OOP Project Report
Prof. DSB

Mantik

Tribhuvan University
Institute Of Engineering
Pulchowk Campus
Department of Electronics And Computer Engineering

Submitted By:

Srijan Poudel (076BCT081)
Sulav Deo (076BCT086)
Sumit Yadav (076BCT088)

Acknowledgment

We would like to express our special thanks of gratitude to our teachers Daya Sagar Baral, Lok Nath Regmi and Shanti Tammang who gave us the guidance and assistance to do our project **Mantik**, which also encouraged us in doing a lot of Research and we came to know about so many new things we are really thankful to them.

Secondly I would also like to thank the Open Source Projects SDL and Dear ImGui (along with ImNodes) who served as a base for our project. Using those libraries we learned a few more programming paradigms than what core c++ allows. Namely Event handling and Immediate Mode Gui. Both of which are discussed later in this document.

- **SDL2** - https://www.libsdl.org/
- **Dear ImGui** - https://github.com/ocornut/imgui
    - **ImNodes** - https://github.com/Nelarius/imnodes

Abstract

Mantik (turk: logic) is a digital logic simulation program designed for extensibility, i.e. by making a few minor changes one can add new circuit elements to the program. One can visualize and simulate any combinational circuit. Even if some circuit components are missing, one can easily add them. I would like to mention this program is specifically designed for simulation of combinational logic. Not for simulating power circuits. That is you cannot use this program to measure circuit currents.

Table of Contents

Objectives

We were aiming for the ability to simulate Digital Combinational circuits. Hence we set out with the following major objectives:

- Add multiple circuit components.
- Add links between them.
- Set input state or carry via links.
- Intuitive User Interface.

Introduction

Digital logic is a foundational course/concept in Digital electronics used to implement combinational calculations as electronic circuits. For rapid prototyping of digital circuits it is wise to use software tools as it is faster than making circuits physically and they allow us to save design from multiple iterations of the prototype. Also it is a lot cheaper to use a software prototyping platform for a multitude of reasons. Hence a software digital prototyping solution makes a lot of sense. For these reasons we settled on making such a program for our 3rd semester project.

As the course demanded, C++ and OOP concepts were going to be implemented in the software. Those will be discussed later in the report. For the interface, this program uses Dear ImGui. This choice was mostly due to the fact that the GUI of this project would be shaped to look unlike most traditional software, i.e. Menu Bar, Toolbar, Tab Bar, document pane etc. Looking around on the internet, Dear ImGui seemed to have a good reputation for being flexible. ImNodes which seemed to fit requirements of this project was also chosen for the core UI of this project. SDL was also used to complement ImGui in the areas it lacked. The choice of SDL was fairly straightforward, we have had previous experience with it.

Application

This program has a wide range of uses for engineering students. While this is not flexible as other simulators out there, this is very effective in implementing short combinational circuits instantly. The GUI albeit, less than competent to truly stand with some of the marketed applications, has enough usability and convenience going for it.

Literature survey

This project is based on the concept of Object Oriented Programming. Accordingly, various books (like The Secrets of Object Oriented Programming in C++, Programming -- Principles and Practice Using C++) exploring the concepts of Object Oriented paradigm were taken as a reference. Apart from these, various online resources (like https://en.cppreference.com/w/, https://stackoverflow.com/, https://lazyfoo.net/ etc.) and examples of ImGui and ImNodes were studied to learn to use those libraries.

Existing System

We are familiar with Proteus and Logisim. They were used as an inspiration for our program.

On the GUI side we saw the box2d demo program and SHADERed IDE. They assured us that Dear ImGui was a perfectly good and flexible library for our use case.

Immediate Mode GUI design

Traditionally GUIs are made in what's called a retained mode paradigm. In this technique the window elements are redrawn only when there is any change that will affect the look/state of the GUI. In 2005 Casey muratori, better known as handmadehero, coined the term Immediate Mode GUI. In this approach you have full control of the loop of your program. As of years IMGUI has been confused with something that it is not. A common misconception goes that the immediate stands for all changes must be acted upon as soon as they happen. But this is rhyme at best. As per my understanding the Immediate Mode GUI community agrees that IMGUI has at least the following benefits over traditional GUIs. The API tries to minimize the application having to retain data related to the UI system and vice versa.

There is a particular reason why IMGUI seems to be popular with games. Because the rendering pipeline can be made very flexible, the UI and game loop can be synchronized very easily. For this reason this very library Dear ImGui has been used by popular developers including but not limited to adobe, ubisoft, mojang, nintendo and more.

Methodology

The project aims to provide a working GUI to perform simulation of combinational circuits while adhering to the object oriented paradigm. Hence, classes representing meaningful entities are created. Since this project uses ImNodes library, a lot of its features were encapsulated in classes to abstract away the complexity in the long run. The nodes provided by ImNodes are used as a container interface for every combinational circuit and gates.

