# Circuit Playground PZ-1: Pizza Box DJ Controller

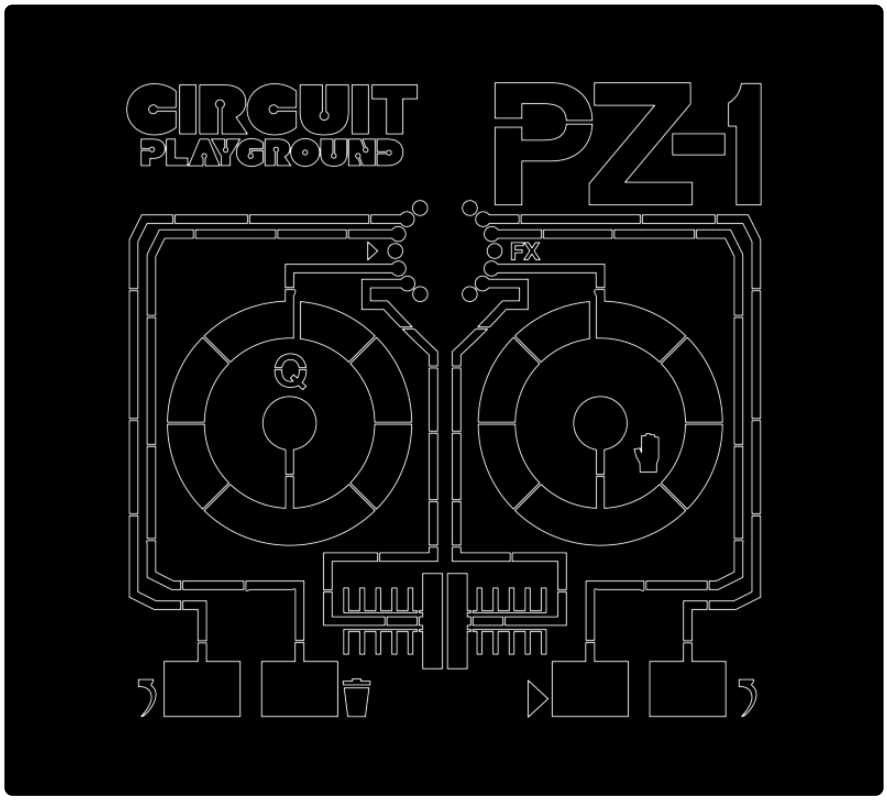
Created by John Park


https://learn.adafruit.com/circuit-playground-pizza-box-dj-controller

Last updated on 2024-06-03 02:00:24 PM EDT

# Table of Contents

# Overview

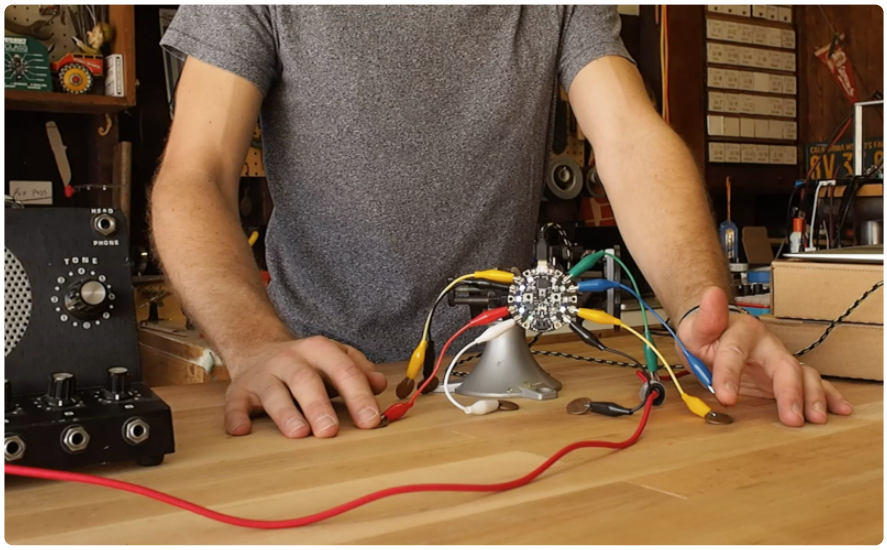



# DJ Controller

DJ software is fun to use -- who doesn't like to get the crowd bumping? -- but controlling it from a laptop keyboard isn't. So, DJ software is often controlled with a peripheral device that has buttons, knobs, sliders, and turntables for a more natural feel. But you don't need to invest in one, you can build your own: the **Circuit Playground PZ-1**!

