Music Cube

ECE 110 Honors Lab Final Report

by

Dave Simley and Bradley Morrell

1. Introduction

a. Problem Statement

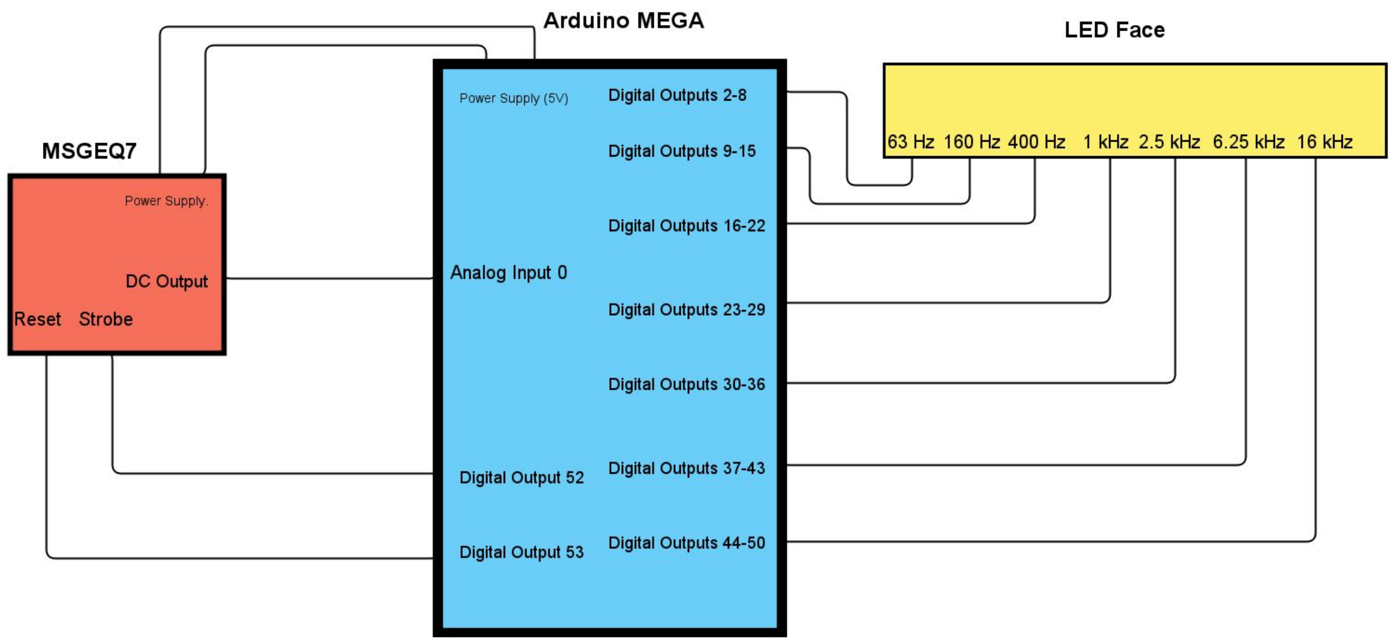
In today's modern age, music is transitioning from being purely auditory to more of an audio-visual experience. For example, many artists, when performing live, incorporate a light show with their performance, because when an extra sense is stimulated, music becomes more captivating and enjoyable. By designing and creating a product that is able to provide a dynamic audio-visual experience, listeners will be more engaged and receptive of their music.

b. Proposed Solution

Our proposed solution to provide this experience is to design and create a small cube that will be able to actively interpret a musical signal and provide some visual animation depending on the frequency and strength of the signal. We plan to do this by creating a seven-by-seven LED display on a face of a cube that will be a graphic equalizer for a musical input, with each column representing a different frequency, and each row representing how loud the respective frequency is in the musical signal. With this device, an extra sense will be engaged while listening to music, providing a more satisfying and captivating musical experience. Also, due to the graphical interpretation of the music, the cube's applicability can extend beyond providing an audio-visual experience. Our project can be used for instructional purposes, demonstrating what different frequencies are found in music, and as a means to interpret the strength of certain frequencies in a signal.

