# Lab 1: 32-Bit ALU and Test-bench

## Introduction

In this lab, you will design the 32-bit Arithmetic Logic Unit (ALU) that is described in Section 5.2.4 of the "Digital Design and Computer Architecture" by "Harris & Harris". Your ALU will become an important part of the MIPS microprocessor that you will build in later labs. In this lab you will design an ALU in Verilog. You will also write a Verilog test-bench with a test vector file to test the ALU.

## Background

You should already be familiar with the ALU from Chapter 5 of the textbook. The design in this lab will demonstrate the ways in which Verilog encoding 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 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 Verilog.

## 1) Verilog code

You will only be doing ModelSim simulation in this lab, so you can use your favorite text editor (such as TextPad).

Create a 32-bit ALU in Verilog. Name the file alu.v. It should have the following module declaration:

```
module alu(input [31:0] a, b,
           input [2:0]  f,
           output [31:0] y,
           output zero);
```

The output `zero` should be TRUE if `y` is equal to zero.

An adder is a relatively expensive piece of hardware. Be sure that your design uses no more than one adder.

## 2) Simulation and Testing

Now you can test the 32-bit ALU in ModelSim. Develop an appropriate set of test vectors to convince a reasonable person that your design is probably correct. Note that part of your grade depends on the ability of your test-bench to cover possible corner cases. Complete Table 1 to verify that all 5 ALU operations work as they are supposed to. Note that the values are expressed in **hexadecimal** to reduce the amount of writing.

| Test | F[2:0] | A | B | Y | Zero |
|---|---|---|---|---|---|
| ADD 0+0 | 2 | 00000000 | 00000000 | 00000000 | 1 |
| ADD 0+(-1) | 2 | 00000000 | FFFFFFFF | FFFFFFFF | 0 |
| ADD 1+(-1) | 2 | 00000001 | FFFFFFFF | 00000000 | 1 |
| ADD FF+1 | 2 | 000000FF | 00000001 | | |
| SUB 0-0 | 6 | 00000000 | 00000000 | 00000000 | 1 |
| SUB 0-(-1) | | 00000000 | FFFFFFFF | 00000001 | 0 |
| SUB 1-1 | | 00000001 | | | |
| SUB 100-1 | | 00000100 | | | |
| SLT 0,0 | 7 | 00000000 | 00000000 | 00000000 | 1 |
| SLT 0,1 | | 00000000 | | 00000001 | 0 |
| SLT 0,-1 | | 00000000 | | | |
| SLT 1,0 | | 00000001 | | | |
| SLT -1,0 | | FFFFFFFF | | | |
| AND FFFFFFFF, FFFFFFFF | | FFFFFFFF | | | |
| AND FFFFFFFF, 12345678 | | FFFFFFFF | 12345678 | 12345678 | 0 |
| AND 12345678, 87654321 | | 12345678 | | | |
| AND 00000000, FFFFFFFF | | 00000000 | | | |
| OR FFFFFFFF, FFFFFFFF | | FFFFFFFF | | | |
| OR 12345678, 87654321 | | 12345678 | | | |
| OR 00000000, FFFFFFFF | | 00000000 | | | |
| OR 00000000, 00000000 | | 00000000 | | | |

**Table 1. ALU operations**

Build a self-checking test-bench to test your 32-bit ALU. Such self-checking test-bench reads the operation and the input values (i.e., test-vectors) from a test-file, computes the outputs, and compares the computed values with the output values also stored in the same test-file. Create a file called alu.tv with all your vectors. For example, the file for describing the first three lines in Table 1 might look like this:

```
2    00000000    00000000    00000000    1
2    00000000    FFFFFFFF    FFFFFFFF    0
2    00000001    FFFFFFFF    00000000    1
```

**Hint:** Remember that each hexadecimal digit in the test vector file represents 4 bits. Be careful when pulling signals from the file that are not multiples of four bits.

You can create the test vector file in any text editor, but make sure you save it as text only, and be sure the program does not append any unexpected characters on the end of your file. For example, in WordPad select **File→Save As**. In the “Save as type” box choose “Text Document

– MS-DOS Format" and type "alu.tv" in the File name box. It will warn you that you are saving your document in Text Only format, click "Yes".

Now create a self-checking test-bench for your ALU. Name it testbench.v.

Compile your ALU and test-bench in ModelSim and simulate the design. Run for a long enough time to check all the vectors. If you encounter any errors, correct your design and rerun. It is a good idea to add a line with an incorrect vector to the end of the test vector file to verify that the test-bench works!

## What to Turn In

Please turn in a single pdf file, which has all the following items in the following order and clearly labeled, along with your source codes:

1. **Please indicate how many hours you spent on this lab.** This will be helpful for calibrating the workload for next time the course is taught. Failure to provide may result in a loss of points.
2. Your table of test vectors (Table 1).
3. Your alu.v file.
4. Your alu.tv file.
5. Your testbench.v file.
6. Images of your test waveforms. Make sure these are readable and that they're printed in hexadecimal. Your test waveforms should include only the following signals in the following order, from top to bottom: `f, a, b, y, zero.` Please change the radix of the f signal to unsigned decimal, and a, b, and y signals to hexadecimal.
7. If you have any feedback on how we might make the lab even better for next quarter, that's always welcome. Please submit it in writing at the end of your lab

## How to Turn In

Lab assignments should be submitted via gradescope by uploading pdf file and providing a hyperlink in the pdf to the code. Make sure that the link is clickable. Note, do not cut and paste Verilog code into the pdf!

## Acknowledgements

This lab was adapted from Harris and Harris textbook's supplementary material for instructors

 