4-bit ALU Design

CIE 239 - Project Report

Mahmoud Mohammed - 202200313
Ahmed Yasser - 202200477

I - Introduction

*A. **P**roject **O**verview*

The project aims to design and implement a 4-bit Arithmetic and Logic Unit (ALU) that has inputs A & B to do the operations listed in Table 1.

Operation	Output (E)
OR	$E = A \vee B$
AND	$E = A \wedge B$
CMP	$E = B'$
ADD	$E = A + B$
SUB	$E = A - B$
XOR	$E = A \oplus B$
ASR	$E_{3:0} = A_3A_3A_2A_1$
INC	$E = B + 1$

Table 1: Operations

The ALU should have the following outputs:

- Output bits E, a 4-bit long output that will contain the results of the operations.
- Output bit V, where V = 1 in case of an overflow.
- Output bit Z, where Z = 1 if E = 0.

The ALU adder module should be based on full adders.

*B. **T**op **L**evel **D**esign and **I**nput/**O**utput **R**elationship*

This section provides a precise technical description of the requirements, focusing on the exact nature and interaction of inputs and outputs. It clarifies the input/output characteristics and their interrelations. The initial design concept, shaped by these input/ output details and additional relevant requirements, is outlined as per the project specifications.

The design features a circuit with three sets of inputs and two sets of outputs, as depicted in Figure 1.

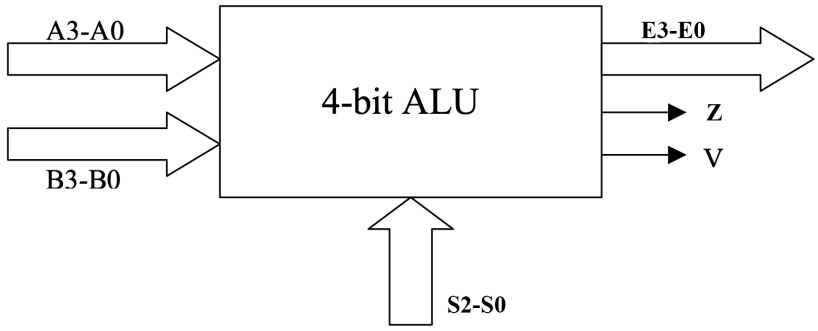

Figure 1. Top-Level Design

As shown in figure 1, the circuit accepts three 4-bit input vectors A, B, and S. More specifically, A and B are 4-bit data input vectors and S is a 3-bit selector input vector. At the output end, the circuit will generate a 4-bit data result as well as two 1-bit status signals Z and V representing the Zero bit and Overflow bit, respectively for the operations.

The choice of three bits for the selector lines is driven by the need to select from eight options, as three bits can encode eight unique states (000 to 111), directly corresponding to each of the eight inputs.

*C. **D**esign **P**rocedure with **B**lock **D**iagrams to **G**et **I**nput/**O**utput **R**elationships*

As the basic requirements and specifications of the problem suggest, the circuit will accepts three sets of inputs (A, B, and S) and giving the values of output (E,Z, and V) accordingly. We can model the the circuit as of a logic function given as f = f(A, B, S) whereas f is the output vector that includes E, Z, and V as a whole. More specifically, we can implement E, Z, and V separately as follows: E = E(A, B, S), Z = Z(A, B, S), and V = V(A, B, S); and f = [E, Z, V]. Doing so, we can approach the task by introducing an initial version of our design of R as given in figure 1.2

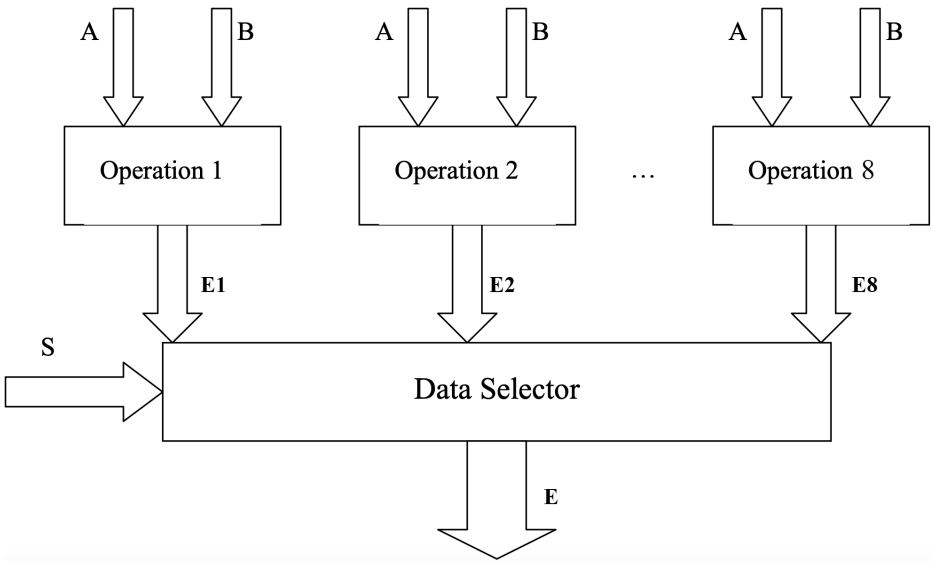

Figure 2. Block diagram to implement data output vector R

As shown in Figure 2, our circuit can comprise 16 operation components , each of which performs one of the 16 operations given by the statement of the problem (see table 1). The basic idea behind this scheme is that because we have 8 operations, we can implement each component as a separate combinational circuit receiving A and/or B as data inputs and giving 8 output vectors E1, E2, ..., E8 respectively, in general sense, at the same time. Because there is only one out of these 8 values is meaningful at a time, the unique bit pattern of S will choose among these the unique correct output vector.

