

Implementation of Memristor Based 4-Bit ALU

Ambika Acharya

Under the guidance of

Dr. Pravanjan Samanta

Submitted in partial fulfilment of the requirement of the Degree of Bachelor of Technology of Maulana Abul Kalam Azad University (formerly West Bengal University of Technology). Kolkata

**Department of Electronics and Communication
Engineering**



St. Thomas' College of Engineering and Technology

4 D.H. Road, Kolkata-700023,

June-2025

Vision of the Department

- To build a strong teaching and research environment to cater to the manpower needs in Industrial and Academic domains of the rapidly growing Electronics and Communication Engineering.

Mission of the Department

- To produce certified industry-ready professional in Electronics and Communication Engineering, through innovative educational programs incorporating laboratory practices and project-based teaching-learning processes, in a modern environment.
- To create knowledge base of advanced technologies through research in the area of Electronics and Communication, for competitive and sustainable development of the country.
- To groom the department as a learning center to inculcate advancement of technology in Electronics and Communication Engineering with social values and environmental awareness.

Program Specific Outcome (PSOs)

After completion of program graduate engineer would have:

- PSO1. Professional skills: An ability to apply the knowledge in Electronics and Communication Engineering in various areas, like Communications, Signal processing, VLSI and Embedded Systems.
- PSO2. Competency: An ability to qualify at the State, National and International level competitive examinations for employment, higher studies and research

Project Outcome (PO)

Format for Project Outcome:

Outcome No.	Outcome Statements	Bloom's Level
CO1	Apply technical knowledges in the solution of complex real-life problems related to public health and safety, culture, society, and environment	3
CO2	Review research literature and use the research-based knowledge to identify, formulate, and analyze the problem	5
CO3	Design innovative solutions for complex engineering problems, which will be published as research paper or developed as a marketable product	6
CO4	Apply modern tools to predict solution, design and develop the solution to problem	4
CO5	Assess societal, health, safety, legal and cultural issues, and the consequent responsibilities relevant to the professional engineering practice.	4

CO7	Apply ethical principles and ethics and responsibilities related to engineering practice.	3
CO8	Function effectively as a member and / or leader in a team	3
CO9	Communicate effectively on professional activities with the team members, superiors and with society at large	3
C10	Plan, manage the project and control finance as a member and leader in a team.	5
C11	Apply the knowledge acquired during the project, in future higher studies or any professional job.	4

Bloom's Level: Remember = 1; Understand = 2; Apply = 3; Analyze = 4; Evaluate = 5; Create = 6

Project Outcomes Vs Program Outcome and Program Specific Outcome (PO) Matrix:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	2	1	3	1	---	3	3	2	3	2	2	---	---	1
CO2	3	2	1	2	---	2	1	---	---	3	1	3	1	2
CO3	3	---	2	1	1	3	3	2	2	1	1	---	2	2
CO4	3	1	1	3	3	---	3	---	2	---	2	---	---	---
CO5	3	2	2	1	---	3	3	3	2	2	1	---	1	3
CO6	3	2	2	1	1	3	2	1	3	2	2	---	---	3
CO7	2	2	2	3	---	2	2	1	2	1	2	1	2	---
CO8	3	1	3	2	---	1	1	---	3	2	1	---	---	3
CO9	3	---	2	---	1	1	1	---	2	1	1	---	1	2
CO10	2	2	3	2	1	2	2	2	3	2	2	---	---	---
CO11	1	1	2	1	---	3	2	1	3	1	2	---	1	---

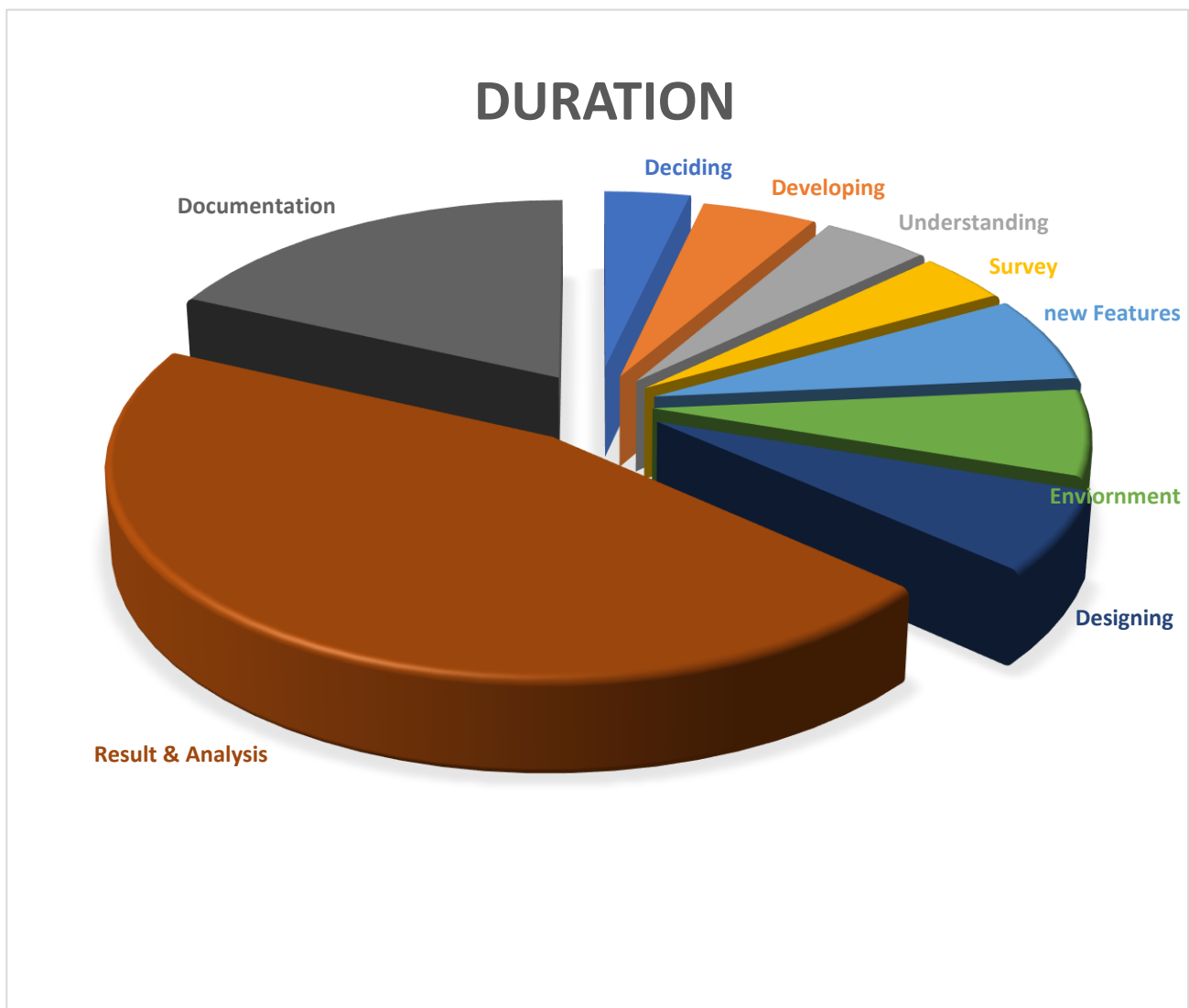
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
Course PO	3	1	2	2	1	2	2	1	2	2	2	---	1	1

Finance:

The project's software-based nature ensures zero finance costs, eliminating the need for significant upfront investments or ongoing financial expenditures. This cost-effective approach enables organizations to allocate resources more efficiently, leveraging technology to drive innovation & growth without incurring substantial financial burdens. By harnessing the power of software, the project can achieve its objective while maintaining a lean financial profile, making it an attractive solution for organizations seeking to optimize their financial resources.

Project Planning:

TASK	Start Date	Duration	End Date
Deciding project idea and finalizing the project	25-07-24	12 days	06-08-24
Developing project idea	07-08-24	16 days	23-08-24
Brief understanding about VLSI	07-08-24	15 days	22-08-24
Starting Literature Survey	09-08-24	13 days	22-08-24
Adding new features to the project	11-09-24	22 days	03-10-24
Familiarization with program environment	11-09-24	22 days	03-10-24
Designing Flow Chart, Block Diagram	03-10-24	21 days	24-10-24
Checking the operations & results	24-10-24	5 months	24-03-25
Project structure finalization and Document presentation	24-03-25	2 months	24-05-25



Certificate

Department of Electronics & Communication Engineering

St. Thomas' College of Engineering and Technology

This is to certify that the project entitled “**Implementation of Memristor Based 4-Bit ALU**”
has been carried out by

Ambika Acharya (12200321012)

Under my guidance during the year July-2024 to May-2025, and accepted for partial fulfilment
of the requirement of the Degree of Bachelor of Technology of Maulana Abul Kalam Azad
University (formerly West Bengal University of Technology). Kolkata

(Dr. Pravanjan Samanta)

Signature of Project Guide

Date:

Acknowledgement

I would like to sincerely thank our project mentor, for their invaluable guidance, constant support, and encouragement throughout the development of my project. I am also deeply grateful to our Head of the Department, for providing me with the resources and motivation needed to carry out this work. My heartfelt thanks go to our respected Principal Ma'am, for her continuous encouragement and for fostering a learning environment that made this project possible. I also extend our gratitude to all the faculty members of the department for their support, insightful suggestions, and motivation during every phase of the project. Their collective guidance and cooperation have played a significant role in the successful completion of my work.

Ambika Acharya

Date:

Table of Content:

Serial No:	Contents:	Page No:
1	Abstract	8
2	Introduction <ul style="list-style-type: none">• Overview• Motivation• Objectives• Application	9-10
3	Background of the project <ul style="list-style-type: none">• Literature Survey	11-12
4	Description of the project <ul style="list-style-type: none">• Specification• Working Principle• Design• Circuit diagram• Component List• Mathematical Analysis• Flowchart• Algorithm• Truth Table• ALU Operations• Graphical Analysis	13-29
5	Results <ul style="list-style-type: none">• Real time Implementation• Future Scope• Benefits	30-34
6	Conclusion	35
7	Reference	36-37

Abstract

This project presents the implementation of a 4-bit Arithmetic Logic Unit (ALU) using memristor-based logic simulation in Python. An ALU is a fundamental component of any computing system, capable of performing arithmetic and logical operations. Traditional CMOS-based ALUs face limitations in terms of scalability, power consumption, and size as technology progresses toward nanoscales. Memristors—non-volatile, two-terminal components that retain resistance based on historical current—offer a promising alternative due to their compact size, low power requirements, and ability to integrate memory and computation. In this project, we simulate the behaviour of memristors to design and implement a 4-bit ALU capable of executing basic arithmetic (addition, subtraction) and logical (AND, OR, XOR, NOT) operations. The entire design and simulation are carried out using Python, showcasing how memristive behaviour can be modelled through programmable resistance and state-dependent logic. This project demonstrates the potential of memristor-based architectures for future high-performance and energy-efficient computing systems.

Introduction

➤ Overview

An Arithmetic Logic Unit (ALU) is a fundamental building block of any digital system, responsible for performing essential arithmetic operations such as addition and subtraction, as well as logical operations like AND, OR, XOR, and NOT. Traditionally, ALUs are implemented using CMOS technology, which, while reliable, faces growing challenges with power consumption, leakage currents, and scalability as devices shrink to nanoscales [1]. To address these limitations, this project explores the implementation of a 4-bit ALU using memristor-based logic circuits. Memristors—non-volatile, nanoscale, two-terminal devices—exhibit the unique capability to combine memory and computation within a single element [2], enabling highly compact, power-efficient, and reconfigurable hardware architectures [3]. Their ability to retain resistance states without power and perform logic directly within memory presents a promising solution to the von Neumann bottleneck [4]. This project demonstrates how a set of arithmetic and logic operations can be executed using only memristor-based logic simulated in Python, thereby validating the potential of memristive technology in designing next-generation ALUs for energy-efficient and compact digital systems [5]. Memristor is a real-time technology. It is still in the research domain but it's our future. To get rid of excessive memory consumption and due to the limited access of cloud its hard to cope up with memory. Its acts as a flow of pipe capable of controlling the waterflow by comparing its radius.

➤ Motivation

With the continuous miniaturization of electronic devices, traditional CMOS technology is reaching its physical and performance limits, including issues like increased power consumption and limited scalability. To overcome these challenges, there is a strong need to explore emerging technologies that offer better efficiency and performance. Memristors, due to their nanoscale size, low power usage, and non-volatile nature, provide an innovative solution by combining memory and logic in a single device.

This project is motivated by the potential of memristor technology to revolutionize digital circuit design. By implementing a 4-bit ALU using memristor-based logic, the aim is to create a compact, energy-efficient, and high-speed computational unit. It also serves as a foundational step toward future applications such as in-memory computing and neuromorphic architectures, where processing and storage occur together. Exploring this technology through a fundamental component like the ALU helps in understanding its practical viability and sets the stage for more complex system designs.

➤ Objective

The main objectives of this project are:

1. **To design and implement a 4-bit ALU** using memristor-based logic gates that can perform a set of basic arithmetic and logical operations.
2. **To replace conventional CMOS-based ALU design** with a more compact and power-efficient memristor-based alternative, utilizing the unique properties of memristors such as non-volatility and scalability.
3. **To simulate the behaviour** of the memristor-based ALU using software tools or custom simulation environments that mimic real-world circuit operations.
4. **To analyse performance characteristics**, such as functional accuracy, switching speed, power consumption, and area efficiency in comparison to traditional implementations.
5. **To demonstrate the feasibility** of memristors in implementing computational logic for next-generation processing units, particularly in low-power or embedded applications.

