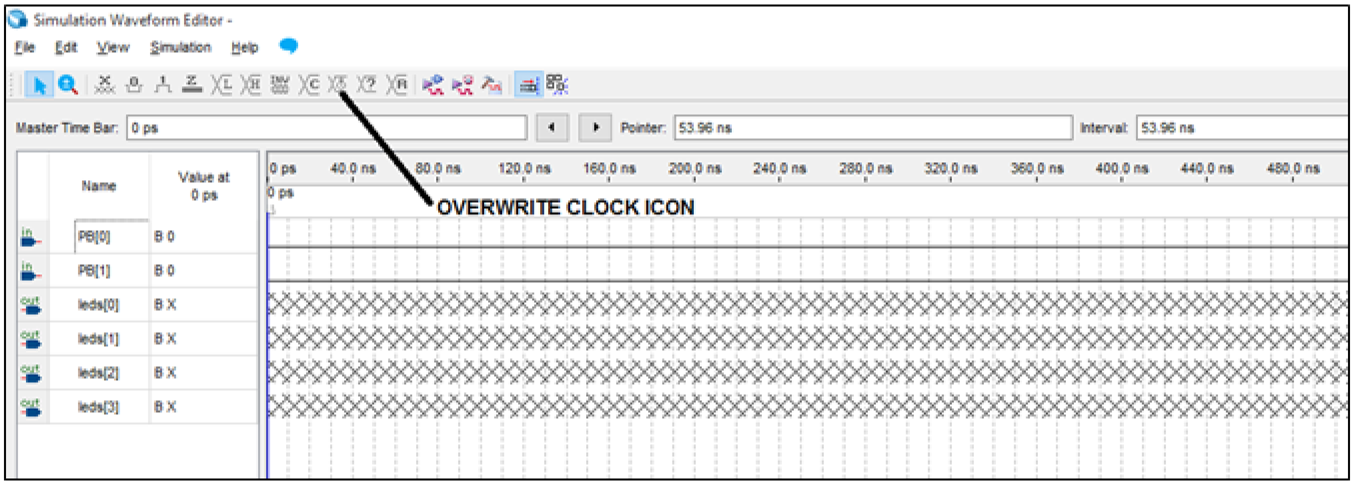

Simulation Waveform Editor -
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer: 53.96 ns
Interval: 53.96 ns
OVERWRITE CLOCK ICON
Name
Value at 0 ps
PB[0] B 0
PB[1] B 0
leds[0] B X
leds[1] B X
leds[2] B X
leds[3] B X
0 ps 40.0 ns 80.0 ns 120.0 ns 160.0 ns 200.0 ns 240.0 ns 280.0 ns 320.0 ns 360.0 ns 400.0 ns 440.0 ns 480.0 ns

Figure 28 Lab1: Adding Node Stimulus

To provide the stimulus waveforms to the input pins SELECT input PB [0] in the NAME column and then Click on the OVERWRITE CLOCK button (shown above in Figure 28) and enter a period of 500 nseconds. Then similarly, for the PB[1] input SELECT the PB[1] in the NAME column and then click on the OVERWRITE CLOCK button (shown above in Figure 29) and enter 1000 nsec for the period value. These two entries should create waveforms for stimulus as shown in Figure 29.

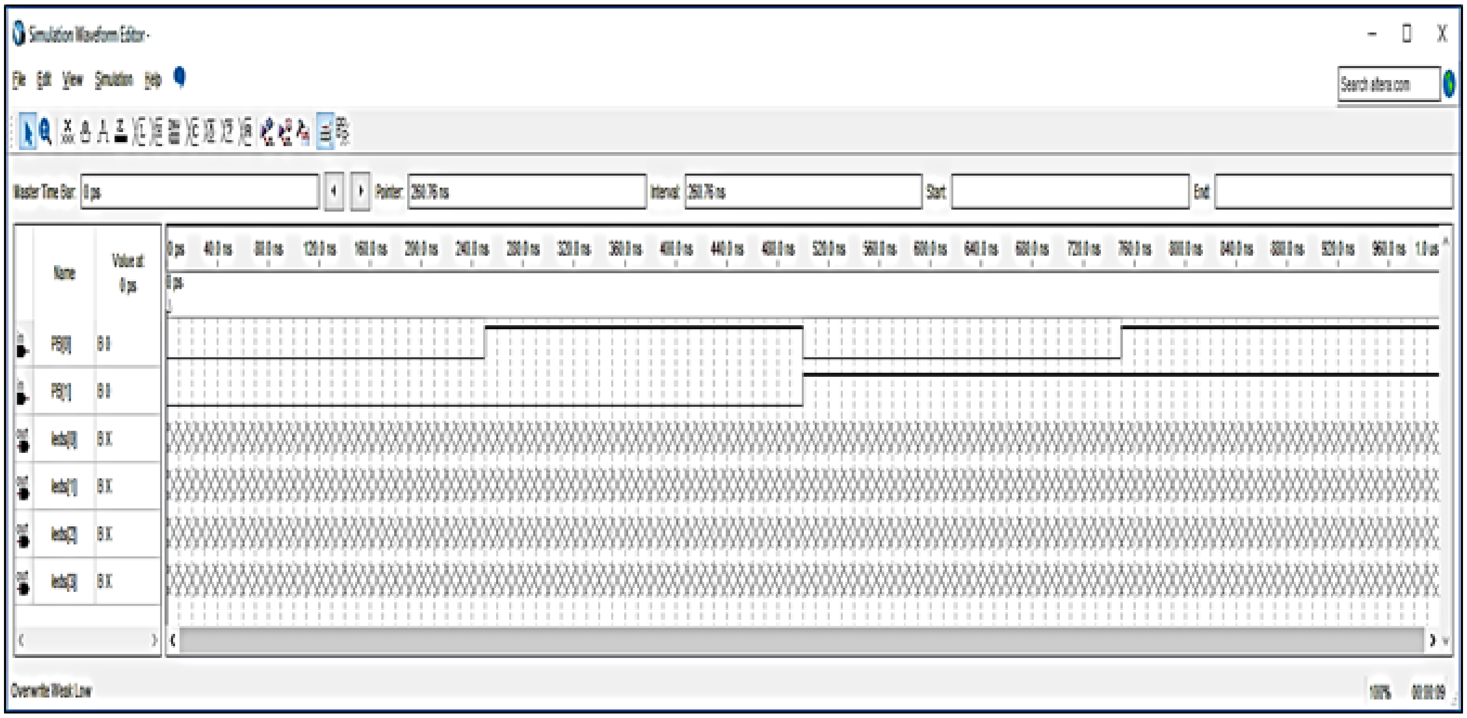


Figure 29 Lab1: Adding more Stimulus

The stimulus is now created. But the outputs are still undefined since the simulation has not yet been run. Save the Simulation file as waveform.vwf by going to the Simulator window FILE Tab and SELECT the File>Save option.

