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

Acknowledgements

First of all, we would like to express our gratitude to our teacher, Anku Jaiswal, for her constant support and guidance throughout the semester. The lectures, assignments and guidelines have been of great help for us to successfully complete our project.

We would also like to thank IOE, Pulchowk campus and Department of Computer and Electronics Engineering for providing us an opportunity to work on this project. We have enhanced our programming skills in C language, and learnt to work as a team in the past few weeks. Our friends and seniors have also been incredibly helpful in figuring out this whole "teamwork for a project" scenario by sharing their knowledge and critique, this helped in shaping the outline and rectifying the elements of our project. We feel immensely thankful to be surrounded by such supportive, kind and obliging people. Our families have also played a huge role by providing us the support we needed-both emotional and mental-without which finishing the project would have been impossible. We would like to thank everyone who has directly or indirectly contributed to the project.

Table of Contents

Chapter 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 of x64 Windows systems 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 can 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.

The complete source code for the project can be found here:
https://github.com/First-Sem-C-Project/Minimalogic

1.1 The C Programming Language

C is a general-purpose, procedural computer programming language which was developed by Dennis Ritchie in Bell Laboratories between 1972 and 1973 AD as a successor to Basic Combined Programming Language (BCPL or the B language). C was designed such that code written in C can be translated efficiently into machine level instructions. Code

written in C somewhat resembles the english language as it uses keywords such as `if, else, for, do`, etc. Along with its speed and syntax, C contains addidtional features which allow it to work at lower level thus it can bridge the gap between machine level and high level languages. Due to this, C has found lasting use in systems programming (e.g writing operating systems). It can also be used for applications programming. Applications made in C are generally much faster and efficient than most other programming languages even with less optimization.

Some characteristics of the C language:

- The language has a small, fixed number of keywords (only 32), including a full set of control flow primitives: if/else, for, do/while, while, and switch.
- It has a rich set of operators (arithmetic, bitwise, relational, logical and some miscellaneous operators).
- It allows users to define functions that return value of certain data type however, value returned by function can be ignored if not needed.
- It also allows for procedures i.e. functions not returning values, by using a return type `void`.
- It is a statically typed language i.e. all data has a type, but implicit conversions are possible.
- Declaration syntax mimics usage context. For eg: C has no "define" keyword; instead, a statement beginning with the name of a type is taken as a declaration.
- It allows for user-defined (`typedef`) and compound data types through `struct`(structure, heterogeneous aggregate data type), `union`(structure with overlapping members), `enum`(enumerated data type) and arrays(homogeneous aggregate of data).
- It allows low-level access to computer memory by converting machine addresses to typed pointers.
- A preprocessor performs macro definition, source code file inclusion, and conditional compilation.
- There is a basic form of modularity: files can be compiled separately and linked together, with control over which functions and data objects are visible to other files via `static` and `extern` attributes.
- The standard library of C provides a rich set of functions which allow for complex functionality such as I/O, string manipulation, and mathematical functions.

1.2 Structure of Code

Generally, a C program is divided into following sections:

- Inclusion / Linking Section
  This section consists of header files to be included in the program. Inclusion of files is done using the `#include` preprocessor directive. It provides instructions to the compiler to link functions, structures, enums etc. from the header file.

- Macro Definition Section
  This sections consists of macro (or symbolic constants) definitions. Macros are defined using the `#define` preprocessor directive.

- Function Definition Section
  This section consists of definitions of all user defined funtions that are to be used in the program to perform. These functions are called from the `main()` function.

- Main Function Section
  There can only be one `main()` function in the entire C program. The `main()` function acts as the entry point to the program i.e. execution of program begins from the main function.

1.2.1 Sample C Program

```c
// Linking section
#include <stdio.h>
// Here, the C standard library stdio.h (standard input output) has been
    included

//Macro definition section
#define MSG "Hello, Peter\n"
//Here, a macro MSG has been defined which expands to "Hello, Peter\n"

//Function definition section
void PrintMessage(){
    printf(MSG);
}
//Here, a function PrintMessage() has been defined
//It calls the printf() function from stdio.h to display value of MSG to the
    screen

//Main Function Section
```

```
17 int main(){
18     PrintMessage();
19     return 0;
20 }
21 //Here, the main function has been defined which calls the PrintMessage()
       function defined previously.
22
23 // Output
24 // Hello, Peter
```

The source code for our program has been divided into 9 files (4 headers and 5 C files). Our source code mostly follows this format however at certain parts of the code the format has not been followed for the sake of convinience.

