CMPE 260
Digital System Design II
Laboratory Exercise 1: Implementation Techniques
Revised: 08-05-2017

This exercise provides an introduction to working with VHDL on FPGAs via several entities in an Arithmetic Logic Unit (ALU) circuit. The objective of this exercise is to be able to program an FPGA and recognize the different styles of writing VHDL. VHDL files are created, or generated, and tested in simulation and on hardware.

Background

Not every component is well suited to every style of VHDL. It is important to be able to recognize whether behavioral, structural, or dataflow would be most appropriate and use that style. In other cases, especially with more complex circuits like USB controllers, using Vendor IP (intellectual property) is the better option.

Prelab Work

- Read the entire lab. Step by step instructions on how to do many of the important operations in Xilinx ISE are provided.
- Write code for NOT, AND, XOR, and Logical Shift Right.
- Draw a schematic of the circuit Logical Shift Left generates when i=4 and N=8. Label all wires with signal names.
- Print the sign off sheet.

Program Specification

The design must meet the following specifications and use all components listed at least once. Table 1 contains the required operations of the ALU.

Table 1: The Operations and Instructions of the ALU

Operation	Instruction
Bit-wise or	OR
Bit-wise not	NOT
Bit-wise and	AND
Bit-wise xor	XOR
Shift Left Logical	SLL
Shift Right Logical	SRL

OR Gate

The OR gate has the following features:

- Function:
  - Bitwise OR Logic
- Parameters:
  - The number of bits of the two inputs. To be called `n`.
- Inputs:
  - `A` (`n` bits): First input
  - `B` (`n` bits): Second input
- Output:
  - `Y` (`n` bits): Result of the OR operation

The VHDL for this gate is given in the Provided Code section. Use this to create the other gates of a similar nature.

AND Gate

The AND gate has the following features:

- Function:
  - Bitwise AND Logic
- Parameters:
  - The number of bits of the two inputs. To be called `n`.
- Inputs:
  - `A` (`n` bits): First input
  - `B` (`n` bits): Second input
- Output:
  - `Y` (`n` bits): Result of the AND operation

The VHDL for this gate should be written based off of the given code for the bitwise OR gate.

XOR Gate

The XOR gate has the following features:

- Function:
  - Bitwise XOR Logic
- Parameters:
  - The number of bits of the two inputs. To be called `n`.
- Inputs:
  - `A` (`n` bits): First input
  - `B` (`n` bits): Second input
- Output:
  - `Y` (`n` bits): Result of the XOR operation

The VHDL for this gate should be written based off of the given code for the bitwise OR gate.

Inverter

The inverter has the following features:

- Function:
  - Bitwise NOT Logic
- Parameters:
  - The number of bits of the input. To be called `n`.
- Inputs:
  - `A` (`n` bits): First input
- Output:
  - `Y` (`n` bits): Result of the NOT operation

The VHDL for this gate should be written based off of the given code for the bitwise OR gate.

Logical Left Shifter

The LLS has the following functions:

- Functions:
  - Shift bits to the left, appending zeros.
- Parameters:
  - The length of the value to be shifted, in bits. To be called `n`.
- Inputs:
  - `A` (`n` bits): Value to be shifted
  - `B` (`n` bits): Number of bits to shift
- Outputs:
  - `Y` (`n` bits): Result of the logical left shift.

The VHDL for this shifter has been given. It should be used to create the Logical Right Shifter.

Logical Right Shifter

The LRS has the following features:

- Functions:
  - Shift bits to the right, appending zeros.
- Parameters:
  - The length of the value to be shifted, in bits. To be called `n`.
- Inputs:
  - `A` (`n` bits): Value to be shifted.
  - `B` (`n` bits): Number of bits to shift.
- Outputs:
  - `Y` (`n` bits): Result of the logical right shift.

The VHDL for this shifter should be based of of the given VHDL for the Logical Left Shifter.

Wrapper

The top level wrapper is responsible for combining the components above.

- Inputs:
  - A, B (n bits): The two numbers on which to operate.
- Outputs:
  - OR_Out (n bits): The result of the OR operation.
  - AND_Out (n bits): The result of the AND operation.
  - XOR_Out (n bits): The result of the XOR operation.
  - NOT_Out (n bits): The result of the NOT.
  - SL_Out (n bits): The result of the Logical Left Shift.
  - SR_Out (n bits): The result of the Logical Right Shift.

Provided Code

The following snippets of code are provided to get you started. Anything that is not given you are expected to write on your own; however, you can base what you write off of the given VHDL.
The code given here will be given as .vhd files on the myCourses page.

