TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A PROJECT REPORT ON

DIGITAL LOGIC SIMULATOR

GUI BASED TOOL

SUBMITTED BY:

NABARAJ BHANDARI (081BCT041)

NIKUNJ BHUSAL (081BCT043)

NIRDESH JOSHI (081BCT044)

SUPERVISED BY:

SENIOR FACULTY MEMBER, DAYA SAGAR BARAL

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

INSTITUTE OF ENGINEERING, PULCHOWK CAMPUS

SUBMISSION DATE:

2082 Bhadra 02, Monday

Acknowledgment

We would like to express our sincere gratitude to our supervisor, **Mr. Daya Sagar Baral**, Senior Faculty Member, Department of Electronics and Computer Engineering, IOE Pulchowk Campus, for his amazing guidance, helpful feedback, and constant support. His knowledge of programming and digital logic helped us understand and build this project better.

This project, **Digital Logic Simulator**, has been carried out as part of the subject **Object-Oriented Programming with C++**, and it would not have been possible without the support provided by our supervisor. His expertise and insights have greatly contributed to the successful design and implementation of this project.

We extend our sincere thanks to the faculty members of the Department of Electronics and Computer Engineering at Pulchowk Campus for providing us with the necessary resources and an excellent learning environment. Their lectures on C++ programming were really helpful for this project. Our heartfelt appreciation goes to our classmates and friends who offered constructive suggestions during its development phases.

Finally, we thank Tribhuvan University for including projects in our studies, letting us use what we learned in a real project. This work has greatly improved our skills in C++ programming, software design, and working as a team.

Abstract

The Digital Logic Simulator is a user-friendly app built with C++ and SFML (Simple Fast Multimedia Library) to provide features like designing, testing, and studying digital logic circuits. It uses object-oriented programming ideas to make a helpful tool for students.

Users can place logic gates like AND, OR, NOT, INPUT, and OUTPUT, then connect them with wires to build circuits. This application can be used to see how circuits work in real time and makes truth tables for the designs. Users can also enter Boolean expressions and see their simplified form on the screen.

This tool is a lighter option compared to expensive software, works well on different systems, and includes key features for learning digital logic. The project shows how to use C++ programming effectively and sets the stage for future improvements in circuit simulation.

Table of Contents

Acknowledgement i

Abstract ii

1 Objectives 1

2 Introduction 2

3 Application 3

4 Literature Survey 4

5 Existing Systems 5

6 Methodology 7

7 Implementation 8

8 Results 9

9 Problems Faced and Solutions 10

10 Limitations and Future Enhancements 11

11 Conclusion and Recommendations 12

References 13

1 Objectives

The primary objective of the **Digital Logic Simulator** project is to develop a practical GUI-based tool that helps students in analyzing digital logic circuits, simplifying Boolean expressions and understanding digital logic concepts.
The specific objectives are:

- To apply object-oriented programming concepts such as encapsulation and abstraction to enhance code maintainability and scalability.
- To use C++ for implementing the core functionalities of the tool to understand its standard libraries.
- To develop and organize custom header files which can help to improve code reusability and modularity throughout the project.
- To learn the basics of graphical user interfaces using SFML which can enable clear and interactive visualization of digital circuits.
- To design an intuitive and accessible interface that allows users to easily construct, modify, and analyze digital logic circuits.
- To make the program run faster and work smoothly in older hardware.
- To gain skills to handle larger, more complex projects in the future.
- To work together as a team to build and complete the project successfully.

2 Introduction

Digital logic design is the foundation of modern electronics and computer engineering. It helps to create systems from basic calculators to complex devices. Tools like Logisim and Digital Works let us design and test digital circuits, but they can be hard to use with command lines, and heavy to run.

The Digital Logic Simulator fixes these problems with a user-friendly tool made with SFML (Simple and Fast Multimedia Library) in C++. Instead of old command-line tools, this one has a clear graphical interface. We can see logic gates, circuits, and truth tables easily, and it uses object-oriented programming.

The Digital Logic Simulator has an interactive canvas where a user can place logic gates like AND, OR, NOT along with INPUT, and OUTPUT components. It offers real-time simulation, so the user can see how the circuit works based on inputs that we choose. The tool automatically creates truth tables for the circuits designed by a user and it can also simplify the boolean expressions that the user enters

The Digital Logic Simulator is designed with modularity, using core object-oriented programming ideas like encapsulation, inheritance, and polymorphism in classes such as Gate, Wire, and Simulator. The SFML graphical interface works on different platforms (tested in Windows and Linux) and renders efficiently.