3.3.2.5 Running the Functional Simulation

On the Simulator window SIMULATION Tab and SELECT the Simulation>Run Functional Simulation option. The simulation results should look like the screenshot in Figure 30 when it has completed.

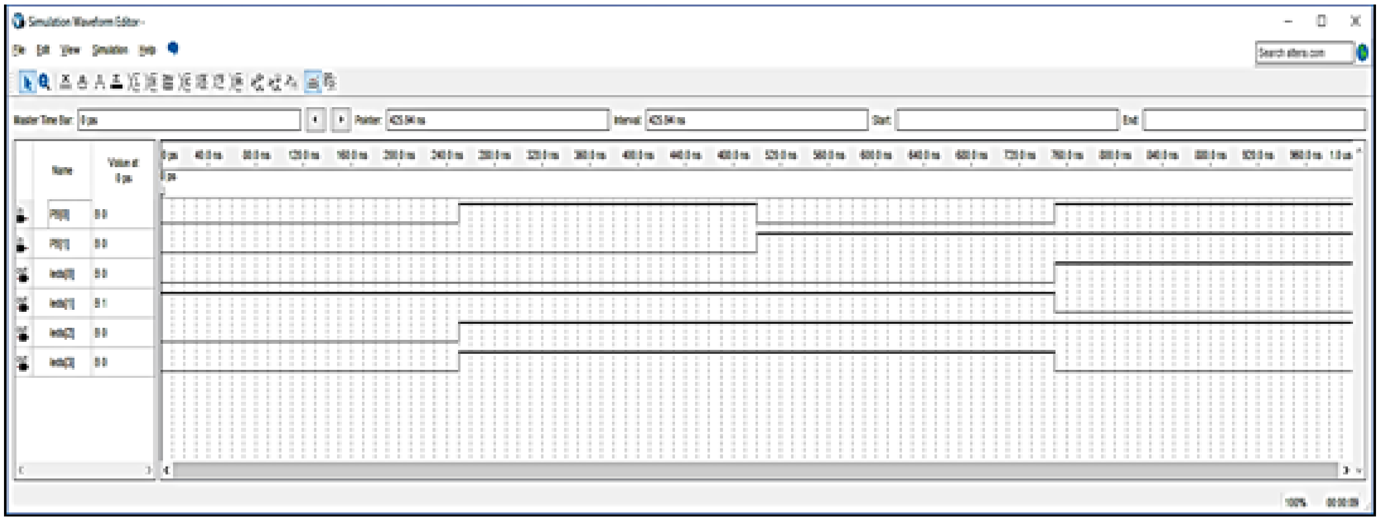


Figure 30 Lab1: Simulation Complete

Recall that the leds[0] pin is connected to the AND_OUT pin of schem_gates in the design. Similarly leds[1] is connected to NAND_OUT, leds[2] with OR_OUT and leds[3] with XOR_OUT. Confirm that the simulation waveforms follow the gate truth tables covered earlier.

3.3.3 Compensating for Active-LOW PB inputs

At this point the functional simulation has proven the design functionality. We can now close the Simulator Windows.

Moving now to a real world operation of the FPGA on the LogicalStep board there will have to be a small modification added to the schematic to adjust for conditions external to the FPGA. We want the inputs to the logic blocks to match the logic levels that were defined as in the simulations. On the LogicalStep board when each PB key is pressed the signal state is ‘0’ for a closed condition. But we want a logical ‘1’ to arrive at the appropriate schem_gates block input when the PB key is pressed.

Therefore we must add inverters to PB[0] and PB[1] pin inputs. The inverters will ensure that this compensation happens. Go to the altera/quartus libraries again to insert the "not" gate from the Primitives/logic folder for each of the inverters. Insert and connect them as shown in Figure 31.

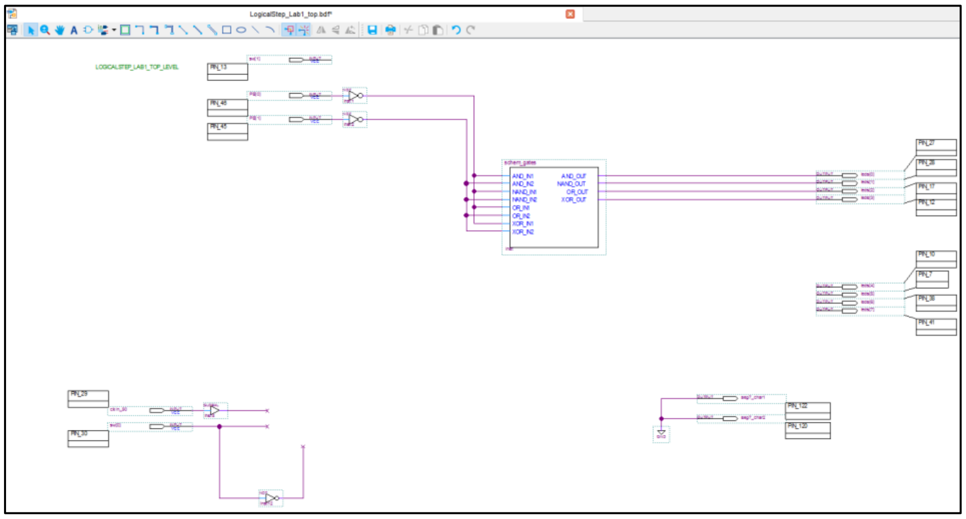
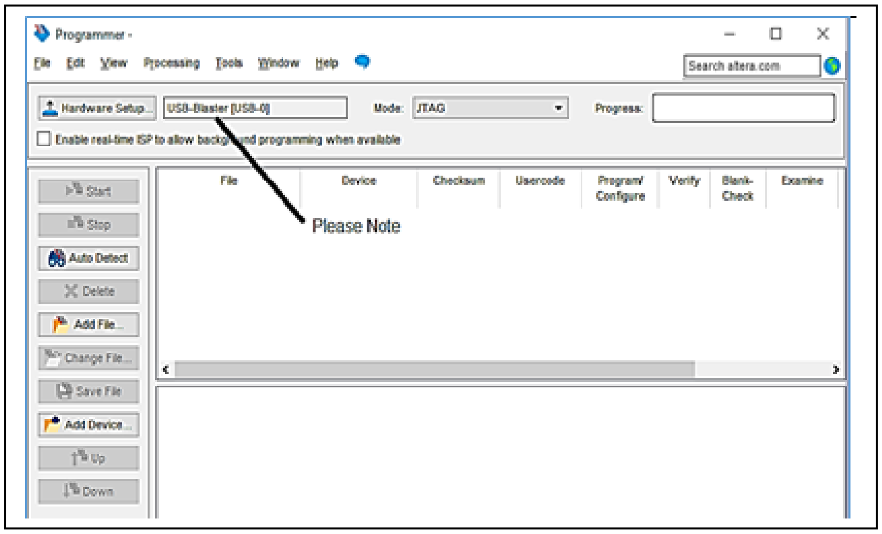


Figure 31 Lab1: Inserting Inverters after PB Key Inputs

3.3.4 FPGA Design Compilation and Download

Now the FPGA Compilation process will be executed. Go to the PROCESSING Tab and SELECT the Processing>Start Compilation option. When the FPGA compilation finishes and if no compilation errors are found (ignore any warnings) then an FPGA load file can be downloaded into the FPGA.

