Department of Electronic and Telecommunication Engineering

University of Moratuwa

EN2160 – Electronic Design Realization

Custom Macropad Report

AMANA M.A.N. - 200022X

This is submitted as a partial fulfillment of the module EN2160: Electronic Design Realization

Department of Electronic and Telecommunication

University of Moratuwa

July 2023

Contents

Abstract

The Custom Macropad is a versatile input device featuring 17 programmable keys and 1 knob, designed to offer exceptional functionality and superior durability. It can be connected via Bluetooth® 4.0 using the ESP32 in BLE mode, making it compatible with various devices such as desktops, laptops, tablets, and smartphones. The device includes two USB ports for charging and power output, along with a power management unit ensuring optimal battery life. This report presents a comprehensive overview of the Custom Macropad, covering its hardware and software design, implementation, functionality, user interface, performance analysis, and conclusion.

Introduction

The custom macro pad is a compact keyboard that incorporates additional programmable keys and a knob. These keys can be assigned to execute various commands, keyboard shortcuts, and macros, allowing users to automate repetitive tasks on their computers. This report aims to detail the design and development process of the macro pad using the ESP32 microcontroller, highlighting its specifications, functionalities, and applications.

Project Overview

The project aims to design and develop a custom macro pad, a small keyboard with extra programmable keys and a knob, to provide users with enhanced efficiency and productivity. The macro pad allows users to assign various commands, keyboard shortcuts, and macros to the programmable keys, enabling the automation of long and repetitive tasks on a computer. This automation can save valuable time and effort, making the custom macro pad a valuable tool for gamers, programmers, and professionals across different fields.

Features and Functionality

The custom macro pad is equipped with the following features:

17 Programmable Keys: The macro pad features 17 programmable keys that users can customize to execute specific commands and macros.

1 Programmable Knob: In addition to the keys, the macro pad also includes a programmable knob that can be assigned to perform various functions.

Macros and Keyboard Shortcuts: Users can record and assign macros, which are sequences of keyboard and mouse actions, to the programmable keys. This allows for easy automation of complex tasks with a single button press.

Mechanical Switches: The keys use mechanical switches, providing durability and satisfying tactile feedback while typing.

Wireless Connectivity: The macro pad utilizes Bluetooth® 4.0 in BLE mode to establish wireless connectivity with compatible devices. This eliminates the need for physical connections and provides a seamless user experience.

Dual USB Ports: The device incorporates two USB ports – a Micro B USB port for charging with a 3.7-5.5V adapter and a USB A port acting as a 5V, 1A output port for powering other devices.

Power Management: To optimize battery life, the macro pad includes a T6845-C power bank module and voltage regulator, ensuring efficient power management and extended usability.

Long Battery Life: With just one charge, the macro pad can operate continuously for up to 12 hours, reducing the need for frequent recharging.

Compact and Portable Design: The product weighs 200g and has a compact size of 100x72x35mm, making it highly portable and suitable for various workspaces.

Versatile Device Support: The macro pad is designed to support desktops, laptops, tablets, and smartphones, making it adaptable and compatible with a wide range of devices.

Implementation of Custom Macro Pad

The implementation of the custom macro pad involves several key components and steps. Let's explore the detailed implementation process:

1.Hardware Components:

Microcontroller:

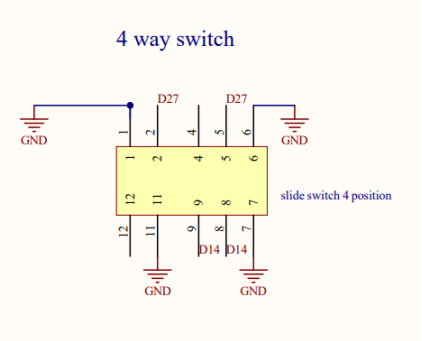
The heart of the Macro Pad is the ESP32 development board. The ESP32 is chosen for its high processing power, low power consumption, built-in Wi-Fi and Bluetooth capabilities, and sufficient GPIO pins to handle the matrix keypad. Additionally, the ESP32's versatility allows for seamless integration of various communication protocols and user interaction options.

Matrix Keypad:

The Macro Pad features a matrix keypad layout with 17 tactile switches. The matrix configuration optimizes the number of GPIO pins used to interface with the switches, making it more efficient than individually wiring each switch. Each switch is assigned a unique position within the matrix, allowing users to press any combination of keys to trigger specific macros. CHERRY G99-0742 mechanical switches are used as the keys to provide durability and tactile feedback during keypresses.

Four Levels of Switch Assignment:

A 10-pin four-position sliding switch is used to enable users to toggle between different sets of macros or commands stored in the custom macro pad. Each position of the sliding switch corresponds to a different level, and each level represents a unique configuration of programmable keys and knob assignments.



D27 and D14 pins are set to input pull-up and each level is identified based on the table below:

Level	D27	D14
1	LOW	HIGH
2	HIGH	LOW
3	HIGH	HIGH
4	LOW	LOW

Programmable Knob:

A rotary encoder with a push-button function is employed as the programmable knob for easy interaction.

LED Indicators:

LEDs are integrated into each key and the knob to provide visual feedback during testing and operation.

Battery and Power Management:

A rechargeable battery and power management circuitry ensure efficient power utilization and extended battery life. Two 3.7V Li-ion 3200mAh batteries are used with the T6845-C power bank module.

