3-bit Arithmetic Logic Unit (ALU) Design and Simulation

Detailed Report

Course: Digital Logic Design
Project: ALU using Logisim and ModelSim
Team Members: [Ahmed Salah], [Mohamed Salah], [Ahmed Akef]
Instructor: [Dr: Amr Hefny]
Date: August 6, 2025

Contents

Chapter 1

Introduction

The Arithmetic Logic Unit (ALU) is a critical component of any digital processing system, including microprocessors, microcontrollers, and digital signal processors. It is responsible for performing arithmetic operations such as addition and subtraction as well as logical operations like AND, OR, XOR, and NOT. In many systems, the ALU is considered the "heart" of the processor as it directly executes the instructions required to process data.

The objective of this project is to design and implement a 3-bit ALU capable of performing a predefined set of arithmetic and logical operations. The design focuses on modularity, where each individual operation is implemented as an independent functional block. These operations are selected through a multiplexer, controlled by a 4-bit selection input ('SEL'), which determines the operation performed on the input operands.

The ALU designed in this project supports the following operations:

1. Addition: $F = A + B$
2. Subtraction: $F = B - A$ (using Two's Complement)
3. Two's Complement: $F = \sim A + 1$
4. Bitwise OR: $F = A|B$
5. Bitwise AND: $F = A\&B$
6. Bitwise XOR: $F = A \oplus B$
7. Bitwise NOT: $F = \sim B$
8. Logical Shift Left: $F = A << 2$
9. Logical Shift Right: $F = A >> 3$
10. Arithmetic Shift Left and Right (Bonus functionality)

In addition to the functional outputs, two flag signals were incorporated to enhance the ALU's usability:

- **Zero Flag (Z):** Indicates when the result of an operation equals zero.
- **Overflow Flag (V):** Indicates overflow during arithmetic operations.

The design and implementation process followed a systematic approach:

- The structural design and block-level implementation were carried out using **Logisim**, which provided an intuitive graphical environment for constructing the digital circuits and verifying the correctness of the functional blocks.
- Simulation and functional verification were performed using **ModelSim** with SystemVerilog-based testbenches. Forcing techniques were used to drive input signals and observe the output behavior.

The main goals of this project are:

1. To demonstrate the ability to design and integrate multiple digital components into a single functional unit.
2. To understand the use of control logic (multiplexer and selection signals) for routing data in digital systems.
3. To verify digital design functionality using industry-standard simulation tools.

This report documents the design methodology, block-level schematics, simulation results, and final observations for the implemented 3-bit ALU.

Chapter 2

ALU Design in Logisim

2.1 Design Overview

The ALU was constructed in Logisim using modular design principles. Each operation was implemented in a separate block:

- 3-bit Adder (ADD)
- 3-bit Subtractor (B - A)
- Two's Complement Generator
- Logical AND, OR, XOR gates
- Bitwise NOT (on B)
- Logical Shift Left and Right
- Arithmetic Shift Left and Right

These blocks are connected through a multiplexer, which selects the output based on a control signal (SEL).

2.2 Flags (Zero and Overflow)

Two additional outputs were added:

- **Zero Flag (Z):** Implemented using a NOT gate connected to the MUX output. This flag is high when the ALU result equals zero.
- **Overflow Flag (V):** Obtained from the carry out (S3) of the 3-bit adder, which indicates signed overflow during addition.

2.3 Schematic Block Diagram

Figure 2.1 shows the overall Logisim schematic.

