Bangladesh University of Engineering and Technology

CSE 210

CSE 210 Computer Architecture Sessional

Assignment-1: 4-bit ALU Simulation

Section - C1

Group - 01

Members of the Group:

i	2105123	- Shatabdi Dutta Chowdhury
ii	2105137	- Arpita Dhar
iii	2105140	- Nusrat Jahan Tamanna
iv	2105141	- Md Mehedi Hasan Khan
v	2105147	- Tasnimzaman Tanmi

A Introduction

An Arithmetic Logic Unit (ALU) is a crucial element within a computer's central processing unit (CPU). It acts as the computational core of the processor, responsible for executing essential arithmetic and logic operations needed to perform various tasks. The ALU's main functions include basic arithmetic operations like addition, subtraction, multiplication, and division, as well as logical operations such as AND, OR, and NOT.

The ALU works with binary data, manipulating 0s and 1s to carry out mathematical computations and decision-making processes. Integrating the ALU into the CPU allows a computer to execute complex calculations, process data, and run program instructions, making it a vital part of a computer's overall functionality. The ALU's performance directly affects the speed and efficiency of the processor, impacting the computer's ability to handle diverse computational tasks.

An ALU consists of two main units: the Arithmetic Unit and the Logic Unit. It can perform operations like addition, subtraction, incrementing, decrementing, and logical operations like NOT, OR, XOR, and AND. To select a specific operation, the ALU uses selection lines or bits. In this implementation, the ALU uses three control selection inputs.

The arithmetic section of the ALU is built using a parallel adder, with various arithmetic operations achieved by using multiplexed inputs to the integrated circuit (IC). Logical operations are handled by appropriate ICs.

The designed ALU operates on 4-bit inputs, producing 4-bit outputs, and includes four additional status flags. These flags are:

- **Carry Flag (C)**: Indicates a carry out of the adder result. A carry sets the flag to 1, otherwise it remains 0.
- **Zero Flag (Z)**: If the ALU's last operation results in zero, this flag is set to 1, otherwise it is 0.
- **Overflow Flag (V)**: When an arithmetic operation exceeds the bit range, this flag is set. After logical operations, the flag is reset to 0.

  The overflow is calculated as:

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

  where C_3 is the carry generated from the third bit, and C_{out} is the final carry from the adder.
- **Sign Flag (S)**: Reflects the most significant bit (MSB) of the result, determining the sign for signed arithmetic operations.

B 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**		
0	0	0	Sub	$A - B$
0	0	1	Add with carry	$A + B + 1$
0	1	0	AND	$A \wedge B$
0	1	1	XOR	$A \oplus B$
1	X	0	Complement A	A'
1	X	1	Decrement A	$A - 1$

Table 1: ALU Control Signals, Functions, and Descriptions

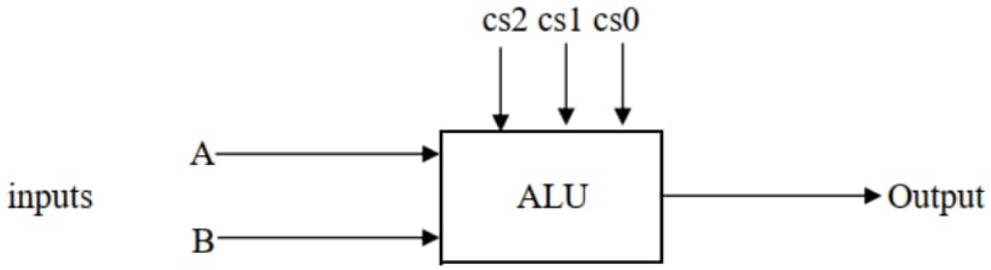


Figure 1: 4 Bit ALU

C Detailed Design Steps With K-map

C.a Design Steps

1. In the given problem there is a combination of 6 operations including 3 logical operations and 3 arithmetic operations.For the opeartions we have directly use the adder(IC-7483).

2. For the adder we have to give input two numbers of 4 bits which we assume A($A_3A_2A_1A_0$) and B($B_3B_2B_1B_0$).

   (a) **Input X_i(Modified version of A_i) :** Here we are give the input of A as the logical operation (xor , and) on A with B or 1.

      i. For this at first we select between B and 1 using the 2X1 Multiplexer (MUX-1) where the selection bit is S_1. If $S_1 = 0$ B will be selected , so the output bits are $B_3B_2B_1B_0$ .Otherwise 1 will be selected , the output bits are all four 1.

      ii. After this selection , there is performed two logical operation(XOR and AND) on the selected bits and the bits of A.

      iii. From the logical operation XOR and AND which will be performed is decided by another 2X1 Multiplexer(MUX-2) where the selection bit is S_2.If $S_2 = 0$ the XOR will be performed and otherwise AND will be performed.

      iv. From this MUX-2 we will get the modified value of the four bits of A (we assume X($X_3X_2X_1X_0$)) ,which are the input as ($A_3A_2A_1A_0$) for the adder.