2. Firmware Development:

The firmware development involves programming the ESP32 microcontroller to handle various functionalities of the custom macro pad. Key steps in firmware development include:

Bluetooth Communication: Implementing BLE communication to allow wireless connectivity between the macro pad and compatible devices.

Macro Recording and Playback: Creating code to record and store sequences of keyboard and mouse actions as macros, and then playing back these macros upon user request.

Handling Programmable Keys and Knob: Writing code to detect keypresses and knob rotation, and mapping these events to specific commands, macros, or shortcuts.

LED Control: Integrating code to control the LED indicators, ensuring they light up in response to keypresses and knob actions.

The custom macro pad is programmed using Arduino IDE version 2.1.0. The project utilizes two essential libraries, "BleKeyboard.h" and "Encoder.h".

3. Functional Testing:

Functional testing focuses on verifying that each feature and functionality of the macro pad works as expected. It involves testing the programmable keys, the knob, and the execution of macros.

Method: Manually test each programmable key, the knob, and the execution of macros.

Procedure:

1. **Testing Mode Activation**: Enter the testing mode by pressing the designated key combination (Num Lock key + Enter key). This activates the testing mode and prepares the macro pad for functional testing.
2. **Testing LED Indicator**: When the testing mode is activated, an LED indicator should light up to indicate that the macro pad is in testing mode. This visual feedback ensures that the tester can identify when the macro pad is ready for testing.
3. **Test Programmable Keys**: One by one, press each programmable key on the macro pad. During this testing phase, verify that each key registers the keypress accurately and executes the assigned command or macro as intended.
4. **Test Programmable Knob**: Rotate the programmable knob in both clockwise and counterclockwise directions. Verify that the knob is responsive and accurately performs the assigned functions.
5. **Macro Execution Testing**: During the functional testing, make use of the recorded macros on the programmable keys. Play back each macro to ensure that they execute the correct sequence of keyboard and mouse actions without any errors.
6. **LED Feedback**: As each key is pressed or the knob is rotated, an LED indicator associated with that key or knob should light up. This LED feedback provides visual confirmation that the macro pad is detecting the input and performing the desired action.
7. **Error Handling**: During the functional testing, pay attention to any unexpected behavior or errors. If any issues or inaccuracies are encountered, they should be noted for further investigation and correction.
8. **Exit Testing Mode**: After completing the functional testing, exit the testing mode by pressing the designated key combination again (e.g., Num Lock key + Enter key). Ensure that the macro pad returns to its regular operational mode after testing.

4.Enclosure Design

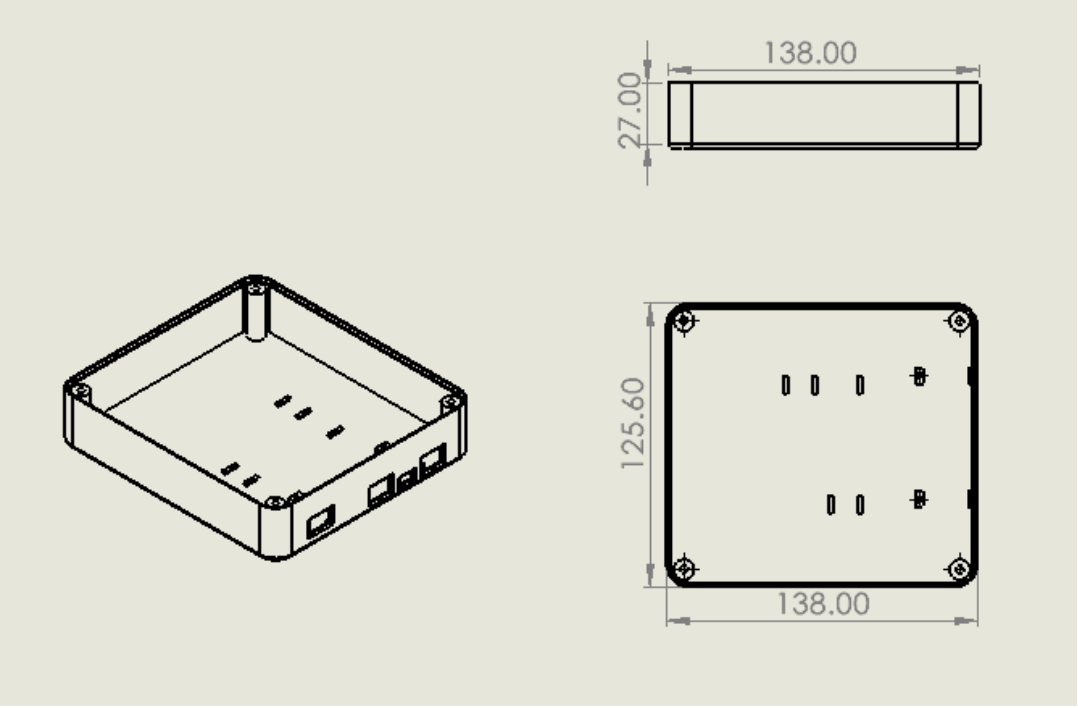
The enclosure design is a crucial aspect of the macropad, providing protection and a user-friendly interface. It should consider the following factors:

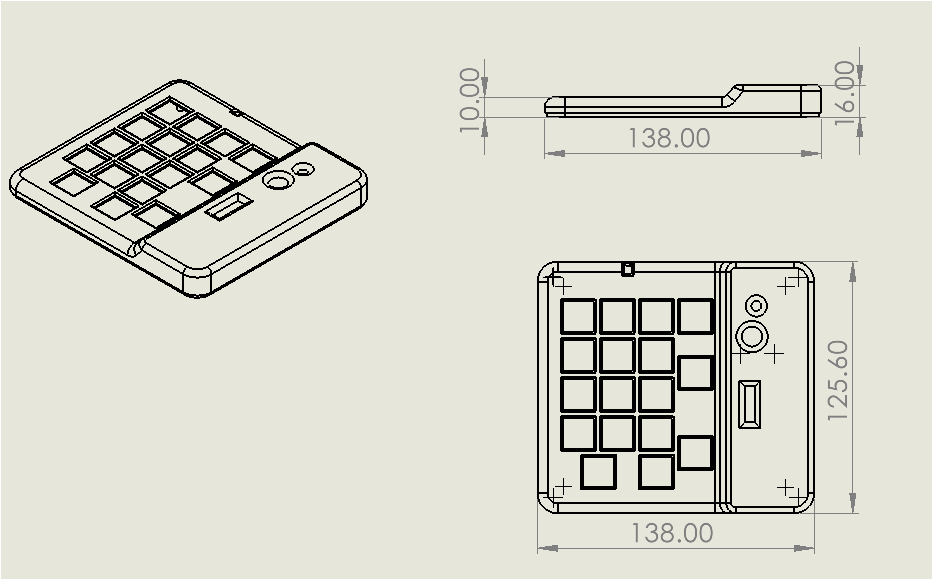
Dimensions: Ensure that the enclosure fits all the components comfortably and allows easy access to switches, the knob, and the 4-position switch.

Material: Select a durable and aesthetically pleasing material. ABS is selected due to its low cost, lightweight, and ease of molding. It offers good impact resistance and can be easily 3D printed or injection molded. ABS come in various colors and textures, allowing for customization.

Mounting: Design suitable mounting points for the PCB and other components.

Port Access: Include openings for the USB connector and any other necessary ports.

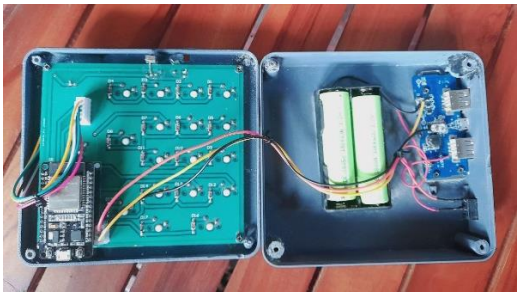




5. Instructions for Assembly:

To assemble the custom macropad, follow these steps:

1. Gather all the required components and ensure that have the necessary tools (soldering iron, screwdrivers, etc.).
2. Solder the mechanical switches and other through-hole components onto the PCB following the provided schematics.
3. Attach the ESP32 module/development board to the PCB.
4. Install the rotary knob and the 4-position switch into their designated positions.
5. Solder the LED indicator for testing/debugging purposes.
6. Double-check all connections and solder joints for any potential errors.
7. Connect the USB connector to the designated pads on the PCB.
8. Mount the PCB securely inside the enclosure.
9. Close and secure the enclosure using the appropriate screws.



Macro Pad Functionality

The custom macro pad offers several functionalities to enhance productivity and automation:

1. Programmable Keys:

Users can assign specific commands, keyboard shortcuts, or macros to each programmable key. When a key is pressed, it triggers the assigned action, enabling quick execution of complex tasks or sequences of actions.

2. Programmable Knob:

The programmable knob can be customized to perform various functions, such as volume control, scrolling, zooming, or switching between applications. Users can set the knob's behavior based on their preferences and frequently used tasks.

3. Macro Recording and Playback:

Users can record sequences of keyboard and mouse actions as macros. These macros can be assigned to specific keys for easy execution. When a macro-enabled key is pressed, the associated macro is played back, automating repetitive tasks, and saving time and effort.

4. Four Levels of Switch Assignment:

One of the standout features of the Macro Pad is its four levels of switch assignment. Users can switch between different sets of macros by toggling through the levels. This functionality provides users with greater flexibility and the ability to customize macros for various tasks or applications, such as programming, graphic design, gaming, or office productivity.

5. Wireless Connectivity:

The macro pad connects wirelessly to compatible devices using Bluetooth® 4.0 in BLE mode. This eliminates the need for physical connections, providing a seamless and clutter-free experience.

6. LED Feedback:

Each key and knob are equipped with LED indicators. These LEDs light up when the corresponding key or knob is pressed or rotated, providing visual feedback during testing and operation.

7. Power Management:

The custom macro pad incorporates a rechargeable battery and efficient power management circuitry. With just one charge, the macro pad can operate for up to 12 hours, ensuring extended usability without frequent recharging.

Applications

The custom macro pad finds applications in various fields, including:

Gaming: Gamers can use the macro pad to execute complex in-game actions or macros, enhancing their gaming performance and convenience.

Programming: Programmers can assign frequently used code snippets and shortcuts to the programmable keys, improving coding efficiency.

Productivity: Professionals can automate repetitive tasks in productivity applications, saving time and effort in their daily work.

