



# University of Engineering and Technology, Lahore

Department of Computer Science

---

## Project Report

### Team Members

Name	Roll Number	Email
M. Shahmeer Khan	2025-CS-173	shahmeerkhan98344@gmail.com
M. Soban Chattha	2025-CS-150	sobanchattha3@gmail.com

**Session:** Fall 2025 Morning

**Section:** C

**Semester:** 1<sup>st</sup>

---

**Course:** CSC 101 – Discrete Mathematics

**Teacher:** Mr. Waqas Ali

January 1, 2025

## Table of Contents

Project Report: .....	3
1. Project Title .....	3
2. Selected Track .....	3
3. Overview .....	3
4. Summary .....	3
5. Technology Stack & Libraries utilized .....	3
6. List of Features .....	4
7. Mathematical Foundations .....	5
8. Examination of Key Functions .....	6
9. Limitations .....	9
10. User Interface .....	9
❖ 10.1 Interface overview.....	9
❖ 10.2 Screen-by-Screen Documentation.....	9
❖ 10.3 Sample Test Cases .....	15
11. Future Improvements.....	17
12. Conclusion .....	17
13. References & Learning Sources .....	17

# Project Report:

Submission Deadline: January 6, 2026

## 1. Project Title

*Simulation Digital logic Circuits*

## 2. Selected Track

- Track 1: Foundation of Logic (Rosen Ch. 1-2)

## 3. Overview

*The Digital Logic Circuit Simulator is an educational tool that allows users to understand, simulate, and analyze basic digital circuits using logic gates. It helps to visualize Boolean logic, truth tables, and circuit behaviour, making discrete mathematics and digital logic concepts easier to understand.*

## 4. Summary

- **Problem & Significance:**

Digital logic and Boolean algebra are fundamental topics in discrete mathematics and computer science, but students often find them difficult to understand due to their abstract nature and lack of visual representation.

- **Approach & Methodology:**

The Digital Logic Circuit Simulator provides an interactive platform where users can design and simulate logic circuits using basic gates such as AND, OR, NOT, XOR, NAND, and NOR. Boolean algebra rules and truth tables are used to calculate accurate outputs.

- **Key Achievements & Outcomes:**

The simulator successfully implements real-time logic gate simulation, input–output visualization, and combinational circuits like half adders and full adders. It offers a simple and user-friendly interface suitable for beginners.

- **Relevance to Discrete Mathematics:**

The project directly applies discrete mathematics concepts including propositional logic, Boolean functions, and logical equivalence. It connects theoretical principles with practical implementation, strengthening conceptual understanding.

- **Overall Impact:**

This project bridges the gap between theory and practice by enabling hands-on learning. It enhances analytical thinking, improves problem-solving skills, and prepares students for advanced studies in digital systems and computer architecture.

## 5. Technology Stack & Libraries utilized

- **Programming Language:** C++ (C++ 17/ C++ 20) used for efficient implementation of Boolean logic and circuit simulation.
- **Libraries:** C++ Standard Template Library (STL) for logic operations.

- **Development Environment:** Visual Studio Code with MinGW compiler.
- **Version Control:** Git and GitHub for source code management.
- **Platform:** Windows operating system for development and testing.

## 6. List of Features

- **AND Gate Simulation:**  
Simulates the AND logic gate by accepting binary inputs (0 or 1) and displaying the output along with its truth table and an ASCII-based gate diagram.  
**[Core Feature]**
- **OR Gate Simulation:**  
Allows users to test OR gate behavior using valid binary inputs and view real-time output with a visual representation.  
**[Core Feature]**
- **XOR Gate Simulation:**  
Implements XOR logic gate functionality, showing correct output based on input combinations and illustrating exclusive OR behavior.  
**[Core Feature]**
- **NAND Gate Simulation:**  
Simulates NAND gate by negating AND gate output and provides a truth table with an ASCII diagram.  
**[Core Feature]**
- **NOR Gate Simulation:**  
Implements NOR gate logic by negating OR output and displays its truth table and visual structure.  
**[Core Feature]**
- **Truth Table Generation:**  
Generates truth table output for all supported logic gates based on user-provided input values.  
**[Core Feature]**
- **Half Adder Simulation:**  
Simulates a half adder circuit using XOR for sum and AND for carry, displaying both truth table and circuit flow.  
**[Core Feature]**
- **Full Adder Simulation:**  
Implements a full adder using multiple logic gates (XOR, AND, OR) and displays sum and carry-out results.  
**[Core Feature]**
- **Boolean Minimization Demonstration:**  
Demonstrates Boolean algebra laws including identity, null, idempotent, and complement laws using logical evaluation.  
**[Core Feature]**
- **Menu-Driven User Interface:**  
Provides a simple console-based menu that allows users to select different logic operations and simulations easily.  
**[Core Feature]**

