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
- Features
- Getting Started
- Usage
- Classes and Their Functions
- Code Structure
- Example
- Future Implementation
- License

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 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/ (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:

- Create ICs.
- Set pin values.
- Connect ICs with wires.
- Simulate the circuit.

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/Wire.hpp"
#include <ANDGateIC.hpp>
#include <ORGateIC.hpp>

// In main.cpp
int main() {
    ANDGateIC andGateIC;
    ORGateIC orGateIC;

    andGateIC += "VCC";
    andGateIC += "GND";

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

    // Set initial values
    andGateIC[1] = 1;
    andGateIC[2] = 1;

    // Connect ICs using wires
    Wire wire1(&andGateIC, 3, &orGateIC, 1); // Connect AND output to OR input
    wire1.connect();

    orGateIC[2] = 0;

    andGateIC.simulate();
    orGateIC.simulate();

    // Results
    cout << "AND IC Output (Pin 3): " << andGateIC[3] << endl;
    cout << "OR IC Output (Pin 3): " << orGateIC[3] << endl;

    // Cleanup
    wire1.disconnect();
    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**.