BlueLive: Livestream Studio switcher controller

Created by Timothy Reese

https://learn.adafruit.com/bluelive

Last updated on 2023-08-29 02:59:24 PM EDT
# Table of Contents

## Overview
- Let's get started!

## 3D Printing

## NeoPixel Arcade Button Hack

## Wiring
- Test!
- MCP Wiring
- Bluefruit Wiring
- Metro 328 (or Arduino Uno)/NeoPixel Wiring
- Powerboost 1000C Wiring
- Momentary Pushbuttons/Arcade Buttons to MCP
- Now Connect the I/Os to the Buttons
- Bakelite 5V and Ground Rails

## Code

## Pairing to Your Machine

## Assembly
- LiPo Battery
- Labeling

## Special Thanks
Sometimes you need one button for the job, and sometimes you need many.

This project is a fully enclosed 3D-printed Bluetooth controller, powered by a METRO 328 and LiPoly battery, mapped to use with Livestream Studio.

Using the serial input on the Bluefruit EZ-Key and some MCP23017 i2c expanders, there's a whole load of buttons. Arcade buttons. Pushbuttons. Buttons and buttons and buttons!
Why Livestream Studio?

Livestream Studio is a powerful, streamlined video streaming software designed for use with PCs and Livestream's own switcher boxes. While incredibly robust, there isn't MIDI support for using just any controller you'd like. Using the hotkey portion of the software, this bypasses the need to use a regular keyboard to get at specific functions.

But what if I don't use Livestream Studio?

Map it to whatever you'd like! The code is pretty heavily documented, so feel free to modify it to your needs.
Tools You'll Need:

- Soldering iron of some sort. I use an Aoyue, but any good ones will work.
- Needle nose pliers. My favorites are on my Leatherman II multitool.
- Flush cutters. Always and forever.
- Third hand tool. The $6 one from Adafruit will do the job, the super fancy one will do the job even better.
- Optional: Blu-Tack for soldering the NeoPixel mini PCBs. Amazon.
- 3D Printer. If you don't have one, you can always source the parts out to Shapeways, or a similar service. Ever upward, never fearing.
- Silicone cover wire. I'm reiterating this here from the side, because it's not optional. Without silicone cover wire, the arcade button hack won't work. Trust me. You can use regular hookup wire for the rest of the project, but honestly, the silicone stuff is incredible. If you've never used it, and especially if you do high temperature RoHS soldering (380°C), you will love it. If you ever have to do a lot of cable wrangling, you'll love it. This project switched me over entirely.
- 2x M2 6mm machine screws.
- 8x M3 8mm (or more, doesn't really matter) flat screws.
- 1x 1000uf 6.3V (or higher) capacitor
- 1x 300-500 Ohm resistor
- 25x 150 Ohm resistors
- 3x 10k Ohm resistors
- Charting tape.
- White paint pen.
- Bluetooth 2.1 Adapter (if you don't have one.)
- A drill of some sort.
- A roughly 1/8" drillbit, or M3 to be specific.
- Scissors.

Disclaimers:

- This is a Bluetooth Raw HID and ASCII input project, not MIDI.
- The casing for this project was 3D printed, and because of its size, you'll want to make sure your bed level is tightly calibrated. It was designed to work within the parameters of a Makerbot Replicator 2, but any 3D printer that has the right build volume will absolutely work.
- This is a very large project, and quite a bit of work. None of the steps are difficult for soldering, or require wave ovens or sensitive scopes. But it is a not insubstantial amount of effort, so bear that in mind.
Let's get started!

3D Printing

When it comes to printing, I usually like to break things into pieces. This way, if a part fails on a large print, it doesn't result in a lot of lost time or material.

For the outer casing, the top, sides, and bottom have all been cut in half. You can get the .STL files below.

While printing mine, I had problems with curling on the bottom of my prints due to the size of the printing jobs, and general thickness of the parts. I had only recently gotten a 3D printer, so instead of continuing to generate lots of scrap, I made adjustments to the sides to compensate. The STLs attached to the project are unadjusted, so if your printer is dialed in, and you use a large enough raft and go slowly, you'll be fine.

Download the Files to 3D Print on Thingiverse
I decided to use a white filament for the sides near the end of the project, so that explains the switch in the next images.

Due to the curling along the bottom of the edges, I needed to get the sides to line up.

