Overwatch Prop Gun: Lucio's Blaster Pt. 2

Created by John Park


Last updated on 2023-08-29 03:16:40 PM EDT
# Table of Contents

**Blaster Integration: Lights and Music**

- Lights and Music

**Blaster Integration: Audio Effects**
Blaster Integration: Lights and Music

This is the second part in the series on creating the Overwatch prop blaster for Lucio. You can check out the first part here.

In learning how to build a multi-function prop we started off by making individual, separate circuits for three things: lighting, music, and sound effects. I like to start off with discreet circuits, each running with its own microcontroller and power supply. This helps to get things up and running quickly and is straightforward to troubleshoot.
These three, separate circuits create the lighting, music, and audio effects for the prop. Let's integrate them into a single system.

Now that those individual parts are working, it's time to combine them into a single, integrated system. This integration covers a number of elements, including doing multiple tasks with a single microcontroller, sharing inputs, such as a trigger pull creating both a sound effect and a lighting change, and sharing power supplies.
Leaping off from the previous lesson, you can first combine the music circuit and the lighting circuit. They each ran from their own Arduino UNO or Adafruit Metro board in part 1, but now we'll run both circuits from a single board.

This time, we're going to use a smaller 3W amplifier connected to the MP3 shield for the background music playing through the three small speakers. We'll continue to use the 20W amp to pump the shooting sound effects from the Audio FX board through
the big speaker in the front of the gun. Keeping these two sound circuits sepeparated allows us to have stereo music channels playing through the left and right small speakers, and still create loud mono effects through the front speaker.

The Arduino sketch needs to be adjusted so that it will both play back music and light the LEDs. Below is a simple sketch to do this, as well as switch modes when the toggle is flipped. Copy it and upload it to the UNO or Metro.

```c
//Lucio Blaster code for UNO and Metro
// by John Park for Adafruit
//based upon:
//****************************************************************************
 Adafruit VS1053 Codec Breakout

 Designed specifically to work with the Adafruit VS1053 Codec Breakout
 ---->&gt; https://www.adafruit.com/products/1381

 Written by Limor Fried/Ladyada for Adafruit Industries.
 BSD license, all text above must be included in any redistribution
 ****************************************************************************/
 #include &lt;Adafruit_VS1053.h&gt;
 #include &lt;SD.h&gt;
 #include &lt;Adafruit_NeoPixel.h&gt;

 #define NEOPIN 2 // pin for NeoPixel ring
 #define NUMPIXELS 24 //adjust depending on ring size

 Adafruit NeoPixel pixels =
 Adafruit_NeoPixel(NUMPIXELS, NEOPIN, NEO_GRBW + NEO_KHZ800);
 int NEODELAY = 0; // 8 millis is 120 BPM

 #define NEOVUPIN 8
 #define NUMVUPIXELS 8
 Adafruit NeoPixel vupixels =
 Adafruit_NeoPixel(NUMVUPIXELS, NEOVUPIN, NEO_GRBW + NEO_KHZ800);
 int NEOVUDELAY = 42; // the tempo for lighting effects

 #define NEOJWLPIN 10
 #define NUMJWLPIXELS 5
 Adafruit NeoPixel jewelpixels =
 Adafruit_NeoPixel(NUMJWLPIXELS, NEOJWLPIN, NEO_GRBW + NEO_KHZ800);
 int NEOJWLDELAY = 84; // the tempo for lighting effects

 // These are the pins used for the music maker shield
 #define SHIELD_RESET -1 // VS1053 reset pin (unused!)
 #define SHIELD_CS 7 // VS1053 chip select pin (output)
 #define SHIELD_DCS 6 // VS1053 Data/command select pin (output)

 // These are common pins between breakout and shield
 #define CARDCS 4 // Card chip select pin
 // DREQ should be an Int pin, see http://arduino.cc/en/Reference/attachInterrupt
 #define DREQ 3 // VS1053 Data request, ideally an Interrupt pin

 Adafruit_VS1053_FilePlayer musicPlayer =
 Adafruit_VS1053_FilePlayer(SHIELD_RESET, SHIELD_CS, SHIELD_DCS, DREQ, CARDCS);

 // state change detection
 const int MODESWITCHPIN = 5; // the pin that the pushbutton is attached to
 int modeSwitchPushCounter = 0; // counter for the number of button presses
 bool modeSwitchState = 0; // current state of the button
 int lastModeButtonState = 0; // previous state of the button

 const char* SONGS[] = {"h.mp3", "s.mp3"}; // change to reflect your file names
```
int songPick = 0;
const int VOL = 1; // volume higher numbers are quieter

