Lab #3: Arithmetic and Logic Unit

EECE 2323 – Prof. Xiaolin Xu

Date: 02/15/2022

Name: Fengyi Quan && Yichen Sun

Northeastern University

1. **Background & Purpose**

The main objective of this lab is to add implementation of the shift and branch operations to the partial ALU in lab 2 to design a complete ALU. The complete ALU is adapted and improved from partial ALU. It is required to extends function from partial ALU by changing a 3-bit selector which responsible for ADD, INV, AND, OR, arithmetic shift right, logical shift left, and two cases (beq or bne) for the branch. Then generate a bitstream to run synthesis and implementation process automatically, and be tested on the TUL PYNQ board and display the results on the LEDs. Each LED represents one single bit of outputs. It has a select signal to indicate which operation should perform and two 8-bit inputs. In order to have full control over the ALU inputs, VIO provides a GUI-based software access to the ALU inputs. Fundamentally, it allows the users to enter the input values while monitoring the output of the ALU.

1. **Prelab**

**2.1. Design the ALU**

It uses a case statement to check if select signal is to indicate which operation to perform, default case should never reach since all possible cases are specified above. The ovf bit is calculated after result f computed, thus, assignment is outside of the case statement. The take\_branch is only activated when operation is beq and bne.

Graphical user interface, text

Description automatically generated

**2.2. Create Test Vector**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | binary\_a | b | binary\_b | s | f | binary\_f | ovf | take\_branch |
| 8’d0 | 0000 0000 | 8’d0 | 0000 0000 | 3’b000 | 8’d0 | 0000 0000 | 0 | 0 |
| 8’d12 | 0000 1100 | 8’d34 | 0010 0010 | 3’b000 | 8’d46 | 0010 1110 | 0 | 0 |
| -8’d12 | 1111 0100 | -8’d34 | 1101 1110 | 3’b000 | -8’d46 | 1101 0010 | 0 | 0 |
| 8’d100 | 0110 0100 | -8’d50 | 1100 1110 | 3’b000 | 8’d50 | 0011 0010 | 0 | 0 |
| -8’d100 | 1001 1100 | 8’d50 | 0011 0010 | 3’b000 | -8’d50 | 1100 1110 | 0 | 0 |
| 8’d100 | 0110 0100 | 8’d100 | 0110 0100 | 3’b000 | -8’d56 | 1100 1000 | 1 | 0 |
| -8’d100 | 1001 1100 | -8’d100 | 1001 1100 | 3’b000 | 8’d56 | 0011 1000 | 1 | 0 |
| 8’d3 | 0000 0011 | 8’d0 | 0000 0000 | 3’b000 | 8’d3 | 0000 0011 | 0 | 0 |
| -8’d3 | 1111 1101 | 8’d0 | 0000 0000 | 3’b000 | -8’d3 | 1111 1101 | 0 | 0 |
| 8’d1 | 0000 0001 | 8’d1 | 0000 0001 | 3’b000 | 8’d2 | 0000 0010 | 0 | 0 |
| 8’d75 | 0100 1011 | 8’d100 | 0110 0100 | 3’b000 | - 8’d81 | 1010 1111 | 1 | 0 |
| 8’d0 | 0000 0000 | 8’d0 | 0000 0000 | 3’b001 | -8’d1 | 1111 1111 | 0 | 0 |
| 8’d12 | 0000 1100 | 8’d34 | 0010 0010 | 3’b001 | -8’d35 | 1101 1101 | 0 | 0 |
| -8’d12 | 1111 0100 | -8’d34 | 1101 1110 | 3’b001 | 8’d33 | 0010 0001 | 0 | 0 |
| 8’d100 | 0110 0100 | -8’d50 | 1100 1110 | 3’b001 | 8’d49 | 0011 0001 | 0 | 0 |
| -8’d100 | 1001 1100 | 8’d50 | 0011 0010 | 3’b001 | -8’d51 | 1100 1101 | 0 | 0 |
| 8’d100 | 0110 0100 | 8’d100 | 0110 0100 | 3’b001 | -8’d101 | 1001 1011 | 0 | 0 |
| -8’d100 | 1001 1100 | -8’d100 | 1001 1100 | 3’b001 | 8’d99 | 0110 0011 | 0 | 0 |
| 8’d2 | 0000 0010 | 8’d3 | 0000 0011 | 3’b001 | -8’d4 | 1111 1100 | 0 | 0 |
| 8’d1 | 0000 0001 | 8’d1 | 0000 0001 | 3’b001 | -8’d2 | 1111 1110 | 0 | 0 |
| 8’d0 | 0000 0000 | 8’d0 | 0000 0000 | 3’b010 | 8’d0 | 0000 0000 | 0 | 0 |
| -8’d1 | 1111 1111 | -8’d1 | 1111 1111 | 3’b010 | -8’d1 | 1111 1111 | 0 | 0 |
| 8’d15 | 0000 1111 | 8’d35 | 0010 0011 | 3’b010 | 8’d3 | 0000 0011 | 0 | 0 |
| -8’d12 | 1111 0100 | -8’d34 | 1101 1110 | 3’b010 | -8’d44 | 1101 0100 | 0 | 0 |