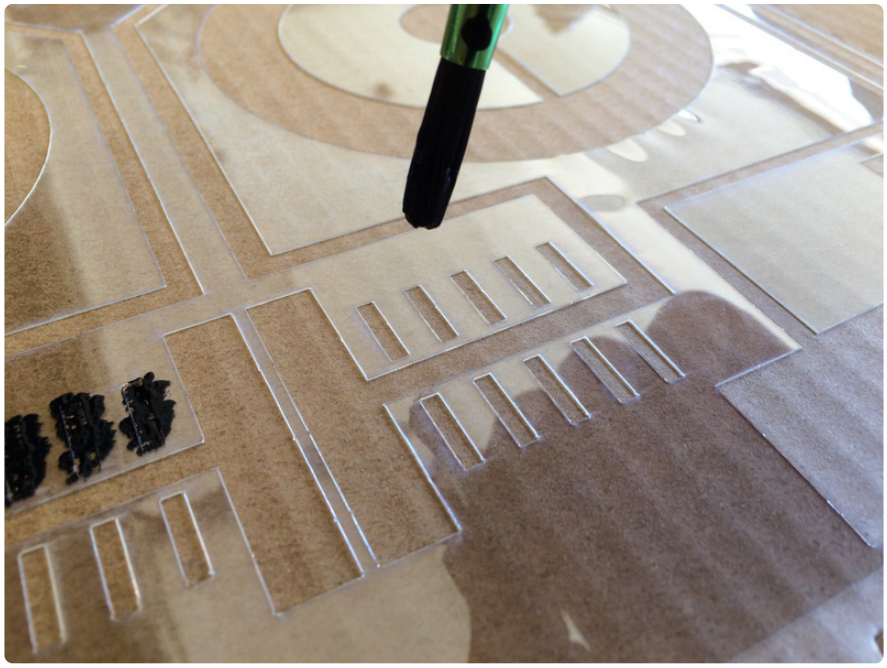
# Make Your Own

You can build your own DJ controller using any microcontroller that speaks MIDI, including the **Circuit Playground**. All of the buttons and sensors on the **Circuit Playground** can be used to trigger MIDI signals that will tell your DJ software what to do. In this project uses the two buttons and eight capacacitive pads, but you could easily modify it to also utilize the accelerometer, temperature sensor, light sensor, and microphone if you like!



# Paint Your Interface

The **Circuit Playground** sending MIDI commands is the brains of the operation, but you'll also want an intuitive interface. So design your own using conductive paint! You can paint any pattern you like, running a different conductive paint circuit trace from each capacitive pad on the **Circuit Playground**. If you want to create precise designs, mask with tape, or create a stencil.



# Parts

- Circuit Playground (http://adafru.it/3000)
- Conductive Paint (http://adafru.it/1305) or Conductive Paint Pen (http://adafru.it/1306)
- USB cable A/microB (http://adafru.it/2008)

# Materials and Tools

- Cardboard 12" pizza box (prefereably new, non-greasy)
- Mylar stencil blank sheets, 12"x12", 7 mil thick
- Painter's tape
- Spray adhesive (optional)
- Utility knife
- Stencil brush (any size from 2 - 6)

# Software

You can use your Circuit Playground PZ-1 with nearly any DJ software. Here are some popular choices:

- Mixxx (OSX, Windows, & Linux)  [Note: Free and open source]
- Traktor (OSX, & Windows)
- Serato DJ (OSX, & Windows)

