BLG252E Object Oriented Programming Project

Spring 2022

1. Objective

The objective of this project is to implement a basic logic simulator using Object-Oriented Programming (OOP) methods and SFML library.

2. Requirements

The project will be implemented using SFML Graphics and Multimedia library (https://www.sfml-dev.org/)

You are recommended to use a Windows machine to do your project. You may use Visual Studio 2017 or later Community version as your build and debug environment. For Visual Studio, download 32-bit version of SFML 2.5.1 (https://www.sfml-dev.org/files/SFML-2.5.1-windows-vc15-32-bit.zip). Refer to lecture 6 recording for the environment setup with Visual Studio on Windows. A sample VC project will be provided with pre-setup environment for you as well. You may use this project as a starter. For Mac and Linux, you need to follow the directions for SFML 2.5.1 for your operating system on this page: https://www.sfml-dev.org/download/sfml/2.5.1/.

3. Project Description

The simulator graphics will be drawn using a set of sprites. A sprite is a 2D graphics drawn at a particular location on screen. There will be a sprite for each type of logic element that will be implemented. You will need to implement the following logic elements:

- 2-input AND gate
- 2-input OR gate
- 2-input XOR gate
- 1-input NOT gate
- D flip-flop
- Logic-0 input
- Logic-1 input
- Clock input
- LED

The simulator window will be as shown in Figure 1. The logic elements will be in a tool pane on the left-hand side of the window. To the right of the tool pane, there will be a workbench on which you can drag and drop elements from the tool pane. At the top of the window, there will be a toolbar with a set of buttons to start and stop the simulation.

The user should be able to connect two logic elements to each other by clicking on (or close to) a terminal pin of the first logic element to select the first pin to connect, then dragging the mouse to extend a wire from that terminal pin, releasing the mouse button on (or close to) a terminal pin of the second logic element. To delete a connection, the user should be able to click anywhere on (or close to) a wire and press the “DEL” key on the keyboard. When the wire is selected, the wire color should change to red.

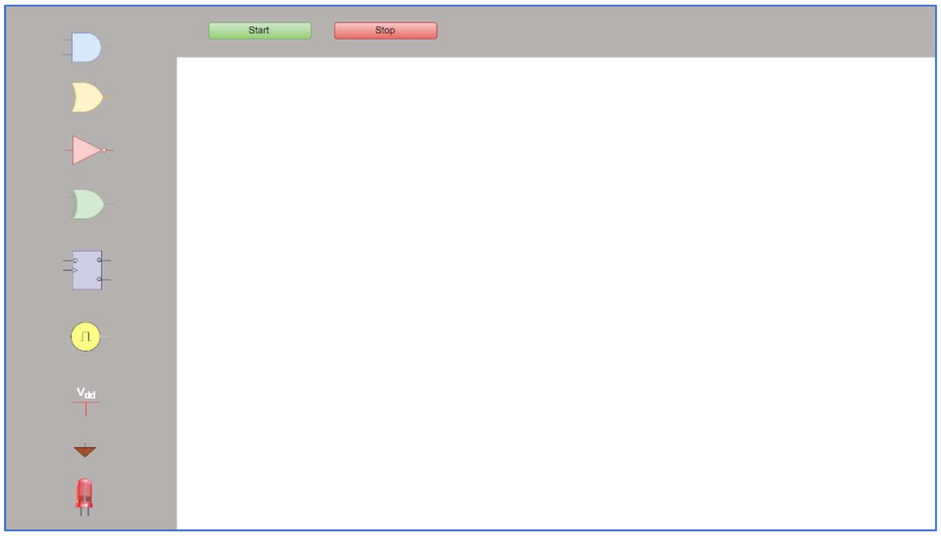

Figure 1. Simulator Window

Similarly, in order to delete a logic element, the user should be able to click on the element and press the "DEL" key on the keyboard. A red rectangle should be drawn around the element when the element is selected. When an element is deleted, all of its wire connections should be deleted as well.

3.1 Classes

You are expected to implement the following classes, but as long as you stick with OOP design rules, the attributes may differ from below depending on your design:

1. Object class

Top level object class. All other drawable elements on screen such as logic elements, wires, toolbar items derive from this.

