adafruit learning system

3d Printed Neopixel Tactile Switch Buttons

Created by Erin St Blaine

Last updated on 2018-08-22 03:54:19 PM UTC

Guide Contents

Introduction

Control your wearable project in style with low-profile, clothing-friendly illuminated push buttons.

These buttons make it easy to control LED brightness, change pattern modes, play sounds, or whatever you can dream up. They're appealing and clicky, adding a fun element of interactivity to any wearable project, and they're small and low-profile enough for even a tight top or pants button.

Check out this fully finished hoodie project (https://adafru.it/xar) using these buttons!

Materials

- Flora RGB Smart NeoPixel version 2 (http://adafru.it/1260) (one per button)
- 6mm Momentary switch (http://adafru.it/367) (one per button)
- 30awg wire in various colors (https://adafru.it/ekF)
- Microcontroller of your choice -- I'm using a Gemma (http://adafru.it/1222) for testing and a Metro Mini (http://adafru.it/2590) for my finished project
- 3d printer (optional)
- Button cover (or 3d print your own)

Tools

- Soldering iron & accessories
- E6000 or other mighty strong glue
- Needle & thread

3d Printing

There are two pieces to this button: the button cover and the backing which holds the neopixel and switch.

I've uploaded a plain version of my button cover as well as a version with an "om" symbol etched into the top. However, you don't need to use the 3d printed button -- jewels, bakelite, plastic, or even coated metal will work just fine. Play with some different ideas to discover what you like.

https://adafru.it/oAF

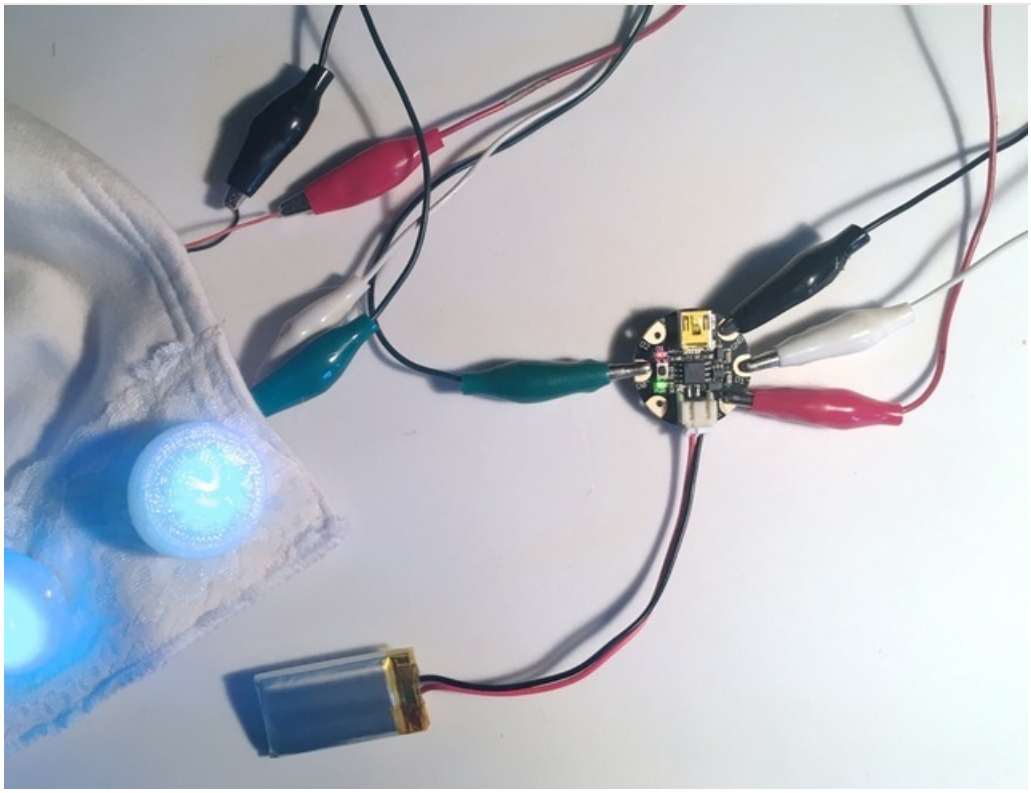
https://adafru.it/oAF

If you don't have a 3d printer you can order the buttons from Shapeways (button front (Om) (https://adafru.it/pfR) or button front (smooth) (https://adafru.it/pfS) and button back (https://adafru.it/oAH)), or you can make your own out of PVC tubing or posterboard.

The backing is just the right size to fit the electronic components inside, and the wire holes will perfectly accommodate 30awg wire. There are a few extra holes in the base for sewing the button to your project.

Print the pieces you'd like in ABS or PLA. I found that white ABS had a nice diffusion to it, but ended up usingglow in the dark PLA (https://adafru.it/mSF) for my final buttons. This stuff has a lovely translucence to it, making the buttons look prettier when the lights are off.. and of course, there's the added bonus that they glow in the dark quite vigorously after the buttons have been turned on for even a minute or two.

