Lab 2 - Design an Arithmetic Logic Unit

For Lab 2, you are required to perform the following activities:

1. Write VHDL code for the ALU specified below.
2. Synthesize a net list of your design.
3. Create a testbench to fully test your ALU.
4. Submit your ALU design electronically such that your TA can perform thorough testing and grade your design after the lab.
5. Test your design in hardware in lab.

1 Background

The Arithmetic Logic Unit (ALU) is one of the most important parts of a microprocessor. It is here that all the mathematical and logic functions are incorporated. The ALU is unique in that it is built entirely of *combinational logic*, without any latches or registers. A simple block diagram showing the input and output signals of the ALU is sketched below, followed by a listing of the required ALU functions.

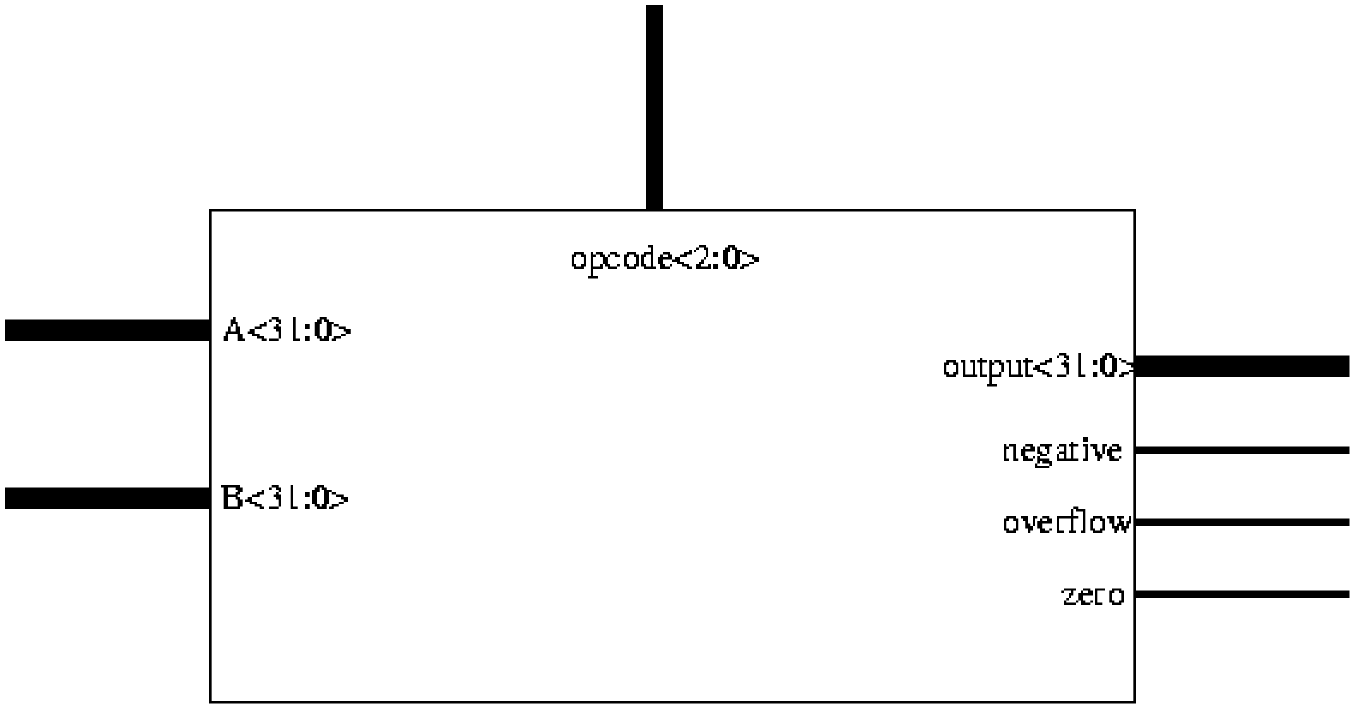
Table 1 ALU Operations

Opcode	Operation	Description
000	SLL	OUTPUT <= A << B
001	SRL	OUTPUT <= A >> B
010	ADD	OUTPUT <= A + B
011	SUB	OUTPUT <= A - B
100	AND	OUTPUT <= A and B
101	NOR	OUTPUT <= A nor B
110	OR	OUTPUT <= A or B
111	XOR	OUTPUT <= A xor B