2. Design

a. Block Diagrams

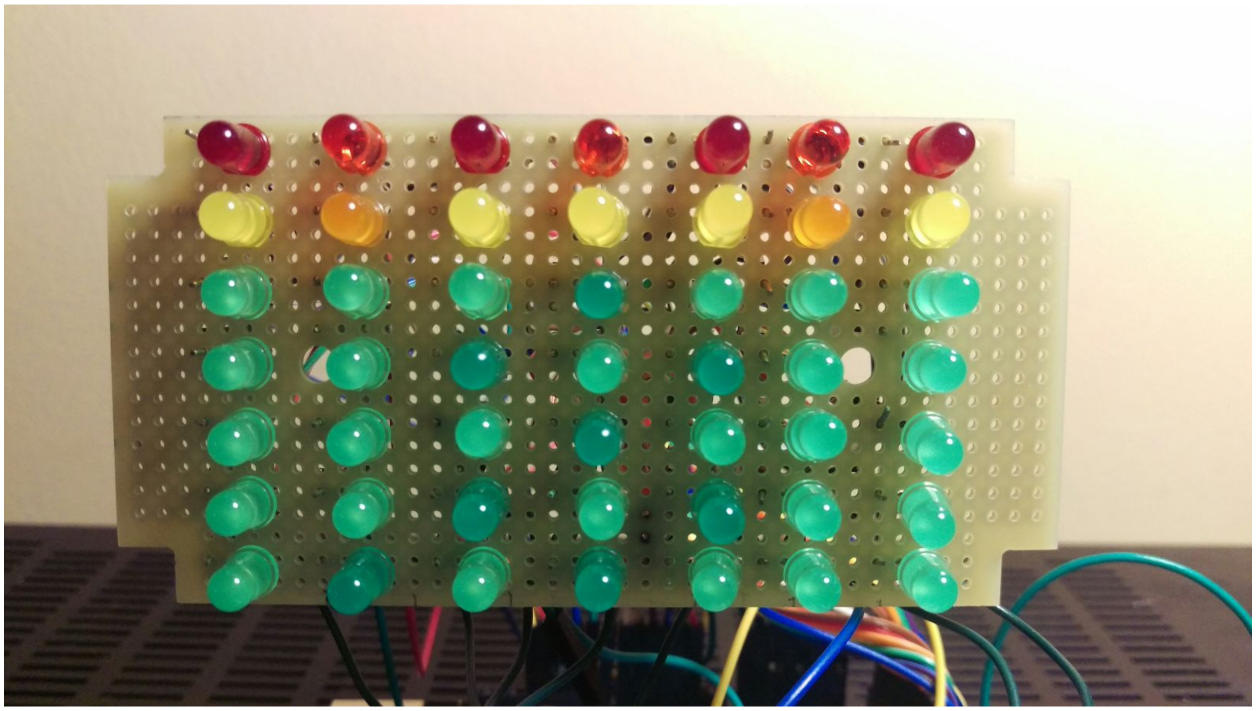


b. Written Description of Block Diagrams

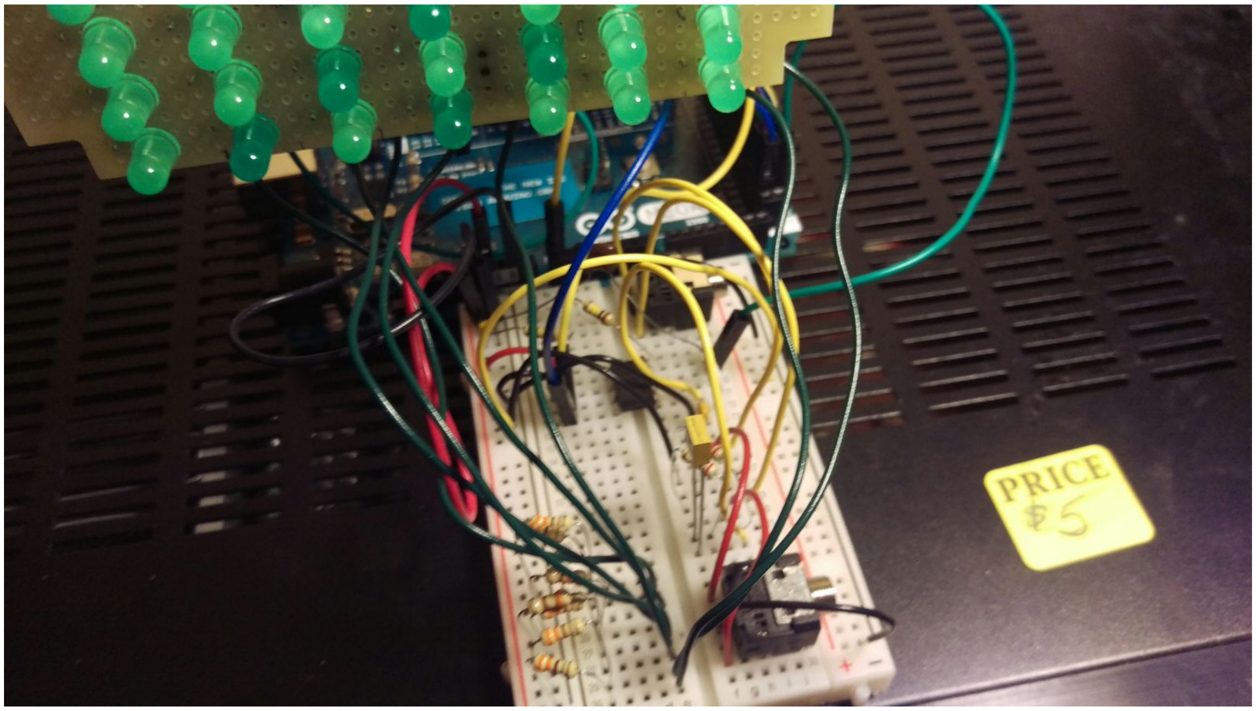
Our project utilized Sparkfun's MSGEQ7 chip, which is able to interpret a musical signal and then send out information pertaining to preset frequencies it had analyzed. When a logical "1" value is applied to the chip's strobe pin, then a logical "0" is applied to the strobe, the chip's DC output will then output a DC voltage that is representative of the amplitude of a certain frequency (the first one being 63 Hz). Then, when a logical "1" followed by a logical "0" is applied to the strobe pin, the chip's multiplexor will then advance to the next frequency (160 HZ) and provide a DC voltage that is representative of its amplitude. When all the frequencies have been analyzed by the Arduino's analog input, the Arduino will then apply a logical "1" to the chip's reset pin, resetting the multiplexor back to the first frequency. Also, by having the chip powered by the Arduino, we save valuable board real estate. When the Arduino analyzes the DC voltage given by the chip pertaining to a frequency, the Arduino will then turn on or off certain digital outputs in order to represent the amplitude of a certain frequency on the LED display. Digital pins 2-8 are used for the 63 Hz frequency, 9-15 are used for 160 Hz, etc. When a loud 63 Hz frequency is detected in the musical signal, digital outputs 2-7 or possibly 2-8 will be turned on, so that the LED's in the first column are all almost fully lit. The Arduino then uses the strobe pin on the chip to cycle through each frequency at a fast speed, and analyzing each one so that a smooth, clear equalizer is seen on the LED display.