   (b) **Input Y_i(Modified version of B_i) :** For the selection of the second input of the adder another 2X1 Multiplexer (MUX-3) is used where the selection bit is S_3.

      i. If $S_3 = 0$ there will be selected $B \oplus cs_0'$. For 0 we will get the modified version of $B(B_3B_2B_1B_0)$.

      ii. For $S_3 = 1$ there will be selected the AND of CS_2 and CS_0.
      Thus the selected output of four bits will be the value of second input (we assume Y($Y_3Y_2Y_1Y_0$)) for the adder.

   (c) **Input Z_i(Modified version of C_{in}) :** Here for the first two arithmetic operation (subtraction and add with carry) we need 1 as C_{in} of the adder. Otherwise we don't need any carry input .That's why we input 0 in the C_{in} of the adder for the rest operations.

3. For the S_1, S_2, S_3, Z_i we have showed the k-map.As we have to use the minimum IC, for decrementing the IC count we have not use the functions from the k-map directly , rather we have do some modifications to implement the selection bits.

4. To implement the flag C(Carry flag) , S(Signed bit) , Z(Zero flag) , V(Overflow flag) we have done the following:

   (a) **Carry Flag (C) :** We compute this by checking if the result of the addition or subtraction exceeds the maximum value that can be represented by the given bit width (for example, 4 bits for a 4-bit ALU).So here the adder's C_{out} basically represents the carry flag.

   (b) **Sign Bit (S) :** The signed bit is typically the most significant bit (MSB) of the result. It indicates whether the result is positive or negative in signed arithmetic. We determine this directly from the MSB of the result. So here the summation bits ($S_3S_2S_1S_0$) is the result from the adder's output and so the most significant bit (S_3) represents the sign bit.

   (c) **Zero Flag (Z) :** The zero flag is set when the result of an arithmetic or logical operation is zero. We implement this by comparing the result to zero and setting the flag if the result is indeed zero.So for the result from the adder if the all four bits ($S_3S_2S_1S_0$) are 0 then the Z flag is 1(set). Otherwise the Z flag remains 0.

   (d) **Overflow Flag (V) :** The overflow flag is used to detect when an arithmetic operation results in an overflow, i.e., when the result is too large to be represented with the given number of bits. This is typically detected by checking whether the carry into the MSB is different from the carry out of the MSB (for addition or subtraction). For the summation of bits in adder:
   Here $C_{in} = C_0$

   $$S_i = A_i \oplus B_i \oplus C_i \tag{1}$$

   So we can also get C_i by using self-inverse property (or involution property) of the XOR (exclusive OR) operation.so

   $$C_i = A_i \oplus B_i \oplus S_i \tag{2}$$

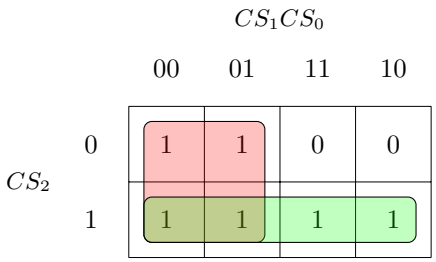
   From this idea we have got the carry for the last bit of inputs (In my assumption C_3) . And the C_{out} (assuming C_4) has been got from the adder's C_{out}. By XOR-ing C_4 and C_{out} we can show if there is any overflow or not. *Overflow is possible if both the inputs are positive or negative.*

C.b K-maps

We will be following Table 2 [Truth Table for ALU Design] to construct the K-maps for intermediate selection bits.

C.b.1 K-map for S_1

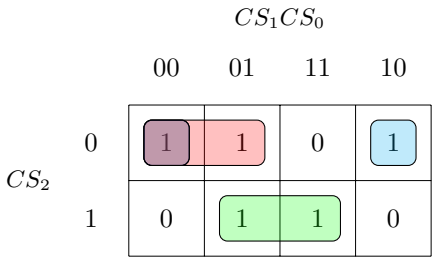
S_1 is the selection bit for the multiplexer that selects B($B_3B_2B_1B_0$) or 1(1111) that will do XOR and AND operation with A($A_3A_2A_1A_0$).



