Bangladesh University of Engineering and Technology
Department of Computer Science and Engineering

CSE 306
Computer Architecture Sessional

Assignment-1: 4-bit ALU Simulation

Section - A1
Group - 03

Members of the Group:

i 2005003 - A.H.M. Towfique Mahmud

ii 2005004 - Md Zim Mim Siddiqee Sowdha

iii 2005006 - Kowshik Saha Kabya

iv 2005018 - Munzer Mahmood

v 2005019 - Jarin Tasneem

1 Introduction

ALU, the abbreviated form of Arithmetic and Logic Unit, can simply be called the mathematical brain of a computer. It is mainly a combinational circuit that can perform both arithmetic and logical operations on two given numbers. It is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPS, and graphics processing units (GPU).
The name clearly shows that it consists of two main units: arithmetic and logical units. These two units allow ALU to perform arithmetic operations such as increment, decrement, addition, subtraction, transfer, and negation, as well as logical operations like OR, XOR, AND, NOT, etc. The source information can be routed to ALU inputs with the help of a control unit. An ALU has some selection lines to help choose a particular operation. For k selection lines we have access to 2^k operations. In our implementation, there are 3 control selection inputs.
The main working unit of an ALU is the parallel adder. By controlling the input flow into it, we can have access to a variety of arithmetic and logical operations. Controlling the input flow to get our desired operation can be achieved by using a simple multiplexer that chooses a particular operation based on the combination of selection lines.
We have also implemented four status flags or outputs. They are called C (Carry Flag), Z (Zero Flag), V (Overflow Flag), and S (Sign Flag). Their respective definitions are as follows:

C: C is simply the C_{out} of the adder used in the ALU. So in any arithmetic operations, any carry-out being 1 sets this flag. For logical operations, its value is irrelevant.

Z: If the last operation of the ALU yields an output of 0, Z is set, otherwise it is 0.

V: In any arithmetic operations, if two positive numbers yield a negative output or two negative numbers yield a positive output, this flag is then set to 1. It is cleared after any logical operations. Its equation is shown below:

$$\begin{aligned} S_3 &= A_3 \oplus Y_3 \oplus C_3 \\ \implies C_3 &= S_3 \oplus A_3 \oplus Y_3 \end{aligned} \tag{1}$$

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

Combining equations 1 and 2,

$$V = A_3 \oplus Y_3 \oplus S_3 \oplus C_{out} \tag{3}$$

Here, Y_3 is either B_3 from input (MSB of B), or anything coming from the multiplexer based on selected operation, S_3 is the MSB of the adder output and C_{out} is the output carry of the adder.

S: It is the output MSB. It shows the signed bit.

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	Output
$cs2$	$cs1$	$cs0$		
0	X	0	Subtract with borrow	$A - B - 1$
0	0	1	Transfer A	A
0	1	1	Subtract	$A - B$
1	0	0	Increment A	$A + 1$
1	0	1	AND	$A \wedge B$
1	1	X	OR	$A \vee B$

