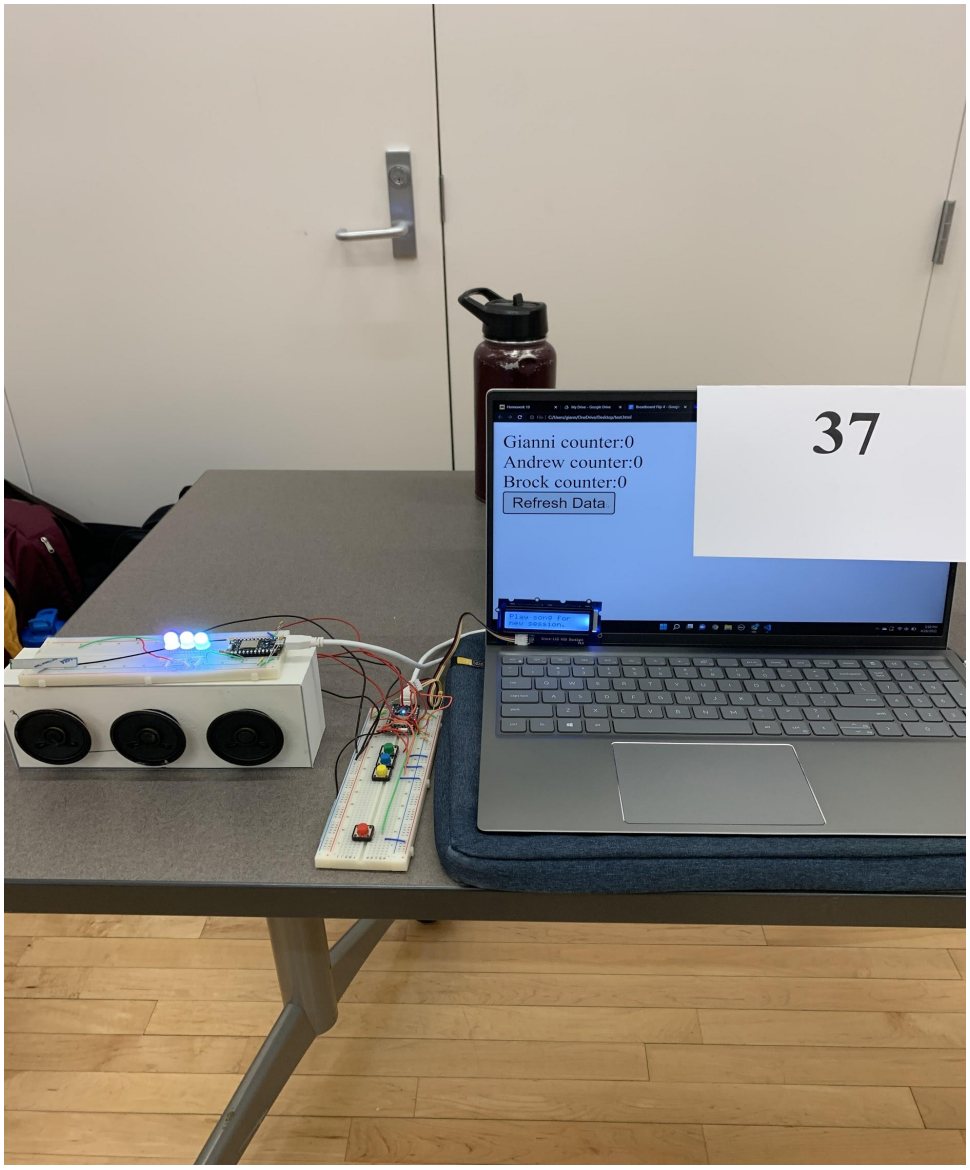
OBJ

BreadBoard Flip 4

Andrew Echlin, Brock Bye, Gianni Guadagno



Overview

Problem

The problem the Breadboard Flip 4 solves is an argument between a group of friends to decide which song is the most listened to among the group, specifically when the group is spending time together in person.

Solution

This project is able to accomplish this goal using 4 sensors and 7 actuators. Those being 4 buttons, 3 speakers, 3 RGB LEDs, 1 LCD display, and additionally a HTML webpage. The Bread Board Flip 4 is a speaker system with 3 songs encoded into the device ready to be played on demand. Each time a button is pressed, the song correlated to that button is played and the counter for that song is increased by one. The counter is displayed via a HTML website and the song currently being played is shown on the LCD display. While this session is occurring, for the show of decorations, we have three RGB LEDs blaring randomly at random intervals because lights are a must for a speaker system. At the end of the listening, someone presses the red button on the edge and it displays the winner, or which song was played the most in that listening session. An email is sent to Gianni Guadagno announcing the winner, and the counters are then set back to zero and the next listening session awaits.

