CSE 210
Computer Architecture Sessional

Assignment-1: 4-bit ALU Simulation

Lab Section - C2
Group - 02

15 September, 2024

Members of the Group:

i. 2105152 - Sudip Kumar Saha

ii. 2105161 - Sadia Naushin

iii. 2105163 - Dibbo Chowdhury

iv. 2105168 - Nusrat Jahan Nidhi

v. 2105174 - Nujhat Sadia

1 Introduction

The Arithmetic Logic Unit (ALU) serves as the computational core of a computer's central processing unit (CPU). Responsible for executing arithmetic and logic operations, the ALU performs essential tasks such as addition, subtraction,bitwise logical operations, negation and many more. Its speed and versatility make it a vital component in processing instructions efficiently. ALUs come in various architectures, adapting to different computational demands across applications. To understand modern computing devices, knowing about the ALU – its role and design – is vital.

The ALU we designed consisted of three control signals, through which we facilitated six operations: subtraction, transfer, add with carry, increment, bitwise AND, decrement. The arithmetic unit was responsible for managing the arithmetic operations, while our logical unit executed the rest. The control signals supervised the entire ALU operations.

We also incorporated 4 status flags , Carry(C), Sign(S), Overflow(V) and Zero(Z) flags, in our design. These flags followed the rules of Assembly Language with some flexibility. They imply :

a) **C** :- C or Carry Flag is the carry out C_{out} of the adder in the ALU. It depends on the carry output of arithmetic operations. It is reset during logical operations.

b) **S** :- S or the Sign bit is the MSB of the output, O_3.

c) **V** :- V is the overflow flag. This flag is only meaningful for signed numbers. If the output of an n-bit ALU exceeds the range of -2^{n-1} to $2^{n-1} - 1$ then an overflow has occured. V is set when the addition of two positive numbers gives negative output and vice-versa. It is always reset during logical operations.

$$O_3 = X_3 \oplus Y_3 \oplus C_3$$

$$\Rightarrow O_3 \oplus C_3 \oplus O_3 = X_3 \oplus Y_3 \oplus C_3 \oplus C_3 \oplus O_3$$

$$C_3 = X_3 \oplus Y_3 \oplus O_3 \tag{1}$$

$$V = C_3 \oplus C_{\text{out}} \tag{2}$$

$$\therefore V = X_3 \oplus Y_3 \oplus O_3 \oplus C_{out}$$

where C_{out} is the output carry, O_3 is the MSB of the output, X_3 and Y_3 are the MSB of the first and second input of the adder respectively.

d) **Z** :- Z is the Zero flag, which is set when the output of the ALU is zero and reset otherwise.

$$Z = \overline{O_3 \vee O_2 \vee O_1 \vee O_0}$$

2 Problem Specification with Assigned Instructions

Design a 4-bit ALU with three selection bits cs_2 , cs_1 and cs_0 for performing the following operations:

Control Signals			**Description**	**Function**
cs_2	cs_1	cs_0		
X	0	0	Sub	$A + \overline{B} + 1$
X	0	1	Transfer A	A
0	1	0	Add with carry	$A + B + 1$
0	1	1	Increment A	$A + 1$
1	1	0	AND	A $\wedge$ B
1	1	1	Decrement A	$A - 1$