Bill Of Materials

Component	Quantity	Unit Price	Total Price	Supplier
ESP32-DevKitC-32UE	1	$10.00	$10.00	Mouser Electronics Datasheet- https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html
CHERRY G99-0742 Switch	17	$0.34	$5.78	DigiKey Datasheet- https://media.digikey.com/pdf/Data%20Sheets/Cherry%20PDFs/EN_CHERRY_ML1A-xxxx.pdf
Slide switch 4 position	1	$0.89	$0.89	Mouser Electronics Datasheet- https://www.mouser.com/datasheet/2/670/ds04_254_smt-1777718.pdf
Rotary Encoder	1	$2.33	$2.33	Mouser Electronics Datasheet- https://www.mouser.com/datasheet/2/54/pec11l-777793.pdf
Li-ion battery 2500mAh	2	$5.21	$10.42	Mouser Electronics Datasheet- https://www.mouser.com/datasheet/2/855/ASR00050_18650_2500mAh-3078640.pdf
Ir receiver	1	$1.77	$1.77	Mouser Electronics Datasheet- https://www.vishay.com/doc?82489
Ir transmitter	1	$1.15	$1.15	Mouser Electronics Datasheet- https://www.vishay.com/doc?84755
LED	1	$0.51	$0.51	Mouser Electronics

				Datasheet- https://www.mouser.com/datasheet/2/216/APHHS1005LSECK_J3_PF-1660327.pdf
Pin header	5 pin - 1, 3- pin 1, 4-pin 1	$0.58	$0.58	Mouser Electronics Datasheet- https://www.mouser.com/datasheet/2/73/HDR100IMP40M_G_V_TH-2525605.pdf
Resistor	1K , 100K , 200K	$0.47	$1.41	Mouser Electronics Datasheet- https://www.vishay.com/doc?28729
Diode 1N4148	17	$0.19	$3.23	Mouser Electronics Datasheet - https://www.mouser.com/datasheet/2/849/1n4148-2577222.pdf
JST connectors	3 pin - 1 5 pin - 1	$1.33	$3.99	Mouser Electronics
PCB	1	$9	$9	JLCPCB
Enclosure	1	$20	$20	
Total cost			$71	

With 15% profit, total price = $82

Conclusion

The project aimed to design and created a versatile and efficient tool that enhances productivity and automation for gamers, programmers, and professionals across various fields. Through the implementation of the custom macro pad, we have delivered a powerful and user-friendly device that fulfills these goals and more.

Future improvements

1. **OLED Display**: Incorporating an intuitive and informative OLED display improves the user interface, making it easier to navigate and providing clear visual feedback.

2. **RGB Lighting Effects**: Customizable RGB lighting effects add visual feedback and enhance the macro pad's aesthetic appeal.

3. **Wireless Charging**: Integrating wireless charging capabilities eliminates the need for constant cable connections, offering convenience and ease of use.

4**. Macro Recording**: Enabling the ability to record and assign macros directly on the macro pad without additional software simplifies the customization process for users.

5. **Haptic Feedback**: Integrating haptic feedback technology provides physical feedback or vibration when buttons are pressed or the knob is turned, enhancing the overall user experience and tactile response. Improved Feature

References

ESP32 BLE Keyboard library

GitHub - T-vK/ESP32-BLE-Keyboard: Bluetooth LE Keyboard library for the ESP32 (Arduino IDE compatible)

ESP32-DevKitC-32UE Datasheet

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html

CHERRY G99-0742 Switch Datasheet-
https://media.digikey.com/pdf/Data%20Sheets/Cherry%20PDFs/EN_CHERRY_ML1A-xxxx.pdf