Electronic Components

Internet connectivity

<u>If This Then That (IFTTT)</u>

We initially decided to use IFTTT to display our song counts and the winner of each round of songs played, but ran into an issue where IFTTT was not notifying Gianni with an email within the same day. So we decided to additionally use an HTML Webpage to display results concurrently with IFTTT sending emails. Interestingly, we got our first IFTTT email about 15 mins after the IoT Showcase ended. In Photon #1 we defined particle variables

```
Particle.variable("Gianni counter", Gcounter);
Particle.variable("Andrew counter", Acounter);
Particle.variable("Brock counter", Bcounter);
```

And each time

```
int McurrButton = digitalRead(MusicState);
    if(McurrButton == HIGH && MprevButton == LOW){  // When red button or last button is pressed
    lcd.clear(); //Clear LCD display
```

```
    lcd.setCursor(0, 0); //Puts cursor in top left corner
    lcd.print("Session Finished:");
    lcd.setCursor(0, 1); //Puts cursor in second line
    if ( Gcounter > Acounter && Gcounter > Bcounter){   //if G counter is
pressed the most displays yesterday as winner
    lcd.print("#1 = Yesterday");
    Particle.publish("Yesterday Wins");
    }else if ( Acounter > Gcounter && Acounter > Bcounter){ //if A counter
is pressed the most display HotlineBling as winner
    lcd.print("#1 = Hlb");
    Particle.publish("Hotlinebling Wins");
    } else if (Bcounter > Gcounter && Bcounter > Acounter){ //if B counter
is pressed the most display Pianoman as winner
     lcd.print("#1 = Pianoman");
     Particle.publish("Pianoman Wins");
     } else {
     lcd.print("Tie"); //if there is a tie for first display tie
     }

     Gcounter = 0;  //resets counters
     Bcounter = 0;
     Acounter =0;
```

HTML Webpage

In order to display the count of each time a song is played, we use an HTML Webpage that dynamically keeps track, displaying the count with each refresh. In Photon #1 we defined particle variables

```
Particle.variable("Gianni counter", Gcounter);
Particle.variable("Andrew counter", Acounter);
Particle.variable("Brock counter", Bcounter);
```

Since Gianni has his photon access code & device ID in the code, the HTML page can dynamically update with each page refresh.

```
var baseURL = "https://api.particle.io/v1/devices/";
```

Push Buttons

Connected to Photon #1, 4 buttons are located on the breadboard. In order to set up these buttons we first need to connect the pin numbers to their own integer value.

```
int ButtonG = A0;
```

Then we also need to set all of the button's previous states to low for when it is initially run.

```
int GprevButton = LOW;
```

This is always going to end up being low and will be explained later. In order, to actually get inputs from the buttons we need to first recognize the buttons in the void setup.

```
pinMode(ButtonG, INPUT_PULLDOWN);
```

Then we need to use the digital read function to take in inputs for the buttons. This takes place within the void loop function.

```
int GcurrButton = digitalRead(ButtonG);
```

3 of these buttons are song buttons for the songs: Yesterday by Atmosphere, Hotline Bling by Drake, and Piano Man by Billy Joel. These are eventually played by pushing the green, yellow and blue buttons. Each button has their own if statements for when they are pressed. After initializing all the buttons' previous and current states to low and reading in inputs from the buttons, we wait until a button is pressed.

```
if(GcurrButton == HIGH && GprevButton == LOW)
```

When one of the 3 song buttons is pressed, many things occur within the if statement. First, the counter for the song is increased by one, which is displayed on the HTML website. Then the LCD display shows the song that is being played, and while this is going on, the songs are being played.

```
Gcounter = Gcounter + 1;
```

Then, each actuator is activated in its own way according to which button is pressed. Finally, the state resets back to low, so that they can be pressed again.

```
GcurrButton = GprevButton;
```

The red button, or the session end button, follows the exact same code to be activated as the three song buttons. However, when this button is pressed the function is very different. First, it notifies the user from the LCD display that the session has finished. Then it runs an if-else statement to figure out which song won. If there is a tie, the display tells us there is a tie for the number one song played in the session. Afterwards all the counters are reset back to zero and the user is prompted to start a session by playing a new song.

In our original plan, we wanted to have state machines for each song, so that when we try to activate a state, the song is easily switched as well as all the other functions along with it. We did this and we found out that some states had priority over other states and that there was a lot going on internally that we did not know.

LCD Display

Also occurring in Photon #1, we have an LCD display (JHD1313M3) with the role of displaying each song being played as well as updating the user on certain events, such as ending a session or being prompted to start a new one. In order to use this we need the LCD's library.

```
#include <Grove_LCD_RGB_Backlight.h>
```

Next, the LCD library is called out using the statement below. This allows the library to be accessed everytime the function "lcd." is called. This function converts human code to an LCD readable statement.

```
rgb_lcd lcd;
```

Within our void setup, we needed to initialize the LCD to have 16 characters and 2 lines. As well as set up the color function to be able to display letters.

```
lcd.begin(16, 2);
lcd.setRGB(colorR, colorG, colorB);
```

Then we were able to print many messages throughout the project's showcase. First, in order to start a message on the first line we use the command

```
lcd.setCursor(0, 0);
```

The first number marking the position of the character the pointer starts at and the second number represents the line the LCD is setting the cursor to. After the cursor is set we clear the screen and output the message that we want to display. An example sequence is shown below.

```
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Now Playing:");
lcd.setCursor(0, 1);
lcd.print("Yesterday");
```

We use this exact style throughout every message we need to display throughout the music session.

