CSE 210
Computer Architecture Sessional
Assignment 1: 4-bit ALU Simulation

26 October 2024

Group: 01
Section: A2

Members of the group:

- 2105041 - Ahnaf Tahmid
- 2105050 - Dipit Saha
- 2105051 - Ruwad Naswan
- 2105052 - Md. Mehedi Hasan
- 2105056 - Shah Mohammad Abdul Mannan

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:

Table 1: Control signals and their corresponding operations.

Control Signals			Functions	Description
CS_2	CS_1	CS_0		
0	0	0	Add with carry	$A + B + 1$
0	X	1	Sub with borrow	$A + B'$
0	1	0	Transfer A	Output is A
1	0	0	Increment A	$A + 1$
1	X	1	OR	$A \cup B$
1	1	X	Decrement A	$A - 1$

3 Truth Table

CS_2	CS_1	CS_0	Functions	X_i	Y_i	C_{in}	S_1	Output
0	0	0	Add with carry	A_i	B_i	1	0	$A_i + B_i + 1$
0	X	1	Sub with borrow	A_i	$\overline{B_i}$	0	0	$A_i + \overline{B_i}$
0	1	0	Transfer A	A_i	0	0	0	A_i
1	0	0	Increment A	A_i	0	1	0	$A_i + 1$
1	X	1	OR	$A_i \cup B_i$	0	0	1	$A_i \cup B_i$
1	1	0	Decrement A	A_i	1	0	0	$A_i - 1$

Table 2: The truth table

4 Detailed Design Steps with K-Maps

4.1 Design Steps

1. One 2-to-1 MUX (for each bit) has been used for determining X_i. The selection bit is $CS_0 \cap CS_2$. This selection bit selects from A_i and $A_i \cup B_i$.

2. To determine Y_i, we will use a 4-to-1 MUX (for each bit). The selection bits are CS_0 and CS_2.

(a) Case 1: $CS_0 = 0$ and $CS_2 = 0$
If $CS_1 = 0$ then $Y_i = B_i$
If $CS_1 = 1$ then $Y_i = 0$
Thus, input $= B_i \wedge CS_1' = (B' \vee CS_1)'$

(b) Case 2: $CS_0 = 0$ and $CS_2 = 1$
If $CS_1 = 0$ then $Y_i = 0$
If $CS_1 = 1$ then $Y_i = 1$
Thus, input $= CS_1$

(c) Case 3: $CS_0 = 1$ and $CS_2 = 0$
If $CS_1 = 0$ then $Y_i = B_i'$
If $CS_1 = 1$ then $Y_i = B_i'$
Thus, input $= B_i'$

(d) Case 4: $CS_0 = 0$ and $CS_2 = 0$
If $CS_1 = 0$ then $Y_i = 0$
If $CS_1 = 1$ then $Y_i = 0$
Thus, input $= 0$

3. Using Kmap we get $C_{in} = C_1'C_0' = (C_1 \vee C_0)'$

4. 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 $C_3 \oplus C_{out}$.