Testing Setup



This project involves lots of small fiddly bits, moving parts, and lots and lots of wires. In order to make success more likely you will definitely want to test your components at each step of the process, so you don't spend hours of time soldering only to find that it all mysteriously doesn't work. Testing at each step means you'll catch any shorts or mistakes early. Plus you get to see your lights come on right away which makes the whole process much more fun and satisfying.

My favorite way to test is using a Gemma microcontroller and some alligator clips. The Gemma is inexpensive and really easy to use for prototyping. You can test any combination of wires without soldering or mucking about with breadboards and headers.

If this is your first time using Arduino

Check out the getting started guide here (https://adafru.it/jDQ). You'll need to make sure you have Adafruit's boards installed as well as the Adafruit Neopixel library (https://adafru.it/nBF).

Plug your Gemma in to your computer using its USB port. Open your Arduino IDE and select Adafruit Gemma from your Boards menu.

Go to File > Examples > Adafruit_Neopixel > buttoncycler and open the buttoncycler code. Find these lines at the top:

```
#define BUTTON_PIN   2    // Digital IO pin connected to the button.  This will be
                          // driven with a pull-up resistor so the switch should
                          // pull the pin to ground momentarily.  On a high -> low
                          // transition the button press logic will execute.

#define PIXEL_PIN    6    // Digital IO pin connected to the NeoPixels.

#define PIXEL_COUNT 16
```

Change BUTTON_PIN to 0 and PIXEL_PIN to 1. Change the PIXEL_COUNT to reflect the number of buttons you are planning to make.

Press the Reset button on the Gemma to get it into bootloader mode, and then immediately press the upload button in Arduino to upload the code.

Then, get your alligator clips out and hook them up thusly:

- Red > Gemma Vout
- Black > Gemma G
- White > Gemma D1
- Green > Gemma D0

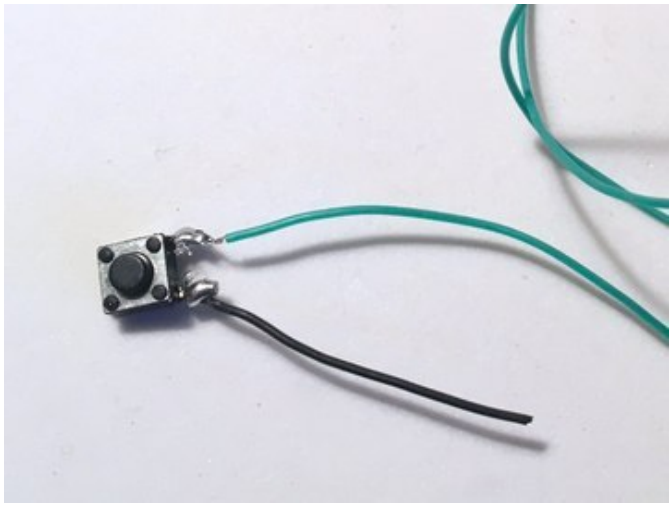
The clips' colors will correspond to the wire colors used in the rest of this project. You can power the Gemma from the USB port or plug a battery in to the JST connector.

Wiring it Up

One leg of the tactile switch will wire to the '-' pin on its corresponding neopixel. The second leg of each button will get wired to a data pin on your microcontroller.

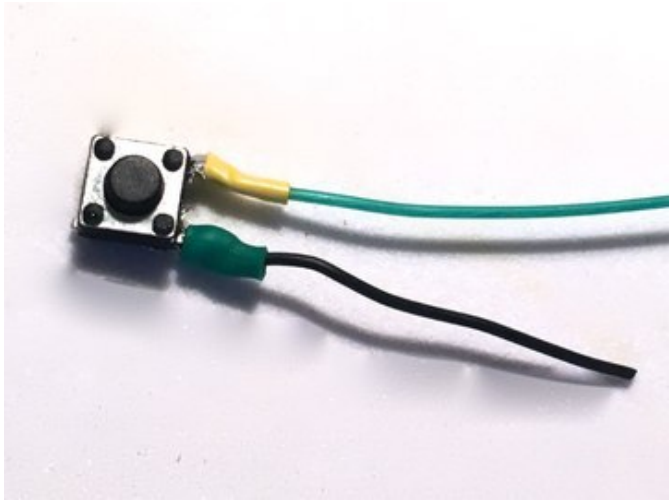
The NeoPixels will be wired in series, connected to + and - and another data pin on your microcontroller.