Node

A generic class *node* is used as an interface with protected access specified members: *input_pins*(a vector of *in_pin* structure), *output_pins*(a vector of *out_pin* structure) and *m_inputs*(a vector of pointer to *node* object) and private access specified members: *m_unique_id* and *node_name*. It has a constructor to initialize private members to give every node a name to display and an unique identifier. A pure virtual function *calculate()* is declared as a template for every derived class to override. This function provides a block of codes to implement relevant calculation for combinational blocks.

Pins

in_pin is a structure of *pin_id*, *pin_name*, *value*(int data to hold 0 for low state, 1 for high state and -1 for invalid/incomplete state) and *counter*(initialized to 0). This completely describes the input pin of any node with their name and unique id. *counter* keeps track of the link from one output pin to that input pin so that a single output is fed to any input. *out_pin* is a structure of *pin_id*, *pin_name*, *value* and *connected_inputs*(a vector of pointer to *in_pin*). *connected_inputs* keeps track of all the inputs connected to any output pin to copy the values from output pin to all connected input pins.

All of the relevant classes and structure related to nodes(discussed above) are stored in **node.h.**

Links

link class describes any link formed between input pins and output pins with members *m_unique_id*, *start_node*, *start_pin*, *end_node* and *end_pin*. Operator '=' is overloaded to copy one link to another. This is completely declared and defined in **link.h** and **link.cpp**.

Node Types

node_types.h defines enumeration containing all the combinational blocks (like Not Gate, AND Gate, different Multiplexer, Demultiplexers, etc). NOT_ADDING is a default state.

Everything below GENERICS is general types of nodes which are specialized. INPUT_1, INPUT_2, etc. are specialized under INPUT.

```
enum node_types
{
   NOT_ADDING = 0,
   INPUT_1,
   INPUT_2,
   INPUT_3,
   INPUT_4,
   INPUT_5,
   INPUT_6,
   INPUT_7,
   INPUT_8,

   OUTPUT,

   AND_GATE_2,
   AND_GATE_3,
   AND_GATE_4,
   AND_GATE_5,
   AND_GATE_6,
   AND_GATE_7,
   AND_GATE_8,

   OR_GATE_2,
   OR_GATE_3,
   OR_GATE_4,
   OR_GATE_5,
```

```
    OR_GATE_6,
    OR_GATE_7,
    OR_GATE_8,

    NOT_GATE,

    ENCODER8_3,
    ENCODER4_2,
    ENCODER2_1,

    DECODER3_8,
    DECODER2_4,
    DECODER1_2,

    MULTIPLEXER8_1,
    MULTIPLEXER4_1,
    MULTIPLEXER2_1,

    DEMULTIPLEXER1_8,
    DEMULTIPLEXER1_4,
    DEMULTIPLEXER1_2,

    GENERICS,
    INPUT,
    AND_GATE,
    OR_GATE,
    ENCODER,
    DECODER,
    MULTIPLEXER,
    DEMULTIPLEXER,
    MAX_GATE_TYPES // useful for for loops
};
```

add_gate.h contains the definitions for all derived classes inherited from *node*. These classes include *input*, *output*, *and_gate*, *or_gate*, *not_gate*, *encoder*, *decoder*, *multiplexer* and *demultiplexer*. Each of these override *calculate()* function. Invalid/Incomplete value '-1' is checked in all calculation blocks for input pins and output pins are set accordingly. A *mapvalues()* function is defined to take in three input binary values to return respective decimal counterparts. This is used in various ways in calculation blocks of decoder, multiplexer and demultiplexer.

Node Adder

It is a structure with member *isAdding*(a bool value to track if the user is adding any nodes) and *new_node_typ*(a *node_types* to store the type of nodes the user is adding). *node_adder* structure is defined in **new_node.h**

Node Manager

node_manager class acts like an encapsulating class and manages all instances of nodes. All the relevant members and functions in this class are declared in **node_manager.h** and defined in **node_manager.cpp**. It has private members *nodes*(a vector of pointer to *node* class) and *links*(a vector of *link* class)**.** This class follows the singleton pattern, i.e only one object of this class is ever created. It has functions defined to add nodes, input pins, output pins and links, remove links, copy the values from output pin to connected input pins, call calculate function of respective derived class and render editor and nodes.

```
int node_manager::add_node (int& node_id, std::string name, node_types
n_typ) {
int temp = 0;
if (n_typ>=INPUT_1 && n_typ<=INPUT_8){
        temp = node_id;
        nodes.push_back (new and_gate{temp, "Input"});
        for ( int iter=0;iter<=(n_typ-INPUT_1);iter++)
                add_output_pins (temp, ++node_id, "I" + (std::to_string (iter)),
0);
}
else if (n_typ == OUTPUT) {
        temp = node_id;
        nodes.push_back (new output{temp, "Output"});
        add_input_pins (temp, ++node_id, "", -1);
}
else if (n_typ >= AND_GATE_2 && n_typ <= AND_GATE_8) {
        temp = node_id;
        nodes.push_back (new and_gate{temp, "AND Gate"});
        for (int iter = 0; iter <= (n_typ-AND_GATE_2+1); iter++)
                add_input_pins (temp, ++node_id, "I" + (std::to_string (iter)),
```

```
-1);
        add_output_pins(temp, ++node_id, "O",-1);
    }
//similar code for OR Gate and NOT Gate
    else if (n_typ >= ENCODER8_3 && n_typ <= ENCODER2_1) {
        temp = node_id;
        int outputno;
        int inputno;
        switch (n_typ) {
        case ENCODER8_3:
                nodes.push_back (new encoder{temp, "Encoder 8_3"});
                outputno = 3, inputno = 8;
                break;
        case ENCODER4_2:
                nodes.push_back (new encoder{temp, "Encoder 4_2"});
                outputno = 2, inputno = 4;
                break;
        case ENCODER2_1:
                nodes.push_back (new encoder{temp, "Encoder 2_1"});
                outputno = 1, inputno = 2;
                break;
        default:
                assert (0);
                break;
        }
        for (int iter = 0; iter < inputno; iter++)
                add_input_pins (temp, ++node_id, "S" + (std::to_string (iter)),
-1);
        for (int iter = 0; iter < outputno; iter++)
                add_output_pins (temp, ++node_id, "Y" + (std::to_string (iter)),
-1);
    }
//similar code for decoder, multiplexer and demultiplexer
else {
        assert (0);
    }
       return 0;
}
```

