Herbert Wertheim College of Engineering
Department of Electrical & Computer Engineering
UNIVERSITY of FLORIDA

**Dr. Christophe Bobda**
Omar Elbasri
**EEL3701C - Fall 2025**

# Lab 4: ALU Design

## Overview

An ALU (Arithmetic Logic Unit) is the heart of a CPU. Most calculations in a computer are performed by an ALU, and computers do hundreds of millions of these calculations per second. In Part 1 of this lab, a combinational ALU will be designed. In Part 2, registers will be added to this ALU to make it a registered ALU, capable of saving data and performing operations in sequence. This lab will be used as a component in Lab 6 to create a simple CPU.

## Pre-Lab Procedure: Part 1 - Combinational ALU

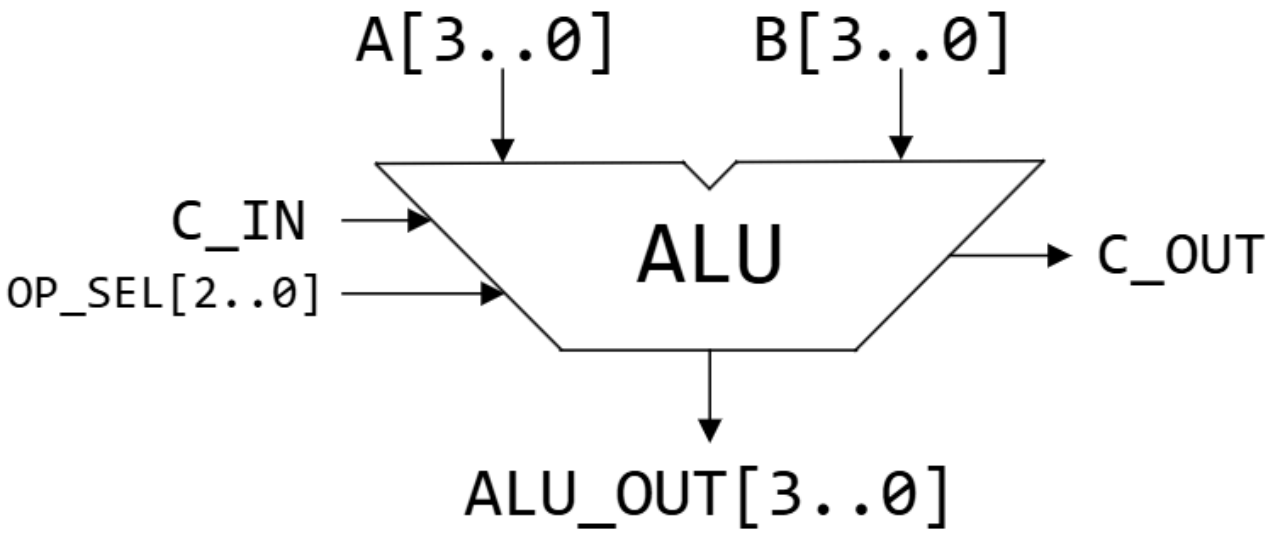



Figure 1: Combinational ALU Block Diagram

In this part, a purely combinational ALU will be created. The inputs to the ALU (A and B) will be 4 bits wide each, with a single bit for a carry input, and a 3 bit input for selecting the operation of the ALU. The output of the ALU is 4 bits, with a single bit for a carry output.

A table for the operations is shown below. First, the output for each operation needs to be generated, then the desired operation should be chosen with a multiplexer. Think carefully about how to implement shifting, and the 74’283 adder component within Quartus should be used, instead of designing your own. All of the generated signals will be **4 bits wide**.