Switch Wiring

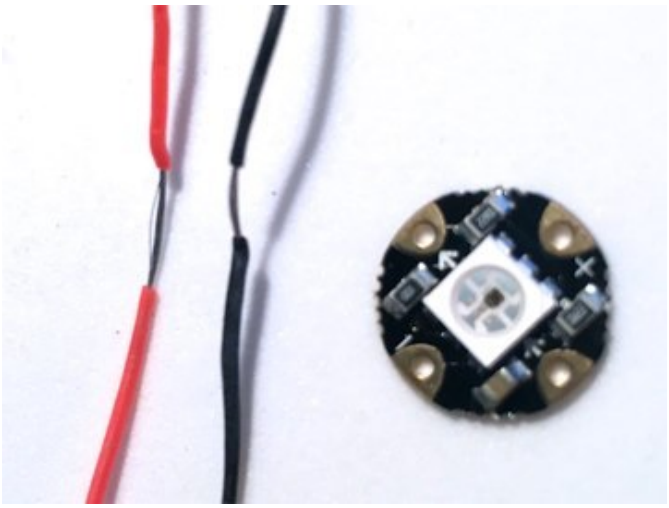


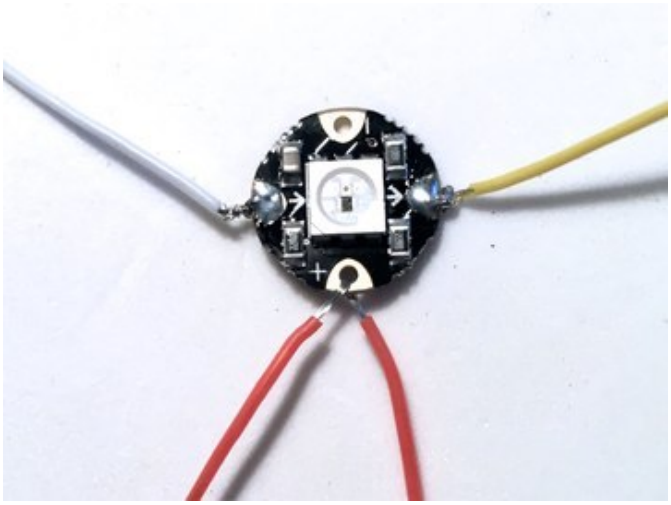
Get out your push button switch and flatten out all the legs. Cut off the two adjacent legs on one side. Trim the remaining two legs to about half their length and solder a short black wire and a long green wire onto these legs.

Cover each leg with a miniscule amount of heat shrink -- as little as you can get away with since we need this to fit into a very small space.



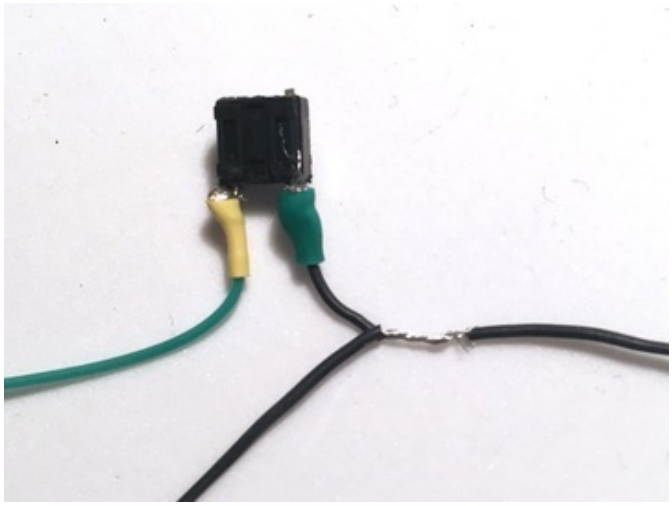
Neopixel Wiring

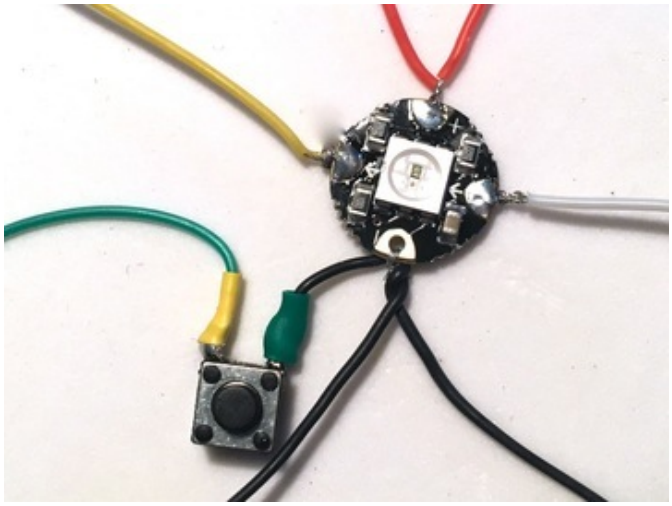




Cut a medium-length red and black wire and use your fingernail or a lighter to break and spread apart the sheilding in the middle of the wire. Thread the red wire through the + hole on the neopixel, twist it around and solder.

