Bangladesh University of Engineering and Technology

Department of Computer Science and Engineering

CSE 210: Computer Architecture Sessional

Assignment-1 : 4-bit ALU Simulation & Design

Submitted By: (A2-Group 6)

- 2105031 - Nabiha Tahseen
- 2105032 - Nawriz Ahmed Turjo
- 2105033 - Abhishek Roy
- 2105035 - Debashri Roy
- 2105048 - Shams Hossain Simanto

Date of Submission: 26 October, 2024

1 Introduction

An Arithmetic Logic Unit (ALU) is a fundamental component of the central processing unit (CPU), serving as the computational core responsible for executing arithmetic and logical operations essential to various computing tasks. The ALU performs basic arithmetic operations such as addition, subtraction, multiplication, and division, as well as logical operations like AND, OR, NOT, and XOR. By manipulating binary data, consisting of 0s and 1s, the ALU facilitates complex calculations, data processing, and the execution of program instructions. The efficiency and speed of the ALU have a significant impact on the overall performance of a processor, influencing how well a computer handles computational tasks.

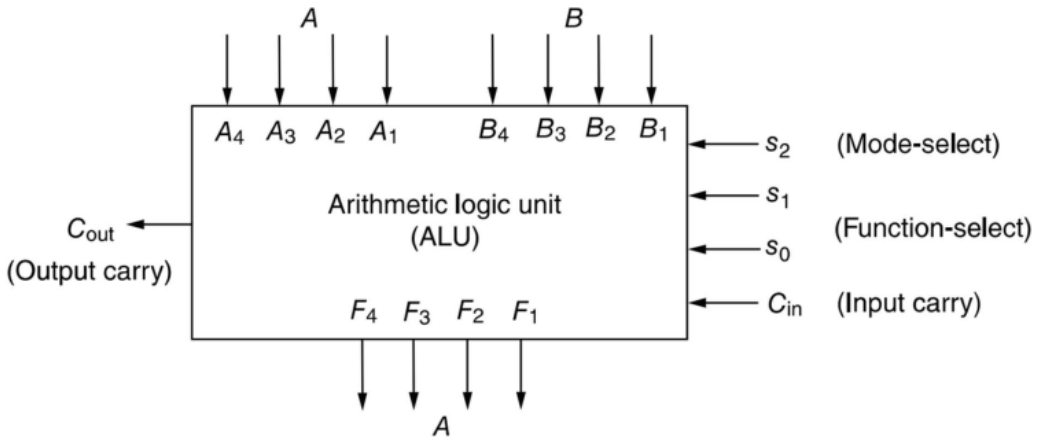


Figure 1: Block Diagram of 4-bit ALU

An ALU is typically divided into two main units: the Arithmetic Unit and the Logic Unit. These units work together to perform operations like addition, subtraction, increment, decrement, and logical functions. In the case of a 4-bit ALU, the input data consists of four bits, with output being similarly represented as a 4-bit number. To control the operations, selection lines or control bits are used to choose between various arithmetic and logical functions. The designed 4-bit ALU operates with three selection inputs to specify different operations. It can carry out up to eight distinct arithmetic and logic operations using a combination of function-select inputs. In addition, four status flags—Carry (C), Zero (Z), Overflow (V), and Sign (S)—indicate important characteristics of the ALU's outputs.

C (Carry Flag): C is the Cout of the adder output in the ALU. If there is any carry out, then C = 1, otherwise it is found to be 0. In Logical operations, it is always 0.

Z (Zero Flag): If the last operation of the ALU gives an output of 0, Z = 1, otherwise it is 0.

V (Overflow Flag): In arithmetic operation, if the result exceeds the range of the used bits, an overflow occurs, that is an overflow and V = 1. In any logical operation, it is 0. The

equation of V is :

$$V = C_3 \oplus C_{\text{out}}$$

S (Sign Flag): It is the MSB of the output.