- Bitwise OR

Listing 1: VHDL for Bitwise OR

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

entity nBitOr is
    generic (n : integer := 16);
        port(A,B : in std_logic_vector(n-1 downto 0);
             Y : out std_logic_vector(n-1 downto 0)
        );
        end nBitOr;

architecture Dataflow of nBitOr is
begin
    Y <= A or B; -- bitwise or
end Dataflow;
```

- Logical Left Shift

Listing 2: VHDL for Logical Left Shifter

```
library ieee ;
```

```
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity nBitLeftShift is
  generic (n : integer := 8);
  port (A,B : in std_logic_vector(n-1 downto 0);
        Y   : out std_logic_vector(n-1 downto 0));
end nBitLeftShift ;
architecture struct of nBitLeftShift is
  -- create array of vectors to hold each of n shifters
  type shifty_array is array (n-1 downto 0) of
     ↪ std_logic_vector (n-1 downto 0);
  signal Y_array : shifty_array;
begin
generate_shifters : for i in 0 to n-1 generate
  Y_array (i)(n-1 downto i) <= A(n-1-i downto 0);
  left_fill : if i > 0 generate
    Y_array(i)(i-1 downto 0) <= ( others => '0');
  end generate left_fill;
end generate generate_shifters;
-- value of B (in binary) determines number of bits A is
   ↪ shifted.
-- Warning: B (in decimal) must not exceed n-1. If n=16
   ↪ bits, the value
--    of B must be 0 to 15 (shifting 16 does not make sense)
   ↪ . If n=17, then
--    the value of B can be 0 to 16.
Y <= Y_array(to_integer(unsigned(B)));
end struct;
```

- Wrapper

Listing 3: VHDL for ALU Top

```
library IEEE ;
use IEEE.STD_LOGIC_1164.all;
--Entity declaration
entity lab1top is
  generic (N : integer := 13);
  port (A,B : in std_logic_vector (N -1 downto 0);
        or_output : out std_logic_vector (N -1 downto 0);
        sl_output : out std_logic_vector (N -1 downto 0));
end lab1top ;
architecture Structural of lab1top is
begin
  nBitOr0 : entity work.nBitOr
    generic map (N => N)
    port map (A => A, B => B, Y => or_output);
  nBitLeftShift_0 : entity work.nBitLeftShift
    generic map (N => N)
    port map (A => A, B => B, Y => sl_output);
end Structural ;
```

Testbench

Use Listing 4 to validate the ALU. Note that this testbench only tests the OR gate and left shifter. You will need to add all of your other entities, each with their own output. See step 2b for instructions on how to use ISE to generate much of the test bench.

Listing 4: Test bench

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY lab1tb IS
END lab1tb ;
ARCHITECTURE behavior OF lab1tb IS
  constant N : integer := 13;
  signal A,B : std_logic_vector (N-1 downto 0) := ( others
    ↪ => '0');
  signal or_output : std_logic_vector (N -1 downto 0);
  signal sl_output : std_logic_vector (N -1 downto 0);
BEGIN
uut : entity work.lab1top
  generic map (N => N)
  port map (A => A, B => B,
            or_output => or_output,
            sl_output => sl_output);

stim_proc : process
begin
  -- the limits for i and j are good for testing the or and
    ↪  shift_left
  -- values can be modified as necessary to test other
    ↪ functions
  for i in 8 to 13 loop
    for j in 2 to 8 loop
      A <= std_logic_vector(to_unsigned(i,N));
      B <= std_logic_vector(to_unsigned(j,N));
      wait for 100 ns;
    end loop;
  end loop;
  wait ;
end process ;
END ;
```

Lab Procedure

1. Create a project.
   (a) Create a new directory for this exercise.
   (b) Open the Xilinx ISE by navigating to Start >Xilinx Design Tools >ISE Design Suite >ISE Design Tools >64-bit Project Navigator.
   (c) Create a new project, using the setting from Table 2.

Table 2: Xilinx ISE Project Settings

Evaluation Development Board	Nexys 3 Board
Family	Spartan 6
Device	XC6SLX16
Package	CSG324
Speed	-3
Top-Level Source Type	HDL
Synthesis Tool	XST
Simulator	Modelsim - SE VHDL
Preferred Language	VHDL

2. Add VHDL code.
   (a) Create a new VHDL module for each in Table 1, except Listing 4. Use Figure 1 as a guide. Give the file names the entity names; meaning, create one entity/architecture for each of the six functions. Don't forget to define the entity appropriately. Code for two of the functions are given in the Provided Code section.

