CSE 210

Computer Architecture Sessional

Assignment-1

Arithmetic Logic Unit(ALU)

Group Information: **Section:** C1 **Group:** 06

Members of the Group:

- 2105128 - Nakib Arman Arzon
- 2105129 - Sarowar Alam Roki
- 2105138 - Dip Saha
- 2105142 -Md Mehedi Hasan Mim
- 2105145 - Ahmed Sajid Hasan

October 19, 2024

1 Introduction

An Arithmetic Logic Unit (ALU) is a critical component of the CPU in a computer. It serves as the processor's core, performing essential arithmetic and logical operations needed for various functions. The ALU executes tasks such as addition, subtraction, multiplication, and division, as well as logical operations like AND, OR, and NOT.

Operating with binary data (0s and 1s), the ALU enables computations and decision-making processes that are vital to a computer's performance. Its integration into the CPU allows for efficient data processing, complex calculations, and the execution of instructions. The speed and efficiency of the ALU play a significant role in determining a computer's overall performance.

Structurally, the ALU consists of two main sections: the Arithmetic Unit and the Logic Unit. It handles tasks like addition, subtraction, incrementing, and decrementing, along with logical operations such as AND, OR, NOT, and XOR. Specific operations are selected using control lines. In a typical ALU, like the one described with three control selection inputs, arithmetic operations are performed through parallel adders, while logical functions are managed by specialized Integrated Circuits (ICs).

In this example of an ALU, it processes 4-bit inputs and generates a 4-bit output. Additionally, it includes four status flags: Carry (C), Zero (Z), Overflow (V), and Sign (S). These flags provide important information about the results of operations, such as whether a carry-out occurred or if the result is zero. Their representations carry out the following meanings:

1. **CF:** CF is the C_{out} of the adder output in the ALU. So, any carry out results in the flag being 1, otherwise it is found to be 0. In Logical operations, bitwise operations do not cause it to be 0.

2. **ZF:** If the last operation of the ALU yielded an output of 0, ZF is set to 1, otherwise it is 0.

3. **VF:** If after any arithmetic operation, if the result exceeds the range of the used bits, an overflow occurs, that is an overflow and VF is set. After any logical operation, it is 0. From arithmetic speculation, V is found to be
$$V = C_3 \oplus C_{out}$$
Here, C_3 is the carry generated during the third bit addition and C_{out} is the output carry of the adder.

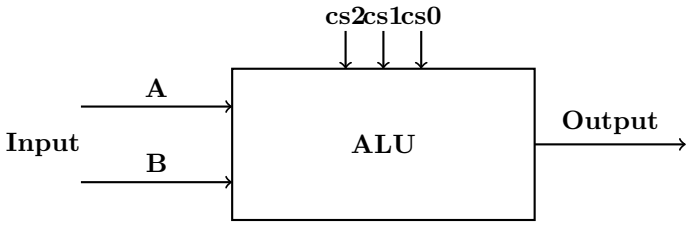
4. **SF:** It reflects the output MSB. For signed arithmetic operation, the sign of the result can be determined from this.

2 Problem Specification with Assigned Instructions

Design a 4-bit ALU with three control selection bits cs_0, cs_1, cs_2. The assigned instructions and corresponding combinations of control selection bits are shown in the table below:

Control Signals			Function	
cs_2	cs_1	cs_0	Operation	Description
0	0	0	Add with carry	$A+B+1$
0	0	1	Subtract	$A-B$
0	1	0	Transfer A	A
0	1	1	XOR	$A \oplus B$
1	X	0	Increment A	$A+1$
1	X	1	Decrement A	$A-1$

Table 1: Problem Specification



4-bit ALU

3 Detailed design steps with k-maps

3.1 Design Steps

1. Here we have to perform **five** types of **arithmetic operations** (Add with carry, Sub, Transfer A, Increment A, Decrement A) and only **one** type of **logical operations**(logical XOR).

2. For the adder,the first input X_i is to be fixed A_i, except for one case. When $c_{s_2}c_{s_1}c_{s_0} = 011$, $A_i \oplus B_i$ is selected as a input of X_i. We used a 2×1 multiplexer for each bit to select from the two options. The MUX has S_1 as selection bit.The selection bit is controlled by control bits(c_{s_2},c_{s_1},c_{s_0}).The details is shown in the truth table and k-Map.

3. For the adder,the possible values of second input Y_i are B_i, $\bar{B}_i$, 0 and 1. We used a 2×1 multiplexer for each bit to choose among these options. For this,The mux has S_2 as its selection bit and en_2 as its enable bit.The selection bit and enable bit are controlled by control bits(c_{s_2},c_{s_1},c_{s_0}).The details in shown in the truth table and k-Map.

