Developer Guide

Pi Music Box

ECE 342-SP20

System Overview

The Pi Music Box is a battery powered system with which users can play audio files from a local library, as well as record and playback their own audio with the built in microphone. Omxplayer is used to play audio files, which offers a wide variety of compatibility with audio codecs, including MP3, FLAC, and AAC. An Arduino is used to control the LEDs on the box, allowing the user to adjust brightness and color. The enclosure is 3D printed and contains all of the components of the music box.

Electrical Specifications

Power Supply

The figure below shows the block diagram for the power supply system in the music box. A battery pack, consisting of 8 AA batteries, supplies a voltage of 12V to the power supply module assembled in Junior Design 1. The JD power supply was implemented to supply a constant 5V to the Arduino, Raspberry Pi, and the Speaker. As the batteries discharge, the total voltage supply will decrease, but the speaker and microprocessors will continue to be supplied a steady 5V from the JD power supply. The Arduino Uno has a maximum current draw of 500 mA to 1A. However, since the arduino is only being used to power

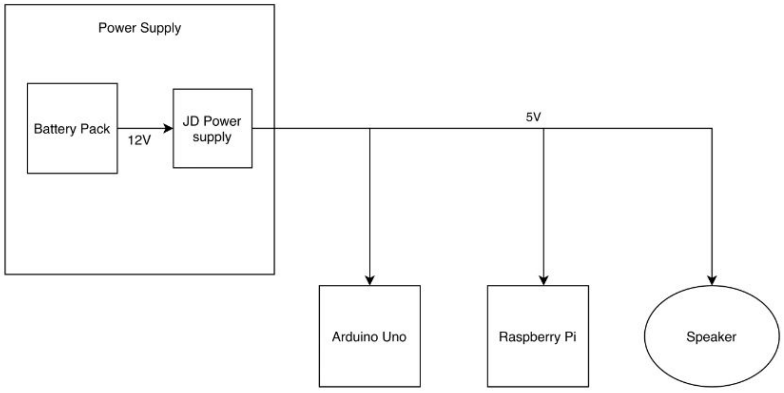

Figure 1: Power Supply Block Diagram

User Guide

Setting up

First and foremost, the batteries need to be installed. This system takes 8 AA batteries to power it. Batteries are installed by opening the top lid of the enclosure, and removing the slide on the two black, square battery packs. Each battery pack contains 4 AA batteries. Secondly the user must load their music onto the box, there are two ways you can do this. You can either load them directly onto the microSD card you're using, or SSH into the Pi to either send files directly over your network, or control the Pi and use the "wget" command to download a file from a url. Once you have your batteries in and your songs preloaded, the music box is ready to go.

Operation

Navigating the Menu System

Navigating the menu system is done by using the left and right buttons. The system prompts the user via the screen while they are browsing. The first menu the user encounters is the Library menu. The user is asked if they would like to view the library. The user can say "yes" by pressing the left button, or "no" by pressing the right button. If the user says yes they are presented with the library, which they can iterate through and choose a song to play. When the user is done viewing the library they will be sent to the second menu.

The second menu the user will encounter is the recording menu. At this menu the user can choose to record a 20 second audio file via the microphone in the music box. After the user has recorded their audio file they are asked if they would like to play back the recorded audio. If the user says yes, the file is played, and if not they are sent to the next and final menu.

The last menu asks the user if they would like to exit the program. If they say yes the program exits, and if they say no they are sent back to the first menu.

Controlling the LED

To control the brightness of the LED use the potentiometer. It looks like a knob. When the knob is set all the way to the left, the LED is off. Turn the knob to the right a little. The LED should turn on. If you want the LED brighter, you turn it to the right. Keep turning until desired brightness. If the knob is turned all the way to the right, the LED is at maximum brightness. Turn it back to the left to dim it.

If you want to change the color of the led you want to press the button. Each time you press the button it will cycle to a different color. There are 10 preprogrammed colors for you to choose from.

Design Artifacts

Speaker Block

The figure below is the schematic for the speaker circuit. This circuit uses an LM386 OpAmp to amplify the audio signal for playback through an 8 Ohm speaker. There is a bandpass filter that allows frequencies between about 300 Hz to approximately 3600 Hz to pass through the circuit. There are also capacitors on the V_In and output of the OpAmp, to filter out any noise in the signal.