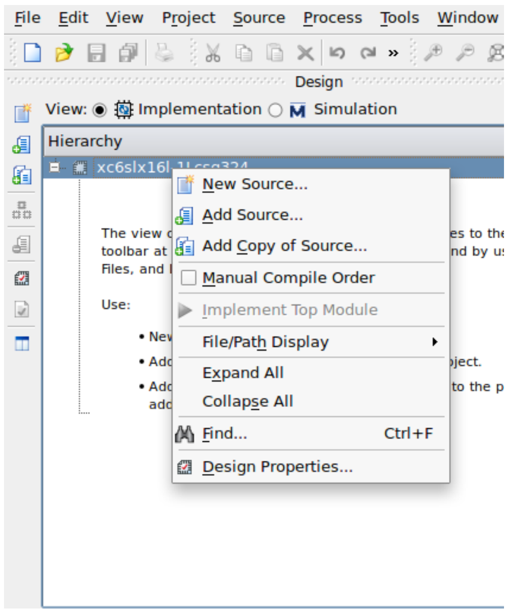


Figure 1: VHDL Module Creation

(b) Check the syntax for each file, as shown in Figure 2.

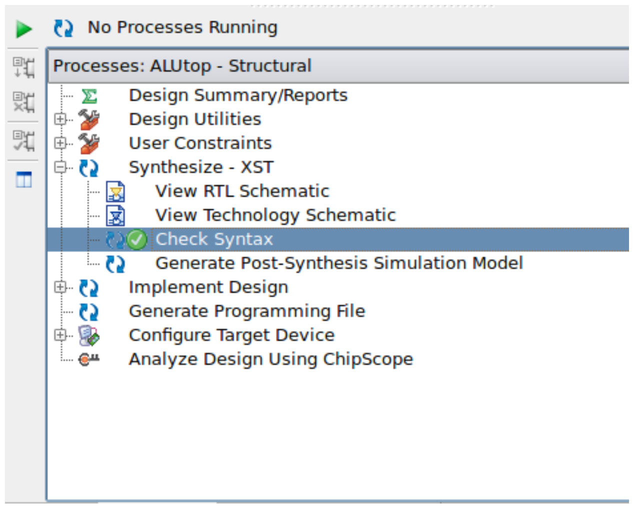
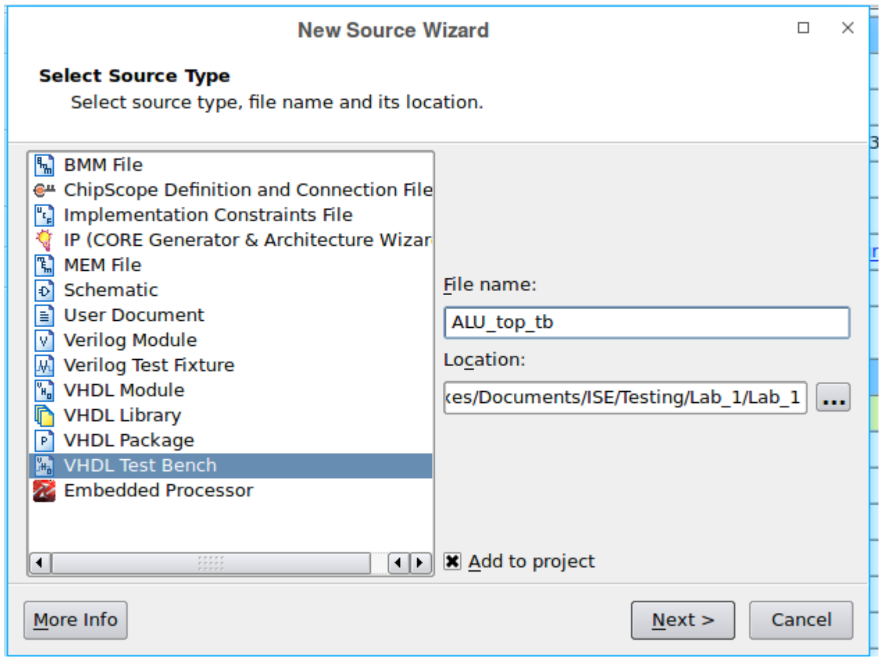


Figure 2: Syntax Checking

(c) Generate a test bench using Figure 3 from the same menu as the other VHDL modules in Figure 1.