void setup() {
pixels.begin(); // This initializes the NeoPixel library.
pixels.setBrightness(30);
pixels.show();

vupixels.begin();
vupixels.setBrightness(20);
vupixels.show();

jewelpixels.begin();
jewelpixels.setBrightness(10);
jewelpixels.show();

pinMode(MODESWITCHPIN, INPUT_PULLUP);

Serial.begin(9600);

// initialise the music player
if (!musicPlayer.begin()) {
    Serial.println(F("Couldn't find VS1053, do you have right pins defined?") );
    while (1);
}

if (!SD.begin(CARDCS)) {
    Serial.println(F("SD failed, or not present"));
    while (1); // don't do anything more
}

Serial.println(F("SD OK!"));

// Set volume for left, right channels. lower numbers == louder volume!
musicPlayer.setVolume(VOL, VOL);

if (!musicPlayer.useInterrupt(VS1053_FILEPLAYER_PIN_INT))
    Serial.println(F("DREQ pin is not an interrupt pin"));

// check the switch state to start up with a NeoPixel color
modeSwitchState = digitalRead(MODESWITCHPIN); // read switch input pin
if (modeSwitchState==HIGH){
    for(int i=(NUMPIXELS-1);i>=0;i--){ // change the color "left to right" sweep
        pixels.setPixelColor(i, pixels.Color(150,120,0,10)); // yellow color
        pixels.show(); // This sends the updated pixel color to the hardware
        delay(NEODELAY); // Delay for a period of time (in milliseconds)
    }

    for(int i=(NUMJWLPIXELS-1);i>=0;i--){ // change the color "left to right" sweep
        jewelpixels.setPixelColor(i, jewelpixels.Color(150,120,0,10)); // yellow color
        jewelpixels.show(); // This sends the updated pixel color to the hardware
        delay(NEOJWLEDelay); // Delay for a period of time (in milliseconds)
    }
}
else {
    for(int i=0;i&lt;NUMPIXELS;i++) { // change the color "right to left" sweep
        pixels.setPixelColor(i, pixels.Color(0,150,0,10)); // green color
        pixels.show();
        delay(NEODELAY);
    }

    for(int i=0;i&lt;NUMJWLPIXELS;i++) { // change the color "right to left" sweep
        jewelpixels.setPixelColor(i, jewelpixels.Color(0,150,0,10)); // green color
        jewelpixels.show();
    }
}
void loop() {

  // Start playing a file, then we can do stuff while waiting for it to finish
  if (! musicPlayer.startPlayingFile(SONGS[songPick])) {
    Serial.println(F("Could not open file"));
    while (1);
  }
  //Serial.println(F("Started playing"));

  while (musicPlayer.playingMusic) {
    modeSwitchState = digitalRead(MODESWITCHPIN); //read switch input
    if (modeSwitchState != lastModeButtonState) { // compare the modeSwitchState to its previous state
      if (modeSwitchState == HIGH) { // if the current state is HIGH then the button went from left to right:
        modeSwitchPushCounter++;
        songPick=0;
        musicPlayer.stopPlaying(); //does this so it'll restart the player with the new song choice

