Digital Circuit Simulation Project

This project simulates a Digital Circuit System that allows users to create, simulate, and analyze the behavior of basic combinational logic ICs using classes representing different types of Integrated Circuits (ICs) like AND, OR, NOT, NOR, NAND, XOR, XNOR gates, and a system to connect and interact with these ICs.

The project demonstrates Object-Oriented Programming concepts in C++ (like Encapsulation, Abstraction, Operator Overloading, Inheritance, Polymorphism and Exception handling) and provides an interactive menu-driven interface for designing circuits dynamically by connecting ICs and wires, setting pin values, and observing results.

Table of Contents

Project Structure

The project consists of the following files:

- `IC.hpp` and `IC.cpp`: Base class `IC` for all integrated circuits.
- `ANDGateIC.hpp` and `ANDGateIC.cpp`: Class `ANDGateIC` to simulate an AND gate IC.
- `ORGateIC.hpp` and `ORGateIC.cpp`: Class `ORGateIC` to simulate an OR gate IC.
- `NOTGateIC.hpp` and `NOTGateIC.cpp`: Class `NOTGateIC` to simulate a NOT gate IC.
- `NORGateIC.hpp` and `NORGateIC.cpp`: Class `NORGateIC` to simulate a NOR gate IC.
- `NANDGateIC.hpp` and `NANDGateIC.cpp`: Class `NANDGateIC` to simulate a NAND gate IC.
- `XORateIC.hpp` and `XORGateIC.cpp`: Class `XORGateIC` to simulate a XOR gate IC.
- `NORGateIC.hpp` and `XNORGateIC.cpp`: Class `XNORGateIC` to simulate a XNOR gate IC.

Features

- **Interactive User Input**:
  The program uses a menu-driven interface to guide users through creating and managing circuits.

  - **Supported ICs**:
  - AND Gate IC
  - OR Gate IC
  - NOT Gate IC
  - XOR Gate IC
  - NAND Gate IC
  - NOR Gate IC
  - XNOR Gate IC

- **Dynamic Circuit Design**:
  Users can:

  - Create new ICs.
  - Set pin values for ICs.
  - Connect power to the ICs.
  - Connect ICs with wires.
  - Simulate the circuit and observe the output.

- **IC Manipulation**: Set and retrieve pin values, connect ICs to each other, and use logic gates.

- **Operator Overloading**: Use operators for pin manipulation, IC comparison, and power connections.

- **Error Handling**: The program handles invalid inputs and operations gracefully.

- **Virtual Functions**: Define a `simulate()` function for IC-specific behavior and execute digital logic. Simulate INDIVIDUAL ICs and view pin states after simulation.

Getting Started

Prerequisites

To compile and run this project, you need:

- C++ compiler supporting C11 or later (e.g., GCC/G)
- CMake (optional for build automation)
- Basic understanding of Digital Circuits

Note: Now the Project can be compiled and run using just 2 commands and clear all the cache files in 1 command using MakeFile concept.

Tests will be implemented soon.

Running the Project

Make sure you have cloned the GitHub repo into your local system if you have not already:

```
git clone https://github.com/Prometheus052404/CIRCUIT.git
```

Use the below commands in `Git Bash` at your Project's Root Directory:

```
git fetch
git pull
```

As you have made sure that you're up-to-date, you are now ready to continue ahead!

Manual Compilation

1. Navigate to the project directory:

   ```
   cd DigitalLogicSimulator
   ```

2. Compile the project:

   ```
   g++ src/*.cpp main.cpp -o simulator
   ```

3. Run the executable:

   ```
   ./simulator
   ```

Using CMake (Optional)

Note: Skip this section if `make --version` gives desired output in your git bash

If you are using Windows and have wsl installed, but not make, then follow the below steps:

- Go to ezwinports, i.e. https://sourceforge.net/projects/ezwinports/files/
- Download `make-4.1-2-without-guile-w32-bin.zip` (get the version without guile)
- Extract zip
- Copy the contents to `C:\Program Files\Git\mingw64\` merging the folders, but do NOT overwrite/replace any exisiting files.

If you are using Ubuntu/Debian,

- Open WSL and update the package list:

```
sudo apt update
```

- Install make:

```
sudo apt install make
```

Navigate to your project directory within WSL and run `make`. OR

```
sudo apt install build-essential
```

`build-essential` includes `make` and other essential development tools like `gcc` and `g++`.

For MacOS,

- If you have Homebrew installed, you can install `make` with:

```
brew install make
```

Verify Installation

- Open a new Git Bash window to refresh your PATH.
- Run:

```
make --version
```

This should display the version of `make` if it’s installed correctly.

Implementation

Create the Project using Makefile:

```
make
```

Execute the code:

```
./DCSimulator
```

Remove all the build / cache files:

```
make clean
```

Note: Run the above commands in git bash, at the project's root directory.

Usage

- **Create IC objects**: Instantiate various IC objects, e.g., `ANDGateIC`, `ORGateIC`.
- **Connect Power**: Connect VCC and GND to the ICs to simulate power supply.
- **Simulate**: Call the `simulate()` function on each IC to execute its digital logic.

Classes and Their Functions

Class `IC`

The base class for all ICs.

- **Constructor**: Initializes pins, VCC, and GND.
- **connectVCC()**: Connects the IC to the power rail.
- **connectGround()**: Connects the IC to the ground rail.
- **setPin(int pin, int value)**: Sets a pin's value.
- **getPin(int pin)**: Gets a pin's value.
- **simulate()**: Pure virtual function for IC-specific logic.

