ECE342, System Documentation

James Wilcock, Aiden Olsen, Blake Wiker

March 8, 2022

Contents

1 Overview

Individually addressable 5x5x7 led cube utilizing layer multiplexing with intuitive user interface and audio reactive visualization.

Our system displays three preprogrammed animations, messages, and music visualization in 10 unique colors. Lattice can be controlled wirelessly with a GUI. This system is capable of controlling each individual led on a given layer, setting its RGB values to a corresponding color. Through the use of multiplexing, it appears the entire cube is on. Wireless connection is web server based.

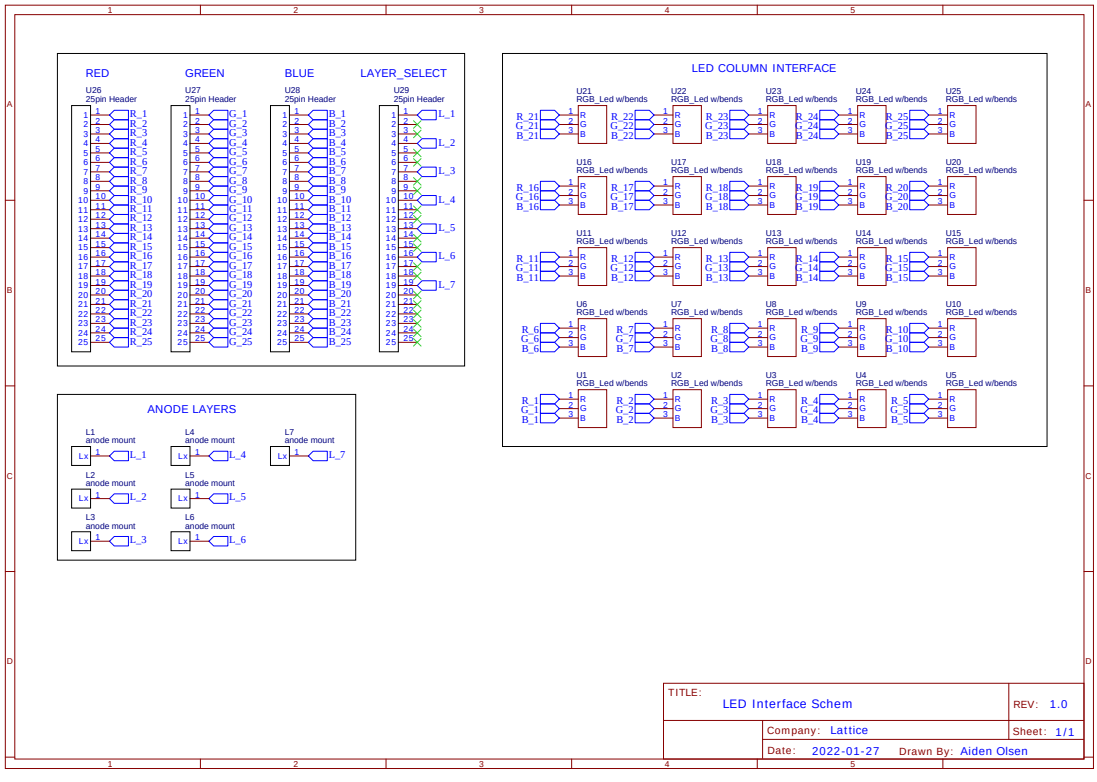
1.1 Engineering Requirements

1. The system should display at least 10 unique colors.
2. The system display should have a resolution of at least 5x5x7.
3. The system should have at least 3 pre-programmed animations.
4. The system should have a GUI that allows users to design at least 3 new messages and animations.
5. The system display should not appear to flicker to at least 2 people other than the project designers.
6. The system should visually respond to normal speaking volume from a distance of at least 5 feet.
7. The system should communicate with a peripheral device from at least 20 feet.

2 Schematic Diagrams

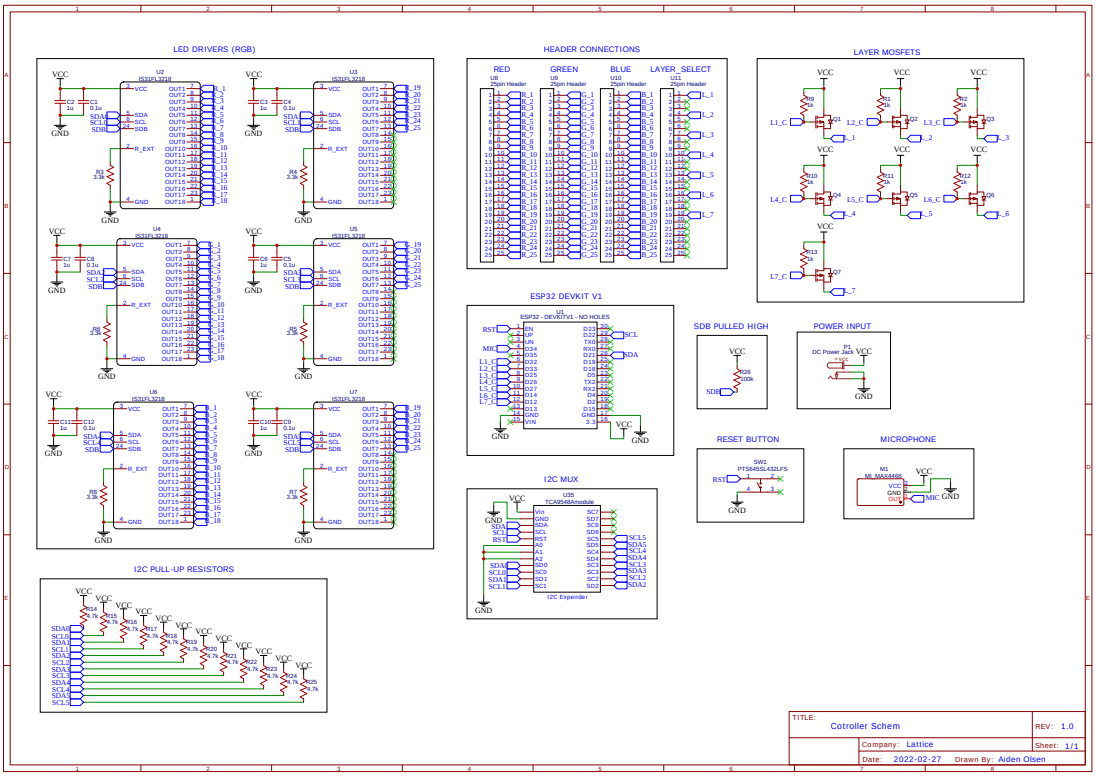
2.1 LED Interface Schematic

The LED Interface Schematic handles the connections for each anode layer as well as the connections between the header pins and LED leads.



2.2 Controller Board Schematic

The Controller Board deals with the ESP32 micro controller's interfaces with the I2C IS31FL3218 LED driver IC's.