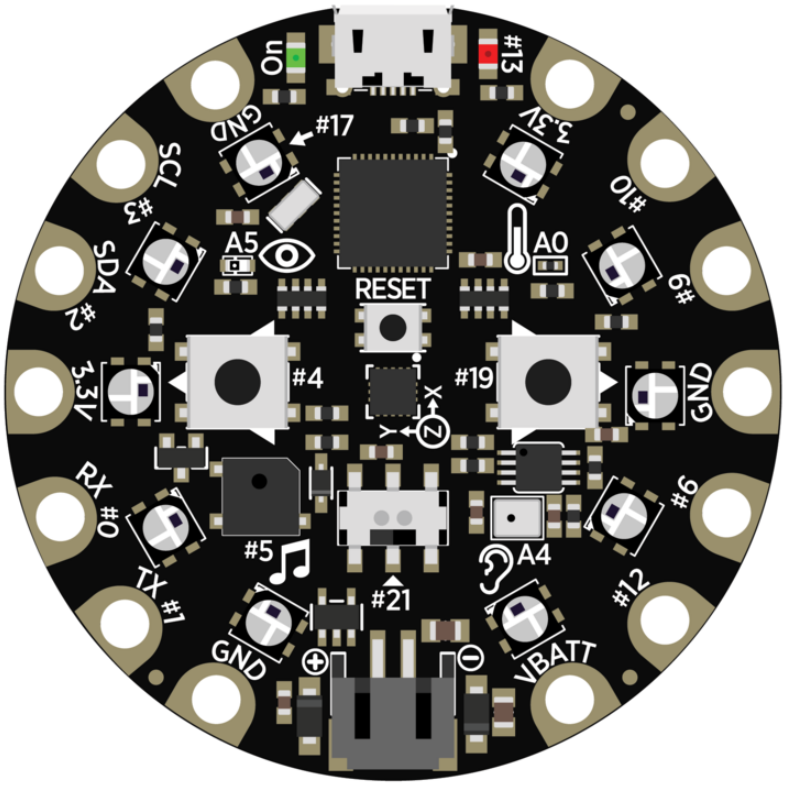
DJ software will usually include an interface for mapping MIDI commands. It can also be helpful to diagnose your MIDI stream with a utility such as:

- MIDI Monitor (https://adafru.it/rBH) for OSX
- MIDI-OX (https://adafru.it/nkF) for Windows
- KMIDIMon (https://adafru.it/rBI) for Linux

---

# Prep the Circuit Playground

Before you get started on making the PZ-1, make sure you've been through the basic tutorials on using the Circuit Playground. You'll need to have installed the Arduino IDE, added the Circuit Playground board to Arduino, and added the Circuit Playground library. This excellent guide, Introducing Circuit Playground (https://adafru.it/pAP) and Circuit Playground lesson #0 (https://adafru.it/r0D) will show you how to do all of that and get you going.




Plug your Circuit Playground into your computer now, and launch the Arduino IDE. Double check that you've selected the board and port as in the Lesson #0 tutorial and are ready to upload code.

# USB MIDI with Teensy Core

There are a few different ways to have the Circuit Playground speak MIDI over USB to your computer. For this project you'll install the Circuit Playground Teensy Core board environment called **TeeOnArdu** into your **Arduino** IDE.

Add the Teensy Core by clicking **Tools** > **Board** > **Board Manager...** in the **Arduino** IDE, and then clicking in the search field (it says Filter your search...) and typing **Adafruit TeeOnArdu**. Click on the board entry and then click the **Install** button.

# How to Use It

- In the **Arduino** IDE, select the new entry under **Tools** > **Board** > **Circuit Playground (TeensyCore)**
- Then, select **Tools** > **USB Type** > **MIDI**
- To upload new sketches you just have to double-press the reset button on the **Circuit Playground** each time you hit the upload button

- If you decide to use your **Circuit Playground** as a regular **Circuit Playground** again, just select it under T**ools** > **Board** >**Circuit Playground** and upload an **Arduino** sketch like **Blink File** > **Examples** > **Basics** > **Digital** > **Blink**. The first time you have to double-press the reset button, but after that your **Circuit Playground** behaves like out of the box.

---

# PZ-1 Pizza Box DJ Code

# MIDI Over USB

Adding the Teensy code to your Arduino environment allows you to use the excellent MIDI over USB commands in your sketches.