Use M3 screws to attach the bottom plates before hot-gluing in the stabilizers. You can use the extra rubber feet from the Metro 328 packaging as the rubber feet for the controller. Handy!
I printed three types of parts I call stabilizers, to keep the sides in alignment, and prevent gaps with the bottom case. Just use a healthy amount of hot glue, and hold into position. These are entirely optional, if you're able to print with minimal curling.

When gluing PLA material together for a high strength bond, don't use cyanoacrylate (...like I did.) Use E6000. It's vastly better, I just didn't have any on hand.

If you’re in a pinch and need to use cyanoacrylate, just know it'll discolor PLA prints.
I got around the discoloring by using a Sharpie to go over the whitened areas on the black PLA prints, and then wiped them down with a 1:1 mixture of 91% isopropyl alcohol and acetone.

Similarly, I printed little stands (inspired by Noe and Pedro) to hold the PCBs. These I also mounted with cyanoacrylate, but it didn't matter if there was discoloration.

By printing these mounts and affixing them later, I was able to get an idea of the internal spacing of the enclosure for fitting the PCBs, without getting lost in hypothetical designs and too much time spent modeling.
All of the PCBs click into place on their mounts, except for the Powerboost 1000C. Because it'll have to handle a USB plug being inserted and pulled for charging, I affixed it with 6mm long M2 screws.

The measurements for where go these aren't especially precise, except for the Powerboost 1000C. The Powerboost 1000C board must be oriented towards the right sidewall, and I glued it into place once I had the walls affixed by machine screws to the base.
NeoPixel Arcade Button Hack

Use silicone wire, or you'll have to cut up and resolder the chain like I did (above.)

**You Must Use Silicone Cover Wire For This Part Of The Project. Must.**

This idea by Noe and Pedro is fantastic, they explain the procedure in their tutorial below.

The V2 connector for the current NeoPixel Mini PCB pack didn't fit mine once the leads were soldered, so I made an adjustment to their design. The V3 is below, and will fit in the caps with no problems.

**Click here to download the V3 mod of the Arcade Button Connector**
Affix all of the bottom portions of the arcade buttons into the top panel, once you've removed the button and plunger.
For soldering to the NeoPixel PCBs, I used a bit of Blutack. It does get tackier when hot, so make sure you don't overheat the pads, or you'll have to use some Blutack to remove residue. But this is a huge time saver, instead of awkwardly using a third hand, or the overkill of a Panavise.

Applying a bit of solder to the pads make them much easier to solder wires to.
When feeding in the LEDs, start with the 6th LED in the bottom row, and then take your time feeding counter clockwise. It takes a bit of wrangling. Once all the LEDs are in and fed through, put in the top caps.

Wiring

The wiring for this project isn't particularly complicated, but it is repetitive.
It's important to strike a balance between using enough wire, but not too much wire. It starts to add up, and get in the way when you're trying to close the project.

Above is the schematic for the MCP23017 i2c 16 input/output port expander. It's the heart of this project, since two pins on an Arduino (or, in my case a Metro 328) into 48 pins.

Test!

Always test your circuits before soldering. Always.
MCP Wiring

We'll be using three of the MCP23017, so to start, we need to identify them amongst themselves on the i2c address bus.

- mcp0: A0 = ground, A1 = ground, A2 = ground
- mcp1: A0 = 5V, A1 = ground, A2 = ground
- mcp2: A0 = ground, A1 = 5V, A2 = ground

For the three chips, use their rails to:

- Connect pin 9 to 5V.
- Connect pin 10 to ground.
- Connect pin 18 to 5V, via a 10k Ohm resistor.
- Connect pin 12 to Analog 5 (of an Arduino Uno, i2c clock)
- Connect pin 13 to Analog 4 (of an Arduino Uno, i2c data)

Bluefruit Wiring

- Connect Ground to ground rail.
- Connect Vin to 5v rail.
- Connect RX to pin 3 (of Arduino Uno or Metro 328.)
- Connect TX (not used in code) to pin 4 (of Arduino Uno or Metro 328.)
Metro 328 (or Arduino Uno)/NeoPixel Wiring

- Connect Ground to ground rail.
- Connect 5V to 5v rail.
Connect a 1000uf 6.3V (or higher) capacitor across the + and - terminals on the Ground and 5V rails of the first protoboard, immediately next to the wires coming in from the Metro 328.

