main report

nafees

October 2024

1 Introduction

An arithmetic logic unit (ALU) is a multioperation, combinational logic digital function. It can perform basic arithmetic operations such as addition, subtraction, increment, decrement, transfer and logic operations such as NOT, OR, XOR, AND etc. The ALU has a number of selection lines to select a particular operation in the unit. The selection lines are decoded within the ALU so that k selection variables can specify up to 2^k distinct operations.

The design of a typical ALU is carried out in three stages. First, the design of the arithmetic section is undertaken. Second, the design of the logic section is considered. Finally, the arithmetic section is modified so that it can perform both arithmetic and logic operations.

The ALU operates on binary data, manipulating 0s and 1s to carry out mathematical computations and decision-making processes. By integrating the ALU into the CPU, a computer can perform complex calculations, process data, and execute program instructions, making it a critical component in the overall functionality of computing systems. The efficiency and speed of an ALU significantly impact the overall performance of a processor, influencing the capability of a computer to handle various computational tasks.

Parallel adders and several basic logic gates can be used to implement an efficient ALU. The implemented ALU operates on 4-bit inputs. Thus the output is comprised of 4 bits. There are additional 4 status flags, which are denoted by C (carry flag), Z (zero flag), V (overflow flag), S (sign flag).

C: It is equivalent to the C_{out} of the adder in the ALU. Logical operations do not effect this flag.

Z: This flag is set if the output of the ALU is 0. Otherwise, the flag is not set.

V: If overflow occurs, i.e. the result exceeds the range of the used bits, v flag is set. Usually, V is determined by the following formula.

$$V = C_3 \oplus C_{out} \tag{1}$$

Here C_3 is the carry produced in the 3rd bit addition.

S: It is equivalent to the leftmost bit of the output. For a signed number, this bit is the sign bit.

2 Problem specification with assigned instructions

Design a 4-bit ALU with three selection bits CS_0, CS_1 and CS_2 for performing the following operations:

Control Signals			Functions	Description
CS_2	CS_1	CS_0		
0	0	0	Add with carry	$A + B + 1$
0	0	1	Transfer A	A
0	1	X	Subtract	$A - B$
1	0	0	XOR	$A \oplus B$
1	0	1	Decrement A	$A - 1$
1	1	X	Increment A	$A + 1$

Table 1: Control signals and their corresponding operations.

3 Detailed design steps with k-maps

3.1 Design steps

1. One MUX has been used for determining X_i. The selection bit of this MUX selects from A_i and $A_i \oplus B_i$.

2. To determine Y_i, two MUX's have been used. One MUX selects from B_i and B_i' and the other selects from the output of the first MUX and 1.

3. To implement the V flag, it is observed that the overflow flag requires previous carry from the adder which can not be directly found from a adder IC. So the adder is split into 2 parts where one adder adds only the 3 bits and another adder only adds the MSBs. The carry generated in the first adder is used as the C_{in} for the second adder. The overflow flag is then determined from $C_3 \oplus C_{out}$.

4. The output bits are denoted by F_i. Z flag was implemented by the following formula: $F_0' \wedge F_1' \wedge (F_2 \vee F_3)'$.

5. The C and S flags were determined from C_{out} and F_3 respectively.

3.2 K-Maps

Equations that we obtained without K-maps are:

$$en_1 = 0$$

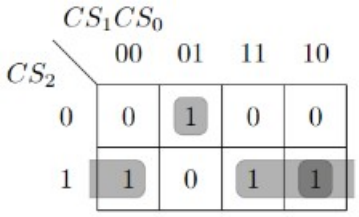
$$S_1 = CS_2\overline{CS_1}CS_0$$

$$S_2 = CS_2\overline{CS_1}CS_0$$

$$S_3 = \overline{CS_2}CS_1$$

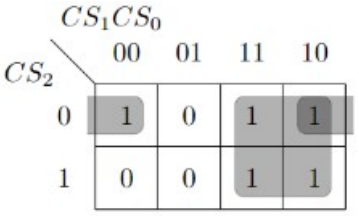
We use the truth table to obtain the following K-maps.

