



**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

Acknowledgements	i
1 INTRODUCTION	1
1.1 The C Programming Language	1
1.2 Structure of Code	3
1.2.1 Sample C Program	3
1.3 Simple DirectMedia Layer(SDL2)	4
1.3.1 Sample SDL Program	5
2 METHODOLOGY	7
2.1 Flowchart	7
2.2 Working of the program	8
2.2.1 Initialize	8
2.2.2 Loop	9
2.2.3 Close	11
3 RESULT	12
4 CONCLUSION	16

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 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 interaction/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 additional 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

```
1 // Linking section
2 #include <stdio.h>
3 // Here, the C standard library stdio.h (standard input output) has been
  included
4
5 //Macro definition section
6 #define MSG "Hello, Peter\n"
7 //Here, a macro MSG has been defined which expands to "Hello, Peter\n"
8
9 //Function definition section
10 void PrintMessage(){
11     printf(MSG);
12 }
13 //Here, a function PrintMessage() has been defined
14 //It calls the printf() function from stdio.h to display value of MSG to the
  screen
15
16 //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 convenience.

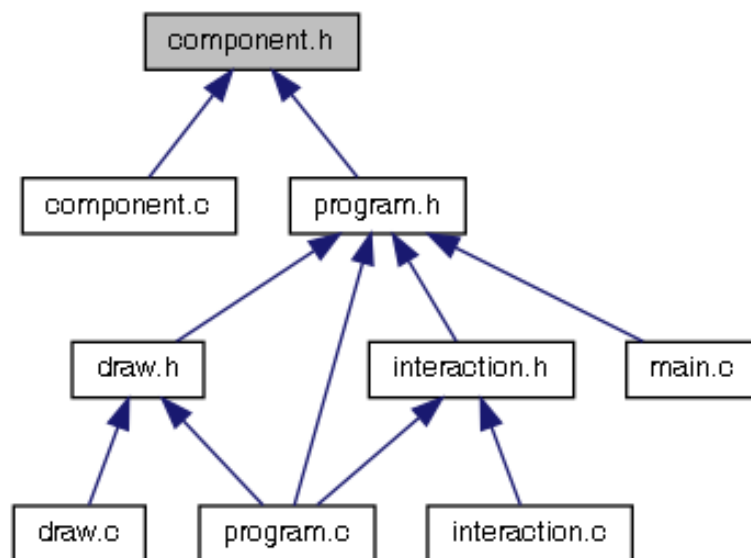


Figure 1.1: File inclusion graph

The files are linked as shown in figure above. Other header files have also been linked which is discussed in the implementation section. 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:

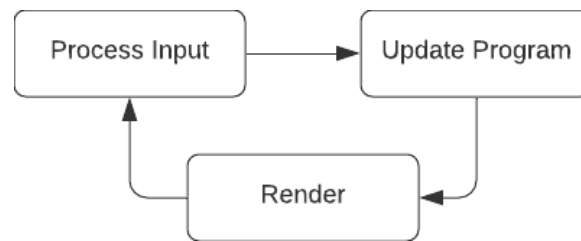


Figure 1.2: SDL program loop

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);