Logic Gate ICs

Each gate IC (`ANDGateIC`, `ORGateIC`, `NOTGateIC`, `NORGateIC`, `NANDGateIC`, `XORGateIC`, `XNNORGateIC`) inherits from `IC` and overrides the `simulate()` method to perform specific logic operations.

Follow the on-screen menu to:

- View ICs.
- Create ICs.
- Set pin values.
- Connect ICs with wires.
- Simulate the circuit.
- Connect Power & Ground to the ICs
- View IC pin states.

Sample Workflow

- **Step 1**: Create an AND Gate IC.
- **Step 2**: Set input pin values to `1` and `0`.
- **Step 3**: Simulate the IC to view the output.
- **Step 4**: Connect the output of the AND Gate to another IC's input using a wire.

Exit the Program

To exit the simulator, choose the `Exit` option from the menu.

Code Structure

```
DigitalLogicSimulator/
├── include/
│   ├── IC.hpp
│   ├── ANDGateIC.hpp
│   ├── ORGateIC.hpp
│   ├── NOTGateIC.hpp
│   ├── XORGateIC.hpp
│   ├── NANDGateIC.hpp
│   ├── NORGateIC.hpp
│   ├── XNORGateIC.hpp
│   └── Wire.hpp
├── src/
```

```
│   ├── ANDGateIC.cpp
│   ├── ORGateIC.cpp
│   ├── NOTGateIC.cpp
│   ├── XORGateIC.cpp
│   ├── NANDGateIC.cpp
│   ├── NORGateIC.cpp
│   ├── XNORGateIC.cpp
│   └── Wire.cpp
├── main.cpp
├── Makefile
└── README.md
```