3.2.1 K-Map for en_2



$$en_2 = \overline{CS_2CS_1}CS_0 + CS_2CS_1 + CS_2\overline{CS_0}$$

3.2.2 K-Map for C_{in}



$$C_{in} = CS_1 + \overline{CS_2CS_0}$$

4 Truth Table

CS_2	CS_1	CS_0	Functions	X_i	S_1	Y_i	S_2	S_3	C_{in}	en_1	en_2
0	0	0	Add with carry	A_i	0	B_i	0	0	1	0	0
0	0	1	Transfer A	A_i	0	0	0	0	0	0	1
0	1	X	Subtract	A_i	0	$\overline{B_i}$	0	1	1	0	0
1	0	0	XOR	$A_i \oplus B_i$	1	0	0	0	0	0	1
1	0	1	Decrement A	A_i	0	1	1	0	0	0	0
1	1	X	Increment A	A_i	0	0	0	0	1	0	1

Table 2: The truth table

5 Block diagram

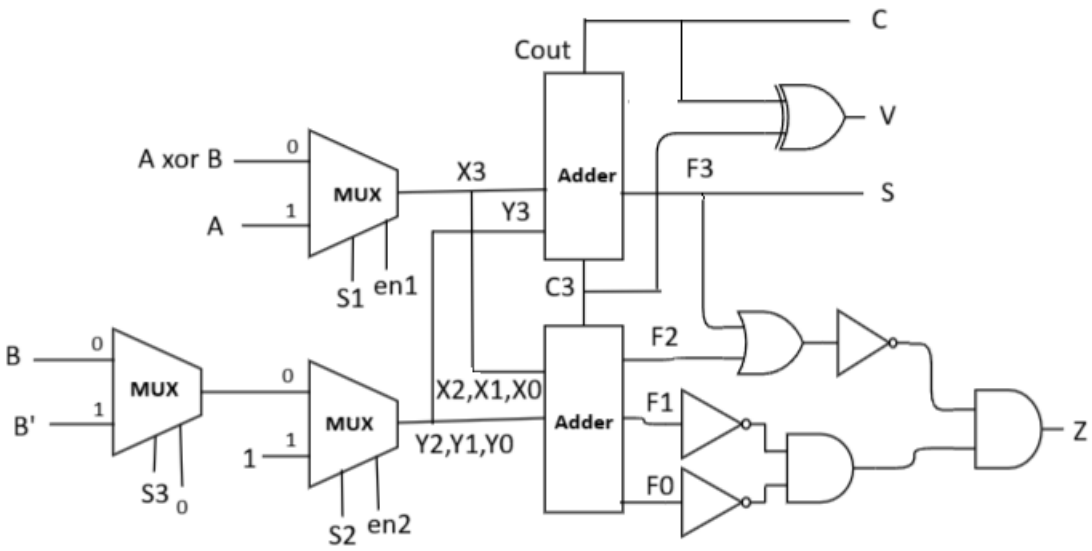


Figure 1: Block diagram of ALU.