Adds respective nodes depending on the argument provided through a series of condition checking.

```
int node_manager::add_input_pins (int& node_id, int pin_id, std::string
pin_name,int value) {
	for(node* n: nodes) {
		if(n->m_unique_id == node_id) {
		n->input_pins.push_back({pin_id, pin_name,value});
		break;
		}
	}
	return 0;
}
```

Adds input pins depending on arguments and pushes them back in the vector of *in_pin*.

```
int node_manager::add_output_pins (int& node_id, int pin_id, std::string
pin_name,int value) {
	for(node* n: nodes) {
		if(n->m_unique_id == node_id) {
		n->output_pins.push_back({pin_id, pin_name,value});
		break;
		}
	}
	return 0;
}
```

Adds output pins.

```
int node_manager::add_link (int& m_id, int st_node, int st_pin, int ed_node, int
ed_pin) {
for (node* n : nodes) {
if (n->m_unique_id == st_node) {
  for (out_pin& opin : n->output_pins) {
    if (opin.pin_id == st_pin) { // identify start/output pin
      for (node* nk : nodes) {
          if (nk->m_unique_id == ed_node) {
             for (in_pin& ipin : nk->input_pins) {
               if (ipin.pin_id == ed_pin)
                  if (ipin.counter == 0) {
                     opin.connected_inputs.push_back (&ipin), ipin.counter++;
                     links.push_back ({m_id, st_node, st_pin, ed_node, ed_pin});
                }
               }
             }
           }
          }
        }
      }
          node* temp;
          for(node* n: nodes) {
          if(n->m_unique_id == st_node) {
          temp = n;
          }
          }
          for(node* n: nodes) {
          if(n->m_unique_id == ed_node) {
          n->m_inputs.push_back(temp);
          }
          }
          return 0;
}
```

For every link creation, it pushes end pins to *connected_inputs* of start pin and keeps count of start pins connected to input pin and ensures it to be 1.

```
int node_manager::remove_link (int l_id) {
in_pin* temp;
for(int n = 0; n < links.size(); ++n) {
  if(links[n].m_unique_id == l_id) {
   for (node* no : nodes) {
    if (no->m_unique_id == links[n].end_node) {
     for (in_pin& pin : no->input_pins) {
       if (pin.pin_id == links[n].end_pin) {
          pin.value = -1; // default condition
            if (pin.counter > 0)
                  pin.counter = 0;
                  temp = &pin;
                  for (node* nod : nodes) {
                    for (out_pin& opin : nod->output_pins) {
                      for (int i = 0; i < opin.connected_inputs.size ( ); i++) {
                        if (opin.connected_inputs[i] == temp) {
                          opin.connected_inputs[i] =
                       opin.connected_inputs[opin.connected_inputs.size ( ) - 1];
                              opin.connected_inputs.pop_back ( );
                                 break;
                            }}}} } }}}
         // bring the last element to the index to remove
         // ie overwrite the useless index
         // and pop the last one
         links[n] = links[links.size() - 1];
         links.pop_back();
         break;
         }
         }
         return 0;
}
```