Example

Below is a sample usage example:

```
#include "./include/ANDGateIC.hpp"
#include "./include/ORGateIC.hpp"
#include "./include/NOTGateIC.hpp"
#include "./include/XORGateIC.hpp"
#include "./include/NANDGateIC.hpp"
#include "./include/NORGateIC.hpp"
#include "./include/XNORGateIC.hpp"
#include "./include/Wire.hpp"
#include <string>
#include <vector>

int main() {
    vector<IC*> icList;
    vector<Wire*> wireList;

    while (true) {
        //making a box for the menu
        cout << "\n-----------------------------------\n";
        cout << "\n--- Circuit Simulator Menu ---\n";
        cout << "1. View ICs\n";
        cout << "2. Create a new IC\n";
        cout << "3. Set pin values\n";
        cout << "4. Connect ICs with a wire\n";
        cout << "5. Simulate IC\n";
        cout << "6. Connect Power and Ground\n";
        cout << "7. View IC pin states\n";
        cout << "8. Exit\n";

        cout << "-----------------------------------\n";

        //taking the input from the user
        cout << "Enter your choice: ";
        int choice;
        cin >> choice;

        if (choice == 1) {
            if (icList.empty()) {
                cout << "No ICs available.\n";
                continue;
            }

            cout << "ICs available: \n";
            for (size_t i = 0; i < icList.size(); ++i) {
                cout << i + 1 << ". " << icList[i]->getName() << endl;
            }
        }
        else if (choice == 2) {
            cout << "Select IC type:\n";
            cout << "1. AND Gate\n2. OR Gate\n3. NOT Gate\n4. XOR Gate\n5. NAND Gate\n6. NOR Gate\n7. XNOR Gate\n";
            int icType;
            cin >> icType;

            IC* newIC = nullptr;
            switch (icType) {
                case 1: newIC = new ANDGateIC(); break;
                case 2: newIC = new ORGateIC(); break;
                case 3: newIC = new NOTGateIC(); break;
                case 4: newIC = new XORGateIC(); break;
                case 5: newIC = new NANDGateIC(); break;
                case 6: newIC = new NORGateIC(); break;
                case 7: newIC = new XNORGateIC(); break;
                default: cout << "Invalid choice.\n"; continue;
            }

            string powerChoice;
            cout << "Connect power to this IC? (y/n): ";
            cin >> powerChoice;

            if (powerChoice == "y" || powerChoice == "Y") {
                newIC->connectVCC();
                newIC->connectGround();
                cout << "Power connected to IC.\n";
            }

            icList.push_back(newIC);
            cout << "IC created and added to the circuit.\n";
        }

        else if (choice == 3) {
            if (icList.empty()) {
                cout << "No ICs available. Create an IC first.\n";
                continue;
            }

            size_t icIndex;
            int pin, value;
            cout<< "ICs available: \n";
            for (size_t i = 0; i < icList.size(); ++i) {
                cout << i + 1 << ". " << icList[i]->getName() << endl;
            }
            cout<< "----------------------------------\n";
            cout << "Select IC index (1-" << icList.size() << "): ";
            cin >> icIndex;
            if (icIndex < 1 || icIndex > icList.size()) {
                cout << "Invalid IC index.\n";
                continue;
            }

            IC* selectedIC = icList[icIndex - 1];
            cout << "Pin values of IC " << icIndex << ":\n";
            for (int pin = 1; pin <= selectedIC->getTotalPins(); ++pin) {
                cout << "Pin " << pin << ": " << selectedIC->getPin(pin) << " ";
            }
            cout << "\n----------------------------------\n";
            cout << "Enter pin number to set (1-" << selectedIC->getTotalPins() << "): ";
            cin >> pin;
            if (pin < 1 || pin > selectedIC->getTotalPins()) {
                cout << "Invalid pin number.\n";
                continue;
            }
            else if(pin == selectedIC->vccPin || pin == selectedIC->groundPin) {
                cout << "Cannot set VCC or GND pin value.\n";
                continue;
            }
            cout << "Enter value for pin " << pin << " (0/1): ";
            cin >> value;

            try {
                selectedIC->setPin(pin, value);
                //changing values for pins connected by wire without accessing the private members
                for (Wire* wire : wireList) {
                    if (wire->getSourceIC() == selectedIC && wire->getSourcePin() == pin) {
                        wire->connect();
                    }
                }

                cout << "Pin value set successfully.\n";
            } catch (const exception& e) {
                cout << "Error: " << e.what() << endl;
            }
        }

        else if (choice == 4) {
            if (icList.size() < 2) {
                cout << "At least two ICs are required for connection.\n";
                continue;
            }

            size_t srcIC, destIC;
            int srcPin, destPin;
            cout << "ICs available: \n";
            for (size_t i = 0; i < icList.size(); ++i) {
                cout << i + 1 << ". " << icList[i]->getName() << endl;
            }
            cout << "----------------------------------\n";
            cout << "Select source IC index (1-" << icList.size() << "): ";
            cin >> srcIC;
            cout << "Pin values of IC " << srcIC << ":\n";
            for (int pin = 1; pin <= icList[srcIC - 1]->getTotalPins(); ++pin) {
                cout << "Pin " << pin << ": " << icList[srcIC - 1]->getPin(pin) << " ";
            }
            cout << "\n----------------------------------\n";
            cout << "Enter source pin: ";
            cin >> srcPin;
            cout << "Select destination IC index (1-" << icList.size() << "): ";
            cin >> destIC;
            cout << "Pin values of IC " << destIC << ":\n";
            for (int pin = 1; pin <= icList[destIC - 1]->getTotalPins(); ++pin) {
                cout << "Pin " << pin << ": " << icList[destIC - 1]->getPin(pin) << " ";
            }
            cout << "\n----------------------------------\n";
            cout << "Enter destination pin: ";
            cin >> destPin;

            if (srcIC < 1 || srcIC > icList.size() || destIC < 1 || destIC > icList.size()) {
                cout << "Invalid IC index.\n";
                continue;
            }

            try {
                Wire* newWire = new Wire(icList[srcIC - 1], srcPin, icList[destIC - 1], destPin);
                newWire->connect();
                wireList.push_back(newWire);
                cout << "Wire connected successfully.\n";
            } catch (const exception& e) {
                cout << "Error: " << e.what() << endl;
            }
        }

        else if (choice == 5) {
            if (icList.empty()) {
                cout << "No ICs available to simulate.\n";
                continue;
            }

            size_t icIndex;
            cout << "ICs available: \n";
            for (size_t i = 0; i < icList.size(); ++i) {
                cout << i + 1 << ". " << icList[i]->getName() << endl;
            }
            cout << "-----------------------------------\n";
            cout << "Select IC index to simulate (1-" << icList.size() << "): ";
            cin >> icIndex;
            if (icIndex < 1 || icIndex > icList.size()) {
                cout << "Invalid IC index.\n";
                continue;
            }

            try {
                icList[icIndex - 1]->simulate();
                cout << "IC simulation completed successfully.\n";
            } catch (const exception& e) {
                cout << "Error: " << e.what() << endl;
            }
        }

        else if (choice == 6) {
            if (icList.empty()) {
                cout << "No ICs available to connect power and ground.\n";
                continue;
            }

            size_t icIndex;
            cout << "ICs available: \n";
            for (size_t i = 0; i < icList.size(); ++i) {
                cout << i + 1 << ". " << icList[i]->getName() << endl;
            }
            cout << "-----------------------------------\n";
            cout << "Select IC index to connect power and ground (1-" << icList.size() << "): ";
            cin >> icIndex;
            if (icIndex < 1 || icIndex > icList.size()) {
                cout << "Invalid IC index.\n";
                continue;
            }

            try {
                icList[icIndex - 1]->connectVCC();
                icList[icIndex - 1]->connectGround();
                cout << "Power and ground connected to IC.\n";
            } catch (const exception& e) {
                cout << "Error: " << e.what() << endl;
            }
        }

        else if (choice == 7) {
            if (icList.empty()) {
                cout << "No ICs available to view.\n";
                continue;
            }

            for (size_t i = 0; i < icList.size(); ++i) {
                cout << "IC " << i + 1 << " (" << icList[i]->getName() << "): ";
                for (int pin = 1; pin <= icList[i]->getTotalPins(); ++pin) {
                    try {
                        cout << "Pin " << pin << ": " << icList[i]->getPin(pin) << " ";
                    } catch (const exception&) {
                        cout << "N/A ";
                    }
                }
                cout << endl;
            }
        }
```

```
        else if (choice == 8) {
            cout << "Exiting program...\n";
            break;
        }

        else {
            cout << "Invalid choice. Try again.\n";
        }
    }

    // Clean up dynamically allocated memory
    for (IC* ic : icList) delete ic;
    for (Wire* wire : wireList) delete wire;

    return 0;
}
```

Future Implementation

Enhancements to Explore:

1. **Breadboard Integration**

- Implement a `Breadboard` class to simulate multiple wires and IC connections.
- Allow dynamic connections to pins on the breadboard and ICs.