c. **Pictures of the Display**

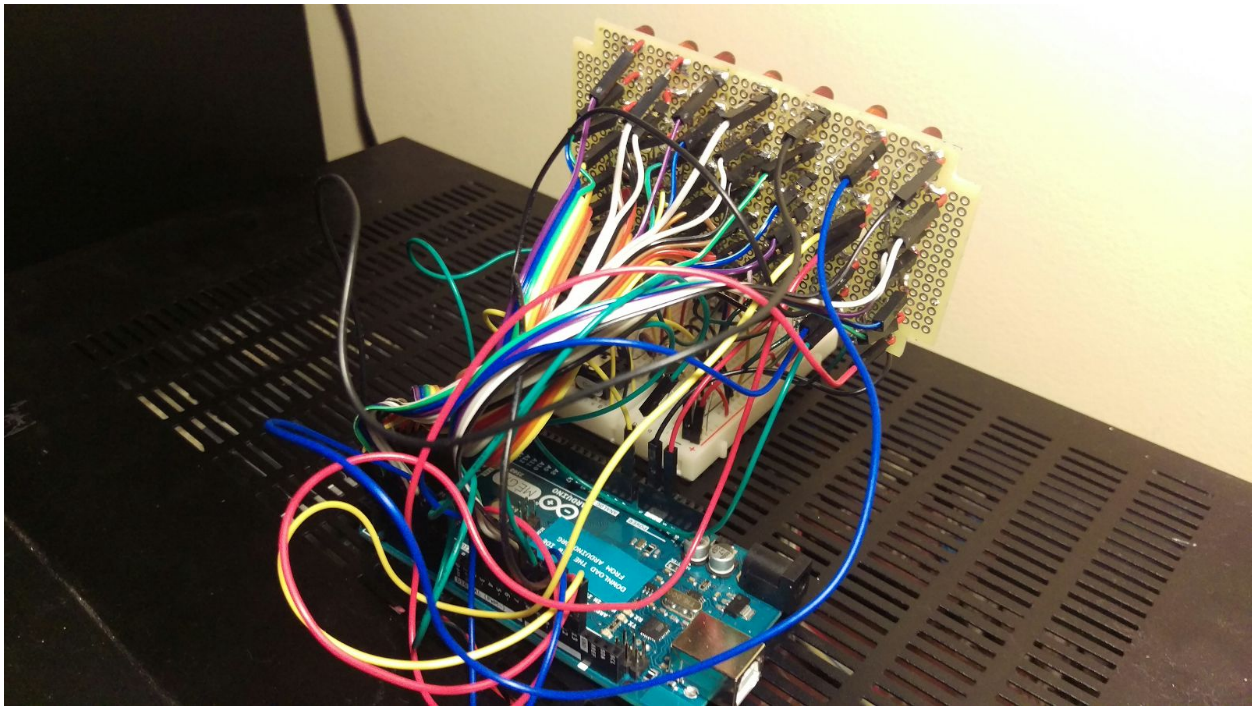
Front of the LED Display



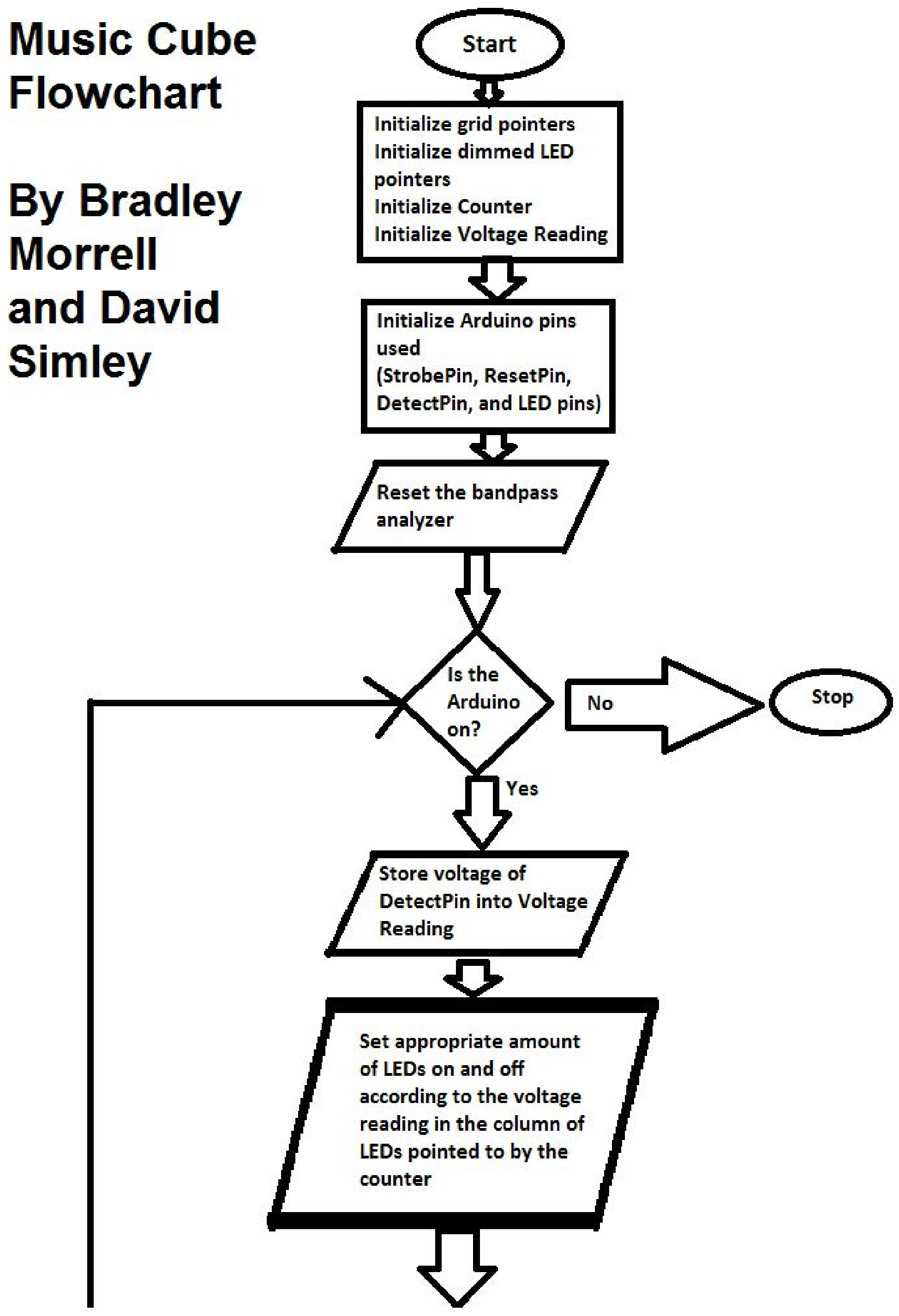
Breadboard containing circuitry of the MSGEQ7 chip

