Computer Organization, Spring 2023

Lab 2 : ALU & Shifter

Due Date: 2023 / 4 / 16

1. Goal

In lab 2, we are going to implement an ALU (Arithmetic Logic Unit) and a shifter by Verilog. Through this lab, students will learn how to design the function unit of a processor to support its instruction set. Note that you should design these circuits in gate- level as combinational logic instead of sequential logic. The ALU and shifter designed in this lab may also be used in most of the succeeding lab units.

2. Lab Description

A. Attached files

The attached file is composed of "ALU.v", "ALU_1bit.v", "Full_adder.v", "Shifter.v", and "TestBench.v". Please use these files to accomplish the design of your ALU and shifter. You may create additional module (.v file) for your design, if necessary.

B. Arithmetic Logic Unit (ALU)

The block diagram of the 32-bit ALU for this lab is shown in Figure 1, and the operations and corresponded control signals for this ALU are described in Table 1.

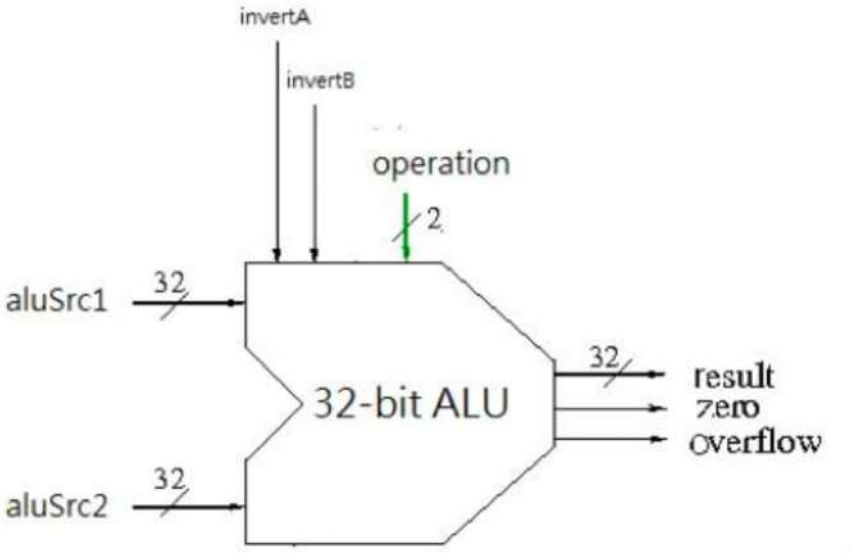

Figure 1: 32-bit ALU

Operation	Control line			Value ofresult
	invertA	invertB	operation	
Add	0	0	10	aluSrc1 + aluSrc2
Sub	0	1	10	aluSrc1 -aluSrc2
And	0	0	01	aluSrc1 & aluSrc2
Or	0	0	00	aluSrc1 | aluSrc2
Nand	1	1	00	~(aluSrc1 & aluSrc2)
Nor	1	1	01	~(aluSrc1 | aluSrc2)
Slt	0	1	11	(aluSrc1 < aluSrc2)? 1: 0

Table 1: Control signals and the output of ALU

Besides computing to “result”, generate two output signals described as follows:

- zero: A 1-bit output control signal. Itis set to 1 when the computing result of ALU, “result”, is 0; else, is clear to 0.
- overflow: A 1-bit output control signal. It is set to 1 when the computing result of ALU is overflow; else, is clear to 0.

Note that all operations in Table1 are 32-bit operations. Therefore, in this lab, you should implement a typical 1-bit ALU first and then build the 32-bit ALU by 32 1-bit ALUs.

a. 1-bit ALU

The logic diagram of a typical 1-bit ALU to be implemented in this lab is shown in Figure 2.

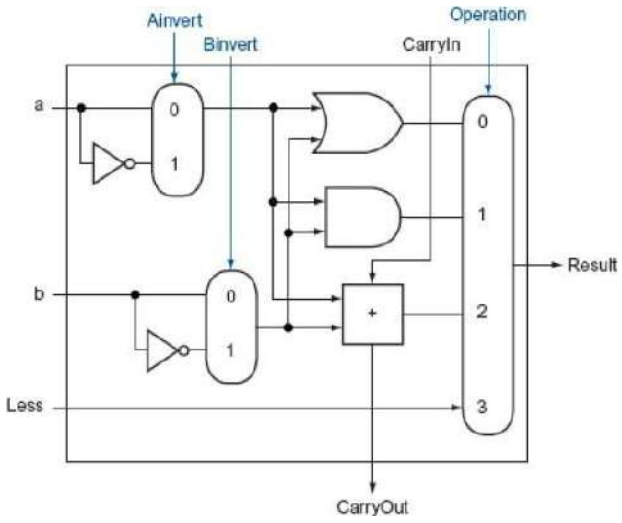

Figure 2: 1-bit ALU

b. 32-bit ALU

The block diagram of the 32-bit ALU to be implemented is shown in Figure 3.