2 Problem Specification

In this assignment, we have built a 4-bit ALU using Logisim as software simulation as well as implementing the hardware. Instead of using the selection input specified in the ALU, we have constructed a customized ALU with distinct selections for arithmetic and logical operations. We have also implemented different flags such as CARRY (C), OVERFLOW (V), ZERO (Z), and SIGN (S) indicating different result specifications of the ALU. For the arithmetic part, we implemented ADD WITH CARRY, SUB EITH BORROW, COMPLEMENT A, DECREMENT A, OR and AND operation. The operations are selected using the following bits:

cs2	**cs1**	**cs0**	**Functions**	**Output**
0	0	0	Add with Carry	A+B+1
0	X	1	OR	A ∪ B
0	1	0	Sub with Borrow	A+B'
1	0	0	Complement A	A'
1	X	1	AND	A ∩ B
1	1	0	Decrement A	A-1

Table 1: Operations of Custom ALU

The schematic diagram for the ALU is given below:
(n.e: the flags are not shown in here)

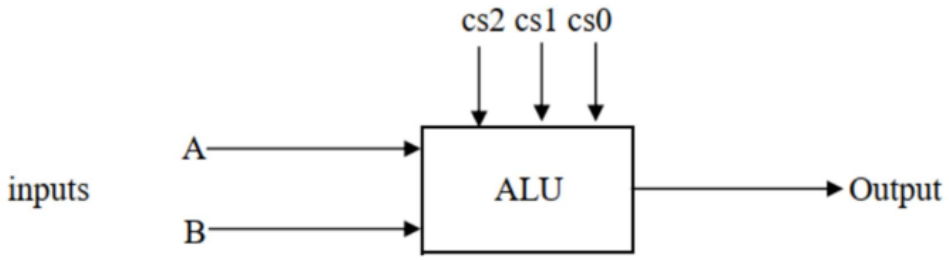


Figure 2: Prototype Block Diagram for the Custom Alu

3 Detailed Design Steps with k-maps

3.1 Design Steps

1. The arithmetic unit computes arithmetic and logical operations with the help of a 4-bit full adder and multiplexer.

2. 4x1 multiplexer is used for each bit to select from the options. For the first input, it can be seen that there are four possible values of X_i where the values are $A, A \cap B, A \cup B$ *and* A' , which are selected by the multiplexer having S_{x2}, S_{x1} selection bit and an enable bit en_x. For the second input, also there are four possible values of Y_i where the values are 0, 1, B, B', which are selected by the multiplexer having S_{y2}, S_{y1} selection bit and an enable bit en_y.

3. To implement the flags, it is observed that the overflow flag requires the previous carry from the adder, which cannot be directly found from an adder IC. So,

$$V = O_3 \oplus A_3 \oplus B_3 \oplus C_{\text{out}}$$

$$\Rightarrow O_3 = A_3 \oplus B_3 \oplus c_3$$

   Hence,

$$V = O_3 \oplus A_3 \oplus B_3 \oplus C_{\text{out}} = C_3 \oplus C_{\text{out}}$$

4. The zero flag, Z, is computed by adding the 4 output bits using OR gates and then inverting it using an NOT gate.

5. The carry flag is simply the *Cout* of the adder.

6. The sign flag can be found from O_4.

3.2 K-maps

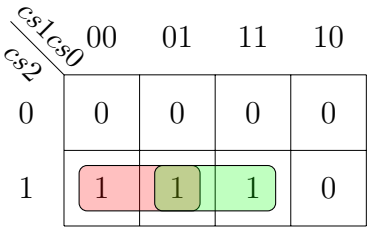
3.2.1 K-maps for S_{x_2}

cs_2 \ cs_1cs_0	00	01	11	10
0	0	1	1	0
1	0	1	1	0

We can express S_{x_2} as following:

$$S_{x_2} = cs_0$$

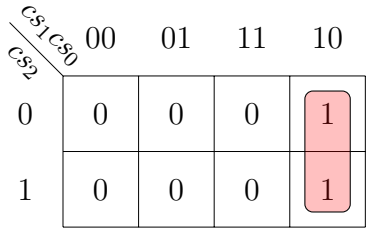
3.2.2 K-maps for S_{x_1}



We can express S_{x_1} as following:

$$S_{x_1} = cs_2(cs_1' + cs_0)$$

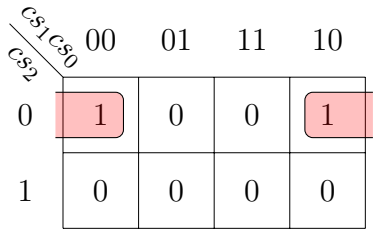
3.2.3 K-maps for S_{y_2}



We can express S_{y_2} as following:

$$S_{y_2} = cs_1\overline{cs_0}$$

3.2.4 K-maps for S_{y_1}



We can express S_{y_1} as following:

$$S_{y_1} = cs_2'cs_0'$$

4 Truth Table

For better interpretation of the variables used, refer to Figure 2.

