CSE 306
COMPUTER ARCHITECTURE SESSIONAL

Offline 1

DESIGNING A 4-BIT ALU

PREPARED BY

1905061 | Nazmus Sakib
1905064 | Sayem Sahad Soummo
1905072 | Mahir Labib Dihan
1905073 | Souvik Ghosh
1905079 | Salman Sayeed

SUBMISSION DATE

December 21, 2022

Contents

1 Introduction

Arithmatic logic unit (ALU) is a special type of combinational circuit that takes a number of inputs and is capabale of doing basic operations which includes both "logical operations" and "arithmatic operations".

An arithmatic logic unit (ALU) takes input from several selection lines. If there are k selection lines, we can do 2^k operations using those lines.

In our experiment, we have used 3 selection lines CS_0, CS_1, CS_2. These lines allow us to use $2^3 = 8$ operations.

As input lines we use 4bits each for inputs A and B. Our combinational circuit uses these inputs to generate a 4bit output F

A 4bit status register has been used to denote the 4 different flags

1. S(sign)
2. V(Overflow)
3. Z(Zero)
4. C(Carry)

These flags change during the operations below :

- **SF :** 1 when highest order bit of output is 1 otherwise 0
- **OF** 1 when XOR of carry bits C_4 and C_5 is 1 otherwise 0
- **ZF :** 1 when all bits of output is 0 otherwise 0
- **CF :** 1 when output carry is 1 otherwise 0

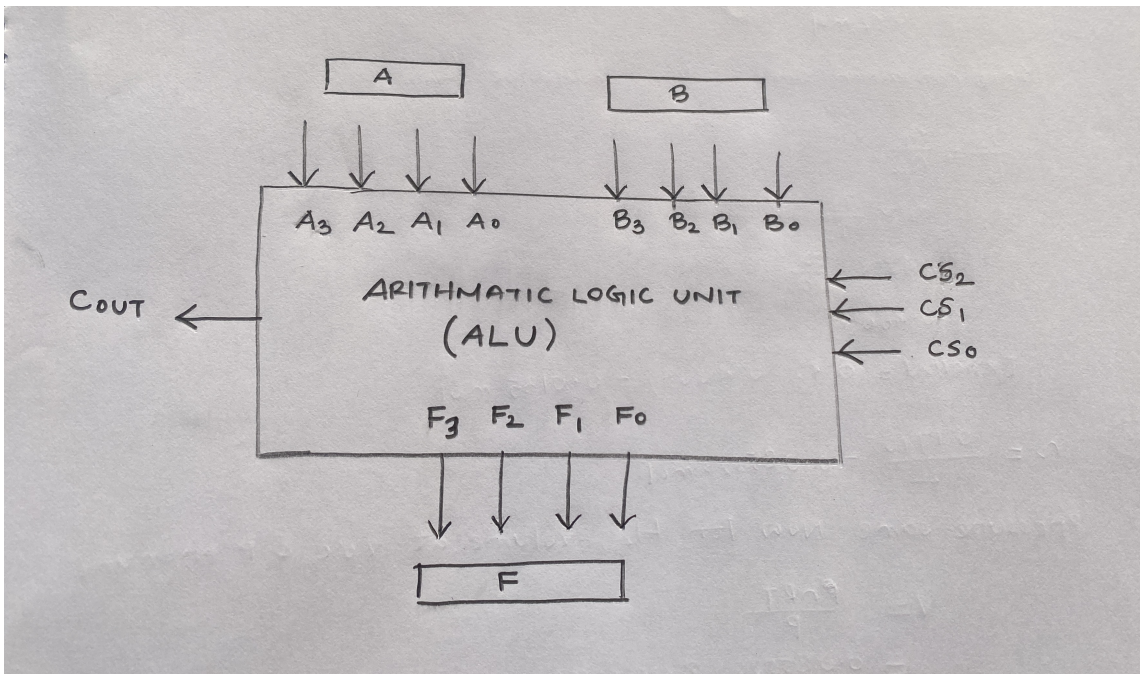


Fig. 1: A block diagram of an ALU with status registers

2 Problem Specification

Design a 4-bit ALU(Arithmatic Logic Unit) with three selection bits CS_0, CS_1, CS_2(S_0, S_1, S_2.) For performing the following operations:

CS_2	**CS_1**	**CS_0**	**Operation**
0	0	0	AND
0	0	1	Decrement A
X	1	0	Sub
0	1	1	Compliment A
1	0	X	Transfer A
1	1	1	Add

Tab. 1: Problem Specification

3 Detailed Design Steps

3.1 Decoder And Terms generation:

Throughout our design, we needed these terms: $D'_0, D_0 + D_3, D_2 + D_6, D_1 + D_7, D'_1 D'_7$

Our decoder was an active low decoder, that means our outputs were $D'_0, D'_1, D'_2, D'_3, D'_4, D'_5, D'_6, D'_7$. After producing these terms, we used nand ic to produce out needed terms. Our nand operations were:
$(D'_0 D'_3)' = D_0 + D_3$
$(D'_2 D'_6)' = D_2 + D_6$
$(D'_1 D'_7)' = D_1 + D_7$
then we used the remaining nand gate to NOT the 3rd term $D_1 + D_7$ and produce the term $D'_1 D'_7$

3.2 Arithmetic Unit:

CS2	CS1	CS0	Operation	Xi	Yi	Cin	Output
0	0	1	Decrement A	Ai	1	0	A+1
0	1	0	Subtraction	Ai	Bi'	1	A+B'+1
1	0	0	Transfer A	Ai	0	0	A
1	0	1	Transfer A	Ai	0	0	A
1	1	0	Subtraction	Ai	Bi'	1	A+B'+1
1	1	1	Addition	Ai	Bi	0	A+B

Tab. 2: Arithmatic Function Table

3.2.1 Y_i Generator:

For four arithmetic operations, we have four different terms for Y_i. Two of them have constant value 0 or 1 , and another two have B_i and B'_i. To select between them, we used CS_1 as selection bit in MUX. then we used XOR gates to xor the output with $D'_1 D'_7$. Ultimately, we got Y_i.