2. **Advanced Signal Propagation**

- Simulate signal propagation delays in wires.
- Introduce realistic electrical behavior (e.g., resistance, capacitance).

3. **Circuit Visualization**

- Create a GUI/CLI-based circuit visualization tool for better interaction.
- Use frameworks like Qt for GUI or text-based interfaces.

4. **Testing Framework**

- Automate testing for IC behaviors with predefined input-output pairs.
- Create detailed error logs for invalid connections or simulations.

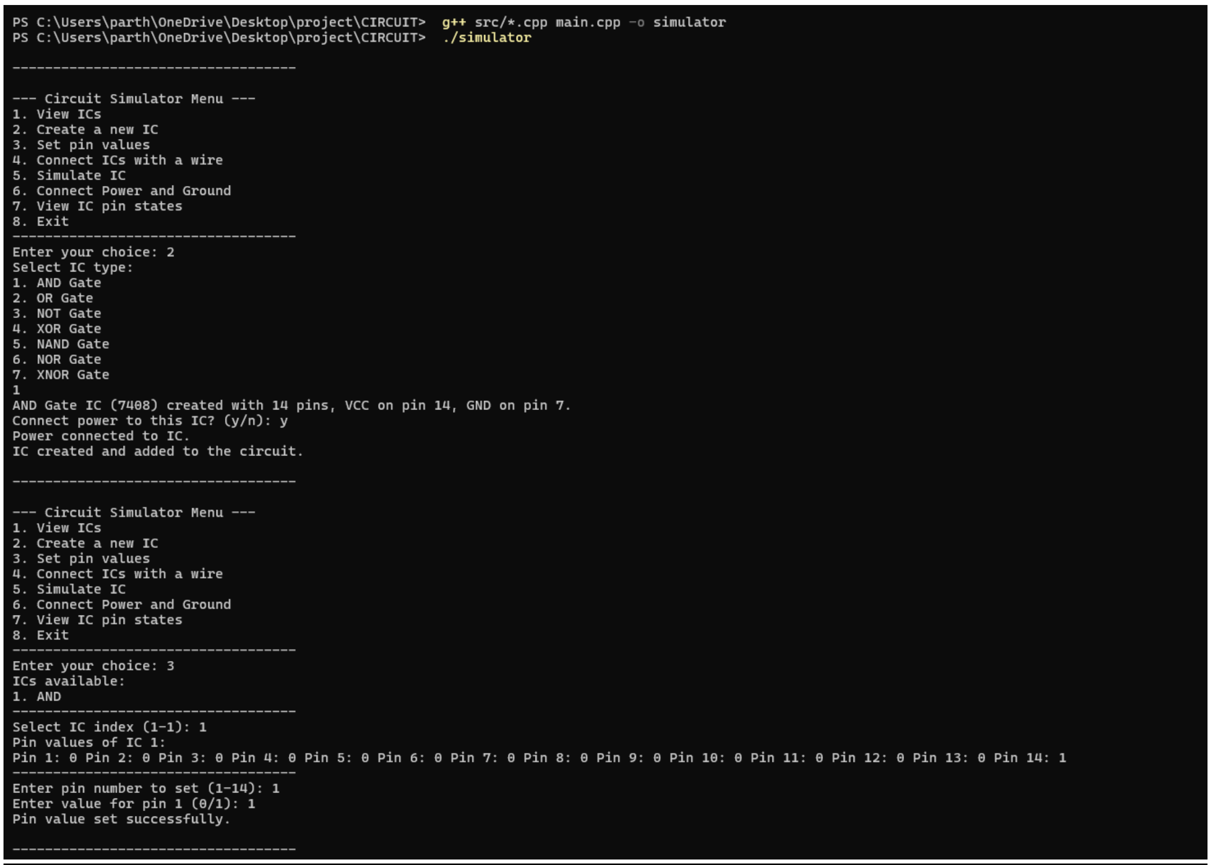
5. **Extensibility**

- Add additional ICs like multiplexers, demultiplexers, flip-flops, etc.
- Support for larger IC pin configurations (e.g., 16-pin, 20-pin ICs).
- Add support for sequential logic circuits (e.g., Flip-Flops, Counters).