6 Complete circuit diagram

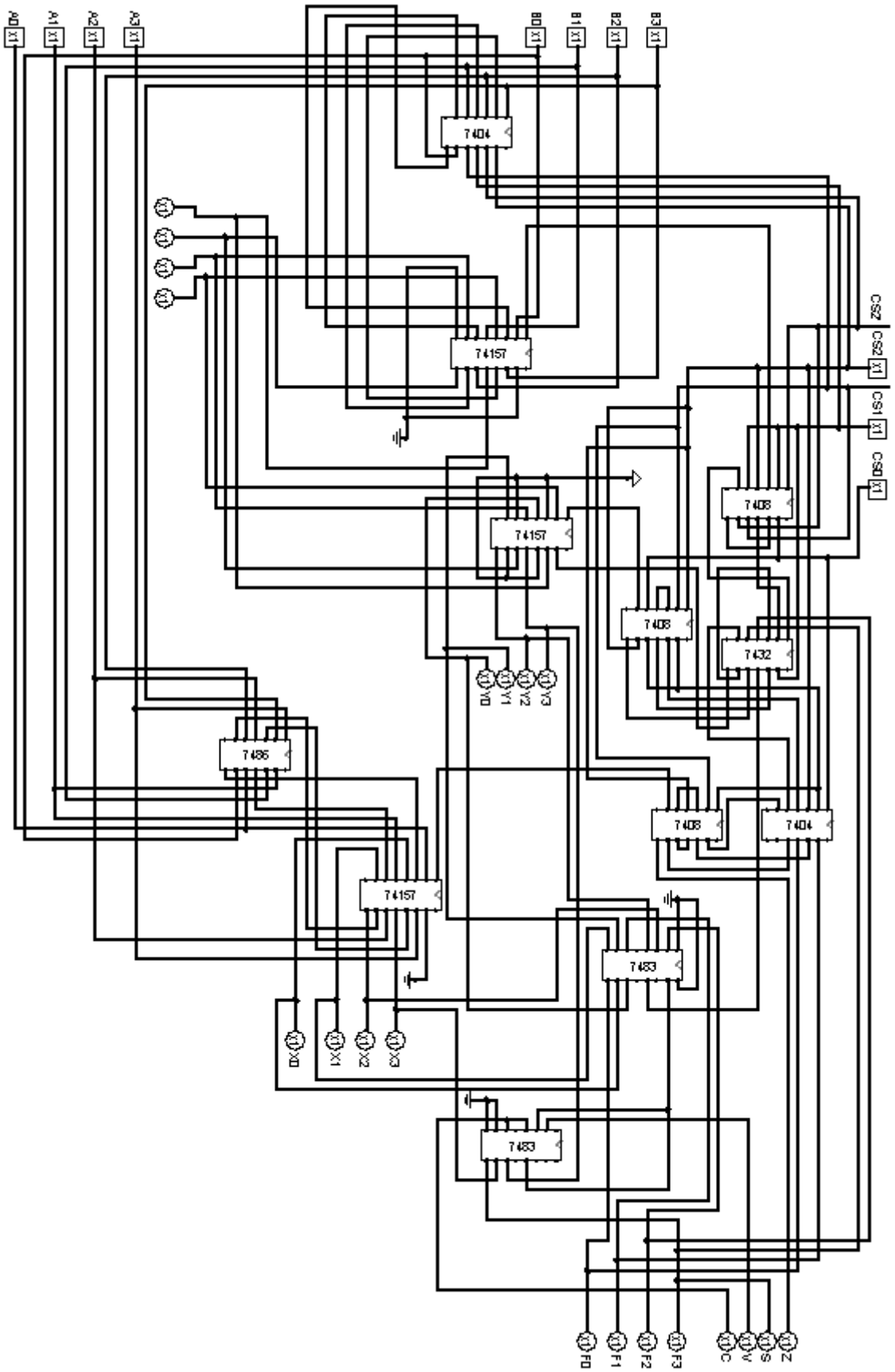


Figure 2: Complete circuit diagram of ALU.

7 ICs used with count as a chart

IC	Quantity
IC 7404	2
IC 7408	3
IC 7432	1
IC 7483	2
IC 7486	1
IC 74157	3
Total	**12**

Table 3: ICs used with quantity

8 The simulator used along with the version number

Logisim 2.7.1

9 Discussion

1. Efficient design choices were made throughout the process to minimize the number of ICs. For example, one xor operation was done with the help of an empty place in an adder, costing one less 7486 IC. Expressions of the flag, enable, and selection bits were modified using laws and theorems to minimize the number of operations.

2. In many cases, the outputs were obtained in such a way that an intermediate value is used in another part, reducing the use of both IC and wire. The process of designing the ALU was iterated multiple times to obtain the most efficient design.

3. After the software implementation, it was thoroughly tested multiple times by group members to avoid errors. The hardware implementation was only initiated after being assured that the software implementation was working properly.

4. The hardware components were tested before using them and handled carefully. The wire connections were made cautiously and within the correct pins of the ICs.

5. Each IC was given power and grounded properly. The switches were also properly grounded so that the ALU receives proper input. If switches are not grounded correctly, the input gets noisy, and low voltage can be received as high voltage, resulting in errors. Sometimes the amplifier inside the input side of the IC also can facilitate these errors.

6. LEDs were connected in the circuit with the help of resistors so that any excessive flow of current does not harm them.

7. The breadboard was arranged as neatly as possible so that the debugging process becomes easier. Each subpart was tested after its completion. In the end, The full implementation was tested with all possible cases and it was ensured that the ALU produced the correct output for every input under the assigned functions.