Table 1: Problem Specification

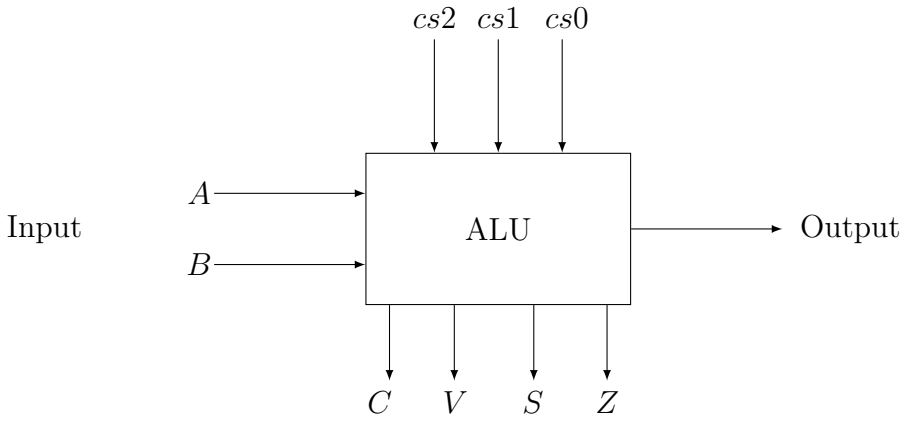

Figure 1: 4-bit ALU

3 Detailed Design Steps with K-maps

3.1 Design Steps

1. The arithmetic unit performs 4 arithmetic operations (Subtract with borrow, Subtract, Transfer A, and Increment A) with the help of a 4-bit full adder and an 8×1 multiplexer.

2. For the adder, the first input A is fixed; the second input is Y which is varied with different combinations of the three selection bits. For subtract with and without borrow operations, $\overline{B_i}$ is selected whereas for Transfer A and Increment A operations, 0 is selected as Y_i. For OR and AND operations, $\overline{A_i}B_i$ and $\overline{A_i}+B$ are sent in as Y_i.

3. In the arithmetic unit, the adder adds A, $\overline{B}$ with $C_{in} = 0$ and $C_{in} = 1$ for subtract with borrow and subtract operations respectively. $Y = 0000, C_{in} = 0$ for transfer operation, so adder outputs $A + 0 + 0 = A$. During increment operation, $Y = 0000$, $C_{in} = 1$ that is adder adds 1 to A which is basically equivalent to $A + 0 + 1 = A + 1$.

4. For eight input variation,two 4×1 multiplexer is used along with one 2×1 multiplexer to work as an 8×1 multiplexer. For this purpose, IC 74157 and IC 74153 are used. $cs1$, $cs0$ are used as selection bits for the first two 4×1 multiplexer while $cs2$ acts as the selection bit for the 2×1 multiplexer.

5. The carry flag is computed from the arithmetic unit. The overflow flag (V) is explicitly handled using equation 3. These flags are determined using a Boolean expression as a selection bit to a 2×1 multiplexer. We will derive this expression in the K-map section. This method was chosen to minimize IC usage.

6. Zero flag, Z is computed by adding (OR) the 4 output bits and inverting the output using NOT gate (IC 7504). This is the NOR operation of all four output bits.

3.2 K-maps

We will be following Table 2 to construct the K-maps for C_{in}, V, and C flag.

3.2.1 K-map for C_{in}

C_{in} is the input carry to the adder. It is changed based on different types of operations.

$cs2cs1$ \ $cs0$	0	1
00	0	0
01	0	1
11	0	0
10	1	1

We can express C_{in} like below:

$$C_{in} = cs2\overline{cs1} + \overline{cs2}cs1cs0$$

3.2.2 K-map for V

It is simply V from equation 2. But it should be cleared after the logical operations, So we can handle that by drawing a k-map like below (We denote the V from equation 2 as V' here):

$cs2cs1$ \ $cs0$	0	1
00	V'	V'
01	V'	V'
11	0	0
10	V'	0

$$V = (\overline{cs2} + \overline{cs1cs0}) \cdot V'$$

The same equation can be derived for CF.

$$C = (\overline{cs2} + \overline{cs1cs0}) \cdot C_{out}$$

Here our derived equation is the actual k-map of V and C flag. In our implementation, we used a 2×1 multiplexer that takes a Boolean expression as a selection bit. When it is true or set to 1, we choose ground as an output - otherwise, we choose V. The k-map for this particular selection bit will look like this:

$cs2cs1$ \ $cs0$	0	1
00	0	0
01	0	0
11	1	1
10	0	1

Now the derived Boolean expression will be:

$$X = cs2 \cdot (cs1 + cs0)$$

It will act as the selection bit for two 2×1 multiplexers - one for V and the other for C flag.