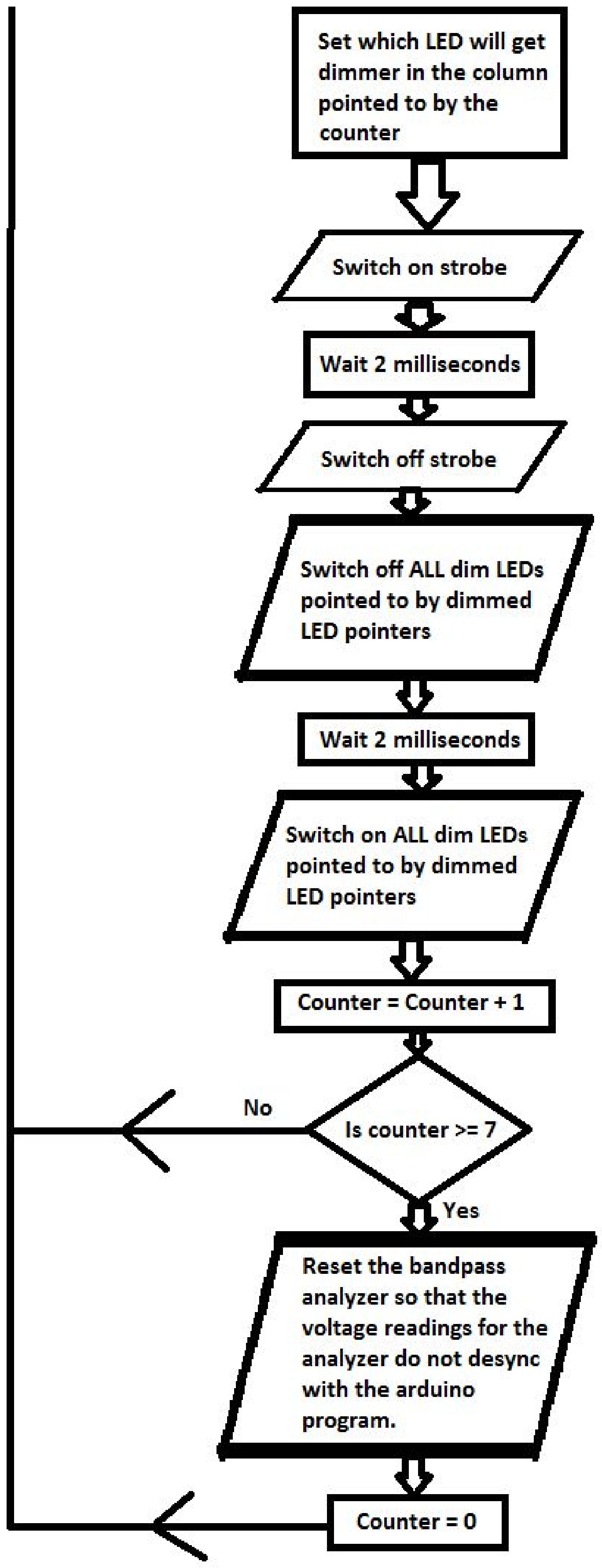


Back Side of the Display and Wiring of the Arduino



d. Flow Chart of Software


Music Cube Flowchart
By Bradley Morrell and David Simley
Start
Initialize grid pointers
Initialize dimmed LED pointers
Initialize Counter
Initialize Voltage Reading
Initialize Arduino pins used (StrobePin, ResetPin, DetectPin, and LED pins)
Reset the bandpass analyzer
Is the Arduino on?
No
Stop
Yes
Store voltage of DetectPin into Voltage Reading
Set appropriate amount of LEDs on and off according to the voltage reading in the column of LEDs pointed to by the counter


Set which LED will get dimmer in the column pointed to by the counter
Switch on strobe
Wait 2 milliseconds
Switch off strobe
Switch off ALL dim LEDs pointed to by dimmed LED pointers
Wait 2 milliseconds
Switch on ALL dim LEDs pointed to by dimmed LED pointers
Counter = Counter + 1
Is counter >= 7
No
Yes
Reset the bandpass analyzer so that the voltage readings for the analyzer do not desync with the arduino program.
Counter = 0