3 Application

The Digital Logic Simulator serves as a versatile tool with applications that are listed below:

- Students can learn digital logic by building and testing circuits visually. It can make understanding gates and circuits easier.
- Users can create and simulate basic electronic systems quickly.
- The tool generates truth tables for circuits automatically. It saves time for students who are studying logic behavior.
- Users can input boolean expressions and get simpler versions. This can help to optimize circuit designs.
- The tool works on multiple platforms, making it easy to use in classrooms or personal projects.
- Users can model and test logic algorithms (e.g., adders, multiplexers) for correctness and understanding.
- Unlike resource-heavy commercial tools, this project's lightweight SFML-based GUI runs smoothly on older hardware too.

4 Literature Survey

Digital logic simulators are widely used simulation models that are used in digital circuit designs to model and test logic circuits before physical implementation. Early simulators, such as Logisim (Berg, 2008), provided a graphical interface for designing combinational and sequential circuits, allowing students to interactively test logic gates and simple circuits. However, it cannot parse Boolean expressions directly, and lacks features for automating analysis or generating truth tables and Karnaugh maps through scripts.

Another digital logic simulator, Digital Work, developed by Mecanique Ltd. enables users to construct and analyze digital logic circuits through real-time simulation. It can perform simple simulations of circuits and show how signals change over time. The software allows users to design circuits using basic gates (AND, OR, NOT, etc.), flip-flops (D, JK, RS). However, it does not support direct Boolean expression evaluation or advanced minimization features.

With the rise of modern graphics libraries, several implementations have explored using frameworks like Qt, SDL, or SFML for improved user interfaces and real-time simulation. Wired Panda is an open-source digital logic simulator developed in C++ using the Qt framework. It provides a real-time, interactive environment for designing and simulating combinational and sequential digital circuits. It supports a variety of logic gates as well as flip flops.

Some websites provide online calculators to evaluate Boolean expressions or generate truth tables. While they can be convenient for quick tasks, they do not support advanced features like K-map minimization, simulation, or visualizing circuits. They also require internet access, which is not always available or reliable in every environment.

Despite the usefulness of these tools, there is a clear gap in the availability of a lightweight, cross-platform suite that combines Boolean expression parsing, truth table generation, K-map minimization, circuit simulation. Our Digital Logic Simulator aims to fill this gap by providing a modular tool that allows users to perform all these tasks in one place, with the possibility of extending the suite in the future.

5 Existing Systems

Several tools for digital logic design are available, but they have limitations that our Digital Logic Simulator aims to address. Some of those systems are listed along with their limitations below:

- **Logisim**: Logisim is a popular tool for learning how to design and simulate digital circuits, widely used by students and educators, but it doesn't work with modern C++. It is built in Java, which makes it slow on older or less powerful computers.
- **Digital Works**: This is a paid software that allows users to simulate basic logic gates and simple circuits. However, it lacks features like generating truth tables or simplifying Boolean expressions, which makes it less helpful for students.
- **Logic Friday**: Logic Friday is a free tool focused on simplifying Boolean expressions. It only works on Windows (does not have a cross-platform support), has no circuit simulation, and uses an old, hard-to-use interface..
- **Online Boolean Calculators**: Tools like CircuitVerse are web-based and allow users to design circuits and perform basic logic analysis online. However, they require a constant internet connection, can be slower than local applications, and may not offer the same performance as a C++-based tool like ours.

Key Improvements in Our System:

- Our Digital Logic Simulator uses SFML to create a clear and responsive graphical interface, making it easy for users to place logic gates and build circuits. Our application runs smoothly even on less powerful computers.
- Our tool supports the full process of designing circuits, simulating them, generating truth tables, and simplifying Boolean expressions. This all-in-one approach makes it more useful for students.
- The Digital Logic Simulator works on multiple operating systems, including Windows, Linux, and macOS, ensuring that users can access it regardless of their device.
- Unlike web-based tools like CircuitVerse that depend on an internet connection, our app runs entirely offline which can make it more reliable and faster for users in areas with poor internet.