Figure 32 Lab1: Quartus FPGA Programmer



Use the Quartus Programmer utility to download your design file into the LogicalStep board FPGA by going to the TOOLS Tab and SELECT the Tools>Programmer option.

A Programmer dialog window will open as shown in Figure 32. Click the ADD File button.

NOTE: If the LogicalStep board is connected the **USB Blaster** should be seen beside the Hardware Setup field (Otherwise speak with the Instructor).

A Select Programming File window will open (Figure 33). Browse to the output_files folder.

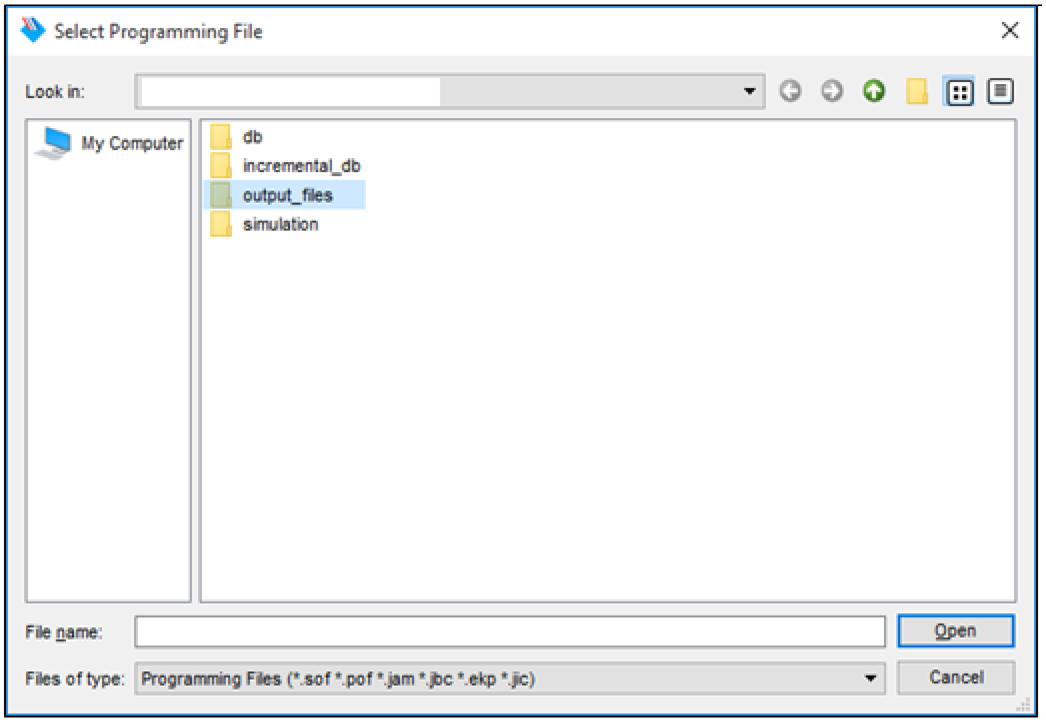
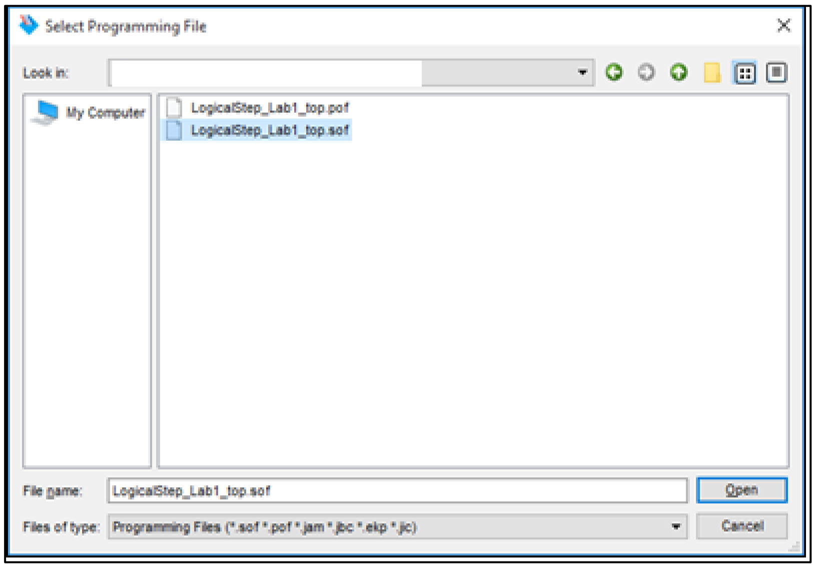


Figure 33 Lab1: Quartus FPGA Programming File Browser

Select the LogicalStep_Lab1_top.**sof file** as shown in Figure 34.

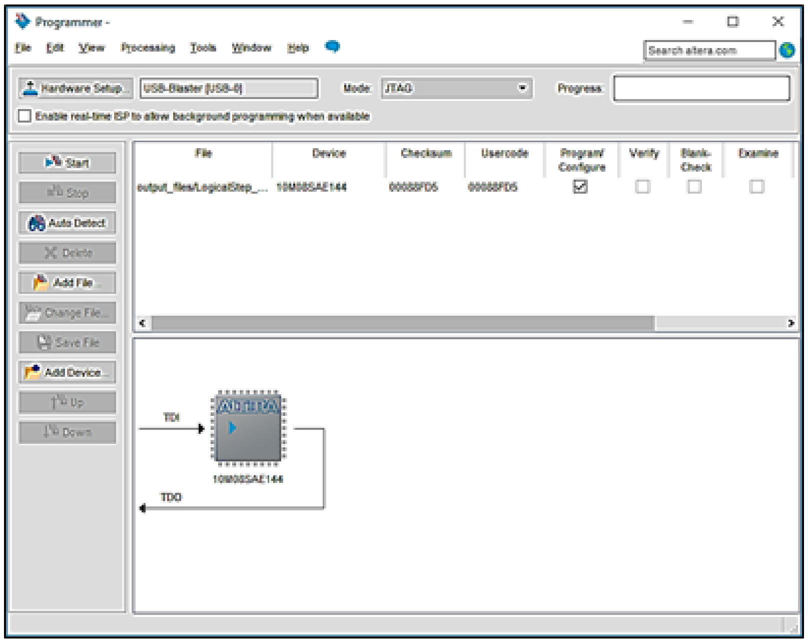


WATCH OUT! **(NOT the .pof file).**

Click Open.

