EE 4321 – Digital Systems Design using HDL

Laboratory Manual

Dr. Hassan Salamy

Spring 2014

Texas State University – San Marcos

Ingram School of Engineering

San Marcos, Texas

Laboratory 1: Hands-on Training
(Groups of 2)

This is the first Lab exercise for EE4321: Digital Systems Design using HDL. This lab is not to teach you VHDL or test your knowledge about VHDL. It is a hands-on-training of how to use the Xilinx ISE 13.3 project software to build and test a VHDL code.

Problem Statement

In this introductory example, we will design a BCD to Excess-3 converter system. The term BCD refers to representing the ten decimal digits in binary forms. The Excess-3 system simply adds 3 to each number to make the codes look different (see Table 1). The truth table for our design is in Table 2. We will not venture to discuss the importance of the Excess-3 BCD system because the discussion would not serve the main objective of Lab 1 which is to introduce Xilinx ISE 13.3. If needed, I encourage you to do some research about BCD to Excess-3 converter on your own.

Table 1: BCD to Excess-3

Decimal Numerals	Binary Numerals	Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Table 2: Truth Table

Input (BCD)				Output (Excess-3)			
X3	**X2**	**X1**	**X0**	**Y3**	**Y2**	**Y1**	**Y0**
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

The VHDL Code

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

entity Converter is
    Port ( X3 : in  STD_LOGIC;
           X2 : in  STD_LOGIC;
           X1 : in  STD_LOGIC;
           X0 : in  STD_LOGIC;
           Y3 : out  STD_LOGIC;
           Y2 : out  STD_LOGIC;
           Y1 : out  STD_LOGIC;
           Y0 : out  STD_LOGIC);
end Converter;

architecture Behavioral of Converter is
begin

     Y3 <= X3 OR (X0 AND X2) OR (X1 AND X2);
     Y2 <= (X0 AND (NOT X2)) OR (X1 AND (NOT X2)) OR ((NOT X0)
     AND (NOT X1) AND X2 AND (NOT X3));
     Y1 <= ((NOT X0) AND (NOT X1)) OR (X0 AND X1);
     Y0 <= (NOT X0);