4 Truth Table

For a better interpretation of the variables used, refer to the figures in section 5.

cs2	cs1	cs0	Function	Y	C_{in}	V	C
0	0	0	Subtract with borrow	$\overline{B}$	0	V	C
0	0	1	Transfer A	0	0	V	C
0	1	0	Subtract with borrow	$\overline{B}$	0	V	C
0	1	1	Subtract	$\overline{B}$	1	V	C
1	0	0	Increment A	0	1	V	C
1	0	1	AND	$\overline{A}+B$	1	0	0
1	1	0	OR	$\overline{A}B$	0	0	0
1	1	1	OR	$\overline{A}B$	0	0	0

Table 2: Truth Table for Intermediate I/O

5 Block Diagram

5.1 Block Diagram for Deriving Y_i

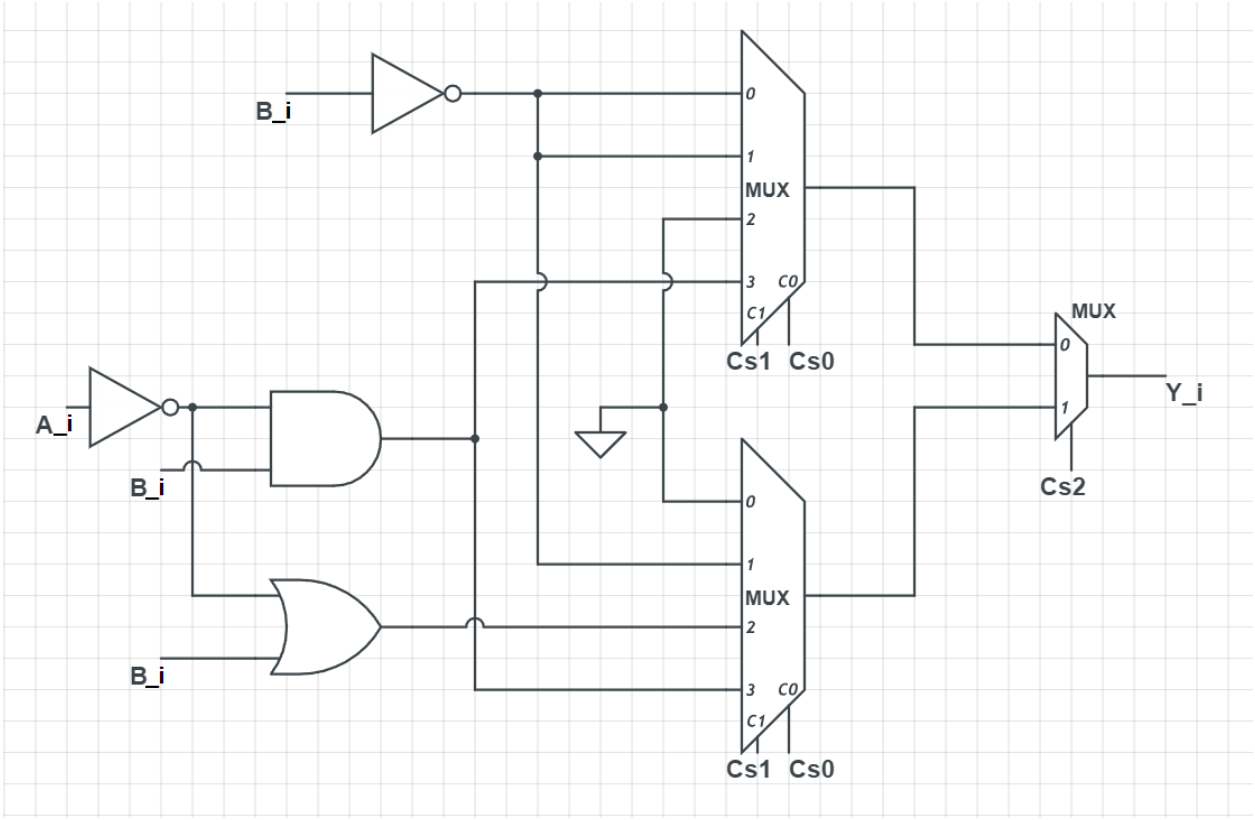

Figure 2: Deriving Y_i from A_i and B_i using control bits

5.2 Block Diagram for Deriving C_{in} and X

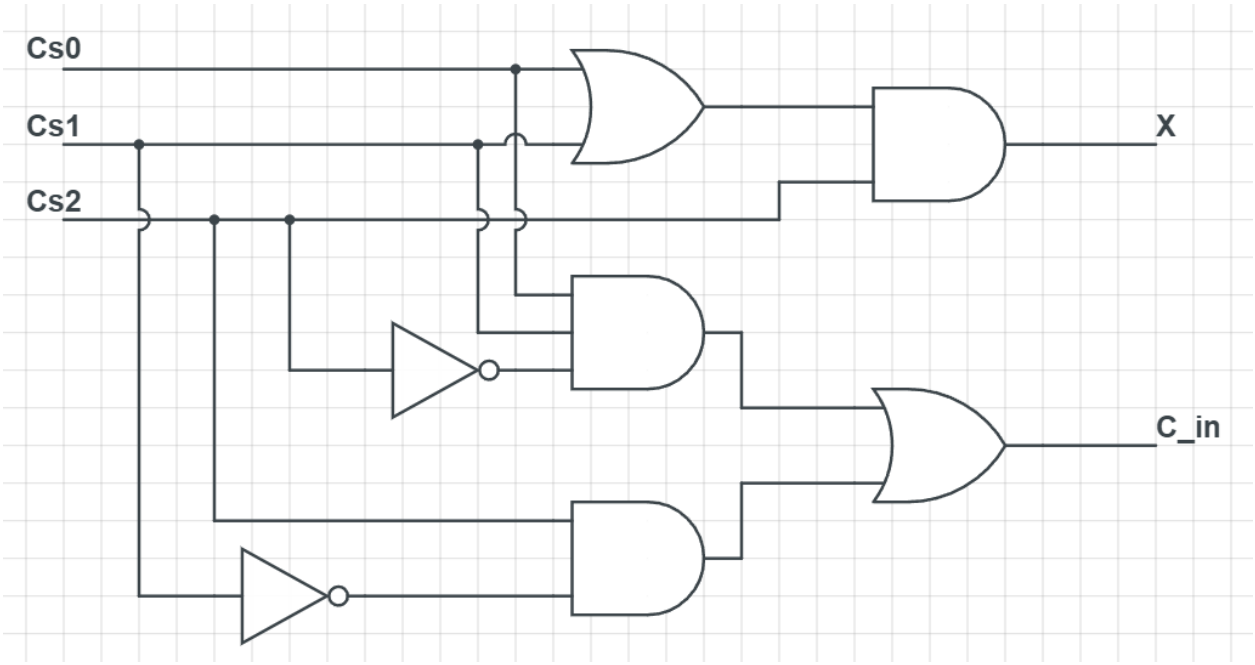

Figure 3: Deriving C_{in} and X from using control bits

5.3 Final Block Diagram

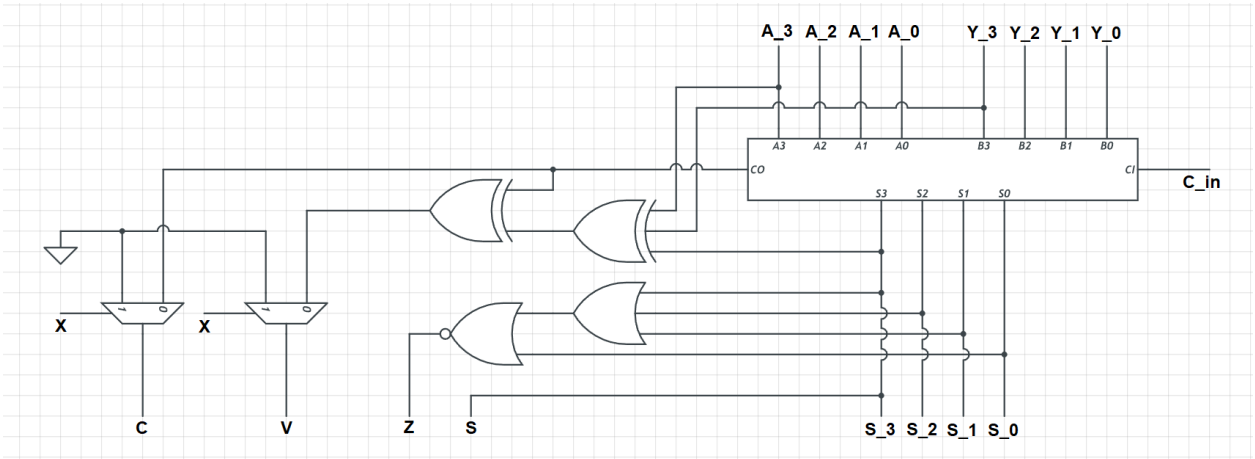

Figure 4: Final block diagram of ALU

6 Complete Circuit Diagram

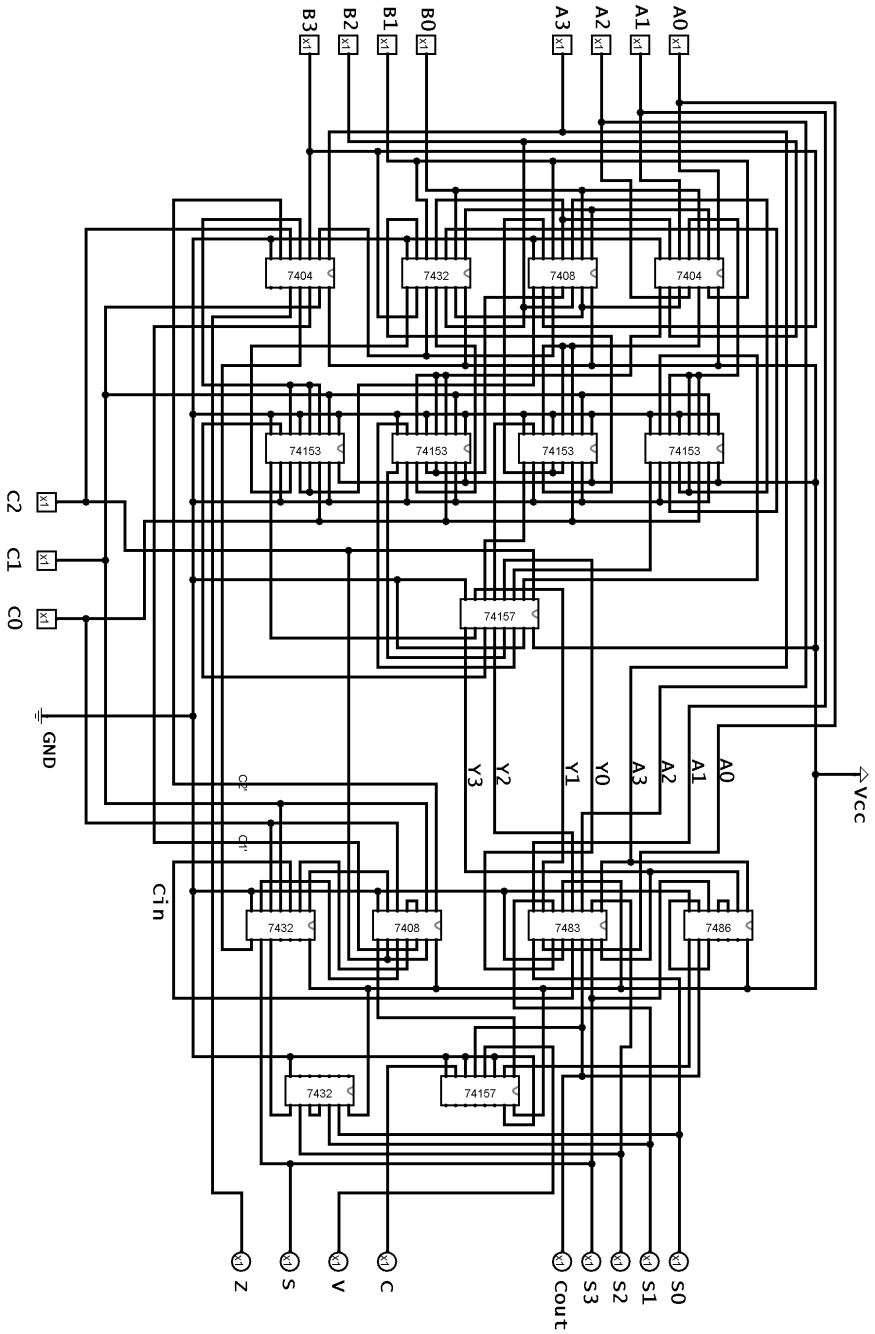

Figure 5: Complete Circuit of ALU

7 ICs Used with Count as a Chart

IC	Quantity
IC 7400	2
IC 7408	2
IC 7432	3