Songs

Within Photon #1, once a song button has been pushed, a for loop is triggered and plays the specified song between three 2" round speakers (16 ohms, 82dB, 2W). We initialized the variables: speakerPin(1, 2, and 3) to their respective digital outputs (D2, D3, D4) as global variables. Key components of information needed to output songs are the specified speaker pin, and two arrays (the Melody, and the Note Duration), which are used to call the tone function and produce the note within the for loop.

```
for (int thisNote = 0; thisNote < arraySize(yesterdayM1); thisNote++)
    tone(speakerPin1, yesterdayM1[thisNote],noteDuration1);
```

The three speakers together will play a chord that goes along with the song's piano sheet music. The melody array contains the frequencies converted from the piano keys.

```
yesterdayM1[thisNote]
```

The note duration array contains the type of note (quarter note is 4 and such), which is later converted to the noteDuration integer into actual time in seconds (1500/4). Using the built in tone function, we output the frequency & duration to the specified speaker.

```
int noteDuration1 = 1500/yesterdayNote1[thisNote];
```

Between each note being played is a pause (pauseBetweenNotes) that smoothly transitions from key to key with this natural pause, like actually playing the piano. We then use the delay function by using the pauseBetweenNotes as the time.

```
int pauseBetweenNotes1 = noteDuration1 * 1.30;
delay(pauseBetweenNotes1);
```

Finally, stop playing the note by calling the noTone function.

```
noTone(speakerPin1);
```

One technical difficulty we encountered was trying to figure out a way to get 8 bit music in code. So what we figured out was to get a song played in piano, and then from the song's sheet music, convert each key into frequency and keep track of the note duration, in separate lists.

LEDs: Digi-Key Part Number: 1568-1215-ND

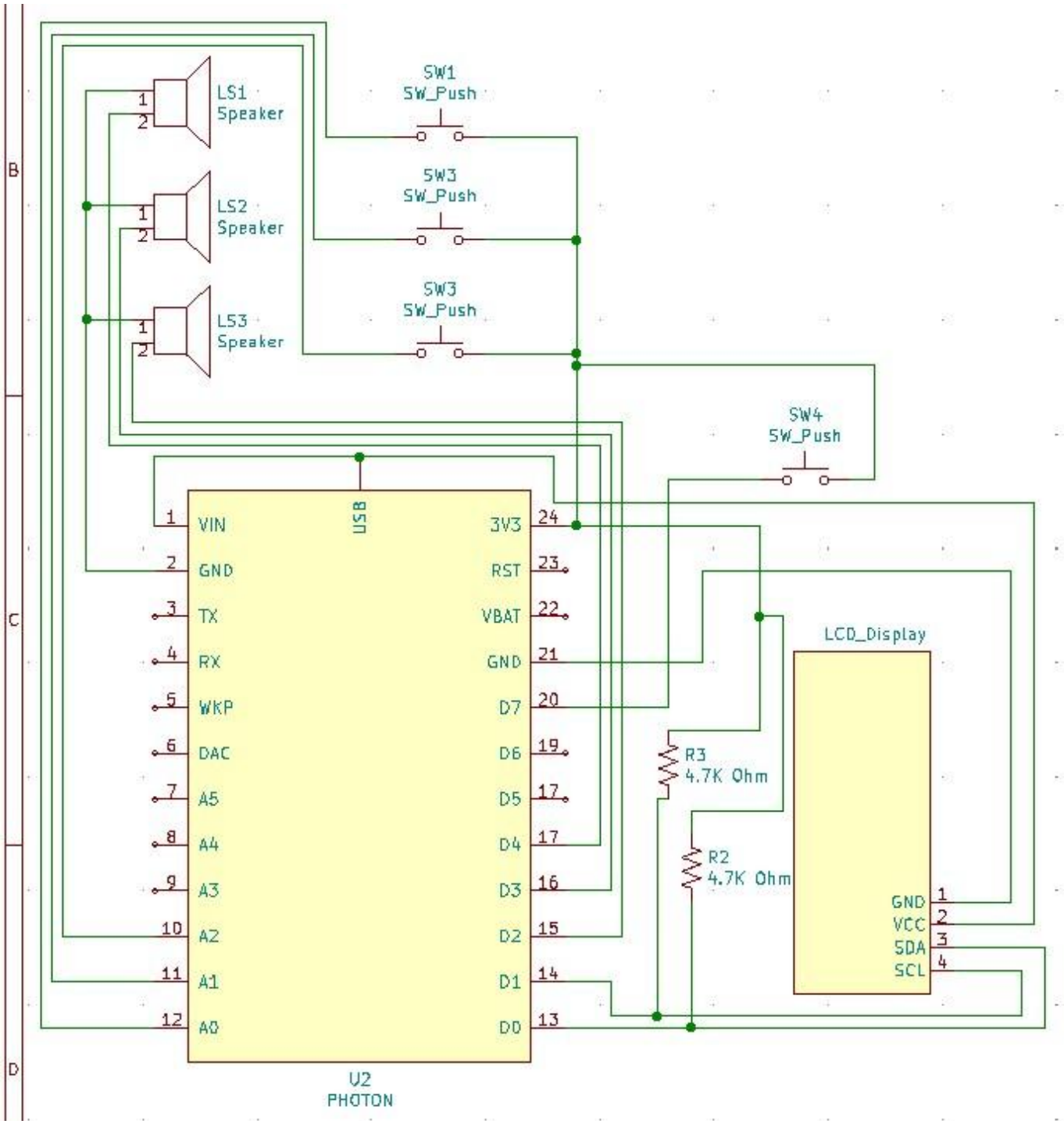
In our project, 3 LEDs were placed on top of our speaker box and flashed continuously at random rates and frequencies. The programming for these uses a loop function that changes the LED color each cycle. Each RGB variable is reassigned each loop using a statement like the following: `B3 = random(256);` The variable "B3" stands for Blue 3, where Blue is the RGB color, and 3 is the LED #. This type of function is used for all LEDs (R1,B1…B3,G3). After this, a random length of delay is called using the function: `delay1 = random(100);`. This variable is later called as a block function `delay(delay1);` and it therefore will keep this color scheme for a random amount of time before repeating the loop. Before the delay function is called, the LEDs colors change using the function `int LED3 = strip.Color( R3, G3, B3);`. Lastly, the LED colors are switched by the statement `strip.setPixelColor(2, LED3);` and then `strip.show();`. The previous statement assigns the color scheme, LED3, to one of the LEDs, #2. In order to use some of these commands, the neopixel library must be downloaded and stated in the program using the statement `#include "neopixel.h".`