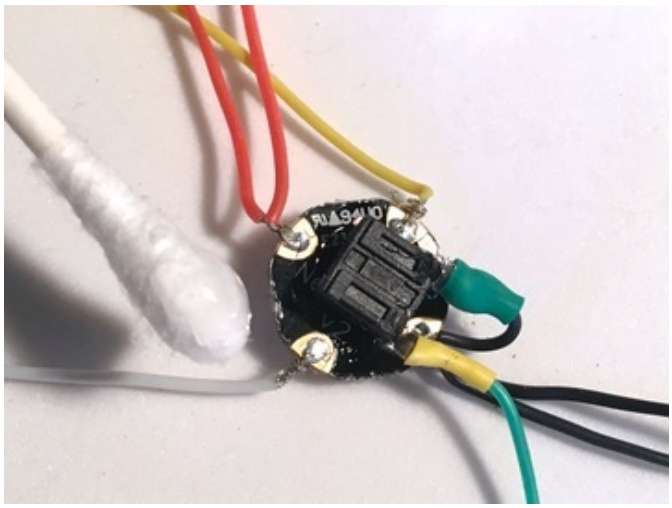
Solder a white wire to the "in" hole and a yellow wire to the "out" hole of each neopixel.

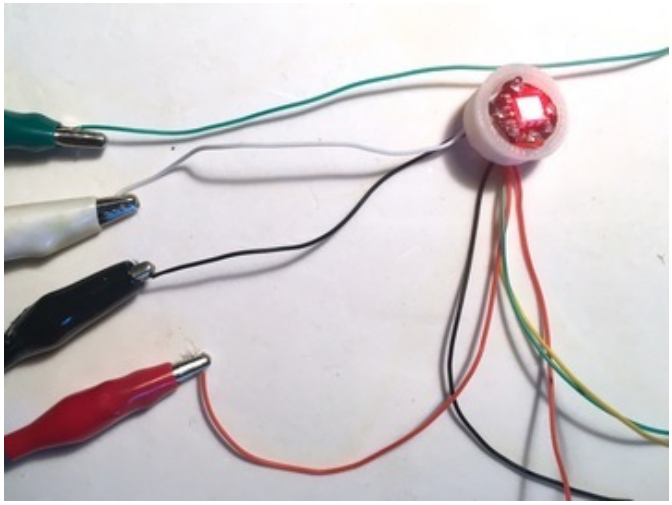




Trim the black wire coming from your button to about 1/2" and strip the end. Twist it neatly around the bare spot in the middle of your black wire.

Thread the black wires through the - hole on the Neopixel, twist around and solder.





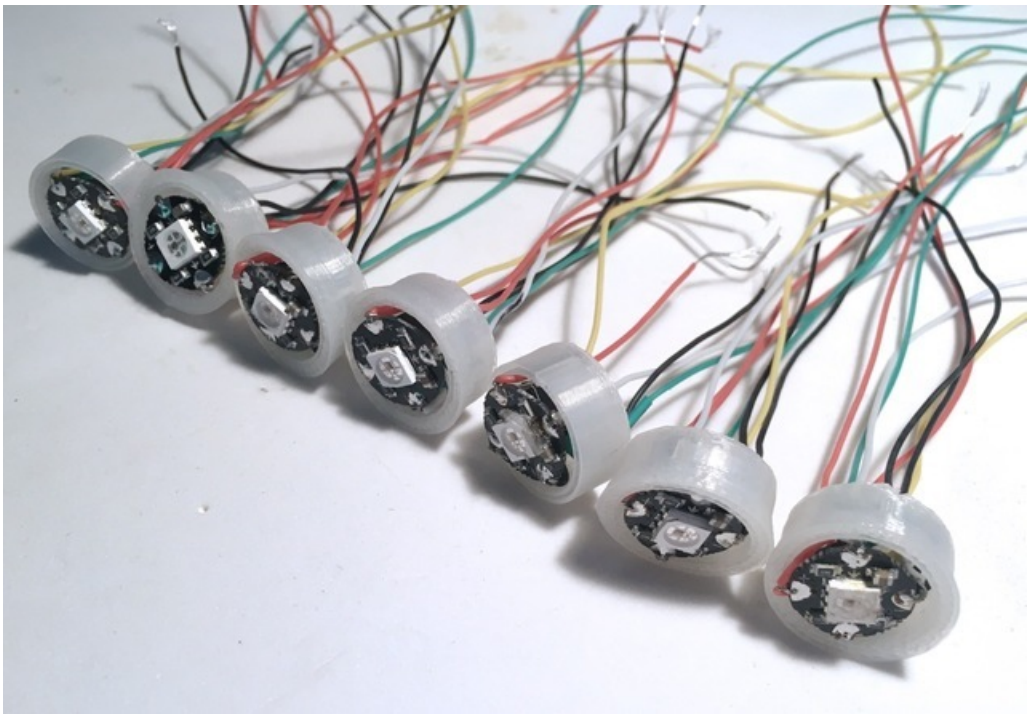
Carefully bend the button around so the clicky part is pressed against the back of the neopixel. Glue it in place with E6000 or epoxy, being careful not to get the glue into the button's workings.