Rotary Encoder Datasheet-

https://www.mouser.com/datasheet/2/54/pec11l-777793.pdf

Appendices

1. Appendix A- Arduino code

```
#include <BleKeyboard.h>
#include <Encoder.h>

const byte output_count = 5; // four rows
const byte input_count = 4;  // four columns

// Define the symbols on the buttons of the keypads
char hexaKeys[output_count][input_count] = {
  {'1', '2', '3', '4'},
  {'5', '6', '7', '8'},
  {'9', '0', 'a', 'emp'},
  {'c', 'd', 'e', 'f'},
  {'g', 'emp2', 'i', 'emp3'}
};

byte outputs[output_count] = {18, 2, 5, 4, 15}; // Connect to the row pinouts of
the keypad
byte inputs[input_count] = {19, 21, 22, 23};     // Connect to the column pinouts
of the keypad

// Rotary encoder pins
const int encoderPinA = 34; // CLK pin connected to D34
const int encoderPinB = 35; // DT pin connected to D35

const int LED = 12;

// Variables to store encoder state
volatile int encoderPos = 0;
volatile bool encoderAState = LOW;
volatile bool lastEncoderAState = LOW;

int key_state;
int key_1 = 27;
int key_2 = 14;

BleKeyboard bleKeyboard;
Encoder myEnc(encoderPinA, encoderPinB);

void setup() {
  Serial.begin(9600);
  bleKeyboard.begin();
```

```
  // KEYS LEVEL
  pinMode( key_1,INPUT_PULLUP);
  pinMode( key_2,INPUT_PULLUP);

  for (int i = 0; i < output_count; i++) { // Declaring all the outputs and
setting them high
    pinMode(outputs[i], OUTPUT);
    digitalWrite(outputs[i], HIGH);
  }

  for (int i = 0; i < input_count; i++) {  // Declaring all the inputs and
activating the internal pull-up resistor
    pinMode(inputs[i], INPUT_PULLUP);
  }

  pinMode(encoderPinA, INPUT_PULLUP);
  pinMode(encoderPinB, INPUT_PULLUP);

  attachInterrupt(digitalPinToInterrupt(encoderPinA), handleEncoderInterrupt,
CHANGE);

  Serial.println("Connected");
}

void loop() {
  digitalWrite(LED, HIGH);
  if (bleKeyboard.isConnected()) {
    // Read the encoder position
    long newPosition = myEnc.read();

    // Check if the encoder position has changed
    if (newPosition != encoderPos) {
      // Update the volume based on the encoder movement
      int volumeChange = newPosition - encoderPos;
      if (volumeChange > 0) {
        // Increase volume
        for (int i = 0; i < volumeChange; i++) {
          bleKeyboard.write(KEY_MEDIA_VOLUME_UP);
        }
      } else {
        // Decrease volume
        for (int i = 0; i > volumeChange; i--) {
          bleKeyboard.write(KEY_MEDIA_VOLUME_DOWN);
        }
      }
```

```
      // Print the current encoder position
      Serial.print("Encoder Position: ");
      Serial.println(newPosition);

      // Update the last encoder position
      encoderPos = newPosition;
    }

    if (digitalRead(key_1)==LOW && digitalRead(key_2)==HIGH)
      key_state=1;
    else if (digitalRead(key_1)==HIGH && digitalRead(key_2)==LOW)
      key_state=2;
    else if (digitalRead(key_1)==HIGH && digitalRead(key_2)==HIGH)
      key_state=3;
    else
      key_state=4;

    Serial.print("key state ");
    Serial.println(key_state);
    Serial.println(digitalRead(key_1));
    Serial.println(digitalRead(key_2));

    for (int i = 0; i < output_count; i++) {
      digitalWrite(outputs[i], LOW);   // Setting one row low
      delayMicroseconds(5);             // Giving electronics time to settle down

      for (int j = 0; j < input_count; j++) {
        if (digitalRead(inputs[j]) == LOW) {
          keyPressed(i, j);             // Calling keyPressed function if one of
the inputs reads low
        }
      }

      digitalWrite(outputs[i], HIGH);
      delayMicroseconds(500);           // Setting the row high and waiting
0.5ms until the next cycle
    }
  }
}

void keyPressed(int row, int col) {
  Serial.print("Output: ");
  Serial.print(row);
  Serial.print(" Input: ");
```

```
Serial.print(col);
Serial.println(" ");

char key = hexaKeys[row][col];


//functionality testing
if (key=='1' && key=='f'){
  switch (key) {
    case '1':
      digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
      delay(1000);                       // wait for a second
      digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
      delay(1000);
      break;
    case '2':
     digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
      delay(1000);                       // wait for a second
      digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
      delay(1000);
      break;




    case '3':
      digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
      delay(1000);                       // wait for a second
      digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
      delay(1000);
      break;
    case '4':
      digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
      delay(1000);                       // wait for a second
      digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
      delay(1000);
      break;
    case '5':
      digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
      delay(1000);                       // wait for a second
      digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
      delay(1000);
      break;
    case '6':
      digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
```

```
  delay(1000);                       // wait for a second
  digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);
  break;
case '7':
  digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);
  break;
case '8':
  digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);
  break;
case '9':
  digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);
  break;
case '0':
  digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);
  break;
case 'a':
  digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);
  break;
case 'c':
  digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);
  break;
case 'd':
  digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
   break;
```

```
    case 'e':
      digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
      delay(1000);                      // wait for a second
      digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW

      break;
    case 'f':
      digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
      delay(1000);                      // wait for a second
      digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
      delay(1000);
      break;
    case 'g':
      digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
      delay(1000);                      // wait for a second
      digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW

      break;
    case 'i':
      digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
      delay(1000);                      // wait for a second
      digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW

      break;
  }
}

if (key_state==1){
  switch (key) {
    case '1':
      bleKeyboard.write('a');
      break;
    case '2':
      bleKeyboard.write('b');
      break;
    case '3':
      bleKeyboard.write('c');
      break;
    case '4':
      bleKeyboard.write('d');
      break;
    case '5':
      bleKeyboard.write('e');
      break;
    case '6':
```

```
      bleKeyboard.write('f');
      break;
    case '7':
      bleKeyboard.write('g');
      break;
    case '8':
      bleKeyboard.write('h');
      break;
    case '9':
      bleKeyboard.write('i');
      break;
    case '0':
      bleKeyboard.write('j');
      break;
    case 'a':
      bleKeyboard.write('k');
      break;
    case 'c':
      bleKeyboard.write('l');
      break;
    case 'd':
      bleKeyboard.write('m');
      break;
    case 'e':
      bleKeyboard.write('n');
      break;
    case 'f':
      bleKeyboard.write('o');
      break;
    case 'g':
      bleKeyboard.write('p');
      break;
    case 'i':
      bleKeyboard.write('q');
      break;
  }
}


if (key_state==3){
  switch (key) {
    case '1':
      //bleKeyboard.write("Level 1 - num pad");
      break;
    case '2':
```