UF Herbert Wertheim College of Engineering
Department of Electrical & Computer Engineering
UNIVERSITY of FLORIDA

**Dr. Christophe Bobda**
Omar Elbasri
**EEL3701C - Fall 2025**

Rather than using a pre-made multiplexer component, you will design a 4-bit 8-to-1 multiplexer, mux8x1.vhd, using VHDL. VHDL is a Hardware Description Language (HDL) that allows digital circuits to be described using text, rather than graphically as you've done previously. This short VHDL tutorial should supplement in-class demonstrations (Hint: with select statements will be very useful for creating a multiplexer). Note that process blocks are **not allowed** in this course. To import this multiplexer component into the greater .bdf file, follow the instructions here at the end of the Lab 3 document for creating a block symbol (.bsf) for a VHDL file.

| Operation | OP_SEL |
|---|---|
| Bitwise AND of A and B | 000 |
| Bitwise OR of A and B | 001 |
| The sum of A and B with carry | 010 |
| Bitwise negation of A | 011 |
| A left shifted by 1 bit | 100 |
| A right shifted by 1 bit | 101 |
| Output A | 110 |
| Output B | 111 |

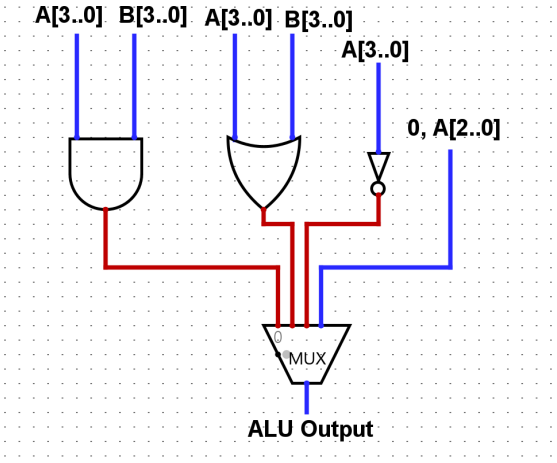



Table 1: ALU operation table and Figure 2: Simple example Logisim ALU

Shown above is an example of a simplified ALU in Logisim with 4 operations: AND, OR, NOT A, and RIGHT SHIFT A. This lab, similar to the previous lab, will have to be built and simulated in Quartus rather than Logisim. Logisim may prove useful for quickly testing designs, but everything else must be done in Quartus.

To ensure functionality, you will have to simulate your design in Quartus. Each operation of the ALU must be demoed with a meaningful test, for example 0000 AND 0000 is not a meaningful test. Both the carry in and carry out for addition should be shown in at least 2 separate operations.

**Part 1 Summary:**

1. Design a 4-bit 8-to-1 multiplexer using VHDL in a file named mux8x1.vhd.
2. Design a 4-bit combinational ALU with the operations described. The ALU must be designed in a file named lab4_alu.bdf and must use the mux8x1 block symbol of mux8x1.vhd

3. Simulate your ALU design with a testbench that demos each operation with functionality shown, showing both the carry in and carry out for addition using 2 separate operations. This should be done using a University Waveform (.vwf) file. It is not necessary to program your DE10 in this section.
4. Submit annotated screenshots of your simulation, detailing the inputs, outputs, and expected results.
5. Submit a screenshot of your lab4_alu.bdf design, including inputs and outputs.

## Pre-Lab Questions: Part 1

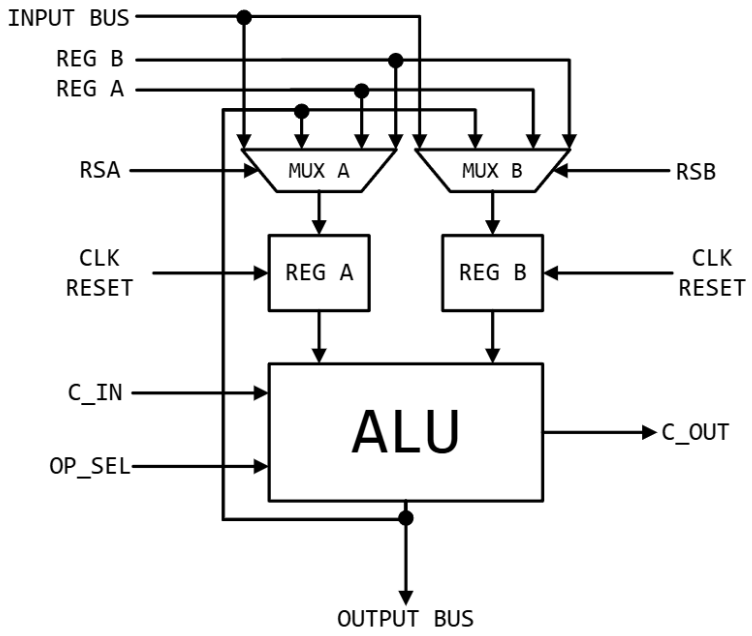
1. How can you divide and multiply by powers of 2 on the ALU?