end Behavioral;
```

Step by Step Guideline

Below are the necessary steps for building, synthesizing, simulating and programming a VHDL code on SPARTAN 3E board. I will be using the "Simple Converter" example to illustrate the steps (steps may slightly differ based on the Xilinx ISE version):

Creating the Project

1. Start the Xilinx ISE Project Navigator software. If there is already a project running, close that project and create a New Project. This is done under the File drop-down menu, just like creating a new file in most other applications.

2. In the dialog box that pops up, set the directory in which you would like all the files you will create to be saved, and enter the name of the project you would like to create. All the files created by the ISE tool will be placed in a folder with the project name specified by you, which itself resides in the directory you picked. In the Top-Level Module Type dropdown, ensure that HDL is listed, and hit Next >.

3. In the next dialog box, under Device Family, select Spartan3E. Under Device, select xc3S500E, which specifies the type of Spartan-III FPGA used on the board. Under Package, select FG320, which specifies the package type for the FPGA. Under Speed Grade, select -4, which specifies the speed grade of the FPGA. For the Synthesis Tool selection, XST (VHDL/Verilog) should be selected. The Simulator selection should be set to ISIM (VHDL/Verilog). The preferred language is VHDL. When all of these selections are verified, hit Next >.

4. You will receive an additional box. You need not worry about adding any sources to the project at this stage, so you may skip through the series of dialog boxes and finish creating the project.

Adding a VHDL Code Module to the Project

5. From the previous steps, you should have a project created in the Project Navigator environment. In the Sources in Project window on the upper left side of the screen, you will see a listing for the device you specified under the project name you entered. Right click the device object and select New Source from the pop-up menu.

6. From the selections provided, choose VHDL Module to create a new VHDL code segment. You will be prompted to enter a name for the file, and then you will be given a screen to enter input and output ports. Enter a port name for each input and each output for the Simple Converter, and select the direction (in or out) for each port. Click Next > to be given a summary of your VHDL module.

7. Hit Finish and a window containing VHDL code will be created. Note that a .vhd file has been added to the device in the Sources in Project window of the Project Navigator.

8. Before doing anything else, look at the beginning of the VHDL code provided to you. Within the entity definition, you will see the port names you previously entered, followed by their direction and data type. Make sure this information is correct.

9. Your code will be entered between the "begin" and the "end" statements in the architecture section of the VHDL code file created for you.