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 Wizar
- MEM File
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

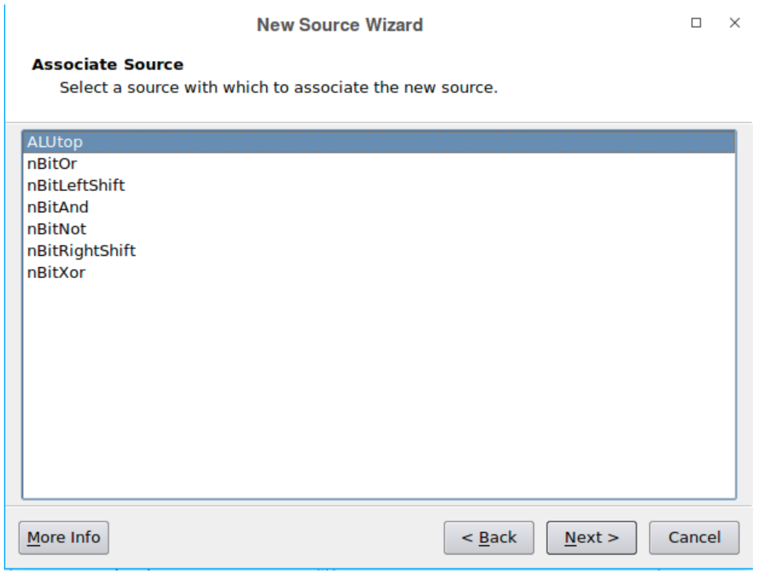
File name: ALU_top_tb

Location: ‹es/Documents/ISE/Testing/Lab_1/Lab_1

[✓ Add to project]

Figure 3: Test Bench Generation

(d) Instruct ISE to use the wrapper to generate the test bench. See Figure 4.



Associate Source

Select a source with which to associate the new source.

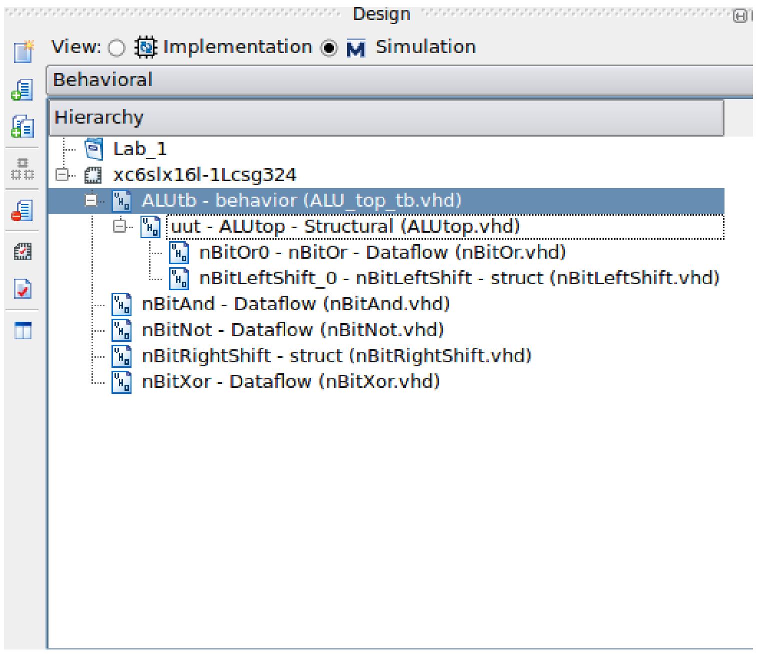
- ALUtop
- nBitOr
- nBitLeftShift
- nBitAnd
- nBitNot
- nBitRightShift
- nBitXor

Figure 4: Test Bench Association

(e) Remove the processes; they will not be needed. Complete the test bench using Listing 4.

3. Simulate.

(a) There must be a completed test bench for simulation to work properly. Select "Simulation", as seen in Figure 5.



Lab_1
xc6slx16l-1Lcsg324
ALUtb - behavior (ALU_top_tb.vhd)
uut - ALUtop - Structural (ALUtop.vhd)
nBitOr0 - nBitOr - Dataflow (nBitOr.vhd)
nBitLeftShift_0 - nBitLeftShift - struct (nBitLeftShift.vhd)
nBitAnd - Dataflow (nBitAnd.vhd)
nBitNot - Dataflow (nBitNot.vhd)
nBitRightShift - struct (nBitRightShift.vhd)
nBitXor - Dataflow (nBitXor.vhd)

Figure 5: Simulation Selection

(b) Right click on “Simulate Behavioral Model” below the Hierarchy box and select “Process Properties”, as shown in Figure 6.

