Bangladesh University of Engineering and Technology

CSE 306
Computer Architecture Sessional

Offline 1: 4-bit ALU Simulation

Section - B1
Group - 5

2005061 - Farriha Afnan
2005072 - Abrar Rahman Abir
2005079 - Ananya Shahrin Promi
2005087 - Iffat Bin Hossain
2005089 - Wahid Al Azad Navid

December 15, 2023

1 Introduction

ALU stands for Arithmetic Logic Unit. It is a fundamental component of a computer processor that performs arithmetic and logical operations, such as addition, subtraction, AND, OR, and XOR, on binary data. The ALU is responsible for carrying out the mathematical and logical computations that are essential for the functioning of a computer.In addition to carrying out fundamental mathematical and logical computations, the ALU produces a set of flags that indicate the status of the result. Its input comprises two binary numbers and control bits; the binary numbers serve as operands, while the control bits determine the operation to be executed.
In this assignment, we have to perform the design and implementation of a 4-bit Arithmetic Logic Unit (ALU) which will encompass both the software simulation and hardware construction of a 4-bit ALU.
A key emphasis in this assignment is on efficiency – specifically, the minimization of integrated circuits (ICs). By adopting a minimalist approach, we aim to create an ALU design that is not only functional but also optimized in terms of component usage.
In the forthcoming segments of this report, we will begin with 'Problem Specification and Assigned Instructions', followed by intricate 'Design Steps with K-Maps', progressing through 'Truth Tables', 'Block Diagrams', and 'Complete Circuit Diagrams'. We'll also include a detailed 'Chart of ICs Used', specify the 'Simulator and Version' used for testing, and conclude with in-depth 'Discussions' and an acknowledgment of 'Each Member's Contribution' to this assignment.

2 Problem Specification with assigned instructions

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

cs_2	cs_1	cs_0	**Function**	**Output**
0	0	0	Decrement A	$F = A - 1$
0	0	1	Transfer A	$F = A$
$\times$	1	0	Complement A	$F = \overline{A}$
0	1	1	Add	$F = A + B$
1	0	$\times$	Add with carry	$F = A + B + 1$
1	1	1	XOR	$F = A \oplus B$

Table 1: Functional Design Specification

In addition, we need to implement the four status outputs, or flags, denoted as C (Carry Flag), S (Sign Flag), V (Overflow Flag), and Z (Zero Flag). Their representations convey specific meanings: C contains the output carry C_{out} of the operation, S contains the sign of the result of the operation, V indicates any overflow has occurred due to the operation, and Z indicates whether the n-bit of the result is 0 or not.
Flags will be affected as per the rules of Assembly Language. Our ALU is required to output flags under the simplified rules:
For NOT Operation: After the NOT operation, which makes the result 0000, if Z becomes 0 from 1, it will not be accepted. However, if Z becomes 1 or if the Z flag remains unchanged, both will be accepted. If S remains unchanged or reflects the highest order bit of the result, both will be accepted. But if the S flag is changed and it is changed to a wrong value, it will not be accepted.

For AND/OR/XOR Operation: C and V should be cleared after the operation. S and Z should be changed according to the output.
Here we are allowed to use only 2 input SSI (AND, OR, NOT, XOR, etc.) and MSI (MUX, Decoder, Adder, etc.). We are required to build an efficient design with the minimum possible number of ICs.
For simulation, we used Logisim (circuit analyzing software).