        for(int i=(NUMPIXELS-1);i>=0;i--){ //change the color "left to right" sweep
          pixels.setPixelColor(i, pixels.Color(150,120,0,10)); // yellow color
        }
        pixels.show(); // show the new values
        for(int i=(NUMVUPIXELS-1);i>=0;i--){ //change the color "left to right" sweep
          vupixels.setPixelColor(i, vupixels.Color(150,120,0,10)); // yellow color
        }
        vupixels.show();
        for(int i=(NUMJWLPixels-1);i>=0;i--){ //change the color "left to right" sweep
          jewelpixels.setPixelColor(i, jewelpixels.Color(150,120,0,10)); // yellow color
        }
        jewelpixels.show();
      }
      else { // if the current state is LOW then the button went from on to off:
        songPick=1;
        musicPlayer.stopPlaying(); // restart the player with the new song choice
        for(int i=0;i&lt;NUMPIXELS;i++){ // change the color "right to left" sweep
          pixels.setPixelColor(i, pixels.Color(0,150,0,10)); // yellow color
        }
        pixels.show(); // show the new values
        for(int i=0;i&lt;NUMVUPIXELS;i++){ // change the color "right to left" sweep
          vupixels.setPixelColor(i, vupixels.Color(0,150,0,10)); // yellow color
        }
        vupixels.show();
        for(int i=0;i&lt;NUMJWLPixels;i++){ // change the color "right to left" sweep
          jewelpixels.setPixelColor(i, jewelpixels.Color(0,150,0,10)); // yellow color
        }
        jewelpixels.show();
      }
    }

    // Delay a little bit to avoid bouncing
    delay(50);
  }
  // save the current state as the last state,
// for next time through the loop
lastModeButtonState = modeSwitchState;

if(modeSwitchState==LOW){
    // VU meter pixel bounce -- brute force, needs to clean up or become function
    vupixels.setPixelColor(0, vupixels.Color(0,150,0,10));
    vupixels.setPixelColor(1, vupixels.Color(0,150,0,10));
    vupixels.setPixelColor(2, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(3, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(4, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(2, vupixels.Color(0,150,0,10));
    vupixels.setPixelColor(3, vupixels.Color(0,150,0,10));
    vupixels.setPixelColor(4, vupixels.Color(0,150,0,10));
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(2, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(3, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(4, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(5, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(6, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(7, vupixels.Color(150,120,0,10));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
}
else{
    // VU meter pixel bounce
    vupixels.setPixelColor(0, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(1, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(2, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(3, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(4, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(2, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(3, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(4, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(5, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(6, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(7, vupixels.Color(150,120,0,10));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(2, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(3, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(4, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(2, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(3, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(4, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(2, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(3, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(4, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
}
One thing to add to the circuit at this point is an on/off switch. The PowerBoost has an enable pin (marked "EN" on the board) which cuts power to the 5V output when it is grounded. You can test this out with a single jumper wire, but we'll use a switch instead.

Note: The PowerBoost 1000C used in these photos looks a little bit different from the 500C in the circuit diagrams.

You can also add a volume knob, such as this. Unlike the 20W amp, there is not an on-board volume pot connector. Instead, we'll use a traditional 10K stereo audio potentiometer, which will act as a middle-man between the output of the MP3 shield and the input of the amplifier. You can wire it as seen in the circuit diagram.
A stereo audio potentiometer is a little different from a typical linear potentiometer. The stereo designation means that it has two gangs of conductors controlled by a single physical shaft, but separated from each other to maintain the different left and right channel integrity. The audio designation means that the resistance response of the pot is logarithmic instead of linear. Due to the biology of human hearing, a logarithmic volume change rate ends up sounding linear to our ears, instead of the sudden dropoff in sound we experience when using a linear pot switch.
You now have two of the three functions integrated. Sweet! You turn it on and off with the PowerBoost enable switch, adjust the volume, and flip modes from health to speed, causing both lights and music to change as needed.

Next up, is integration of the Audio FX board for shooting sounds.
Blaster Integration: Audio Effects

You've combined lights and music, so the next step is to add the sound effects sub-system. Here's what it looked like in part 1:

When joining separate parts into a whole system, you'll want to make sure they all have a common ground, so that's one connection we'll make between the Audio FX board and the rest.

In this case we'll also share the power input between these parts -- except for the 20W amp which will have its own power supply.

And there is one more connection to be made: the Audio FX board has an activity pin -- marked "ACT" -- which you'll use to inform the Arduino UNO or Metro when it is playing a triggered sound so that the lights will flash appropriately.

