University of California, Davis
Department of Electrical and Computer Engineering

EEC180B DIGITAL SYSTEMS II Spring 2006

LAB 3: ALU Design

Objective: In this lab you will design, simulate and synthesize a 16-bit ALU (Arithmetic Logic Unit) that you will use later in a processor design. You will seek to design a functionally correct ALU that uses minimal FPGA resources. This lab will also introduce the concept of saturating arithmetic. You are **not** required to download this design.

Pre-lab (15 points)

Complete part 1 of the Lab Requirements section before your first lab session. You will turn in the pre-lab to be graded at your first lab session.

ALU Specifications

The external interface of the ALU is shown in Figure 1. The inputs to the ALU consist of two 16-bit two's complement operands, A and B, and some arbitrary number of control signals that are left for you to specify. The output of the ALU consists of a 16-bit two's complement result, Y, and two status flags, Z and N.

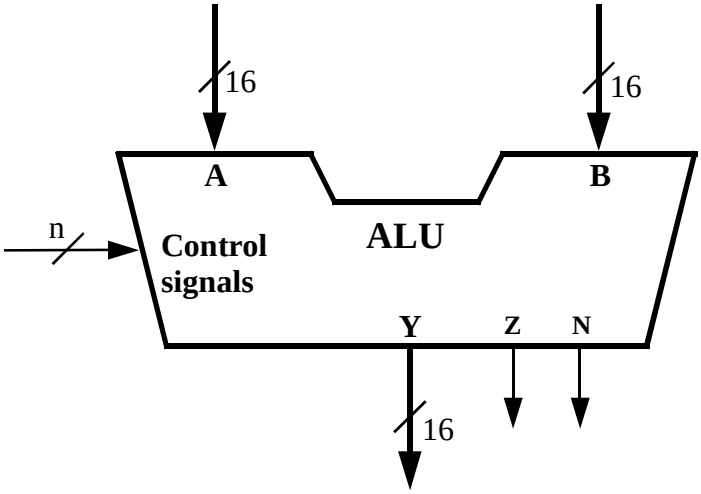

Figure 1 : External Interface of the ALU

The ALU is a combinational circuit - there are no registers or latches within the ALU. This ALU has only seven operations: Add, Add with saturation, Subtract, Subtract with saturation, And, Complement A and Pass B.

In saturating arithmetic, if addition or subtraction results in an overflow or underflow, meaning that the correct answer cannot be represented within the given number of bits, the result is

clamped to the maximum or minimum value. For 16-bit two's complement numbers, the maximum value is 0x7fff and the minimum value is 0x8000. (What does 0x8000 represent in decimal?)[1] As an example, 0x4000 + 0x4000 = 0x7fff using saturated arithmetic since the addition causes an overflow. Although negating the minimum value 0x8000 causes an overflow and gives the erroneous result 0x8000, this number can still be used in additions and subtractions as long as the final result does not produce an overflow. Saturating arithmetic is important for multimedia applications, for example, to prevent a black pixel from inadvertently turning white due to an overflow in a pixel calculation. The MMX technology developed by Intel employs instructions with saturating arithmetic.

The set of ALU operations that you are required to implement is listed in Table 1. You must define the set of control signals you will use to specify each operation. In other words, your VHDL model must have an entity that defines all the input and output signals (including control signals) of the ALU. Show how the control signals that you define will be used to specify each of the ALU operations.

Operation	**Y**	**Control Signals**
Add	A + B	
Add with saturation	A + B (saturating)	
Subtract	A – B	
Subtract with saturation	A – B (saturating)	
And (bitwise)	A AND B	
Com A (one's complement)	NOT A	
Pass B	B	

Table 1. ALU Operations

The ALU status flags, Z and N, should reflect the status of the ALU output. The Zero Flag, Z, should be 1 when the ALU result is zero and 0 otherwise. The Negative Flag, N, should be 1 when the ALU result is negative and 0 otherwise.

Lab Requirements

1. Before you design your ALU, demonstrate your understanding of two's complement arithmetic by completing the following table. Operands A and B are two's complement numbers expressed in hexadecimal, which is how you should express your answers. (Note: This is your pre-lab assignment!)

[1] -2^{15} If you did not get this immediately, reread material on two's complement arithmetic. Otherwise this lab will go badly and you will be fired.

A	B	Add	Add (sat)	Sub	Sub (sat)	And	Com A	Pass B
0000	8000							
FFFF	8000							
8000	8000							
7000	2000							
3333	8000							
8000	CC55							

2. Write a complete VHDL model for the ALU.

3. Write a VHDL test bench to verify your ALU model. At a minimum, your test bench must test your ALU with all of the cases listed in the table in item 1. You may include other test cases as desired. A convenient method for coding your test bench, although not required, is to use two constant arrays for the A and B inputs and then use a for loop to cycle through the test cases. The test bench given on p. 139 in your VHDL text by Roth illustrates this technique.

4. Simulate your test bench using ModelSim. Compare your simulation results with your hand calculations to make sure that your ALU functions correctly.

5. Synthesize your ALU model using Synplicity's Amplify software. Since you will not download this circuit to the Xilinx Spartan-3 board, you do not need to use attribute statements to assign pad locations. Examine the RTL view (.srs file) of your design from within Amplify. If your circuit uses more than one adder and one subtractor, you should recode your VHDL model in order to share adder and subtractor resources among the different arithmetic functions. If desired, you can try to code your VHDL model to specify a single adder/subtractor module. Print the RTL view to turn in with your lab report.

6. Run the Xilinx ISE software on the edif file (.edf) to determine how many FPGA resources your design requires. Check the Design Summary or the Map Report to find out how many 4-input Lookup Tables (LUTs) are needed. Print the Design Summary to turn in with your lab report.

7. Demonstrate your simulation for your TA and show him or her the RTL view from part 5 and the Design Summary from part 6. Have the TA assign a score and sign the verification part of the Lab Cover Sheet. **(50 points)**

Lab Report (35 points)

For your lab report, include the following:

- Lab Cover Sheet with signed TA verification for successful simulation
- Complete VHDL source code, including package code (if any), for your ALU model.
- Complete VHDL test bench source code.

- RTL view of your ALU circuit printed from Amplify.
- Design Summary of your ALU printed from Xilinx ISE.
- Simulation waveform of your ALU simulations printed from Modelsim.
- Answer the following questions:

    1. Design a circuit that uses a single 16-bit adder module to implement a 16-bit adder/subtractor using the principle that two's complement subtraction can be implemented by addition of a negative number. (That is, your accomplish subtraction by negating the subtrahend and adding it with the minuend.) Draw a schematic with logic gates and an adder module. Assume that the adder module has a carry-in input.

    2. Design a simple circuit to detect the overflow condition for binary addition or subtraction. Hint: It can be done with a single gate.