3 Code

3.1 PySimpleGUI Code

Code for controlling the controlling the led cube over wifi. Uses the requests library to make post requests of data, such as color, animation type, message and audio mode to the web server running on the ESP32.

```
#-------------------------------------------------------#
# Blake Wiker
# 02/11/2021
# LED Cube GUI using PySimpleGUI
#-------------------------------------------------------#

import PySimpleGUI as sg
import requests
from PIL import ImageColor

sg.theme('DarkBlack1') # Add Dark Theme

URL = "http://192.168.4.1/posttest" # URL for ESP32 Server

font = ('Arial', 15) # font type and size

# All the stuff inside your window.
layout = [[sg.Button('Random', button_color=('#983030'),
            size = (10,2), font = ('Arial', 15),
            key = 'animation1'),
```

```
            sg.Button('Rise', button_color=('#983030'),
             size = (10,2), font = ('Arial', 15),
             key = 'animation2'),
            sg.Button('Rain', button_color=('#983030'),
             size = (10,2), font = ('Arial', 15),
             key = 'animation3')],
           [sg.Text('')],
           [sg.Text('Enter Messsage', font = font),
            sg.InputText(font = font, size = (42,5))],
           [sg.Text('')],
           [sg.In("", visible=False, enable_events=True,
             key='set_line_color'),
            sg.ColorChooserButton('Color Picker',
             button_color=('#000000'), target='set_line_color',
             key='set_line_color_chooser',
             font = font, size = (10,2)),
            sg.Button('Audio Mode', button_color=('#375ddb'),
             key='audio_color', font = font, size = (10, 2)),
            sg.Button('Send', font = font, size = (10,2),
             button_color=('#1eb37c'))],
           [sg.Text('')]
          ]

# Create the Window
window = sg.Window('Lattice', layout)

# data json: stores color, animation, message, and audio mode
data = {'red': 255, 'green': 255, 'blue': 255, 'animation': 0,
        'message':'none', 'choice': 0, 'audio': 0};

# Event Loop to process "events" and get the "values" of the inputs
while True:
    event, values = window.read()
    if event == sg.WIN_CLOSED: # if user closes window
        break

    if event == 'animation1': # if animation1 button is clicked
       if data['animation'] != 1:
           data['animation'] = 1 # update json
           # update button colors to show selection
           window.find_element('animation1').Update(
                   button_color=('#983030','#ffffff'))
           window.find_element('animation2').Update(
                   button_color=('#ffffff','#983030'))
           window.find_element('animation3').Update(
                   button_color=('#ffffff','#983030'))

       else: # if already selected, deselect button
           data['animation'] = 0
           window.find_element('animation1').Update(
                   button_color=('#ffffff', '#983030'))

    if event == 'animation2': # if animation2 button is clicked
       if data['animation'] != 2:
```

```
            data['animation'] = 2 # update json
            # update button colors to indicate selection
            window.find_element('animation1').Update(
                    button_color=('#ffffff','#983030'))
            window.find_element('animation2').Update(
                    button_color=('#983030','#ffffff'))
            window.find_element('animation3').Update(
                    button_color=('#ffffff','#983030'))

        else: # if already selected, deselect button
            data['animation'] = 0
            window.find_element('animation2').Update(
                    button_color=('#ffffff', '#983030'))

    if event == 'animation3': # if animation3 button is clicked
        if data['animation'] != 3:
            data['animation'] = 3 # update json
            # update button colors to indicate selection
            window.find_element('animation1').Update(
                    button_color=('#ffffff','#983030'))
            window.find_element('animation2').Update(
                    button_color=('#ffffff','#983030'))
            window.find_element('animation3').Update(
                    button_color=('#983030','#ffffff'))

        else: # if already selected, deselect button
            data['animation'] = 0
            window.find_element('animation3').Update(
                    button_color=('#ffffff', '#983030'))

    if event == 'set_line_color': # if color picker is clicked
         if values[event] == 'None': # if no color selected
            values[event] = '#000000' # default color is black

         #update button color to indicate selection and update json
         window['set_line_color_chooser'].Update(
                 button_color=('#ffffff', values[event]))
         RGB = ImageColor.getcolor(values[event], 'RGB')
         data['red'] = RGB[0]
         data['green'] = RGB[1]
         data['blue'] = RGB[2]

    if event == 'audio_color': # if audio button is clicked
        if data['audio'] == 0: # if not selected
            data['audio'] = 1  #update json and change color
            window.find_element('audio_color').Update(
                    button_color=('#375ddb','#ffffff'))
        else: # if selected, update json and change color
            data['audio'] = 0
            window.find_element('audio_color').Update(
                    button_color=('#ffffff', '#375ddb'))

    if event == 'Send': # if send button is clicked
         data['message'] = values[0]
```

```
            if data['message'] != '':
              data['message'] = "␣" + values[0] # update message json

            # if animation and message selected, popup error occurs
            if data['animation'] != 0 and data['message'] != '':
                sg.Popup("Cannot␣send␣animation␣and␣message",
                        title = "Error")

            # if no animation or message selected, popup error occurs
            elif data['animation'] == 0 and data['message'] == '' and data['audio'] == 0:
                sg.Popup("Choose␣animation,␣message␣or␣audio␣mode", title = "Error")

            # if animation or message selected with audio mode, popup error occurs
            elif (data['animation'] != 0 or data['message'] != '') and data['audio'] == 1:
                sg.Popup("Deselect␣animation␣or␣message␣for␣audio␣mode", title = "Error")

           # try to post json if no errors
            else:
                try:
                    requests.post(URL, json = data, timeout = 2)
                except: # error if not connected
                    sg.Popup("Not␣connected␣to␣Lattice", title = "Error")

window.close() # close window on exit
```

3.2 ESP32 Code

Code for the ESP32 which includes the web server that the GUI interfaces with and the code for controlling the mosfets for powering each layer. As well as the controls for updating the led drivers with the color data

3.2.1 Receiver.cpp - Server and Main Loop Code

```
/*
  Rui Santos
  Complete project details at
  https://RandomNerdTutorials.com/esp32-client-server-wi-fi/

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files.

  The above copyright notice and this permission notice shall be included in all
  copies or substantial portions of the Software.
*/

/*
  James Wilcock
  3D LED Cube driver code
  Code to take in input from the ESP32 webserver and
  translate it to colors and animations on the ledcube.
  Involves updating the LED driver registers on our pcb
  with the values from the ledarray, which is store of the values
  that should be lit up on the cube
*/
```

```
#include "audiomode.h"
#include "leddriver.h"
#include <Arduino.h>
#include <ArduinoJson.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <SparkFun_I2C_Mux_Arduino_Library.h>
#include <WiFi.h>
#include <Wire.h>
#include <pgmspace.h>
static QWIICMUX myMux; // creates object for i2c mux
static int ledarray[5][5][7][3];

// Set your access point network credentials
const char *ssid = "Lattice";
const char *password = "123456789";
static int rgb[] = {0, 0, 100};
static int animation = -1;
static int audiomode = 0;
static char usermessage[2000];
static int volume = 1;

DynamicJsonDocument doc(2048);
const char *PARAM_MESSAGE = "message";

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

void onRequest(AsyncWebServerRequest *request) {
    // Handle Unknown Request
    request->send(404);
}

void onBody(AsyncWebServerRequest *request, uint8_t *data, size_t len,
            size_t index, size_t total) {
    // Handle body
}

void onUpload(AsyncWebServerRequest *request, String filename, size_t index,
              uint8_t *data, size_t len, bool final) {
    Serial.print("OH␣yeah");
}

void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client,
             AwsEventType type, void *arg, uint8_t *data, size_t len) {
    // Handle WebSocket event
}

void setup() {

    myMux = initialSetUp();
    Serial.print("Setting␣AP␣(Access␣Point)   ");
    // Remove the password parameter, if you want the AP (Access Point) to be
    // open
    WiFi.softAP(ssid, password);

    // attach AsyncEventSource
    IPAddress IP = WiFi.softAPIP();
```

```
    Serial.print("AP␣IP␣address:␣");
    Serial.println(IP);

    server.on(
        "/posttest", HTTP_POST, [](AsyncWebServerRequest *request) {}, NULL,
        [](AsyncWebServerRequest *request, uint8_t *data, size_t len,
           size_t index, size_t total) {
            static char message[200];
            int count = 0;
            for (size_t i = 0; i < len; i++) {
                Serial.write(data[i]);
                message[i] = data[i];
                count = i;
            }
            message[count + 1] = '\0';
            Serial.println();
            Serial.println(message);
            DeserializationError error = deserializeJson(
                doc, message); // translates the json message into array

            // Test if parsing succeeds.
            if (error) {
                Serial.print(F("deserializeJson()␣failed:␣"));
                Serial.println(error.f_str());
                return;
            }

            int red = doc["red"];
            int blue = doc["blue"];
            int green = doc["green"];

            int choice = doc["animation"]; // gets which animation from json
            rgb[0] = red;
            rgb[1] = green;
            rgb[2] = blue;
            animation = choice;
            const char *word = doc["message"];
            audiomode = doc["audio"];
            strcpy(usermessage, word);
            // print json parsed data
            Serial.printf("color␣=␣%d,%d,%d,␣animation␣=␣%d,␣message␣=␣%s\n",
                          red, green, blue, choice, word);
            request->send(200);
        });

    initarray(&ledarray);

    // Start server
    animation = 3;
    server.begin();
}

void printdata() {
    Serial.printf(
        "color␣=␣%d,%d,%d,␣animation␣=␣%d,␣message␣=␣%s,audiomode␣=␣%d\n",
        rgb[0], rgb[1], rgb[2], animation, usermessage, audiomode);
}
```

```
void loop() {

    // loops animation randomly turing on and off leds
    while (animation == 1) {
        initarray(&ledarray);
        Serial.println("Doing␣anim␣1");
        while (animation == 1) {
            animation1(rgb, &ledarray);
            int time = millis();
            while (millis() - time < 10) {

                arraytolight(myMux, ledarray);
            }
        }
    }

    // loops animation of moving plane of leds
    // from bottom to top and then looping to bottom
    while (animation == 2) {
        initanim2(rgb, &ledarray);
        Serial.println("Doing␣anim␣2");
        while (animation == 2) {
            animation2(myMux, rgb, &ledarray);
            int time = millis();
            while (millis() - time < 200) {
                arraytolight(myMux, ledarray);
            }
        }
    }

    // loops animation of falling rain
    while (animation == 3) {
        initarray(&ledarray);
        Serial.println("Doing␣anim␣3");
        while (animation == 3) {
            dorain(rgb, &ledarray);
            int time = millis();
            while (millis() - time < 100) {
                arraytolight(myMux, ledarray);
            }
        }
    }

    // checks for if there is a message
    if (usermessage != NULL) {
        initarray(&ledarray);
        for (int x = 0; x <= strlen(usermessage); x++) {
            if (usermessage[x] != '\0') {
                // updates ledarray with letter data
                sendletter(usermessage[x], rgb, &ledarray);
            }
            int time = millis();
            // delay of 1 second to next letter
            while (millis() - time < 1000)
                arraytolight(myMux, ledarray);
        }
        // sets user message to null
        memset(usermessage, 0, strlen(usermessage));
```

```
    }

    // notes on the mic chip
    //  turning towards gnd is more gain
    // uses mic to generate respones in leds
    if (audiomode) {
        int slowest = 0;
        int prevvolume = -1;
        int samples[10];
        for (int i = 0; i < 10; i++) {
            samples[i] = 0;
        }
        int len = 0;
        int marker = 0;
        while (audiomode) {
            int time = micros();
            // gets volume from sampling  mic
            volume = getvolume(&marker, &len, &samples);
            if (prevvolume != volume) {
                // handles manipulation of ledarray based on volume
                prevvolume = handlevolume(volume, rgb, &ledarray);
            }
            arraytolight(myMux, ledarray);
            int timetocomplete = micros() - time;

            if (timetocomplete > slowest) {
                slowest = timetocomplete;
                Serial.printf("New␣slowest:␣%d\nFrequency␣=␣%f\n", slowest,
                              (1 / (timetocomplete * pow(10, -6))));
            }
        }
    }
}
```