For every link destruction, the link is overwritten with the last link in the vector and the last member is popped. All connected pins are set with -1 value and the information of the connected input pin is removed from the output pin.

```
int
node_manager::render ( ) {
    ImVec4 color = {0.36f, 0.36f, 0.36f, 1.0f};
    bool   iflag     = false; // flag to check if input pins/out pins exist
    bool   oflag = false;
    for (node* n : nodes) {

            ImNodes::BeginNode (n->m_unique_id);
            ImNodes::BeginNodeTitleBar ( );
            ImGui::Text ("%s", n->node_name.c_str ( ));
            ImNodes::EndNodeTitleBar ( );
            for (in_pin& pin : n->input_pins) {
                    if (pin.value == 0) // checking value and render proper color
                            color = {0.992f, 0.266f, 0.423f, 1.0f};
                    if (pin.value == 1)
                            color = {0.5f, 1.0f, 0.25f, 1.0f};
                    if (pin.value == -1)
                            color = {0.36f, 0.36f, 0.36f, 1.0f};
                    ImNodes::BeginInputAttribute (pin.pin_id);
                    ImGui::ColorButton ("1", color,
ImGuiColorEditFlags_NoTooltip, {15, 15});
                    ImGui::SameLine ( );
                    ImGui::Text ("%s", pin.pin_name.c_str ( ));
                    ImNodes::EndInputAttribute ( );
                    iflag = true;
            }
            if (iflag==true) ImGui::NewLine ( );
            for (out_pin& pin : n->output_pins) {
                    if (pin.value == 0)
                            color = {0.992f, 0.266f, 0.423f, 1.0f};
                    if (pin.value == 1)
                            color = {0.5f, 1.0f, 0.25f, 1.0f};
                    if (pin.value == -1)
                            color = {0.36f, 0.36f, 0.36f, 1.0f};
                    ImNodes::BeginOutputAttribute (pin.pin_id);
                    if (iflag == false) {                                   // if input
pin doesn't exist i.e for input node
                            if (ImGui::Button ("", {15, 15})) {     // create a button
and check if its pressed
                                    pin.value = ( int )!bool (pin.value); // flip 0 and 1
//could use review as I'm unsure how safe it is
                            }
                            ImGui::SameLine ( );
```

```
                    }
                    ImGui::Indent ( );
                    ImGui::Text ("%s", pin.pin_name.c_str ( ));
                    ImGui::SameLine ( );
                    ImGui::ColorButton ("", color,
ImGuiColorEditFlags_NoTooltip, {15, 15});
                    ImNodes::EndOutputAttribute ( );
                    oflag = true;
            }
            if (oflag == false) {
                    ImGui::Indent;
                    if (n->input_pins[0].value==-1)
                            ImGui::Button ("", {30, 30});
                    else
                            ImGui::Button
((std::to_string(n->input_pins[0].value)).c_str() , {30, 30});
            }
            ImNodes::EndNode ( );
            iflag = false;
            oflag = false;
    }
    for (link& l : links) {
            ImNodes::Link (l.m_unique_id, l.start_pin, l.end_pin);
    }
           return 0;
}
```

Renders every pin, links and nodes.

```
int node_manager::copyover ( ) {
for (node* n : nodes) {
 for (out_pin& opin : n->output_pins) {
  for (int i = 0; i < opin.connected_inputs.size ( ); i++) {
 opin.connected_inputs[i]->value = opin.value;
}
}
}
return 0;
}
```

Copies value of output pin to all connected input pins.

```
int node_manager::calculate ( ) {
    for (node* n : nodes) {
            if (n->node_name != "Input") {
                    n->calculate ( );
            }
    }
    return 0;
}
```

Calls *calculate()* function of respective classes by runtime **polymorphism**.

Implementation

This program uses SDL2 and Dear ImGui as its graphical library. ImNodes as an extension to ImGui for node based UI.

Window Creation

First the event handling and window management of this program is handled by SDL. A SDL_Window is created after setting many OpenGL parameters. OpenGL is strictly used by ImGui and ImNodes to create their widgets. Also the responsibility of loading OpenGL functions is handled by glew. After this OpenGL is set up for ImGui.

```
SDL_Window*     window = nullptr;
SDL_GLContext gl_context;
```

Initializes SDL Window and sets up context for OpenGL

```
if (SDL_Init (SDL_INIT_VIDEO | SDL_INIT_TIMER) != 0) {
        printf ("Error: %s\n", SDL_GetError( ));
        return -1;
}
```

Checks if SDL is properly initialized and returns -1 if it fails.

```
// GL 3.0 + GLSL 130
SDL_GL_SetAttribute (SDL_GL_CONTEXT_FLAGS, 0);
SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK,
SDL_GL_CONTEXT_PROFILE_CORE);
SDL_GL_SetAttribute (SDL_GL_CONTEXT_MAJOR_VERSION, 3);
SDL_GL_SetAttribute (SDL_GL_CONTEXT_MINOR_VERSION, 3);
SDL_GL_SetAttribute (SDL_GL_DOUBLEBUFFER, 1);
SDL_GL_SetAttribute (SDL_GL_DEPTH_SIZE, 24);
SDL_GL_SetAttribute (SDL_GL_STENCIL_SIZE, 8);
SDL_DisplayMode current;
SDL_GetCurrentDisplayMode (0, &current);
```

Sets up OpenGL attributes and SDL Display Mode.

```
window = SDL_CreateWindow ("Mantik", SDL_WINDOWPOS_CENTERED,
                    SDL_WINDOWPOS_CENTERED, 1280, 720,
                    SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE |
SDL_WINDOW_ALLOW_HIGHDPI);
gl_context = SDL_GL_CreateContext (window);
SDL_GL_MakeCurrent (window, gl_context);
SDL_GL_SetSwapInterval (1); // Enable vsync
```

This creates a SDL Window with name “Mantik” and sets its size as 1280x720. This also provides the window created as context to OpenGL.

```
if (glewInit( )) {
fprintf (stderr, "Failed to initialize OpenGL loader!\n");
return 1;
}
```

Checks if OpenGL is properly initiated.

```
const ImVec4 clear_color = ImVec4 (0.45f, 0.55f, 0.60f, 1.00f);
glClearColor (clear_color.x, clear_color.y, clear_color.z, clear_color.w);
```

Defines a vector(essentially array in this context) of rgba color format to set OpenGL context color.