$$S_1 = CS_2 + CS_1'$$

C.b.2 K-map for S_2

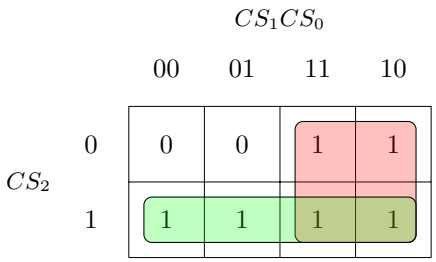
S_2 is the selection bit for the multiplexer that that selects the first input of the adder between the XOR and AND operation of A and B or 1.



$$S_2 = (CS_2 + CS_1)' + (CS_2 \oplus CS_0)'$$

C.b.3 K-map for S_3

S_3 is the selection bit for the multiplexer that that selects the second input of the adder between the B XOR $(CS_0)'$ and CS_0CS_2.



$$S_3 = (CS_2 + CS_1)$$

For decrementing the IC count, CS_0CS_2 is not used directly as the second input in MUX-3. There are some equational modifications to use the remaining empty gate of the ICs.

$$\begin{aligned}
&((CS_2 \oplus CS_0) + CS_0')' \\
&= (CS_2'CS_0 + CS_0'CS_2 + CS_0')' \\
&= (CS_2'CS_0 + CS_0'(CS_2 + 1))' \\
&= (CS_2'CS_0 + CS_0')' \\
&= ((CS_2' + CS_0')(CS_0 + CS_0'))' \\
&= (CS_2' + CS_0')' \\
&= CS_2CS_0
\end{aligned}$$

C.b.4 K-map for C_{in}

It is the input carry input of the adder. It will be 1 only at the time of $CS_2CS_1CS_0 = 000$ and $CS_2CS_1CS_0 = 001$. In all other cases, it is set to be 0.

CS_2 \ CS_1CS_0	00	01	11	10
0	1	1	0	0
1	0	0	0	0

$$C_{in} = (CS_2 + CS_1)'$$

D Truth Table

For better interpretation of the variables used,refer to figure 2 [Block Diagram]

CS_2	CS_1	CS_0	**Function**	x_i	y_i	z_i	S_1	S_2	S_3
0	0	0	Subtraction	$A_i \wedge 1$	B'	1	1	1	0
0	0	1	Add with carry	$A_i \wedge 1$	B	1	1	1	0
0	1	0	AND	$A_i \wedge B_i$	0	0	0	1	1
0	1	1	XOR	$A_i \oplus B_i$	0	0	0	0	1
1	0	0	Complement A	$A_i \oplus 1$	0	0	1	0	1
1	0	1	Decrement A	$A_i \wedge 1$	1	0	1	1	1
1	1	0	Complement A	$A_i \oplus 1$	0	0	1	0	1
1	1	1	Decrement A	$A_i \wedge 1$	1	0	1	1	1