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

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

4.2 K-Map

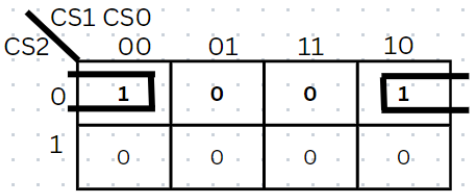

Figure 1: Cin

5 Complete circuit diagram

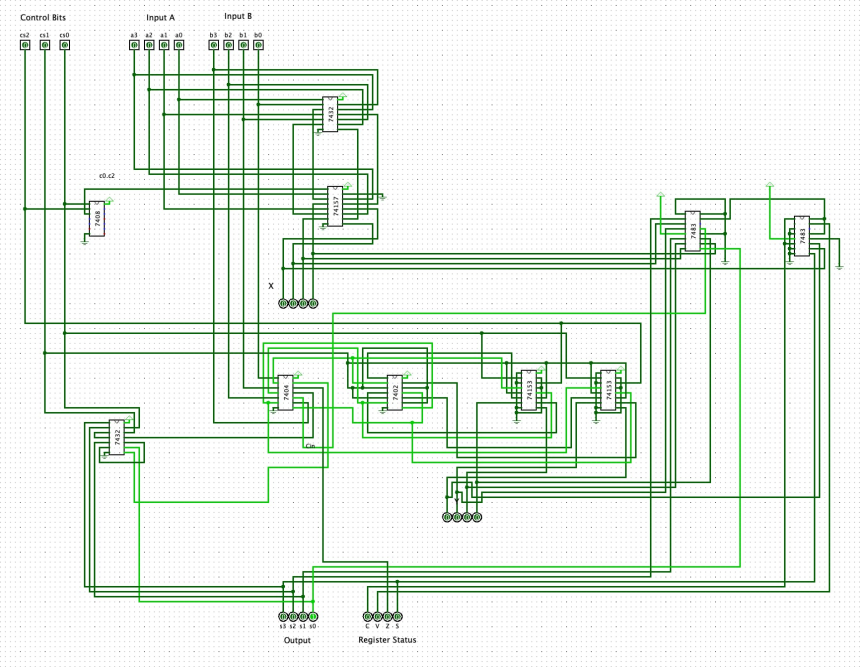

Figure 2: Circuit diagram

6 Block diagram

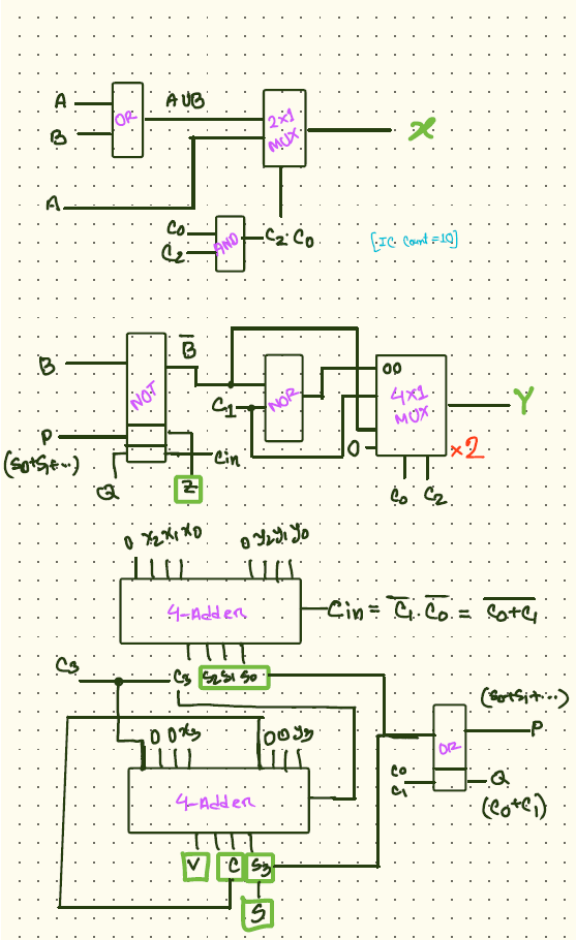

Figure 3: Block Diagram

7 ICs Used with Quantity as a Chart

IC	Quantity
IC 7402	1
IC 7404	1
IC 7408	1
IC 7432	2
IC 7483	2
IC 74153	2
IC 74157	1
Total	**10**

Table 3: ICs Used with Quantity

8 The Simulator Used Along with the Version Number

Logisim 2.7.1

9 Discussion

1. Throughout the design process, careful choices were made to reduce the overall count of ICs used. For instance, an XOR operation was implemented by utilizing an unused section within an adder, eliminating the need for an additional 7486 IC. Logical expressions for control flags, enable signals, and selector bits were simplified using Boolean laws and theorems to decrease the number of required operations.

2. In several instances, outputs were generated strategically so that intermediate values could be reused across different parts of the ALU. This approach minimized both the IC and wiring requirements. The ALU design underwent multiple iterations to arrive at the most resource-efficient structure.

3. The software model of the ALU was extensively tested by the team to verify correct functionality before starting the hardware implementation. This step ensured that any issues could be resolved early, providing a reliable reference for the physical build.

4. All hardware components were pre-tested to confirm they functioned correctly and were handled with care during assembly. Wiring connections were carefully organized to ensure proper alignment with IC pin configurations.

5. Each IC was supplied with power and correctly grounded, and switches were also grounded to ensure clean, stable input signals. Improper grounding can introduce noise, where a low voltage signal may be misinterpreted as high, causing errors. In some cases, internal amplifiers within the IC's input stage could amplify such errors.

6. LEDs were added to the circuit with resistors to limit current, preventing potential damage from excessive current flow.

7. The breadboard layout was organized as neatly as possible, simplifying the debugging process. Each submodule was tested individually upon completion. Finally, the entire ALU circuit was tested with a comprehensive range of inputs, ensuring that it generated accurate outputs for all expected operations.

10 Contribution

Software implementation:

Implementation: 2105056
Testing and Cross-checking: Everyone

Hardware implementation:
Hands on: 2105052, 2105056
Monitoring and Cross-checking: 2105051,2105041,2105050
Testing : Everyone
Report: 21050041, 2105050, 2105051, 2105052, 2105056