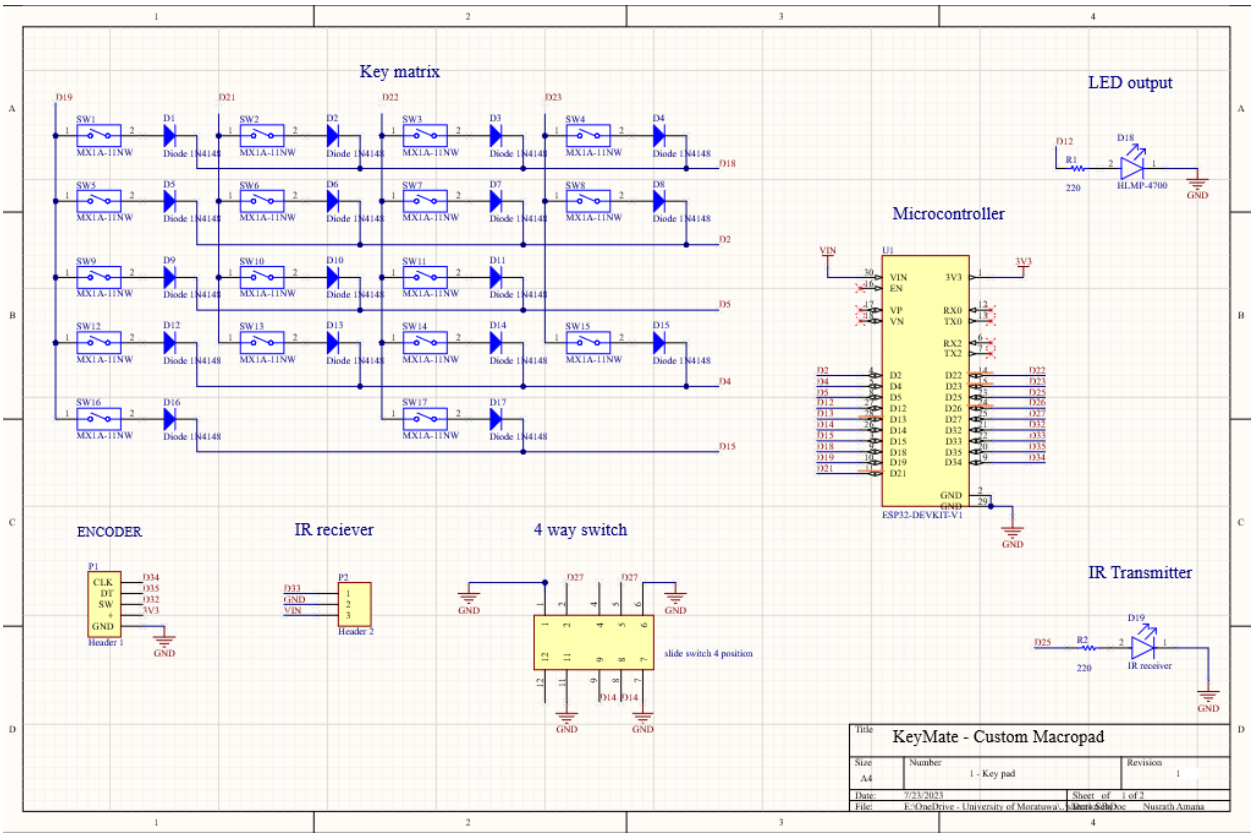
```
  bleKeyboard.press(KEY_NUM_SLASH);
  break;
case '3':
  bleKeyboard.press(KEY_NUM_ASTERISK);
  break;
case '4':
  bleKeyboard.press(KEY_NUM_MINUS);
  break;
case '5':
  bleKeyboard.press(KEY_NUM_7);
  break;
case '6':
  bleKeyboard.press(KEY_NUM_8);
  break;
case '7':
  bleKeyboard.press(KEY_NUM_9);
  break;
case '8':
  bleKeyboard.press(KEY_NUM_PLUS);
  break;
case '9':
  bleKeyboard.press(KEY_NUM_4);
  break;
case '0':
  bleKeyboard.press(KEY_NUM_5);
  break;
case 'a':
  bleKeyboard.press(KEY_NUM_6);
  break;
case 'c':
  bleKeyboard.press(KEY_NUM_1);
  break;
case 'd':
  bleKeyboard.press(KEY_NUM_2);
  break;
case 'e':
  bleKeyboard.press(KEY_NUM_3);
  break;
case 'f':
  bleKeyboard.press(KEY_NUM_ENTER);
  break;
case 'g':
  bleKeyboard.press(KEY_NUM_0);
  break;
case 'i':
```

