TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Report

On

Logic Gate simulator using C++

Submitted By:

Saksham Neupane (THA081BCT032)

Sandeep Dhungana (THA081BCT034)

Sandesh Acharya (THA081BCT035)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

August, 2025

DECLARATION

We hereby declare that the report of the project entitled **"Logic Gate Simulator using C++"** which is being submitted to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus,** in the partial fulfillment of the requirements for the award of the Degree of Bachelor of Engineering in **Computer Engineering,** is a Bonafide report of the work carried out by us

. The materials contained in this report have not been submitted to any University or Institution for the award of any degree and we are the only author of this complete work and no sources other than the listed here have been used in this work.

Saksham Neupane (THA081BCT032) _____

Sandeep Dhungana (THA081BCT034) _____

Sandesh Acharya (THA081BCT035) _____

Date: August, 2025

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus**, a major project work entitled **"Logic Gate simulator using C++"** submitted by **Saksham Neupane**, **Sandeep Dhungana** and **Sandesh Acharya** in partial fulfillment for the award of the **Bachelor's Degree in Electronics and Computer Engineering**. The project was carried out under special supervision and within the time frame prescribed by the syllabus.

We found the students to be hardworking, skilled, and ready to undertake any related work to their field of study, and hence we recommend the award of partial fulfillment of the Bachelor's degree in Electronics and Computer Engineering.

_____

Project Supervisor

Er. Prajwol Pakka

Department of Electronics and Computer Engineering, Thapathali Campus

_____

Head of Department

Er. Umesh Kanta Ghimire

Department of Electronics and Computer Engineering, Thapathali Campus

COPYRIGHT

The author has agreed that the library, Department of Electronics and Computer Engineering, IOE, Thapathali Campus, may make this report freely available for inspection. Moreover, the author has agreed that the permission for extensive copying of this project work for scholarly purposes may be granted by the professor/lecturer who supervised the project work recorded herein or, in their absence, by the head of the department. It is understood that recognition will be given to the author of this report and the Department of Electronics and Computer Engineering, IOE, Thapathali Campus in any use of the material of this report. Copying or publication or other use of this report for financial gain without the approval of the Department of Electronics and Computer Engineering, IOE, Thapathali Campus, and the author's written permission is prohibited. Request for permission to copy or to make any use of the material in this project in whole or part should be addressed to the Department of Electronics and Computer Engineering, IOE, Thapathali Campus.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude towards the Institute of Engineering, Tribhuvan University, for the inclusion of the major project in the course of Bachelor's in Electronics and Computer Engineering. We are also thankful towards our teachers and supervisor Er. Prajwal Pakka, Er. Anup Shrestha and the Department of Electronics and Computer Engineering, Thapathali Campus, for providing us with the resources and support needed for this project.

Saksham Neupane (THA081BCT032)

Sandeep Dhungana (THA081BCT034)

Sandesh Acharya (THA081BCT035)

ABSTRACT

This project describes the design and implementation of an interactive logic gate simulator using C++. The application provides a graphical user interface (GUI) for users to place, connect, and observe the behavior of fundamental logic gates, including AND, OR, NOT, NAND, and NOR gates. The simulator is designed as an educational tool to help students and enthusiasts visually understand digital logic concepts in a real-time environment. It uses the Raylib library for GUI and graphics, enabling real-time visual feedback on input and output signals as users interact with the circuit. The system is built on a modular, object-oriented design to facilitate future enhancements and the addition of new components.

Keywords: Logic gates, C++, GUI, Raylib, Digital logic, Simulation, Object-oriented programming, Educational tool

Table of Contents

List of Abbreviations

GUI: Graphical User Interface

IDE: Integrated Development Environment

OOP: Object-Oriented Programming

CPU: Central Processing Unit

1. INTRODUCTION The field of digital electronics is foundational to modern computing and communication systems. Understanding the behavior of logic gates is a critical first step for students and engineers. While physical circuits offer a hands-on learning experience, they can be costly and time-consuming to set up and modify. This project addresses these challenges by developing an interactive, software-based logic gate simulator that makes learning digital logic more accessible and visual.