➤ Application

The memristor-based 4-bit ALU, while basic in complexity, lays the foundation for a wide range of applications in modern computing systems. Some potential applications include:

1. **Low-power embedded systems:** Due to their small size and low energy requirements, memristor-based ALUs are ideal for battery-operated or wearable electronic devices [1].
2. **Neuromorphic computing:** Memristors can mimic the behaviour of biological synapses, making them suitable for implementing logic operations in neuromorphic architectures where computation and memory are integrated [2].
3. **Edge computing and IoT devices:** With reduced power and area, these ALUs can be embedded in smart sensors and IoT modules to perform local processing without the need to communicate with central servers [3].
4. **Non-volatile processors:** By integrating logic and memory, processors built with memristor-based ALUs can retain their operational states, supporting instant-on computing systems [4].
5. **Educational and experimental use:** This project serves as a model for learning, teaching, and further research in advanced digital circuit design and emerging nanotechnologies.
6. **In-memory computation systems:** The architecture supports parallel and in-memory data processing, reducing latency and improving system speed for big data and AI workloads [5].

Background of the Project

➤ Literature Survey

[6] Leon Chua (1971) in “Memristor—The Missing Circuit Element” laid the theoretical foundation for the memristor as the fourth fundamental circuit element, complementing resistor, capacitor, and inductor. He proposed the memristor as the missing link between charge and magnetic flux, defining its unique behaviour through the relationship. Although no physical device existed at the time, his circuit-theoretic formulation-initiated decades of research and ultimately enabled the practical development of memristor-based logic and memory circuits.

[7] Shirinzadeh et al. (2018) in “Logic Design Using Memristors: An Emerging Technology” explored memristor-based logic implementation approaches such as IMPLY and MAGIC. The study detailed methods of integrating memristors into in-memory computing systems and discussed challenges like sneak-path issues in crossbar arrays. The paper also categorized memristor logic synthesis strategies and emphasized their suitability for compact, high-speed logic like that in ALUs.

[8] Mohanty (2013) in “Memristor: From Basics to Deployment” presented a comprehensive overview of memristor development, from Chua’s theory to HP’s physical realization. It covered memristor properties such as hysteresis and dynamical resistance and evaluated different types (e.g., spintronic, manganite, polymeric). The article also outlined practical applications in digital and analog circuits, reinforcing memristor potential in logic circuits, including ALUs.

[9] Huang et al. (2010) in “Memristor System Properties and Its Design Applications” focused on memristor behaviour under nonlinear drift, and presented analysis and design strategies for memristor-based memory and logic circuits. The paper discussed voltage-induced ionic transport and proposed models to address read/write reliability. It supported the use of memristors in logic systems like ALUs by demonstrating robust modelling from basic equations to practical implementations.

[10] Mladenov & Kirilov (2015) in “Memristor Modelling in MATLAB® & PSPICE®” proposed a modified Joglekar model for enhanced simulation accuracy. The comparison between Joglekar and Pickett models using window functions allowed better tuning for real-world behaviour of memristor devices in logic circuits. Their simulation framework supports efficient and realistic ALU design modelling.

[11] Abdalla (2020) in “Memristor Operation and Applications” explains memristor operation using analogies and circuit behaviour. The paper highlights the non-volatile and analog switching properties of memristors that make them excellent for storing logic states, such as in ALU circuits. It also simplifies understanding through examples, making it useful for both technical and educational applications.

[12] Apollos (2019) in “Memristor Theory and Mathematical Modelling” explored a range of memristor models including Simmons Tunnel Barrier, TEAM, and VTEAM. The work detailed memristor mathematical expressions and window functions, and emphasized how these models align with physical characteristics. It also discussed practical simulation tools like Cadence and Verilog-A, aiding accurate ALU gate modelling.

[13] Gao et al. (2020) in “Memristor-Based Logic Gate Circuit” demonstrated PSPICE-based implementation of logic gates using NiO memristors. The results aligned closely with expected truth tables, validating that memristors can replace conventional CMOS logic blocks. This practical realization forms a building block for more complex designs such as ALUs.

[14] Sahoo & Prabakaran (2020) in “Nanoionics Memristor Equipped Arithmetic Logic Unit using VTEAM Model” specifically implemented a 1-bit ALU using the VTEAM model in Cadence Virtuoso. The design integrated arithmetic (addition, subtraction) and logic operations (AND, OR, XOR), and demonstrated advantages like area and power efficiency over CMOS. This paper directly supports and validates the approach of building 4-bit ALUs using memristor-based technology.

Short comparative table between Memristor-based and CMOS-based 4-bit ALUs:

Aspect	Memristor-Based ALU	CMOS-Based ALU
Power Consumption	Very low (non-volatile)	High (leakage and switching losses)
Area Efficiency	Highly compact	Requires more space
Speed	Faster (in-memory computing)	Slower (separate memory and logic)
Data Retention	Retains data without power	Loses data when power is off
Scalability	Easily scalable (2-terminal devices)	Limited by transistor scaling
Design Complexity	Simpler logic implementation	Complex with more components
Energy Efficiency	Highly energy-efficient	Less efficient
ALU Suitability	Proven for efficient arithmetic and logic functions	Traditional but less optimized

Description of the Project

This project aims to model, simulate, and visualize the **memristor device**, focusing on its characteristic *non-linear hysteresis behaviour*, and to integrate this device conceptually with a **4-bit Arithmetic Logic Unit (ALU)**.

The memristor, short for *memory resistor*, is a two-terminal passive circuit element that retains memory of its past voltages and currents. It is considered a key enabler for next-generation non-volatile memory and neuromorphic computing.

Additionally, an interactive 4-bit ALU simulator is developed, visualizing the waveform of selected operations with an associated memristor circuit diagram for educational purposes.

➤ Specifications:

Table 1: Parametric table

<i>Parameter</i>	<i>Value/Description</i>
<i>Memristor model</i>	Sinusoidal voltage-driven I-V model
<i>Memristor visualization</i>	4-quadrant "8-shaped" I-V hysteresis loop
<i>ALU bit-width</i>	4-bit
<i>ALU operations supported</i>	16 operations (add, sub, and/or/xor, shift, etc.)
<i>Programming language</i>	Python 3.x
<i>Visualization library</i>	Matplotlib
<i>Numerical integration</i>	Trapezoidal integration / cumulative sum
<i>User interface</i>	Text-based console interface with waveform plot
<i>Target application</i>	Educational simulation & visualization

➤ Memristor Working Principle:

- The TiO_2 (Titanium Dioxide) layer used in the memristor is divided into two regions:
 - **Doped Region:** Contains oxygen vacancies (e.g., Ti_4O_7 or TiO_{2-x}), making it **highly conductive** (resistance = R_{on}).
 - **Undoped Region:** Made of pure TiO_2 , which is **highly resistive** (resistance = R_{off}).
- The **memristance** $M(x)$ is calculated as:

$$M(x) = R_{on} \cdot \frac{x}{D} + R_{off} \cdot \left(1 - \frac{x}{D}\right)$$

where:

- x : Width of the doped region
- D : Total width of the TiO_2 layer

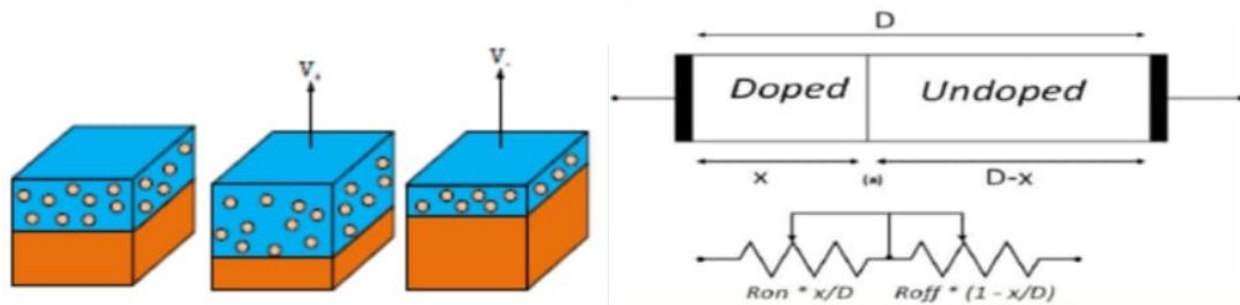


Figure 1 : Diagrams illustrate how the resistance varies with the distribution of doped and undoped regions and the applied voltage.

➤ Design:

a. ALU Design

- 4-bit ALU supports:
 - Addition
 - Subtraction
 - Bitwise AND/OR/XOR
 - Complement ($\sim A$, $\sim B$)
 - Shifts (\ll , \gg)
 - Increment/Decrement
 - Pass-through (A , B)
 - Equality check ($=$)
- Implemented in Python using bit manipulation and logical operators.

b. Memristor Device Model

1. HP Model: Linear ion drift model for memristors.
2. Stanford Model: Specific memristor model (less commonly referenced).
3. Ion Drift Model: Describes memristor behavior based on ion movement.
4. VTEAM Model: Flexible memristor model for simulating various devices.
5. TEAM Model: Memristor model similar to VTEAM, with different fitting parameters.

➤ Circuit Diagram

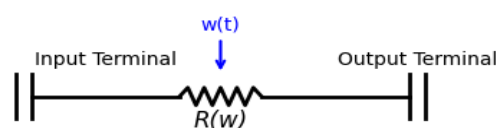


Figure 2: memristor equivalent Circuit

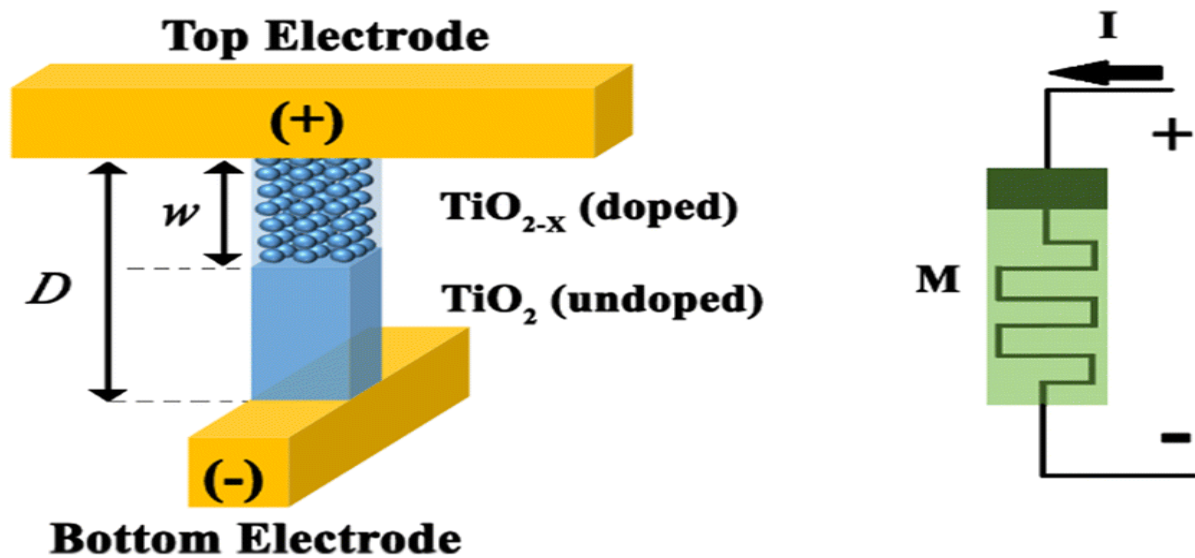


Figure 3: Internal Structure & It's Symbol

Explanation:

- The memristor acts like a time-dependent resistor whose resistance is a function of the charge that has passed through it.
- A simple variable resistor $R(w)$ is used to depict this in the diagram.
- The circuit diagram of a memristor is relatively simple, typically represented as a two-terminal passive device. It consists of a thin layer of memristive material, such as titanium dioxide (TiO_2), sandwiched between two metal electrodes—commonly platinum or gold.
- Internally, the memristive layer is divided into two regions: a doped region with a high concentration of oxygen vacancies (conductive), and an undoped region with low vacancy concentration (insulating). The boundary between these regions shifts depending on the polarity and magnitude of the applied voltage. When a positive voltage is applied across the electrodes, mobile oxygen vacancies drift, expanding the doped region and reducing the device's resistance. Conversely, a negative voltage causes the doped region to shrink, increasing resistance. This behaviour results in a pinched hysteresis loop in the I-V (current-voltage) characteristic curve.

➤ Component List

Table 2: Component Table

<i>Component</i>	<i>Quantity</i>	<i>Remarks</i>
<i>Memristor device model</i>	Simulated	Using Python-based numerical modeling
<i>ALU</i>	Simulated	Python code
<i>Visualization tools</i>	Software	Matplotlib
<i>User interface</i>	Software	Text-based Python console

➤ Mathematical Analysis

Input Output Voltage characteristics & Hysteresis loop are the main operations shown in figure 4. We use python simulator to design Ion drift memristor model.