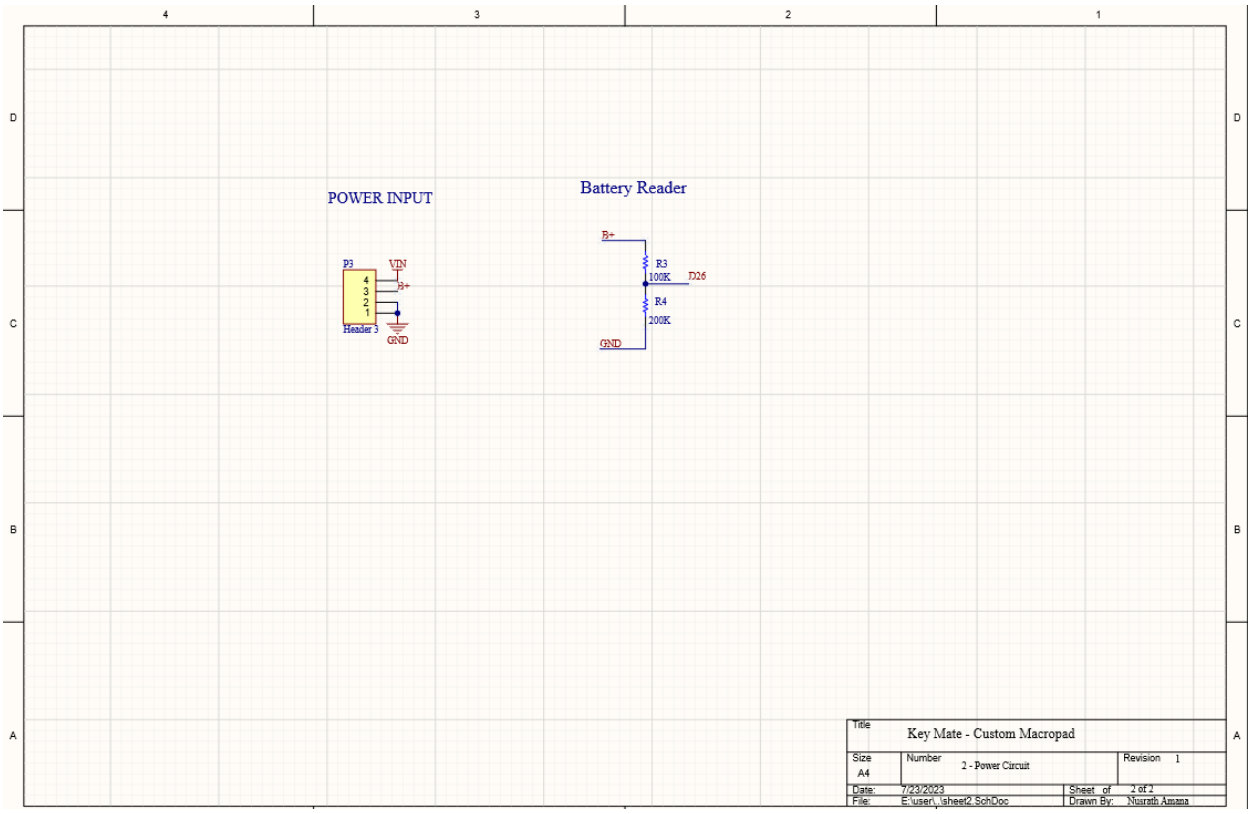
```
      bleKeyboard.press(KEY_NUM_PERIOD);
      break;
  }
}

if (key_state==2){
  switch (key) {
    case '1':
      bleKeyboard.press(KEY_LEFT_CTRL);
      bleKeyboard.write('a');
      break;
    case '2':
      bleKeyboard.press(KEY_LEFT_CTRL);
      bleKeyboard.write('c');
      break;
    case '3':
      bleKeyboard.press(KEY_LEFT_CTRL);
      bleKeyboard.write('x');
      break;
    case '4':
      bleKeyboard.press(KEY_LEFT_CTRL);
      bleKeyboard.write('v');
      break;
    case '5':
      bleKeyboard.press(KEY_HOME);

      break;
    case '6':
      bleKeyboard.press(KEY_UP_ARROW);
      break;
    case '7':
      bleKeyboard.press(KEY_PAGE_UP);
      break;
    case '8':
      bleKeyboard.press(KEY_LEFT_CTRL);
      bleKeyboard.write('z');
      break;
    case '9':
      bleKeyboard.press(KEY_LEFT_ARROW);
      break;
    case '0':
      bleKeyboard.press(KEY_LEFT_CTRL);
      bleKeyboard.write('Y');
      break;
    case 'a':
```

```
        bleKeyboard.press(KEY_RIGHT_ARROW);
        break;
      case 'c':
        bleKeyboard.press(KEY_END);
        break;
      case 'd':
        bleKeyboard.press(KEY_DOWN_ARROW);
        break;
      case 'e':
        bleKeyboard.press(KEY_PAGE_DOWN);
        break;
      case 'f':
        bleKeyboard.press(KEY_NUM_ENTER);
        break;
      case 'g':
        bleKeyboard.press(KEY_BACKSPACE);
        break;
      case 'i':
        bleKeyboard.press(KEY_DELETE);
        break;
    }

  delay(300);
  bleKeyboard.releaseAll();
}

void handleEncoderInterrupt() {
  // Read the current state of the CLK pin
  bool currEncoderAState = digitalRead(encoderPinA);

  // Check if the CLK pin state has changed
  if (currEncoderAState != lastEncoderAState) {
    // Check the state of the DT pin to determine the rotation direction
    if (digitalRead(encoderPinB) != currEncoderAState) {
      // Increase volume
      myEnc.write(myEnc.read() + 1);
    } else {
      // Decrease volume
      myEnc.write(myEnc.read() - 1);
    }
  }

  // Update the last CLK pin state
  lastEncoderAState = currEncoderAState;
}
```