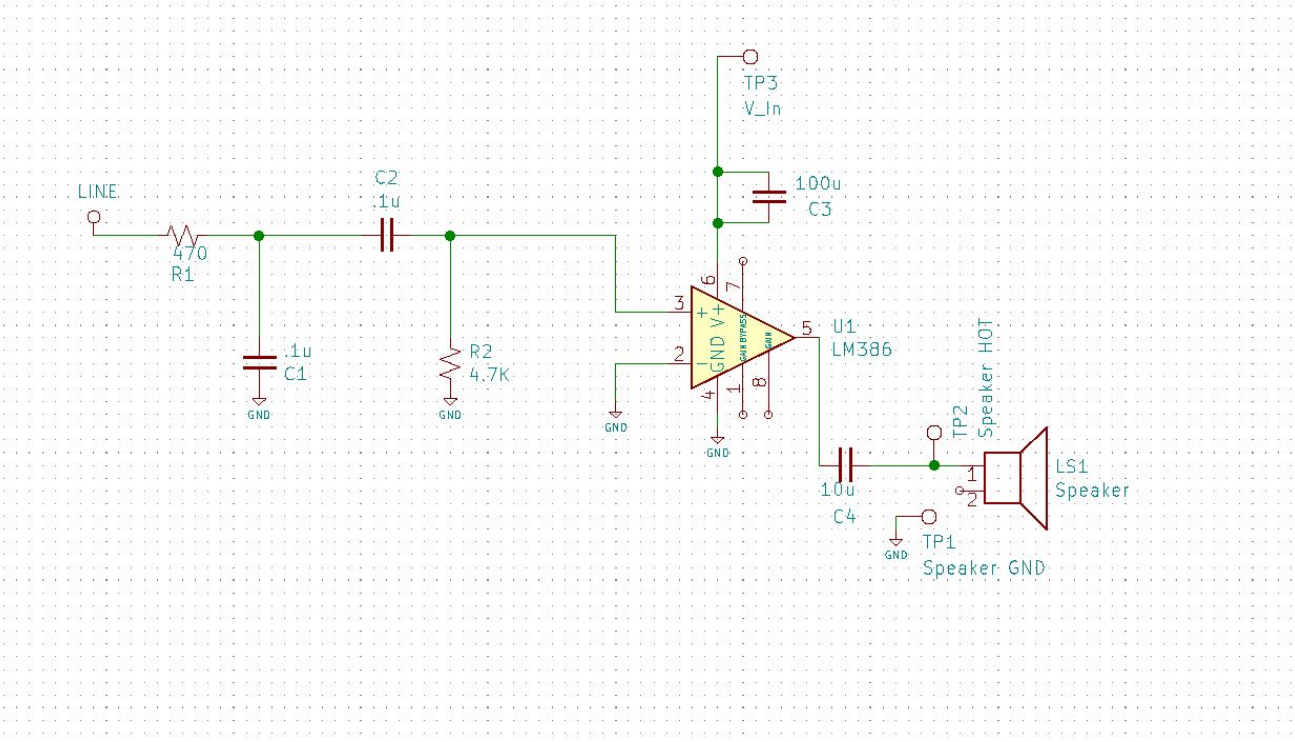

Figure 2: Speaker Schematic

The figure below is a screenshot of the CAD design of the PCB used for the speaker circuit. It is an implementation of the schematic above. There are test pads where the leads for the speaker output can be soldered on, as well as a test pad for the voltage in. The positive lead for the audio input signal from the Pi is soldered to the resistor R1, and the ground for V_In and the audio input are shared with the speaker GND.