2 Operation

The ALU Opcode determines the operation the ALU should perform to the two operands namely A and B. The zero flag is active ('1') *if and only if* the output is equal to zero (represented by a string of zeros). The overflow flag is to be active ('1') when an arithmetic operation is in overflow (notice that only add and subtract can cause an overflow). Do a little research to find out what you need to look for to detect an overflow (Hint: carry bits.)

Figure 1 ALU Block Diagram



NOTE

Note that since the ALU is composed wholly of combinational logic, it would be wise to have the ALU do all operations simultaneously, then choose which result you need by using a multiplexer (with the opcode selecting the line to be used).

You need to implement these functions in an ALU designed in VHDL code. It is required that you implement the shift left and shift right functions via a barrel shifter. You might want to look at your ECE 270 book OR ECE 337 notes for a VHDL description on barrel shifer.

A barrel shifter is a combinational logic device that incrementally shifts in stages. In our barrel shifter, we will have 5 stages. Each stage can choose to shift or not to shift. For example, to shift an input by 9, stages one and four will shift 1 time and 8 times, respectively. Stages two, three, and five will not shift. Notice that the binary number 9 is "01001". How convenient that the fourth bit and the first bit are 1 and the second, third, and fifth bits are 0.

NOTE

Don't forget about all possible input combinations. Your ALU has full 32 bit inputs on both A and B ports. This means that you can shift by very large amounts. What happens if you shift by 13,000?

Do NOT do shift around/shift rotation, but do IGNORE the bits of B beyond the 5th bit when do shift operations, which means when given B = 34(100010), you only want shift by 2(00010). you will see why this is the case when you learn decoding instruction or doing your lab4.

Add and Subtract can be done with normal signed adders. Since your processors won't be running at Ghz speeds, a slower adder will not impact your performance and will be sufficient. Make sure you think about all the possibilities that will cause an overflow to happen.

The logic functions NOR, OR, AND, and XOR are just bitwise logic functions. Each bit of A is NORed, ANDed, ORed, or XORed with each corresponding bit of input B.

On the next page is a sample entity declaration for the arithmetic logic unit. Note: unlike the previous lab, there is little VHDL provided to guide you. As always, it is highly recommended that before writing any VHDL code, you sketch out a block diagram for your design in such a way that each block can be easily understood in terms of hardware.

```
entity alu is
  port ( opcode:         IN STD_LOGIC_VECTOR (2 downto 0);
         A, B:           IN STD_LOGIC_VECTOR (31 downto 0);
         aluout:         OUT STD_LOGIC_VECTOR (31 downto 0);
         negative:       OUT STD_LOGIC;
         overflow, zero: OUT STD_LOGIC)
end alu;
```

If you already know how to create block diagrams, you may wish to draw a block diagram in HDL Design and create a separate VHDL entity for each block. In fact, structural VHDL (such as is generated from a block diagram) is to be preferred over using large number of interacting process blocks within the same entity.

NOTE

For this lab exercise you should continue to use the directory structure that was described in Lab 1.

You should create all your VHDL source files inside the source directory.

It is *required*, by the course scripts that you name the entity of the VHDL source code the same name as the file name without the ".vhd" extension. For example, the alu.vhd should have an entity name of "alu".

3 Electronic Submission

3.1 Updating The Makefile

In order for the grading program to interface with your ALU, the port specification and entity must be specified exactly. We have given these strict definitions in the lab. We have, however, not specified the names of other files needed for the ALU design. For example, you may have defined the adder in a file called `adder.vhd` (or something else) and the shifter in `shifter.vhd` (or something else). Both of these entities are needed to compile the ALU, so we need to compile these first. The Makefile is structured as follows:

```
vhdl file : its dependent files
```

NOTE

Remember your entity names are the same as your file names minus the .vhd extension. i.e. entity `alu` and file name `alu.vhd`.

