Pulchowk Campus
Institute Of Engineering
Tribhuvan University

C Project Report on

MinimaLogic

A Digital Logic Simulator

Submitted To:

Department of Electronics and Computer Engineering

Submitted By:
Aditi Kharel – PUL077BEI008
Ashutosh Bohara – PUL077BEI012
Pallavi Paudel – PUL077BEI027
Rijan Shrestha – PUL077BEI034

1 INTRODUCTION

Logic circuits are one of the core components of CPUs. These circuits allow manipulating binary data and carrying out various logical operations. Programs such as Proteus and Logisim can be used to simulate logic circuits. For our project, we decided to create a similar (albeit heavily simplified) logic simulator named **MinimaLogic**.

MinimaLogic is a GUI based logic simulator that allows simulation of various logic gates and circuits. It allows users to create and simulate circuits ranging from a simple 1-bit adder to more complex circuits like 4-bit counters. In fact, one could create any circuit that fits within the available area using the components provided in the program. The program allows the user to interact via mouse and keyboard. To make the program user friendly, we tried to keep the controls as intuitive as possible.

This program heavily relies on the Simple DirectMedia Layer(SDL2) Library for GUI elements as well as user interation/input. Alongside SDL2, the program also utilizes SDL2_ttf for font rendering as well as the Windows API for opening/saving files. Other standard headers that are available with all modern development environments have also been used.

1.1 The C language

1.2 Simple DirectMedia Layer(SDL2)

1.3 Structure of Code

1.4 Building

2 METHODOLOGY

2.1 Planning and Gathering resources

In the planning phase of the project, we made a list of features that we wanted to implement in the program. User interaction (placing, deleting, simulating components, wiring) and drawing were of highest priority on our list. This process of planning persisted throughout the duration of the project as we kept on adding new ideas (undo, redo) and abandoning some old ones (zomming, panning).

As for gathering resources, we couldnt get our hands on any books/ebooks to refer to, so we instead stuck to online tutorials (GeeksforGeeks, tutorialspoint, GitHub gists) as well as the offical documentation of the libraries (libsdl, MSDN).

2.2 Developing

Development of the project consisted of implementing features by writing code and then compiling/building them. The code was written using Visual Studio Code and Visual Studio. Since the project uses SDL2 and SDL2_ttf libraries, we downloaded development libraries for both SDL2 and SDL2_ttf and setup our IDEs to work with them. During the development process, each team member picked worked on implementing different features individually. Since certain features overlapped or had to interact with each other (for eg: wiring and updating components), we also had to collaborate with each other so that the code would fit together nicely. We used git for version control and GitHub to share and collaborate on code.

For building, we wrote a batchfile that would automate the build process. The batchfile would build using GCC, Clang and MSVC of they any of these compilers is found. It also downloads and copies the necessary libraries by itself. Since the project relies on Windows API, it cannot be built on unix or linux systems.

2.3 Testing & Debugging

Testing the project was done manually. We often asked each other to test the features that we implemented. We also asked some friends to help test the program and obtain some feedback. Although we were unable to perform any automated tests, this method of manually testing features proved to be quite effective as we were able to quickly find and fix many bugs in the program.

Through our testing process we were able to find lots of bugs and bottle necks in our program such as memory leaks, high gpu usage etc. We utilized various debugging and profiling tools provided by Visual Studio Code and Visual Studio to locate and fix these issues. These tools (such as call stack, watch values etc) allowed us to pinpoint errors and correct the quickly.

3 IMPLEMENTATION

The source code for this project can be accessed through the GitHub repository of the project:
https://github.com/nantoka/kantoka

3.1 Flowchart

Result

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

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: Revert to the last change made on the grid (Undo).
- Ctrl+R: Redo the last change made by Undo.
- 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.

Some important snapshots from the program are shown below:

Experience and Insights Gained

When we started working on this project, we were sure it would be an interesting experience, but what we did not anticipate was the sheer number of challenges that would come along. Challenges that proved the knowledge gained from our course to be insufficient for a project of this level. Algorithms that were unfamiliar to us had to be used for various parts of the program. But, along the way, we learned that these challenges were part of the learning experience. As futile as it may be, we've tried to make a list of what we learned by doing this project:

1. Organizing the project and making the source code readable by using multiple files and using a consistent naming convention.
2. Collaborative work with VCS like Git and remote hosting services like GitHub.
3. Using the official documentation of Libraries and the C language itself.
4. Unique algorithms for things like drawing curves, collision detection, etc.

5. Avoiding memory leaks and other vulnerabilities that are exposed in low level language such as C.