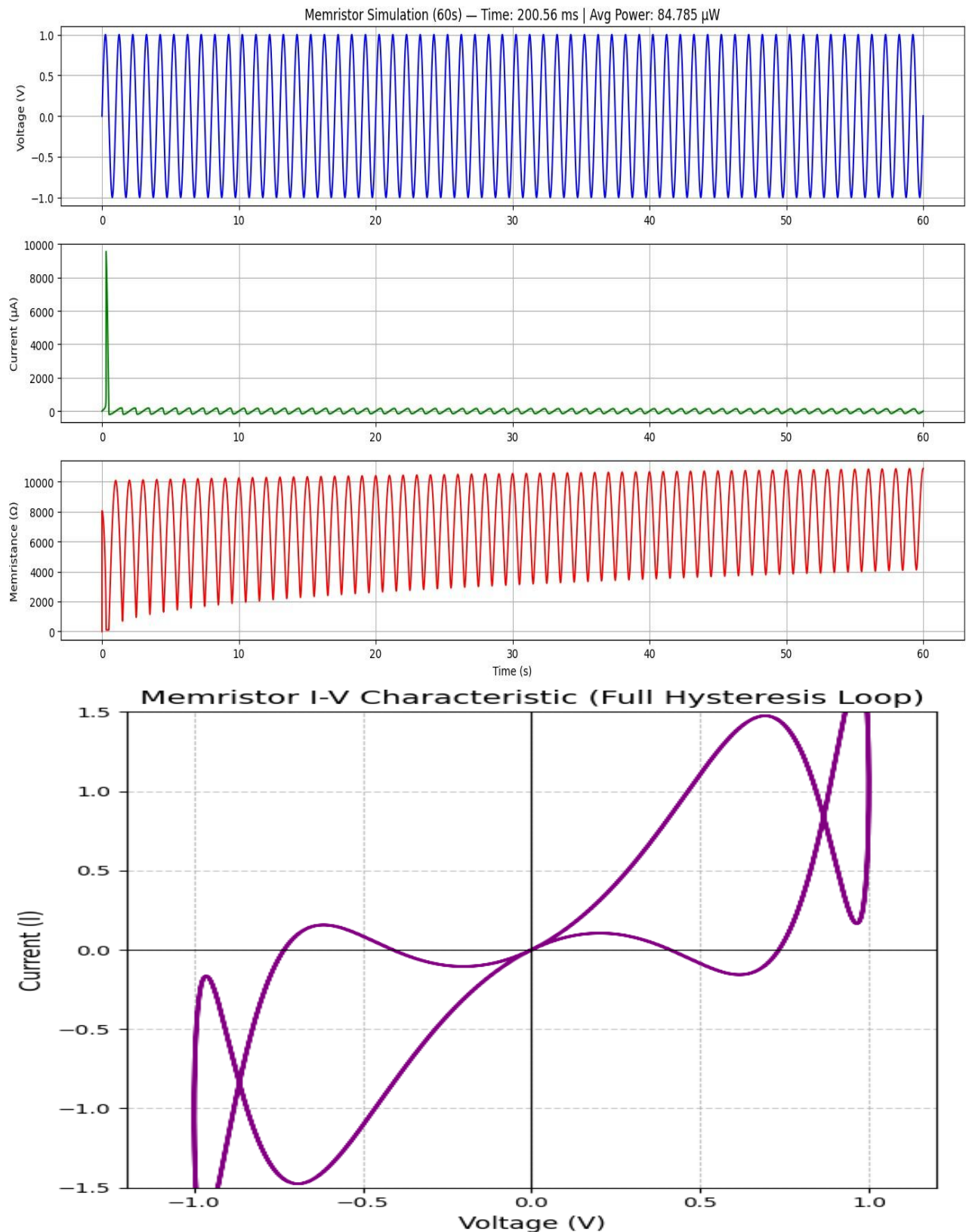
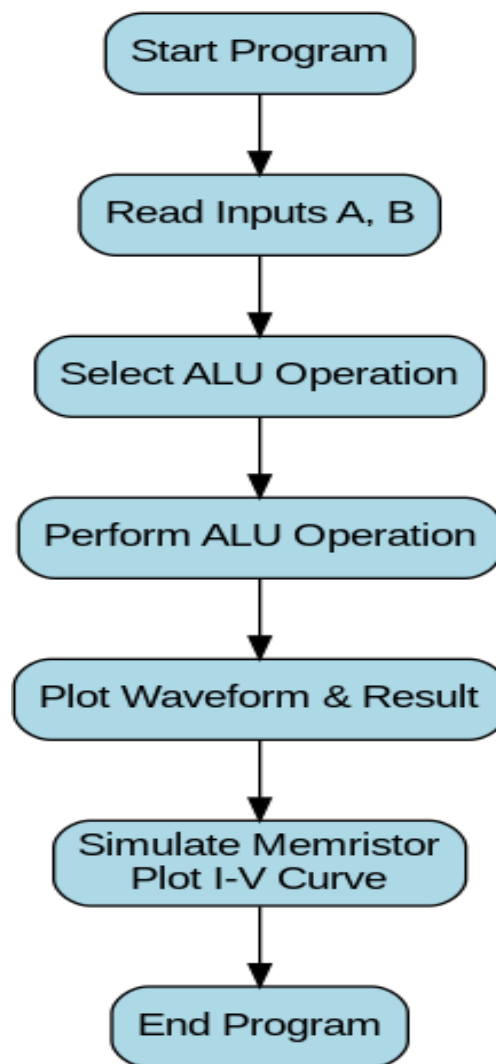


Figure 4: Memristor response & Hysteresis loop of memristor

➤ **Flowchart**



➤ **Algorithm**

ALU Algorithm:

1. Input A, B.
2. Map selected operation to control signal SEL.
3. Perform operation using Python bitwise/arithmetic ops.
4. Output result and visualize waveform.

Memristor Algorithm:

1. Generate time vector t .
2. Compute $V(t) = \sin(t)$.
3. Numerically integrate $V(t) \rightarrow q(t)$.
4. Compute $I(t)$.
5. Plot I vs V .

➤ Truth Table Analysis:

Table 3: Operational Truth Table

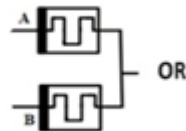
Operation	Symbo l	SEL code (4- bit)	Result	Carry / Flag (if applicable)	Description
Addition (A + B + CIN)	+	0000	(A+B+CIN) % 16	Carry if overflow	Adds A and B with optional CIN
Subtraction (A - B - CIN)	-	0001	(A-B-CIN) % 16	Flag = 1 if negative	Subtracts B from A with CIN
AND	&	0010	A & B	0	Bitwise AND
OR		0011	A B	0	Bitwise OR
XOR	^	0100	A ^ B	0	Bitwise XOR
NOT A	~A	0101	~A % 16	0	Bitwise complement of A
NOT B	~B	0110	~B % 16	0	Bitwise complement of B
Left Shift A	<<	0111	(A << 1) % 16	0	Shifts A left by 1 bit
Right Shift A	>>	1000	A >> 1	0	Shifts A right by 1 bit
Increment A	++	1001	(A+1) % 16	Carry if overflow	Increments A by 1
Decrement A	--	1010	(A-1) % 16	Flag = 1 if negative	Decrements A by 1
Clear (Zero)	0	1011	0000	0	Forces result to 0
Set (One)	1	1100	1111	0	Forces result to 15 (all bits 1)
Pass A	A	1101	A	0	Passes A directly
Pass B	B	1110	B	0	Passes B directly
Equality (A == B)	==	1111	1 if A==B, else 0	0	Compares A and B

An Arithmetic Logic Unit (ALU) is a core component of a computer's central processing unit (CPU) that performs a wide range of arithmetic and logical operations. It takes two input operands, commonly labeled A and B, and executes operations based on a control signal or opcode. The arithmetic operations typically include addition, subtraction, increment, and decrement, allowing the ALU to perform basic mathematical computations. Logical operations include AND, OR, XOR, and NOT, which are essential for decision-making and bitwise manipulation. Some ALUs also support operations like clearing the output (setting it to 0), setting it to 1, or passing one of the inputs directly as output. The result of the selected operation is a binary output, often restricted to a fixed number of bits (such as 4-bit or 8-bit), with overflow or carry handled by additional flags. The flexibility and speed of the ALU enable it to execute fundamental tasks required for instruction execution, making it a critical building block in digital systems, microprocessors, and embedded devices.

➤ ALU Basic Operations

○ OR gate

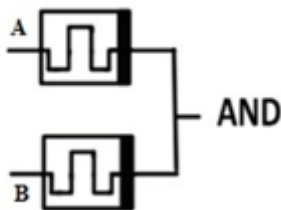
The below figure shows the structure of the OR gate using memristors, here the two memristors are connected in parallel to the polarities and when the current flows into the memristor, the resistance decreases and reaches RON state (minimum resistance) and when the current flows out of the memristor, the resistance increases and reaches ROFF state (maximum resistance). The OR gates output voltage is given by the below equation.



$$V_{out} = (R_{off} / (R_{off} + R_{on})) V_{DD}$$

○ AND gate:

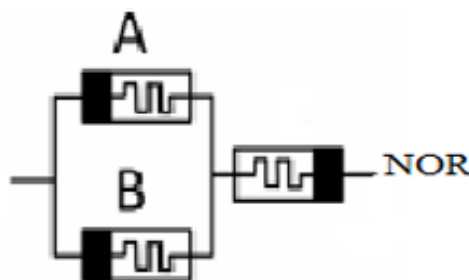
The above fig shows the structure of the AND gate using memristors, here the two memristors are connected in parallel at opposite polarities to that of the OR gate, when the current flows into the memristors, the resistance increases and reaches ROFF state and when the current flows out of the memristor, the resistance decreases and reaches RON state. The AND gates output voltage is given by the below equation.



$$V_{out} = (R_{ON} / (R_{OFF} + R_{ON})) V_{DD}$$

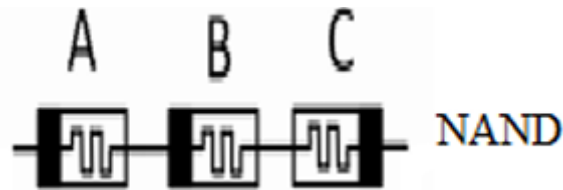
○ NOR gate:

Figure 9 depicts a NOR gate built using memristors, where two input memristors are connected in parallel and then in series with an output memristor. Initially, the output memristor is set to a low-resistance (logic 1) state. Inputs are applied by programming the input memristors with either high or low resistance. A high input corresponds to high resistance. Only when both inputs are high (i.e., both memristors have high resistance), a sufficient voltage drop appears across the output memristor, switching it from low resistance (logic 1) to high resistance (logic 0), thus realizing NOR logic behavior.



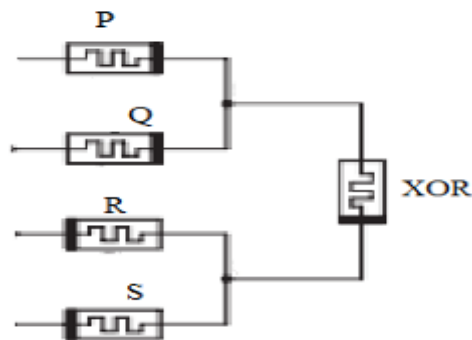
- NAND gate:

Fig -:10 NAND gate structure using memristors The above figure represents the structure of a NAND gate using memristors, here the two input memristors are connected in series and also the output memristor is connected to these memristor in series with an output memristor. In this NAND gate, when the memristance of both the input memristor are low, then the voltage across the output memristor should be high to change the configuration of the output memristor from logic 1 to logic 0.



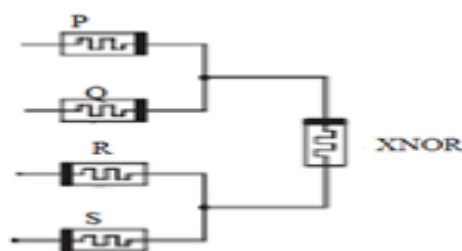
- XOR gate:

The XOR gate is built using memristor-based AND and OR gates. Initially, the output memristor has high resistance (logic 0). When both inputs are 0 or 1, the output remains high resistance. But when inputs differ (one is 1, the other is 0), the voltage difference causes the output memristor to switch to low resistance (logic 1), thus performing XOR logic.



- XNOR

The figure shows the XNOR gate structure using memristors, which is similar to the XOR gate (one AND and one OR gate with an output memristor). The key difference is that the output memristor is connected in reverse direction. This reverse connection and polarity cause the XNOR output to be the opposite of the XOR output.



