**PROJECT REPORT**

# PULCHOWK CAMPUS

# INSTITUTE OF ENGINEERING

# TRIBHUVAN UNIVERSITY

****

**Project: Digital Logic Simulator (MinimaLogic)**

SUBMITTED TO

# DEPARTMENT OF ELECTRONICS AND

# COMPUTER ENGINEERING

**Submitted By:**

Aditi Kharel – PUL077BEI008

Ashutosh Bohara – PUL077BEI012

Pallavi Paudel – PUL077BEI027

Rijan Shrestha – PUL077BEI034

**Introduction:**

This project is a simple Logic Simulator, named **MinimaLogic,** which uses the SDL Library for graphics elements. The users can interact with the program using keyboard or mouse inputs. It allows users to create circuits ranging from a simple 1-bit adder to more complex circuits like 4-bit counters. In fact, one can create any circuit that fits within the available area using the components provided in the program.

# Dependencies:

As mentioned, this program uses the SDL Library for GUI elements. The version of SDL used is SDL2-2.0.14 available at libsdl.org. The program also uses SDL\_ttf, which is an extension to the standard SDL library for font rendering. For this reason, the header files **<SDL.h>** and **<SDL\_ttf.h>** must be included. Besides these, the program also uses the Win32 API available in **<windows.h>** header file. Other standard libraries that are available with all modern development environments have also been used.

# Source Code:

The complete source code for this program is available on GitHub. It can be downloaded using <tinyurl.com/MinimaLogic>. The source code consists of 9 files altogether each serving specific purposes.

* component.c and component.h:

As the name suggests, these files contain all the necessary information about the components used in the program. The header file defines a structure named Component that encompasses the details about a component including its size, position, input source, number of inputs, input state(s), output state(s) and other information which is later used.

The output of any component (except *clock* and *state*) depends on its input(s). To get the desired output for any component from its inputs, the source file defines different component specific functions. The working of these functions is pretty straight-forward as they follow the standard logic operations available in C. As for the *clock*, its output is generated based on the value of *time* variable, which changes as the program progresses, defined in program.c. The *clock* inverts its current state when *time* reaches a certain value. The output of *state* is inverted when the user clicks on it.

* draw.h and draw.c:

These two files contain the variables and functions that are responsible for drawing all the elements that are visible on the screen such as **Buttons**, **Components**, and **Wires**. It also handles rendering text in the SDL window where necessary. The header file defines an enumeration of confirmation flags that are later used to ask the user for confirmation on certain operations.

The standard rendering functions available in the SDL library are used in order to draw **Buttons** and **Components**. However, SDL does not offer the functionality to draw curves. So, a simple algorithm that approximates a cubic Bezier Curve is used to draw wires.

As for displaying text, a character map consisting of all the ASCII characters is predefined when the program starts. The font used is robotoo.ttf. The character map is later used to display any text (ASCII based) on the screen.

* interaction.h and interaction.c:

User interaction is an integral part of any program, even more so for programs that use both mouse and keyboard to take input. These two files are responsible for handling such interactions. The header file defines various structures that are necessary for the **Undo/Redo** functionalities.

The source file defines different functions that determine what will happen when a certain button is pressed or when a component is placed on the grid. Since these functions handle interaction with the user, they are usually only called when an event occurs. An SDL event encompasses mouse clicks, keyboard presses, etc. Different functions are called for different events. This co-ordination is handled in the file program.c.

* program.h and program.c:

To keep the main.c file clean, the main program loop is defined in this file. For this reason, it acts as the central piece of the program that co-ordinates the functions of all other files. To begin with, the header file defines macros for configuring the main window and different elements inside it. Also, the colors that are frequently used in the program are defined here.

The source file can be vaguely divided in to two parts: **Initialization** and **Main Program Loop**. The initialization part is responsible for setting up all the necessary elements needed for the program to function properly. This is a one-time process that runs when the program is initially opens.

