Digital Design and Computer Architecture

J. Spjut & M. Spencer

Lab 5: 32-Bit ALU and Testbench

Introduction

In this lab, you will design the 32-bit Arithmetic Logic Unit (ALU) that is described in Section 5.2.4 of the text. 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 SystemVerilog. You will also write a SystemVerilog testbench and testvector 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 SystemVerilog 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 your full adder from Lab 1, 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 SystemVerilog.

1) SystemVerilog code

You will only be doing ModelSim simulation in this lab, so you can use your favorite text editor (such as WordPad) instead of Quartus if you like.

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

```
module alu(input  logic [31:0] a, b,
           input  logic [1:0]  ALUControl,
           output logic [31:0] Result,
           output logic [3:0]  Flags);
```

The four bits of `Flags` should be TRUE if a condition is met. The four flags are as follows:

Flag bit	Meaning
3	Result is negative
2	Result is 0
1	The adder produces a carry out
0	The adder results in overflow

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. It is prudent to think through a set of input vectors

Develop an appropriate set of test vectors to convince a reasonable person that your design is probably correct. 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	ALUControl[1:0]	A	B	Y	Flags
ADD 0+0	0	00000000	00000000	00000000	4
ADD 0+(-1)	0	00000000	FFFFFFFF	FFFFFFFF	8
ADD 1+(-1)	0	00000001	FFFFFFFF	00000000	6
ADD FF+1	0	000000FF	00000001		
SUB 0-0	1	00000000	00000000	00000000	6
SUB 0-(-1)		00000000	FFFFFFFF	00000001	0
SUB 1-1		00000001			
SUB 100-1		00000100			
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 testbech to test your 32-bit ALU. To do this, you'll need a file containing test vectors. 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:

0_00000000_00000000_00000000_4
0_00000000_FFFFFFFF_FFFFFFFF_8
0_00000001_FFFFFFFF_00000000_6

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 testbench for your ALU. Name it testbench.sv.

Compile your alu and testbench in ModelSim and simulate the design. Run for a long enough time to check all of 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 testbench works!

What to Turn In

Please turn in each of the following items as a single pdf to the class Sakai site (in the following order and clearly labeled):

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.sv file.
4. Your alu.tv file.
5. Your testbench.sv 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: `ALUControl, a, b, Result, Flags.`
7. If you have any feedback on how we might make the lab even better for next semester, that's always welcome. Please submit it in writing at the end of your lab