The LED code was programmed on a different photon. And for this reason, it has no correlation with the rest of the code and runs on its own. The LEDs are on a separate photon because the blocking functions discussed earlier would interrupt the playing of the music, therefore, making it extremely difficult to combine these programs into one photon with our time frame.

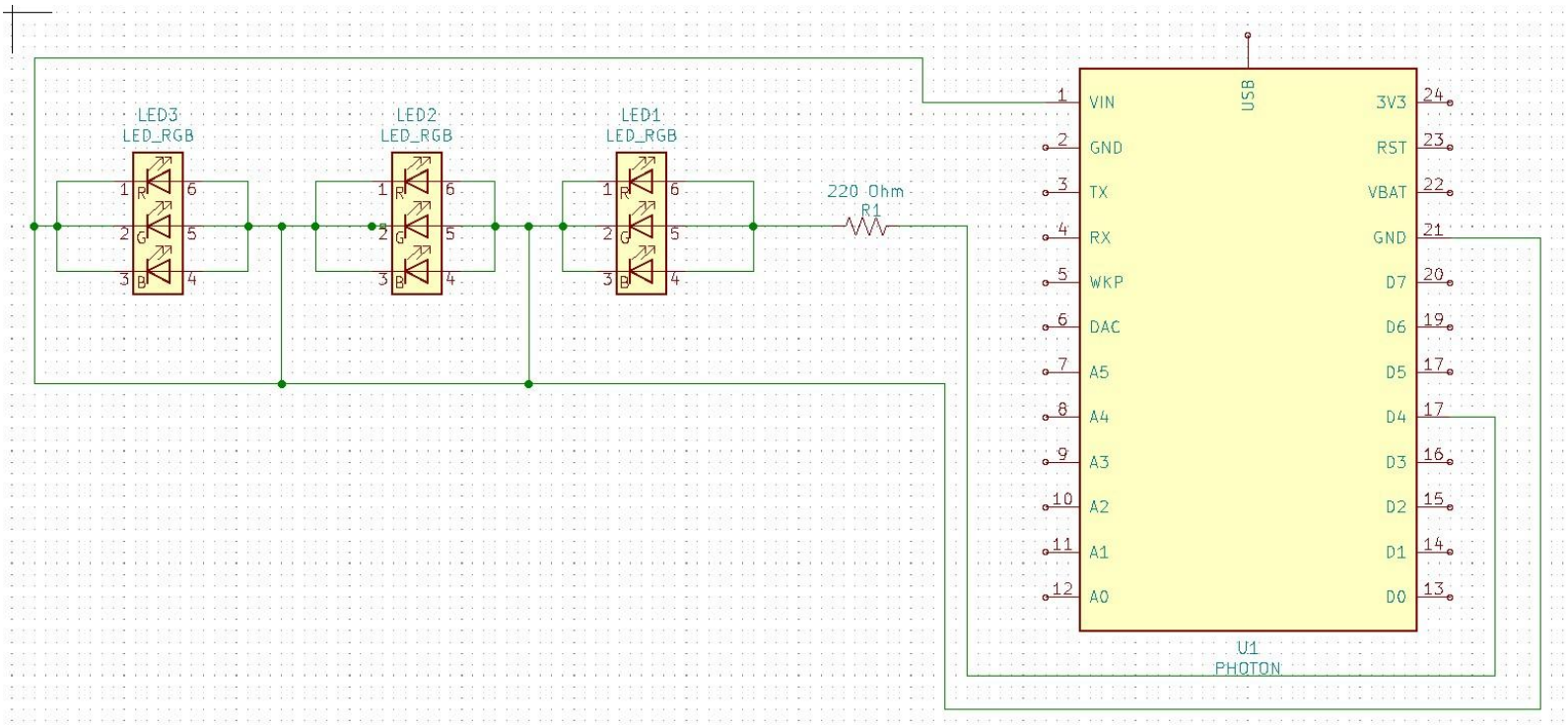
Appendix

Schematics:

Photon #1: Includes 4 buttons, 3 speakers, 3 resistors, and an LCD display



Photon #2: Includes 3 RGB LEDs, and a resistor



Main Code:

```
// Breadboard Flip 4
//Gianni Guadagno, Brock Bye, Andrew Echlin
// Photon #1

#include <Grove_LCD_RGB_Backlight.h>

rgb_lcd lcd;

const int colorR = 255;
const int colorG = 255;
const int colorB = 255;

int ButtonG = A0; //Lists pins for each button
int ButtonA = A1;
int ButtonB = A2;
int MusicState = D7;
```

```
int GprevButton = LOW;  // Initalized all the buttons to low before being pressed
int AprevButton = LOW;
int BprevButton = LOW;
int MprevButton = LOW;

int Gcounter = 0; //Initalized each song counter Yesterday,  and .
int Acounter= 0;
int Bcounter = 0;

 int speakerPin1 = D2;  //Initalized the 3 pins for out speakers.
 int speakerPin2 = D3;
 int speakerPin3 = D4;

//Music pitches arrays for speaker 1,2 and 3 for the song yesterday
 double yesterdayM1[] =
{1244.508,1318.51,1174.659,1046.502,1046.502,783.9909,622.254,783.9909,622.254,783.9909,587.3
295,523.2511,440,391.9954,440,523.2511,523.2511,523.2511,523.2511,391.9954,440,493.8833,415.3
047,523.2511,415.3047,523.2511,523.2511,783.9909,880,1046.502
};
double yesterdayM2[] =
{0,440,391.9954,329.6276,293.6648,261.6256,0,220,0,220,195.9977,164.8138,329.6276,0,146.8324,
146.8324,195.9977,174.6141,174.6141,130.8128,0,0,0,0,0,0,0,0,0,0
};
double yesterdayM3[] = {
0,220,195.9977,164.8138,146.8324,130.8128,0,110,0,110,97.99886,82.40689,293.6648,0,73.41619,7
3.41619,97.99886,87.30706,87.30706,65.40639,0,0,0,0,0,0,0,0,0,0
};
//Note duration arrays for yesterday
double yesterdayNote1[] =
{64,8,8,8,4,8,64,4,64,8,8,8,2.6667,16,16,8,8,4,2.6667,16,16,8,64,8,64,4,2.6667,16,16,8
};
double yesterdayNote2[] =
{64,8,8,8,4,8,64,4,64,8,8,8,2.6667,4,8,4,1.6,8,4,1.6,16,16,8,64,8,64,4,2.6667,16,16,8
};
double yesterdayNote3[] =
{64,8,8,8,4,8,64,4,64,8,8,8,2.6667,4,8,4,1.6,8,4,1.6,16,16,8,64,8,64,4,2.6667,16,16,8
};
```

```
// //Hotline Bling Melody & Note Duration Arrays
double hotlineBlingM1[] =
{698.45,1046.5,1046.5,1046.5,880,1046.5,1046.5,1174.6,0,349.23,698.45,880,784,698.45,784,784,
880,0,0,1046.5,1046.5,1046.5,880,1046.5,1046.5,1174.7,0,0,0,698.45,880,784,698.45,784,784,880
,1046.5,880,1046.5,880,1046.5,880,1046.5,880,1174.66,880,880,880,880,932.32,880,784,698.45,78
4,784,880,880,880,880,932.32,880,784,698.45,784,784,880,880,880,880,932.32,880,784,698.45,784
,784,698.45,1046.5,880,1046.5,880,1046.5,880,1046.5,880,1174.66,1174.66,880,880,880,880,932.3
2,880,784,698.45,784,784
};
double hotlineBlingM2[] =
{0,0,0,0,0,698,698,698,0,0,440,0,0,523,523,523,0,0,0,0,0,0,698,698,698,0,0,0,698,880,784,698,
784,784,880,1046,880,1046,880,1046,880,1046,880,587,587,0,0,0,0,0,0,0,523,523,659,0,0,0,0,0,0
,0,466,466,466,0,0,0,0,0,0,0,523,523,0,0,0,0,0,0,0,0,0,587,587,466,0,0,0,0,0,0,0,523,523
};
double hotlineBlingM3[] =
{0,0,0,220,220,349,440,0,0,0,0,116.5,110,329.6,392,110,329.6,392,110,116.5,174.6,220,116.5,17
4.6,220,116.5,110,165,196,0,116.5,147,233,261.6,293.7,0,0,174.6,110,165,220,261.6,329.6,0,0,0
,0,165,110,98,147,196,233,293.7,0,0,196,110,165,220,261.6,329.6,0,0,116.5,174.6,220,233,261.6
,293.7,349,440,0,0,0,0,0,174.6,110,165,220,261.6,329.6,220,261.6,0,0,0,0,0,0,0,0,0,0
};
double hotlineBlingNote1[] =
{8,8,8,8,8,8,8,1.66,2,8,8,8,8,8,8,8,1.66,2,8,8,8,8,8,8,8,1.2658,8,2,8,8,8,8,8,8,0.8,8,8,8,8
,8,8,8,8,1,8,8,8,8,8,8,8,8,4,1.3333,8,8,8,8,8,8,8,8,4,1.333,8,8,8,8,8,8,8,8,4,1.333,8,8,8,8,8
,8,8,8,8,1.1428,8,8,8,8,8,8,8,8,8,4,2.125
};
double hotlineBlingNote2[] =
{8,8,8,8,8,8,8,1.662,8,4,8,8,8,8,1.66,2,8,8,8,8,8,8,8,1.2658,8,2,8,8,8,8,8,8,0.8,8,8,8,8,8,
8,8,8,1,8,8,8,8,8,8,8,8,4,1.3333,8,8,8,8,8,8,8,8,4,1.333,8,8,8,8,8,8,8,8,4,1.333,8,8,8,8,8,8,
8,8,8,1.1428,8,8,8,8,8,8,8,8,8,4,2.125
};
double hotlineBlingNote3[] =
{8,8,8,4,2.666,8,2.666,2,8,8,8,4,2.666,8,2,2.666,8,4,4,2.66,8,2,2.666,8,4,4,2.666,8,1.15,1,8,
8,8,8,2,4,4,2,8,8,8,8,2,8,8,8,8,4,4,8,8,8,8,2,4,4,2,8,8,8,8,2,1,8,8,8,8,8,8,8,8,8,4,8,8,8,4
,8,8,8,8,8,8,8,8,8,8,4,8,8,8,8,8,8,8
};

// //Piano Man Melody & Note Duration Arrays
double pianoManM1[] =
{0,0,392,392,392,392,349,329.6,349,329.6,261.6,0,0,261.6,261.6,261.6,261.6,261.6,261.6,293.6,
```

```
293.6,0,392,392,392,392,392,349,329.6,349,329.6,261.1,220,220,220,261.6,329.6,349.2,329.6,293
.6,261.6,0,0,0,0,0,0,0,0
};
double pianoManM2[] =
{349,329.6,293.6,0,329.6,329.6,0,293.6,293.6,0,261.6,261.6,0,261.6,261.6,0,261.6,261.6,0,261.
6,261.6,0,370,370,392,293.6,247,293.6,0,329.6,329.6,0,293.6,293.6,0,261.6,261.6,0,261.6,261.6
,0,261.6,261.6,0,261.6,261.6,0,329.6,329.6
};
double pianoManM3[] =
{174.6,164.8,146.8,130.8,261.6,261.6,123.5,247,247,110,220,220,98,196,196,87.3,220,220,82.4,1
96,196,73.4,220,220,98,196,247,131,261.6,261.6,123.47,247,247,110,220,220,98,196,196,87.3,220
,220,98,196,196,0,0,0,0
};
double pianoManNote1[] =
{4,4,4,2,4,4,2.66,8,8,8,2,4,4,4,4,2.66,8,4,8,2.66,1,8,8,8,8,4,1,8,8,4,8,1.66,8,8,2,8,8,4,4,4,
1.33,4,4,4,4,4,4,4,4
};
double pianoManNote2[] =
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,8,8,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4
};
double pianoManNote3[] =
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4
};

// setup() runs once, when the device is first turned on.
void setup() {
    lcd.begin(16, 2);

    lcd.setRGB(colorR, colorG, colorB);

    pinMode(ButtonG, INPUT_PULLDOWN); // INPUT mode with internal pull-down resistor
    pinMode(ButtonA, INPUT_PULLDOWN);
    pinMode(ButtonB, INPUT_PULLDOWN);
    pinMode(MusicState, INPUT_PULLDOWN);
    Particle.variable("Gianni counter", Gcounter); //Puts counter Variables into the cloud
    Particle.variable("Andrew counter", Acounter);
    Particle.variable("Brock counter", Bcounter);