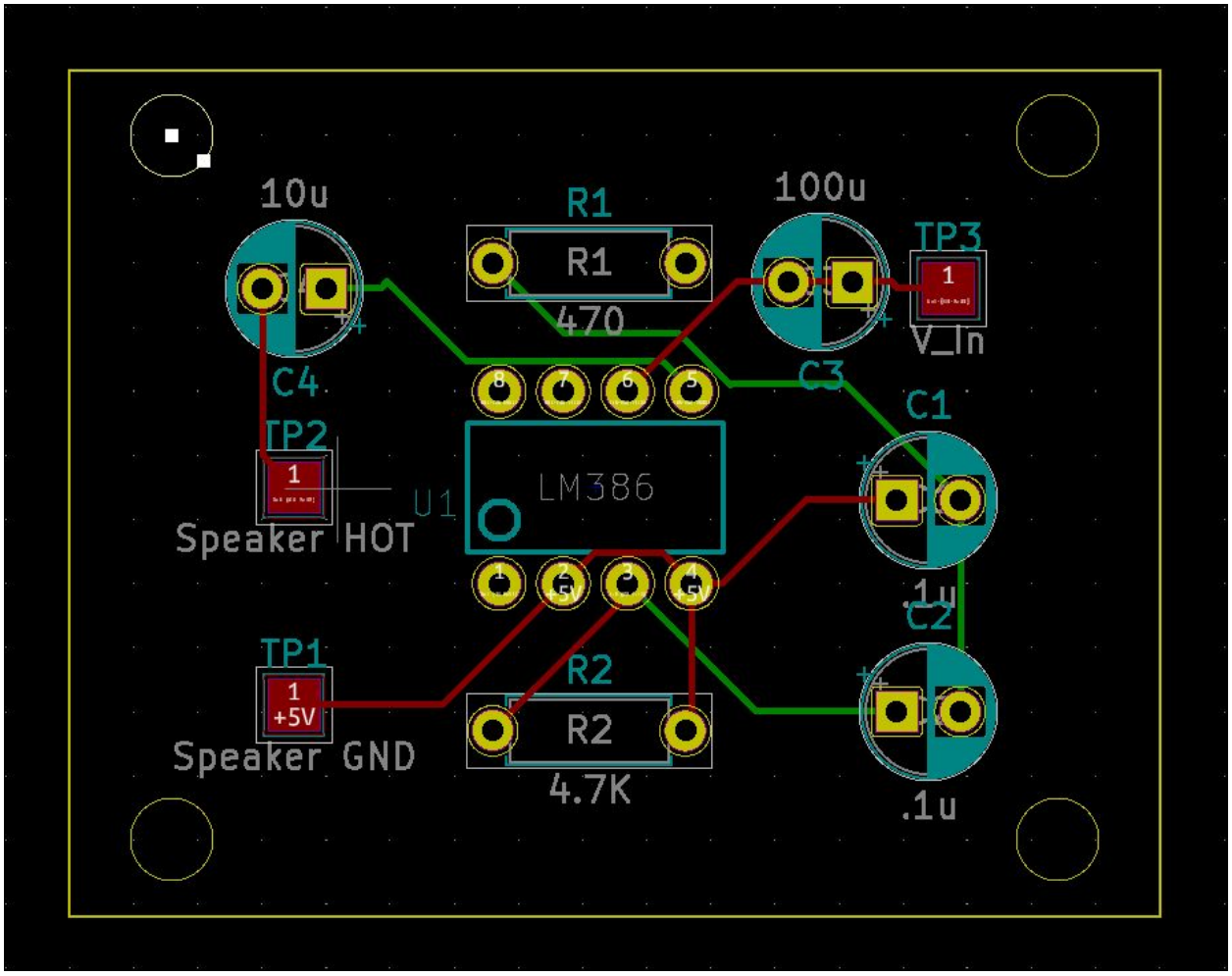

Figure 3: Speaker Circuit PCB

Below is a 3D render of what this PCB will look like when manufactured. It is designed with screw holes for mounting into an enclosure. It also has silk screening indicating where each component from the schematic is placed, and the orientation of the polarized capacitors and OpAmp. This PCB is designed to be relatively compact, and is only 1.7 x 1.32 inches, or 43.3 x 33.6 mm. Included is also a picture of the PCB after fabrication.