6 Methodology

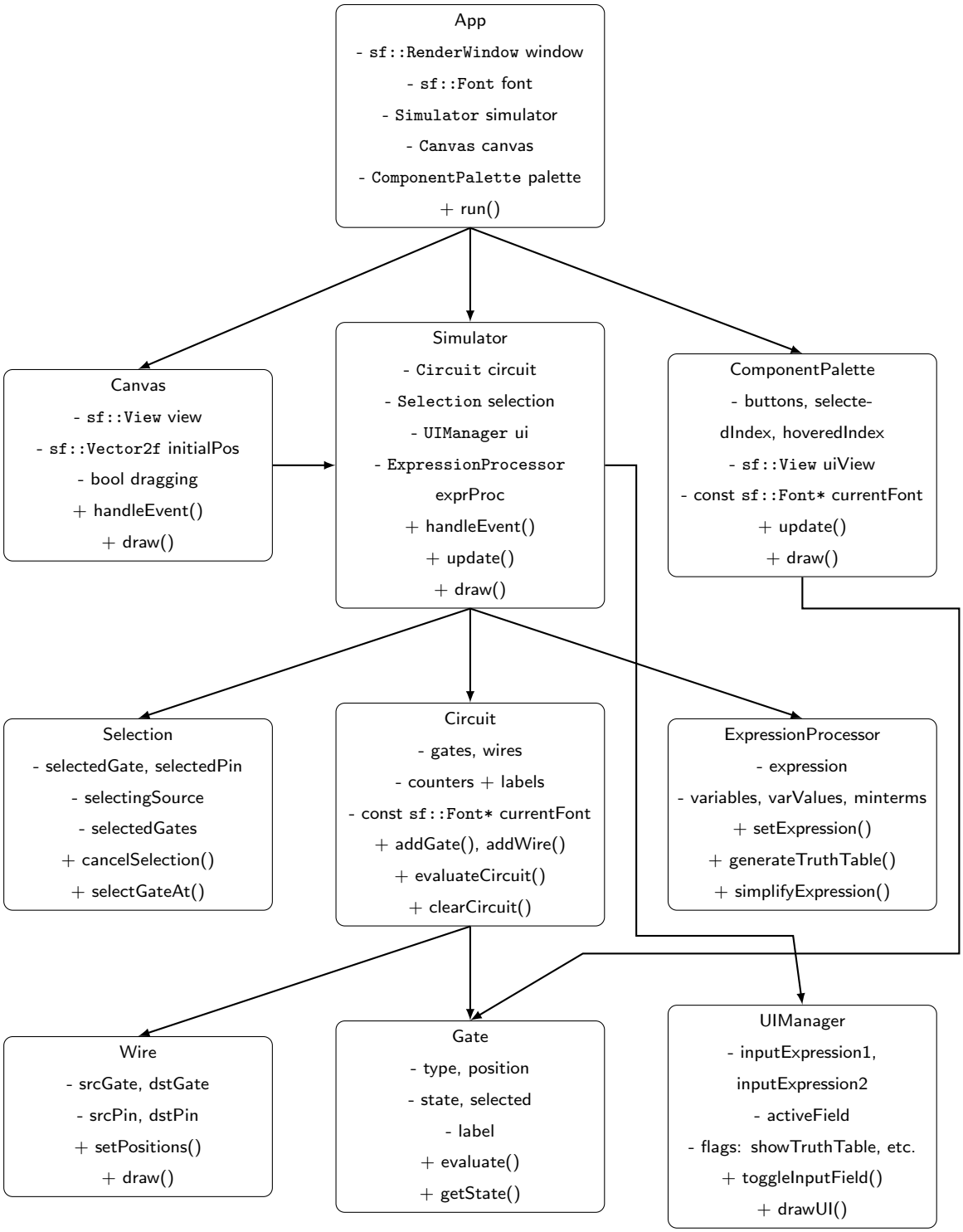
This project, “Digital Logic Simulator,” is a C++ application that uses the SFML graphics library and follows object-oriented programming (OOP) paradigm. We made different classes with private and public members to keep the code organized and safe. We used code reusability and data abstraction to make the project better. Each object has its own functions to handle events, and these work together to create the interactive simulation.

The Digital Logic Simulator solves problems found in old digital logic simulators by giving users an easy-to-use tool made in C++ with SFML (Simple and Fast Multimedia Library). Unlike older tools that use only the command line, this simulator has a clear graphical interface. Users can see logic gates, circuit connections, and truth tables easily. By using OOP, we organized the code into classes, making it easier to reuse and maintain. Users can interact with the simulation, place components, and see how circuits work in real time. This makes it useful for learning and designing digital logic circuits.

Basic C++ graphics were not enough for our needs. To make a better and more interactive interface, we used SFML version 3.0.0. SFML gave us good graphics, worked on different systems, and it was easy to use for drawing, handling events, and getting input. We used VS Code as our IDE and the GCC compiler, which helped us to code, compile, and debug on different platforms.

Before starting, we learned SFML by reading tutorials, official documents, and online guides. We also reviewed OOP concepts to design a system that is easy to manage. SFML is used for features like making the grid layout, displying connections with wires, simulating logic gates, showing the component palette, drawing the window, creating buttons and the visual layout of the program.

7 Implementation



8 Results

With the completion of the project, the objectives of the Digital Logic Simulator were successfully achieved, and the simulator was developed. The project was implemented with the necessary features, including placing logic gates, connecting components, real-time simulation, truth table generation, Boolean expression simplification, and a user-friendly graphical interface.