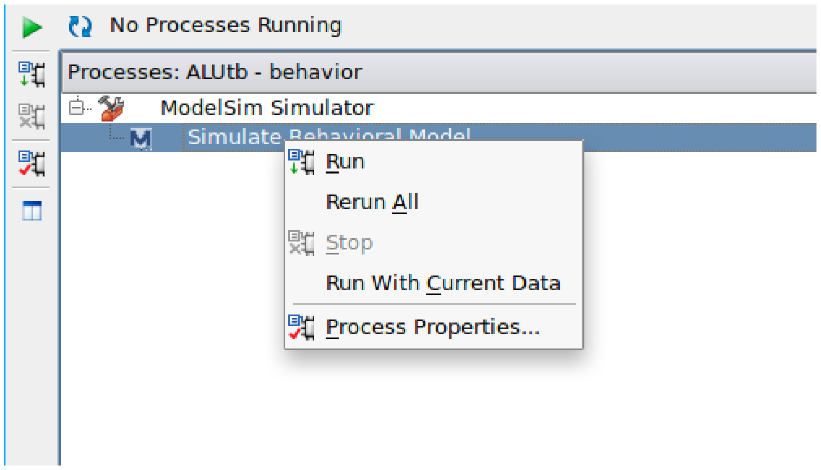
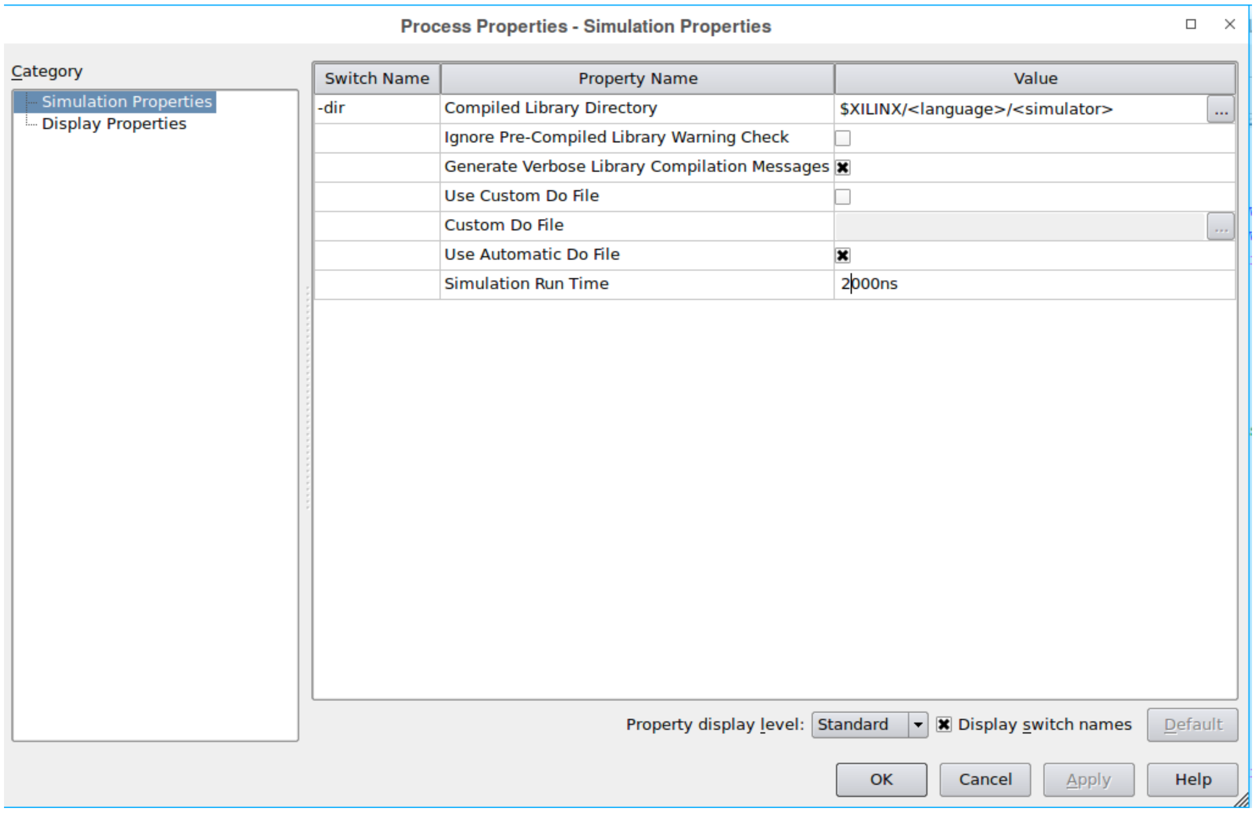


Figure 6: Simulation Properties

(c) Enter 1600 ns in the “Time” box.