Thread the wires through the holes in your 3d printed cup, leaving every other hole open (you can fit 2 wires through each hole as needed).

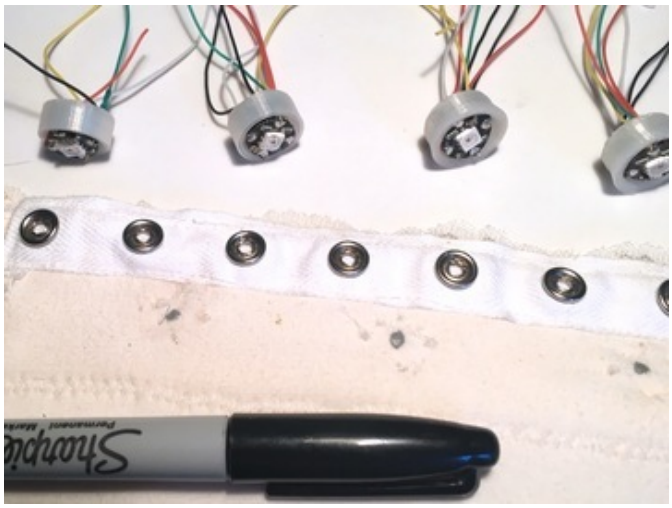
Clip the button's wires to your Gemma running the buttoncycle test code. Your button should turn on with the first press and change colors with each subsequent press.

If everything is working, add a dab of glue inside the cup to hold the button and neopixel assembly down tight.

Repeat until all your buttons are wired, glued, and tested.

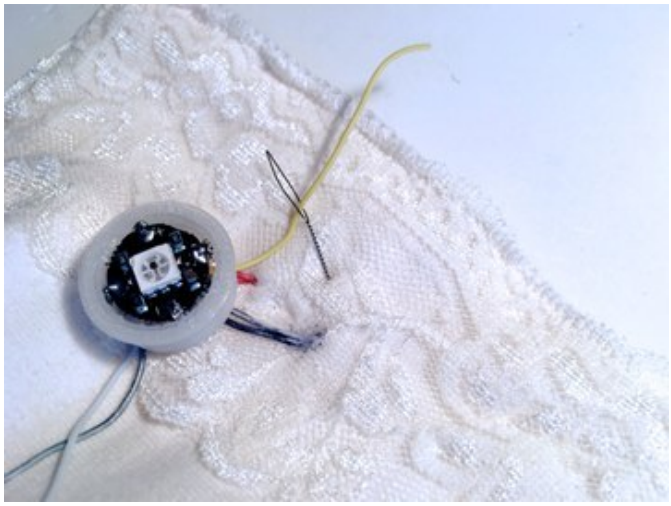


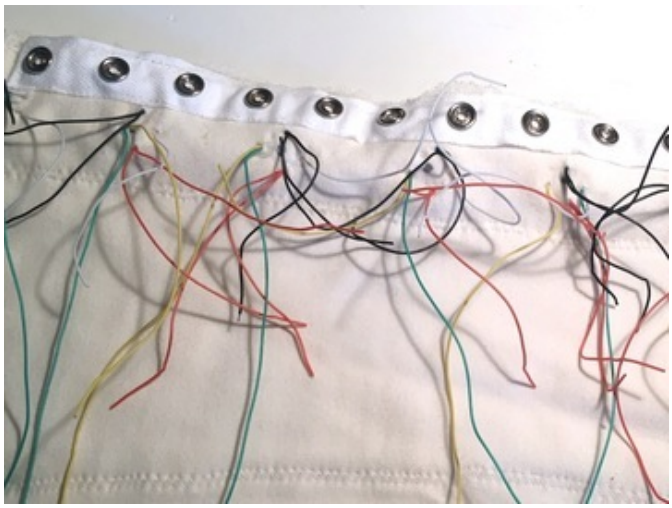
Install Buttons

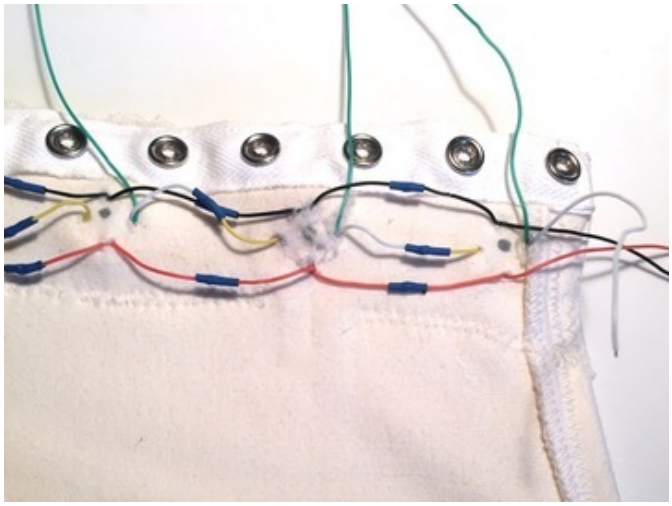


Mark the inside of your garment where you want the buttons to go. Use a thread ripper or an awl to make 4 holes around your mark, at 12:00, 3:00, 6:00 and 9:00.