3.2.2 LEDDriver.cpp - LED Driver Code

```
#include "leddriver.h"
#include <font.h>

// Initializes the ledarray to all zero
void initarray(int (*ledarray)[5][5][7][3]) {
    // initialize array
    for (int x = 0; x < 5; x++) {
        for (int y = 0; y < 5; y++) {
            for (int z = 0; z < 7; z++) {
                for (int color = 0; color < 3;
                     color++) {
                    (*ledarray)[x][y][z][color] =
                        0;
                }
            }
        }
    }
}

// function to get the correct letter from the
// font file and shift and update the ledarray
// accordingly
void sendletter(char letter, int color[3],
```

```
            int (*ledarray)[5][5][7][3]) {
// Want to first shift the array in order to
// move message Then assign the letter to the
// array using font.h

// shifting array in y direction
for (int y = 4; y > 0; y--) {
    for (int x = 0; x < 5; x++) {
        for (int z = 0; z < 7; z++) {
            for (int color = 0; color < 3;
                 color++) {
                (*ledarray)[x][y][z][color] =
                    (*ledarray)[x][y - 1][z]
                                [color];
            }
        }
    }
}

// setting first part to 0
for (int x = 0; x < 5; x++) {
    for (int z = 0; z < 7; z++) {
        for (int color = 0; color < 3;
             color++) {
            (*ledarray)[x][0][z][color] = 0;
        }
    }
}

int match = 0;
for (int l = 0; font[l].letter != 0; l++) {
    if (font[l].letter == letter) {
        match = l;
        break;
    }
}
for (int z = 0; z < 7; z++) {
    for (int x = 0; x < 5; x++) {
        if (font[match].code[z][x] == '#') {
            (*ledarray)[x][0][6 - z][0] =
                (color[0]);
            (*ledarray)[x][0][6 - z][1] =
                (color[1]);
            (*ledarray)[x][0][6 - z][2] =
                (color[2]);
        } else {
            (*ledarray)[x][0][6 - z][0] = 0;
            (*ledarray)[x][0][6 - z][1] = 0;
            (*ledarray)[x][0][6 - z][2] = 0;
        }
    }
}
for (int z = 0; z < 7; z++) {
    for (int y = 0; y < 5; y++) {
        for (int x = 0; x < 5; x++) {
            for (int color = 0; color < 3;
                 color++) {
                if (y != 0)
```

```
                        (*ledarray)[x][y][z]
                                  [color] = 0;
                }
            }
        }
    }
}

// code for animation 1 which is to turn on random
// leds and if they are on turn them off
void animation1(int rgb[3],
                int (*ledarray)[5][5][7][3]) {
    // turns random led on and turns leds that are
    // on off
    int randx = rand() % 5;
    int randy = rand() % 5;
    int randz = rand() % 7;
    int sum = 0;
    for (int color = 0; color < 3; color++) {
        sum +=
            *ledarray[randx][randy][randz][color];
    }
    int onoroff = sum > 0;
    for (int color = 0; color < 3; color++) {
        // if led on turn it off
        if (onoroff)
            (*ledarray)[randx][randy][randz]
                       [color] = 0;
        else
            (*ledarray)[randx][randy][randz]
                       [color] = rgb[color];
    }
}

// initializes the array for animation 2, setting
// a plane of leds on z=0 on with the user given
// color
void initanim2(int rgb[3],
               int (*ledarray)[5][5][7][3]) {
    initarray(ledarray);
    for (int x = 0; x < 5; x++) {
        for (int y = 0; y < 5; y++) {
            for (int color = 0; color < 3;
                 color++) {
                (*ledarray)[x][y][0][color] =
                    rgb[color];
            }
        }
    }
}

// code to move a plane of leds up the cube
// z=0->z=7 once the plane is moved up from 7 it
// appears on
//  the bottom to achieve a repeating effect
void animation2(QWIICMUX myMux, int rgb[3],
                int (*ledarray)[5][5][7][3]) {
    for (int x = 0; x < 5; x++) {
```

```
            for (int y = 0; y < 5; y++) {
                for (int z = 6; z > 0; z--) {
                    for (int color = 0; color < 3;
                         color++) {
                        (*ledarray)[x][y][z][color] =
                            (*ledarray)[x][y][z - 1]
                                       [color];
                    }
                }
            }
        }
        // clear bot layer
        for (int x = 0; x < 5; x++) {
            for (int y = 0; y < 5; y++) {
                for (int color = 0; color < 3;
                     color++) {
                    (*ledarray)[x][y][0][color] = 0;
                }
            }
        }
        int sum = 0;
        for (int x = 0; x < 5; x++) {
            for (int y = 0; y < 5; y++) {
                for (int z = 0; z < 7; z++) {
                    for (int color = 0; color < 3;
                         color++) {
                        sum += (*ledarray)[x][y][z]
                                          [color];
                    }
                }
            }
        }
        if (sum == 0) {
            initanim2(rgb, ledarray);
        }
    }

    // does the rain animation which consists of
    // randomly generating some "raindrops" on the top
    // layer each cycle and moving them down a layer,
    // if they are shifted down on the bottom they go
    // away.
    void dorain(int rgb[3],
                int (*ledarray)[5][5][7][3]) {
        // shift array down
        for (int z = 0; z < 6; z++) {
            for (int x = 0; x < 5; x++) {
                for (int y = 0; y < 5; y++) {
                    for (int color = 0; color < 3;
                         color++) {
                        (*ledarray)[x][y][z][color] =
                            (*ledarray)[x][y][z + 1]
                                       [color];
                    }
                }
            }
        }
        // clear top layer
```

```
    for (int x = 0; x < 5; x++) {
        for (int y = 0; y < 5; y++) {
            for (int color = 0; color < 3;
                 color++) {
                (*ledarray)[x][y][6][color] = 0;
            }
        }
    }
    // randomize number of "drops"
    int randDropNum = rand() % 7;
    // if randDropNum is 2 or less dont spawn a
    // drop
    if (randDropNum <= 2) {
        randDropNum = 0;
    }
    for (int drops = 0; drops < randDropNum - 2;
         drops++) // randDropNum - 2, so only up
                  // to 5 drops
    {
        int randx = rand() % 5;
        int randy = rand() % 5;
        for (int color = 0; color < 3; color++) {

            (*ledarray)[randx][randy][6][color] =
                rgb[color];
        }
    }
}

// takes in a given volume measurement from the
// audio driver code and updates the ledarray
// accordingly.
//  for volume 0 all leds off
//  for volume 1 middle led on
//  for volume 2 middle 9 on
//  for volume 3 all on
int handlevolume(int volume, int color[3],
                 int (*ledarray)[5][5][7][3]) {
    if (volume == 0) {
        // turn all off
        for (int x = 0; x < 5; x++) {
            for (int y = 0; y < 5; y++) {
                for (int z = 0; z < 7; z++) {
                    (*ledarray)[x][y][z][0] = 0;
                    (*ledarray)[x][y][z][1] = 0;
                    (*ledarray)[x][y][z][2] = 0;
                }
            }
        }
    } else if (volume == 1) {
        // turn middle on
        for (int x = 0; x < 5; x++) {
            for (int y = 0; y < 5; y++) {
                for (int z = 0; z < 7; z++) {
                    if (x == 2 && y == 2) {
                        (*ledarray)[x][y][z][0] =
                            color[0];
                        (*ledarray)[x][y][z][1] =
```

```
                            color[1];
                        (*ledarray)[x][y][z][2] =
                            color[2];
                    } else {
                        (*ledarray)[x][y][z][0] =
                            0;
                        (*ledarray)[x][y][z][1] =
                            0;
                        (*ledarray)[x][y][z][2] =
                            0;
                    }
                }
            }
        }
    } else if (volume == 2) {
        // turn 3x3 middle on
        for (int x = 0; x < 5; x++) {
            for (int y = 0; y < 5; y++) {
                for (int z = 0; z < 7; z++) {
                    if ((x >= 1 && x <= 3) &&
                        (y >= 1 && y <= 3)) {
                        (*ledarray)[x][y][z][0] =
                            color[0];
                        (*ledarray)[x][y][z][1] =
                            color[1];
                        (*ledarray)[x][y][z][2] =
                            color[2];
                    } else {
                        (*ledarray)[x][y][z][0] =
                            0;
                        (*ledarray)[x][y][z][1] =
                            0;
                        (*ledarray)[x][y][z][2] =
                            0;
                    }
                }
            }
        }
    } else if (volume == 3) {
        // all on
        for (int x = 0; x < 5; x++) {
            for (int y = 0; y < 5; y++) {
                for (int z = 0; z < 7; z++) {
                    (*ledarray)[x][y][z][0] =
                        color[0];
                    (*ledarray)[x][y][z][1] =
                        color[1];
                    (*ledarray)[x][y][z][2] =
                        color[2];
                }
            }
        }
    }
    return volume;
}

// does operations necessary for the setup of the
// program such as setting clock rate initializing
```

```
// the led driver registers
//  setting the mposfet pins for output
QWIICMUX initialSetUp() {
    QWIICMUX myMux;
    Serial.begin(115200);
    Serial.println();
    Serial.println("Setting up");
    Wire.begin();
    Wire.setClock(1000000); // I2C 800kHz

    if (myMux.begin() == false) {
        Serial.println(
            "Mux not detected. Freezing...");
    }
    Serial.println("Mux detected");

    byte currentPortNumber = 0;
    for (int i = 0; i < 8; i++) {
        Serial.printf("Initializing board %d\n",
                      i);
        myMux.setPort(i);
        currentPortNumber = myMux.getPort();
        Serial.print("CurrentPort: ");
        Serial.println(currentPortNumber);
        WriteLedDriverByte(0x17, 0x00); // reset
        WriteLedDriverByte(0x00, 0x01); // enable
        WriteLedDriverByte(0x13,
                           0x3f); // enable leds
        WriteLedDriverByte(0x14,
                           0x3f); // enable leds 2
        WriteLedDriverByte(0x15,
                           0x3f); // enable led 3
    }

    // TODO need to change 2 to 7 for all layers
    for (int mosfet = 0; mosfet < 7; mosfet++) {
        pinMode(moslist[mosfet], OUTPUT);
        digitalWrite(moslist[mosfet], LOW);
    }
    return myMux;
}

// writes a given register with the given data on
// the led driver
void WriteLedDriverByte(uint8_t Reg_Add,
                        uint8_t Reg_Dat) {
    Wire.beginTransmission(0xA8 /
                           2); //(MUX_Address/2);
    Wire.write(Reg_Add);        // sends regaddress
    Wire.write(Reg_Dat);        // sends regaddress
    Wire.endTransmission(); // stop transmitting
}

// translates the ledarray into updating registers
// on the led drivers, does layer by layer by
// selecting a single mosfet on to power only 1
// layer.
void arraytolight(QWIICMUX myMux,
```

```
                int ledarray[5][5][7][3]) {
    // 6 led drivers
    // 0,1 red
    // 2,3 green
    // 4,5 blue
    int ledcount = 0;
    for (int z = 0; z < 7; z++) {
        selectLayer(z);
        for (int color = 0; color < 3; color++) {
            myMux.setPort(color * 2);
            // tcaset(color * 2);
            //   int currport = myMux.getPort();
            //   Serial.println(currport);
            Wire.beginTransmission(0xA8 / 2);
            ledcount = 1;
            Wire.write(1);
            for (int y = 0; y < 5; y++) {
                for (int x = 0; x < 5; x++) {
                    if (ledcount == 19) {
                        Wire.endTransmission();
                        WriteLedDriverByte(0x16,
                                           00);
                        myMux.setPort(color * 2 +
                                      1);
                        // tcaset(color * 2 + 1);
                        //   currport =
                        //   myMux.getPort();
                        //   Serial.println(currport);
                        Wire.beginTransmission(
                            0xA8 / 2);
                        Wire.write(1);
                    }
                    Wire.write(
                        ledarray[x][y][z][color]);
                    ledcount++;
                }
            }
            Wire.endTransmission();
            WriteLedDriverByte(0x16, 00);
        }
        // resetting registers after selecting new
        // layer set all ports on
        tcaallon();
        WriteLedDriverByte(0x17, 00);
        WriteLedDriverByte(0x00, 0x01); // enable
        Wire.beginTransmission(0xA8 / 2);
        Wire.write(0x13);
        Wire.write(0x3f);
        Wire.write(0x3f);
        Wire.write(0x3f);
        Wire.endTransmission();
    }
}

// sets a specific channel on the i2c mux on
void tcaset(int channel) {
    Wire.beginTransmission(0x70);
    Wire.write(1 << channel);
```

```
    Wire.endTransmission();
}

// sets all i2c mux channels on
void tcaallon() {
    Wire.beginTransmission(0x70);
    Wire.write(0xFF);
    Wire.endTransmission();
}

// turns "on" a single mosfet to only power a
// single layer mosfets are active low as they are
// PMOS
void selectLayer(int layernum) {

    for (int x = 0; x < 7; x++) {
        if (x == layernum)
            digitalWrite(moslist[x], LOW);
        else
            digitalWrite(moslist[x], HIGH);
    }
}
```