```

```
    Serial.begin(9600);               // Use Serial port for debugging

}
// loop() runs over and over again, as quickly as it can execute.
void loop() {

    int GcurrButton = digitalRead(ButtonG);
    if(GcurrButton == HIGH && GprevButton == LOW){  //When G Button or the First Button is
pressed
      Gcounter = Gcounter + 1; // Increase G counter
      lcd.clear();  //Clear LCD display
      lcd.setCursor(0, 0); //Puts cursor in top left corner
      lcd.print("Now Playing:");
      lcd.setCursor(0, 1); //Puts cursor in second line
      lcd.print("Yesterday");
       for (int thisNote = 0; thisNote < arraySize(yesterdayM1); thisNote++) {

    // to calculate the note duration, take one second
    // divided by the note type.
    //e.g. quarter note = 1500 / 4, eighth note = 1000/8, etc.
      int noteDuration1 = 1500/yesterdayNote1[thisNote];
      int noteDuration2 = 1500/yesterdayNote2[thisNote];
      int noteDuration3 = 1500/yesterdayNote3[thisNote];

      tone(speakerPin1, yesterdayM1[thisNote],noteDuration1);
      tone(speakerPin2, yesterdayM1[thisNote],noteDuration2);
      tone(speakerPin3, yesterdayM1[thisNote],noteDuration3);
    // to distinguish the notes, set a min time between them.
    // the note's duration + 30% seems to work well:
      int pauseBetweenNotes1 = noteDuration1 * 1.30;
      delay(pauseBetweenNotes1);

    // stop the tone playing:
      noTone(speakerPin1);
      noTone(speakerPin2);
      noTone(speakerPin3);
    }
    }
```

```
GcurrButton = GprevButton; //Sets button state



int AcurrButton = digitalRead(ButtonA);
    if(AcurrButton == HIGH && AprevButton == LOW){     //When A Button or the Second Button
is pressed
    Acounter = Acounter + 1;  // Increase G counter
    lcd.clear();   //Clear LCD display
    lcd.setCursor(0, 0);     //Puts cursor in top left corner
    lcd.print("Now Playing:");
    lcd.setCursor(0, 1);    //Puts cursor in second line
    lcd.print("HotlineBling");
    for (int thisNote = 0; thisNote < arraySize(hotlineBlingM1); thisNote++) {

    // // to calculate the note duration, take one second
    // // divided by the note type.
    //e.g. quarter note = 1500 / 4, eighth note = 1000/8, etc.
      int noteDuration1 = 1500/hotlineBlingNote1[thisNote];
      int noteDuration2 = 1500/hotlineBlingNote2[thisNote];
      int noteDuration3 = 1500/hotlineBlingNote3[thisNote];

      tone(speakerPin1, hotlineBlingM1[thisNote],noteDuration1);
      tone(speakerPin2, hotlineBlingM2[thisNote],noteDuration2);
      tone(speakerPin3, hotlineBlingM3[thisNote],noteDuration3);
    // to distinguish the notes, set a min time between them.
    // the note's duration + 30% seems to work well:
      int pauseBetweenNotes1 = noteDuration1 * 1.30;
      delay(pauseBetweenNotes1);

    // stop the tone playing:
      noTone(speakerPin1);
      noTone(speakerPin2);
      noTone(speakerPin3);
    }
    }
AcurrButton = AprevButton; //Sets button state

int BcurrButton = digitalRead(ButtonB);
```

```
    if(BcurrButton == HIGH && BprevButton == LOW){ //When B Button or the Third Button is
pressed
    Bcounter = Bcounter + 1; // Increase B counter
    lcd.clear(); //Clear LCD display
    lcd.setCursor(0, 0); //Puts cursor in top left corner
    lcd.print("Now Playing:");
    lcd.setCursor(0, 1); //Puts cursor in second line
    lcd.print("Pianoman");
    for (int thisNote = 0; thisNote < arraySize(pianoManM1); thisNote++) {

    // to calculate the note duration, take one second
    // divided by the note type.
    //e.g. quarter note = 1500 / 4, eighth note = 1000/8, etc.
      int noteDuration1 = 1500/pianoManNote1[thisNote];
      int noteDuration2 = 1500/pianoManNote2[thisNote];
      int noteDuration3 = 1500/pianoManNote3[thisNote];

      tone(speakerPin1, pianoManM1[thisNote],noteDuration1);
      tone(speakerPin2, pianoManM2[thisNote],noteDuration2);
      tone(speakerPin3, pianoManM3[thisNote],noteDuration3);
    // to distinguish the notes, set a min time between them.
    // the note's duration + 30% seems to work well:
      int pauseBetweenNotes1 = noteDuration1 * 1.2;
      delay(pauseBetweenNotes1);

    // stop the tone playing:
      noTone(speakerPin1);
      noTone(speakerPin2);
      noTone(speakerPin3);
    }
    }
BcurrButton = BprevButton; //Sets button state

int McurrButton = digitalRead(MusicState);
    if(McurrButton == HIGH && MprevButton == LOW){  // When red button or last button is
pressed
    lcd.clear(); //Clear LCD display
    lcd.setCursor(0, 0); //Puts cursor in top left corner
    lcd.print("Session Finished:");
```

```
    lcd.setCursor(0, 1); //Puts cursor in second line
    if ( Gcounter > Acounter && Gcounter > Bcounter){   //if G counter is pressed the most
displays yesterday as winner
    lcd.print("#1 = Yesterday");
    Particle.publish("Yesterday Wins");
    }else if ( Acounter > Gcounter && Acounter > Bcounter){ //if A counter is pressed the
most display HotlineBling as winner
    lcd.print("#1 = Hlb");
    Particle.publish("Hotlinebling Wins");
    } else if (Bcounter > Gcounter && Bcounter > Acounter){ //if B counter is pressed the
most display Pianoman as winner
     lcd.print("#1 = Pianoman");
     Particle.publish("Pianoman Wins");
     } else {
     lcd.print("Tie"); //if there is a tie for first display tie
     }

     Gcounter = 0;  //resets counters
     Bcounter = 0;
     Acounter =0;
     delay(5000);
     lcd.clear(); //Clear LCD display
     lcd.setCursor(0, 0); //Puts cursor in top left corner
     lcd.print("Play song for");
     lcd.setCursor(0,1); //Puts cursor in second line
     lcd.print("new session."); //ready for new session to begin
     }
}
```