II - Project Breakdown and Characterization of Each Sub-Module

Partitioning of Project into Smaller and Manageable Sub-Modules

*A. **D**esign **B**locks*

Based on the previously mention requirements, the following blocks are needed to build the ALU:

1- Adder-Subtractor block

2- Logic block

3- Increment block

4- Shifter block

5- Selector block (MUX)

Implementation of the Adder-Subtractor block

In our design, we merged the addition and subtraction circuits into a single unit. This approach reduced the number of gates needed, leading to lower power consumption and less space used, making the design more efficient.

The design is based on full adders, as ripple carry adders, The circuit consists of 4 full adders since we are performing operations on 4-bit numbers. There is a control line K that holds a binary value of either 0 or 1 which determines that the operation is carried out is addition or subtraction.

If the value of K (Control line) is 1, the output of B0 ⊕ K = B0′. Thus the operation would be A + (B0′). Now 2's complement subtraction for two numbers A and B is given by A+B'+Cin. This suggests that when K=1, the operation being performed on the four-bit numbers is subtraction. Similarly If the Value of K=0, B0 ⊕ K=B0. The operation is A+B which is simple binary addition. This suggests that When K=0, the operation is performed on the four-bit numbers in addition.

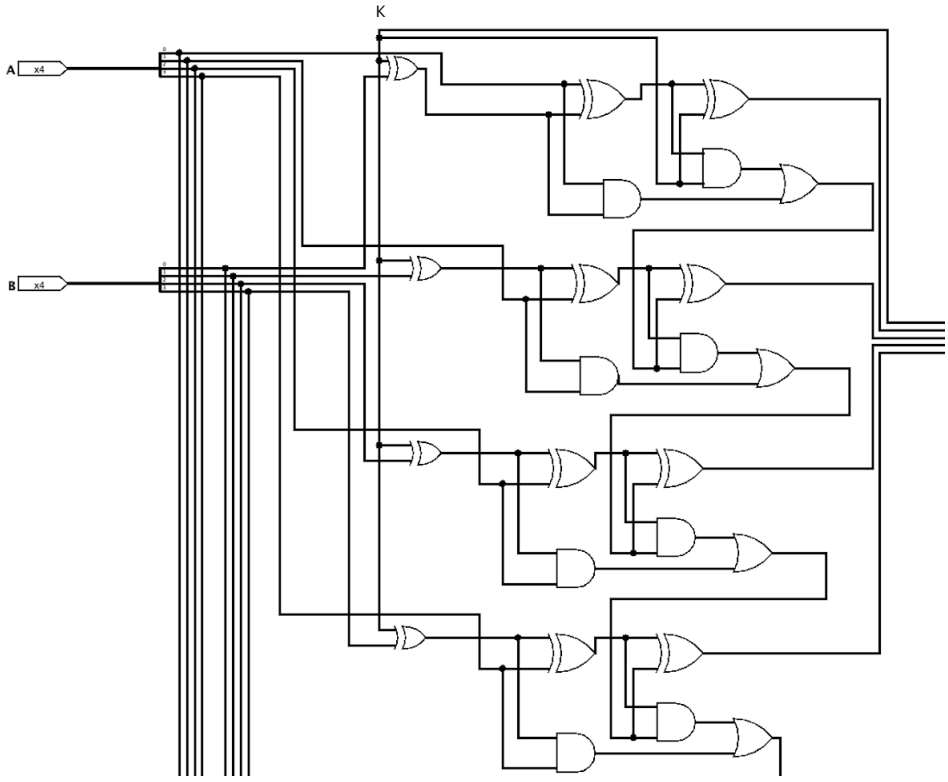

Figure 3. Schematic of the Adder-Subtractor block

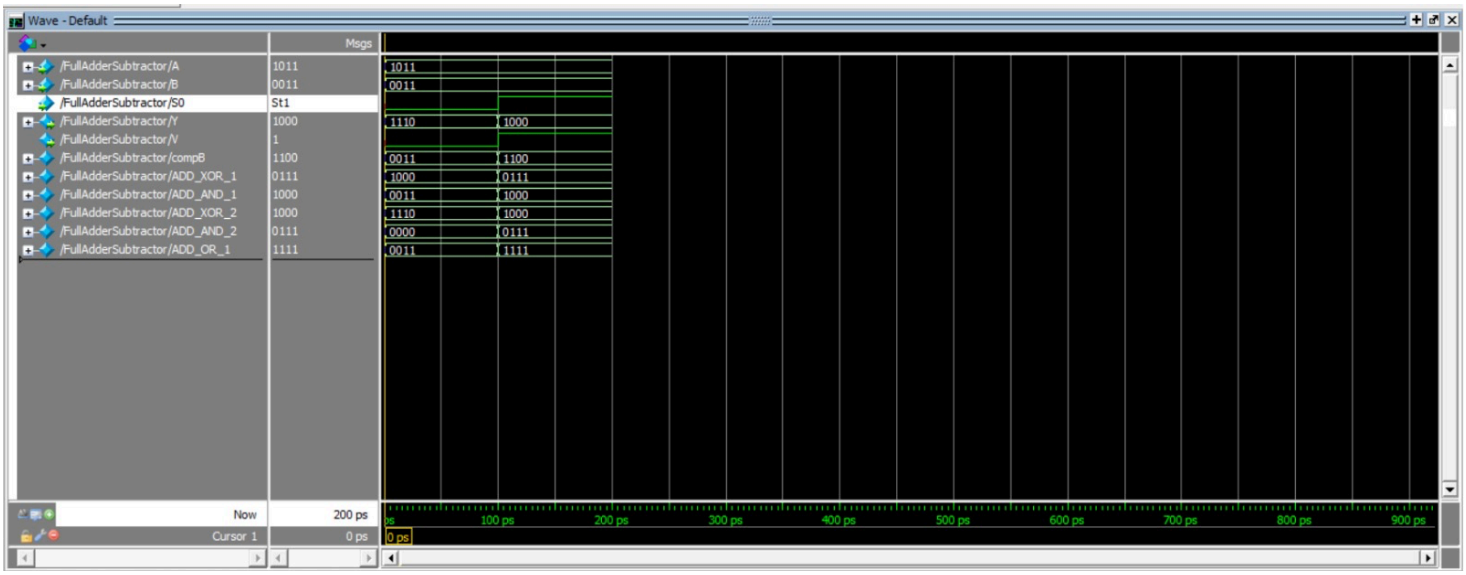

Figure 4. Waveform of the Adder-Subtractor module
First Case - Addition 1011 + 0011 = 1011 (From the figure)
Second Case - Subtraction 1011 - 0011 = 1000 (From the figure)

Implementation of Logic block

The logic block consists of four smaller sub-blocks, OR, AND, XOR and NOT. Outputs are from E3-E6 respectively. Each block takes 2 inputs A & B, except for the NOT block which takes B only as an input.