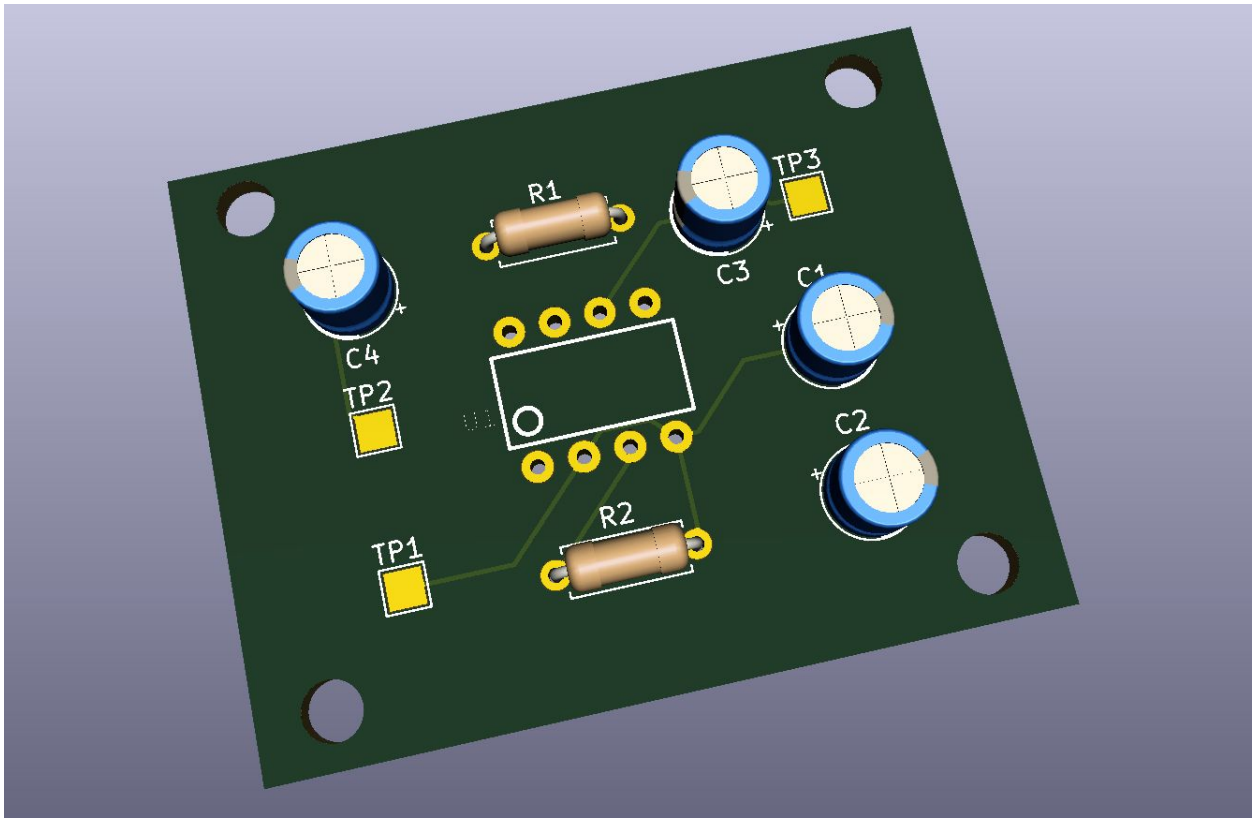

Figure 4: Speaker Circuit PCB Rendering

Figure 5: Assembled Speaker PCB

Pi Block

This is the main code for the Pi. It is written in Python and operates by using loops for different processes. It iterates through three main states, the first state is the library state, the second state is the recording state, and the last state is the exit state. At the beginning of each state the user is prompted if they would like to proceed to the next state, or use the functions of their current state. For example, if the user is at the library state they can choose to view the library and pick a song to play, or they can choose to go to the next state. This code also calls a C++ program that captures the filenames of songs in the library folder. This populates a text file with the names and is used for viewing the library and playing specific songs.

```
            fname = Library.readline()
            Library.readline()

            # checks to see if there is no file to display, if there is no file
            # it displays to the user that they have reached the end of the library
            # and goes to the next state of the program
            if fname == '':
                lcd.clear()
                lcd.message = "End of\nLibrary"
                time.sleep(1)
                looping=0
                libloop=0
                break

            # if there is a file to display, then the lcd will display the file name
            # and prompt the user if they want to play the song
            lcd.message=fname
            time.sleep(2);
            lcd.clear()
            lcd.message="Play song?\n<- Yes     No ->"

            # creates loop to wait for user input
            # if user hits left button, the song plays
            # if user hits right button, song doesn't play and the library loop restarts

            lupin=1
            while lupin==1:
                if lcd.left_button:

                    # displays to the user that the device is playing a specific song
                    lcd.clear()
                    lcd.message=f"Playing\n{fname}"
                    played = os.system(f"omxplayer -o local ~/library/{fname}")

                    # when the system call to play the song executes, the lcd displays
                    # that the song has finished playing

                    if played==2:
                        lcd.clear()
                        lcd.message=f"Done Playing\n{fname}"
                    lupin=0
                if lcd.right_button:
                    lupin=0
            time.sleep(1)

            # asks the user if they would like to continue viewing the library
            # if they hit left button the library loop keeps looping
            # if they hit right button the program moves to the next state
            lcd.clear()
            lcd.message="Continue viewing\n<- Yes     No ->"
            lupin=1
            while lupin==1:
                if lcd.left_button:
                    libloop=1
                    lupin=0
                if lcd.right_button:
                    libloop=0
                    lupin=0
    if lcd.right_button:
        choice_wheel = choice_wheel +1
        looping =0
```

```
# begins second state of program, prompting the user about recording audio
# if the user chooses to record audio, 20 seconds of audio is recorded
# and stored as recording.wav, the user can choose to play the audio back in this
# state as well.


if choice_wheel ==1:
    lcd.clear()
    lcd.message = "Record Audio?\n<- Yes    No ->"

    # loops while waiting for user input
    looping=1
    while looping ==1:

        # if user selects yes the audio is recorded
        # if user selects no they are sent to the next state

        if lcd.left_button:
            lcd.clear()
            lcd.message = "Recording\n Audio.."
            played = os.system("arecord --format=S16_LE --duration=20 --rate=16000 --file-type=raw recording.wav")

            if played==0:
                lcd.clear()
                lcd.message="Done recording"
                time.sleep(1)
                lcd.clear()

                # prompts user if they would like to play back the recorded audio
                # loops while waiting for user response, if user says yes audio is
                # played back, otherwise they are sent to the next state

                lcd.message="Play recording?\n<- Yes    No ->"
                lupin = 1
                while lupin==1:
                    if lcd.left_button:
                        lcd.clear()
                        lcd.message = "Playing\n recording.."
                        played = os.system("omxplayer -o local recording.wav")
                        print (played)

                        if played==256:
                            lcd.clear()
                            lcd.message="Done playing\n recording"
                            lupin =0
                            time.sleep(2)

                choice_wheel = choice_wheel + 1
                looping=0

        if lcd.right_button:
            choice_wheel= choice_wheel+1
            looping=0







# begins third state of the program and prompts user to exit
```

```
# if user hits left button, system exits
# if user hits right button, system returns to first state

if choice_wheel == 2:
    lcd.clear()
    lcd.message = "Exit\n<- Yes    No ->"
    looping=1
    while looping ==1:
        if lcd.left_button:
            lcd.clear()
            lcd.message = "Have a nice\nday!"
            sys.exit()
        if lcd.right_button:
            choice_wheel = 0
            looping =0
```