The Makefile compiles the lower level dependencies first, then compiling the higher level files later when the compiler dependencies have been resolved. For our example, the `Makefile` would look like:

```
...
alu.vhd : shifter.vhd adder.vhd
tb_alu.vhd: alu.vhd
...
lab2 : tb_alu.vhd
...
```

To check to see if it works, first remove all of the compiled files with:

```
> make clean
```

To test your "Makefile", type:

```
> make lab2
```

You should see all the dependent files compile (if you have any) without any warnings. Make sure your `Makefile` is in your project1 directory, that is where our grading program will expect it.

NOTE

Incomplete compilation will cause ALL tests to fail when graded with the grading script and you will get *ZERO* credit for the lab. It is your duty to insure that all files compile with NO warnings and that your makefile is correct.

3.2 Submission

Run the following command from your mg account:

```
submit -p lab2
```

The "2" is for lab 2. You can run this command as many times as you want. A tar file containing your source and asmFiles directories along with your Makefile will be placed on the course account. Each submit will overwrite the previous submit, only the latest is kept on the course account.

WARNING

make sure your top level file name is alu.vhd before you do your submission.

4 Setup for Downloading To Prototyping System.

4.1 More Makefile Modifications

There are three files you need to have for successful hardware synthesis. The first file, `aluTest.pins`, contains the mapping of signal names to pins on the fpga. The second file, `aluTest.vhd`, is the file containing the interface that maps the board to your design. The third file `bintohexDecoder.vhd` allows your value to be placed on the 7segment displays available on the board.

Add the following lines to your `Makefile`:

```
# begin VHDL files (keep this)
...
aluTest.vhd: alu.vhd bintohexDecoder.vhd
...
# end VHDL files (keep this)
```

Download `bintohexDecoder.vhd`, `aluTest.vhd` and `aluTest.pins` from the lab website. Remember that `bintohexDecoder.vhd` and `aluTest.vhd` should be in your `source` directory and `aluTest.pins` in `scripts` directory. The file `bintohexDecoder.vhd` is called `7segDecoder.vhd` on the website, in case you are looking for it and can't find it.

4.2 Route & Map for Hardware

From the `~/ece437/project1/` directory, execute the command:

```
> compile aluTest
```

This command will create `aluTest.sof` in the `~/ece437/project1/.aluTest/` directory.

5 Downloading The Design To The Prototyping System

You are now ready to download your design to the fpga and test it.
Download the `aluTest.sof` file to the hardware as in lab 1.

6 Verify the Correct Behavior of the Design in Hardware

The ALU you coded for the lab had several inputs and outputs: OPCODE, A, B, ALUOUT, NEGATIVE, OVERFLOW and ZERO.

OPCODE Push buttons 2, 1, and 0 (KEY2 - KEY0) will be used for the lowest order three bits of the opcode. The push buttons on the board are placed as follows: |3|2|1|0|(they are also labeled KEY#). Pressing the button is a HIGH signal.

A To store a value in A, select the required input value using the 18 switches and then press push button 3 (KEY3). The last 14 bits are set to zero as there are not enough switches for bits.

B It is connected directly to the dip switches. The last 14 bits are just a copy of the last 14 bits from switches (SW17 - SW4). Again out of switches for bits.

ALUOUT The output is connected directly to the 7seg displays. Whatever operation is performed, the result will be shown on the 7seg displays.

OVERFLOW Is connected to the green LED between the 7seg displays (LEDG8).

ZERO Is connected to the left most red LED (LEDR17).

NEGATIVE Is connected to the next red LED (LEDR16).

Please have your TA check the behavior of the design.

7 Deliverables

You will be expected to have the following at the beginning of lab 2:

1. Pre-synthesis simulation results, and post-synthesis simulation results.
2. Source code that is reasonably (not excessively) commented. In addition, all process blocks, and any non-obvious blocks of code must include comments briefly explaining the function of that block of code.
3. A comprehensive test bench coded in VHDL. This should be used as an aid in debugging your ALU design.

The evaluation sheet is due by the end of lab.