3.2.3 LEDDriver.h - LED Driver Header

```
#ifndef LEDDRIVER_H
#define LEDDRIVER_H

#include <Arduino.h>
#include <SparkFun_I2C_Mux_Arduino_Library.h>
#include <Wire.h>
#include <stdlib.h>
// TODO CHANGE PINS FOR PCB
#define L_0 32
#define L_1 33
#define L_2 25
#define L_3 26
#define L_4 27
#define L_5 14
#define L_6 13
#define FX_SPEED 50
static int moslist[] = {L_0, L_1, L_2, L_3,
                        L_4, L_5, L_6};

void WriteLedDriverByte(uint8_t Reg_Add,
                        uint8_t Reg_Dat);

QWIICMUX initialSetUp();
void sendletter(char letter, int color[3],
                int (*ledarray)[5][5][7][3]);
void initarray(int (*ledarray)[5][5][7][3]);
void animation1(int rgb[3],
                int (*ledarray)[5][5][7][3]);
void animation2(QWIICMUX myMux, int rgb[3],
                int (*ledarray)[5][5][7][3]);
void initanim2(int rgb[3],
               int (*ledarray)[5][5][7][3]);
void multiplexLayers();
void clearLayer();
```

```
void tcaallon();
void tcaset(int channel);
void setallon(QWIICMUX myMux);
void setalloff(QWIICMUX myMux);
void selectLayer(int layernum);
void dorain(int rgb[3],
            int (*ledarray)[5][5][7][3]);
int handlevolume(int volume, int color[3],
                 int (*ledarray)[5][5][7][3]);
void arraytolight(QWIICMUX myMux,
                  int ledarray[5][5][7][3]);
#endif
```

3.2.4 Audiomode.cpp - Audio Mode Code

```
#include "audiomode.h"
// green is out
// brown is GND
// Red is vcc
int getvolume(int *marker, int *len,
              int (*samples)[samplenum]) {
    float normalized = 0;
    int count = 0;
    int sum = 0;
    int sensorValue = 0;
    int time = micros();
    sensorValue = analogRead(sensorPin);
    // checks if sensor value is loud enough to be
    // greater than noise
    if (sensorValue > BASELINE - NOISE &&
        sensorValue < BASELINE + NOISE) {
        sensorValue = 0;
    } else {
        if (sensorValue > BASELINE)
            sensorValue =
                sensorValue - BASELINE - NOISE;
        else
            sensorValue =
                (BASELINE - sensorValue) - NOISE;
    }

    // checks len of sample array
    // if less than max then just set the current
    // sample if the array has been filled keep a
    // marker of the current element and start
    // updating the array from 0 incrementing each
    // time until max length
    if ((*len) < samplenum) {
        (*samples)[*marker] = sensorValue;
        (*len)++;
        (*marker)++;
    } else if ((*len) == samplenum) {
        if (*marker == samplenum) {
            *marker = 0;
        }
        (*samples)[*marker] = sensorValue;
        (*marker)++;
    }
```

```
    if (*len == 0) {
        count = 1;
    } else {
        count = (*len);
    }
    for (int i = 0; i < *len; i++) {
        sum += (*samples)[i];
    }

    // 1750 seems to be the range at which the mic
    // sits without stimulation
    normalized =
        abs((float(sum / count)) / (1750));
    int volume =
        int(normalized * 100); // 0-100 volume
    int level = 0;
    if (volume < 20) {
        level = 0;
    } else if (volume < 25) {
        level = 1;
        // turn on [2][2]
    } else if (volume < 30) {
        level = 2;
        // turn on [1-3][1-3]
    } else {
        level = 3;
        // turn all on
    }
    return level;
}
```

3.2.5 Audiomode.h - Audio Mode Header

```
#ifndef AUDIOMODE_H
#define AUDIOMODE_H
#define samplenum 10
#define NOISE 200
#define BASELINE 1950
#include <Arduino.h>
#define sensorPin 34
int getvolume(int *marker, int *len, int (*samples)[samplenum]);
#endif
```