Figure 6: Main Pi Code

This is the library capture code, which is written in C++ and called within the main program. It works by creating a text file called “lib.txt” which it writes to, it then opens a pipe stream and writes each file name to the text file until there are no more files left. It then closes the pipe stream and ends the program. It is simple but effective at capturing and populating the library of the music box.

```
using namespace std;

#include <iostream>
#include <ostream>
#include <string>
#include <wiringPi.h>
#include <fstream>


int main()
{

        // declares necessary variables
        // FILE object, string and character array

        FILE *lib;
        char sysRet[255];
        string cmd;


                // creates text file to write out to

                ofstream out("lib.txt");

                // opens pipe stream

                lib = popen("ls ~/library","r");
                int num =1;

                // while there is still information coming from
                // pipe stream the loop outputs the file names
                // into the file 'lib.txt'
                while(fgets(sysRet,255,lib) != NULL){
                        string a(sysRet,255);
                        out << sysRet << endl;
                        //cout << a << endl;

                        // This is here to check that the library
                        // file is populated correctly
                        //printf("Song %d is %s",num, sysRet);



                        num = num+1;
                }

                // closes the pipe stream
                pclose(lib);
        }
```

Figure 7: Library Capture

Power Supply Block

The Power Supply block is responsible for supplying 5V to the Arduino Uno, Raspberry Pi, and Speaker Circuit from a battery pack supplying 12 volts. The Junior Design Power Supply Module is implemented to regulate the main supply voltage keeping it at a steady 5V. The Arduino is expected to draw a 500 mA output current, plus 40 mA through the A0 output pin, the Pi will draw a maximum current of 400 mA, and the speaker circuit will draw a current of 4 mA. This makes the maximum load current experienced by the JD power supply 944mA, which is well under the maximum output current of 1.9A for the JD Power Supply.

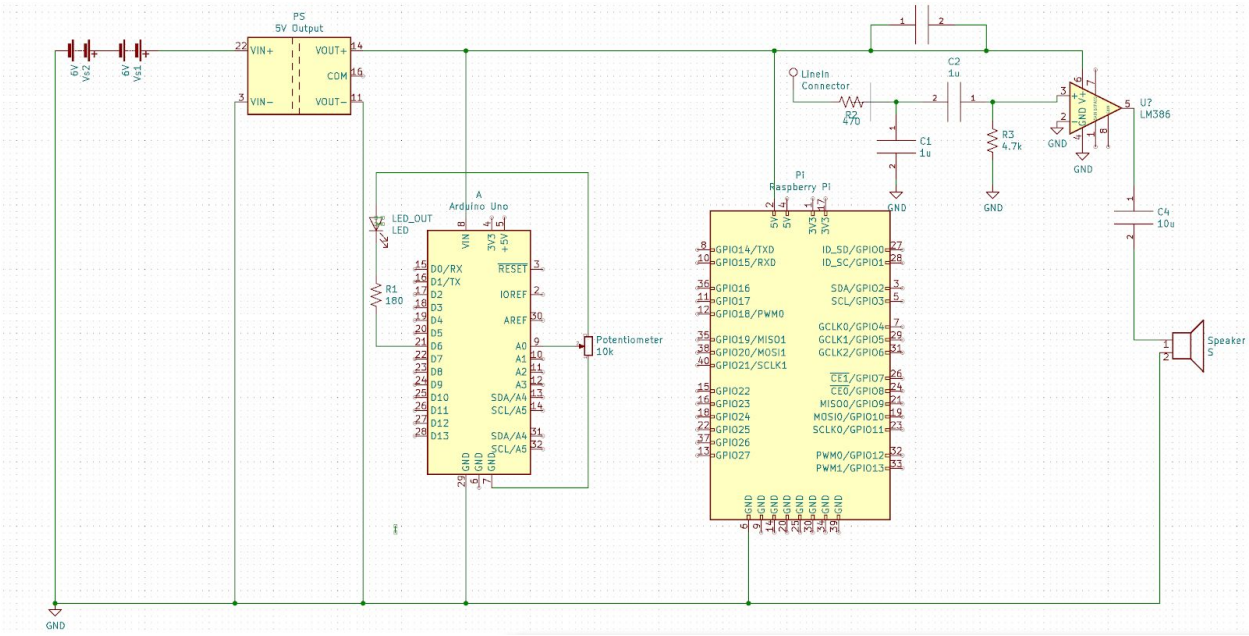

Figure 8: Final Schematic (Power Supply Orientation shown)

The PCB applied to the power supply block was simply a rail, designed to make connections between the JD Power Supply Module and the Arduino Uno, Raspberry Pi, and the Speaker Circuit without needing a breadboard, as well as supplying a common ground for all components receiving power directly from the Power Supply Block. The Power Supply PCB's dimensions are 5.35'' x 3.94''. The main rail of the PCB contains three traces in parallel from the voltage input pin, to three voltage output pins. On the bottom of the PCB is a single trace with several pins that serves as a common ground for all related components.