cs2	cs1	cs0	Fuction	Operation	Xi	Yi	Zi	S_{x_2}	S_{x_1}	Ex	S_{y_2}	S_{y_1}	Ey
0	0	0	Add with carry	A+B+1	A	B	1	0	0	0	0	1	0
0	0	1	OR	A∪B	A∪B	0	0	1	0	0	0	0	0
0	1	0	Sub with borrow	A+B'	A	B'	0	0	0	0	1	1	0
0	1	1	OR	A∪B	A∪B	0	0	1	0	0	0	0	0
1	0	0	Complement A	A'	A'	0	0	0	1	0	0	0	0
1	0	1	AND	A∩B	A∩B	0	0	1	1	0	0	0	0
1	1	0	Decrement A	A-1	A	1	0	0	0	0	1	0	0
1	1	1	AND	A∩B	A∩B	0	0	1	1	0	0	0	0

Table 2: Truth Table for Intermediate I/0

5 Block Diagram

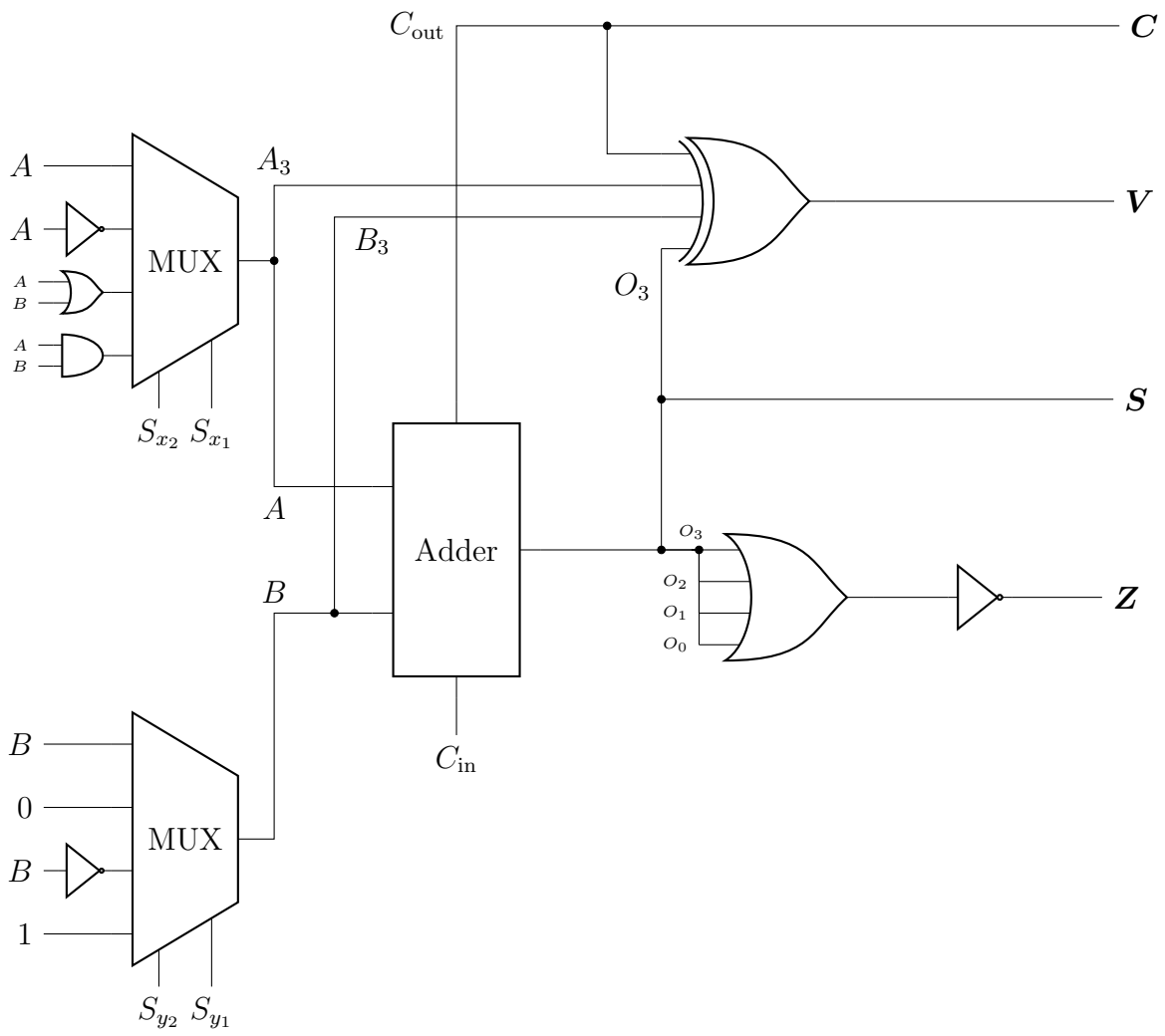


Figure 3: Block Diagram of ALU

6 Complete Circuit diagram

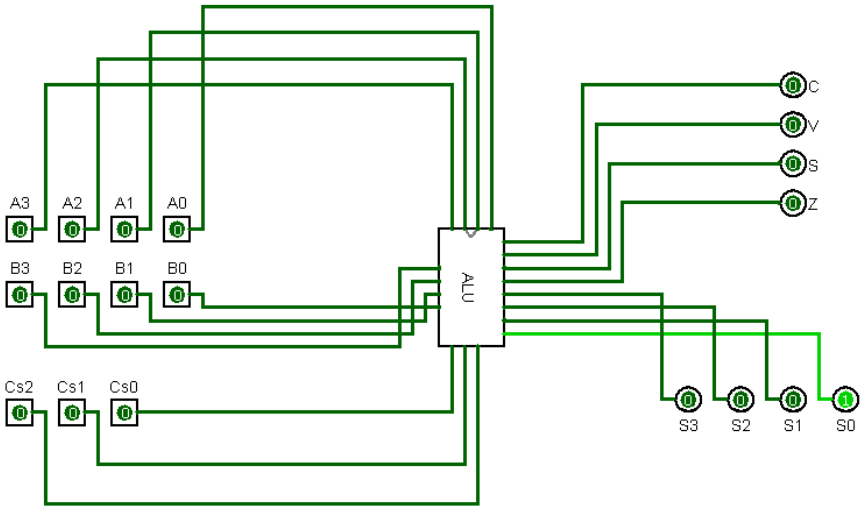


Figure 4: High-level view of the ALU, illustrating the inputs, outputs, control signals, and flags used in our design.