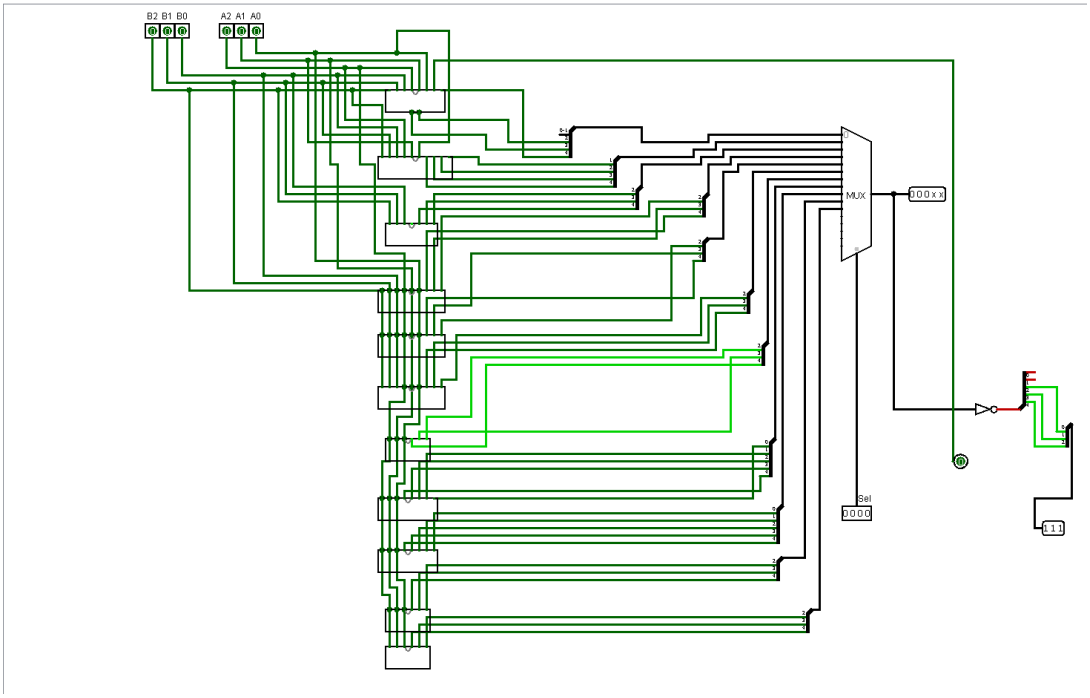

Figure 2.1: Logisim-based ALU Design showing all functional blocks.

Chapter 3

Detailed Module Descriptions

This chapter provides a detailed breakdown of each functional module used in the ALU design. Each module is implemented independently and later integrated into the top-level ALU. Images shown are placeholders for the actual Logisim schematics.

3.1 Full Adder Module

The Full Adder (FA) is the basic building block for performing addition. It takes three inputs: two operands (A, B) and a carry input (C_{in}), and produces two outputs: the sum (S) and carry out (C_{out}). The Boolean equations are:

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = (A \wedge B) \vee (C_{in} \wedge (A \oplus B))$$

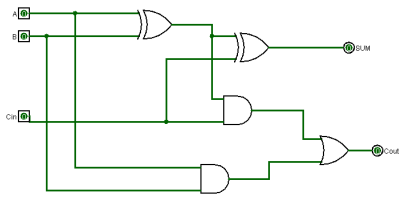

Figure 3.1: Full Adder circuit implemented in Logisim.

3.2 3-bit Ripple Carry Adder

The 3-bit Ripple Carry Adder is constructed using three Full Adders connected in series. The carry output of one adder is connected to the carry input of the next. This allows the addition of two 3-bit numbers, producing a 4-bit result (3-bit sum + carry out).

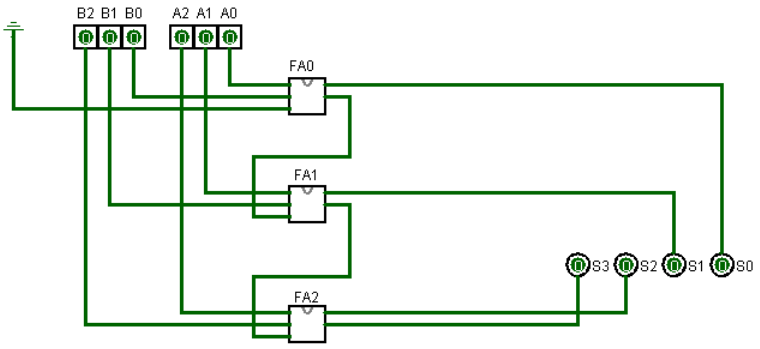

Figure 3.2: 3-bit Ripple Carry Adder using three Full Adders.