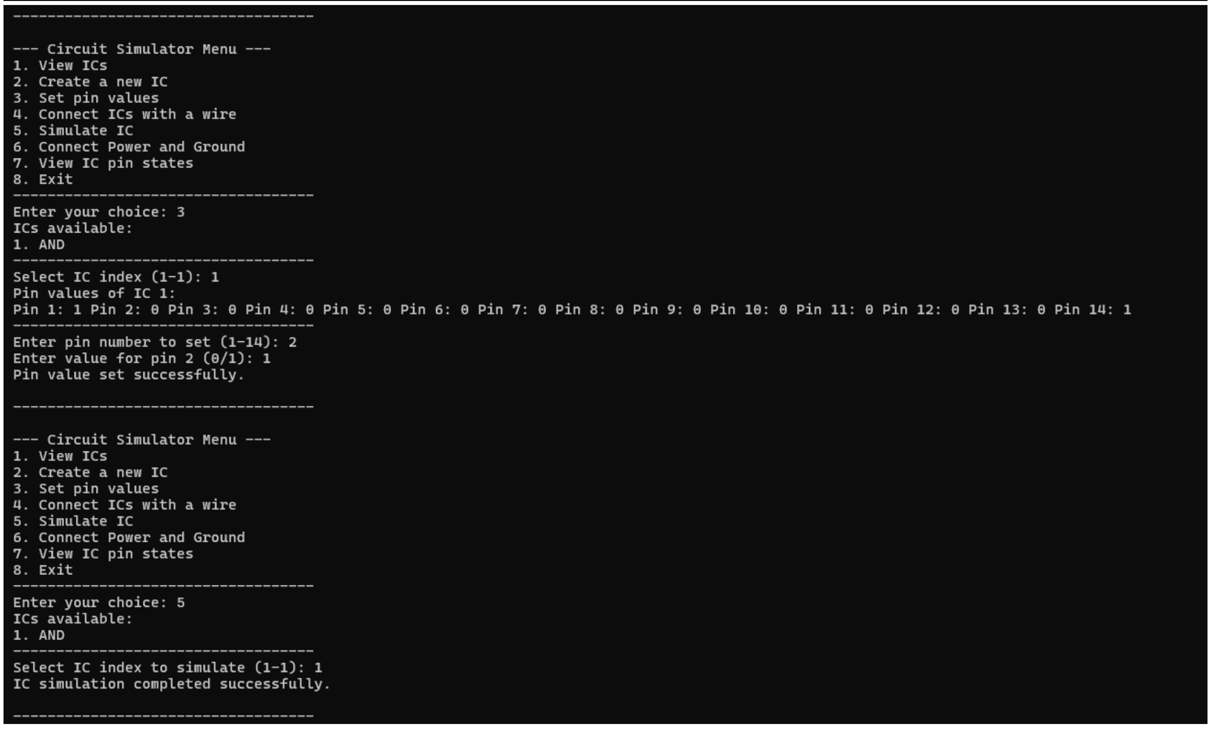
License

This project, **Digital Circuit Simulator**, was created by OOPS Team - 64, **Harith Yerragolam** and **Parth Pandey**.