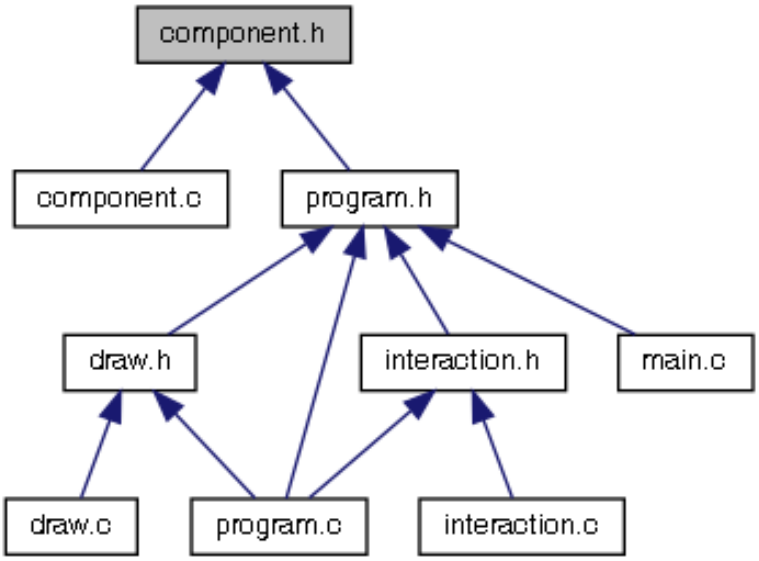

Figure 1.1: File inclusion graph

The files are linked as shown in figure above. Functions, structs and enums shared across multiple files have been defined in the relevant header file.

1.3 Simple DirectMedia Layer(SDL2)

Simple DirectMedia Layer is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D. It is used by video playback software, emulators, and popular games including Valve's award winning catalog and many Humble Bundle games.

SDL2 libraries also contains extension to keep SDL as light as possible. Some of the libraries are SDL2_image, SDL2_net, SDL2_mixer, SDL2_ttf, true type SDL2_rtf.

SDL2_image is used to load different images, SDL2_net is used for cross platform networking, SDL2_mixer is an audio mixer library that supports WAV, MP3, MIDI and OGG. SDL2_ttf is used to write using fonts in the program and SDL2_rtf is Rich Text Format library. Out of these libraries, we only used SDL2_ttf.

The SDL2 library is more oriented towards game development and it provides a vast quantity of features such as 3D rendering, hardware accelerated 2D graphics, multiple windows, multiple audio devices, multiple input devices etc. Since our program is much simpler than a full-fledged game, we were only able to utilize a small fraction of SDL's features.

SDL programs run continuously in a loop which is often referred to as the game loop. The loop can be summarized by the following diagram:

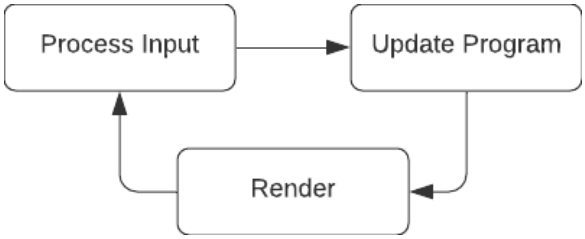

Figure 1.2: SDL program loop

1.3.1 Sample SDL Program

```
1  // This programs opens a SDL window which shows a blank screen until the user
       quits
2  #include <SDL2/SDL.h>
3  #include <stdio.h>
4
5  // Declare SDL window and renderer
6  SDL_Window *window;
7  SDL_Renderer *renderer;
8
9  int main(int argc, char ** argv) {
10     // Display error and quit program if SDL cannot be initialized
11     if (SDL_Init(SDL_INIT_EVERYTHING) != 0) {
12         printf("Failed to initialize SDL\n");
13         return -1;
14     }
15     // Create a window
16     window = SDL_CreateWindow("Demo", 0, 0, 500, 500, SDL_WINDOW_RESIZABLE);
```

```
    // Display error and quit program if window cannot be created
    if (!window) {
        printf("Could not create a window: %s", SDL_GetError());
        return -1;
    }
    // Create a renderer
    renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_SOFTWARE);
    // Display error and quit program if renderer cannot be created
    if (!renderer) {
        printf("Could not create a renderer: %s", SDL_GetError());
        return -1;
    }
    // Main program loop
    while (true) {
        // Get the next event
        SDL_Event event;
        if (SDL_WaitEventTimeout(&event, 10)) // Wait for event (user input) {
            if (event.type == SDL_QUIT) {
                // Exit loop if user presses quit
                break;
            }
                /* User Input Processing happens here */
        }
        // Clear the screen
        SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255);
        SDL_RenderClear(renderer);

        /* Rendering/drawing code goes here */
        /* Updating code goes here */

        SDL_RenderPresent(renderer); // Update the screen
        SDL_Delay(10); // Limit frame rate to reduce CPU load
    }
    // Clean up
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
    return 0;
}
```