Then requested file will then show up in the Programmer window as shown below in Figure 35.

Figure 34 Lab1: Selecting the FPGA Programming File (.sof)



Hardware Setup... USB-Blaster [USB-0] Mode: JTAG Progress:

☐ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
output_files/LogicalStep_...	10M08SAE144	00088FD5	00088FD5	☑	☐	☐	☐

Figure 35 Lab1: Starting the Quartus FPGA Programming

Click on the START button (see Figure 35) to begin the FPGA download. The progress window should indicate 100% after the downloading is completed.

After downloading you can test your FPGA design by using board-level inputs (PB [1..0] Keys) and also observing the outputs (LEDs) according to your logic gate truth tables.

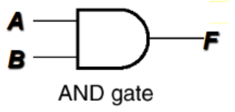
3.3.5 VHDL Design Entry

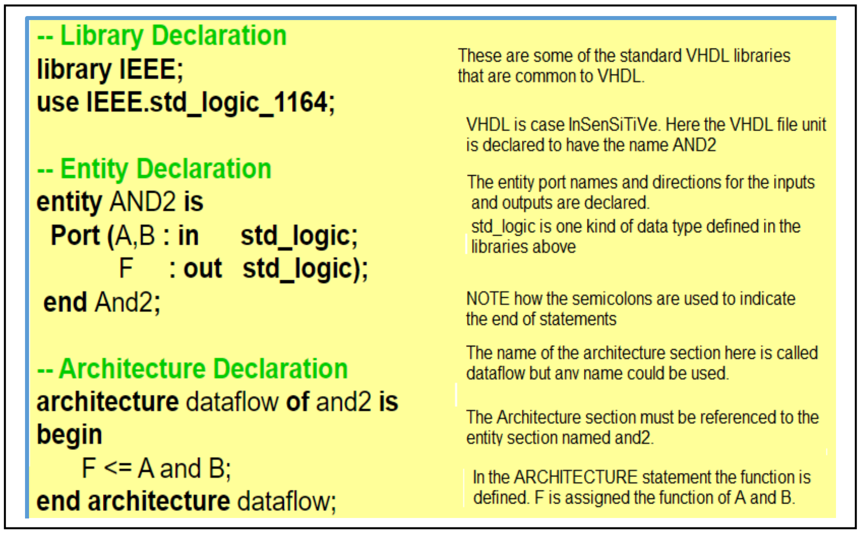
The main areas that will be covered on VHDL in this course have to do with the parts of the VHDL design unit (hardware block). For ECE-124 there are just two main areas of focus. These are the ENTITY and ARCHITECTURE constructs. (Library declarations are also required, as shown in Figure 36 below but just a few variants of these library declarations will be provided to you for use):

1. ENTITY: declares the design unit name and the ports (which are inputs and outputs of the entity or design unit) associated with it. Each port name, type (input or output) and width (number of bits) is declared in the entity.

2. ARCHITECTURE: specifies the actual functionality of the entity. Notice that the entity has no information about how the hardware block uses the inputs or how to produce the outputs - that is the role of the architecture associated with the entity.

   Figure 36 is an example of a complete VHDL unit for a two input AND gate:





```
-- Library Declaration
library IEEE;
use IEEE.std_logic_1164;

-- Entity Declaration
entity AND2 is
  Port (A,B : in     std_logic;
          F    : out  std_logic);
 end And2;

-- Architecture Declaration
architecture dataflow of and2 is
begin
      F <= A and B;
end architecture dataflow;
```

These are some of the standard VHDL libraries that are common to VHDL.

VHDL is case InSenSiTiVe. Here the VHDL file unit is declared to have the name AND2

The entity port names and directions for the inputs and outputs are declared.
std_logic is one kind of data type defined in the libraries above

NOTE how the semicolons are used to indicate the end of statements

The name of the architecture section here is called dataflow but anv name could be used.

The Architecture section must be referenced to the entity section named and2.

In the ARCHITECTURE statement the function is defined. F is assigned the function of A and B.

Figure 36 Lab1: VHDL Example for a Simple AND Gate

Within the Architectural construct there are two styles used to describe the functionality of a VHDL design unit. These are:

1. Behavioral: where the relation between input and output is declared using logical equations.
2. Structural: where you can use previously created entities in your design unit as components. For example if you built an adder unit you can use it as a component in designing a microprocessor.

For Lab1 we will just be using the Behavioral style.
The VHDL design entry method within Quartus will now be covered. Similar to how we created the design for the schem_gates block we will now create a VHDL design block.

Returning to the LogicalStep_Lab1_top design in Quartus go to the FILE Tab and Select File>New.

The dialog box shown below in Figure 37 will open:

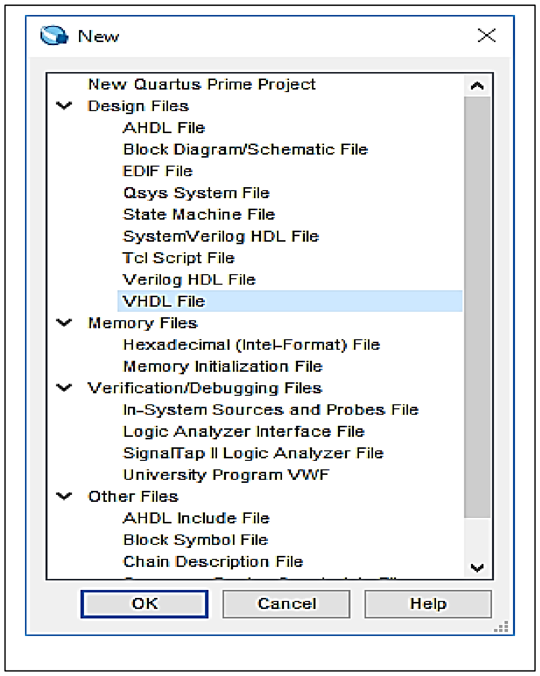


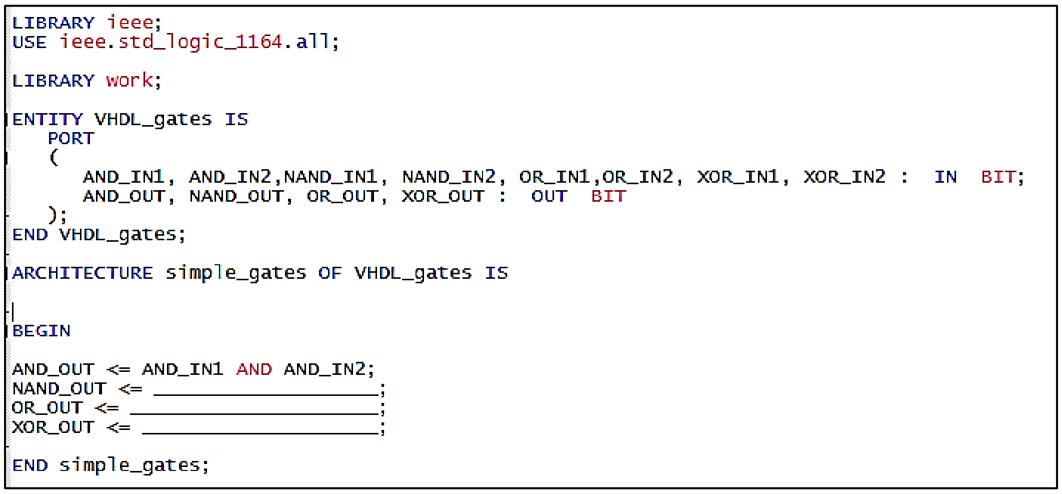
Figure 37 Lab1: Starting a VHDL Design Entry File

SELECT the VHDL File option. A blank VHDL window will then open in Quartus.

Save this VHDL file as “VHDL_gates.vhd” by going to the FILE Tab and SELECTING the File> Save As option.

This VHDL design file is to be an exact functional replica of the schem_gates circuit that was done earlier (for easier comparison during the demo) but entered with VHDL coding.

You must enter the all of the VHDL code shown below in Figure 38 and then fill in the VHDL coding in the ARCHITECTURE section for the remaining gate function types (same as “schematic_gates”).



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY VHDL_gates IS
   PORT
   (
      AND_IN1, AND_IN2,NAND_IN1, NAND_IN2, OR_IN1,OR_IN2, XOR_IN1, XOR_IN2 :  IN  BIT;
      AND_OUT, NAND_OUT, OR_OUT, XOR_OUT :  OUT  BIT
   );
END VHDL_gates;

ARCHITECTURE simple_gates OF VHDL_gates IS


BEGIN

AND_OUT <= AND_IN1 AND AND_IN2;
NAND_OUT <= ____________________;
OR_OUT <= _____________________;
XOR_OUT <= ____________________;