➤ ALU Operations Graphical Analysis:

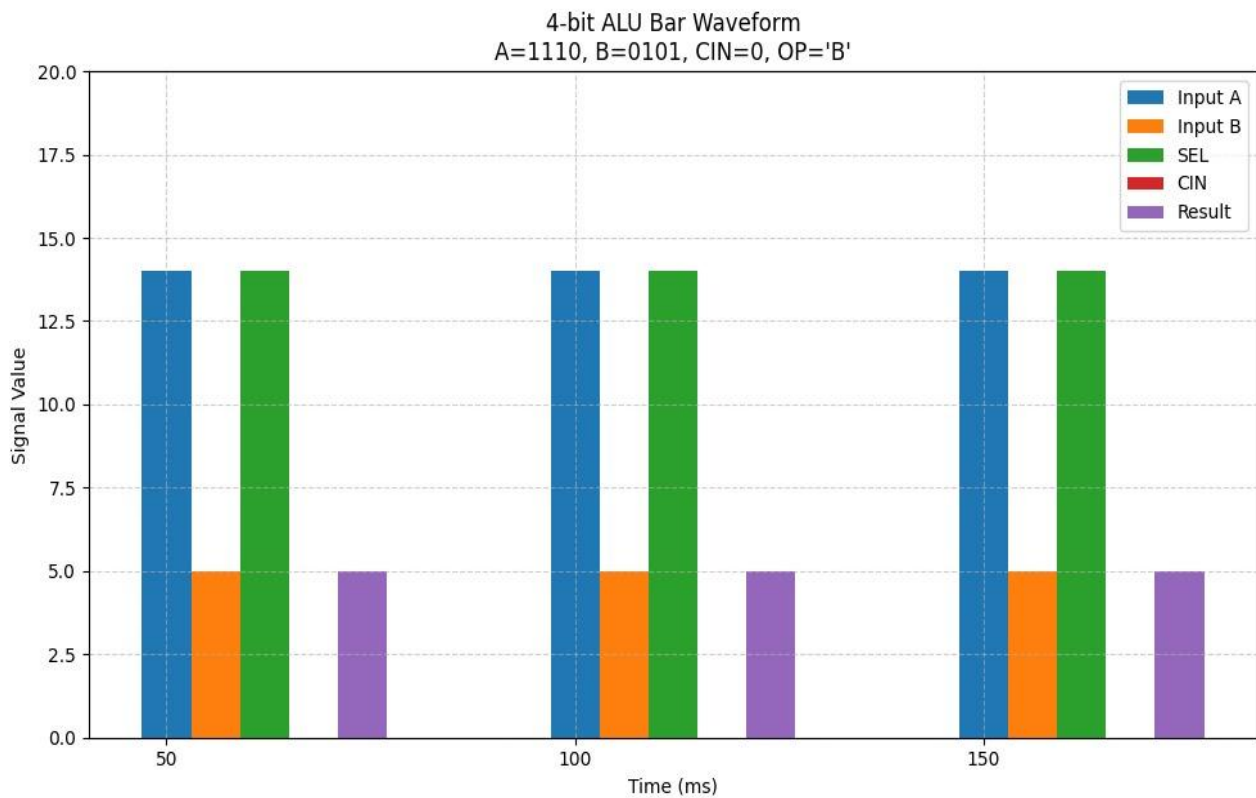


Figure 5: Operation B

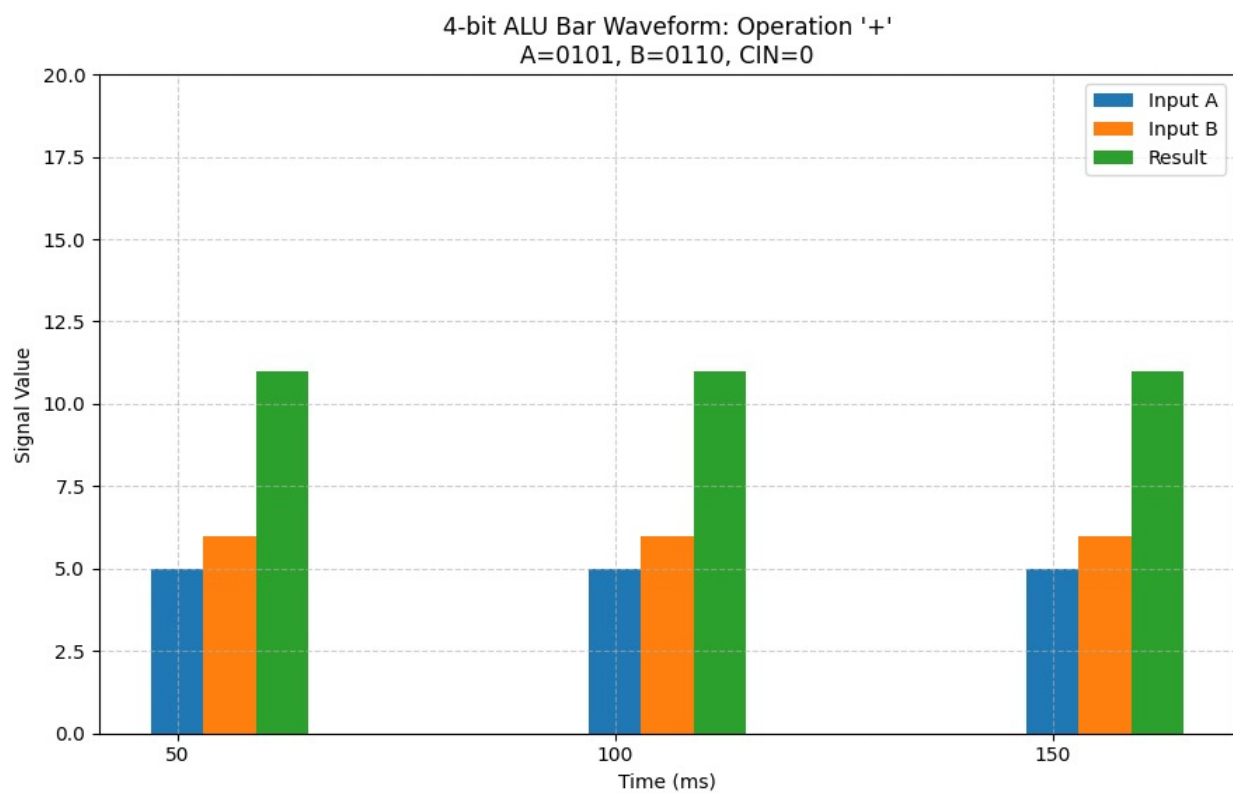


Figure 6: A+B

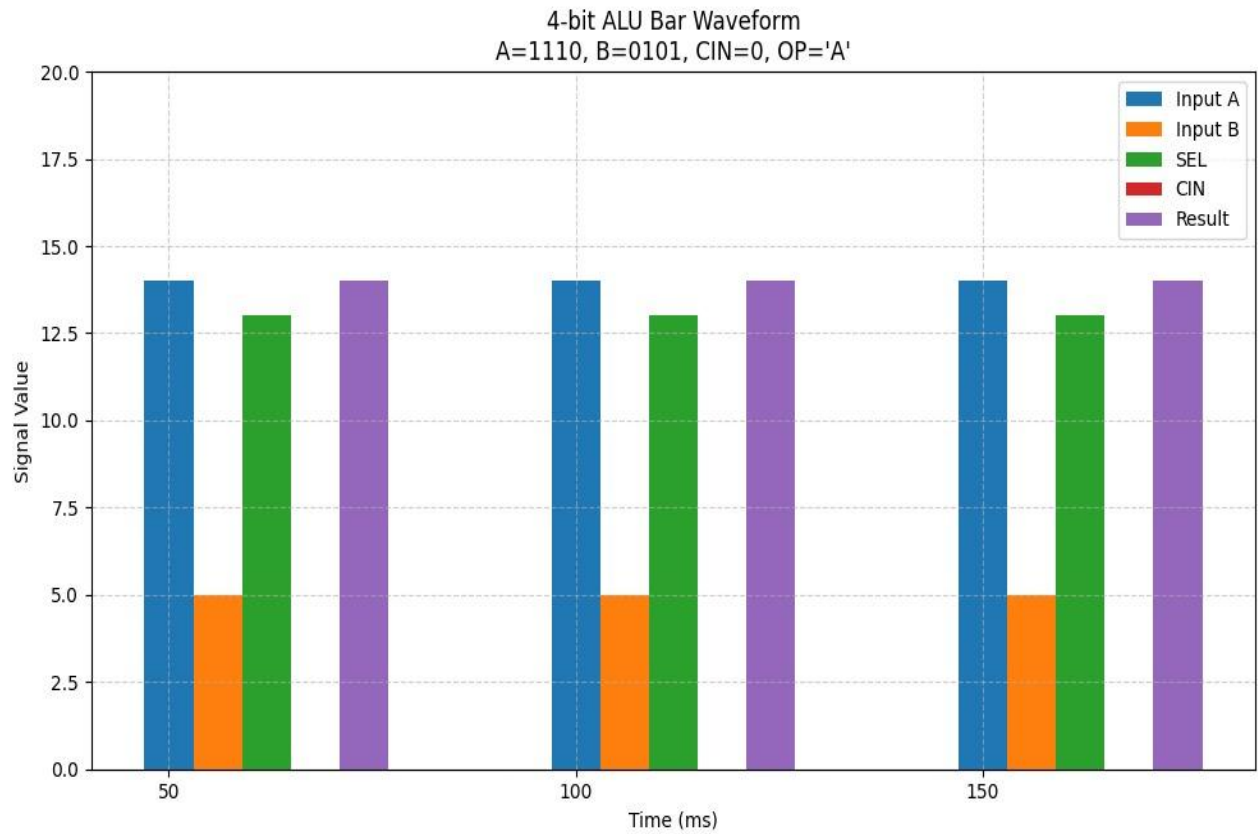


Figure 7: Operation A

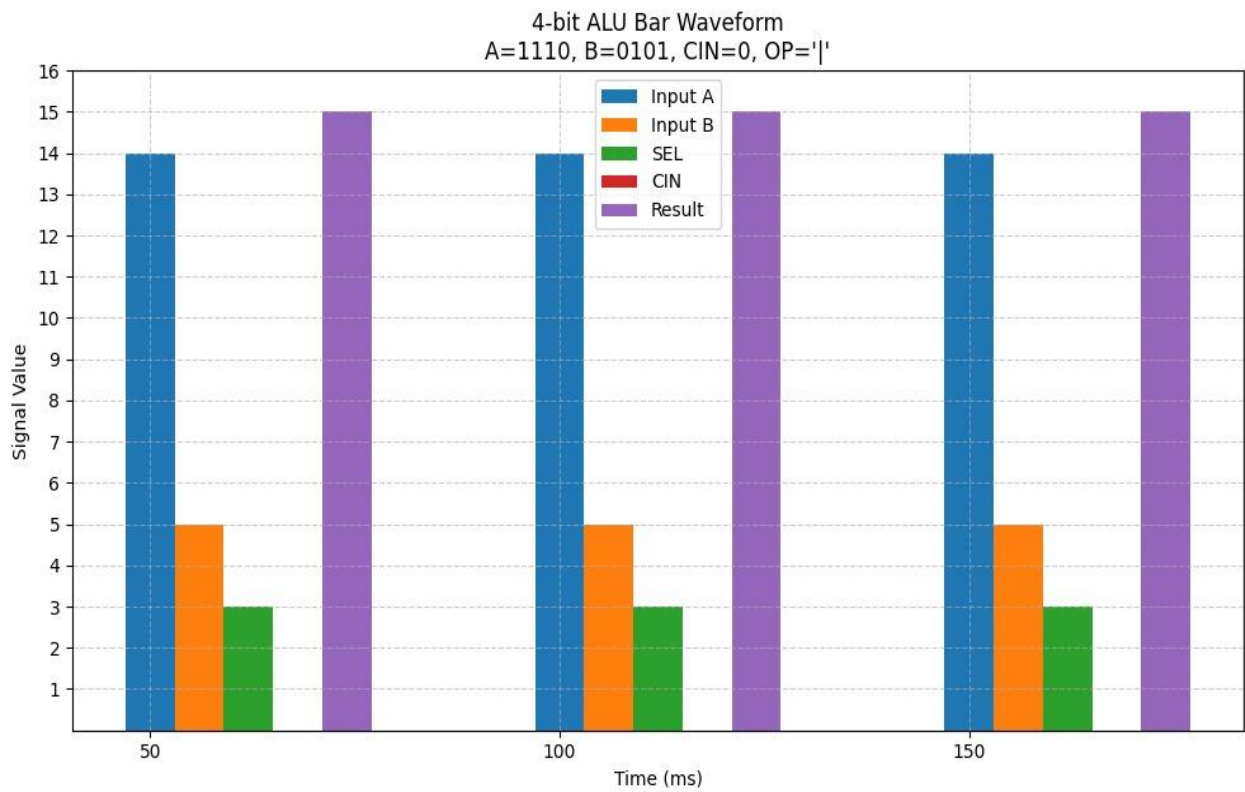


Figure 8: OR Operation

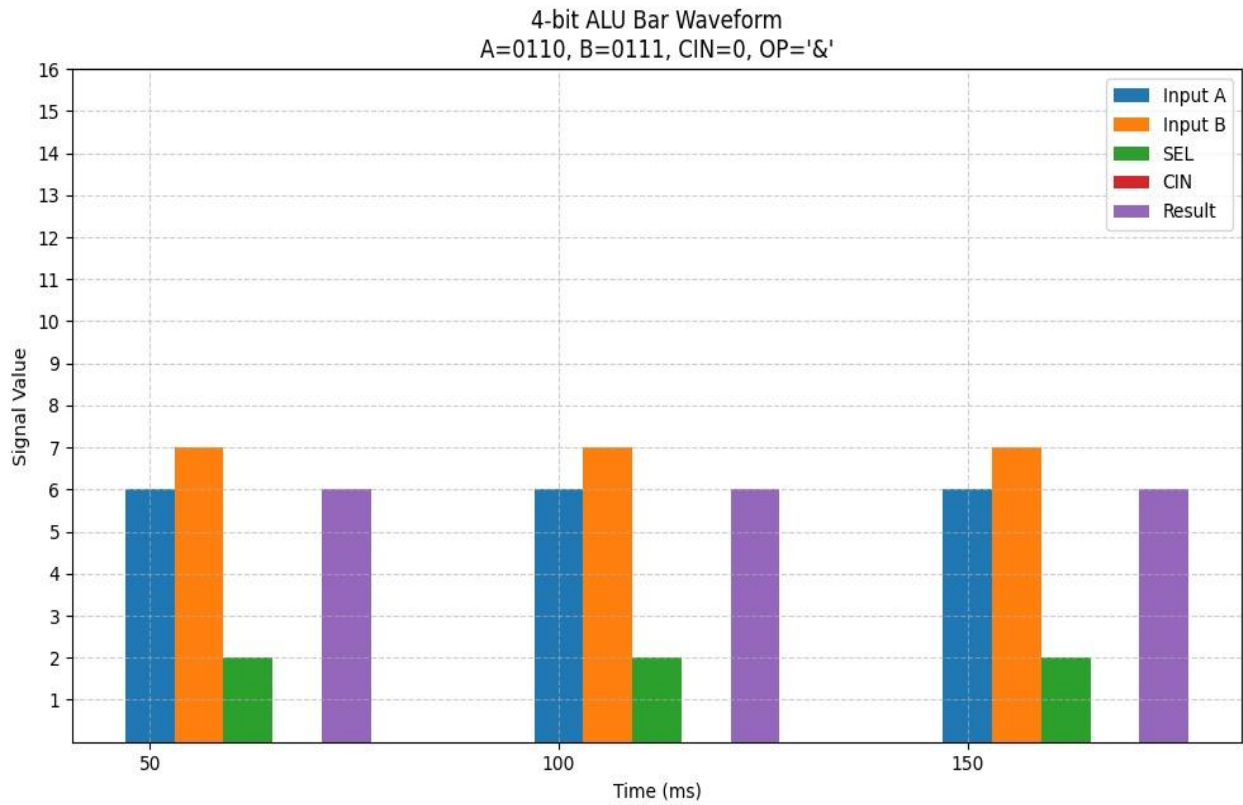


Figure 9: AND Operation

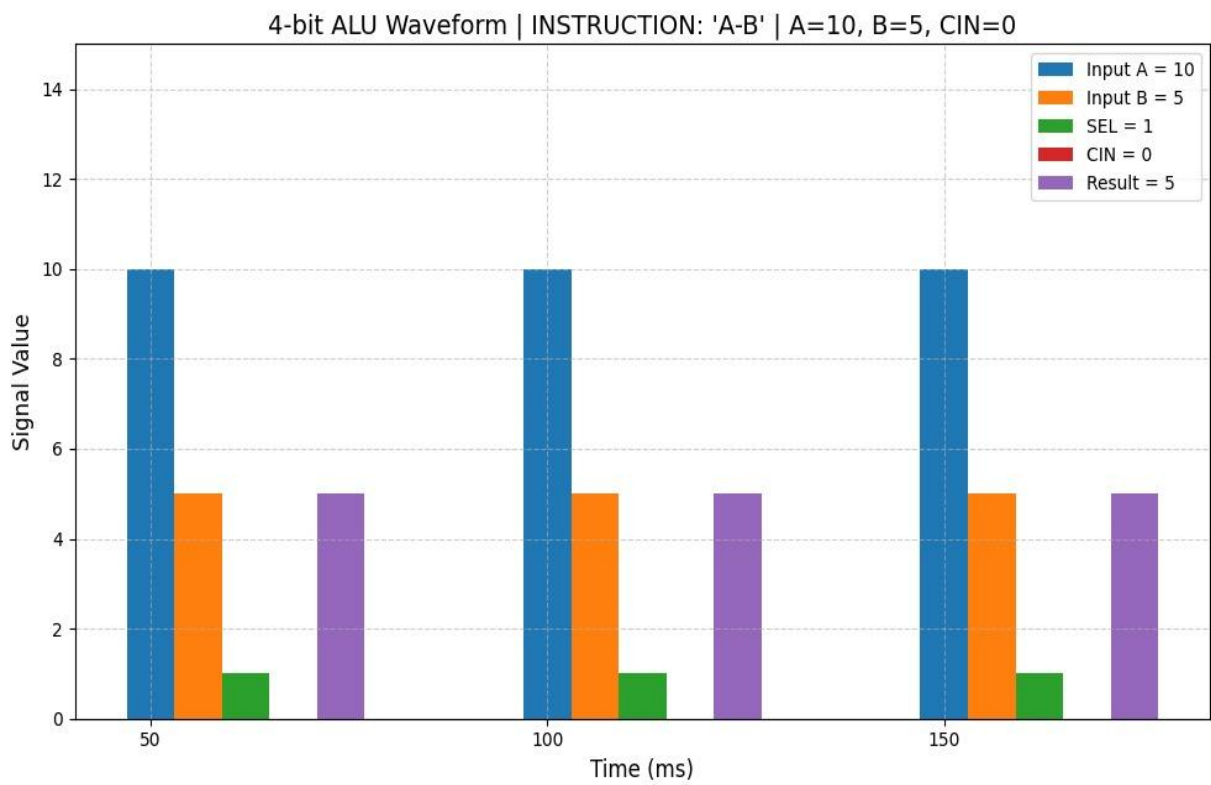


Figure 10: A-B

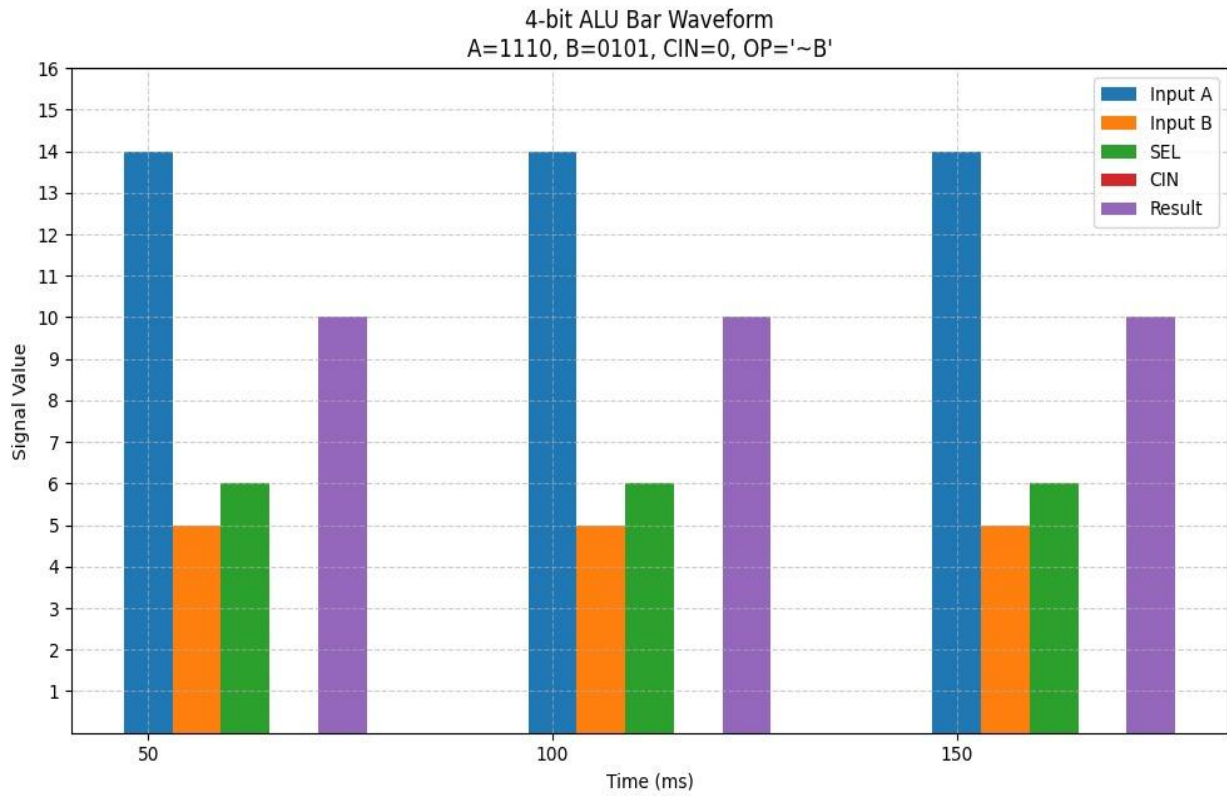


Figure 11: Operation NOT B

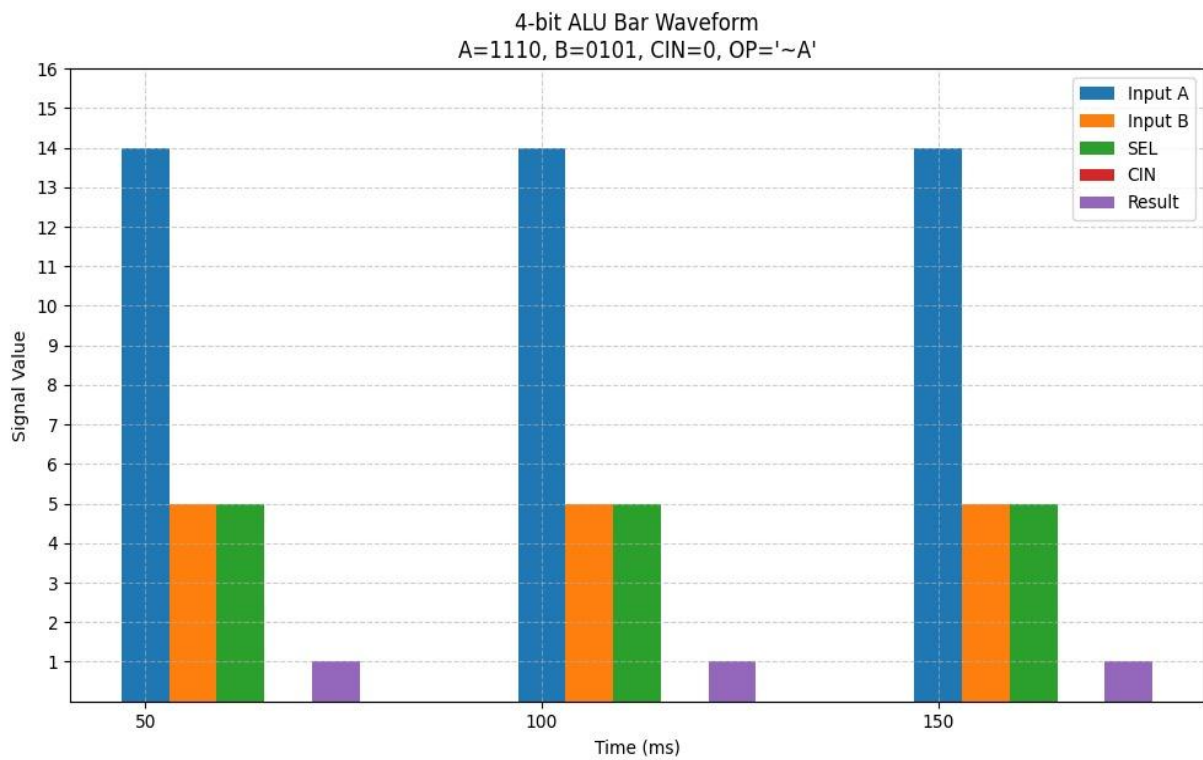


Figure 12: Operation NOT A

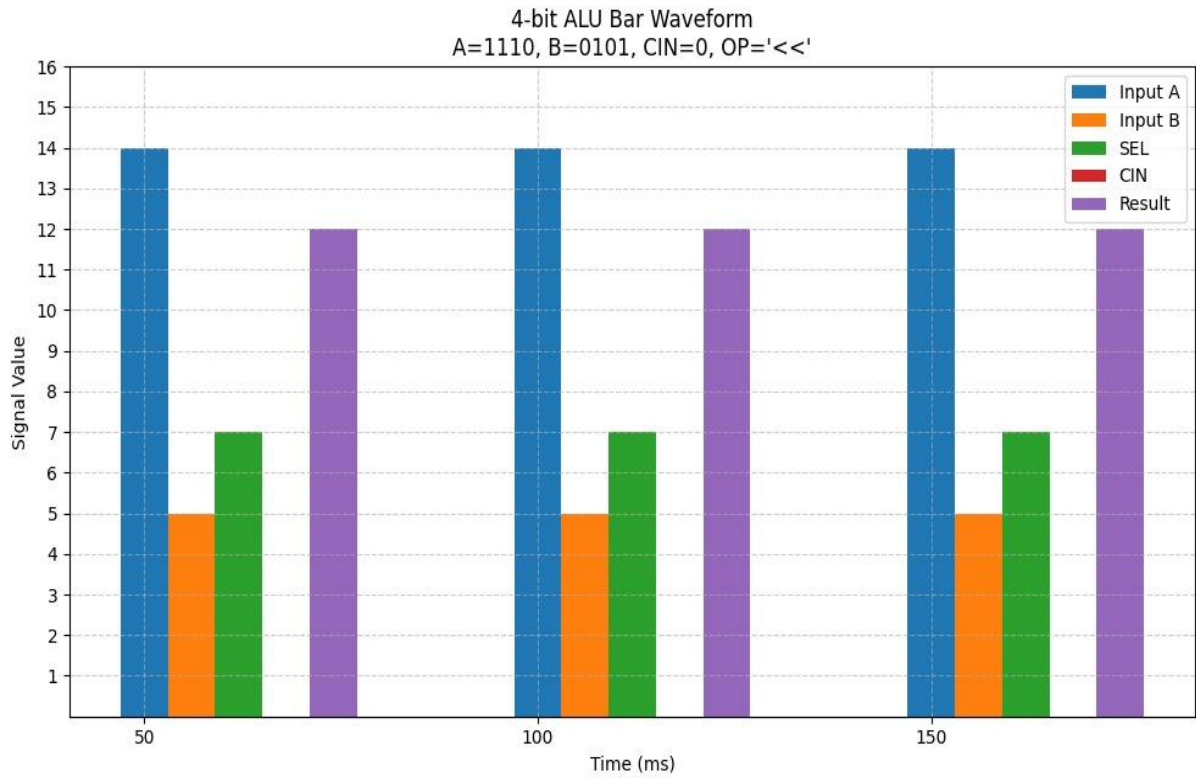


Figure 13: LEFT SHIFT A

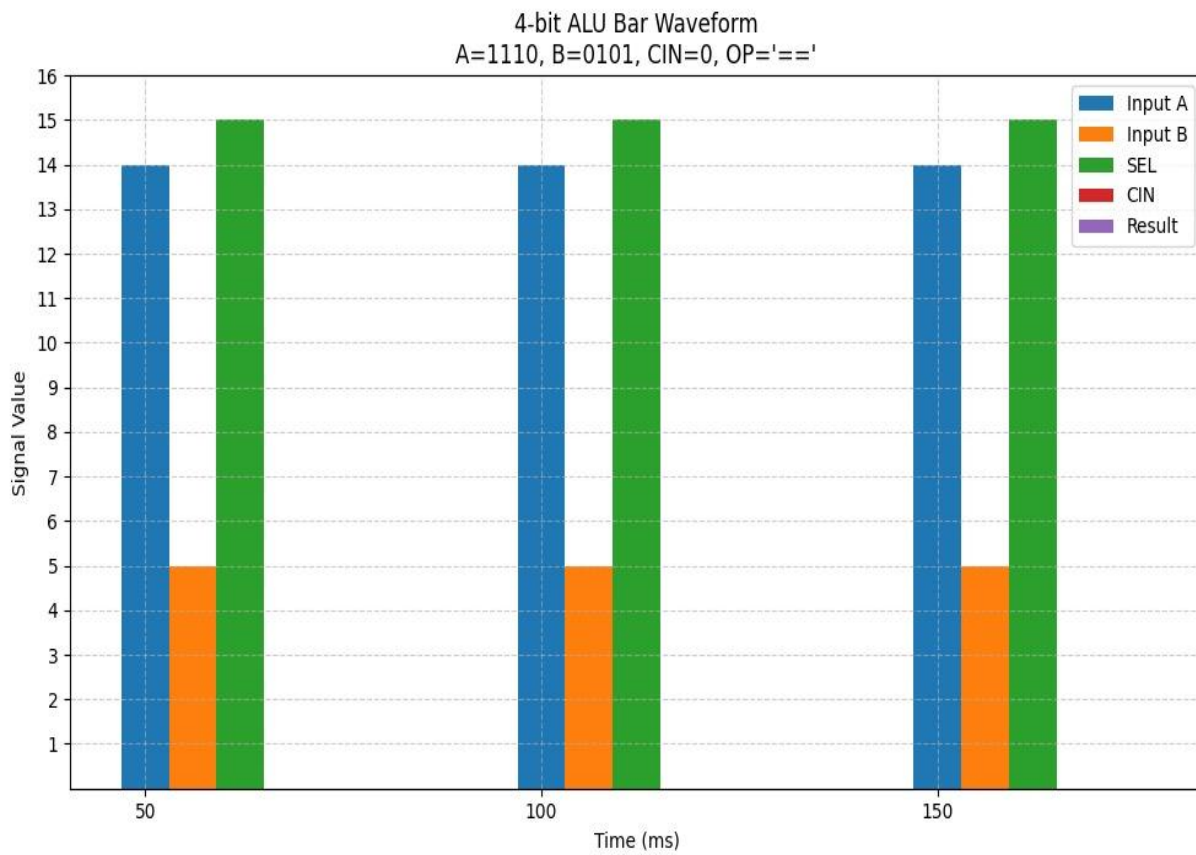


Figure 14: EQUALITY Operation

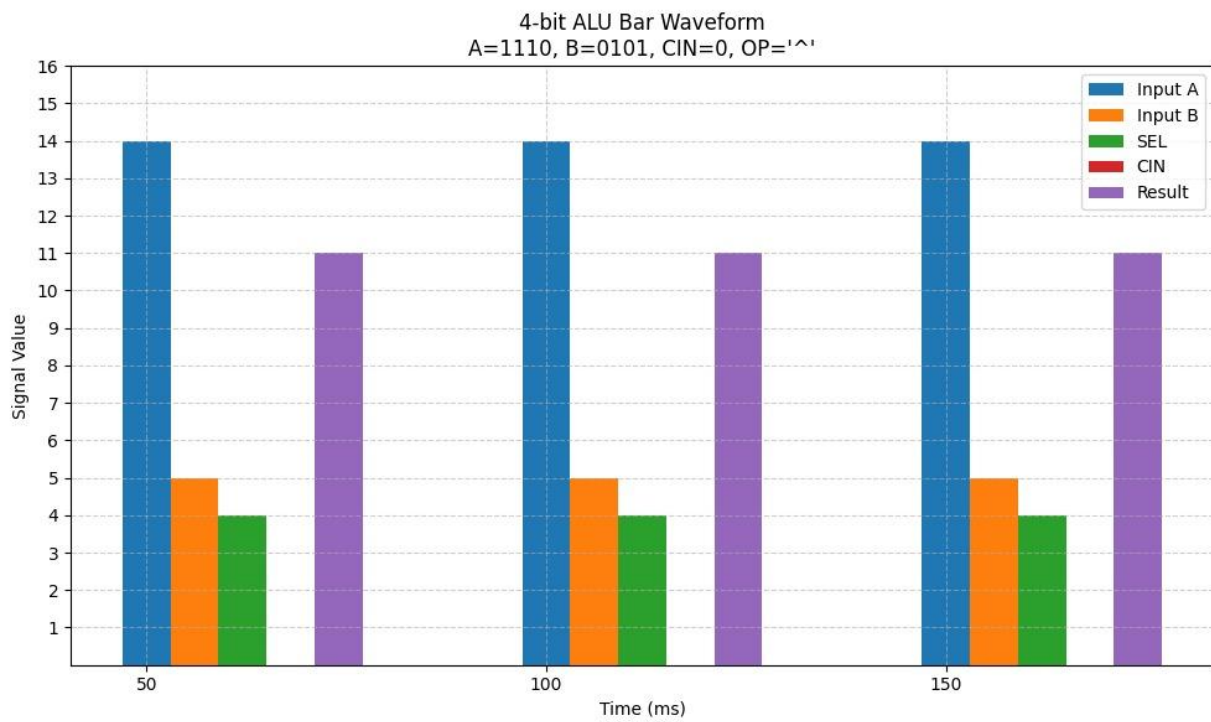


Figure 15: XOR Operation

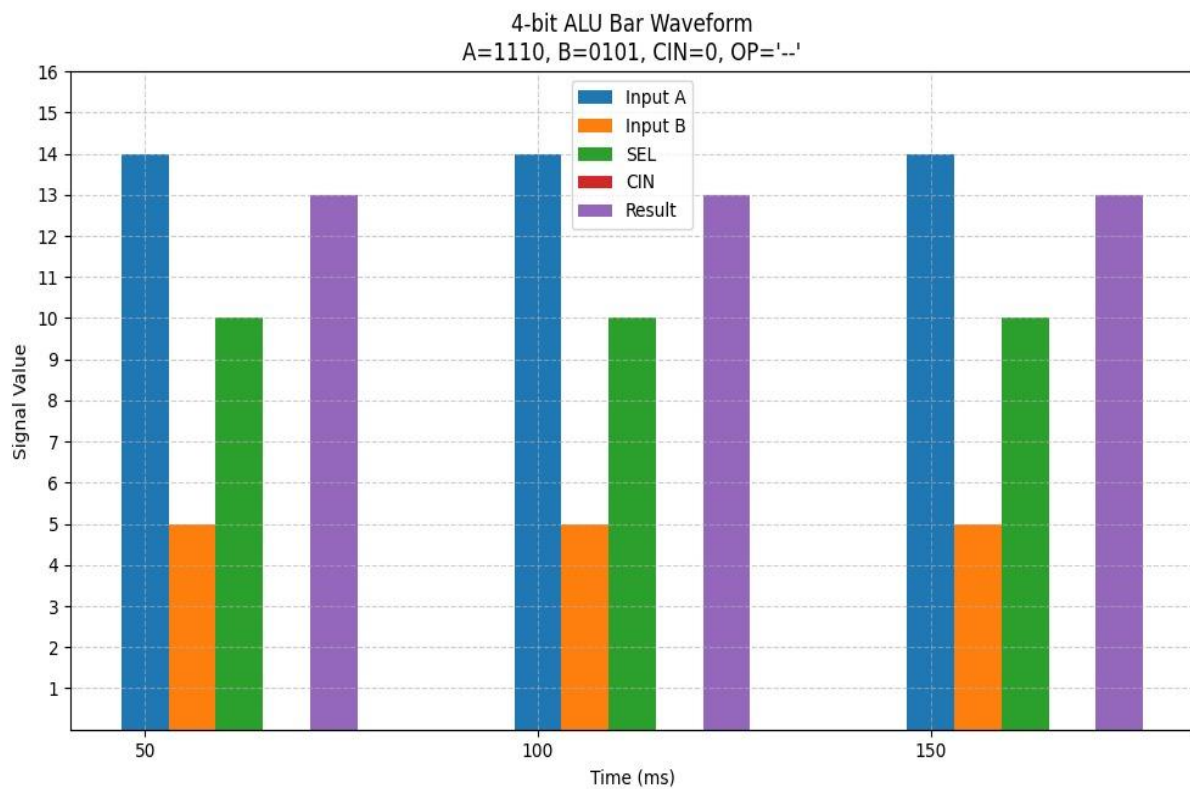


Figure 16: Decrement A

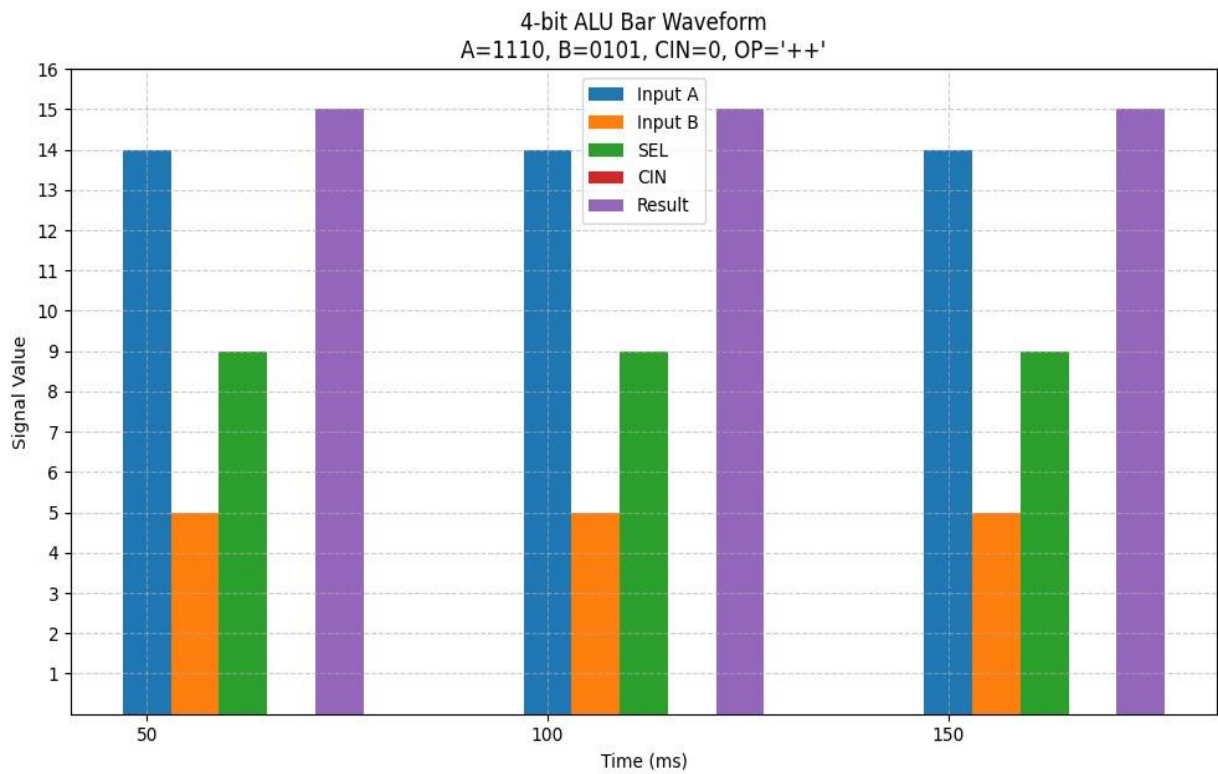


Figure 17: Increment operation

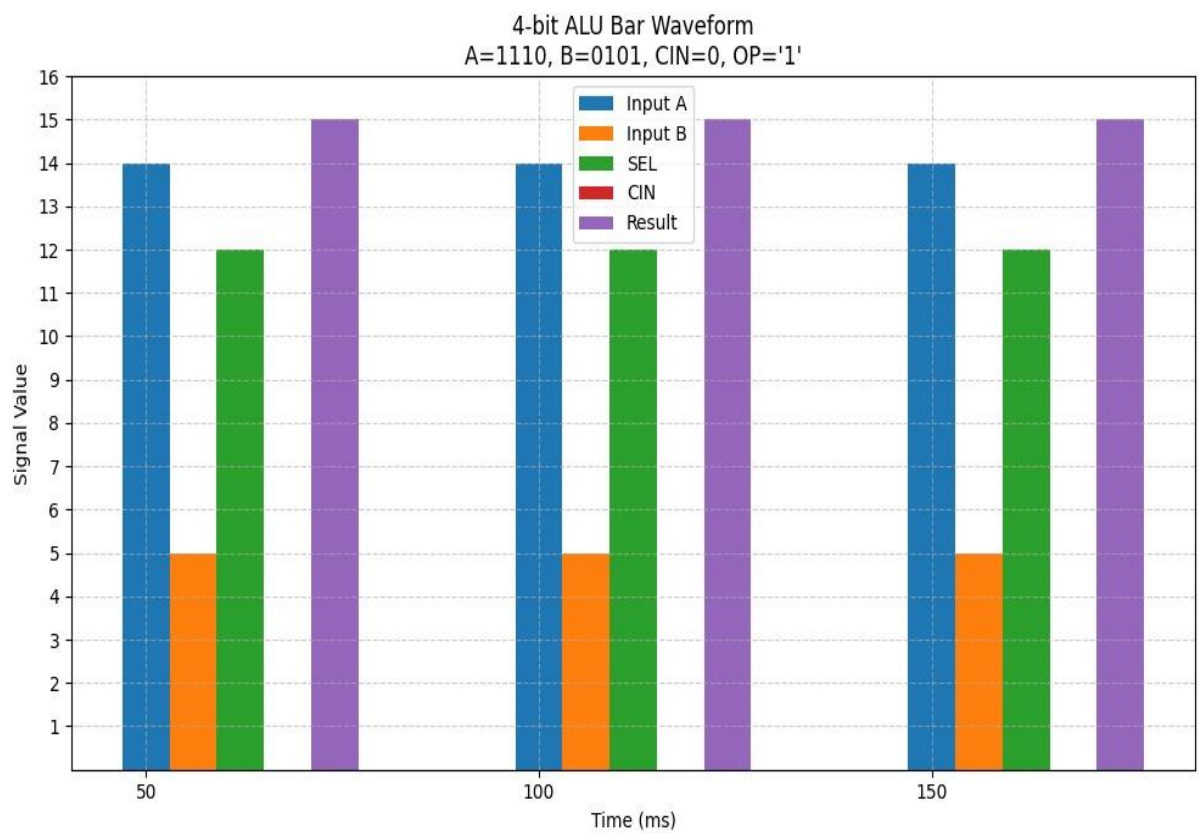


Figure 18: SET Operation

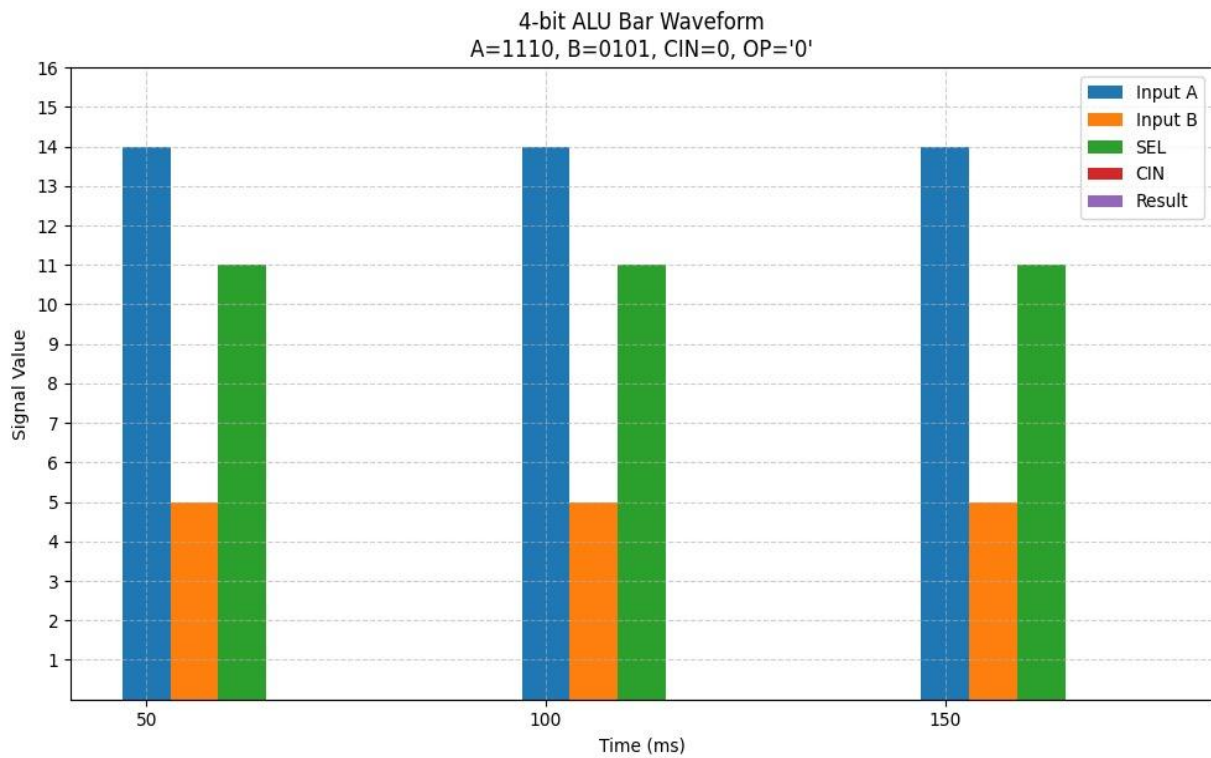


Figure 19: CLR Operation

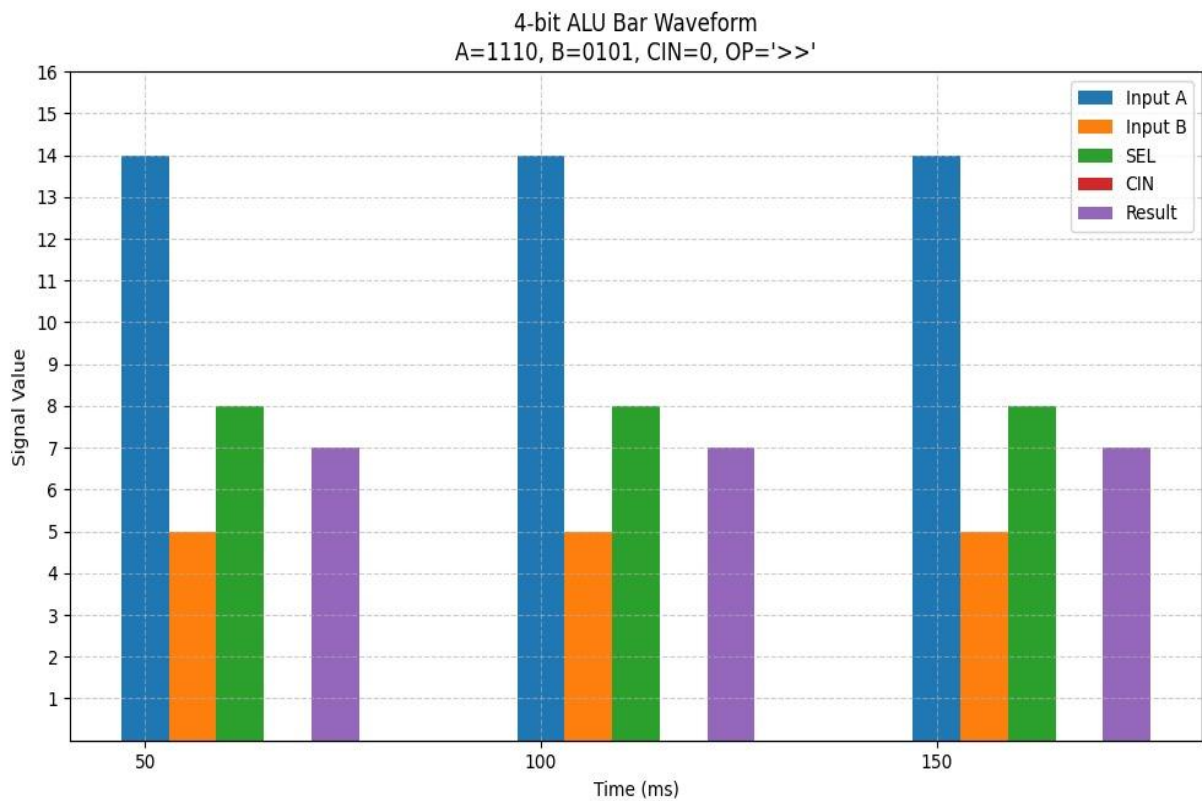


Figure 20: RIGHT SHIFT A

16 operations performed according to the Truth- Table is from the figure 5 to figure 20.

Result

➤ **Real Time Implementation:**