The Audio FX board's activity pin (ACT) looks like any other button to an Arduino -- just plug it into an Arduino digital pin as an INPUT_PULLUP (making sure the boards share ground) and the ACT will go "low" when a sound is playing. Your
One PowerBoost 1000C and battery will provide 5V at 1A, which is adequate to power the lights and music subsystems, as well as the Audio FX board. However, adding in the 20W amp for the shooting effects is too much and you'll hear clipping at higher volumes as the amp gets starved for power. A simple, expedient way to solve this is to supply power to the 20W amp from a separate, secondary PowerBoost and battery.

Here's a lay of the land for this circuit diagram:

- Two batteries, each connected to their own PowerBoost
- PowerBoost USB pins are connected to allow both batteries to be charged by plugging USB micro into one board
- Everything is powered on/off by the switch connected to the first PowerBoost's enable pin
- Smoothing capacitor to protect NeoPixels
- First PowerBoost 5V running on all breadboard power rails (all boards powered on this rail except the 20W amp, which is powered by second PowerBoost)
- UNO/Metro controlling all NeoPixels
- UNO/Metro (via MP3 shield) outputs stereo music through volume pot into 3.7W amplifier
- 3.7W amplifier outputs to three small speakers which will be arrayed on sides and top of prop gun
- Input toggle switch connected to UNO/Metro controlling light and music modes

Arduino code can then use that information as input to do something else, in this case to adjust the NeoPixel brightness.
• Input activity pin from Audio FX board connected to UNO/Metro controlling light flashes
• Audio FX board sounds triggered by four buttons (two to be tilt switches, not pictured in the diagram)
• Audio FX board outputs sound to 20W amplifier
• 20W amplifier powered by second PowerBoost
• 20W amplifier volume adjusted with Pot. Vol knob
• Large 20W speaker driven by 20W amplifier
• All boards/amps-switches power grounds connected to common ground

Here's the system put together on a large prototyping breadboard:

Note: The PowerBoost 1000C used in these photos looks a little bit different from the 500C in the circuit diagrams. Also, the tilt ball switches have been connected.

The only change to the software will be the addition of the Audio FX activity pin as an input to the Arduino, so it will brighten the NeoPixels when the sound effects are triggered.

Download the code below and upload it to your Arduino UNO or Metro board. This adds a few lines of code that allows the Arduino to read the state of the Audio FX board and brighten the lights when a sound is triggered.

```cpp
//Lucio Blaster code for UNO and Metro
// by John Park for Adafruit
//based upon:
/*****************************************************************************/
Adafruit VS1053 Codec Breakout
```
Designed specifically to work with the Adafruit VS1053 Codec Breakout
---&gt; https://www.adafruit.com/products/1381
Written by Limor Fried/Ladyada for Adafruit Industries.
BSD license, all text above must be included in any redistribution
******************************************************************************
#include &lt;Adafruit_VS1053.h&gt;
#include &lt;SD.h&gt;
#include &lt;Adafruit_NeoPixel.h&gt;

#define NEOPIN 2 // pin for NeoPixel ring
#define NUMPIXELS 24 //adjust depeding on ring size
Adafruit_NeoPixel pixels =
    Adafruit_NeoPixel(NUMPIXELS, NEOPIN, NEO_GRBW + NEO_KHZ800);
int NEODELAY = 0; //8 millis is 120BPM

#define NEOVUPIN 8
#define NUMVUPIXELS 8
Adafruit_NeoPixel vupixels =
    Adafruit_NeoPixel(NUMVUPIXELS, NEOVUPIN, NEO_GRBW + NEO_KHZ800);
int NEOVUDELAY = 42; //the tempo for lighting effects

#define NEOJWLPIN 10
#define NUMJWLPIXELS 5
Adafruit_NeoPixel jewelpixels =
    Adafruit_NeoPixel(NUMJWLPIXELS, NEOJWLPIN, NEO_GRBW + NEO_KHZ800);
int NEOJWLDelay = 84; //the tempo for lighting effects

// These are the pins used for the music maker shield
#define SHIELD_RESET  -1      // VS1053 reset pin (unused!)
#define SHIELD_CS     7      // VS1053 chip select pin (output)
#define SHIELD_DCS    6      // VS1053 Data/command select pin (output)