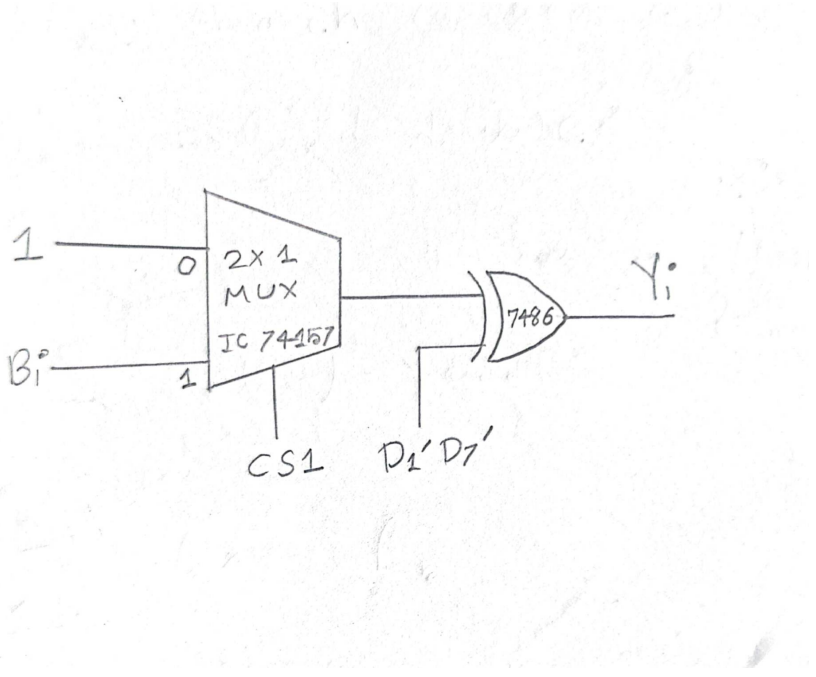


Fig. 2: Y-generator circuit

3.2.2 C_{in} generation:

For our adder, C_{in} is 1 when $D_2 + D_6$ is 1. So, we passed $D_2 + D_6$ as C_{in}

3.3 Logical Unit:

When CS_1 is 0, we do logical AND. Otherwise, we do Complement A. So, we would use a MUX to selection our operation and use CS_1 as selection bit. then our circuit should be:

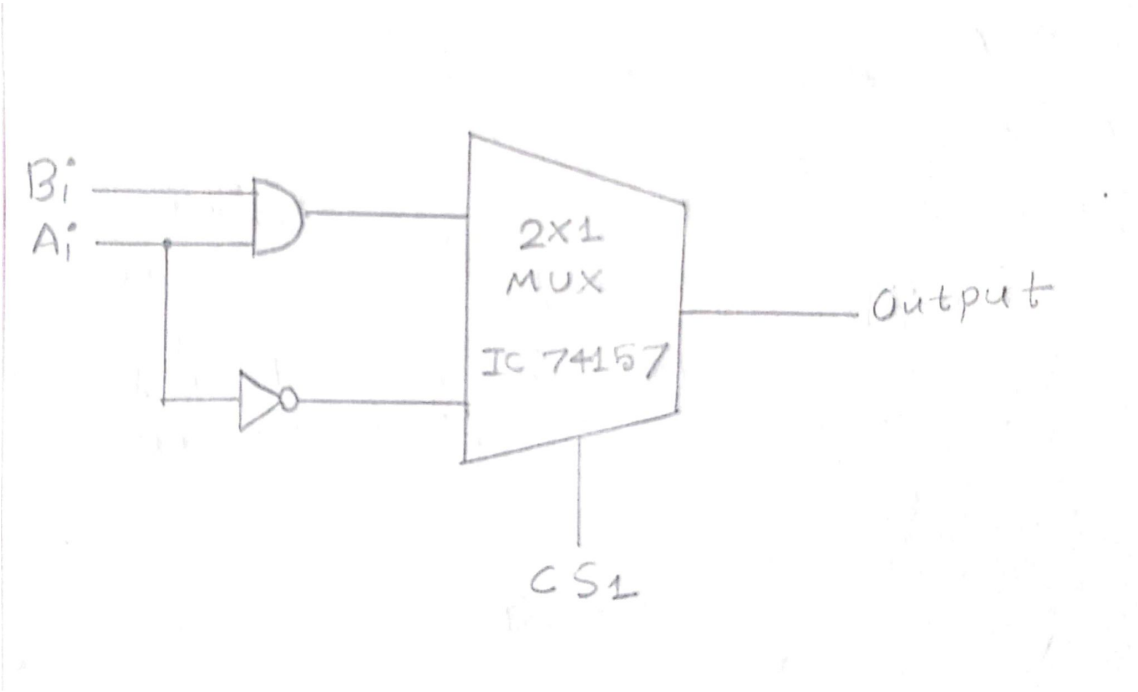


Fig. 3: Un-optimized version of logical unit

To ensure less IC usage, We used inverted MUX. As a result, out efficient design for logical Unit was:

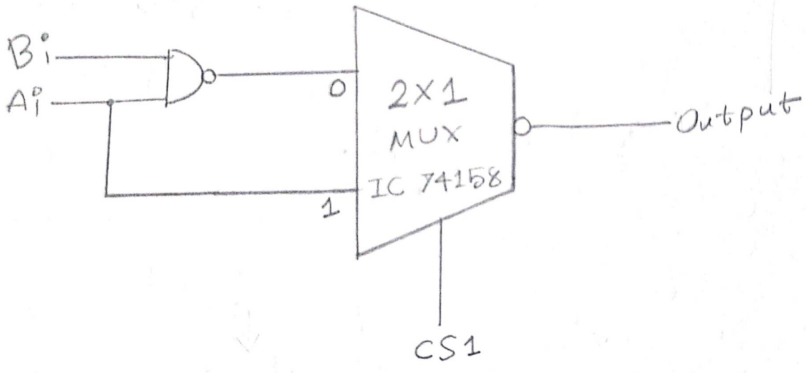


Fig. 4: Optimized version of logical unit

3.4 Flag generator:

Equations for the flags:
$S = F_3$
$Z = (F_3 + F_2 + F_1 + F_0)'$
$C = C_{in} * D_0'$
V= $(X_3$ XOR Y_3 XOR S_3 XOR C_{out} $)D_0'$