| 8’d100 | 0110 0100 | -8’d50 | 1100 1110 | 3’b010 | 8’d68 | 0100 0100 | 0 | 0 |
| 8’d100 | 0110 0100 | 8’d100 | 0110 0100 | 3’b010 | 8’d100 | 0110 0100 | 0 | 0 |
| -8’d100 | 1001 1100 | -8’d100 | 1001 1100 | 3’b010 | -8’d100 | 1001 1100 | 0 | 0 |
| 8’d127 | 0111 1111 | 8’d1 | 0000 0001 | 3’b010 | 8’d1 | 0000 0001 | 0 | 0 |
| 8’d75 | 0100 1011 | 8’d102 | 0110 0110 | 3’b010 | 8’d66 | 0100 0010 | 0 | 0 |
| 8’d0 | 0000 0000 | 8’d0 | 0000 0000 | 3’b011 | 8’d0 | 0000 0000 | 0 | 0 |
| 8’d12 | 0000 1100 | 8’d34 | 0010 0010 | 3’b011 | 8’d46 | 0010 1110 | 0 | 0 |
| -8’d12 | 1111 0100 | -8’d34 | 1101 1110 | 3’b011 | -8’d2 | 1111 1110 | 0 | 0 |
| -8’d100 | 1001 1100 | 8’d50 | 0011 0010 | 3’b011 | -8’d66 | 1011 1110 | 0 | 0 |
| 8’d100 | 0110 0100 | 8’d100 | 0110 0100 | 3’b011 | 8’d100 | 0110 0100 | 0 | 0 |
| 8’d3 | 0000 0011 | 8’d0 | 0000 0000 | 3’b011 | 8’d3 | 0000 0011 | 0 | 0 |
| 8’d1 | 0000 0001 | 8’d1 | 0000 0001 | 3’b011 | 8’d1 | 0000 0001 | 0 | 0 |
| -8’d1 | 1111 1111 | -8’d1 | 1111 1111 | 3’b011 | -8’d1 | 1111 1111 | 0 | 0 |
| 8’d0 | 0000 0000 | 8’d0 | 0000 0000 | 3’b100 | 8’d0 | 0000 0000 | 0 | 0 |
| 8’d12 | 0000 1100 | 8’d34 | 0010 0010 | 3’b100 | 8’d6 | 0000 0110 | 0 | 0 |
| -8’d12 | 1111 0100 | -8’d34 | 1101 1110 | 3’b100 | -8’d6 | 1111 1010 | 0 | 0 |
| -8’d100 | 1001 1100 | 8’d50 | 0011 0010 | 3’b100 | -8’d50 | 1100 1110 | 0 | 0 |
| 8’d100 | 0110 0100 | 8’d100 | 0110 0100 | 3’b100 | 8’d50 | 0011 0010 | 0 | 0 |
| 8’d3 | 0000 0011 | 8’d0 | 0000 0000 | 3’b100 | 8’d1 | 0000 0001 | 0 | 0 |
| 8’d1 | 0000 0001 | 8’d1 | 0000 0001 | 3’b100 | 8’d0 | 0000 0000 | 0 | 0 |
| -8’d1 | 1111 1111 | -8’d1 | 1111 1111 | 3’b100 | -8’d1 | 1111 1111 | 0 | 0 |
| 8’d0 | 0000 0000 | 8’d1 | 0000 0001 | 3’b101 | 8’d0 | 0000 0000 | 0 | 0 |
| 8’d12 | 0000 1100 | 8’d2 | 00000010 | 3’b101 | 8’d48 | 0011 0000 | 0 | 0 |
| -8’d12 | 1111 0100 | 8’d3 | 00000011 | 3’b101 | -8’d96 | 1010 0000 | 0 | 0 |
| -8’d100 | 1001 1100 | 8’d4 | 00000100 | 3’b101 | -8’d64 | 1100 0000 | 0 | 0 |
| 8’d100 | 0110 0100 | 8’d1 | 0000 0001 | 3’b101 | -8’d56 | 1100 1000 | 0 | 0 |
| 8’d3 | 0000 0011 | 8’d2 | 00000010 | 3’b101 | 8’d12 | 0000 1100 | 0 | 0 |
| 8’d1 | 0000 0001 | 8’d7 | 00000111 | 3’b101 | -8’d128 | 1000 0000 | 0 | 0 |
| -8’d1 | 1111 1111 | 8’d1 | 0000 0001 | 3’b101 | -8’d2 | 1111 1110 | 0 | 0 |
| -8’d1 | 1111 1111 | 8’d0 | 0000 0001 | 3’b101 | -8’d1 | 1111 1111 | 0 | 0 |
| 8’d0 | 0000 0000 | 8’d0 | 0000 0000 | 3’b110 | 8’d0 | 0000 0000 | 0 | 1 |
| 8’d12 | 0000 1100 | 8’d34 | 0010 0010 | 3’b110 | 8’d0 | 0000 0000 | 0 | 0 |
| -8’d12 | 1111 0100 | -8’d12 | 1111 0100 | 3’b110 | 8’d0 | 0000 0000 | 0 | 1 |
| -8’d100 | 1001 1100 | 8’d50 | 0011 0010 | 3’b110 | 8’d0 | 0000 0000 | 0 | 0 |
| 8’d100 | 0110 0100 | 8’d100 | 0110 0100 | 3’b110 | 8’d0 | 0000 0000 | 0 | 1 |
| 8’d3 | 0000 0011 | 8’d3 | 0000 0011 | 3’b110 | 8’d0 | 0000 0000 | 0 | 1 |
| 8’d1 | 0000 0001 | 8’d1 | 0000 0001 | 3’b110 | 8’d0 | 0000 0000 | 0 | 1 |
| -8’d1 | 1111 1111 | -8’d1 | 1111 1111 | 3’b110 | 8’d0 | 0000 0000 | 0 | 1 |
| 8’d0 | 0000 0000 | 8’d0 | 0000 0000 | 3’b111 | 8’d0 | 0000 0000 | 0 | 0 |
| 8’d12 | 0000 1100 | 8’d34 | 0010 0010 | 3’b111 | 8’d0 | 0000 0000 | 0 | 1 |
| -8’d12 | 1111 0100 | -8’d34 | 1101 1110 | 3’b111 | 8’d0 | 0000 0000 | 0 | 1 |
| -8’d100 | 1001 1100 | 8’d50 | 0011 0010 | 3’b111 | 8’d0 | 0000 0000 | 0 | 1 |
| 8’d100 | 0110 0100 | 8’d100 | 0110 0100 | 3’b111 | 8’d0 | 0000 0000 | 0 | 0 |
| 8’d3 | 0000 0011 | 8’d0 | 0000 0000 | 3’b111 | 8’d0 | 0000 0000 | 0 | 1 |
| 8’d1 | 0000 0001 | 8’d1 | 0000 0001 | 3’b111 | 8’d0 | 0000 0000 | 0 | 0 |
| -8’d1 | 1111 1111 | -8’d1 | 1111 1111 | 3’b111 | 8’d0 | 0000 0000 | 0 | 0 |

