Digital Circuit Simulation Project

This project simulates a Digital Circuit System 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 allows users to manipulate and simulate the behavior of various digital ICs.

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

- **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.
- **Virtual Functions**: Define a `simulate()` function for IC-specific behavior and execute digital logic.

Getting Started

Prerequisites

To compile and run this project, you need:

- C++ compiler supporting C++11 or later (e.g., g++)
- 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!

Make installation

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:

```
./DigitalCircuitSimulator
```

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.

Example

Below is a sample usage example:

```
 #include "NANDGateIC.hpp"
#include "NORGateIC.hpp"
#include "XNORGateIC.hpp"
#include "ANDGateIC.hpp"
#include "ORGateIC.hpp"
#include "XORGateIC.hpp"
#include "NOTGateIC.hpp"

int main()
{
    //implementing ICs
    //A MORE EXTENSIVE TEST DRIVER WILL BE PROVIDED IN THE FINAL SUBMISSION
    ANDGateIC andGateIC;

    ORGateIC orGateIC;

    NOTGateIC notGateIC;

    XORGateIC xorGateIC;

    NANDGateIC nandGateIC;

    NORGateIC norGateIC;

    XNORGateIC xnorGateIC;

    //connecting ICs
    andGateIC += "VCC";
    andGateIC += "GND";

    orGateIC += "VCC";
    orGateIC += "GND";

    notGateIC += "VCC";
    notGateIC += "GND";

    xorGateIC += "VCC";
    xorGateIC += "GND";

    nandGateIC += "VCC";
    nandGateIC += "GND";

    norGateIC += "VCC";
    norGateIC += "GND";

    xnorGateIC += "VCC";
    xnorGateIC += "GND";

    //setting pins

    andGateIC[1] = 1;
    andGateIC[2] = 1;

    cout << "AND Gate inputs: pin - 1: " << andGateIC[1] << " pin - 2: " << andGateIC[2] << endl;

    orGateIC[1] = 1;
    orGateIC[2] = 1;
```