Use a needle threader or a beading needle to pull the wires through the holes. It helps to keep all the buttons oriented the same way (i.e. the red wire is always on the left).

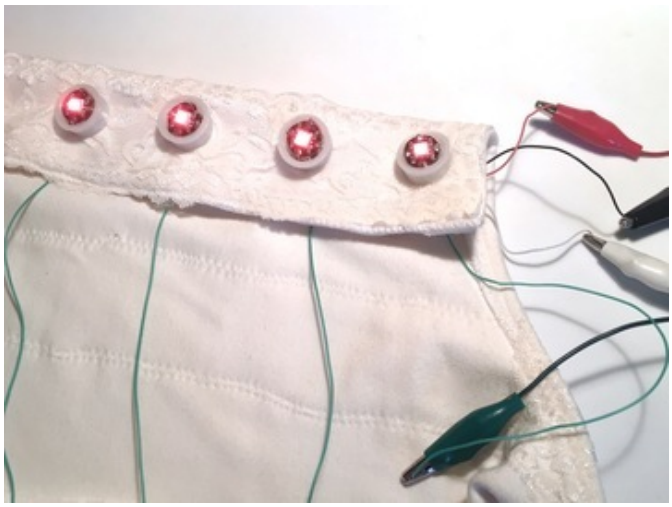




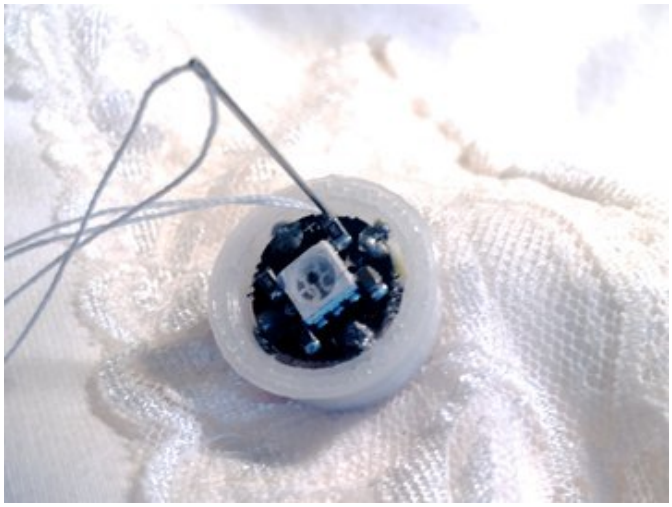


Flip your garment over and pat yourself on the back for being consistent with your wire color coding.

Pull the green button wires out of the way for now. Then starting at the "in" end, hook up all the power and ground wires, then hook all the "out" yellow wires to the "in" white wires of the next button in line.

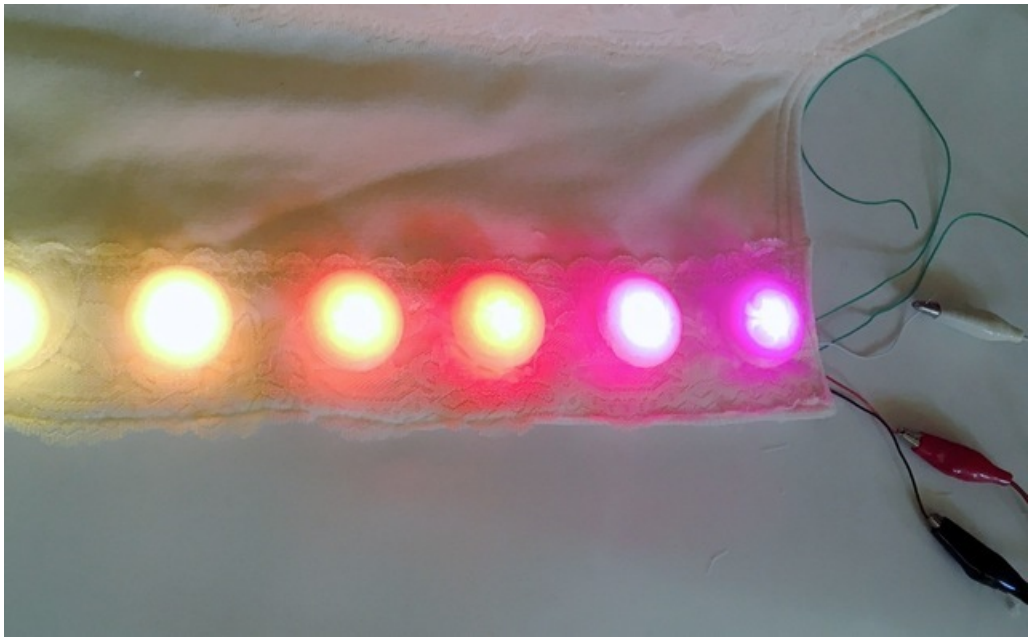


Hook up the power, ground, data-in and button wires coming from your first button to your tester Gemma once again. Be sure the whole strand comes on and reacts when you press the first button. Test the other buttons by hooking each green wire up to your gemma one at a time.