Process Properties - Simulation Properties

Category
- Simulation Properties
- Display Properties

Switch Name	Property Name	Value
-dir	Compiled Library Directory	$XILINX/<language>/<simulator>
	Ignore Pre-Compiled Library Warning Check	☐
	Generate Verbose Library Compilation Messages	☒
	Use Custom Do File	☐
	Custom Do File	
	Use Automatic Do File	☒
	Simulation Run Time	2000ns

Property display level: Standard ☒ Display switch names

Figure 7: Simulation Run Time

(d) Click OK to save these changes.

(e) Right click on the "Simulate Behavioral Model" option below the Hierarchy box

(f) Click the "restart" icon to the left of the time box to restart the simulation.

(g) Click the "run" icon to the right of the time box to run the simulation.

(h) Capture the waveform of correct comparator operation.

(i) Close the Simulation

4. Navigate to the lowest level of OR and Shift_Left and capture each one's design. This will cause the design to synthesize, which involves the software determining what gates are required.

   (a) Select the top level design file.

   (b) Select implementation at the top of the design hierarchy box.

   (c) Double click on "View RTL Schematic" under "Synthesize - XST" in the processes window.

   (d) Select the "Start with a schematic of the top level block" option.

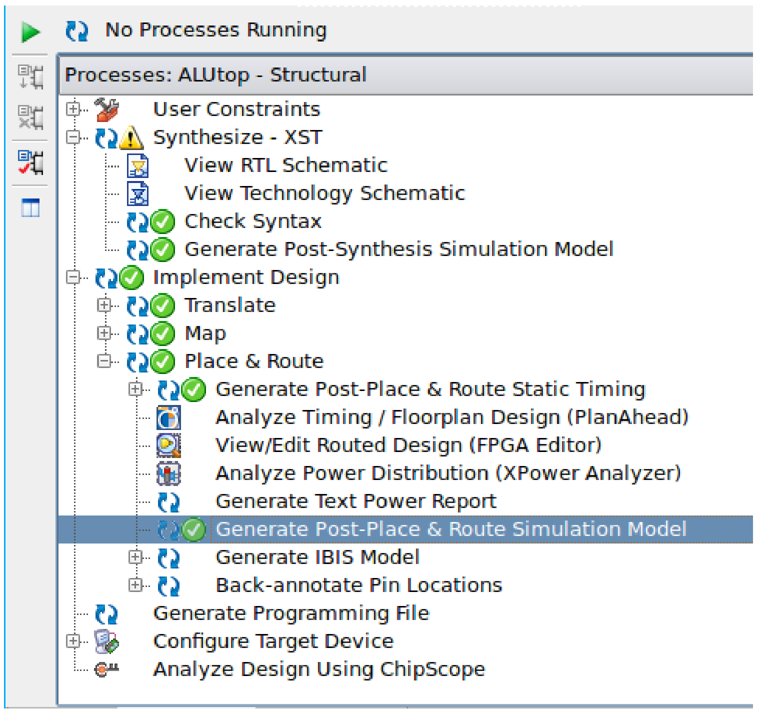
   (e) Double click on each block until the lowest level is reached.

   (f) Capture the lowest level of OR and Shift_Left entities.

5. Perform a post-route simulation. This will account for the delays in the hardware.

   (a) Add the "simprim" library to modelsim. Follow the unisim library instructions in the Troubleshooting section, replacing unisim with simprim.

(b) Double click the "Generate Post-Place & Route Static Timing" under the "Place & Route" option in the Processes window under the Implement Design option. See Figure 8.



No Processes Running

Processes: ALUtop - Structural

```
User Constraints
Synthesize - XST
    View RTL Schematic
    View Technology Schematic
    Check Syntax
    Generate Post-Synthesis Simulation Model
Implement Design
    Translate
    Map
    Place & Route
        Generate Post-Place & Route Static Timing
        Analyze Timing / Floorplan Design (PlanAhead)
        View/Edit Routed Design (FPGA Editor)
        Analyze Power Distribution (XPower Analyzer)
        Generate Text Power Report
        Generate Post-Place & Route Simulation Model
        Generate IBIS Model
        Back-annotate Pin Locations
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
```

Figure 8: Place and Route

(c) Change the view from Implementation to Simulation and select "Post-Route" in the drop-down box below.

(d) Double click on the "Simulate Post-Place & Route model" under the Model-sim option in the Processes window. See Figure 9.