Chapter 2

METHODOLOGY

2.1 Flowchart

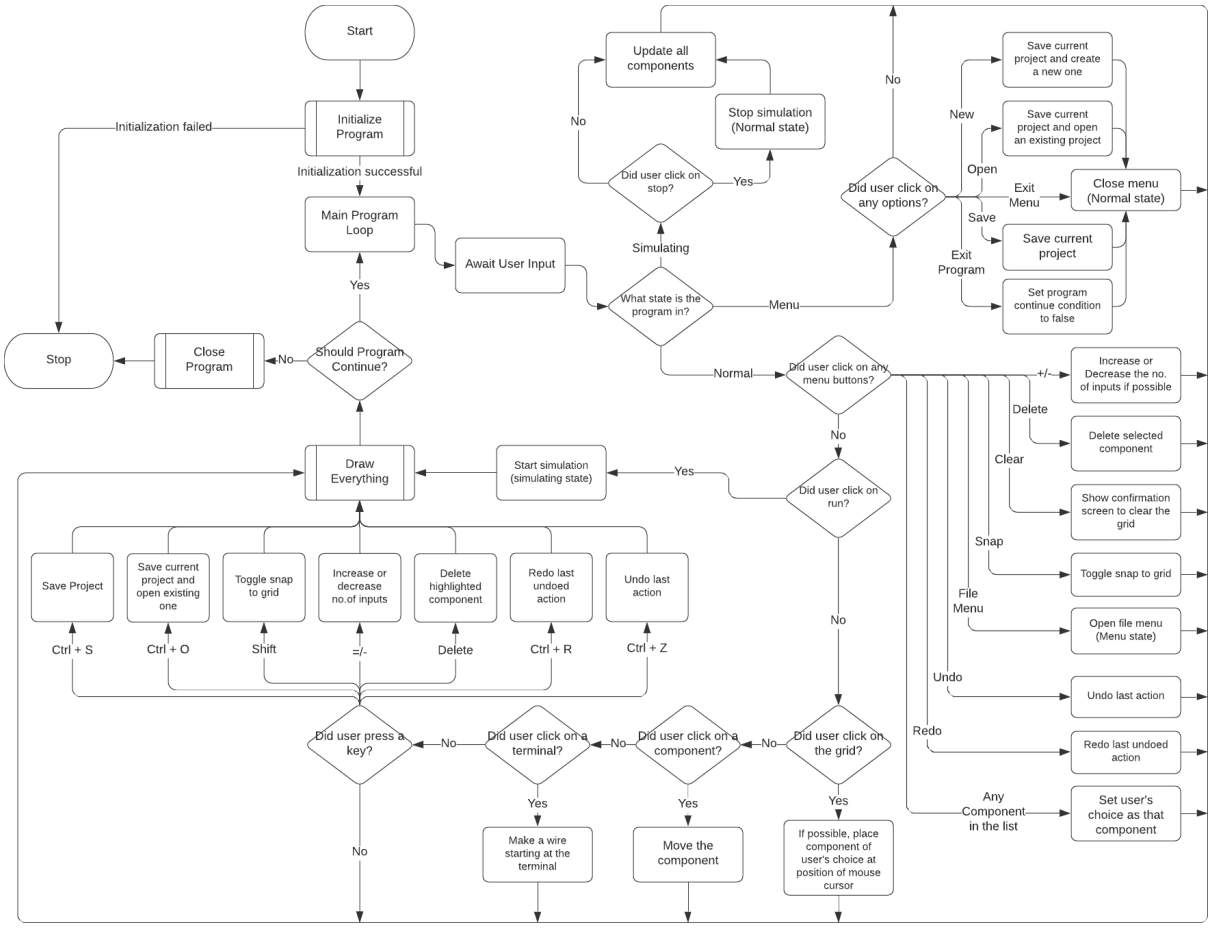

Figure 2.1: Flowchart of the program

2.2 Working of the program

Working of the program can be described in the following three steps:

2.2.1 Initialize

When the program is launched, initialization is carried out. If initialization is unsucessful, the program will display and error and quit. During this process, the libraries used by the program (SDL2 and SDL2_ttf) are initialized. Then the SDL window and renderer are created. Fonts used to display text in the program are loaded and then a character map is created using the fonts. Since SDL2_ttf ant the fonts are no longer required after the character map has been created, the fonts get closed and SDL2_ttf is quit.