17 // Display error and quit program if window cannot be created
18 if (!window) {
19     printf("Could not create a window: %s", SDL_GetError());
20     return -1;
21 }
22 // Create a renderer
23 renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_SOFTWARE);
24 // Display error and quit program if renderer cannot be created
25 if (!renderer) {
26     printf("Could not create a renderer: %s", SDL_GetError());
27     return -1;
28 }
29 // Main program loop
30 while (true) {
31     // Get the next event
32     SDL_Event event;
33     if (SDL_WaitEventTimeout(&event, 10)) // Wait for event (user input) {
34         if (event.type == SDL_QUIT) {
35             // Exit loop if user presses quit
36             break;
37         }
38         /* User Input Processing happens here */
39     }
40     // Clear the screen
41     SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255);
42     SDL_RenderClear(renderer);
43
44     /* Rendering/drawing code goes here */
45     /* Updating code goes here */
46
47     SDL_RenderPresent(renderer); // Update the screen
48     SDL_Delay(10); // Limit frame rate to reduce CPU load
49 }
50 // Clean up
51 SDL_DestroyRenderer(renderer);
52 SDL_DestroyWindow(window);
53 SDL_Quit();
54 return 0;
55 }

```

Chapter 2

METHODOLOGY

2.1 Flowchart

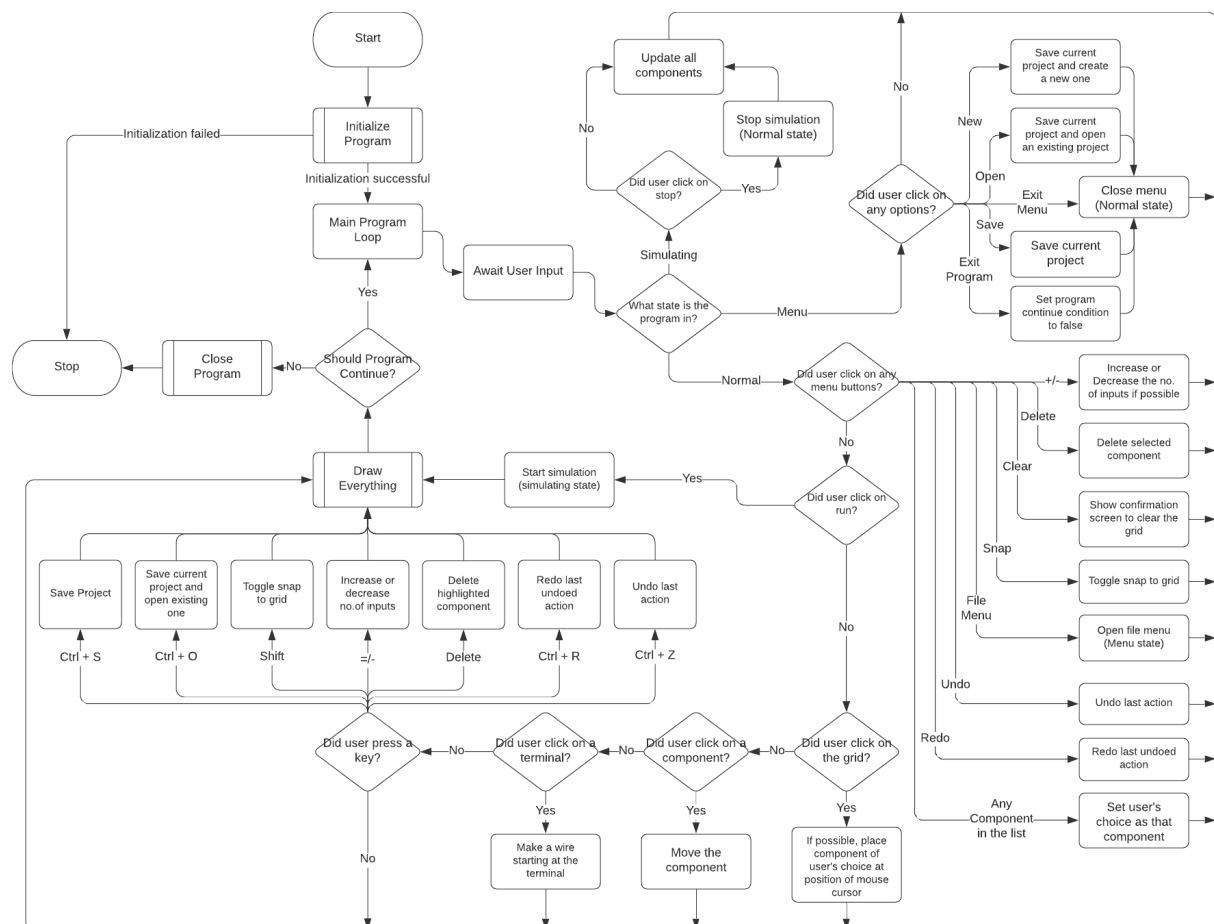


Figure 2.1: Flowchart of the program

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 unsuccessful, the program will display an 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 and the fonts are no longer required after the character map has been created, the fonts get closed and SDL2_ttf is quit.

