# Overall design flow:[[1]](#footnote-1)

**Big Picture:** In this computer exercise, you will design, write, and test a 32-bit Arithmetic Logic Unit (ALU). You will use this ALU in future assignments as you build a fully-functional MIPS microprocessor. You will design and write a testbench of the ALU using VHDL.

# Background

You should already be familiar with the ALU from Section 5.2.4 of the textbook (Figure 1). The design in this lab will demonstrate the ways in which VHDL makes hardware design more efficient. It is possible to design a 32-bit ALU from 1-bit ALUs (i.e., you could program a 1-bit ALU incorporating your half adder from earlier in the course, combine it to create a full adder, chain four of these together to make a 4-bit ALU, and chain 8 of those together to make a 32-bit ALU.) However, it is altogether more efficient, both in time and lines of code, to code it succinctly in VHDL.



|  |  |  |
| --- | --- | --- |
| F2:0 | Function |  |
| 000 | A & B |  |
| 001 | A | B |  |
| 010 | A + B |  |
| 011 | not used |  |
| 100 | A & ~B |  |
| 101 | A | ~B |  |
| 110 | A - B |  |
| 111 | SLT |  |

Figure 1 –These diagrams from your textbook (Figure 5.15, Table 5.1) describe the hardware you will create using behavioral VHDL code. Look carefully at the layout/configuration of the hardware as it will make it easier to implement while meeting the requirements for this exercise.

# VHDL Design Code

You will only be doing ISim simulation in this lab, so you can use your favorite text editor (such as Notepad++) instead of Xilinx ISE if you like.

You must implement the ALU from Figure 1 with behavioral VHDL code. The encoding scheme for arithmetic operations, f, was chosen to make the hardware solution elegant and efficient.

An adder is a relatively expensive piece of hardware. Be sure your design uses *no more than one adder*. Figure 1 should be useful in helping you write your behavioral VHDL code. HDL Example 4.6 in your textbook provides a good example of how to behaviorally describe a multiplexer.

You will need to add the following line to your import statements so that you can perform addition of std\_logic\_vector ports/signals: “**use** ieee**.**std\_logic\_signed**.all;**”.

Name the file alu.vhd. It should have the below entity declaration. (Failure to adhere to either of these two requirements will give you grief on Lab 4). The output zero should be ‘1’ if y is equal to zero.

|  |
| --- |
| **entity** alu **is**  **port(**  a**,** b **:** **in** std\_logic\_vector**(**31 **downto** 0**);**  f **:** **in** std\_logic\_vector**(**2 **downto** 0**);**  y **:** **out** std\_logic\_vector**(**31 **downto** 0**);**  zero **:** **out** std\_logic  **);**  **end** alu**;** |

# VHDL Testbench

Now you can test the 32-bit ALU in ISim. It is prudent to think through a set of input vectors.

We need to use an appropriate set of test vectors to convince a reasonable person that your design is probably correct. Complete Table 1 to verify that all five ALU operations work as they are supposed to. Note that the values are expressed inhexadecimal. Why do you think each of these test vectors is appropriate?