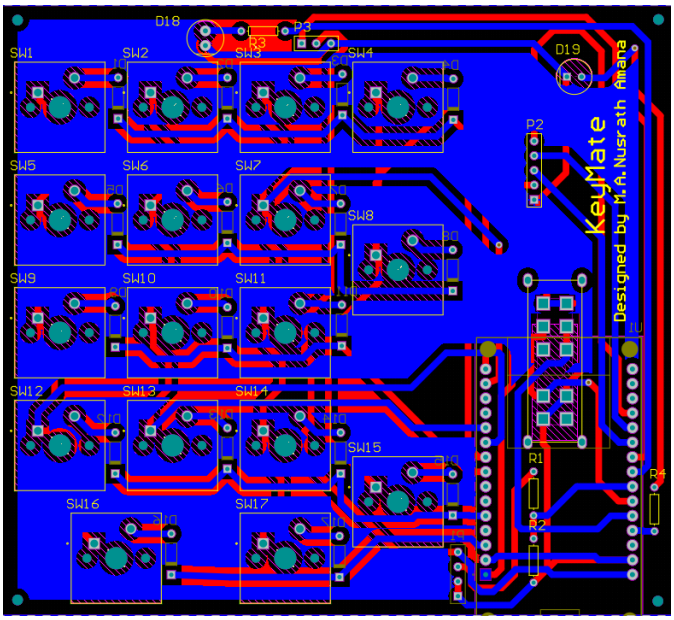
2. Appendix B- PCB Design

2.1. Schematic

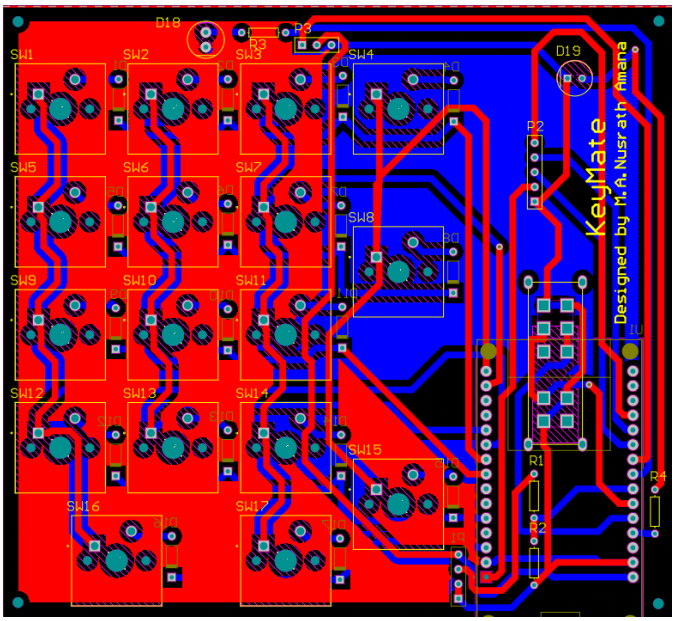




2.2. PCB Layout

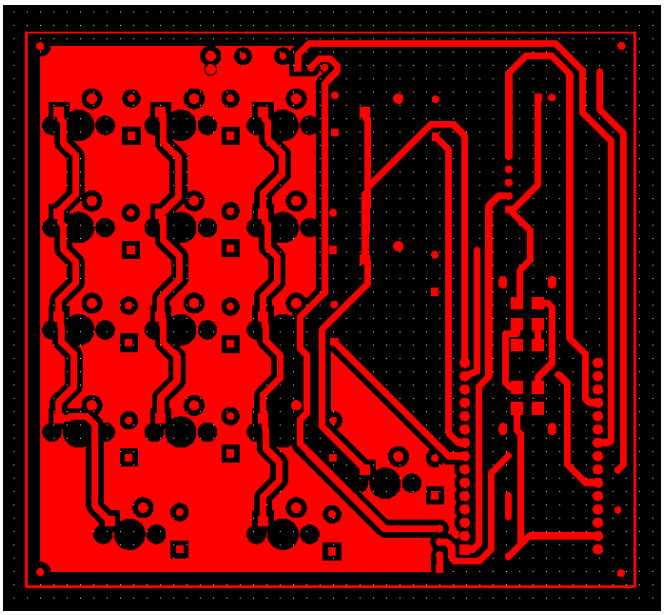


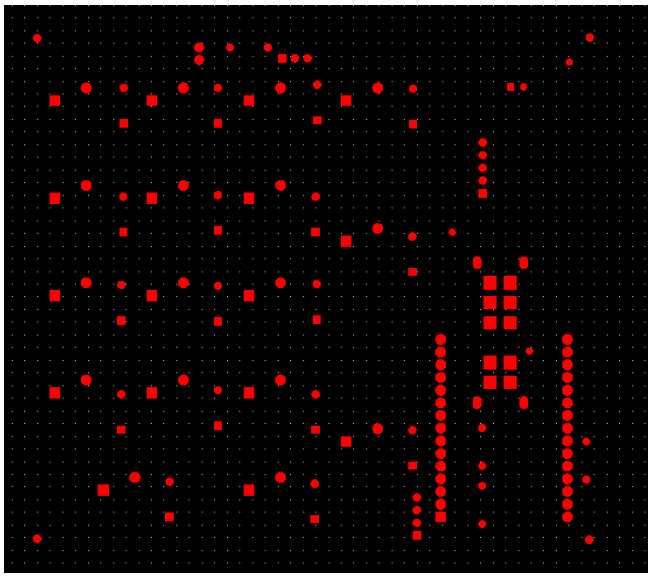
Bottom layer

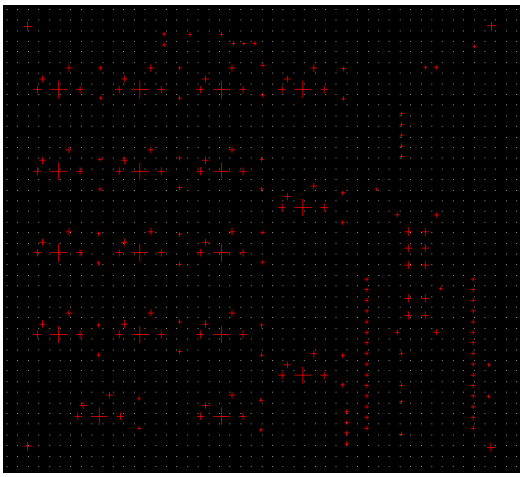


Top layer

2.3. Greber Files







3. Appendix C- Enclosure Design