Algorithm for Initializing

```
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
initialize menu
    set dimensions and positions for all buttons
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 undone 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 function which destroy the textures, window and renderer and frees memory.

Algorithm for Closing

destroy all textures

destroy window

destroy renderer

quit SDL

exit program

Chapter 3

RESULT

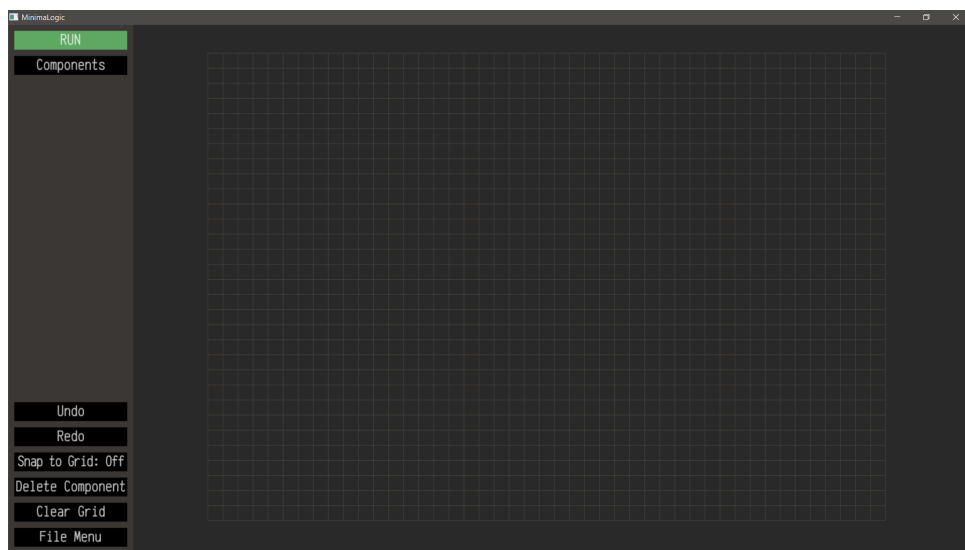