For overflow flag, we need C_3 XOR C_{out} but 4 bit full adder can't provide us C_3 . We know, $S_3 = X_3$ XOR Y_3 XOR C_3

So, we can say,
$C_3 = X_3$ XOR Y_3 XOR S_3

As a result,
C_3 XOR $C_{out} = X_3$ XOR Y_3 XOR S_3 XOR C_{out}

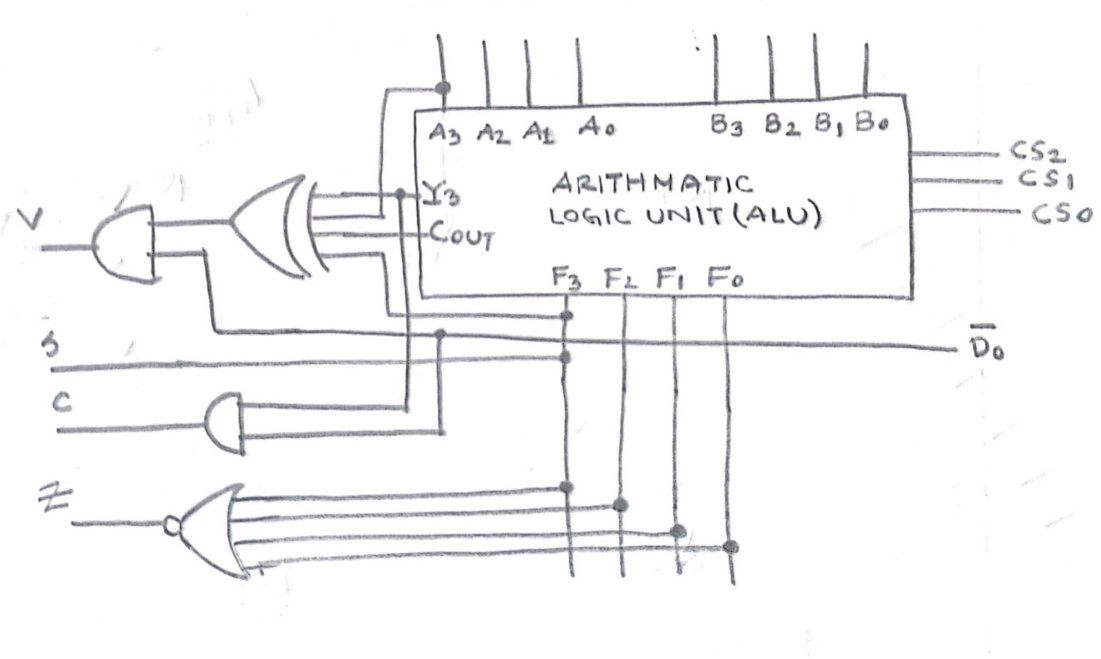


Fig. 5: Flag generator

4 Truth tables

CS2	CS1	CS0	Operation	Xi	Yi	Cin	Output
0	0	0	Logical AND	-	-	-	-
0	0	1	Decrement A	Ai	1	0	A+1
0	1	0	Subtraction	Ai	Bi'	1	A+B'+1
0	1	1	Complement A	-	-	-	-
1	0	0	Transfer A	Ai	0	0	A
1	0	1	Transfer A	Ai	0	0	A
1	1	0	Subtraction	Ai	Bi'	1	A+B'+1
1	1	1	Addition	Ai	Bi	0	A+B