- **Input Validation:**  
Ensures only valid binary inputs (0 or 1) are accepted for logical operations, improving correctness and reliability.  
**[Core Feature]**
- **ASCII-Based Circuit Visualization:**  
Displays basic circuit and gate shapes using text-based diagrams to help users visualize logic behavior.  
**[Optional Enhancement]**

## 7. Mathematical Foundations

### ➤ Relevant Concepts

- **Propositional Logic:**  
The project is based on propositional logic, where inputs and outputs take binary values (0 or 1), representing false and true respectively.
- **Boolean Algebra:**  
Boolean algebra is used to define and compute logic gate operations such as AND, OR, NOT, XOR, NAND, and NOR. These operations follow well-defined algebraic laws.
- **Boolean Functions:**  
Each logic gate is modelled as a Boolean function that maps binary inputs to a binary output.
- **Combinational Circuits:**  
Half adders and full adders are examples of combinational circuits, where outputs depend only on current inputs.

### ➤ Formal Definitions

- **AND Gate:**  
 $A \wedge B = 1$ , if and only if  $A=1$  and  $B=1$
- **OR Gate:**  
 $A \vee B = 1$ , if at least one of A or B is 1
- **NOT Gate:**  
 $\neg A = 1$ , if  $A=0$  and vice versa.
- **XOR Gate:**  
 $A \oplus B = 1$ , if  $A \neq B$
- **Half Adder:**
  - Sum:  $S = A \oplus B$
  - Carry:  $C = A \wedge B$
- **Full Adder:**
  - Sum:  $S = A \oplus B \oplus C_{in}$
  - Carry:  $C_{out} = (A \wedge B) \vee (C_{in} \wedge (A \oplus B))$

### ➤ Proofs & Correctness

- **Correctness of Logic Gates:**  
Each gate function directly implements its Boolean definition using bitwise operators, ensuring correctness for all valid inputs.

- Adder Correctness:

The half adder and full adder equations are standard Boolean expressions proven to correctly model binary addition.

## ➤ Illustrative Examples

- For inputs  $A=1$ ,  $B=0$ :
    - AND = 0
    - OR = 1
    - XOR = 1
  - For full adder inputs  $A=1$ ,  $B=1$ ,  $Cin=0$ :
    - Sum = 0
    - Carry = 1

## **8. Examination of Key Functions**

## ❖ AND Gate Function

- 1. Function Name:** AND
  - 2. Purpose:** Computes the logical AND of two inputs (0 or 1).
  - 3. Algorithm Description:**

- Take two inputs a and b.
  - Apply bitwise AND:  $a \& b$ .
  - Return the result.

#### **4. Time and Space Complexity:**

- **Time Complexity:**  $O(1)$  – single operation.
  - **Space Complexity:**  $O(1)$  – no extra memory used.

**5. Implementation Details:** Uses C++ bitwise AND operator &. Valid for binary inputs.

## **6. Code Snippet:**

```
void truthTableAND()
{
    int a;
    int b;
    cout<<"Enter A and B :";
    while(cin>>a>>b && (a==1 || a==0 && b==1 || b==0 ))
    {
        cout << "\nAND Gate Truth Table\n";
        cout << "A B | Y\n";
        cout << a << " " << b << " | " << AND(a, b) << "\n\n";

        cout<<a<<" **** * " <<endl;
        cout<<" *      * " <<endl;
        cout<<" *      * " <<endl;
        cout<<" *      **** " <<AND(a,b)<<endl;
        cout<<" *      * " <<endl;
        cout<<" *      * " <<endl;
        cout<<b<<" **** * " <<endl;
        cout<<"          AND GATE           " <<endl;
        cout<<"Enter value other than 1 and 0 to exist :";
    }
}
```

## ❖ NAND Gate Function

- 1. Function Name:** NAND
- 2. Purpose:** Computes NOT AND operation.
- 3. Algorithm Description:**

- Compute AND of inputs.
- Negate result using logical NOT !