Figure 3.1: Initial / Starting Screen

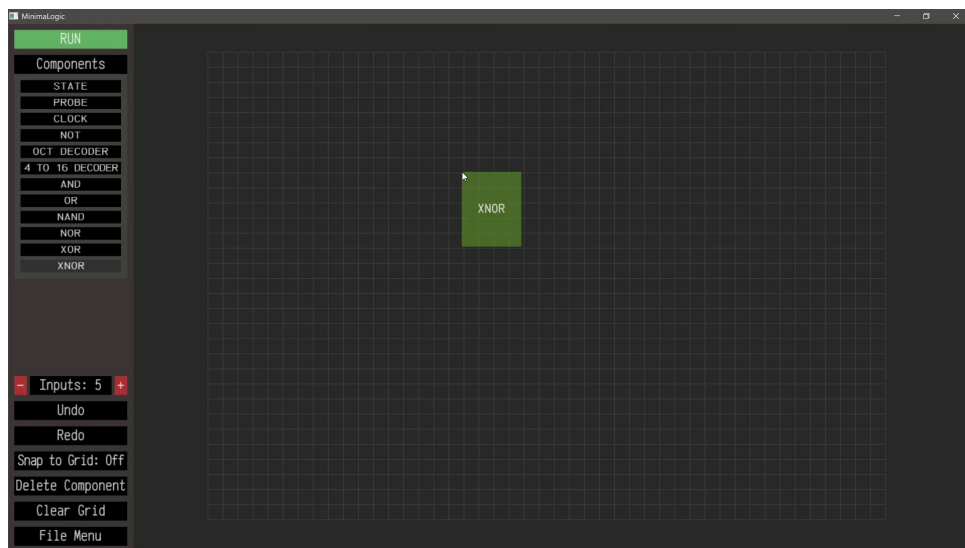


Figure 3.2: Component preview before placing

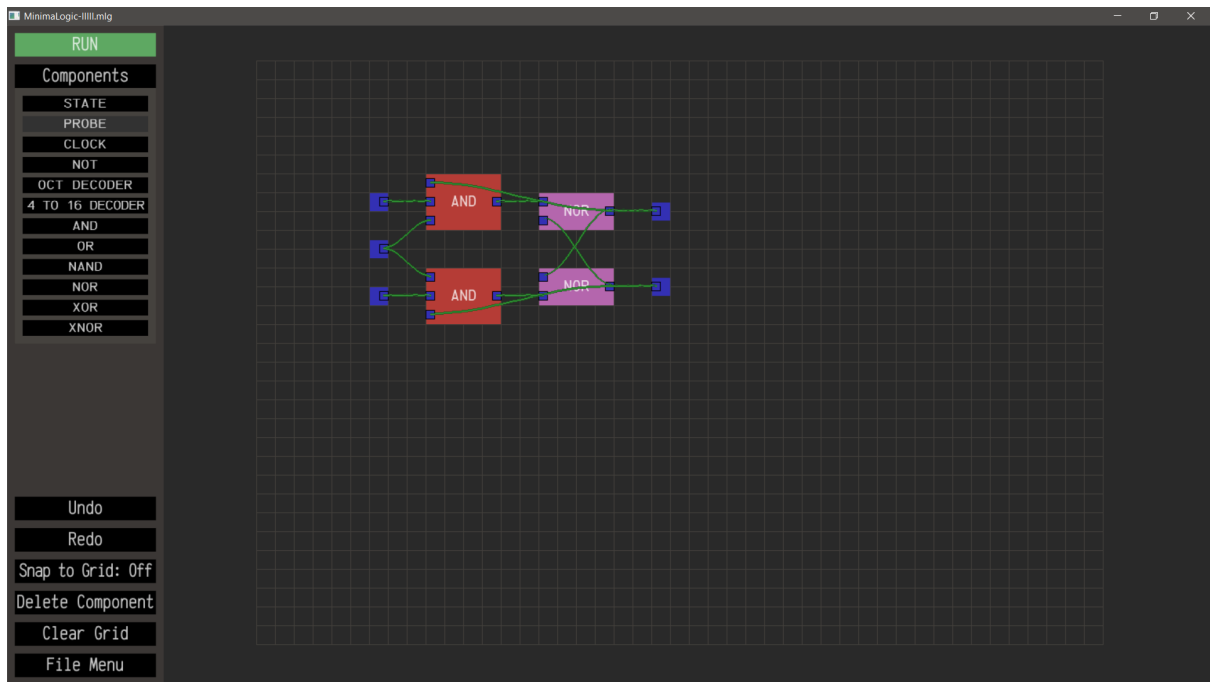


Figure 3.3: Circuit in normal state (not simulated)

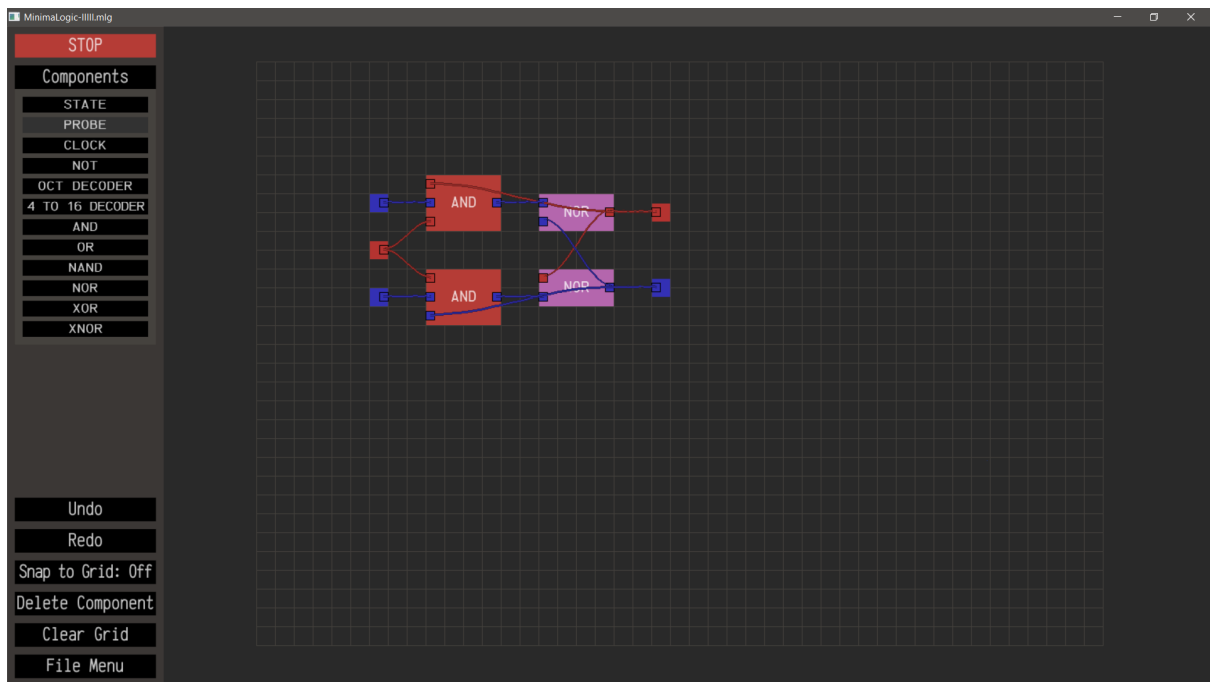