NeoPixel arrangement below:

- Connect pin 6 to free column on protoboard.
- Connect 300-500 Ohm resistor to free column.
- Connect free hole on other side of 500 Ohm resistor to Din on first NeoPixel PCB.
- Connect 5V rail to 5V on first NeoPixel PCB.
- Connect Ground rail to GND on first NeoPixel PCB.
You then connect every button to the correct GPA (or GPB) pin in sequence, with the other lead connected to ground. When the button is pressed and the GPA (or GPB) of each MCP connects to ground, that input is translated on the Arduino into a serial command.

**Powerboost 1000C Wiring**

- Connect GND to one of the outside pins on the slide switch.
- Connect EN to the inside pin on the slide switch.
- Solder the USB-B style jack into the Powerboost 1000C, and run the short USB cable to the METRO 328. (If you use an Arduino Uno with USB-A type plugs, you'll need to find one short enough to not take up unnecessary space.)

Even though you see the battery attached here, do not attach the LiPo battery until the very end of assembly. Never, ever, ever solder a circuit with the battery attached.
Momentary Pushbuttons/Arcade Buttons to MCP

- Connect all of the I/O Ground pins to each other as a daisy chain. It's a much cleaner way to wire. You can also connect these to the arcade buttons. Then connect the final one to the Ground rail.
- Connect each of the other I/O pins to their respective GPA/GPB pins on the MCP23017. See diagram above.
- Connect the LED Ground (-) to a 150 Ohm resistor, then this resistor to the Bakelite Ground rail.
- Connect the LED Positive (+) to the Bakelite 5V rail.

I repeat, do not solder the circuit with the LiPo battery attached. Plug it in at final case assembly.
Now Connect the I/Os to the Buttons
Bakelite 5V and Ground Rails

To limit the amount of wiring mess in this project, I cut a length of Bakelite to secure to the top of the case by printed 6mm standoffs.

When cutting Bakelite, use scissors, and cut through holes you won't need. It's very crumbly and unpredictable to cut, but also cuts with little force, so take your time and give yourself margin.

I then drilled two holes for the 6mm screws to secure to the standoffs. Do not glue the standoffs to the top of the case until you're finished soldering.
Solder a wire from the common Ground and 5V rails on the protoboards, and then connect each individual Ground wire and 5V wire from the pushbuttons to the Bakelite rail.

Once you've soldered each in a row, bridge the joints (this is so much fun to do!)

A note about soldering to Bakelite: it isn't easy. The pads aren't very robust, so be deliberate with your iron placement, and patient if the solder doesn't immediately leach.
Now that you've finished, glue the 6mm standoffs (already screwed to the Bakelite rail) to the top of the case, inbetween the arcade buttons and pushbuttons.
The three main sections of this code are for controlling the NeoPixels, using the MCP23017’s, and sending serial commands to the Bluefruit EZ-Key.

You can find all of the code documented below.

Click here to get the code on GitHub

To reference the tables I used for the serial commands, should you want to adapt the code for some other use, go to the wonderfully table-y page below.

Click here to get the MCP23017 Library from Adafruit

Click here to get the NeoPixel Library from Adafruit

Click here to see the BlueFruit EZ-Key Keys/Mouse via Serial page

//Written by Timothy Reese, October 11th, 2015. //This code is a mix of Adafruit examples and my own customization.

// Connect pin #12 of the expander to Analog 5 (i2c clock) // Connect pin #13 of the expander to Analog 4 (i2c data)
// Connect pins #15, 16 and 17 of the expander to ground (address selection)
// Connect pin #9 of the expander to 5V (power)
// Connect pin #10 of the expander to ground (common ground)
// Connect pin #18 through a ~10kohm resistor to 5V (reset pin, active low)

// Input #0 is on pin 21 so connect a button or switch from there to ground

#include <Wire.h>
#include "Adafruit_MCP23017.h"
#include <SoftwareSerial.h>
#include <Adafruit_NeoPixel.h>

#ifdef __AVR__

 include &lt;avr/power.h&gt;
#endif

Adafruit_MCP23017 mcp0;
Adafruit_MCP23017 mcp1;
Adafruit_MCP23017 mcp2;

#define PIN 6 // for the NeoPixel data line.
Adafruit_NeoPixel strip = Adafruit_NeoPixel(14, PIN, NEO_GRB + NEO_KHZ800);
SoftwareSerial BT(4,3); //RX, and then TX. These are not your hardware serials, don't get them confused or gibberish will ensue on boot.