4. The input carry(C_{in}) of the adder IC is either 0 or 1.It should be 1 for add with carry, for sub and for increment A functions.This is also controlled by control bits(c_{s_2},c_{s_1},c_{s_0}).The details is shown in the truth table and k-Map.

5. Transfer A could have been operated in multiple ways.But both Y_i and C_{in} were chosen to be 0 because of an easier design.

6. During logical operations,X_i is transferred as Y_i=0 and C_{in}=0.

7. Zero flag, ZF is computed by adding the 4 output bits using 3 OR gates and then inverting $S_0 + S_1 + S_2 + S_3$ by 1 NOT gate.

8. The carry flag(C) is directly obtained from C_{out} of the parallel adder

9. The sign flag(SF) is obtained from the MSB of the sum(S_3).

10. To implement the flags, it is observed that the overflow flag requires previous carry from the adder which can not be directly found from a adder IC. So the adder is split in to 2 parts where one adder adds only the 3 bits and another adder only adds the MSB's. Dividing the adder into two parts it only a design choice. The carry generated in the first adder is used as the carry in for the second adder. The overflow flag is the determined from the $C_3 \oplus C_{out}$.

3.2 K-maps

We will be following table 2 to construct the K-maps for selection bits, enable bit of multiplexers and C_{in}.

The IC of parallel adder takes X_i and Y_i and C_{in} as input.We need X_i as A_i or $A_i \oplus B_i$ for logical changes. These values are received as X_i(output of the 2 to 1 multiplexer) and the selection bits for the multiplexer are S_1. For Y_i we want 0, 1, B_i and its compliment as the output of 2 to 1 multiplexer. The selection and enable bit of this(2 to 1 MUX) multiplexer are S_2 and en_2 respectively. The k-map for the selection bits, enable bit of the multiplexer and carry input C_{in} are as follows:

3.2.1 K-map for S_1

S_1 is the selection bit for the multiplexer that selects the first input X_i of the adder in the arithmetic unit.

C_{S_2} \ $C_{S_1}C_{S_0}$	00	01	11	10
0	0	0	1	0
1	0	0	0	0

We can easily express S_{11} as following:

$$S_1 = \overline{C_{S_2}} C_{S_1} C_{S_0}$$

3.2.2 K-map for S_2

S_2 is the selection bit for the multiplexer that selects B, $\bar{B}$ and 1 as input of a 2 to 1 MUX . B and $\bar{B}$ were represented in XOR with C_{S_0}. To get 0, the enable was set accordingly.

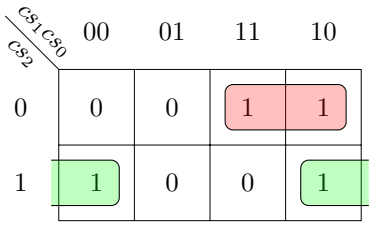
C_{S_2} \ $C_{S_1}C_{S_0}$	00	01	11	10
0	0	0	0	0
1	0	1	1	0

We can easily express S_2 as following:

$$S_2 = C_{S_2} C_{S_0}$$

3.2.3 K-map for en_2

en_2 is the enable bit for the 2 to 1 multiplexer that decides when the multiplexer will be functioning(will select one of its inputs) or not(in ground state).

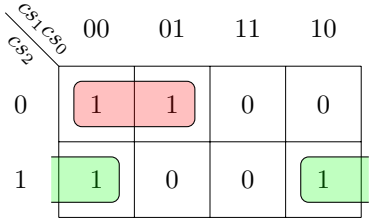


The simplified form for en_2 will be:

$$en_2 = \overline{C_{S_2}} C_{S_1} + C_{S_2} \overline{C_{S_0}}$$

3.2.4 K-map for C_{in}

It is the input carry bit of the adder used inside arithmetic unit.



The simplified form for C_{in} will be:

$$C_{in} = \overline{C_{S_2}}\,\overline{C_{S_1}} + C_{S_2} \overline{C_{S_0}}$$

4 Truth Table

cs2	**cs1**	**cs0**	**Function**	X_i	S_1	en_1	Y_i	S_2	en_2	C_{in}
0	0	0	Add with carry	A_i	0	0	B_i	0	0	1
0	0	1	Sub	A_i	0	0	$\overline{B_i}$	0	0	1
0	1	0	Transfer A	A_i	0	0	0	0	1	0
0	1	1	XOR	$A_i \oplus B_i$	1	0	0	0	1	0
1	X	0	Increment A	A_i	0	0	0	0	1	1
1	X	1	Decrement A	A_i	0	0	1	1	0	0