Using ImGui Widgets

Now ImGui is set up. ImGui works on a stack based architecture. That is every component has an opening and a closing function. And all those functions must be balanced. That is the open function for a component must be complemented by its closing function. And components opened inside a component must be closed inside the same component. HTML is a perfect comparison.

A small example

```
ImGui::Begin();
//some block of code
ImGui::End();
```

```
IMGUI_CHECKVERSION( );
ImGui::CreateContext( );
ImGui_ImplSDL2_InitForOpenGL (window, gl_context);
ImGui_ImplOpenGL3_Init (glsl_version);
```

Creates a context for ImGui and passes SDL window and OpenGL context to ImGui for initialisation.

```
ImNodes::CreateContext( );
// Setup style
ImGui::StyleColorsDark( );
ImNodes::StyleColorsDark( );
ImNodes::PushAttributeFlag(ImNodesAttributeFlags_EnableLinkDetachWithD
ragClick | ImNodesAttributeFlags_EnableLinkCreationOnSnap);
ImNodesEditorContext* e1 = ImNodes::EditorContextCreate( );
```

This sets up style for ImGui and ImNodes. ImNodes is used with ImGui to create nodes in our program.

Event Loop

```
bool done;
while (!done) {
//some block of code
}
```

done is used to check the condition for the while loop. This loop contains every line of code related to interactivity. All the events that happen at runtime are handled in this loop. An event is an action or occurrence recognized by software, often originating asynchronously from the external environment, that may be handled by the software. This includes input from any input devices, interaction with items, etc. Event loop is a programming construct or design pattern that waits for and dispatches events or messages in a program. This 'while' loop acts as an event loop for this program.

Before entering the event loop, an instance of *node_manager* class is created as *node_man*.

Event Handling

Event handling is done by SDL. In each iteration of the *event loop*, events are polled. For every relevant event to the program, the state of a few variables are changed. Later depending upon the state of those variables, appropriate operations are performed on classes. Specifically, an instance of *node_manager* named *node_man.*

```
SDL_Event event;
while (SDL_PollEvent (&event)) {
        ImGui_ImplSDL2_ProcessEvent (&event);
       //event handling code
}
```

This creates an SDL event variable to handle all events that occur in the program and create a loop for polling events in order till all the events are handled. It also passes *event* variable to ImGui for taking input from window events.

The event handling code includes

```
if (event.type == SDL_QUIT || (event.key.keysym.sym == SDLK_ESCAPE))
done = true;
if (event.type == SDL_WINDOWEVENT && event.window.event
== SDL_WINDOWEVENT_CLOSE && event.window.windowID
== SDL_GetWindowID (window)) done = true;
```

This block of code handles exit from the program either by close button or ‘esc’ button by setting *done* true which stops the event loop.

```
if (isAdding.isAdding && isAdding.new_node_typ>GENERICS ) {
        c = isAdding.new_node_typ;
}
else if (isAdding.isAdding && event.type == SDL_MOUSEBUTTONDOWN)
{
        unique_number++;
        ImNodes::SetNodeScreenSpacePos(unique_number,
            ImGui::GetMousePos());
        node_man.add_node(unique_number, "Add Node",
            isAdding.new_node_typ);
        isAdding = {false, NOT_ADDING};
        c  = NOT_ADDING;
}
```

If *isAdding* is true and *new_node_type* is greater than GENERICS, a dummy variable *c*(to store generic node type) is set to required generic node type provided by the user through buttons.

Once specialization of generic types is provided, a unique number is provided to *add_node* function to create necessary nodes and place it to the active mouse location through ImNodes function.

ImGui Context and Windows

Then, ImGui frames and windows are created. The size data is extracted from the SDL window and fed to ImGui. Windows, frames and ImNodes sections are created next. Then, link creations are checked and on the event they are created, a link is added in node_man. Conversely, link destruction is also checked and removed from node_man.

```
ImGui_ImplOpenGL3_NewFrame( );
ImGui_ImplSDL2_NewFrame (window);
ImGui::NewFrame( );
ImGuiTabBarFlags tab_bar_flags = ImGuiTabBarFlags_None;
SDL_GetWindowSize (window, &s_w, &s_h);
ImVec2 window_size{( float )s_w * 0.75f, ( float )s_h};
ImGui::SetNextWindowSize (window_size);
ImGui::SetNextWindowPos ({0, 0});
ImGui::Begin ("Encloser", NULL,
ImGuiWindowFlags_NoBringToFrontOnFocus | ImGuiWindowFlags_NoMove
| ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoTitleBar);
ImNodes::EditorContextSet (e1);
ImNodes::BeginNodeEditor( );
node_man.render();
ImNodes::MiniMap();
ImNodes::EndNodeEditor( );
```