e. Code Used

```
//Code Copyright Bradley Morrell and Dave Simley
//Licensed under GPL version 3.
int LEDGridLocation[7][7];
int DimmedLEDs[7];
int StrobePin = 52;
int ResetPin = 53;
int DetectPin = 0;
int Counter = 0;
int Reading = 1;

void setupGrid()
{
    for (int i=0;i<7;i++)//set up gridlocation as pointers to grid
    {
        for (int j=0;j<7;j++)
        {
            LEDGridLocation[i][j] = 7*i + j + 2;
        }
    }
    for (int i=0;i<49;i++)//setup PinMode for all pins in grid
    {
        pinMode(i,OUTPUT);
    }
}

void setPinValues() // set pins on or off according to reading
{
    if(Reading >= 0 && Reading < 128)
    {
        for (int i=0;i<7;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],LOW);
        }
            DimmedLEDs[Counter] = -1;
    }
    else if(Reading >= 128 && Reading < 192)
    {
        for (int i=0;i<1;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],HIGH);
        }
        for (int i=1;i<7;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],LOW);
        }
            DimmedLEDs[Counter] = 0;
    }
    else if(Reading >= 192 && Reading < 256)
    {
        for (int i=0;i<1;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],HIGH);
        }
        for (int i=1;i<7;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],LOW);
        }
            DimmedLEDs[Counter] = -1;
    }
    else if(Reading >= 256 && Reading < 320)
    {
        for (int i=0;i<2;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],HIGH);
        }
        for (int i=2;i<7;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],LOW);
        }
            DimmedLEDs[Counter] = 1;
    }
    else if(Reading >= 320 && Reading < 384)
    {
        for (int i=0;i<2;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],HIGH);
        }
        for (int i=2;i<7;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],LOW);
        }
            DimmedLEDs[Counter] = -1;
    }
    else if(Reading >= 384 && Reading < 448)
    {
        for (int i=0;i<3;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],HIGH);
        }
        for (int i=3;i<7;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],LOW);
        }
            DimmedLEDs[Counter] = 2;
    }
    else if(Reading >= 448 && Reading < 512)
    {
        for (int i=0;i<3;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],HIGH);
        }
        for (int i=3;i<7;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],LOW);
        }
            DimmedLEDs[Counter] = -1;
    }
    else if(Reading >= 512 && Reading < 576)
    {
        for (int i=0;i<4;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],HIGH);
        }
        for (int i=4;i<7;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],LOW);
        }
            DimmedLEDs[Counter] = 3;
    }
    else if(Reading >= 576 && Reading < 640)
    {
        for (int i=0;i<4;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],HIGH);
        }
        for (int i=4;i<7;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],LOW);
        }
        DimmedLEDs[Counter] = -1;
}
else if(Reading >= 640 && Reading < 704)
{
    for (int i=0;i<5;i++)
    {
        digitalWrite(LEDGridLocation[Counter][i],HIGH);
    }