In this research, we propose the design and implementation of a real-time calculator using memristor-based logic that is capable of performing fundamental arithmetic operations—addition (+), subtraction (−), multiplication (*), and division (/). The architecture utilizes four user-defined inputs (a, b, c, d), allowing for flexible computational modeling. What distinguishes this system is that the arithmetic operation itself is also dynamically selected by the user through custom-defined operational equations. The calculator not only processes these operations efficiently using memristor logic circuits—which offer advantages in terms of non-volatility, low power consumption, and compact design—but also provides a visual output in the form of a graphical representation of the results. This integration of memristive computing with real-time graph plotting enhances user interaction and enables the analysis of arithmetic behavior under different input scenarios, making it particularly relevant for applications in neuromorphic computing, embedded systems, and low-power processors. The design showcases the potential of memristors to perform logic-in-memory operations while simultaneously supporting real-time visualization.

Algorithm: Memristor-Based Real-Time Calculator

Step 1: Initialization

- 1.1 Start the system.
- 1.2 Initialize memristor-based logic components.
- 1.3 Set up input interface to accept user-defined inputs (a, b, c, d).
- 1.4 Set up input interface for selecting the arithmetic operation or user-defined equation.

Step 2: Input Collection

- 2.1 Prompt the user to input four values: a, b, c, and d.
- 2.2 Prompt the user to input the desired arithmetic operation or expression (e.g., $a + b$, $a * c - d$, etc.).

Step 3: Parsing and Validation

- 3.1 Parse the user-defined expression.
- 3.2 Check for syntax errors or invalid operations (e.g., division by zero).
- 3.3 Validate the presence of only defined variables (a, b, c, d) and arithmetic operators (+, −, *, /).

Step 4: Logic Implementation Using Memristors

4.1 Map each arithmetic operation to its corresponding memristor logic circuit:

- Use memristor-based full adder/subtractor for + and -
- Use memristor logic gates (e.g., NAND-based multiplier units) for *
- Use iterative division circuits for /

4.2 Convert input variables into logic-compatible binary signals.

4.3 Process the expression using the configured memristor logic circuits.

4.4 Store the output result from the memristor circuit.