1.1 Background Introduction

Digital logic forms the basis of all digital circuits and systems. At its core, it relies on fundamental components known as logic gates, which perform basic logical operations on binary inputs. Software simulators have become indispensable tools for learning and design, allowing users to build and test virtual circuits without the need for physical hardware. These simulators provide a safe, flexible, and interactive environment for experimentation.

1.2 Motivation

The primary motivation for this project is to create an educational tool that simplifies the learning process for digital logic. By providing a real-time, visual platform, the simulator aims to make abstract concepts tangible. Furthermore, the project offered an opportunity to gain practical experience in C++ application development, GUI design, and object-oriented programming (OOP) principles. It combines theoretical knowledge with practical skills to produce a functional and demonstrable project.

1.3 Problem Definition

Traditional methods for teaching and experimenting with digital logic often rely on physical hardware, which can be expensive, prone to damage, and difficult to reconfigure. This limits the

scale and complexity of circuits that can be built and tested in a typical classroom or home environment. The problem this project solves is the need for an accessible, cost-effective, and flexible alternative that allows users to quickly design, test, and understand digital circuits without these physical constraints.

1.4 Objectives

The main objectives of this project are:

- To develop a real-time logic gate simulator supporting fundamental gates (AND, OR, NOT, NAND, NOR).
- To implement a modular and object-oriented design for representing gates and circuits.
- To create a graphical user interface (GUI) for intuitive interaction, allowing users to place and wire components.
- To provide real-time visual feedback on signal propagation through the circuit.

1.5 Scope and Applications

The project is a standalone desktop application that runs on Windows. It supports the simulation of combinational logic circuits using the five specified gates. The application is intended as an educational tool for students, a testing environment for simple circuit designs, and a platform for further development with more advanced components.

2. LITERATURE REVIEW

2.1 Digital Logic Simulators Digital logic simulators have a long history, evolving from complex, text-based systems to user-friendly graphical interfaces. While early simulators focused on accurate waveform analysis and circuit timing, modern educational simulators prioritize **ease of use, visual clarity, and real-time interaction** to enhance the learning experience. This shift recognizes that intuitive visual feedback is critical for helping students build a concrete understanding of abstract logical concepts.

2.2 C++ for Application Development C++ is a high-performance, object-oriented language widely used in systems programming, game development, and desktop applications. Its performance and fine-grained control over system resources make it an excellent choice for a real-time simulation application where speed is paramount. The use of **Object-Oriented Programming (OOP) principles** in C++ allows for a modular and scalable design, where each logic gate can be represented as an independent object with its own properties and methods, simplifying the overall architecture.

2.3 GUI Libraries for C++ GUI development in C++ can be complex and time-consuming. While powerful libraries like **Qt** and **wxWidgets** offer extensive features, their larger scope and steeper learning curves can be a barrier for smaller projects. For this project, we chose **Raylib**, a simple and easy-to-use library that is specifically well-suited for interactive, educational, and game-like applications. Raylib provides all the necessary functions for drawing shapes, handling user input, and creating a **responsive visual experience** without the overhead of a more comprehensive framework, which aligns perfectly with our goal of a lightweight and focused learning tool.

3. SYSTEM ARCHITECTURE AND METHODOLOGY

The Logic Gate Simulator is a single-threaded desktop application with a clear, modular architecture. The methodology follows a **top-down approach**, where the high-level components were designed first, followed by the detailed implementation of the classes and functions within each component.

3.1 Block Diagram and System Architecture

The application's architecture is based on three main, interconnected components: the **Logic Core**, the **User Interface (GUI)**, and the **Input/Output Manager**. The **Input/Output Manager** acts as the crucial link, translating user interactions from the GUI into commands that the **Logic Core** can process, and then returning updated state information to the GUI for real-time display.