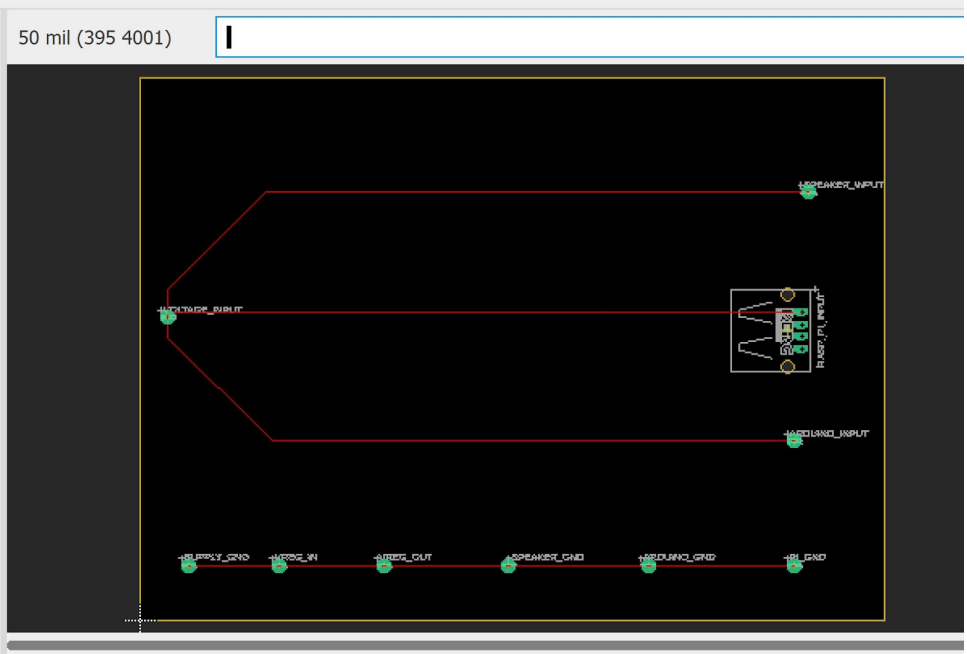

Figure 9: Power Supply PCB Board layout

Enclosure Block

For the enclosure, a 3-D printable box with a hinged top was modeled in fusion 360. This box was designed to be 30x20x12cm, in order to fit in all necessary components with a little extra room for easy inspection. The hinged top allows for the user, and/ or designer to get an easy look at the interior and design of the overall system, and to grant easy access to the battery pack. The box's large design allows for several different prototype options for LED orientations. In the image below, the box would support a small LED strip running vertically along the left side of the face. The surface area of the base of the box would be 600 square centimeters. Of the components that will rest on the base of the box, about 450 square centimeters will be taken up.

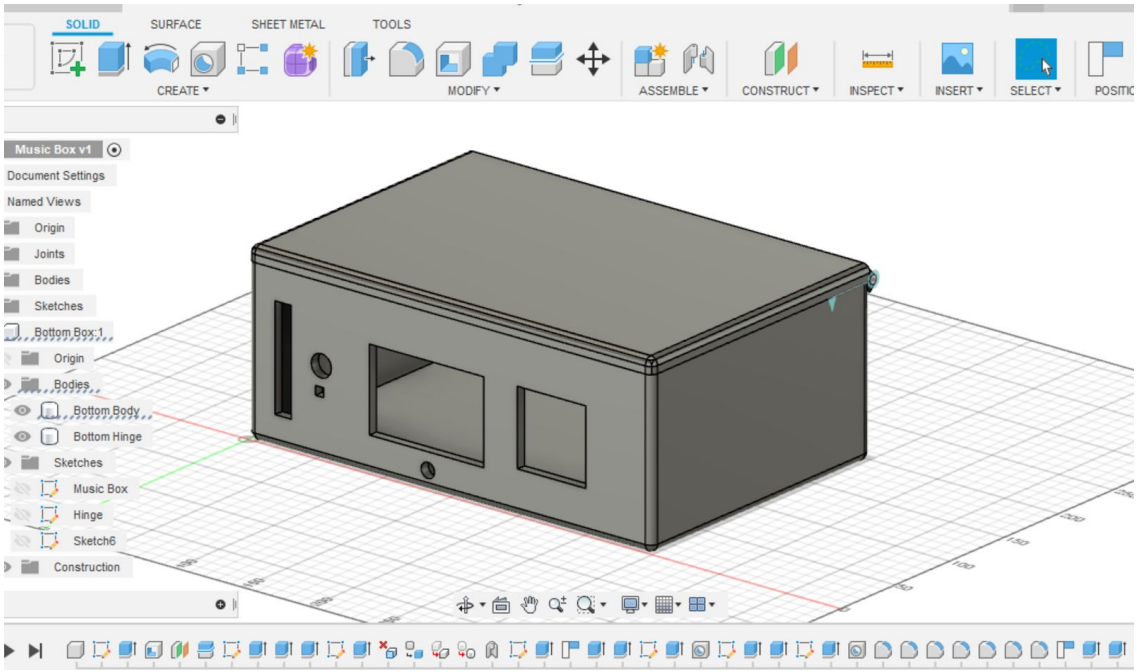

Figure 10: Enclosure Model (Fusion 360)

LED Block

The Figure below shows the schematic for the LED control. An LED is connected to a potentiometer and resistor and the Arduino. The potentiometer is also connected to the Arduino. There is also a switch connected to the Arduino.