- 4. Time/Space Complexity:** O(1), O(1)

- 5. Implementation Details:** ! returns 0 or 1, works for binary inputs only.

- 6. Code Snippet:**

```
void truthTableAND()
{
    int a;
    int b;
    cout<<"Enter A and B :";
    while(cin>>a>>b && (a==1 || a==0 && b==1 || b==0 ))
    {
        cout << "\nAND Gate Truth Table\n";
        cout << "A B | Y\n";
        cout << a << " " << b << " | " << AND(a, b) << "\n\n";

        cout<<a<<" **** * ** <<endl;
        cout<<"          **      ** <<endl;
        cout<<"          **      **<<endl;
        cout<<"          **      ***** " <<AND(a,b)<<endl;
        cout<<"          **      **<<endl;
        cout<<"          **      ** <<endl;
        cout<<b<<" **** * ** <<endl;
        cout<<"          AND GATE      " <<endl;
        cout<<"Enter value other than 1 and 0 to exist :";
    }
}
```

## ❖ Half Adder Function

- 1. Function Name:** Half Adder
- 2. Purpose:** Computes sum and carry for two inputs.
- 3. Algorithm Description:**

- Input: A, B.
- Sum = XOR (A, B)
- Carry = AND (A, B)
- Print truth table.

- 4. Time/Space Complexity:** O(1), O(1)

- 5. Implementation Details:** Uses previously defined AND and XOR functions. Allows repeated input until invalid value.

- 6. Code Snippet:**

```

void halfAdder()
{
    int a,b;
    cout<<"Enter the value of A and B :";

    while(cin>>a>>b && (a==1 || a==0 && b==1 || b==0 ))
    {
        cout << "\nHalf Adder Truth Table\n";
        cout << "A B | Sum Carry\n";
        bool sum = XOR(a, b);
        bool carry = AND(a, b);
        cout << a << " " << b << " | " << sum << " " << carry << endl;

        cout << "\nHalf Adder Circuit\n";
        cout << a<<"---> XOR --->" <<sum<<"\n";
        cout << " \\\n";
        cout << b<<" --> AND ---->" <<carry<<"\n";
        | cout<<"Enter value other than 1 and 0 to exist :";
    }
}

```

## ❖ Boolean Minimization Function

**1. Function Name:** Boolean Minimization

**2. Purpose:** Demonstrates basic Boolean algebra laws (Identity, Null, Idempotent, Complement).

**3. Algorithm Description:**

- Input: law choice and variable A.
- Apply corresponding law using AND, OR, NOT.
- Print minimized result.

**4. Time/Space Complexity:** O(1), O(1)

**5. Implementation Details:** Only handles single-variable simplifications.

**6. Code Snippet:**

```

void booleanMinimization()
{
    int choice;
    int a;

    cout << "\n--- BOOLEAN MINIMIZATION CALCULATOR ---\n";
    cout << "1. Identity Law (A + 0 = A | A . 1 = A)\n";
    cout << "2. Null Law      (A + 1 = 1 | A . 0 = 0)\n";
    cout << "3. Idempotent     (A + A = A | A . A = A)\n";
    cout << "4. Complement      (A + A' = 1 | A . A' = 0)\n";
    cout << "Enter the law you want to test: ";
    cin >> choice;

    cout << "Enter input value for A (0 or 1): ";
    cin >> a;

    if (a != 0 && a != 1) {
        cout << "Invalid input! Please enter 0 or 1.\n";
        return;
    }
}