3.3 Two's Complement Module

The Two's Complement module inverts all bits of the input and adds one using the previously implemented 3-bit Adder. This module is primarily used for subtraction ($B - A$) and generating the negative equivalent of a value.

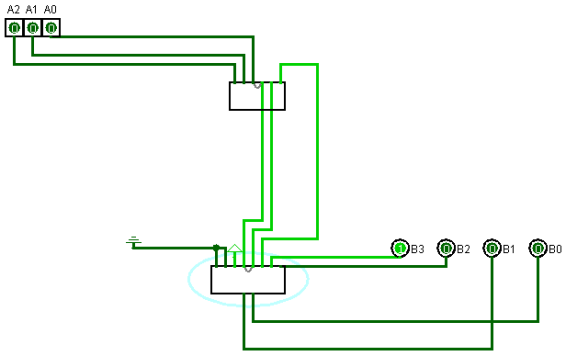

Figure 3.3: Two's Complement generator using bitwise NOT and adder.

3.4 Subtractor Module

The Subtractor module computes $B - A$ by using the Two's Complement of A and adding it to B. It reuses the 3-bit Ripple Carry Adder as its main arithmetic engine.

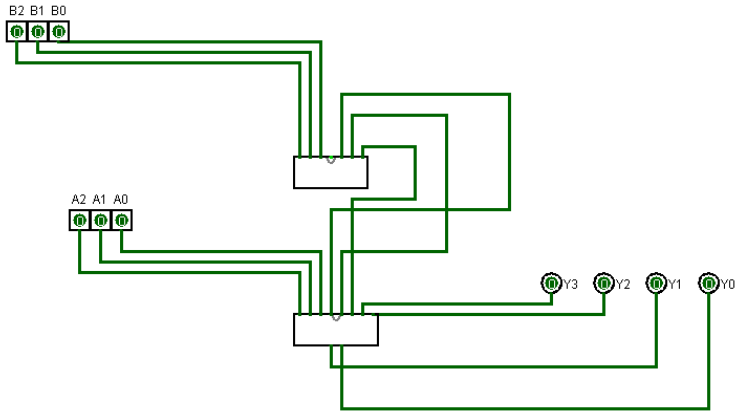

Figure 3.4: 3-bit Subtractor implemented using Two's Complement and Adder.

3.5 Logic Gate Modules (AND, OR, XOR, NOT)

These modules perform bitwise logical operations:

- AND: $F = A \wedge B$
- OR: $F = A \vee B$
- XOR: $F = A \oplus B$
- NOT: $F = \sim B$

Each module is designed using basic two-input gates available in Logisim.