Once you're sure everything works, sew the buttons tightly to your garment through the remaining holes in the 3d printed cups. Be careful not to get thread tangled in the button workings!

Glue the caps onto the buttons using E6000 or epoxy.



Play with your clicky buttons and watch them twinkle!

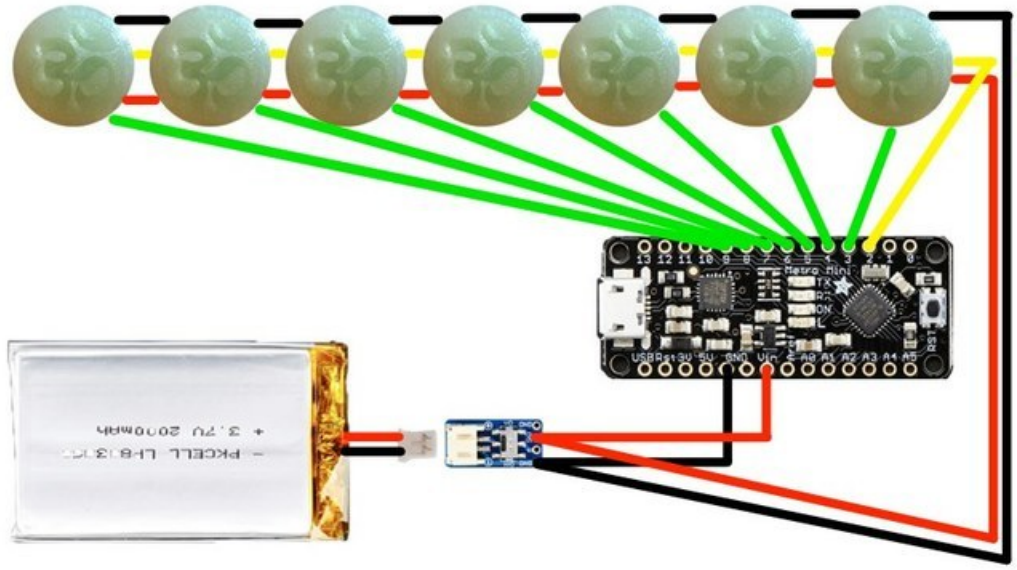
You'll notice that the buttons respond very nicely to the first few presses -- the non-animated patterns. As soon as the neopixel patterns become animated the buttons don't respond as well. Neopixels don't like to be interrupted when they've been given a task, and the Adafruit Neopixel library's animated modes don't play well with button presses. We'll fix this by using the FastLED library and fancier microcontrollers in the next section.

Make them Do Stuff

How do you want to use these buttons in your project? With a bank of blinky buttons, the sky is the limit.

- Control the colors or properties of the buttons themselves
- Control other LED arrays in the same project
- Trigger sound events
- Send someone an SMS message with Fona
- Trigger a web based event with a Feather Wifi

Here is one example of how to wire 7 buttons to an Adafruit Metro Mini (https://adafru.it/fto).



In this example, the data line is wired to pin 2 on the Metro Mini, and the buttons are wired to pins 3-9. Pins 0 and 1 are left available for serial connections or debugging.

Sample Code

Here is some sample code that shows one simple way to address all 7 buttons. In this code, the buttons are running a color palette-based gradient animation, and the animation's color changes whenever you press a button.

Install the FastLED Library

FastLED is a very powerful tool for creating all kinds of LED lighting patterns and effects. Learn more and find lots of yummy code samples at the FastLED Google Plus (https://adafru.it/ebn) community page.

Use the Library Manager in the Arduino IDE to install the FastLED library (Sketch→Include Library→Manage Libraries...).