Table 1: Problem Specification

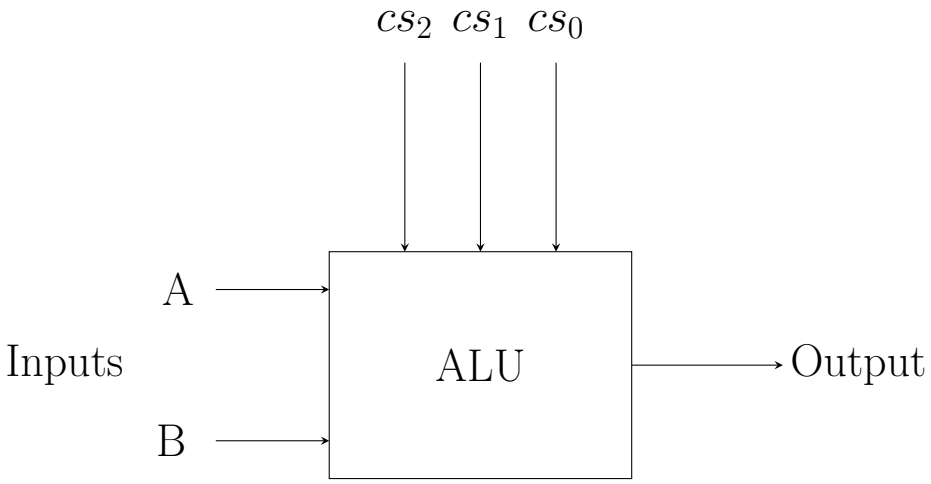

Figure 1: 4-bit ALU

3 Detailed Design Steps with K-maps

3.1 Design Steps

i) Our design comprises 4 parts: arithmetic unit, logical unit, selection control, and status flags.

ii) The arithmetic unit computes 5 arithmetic operations (subtract, transfer A, add with carry, increment A, decrement A) with the help of a 4-bit parallel adder, basic gates and two multiplexer.

iii) Here we have modified the input into the adder in such a way so that we don't need to modify the output after getting the output from adder.

iv) From the truth table 3, we see that the X_i can be either A or $A \wedge B$. MUX:3 is just doing that.

v) For Y_i as there are four possible values, we constructed another truth table with two **Intermediate Selection Bits** namely S_1, S_2 in Table 2. And as we are using 2 to 1 MUX for MUX:3, so we stuck with 2 to 1 MUX for the rest of the design. Using 4 to 1 MUX would not have lessened the amount of ICs.

S_2	S_1	Y_i
0	0	0
0	1	1
1	0	B
1	1	$\overline{B}$

Table 2: Determining Y_i

vi) Now the logical unit is comprised of a 4-bit 2X1 multiplexer with S_0 selection bit. Logical operation AB is selected when S_0 is 1 and A is selected when S_0 is 0. The only logical operation occurs when the combination of C_{S2}, C_{S1} and C_{S0} is 110. So the value of S_0 will be 1 when $S_0 = C_{S2}C_{S1}\overline{C_{S0}}$

vii) For calculating the flags, we used the formula described in section:1.

3.2 Truth Table

C_{S2}	C_{S1}	C_{S0}	X_i	Y_i	Z	S_2	S_1	S_3	Output
X	0	0	A_i	$\overline{B_i}$	1	1	1	0	$A-B$
X	0	1	A_i	0	0	0	0	0	A
0	1	0	A_i	B_i	1	1	0	0	$A+B+1$
0	1	1	A_i	0	1	0	0	0	$A+1$
1	1	0	A_iB_i	0	0	0	0	1	AB
1	1	1	A_i	1	0	0	1	0	$A-1$

Table 3: Truth Table for S_1, S_2 selection bits and carry input of the adder

3.3 K-maps

3.3.1 K-map for S_1

S_1 is the selection bit with input B and $\overline{B}$ and it's output goes to the multiplexer with S_2

C_{S2} \ C_{S1},C_{S0}	00	01	11	10
0	1	0	0	0
1	1	0	1	0

From the K-map: $S_1 = \overline{C_{S1}}\,\overline{C_{S_0}} + C_{S2}C_{S_1}C_{S_0}$

3.3.2 K-map for S_2

S_2 is the selection bit with the inputs: S_1 , Output of S_1. The output of this multiplexer determines Y_i.

C_{S2} \ C_{S1},C_{S0}	00	01	11	10
0	1	0	0	1
1	1	0	0	0

From the K-map: $S_2 = \overline{C_{S1}}\,\overline{C_{S_0}} + \overline{C_{S2}}\,\overline{C_{S_0}}$

3.3.3 K-map for Z

Z is the carry input of our adder.

C_{S2} \ C_{S1}, C_{S0}	00	01	11	10
0	1	0	1	1
1	1	0	0	0

From the K-map: $Z = \overline{C_{S1}}\,\overline{C_{S_0}} + \overline{C_{S2}}C_{S_1}$