LED CODE:

```
// Breadboard Flip 4
//Gianni Guadagno, Brock Bye, Andrew Echlin
// Photon #2
```

```
#include "Particle.h"
#include "neopixel.h"

SYSTEM_MODE(AUTOMATIC);

// IMPORTANT: Set pixel COUNT, PIN and TYPE
#define PIXEL_PIN D4
#define PIXEL_COUNT 3
#define PIXEL_TYPE WS2812

Adafruit_NeoPixel strip(PIXEL_COUNT, PIXEL_PIN, PIXEL_TYPE);

// Prototypes for local build, ok to leave in for Build IDE
void rainbow(uint8_t wait);
uint32_t Wheel(byte WheelPos);

void setup()
{
  strip.begin();
  strip.show();// Initialize all pixels to 'off'
}
 void loop()
{
    int delay1;
    int R1,R2,R3;//Initialize all Variables
    int G1,G2,G3;
    int B1,B2,B3;

   //set delay time to rand variable 0-.1s
   //set LED frequency randomly between 0-255
    delay1 = random(100);
    R1 = random(256);
    R2 = random(256);
    R3 = random(256);
    G1 = random(256);
    G2 = random(256);
    G3 = random(256);
    B1 = random(256);
    B2 = random(256);
```

```
    B3 = random(256);

   //Assign the LED to generated frequencies
    int LED1 = strip.Color(R1, G1, B1);
    int LED2  = strip.Color(R2, G2, B2);
    int LED3 = strip.Color(R3, G3, B3);

   //Refine Pixel Color of LED
    strip.setPixelColor(0, LED1);
    strip.setPixelColor(1, LED2);
    strip.setPixelColor(2, LED3);
    strip.show();//Show the new color
    delay(delay1);//Wait for random time
}
```