10. Save your VHDL file and synthesize it. This can be done by highlighting the .vhd file in the Sources in Project window, and then double-clicking the *Synthesize XST* process in the Processes for Current Source window. The development tools will detect any coding errors during this process.

11. After synthesizing the completed VHDL file, pin numbers must be assigned to each input and output port for programming the Spartan-3E Starter Kit board later. To do this, a user constraints file must be added to the project, and modified with the pin number information. Right-click the device in the Sources in Project window, and select New Source. Select *Implementation Constraints File*, and give it a name. In the next Dialog Box, you must select the VHDL source file that the constraints will apply to. Since there is only one file in the project, make sure it is selected and finish the source creation process.

12. Expand the User Constraints and double-click the Plan ahead post synthesis. Under *I/O ports -> Expand all ports -> Scalar ports*, you will see a list of the port names in your project. Select each port and assign to it a pin location (*Site* column) based upon the table found in Appendix A on the class website. Pin numbers may be entered manually

13. After assigning pin numbers to each port, save the file and close the pin assignment window. Highlight the .vhd file in the Sources in Project window and double-click the Implement Design process in the Processes for Current Source window. This will map your design to the Spartan-3E device you selected.

14. In order to simulate the design you have entered, another new source must be added to the project. Right-click the device and add a New Source. Select VHDL Test Bench and give it a name different from the name of the VHDL file it will be testing. This naming difference is important so as not to confuse the simulator. As with the Test Bench file, the VHDL source to which the simulation file will

apply must be selected in the next dialog box. Select your VHDL file, and finish creating the new file.

15. A VHDL test bench file is now created. Do changes to the Test bench file to serve as a test bench for our VHDL example.

16. Check the *simulation* button and verify that *Behavioral* is selected from the drop-down list in the Sources window. Select the test bench file to simulate the design. Under ISim, double click *Behavioral Check Syntax*. The development tools will detect any coding errors during this process.

17. If successful, double click *Simulate Behavioral Model*. The ISim window will be opened.

18. When you are certain that the functionality of the Simple Converter is correct, you may exit the ISim simulator and proceed to programming the Spartan-3E Starter Kit board.

Programming the Spartan-3E Starter Kit board

19. Select *Implementation* under View.

20. After setting this, double-click the Generate Programming File process and wait for the program to complete.

21. Ensure that the Spartan-3E Starter Kit board is connected to power and to your computer's USB port and run this program.

22. After the program file is generated, double-click *Configure Target Device*. This will bring up iMPACT.