// These are common pins between breakout and shield
#define CARDCS 4     // Card chip select pin
// DREQ should be an Int pin, see http://arduino.cc/en/Reference/attachInterrupt
#define DREQ 3       // VS1053 Data request, ideally an Interrupt pin

Adafruit_VS1053_FilePlayer musicPlayer =
    Adafruit_VS1053_FilePlayer(SHIELD_RESET, SHIELD_CS, SHIELD_DCS, DREQ, CARDCS);

//state change detection
const int MODESWITCHPIN = 5;    // the pin that the pushbutton is attached to
int modeSwitchPushCounter = 0;   // counter for the number of button presses
bool modeSwitchState = 0;         // current state of the button
int lastModeButtonState = 0;     // previous state of the button

const int AFXPIN = 9; //pin connected to AudioFX board ACT pin
bool afxState = HIGH; //defaults to high, is pulled low during activity

const char* SONGS[] = {"h.mp3", "s.mp3"};//change to reflect your file names
int songPick = 0;

const int VOL = 1; //volume higher numbers are quieter

void setup() {
    pixels.begin(); // This initializes the NeoPixel library.
    pixels.setBrightness(30);
    pixels.show();

    vupixels.begin();
    vupixels.setBrightness(20);
    vupixels.show();

    jewelpixels.begin();
}
jewelpixels.setBrightness(10);
jewelpixels.show();

pinMode(MODESWITCHPIN, INPUT_PULLUP);
pinMode(AFXPIN, INPUT_PULLUP);//set up the pin connected to AudioFX ACT pin

Serial.begin(9600);

// initialise the music player
if (! musicPlayer.begin()) { // initialise the music player
    Serial.println(F("Couldn't find VS1053, do you have right pins defined?"));
    while (1);
}
if (!SD.begin(CARDCS)) {
    Serial.println(F("SD failed, or not present"));
    while (1);  // don't do anything more
} Serial.println(F("SD OK!
"));

// Set volume for left, right channels. lower numbers == louder volume!
musicPlayer.setVolume(VOL,VOL);

if (! musicPlayer.useInterrupt(VS1053_FILEPLAYER_PIN_INT))
    Serial.println(F("DREQ pin is not an interrupt pin"));

//check the switch state to start up with a NeoPixel color
modeSwitchState = digitalRead(MODESWITCHPIN); //read switch input pin
if (modeSwitchState==HIGH)
for(int i=(NUMPIXELS-1);i&gt;=0;i--){ //change the color "left to right"
    //sweep
    pixels.setPixelColor(i, pixels.Color(150,120,0,10)); //yellow color
    pixels.show(); // This sends the updated pixel color to the hardware
    delay(NEODELAY); // Delay for a period of time (in milliseconds)
}
for(int i=(NUMJWLPIXELS-1);i&gt;=0;i--){ //change the color "left to right"
    //sweep
    jewelpixels.setPixelColor(i, jewelpixels.Color(150,120,0,10)); //yellow color
    jewelpixels.show(); // This sends the updated pixel color to the hardware
    delay(NEOJWLDELAY); // Delay for a period of time (in milliseconds)
}
else {
for(int i=0;i&lt;NUMPIXELS;i++)\{ //change the color "right to left" sweep
    pixels.setPixelColor(i, pixels.Color(0,150,0,10)); // green color
    pixels.show();
    delay(NEODELAY);
}
for(int i=0;i&lt;NUMJWLPIXELS;i++)\{ //change the color "right to left" sweep
    jewelpixels.setPixelColor(i, jewelpixels.Color(0,150,0,10)); // green color
    jewelpixels.show();
    delay(NEOJWLDELAY);
}
}

void loop() {

    // Start playing a file, then we can do stuff while waiting for it to finish
if (! musicPlayer.startPlayingFile(SONGS[songPick])) {
    Serial.println(F("Could not open file"));
    while (1);
}
    //Serial.println(F("Started playing"));
while (musicPlayer.playingMusic) {
    afxState=(digitalRead(AFXPIN)); //check the Audio FX sensor