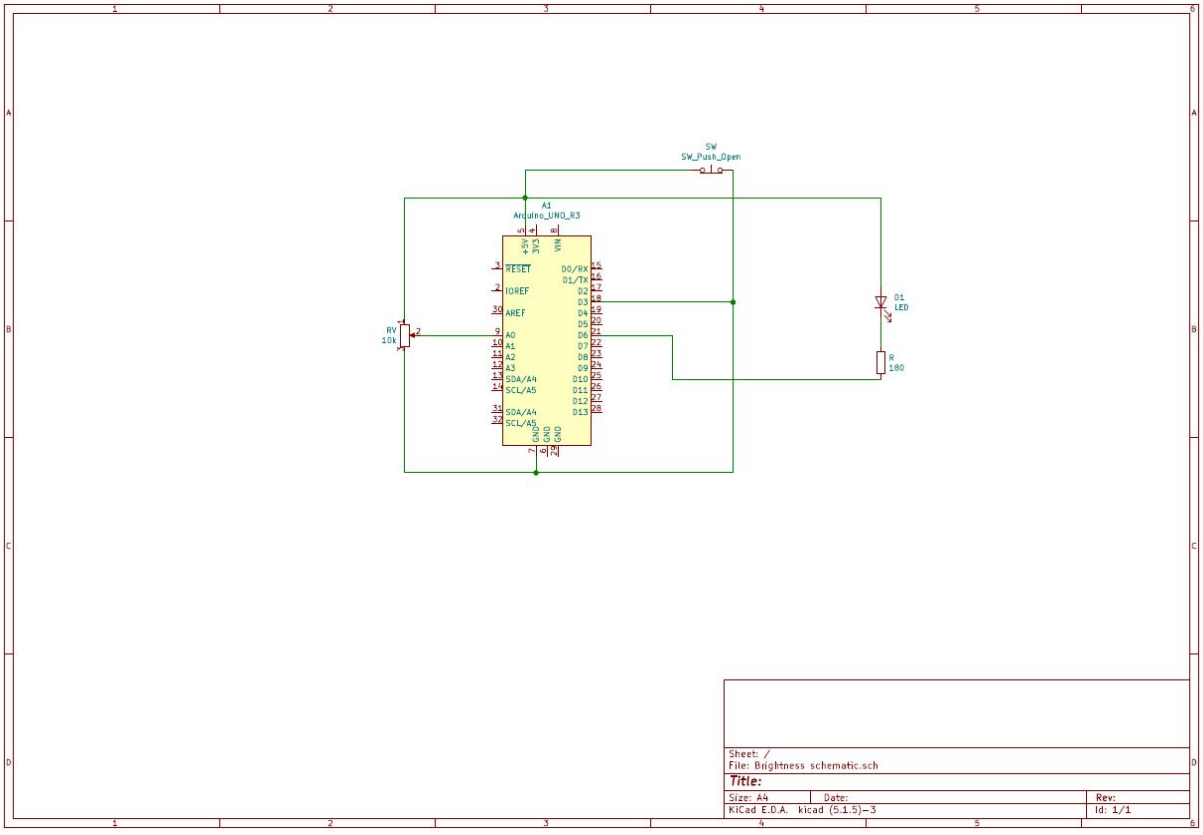

Figure 11: Arduino LED Schematic

Below is the PCB for the LED control. It has the LED, potentiometer, button, and resistor. It also has a couple of connector pins for the power supply.