3.2.6 Font.h - Font code

```
/*
MIT License

Copyright (c) 2020 Petabyte

Permission is hereby granted, free of charge, to
any person obtaining a copy of this software and
associated documentation files (the "Software"),
to deal in the Software without restriction,
including without limitation the rights to use,
copy, modify, merge, publish, distribute,
sublicense, and/or sell copies of the Software,
and to permit persons to whom the Software is
furnished to do so, subject to the following
conditions:
```

```
The above copyright notice and this permission
notice shall be included in all copies or
substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY
OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT
LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES
OR OTHER LIABILITY, WHETHER IN AN ACTION OF
CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

    This file is C89.

    Compiler command line options:
    -DNO_LOWERCASE
    -DNO_SYMBOL
    -DNO_NUMBERS
*/

// font file to take and translate into the
// ledarray
#ifndef FONT_H
#define FONT_H

struct Font {
    char letter;
    char code[7][6];
};

struct Font font[] = {
    {'␣',
     {/* Processor should ignore this */
      "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣",
      "␣␣␣␣␣", "␣␣␣␣␣"}},
    {'A',
     {"␣###␣", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "#####",
      "#␣␣␣#", "#␣␣␣#"}},
    {'B',
     {"####␣", "#␣␣␣#", "#␣␣␣#", "####␣", "#␣␣␣#",
      "#␣␣␣#", "####␣"}},
    {'C',
     {"␣####", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣",
      "#␣␣␣␣", "␣####"}},
    {'D',
     {"####␣", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#",
      "#␣␣␣#", "####␣"}},
    {'E',
     {"#####", "#␣␣␣␣", "#␣␣␣␣", "#####", "#␣␣␣␣",
      "#␣␣␣␣", "#####"}},
    {'F',
     {"#####", "#␣␣␣␣", "#␣␣␣␣", "#####", "#␣␣␣␣",
      "#␣␣␣␣", "#␣␣␣␣"}},
    {'G',
```

```
 {"␣####", "#␣␣␣␣", "#␣␣␣␣", "#␣␣##", "#␣␣␣#",
  "#␣␣␣#", "␣####"}},
{'H',
 {"#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "#####", "#␣␣␣#",
  "#␣␣␣#", "#␣␣␣#"}},
{'I',
 {"#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣",
  "#␣␣␣␣", "#␣␣␣␣"}},
{'J',
 {"␣␣␣␣#", "␣␣␣␣#", "␣␣␣␣#", "␣␣␣␣#", "␣␣␣␣#",
  "␣␣␣␣#", "####␣"}},
{'K',
 {"#␣␣␣#", "#␣␣#␣", "#␣#␣␣", "##␣␣␣", "#␣#␣␣",
  "#␣␣#␣", "#␣␣␣#"}},
{'L',
 {"#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣",
  "#␣␣␣␣", "#####"}},
{'M',
 {"#␣␣␣#", "##␣##", "#␣#␣#", "#␣␣␣#", "#␣␣␣#",
  "#␣␣␣#", "#␣␣␣#"}},
{'N',
 {"#␣␣␣#", "##␣␣#", "#␣#␣#", "#␣␣##", "#␣␣␣#",
  "#␣␣␣#", "#␣␣␣#"}},
{'O',
 {"␣###␣", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#",
  "#␣␣␣#", "␣###␣"}},
{'P',
 {"####␣", "#␣␣␣#", "#␣␣␣#", "####␣", "#␣␣␣␣",
  "#␣␣␣␣", "#␣␣␣␣"}},
{'Q',
 {"␣###␣", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#",
  "#␣␣##", "␣####"}},
{'R',
 {"####␣", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "####␣",
  "#␣␣␣#", "#␣␣␣#"}},
{'S',
 {"␣####", "#␣␣␣␣", "#␣␣␣␣", "␣###␣", "␣␣␣␣#",
  "␣␣␣␣#", "####␣"}},
{'T',
 {"#####", "␣␣#␣␣", "␣␣#␣␣", "␣␣#␣␣", "␣␣#␣␣",
  "␣␣#␣␣", "␣␣#␣␣"}},
{'U',
 {"#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#",
  "#␣␣␣#", "␣###␣"}},
{'V',
 {"#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#",
  "␣#␣#␣", "␣␣#␣␣"}},
{'W',
 {"#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "#␣#␣#",
  "##␣##", "#␣␣␣#"}},
{'X',
 {"#␣␣␣#", "#␣␣␣#", "␣#␣#␣", "␣␣#␣␣", "␣#␣#␣",
  "#␣␣␣#", "#␣␣␣#"}},
{'Y',
 {"#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "␣###␣", "␣␣#␣␣",
  "␣␣#␣␣", "␣␣#␣␣"}},
{'Z',
 {"#####", "␣␣␣␣#", "␣␣␣#␣", "␣␣#␣␣", "␣#␣␣␣",
```

```
      "#␣␣␣␣", "#####"}},
#ifndef NO_LOWERCASE
    {'a',
     {"␣␣␣␣␣", "␣␣␣␣␣", "␣###␣", "␣␣␣␣#", "␣####",
      "#␣␣␣#", "␣####"}},
    {'b',
     {"#␣␣␣␣", "#␣␣␣␣", "####␣", "#␣␣␣#", "#␣␣␣#",
      "#␣␣␣#", "####␣"}},
    {'c',
     {"␣␣␣␣␣", "␣␣␣␣␣", "␣###␣", "#␣␣␣#", "#␣␣␣␣",
      "#␣␣␣#", "␣###␣"}},
    {'d',
     {"␣␣␣␣#", "␣␣␣␣#", "␣####", "#␣␣␣#", "#␣␣␣#",
      "#␣␣␣#", "␣####"}},
    {'e',
     {"␣␣␣␣␣", "␣␣␣␣␣", "␣###␣", "#␣␣␣#", "#####",
      "#␣␣␣␣", "␣####"}},
    {'f',
     {"␣␣##␣", "␣#␣␣#", "␣#␣␣␣", "###␣␣", "␣#␣␣␣",
      "␣#␣␣␣", "␣#␣␣␣"}},
    {'g',
     {"␣␣␣␣␣", "␣####", "#␣␣␣#", "#␣␣␣#", "␣####",
      "␣␣␣␣#", "####␣"}},
    {'h',
     {"#␣␣␣␣", "#␣␣␣␣", "####␣", "#␣␣␣#", "#␣␣␣#",
      "#␣␣␣#", "#␣␣␣#"}},
    {'i',
     {"#␣␣␣␣", "␣␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣",
      "#␣␣␣␣", "#␣␣␣␣"}},
    {'j',
     {"␣␣␣␣#", "␣␣␣␣␣", "␣␣␣##", "␣␣␣␣#", "␣␣␣␣#",
      "#␣␣␣#", "␣###␣"}},
    {'k',
     {"#␣␣␣␣", "#␣␣␣␣", "#␣␣␣#", "#␣␣#␣", "###␣␣",
      "#␣␣#␣", "#␣␣␣#"}},
    {'l',
     {"#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣",
      "#␣␣␣␣", "##␣␣␣"}},
    {'m',
     {"␣␣␣␣␣", "␣␣␣␣␣", "####␣", "#␣#␣#", "#␣#␣#",
      "#␣#␣#", "#␣#␣#"}},
    {'n',
     {"␣␣␣␣␣", "␣␣␣␣␣", "####␣", "#␣␣␣#", "#␣␣␣#",
      "#␣␣␣#", "#␣␣␣#"}},
    {'o',
     {"␣␣␣␣␣", "␣␣␣␣␣", "␣###␣", "#␣␣␣#", "#␣␣␣#",
      "#␣␣␣#", "␣###␣"}},
    {'p',
     {"␣␣␣␣␣", "␣###␣", "␣#␣␣#", "␣#␣␣#", "␣###␣",
      "␣#␣␣␣", "␣#␣␣␣"}},
    {'q',
     {"␣␣␣␣␣", "␣␣␣␣␣", "␣␣###", "␣#␣␣#", "␣␣###",
      "␣␣␣␣#", "␣␣␣␣#"}},
    {'r',
     {"␣␣␣␣␣", "␣␣␣␣␣", "#␣###", "##␣␣␣", "#␣␣␣␣",
      "#␣␣␣␣", "#␣␣␣␣"}},
    {'s',
     {"␣␣␣␣␣", "␣␣␣␣␣", "␣####", "#␣␣␣␣", "␣###␣",
```

```
      "␣␣␣␣#", "####␣"}},
    {'t',
     {"␣#␣␣␣", "␣#␣␣␣", "###␣␣", "␣#␣␣␣", "␣#␣␣␣",
      "␣#␣␣␣", "␣␣##␣"}},
    {'u',
     {"␣␣␣␣␣", "␣␣␣␣␣", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#",
      "#␣␣␣#", "␣###␣"}},
    {'v',
     {"␣␣␣␣␣", "␣␣␣␣␣", "#␣␣␣#", "#␣␣␣#", "#␣␣␣#",
      "␣#␣#␣", "␣␣#␣␣"}},
    {'w',
     {"␣␣␣␣␣", "␣␣␣␣␣", "#␣␣␣#", "#␣␣␣#", "#␣#␣#",
      "#␣#␣#", "␣#␣#␣"}},
    {'x',
     {"␣␣␣␣␣", "␣␣␣␣␣", "#␣␣␣#", "␣#␣#␣", "␣␣#␣␣",
      "␣#␣#␣", "#␣␣␣#"}},
    {'y',
     {"␣␣␣␣␣", "␣␣␣␣␣", "#␣␣␣#", "#␣␣␣#", "␣####",
      "␣␣␣␣#", "####␣"}},
    {'z',
     {"␣␣␣␣␣", "␣␣␣␣␣", "#####", "␣␣␣#␣", "␣␣#␣␣",
      "␣#␣␣␣", "#####"}},
#endif
#ifndef NO_NUMBERS
    {'0',
     {"␣###␣", "#␣␣␣#", "#␣␣##", "#␣#␣#", "##␣␣#",
      "#␣␣␣#", "␣###␣"}},
    {'1',
     {"##␣␣␣", "␣#␣␣␣", "␣#␣␣␣", "␣#␣␣␣", "␣#␣␣␣",
      "␣#␣␣␣", "␣#␣␣␣"}},
    {'2',
     {"␣###␣", "#␣␣␣#", "␣␣␣␣#", "␣␣##␣", "␣#␣␣␣",
      "#␣␣␣␣", "#####"}},
    {'3',
     {"####␣", "␣␣␣␣#", "␣␣␣␣#", "␣###␣", "␣␣␣␣#",
      "␣␣␣␣#", "####␣"}},
    {'4',
     {"#␣␣␣#", "#␣␣␣#", "#␣␣␣#", "#####", "␣␣␣␣#",
      "␣␣␣␣#", "␣␣␣␣#"}},
    {'5',
     {"#####", "#␣␣␣␣", "#␣␣␣␣", "####␣", "␣␣␣␣#",
      "␣␣␣␣#", "####␣"}},
    {'6',
     {"␣####", "#␣␣␣␣", "#␣␣␣␣", "␣###␣", "#␣␣␣#",
      "#␣␣␣#", "␣###␣"}},
    {'7',
     {"#####", "␣␣␣␣#", "␣␣␣␣#", "␣␣##␣", "␣#␣␣␣",
      "␣#␣␣␣", "␣#␣␣␣"}},
    {'8',
     {"␣###␣", "#␣␣␣#", "#␣␣␣#", "␣###␣", "#␣␣␣#",
      "#␣␣␣#", "␣###␣"}},
    {'9',
     {"␣###␣", "#␣␣␣#", "#␣␣␣#", "␣####", "␣␣␣␣#",
      "␣␣␣␣#", "␣###␣"}},
#endif
#ifndef NO_SYMBOLS
    {'!',
     {"#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣",
```