HTML File Code:

```
<!DOCTYPE html>
<html>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"
type="text/javascript" charset="utf-8"></script>
<body>
    Gianni counter:<span id="Gianni counter"></span><br>
    Andrew counter:<span id="Andrew counter"></span><br>
    Brock counter:<span id="Brock counter"></span><br>


    <button id="connectbutton" onclick="start()">Refresh Data</button>

    <script type="text/javascript">
    function start(objButton) {
        var deviceID = "2a0032001047393334363636";
        var accessToken = "13f8d7aa25ff232aa1c47c4882d729ec7856720f";
        var baseURL = "https://api.particle.io/v1/devices/";

        var varName = "Gianni counter";
        requestURL = baseURL + deviceID + "/" + varName + "/?access_token=" + accessToken;
        $.getJSON(requestURL, function(json) {
```

```
            document.getElementById("Gianni counter").innerHTML = + json.result;
            });

        var varName = "Andrew counter";
        requestURL = baseURL + deviceID + "/" + varName + "/?access_token=" + accessToken;
        $.getJSON(requestURL, function(json) {
            document.getElementById("Andrew counter").innerHTML = + json.result;
            });

        var varName = "Brock counter";
        requestURL = baseURL + deviceID + "/" + varName + "/?access_token=" + accessToken;
        $.getJSON(requestURL, function(json) {
            document.getElementById("Brock counter").innerHTML = + json.result;
            });
    }
    </script>
<br>
</body>
</html>
```