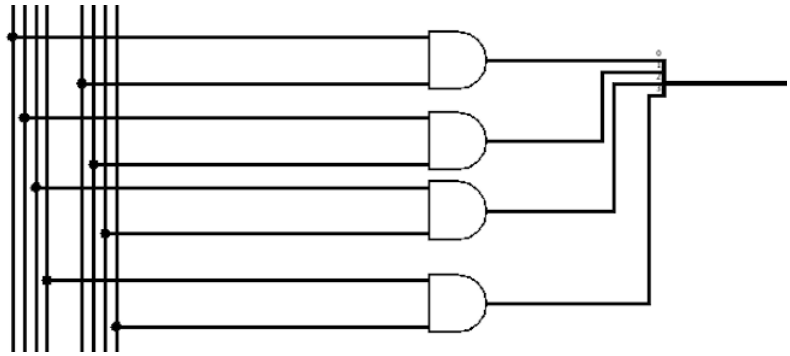

Figure 5. Schematic of the AND block

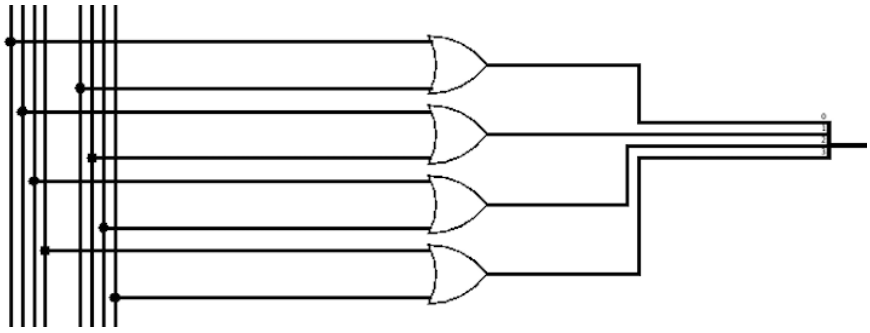

Figure 6. Schematic of the OR block

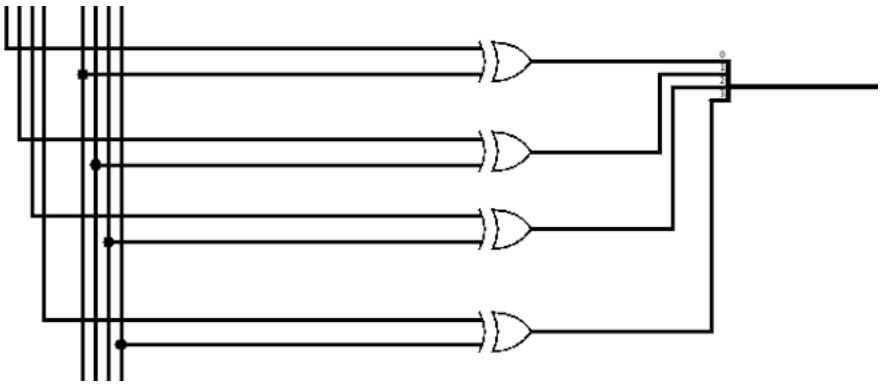

Figure 7. Schematic of the XOR block

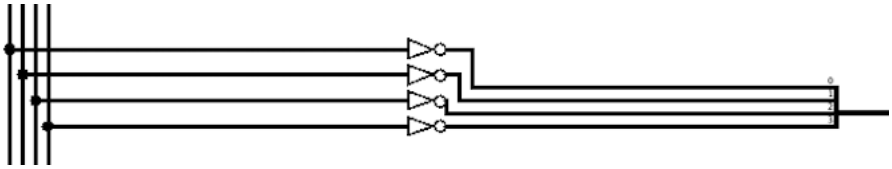