END simple_gates;
```

Figure 38 Lab1: Initial VHDL_gates File

Save the VHDL file by browsing to your **Lab1 project folder** and Save the VHDL_gates.vhd file and just leave the file active (current). Create a schematic block symbol for the VHDL_gates design.file. Do this as before (using File>Create / Update>Create Symbol files for Current File). Save the VHDL_gates.bsf symbol.

Go back to the top level schematic design to insert ("instantiate") the new VHDL_gates symbol as in Figure 39. Recall that to select the symbol RIGHT-CLICK anywhere on the top level schematic again and SELECT the Insert>Symbol option. Browse to the Project folder in the Symbol Window and Select the VHDL_gates symbol and Click OK.

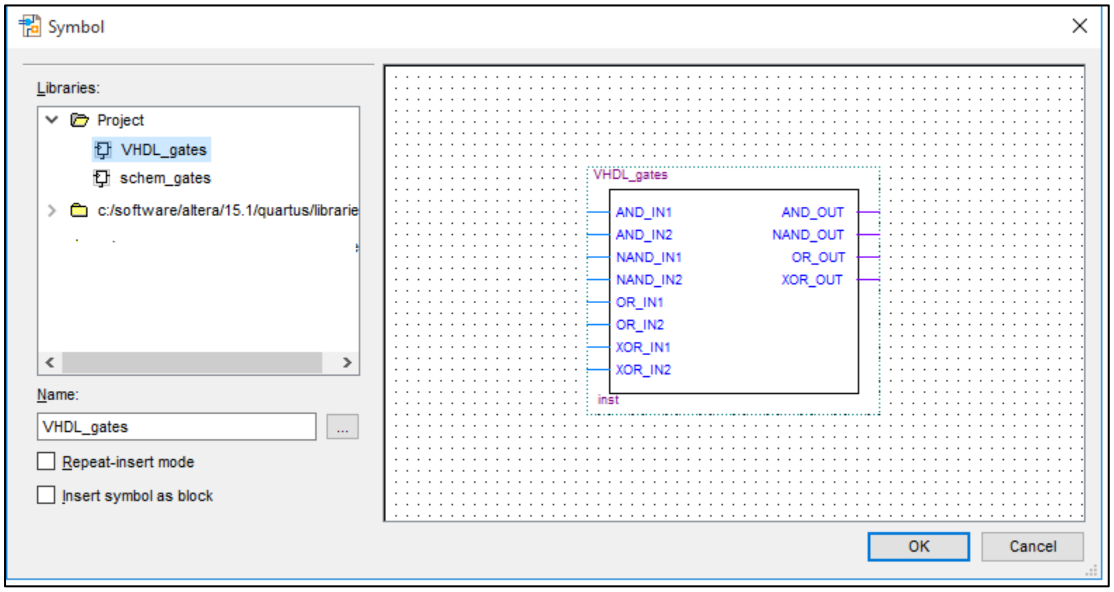


Figure 39 Lab1: Selecting the VHDL_gates Symbol for Insertion

Connect the VHDL block input to the same input connections and connect the VHDL block outputs to the other remaining LogicalStep board LEDs as shown in Figure 40 by using the ORTHOGONAL NODE TOOL as before.

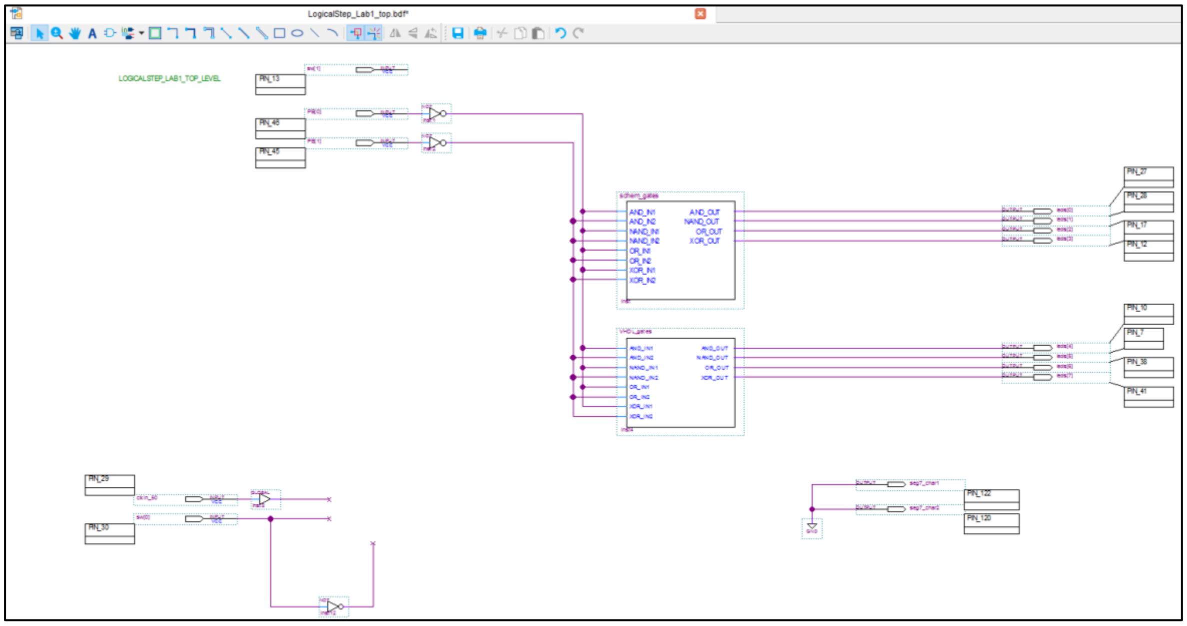


Figure 40 Lab1: Adding Connections to VHDL_gates

Save the design and run a FULL compile of the new FPGA design (Processing>Start Compilation). Download the new FPGA design into the FPGA with the programmer and then test with the PB[1..0] keys. Confirm that both the schematic and VHDL implementations work the same by observing the patterns on the two sets of LED outputs.

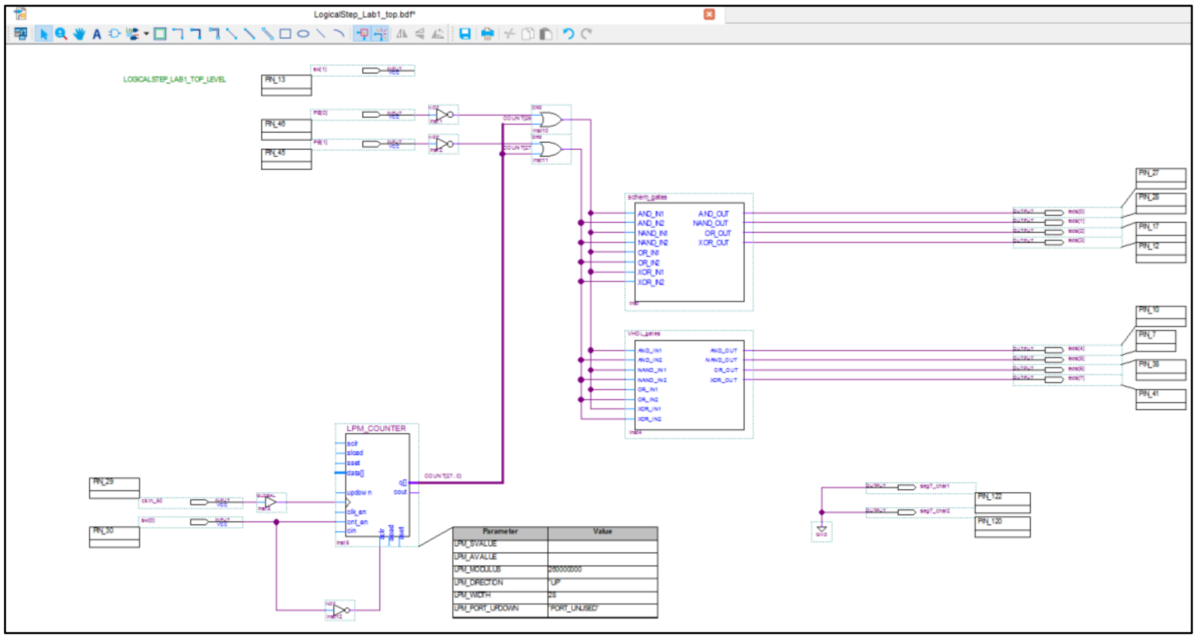
3.3.6 Adding Some Automation

As a next step to the LogicalStep_Lab1_top design add an LPM_counter from the library in (altera/quartus/megafunctions/arithmetic/lpm_counter). The counter will use the clkin_50 input pin signal to increment. The clock on the LogicalStep board runs at 50 MHz. The counter is being added to automate the operation of the PB keys and to slow down the logic activity of the schem_gates and VHDL gates blocks so that you can actually see them switching. The parameters for the counter can be observed in the diagram below in Figure 41.

Double-Click the LPM_Parameter block and modify its properties.

LPM_MODULUS: 260000000
LPM_DIRECTION: "UP" (include quotes)
LPM_WIDTH: 28
LPM_PORT_UPDOWN: "PORT_UNUSED" (include quotes)

Connect the LPM_Counter clock input (pin with a "I>" on the left side of the symbol) to the GLOBAL buffer that is used by the CLKIN_50 input pin using the Orthogonal Node Tool as before.

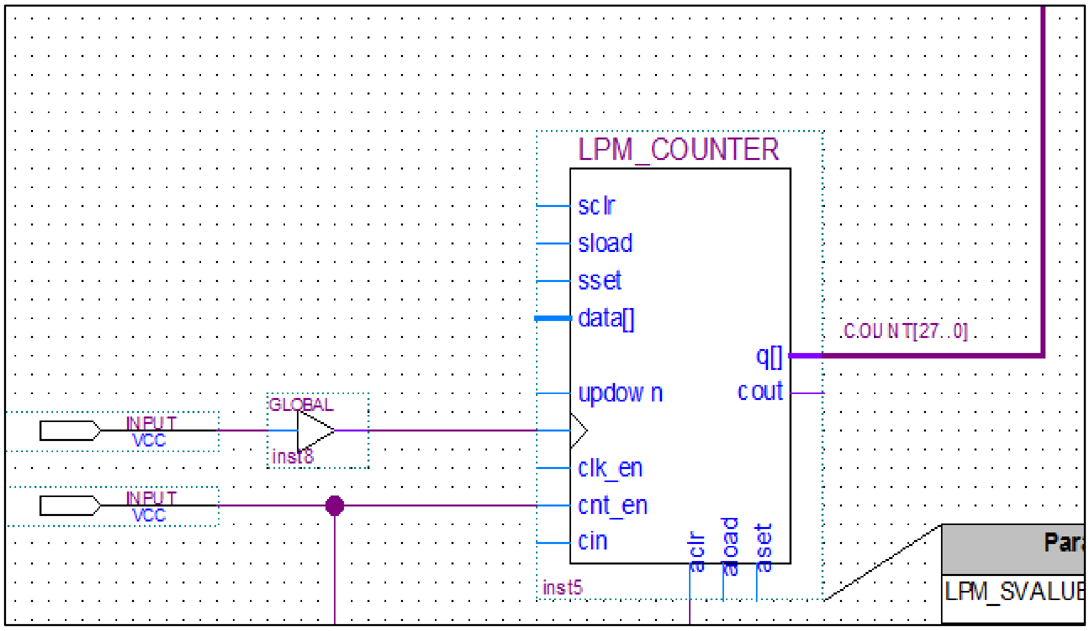


Parameter	Value
LPM_SVALUE	
LPM_AVALUE	
LPM_MODULUS	260000000
LPM_DIRECTION	"UP"
LPM_WIDTH	28
LPM_PORT_UPDOWN	"PORT_UNUSED"

Figure 41 Lab1: Adding Automation to LogicalStep_Lab1_top Design

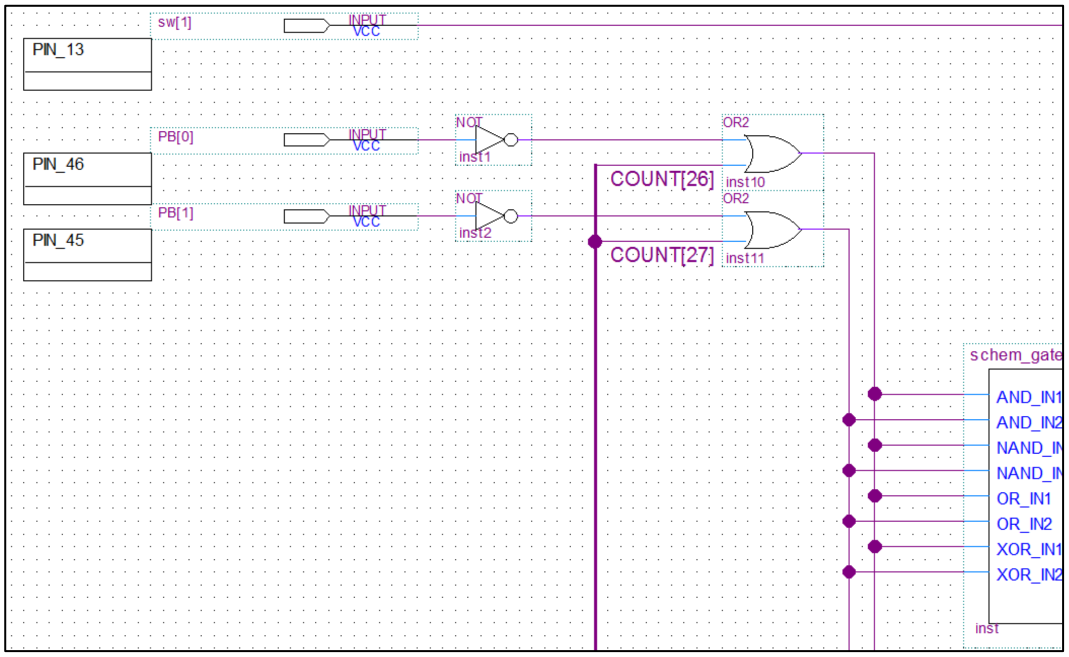
Next connect the counter "cnt_en" inputs to the sw[0] input pin. Also make sure to connect the sw[0] INVERTER to the counter "aclr" pin. These two connections will turn the counter on and off.