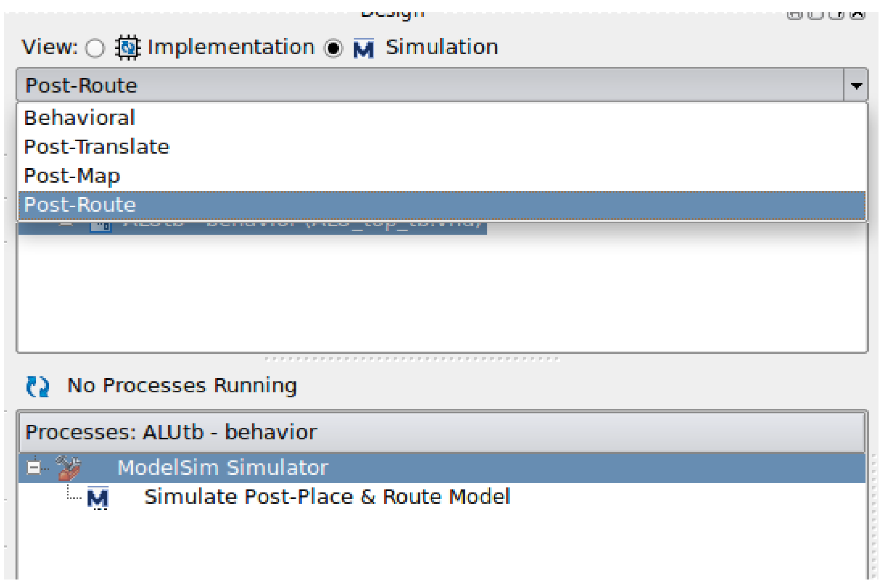

View: Implementation ● Simulation
Post-Route
Behavioral
Post-Translate
Post-Map
Post-Route
No Processes Running
Processes: ALUtb - behavior
ModelSim Simulator
Simulate Post-Place & Route Model

Figure 9: Performing a Post-Route Simulation

(e) Restart and run the post-route simulation, just like the behavorial one.

(f) Capture the waveform.

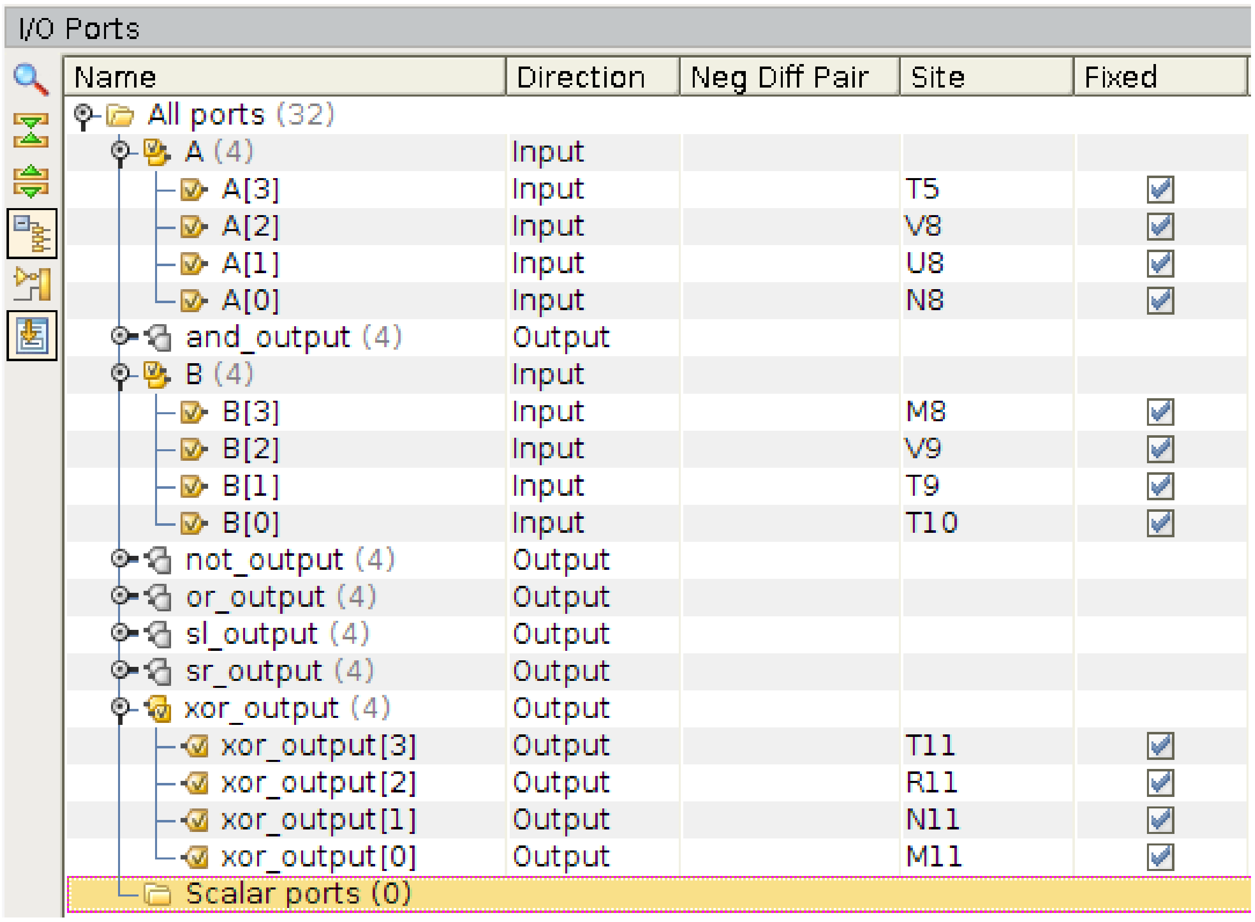
6. Prepare the board for programming by selecting the I/O pins.