Figure 8. Schematic of the XOR block

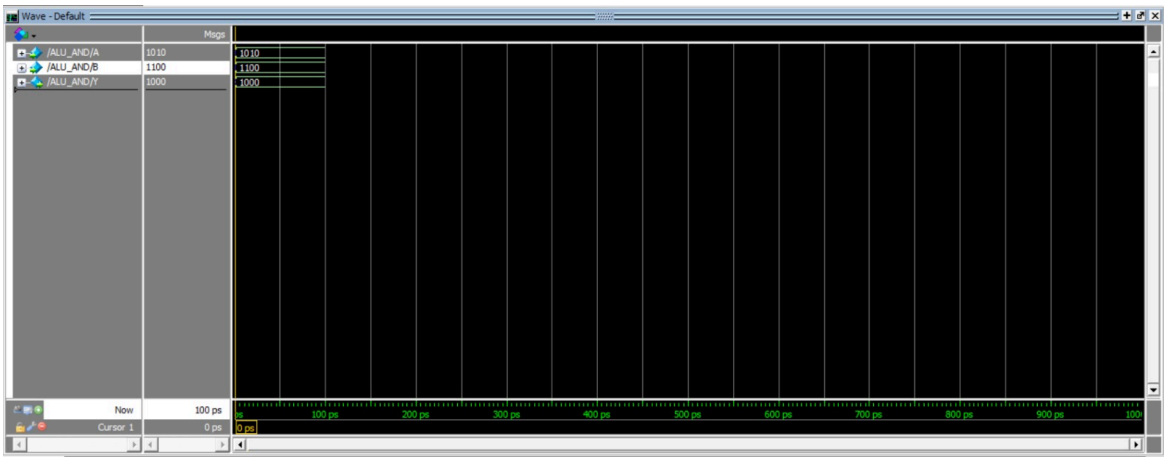

Figure 8. Waveform of the AND block
Case: 1010 AND 1100 = 1000 (From figure)

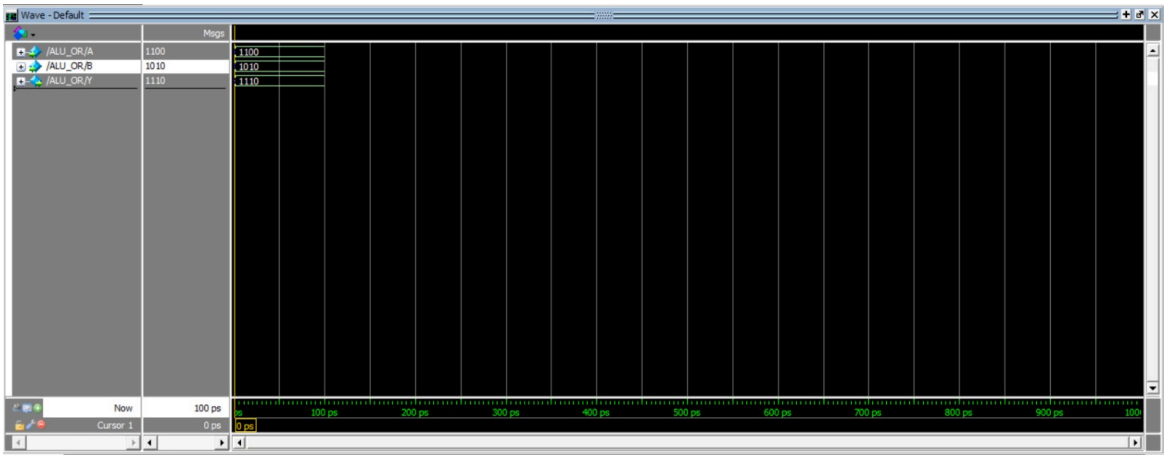

Figure 9. Waveform of the OR block
Case: 1010 OR 1100 = 1110 (From figure)