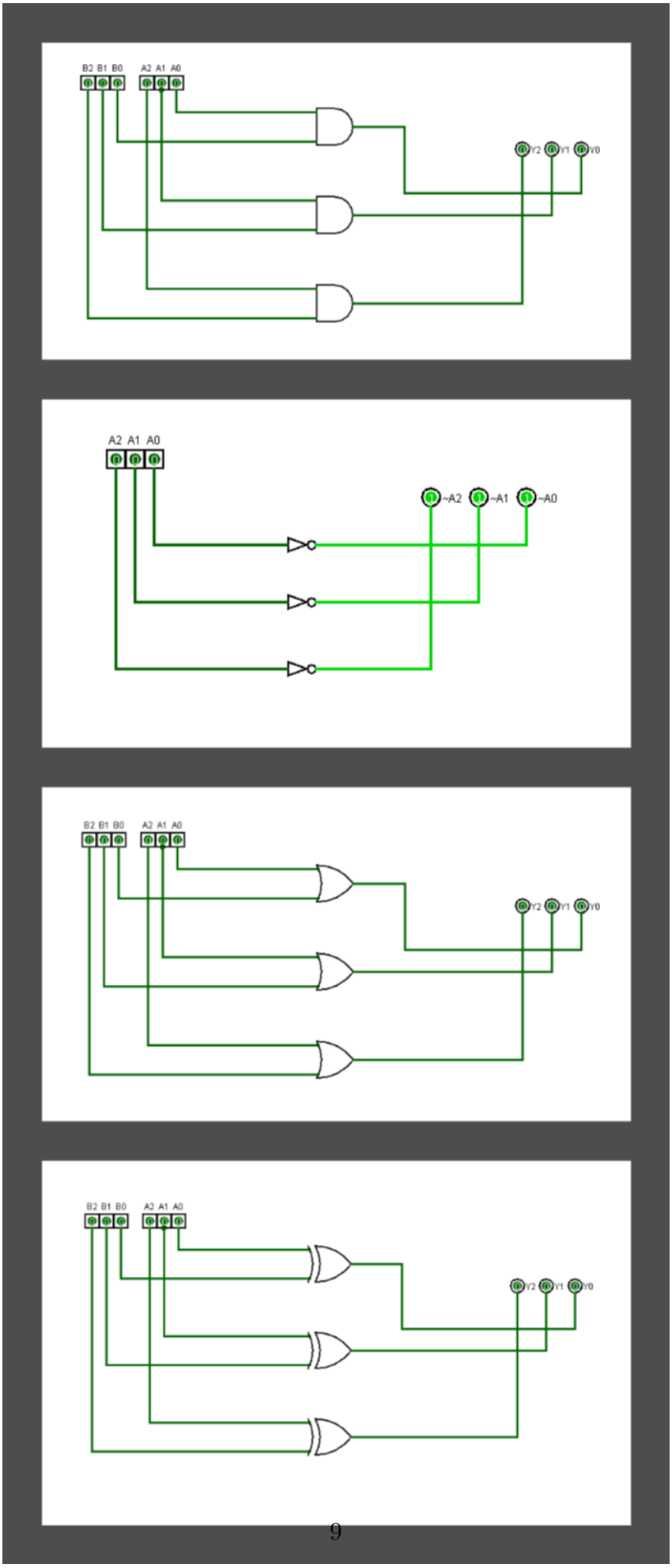

Figure 3.5: Basic Logic Gate Modules used in ALU.

3.6 Shift Modules

Two types of shift modules are implemented:

- Logical Shift Left (LSL): shifts input bits left and fills with zeros.
- Logical Shift Right (LSR): shifts input bits right and fills with zeros.
- Arithmetic Shift Right (ASR): preserves the sign bit during right shift.
- Arithmetic Shift Left (ASL): same as logical shift left for positive numbers.

These shifts are implemented bit-by-bit using direct wire connections in Logisim.