    for (int i=5;i<7;i++)
    {
        digitalWrite(LEDGridLocation[Counter][i],LOW);
    }
        DimmedLEDs[Counter] = 4;
}
else if(Reading >= 704 && Reading < 768)
{
    for (int i=0;i<5;i++)
    {
        digitalWrite(LEDGridLocation[Counter][i],HIGH);
    }
    for (int i=5;i<7;i++)
    {
        digitalWrite(LEDGridLocation[Counter][i],LOW);
    }
        DimmedLEDs[Counter] = -1;
}
else if(Reading >= 768 && Reading < 832)
{
    for (int i=0;i<6;i++)
    {
        digitalWrite(LEDGridLocation[Counter][i],HIGH);
    }
    for (int i=6;i<7;i++)
    {
        digitalWrite(LEDGridLocation[Counter][i],LOW);
    }
        DimmedLEDs[Counter] = 5;
}
else if(Reading >= 832 && Reading < 896)
{
    for (int i=0;i<6;i++)
    {
        digitalWrite(LEDGridLocation[Counter][i],HIGH);
    }
    for (int i=6;i<7;i++)
    {
        digitalWrite(LEDGridLocation[Counter][i],LOW);
    }
        DimmedLEDs[Counter] = -1;
}
else if(Reading >= 896 && Reading < 960)
{
    for (int i=0;i<7;i++)
    {
        digitalWrite(LEDGridLocation[Counter][i],HIGH);
    }
        DimmedLEDs[Counter] = 6;
}
else if(Reading >= 960 && Reading < 1024)
{
    for (int i=0;i<7;i++)
    {
        digitalWrite(LEDGridLocation[Counter][i],HIGH);
    }
        DimmedLEDs[Counter] = -1;
}
```

```
    else
    {
        for (int i=0;i<7;i++)
        {
            digitalWrite(LEDGridLocation[Counter][i],LOW);
        }
            DimmedLEDs[Counter] = -1;
    }
}
void turnOffDimLeds()
{
    //this will turn off the power for all pins that are in the dim state.
    for (int i=0;i<7;i++)
    {
        if (DimmedLEDs[i] != -1)
        {
            digitalWrite(LEDGridLocation[i][DimmedLEDs[i]],LOW);
        }
    }
}
void turnOnDimLeds()
{
    //this will turn on the power for all pins that are in the dim state.
    for (int i=0;i<7;i++)
    {
        if (DimmedLEDs[i] != -1)
        {
            digitalWrite(LEDGridLocation[i][DimmedLEDs[i]],HIGH);
        }
    }
}

void setup() {
    Serial.begin(9600);
    setupGrid();
    pinMode(StrobePin,OUTPUT); //NEED PINMODE CRAP
    pinMode(ResetPin,OUTPUT);
    pinMode(DetectPin,INPUT);
    digitalWrite(ResetPin,HIGH); //turn on reset
    delay(2);
    digitalWrite(ResetPin,LOW); //turn off reset and turn on strobe
    digitalWrite(StrobePin,HIGH);
    delay(2);
    digitalWrite(StrobePin,LOW); //turn off strobe
}
void loop() {
    // put your main code here, to run repeatedly:
    Reading = analogRead(DetectPin); //storeAnalogReading into variable
    setPinValues(); //TurnOnLEDs
    digitalWrite(StrobePin,HIGH); //switch on the strobe
    delay(2);
    digitalWrite(StrobePin,LOW); //switch off the strobe
    turnOffDimLeds(); //SwitchoffDimLeds
    delay(2);
    turnOnDimLeds(); //switchonDimLeds

    Counter++;
    //check if counter is 7 or more, if it is, then reset the filter (includes a delay)
switch on strobe and switch off strobe.
    if (Counter >= 7)
    {
        digitalWrite(ResetPin,HIGH); //turn on reset
        delay(2);
        digitalWrite(ResetPin,LOW); //turn off reset and turn on strobe
        digitalWrite(StrobePin,HIGH);
        delay(2);
        digitalWrite(StrobePin,LOW); //turn off strobe
        Counter = 0;
    }
}
```