Table 2: Truth Table for Intermediate I/O

5 Block Diagram

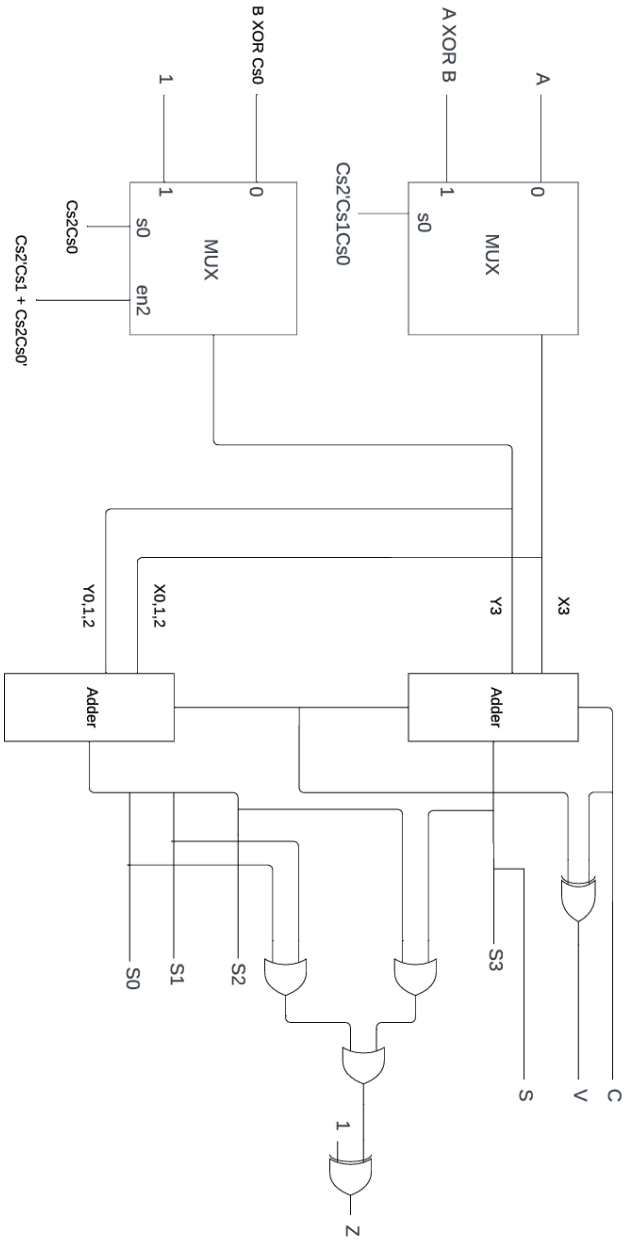


Figure 1: Block Diagram of ALU

6 Circuit Diagram

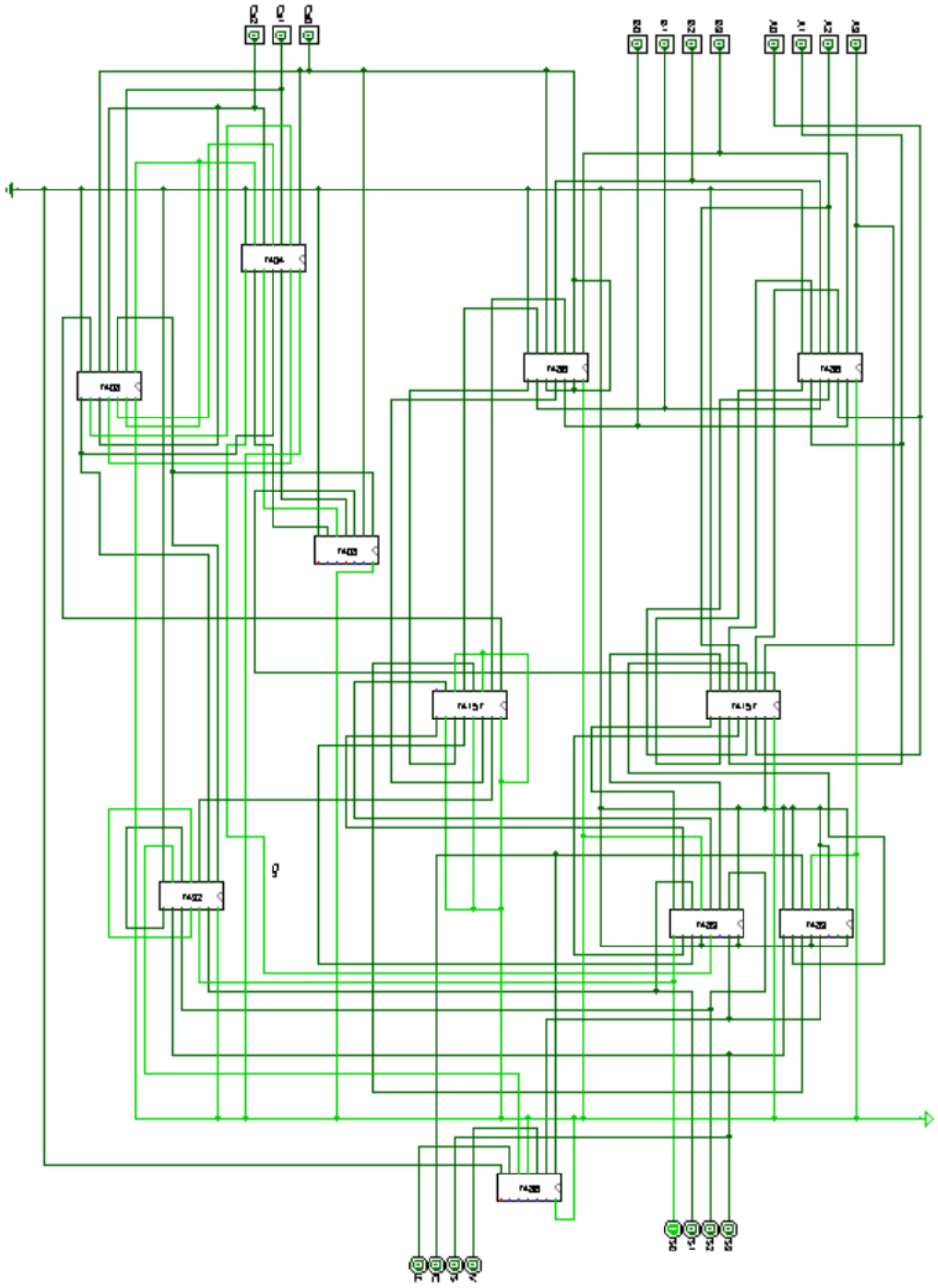


Figure 2: Complete Circuit Diagram of ALU

7 ICs Used with Count as a Chart

IC	Quantity
7404	1
7408	2
7432	1
7483	2
7486	3
74157	2
Total	11

Table 3: ICs used with quantity

8 The Simulator Used along with the Version Number

Logisim - 2.7.1

9 Discussions

In this assignment, we were tasked with creating a 4-bit Arithmetic Logic Unit (ALU) that performs five arithmetic and one logical operations using as few Integrated Circuits (ICs) as possible. Our approach included using an IC7486 (XOR) for the logical XOR operations so that IC counts do not increase and also two IC7483 adders to form four single-bit Full Adders in parallel. These ideas decreased the overall IC count by a lot. We first implemented the minimized design idea onto Logisim, and checked with numerous test cases to confirm that the design was perfect. Later, we started to implement this design on hardware.

After assembling the circuit, we faced a significant hurdle and struggled a lot when it did not work as anticipated, leading to disappointment among our team members. However, a detailed examination revealed that some components were improperly grounded. Correcting this grounding issue was crucial and once addressed, the circuit functioned perfectly.

The successful completion of our 4-bit Arithmetic Logic Unit (ALU) not only demonstrated our capability to implement complex designs with minimal resources but also highlighted the critical importance of teamwork and iterative testing in engineering projects. The issues we encountered, such as the grounding problem, served as a stark reminder that theoretical knowledge must be complemented with practical implementation.