IC 7483	1
IC 7486	1
IC 74153	4
IC 74157	2
Total	15

Table 3: ICs Used with Quantity

8 Essential Softwares

We used **Logisim - 2.7.1** to design and simulate the ALU before hardware implementation.

9 Discussion

In this assignment, we were tasked to implement a 4-bit ALU which performs 4 arithmetic and 2 logical operations.
Throughout the designing process, our major challenge was deriving the logical operations from the arithmetic operations while keeping the number of ICs used at a minimum. The procedure was complicated particularly because in the IC 77483 Parallel Adder IC, the internal carry signals are not accessible. Therefore, we derived equations for the second input of the adder, Y, as shown in Table 2 so that manipulating the C_{in} will be enough for logical operations. For the AND operation, we set $C_{in} = 1$ and designed the circuit so that all the internal carry signals are also 1. For the OR operation, we set $C_{in} = 0$ and designed the circuit so that all the internal carry signals are also 0.
The hardware implementation also posed challenges like planning the placement of different modules. We divided the whole circuit into three levels and implemented them in three breadboards, which were merged later on. Moreover, we dedicated two other breadboards for input switches and output LED lights. To keep the design clean, we connected the wires so that they crossed each other as rarely as possible. We connected the power and ground connections with caution to prevent the IC or other components from getting damaged.

10 Contribution

i **2005003**

- Logical design of the status flags C and V
- Software simulation of the adder and status flags implementations
- Hardware implementation of the adder and status flags circuit
- Merging, checking, and debugging the final circuit implementation

ii **2005004**

- Logical design of the status flags S and Z
- Merging all 1-bit implementation simulations and designing the final circuit
- Hardware implementation of the MUX circuit that connects the input circuit with the adder circuit
- Figures, Discussion, and miscellaneous sections in the final report

iii **2005006**

- Logical design of input manipulation (A_i, Y_i) for logical operations
- Software simulation of the MUX and basic logical gates implementations
- Hardware implementation of the MUX circuit that connects the input circuit with the adder circuit
- Checking and debugging the MUX circuit implementation

iv **2005018**

- Logical design of control bit manipulation (C_{in}, X) for both arithmetic and logical operations
- Software simulation of the MUX and basic logical gates implementations
- Hardware implementation of the MUX circuit that connects the input circuit with the adder circuit
- Design steps, truth tables, and K-maps in the final report

v **2005019**

- Logical design of input manipulation (A_i, Y_i) for arithmetic operations
- Software simulation of the MUX and basic logical gates implementations
- Hardware implementation of the input circuit for both arithmetic and logical operations
- Checking and debugging the input circuit implementation