This handles all the context created for ImGui and ImNodes and renders everything related to ImNodes.

```
int link_id = -1, start_node_id = -1, start_pin_id = -1, end_node_id = -1,
end_pin_id = -1;
bool created_from_snap;
if (ImNodes::IsLinkCreated(&start_node_id, &start_pin_id, &end_node_id,
&end_pin_id, &created_from_snap)) {
    unique_number++;
    node_man.add_link(unique_number, start_node_id, start_pin_id,
            end_node_id, end_pin_id);
}
if (ImNodes::IsLinkDestroyed(&link_id)) {
    node_man.remove_link(link_id);
}
ImGui::End( );
```

This calls respective functions to handle link creation and link destruction by using ImNodes function to detect them.

Buttons

Various buttons are created in order to allow users to create specific blocks on the fly.

```
if (ImGui::Button("Or Gate"))
                        isAdding = {true, OR_GATE};
```

As you can see they are used as the conditions of if statements. Thus when they are clicked the enclosing code gets executed. In this case we transition to a state where we are adding an or gate.

Calculation and wrapping up loop

Next, business logic is done. The values of pins are copied via the links. Then, with enough info, calculation is performed for the output of every node. After calculation, rendering is done. Lastly OpenGL is deinitialized, SDL renderer and window are destroyed.

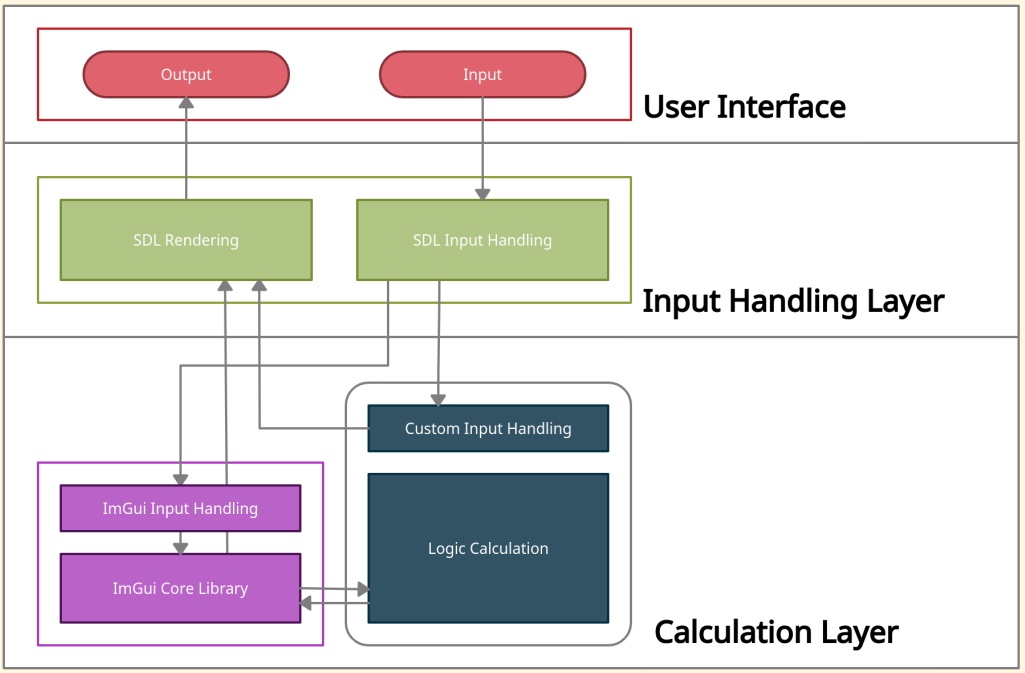
```
node_man.copyover ( );
node_man.calculate ( );
ImGui::Render( );
int fb_width, fb_height;
SDL_GL_GetDrawableSize (window, &fb_width, &fb_height);
glViewport (0, 0, fb_width, fb_height);
glClear (GL_COLOR_BUFFER_BIT);
ImGui_ImplOpenGL3_RenderDrawData (ImGui::GetDrawData( ));
SDL_GL_SwapWindow (window);
```

Calls respective functions to copy values and perform calculations to render everything.

```
ImNodes::EditorContextFree (e1);
ImGui_ImplOpenGL3_Shutdown( );
ImGui_ImplSDL2_Shutdown( );
ImGui::DestroyContext( );
SDL_GL_DeleteContext (gl_context);
SDL_DestroyWindow (window);
SDL_Quit( );
```

Frees ImNodes context and destroys OpenGL context as well SDL window.

Block diagram



Results

With everything wrapping up neatly, the end result was a working simulator that could simulate basic gates and some combinational circuits on the go. It has the necessary feature of having multiple input gates and a single node with multiple nodes as well as different types of multiplexer(like 8:1, 4:1, 2:1) and other combinational circuit subtypes.