    //brighten LEDs if Audio FX board is triggered
    if(afxState==LOW){
        pixels.setBrightness(80); //brightness up
        pixels.show();
    } else{
        pixels.setBrightness(30);
        pixels.show();
    }

    modeSwitchState = digitalRead(MODESWITCHPIN); //read switch input
    if (modeSwitchState != lastModeButtonState) { // compare the modeSwitchState
        //to its previous state
        if (modeSwitchState == HIGH) {
            // if the current state is HIGH then the button
            // went from left to right:
            modeSwitchPushCounter++;
            songPick=0;
            musicPlayer.stopPlaying(); //does this so it'll restart the player with
            // the new song choice
            for(int i=(NUMPIXELS-1);i>=0;i--){ //change the color "left to right"
                //sweep
                pixels.setPixelColor(i, pixels.Color(150,120,0,10)); // yellow color
            }
            pixels.show(); //show the new values
            for(int i=(NUMVUPIXELS-1);i>=0;i--){ //change the color "left to right"
                vupixels.setPixelColor(i, vupixels.Color(150,120,0,10)); //yellow
            }
            vupixels.show();
            for(int i=(NUMJWLPIXELS-1);i>=0;i--){ //change the color "left to right"
                jewelpixels.setPixelColor(i, jewelpixels.Color(150,120,0,10)); //yellow
            }
            jewelpixels.show();
        } else {
            // if the current state is LOW then the button
            // went from on to off:
            songPick=1;
            musicPlayer.stopPlaying(); //restart the player with the new song choice
            for(int i=0;i<NUMPIXELS;i++){ //change the color "right to left" sweep
                pixels.setPixelColor(i, pixels.Color(0,150,0,10)); // yellow color
            }
            pixels.show(); //show the new values
            for(int i=0;i<NUMVUPIXELS;i++){ //change the color "right to left" sweep
                vupixels.setPixelColor(i, vupixels.Color(0,150,0,10)); // yellow color
            }
            vupixels.show();
            for(int i=0;i<NUMJWLPIXELS;i++){ //change the color "right to left" sweep
                jewelpixels.setPixelColor(i, jewelpixels.Color(0,150,0,10)); // yellow color
            }
            jewelpixels.show();
        }
        // Delay a little bit to avoid bouncing
        delay(50);
    } else {
        // save the current state as the last state,
        // for next time through the loop
        lastModeButtonState = modeSwitchState;
    }
}
if(modeSwitchState==LOW){
    //VU meter pixel bounce -- brute force, needs to clean up or become function
    vupixels.setPixelColor(0, vupixels.Color(0,150,0,10));
    vupixels.setPixelColor(1, vupixels.Color(0,150,0,10));
    vupixels.setPixelColor(2, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(3, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(4, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(2, vupixels.Color(0,150,0,10));
    vupixels.setPixelColor(3, vupixels.Color(0,150,0,10));
    vupixels.setPixelColor(4, vupixels.Color(0,150,0,10));
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(5, vupixels.Color(0,150,0,10));
    vupixels.setPixelColor(6, vupixels.Color(0,150,0,10));
    vupixels.setPixelColor(7, vupixels.Color(0,150,0,10));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
}
else{
    //VU meter pixel bounce
    vupixels.setPixelColor(0, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(1, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(2, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(3, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(4, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(2, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(3, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(4, vupixels.Color(150,120,0,10));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(5, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(6, vupixels.Color(150,120,0,10));
    vupixels.setPixelColor(7, vupixels.Color(150,120,0,10));
    vupixels.show();
    delay(NEOVUDELAY);
    vupixels.setPixelColor(5, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(6, vupixels.Color(0,0,0,0));
    vupixels.setPixelColor(7, vupixels.Color(0,0,0,0));
    vupixels.show();
}
}
Now you can try it all out! Turn on the power switch and go. You'll have lights and music running, you can adjust the volume, switch modes, and pull the triggers.

In the next part in this series we'll move from breadboards to permanent proto boards, and make the system as compact as possible in preparation for fitting it inside the 3D printed prop blaster.