Table 2: Truth Table for ALU Design

E Block Diagram

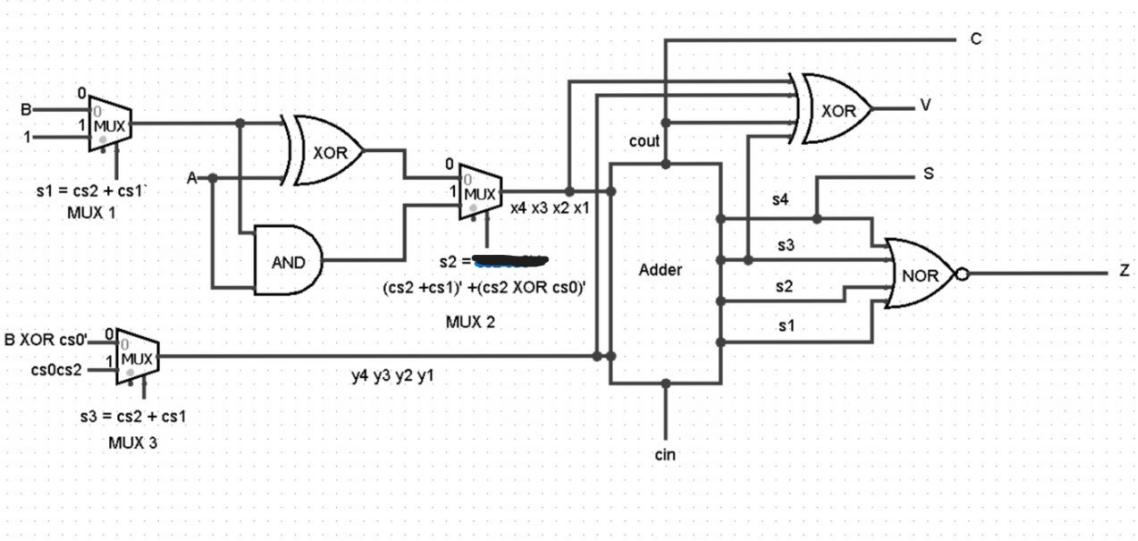


Figure 2: Block Diagram

F Complete Circuit diagram

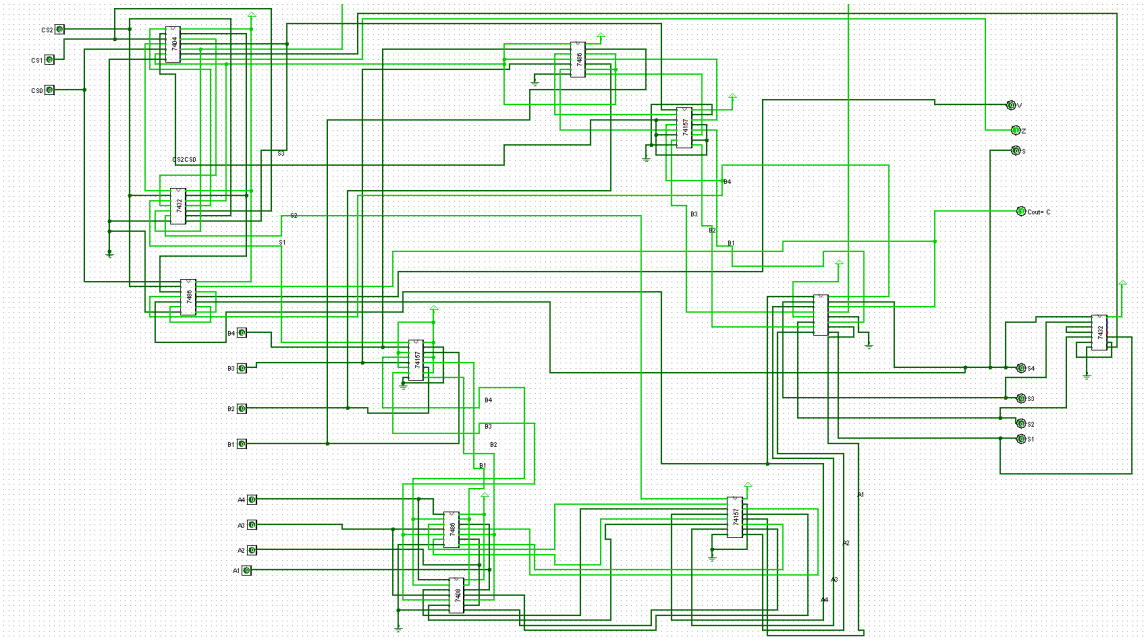


Figure 3: Logism Circuit

G ICs Used with Count as a Chart

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

Table 3: ICs Used with Quantity

H The Simulator Used along with the Version Number

Logisim - 2.7.1

I Description of 4-bit ALU Simulation

A 4-bit ALU (Arithmetic Logic Unit) simulation is designed to perform fundamental operations on two 4-bit binary numbers. The ALU can execute both arithmetic operations, such as addition and subtraction, and logical operations, such as AND, OR, and XOR. In this assignment, we were tasked to implement a 4-bit ALU which performs 3 arithmetic (SUB,Add with carry,Decrement)and 3 logical operations(AND,XOR,Complement).

Through rigorous scrutiny, we had to strive hard to obtain the design with the minimum number of ICs. The total IC count of the final design is found to be 11.

After the software implementation, it was thoroughly tested for multiple times by multiple group members to avoid all kinds of error or corner cases. The hardware implementation was only initiated after being assured that the software is fully functional.

The hardware implementation requires both working instruments and proper connections. Each component was tested before its use. Each connection was implemented with extra caution so that the correctness of the circuit remains intact. To keep the hardware design clean, we had to connect the wires so that they cross as little length as possible between the connections.

It can be said that after multiple iterations of design process and efficient design choices, the obtained ALU is an efficient implementation for the given task.

J Contribution

Software Implementation

Xi: 2105123,2105137,2105147

Yi: 2105123,2105137,2105147

flag:2105123,2105147,2105140

Adder:2105123,2105140,2105141

Testing: 2105137,2105140,2105147

Hardware Implementation

Xi: 2105123,2105137,2105147

Yi: 2105123.2105140,2105147

Adder: 2105123,2105141,2105147

Flag: 2105123,2105137,2105147

Testing:2105123,2105137,2105140,2105141,2105147

Report: 2105123,2105137,2105140,2105141,2105147