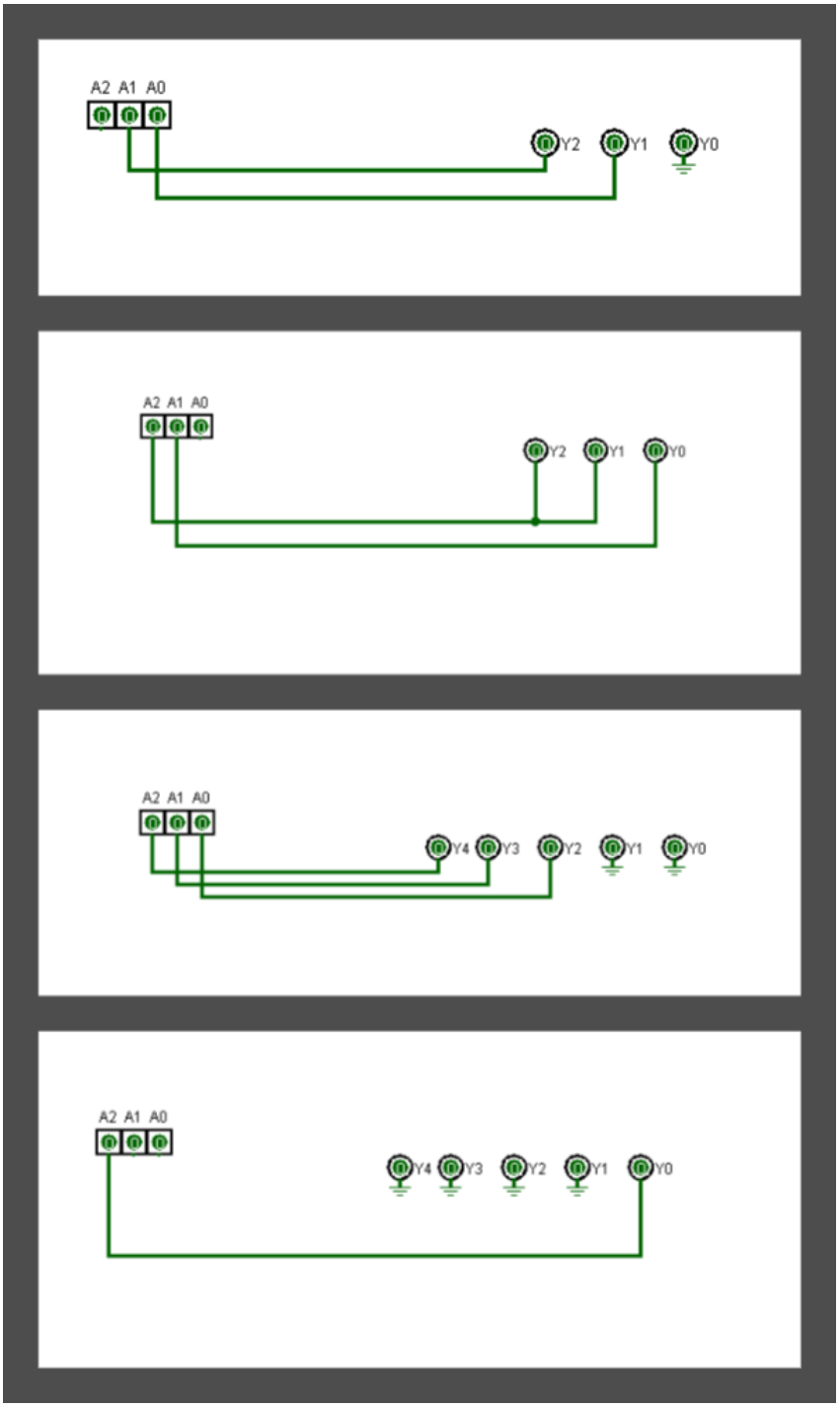

Figure 3.6: Shift modules for logical and arithmetic operations.

3.7 Multiplexer and Control Logic

A multiplexer (MUX) is used to select one of the module outputs based on the 4-bit control input (SEL). This selection mechanism integrates all functional modules into a single ALU output. The MUX output is also connected to a Zero flag detector (NOT gate output) and an Overflow flag derived from the 3-bit Adder.