Step 5: Output Handling

5.1 Convert the binary output back to decimal format.

5.2 Display the result to the user.

Step 6: Graphical Representation

6.1 Initialize the graph plotting interface (e.g., using matplotlib or GUI graph tools).

6.2 Plot the input values and the corresponding output result(s).

- X-axis: Inputs or operation steps
- Y-axis: Output results

6.3 Display the graph in real time.

Step 7: Loop or Terminate

7.1 Ask the user if another calculation is needed.

- If Yes, go back to Step 2.
- If No, proceed to Step 8.

Step 8: Termination


8.1 Shut down the memristor circuit operations.

8.2 Clear all buffers and release memory.

8.3 End the program.

Output Analysis:

```
Enter integer for A (0-15): 1
Enter integer for B (0-15): 2
Enter integer for C (0-15): 3
Enter integer for D (0-15): 4
Enter expression using A, B, C, D (e.g. A/B + C*D - A): A+B*C/A+D
```

 Result = 11.0

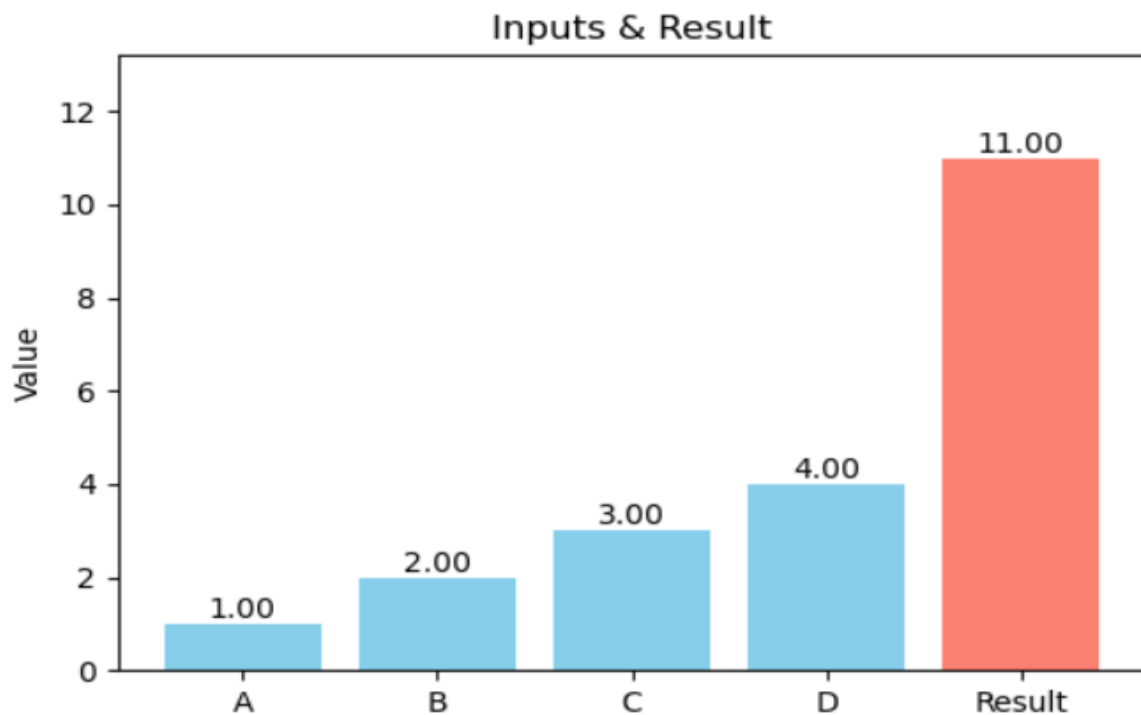


Figure 21: User define INPUT, Equation Operation along with Graph

```
Continue? (y/n): y
Enter integer for A (0-15): 16
Error: Must be 0-15
Continue? (y/n): y
Enter integer for A (0-15): 6
Enter integer for B (0-15): 9
Enter integer for C (0-15): 18
Error: Must be 0-15
Continue? (y/n): y
```