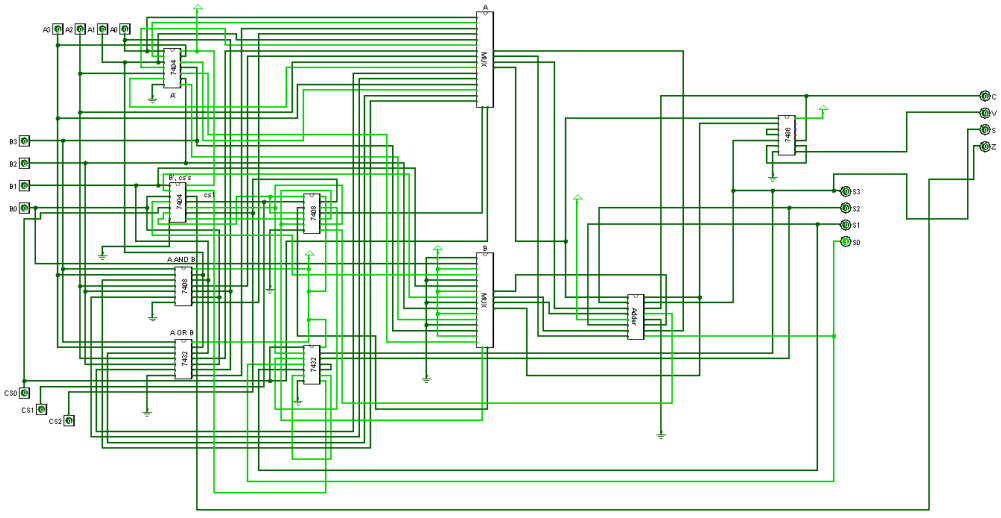


Figure 5: Complete Design of Arithmetic Logic Unit (ALU)

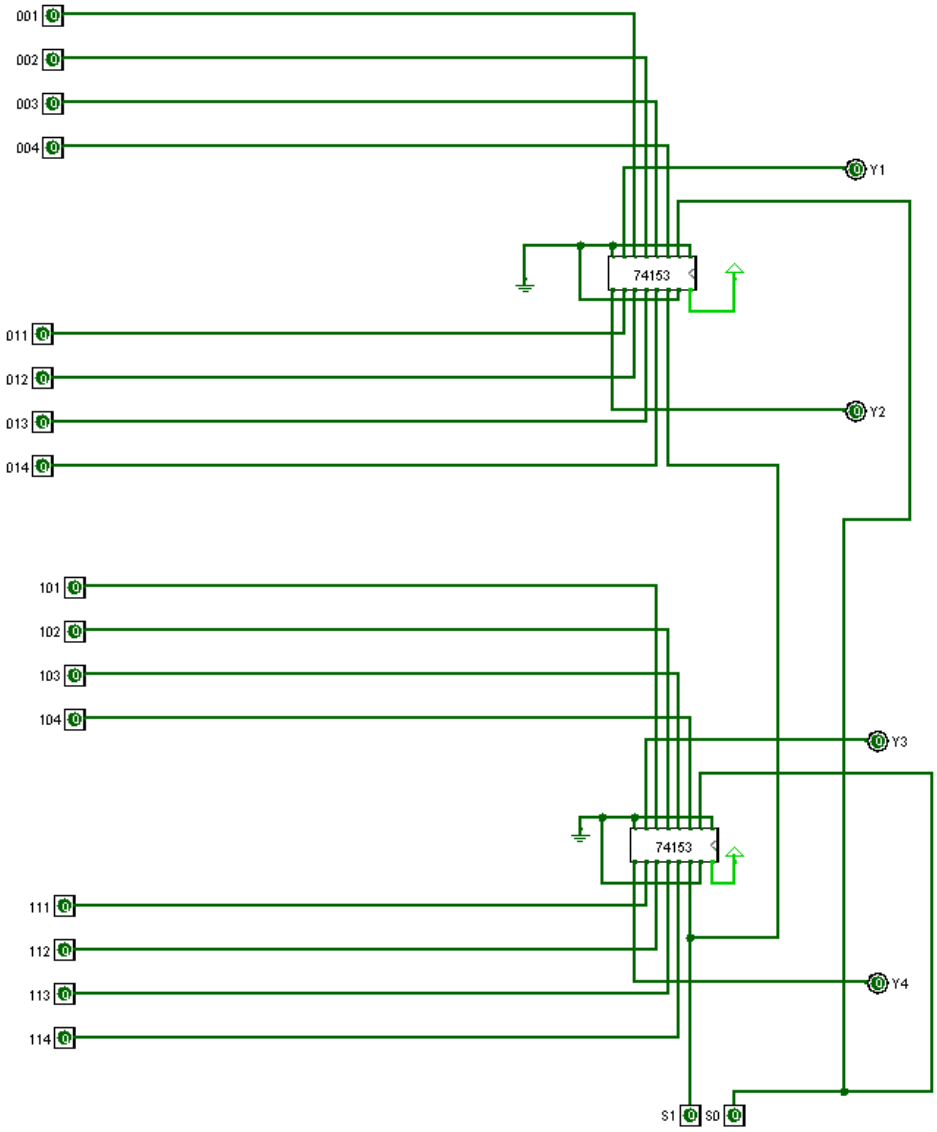


Figure 6: Custom multiplexer design used for selecting X and Y inputs in the ALU

7 IC Count

No.	IC NUMBER	IC NAME	COUNT
1.	7404	Hex inverter	2
2.	7408	Quad 2-Input AND Gate	2
3.	7432	Quad 2-Input OR Gate	2
4.	7486	Quad 2-Input XOR Gate	1
5.	7483	4-BIT Binary Full Adder	1
6.	74153	Dual 4-Line to 1-Line Data Multiplexer	4