Tab. 3: Truth Table

5 Block Diagrams

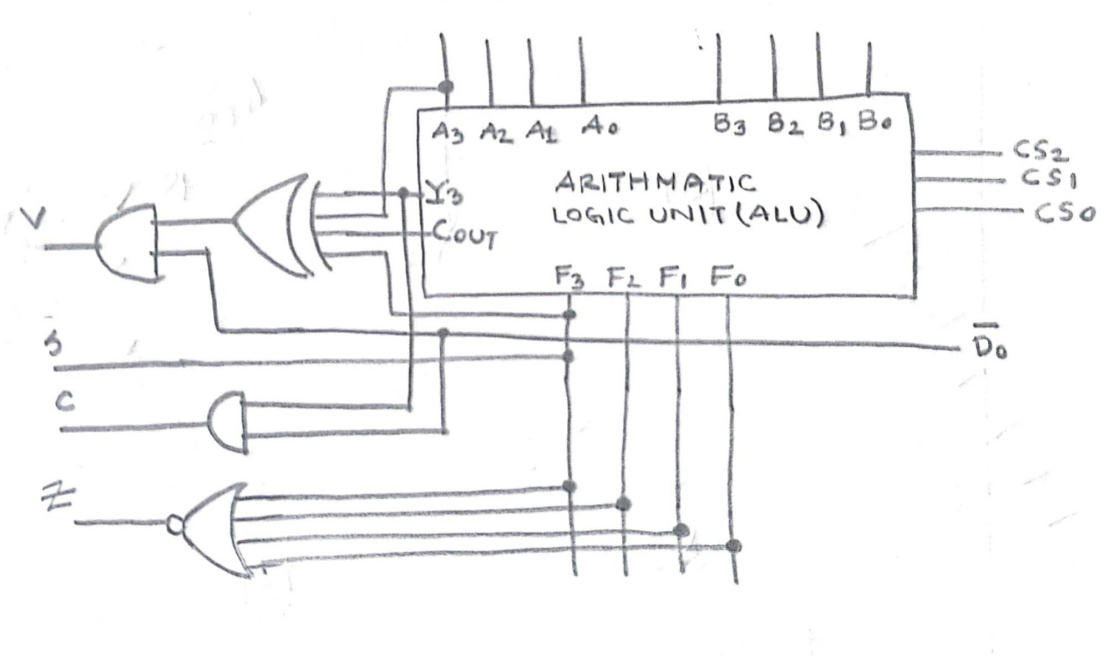


Fig. 6: Status Register

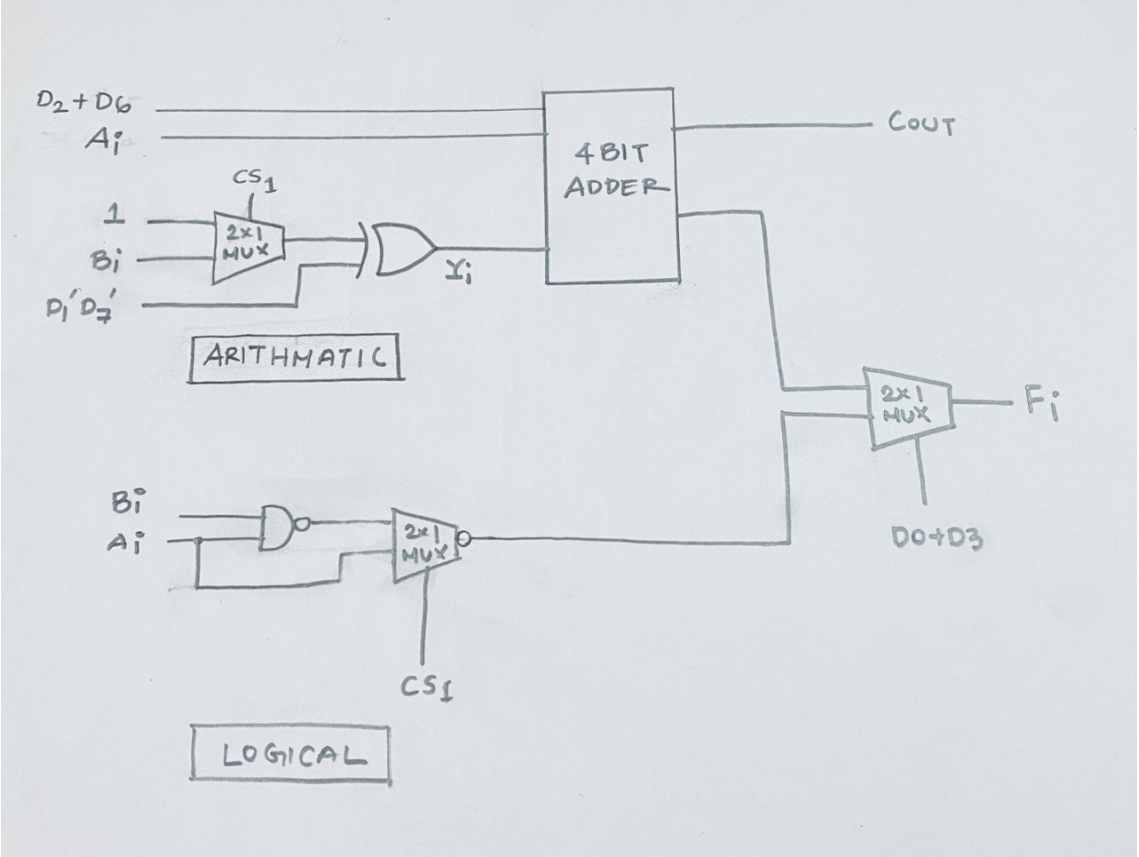
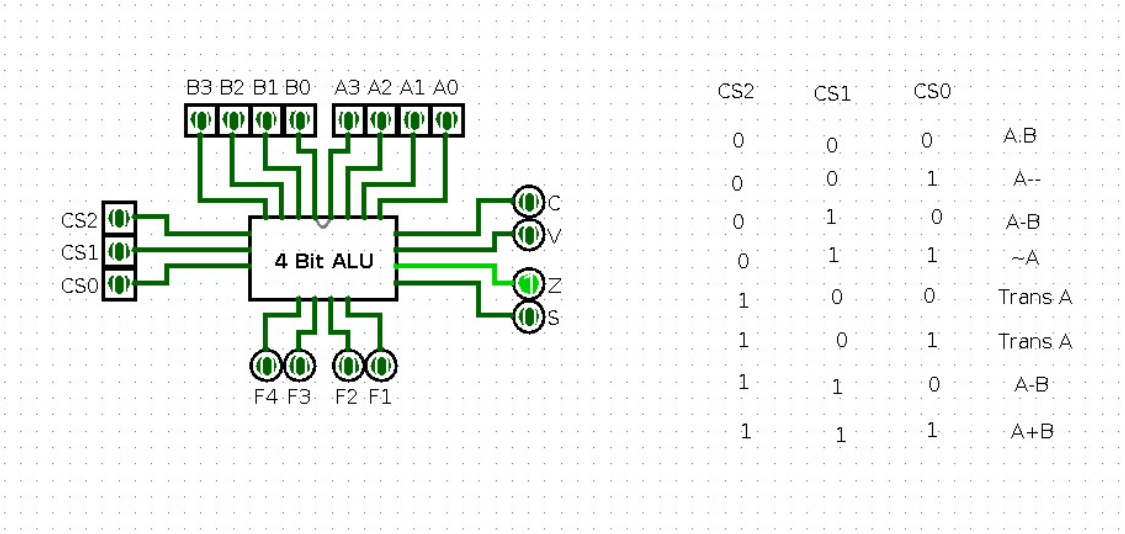


Fig. 7: Multiplexing of arithmatic and logical components

6 Circuit Diagram



CS2	CS1	CS0	
0	0	0	A.B
0	0	1	A--
0	1	0	A-B
0	1	1	~A
1	0	0	Trans A
1	0	1	Trans A
1	1	0	A-B
1	1	1	A+B