```

```

cout << "\n--- Minimization Result ---\n";

switch (choice)
{
    case 1:
        cout << "Operation: A + 0 => " << a << " + 0 = " << OR(a, 0) << " (Result matches A)\n";
        cout << "Operation: A . 1 => " << a << " . 1 = " << AND(a, 1) << " (Result matches A)\n";
        break;
    case 2:
        cout << "Operation: A + 1 => " << a << " + 1 = " << OR(a, 1) << " (Result is always 1)\n";
        cout << "Operation: A . 0 => " << a << " . 0 = " << AND(a, 0) << " (Result is always 0)\n";
        break;
    case 3:
        cout << "Operation: A + A => " << a << " + " << a << " = " << OR(a, a) << " (Simplified to A)\n";
        cout << "Operation: A . A => " << a << " . " << a << " = " << AND(a, a) << " (Simplified to A)\n";
        break;
    case 4:
        cout << "Operation: A + A' => " << a << " + " << NOT(a) << " = " << OR(a, NOT(a)) << " (Always 1)\n";
        cout << "Operation: A . A' => " << a << " . " << NOT(a) << " = " << AND(a, NOT(a)) << " (Always 0)\n";
        break;
    default:
        cout << "Invalid choice!\n";
}

```

## 9. Limitations

### 1) Scope Limitation:

- o Only basic gates and single-bit adders are supported.
- o Boolean minimization works for one variable only.
- o No multi-bit adders or graphical circuits.

### 2) Performance Constraints:

- o Manual input is slow for many tests.
- o ASCII diagrams take extra time to display.

### 3) Algorithm Limitations:

- o Works only with 0 or 1 inputs.
- o Adders and minimization can't handle multiple bits or variables.

### 4) Design Trade-Offs:

- o Code is simple but repetitive.
- o Diagrams are visual but not easy to read or scale.
- o Strict input rules make it less flexible.

### 5) Known Issues:

- o Input checks can fail sometimes.
- o Diagrams may look messy.
- o Program can crash if wrong input is entered.
- o Boolean minimization is very basic.

## 10. User Interface

### ❖ 10.1 Interface overview

The Digital Logic Circuit Simulator is designed with **simplicity and clarity** in mind, using a **command-line interface (CLI)** to make interaction straightforward and accessible. The main design philosophy is to provide a **menu-driven, step-by-step workflow**, allowing users to easily select logic gates, adders, or Boolean laws and input values for immediate feedback. The simulator emphasizes **visual understanding** through ASCII circuit diagrams alongside truth tables, helping users connect theoretical logic concepts with practical results. The user experience prioritizes **guided input, repeated testing, and immediate results**, while maintaining a lightweight, responsive, and easy-to-navigate interface, suitable for learning and experimentation in digital logic.

### ❖ 10.2 Screen-by-Screen Documentation

#### Screen 1: Main Menu Screen

**Screen Name/Title:**

Digital Logic Circuit Simulator – Main Menu

**Purpose:**

This screen allows the user to select which digital logic operation or circuit they want to simulate.

**User Actions:**

The user selects one option by entering a number:

- 1 → AND Gate
- 2 → OR Gate
- 3 → XOR Gate
- 4 → Half Adder
- 5 → Full Adder
- 6 → Boolean Minimization
- 7 → NAND Gate
- 8 → NOR Gate
- 0 → Exit the program

**Screenshot:**

```
--- DIGITAL LOGIC CIRCUIT SIMULATOR ---
1. AND Gate
2. OR Gate
3. XOR Gate
4. Half Adder
5. Full Adder
6. Boolean Minimization
7. NAND GATE
8. NOR GATE
0. Exit
Enter your choice: |
```

**Input Description:**

- A single integer representing the menu choice.

**Output Description:**

- Based on the selected option, the program navigates to the corresponding gate or circuit screen.
- If an invalid number is entered, an error message is displayed.

**Navigation:**

- Automatically redirects to the selected operation screen.
- Selecting 0 exits the program and terminates execution.

## Screen 2: Logic Gate Input Screen (AND, OR, XOR, NAND, NOR)

### Screen Name/Title:

Logic Gate Input Screen

### Purpose:

Allows the user to input binary values for two inputs (A and B) to simulate the selected logic gate.