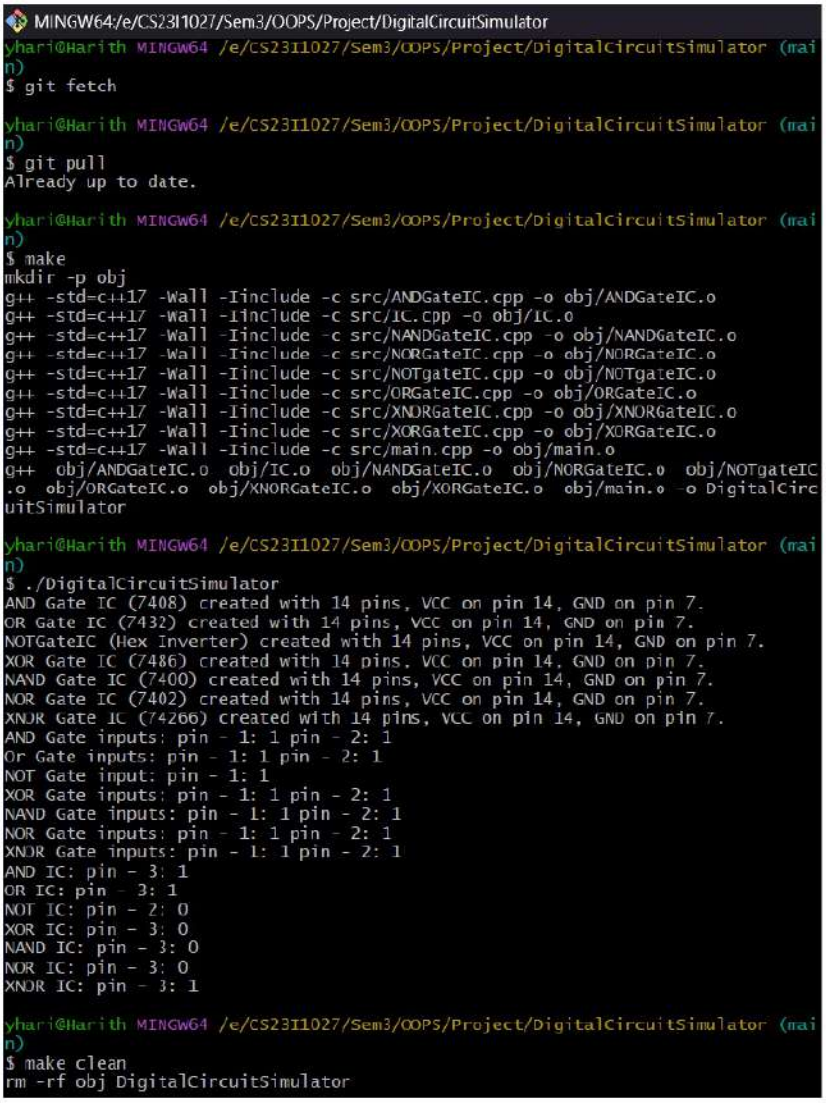
```
    cout << "Or Gate inputs: pin - 1: " << andGateIC[1] << " pin - 2: " << andGateIC[2] << endl;

    notGateIC[1] = 1;

    cout << "NOT Gate input: pin - 1: " << notGateIC[1] << endl;

    xorGateIC[1] = 1;
    xorGateIC[2] = 1;

    cout << "XOR Gate inputs: pin - 1: " << xorGateIC[1] << " pin - 2: " << xorGateIC[2] << endl;

    nandGateIC[1] = 1;
    nandGateIC[2] = 1;

    cout << "NAND Gate inputs: pin - 1: " << nandGateIC[1] << " pin - 2: " << nandGateIC[2] << endl;

    norGateIC[1] = 1;
    norGateIC[2] = 1;

    cout << "NOR Gate inputs: pin - 1: " << norGateIC[1] << " pin - 2: " << norGateIC[2] << endl;

    xnorGateIC[1] = 1;
    xnorGateIC[2] = 1;

    cout << "XNOR Gate inputs: pin - 1: " << xnorGateIC[1] << " pin - 2: " << xnorGateIC[2] << endl;

    //simulating ICs
    andGateIC.simulate();
    orGateIC.simulate();
    notGateIC.simulate();
    xorGateIC.simulate();
    nandGateIC.simulate();
    norGateIC.simulate();
    xnorGateIC.simulate();

    //Results
    cout << "AND IC: pin - 3: " << andGateIC[3] << endl;
    cout << "OR IC: pin - 3: " << orGateIC[3] << endl;
    cout << "NOT IC: pin - 2: " << notGateIC[2] << endl;
    cout << "XOR IC: pin - 3: " << xorGateIC[3] << endl;
    cout << "NAND IC: pin - 3: " << nandGateIC[3] << endl;
    cout << "NOR IC: pin - 3: " << norGateIC[3] << endl;
    cout << "XNOR IC: pin - 3: " << xnorGateIC[3] << endl;

    return 0;
}
```

Future Implementation

- A `Breadboard` Class is planned to be implemetated, along with various other features, like **integrating the IC and the `Breadboard` Class**.
- More ICs to be implemented (Adders, Subtractors, Comparators, Encoders, Decoders, Multiplexers, Demultiplexers, Flip Flops, Counters, Shift Registers
- Inputs shall be user given entirely.