Algorithm for Inititalizing

```
initialize SDL
did SDL initialize successfully?
  yes: continue
  no : display error
     exit program
create SDL window
create SDL renderer
were the window and renderer created successfully?
  yes: continue
  no: display error
    exit program
initialize the grid
  fill grid with -1
had user clicked on a file to open?
  read from file and update grid
  update window title
initialize menu
  set dimensions and positions for all buttons
change directory to directory of executable
initialize SDL_ttf
did SDL_ttf initialize successfully?
  yes: continue
  no : display error
       exit program
load fonts
were fonts loaded successfully?
```

```
  yes: continue
  no : display error
       exit program
create character map
  create textures for all characters to be used
close fonts
close SDL_ttf
```

2.2.2 Loop

All user interaction, simulation and rendering happen inside the main program loop. During each iteration of the loop, the program waits for user input and then processes it as shown in the flowchart. Then all components of the program are rendered (drawn) onto the screen. If the simulation is running then all the components get updated. The loop continues to run until the user quits (presses the X button in title bar or presses Alt+F4) or exits program from the menu.

Algorithm for Loop

```
loop if program continue condition is true{
  await user input
  what state is program in?
    normal:
      if user clicked on RUN
        start simulation
      if user clicked on + or user pressed = key
        increase no. of inputs of current choice if possible
      if user clicked on - or user pressed - key
        decrease no. of inputs of current choice if possible
      if user clicked on Undo or user pressed Ctrl + z
        undo last action
      if user clicked on Redo or user pressed Ctrl + r
        redo last undoed action
      if user clicked on Delete Component or user pressed Delete key
        delete selected component on the grid
      if user clicked on Clear Grid
        show confirmation screen
          user clicked yes: empty the grid
      if user clicked on Snap to Grid or user held Shift
        toggle snapping to grid
```

```
      is snapping toggled on?
        yes: set snap button text to "Snap to Grid: On"
        no : set snap button text to "Snap to Grid: Off"
    if user clicked on File Menu
      open the menu
    if user clicked on any component in the list
      set user's choice to be that component
    if user pressed Ctrl + o
      save current project and open existing one
    if user pressed Ctrl + s
      save project
    if user clicked on component on the grid
      move the component
    if user clicked on terminal
      make wire starting at that terminal
    if user clicked on the grid
      if possible place user's choice of component on the grid
      at current mouse cursor position
  simulating:
    if user clicked on STOP
      stop simulation
    update all components
  menu:
    if user clicked on Save:
      save current project
      close menu
    if user clicked on New:
      save current project and create a new one
      close menu
    if user clicked on Open:
      save current project and open an existing one
      close menu
    if user clicked on Exit Menu:
      close menu
    if user clicked on Exit Program:
      set program continue condition to false
      close menu
draw everything
  draw menu
```

```
    draw grid
    draw components
    draw wires
}
```

2.2.3 Close

After the program exits from the loop, it calls some clean up functions which destroy the textures, window and renderer, and free memory.

Algorithm for Closing

```
destroy all textures
destroy window
destroy renderer
quit SDL
exit program
```

2.3 Brief Descriptions of the source files

2.3.1 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.

2.3.2 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.

2.3.3 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 coordination is handled in the file program.c.

2.3.4 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 centerpiece of the program that coordinates 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 into 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 occurs when the program is launched. 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. During each loop, the program checks for events, performs necessary operations based on them, updates the elements on the screen if required, and redraws all of those elements.

2.3.5 main.c

As mentioned earlier, this file is kept as clean as possible by defining the main loop in program.c. The main function calls functions for initialization, the main program loop, and finally closing the program.