Figure 22: For Wrong INPUT


```

Continue? (y/n): y
Enter integer for A (0-15): 7
Enter integer for B (0-15): 5
Enter integer for C (0-15): 6
Enter integer for D (0-15): 9
Enter expression using A, B, C, D (e.g. A/B + C*D - A): A+B+C+D

```

Result = 27

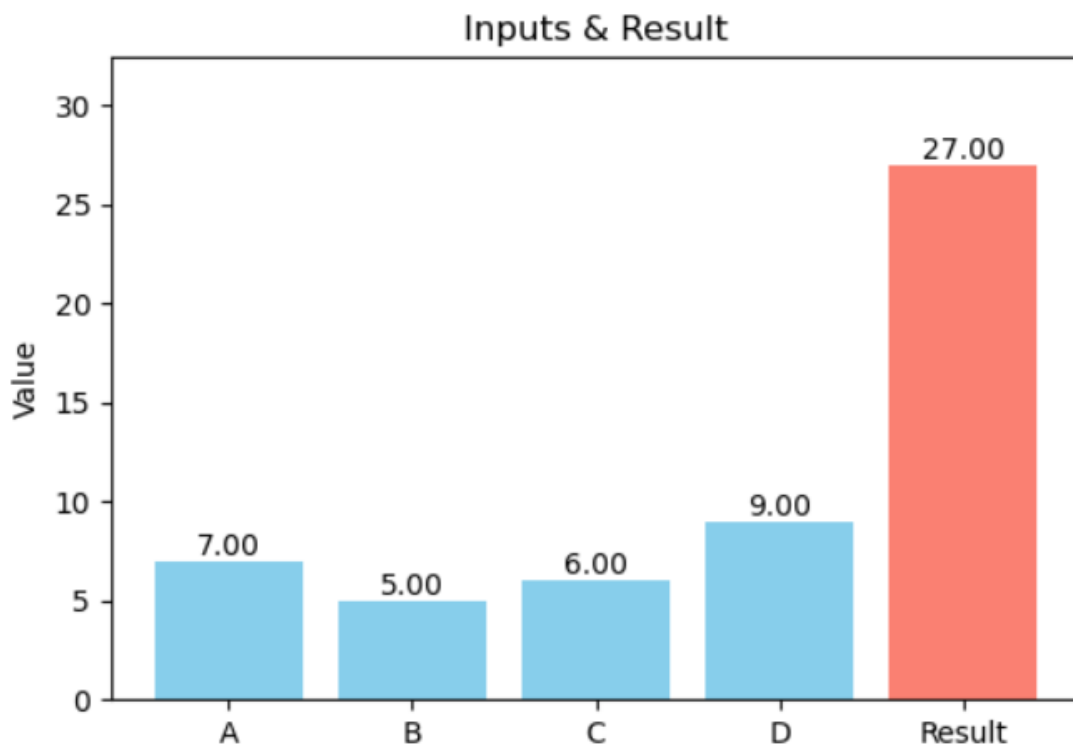


Figure 23: User define INPUT, Equation Operation along with Graph

➤ **Future Scope:**

The memristor-based 4-bit ALU project represents a significant step toward next-generation computing hardware. With the explosive growth in data-driven applications, IoT devices, artificial intelligence, and edge computing, there is a compelling need for compact, fast, and energy-efficient processing units. This project aligns with these trends and opens several research and industrial pathways.

1. Expansion to multi-bit ALUs and Complex Processors

The current 4-bit design can be scaled to 8-bit, 16-bit, 32-bit, or even 64-bit ALUs. Memristor devices offer ultra-small form factors, making them ideal for high-density logic circuits.[15][16].

2. Integration with Neuromorphic Computing Platforms

Memristors can emulate synaptic plasticity, making them a natural fit for neuromorphic computing. A memristor-based ALU could be integrated into a neuromorphic chip to perform both learning and arithmetic tasks natively on hardware, eliminating the need for von Neumann-style architecture [17][18].

3. Reconfigurable Logic and Adaptive ALUs

Future ALUs could leverage the reconfigurable resistance states of memristors to build adaptive systems. These could be used in FPGAs or reconfigurable computing fabrics to change logic behaviour dynamically, making them ideal for mission-critical or space-based systems where remote updates are crucial [19][20].

4. Edge Computing and Ultra-Low Power Devices

IoT and wearable devices require computation at the edge, where power and area constraints dominate. Memristor-based ALUs are inherently low-power and compact, offering an optimal solution for embedded AI inference engines, wearable health monitors, and environmental sensors [21][22].

5. 3D Integrated Circuits and Crossbar Architectures

Memristors support 3D stacking and crossbar architectures, which can be exploited to integrate the ALU directly above memory layers or alongside other compute units. This approach drastically reduces latency, increases throughput, and can enable brain-inspired architectures [23][24].

6. Quantum-Classical Hybrid Co-Processing

As quantum computers evolve, memristor-based ALUs can be developed as classical co-processors to support quantum operations like pre- and post-processing of data or error correction, due to their high-speed, non-volatile characteristics [25].

7. Resilient and Fault-Tolerant Systems

Given their endurance and multi-state capability, future ALUs can use redundancy in memristor circuits to enable self-repair, dynamic rerouting, and fault-tolerant execution. This is particularly useful in aerospace, military, and nuclear applications where reliability is critical [26].

8. Hardware Implementation of Cryptographic Logic

Memristor logic units can implement cryptographic functions like modular arithmetic, prime generation, and hashing at hardware level with enhanced speed and resistance to reverse engineering, benefiting secure computing [27].

9. AI-Driven Design Automation for Memristive Circuits

As memristor modeling becomes more standardized, machine learning techniques can be used to optimize layouts, reduce leakage, and improve performance through AI-based electronic design automation (EDA) tools [28].

➤ **Benefits:**

1. Low Power Consumption

Memristor-based circuits consume significantly less power due to their near-zero static leakage and energy-efficient switching behaviour. This makes them ideal for battery-operated systems and remote sensors where energy efficiency is crucial [15][21].

2. Non-Volatile Logic-in-Memory Design

Memristors retain their state even when power is off. This allows them to function as both logic and memory, reducing latency and energy associated with data movement between processor and memory, overcoming the von Neumann bottleneck [17][22].

3. Smaller Footprint and Higher Integration Density

Memristors can be fabricated at nanometre scale, enabling very high-density logic circuits. Their small size and compatibility with 3D integration provide an advantage over traditional CMOS transistors [18][23].

4. Faster Computational Speed

Due to reduced interconnect lengths and the ability to perform operations directly in memory, memristor-based ALUs can achieve faster speeds, especially in highly parallel or reconfigurable environments [19][24].

5. Durability and Endurance

Modern memristors have been shown to endure over 10^{12} write cycles, making them suitable for long-term industrial use, especially in environments demanding high reliability [26].

6. Green and Sustainable Technology

With reduced silicon area, lower fabrication complexity, and ultra-low power operation, memristors support environmentally friendly and sustainable computing practices [27].

7. Security Advantages

The non-linearity and variability of memristors can be exploited to build physically unclonable functions (PUFs) and secure encryption hardware. A memristor-based ALU inherently offers better resistance to side-channel and reverse engineering attacks [28].

8. Cost-Effective Fabrication

Memristors can be fabricated using existing CMOS-compatible processes, which lowers the barrier for industrial adoption and mass production. This makes them attractive for consumer electronics and low-cost embedded systems [20][25].

Conclusion:

In this project, we have successfully implemented a 4-bit Arithmetic Logic Unit (ALU) using memristor-based logic principles simulated through Python. The approach combined the fundamental advantages of memristor technology—such as non-volatility, low power consumption, and compact size—with the flexibility and clarity of Python for modelling and visualization.

The designed ALU is capable of performing fundamental arithmetic operations, including addition, subtraction, multiplication, and division, based on user-defined inputs and logic expressions. Furthermore, the simulation not only executed the operations correctly but also provided graphical representations of the input-output behaviour, showcasing real-time operation tracking and result visualization.

This project marks an essential step in validating the potential of memristor logic in computational systems. By leveraging software-based simulations, we demonstrated how memristor devices can emulate logic functions traditionally performed by CMOS circuits, thereby offering an alternative paradigm for next-generation hardware design.

Our implementation provides a foundational framework for extending the ALU to more complex and scalable designs, including 8-bit or 16-bit systems, parallel computation arrays, or neuromorphic processing units. The outcomes also highlight the feasibility of integrating memristor logic into reconfigurable systems, edge devices, and AI-centric architectures.

In summary, this project not only reinforces the theoretical promise of memristors in digital logic design but also provides a practical, programmable, and visual platform to explore their real-world applications in computing.

Reference:

1. Wong, H. S. P., & Salahuddin, S. (2015). *Memory leads the way to better computing*. Nature Nanotechnology, 10(3), 191–194. <https://doi.org/10.1038/nnano.2015.29>
2. Chua, L. (1971). *Memristor - The missing circuit element*. IEEE Transactions on Circuit Theory, 18(5), 507–519. <https://doi.org/10.1109/TCT.1971.1083337>
3. Zidan, M. A., Strachan, J. P., & Lu, W. D. (2018). *The future of electronics based on memristive systems*. Nature Electronics, 1(1), 22–29. <https://doi.org/10.1038/s41928-017-0006-8>
4. Borg Hetti, J., et al. (2010). “*Memristive*” switches enable “*stateful*” logic operations via material implication. Nature, 464(7290), 873–876. <https://doi.org/10.1038/nature08940>
5. Strukov, D. B., Snider, G. S., Stewart, D. R., & Williams, R. S. (2008). *The missing memristor found*. Nature, 453(7191), 80–83. <https://doi.org/10.1038/nature06932>
6. Chua, L. O. (1971). Memristor—The Missing Circuit Element. IEEE Transactions on Circuit Theory.
7. Shirinzadeh, S. et al. (2018). Logic Design Using Memristors: An Emerging Technology. IEEE ISMVL.
8. Mohanty, S. P. (2013). Memristor: From Basics to Deployment. IEEE Potentials.
9. Huang, G. M., Ho, Y., & Li, P. (2010). Memristor System Properties and Its Design Applications. IEEE.
10. Mladenov, V. M., & Kirilov, S. M. (2015). Memristor Modeling in MATLAB® & PSPICE®. ECMS Proceedings.
11. Abdalla, S. (2020). Memristor Operation and Applications. University of Nevada, Las Vegas.
12. Apollos, E. (2019). Memristor Theory and Mathematical Modelling. IJCA, Vol. 178.
13. Gao, C. et al. (2020). Memristor-Based Logic Gate Circuit. IEEE Conference.
14. Sahoo, S., & Prabakaran, S. R. S. (2020). Nanoionic Memristor Equipped Arithmetic Logic Unit using VTEAM Model. IEEE IC-GET.
15. Chua, L. (1971). Memristor—The Missing Circuit Element. IEEE Transactions on Circuit Theory, 18(5), 507–519.
16. Strukov, D. B., Snider, G. S., Stewart, D. R., & Williams, R. S. (2008). The missing memristor found. Nature, 453(7191), 80–83.
17. Yang, J. J., Strukov, D. B., & Stewart, D. R. (2013). Memristive devices for computing. Nature Nanotechnology, 8(1), 13–24.
18. Zidan, M. A., Strachan, J. P., & Lu, W. D. (2018). The future of electronics based on memristive systems. Nature Electronics, 1, 22–29.
19. Gao, L., et al. (2013). Programmable CMOS/memristor threshold logic. IEEE Transactions on Nanotechnology, 12(2), 115–126.
20. Borghetti, J., et al. (2010). A hybrid nanomemristor/transistor logic circuit capable of self-programming. PNAS, 107(28), 12741–12745.
21. Mehonic, A., & Kenyon, A. J. (2016). Emulating the electrical activity of the brain with memristors. Frontiers in Neuroscience, 10, 57.
22. Hu, M., et al. (2016). Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. DAC '16.
23. Lin, P., et al. (2020). Three-dimensional memristor crossbar arrays. Nature Electronics, 3(5), 225–232.

24. Li, C., et al. (2018). Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature Communications*, 9, 2385.
25. Suri, M., et al. (2013). Bio-inspired stochastic computing using binary CBRAM synapses. *IEEE Transactions on Electron Devices*, 60(7), 2402–2409.
26. Prezioso, M., et al. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550), 61–64.
27. Wang, Z., et al. (2018). Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing. *Nature Materials*, 17(9), 873–879.
28. Chakraborti, R., et al. (2022). Evolutionary Design of Memristive Circuits for Arithmetic Processing. *IEEE Access*, 10, 14582–14591.