Duke ECE496 – Spring 2013 – Project Part 4: ALU

75 Points. Due electronically by 11:59pm on Thu, Feb 21.

In this part of the project, you will implement the integer Arithmetic Logic Unit (ALU) that you will use in your processor.

Project Part 4: Arithmetic Logic Unit

The ALU implements all of the arithmetic and logical operations specified in the Instruction Set Architecture, including addition, 2's complement subtraction, bitwise logical and, bitwise logical or, left shift, zero-extending right shift as well as left and right rotation. In addition to the above operations, the ALU also generates a signal isNotEqual when both operands are not equal to each other; this is used for the bne (branch-if-not-equal) instruction. Lastly, the ALU generates a signal isLessThan when data_operandB is less than data_operandA and the subtract operation is selected (behavior when the subtract operation is not selected is undefined / don't-care); this is used for the blt (branch-if-less-than signed 2's complement) instruction. Be sure to check for under- / over-flow and generate the correct output when calculating isLessThan. The ALU receives a 5-bit ALU operation code (ALU opcode) from the processor's control logic that denotes which operation should be performed. Note that the for some of the instructions, the ALU opcode is already present in the instruction word. The shift/rotate operations require only the lower 5 bits of input data_operandB.

You will implement your ALU in Structural VHDL using the Quartus II software. You should create one VHDL (alu.vhd) or Block-Diagram (alu.bdf) file that has exactly the same format as the diagram in Figure 1. This top-level alu.vhd or alu.bdf file is likely to refer to other lower-level files (e.g., adder, shifter, etc.). The top-level file alu.vhd or alu.bdf file is what you will then use later in the semester when you need an ALU for your processor. Figure 1 is a screenshot of the alu component in Quartus, and it shows the signal names that you MUST use in your design to facilitate testing and grading.

After implementing your ALU, you should test it thoroughly to verify that it works correctly. One test waveform is provided for your ALU on Sakai. In addition, this assignment will be graded by running additional tests that are not provided, so do not assume that you can ignore bugs that do not manifest themselves on the one test that is provided.

Submitting This Assignment

To submit this assignment, create a Quartus Archive (Project ▸ Archive Project) named project4.qar of all the files needed to implement your design. Make sure that your top-level file is named alu.vhd or alu.bdf. Names of lower-level files are unrestricted, but be sure to include them along with your top-level design entity in the Quartus Archive file. Submit the archive on the assignment section in Sakai.

Table 1: ALU Opcodes

Operation	ALU Opcode
add	00000
sub	00001
and	00010
or	00011
sll	00100
srl	00101
rol	00110
ror	00111

Figure 1: alu VHDL declaration

```
ENTITY alu IS
     PORT ( data_operandA, data_operandB   : IN STD_LOGIC_VECTOR(31 DOWNTO 0); -- 32bit inputs
            ctrl_ALUopcode                 : IN STD_LOGIC_VECTOR(4 DOWNTO 0); -- 5bit ALU opcode
            data_result                    : OUT STD_LOGIC_VECTOR(31 DOWNTO 0); -- 32bit output
            isNotEqual, isLessThan         : OUT STD_LOGIC);
END alu;
```