void setup() {
    Serial.begin(9600);
    BT.begin(9600);
    mcp0.begin(0);      // use default address 0
    mcp1.begin(1);      // use address 1
    mcp2.begin(2);      // use address 2

    strip.begin();
    strip.show(); // Initialize all pixels to 'off'

    // This could probably be an array
    mcp0.pinMode(0, INPUT);
    mcp0.pinMode(1, INPUT);
    mcp0.pinMode(2, INPUT);
    mcp0.pinMode(3, INPUT);
    mcp0.pinMode(4, INPUT);
    mcp0.pinMode(5, INPUT);
    mcp0.pinMode(6, INPUT);
    mcp0.pinMode(7, INPUT);
    mcp0.pinMode(8, INPUT);
    mcp0.pinMode(9, INPUT);
    mcp0.pinMode(10, INPUT);
    mcp0.pinMode(11, INPUT);
    mcp0.pinMode(12, INPUT);
    mcp0.pinMode(13, INPUT);
    mcp0.pinMode(14, INPUT);
    mcp0.pinMode(15, INPUT);
}
mcp0.pinMode(15, INPUT); // turn on a 100K pullup internally
mcp0.pinMode(0, INPUT);
mcp0.pullUp(0, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(1, INPUT);
mcp0.pullUp(1, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(2, INPUT);
mcp0.pullUp(2, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(3, INPUT);
mcp0.pullUp(3, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(4, INPUT);
mcp0.pullUp(4, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(5, INPUT);
mcp0.pullUp(5, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(6, INPUT);
mcp0.pullUp(6, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(7, INPUT);
mcp0.pullUp(7, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(8, INPUT);
mcp0.pullUp(8, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(9, INPUT);
mcp0.pullUp(9, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(10, INPUT);
mcp0.pullUp(10, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(11, INPUT);
mcp0.pullUp(11, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(12, INPUT);
mcp0.pullUp(12, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(13, INPUT);
mcp0.pullUp(13, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(14, INPUT);
mcp0.pullUp(14, HIGH); // turn on a 100K pullup internally
mcp0.pinMode(15, INPUT);
mcp0.pullUp(15, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(0, INPUT);
mcp1.pullUp(0, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(1, INPUT);
mcp1.pullUp(1, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(2, INPUT);
mcp1.pullUp(2, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(3, INPUT);
mcp1.pullUp(3, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(4, INPUT);
mcp1.pullUp(4, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(5, INPUT);
mcp1.pullUp(5, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(6, INPUT);
mcp1.pullUp(6, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(7, INPUT);
mcp1.pullUp(7, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(8, INPUT);
mcp1.pullUp(8, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(9, INPUT);
mcp1.pullUp(9, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(10, INPUT);
mcp1.pullUp(10, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(11, INPUT);
mcp1.pullUp(11, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(12, INPUT);
mcp1.pullUp(12, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(13, INPUT);
mcp1.pullUp(13, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(14, INPUT);
mcp1.pullUp(14, HIGH); // turn on a 100K pullup internally
mcp1.pinMode(15, INPUT);
mcp1.pullUp(15, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(0, INPUT);
mcp2.pullUp(0, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(1, INPUT);
mcp2.pullUp(1, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(2, INPUT);
mcp2.pullUp(2, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(3, INPUT);
mcp2.pullUp(3, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(4, INPUT);
mcp2.pullUp(4, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(5, INPUT);
mcp2.pullUp(5, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(6, INPUT);
mcp2.pullUp(6, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(7, INPUT);
mcp2.pullUp(7, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(8, INPUT);
mcp2.pullUp(8, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(9, INPUT);
mcp2.pullUp(9, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(10, INPUT);
mcp2.pullUp(10, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(11, INPUT);
mcp2.pullUp(11, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(12, INPUT);
mcp2.pullUp(12, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(13, INPUT);
mcp2.pullUp(13, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(14, INPUT);
mcp2.pullUp(14, HIGH); // turn on a 100K pullup internally
mcp2.pinMode(15, INPUT);
mcp2.pullUp(15, HIGH); // turn on a 100K pullup internally