Table 3: ICs used with count

8 The Simulator Used along with the Version Number

logisim-generic-2.7.1

9 Discussion

We used 12 ICs in our final ALU design, which was achieved after optimizing and reducing the number of gates in our initial attempts. Initially, we separated the arithmetic and logic units, which resulted in a higher IC count of around 14-15. To improve this, we explored a MUX-based approach, starting with 2x1 MUXs (IC 74157). However, this design still required around 14 ICs due to the complex selection inputs needed for the diverse operations of the ALU, particularly requiring XOR gates (IC 7486). We then shifted to 4x1 MUXs (IC 74153), which simplified the selection bit equations, reducing gate complexity and bringing the IC count down to 12, optimizing the overall design.

After confirming the ALU software was correct, we began the hardware design. We sourced most of our hardware and ICs from our seniors and the lab. To ensure smooth progress, we tested each IC in small circuits beforehand, identifying faulty ones early. Some problems we faced during implementation:

- In our simulation software, we encountered an issue with the adder. In Logisim, the adder IC's C_{out} (carry-out) bit had an input terminal, which caused complications. To resolve this, we designed our own adder circuit in the software, bypassing the flaw and ensuring proper functionality.
- One issue we encountered was that the ICs required a DC current, which is typically supplied by batteries. However, many batteries available in the market have a higher voltage (9V), which can damage sensitive components. In fact, we ended up destroying about 10-20 LEDs initially. To prevent further damage, we switched to using an adapter with a Power Supply Unit (PSU) to ensure a constant 3V DC flow for safe operation

- To address noise issues in some ICs, such as the XOR gate, which are particularly sensitive to noise and can mistakenly register it as a valid input, we grounded one pole of our DPDT switches. This helped stabilize the signal and prevent false triggering in the circuit.

- To create the overflow flag (V), we needed both the C_3 and C_{out} from our adder circuit. However, due to the physical design of the adder, we couldn't access C_3 directly. To work around this, we manually calculated C_3 using the inputs with the following equation:

$$C_3 = A_3 \oplus B_3 \oplus S_3 \tag{1}$$

where A_3 and B_3 are the MSB of our 4 bit inputs and S_3 is the adder output.

$$S_3 = A_3 \oplus B_3 \oplus C_3$$

putting S_3 in R.H.S of equation (1) :

$$A_3 \oplus B_3 \oplus (A_3 \oplus B_3 \oplus C_3) = (A_3 \oplus A_3) \oplus (B_3 \oplus B_3) \oplus C_3 = 0 \oplus C_3 = C_3$$

After obtaining C_3, we created our overflow flag (V) using :

$$V = C_3 \oplus C_{out}$$

In conclusion, our journey through the software and hardware evaluation has been immensely rewarding. We learned a great deal about the importance of proper team collaboration and efficient design practices. While we successfully navigated many challenges, we also recognize a small regret regarding the messy wire connections that could have been better organized. Overall, this experience has equipped us with valuable insights and skills for future projects.

10 Contribution of Each Member

1. **Software Implementation:**
    - Circuit Design: 2105031, 2105032, 2105033, 2105035, 2105048
    - Simulation and implementation : 2105032, 2105033
    - Verification and Testing : 2105031, 2105035, 2105048
2. **Hardware Implementation:** 2105031, 2105032, 2105033, 2105035, 2105048
3. **Report Writting:**
    - 2105031 : Section 1 , 2
    - 2105032 : Section 5
    - 2105033 : Section 6, 9, 10
    - 2105035 : Section 4, 7
    - 2105048 : Section 3, 8