Fig. 8: Main ALU Circuit

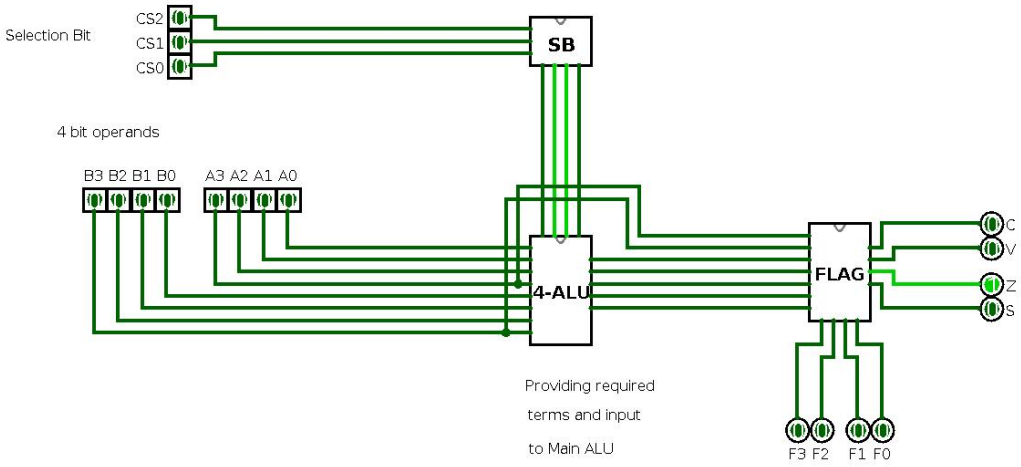


Fig. 9: Arithmatic Logic Unit Flag

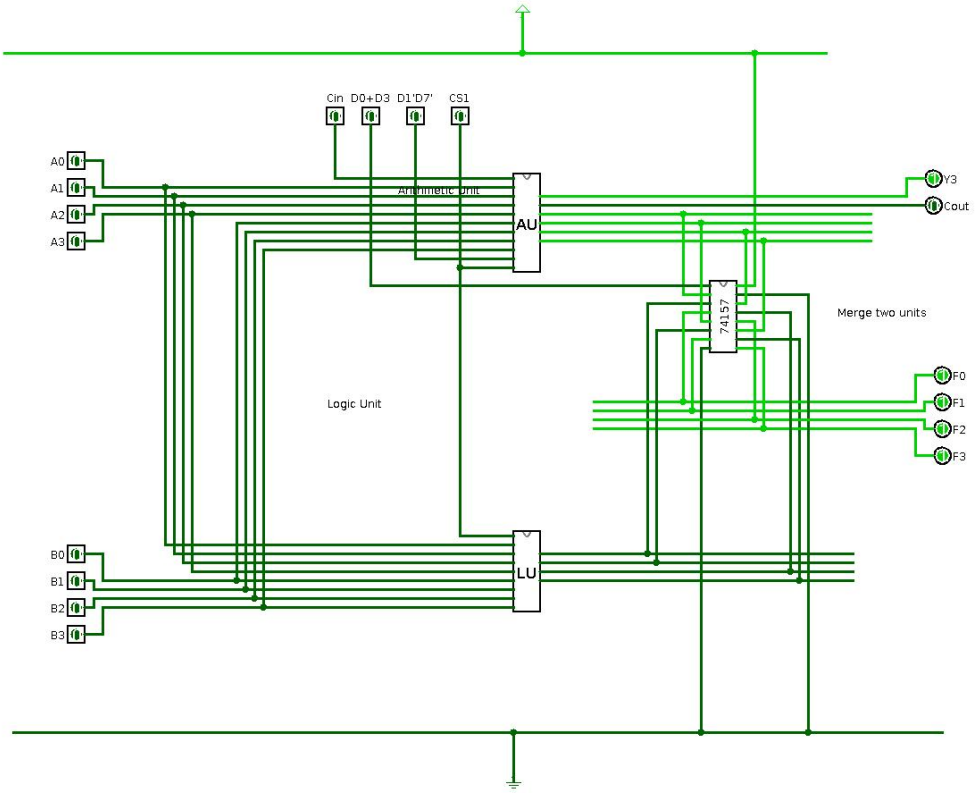


Fig. 10: Arithmatic Logic Unit

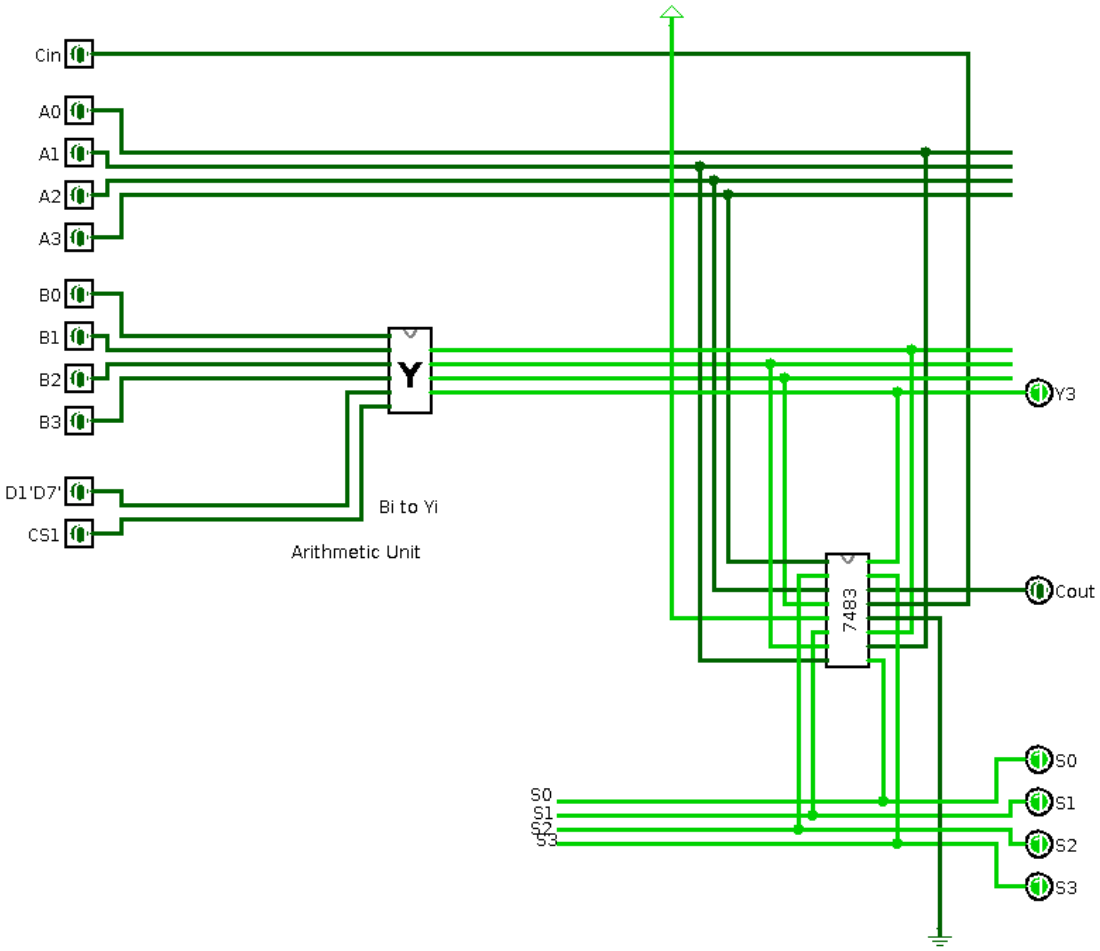


Fig. 11: Combinational Circuit for Arithmatic Component

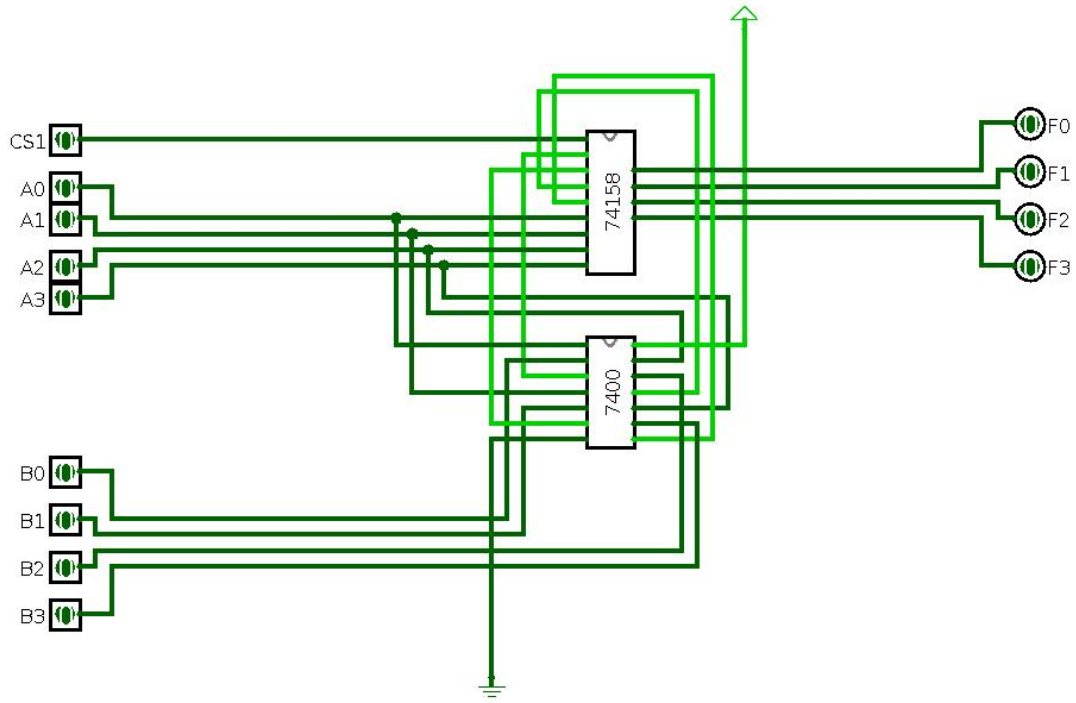


Fig. 12: Combinational Circuit for Logical Component

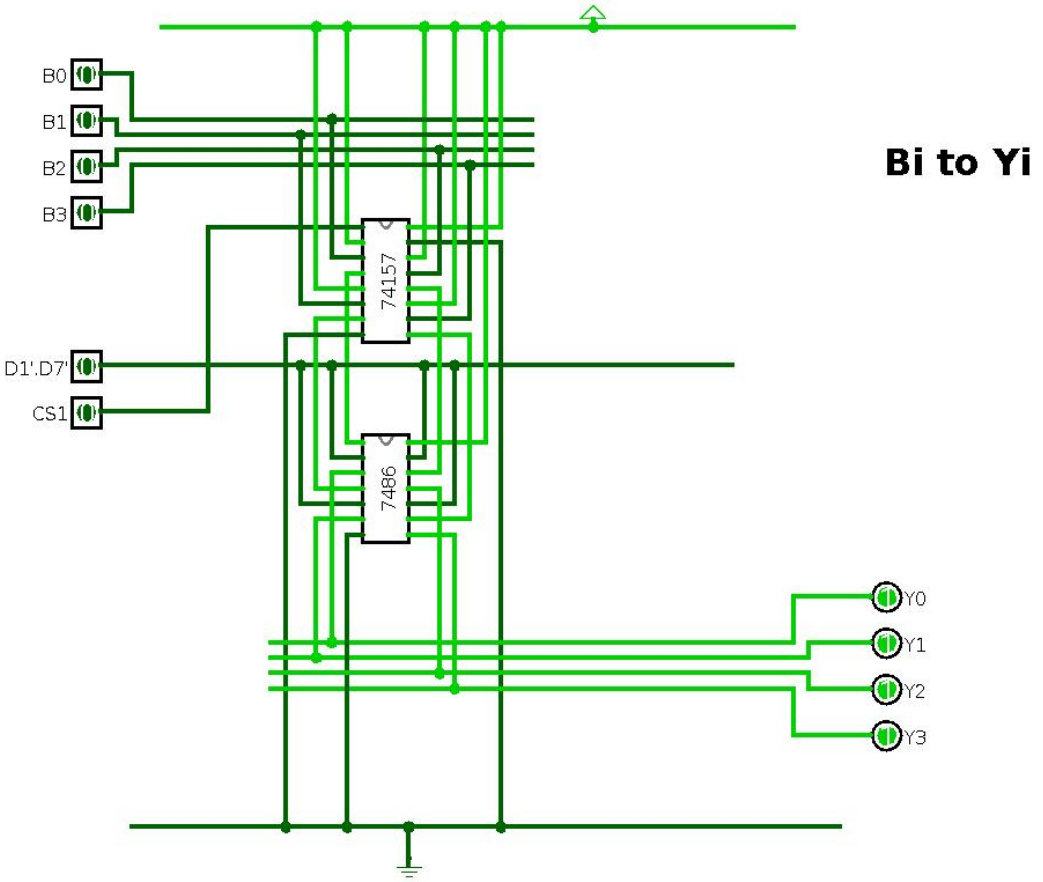


Fig. 13: Combinational Circuit to generate Y_i

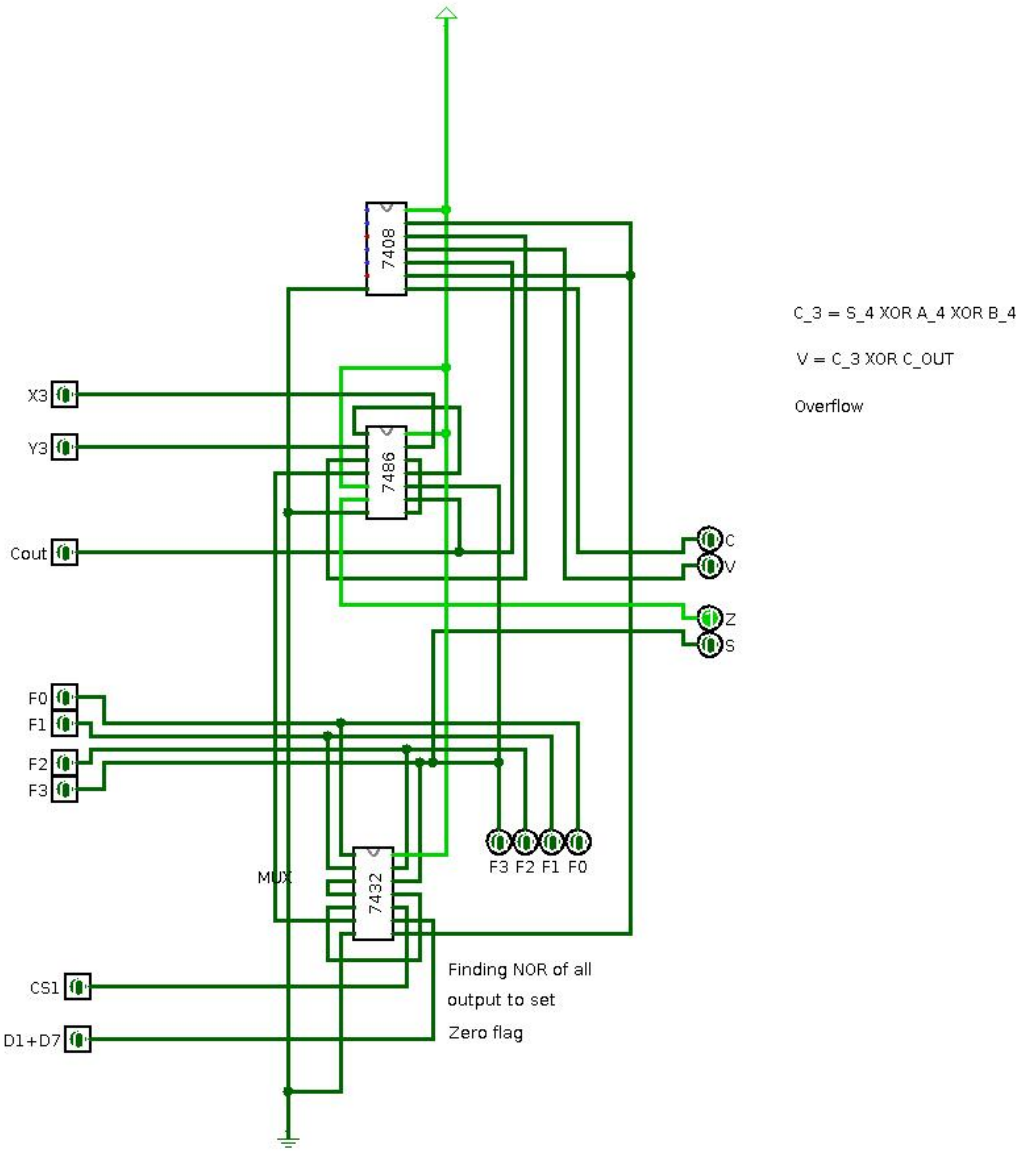


Fig. 14: Combinational Circuit to generate flag

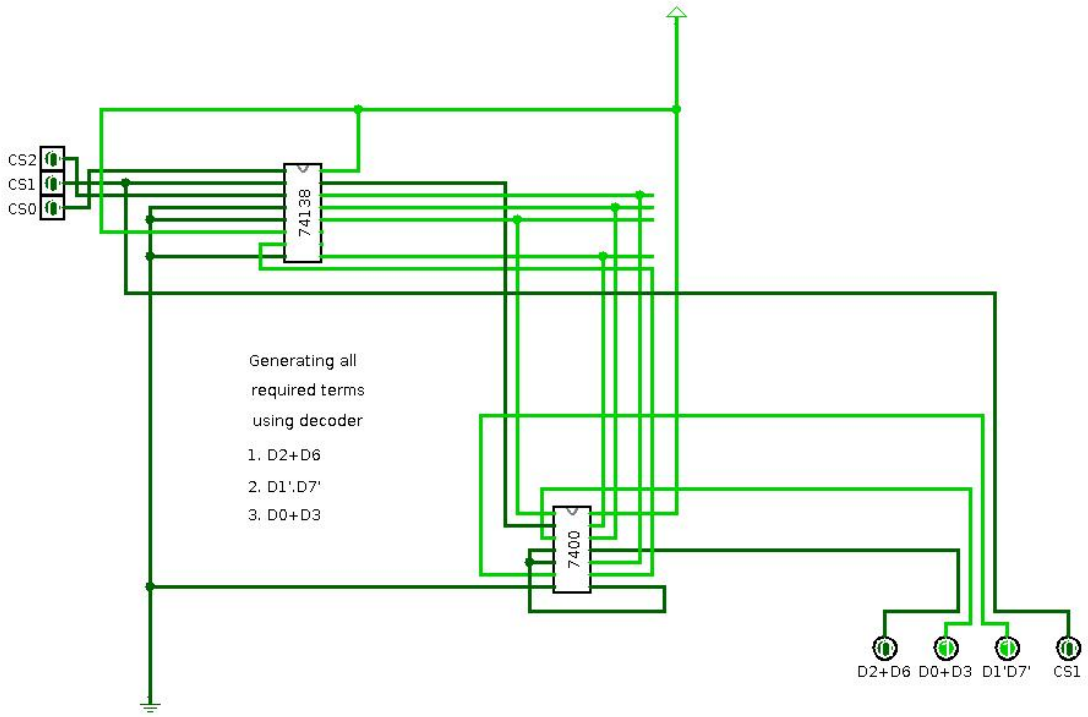