3. Results

a. Qualitative Analysis of Results

i. We were able to create the music visualizer. It worked! The visualization was smooth and followed the amplitude of the music as expected. The dimming of the LEDs to show the music amplitude was in between a row of LEDs worked nicely. We decided against putting the LEDs on a cube and instead put them on a flat screen.

b. Quantitative Analysis of Results

i. When a strong musical signal was sent into the MSGEQ7 chip, the DC output would give off around 5 Volts (1023 as read by the Arduino). When no music was playing, the chip would give off a DC voltage around .1 Volts (around 32 as read by the Arduino), and when a moderate volume of music was being played, the chip would output a DC voltage around 3-4 Volts (600-800 as read by the Arduino). We then assigned these values to then be accurately displayed on the LED display so that strong signals would turn on all of the LEDs, whereas no signal would turn off all of the LEDs. We were also able to get one column of the cube's screen to update 2 milliseconds at a time. Then after updating the rightmost column of the cube's screen, it takes 4 milliseconds to start updating the leftmost column. All in all it takes 16 milliseconds to update each column after last updating it. This is obviously too fast for human eyes, so the visualizer appears smooth.

4. Future Work

i. In the future we will create a housing for the music visualizer, so that it actually is a cube instead of a screen made of LEDs. We also could add more features like an on-board microphone, scrolling text across the screen, or adding pre-programmed light shows.

5. Conclusion

a. What Worked

i. The visualizer idea we had worked. The screen updates smoothly and nicely. The visualizer actually looks like a visualizer. The code worked after a few bug fixes and there were no soldering errors.

The LED dimming function works as intended. It is as if the stars aligned and the universe wanted the project to work.

b. **What Didn't Work**

   i. First we had problems with the wiring of the bandpass filter. We soon figured out that the problem was the filter was receiving no power. Then we had problems with the code, first we realized we could not use ports 0 and 1 on the arduino because they could not send power through both ports at once. Then we realized that the visualizer was not updating properly because we forgot to reset the counter to zero after resetting the filter, so the counter kept going up and up. Also we decided to make a screen instead of a cube to allow people to see the visualization more clearly.

c. **What We Learned**

   i. We learned to combine and apply our knowledge learned in ECE 110 about circuits, knowledge learned in ECE 120 about multiplexers and logic, and our previously learned knowledge about coding and soldering to make a fully functional project. We also learned more about the process of doing a project over an entire semester and how to get our project working by a specific time. We also learned that sometimes electronic projects do work out with minimal issues, which is a rare occurrence for both of us.