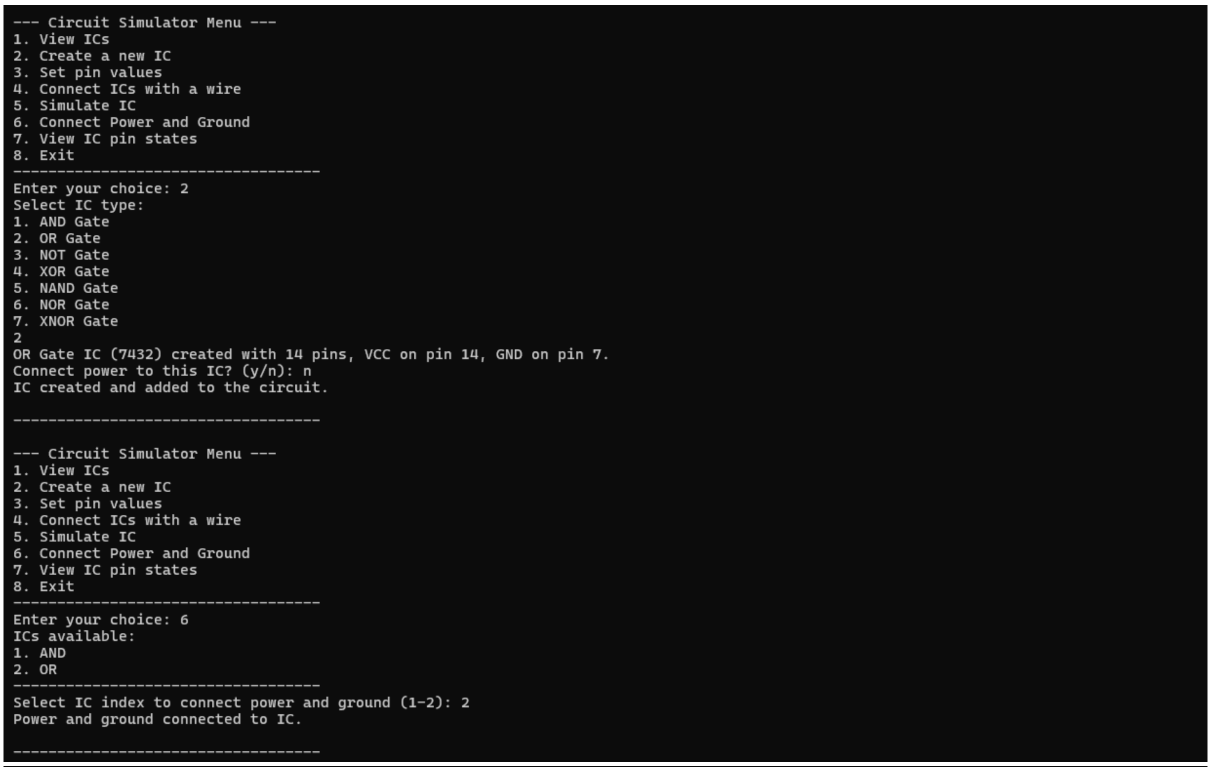
OUTPUT SCREENSHOTS



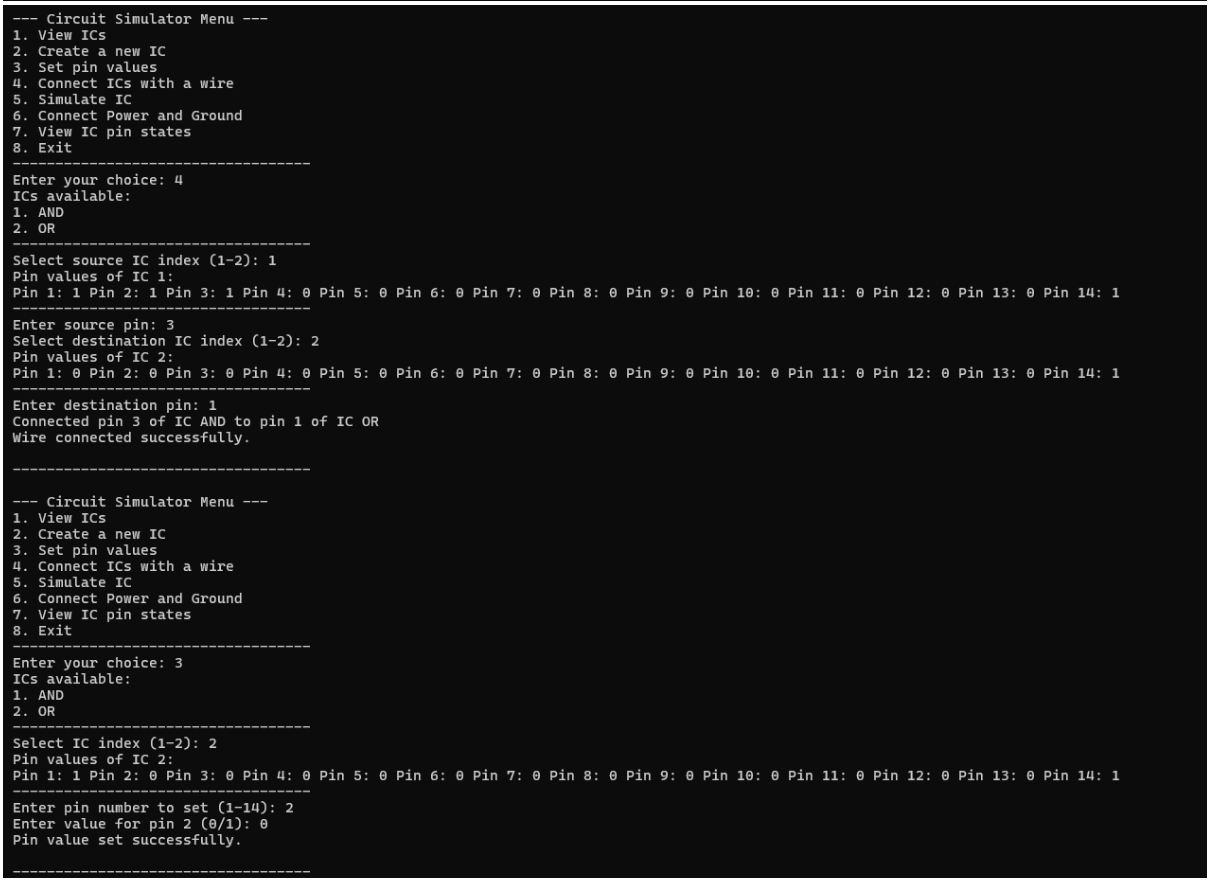
```
PS C:\Users\parth\OneDrive\Desktop\project\CIRCUIT>  g++ src/*.cpp main.cpp -o simulator
PS C:\Users\parth\OneDrive\Desktop\project\CIRCUIT>  ./simulator

-----------------------------------

--- Circuit Simulator Menu ---
1. View ICs
2. Create a new IC
3. Set pin values
4. Connect ICs with a wire
5. Simulate IC
6. Connect Power and Ground
7. View IC pin states
8. Exit
-----------------------------------
Enter your choice: 2
Select IC type:
1. AND Gate
2. OR Gate
3. NOT Gate
4. XOR Gate
5. NAND Gate
6. NOR Gate
7. XNOR Gate
1
AND Gate IC (7408) created with 14 pins, VCC on pin 14, GND on pin 7.
Connect power to this IC? (y/n): y
Power connected to IC.
IC created and added to the circuit.

-----------------------------------

--- Circuit Simulator Menu ---
1. View ICs
2. Create a new IC
3. Set pin values
4. Connect ICs with a wire
5. Simulate IC
6. Connect Power and Ground
7. View IC pin states
8. Exit
-----------------------------------
Enter your choice: 3
ICs available:
1. AND
-----------------------------------
Select IC index (1-1): 1
Pin values of IC 1:
Pin 1: 0 Pin 2: 0 Pin 3: 0 Pin 4: 0 Pin 5: 0 Pin 6: 0 Pin 7: 0 Pin 8: 0 Pin 9: 0 Pin 10: 0 Pin 11: 0 Pin 12: 0 Pin 13: 0 Pin 14: 1
-----------------------------------
Enter pin number to set (1-14): 1
Enter value for pin 1 (0/1): 1
Pin value set successfully.

-----------------------------------
```