// pinMode(13, OUTPUT); // use the p13 LED as debugging
// For the Raw HID commands, here's the function.
void keyCommand(uint8_t modifiers, uint8_t keycode1, uint8_t keycode2 = 0, uint8_t keycode3 = 0,
    uint8_t keycode4 = 0, uint8_t keycode5 = 0, uint8_t keycode6 = 0) {
    BT.write(0xFD);       // our command
    BT.write(modifiers);  // modifier!
    BT.write((byte)0x00); // 0x00
    BT.write(keycode1);   // key code #1
    BT.write(keycode2); // key code #2
    BT.write(keycode3); // key code #3
    BT.write(keycode4); // key code #4
    BT.write(keycode5); // key code #5
    BT.write(keycode6); // key code #6
}

void loop() {
    // The LED will 'echo' the button - used for debugging.
    // digitalWrite(13, mcp0.digitalRead(0));

    // NeoPixel Arcade Button Colors
    strip.setPixelColor(0, 255, 165, 0);
    strip.setPixelColor(1, 255, 165, 0);
    strip.setPixelColor(2, 255, 165, 0);
    strip.setPixelColor(3, 255, 165, 0);
    strip.setPixelColor(4, 255, 165, 0);
    strip.setPixelColor(5, 255, 165, 0);
    strip.setPixelColor(6, 255, 0, 0);
    strip.setPixelColor(7, 255, 0, 0);
    strip.setPixelColor(8, 255, 0, 0);
    strip.setPixelColor(9, 255, 0, 0);
    strip.setPixelColor(10, 255, 0, 0);
    strip.setPixelColor(11, 255, 0, 0);
    strip.setPixelColor(12, 255, 0, 0);
    strip.setPixelColor(13, 255, 165, 0);
    strip.show();

    // All keyCommands are from the Raw HID Keyboard Reports section, and you just
    // starting at 4 on the list. So "B" would be a 5.
    // Likewise "I" would be a 12.
    // The BT.write commands are using the Hex values for ASCII.
    // I've included both because I used both while writing the original code, and also
    // so you can
    // pick and choose if you wish to use it for another project.

    // Preview 1
    if (mcp0.digitalRead(0) == LOW) {
        Serial.println("Success Preview 1!");
        keyCommand(0,30);
        strip.setPixelColor(6, 255, 255, 255);
        strip.show();
        delay(250);
        keyCommand(0,0);
    }

    // Preview 2
    if (mcp0.digitalRead(1) == LOW) {
        Serial.println("Success Preview 2!");
        keyCommand(0,31);
        strip.setPixelColor(7, 255, 255, 255);
        strip.show();
        delay(250);
keyCommand(0,0);

//Preview 3
if (mcp0.digitalRead(2) == LOW) {
    Serial.println("Success Preview 3!");
    keyCommand(0,32);
    strip.setPixelColor(8, 255, 255, 255);
    strip.show();
    delay(250);
    keyCommand(0,0);
}

//Preview 4
if (mcp0.digitalRead(3) == LOW) {
    Serial.println("Success Preview 4!");
    keyCommand(0,33);
    strip.setPixelColor(9, 255, 255, 255);
    strip.show();
    delay(250);
    keyCommand(0,0);
}

//Preview 5
if (mcp0.digitalRead(4) == LOW) {
    Serial.println("Success Preview 5!");
    keyCommand(0,34);
    strip.setPixelColor(10, 255, 255, 255);
    strip.show();
    delay(250);
    keyCommand(0,0);
}

//Preview 6
if (mcp0.digitalRead(5) == LOW) {
    Serial.println("Success Preview 6!");
    keyCommand(0,35);
    strip.setPixelColor(11, 255, 255, 255);
    strip.show();
    delay(250);
    keyCommand(0,0);
}

//Auto Transition (Space)
if (mcp0.digitalRead(6) == LOW) {
    Serial.println("Success Auto Transition!");
    keyCommand(0,44);
    strip.setPixelColor(12, 255, 255, 255);
    strip.show();
    delay(250);
    keyCommand(0,0);
}

//Prog 1
if (mcp0.digitalRead(7) == LOW) {
    Serial.println("Success F1!");
BT.write(0x0F);
strip.setPixelColor(5, 255, 255, 255);
strip.show();
delay(250);
}

//Prog 2
if (mcp0.digitalRead(8) == LOW) {
  Serial.println("Success F2!");
  BT.write(0x10);
  strip.setPixelColor(4, 255, 255, 255);
  strip.show();
delay(250);
}