**2.3. Testbench**

Please see Appendix B: ALU testbench.

**2.4. Simulation**

For waveform, please see Appendix C: ALU testbench simulation waveform. For sake of testing easily, we print test result to TCL console to visualize if the output matches the expected result. It prints the details of each signal change, the result = 1 if it matches the expected result and result = 0 if it does not match. In all test cases, all test passed and matched the expected result. For more information, please seeAppendix D: ALU testbench TCL console output.

1. **Results and Analysis**

The ALU has three input signals, including two operands a and b, and one operator select signal. Since it has been extended to offer 8 different operations, the s select is set to be 3-bit. Arithmetic shift right, logical shift left, branch if equal and branch if not equal has been added to the module. The difference between arithmetic and logical shift is that arithmetic shift treats the number as signed number and retains the most significant bit which means it keeps the sign of a number. And logical shift treats the number as plain bits and simply shift all by given index. In lab, some students encountered the issue that arithmetic shift does not recognize the most significant bit. It can be solved by specifying the signed keyword to the input a and b to manually set the signed binary to 2’s complement. By observing the result, it can tell that shift left by n bit is equivalent to multiply the number by 2n and shift right is equivalent to divide the number by 2n but it always rounds down (towards negative infinity), because right shift would eliminate the least significant bit which could loss precision. Bit shifting is faster than arithmetic operations, such as multiplication and division in programming. Thus, it might be useful to substitute the arithmetic operations by bit shifting if it matches the above requirement. The beq and bne outputs to take\_branch. The beq is to check if two binaries are equal and the bne is to check if two binaries are not equal. The f output and ovf are always set to be 0 because they did not involve at all. The take\_branch is set only for these two operations and set to be 0 in any other case.

The ALU was tested using the Virtual Input/Output (VIO) functionality provided by the Xilinx Vivado. This provides a more efficient way to test the hardware behavior of the module. A comprehensive suite of test cases was performed to ensure that all input and output bits were either set or unset during the test case.

1. **Conclusion & Recommendations**

The objective of this lab is to extend the functionalities on a partial ALU in Verilog and simulated logic circuit with testbench in the Xilinx, then tested the ALU by using the Virtual Input/Output (VIO). The ALU performed 8 different operations with two operands successfully. The VIO functionality in the Xilinx provided more useful method to test the complex hardware and circuit. The complete design of the ALU is beneficial to implement other digital systems.

1. **Appendices**

Appendix A: ALU Module

Graphical user interface, text

Description automatically generated

Appendix B: ALU testbench

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Appendix C: ALU testbench simulation waveform

A screenshot of a computer

Description automatically generatedGraphical user interface

Description automatically generated

Appendix D: ALU testbench TCL console output

Table

Description automatically generated Text

Description automatically generated