Chapter 3

RESULT

Figure 3.1: Initial / Starting Screen

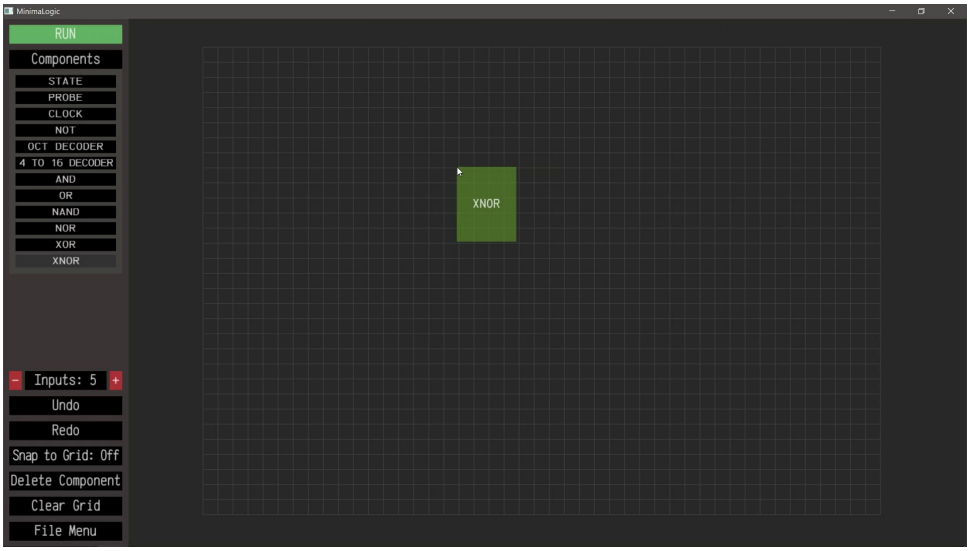

Figure 3.2: Component preview before placing

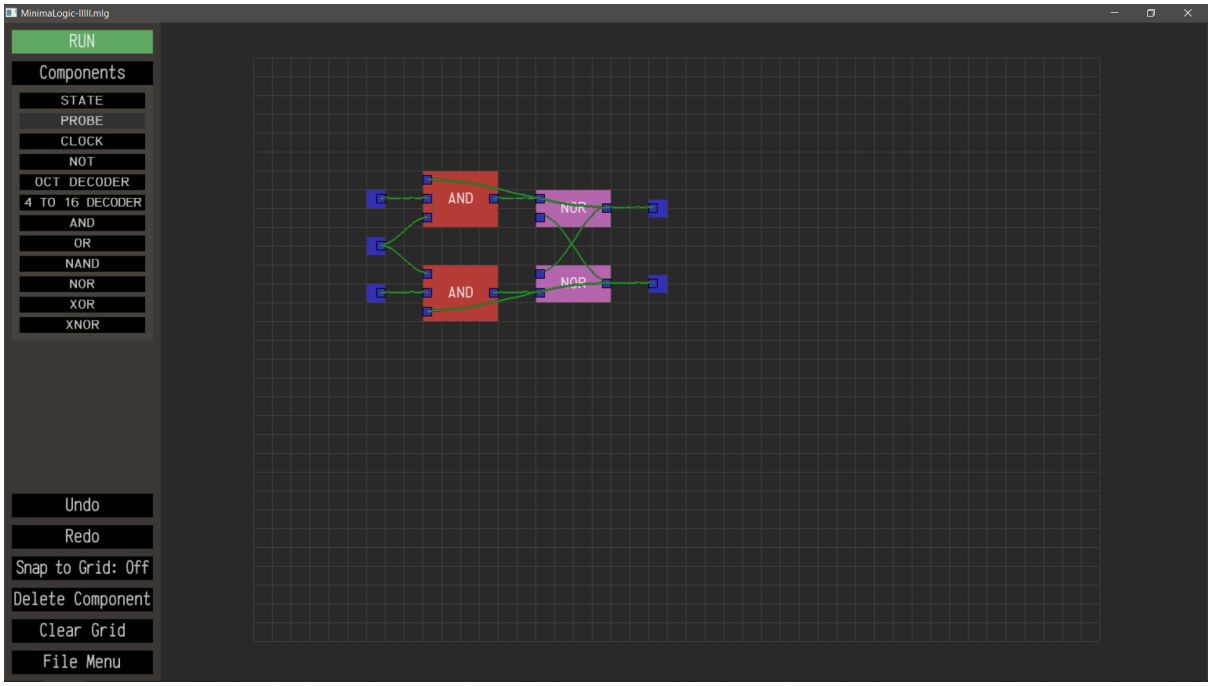

Figure 3.3: Circuit in normal state (not simulated)