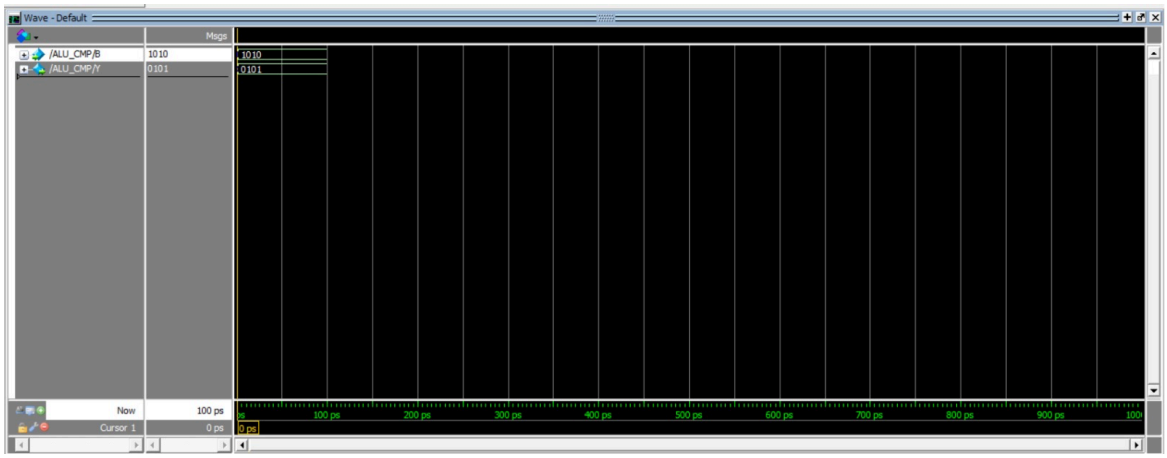

Figure 10. Waveform of the NOT block (CMP B)
Case: NOT (1010) = 0101 (From figure)

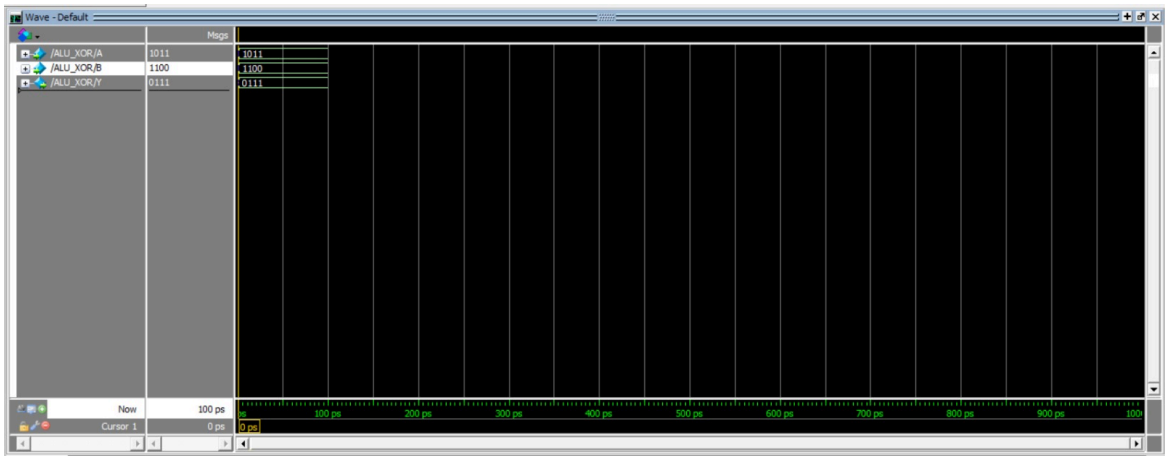

Figure 11. Waveform of the XOR block
Case: 1011 XOR 1100 = 0111

Implementation of Increment Block

The INC block in our design, performing the operation E = B + 1, is constructed using four half-adders arranged sequentially. Each half-adder has two inputs and two outputs. For the least significant bit (LSB) half-adder, or the topmost one, the inputs are '1' and B0 (the first bit of the input), resulting in two outputs: the sum (S0) and the carry (C).

This carry from each half-adder is then fed as an input to the next higher order half-adder. In this cascading manner, the carry output from a lower order half-adder becomes the input for the subsequent one.

Given that the circuit incorporates four half-adders, it processes four bits (B0, B1, B2, B3). By adding '1' to these inputs, the circuit effectively increments the input value.