4 Block Diagram

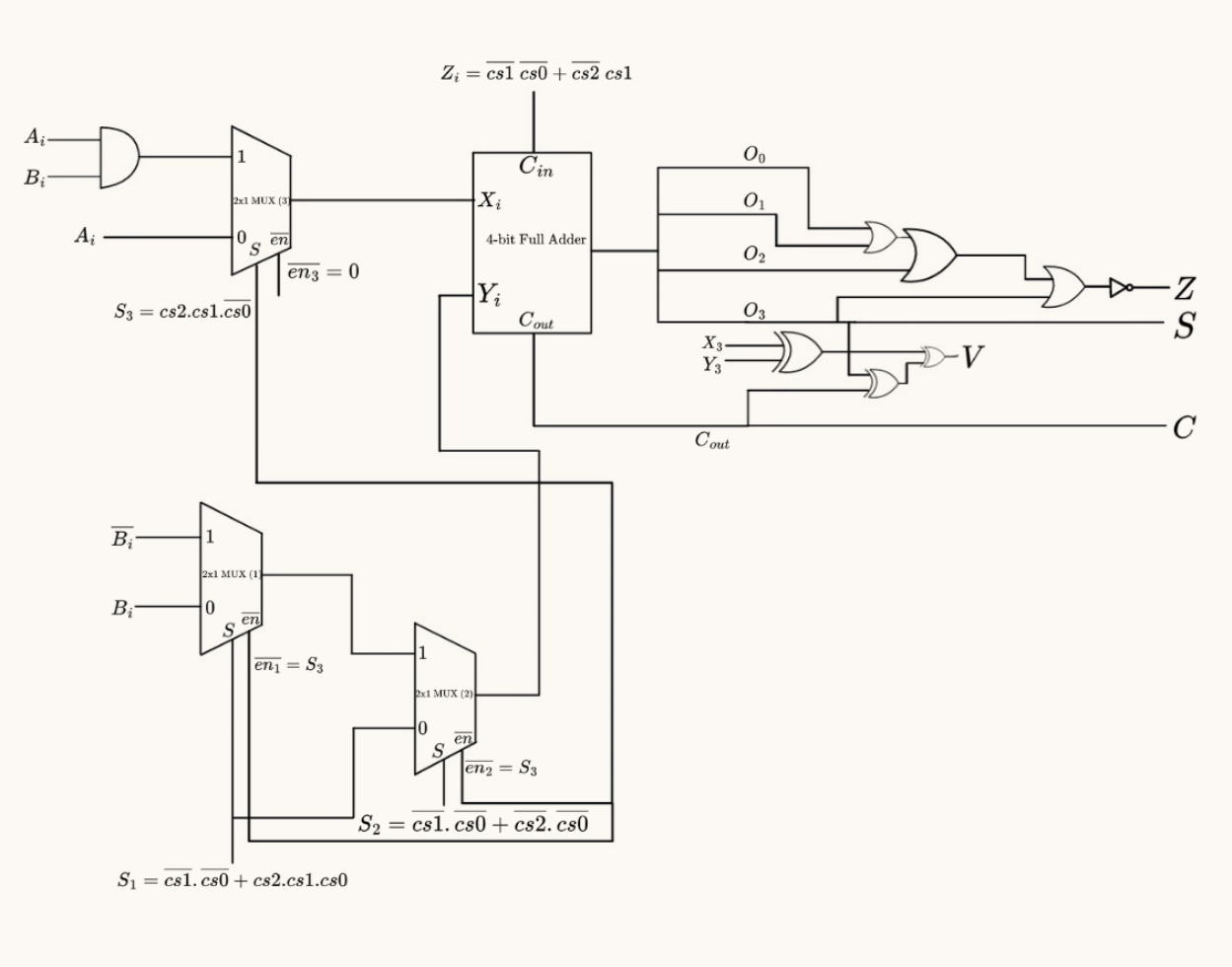

Figure 2: Block Diagram of ALU

5 Complete Circuit diagram

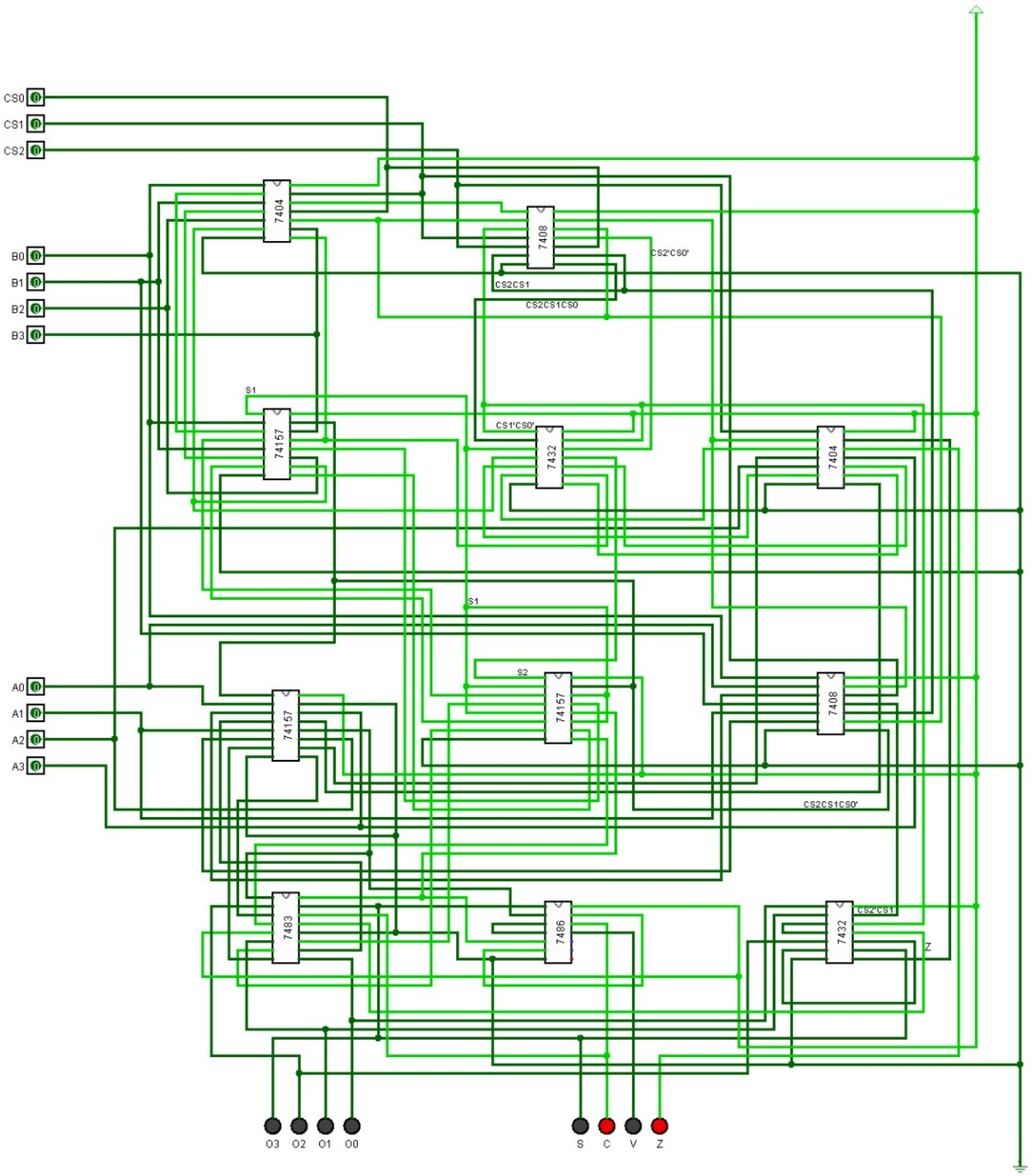

Figure 3: Complete ALU circuit

6 TinkerCAD Design

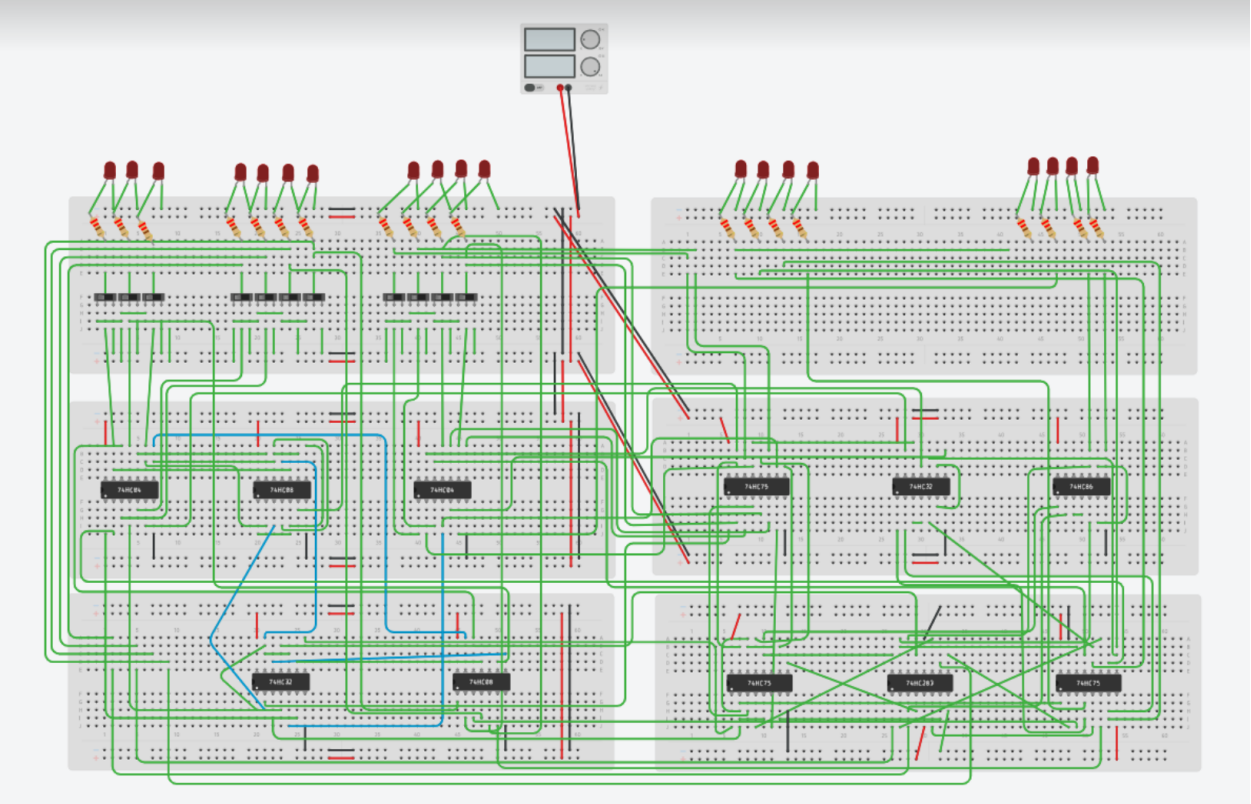

Figure 4: TinkerCAD design for hardware implementation

7 ICs used with count as a chart

IC	Number of ICs
7483	1
74157	3
7432	2
7486	1
7404	2
7408	2
Total	11

Table 4: ICs and their number

8 The Simulator Used along with the Version Number

Logisim - generic - 2.7.1

9 Discussions

In this 4-bit ALU design, we have efficiently combined both arithmetic and logical operations using basic gates, multiplexers, and a parallel adder.

We started off by making the truth table and finding equations. But in this traditional way in which we tried to find the values of X_i and Y_i separately, we