Figure 3.4: Circuit being simulated

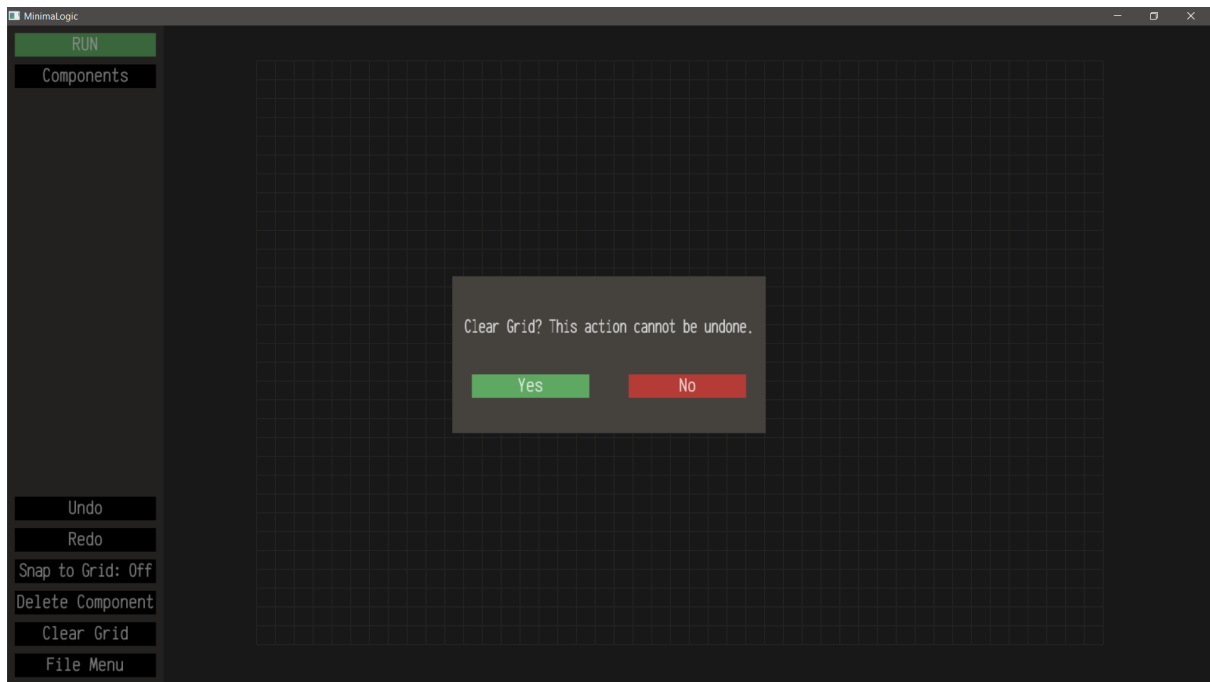


Figure 3.5: Confirmation screen before clearing

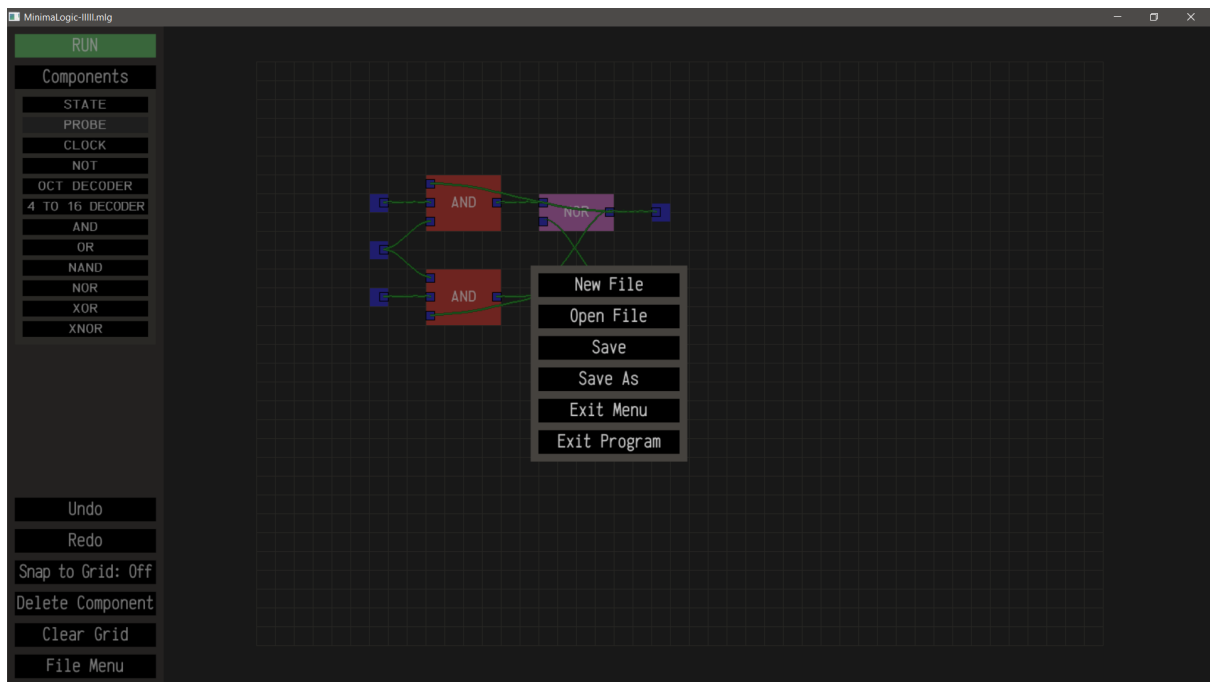


Figure 3.6: File Menu

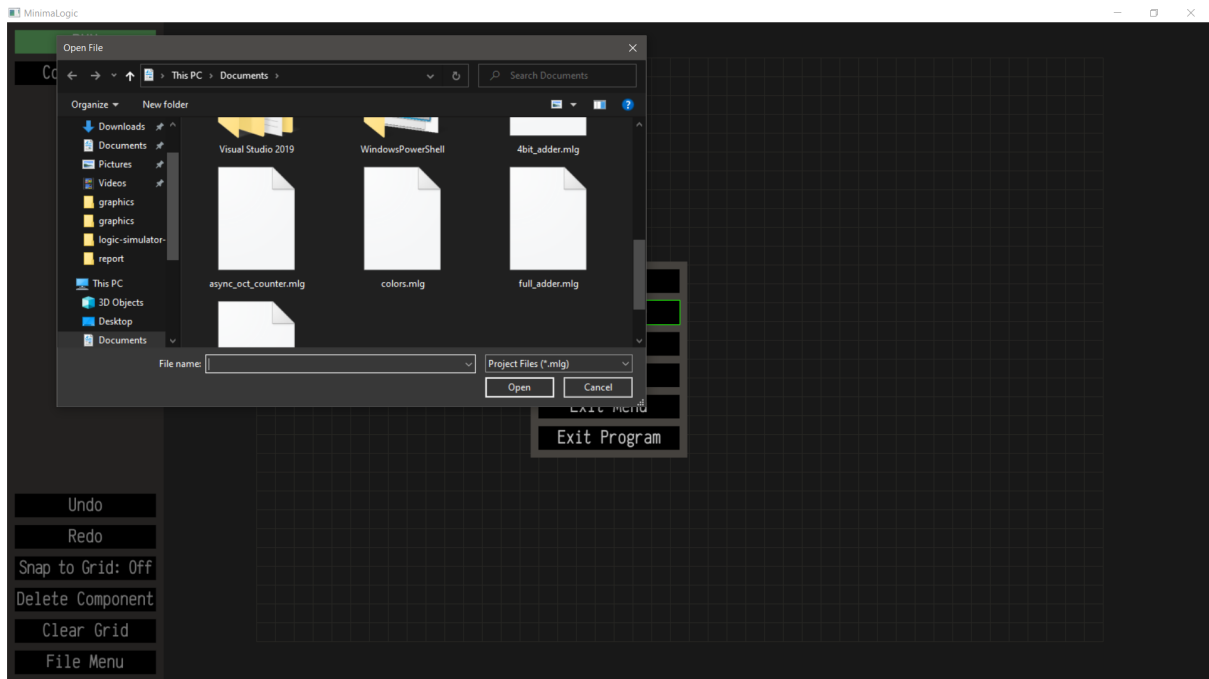


Figure 3.7: Opening a file

