BLG 252E - Object Oriented Programming
Assignment #3

Due: April 12th 23:59

Introduction

This homework assignment tasks you with implementing a simple digital logic simulator in C++. The simulator will allow the user to define components such as logic gates and pins, connect them with wires, and simulate the behavior of complex logic expressions. The focus will be on practicing object-oriented programming (OOP) principles, including inheritance, encapsulation, and polymorphism.

For any issues regarding the assignment, you can ask your questions on Ninova Message Board. Before writing your question, please check whether your question has been asked. You can also contact T.A. M. Alpaslan Tavukçu (**tavukcu22@itu.edu.tr**).

1 Implementation Details

For this project, you'll start with skeleton code that helps build your final project. It's important to understand how this code is organized and how different parts connect. The code already has all the necessary libraries, so you do not need to add any more.

Your task is to create several classes: **Pin**, **Component**, **Gate**, **ANDGate**, **ORGate**, **NOTGate**,**Wire**, and **Circuit**. These classes work together to create a digital logic simulator. As you go through the provided code, you will see how these classes interact.

Each class has specific functions and roles, which are briefly described here. To get a full understanding and to complete your project well, make sure to study the provided code in detail.

1.1 Component

The Component class serves as an abstract base class within a digital circuit simulator framework, designed to outline the fundamental structure and behavior that all specific circuit components must inherit and implement.

1.2 Pin

The Pin class is a foundational component of digital circuit simulations, representing individual pins that can hold and transmit digital signals (either true or false). Each pin functions primarily as an interface point for components in the circuit, serving either as an input or an output.

1.3 Gate

The Gate class is an abstract base class in a digital circuit simulation framework, designed to represent logic gates, which are fundamental components in digital circuits. This class extends the Component class, inheriting its basic properties and adding specific functionality for gates. It manages multiple input pins and one output pin, handling how signals are received and produced based on logical operations specific to the type of gate (like AND, OR, etc.). The Gate class requires derived classes to implement their own logic in the evaluate method, ensuring each gate type performs its intended function. This structure allows for the easy addition of different types of gates and facilitates the simulation of complex logical operations within circuits.

1.4 ANDGate

The ANDGate class is a specific type of logic gate used in digital circuit simulations. It inherits from the general Gate class, to perform the logical AND operation. This gate takes multiple input signals and produces an output signal.

1.5 ORGate

The ORGate class is designed to simulate the behavior of an OR logic gate within digital circuits. This class extends the general Gate class, incorporating the specific functionality needed for the OR operation.

1.6 NOTGate

The NOTGate class represents a NOT logic gate in digital circuit simulations. It is a specialized form of the generic Gate class, to perform the logical NOT operation. This type of gate has a single input and one output.

1.7 Wire

The Wire class models the connections between components in a digital circuit simulation, particularly between the pins of these components. It serves as a conduit for transferring electrical signals from one component to another, typically from an output pin of one component to the input pin of another. In the simulation context, a Wire class includes attributes for both a source and a destination pin, defining the direction of signal flow. It also contains a method called propagateSignal, which is responsible for moving the signal from the source pin to the destination pin.

1.8 Circuit

The Circuit class acts as a container for managing and simulating a complete digital circuit within a simulation framework. It is designed to hold and orchestrate various components such as gates and wires that collectively make up the circuit. The class has a list of Component pointers, representing different circuit elements, and a list of Wire pointers, which connect these components by transmitting signals between them.

Key functionalities of the Circuit class include methods to add components (addComponent) and wires (addWire) to the circuit. This allows for dynamic construction and modification of the circuit as needed. Another critical method is simulate, which initiates the circuit operation by sequentially activating the evaluate method of each component.

In designing the simulate method for a digital circuit simulation, a key assumption is that components are added to the circuit in an order that correctly supports the logical dependencies and sequence of operations. This means that each component should be evaluated in a way that respects the flow of signals through the circuit, ensuring that any component's output used as an input for another is already updated. Therefore, when you add components to your simulation, it's crucial to consider the timing and dependency of signals between these components. This ordered approach helps in achieving accurate simulations and mimicking the behavior of a physical circuit where signal propagation has a natural sequence based on the circuit design.