- **Logic Core:** This component is the brain of the simulator, responsible for the backend logic. It contains a collection of classes for each type of logic gate (AND, OR, NOT, etc.) and a **Circuit Manager** that handles the connections and state propagation throughout the circuit.
- **User Interface (GUI):** Built with the Raylib library, the GUI handles all visual aspects of the application. This includes rendering the gates, drawing the connecting wires, and displaying the state of all inputs and outputs.
- **Input/Output Manager:** This component processes user interactions such as mouse clicks and key presses. It translates these actions into specific commands—like placing a gate or toggling an input switch—that are sent to the Logic Core. It also handles the real-time visualization of the simulation results back to the user via the GUI.

3.2 Data Flow Diagram

The simulation process follows a simple, cyclical workflow that ensures a responsive and interactive experience. User actions on the GUI are captured by the **Input/Output Manager**, which triggers a calculation within the **Logic Core**. The Logic Core processes the new input states and propagates the logical changes throughout the circuit. Once the calculations are complete, the updated circuit state is sent back to the GUI for real-time visualization, completing the loop.

- **Placement Mode:** The user selects and places a gate on the canvas via the GUI.
- **Wiring Mode:** The user connects the outputs and inputs of different gates to form a circuit.
- **Toggling Phase:** The user can interactively toggle input switches to change the circuit's input.
- **Logic Propagation:** The **Logic Core** calculates the new output state of each gate based on its inputs, repeating this process until all signals have stabilized.
- **Real-time Visualization:** The GUI updates the visual representation of the circuit to show the new state of the wires and gate outputs instantly.

3.3 Tools and Environment

- **Programming Language: C++** was chosen for its performance, which is essential for a real-time simulation application.

- **GUI Library: Raylib** was selected for its simplicity and efficiency in creating interactive graphical applications.
- **IDE: Visual Studio** was used for its robust debugging tools and comprehensive development environment.
- **Version Control: GitHub** was used to manage the project's code, facilitate version tracking, and handle collaborative development.

4. FEASIBILITY ANALYSIS

A feasibility study was conducted to evaluate the practicality and viability of developing the Logic Gate Simulator project. The analysis considered technical, operational, economic, and time-based factors.

4.1 Technical Feasibility

The project's technical feasibility is confirmed by the availability of the necessary tools and technologies. C++ is a powerful language well-suited for simulation and GUI development. The Raylib library provides a robust and easy-to-use framework for creating the interactive interface. Furthermore, the object-oriented design allows for a clean separation of concerns, making the system easier to manage and debug.

4.2 Operational Feasibility

The simulator is designed to be an intuitive and user-friendly tool. The graphical interface allows for simple drag-and-drop placement of gates and click-and-drag wiring. The real-time visual

feedback of signal propagation provides immediate reinforcement of digital logic concepts. These features ensure that the system will be effective in achieving its educational goals.

4.3 Economic Feasibility

The project is highly economically feasible. It relies on free and open-source software, including the C++ compiler (GCC or MSVC), the Raylib library, and the Visual Studio IDE. There are no licensing or hardware costs beyond a standard computer, making the project cost-effective to develop and deploy.

4.4 Time Feasibility

Time feasibility ensures the project can be completed within the time constraints. The project timeline includes distinct phases for requirement analysis, design, coding, and testing, all of which are achievable within a typical academic schedule. By leveraging established libraries and tools, such as the **Raylib library** to simplify GUI development and **GitHub** to streamline collaboration, the project's development complexity is reduced. This proactive approach ensures adherence to the schedule, confirming its overall feasibility.

5. SCOPE AND APPLICATIONS

This section defines the boundaries and limitations of the project, as well as the potential uses and target audience for the developed software.

5.1 Scope

The project focuses on developing a functional, standalone desktop application for simulating combinational digital logic circuits. The scope is defined by the following features:

- **Core Logic Gates:** The simulator will support fundamental gates, including AND, OR, NOT, NAND, and NOR.
- **Graphical Interface:** A user-friendly graphical user interface (GUI) will be implemented to allow for intuitive circuit design through drag-and-drop placement and click-and-drag wiring.
- **Real-time Simulation:** The application will provide real-time visual feedback on signal propagation and logic states (high/low).
- **Combinational Logic Only:** The initial version of the simulator is limited to combinational circuits and does not include components for sequential logic (e.g., flip-flops or memory).