   (a) Using the generics, resize the design to fit on the board. Both inputs and the outputs should all be set to 4 bits wide for this.

   (b) In the Implementation view, select the top-level wrapper.

   (c) Double click on “Floorplan Area/IO/Logic (Plan Ahead)” under the User Constraints option.

   (d) Using the schematic of the FPGA board and Figure 10, select the appropriate site cells for each input and output. Then save and close the Plan Ahead file. The Nexys_3_IO PDF on myCourses may also be of assistance.



I/O Ports

Name	Direction	Neg Diff Pair	Site	Fixed
All ports (32)				
A (4)	Input			
A[3]	Input		T5	✓
A[2]	Input		V8	✓
A[1]	Input		U8	✓
A[0]	Input		N8	✓
and_output (4)	Output			
B (4)	Input			
B[3]	Input		M8	✓
B[2]	Input		V9	✓
B[1]	Input		T9	✓
B[0]	Input		T10	✓
not_output (4)	Output			
or_output (4)	Output			
sl_output (4)	Output			
sr_output (4)	Output			
xor_output (4)	Output			
xor_output[3]	Output		T11	✓
xor_output[2]	Output		R11	✓
xor_output[1]	Output		N11	✓
xor_output[0]	Output		M11	✓
Scalar ports (0)				

Figure 10: I/O Pin Selection

(e) This will need to be done twice, for showing two outputs on the board. Complete this and all of Step 7 once for the XOR operation, then reassign the output LEDs for the Logical Left Shift operation and repeat Step 7 again. Have your TA or Lab Instructor check both of these.

(f) After assigning the pins, double click on the "Generate Programming File" option in the Processes window.

7. Program the board

(a) Make sure SW8 is UP, selecting VUSB (powered by USB).

(b) Connect the USB cable to the computer and the board.

(c) Double click the "Manage Configuration Project (iMPACT)" option in the Processes window.

(d) Once iMPACT has launched, select "Boundary Scan".

(e) Right click on the main screen and choose the "Initialize Chain" option.

(f) In the "Assign New Configuration File" window, select the programming file (.bit extension) from the project folder.

(g) If a prompt appears asking to attach SPI or BPI PROM, select "No".

(h) Right click on the icon representing the FPGA board in the Boundary Scan

window and select the "Program" option.
   (i) iMPACT will program the board using the programming file.
   (j) Once the program is finished uploading to the board, the board will be ready for testing. Be sure to leave it plugged in, or these steps will need to be repeated.
8. Demonstrate a working board, along with the waveforms from the behavioral and post-route simulations, to the instructor, who will sign the grading sheet.
9. Submit the source code and report online to the appropriate myCourses dropbox.

Troubleshooting

If the Unisim library is not found:

1. In ModelSim, go to File >New >Library.
2. Select "Map to an existing library".
3. Browse to C:\MentorGraphics\xilinx_lib\unisim and add

If Modelsim does not open, go to the Xilinx window:

1. Xilinx>Edit>Preferences>ISE Generals>Integrated Tools
2. For the ModelTech simulator choose: C:\Mentor Graphics\win64\modelsim.exe
3. Apply

Lab Report

Write a report that meets the rules of professional technical writing and follows the lab report format on myCourses. Additional items that must be included:

– Marked simulation results, both the correct and incorrect versions. The incorrect simulation is only required for the behavorial. **FIGURES WITH BLACK BACKGROUNDS MUST BE INVERTED TO RECEIVE CREDIT**. This means a white background with dark waves.
– A block diagram of the design, which is not hand drawn and is not taken from the RTL schematic.
– RTL diagrams of each VHDL component you created.