License

Output Screenshots Attached below:



```
MINGW64:/e/CS23I1027/Sem3/OOPS/Project/DigitalCircuitSimulator
yhari@Harith MINGW64 /e/CS23I1027/Sem3/OOPS/Project/DigitalCircuitSimulator (mai
n)
$ git fetch

yhari@Harith MINGW64 /e/CS23I1027/Sem3/OOPS/Project/DigitalCircuitSimulator (mai
n)
$ git pull
Already up to date.

yhari@Harith MINGW64 /e/CS23I1027/Sem3/OOPS/Project/DigitalCircuitSimulator (mai
n)
$ make
mkdir -p obj
g++ -std=c++17 -Wall -Iinclude -c src/ANDGateIC.cpp -o obj/ANDGateIC.o
g++ -std=c++17 -Wall -Iinclude -c src/IC.cpp -o obj/IC.o
g++ -std=c++17 -Wall -Iinclude -c src/NANDGateIC.cpp -o obj/NANDGateIC.o
g++ -std=c++17 -Wall -Iinclude -c src/NORGateIC.cpp -o obj/NORGateIC.o
g++ -std=c++17 -Wall -Iinclude -c src/NOTgateIC.cpp -o obj/NOTgateIC.o
g++ -std=c++17 -Wall -Iinclude -c src/ORGateIC.cpp -o obj/ORGateIC.o
g++ -std=c++17 -Wall -Iinclude -c src/XNORGateIC.cpp -o obj/XNORGateIC.o
g++ -std=c++17 -Wall -Iinclude -c src/XORGateIC.cpp -o obj/XORGateIC.o
g++ -std=c++17 -Wall -Iinclude -c src/main.cpp -o obj/main.o
g++  obj/ANDGateIC.o  obj/IC.o  obj/NANDGateIC.o  obj/NORGateIC.o  obj/NOTgateIC
.o  obj/ORGateIC.o  obj/XNORGateIC.o  obj/XORGateIC.o  obj/main.o  -o DigitalCirc
uitSimulator

yhari@Harith MINGW64 /e/CS23I1027/Sem3/OOPS/Project/DigitalCircuitSimulator (mai
n)
$ ./DigitalCircuitSimulator
AND Gate IC (7408) created with 14 pins, VCC on pin 14, GND on pin 7.
OR Gate IC (7432) created with 14 pins, VCC on pin 14, GND on pin 7.
NOTGateIC (Hex Inverter) created with 14 pins, VCC on pin 14, GND on pin 7.
XOR Gate IC (7486) created with 14 pins, VCC on pin 14, GND on pin 7.
NAND Gate IC (7400) created with 14 pins, VCC on pin 14, GND on pin 7.
NOR Gate IC (7402) created with 14 pins, VCC on pin 14, GND on pin 7.
XNOR Gate IC (74266) created with 14 pins, VCC on pin 14, GND on pin 7.
AND Gate inputs: pin - 1: 1 pin - 2: 1
Or Gate inputs: pin - 1: 1 pin - 2: 1
NOT Gate input: pin - 1: 1
XOR Gate inputs: pin - 1: 1 pin - 2: 1
NAND Gate inputs: pin - 1: 1 pin - 2: 1
NOR Gate inputs: pin - 1: 1 pin - 2: 1
XNOR Gate inputs: pin - 1: 1 pin - 2: 1
AND IC: pin - 3: 1
OR IC: pin - 3: 1
NOT IC: pin - 2: 0
XOR IC: pin - 3: 0
NAND IC: pin - 3: 0
NOR IC: pin - 3: 0
XNOR IC: pin - 3: 1

yhari@Harith MINGW64 /e/CS23I1027/Sem3/OOPS/Project/DigitalCircuitSimulator (mai
n)
$ make clean
rm -rf obj DigitalCircuitSimulator
```

```
PS D:\Parth\CIRCUIT> g++ *.hpp *.cpp
PS D:\Parth\CIRCUIT> ./a.exe
AND Gate IC (7408) created with 14 pins, VCC on pin 14, GND on pin 7.
OR Gate IC (7432) created with 14 pins, VCC on pin 14, GND on pin 7.
NOTGateIC (Hex Inverter) created with 14 pins, VCC on pin 14, GND on pin 7.
XOR Gate IC (7486) created with 14 pins, VCC on pin 14, GND on pin 7.
NAND Gate IC (7400) created with 14 pins, VCC on pin 14, GND on pin 7.
NOR Gate IC (7402) created with 14 pins, VCC on pin 14, GND on pin 7.
XNOR Gate IC (74266) created with 14 pins, VCC on pin 14, GND on pin 7.
AND Gate inputs: pin - 1: 1 pin - 2: 1
Or Gate inputs: pin - 1: 1 pin - 2: 1
NOT Gate input: pin - 1: 1
XOR Gate inputs: pin - 1: 1 pin - 2: 1
NAND Gate inputs: pin - 1: 1 pin - 2: 1
NOR Gate inputs: pin - 1: 1 pin - 2: 1
XNOR Gate inputs: pin - 1: 1 pin - 2: 1
AND IC: pin - 3: 1
OR IC: pin - 3: 1
NOT IC: pin - 2: 0
XOR IC: pin - 3: 0
NAND IC: pin - 3: 0
NOR IC: pin - 3: 0
XNOR IC: pin - 3: 1
PS D:\Parth\CIRCUIT>
```