## Pre-Lab Procedure: Part 2 - Registered ALU

To be able to do operations more effectively, we need to be able to save data, whether it be from an input or the result of an operation. At the moment, the combinational ALU is a calculator, but it cannot do operations in sequence without the addition of registers.

In this part of the lab, you will add two registers and supporting circuitry to your ALU in order to increase functionality and flexibility. The register's input will be multiplexed to change where the data going into the registers is coming from, outlined in the table below. Rather than inputting straight into the ALU, the inputs into the ALU will come from the registers, and the data in the registers will come from an input bus.

This part should be done in the same project. To use the ALU from part 1 within the project, with the file open, go to File -> Create/Update -> Create Symbol File. Afterwards, double click to add a new component, then you should see your ALU from part 1 under the project folder. This process is very similar to how to create a .bsf for a VHDL file. To change which file is the "top-level", or the one that gets simulated and programmed to the board, change from "Hierarchy" to "Files" on the left, right click your new BDF file and click "Set as Top-Level Entity". Alternatively, with your new BDF file open, hit CTRL+SHIFT+J on your keyboard.




| RSA, RSB | Data Source |
|---|---|
| 00 | Input Bus |
| 01 | Output Bus (ALU Out) |
| 10 | Register A Output |
| 11 | Register B Output |

Figure 3: RALU Block Diagram and Table 2: RSA/RSB Table

Using RSA and RSB, we can select where the data going into Register A and Register B are coming from. For example, if we want to load data from the input bus into Register B while taking the output of the ALU into Register A, we set RSA = 01 (output bus) and RSB = 00 (input bus). On the next clock cycle, if we want to hold the value of Register A and have Register B get data from the ALU, we would set RSA = 10 (Register A output) and RSB = 01 (output bus).

From here, we can take RSA and RSB and combine it with the OP_SEL of the ALU to form a 7-bit control word that determines the operation of the ALU on a given clock cycle. We put these control words in a sequence and we now have a sequence of operations the ALU will perform on a given input.

As an example, let's say we want to create a program to add the number 2 to the number 3, then add 2 to the output of that. Each row represents one clock cycle, so after setting the proper inputs, the clock should toggle, and then this process should be repeated.

| RSA | RSB | OP_SEL | INPUT_BUS | Explanation |
|---|---|---|---|---|
| 00 | 11 | 000 | 0010 | Register A needs the value 2 within it, so we set the input bus to 2 and make Register A take data from the input bus. Register B can be left alone as we don't do anything with it this cycle. OP_SEL can be anything as we don't use the output of the ALU. |
| 10 | 00 | 000 | 0011 | Like the previous cycle, we put the value of 3 on the input bus and put it into register B. OP_SEL can be anything because the output of the ALU isn't being used. |
| 10 | 01 | 010 | 0000 | Register A has the value of 2, so we leave it alone. OP_SEL = 010 which is the addition operation. Currently, Register A contains 2 and Register B contains 3 so the output of the ALU is 5 before the clock toggles. RSA and RSB determine where the data is coming from **after** the clock edge, so setting RSB = 01 results in 5 getting put into Register B, as that is the output of the ALU when the clock toggles. |
| 10 | 01 | 010 | 0000 | This is the same as last cycle, and this will result in 7 getting put into Register B. This can continue as long as we want. |

Table 3: Example program explanation table

In addition to designing the Registered ALU, you will have to devise a program to perform this sequence of operations listed: **Add 5 and 2, take that result and multiply it by 2, bitwise AND with 7, divide that result by 4, then add 3, and store the result in B.** This entire sequence should take no more than 10 clock cycles to complete. Think carefully about what RSA, RSB, OP_SEL, and the Input Bus should be. Carry in and carry out can be ignored for this sequence of operations, but be sure to set C_IN to 0 in Quartus, as the input left floating could cause errors. You will submit an annotated screenshot of this simulation, notating which operation is being done on every clock cycle.