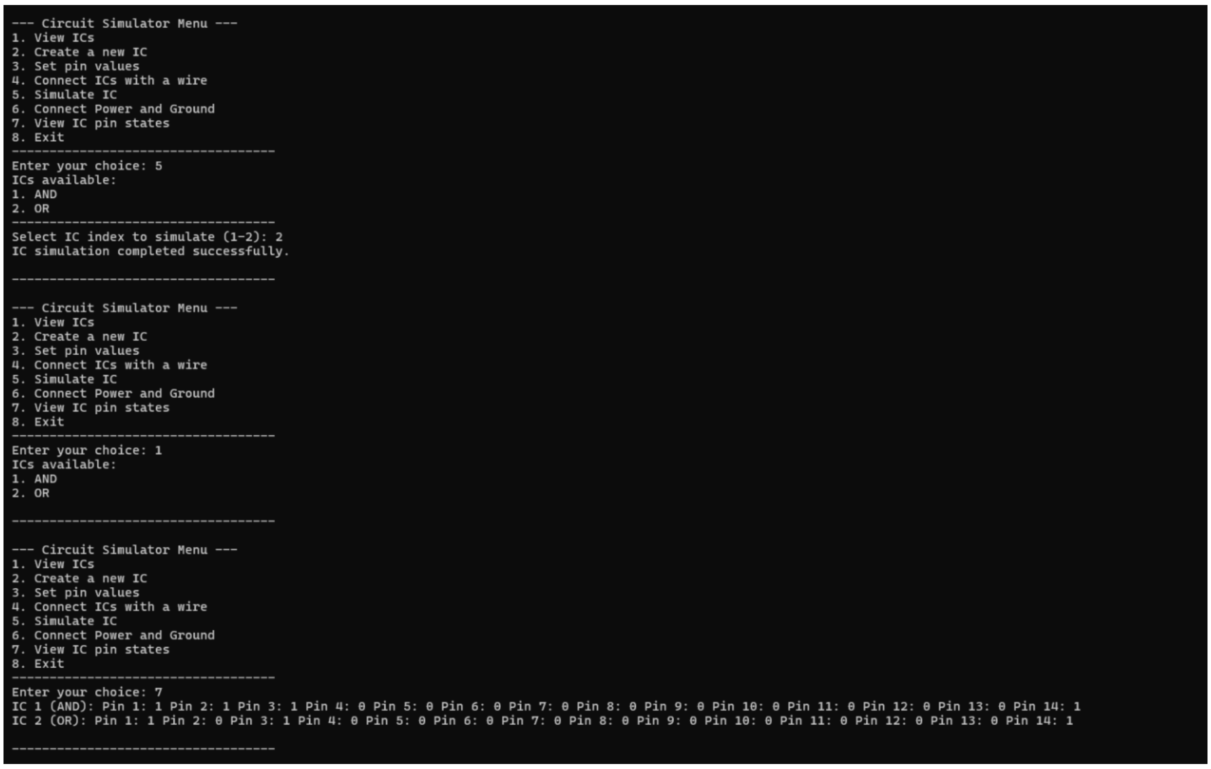
```
-----------------------------------

--- Circuit Simulator Menu ---
1. View ICs
2. Create a new IC
3. Set pin values
4. Connect ICs with a wire
5. Simulate IC
6. Connect Power and Ground
7. View IC pin states
8. Exit
-----------------------------------
Enter your choice: 3
ICs available:
1. AND
-----------------------------------
Select IC index (1-1): 1
Pin values of IC 1:
Pin 1: 1 Pin 2: 0 Pin 3: 0 Pin 4: 0 Pin 5: 0 Pin 6: 0 Pin 7: 0 Pin 8: 0 Pin 9: 0 Pin 10: 0 Pin 11: 0 Pin 12: 0 Pin 13: 0 Pin 14: 1
-----------------------------------
Enter pin number to set (1-14): 2
Enter value for pin 2 (0/1): 1
Pin value set successfully.

-----------------------------------

--- Circuit Simulator Menu ---
1. View ICs
2. Create a new IC
3. Set pin values
4. Connect ICs with a wire
5. Simulate IC
6. Connect Power and Ground
7. View IC pin states
8. Exit
-----------------------------------
Enter your choice: 5
ICs available:
1. AND
-----------------------------------
Select IC index to simulate (1-1): 1
IC simulation completed successfully.

-----------------------------------
```