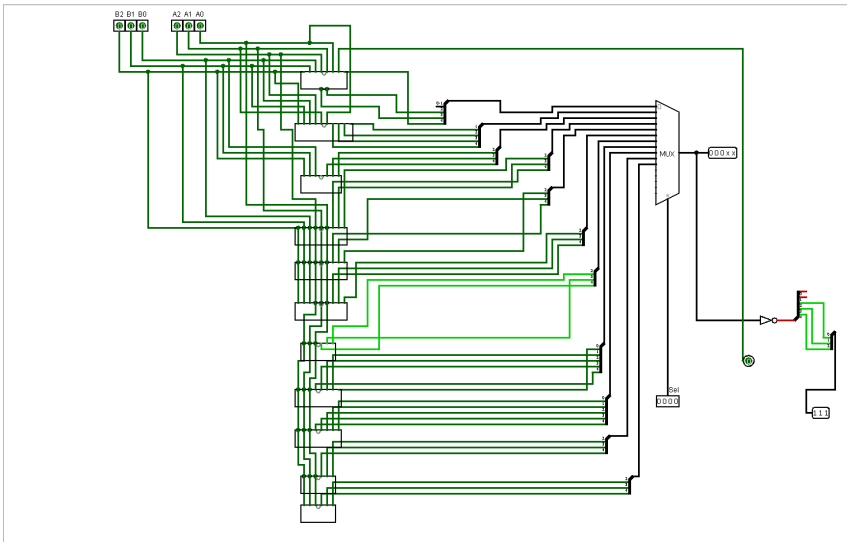

Figure 3.7: Multiplexer and Control Logic selecting the ALU operation.

Chapter 4

ModelSim Simulation

4.1 Simulation Objective

The purpose of ModelSim simulation is to verify the functional correctness of the designed 3-bit ALU. The simulation validates:

- Arithmetic and logical operations produce expected results.
- Zero (Z) and Overflow (V) flags operate correctly.
- Edge cases (e.g., overflow, zero output, negative results) are handled correctly.

4.2 Simulation Setup

The simulation environment consists of:

- ALU design and all functional submodules compiled in ModelSim.
- A SystemVerilog testbench for automated testing.
- Manual signal forcing for specific edge-case scenarios.

4.2.1 Input and Output Signals

- **Inputs:**
  - `A[2:0]` : Operand A (3-bit)
  - `B[2:0]` : Operand B (3-bit)
  - `SEL[3:0]` : Operation selection
- **Outputs:**
  - `F[4:0]` : Result of the operation
  - `Z` : Zero flag
  - `V` : Overflow flag

4.3 Forcing-Based Simulation

Using the `force` command in ModelSim, different values were applied directly:

- Example forcing command:

```
force A 3'b011
force B 3'b101
force SEL 4'b0000
run 100
```

- Waveforms and outputs were recorded for each operation.