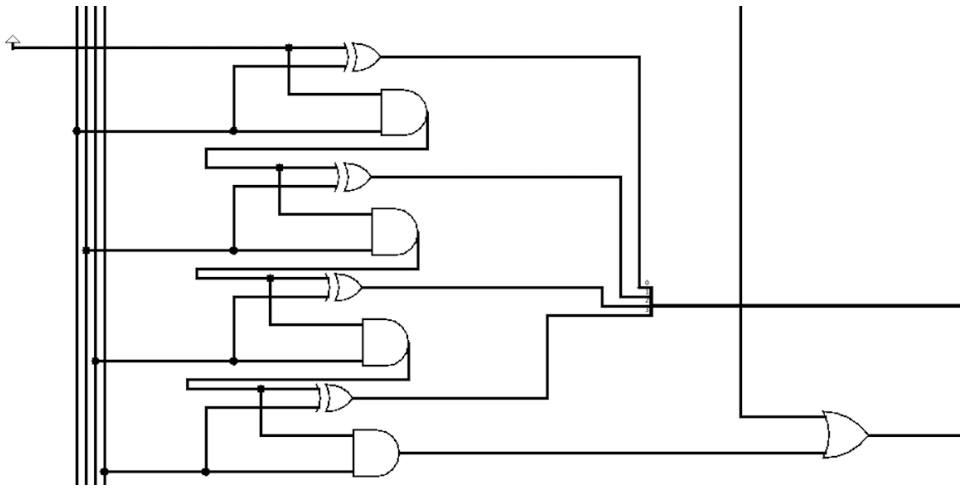

Figure 12. Schematic of the INC block

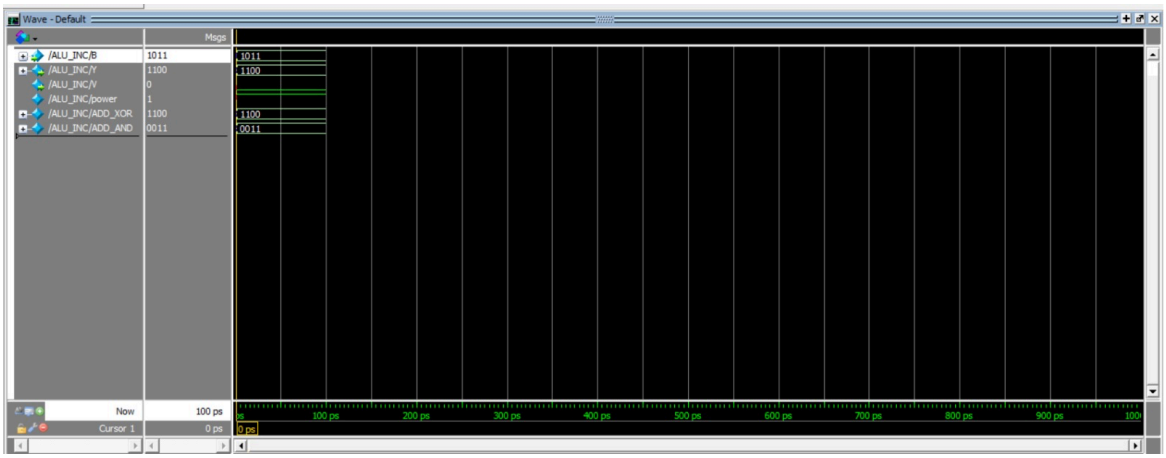

Figure 13. Waveform of the INC block
Case: INC(1011) = 1100

Implementation of Arithmetic Shift block

The arithmetic shift right (ASR) operations happens without the need to use any gates, the operation is achieved by copying the MSB of B (B3) to (E3 & E2), while ignoring the least significant bit.