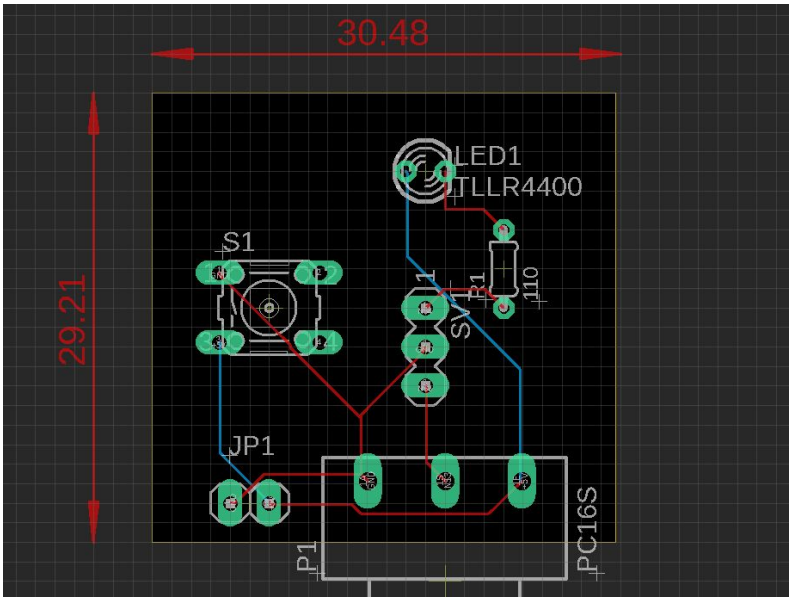

Figure 12: Arduino Control PCB

Here is the code for the Arduino. This code explains how the LED brightness is changed and what colors are coded in for the LED. The Arduino takes in input from the potentiometer. It then divides this input by four so the values are in range for the LED. Depending on the value, the LED brightness will change accordingly. There is also a function that will change the color if the button is pressed. The counter starts at zero and increases by one for each time the button is pressed. The color corresponds to what number the counter is at. The counter resets after nine.

LED_Brightness_and_Color_Change

```
int sensor = A0;
int redPin = 9;
int greenPin = 10;
int bluePin = 11;
int buttonValue = 0;
int button = 3;
int counter = 0;
int output = 6;

void setup()
{
pinMode(redPin, OUTPUT);
pinMode(greenPin, OUTPUT);
pinMode(bluePin, OUTPUT);
pinMode(button, INPUT);
pinMode(output, OUTPUT);
}

void loop()
{
if(counter == 0){
setColor(255, 0, 0);//sets default color to red
}

int reading = analogRead(sensor); //input from potentiometer
int bright = reading/4; //divide by 4 to bring reading in range of 0-255
delay(500);
analogWrite(output, bright); //sets led output to bright, i.e changes the brightness

buttonValue = digitalRead(button); //sets the button value to button input
if(buttonValue == HIGH){
  counter = counter + 1;
}

if(counter == 1){
  setColor(255, 128, 0);//sets color to orange
}

if(counter == 2){
  setColor(255, 255, 0);//sets color to yellow
}
```

```
if(counter == 3){
  setColor(0, 255, 0);//sets color to green
}

if(counter == 4){
  setColor(0, 255, 128);//sets color to spring green
}

if(counter == 5){
  setColor(0, 255, 255);//cyan
}

if(counter == 6){
  setColor(0, 128, 255);//azure
}

if(counter == 7){
  setColor(0, 0, 255);//blue
}

if(counter == 8){
  setColor(128, 0, 255);//violet
}

if(counter == 9){
  setColor(255, 0, 255);//magenta
}

if(counter > 9){
  counter = 0;
}

}

void setColor(int redValue, int greenValue, int blueValue){
  analogWrite(redPin, redValue);
  analogWrite(greenPin, greenValue);
  analogWrite(bluePin, blueValue);
}
```

Figure 13: Arduino LED Code

Microphone Block

The only artifact for this block is code for the microphone. We connected a usb microphone to the Raspberry Pi. The code is just setting up the microphone so it can be used with the Raspberry Pi. It also contains the command for recording and playback sound.

```
////////////////////////////////////////////////////////
Pi Recorder Program
Records audio for the Raspberry Pi
////////////////////////////////////////////////////////

pcm.!default {
   type asym
   capture.pcm "mic"
   playback.pcm "speaker"
}

pcm.mic { //sets up the microphone
   type plug
   slave {
        pcm "hw:1, 0"
   }
}

pcm.speaker { //sets up the speaker
   type plug
   slave {
        pcm "hw:0, 0"
   }
}



////////////////////////////////////////////////////////////////////////////
Commands for recording and audio playback
////////////////////////////////////////////////////////////////////////////

command for record: arecord --format=S16_LE --duration=20 --rate=16000 --file-type=raw
sample.wav

command for playback: aplay --format=S16_LE --rate=16000 sample.wav
```

Figure 14: Microphone Module Code

Part Info

Component List

Component #	Part Name	Quantity	Manufacturer
1	Raspberry Pi 3B	1	Raspberry Pi
2	Mini USB Mic	1	Adafruit Industries LLC
3	8 Ohm Speaker	1	PUI Audio, INC
4	470 Ohm Res.	1	Vishay BC Components
5	4.7K Ohm Res.	1	Vishay BC Components
6	.1uF Cap.	2	KEMET
7	10uF Cap.	1	KEMET
8	100uF Cap.	1	TDK Corporation
9	LM386	1	Texas Instruments
10	Arduino Uno Clone	1	HK Shanghai Group Limited
11	Omron Tact Switch	1	Omron Electronics
12	Red LED, 5mm	1	NTE Electronics
13	180 Ohm REs.	1	Stackpole Electronics
14	10K Potentiometer	1	Jameco ValuePRo
15	Adafruit 16x2 LCD	1	Adafruit Industries LLC
16	32GB microSD	1	Sandisk
17	JD Power Supply	1	OSU
18	6V AA Battery Case	2	Ogrmar
19	AA Alkaline Battery	8	*User Dependent