3 Detailed Design Steps with K-maps

3.1 Design Steps

Initially, we designate the four bits of operands A and B as $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ respectively. Here, A_3 and B_3 represent the most significant bits of A and B, respectively. Likewise, the selector bits cs_2, cs_1, and cs_0 are defined to determine the ongoing operations.
There are two types of operations: Arithmetic and Logical. We are asked to implement four arithmetic operations: Addition, Addition with carry, Decrement, and Transfer. And two logical operations: exclusive-OR, and complement. We are using 4-bit full adders as 2-bit full adders (Because for logical operation, we need to modify out carry in every bit).
Addition: For addition, we take inputs A and B and set the carry input to 0.
Addition with carry: For addition with carry, we change the carry input to 1 and keep inputs as A and B.
Decrement: For the decrement operation on A, we set the inputs to A and 1111 (which represents -1), with the carry input set to 0.
Transfer: To transfer A, we use inputs A and 0000 with a carry input of 0.
Exclusive OR: For the logical XOR operation, we utilize inputs A and B to obtain the bit-wise XOR of A and B. In each step, a carry of 0 is generated using a flag. Additionally, we set the carry input to 0 to obtain the result.
Complement: To complement A, we use A and 1111 as inputs, with the carry input set to 0. Utilizing the property $x \oplus 1 = \overline{x}$, the result is $\overline{A}$.
Thus, we constructed the circuit for the final 4-bit output F. Additionally, we were tasked with determining four flag bits: Carry bit (C), Unsigned Overflow bit (S), Signed Overflow bit (V), and Zero bit (Z).
Carry Flag: Carry Flag (C) is the carry output of the arithmetic unit. During logical operations, every carry is set 0 so there is no chance of overflow and carry keeping $C = 0$ for logical operations.
Sign Flag: Sign Flag (S) can be directly accomplished from the F_3 output bit.
Overflow Flag: The equation for the overflow Flag (V) flag can be determined from existing intermediate results. There is no chance of overflow and carry keeping $C = 0$ for logical operations.
Zero Flag: This flag must be set when all four bits of F are zero and must be unset otherwise. Therefore, the equation for the Z flag is simply: $Z = \overline{F_3 + F_2 + F_1 + F_0}$

Using A/L to handle Flags :

To align with the specific instructions of our assignment, we've introduced a variable A/L, where 1 signifies an arithmetic operation, and 0 denotes a logical operation. To meet the assignment's requirements, we strategically utilize A/L for managing the carry flag (C) and overflow flag (V). We perform bitwise AND operations, clearing C and V when A/L is 0 (indicating a logical operation) and preserving their values for arithmetic operations ($A/L = 1$). This approach ensures that both C and V are reset to 0 after logical operations while retaining their states post-arithmetic operations.

Optimization techniques

Our emphasis is on minimizing ic numbers. So here are the optimizations we have done :

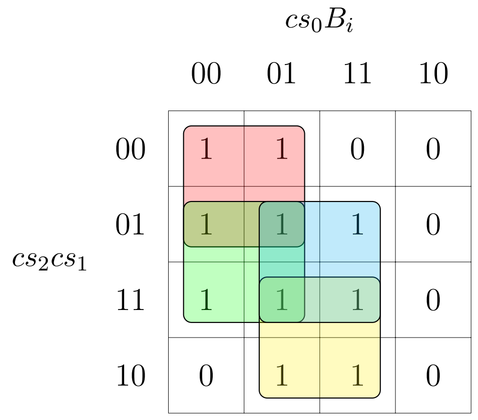
- The expression for Y_i is $Y_i = \overline{cs_2}\,\overline{cs_0} + cs_1\overline{cs_0} + cs_1 B_i + s_2 B_i$ and Z_{in} is $Z_{in} = cs_2\overline{cs_1}$. Here we use $\overline{Z_{in}} = \overline{cs_2} + cs_1$ in the expression of Y_i as $Y_i = \overline{cs_0}\cdot(cs_1 + \overline{cs_2}) + cs_1 B_i + s_2 B_i$ and finally reduces to $Y_i = \overline{cs_0}\cdot\overline{Z_{in}} + B_i\cdot(cs_1 + s_2)$
- Zero flag, $Z = \overline{F_4 + F_3 + F_2 + F_1}$ and we optimize it as $Z = \overline{F_1 + F_2}\cdot\overline{F_3 + F_4}$

3.2 K-maps

We will be following Table 2 to construct the K-maps. We are keeping A fixed (Hence, $X_i = A_i$) and changing B to generate different operations.

3.2.1 K-map for Y_i

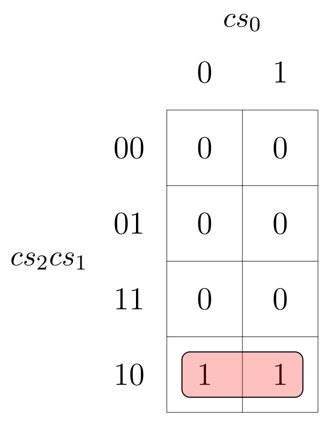
Y_i represents modified representation of B_i and thus Y represents B. So, the following 4 combinations can be obtained: Keeping B as it is ($Y_i = B_i$), Inverting all bits of B ($Y_i = \overline{B_i}$), Changing each bit of B to 0 ($B_i = 0$), and Changing each bit of B to 1 ($B_i = 1$).



$$Y_i = \overline{cs_2}\,\overline{cs_0} + cs_1\overline{cs_0} + cs_1 B_i + s_2 B_i$$

3.2.2 K-map for Z_{in}

Z_{in} represents the modified carry input, denoted as C_{in}, provided to the adder circuit based on the specific operation being performed. The modification of Z_{in} depends on the nature of the operation being carried out in the circuit.