got huge equations and led to extensive use of ICs. So we had to optimize the situaation by using MUX. We narrowed the number of ICs down to 11.

Some difficulties did arise while software simulation using **Logisim**. Changes had to be made in the 7400-lib.circ. The C_{out} pin of the IC7483 in the library was faulty. This problem was corrected using a seperate IC7483.circ file in our final ALU software simulation.

Finally, we addressed challenges in the hardware implementation by correcting issues with LEDs, where the use of different types initially caused a power drop. Overheating problems due to connecting LEDs without resistances were resolved through the implementation of proper corrections. The challenge of maintaining a stable power flow across six breadboards was successfully addressed through multiple correction attempts. The flags are calculated based on the output and control signals, ensuring accurate status reporting for operations. This design approach balances functionality and simplicity, providing a solution for basic arithmetic and logic tasks in a compact 4-bit system.

10 Contribution of Each Member

- 2105152 : Sudip Kumar Saha
  - Logisim circuit design
  - Software simulation of ALU circuit
  - Hardware
  - Report writing

- 2105161 - Sadia Naushin
  - Software simulation of ALU circuit
  - Hardware
  - Report writing

- 2105163 - Dibbo Chowdhury
  - Circuit design and optimization
  - TinkerCAD design
  - Hardware
  - Report writing

- 2105168 - Nusrat Jahan Nidhi
  - Hardware
  - Report writing

- 2105174 - Nujhat Sadia
  - Circuit design and optimization
  - Software simulation of ALU circuit
  - Hardware implementation
  - Report writing