CSE 210
(Computer Architecture Sessional)

4-bit ALU Simulation

Samiul Hasan Nihal

2105105

Department of Computer Science

`Bangladesh University of Engineering and Technology`

1 Introduction

An Arithmetic and Logic Unit (ALU) is the core computational component of a computer, often referred to as the mathematical brain of digital systems. It is a combinational digital circuit designed to perform arithmetic operations (such as addition and subtraction) and logical operations (such as AND, OR, and XOR) on binary data. The ALU is an essential building block found in almost every processing unit, including Central Processing Units (CPUs) and Graphics Processing Units (GPUs), playing a critical role in executing instructions and handling data manipulation tasks.
As its name implies, the ALU is divided into two main parts: the Arithmetic Unit and the Logic Unit. The Arithmetic Unit performs operations such as addition, subtraction, incrementing, and decrementing, while the Logic Unit handles operations like bitwise NOT, AND, OR, and XOR. The selection of operations is controlled by a set of control lines or bits, which determine the specific operation to be executed by the ALU. In many designs, k control bits allow the ALU to support 2^k different operations. For example, the ALU we've implemented uses three control selection inputs, allowing it to perform up to eight distinct operations. Additionally, our ALU includes four status outputs, also known as flags: the Carry Flag (C), Zero Flag (Z), Overflow Flag (V), and Sign Flag (S). These flags provide important information about the result of an operation:

- **C (Carry Flag)**: Indicates whether a carry-out has occurred after an arithmetic operation. If there is a carry, the flag is set to 1; otherwise, it remains 0.
- **Z (Zero Flag)**: Set to 1 if the ALU operation results in a zero output, indicating that the result of the operation was zero.
- **V (Overflow Flag)**: Set when an arithmetic overflow occurs, such as when adding two positive numbers results in a negative number or adding two negative numbers results in a positive number. After any logical operation, it is cleared.
- **S (Sign Flag)**: Reflects the output Most Significant Bit (MSB).

$$S_3 = A_3 \oplus B_3 \oplus C_3 \tag{1}$$
$$C_3 = A_2B_2 + (A_2 \oplus B_2)C_2 \tag{2}$$

Combining 1 and 2,

$$V = C_3 \oplus C_{out} \tag{3}$$

Here, S_3 is the MSB of the adder output and C_{out} is the output carry of the adder.