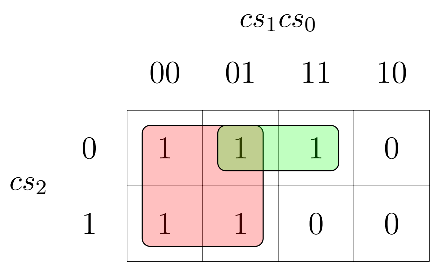


cs_2cs_1 \ cs_0	0	1
00	0	0
01	0	0
11	0	0
10	1	1

$$Z_{in} = cs_2\overline{cs}_1$$

3.2.3 K-map for A/L

We also maintain a flag A/L to determine the type of operation, whether it is an arithmetic or logical operation. We set $A/L = 1$ for arithmetic operations and $A/L = 0$ for logical operations.



cs_2 \ cs_1cs_0	00	01	11	10
0	1	1	1	0
1	1	1	0	0

$$A/L = \overline{cs}_1 + \overline{cs}_2cs_0$$

4 Truth Table

For a clearer interpretation of the variables employed, we constructed this truth table:

cs_2	cs_1	cs_0	X_i	Y_i	Z_{in}	**Function**	A/L
0	0	0	A_i	1	0	$F = A - 1$	1
0	0	1	A_i	0	0	$F = A$	1
0	1	0	A_i	1	0	$F = \overline{A}$	0
0	1	1	A_i	B_i	0	$F = A + B$	1
1	0	0	A_i	B_i	1	$F = A + B + 1$	1
1	0	1	A_i	B_i	1	$F = A + \overline{B} + 1$	1
1	1	0	A_i	1	0	$F = \overline{A}$	0
1	1	1	A_i	B_i	0	$F = A \oplus B$	0