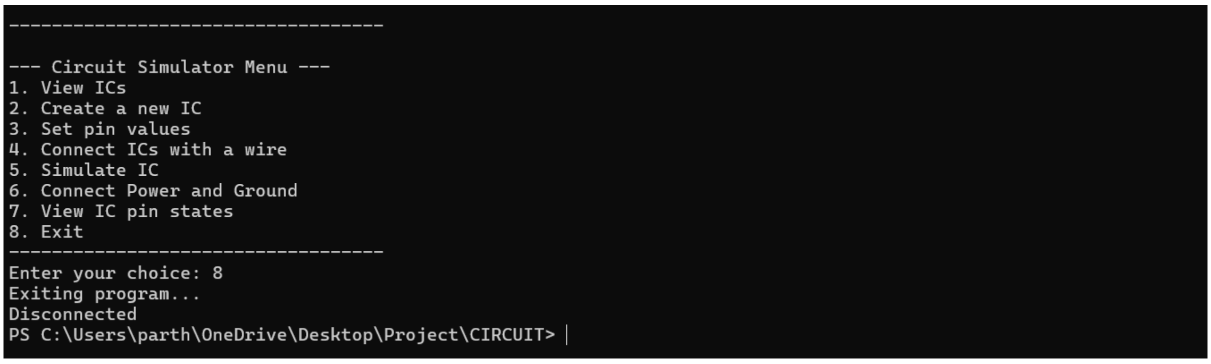
```
--- Circuit Simulator Menu ---
1. View ICs
2. Create a new IC
3. Set pin values
4. Connect ICs with a wire
5. Simulate IC
6. Connect Power and Ground
7. View IC pin states
8. Exit
-----------------------------------
Enter your choice: 2
Select IC type:
1. AND Gate
2. OR Gate
3. NOT Gate
4. XOR Gate
5. NAND Gate
6. NOR Gate
7. XNOR Gate
2
OR Gate IC (7432) created with 14 pins, VCC on pin 14, GND on pin 7.
Connect power to this IC? (y/n): n
IC created and added to the circuit.

-----------------------------------

--- Circuit Simulator Menu ---
1. View ICs
2. Create a new IC
3. Set pin values
4. Connect ICs with a wire
5. Simulate IC
6. Connect Power and Ground
7. View IC pin states
8. Exit
-----------------------------------
Enter your choice: 6
ICs available:
1. AND
2. OR
-----------------------------------
Select IC index to connect power and ground (1-2): 2
Power and ground connected to IC.

-----------------------------------
```



```
--- Circuit Simulator Menu ---
1. View ICs
2. Create a new IC
3. Set pin values
4. Connect ICs with a wire
5. Simulate IC
6. Connect Power and Ground
7. View IC pin states
8. Exit
-----------------------------------
Enter your choice: 4
ICs available:
1. AND
2. OR
-----------------------------------
Select source IC index (1-2): 1
Pin values of IC 1:
Pin 1: 1 Pin 2: 1 Pin 3: 1 Pin 4: 0 Pin 5: 0 Pin 6: 0 Pin 7: 0 Pin 8: 0 Pin 9: 0 Pin 10: 0 Pin 11: 0 Pin 12: 0 Pin 13: 0 Pin 14: 1
-----------------------------------
Enter source pin: 3
Select destination IC index (1-2): 2
Pin values of IC 2:
Pin 1: 0 Pin 2: 0 Pin 3: 0 Pin 4: 0 Pin 5: 0 Pin 6: 0 Pin 7: 0 Pin 8: 0 Pin 9: 0 Pin 10: 0 Pin 11: 0 Pin 12: 0 Pin 13: 0 Pin 14: 1
-----------------------------------
Enter destination pin: 1
Connected pin 3 of IC AND to pin 1 of IC OR
Wire connected successfully.

-----------------------------------

--- Circuit Simulator Menu ---
1. View ICs
2. Create a new IC
3. Set pin values
4. Connect ICs with a wire
5. Simulate IC
6. Connect Power and Ground
7. View IC pin states
8. Exit
-----------------------------------
Enter your choice: 3
ICs available:
1. AND
2. OR
-----------------------------------
Select IC index (1-2): 2
Pin values of IC 2:
Pin 1: 1 Pin 2: 0 Pin 3: 0 Pin 4: 0 Pin 5: 0 Pin 6: 0 Pin 7: 0 Pin 8: 0 Pin 9: 0 Pin 10: 0 Pin 11: 0 Pin 12: 0 Pin 13: 0 Pin 14: 1
-----------------------------------
Enter pin number to set (1-14): 2
Enter value for pin 2 (0/1): 0
Pin value set successfully.

-----------------------------------
```



```
--- Circuit Simulator Menu ---
1. View ICs
2. Create a new IC
3. Set pin values
4. Connect ICs with a wire
5. Simulate IC
6. Connect Power and Ground
7. View IC pin states
8. Exit
-----------------------------------
Enter your choice: 5
ICs available:
1. AND
2. OR
-----------------------------------
Select IC index to simulate (1-2): 2
IC simulation completed successfully.

-----------------------------------

--- Circuit Simulator Menu ---
1. View ICs
2. Create a new IC
3. Set pin values
4. Connect ICs with a wire
5. Simulate IC
6. Connect Power and Ground
7. View IC pin states
8. Exit
-----------------------------------
Enter your choice: 1
ICs available:
1. AND
2. OR

-----------------------------------

--- Circuit Simulator Menu ---
1. View ICs
2. Create a new IC
3. Set pin values
4. Connect ICs with a wire
5. Simulate IC
6. Connect Power and Ground
7. View IC pin states
8. Exit
-----------------------------------
Enter your choice: 7
IC 1 (AND): Pin 1: 1 Pin 2: 1 Pin 3: 1 Pin 4: 0 Pin 5: 0 Pin 6: 0 Pin 7: 0 Pin 8: 0 Pin 9: 0 Pin 10: 0 Pin 11: 0 Pin 12: 0 Pin 13: 0 Pin 14: 1
IC 2 (OR): Pin 1: 1 Pin 2: 0 Pin 3: 1 Pin 4: 0 Pin 5: 0 Pin 6: 0 Pin 7: 0 Pin 8: 0 Pin 9: 0 Pin 10: 0 Pin 11: 0 Pin 12: 0 Pin 13: 0 Pin 14: 1

-----------------------------------
```



```
-----------------------------------

--- Circuit Simulator Menu ---
1. View ICs
2. Create a new IC
3. Set pin values
4. Connect ICs with a wire
5. Simulate IC
6. Connect Power and Ground
7. View IC pin states
8. Exit
-----------------------------------
Enter your choice: 8
Exiting program...
Disconnected
PS C:\Users\parth\OneDrive\Desktop\Project\CIRCUIT>
```