5.2 Applications

The logic gate simulator is designed to be a valuable educational tool with several key applications:

- **Educational Tool:** It can be used by students and instructors in academic settings to visually demonstrate and experiment with fundamental digital logic concepts.
- **Self-Study:** It serves as a personal learning tool for anyone interested in understanding how logic gates work without the need for physical components or expensive hardware.
- **Prototyping:** It can be used by hobbyists and beginners to quickly prototype and test simple circuit designs before moving to a physical implementation.

- **Project Foundation:** The modular design of the simulator makes it a solid foundation for future development, allowing for the addition of more complex gates, sequential logic components, or advanced features.

6. IMPLEMENTATION DETAILS

This section provides specifics about the architectural design and code implementation of the Logic Gate Simulator, detailing how the software was constructed to deliver a real-time, interactive user experience.

6.1 Architectural Overview

The core of the application is a **main event loop** that continuously checks for user input, updates the state of the circuit's gates, and renders the scene. This loop ensures a real-time, responsive experience by processing events and redrawing the screen in rapid succession. The system's design is highly modular, with different components handling distinct responsibilities, which makes it easy to maintain and extend.

6.2 Gate Classes and Object-Oriented Design

Each type of logic gate is implemented as a separate **class**. This object-oriented approach encapsulates the gate's properties (position, input/output states) and its behavior (the specific logical operation it performs). This design allows for the easy addition of new gates or circuit components in the future by simply creating a new class that inherits from a common base class.

6.3 User Interface (GUI) Implementation

The GUI is the user's primary point of interaction. It is implemented using **Raylib's drawing functions** for geometric shapes, text, and lines, which creates the visual canvas. The interface includes a sidebar for selecting different gates and a main canvas where the circuit is built. User input from the mouse is used to handle placement, wiring, and toggling of inputs, which the application translates into commands for the simulation engine.

6.4 Logic Propagation and Real-Time Visualization

When an input value changes, a **propagation algorithm** is triggered. This algorithm traverses the circuit's data structure (a list of connected components), starting from the changed input and moving through each connected gate. The algorithm updates the state of each wire and gate in sequence. The GUI then reads these new states and changes the color of the wires and gate output indicators to visually represent the logical 0 and 1 states, providing instant feedback to the user.

6.5 Error Handling

This section describes the measures taken to ensure the stability and reliability of the Logic Gate Simulator by addressing potential errors and unexpected user actions.

6.5.1 User Input and Validation

The application is designed to prevent invalid user inputs that could corrupt a circuit. The system includes validation checks to ensure that gates are placed within the canvas boundaries and that wires are connected only to valid input and output pins. For example, the software will not allow a user to connect a wire from one gate's output to another's, ensuring the integrity of the circuit's data flow. Any attempt at an invalid action is either ignored or prompts a clear, non-intrusive message to the user.

6.5.2 Logic and Runtime Errors

The core simulation loop is designed to be resilient against logical and runtime errors. The propagation algorithm includes a built-in safety mechanism to prevent infinite loops that could be caused by complex feedback circuits. By managing the number of propagation steps, the system ensures that the application remains responsive and stable. Additionally, a robust error-handling system is in place to catch and manage unexpected issues that might arise during the simulation process.

6.5.3 Resource Management

As the project is implemented in C++, meticulous attention was paid to resource management. The application is designed to prevent common errors such as memory leaks by ensuring that all dynamically allocated memory for gates, wires, and other components is properly deallocated when no longer needed. This careful management of system resources allows the simulator to handle complex, large-scale circuits without compromising performance or causing the application to crash.

7. RESULTS AND ANALYSIS

This section presents the results obtained from the implementation and testing of the Logic Gate Simulator. The analysis focuses on the application's functionality, performance, and its effectiveness as an educational tool for digital logic.