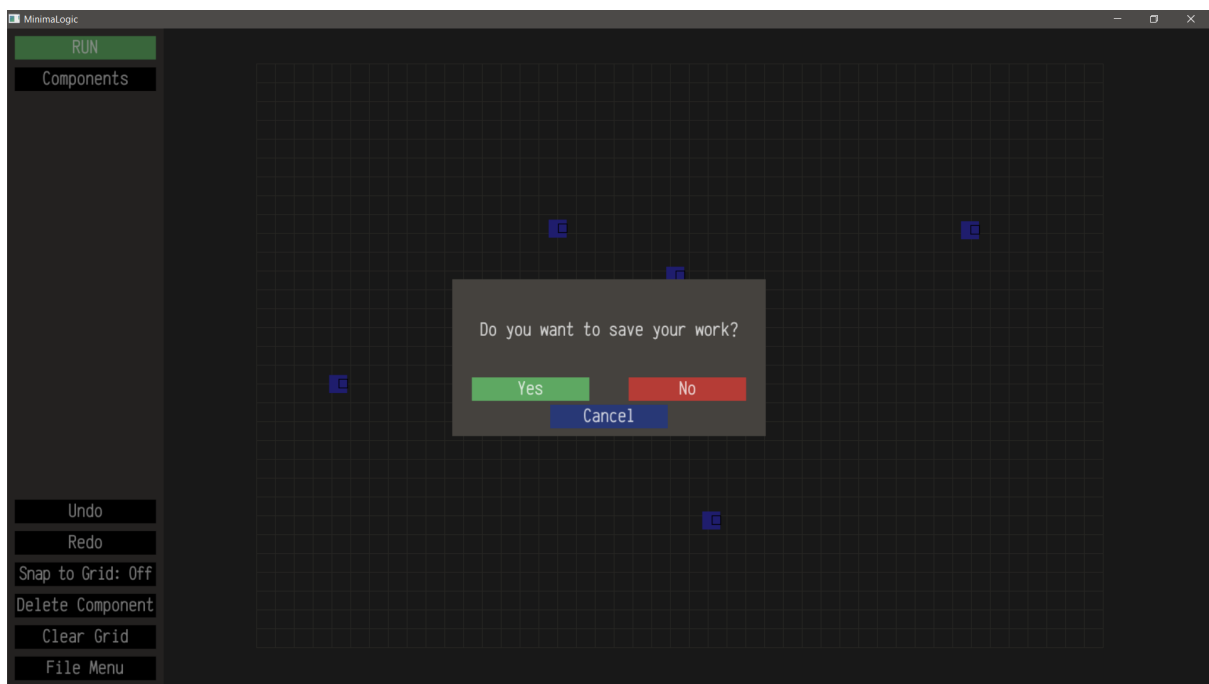


Figure 3.8: Asking to save before exiting program or opening another file

Chapter 4

CONCLUSION

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.
6. Making sure the codes were methodical, operational and compatible.