Table 2: Truth Table

5 Block Diagram

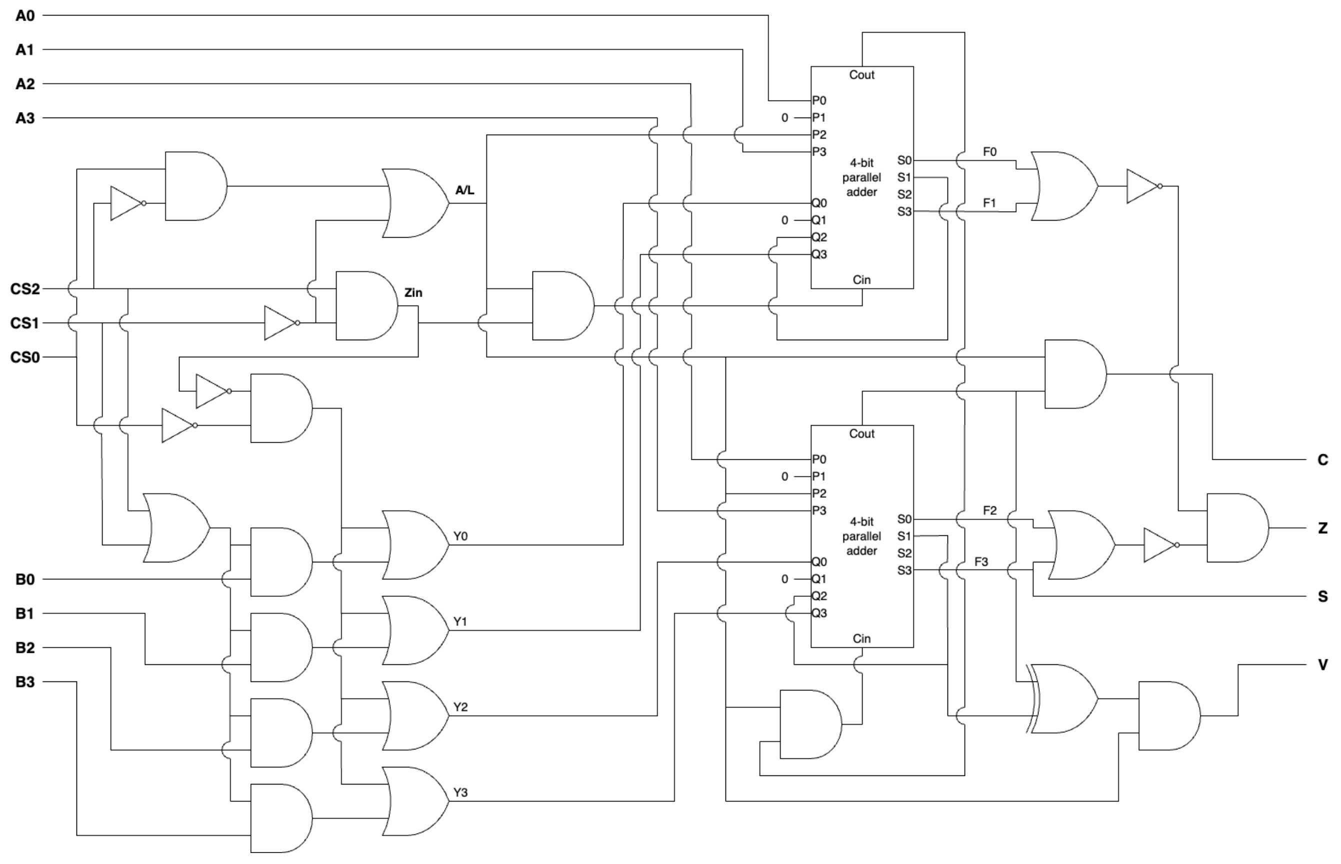


Figure 1: Block Diagram of ALU

6 Complete Circuit diagram

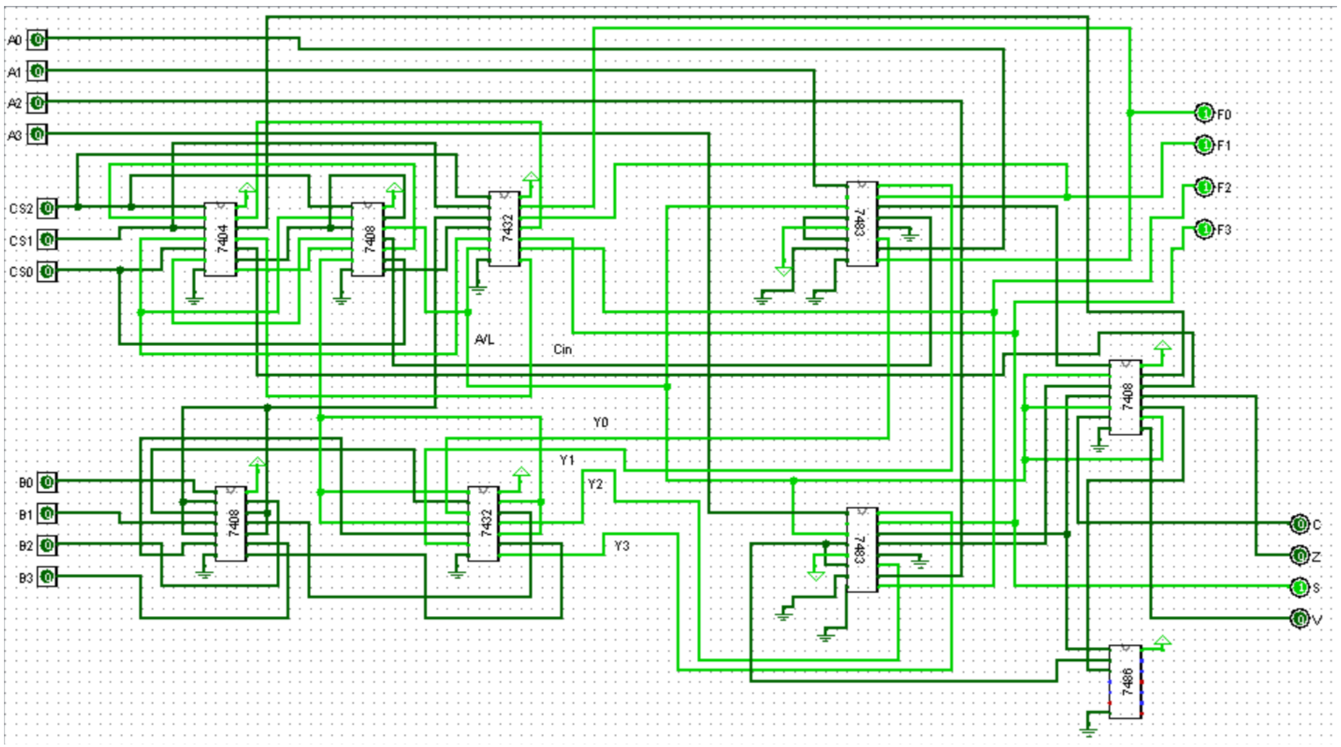


Figure 2: Complete Diagram of ALU (Logisim)

7 ICs used with count as a chart

In this section, the integrated circuits (ICs) utilized in the implementation, along with their respective quantities, are presented in a concise chart:

IC Number	IC Name	Quantity
SN74HC08N	Quad 2-Input AND	3
SN74HC32N	Quad 2-Input OR	2
SN74HC86N	Quad 2-Input XOR	1
SN74HC04N	1-Input Hex-Inverter	1
SN74HC83N	4-bit Binary Full Adder	2

Table 3: ICs Used with Quantity

8 The simulator used along with the version number

Software: Logisim
Version: Logisim-win-2.7.1

9 Discussions

For this assignment, our objective was to design a 4-bit ALU capable of performing 4 arithmetic and 2 logical operations. The hardware implementation presented challenges, particularly in strategically planning the placement of various modules.

- Throughout the process, we undertook a meticulous examination, putting in considerable effort to achieve a design that utilized the minimum number of integrated circuits (ICs).
- To maintain a clean hardware design, significant attention was given to wiring, minimizing the length of wire crossings between connections.
- All of the ICs and breadboards were carefully checked before using them in the circuit to ensure that all of those were working perfectly.
- Additional effort was invested not only to ensure the functionality of the hardware but also to achieve an aesthetically pleasing design.
- Careful consideration was taken in establishing power and ground connections to prevent potential damage to integrated circuits (ICs) and other components.
- The outputs were checked multiple times for a large number of input test cases to ensure that our circuit satisfies the expected output values from the truth table.
- The integration of both software simulation and hardware construction provided a comprehensive understanding of the ALU's operations. The software implementation allowed us to validate the logic and functionality before moving to the hardware phase.