Fig. 15: Combinational Circuit for selection bits

7 IC Diagram

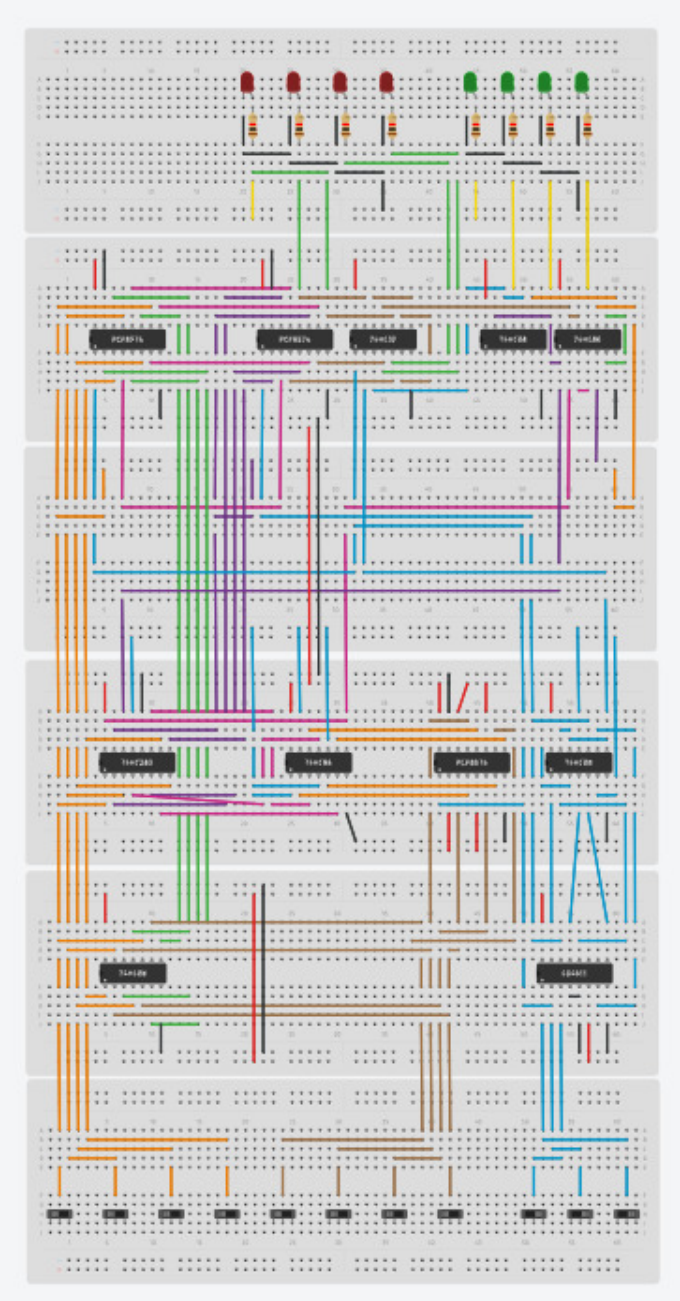


Fig. 16: IC implementation using TinkerCAD

8 ICs used

Here is the list of ICs that have been used during the completion of this experiment

IC Name	IC Count
7400	2
74157	2
74158	1
7483	1
74138	1
7486	2
7432	1
7408	1

Tab. 4: List of ICs used

9 Simulator used

The simulators and software that we used to generate our ALU are :

1. Logisim 2.7.1
2. TinkerCAD

10 Discussion

Our circuit was first designed and simulated in "Logisim". During the design process, we were careful about the ICs that we had used. We also modified the IC 74157 to generate 74158 which was not initially available in the provided library. All the wiring and connections were done carefully and we also modularized the components of the circuit to reduce complexity of the circuits.

During the designing process, we divided our combinational circuit into two components; the logical component and the combinational component for simplifying the composition. We also used decoder to select proper bits at different places

We also used TinkerCAD to design the ALU in-soft to make the task of hardware implementation less tedious. During the TinkerCAD designing process we had prioritized the decluttering of wires and avoided excessive use of wiring.

Finally, we implemented everything using proper ICs. We have used professional wires instead of jumper-cables for better debugging and understanding of the circuit. This also helped us overcome loose wires and other component-related issues to greater extent.

Both our software and hardware implementations ran without any bugs during the testing. Thus it can be concluded that the experiment of implementing a 4-bit ALU using three selection bits was a success.