Your registers should use the "dff" (D Flip-Flop) component in Quartus. The multiplexers going into Register A and Register B should be a 4-bit 4-to-1 multiplexer made in VHDL in a file named mux4x1.vhd. There should be an asynchronous, active low clear input for the registers, connected to the CLRN input of the D Flip-Flops.

UF Herbert Wertheim College of Engineering
Department of Electrical & Computer Engineering
UNIVERSITY of FLORIDA

**Dr. Christophe Bobda**
Omar Elbasri
**EEL3701C - Fall 2025**

The Registered ALU must be programmed onto your DE10, and you should use buttons and switches for the inputs, as well as LEDs and seven segment displays for the output. The switches of the DE10 should have RSA, RSB, OP_SEL, the lower 2 bits of INPUT_BUS, and CLK. The 2nd highest bit of INPUT_BUS should be on one of the buttons. Recall that the buttons are active low, so holding it down will result in a value of 0, and letting it go will result in a value of 1.

**Part 2 Summary**:

1. Design and implement the Registered ALU in a Quartus .bdf named lab4_ralu.bdf, using the ALU from Part 1 as a component. Design a 4-bit 4x1 multiplexer titled mux4x1.vhd. Your lab document should include the following:
    a. Annotated screenshots of your .vwf simulation showing the program described in part 2.
    b. Screenshots of your part 2 design including inputs and outputs
2. Program the Registered ALU to the DE10 Lite and make sure it is functional.
    a. At minimum, you should have the output of Register A, Register B, and the ALU on 3 seven segment displays, using the ssgdecoder.vhd from Lab 3.
    b. The switches should be mapped to control RSA[1..0], RSB[1..0], OP_SEL[2..0], the lower 2 bits of the INPUT_BUS (INPUT_BUS[1..0]), and CLK.
    c. Map the 2nd highest bit of the INPUT_BUS (INPUT_BUS[2]) to one of the buttons. Recall that the buttons are active low, so holding it down will result in a value of 0, and letting it go will result in a value of 1.

## Pre-Lab Questions: Part 2

1. How can you get the 2's complement of a number in Register A and put it in Register B?
2. Using the table columns below, make control sequences to do the following operations. These programs are **not** to be simulated. The values in the REG_A and REG_B columns should be the values of registers **after** the clock has toggled. The process of loading values into registers should also be shown. If it is unknown what the value of a register will be, or dependent on the previous state of the RALU, put a question mark (?). If the result location is not explicitly stated, either register is fine. Add rows as necessary, and keep each table separate.
    a. Add 6 and A together.
    b. Bitwise OR of E and 3.
    c. Divide 7 by 4, take the result and add it to 5, then multiply that result by 2.
    d. Take the 2's complement of 6.
    e. The control sequence from Part 2 of the lab: Add 5 and 2, take that result and multiply it by 2, bitwise AND with 7, divide that result by 4, then add 3, and store the result in B.

Herbert Wertheim College of Engineering
Department of Electrical & Computer Engineering
UNIVERSITY of FLORIDA

**Dr. Christophe Bobda**
Omar Elbasri
**EEL3701C - Fall 2025**

| RSA | RSB | OP_SEL | INPUT_BUS | REG_A | REG_B | Explanation |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |

## In-Lab Procedure

1. Demo the sequence in Part 2 to your PI using the DE-10.
2. Complete the in-lab quiz as specified by your PI.

## Submission Files

- lab4.qar (Quartus project archive file)
- lab4.pdf

## Appendix

VHDL Tutorial: https://nandland.com/introduction-to-vhdl-for-beginners-with-code-examples/
VHDL symbol tutorial (Appendix B): https://eel3701.ece.ufl.edu/assignments/labs/lab3.html
ssgdecoder.vhd: https://eel3701.ece.ufl.edu/assets/misc/ssgdecoder.vhd