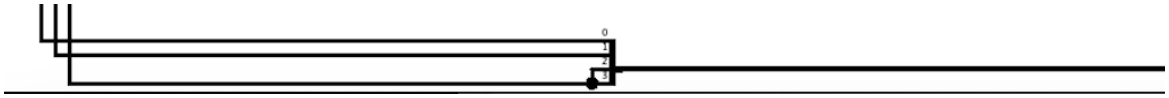

Figure 13. Schematic of the ASR block

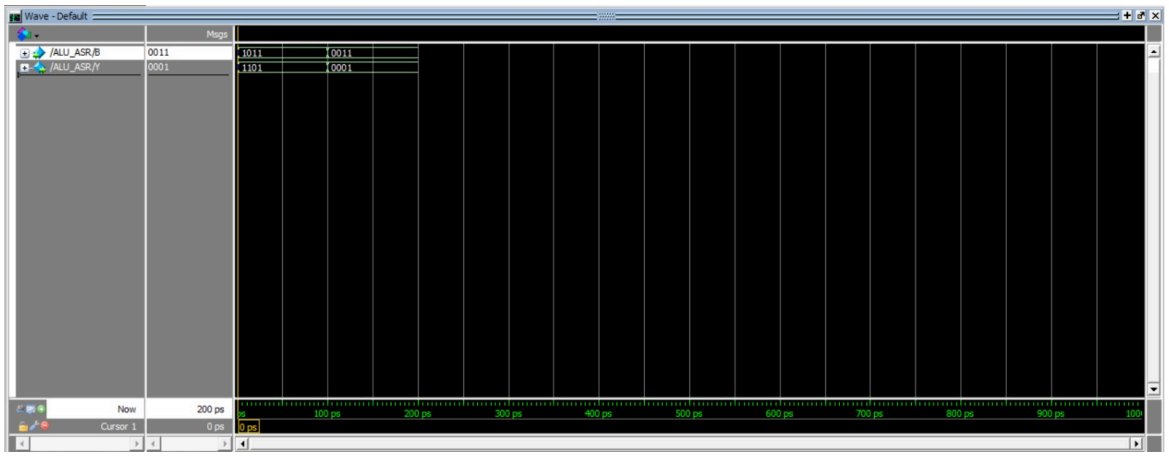

Figure 14. Waveform of the ASR block
Case: ASR(0011) = 0001

Implementation of Selector Block

The ALU utilized an 8 to 1 MUX, with 3 selector lines S2-S0, that corresponds to the following table:

S2	S1	S0	Operation
0	0	0	ADD
0	0	1	SUB
0	1	0	AND
0	1	1	OR
1	0	0	XOR
1	0	1	NOT
1	1	0	INC
1	1	1	ASR

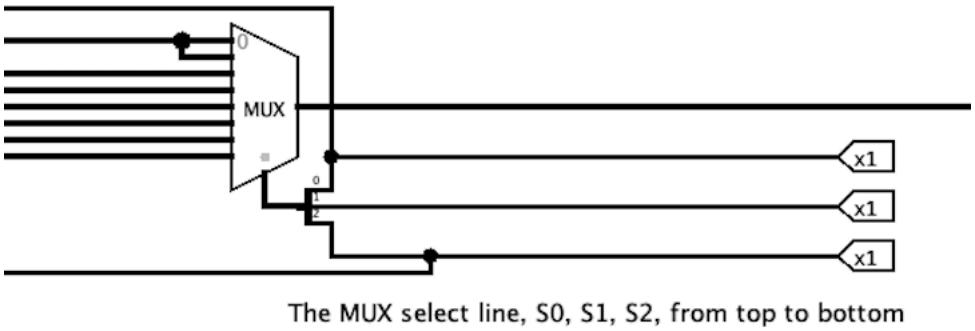

Figure 15. Schematic of the Selector Block

Selector line connections

Selector line **S0** is connected to K, the control line for the Adder-Subtractor block, so it activates subtraction once the SUB operation is selected. Meanwhile, line **S2** is connected to a 2to1 MUX so it selects which carry out signal to use between the ADD operation or the INC operation.

Implementation of ALU Flags

Z flag

The Z flag, which equals 1 if E = 0, is made using a 4-input OR gate connected to a NOT gate, so any non-zero bit of E can turn it into 0.

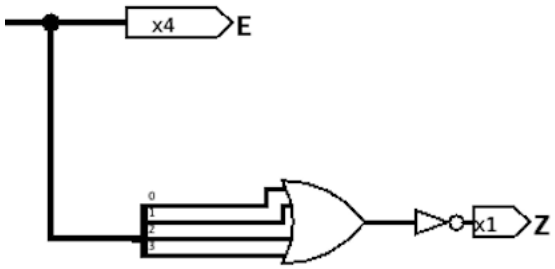

Figure 16. Schematic of Z flag output unit

V flag

This flag, which indicates overflow is an output of a MUX, connected to carry out signals of the ADD and INC operations, S0 of the MUX is S2 of the main 8 to 1 MUX, so it can select which carry out signal to display.