The Main Program Loop, as the name suggests, is a loop that runs over and over until the user exits the program. Everything that the user does inside the program is handled in this section. Within each loop, the programs checks for events, performs necessary operations based on them, updates the elements on the screen if required, and redraws all of those elements.

* main.c:

As mentioned earlier, this file is kept as clean as possible by defining the main loop in program.c. Inside this file, the current working directory is changed to the folder containing the executable and font files, so that the font files are always found regardless of where the program is run from. This is done using the **<direct.h>** library.

The main function calls functions for initialization, the main program loop, and finally closing the program.

# Building:

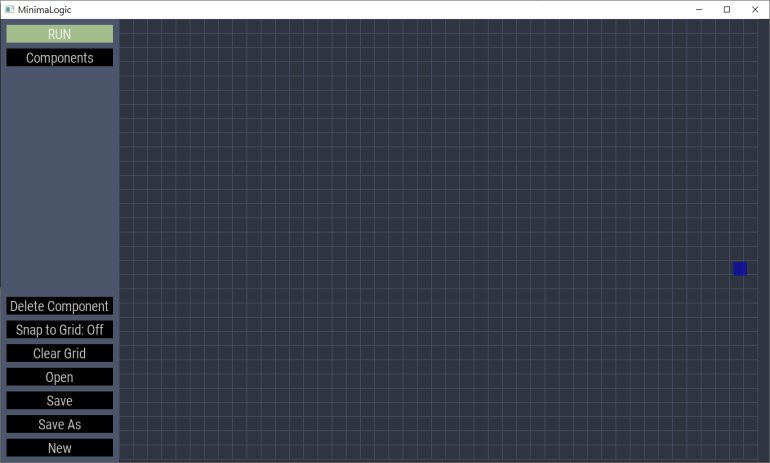
Since this program uses external libraries that are not available in any standard development environment, building the program is not straight-forward. It cannot be built from an IDE without first configuring it to work with SDL library. So, to make it easier for the user to build the program, a windows batch script is provided alongside the source code. Simply running the **build.bat** file, either from terminal or directly from the explorer window, will create an executable file inside the **bin** directory, given that a compiler is installed on the system. Depending on the compilers available, at most 3 folders will be created inside the **bin** directory each containing an executable file named **MinimaLogic.exe**, along with two font files and extensions that are necessary for the program to run.

# Runtime Walkthrough:

Using this program is pretty straight-forward. The main menu is laid out on the left side of the canvas and all the available options are self-explanatory. However, this program is also capable of taking inputs from the keyboard. A list of valid keyboard commands is given below:

* **Shift:** When pressed during the placement of a new component, the component will be aligned with the grid.
* **Delete:** This will delete a component that is either being hovered over or selected, the priority being the one that is hovered over.
* **Equal:** If possible, the number of inputs of the component being placed on the grid will be increased by 1, maximum value being 5.
* **Minus:** If possible, the number of inputs of the component being placed on the grid will be decreased by 1, minimum value being 2.
* **Ctrl+Z:** Reverts the last change made on the grid (Undo).
* **Ctrl+R:** Redo the last change reverted by Undo command.
* **Ctrl+S:** Save changes made to the file currently open, if no file is open then prompt the user to create a new file.
* **Ctrl+O:** Creates a windows dialog box to open an existing file.

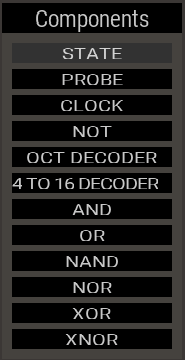
On running the program, the user we be greeted with a window as shown in Figure 1.