The main objective for the development of this project was fulfilled, but not everything was on par with initial vision for the project. This was merely because of overestimating the difficulty of the project mixed in with time constraint. Much time ended up being spent on learning the necessary skills due to our lack of experience and knowledge. While it certainly isn't the best simulator out there, it very much aligns with the initial vision and leaves enough room for improvement. Here are some screenshots of our Mantik:

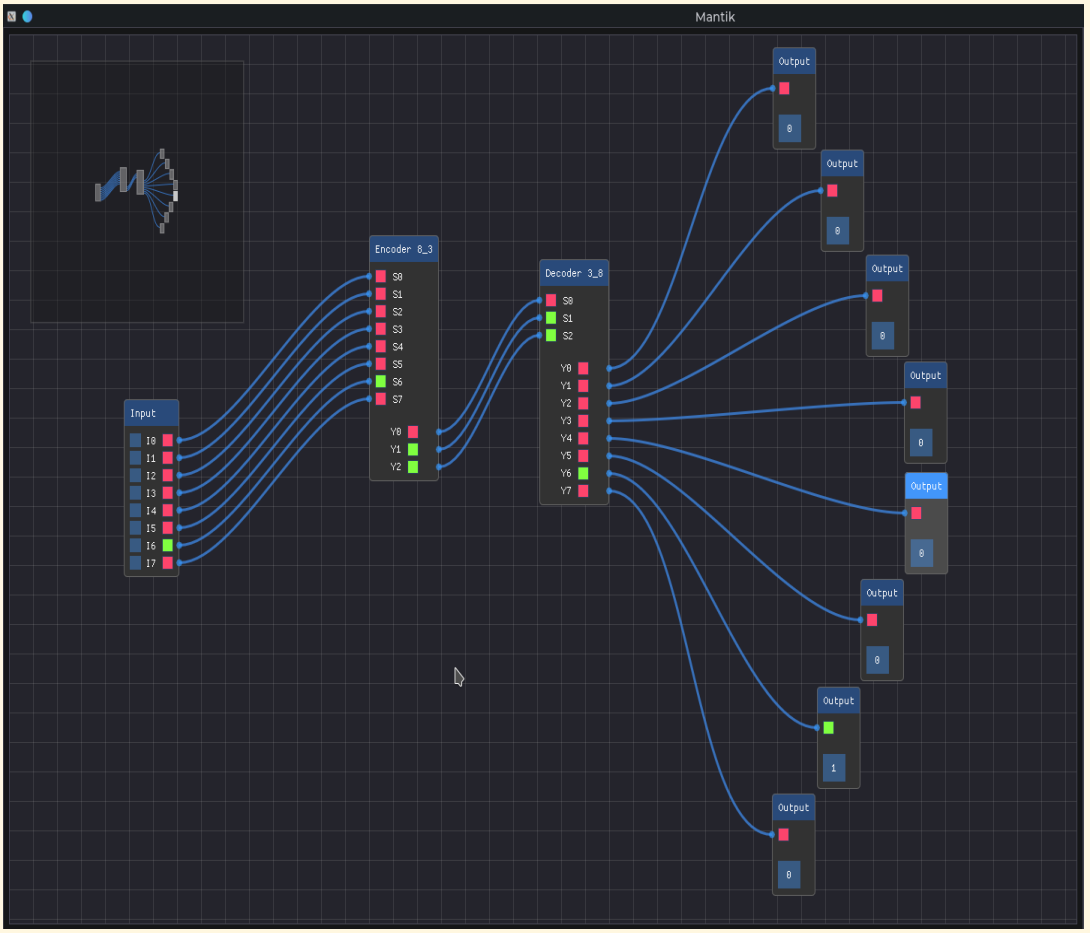


Fig.Rendering Windows

fig.Nodes Selection Tab

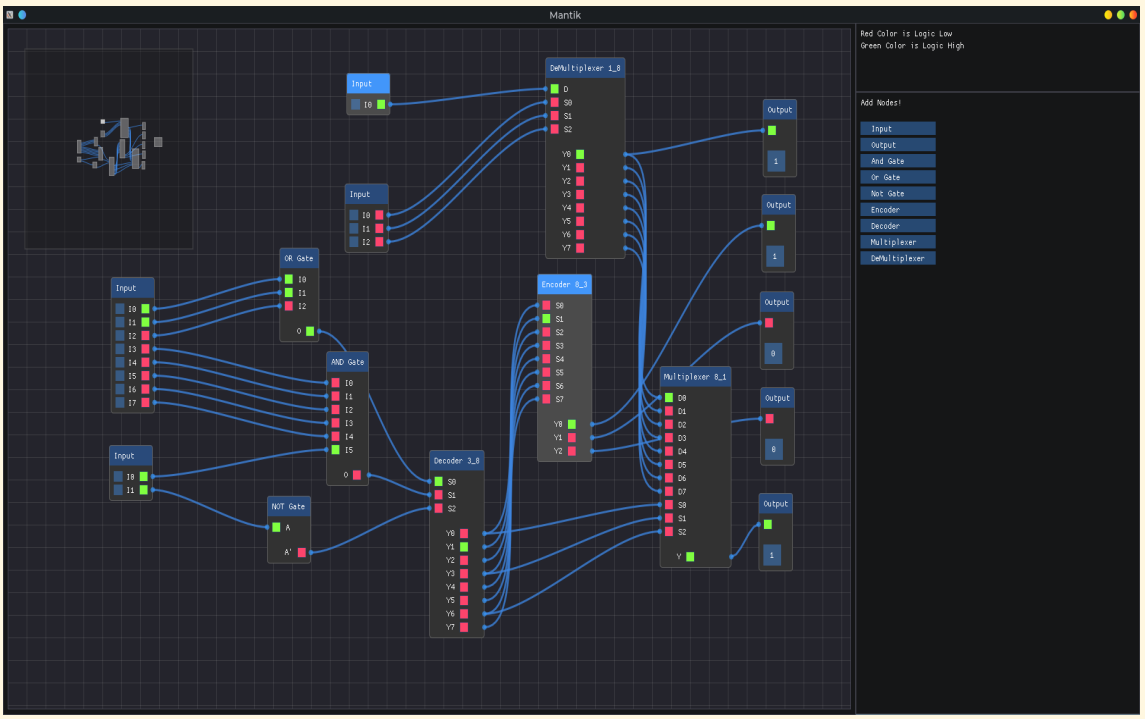


fig.Some Glimpse of Components

Problems Faced and solutions

While it might not look much to outsiders, everyone involved in the project worked their best with limited skill but enough motivation and passion to back it up. This doesn't mean the journey was smooth sailing. A lot of hindrances interrupted the team from giving their best. While that also meant it gave an opportunity to everyone to do something more.

Some of the common problems we came across throughout the development of this project:

- **Issues regarding standard template's type trait compatibility.**
  C++ standard template seems to require classes to maintain certain requirements to enable all of a container's features. Failing to meet them gives you a compilation error.
- **Unknown Libraries and paradigm.**
  As ImGui and to that extension ImNodes were unventured grounds for us, it took a lot of effort spending hours on end scrolling through documents and tutorials to get the general sense of idea.
- **Version Control.**
  As we never had used GIT to manage a project of this scope and lack of proper plan on our part, some minor merge issues were inevitable.
- **Object Oriented Programming.**
  While OOP is fascinating, familiarizing with it might not have been the easiest thing for us, often giving us the temptation to break down encapsulation and return to the good old procedural paradigm.
- **Linker and Runtime errors.**
  What's C++ programming if not linking errors with runtime errors on top. Classic memory errors were encountered which were easily mitigated using standard vectors provided by STL(Standard Template Library) following proper directions.

Limitations and future enhancements

A project of this scope with the time constraints was bound to have limited functionality. The goal wasn't to create a fully functional simulation software, rather to lay a groundwork for something that can be improved over time and learn about OOP in the meantime. Some of the things we believe this project lacks and could use further work on:

- **Simulate sequential circuits.**

  As it is now, it is limited to combinational circuits by its design. Since the calculations are performed every frame to have a simulation that's real time with every modification in circuit, we limit ourselves to a design which features heavy checking of connection and in turn limit inputs driven by its own output.
- **Remove components.**

  A very important feature ended up being shelved due to the time constraint. But this doesn't mean we didn't lay a groundwork to implement in the future.
- **Multiple instances of editors**

  The simulator now lacks the ability to handle multiple instances due to *node_manager* acting like a singleton. This is again that's something within our reach to improve upon.
- **Save and Load circuits**

  While this feature is something that's more than an easy fix, it's very much possible to implement with file handling.
- **Flexible UI**

  While the UI is serviceable, it certainly isn't the best. We fully intend to revisit it when we are more familiar with UI programming.

Conclusion and recommendations

This project was undeniably a good learning experience. It gave us the chance to strengthen and further our knowledge in Object Oriented Programming. Breaking the pre established notions is definitely not an easy feat which this project gave us the opportunity to. It hammered one of the most important and yet one of the most disregarded steps of application development: Planning. Coding is not the initial step for developing any application, rather a proper grasp of the situation and acknowledging the limitation is. Had we realized the extent of the project very well and our lack of skill beforehand and put forward a plan that would be effective in balancing the work pressure and quality control, this project might have been something more and even surpass our initial vision.

Thus, after the completion of this project, we conclude proper judgment of the situation is of utmost importance. While poor planning might have been an obstacle, it also made us push ourselves beyond our limit.

References

- Robert Lafore, "Object Oriented Programming with C++", Sams Publication
- Daya Sagar Baral and Diwakar Baral, "The Secrets of Object oriented Programming", Bhundipuran Prakasan
- Harvey M. Dietel and Paul J. Dietel, "C++ How to program", Pearson Education Inc.
- ImGui Github repository. https://github.com/ocornut/imgui
- ImNodes Github repository. https://github.com/Nelarius/imnodes
- SDL documentation. https://wiki.libsdl.org/
- C++ Standard Library Reference https://en.cppreference.com/w/