During the testing phase, various sample circuits were constructed to evaluate the simulator's accuracy and performance. The simulator correctly computed outputs for all tested logic combinations, and the graphical interface responded to the user interactions. Performance remained stable even with complex circuits. This confirmed that the project could be run on older hardware without significant issues which further confirmed that the objectives were achieved.

The project also provided valuable learning outcomes. We gained hands-on experience in object-oriented programming, real-time simulation design, and graphics rendering with SFML. Additionally, collaboration and problem-solving skills were enhanced, learning to work in cooperation with a team and collectively create solutions to problems as a team.

9 Problems Faced and Solutions

The Digital Logic Simulator works well but has some limitations and ideas for future improvements. They are listed below as follows:

1. **Limitations**

    - **Gate Variety**: Only basic, universal and XOR gates are available; advanced components combinational and sequential circuits are missing.
    - **No Circuit Persistence**: Currently, users cannot save or load their circuit designs for future editing or sharing.
    - **Limited User Interface**: The interface is basic and may not be intuitive for all users, lacking features like drag-and-drop and right-click context menus.
    - **No Error Checking**: The application does not provide feedback or warnings for invalid connections or logical errors in the circuit.
    - **Limited Customization**: Users have minimal options to customize gate properties or circuit appearance which might reduce flexibility in design and presentation.

2. **Future Enhancements**

    - **Advanced Components**: Add latches, flip-flops, registers and other logical components for more complex circuit simulations.
    - **Save/Load Functionality**: Implement the ability to save, export, and import circuit designs.
    - **Enhanced User Interface**: Improve usability with features like drag-and-drop, actual gate shapes instead of squares, and better visual feedback.
    - **Error Detection**: Add real-time error checking and helpful messages for invalid circuit configurations.
    - **Simulation Features**: Support step-by-step simulation, timing analysis, and visualization of signal propagation with the help of timing diagrams.

10 Limitations and Future Enhancements

While the Digital Logic Simulator successfully implements the core functionalities, there are several areas that could be improved or added in the future to enhance usability and performance. One limitation encountered during the project was inconsistent code formatting among team members, which occasionally caused difficulties during Git merges and slowed down collaborative development. Also, due to time constraints, more desired features couldn't be added. Future enhancements that could be implemented include are listed below as follows:

- **Extended Component Library**: Adding more advanced logic gates, multiplexers, decoders, and sequential circuit components.
- **Enhanced Sequential Simulation**: Support for complex sequential circuits with timing analysis and clocked operations.
- **Improved User Interface**: Features like drag-and-drop palettes, zooming, advanced grid snapping, and signal indicators to improve circuit visualization.
- **Cross-Platform Packaging**: Standalone installers for Windows, Linux, and macOS to make the tool more accessible.
- **Educational Features**: Integration of tutorials, guided exercises, or challenges to make the simulator more effective for learning purposes.

By addressing these limitations and implementing future enhancements, the Digital Logic Simulator can become a more robust, user-friendly, and educational tool for students, hobbyists, and professionals in the field of digital logic design.

11 Conclusion and Recommendations

The Digital Logic Simulator successfully addresses the limitations of traditional digital logic simulators by providing a lightweight, interactive, and user-friendly platform for designing, simulating, and analyzing digital circuits. By using C++ and the SFML graphics library, the project combines real-time simulation, intuitive graphical representation, and object-oriented design to create a modular system. Users can easily place logic gates, connect components, generate truth tables, and simplify Boolean expressions, all within a single integrated environment.

The project demonstrates the advantages of using object-oriented programming principles, such as encapsulation, and data abstraction, to structure the code effectively. Cross-platform support was achieved through SFML, making the simulator functional on both Windows and Linux. Overall, the Digital Logic Simulator provides an efficient, educational, and practical tool for students, hobbyists, and professionals working with digital logic circuits.

To further enhance the Digital Logic Simulator, additional features could be incorporated, such as more advanced gates and sequential components, improved Boolean expression parsing with automated K-map simplification, and the ability to save and load projects. Enhancements to the user interface, multiple inputs and outputs, addition of components like flipflops, MUX, DEMUX, etc. would improve usability. Expanding cross-platform support and integrating interactive tutorials could also make the simulator more accessible and educational for students and any user.

References

[1] Official SFML Website: *SFML 3.0 Tutorials*.
Available at: https://www.sfml-dev.org/tutorials/3.0/

[2] @hopzbie (YouTube channel): *C++ SFML 3 — Programming (YouTube Playlist)*. Available at: https://www.youtube.com/playlist?list=PLkX_-fCkj2di5WrSIBE66j5Yq0xmHvpAv