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | Test | F[2:0] | A | B | Y | Zero |
| 1 | ADD 0+0 | 2 | 00000000 | 00000000 | 00000000 | 1 |
| 2 | ADD 0+(-1) | 2 | 00000000 | FFFFFFFF | FFFFFFFF | 0 |
| 3 | ADD 1+(-1) | 2 | 00000001 | FFFFFFFF | 00000000 | 1 |
| 4 | ADD FF+1 | 2 | 000000FF | 00000001 | 00000100 | 0 |
| 5 | SUB 0-0 | 6 | 00000000 | 00000000 | 00000000 | 1 |
| 6 | SUB 0-(-1) | 6 | 00000000 | FFFFFFFF | 00000001 | 0 |
| 7 | SUB 1-1 | 6 | 00000001 | 00000001 | 00000000 | 1 |
| 8 | SUB 100-1 | 6 | 00000100 | 00000001 | 000000FF | 0 |
| 9 | SLT 0,0 | 7 | 00000000 | 00000000 | 00000000 | 1 |
| 10 | SLT 0,1 | 7 | 00000000 | 00000001 | 00000001 | 0 |
| 11 | SLT 0,-1 | 7 | 00000000 | FFFFFFFF | 00000000 | 1 |
| 12 | SLT 1,0 | 7 | 00000001 | 00000000 | 00000000 | 1 |
| 13 | SLT -1,0 | 7 | FFFFFFFF | 00000000 | 00000001 | 0 |
| 14 | AND FFFFFFFF, FFFFFFFF | 0 | FFFFFFFF | FFFFFFFF | FFFFFFFF | 0 |
| 15 | AND FFFFFFFF, 12345678 | 0 | FFFFFFFF | 12345678 | 12345678 | 0 |
| 16 | AND 12345678, 87654321 | 0 | 12345678 | 87654321 | 02244220 | 0 |
| 17 | AND 00000000, FFFFFFFF | 0 | 00000000 | FFFFFFFF | 00000000 | 1 |
| 18 | OR FFFFFFFF, FFFFFFFF | 1 | FFFFFFFF | FFFFFFFF | FFFFFFFF | 0 |
| 19 | OR 12345678, 87654321 | 1 | 12345678 | 87654321 | 97755779 | 0 |
| 20 | OR 00000000, FFFFFFFF | 1 | 00000000 | FFFFFFFF | FFFFFFFF | 0 |
| 21 | OR 00000000, 00000000 | 1 | 00000000 | 00000000 | 00000000 | 1 |

Table - Test vectors to verify the functionality of the ALU.

Build a self-checking testbench – HDL Example 4.38 in your textbook – to test your 32-bit ALU. You want to fully test all functions of your ALU, so carefully consider your desired test vectors. Take a screenshot of the simulation console window and of the test vector waveforms.

You may need more than one screenshot to make the test vector waveforms readable for all tests. Your signals should be displayed using “hexadecimal” radix and in the following order (top to bottom): a, b, f, y, zero. Make the output signals a different color to make help differentiate the signals. Clearly label each test on your simulation (differently colored boxes around each test, associated with test numbers, can very clearly tell a story) for full points. Compare your simulations results to the predictions in your initial test vector table.

# Analysis

Now you need to determine how efficient your design was. Open up the “synthesis report” in the Design tab. Look through this log file until you find a section *like* (not identical, as the results of your synthesis will be slightly different) this:

|  |
| --- |
| =========================================================================  HDL Synthesis Report  Macro Statistics  # Adders/Subtractors : 23  16-bit adder : 23  # Multiplexers : 4  16-bit 64-to-1 multiplexer : 4  ========================================================================= |

Take a screenshot of these results for your lab notebook. What do these results mean?

|  |  |
| --- | --- |
| CE4 Cut Sheet | **Name:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Instructor:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Section:** \_\_\_\_\_\_\_\_\_\_\_ |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Points** | **Out of** | **Initials** | **Late Date** |
| Lab notebook |  | 10 | ----------------- |  |
| ALU schematic |  | 5 | ----------------- |  |
| Test vectors |  | 15 | ----------------- |  |
| Verification/Testing of ALU |  | 10 | ----------------- |  |
| VHDL code |  | 15 | ----------------- |  |
| Testbench code |  | 15 | ----------------- |  |
| Simulation waveform |  | 30 | ----------------- |  |
| **Total** |  | **100** |  |  |

**Number of hours spent on CE4:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (no points associated with this unless you leave it blank)

**Suggestions to improve CE4 in future years:** (use blank space below)

1. Modeled after a lab provided with instructor notes for Digital Design and Computer Architecture, David Money Harris & Sarah L. Harris, 2nd Edition [↑](#footnote-ref-1)