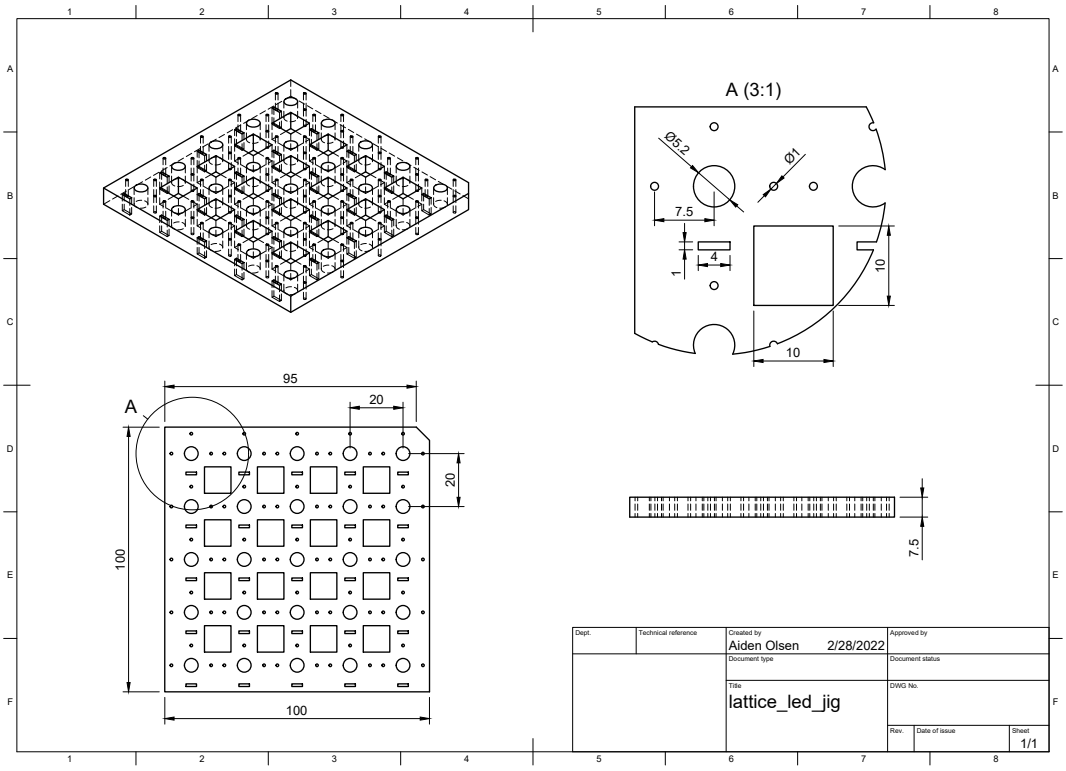
```
  "␣␣␣␣␣", "#␣␣␣␣"}},
{'.',
 {"␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣",
  "␣␣␣␣␣", "#␣␣␣␣"}},
{',',
 {"␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣",
  "␣␣#␣␣", "␣#␣␣␣"}},
{'?',
 {"␣##␣␣", "#␣␣#␣", "␣␣␣#␣", "␣␣#␣␣", "␣#␣␣␣",
  "␣␣␣␣␣", "␣#␣␣␣"}},
{1,
 {"␣␣␣␣␣", "␣#␣#␣", "␣#␣#␣", "␣␣␣␣␣", "␣␣#␣␣",
  "#␣␣␣#", "␣###␣"}},
{'%',
 {"␣␣␣␣#", "␣#␣␣#", "␣␣␣#␣", "␣␣#␣␣", "␣#␣␣␣",
  "#␣␣#␣", "#␣␣␣␣"}},
{'#',
 {"␣#␣#␣", "␣#␣#␣", "#####", "␣#␣#␣", "#####",
  "␣#␣#␣", "␣#␣#␣"}},
{'_',
 {"␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣",
  "␣␣␣␣␣", "#####"}},
{'-',
 {"␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣###␣", "␣␣␣␣␣",
  "␣␣␣␣␣", "␣␣␣␣␣"}},
{';',
 {"␣␣␣␣␣", "␣#␣␣␣", "␣␣␣␣␣", "␣#␣␣␣", "␣#␣␣␣",
  "#␣␣␣␣", "␣␣␣␣␣"}},
{'`',
 {"␣#␣␣␣", "␣#␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣",
  "␣␣␣␣␣", "␣␣␣␣␣"}},
{'=',
 {"␣␣␣␣␣", "#####", "␣␣␣␣␣", "␣␣␣␣␣", "#####",
  "␣␣␣␣␣", "␣␣␣␣␣"}},
{'_',
 {"␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣",
  "␣␣␣␣␣", "#####"}},
{':',
 {"␣␣␣␣␣", "␣#␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣#␣␣␣",
  "␣␣␣␣␣", "␣␣␣␣␣"}},
{'<',
 {"␣␣␣#␣", "␣␣#␣␣", "␣#␣␣␣", "#␣␣␣␣", "␣#␣␣␣",
  "␣␣#␣␣", "␣␣␣#␣"}},
{'>',
 {"␣#␣␣␣", "␣␣#␣␣", "␣␣␣#␣", "␣␣␣␣#", "␣␣␣#␣",
  "␣␣#␣␣", "␣#␣␣␣"}},
{'~',
 {"␣␣␣␣␣", "␣␣␣␣␣", "#␣#␣#", "␣#␣#␣", "␣␣␣␣␣",
  "␣␣␣␣␣", "␣␣␣␣␣"}},
{'*',
 {"␣␣#␣␣", "␣###␣", "␣␣#␣␣", "␣#␣#␣", "␣␣␣␣␣",
  "␣␣␣␣␣", "␣␣␣␣␣"}},
{'/',
 {"␣␣␣␣#", "␣␣␣␣#", "␣␣␣#␣", "␣␣#␣␣", "␣#␣␣␣",
  "␣#␣␣␣", "#␣␣␣␣"}},
{'\'',
 {"#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣",
  "␣␣␣␣␣", "␣␣␣␣␣"}},
```

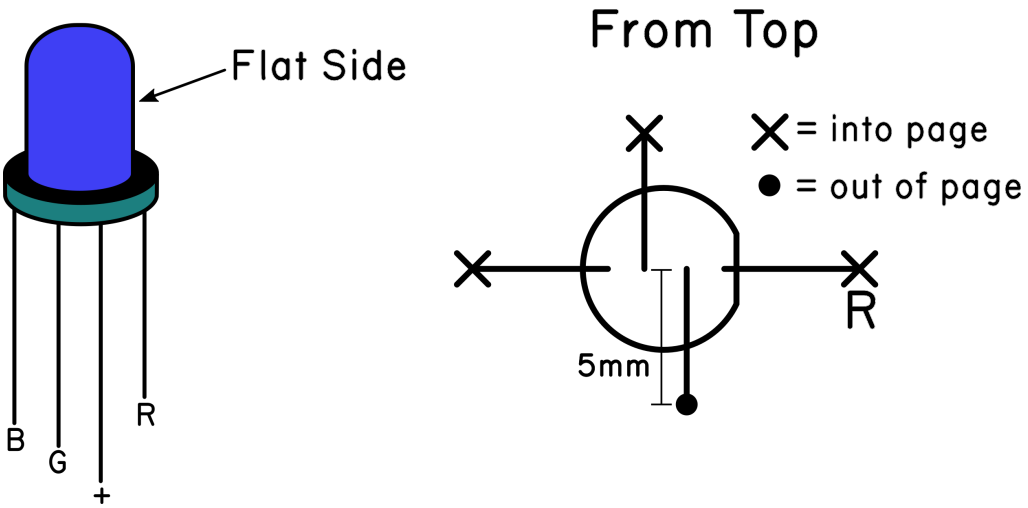
```
    {'"',
     {"#␣#␣␣", "#␣#␣␣", "#␣#␣␣", "␣␣␣␣␣", "␣␣␣␣␣",
      "␣␣␣␣␣", "␣␣␣␣␣"}},
    {'(',
     {"␣␣#␣␣", "␣#␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣",
      "␣#␣␣␣", "␣␣#␣␣"}},
    {')',
     {"#␣␣␣␣", "␣#␣␣␣", "␣␣#␣␣", "␣␣#␣␣", "␣␣#␣␣",
      "␣#␣␣␣", "#␣␣␣␣"}},
    {'}',
     {"#␣␣␣␣", "␣#␣␣␣", "␣#␣␣␣", "␣␣#␣␣", "␣#␣␣␣",
      "␣#␣␣␣", "#␣␣␣␣"}},
    {'{',
     {"␣␣#␣␣", "␣#␣␣␣", "␣#␣␣␣", "#␣␣␣␣", "␣#␣␣␣",
      "␣#␣␣␣", "␣␣#␣␣"}},
    {'+',
     {"␣␣␣␣␣", "␣␣␣␣␣", "␣␣#␣␣", "␣␣#␣␣", "#####",
      "␣␣#␣␣", "␣␣#␣␣"}},
    {'-',
     {"␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "␣␣␣␣␣", "#####",
      "␣␣␣␣␣", "␣␣␣␣␣"}},
#endif

#ifdef EXTENDED_ASCII
#define FNT_AN "\xa4"
    {0xa4,
     {"####␣", "␣␣␣␣␣", "####␣", "#␣␣␣#", "#␣␣␣#",
      "#␣␣␣#", "#␣␣␣#"}},
#define FNT_AI "\xad"
    {0xad,
     {"#␣␣␣␣", "␣␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣",
      "#␣␣␣␣", "#␣␣␣␣"}},
#define FNT_AQST "\xa8"
    {0xa8,
     {"␣#␣␣␣", "␣␣␣␣␣", "␣#␣␣␣", "␣␣#␣␣", "␣␣␣#␣",
      "#␣␣#␣", "␣##␣␣"}},
#define FNT_AEX "\xa1"
    {0xa1,
     {"##␣␣␣", "␣␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣", "#␣␣␣␣",
      "#␣␣␣␣", "#␣␣␣␣"}},
#endif

    /* Fallback/end Char. If you don't know the
    font size, use this as the "null terminator"
  */
    {0,
     {"#####", "#####", "#####", "#####", "#####",
      "#####", "#####"}},
};

#endif
```