### User Actions:

- Enter values for inputs A and B.
- Can repeat input multiple times.
- Entering any value other than 0 or 1 exits back to the main menu.

### Screenshot:

```
Enter A and B :1 1

AND Gate Truth Table
A B | Y
1 1 | 1

1 ****
**   **
**   **
**   ****    1
**   **
**   **

1 ****
      AND GATE
Enter value other than 1 and 0 to exist :2 2
```

### Input Description:

- Two integers (A and B)
- Valid values: 0 or 1

### Output Description:

- Displays:
  - Truth table result (A, B, Output)
  - Calculated gate output
  - ASCII-based visual representation of the selected gate

**Navigation:**

- Repeats input loop until invalid input is entered.
- Returns automatically to the main menu.

**Screen 3: Half Adder Screen****Screen Name/Title:**

Half Adder Simulation Screen

**Purpose:**

Simulates a Half Adder circuit using XOR (Sum) and AND (Carry) logic.

**User Actions:**

- Enter binary values for inputs A and B.
- Repeat inputs to test multiple cases.
- Entering values other than 0 or 1 exits the screen.

**Screenshot:**

```

Enter the value of A and B :1 1

Half Adder Truth Table
A B | Sum Carry
1 1 |  0     1

Half Adder Circuit
1----> XOR ---->0
 \
1 --> AND ---->1
Enter value other than 1 and 0 to exist :2 2

--- DIGITAL LOGIC CIRCUIT SIMULATOR ---

```

**Input Description:**

- Two binary inputs: A and B

**Output Description:**

- Displays:
  - Half Adder truth table (Sum and Carry)
  - Logical flow showing XOR and AND gate connections
  - Calculated Sum and Carry outputs

**Navigation:**

- Continues accepting input until invalid input is provided.
- Returns to the main menu.

#### Screen 4: Full Adder Screen

##### Screen Name/Title:

Full Adder Simulation Screen

##### Purpose:

Simulates a Full Adder circuit using two XOR gates, two AND gates, and one OR gate.

##### User Actions:

- Enter three binary values: A, B, and Cin.
- Repeat inputs for different test cases.
- Invalid input exits the screen.

##### Screenshot:

```

Enter the value of A , b and Cin: 1 1 1

Full Adder Truth Table
A B Cin | Sum Cout
1 1 1 | 1   1

Full Adder Circuit
1 ----> XOR ----> XOR ----> Sum
1
1 \
--> AND ---\ OR ---->
11 --> AND ---/
Enter value other than 1 and 0 to exist :2 2 2

--- DIGITAL LOGIC CIRCUIT SIMULATOR ---
1 AND Gate

```

##### Input Description:

- Three binary values:
  - A (Input)
  - B (Input)
  - Cin (Carry In)

##### Output Description:

- Displays:
  - Full Adder truth table (Sum and Carry Out)
  - Step-by-step logic using intermediate carries
  - ASCII circuit-style output flow

##### Navigation:

- Loops until invalid input is entered.

- Automatically returns to the main menu.

### Screen 5: Boolean Minimization Screen

**Screen Name/Title:**

Boolean Minimization Calculator

**Purpose:**

Demonstrates basic Boolean algebra laws used for logic simplification.

**User Actions:**

- Select a Boolean law by entering a number (1–4).
- Enter a binary value for variable A.

**Screenshot:**

```

--- BOOLEAN MINIMIZATION CALCULATOR ---
1. Identity Law (A + 0 = A | A . 1 = A)
2. Null Law      (A + 1 = 1 | A . 0 = 0)
3. Idempotent    (A + A = A | A . A = A)
4. Complement    (A + A' = 1 | A . A' = 0)
Enter the law you want to test: 1
Enter input value for A (0 or 1): 1

--- Minimization Result ---
Operation: A + 0 => 1 + 0 = 1 (Result matches A)
Operation: A . 1 => 1 . 1 = 1 (Result matches A)

```

**Input Description:**

- Integer choice for Boolean law
- One binary input value (A)

**Output Description:**

- Displays:
  - Selected Boolean law
  - Step-by-step evaluation
  - Simplified Boolean result

**Navigation:**

- After displaying results, returns to the main menu.