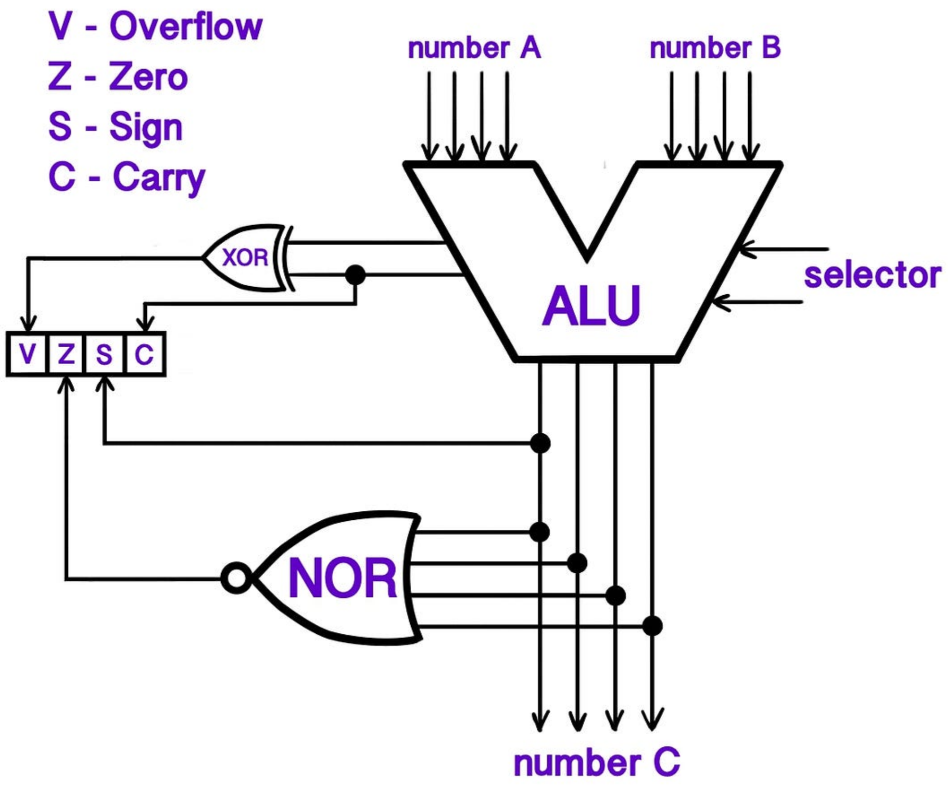

Figure 1: ALU Status Flags

2 Problem Specification with Assigned Instructions

Design a 4-bit ALU with three selection bits cs0, cs1 and cs2 for performing the following operations:

Control Signals			**Functions**	**Description**
cs2	**cs1**	**cs0**		
x	0	0	Add	$A + B$
x	0	1	Add With Carry	$A + B + 1$
0	1	0	Sub With Borrow	$A + \bar{B}$
0	1	1	XOR	$A \oplus B$
1	1	0	Complement$A$	$\bar{A}$
1	1	1	Increment$A$	$A + 1$

Table 1: Problem Specification

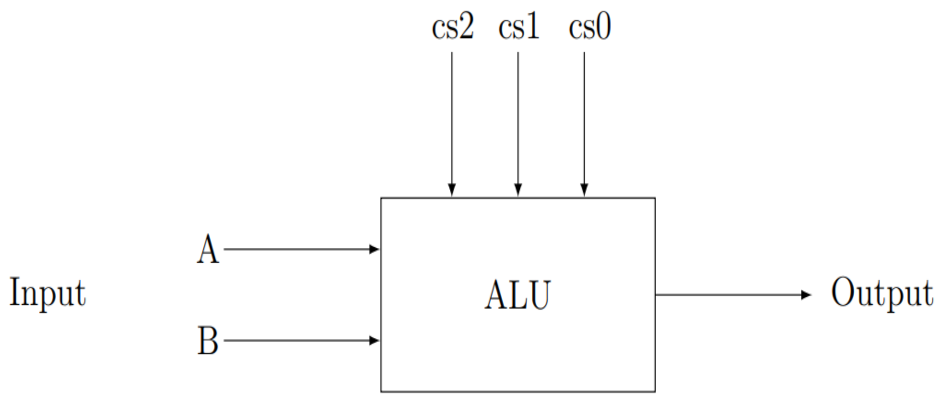

Figure 2: 4-bit ALU

3 Detailed Design Steps with K-maps

3.1 Design Steps

1. The Arithmetic Unit performs four arithmetic operations: Addition, Addition with Carry, Subtraction with Borrow, and Increment A. It also executes two logical operations: XOR and Complement of A. This is achieved using a 4-bit full adder (7483 IC), a multiplexer (74157 IC), a hex inverter (7404 IC), a quad 2-input AND gate (7408 IC), and a quad 2-input OR gate (7432 IC).

2. To design the ALU, I first derived the expressions for X_i, Y_i, and Z_i required to implement the functions.

3. The variables X_i and Y_i serve as the inputs to the 4-bit full adder IC, while Z_i represents the C_{in} for the first 4-bit full adder IC. The expressions for X_i, Y_i, and Z_i were derived using Karnaugh maps, yielding the following:

$$X_i = A \oplus (\bar{cs}2cs1cs0)B \oplus (cs2cs1\bar{cs}0)1 \tag{4}$$

$$Y_i = B\bar{cs}1 + \bar{B}\bar{cs}2cs1\bar{cs}0 \tag{5}$$

$$Z_i = cs0(\bar{cs}1 + cs2) \tag{6}$$

4. In one quad 2-input multiplexer (74157 IC), the expression $\bar{cs}2cs1\bar{cs}0$ is utilized as the select input. If the select input is high, then the expression $(B\bar{cs}1 + \bar{B})$ is set; otherwise, $B\bar{cs}1$ is selected. The expression $(\bar{cs}2cs1cs0)$ is passed as the select input in the second quad 2-input multiplexer (74157 IC) to perform the XOR operation between A and B. When the select input is high, the operation performed will be $A \oplus B$; if low, it will simply pass A.