4 Mechanical Drawings

4.1 LED Bending JIG

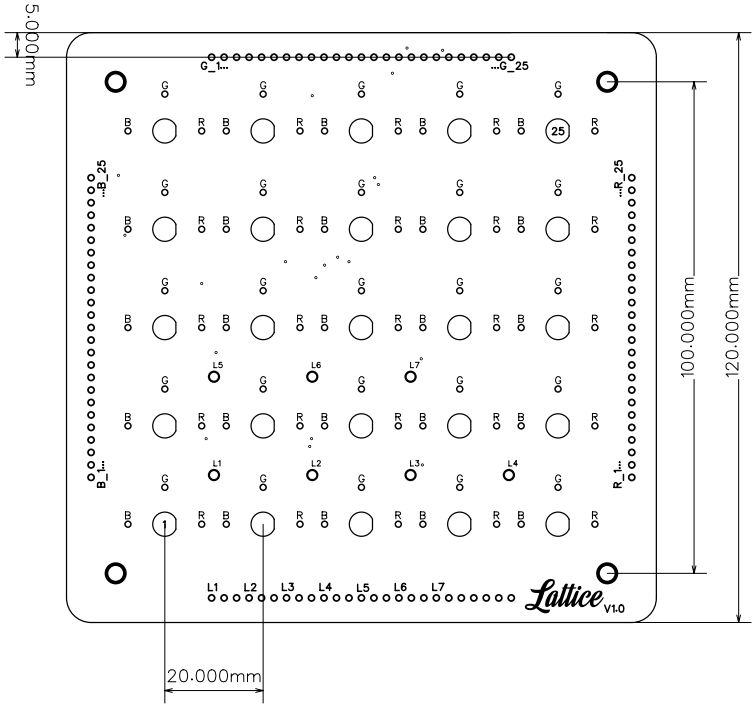
The LED Bending JIG is necessary during the construction of the led cube. The JIG allows for precise bends leading to a nicer looking construction.