### Screen 6: Exit Screen

**Screen Name/Title:**

Program Exit

**Purpose:**

Terminates the simulator safely.

**User Actions:**

- Select option 0 from the main menu.

**Screenshot:**

```
--- DIGITAL LOGIC CIRCUIT SIMULATOR ---
1. AND Gate
2. OR Gate
3. XOR Gate
4. Half Adder
5. Full Adder
6. Boolean Minimization
7. NAND GATE
8. NOR GATE
0. Exit
Enter your choice: 0
Program Exited Successfully.
```

**Input Description:**

- Integer 0

**Output Description:**

- Displays confirmation message:  
“*Program Exited Successfully.*”

**Navigation:**

- Program execution ends.

## ❖ 10.3 Sample Test Cases

➤ **Test Case 1: AND Gate Operation****Test Case Name:**

Basic AND Gate Evaluation

**Input:**

- Main Menu Choice: 1 (AND Gate)

- Input A: 1
- Input B: 1

**Expected Output:**

- Truth table displays  $A=1, B=1, Output=1$
- ASCII representation of AND gate shown
- Correct logical output displayed

**Actual Output:**

- Program displays output 1 for AND operation
- Truth table and gate diagram printed correctly

**Status:**

Pass

**➤ Test Case 2: NAND Gate Operation****Test Case Name:**

NAND Gate Both Inputs High

**Input:**

- Main Menu Choice: 7 (NAND Gate)
- Input A: 1
- Input B: 1

**Expected Output:**

- NAND output = 0
- Gate diagram shown with inversion bubble

**Actual Output:**

- Output correctly displayed as 0
- Truth table and diagram printed

**Status:**

Pass

**➤ Test Case 3: Half Adder Simulation****Test Case Name:**

Half Adder with Both Inputs High

**Input:**

- Main Menu Choice: 4 (Half Adder)
- Input A: 1
- Input B: 1

**Expected Output:**

- Sum = 0
- Carry = 1
- Half Adder truth table displayed

**Actual Output:**

- Program outputs Sum 0 and Carry 1
- Logical flow diagram displayed correctly

**Status:**

Pass

## 11. Future Improvements

- Support for Multi-bit Adders: Extend the current single-bit half and full adders to support 4-bit or 8-bit ripple-carry adders to simulate more complex arithmetic operations.
- Graphical User Interface (GUI): Transition from a command-line interface (CLI) to a graphical platform where users can drag and drop gates to build circuits visually.
- Advanced Boolean Minimization: Upgrade the current basic minimization demonstration to include a full implementation of the Karnaugh Map (K-Map) or Quine-McCluskey algorithm for expressions with 3 or 4 variables.
- Sequential Logic Support: Introduce flip-flops, latches, and counters to move beyond combinational logic and simulate memory-based circuits.

## 12. Conclusion

The Digital Logic Circuit Simulator successfully achieves its goal of bridging the gap between abstract Boolean theory and practical implementation. By simulating core logic gates and combinational circuits like adders, the project demonstrates the fundamental principles of propositional logic and Boolean functions found in discrete mathematics. This development process has not only reinforced our understanding of logical correctness and bitwise operations but also highlighted the importance of algorithm efficiency and user-centric design in educational software.

## 13. References & Learning Sources

- **GitHub Repository link:**  
➤ [https://github.com/shahmeer Khan98344-ai/DM\\_final\\_project.git](https://github.com/shahmeer Khan98344-ai/DM_final_project.git)

- **Demo Video Link:**

➤ <https://share.zight.com/Qwu1jn9m>

- **Textbook:**

Rosen, K. H. (2019). Discrete Mathematics and Its Applications (8th ed.). McGraw-Hill Education. (Used for formal definitions of logical operators and circuit properties) .

- **C++ Documentation:**

ISO/IEC. (2020). Standard for Programming Language C++ (C++20). (Used for bitwise operator implementation details) .

- **AI Assistance:**

ChatGPT (OpenAI) was utilized to assist in drafting the project report structure, generating ASCII gate diagrams, and refining the logic for the Boolean minimization demonstration.

- **Development Tools:**

Visual Studio Code and the MinGW compiler suite were used for building and testing the source code.