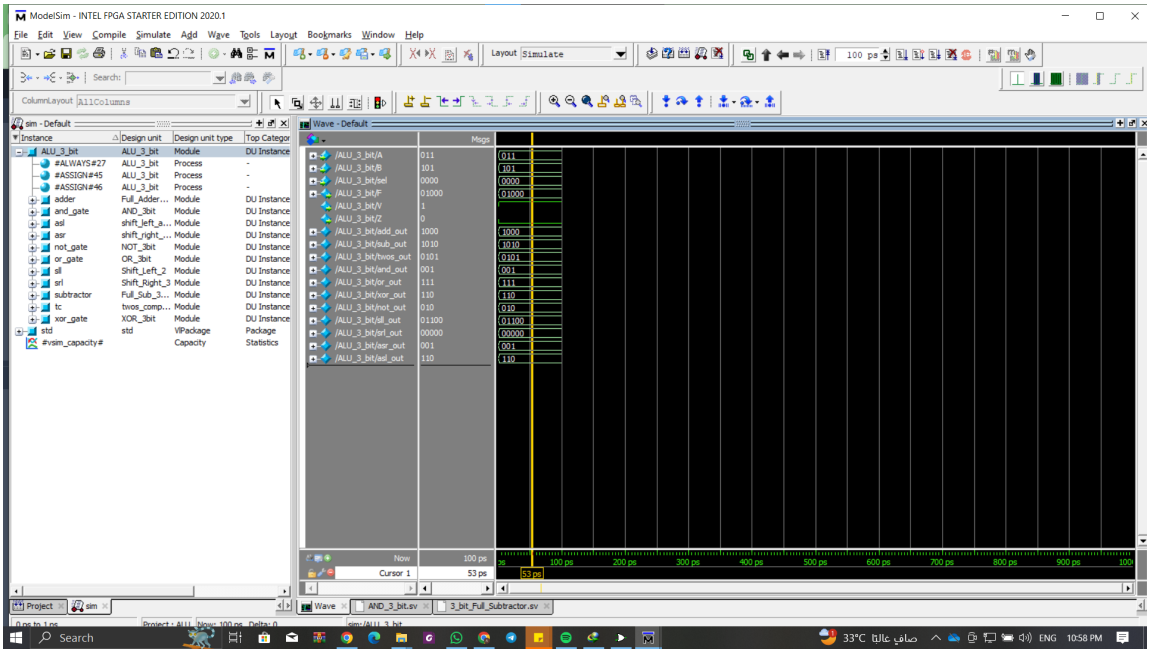

Figure 4.1: Waveform output of ALU using manual forcing commands.

4.4 Testbench-Based Simulation

A dedicated SystemVerilog testbench was created to automatically apply stimulus and capture results. It sequentially applies multiple input vectors covering all operations:

- Arithmetic operations (ADD, SUB, Two's Complement)
- Logical operations (AND, OR, XOR, NOT)
- Shifting operations (Logical and Arithmetic)

4.4.1 Waveform Output from Testbench

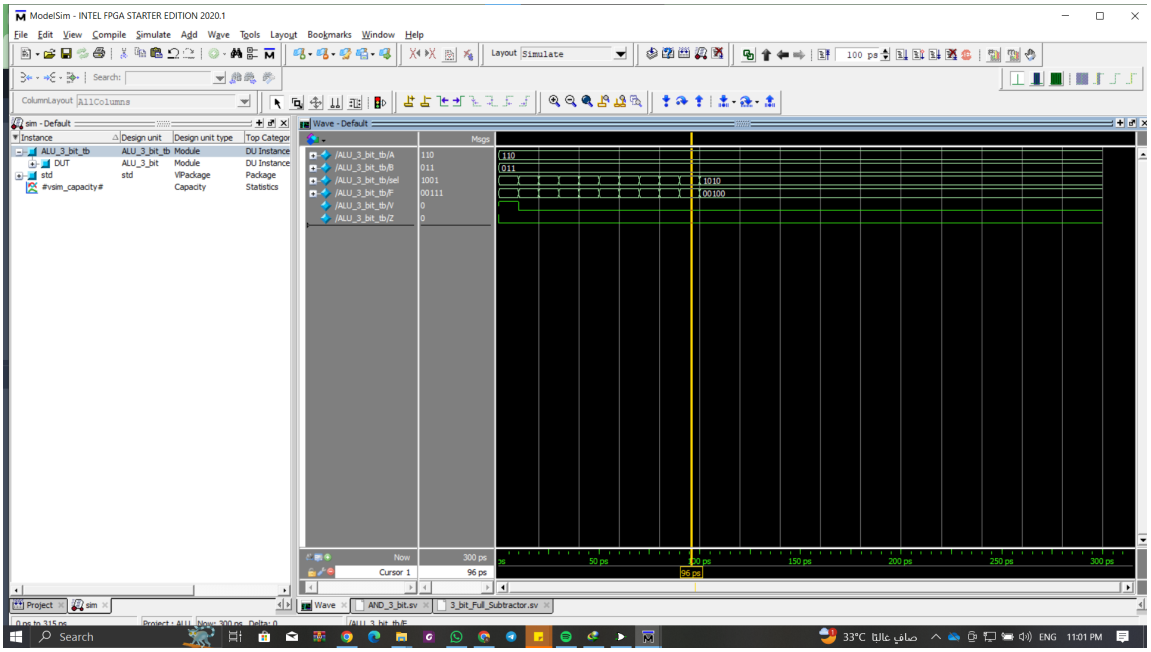

Figure 4.2: Waveform from automated testbench execution.