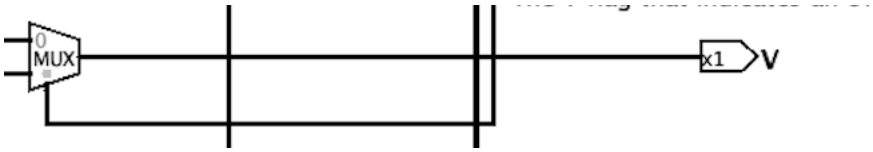

Figure 17. Schematic of V flag output unit

Full scheme overview

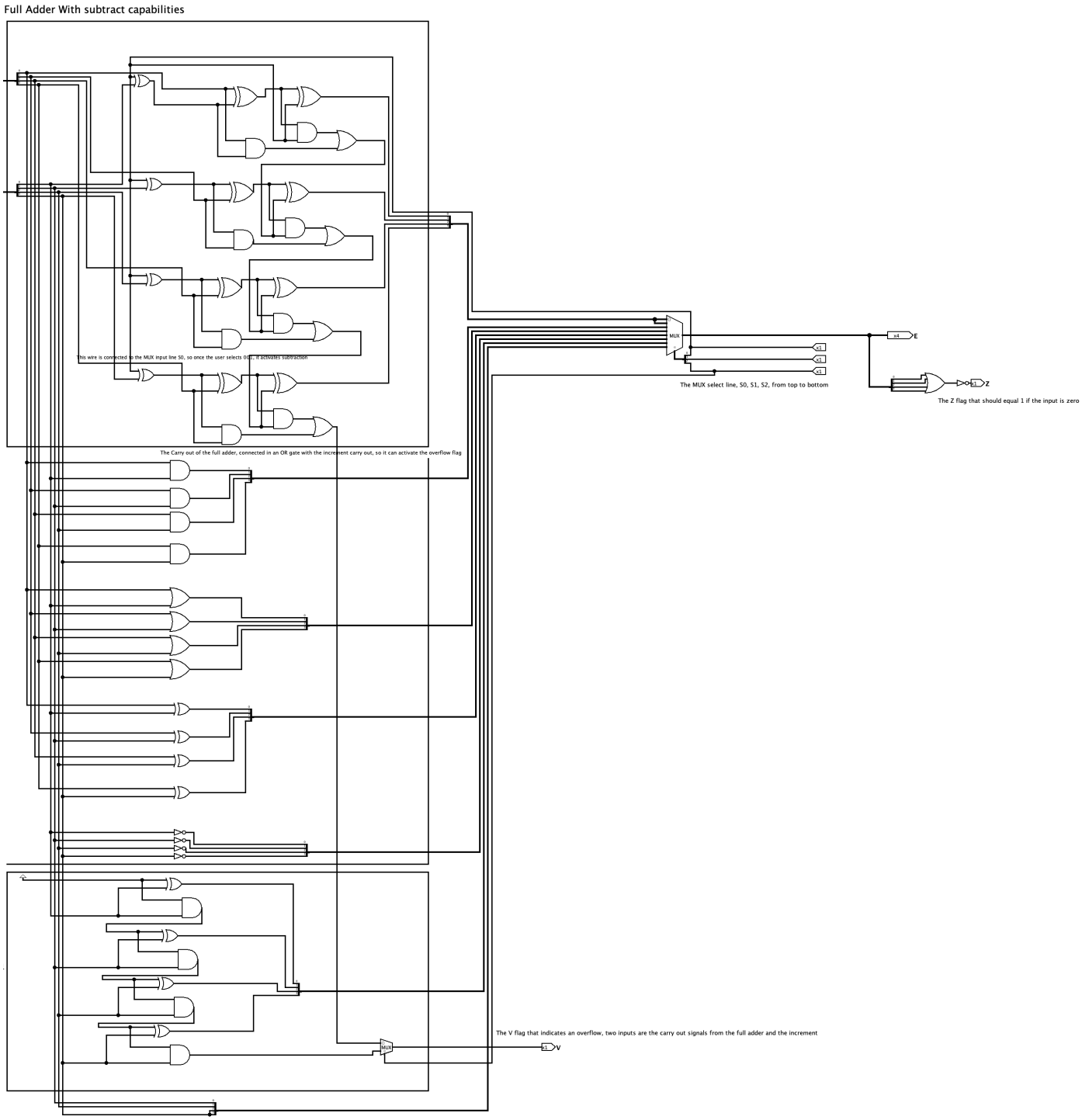

Figure 17. ALU schematic

Test Bench (Bonus)

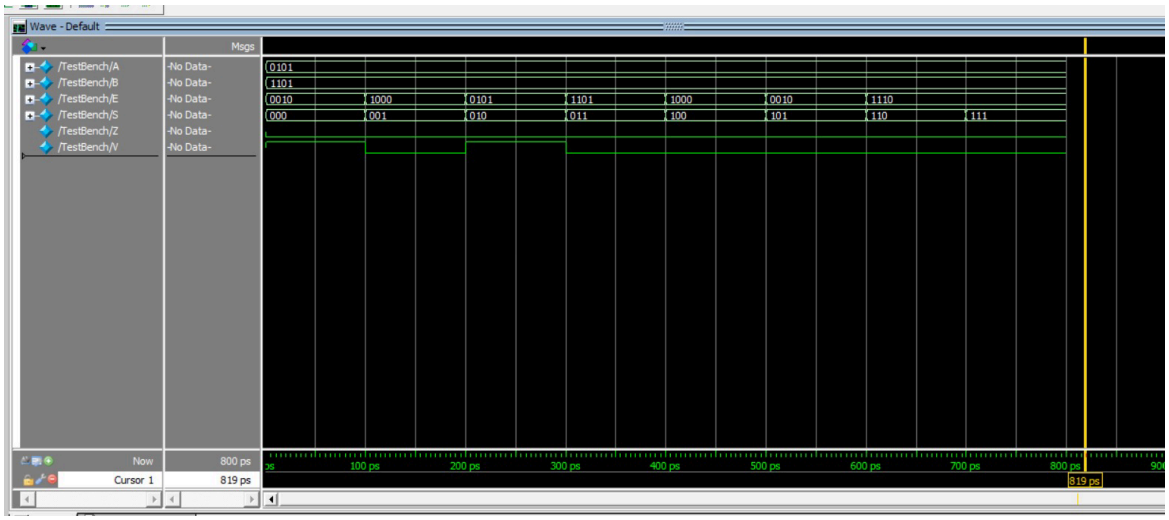

Figure 18. Waveform Test bench