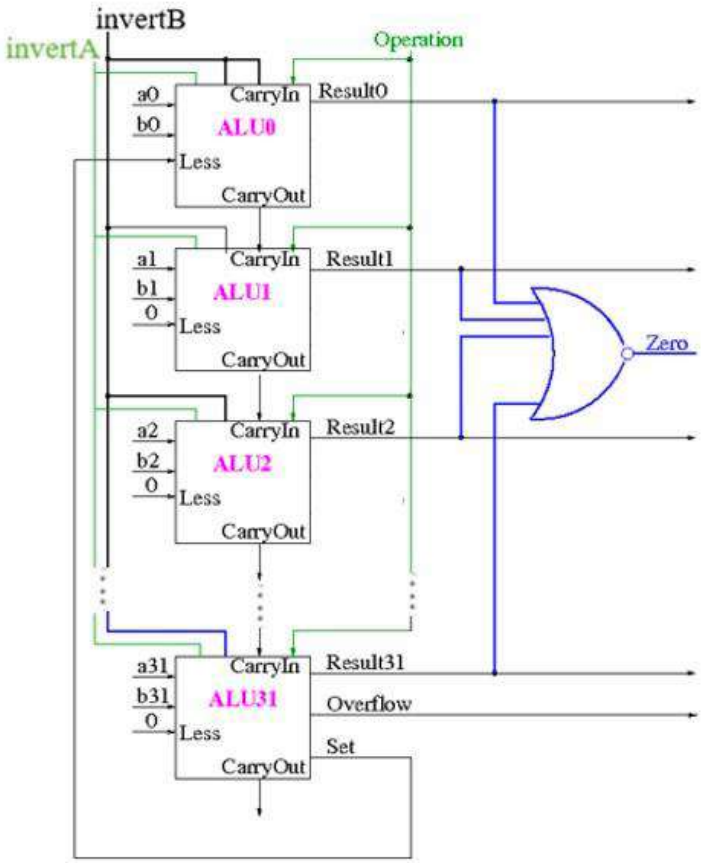

Figure3: Connection between 32 1-bit ALU

As shown in Figure 3, each input signal of ALU.v will be connected to the corresponding input of 1-bit ALU. Moreover, carryOut of ALU_i will be connected to carryIn of ALU_{i+1}.

The design of ALU31 is different from that of other typical 1-bit ALUs. There are two additional signals generated by ALU31 and are described as follows:

- set : A 1-bit control line, is generated by ALU31 and send to ALU0.
- overflow: A 1-bit output control signal, is set to 1 when overflow occurs.

C. Shifter

The block diagram of the shifter to be implemented in this lab is shown in Figure 4, and the basic operations and the corresponded control signals of the shifter are described in Table 2.

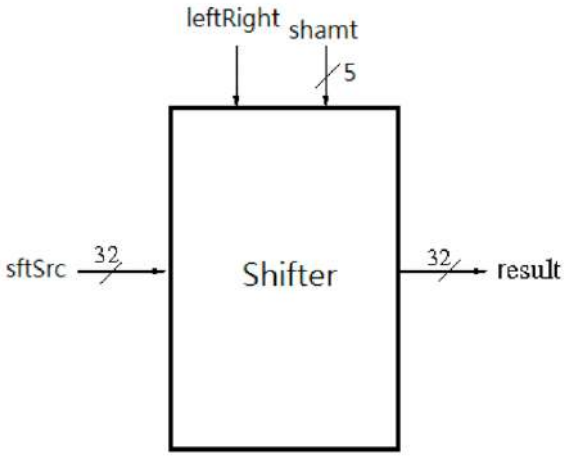

Figure 4: Shifter

Operation	Control line		Value of result
	leftRight	shamt	
Shift right logical by 1 bit	0	X (don't care)	sftSrc >> 1
Shift left logical by 1 bit	1	X (don't care)	sftSrc << 1

Table 2: Control signals and the output of shifter

This shifter is 32bits, which may logical left/right shift by one bit position each time.

Please implement it as a combinational circuit.

Shifter.v contains the following inputs and outputs:

- sftSrc: A 32-bit input data, is the source data of the shifter
- leftRight: A 1-bit input control signal. When it is set to 1, the shifter perform logical left shift; else, does logical right shift.
- shamt: A 5-bit input data, represents the number of bit positions to be shifted (invalid in the design of this basic shifter).
- result: A 32-bit output data, which represents the shifting result of the shifter.

3. TestBench

- Follow the steps in attached ppt to install and run Xilinx Vivado.
- You can use TestBench.v and the test data to test your code.
- Put all .v files and .txt(test and ans file) in the same path and use vivado to compile TestBench.v.
- If you create additional .v file, you need to include it in TestBench.v.

4. Grading

After you hand in your code, TAs will use the similar TestBench module and different test data to verify the correctness of your design of ALU and shifter.
If you have attached additional modules for your design, do ensure that those modules would not affect our testing.
The format of the test data for ALU is shown below.

(invertA)(invertB)(operation)(aluSrc1)(aluSrc2)

For example:

00100000000000000000000000000001000000000000000000000000000000011

means

add 2 3

The format of the test data for shifter is shown below.

(leftRight)(shamt)(sftSrc)

For example:

10000100000000000000000000000001001

means

shift 9 left by 1

- Total : 100 points (ALU 70%, Shifter 30%)
- The more test cases you pass, the higher score you will get.
  (The evaluation will include the public test cases provided)

5. Notice

a. Develop your lab in Xilinx Vivado.

b. Use “ALU.v” , “ALU_1bit.v, “Shifter.v”, “Full_adder.v”, and “TestBench.v” provided to implement your ALU and shifter.

c. Do not modify any existing code except the input filename in TestBench.v.

d. Write your code in “/*your code here*/”.

6. Submission

a. One person per group for this lab. Please upload your files onto new E3.

b. Please compress the files of “ALU.v ”, “ALU_1bit.v ”, “Shifter.v ”, or any files you need in your design into one zip file (only zip file can be accepted). Please names your zip file as “HW2_StudentID”. (e.g. “HW2_01234567.zip”)

c. Deadline : 2023 / 4 / 16

d. Late submissions will be penalized 20% for one day, 40% for two days, and so on.

e. Any assignment work that is fraudulent will receive a grade of zero.