//Prog 3
if (mcp0.digitalRead(9) == LOW) {
  Serial.println("Success F3!");
  BT.write(0x11);
  strip.setPixelColor(3, 255, 255, 255);
  strip.show();
delay(250);
}

//Prog 4
if (mcp0.digitalRead(10) == LOW) {
  Serial.println("Success F4!");
  BT.write(0x12);
  strip.setPixelColor(2, 255, 255, 255);
  strip.show();
delay(250);
}

//Prog 5
if (mcp0.digitalRead(11) == LOW) {
  Serial.println("Success F5!");
  BT.write(0x13);
  strip.setPixelColor(1, 255, 255, 255);
  strip.show();
delay(250);
}

//Prog 6
if (mcp0.digitalRead(12) == LOW) {
  Serial.println("Success F6!");
  BT.write(0x14);
  strip.setPixelColor(0, 255, 255, 255);
  strip.show();
delay(250);
}

//Cut (enter)
if (mcp0.digitalRead(13) == LOW) {
  Serial.println("Success Cut!");
  BT.write(0x0A);
  strip.setPixelColor(13, 255, 255, 255);
  strip.show();
delay(250);
}

// Record (home)
if (mcp0.digitalRead(14) == LOW) {
    Serial.println("Success Record!");
    BT.write(0x02);
    delay(250);
}

// Go Live (end)
if (mcp0.digitalRead(15) == LOW) {
    Serial.println("Success Go Live!");
    BT.write(0x05);
    delay(250);
}

// Edit Media 1 (z)
if (mcp1.digitalRead(0) == LOW) {
    Serial.println("Success Edit Media 1!");
    BT.write(0x7A);
    delay(250);
}

// Edit Graphic 1 (q)
if (mcp1.digitalRead(1) == LOW) {
    Serial.println("Success Edit Graphic 1!");
    BT.write(0x71);
    delay(250);
}

// Edit Graphic 2 (w)
if (mcp1.digitalRead(2) == LOW) {
    Serial.println("Success Edit Graphic 2!");
    BT.write(0x77);
    delay(250);
}

// Edit Graphic 3 (e)
if (mcp1.digitalRead(3) == LOW) {
    Serial.println("Success Edit Graphic 3!");
    BT.write(0x65);
    delay(250);
}

// Go to Start (g)
if (mcp1.digitalRead(4) == LOW) {
    Serial.println("Success Go to Start!");
    BT.write(0x67);
    delay(250);
}

// Play/Pause (k)
if (mcp1.digitalRead(5) == LOW) {
    Serial.println("Success Play/Pause!");
    BT.write(0x6b);
    delay(250);
}
// Go to End (')
if (mcp1.digitalRead(6) == LOW) {
    Serial.println("Success Go to End!");
    BT.write(0x27);
    delay(250);
}

// Edit Media 2 (x)
if (mcp1.digitalRead(7) == LOW) {
    Serial.println("Success Edit Media 2!");
    BT.write(0x78);
    delay(250);
}

// Edit Audio (a)
if (mcp1.digitalRead(8) == LOW) {
    Serial.println("Success Edit Audio!");
    BT.write(0x61);
    delay(250);
}

// Edit Stream (s)
if (mcp1.digitalRead(9) == LOW) {
    Serial.println("Success Edit Stream!");
    BT.write(0x73);
    delay(250);
}

// Edit Transition (d)
if (mcp1.digitalRead(10) == LOW) {
    Serial.println("Success Edit Transition!");
    BT.write(0x64);
    delay(250);
}

// Go to In (u)
if (mcp1.digitalRead(11) == LOW) {
    Serial.println("Success Go to In!");
    BT.write(0x75);
    delay(250);
}

// Set In (i)
if (mcp1.digitalRead(12) == LOW) {
    Serial.println("Success Set In!");
    BT.write(0x69);
    delay(250);
}

// Set Out (o)
if (mcp1.digitalRead(13) == LOW) {
    Serial.println("Success Set Out!");
    BT.write(0x6F);
    delay(250);
}

// Graphics 1 Preview (Numpad 1)
if (mcp1.digitalRead(14) == LOW) {
    Serial.println("Success Graphics 1 Preview!");
keyCommand(0,89);
delay(250);
keyCommand(0,0);
}