Once you've got the library installed, select your Arduino board from the Tools→Board menu and upload the code.

```
#include <FastLED.h>
// Because conditional #includes don't work w/Arduino sketches...
```

```
//#include <SPI.h>         // COMMENT OUT THIS LINE FOR GEMMA OR TRINKET
#include <avr/power.h> // ENABLE THIS LINE FOR GEMMA OR TRINKET

#define DATA_PIN      2
#define COLOR_ORDER GRB
#define NUM_LEDS      7
#define DEBOUNCE     10 // button debouncer, how many ms to debounce, 5+ ms is usually plenty

int HUE=15;
int SATURATION=255;
int BRIGHTNESS=150;
int SPEEDO=10;
int STEPS=25;

CRGB leds[NUM_LEDS];
TBlendType    currentBlending;
CRGBPalette16 currentPalette;

// here is where we define the buttons that we'll use. button "1" is the first, button "6" is the 6th, et
byte buttons[] = {3,4,5,6,7,8,9};
// This handy macro lets us determine how big the array up above is, by checking the size
#define NUMBUTTONS sizeof(buttons)
// we will track if a button is just pressed, just released, or 'currently pressed'
byte pressed[NUMBUTTONS], justpressed[NUMBUTTONS], justreleased[NUMBUTTONS];

void setup() {
  byte i;

  // Make input & enable pull-up resistors on switch pins
  for (i=0; i<NUMBUTTONS; i++){
    pinMode(buttons[i], INPUT_PULLUP);
  }

  // pin13 LED
  pinMode(13, OUTPUT);

  //FastLED.setMaxPowerInVoltsAndMilliamps( VOLTS, MAX_MA);
  FastLED.addLeds<WS2812B,DATA_PIN,COLOR_ORDER>(leds, NUM_LEDS).setCorrection(TypicalLEDStrip);
}




void loop() {

Gradient();
digitalWrite(13, LOW);

check_switches();      // when we check the switches we'll get the current state

  for (byte i = 0; i<NUMBUTTONS; i++){
    if (pressed[i]) {
      digitalWrite(13, HIGH);
      // is the button pressed down at this moment
    }
    if (justreleased[i]) {
      if (i == 0){
        HUE=190;
      }else if (i == 1){
```

```
        HUE=170;
      }else if (i == 2){
        HUE=140;
      }else if (i == 3){
        HUE=100;
      }else if (i == 4){
        HUE=70;
      }else if (i == 5){
        HUE=30;
      }else if (i == 6){
        HUE=0;
  }
  for (byte i=0; i<NUMBUTTONS; i++){  // remember, check_switches() will necessitate clearing the 'just p
    justpressed[i] = 0;
  }
}
  }
}

void check_switches()
{
  static byte previousstate[NUMBUTTONS];
  static byte currentstate[NUMBUTTONS];
  static long lasttime;
  byte index;

  if (millis() < lasttime){ // we wrapped around, lets just try again
     lasttime = millis();
  }

  if ((lasttime + DEBOUNCE) > millis()) {
    // not enough time has passed to debounce
    return;
  }
  // ok we have waited DEBOUNCE milliseconds, lets reset the timer
  lasttime = millis();

  for (index = 0; index<NUMBUTTONS; index++){ // when we start, we clear out the "just" indicators
    justreleased[index] = 0;

    currentstate[index] = digitalRead(buttons[index]);   // read the button

    if (currentstate[index] == previousstate[index]) {
      if ((pressed[index] == LOW) && (currentstate[index] == LOW)) {
          // just pressed
          justpressed[index] = 1;
      }
      else if ((pressed[index] == HIGH) && (currentstate[index] == HIGH)) {
          // just released
          justreleased[index] = 1;
      }
      pressed[index] = !currentstate[index];  // remember, digital HIGH means NOT pressed
    }
    //Serial.println(pressed[index], DEC);
    previousstate[index] = currentstate[index];   // keep a running tally of the buttons
  }
}


// GRADIENT ------------------------------------------------------------
```

```
void Gradient()
{
  SetupGradientPalette();

  static uint8_t startIndex = 0;
  startIndex = startIndex + 2;  // motion SPEEDO

  FillLEDsFromPaletteColors( startIndex);
  FastLED.show();
  FastLED.delay(SPEEDO);
}

void SetupGradientPalette()
{
  CRGB light = CHSV( HUE + 5, SATURATION - 15, BRIGHTNESS);
  CRGB light1 = CHSV( HUE + 10, SATURATION - 10, BRIGHTNESS);
  CRGB light2 = CHSV( HUE + 15, SATURATION - 20, BRIGHTNESS);
  CRGB medium = CHSV ( HUE - 3, SATURATION, BRIGHTNESS);
  CRGB medium1 = CHSV ( HUE - 7, SATURATION, BRIGHTNESS);
  CRGB medium2 = CHSV ( HUE - 11, SATURATION, BRIGHTNESS);
  CRGB dark  = CHSV( HUE + 3, SATURATION - 30, BRIGHTNESS);
  CRGB dark1  = CHSV( HUE, SATURATION - 20, BRIGHTNESS);
  CRGB dark2  = CHSV( HUE -3, SATURATION - 15, BRIGHTNESS);



  currentPalette = CRGBPalette16(
    light,  light1,  light2,  light1,
    medium, medium1, medium2,  medium1,
    dark,  dark1,  dark2,  dark1,
    medium, medium1, medium2,  medium1 );
}

void FillLEDsFromPaletteColors( uint8_t colorIndex)
{
  uint8_t brightness = BRIGHTNESS;

  for( int i = 0; i < NUM_LEDS; i++) {
    leds[i] = ColorFromPalette( currentPalette, colorIndex, brightness, currentBlending);
    colorIndex += STEPS;
  }
}
```

Having a button array is a very powerful, effective and interactive way to control your project. What will you control?