7.1 Functional Results

The testing of the application confirmed that all core functionalities were implemented as designed, providing a reliable and interactive experience.

- **Gate Functionality:** The simulator accurately replicated the behavior of all specified logic gates (AND, OR, NOT, NAND, NOR). The output states correctly reflected the inputs according to each gate's truth table.
- **Circuit Construction:** Users can successfully place gates on the canvas and connect them with wires. The system ensures that all connections are valid, preventing illogical circuit configurations.
- **Real-time Visualization:** The GUI provides immediate and clear visual feedback. The wires and outputs of gates change color in real-time to reflect a logical 0 (low) or 1 (high`) state, making the circuit's behavior easy to understand.
- **User Interface:** The interface is intuitive and responsive. Users can easily toggle input switches, place new components, and create complex circuits without any noticeable lag.

7.2 Performance and Usability Analysis

The simulator's performance was evaluated to ensure a smooth and efficient user experience, even with complex circuits.

- **Performance:** The simulation engine's propagation algorithm is highly efficient. The application maintained a high frame rate and immediate response times, even with circuits containing multiple gates and multiple inputs. No significant performance degradation was observed.
- **Usability:** The design of the GUI and the component-based architecture contribute to the application's high usability. The tool is effective as both a learning aid for students and a quick prototyping environment for hobbyists. The visual feedback makes the abstract concepts of digital logic tangible and easy to grasp.
- **Reliability:** The implemented error-handling mechanisms proved effective in preventing crashes and illogical states. The application remained stable under various user inputs, confirming its reliability.

8. OVERALL SYSTEM PERFORMANCE

The performance of the Logic Gate Simulator was evaluated based on its functionality, efficiency, and usability under different scenarios. The analysis focuses on how the application delivers a seamless and responsive experience for the user.

8.1 Scalability and Reliability

The simulator's design demonstrated excellent scalability by handling increasingly complex circuits without a significant drop in performance. The component-based and object-oriented architecture allows the application to manage a large number of gates and connections,

showcasing its ability to handle more than just simple circuits. The system proved to be highly reliable, with the robust error-handling mechanisms preventing crashes and logical errors, ensuring that the simulator remains stable even during intensive user interaction.

8.2 Speed and Usability

The application's speed is a key factor in its effectiveness as an educational tool. The simulation engine's propagation algorithm is highly efficient, allowing for a near-instantaneous update of the circuit's state when an input is changed. This provides the user with real-time feedback, which is crucial for learning. The usability of the system is a direct result of this speed, as the fluid and responsive GUI makes the process of building and testing circuits intuitive and engaging. The combination of speed and usability ensures a seamless and productive experience for the user.

9. FUTURE ENHANCEMENTS

The current implementation provides a solid foundation, and several enhancements could be made in the future to expand its functionality and utility.

- **Add More Logic Components**: The project could be extended to include more advanced gates like XOR and XNOR, as well as components for arithmetic operations (e.g., half-adders, full-adders).
- **Sequential Circuits**: The addition of sequential components like clocks and flip-flops would allow users to simulate memory and state-based circuits.
- **Automatic Truth Table Generation**: A feature to automatically generate and display a truth table for any user-created circuit would be a valuable educational tool.

- **Save/Load Functionality**: The ability to save a circuit design and load it later would allow users to work on more complex projects over time.
- **Multi-User Collaboration**: Implementing network functionality could enable real-time, multi-user editing for remote learning and collaborative projects.

10. CONCLUSION

The Logic Gate Simulator successfully fulfills its objectives by providing an interactive and effective educational tool for understanding fundamental digital logic concepts. The implementation demonstrates:

- An accurate and functional simulation engine for combinational circuits.
- A user-friendly and responsive graphical interface for intuitive circuit design.
- A robust, modular architecture that ensures stability and ease of maintenance.

While the current version meets all core requirements, the project serves as a solid foundation for future enhancements. Overall, the project highlights the practical application of C++ programming, GUI development, and object-oriented design principles. This system offers a valuable starting point for further advancements in computer-aided learning and simulation.