Semester 2 - Programming in C#

Assignment 2 - Combinatorial Logic Simulator

Jaap Geurts

02 May 2020

Requirements

Visual Studio 2010 or higher

Goal

In this assignment you'll build a C# console application in which you'll apply the concepts and constructs that you've learned in the past 12 weeks.

You'll research what composition is in object oriented design by programming a combinatorial logic simulator.

In your work you must demonstrate your understanding of the following topics

1. Inheritance
2. Composition
3. Polymorphism
4. Lists and Arrays
5. Interfaces / Abstract classes
6. Assertions
7. Custom Exceptions

Required Research

1. Inheritance and Composition
2. Combinatorial Logic
3. Truth Tables
4. Half adder

Format

- This assignment is an individual assignment.
- Submission deadline: Monday of week 14, 8am.

- Hand-in through Canvas.
- Upload a ZIP file of the project folder.
- Only hand in your report, diagrams and project code (don't send executables. So clean your project before submission)
- Code must have concise comments, use `/// summary` comments for all methods.
- Documents must have a title block with
  - A title,
  - The date,
  - Your name,
  - Student number

Description

A logic circuit is composed of different logic gates with inputs and outputs. Each of these gates have a variable number of input and output pins. Output pins can be connected to (multiple) input pins to build a larger circuit. The state of the output pins depends on the state of the input pins.

The most common elementary logic gates are: AND, OR, NOT, XOR. Elementary gates can also be combined to make more complex logic components. Some example include half adder, full adder, shift register, d-flipflop to name a few.

There are two special types of elementary gates: NAND(not and) and NOR(not or). They are called universal gates because they can be used to build any other gate.

The symbol for an AND gate is:

Figure 1: And gate

The inputs are A and B, the output is Q.

The truth table for the AND gate is:

A	B	Q
F	F	F
F	T	F
T	F	F
F	F	T

Question 1

Research combinatorial logic

a. Give the definition of combinatorial logic.
b. Describe in your own words what it is. What are the limits of what can be computed with combinatorial logic? Give two examples of what can be, and what cannot be computed with combinatorial logic.

Question 2

Design a program to simulate the basic logic ports: NOT, AND, OR and XOR. The program should produce a truth table for all the inputs and outputs.

We want this application to be extensible. We would like to be able to combine gates to make a larger circuit. We also would like to add extra logic components later. Therefore we'll define an interface:

```
public interface ILogicComponent
{
    // Returns the state of an input pin.
    bool GetInput(int pin);
    // Returns the state of an output in.
    bool GetOutput(int pin);

    // Set the state of an input pin.
    void SetInput(int pin, bool value);

    // Connect an output of this component to an input of another component.
    // Allows multiple connections from the same output to other output pins
    void ConnectOutput(int outputPin, ILogicComponent other, int inputPin);
}
```

Each of your gates should (indirectly) inherit from this interface. Note: You can insert an abstract class to add common behaviour to all classes if you need to.

The main program could look something like this:

```
// Simple and gate
AndGate and = new AndGate();

Console.WriteLine("A B Q");
and.SetInput(AndGate.A, false);
and.SetInput(AndGate.B, false);
Console.WriteLine(String.Format("F F {0}", and.GetOutput(AndGate.Q) ? "T" : "F"));

and.SetInput(AndGate.A, false);
and.SetInput(AndGate.B, true);
Console.WriteLine(String.Format("F F {0}", and.GetOutput(AndGate.Q) ? "T" : "F"));
```

```
and.SetInput(AndGate.A, true);
and.SetInput(AndGate.B, false);
Console.WriteLine(String.Format("F F {0}", and.GetOutput(AndGate.Q) ? "T" : "F"));

and.SetInput(AndGate.A, true);
and.SetInput(AndGate.B, true);
Console.WriteLine(String.Format("F F {0}", and.GetOutput(AndGate.Q) ? "T" : "F"));
```

a. Create a class diagram for your version of the interface and basic logic components.
b. Draw a sequence diagram for producing a truth table for an XOR gate
c. Create an application that simulates the required gates
d. Add a custom exception `InvalidPinException`: thrown when any of the logic gates receive input on a non existing pins
e. Update your class diagram and include the exception.

Question 3

Research half-adder.

a. Give the definition of a half adder.
b. Write down in your own words what a half adder is, what it is used for and what its limits are.
c. Draw the circuit diagram of a half-adder
d. Describe its inputs and outputs.
e. Draw a truth table for a half adder.

Question 4.

We would like to add a half adder to our simulator. To do so we create a new component that inherits from `LogicComponent`. To implement the half adder we have two choices, we program its behaviour directly or, we can use existing gates that we've already built and use those inside our HalfAdder class to program the desired behaviour. The last method is an example of composition.

Research composition.

a. Give the definition of composition in the context of object oriented programming.
b. Write down in your own words what composition is.
c. Give an example of composition.
d. Write what the advantages and disadvantages of composition are.
e. Based on the advantages and disadvantages would you use composition to build the half adder? Explain why.
f. Program the half adder using composition.
g. Add a custom exception `ConnectionAlreadyCreated`: thrown when a connection from an output to an input already exists

h. Update your class diagram and include the exception.

Question 5

a. Add a full adder component to the system. You can either use composition or program it directly.
b. Write a demonstration program to show that your full adder works correctly.

Extra

3 bit full adder

Write a demonstration program which shows a working 3 bit full adder. For example: $3 + 4 = 7$. If the adder overflows, show the value of the carry bit.

Hint: you can write a special input component which takes a number as an input and converts it into a 3 bit output. You can do the opposite for the output.

Build a d-flopflop

Build a d-flipflop component.

Build a graphical interface

Build a graphical application to show a live version of the simulator