Disconnect the two PB inverter outputs from the schem_gates and VHDL_gates block inputs. Insert and connect a single, two-input OR gate to each of those inverter outputs. Then connect the OR gate outputs back to the wires connected to the schem_gates and VHDL_gates block inputs.



Using the Orthogonal **BUS** tool (icon is located beside the Orthogonal NODE Tool) connect a bus (a thick wire) to the LPM_Counter "q[]" output (See Figure 42). Draw it to up close to the OR gates. Select this bus and Right-click to change its properties. Label this bus as COUNT[27..0].

Figure 42 Lab 1: Creating a 28 Bit Signal Bus for the Counter Output



Using the Orthogonal **NODE** tool draw two thin wires from the new bus (COUNT[27..0]) to the open OR gate inputs as shown in Figure 43. Select each of these thin wires and change their properties to label them as COUNT[26] and COUNT[27] as shown.

Why do you think we connect to the two highest COUNT bits from the counter??

Figure 43 Lab1: Using Bits from the Counter Signal Bus

We will NOT be discussing the internal functionality of the counter during this lab. We will only be using it as a “generic engine” to automate the PB inputs to the design.

Other schematic-based functions can be viewed in the library for your future reference.

So the way this automation should work is:

1) If sw[0] is OFF then the counter does not count and the PB inputs can be used as before.
2) If sw[0] is ON then the counter is enabled and the PB inputs are not required to be manually operated.

Recompile the design (Processing>Start Compilation) and then download the load file (using Tools>Programmer) into the FPGA and then confirm the automatic operation of the blocks on the LogicalStep LEDs.

Again, the VHDL design driven LED’s should match the operation of the schematic design driven LEDs.

3.4 POST - Lab1 Activities

For your **DEMO** design in the next Lab session you must add another two blocks (one in schematic form and one VHDL design form) that allows **sw**[1] to be used as an output polarity change control on each of the output pins. Further, a new block function must be entered to use a **single 2 pin gate per path**. (One pin will be connected to the Polarity Control and the second pin will be connected to the upstream schem_gates or VHDL_gates output).

Hint: Do the schematic one first and create the gate truth tables for various 2 input gates in the library to determine what kind of 2 input gate to use. The connections are shown below in Figure 44 but with the gates missing.

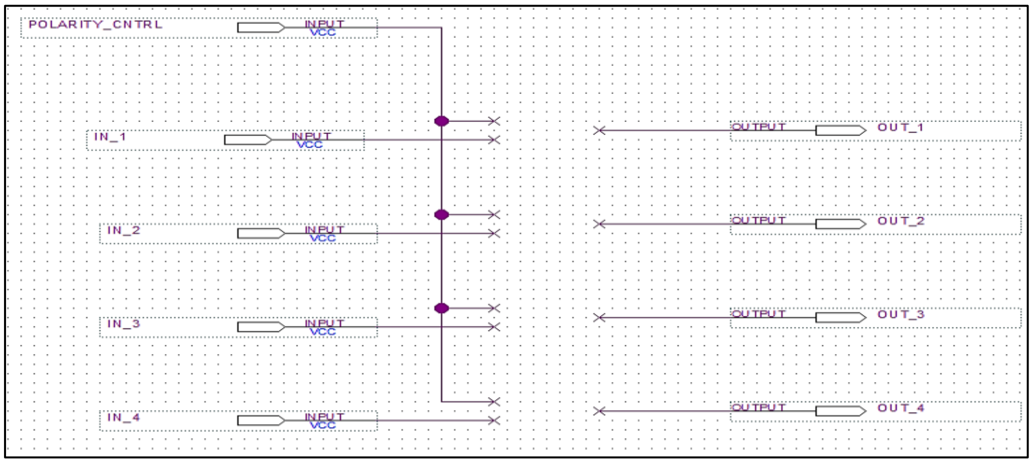
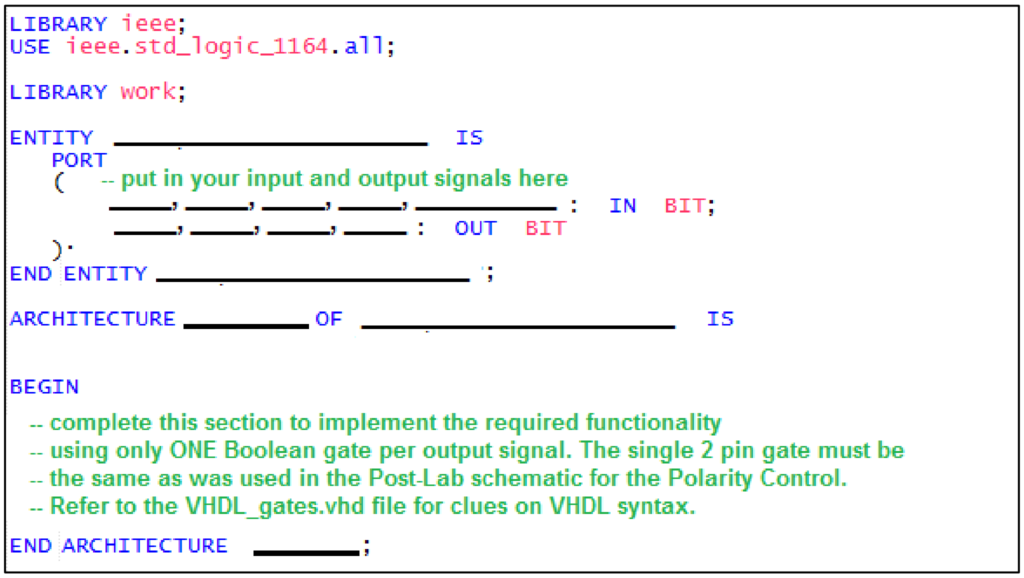


Figure 44 Lab1: Initial Schematic Version of Polarity Control

For the VHDL version you may use the following info below in Figure 45:

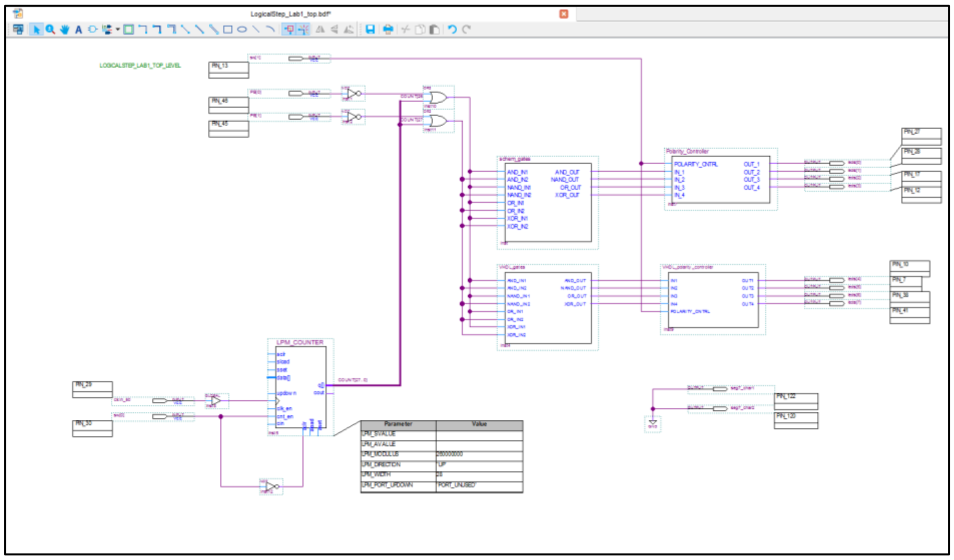