2 Example Usage

In the following code snippets, you can see the main function and how the Logic Simulator framework is used to implement the logic circuit for: (A and B) or (C and not D).

```
int main() {
    // Initialize a new digital circuit container
    Circuit circuit;

    // Create input pins as components with unique identifiers
    Pin * A = new Pin("A");
```

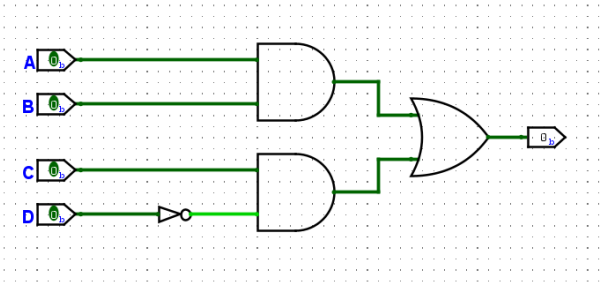

Figure 1: Logisim diagram for: (A and B) or (C and not D)

```
Pin * B = new Pin("B");
Pin * C = new Pin("C");
Pin * D = new Pin("D");

// Set initial signals for all pins to false (LOW state)
A->setSignal(false);
B->setSignal(false);
C->setSignal(false);
D->setSignal(false);

// Create gates, each with a unique identifier
ANDGate * and1 = new ANDGate("AND1");
ANDGate * and2 = new ANDGate("AND2");
NOTGate * not1 = new NOTGate("NOT1");
ORGate * or1 = new ORGate("OR1");

// Add components to the circuit in logical order to ensure correct dependencies
circuit.addComponent(A);
circuit.addComponent(B);
circuit.addComponent(C);
circuit.addComponent(D);
circuit.addComponent(not1);
circuit.addComponent(and1);
circuit.addComponent(and2);
circuit.addComponent(or1);

// Create wires that connect outputs from some components to inputs of others
circuit.addWire(new Wire(A, and1->getInputPin(0)));
circuit.addWire(new Wire(B, and1->getInputPin(1)));
circuit.addWire(new Wire(C, and2->getInputPin(0)));
circuit.addWire(new Wire(D, not1->getInputPin(0)));
circuit.addWire(new Wire(not1->getOutputPin(), and2->getInputPin(1)));
circuit.addWire(new Wire(and1->getOutputPin(), or1->getInputPin(0)));
circuit.addWire(new Wire(and2->getOutputPin(), or1->getInputPin(1)));

// Loop through all possible combinations of input signals (0 to 15)
for (int i = 0; i < 16; ++i) {
    // Set input signals based on the current count in binary form
    A->setSignal(i & 1);
    B->setSignal(i & 2);
    C->setSignal(i & 4);
    D->setSignal(i & 8);

    // Perform a simulation of the circuit with the current input setup
```

```
        circuit.simulate();

        // Print the input values and the resulting output of the circuit
        std::cout << "Inputs: A=" << (i & 1) << ", B=" << ((i & 2) >> 1) << ", C=" << ((i & 4) >> 2)
                  << " | Output: " << or1->getOutputPin()->getSignal() << std::endl;
    }

    return 0;
}
```

3 Test Cases

3.1 Case 1

Try to implement and simulate following logic circuit: (A and B) or (A or B)

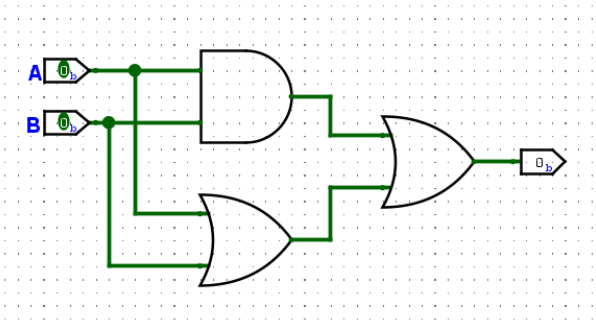

Figure 2: Logisim diagram for: (A and B) or (A or B)

3.2 Case 2

Try to implement and simulate following logic circuit: (A and B) or ((B and C) and (B or C))

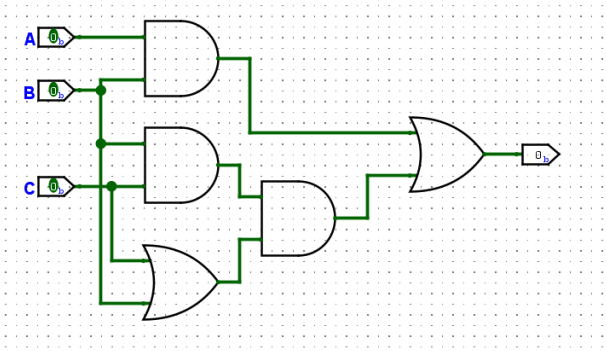

Figure 3: Logisim diagram for: (A and B) or ((B and C) and (B or C))

4 How to compile, run, and test your code

If you want to compile and run the provided code on a terminal, you can use these commands:

```
g++ -Wall -Werror src/*.cpp -Iinclude -o bin/main

./bin/main
```

You should run Valgrind on a terminal to check your assignment with the command:

```
valgrind --tool=memcheck --leak-check=full --show-leak-kinds=all \
--track-origins=yes --log-file=valgrind_log.txt ./bin/main
```

5 Submission

Compress your whole project folder and submit it as a zip file on Ninova. Be aware that your grades will be based on the quiz that is going to be held two weeks later. The exact date will be announced.