4.5 Test Vector Summary

Table 4.1 shows a summary of key tested cases with expected and observed outputs (to be filled after simulation).

Table 4.1: Test vector summary from simulation.

A	B	SEL	Observed F	Flags (Z, V)
3'b110	3'b101	4'b0000	5'01001	(0,1)
3'b110	3'b101	4'b0001	5'00101	(0,0)
3'b110	3'b101	4'b0010	5'00010	(0,0)
3'b110	3'b101	4'b0011	5'00111	(0,0)
3'b110	3'b101	4'b0100	5'00010	(0,0)
3'b110	3'b101	4'b0101	5'00101	(0,0)
3'b110	3'b101	4'b0110	5'00100	(0,0)
3'b110	3'b101	4'b0111	5'11000	(0,0)
3'b110	3'b101	4'b1000	5'00001	(0,0)
3'b110	3'b101	4'b1001	5'00111	(0,0)
3'b110	3'b101	4'b0001	5'00101	(0,0)
3'b110	3'b101	4'b1010	5'00100	(0,0)
3'b110	3'b101	4'b0001	5'00101	(0,0)

4.6 Analysis

- **Arithmetic operations:** Verified correct handling of carry and borrow.
- **Logical operations:** Output bits matched expected truth tables.
- **Shift operations:** Proper bit movements and sign preservation observed.
- **Flags:** Zero (Z) asserted only when output equals zero; Overflow (V) detected in signed addition overflow cases.

Chapter 5

Conclusion

This project successfully demonstrated the design, implementation, and verification of a modular 3-bit Arithmetic and Logic Unit (ALU). The ALU was designed using **Logisim**, leveraging its visual circuit building environment, and simulated using **ModelSim** to validate functionality through waveform analysis and automated testbench execution.

The ALU supports the following operations:

- Arithmetic operations: Addition, Subtraction (using Two's Complement)
- Logical operations: AND, OR, XOR, NOT
- Shift operations: Logical left/right shifts and arithmetic shifts
- Utility operation: Two's complement generation

In addition to performing operations, the ALU incorporates two key flags:

- **Zero Flag (Z):** Indicates when the ALU output equals zero.
- **Overflow Flag (V):** Detects signed overflow during arithmetic operations.

The modular approach ensured each functionality (adders, subtractor, logic gates, and shifters) was built and verified independently before integration. The Logisim implementation visually confirmed logic correctness, while ModelSim simulation provided detailed timing behavior and allowed for extensive testing using both manual forcing and automated testbenches.

The results confirm that:

1. All operations performed as expected under normal and edge-case inputs.
2. The flag outputs (Zero and Overflow) behaved correctly, improving ALU reliability.
3. The design is scalable and could be easily expanded to support more bits or additional functions.

Future Work: Potential extensions include:

- Expanding the design to an 8-bit or 16-bit ALU.
- Adding more complex operations such as multiplication, division, or rotate.
- Integrating the ALU into a complete processor datapath and testing it under instruction execution scenarios.
- Optimizing the design for FPGA implementation and hardware synthesis.

Overall, this project provided practical exposure to both digital circuit design (Logisim) and HDL simulation (ModelSim), bridging theoretical concepts with real-world implementation and verification.

Bibliography

[1] Logisim Digital Logic Simulator, *http://www.cburch.com/logisim/*

[2] ModelSim Simulation Software, *https://www.mentor.com/products/fv/modelsim/*

[3] SystemVerilog IEEE Standard, *https://ieeexplore.ieee.org/document/8299595*

[4] Digital Design Principles and FPGA Applications, *https://doi.org/10.1007/978-1-4614-1372-7*