   After executing the operation in the second quad 2-input multiplexer (74157 IC), the outputs are then passed into the third quad 2-input multiplexer (74157 IC), where $(cs2cs1\bar{cs}0)$ serves as the select input. If the select input is high, it will perform XOR on the inputs received from the second 74157 IC; otherwise, the output will remain unchanged.

5. The variable Z_i is passed into the first 4-bit full adder IC (7483) as C_{in}. In this stage, I worked with the three bits of the inputs X_i and Y_i from the previous step, setting the fourth bit of the inputs to zero to compute C_3, which corresponds to S_4 of the first full adder. Subsequently, I used S_4 or C_3 as the C_{in} for the second full adder. Here, I worked with the last bit of both inputs, placing it at the zeroth position and setting all the remaining three bits to zero, which produced the actual C_{out} from S_1.

6. The zero flag, Z, is computed by summing the four output bits using three OR gates, followed by inverting $S_1 + S_2 + S_3 + S_4$ with an inverter IC (7404).

7. The overflow flag VF and carry flag CF are determined from the arithmetic unit, ensuring that there is no potential for overflow or carry, thus maintaining $V = 0$ and $C = 0$ for logical operations. The sign flag is indicated by S_4.

3.2 K-maps

We will be following Table **??** to construct the K-maps for intermediate selection bits:

3.2.1 K-map for X_i

	cs0	
	1	1
cs2cs1	1	1
	0	1
	1	1

Table 2: K-map for A

	cs0	
	0	0
cs2cs1	0	0
	1	0
	0	0

Table 3: K-map for $\bar{A}$

3.2.2 K-map for Y_i

	cs0	
	1	1
cs2cs1	0	0
	0	0
	1	1

Table 4: K-map for B

	cs0	
cs2cs1	0	0
	1	0
	0	0
	0	0

Table 5: K-map for $\bar{B}$

3.2.3 K-map for Z_i

	cs0	
cs2cs1	0	1
	0	0
	0	1
	0	1

Table 6: K-map for Z_i

4 Truth Table

For better interpretation of the variables used, refer to Figure 3

cs2	**cs1**	**cs0**	$\boldsymbol{Xi}$	$\boldsymbol{Yi}$	$\boldsymbol{Zi}$	**Function**
0	0	0	A	B	0	$A+B$
0	0	1	A	B	1	$A+B+1$
0	1	0	A	$\bar{B}$	0	$A+\bar{B}$
0	1	1	$A \oplus B$	0	0	$A \oplus B$
1	0	0	A	B	0	$A+B$
1	0	1	A	B	1	$A+B+1$
1	1	0	$\bar{A}$	0	0	$\bar{A}$
1	1	1	A	0	1	$A+1$