23. Double-click on the iMPACT **Boundary Scan** line in the **iMPACT Flows** dialog box to see detected devices (PROMs, FPGAs, etc). Then right-click and select **Initialize Chain** and walk through the dialogs selecting the .bit file

24. After choosing the configuration file to be used, the program will present you with an icon representing the .bit programming file that was generated from your code. Right-click this icon and select Program. A dialog box will appear with some programming settings; uncheck the verify option if it is checked, and say OK. The board should program, allowing you to test the functionality of your Simple Converter circuit physically.

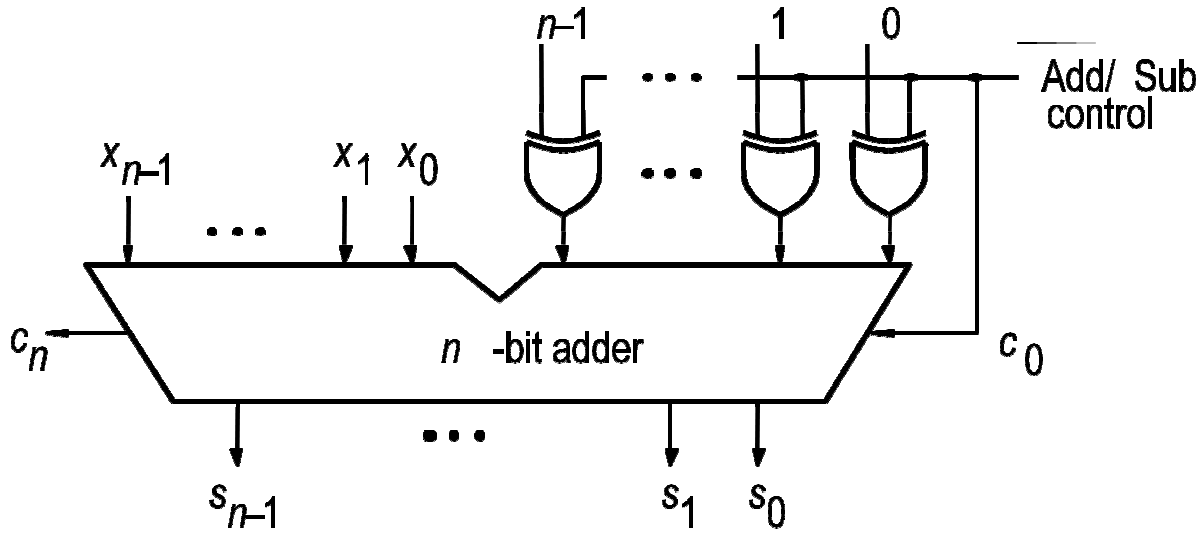
Report (40% of grade)

The report should at least include the following:

1. Introduction
2. Discussion about BCD and excess-3
3. A detailed design of the converter.
4. The testbench file.
5. The waveform simulation results using ISim.
6. Any analysis and thoughts.

Laboratory 2
4-bit Adder/Subtractor
Ripple Carry Adder
(Groups of 2)

In this Lab, you will design a 4-bit adder/subtractor based on Ripple-Carry adder. A general structure of an n-bit adder/subtractor circuit is presented in the figure below.



4-bit Ripple-Carry Adder/Subtractor

For this Lab, you are to design a 4-bit adder/subtractor based on the Ripple-Carry adder as discussed in the class lectures. A quick start guide of how to create this in Xilinx ISE is as follows:

1. Start a new project in the Xilinx ISE Project Navigator.
2. Create a new VHDL module file and design a 1-bit adder/subtractor.
3. Create a testbench for the 1-bit adder/subtractor and simulate the design using ISim. Make sure the 1-bit adder/subtractor works before you proceed.
4. Create a new VHDL module file for the 4-bit adder/subtractor.
5. Use the 1-bit adder/subtractor as a building component for the 4-bit adder/subtractor.

6. Save your new file and look at the Sources in Project window. If you have done everything correctly, you should notice that your one-bit adder VHDL file is now nested below the four-bit adder VHDL file. This means that your new file utilizes the code in the previous file in part of its functionality.

7. Create a testbench for the 4-bit adder/subtractor and make sure the simulation works using ISim.

8. Program the SPARTAN-3E board.

Report (40 %)

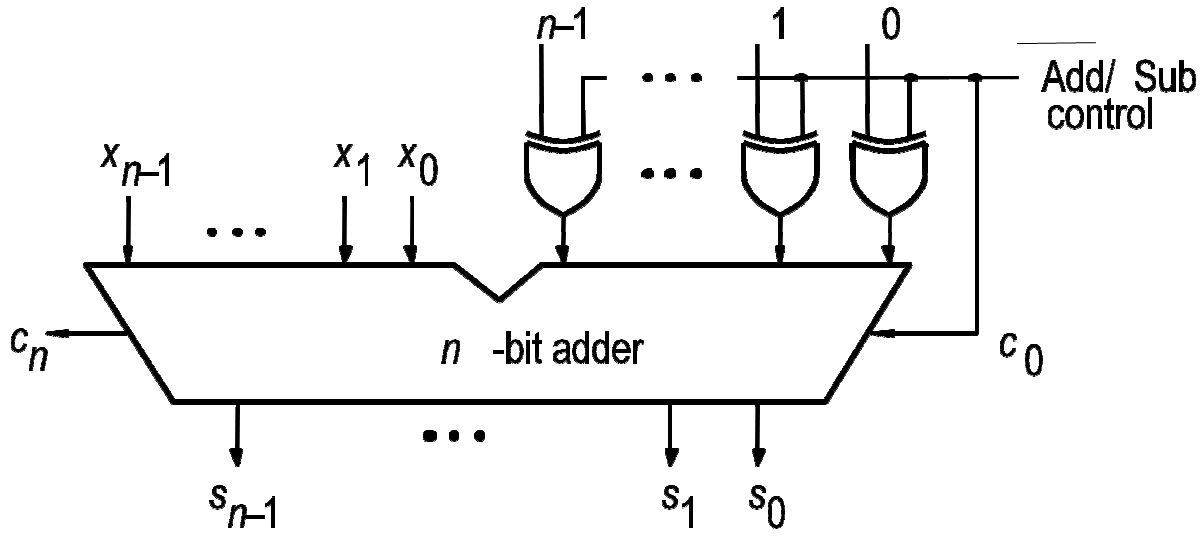
The report should at least include the following:

1. Introduction
2. A detailed design of the 4-bit adder/subtractor using the ripple-carry.
3. The VHDL codes with comments.
4. Analysis of the VHDL code.
5. The testbench files.
6. The two waveform simulation results using ISim for the 1-bit and 4-bit adder/subtractor.
7. Any analysis and thoughts.

Please upload to DropBox the report, VHDL, and testbench files.

Laboratory 3
4-bit Adder/Subtractor
Carry-Look-ahead Adder
(Groups of 2)

In this Lab, you will design a 4-bit adder/subtractor based on Carry Look-ahead adder. A general structure of an n-bit adder/subtractor circuit is represented in the figure below.



4-bit Carry-Look-ahead Adder/Subtractor