- The software implementation allowed for swift iterations and debugging, contributing to the efficiency of the overall design process.
- Documentation and organization were maintained, facilitating clear communication and collaboration among team members across both software and hardware aspects.

After addressing these considerations, we are optimistic about the successful implementation of the 4-bit Arithmetic and Logic Unit (ALU) with a minimal number of ICs.
Through the course of this assignment, we gained invaluable insights into the intricate world of digital system design. The dual focus on software and hardware implementations provided a holistic learning experience, allowing us to bridge the gap between theoretical concepts and real-world applications. The challenges encountered during the strategic planning of hardware modules underscored the importance of meticulous design, emphasizing efficiency in utilizing integrated circuits (ICs). The integration of software simulation not only streamlined the debugging process but also showcased the significance of iterative development in complex projects. Additionally, the emphasis on clean wiring, aesthetic considerations, and thorough testing highlighted the importance of attention to detail in achieving a functional and visually pleasing hardware design. Collaborative documentation practices further reinforced the necessity of clear communication within a team. In essence, this assignment served as a practical laboratory for honing our skills in digital system design, offering valuable lessons that extend beyond the realm of ALU construction to various facets of hardware and software integration.

10 Contribution of Each Member

Activity	**Roll**
Steps Designing	87, 89
Truth Table and K-Map	79, 87
Optimization and IC Minimization	72, 89
Block Diagram	79, 87
Logisim Implementation	61, 87, 89
Equipment Management	61, 72, 89
Hardware Implementation	61, 72, 79, 87
Hardware Circuit Checking (step by step)	61, 72, 89
Finding and Fixing Errors in Circuit	61, 79
Report Writing	72, 79

Table 4: Contribution Chart