Figure

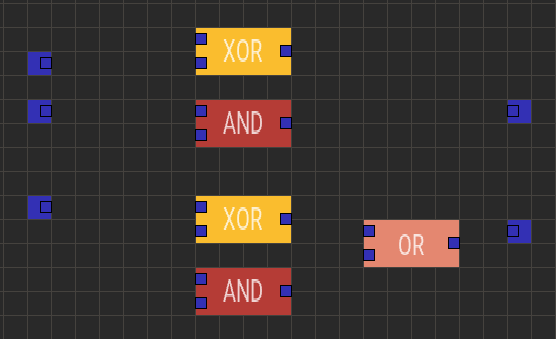
For demo purpose, we'll look at how one would go about creating a simple 1-bit adder circuit in this program.

1. By default, the component selected is *state*. To change this, simply click on the **Components** button on the menu which will open up a list of available components as shown if Figure 2. Click on the button respective to the component you wish to select.



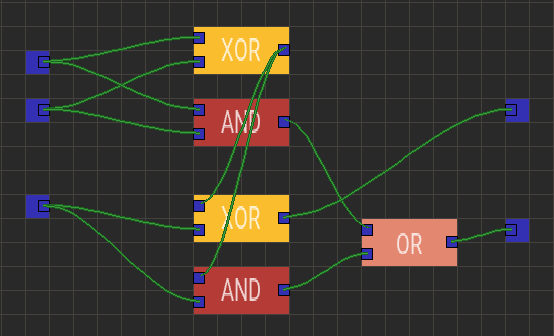
Figure

1. If the component can be placed at the position of the cursor, a slightly unsaturated version of the component will show up. To place the selected component on the grid, simply click at any available position.
2. By repeating steps 1 and 2, place all the components necessary for the full adder. The circuit will look like as shown in Figure 3.



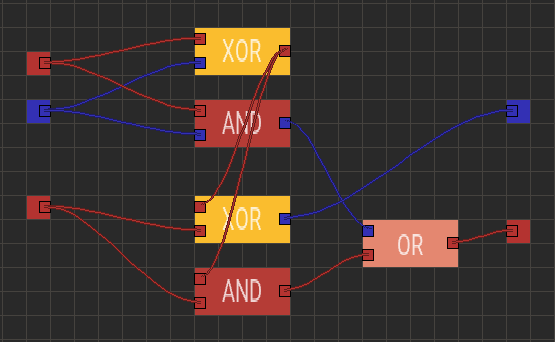
Figure

1. Once all the components are placed connect the appropriate terminals by simply dragging and dropping from one terminal to the other. A green colored wire will be drawn between the terminals that are connected. The final circuit with wires will look like as shown in Figure 4.



Figure

1. Once the circuit is complete, click on the **Run** button on the menu to start the simulation. The wires will change color based on the state of the terminals they are connected to.
2. To change the state of *state*, simply clicking on it will invert its current state. The circuit during simulation is shown in Figure 5.



Figure

Here, the **Red** color represents that the logic level is HIGH and **Blue** color represents logic state LOW.

Similar methods can be used to create different kinds of circuit, given that they fit inside the grid space available.

The **Undo** and **Redo** buttons perform same function as pressing **Ctrl+Z** and **Ctrl+R**.

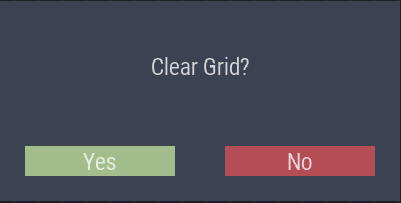
The **Snap to Grid: On (or Off)** indicated whether the component being placed will be aligned to the grid or not.

The **Delete Component** button will delete the selected component (indicated by a faint green border) from the grid.

The **Clear Grid** button will remove all the components from the grid, after a confirmation from the user (Figure 6).

The **File Menu** button, when pressed, will open up a small menu with some general options related to files.

* The **New** button starts a completely new project the user can work on.
* The **Open** button will display a windows dialog box from where the user can select a project file having extension **.mlg.**
* The **Save** and **Save As** button perform the typing functions of saving a file. The file must be saved either without any extension or with the extension **.mlg.**



Figure