// Graphics 2 Preview (Numpad 2)
if (mcp1.digitalRead(15) == LOW) {
    Serial.println("Success Graphics 2 Preview!");
    keyCommand(0,90);
delay(250);
keyCommand(0,0);
}

// Graphics 3 Preview (Numpad 3)
if (mcp2.digitalRead(0) == LOW) {
    Serial.println("Success Graphics 3 Preview!");
    keyCommand(0,91);
delay(250);
keyCommand(0,0);
}

// Graphics 1 Push (Numpad 4)
if (mcp2.digitalRead(1) == LOW) {
    Serial.println("Success Graphics 1 Push!");
    keyCommand(0,92);
delay(250);
keyCommand(0,0);
}

// Graphics 2 Push (Numpad 5)
if (mcp2.digitalRead(2) == LOW) {
    Serial.println("Success Graphics 2 Push!");
    keyCommand(0,93);
delay(250);
keyCommand(0,0);
}

// Graphics 3 Push (Numpad 6)
if (mcp2.digitalRead(3) == LOW) {
    Serial.println("Success Graphics 3 Push!");
    keyCommand(0,94);
delay(250);
keyCommand(0,0);
}

// Graphics 1 Pull (Numpad 7)
if (mcp2.digitalRead(4) == LOW) {
    Serial.println("Success Graphics 1 Pull!");
    keyCommand(0,95);
delay(250);
keyCommand(0,0);
}

// Graphics 2 Pull (Numpad 8)
if (mcp2.digitalRead(5) == LOW) {
Serial.println("Success Graphics 2 Pull!");
keyCommand(0,96);
delay(250);
keyCommand(0,0);
}

// Graphics 3 Pull (Numpad 9)
if (mcp2.digitalRead(6) == LOW) {
    Serial.println("Success Graphics 3 Pull!");
    keyCommand(0,97);
    delay(250);
    keyCommand(0,0);
}

// This is a breakdown of the Function Commands via Hex,
// since they're not spelled out on the Bluefruit guide.
// 0x0F  = F1
// 0x10  = F2
// 0x11  = F3
// 0x12  = F4
// 0x13  = F5
// 0x14  = F6
// 0x15  = F7
// 0x16  = F8
// 0x17  = F9
// 0x18  = F10
// 0x19  = F11
// 0x1A  = F12

// https://learn.adafruit.com/introducing-bluefruit-ez-key-diy-bluetooth-hid-keyboard/sending-keys-via-serial
// http://www.instructables.com/id/different-ways-to-count/

---

Pairing to Your Machine

A note about pairing:
The current line of Livestream Studio hardware (HD31, HD51, HD510, HD550, and HD1710) all ship with Windows 7. Windows 7 doesn't natively support Bluetooth 4.0, and in my experience, I could not get it to pair with the Bluetooth 4.0 adapter available in the Adafruit shop. If you are an advanced user, you might be able to get it to work. Maybe. I really tried!

So as a result, I had to buy a Bluetooth 2.1 adapter, link here: AZIO BTD-V201

I just plugged it in, and it worked.
Assembly

LiPo Battery

I like to wrap some kapton tape around LiPo batteries, before attaching velcro. This gives me peace of mind when removing the battery, that I'm not placing direct force onto the foil wrapping the battery.

For final assembly, make sure none of the wiring is snagging between the top and bottom panels.

Insert the M3 flat machine screws into the top 4 corners.
Because of some tolerance issues with curling on mine, there was a bulge in the front. I fixed this by affixing a facade plate with cyanoacrylate glue, and then sanding down any protrusions. This single fix saved an enormous amount of time.
Labeling

For the labeling, I used thin charting tape, 1/8" wide. It's available at craft stores. I chose black, because black is all that was available.

I then used a white paint pen.

For ease of reference, I labeled all of the buttons. It makes it much easier to remember some of the hot keys on the fly.
Charting tape does not have a strong adhesive, so I dabbed a small amount of cyanoacrylate on the ends of each strip to prevent it from curling up.

Underneath the Cut arcade button, you'll see two lights visible on the frame. The blue light means that the circuit is on, and if plugged in, the amber light means that the LiPo battery is charging.

And just like that, you're done!
Special Thanks

My special thanks on this project go to an innumerable number of internet forums I had to research, and my 3D printer, that tolerated an unbelievable number of test prints.

Also, to Alex of RasPi.TV, for an excellent breakdown of the i2c pattern for the MCP23017 chips.