Table 7: Truth Table for Intermediate I/0

5 Block Diagram

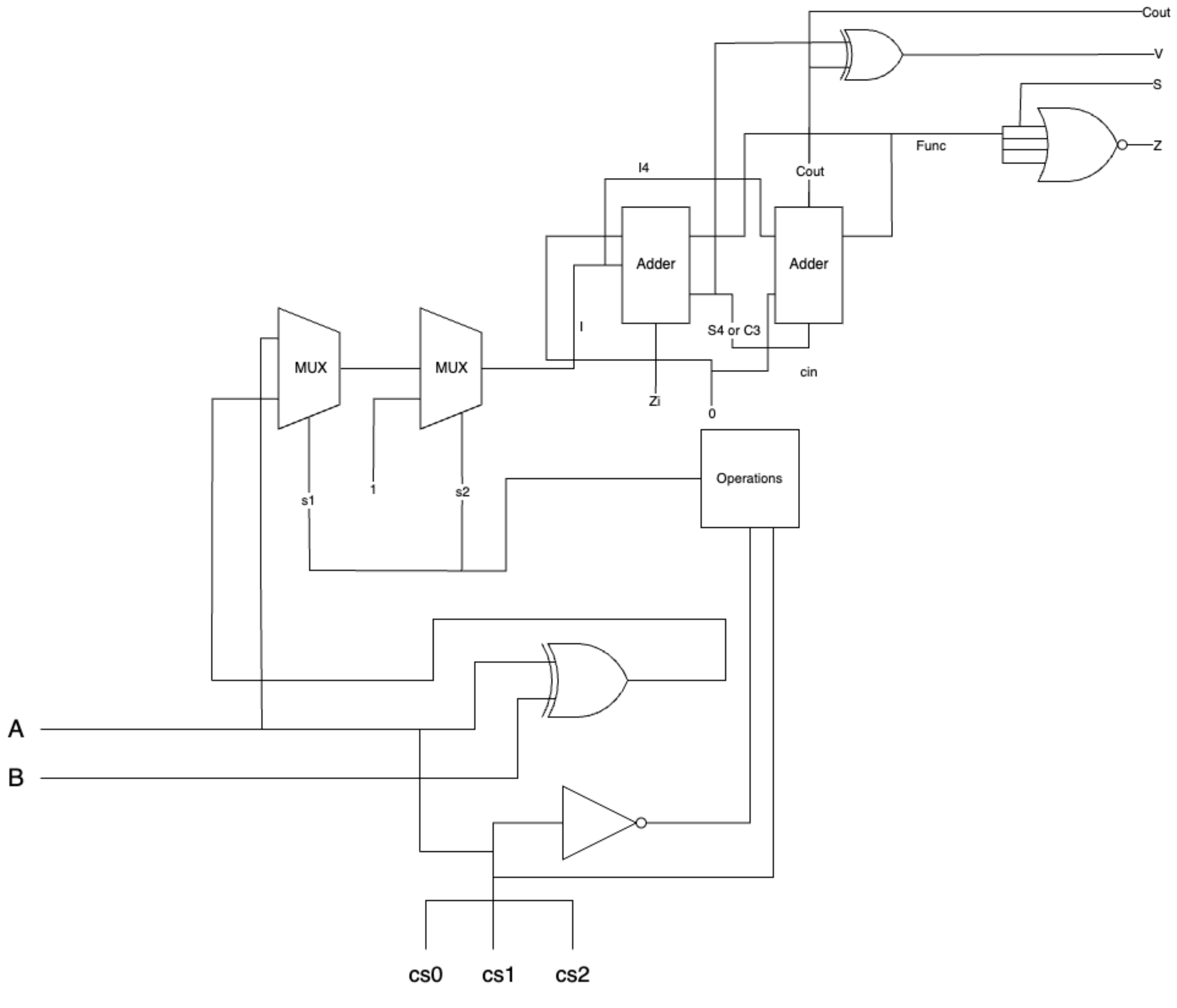

Figure 3: Block Diagram of ALU

6 Complete Circuit Diagram

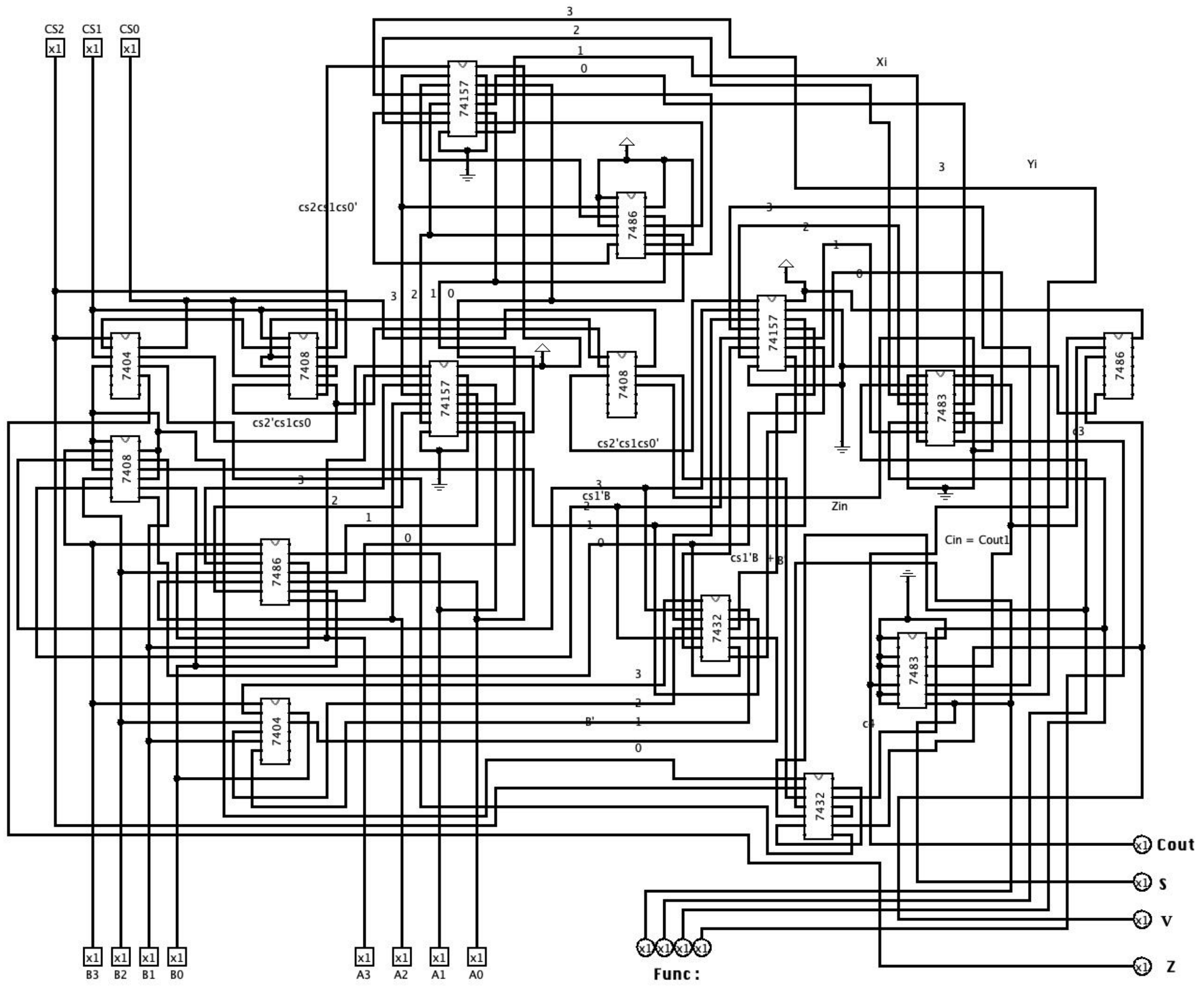

Figure 4: The Complete Design

7 ICs Used with Count as a Chart

ICs	Quantity
7404	2
7408	3
7432	2
7486	3
7483	2
74157	2
Total	14

Table 8: ICs Used with Quantity

8 The Simulator Used along with the Version Number

Logisim -Version 2.7.1 (10.2)

9 Discussion

The design and implementation of the Arithmetic Logic Unit (ALU) presented in this project was a result of individual effort, as I was not assigned to a group. This situation presented both challenges and opportunities. On one hand, working independently allowed me complete control over the design decisions and implementation process. On the other hand, it required me to address all technical and design challenges on my own, without the benefit of collaboration or shared workload. The ALU was designed to perform four arithmetic operations (addition, addition with carry, subtraction with borrow, and increment) and two logical operations (XOR and complement of A). The core of the design is built around a 4-bit full adder (7483 IC), along with multiplexers (74157 IC), hex inverters (7404 IC), and logic gates (7408 IC for AND, and 7432 IC for OR). By using a combination of hardware and logical control signals, the ALU achieves the desired functionality.
One of the key features of this ALU is its ability to perform arithmetic and logical operations on 4-bit inputs, while handling carry and borrow conditions efficiently. The expressions for X_i , Y_i , Z_i were derived using Karnaugh maps to ensure optimal minimization of logic gates, enhancing the efficiency and speed of the unit. This approach also allowed for straightforward control of operations through select signals (cs0, cs1, cs2), which were managed by the multiplexers.
Designing this ALU single-handedly provided me with a deep understanding of the underlying hardware architecture and logical decision-making process. From deriving the logical expressions using Karnaugh maps to assembling the components into a coherent unit, the entire process required meticulous attention to detail. I managed both the conceptual design and practical implementation, including testing the functionality of each operation independently to ensure correctness.
Despite the lack of group support, I was able to complete the design on schedule and ensure that the ALU met all functional requirements. This experience highlighted the importance of self-reliance, troubleshooting skills, and the ability to manage complex tasks independently.

The final ALU design is a testament to the efficiency and effectiveness of well-planned digital circuits. Through careful consideration of logical operations, efficient use of multiplexers, and optimization of carry and borrow handling, I was able to design an ALU that performs the required operations with minimal complexity. Working independently on this project allowed me to develop a deeper understanding of digital design and the satisfaction of solving complex problems on my own.