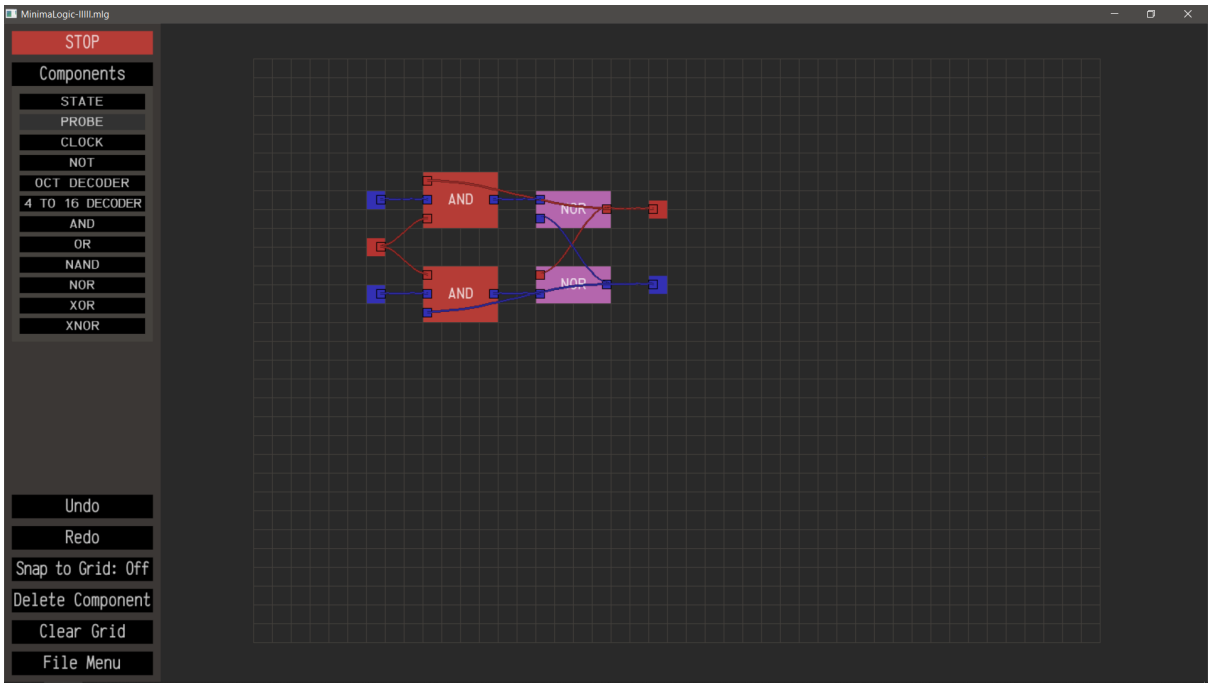

Figure 3.4: Circuit being simulated

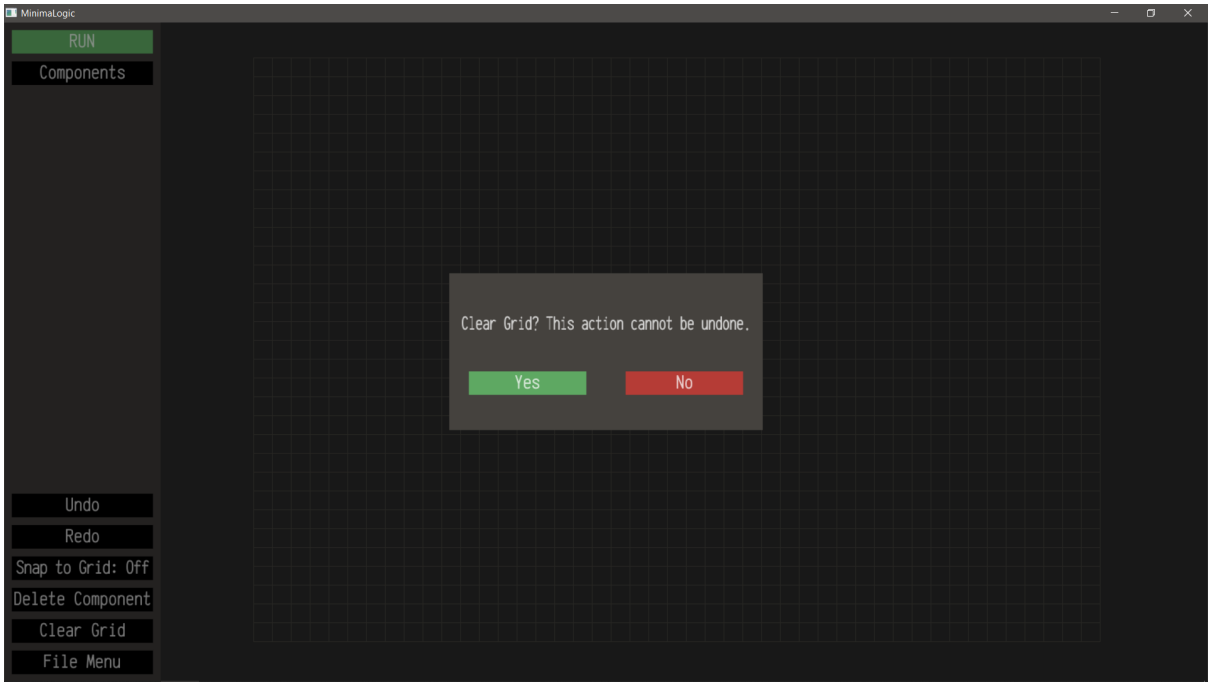

Figure 3.5: Confirmation screen before clearing

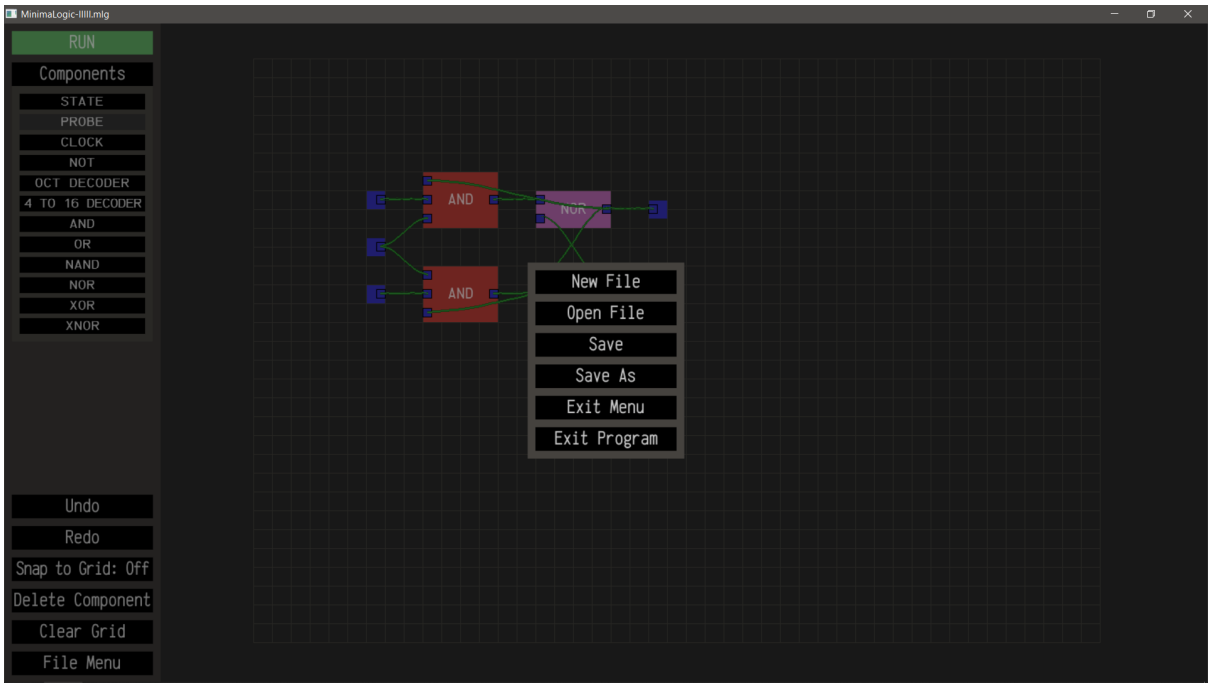

Figure 3.6: File Menu

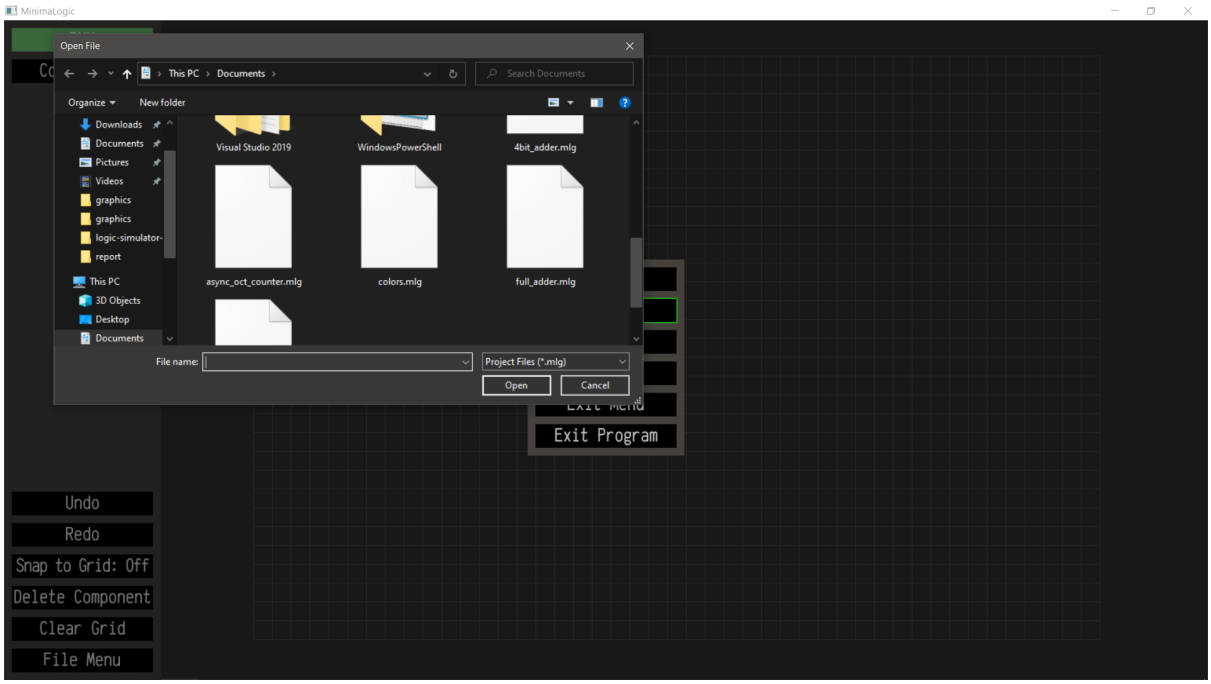

Figure 3.7: Opening a file

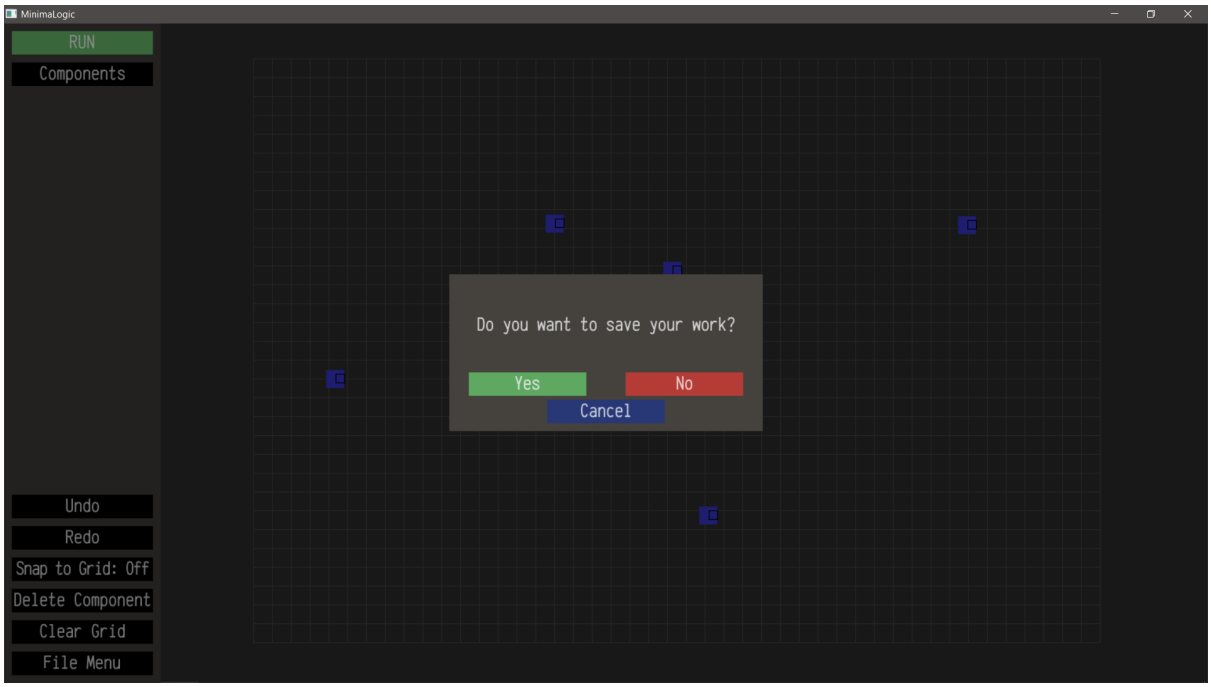

Figure 3.8: Asking to save before exiting program or opening another file

Chapter 4

CONCLUSION

4.1 Experience

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. All in all, this was a very helpful project that introduced us to different paradigms of programming. 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.
6. Making sure the codes were methodical, operational and compatible.

4.2 Possible Improvements

While this program is stable as far as we have tested it, it is nowhere near perfect. Many features were sratched off of the to-to list, some due to the lack of time and others because they were too complicated. Here is a list of improvements that can be made to this program.

1. Better graphics for components and wires. The wires look jagged right now which can be fixed with some anti-aliasing techniques.

2. Selection of multiple components on the grid and the functionality to copy paste the selection.

3. Zooming and panning to provide a larger canvas.

4. More components. Currentlty, there are very few, basic components which makes drawing more complex circuits difficult.

5. Letting users create custom components.