This project also underscored the significance of a systematic approach to troubleshooting in electronic design. Initially, the failure of the circuit to perform as expected was a discouraging setback. Yet, this challenge prompted us to methodically check each component's connections and functionality, which was an invaluable exercise in patience and attention to detail. This systematic checking might become a standard procedure in our future projects, ensuring that we maintain high standards of quality and reliability.

Moreover, the necessity to minimize the use of Integrated Circuits (ICs) in our design taught us to be innovative and to optimize our solutions. This constraint led us to a deeper understanding of each IC's capabilities and limitations, which is crucial when scaling up designs or working under resource constraints in professional settings.

Reflecting on the teamwork aspect, the project strengthened our collaboration skills. The reso- lution of the grounding issue was not just a technical victory but a testament to our team's commu- nication and collective problem-solving abilities. Each member's unique perspective and expertise contributed to diagnosing and fixing the issue, reinforcing the idea that diverse teams can lead to more effective solutions.

10 Contribution of Each Member

2105128:

Initial Design making,Truth Table design, K-map for Simplification,Hard- ware Implementation,Software Testing,Logisim Design,Debugging,Report Writing(Section 5,6)

2105129:

Initial Design making,Hardware collection, Hardware Implementation, Debugging,Report Writing(Section 1,2)

2105138:

Initial Design making, Design Minimization (MUX),K-map for Simplification,Hardware collection, Hardware Implementation, Debugging,Report Writing(Section 3,4)

2105142:

Initial Design making,Hardware collection, Hardware Implementation, Debugging ,Hardware Collection,Report Writing(Section 7,8)

2105145:

Software Testing ,Logisim Design , Truth Table Design , Hardware Implementattion,Report Writing (Section 9,10)