```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY ______________________ IS
   PORT
   (   -- put in your input and output signals here
       ____, _____, _____, _____, __________ :  IN  BIT;
       ____, _____, _____, _____ :  OUT  BIT
   );
END ENTITY ______________________ ;

ARCHITECTURE _________ OF ______________________  IS


BEGIN
  -- complete this section to implement the required functionality
  -- using only ONE Boolean gate per output signal. The single 2 pin gate must be
  -- the same as was used in the Post-Lab schematic for the Polarity Control.
  -- Refer to the VHDL_gates.vhd file for clues on VHDL syntax.
END ARCHITECTURE ______;
```

Figure 45 Lab1: Initial VHDL File of Polarity Control

For the VHDL version make sure that you save that file with the same name as the declared ENTITY name used.

Create, save and add a schematic block symbol for <u>EACH</u> of your new blocks as before (using File>Create / Update>Create Symbol files for Current File) and add it to the LogicalStep_Lab1_top schematic with the required connections mentioned above.



Parameter	Value
LPM_SVALUE	
LPM_AVALUE	
LPM_MODULUS	250000000
LPM_DIRECTION	"UP"
LPM_WIDTH	28
LPM_PORT_UPDOWN	"PORT_UNUSED"

Your final schematic design will look something like the following in Figure 46.

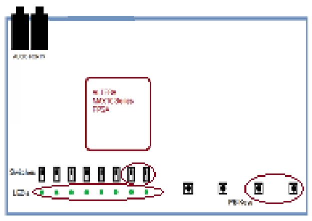


Figure 46 Lab1: Demo Design

Complete the work for the Lab1 Demo design (ie: get it compiled and tested on the LogicalStep board) since it will be required at the next lab session in Lab 2. Make sure everything is saved to project folder ("Lab1") on your N:drive. Select the FILE>Close Project option.

There will be no typical report for this lab but you will need to answer some basic Lab1 related questions listed on the Lab1 Submission form during the time of your Lab1 DEMO.

NEXT LAB SESSION: We will get into more challenging logic designing where we learn and build functions that are used in simple computers today.