Attributes:

```
Object* next                //Pointer to next object in the list
bool locked                 //Whether the object can move on screen or is fixed
                           //You can use this flag for toolbar items which cannot move
sf::RenderWindow* window    //Pointer to SFML render window
sf::Texture textures[2]     //SFML texture list for the object (if any). Some objects
                            such as LED element may have multiple textures (for on
                            state and off state), hence this is a list

sf::Sprite sprite           //SFML sprite for the object (if any)
bool state;                 //State of the logic element (may be used to designate
                            button state, D-flipflop state or whether LED is on or off)
bool selected;              //Whether the object is selected for deletion
```

2. Wire class

The wire class that connects two elements through their pins. This class should inherit from the Object class.

Attributes:

```
sf::Vertex line[2]          //End point vertices for the wire
Pin* pins[2];               //A list of pins that this wire is connected to
```

3. Pin class

The pin class that represents the terminal pin of an element. This class should not derive from any other class.

Attributes:

```
enum pinType { INPUT, OUTPUT };          //enum for pin type (input or output pin)
enum pinState { HIGHZ, LOW, HIGH };      //enum for pin state


int index;                               //Index of the pin for an element
                                         //First input pin has index 0, second input
                                         pin has index 1, output pin has index 2, so
                                         on…
pinType type;                            //whether this pin is input or output
bool isSrc[MAX_CONNECTIONS];             //whether this pin is the starting point of
                                         the wire connected to it or the ending point
                                         for that wire for every connection it has
Pin* connectedTo[MAX_CONNECTIONS];       //List of other pins this pin is connected to
Object* wires[MAX_CONNECTIONS];          //Wires connected to this pin
int numConnections;                      //Number of connections on this pin
sf::Vector2f pos;                        //Screen position of the pin

pinState state;                          //Logic state of the signal on this pin
```

4. LogicElement class

Top level class that all other logic elements derive from. Inherits from the Object class.

Attributes:

```
Pin pins[4];                             //List of pins of the logic element
int numPins;                             //Number of pins of the logic element
```

5. AndGate class

Represens an AND gate. Inherits from the LogicElement class. There should be a separate class like this for all other logic elements such as OrGate, XorGate, LED, Logic_0, Logic_1, Clock etc.

The Clock logic element class should have an additional sf::Clock clock attribute to hold the current time. You may use SFML clock and time functions together with this attribute to implement the clock. For simplicity, the clock should have 1Hz frequency.

6. Simulator class

The main simulator class. This class does not inherit from any other class. It should contain the following attributes.

Attributes:

```
sf::RenderWindow* window;                //Pointer to SFML render window
Object* objects;                         //Pointer to a list of objects on screen
```

5. Bonus

There are various bonus opportunities in this project.

1. If you implement wires that can be stretched and bent at 90-degree angles, you will get extra 5 points.

2. If you implement an extra scope element that shows the waveform of a logic signal on the pin that it is connected to, you will get extra 10 points. The waveform should show up when the scope element is double-clicked. It should have simulation time on the x-axis, and the logic level on the y-axis. If the waveform graph can be zoomed in/out (using mouse wheel) and scrolled horizontally (using mouse drag), you will get 5 points more.

6. Project Deliverables

Your submission should include a zip file containing a project report and the contents of your project folder. If your project folder is too large for ninova, please delete the contents of the hidden .vs folder in your project tree before compressing your project. To be able to see this folder, you will need to enable show hidden folders option from file explorer.

1. You need to add comments to your code. Uncommented code will get partial credit. Be reasonable with the number of comments you add. Do not try to comment every line.
2. DO NOT submit files individually. Put them into a compressed zip archive and submit the zip archive. Name your zip archive with your team name such as awesome_team.zip
3. Include your project report in pdf format in your submission.
4. **Only one student** from each group should submit the project.
5. **Sharing code among project teams is NOT allowed. Doing so will cause credit cut.**

Below is the template for the project report.

Introduction
Briefly describe the project goals here.

Team Info
Describe team members and their roles during the completion of the project here.

Implementation
Here, you should describe the classes you implemented, how you draw the logic elements and wires between them, how you keep track of which elements are connected to which and how the simulation works. Point out the object-oriented programming features you used in your implementation.

Try to use block diagrams, flow charts and any other visuals that will help explain your algorithm to a reader who does not know anything about your project.

Discussion
Briefly describe the problems you faced in the implementation of your project and how you could improve it.