The main problem with the Ripple-Carry adder design in Lab 2 is the delays needed to produce the carry out signal and the most significant bits. These delays increase with the increase in the number of bits to be added. To resolve the delay problem, design the 4-bit adder/subtractor using carry-lookahead adder.

1. Design using VHDL a 4-bit carry-look-ahead adder/subtractor in Xilinx ISE.

2. Create a testbench and make sure the design works fine using the ISim simulator.

Report (40 %)

The report should at least include the following:

1. Introduction
2. A detailed design of the 4-bit adder/subtractor using the carry-look-ahead adder.
3. The VHDL code with comments.
4. Analysis of the VHDL code.
5. The testbench file.
6. The waveform simulation results using ISim.
7. Any analysis and thoughts.

Please upload to DropBox the VHDL and testbench files.

Laboratory 4/ Project 1
4-bit Arithmetic Logic Unit
(Individual Lab)

In this lab, you are to design an optimized 4-bit Arithmetic Logic Unit (ALU) in VHDL based on Figure 1. The ALU has 3 inputs **A** and B and **op**, and 2 outputs result **R** and carryout $\mathbf{C_o}$. Inputs **A** and **B** are 4-bit each whereas **op** is 6 bits. The control circuit is such that $\mathbf{C_o}$ is zero when the operation does not generate a carry out; Otherwise, $\mathbf{C_o}$ is the carry out. The 4-bit ALU is to perform the operations in Table 1. You are to program your VHDL design on the SPARTAN 3E board.

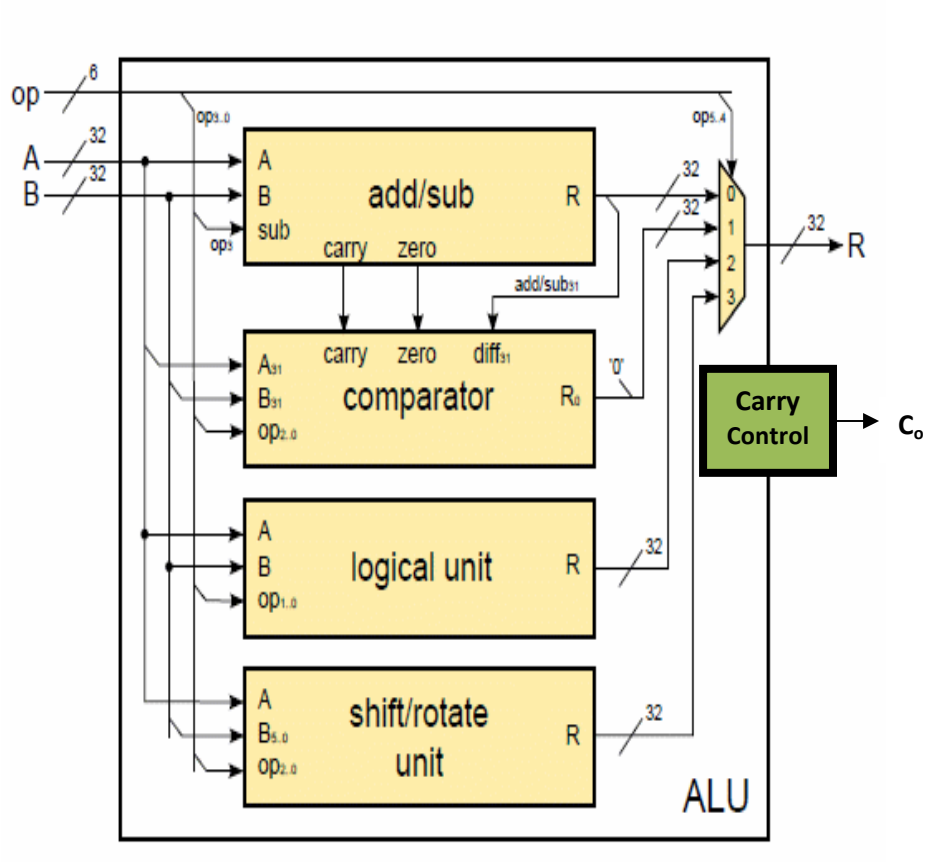


Figure 1: 32-bit ALU

Table 1: ALU operations.

Operation	Operation Type	Opcode
$A + B$	Add/Sub	$000\phi\phi\phi$
$A - B$		$001\phi\phi\phi$
$A \geq B$ (signed)	Comparison	011001
$A < B$ (signed)		011010
$A \neq B$		011011
$A = B$		011100
$A \geq B$ (unsigned)		011101
$A < B$ (unsigned)		011110
A nor B	Logical	$10\phi\phi00$
A and B		$10\phi\phi01$
A or B		$10\phi\phi10$
A xor B		$10\phi\phi11$
A rol B	Shift/Rotate (Optional)	$11\phi000$
A ror B		$11\phi001$
A sll B		$11\phi010$
A srl B		$11\phi011$
A sra B		$11\phi111$

ϕ = *don't care*

Report (35 %)

The report should at least include the following:

1. Introduction
2. A detailed design of the 4-bit ALU.
3. The VHDL code with comments.
4. Analysis of the VHDL code.
5. The test bench file.
6. The waveform simulation results using ISim.
7. Any analysis and thoughts.

Please upload to DropBox the report, VHDL and test bench files.

Laboratory 5
Sequence Detector

(Groups of 2)

Implement in VHDL a MOORE state machine for a circuit that meets the following specifications,

a) The circuit has one input **w** and one output **z**.

b) The output **z** = 1 only if the input **w** had the values 1,0,1,1,0,0,1 during the last seven clock cycles.

c) Count the number of times the sequence in part (b) has been detected so far.

Report (40 %)

The report should at least include the following:

1. Introduction
2. A detailed design of the sequence detector and the associated FSM.
3. The VHDL code with comments.
4. Analysis of the VHDL code.
5. The test bench file.
6. The waveform simulation results using ISim.
7. Any analysis and thoughts.

Please upload to DropBox the report, VHDL and test bench files.