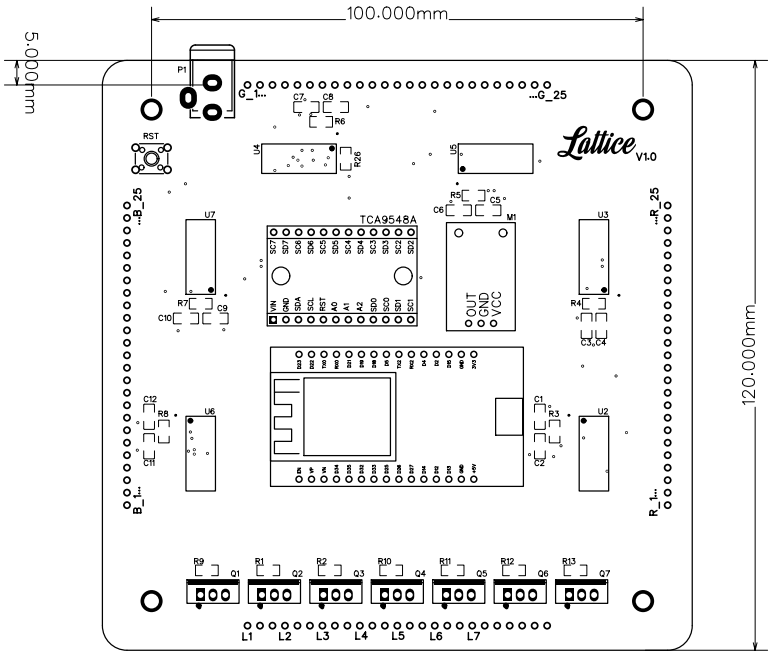


4.2 PCB Dimensions

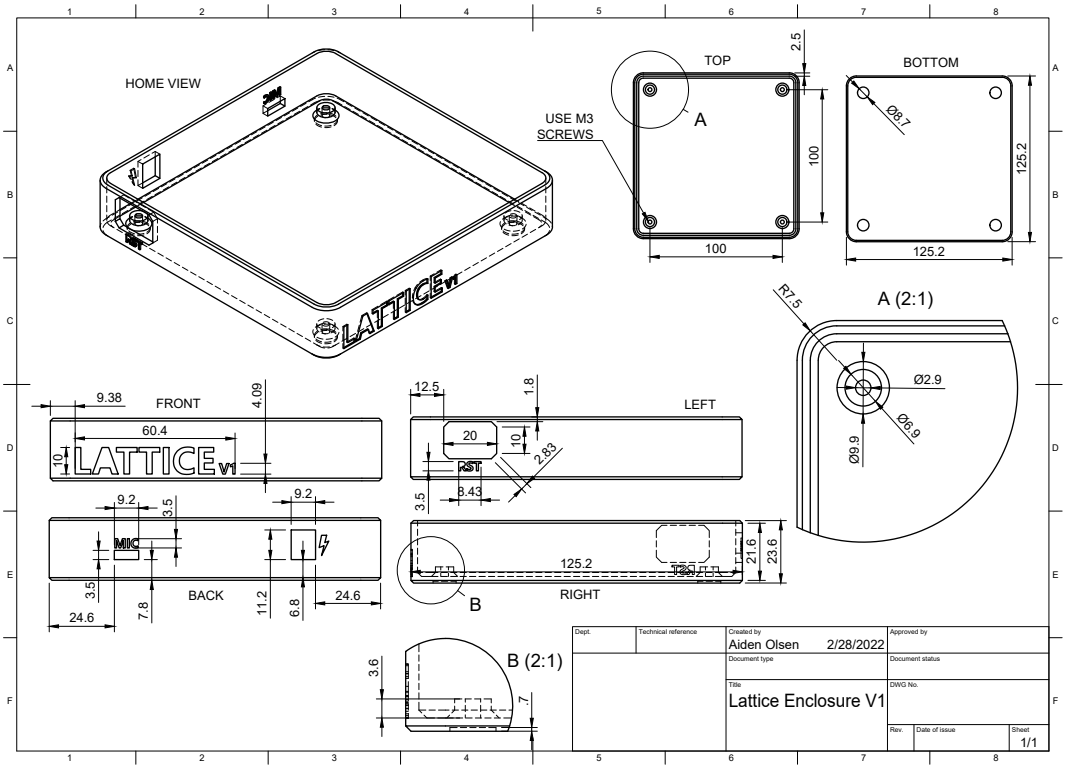
4.2.1 LED Interface Board



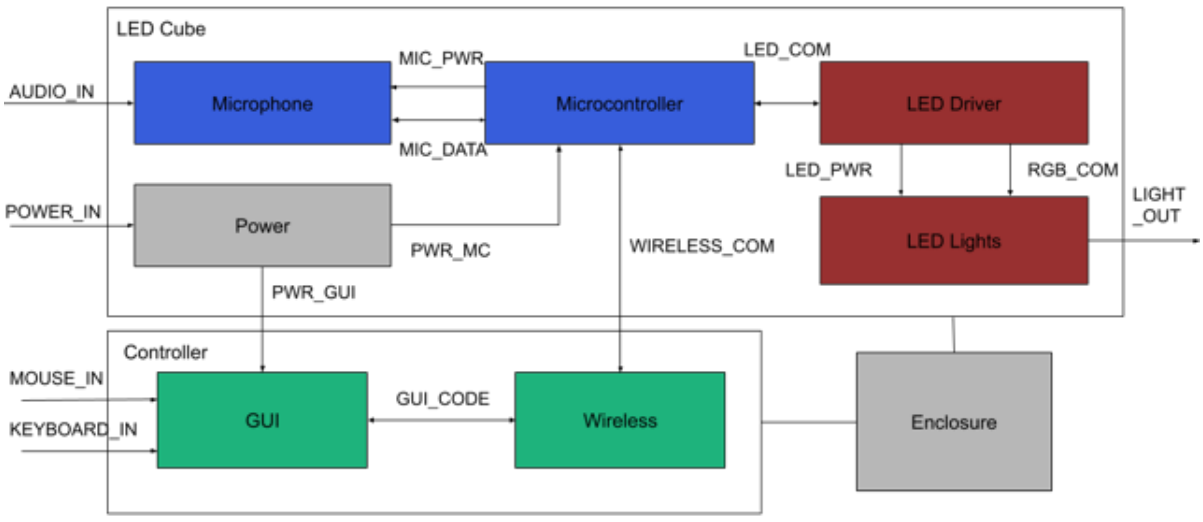
4.2.2 LED Controller Board



4.3 Enclosure Dimensions



5 Top Level Block Diagram



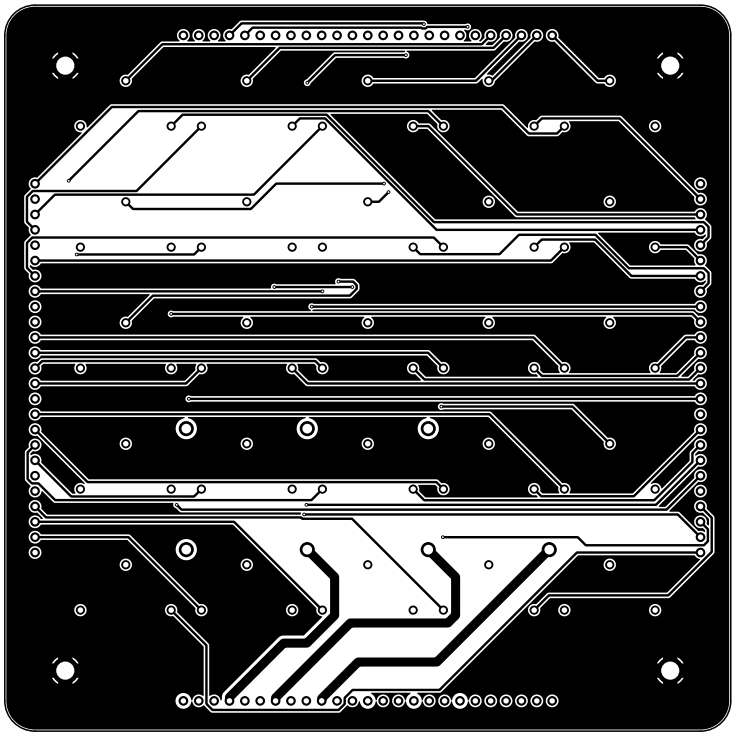
6 Interfaces

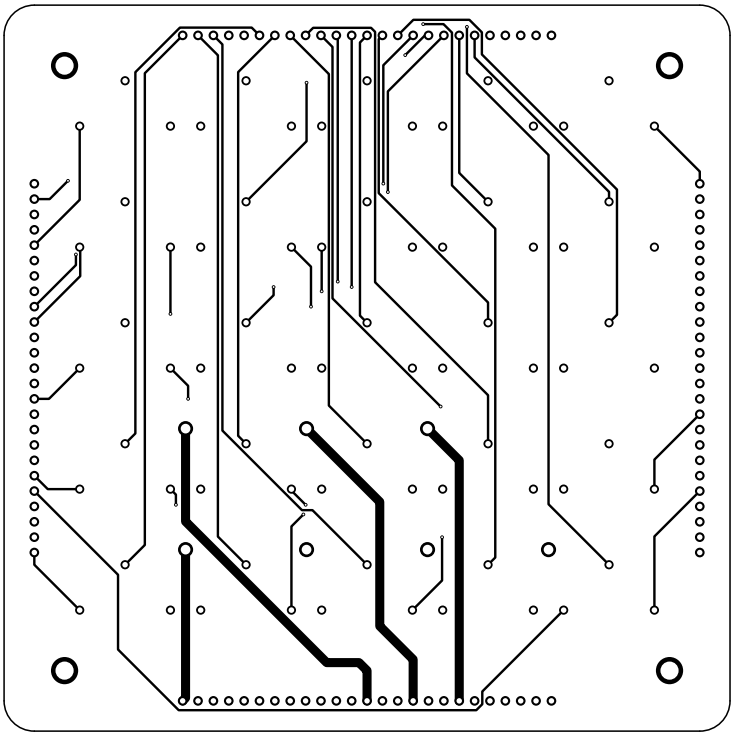
Interface Definitions

Interface Type	Interface Specification (See next page)
POWER_IN	AC Power • Vnom: 120VAC • Other: NEMA 5-15R • Ipeak: 15A • Inominal: 500mA
PWR_MC	USB Power
PWR_GUI	USB Power • Vmax: 5.25V • Vmin: 4.4V • Ipeak: 500mA • Inominal: 100mA
LED_COM	I2C communication
LED_PWR	Vmax: 5V Inom per layer: 1.5 A
MIC_DATA	0V to 5V peak to peak analog signal 25x to 125x adjustable gain
MIC_PWR	Vmax: 3.3V Imax: 40mA
RGB_COM	PWM (0-255) digital signal One connection for each color
LIGHT_OUT	Light waves from the LED Cube
AUDIO_IN	Represents the input of audio into the Microphone, this could be sound in the room or music playing.
MOUSE_IN	User uses mouse or trackpad to move cursor and select GUI buttons, cursor can also be used to manipulate text
KEYBOARD_IN	User uses keyboard to type message in the message window on the GUI
WIRELESS_COM	Server/Receiver HTTP request connection between computer and ESP32, data is sent as a json struct (see images below) ESP32 Wifi Specs: IEEE 802.11ax (Wi-Fi 6) on 2.4 GHz, supporting 20 MHz bandwidth in 11ax mode, speeds up to 150 Mbps.
GUI_CODE	This code defines what the GUI will look like and what signals it will send to the microcontroller. Written using the pysimplegui libraries and functions.

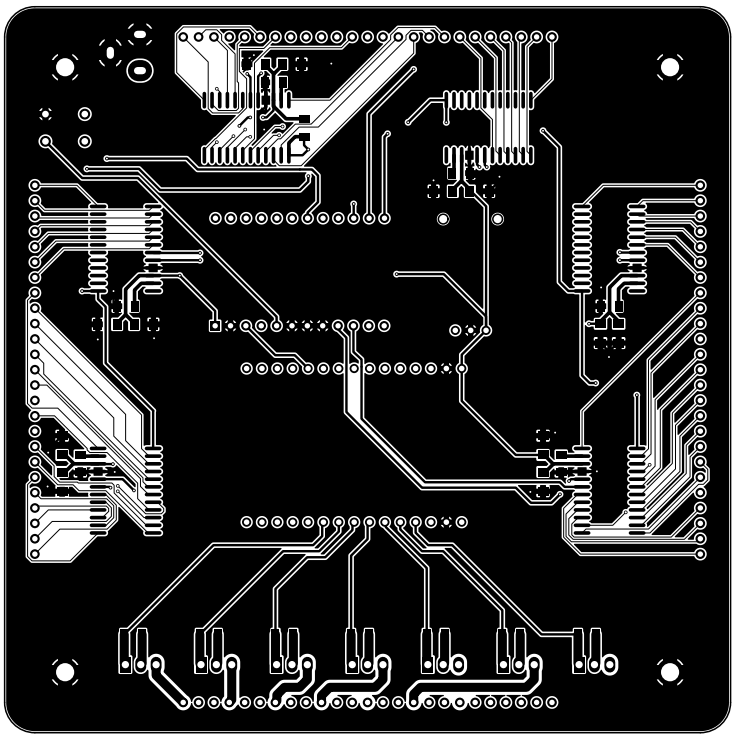
7 PCB Layers

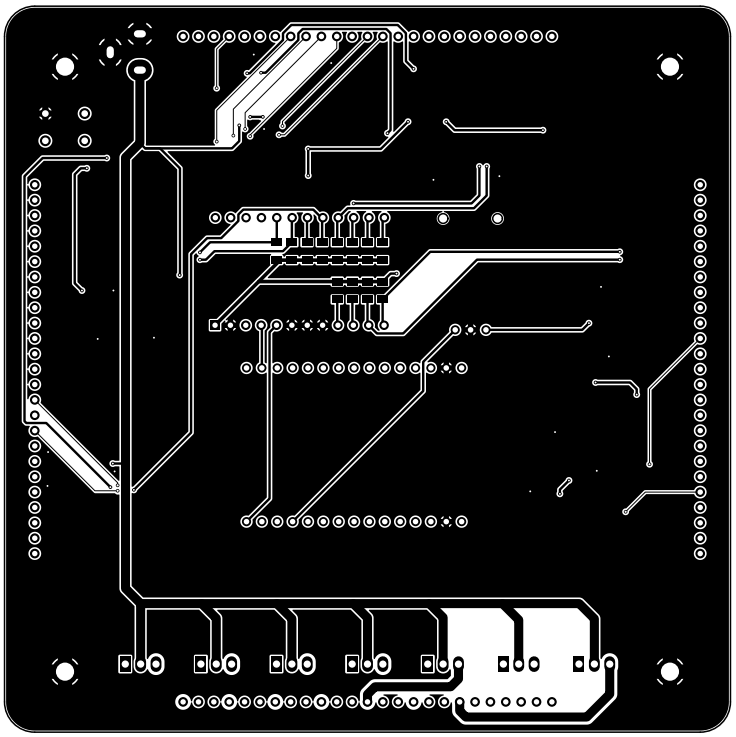
7.1 LED Interface PCB





7.2 LED Controller PCB





8 Bill of Materials

Name	Manufacturer	Supplier	Quantity	Price Per	Total Price
0.1uF	KEMET	Digi-Key	6	0.08	0.48
1u	KEMET	Digi-Key	6	0.17	1.02
MI_MAX4466 MIC	MAX	Amazon	1	4.39	4.39
DC Power Jack	CUI Inc	Adafruit	1	0.95	0.95
IRF9540NPBF	IR	LCSC	7	0.69	4.83
1k	Stackpole	Digi-Key	7	0.03	0.21
3.3k	YAGEO	Digi-Key	6	0.07	0.42
4.7k	TE Connectivity	Digi-key	12	0.03	0.36
100k	TE Connectivity	Digi-Key	1	0.08	0.08
PTS645SL432LFS (switch)	C&K	LCSC	1	0.08	0.08
ESP32-DEVKITV1	Espressif	Amazon	1	7.5	7.5
IS31FL3218	ISSI	Digi-Key	6	1.47	8.82
25 PIN HEADER	Generic	Amazon	2	4.99	9.98
TCA9548A module	HiLetGo	Amazon	1	1.83	1.83
LED (Common Anode)	EDGELEC	Amazon	175	0.09	15.75
RUBBER FEET (enclosure)	AUSTOR	Amazon	4	0.2	0.8
36W 3V Power Subbly	Belker	Amazon	1	15.9	15.9
PLA Plastic	Amazon Basics	Amazon	1	0.75	0.75
				Total:	74.15

9 Time Report

	Week 11	Week 12	Week 13	Week 14	Week 15
James Wilcock	3	2	5	9	10
Aiden Olsen	8	5	10	5	10
Blake Wiker	2	2	3	7	6
	Week 16	Week 17	Week 18	Week 19	Week 20
James Wilcock	20	15	20	5	2
Aiden Olsen	24	10	15	5	2
Blake Wiker	10	6	4	3	2
	Total Hours				
James Wilcock	91				
Aiden Olsen	94				
Blake Wiker	45				