You can transmit MIDI messages from your **Circuit Playground** using a few different commands. (Here's more detailed info (https://adafru.it/rBJ) from the PJRC Teensy page.) For our needs, the most commons ones are:

```
usbMIDI.sendNoteOn(note, velocity, channel)
usbMIDI.sendNoteOff(note, velocity, channel)
usbMIDI.sendControlChange(control, value, channel)
```

You can test out a basic sketch to see if things are all working.

```
#include &lt;Adafruit_CircuitPlayground.h&gt;

void setup() {
  CircuitPlayground.begin();
}

void loop() {
  /************* TEST BOTH BUTTONS */
  if (CircuitPlayground.leftButton()) {
    CircuitPlayground.redLED(HIGH);
    usbMIDI.sendNoteOn(60, 127, 0);  // 60 = C4 velocity 127 (max)
    delay(100);
    usbMIDI.sendNoteOff(60, 0, 0);
    CircuitPlayground.redLED(LOW);
  }
  if (CircuitPlayground.rightButton()) {
    CircuitPlayground.redLED(HIGH);
    usbMIDI.sendNoteOn(61, 127, 0);
    delay(100);
    usbMIDI.sendNoteOff(61, 0, 0);
    CircuitPlayground.redLED(LOW);
  }

  // dont listen to any incoming MIDI!
  while (usbMIDI.read()) {
```

```
  }
}
```

Copy that code into your **Arduino** IDE and then upload it to the Circuit Playground (TeensyCore). Make sure you are in MIDI mode (**Tools** > **USB type** > **MIDI**) and have the correct port selected for your **Circuit Playground**.

Once uploaded, launch your MIDI monitor software and a music application such as Garage Band. When you press the left and right buttons on the **Circuit Playground** you should hear see the C4 note indicated in the data stream and hear a C.

---

# DJ Software MIDI Commands

# Control Change

Instead of playing musical notes, you'll be sending more generic MIDI commands from the **Circuit Playground** and then configuring your DJ software to use them. The MIDI Control Change command can be used to send numbered controls and their values.

For example, `usbMIDI.sendControlChange(12, 127, 1)` will send a MIDI command number 12 with a value of 127 over channel 1. You can configure your DJ software to listen for command number 12 on channel 1 and when the value sent is 127, it can raise the volume fader on deck B to full volume. You can stream a series of values from 0-to-127 over that command number 12 and depending on the capacative pad reading on your Circuit Playground that is triggering that MIDI command, you and raise and lower the volume.

# PZ-1 Code

Here is the **Circuit Playground PZ-1 Pizza Box DJ** code. Copy it and paste it into a new **Arduino** sketch.

```
//Circuit Playground PZ-1 Pizza Box DJ
// by John Park
// a.k.a. DJ Sternocleidomastoid
// for Adafruit Industries
// MIT License
////////////////////////////////////////////////////////////////////////////
// MIDI controller for Traktor, Mixxx and other DJ software
// Uses Circuit Playground (Teensy Core) to send MIDI over
// USB
//
// MIDI controller command mapping must be set up in your DJ software, check
// below for MIDI signal outputs
////////////////////////////////////////////////////////////////////////////
```

```
#include &lt;Adafruit_CircuitPlayground.h&gt;
#include &lt;Wire.h&gt;
#include &lt;SPI.h&gt;

////////////////////////////////////////////////////////////////////////
//MIDI CC (change control) assignments
const int CHANNEL         = 14; //MIDI channel
const int LEFT_BUTTON_CC  =  4; //Play/Pause Deck A
const int RIGHT_BUTTON_CC = 19; //MIDI CC for left button: FX 1 On
const int CAP1_CONTROL    = 1;  //Select/Set+Store Hotcue 2 (1 is load point)
const int CAP0_CONTROL    = 0;  //Left fader mixes Deck B volume down
const int CAP2_CONTROL    = 2;  //Delete hotcue 2 point
const int CAP3_CONTROL    = 3;  //Remix Trigger 6-1 or Loop Size /2
const int CAP12_CONTROL   = 12; //Right fader mixes Deck A volume down
const int CAP6_CONTROL    = 6;  //Scratch (jog turn) Deck B
const int CAP9_CONTROL    = 9;  //Play/Pause Deck B
const int CAP10_CONTROL   = 10; //Remix Trigger 6-4 or Loop Set 4 bar

int leftButtonState = 0;
int rightButtonState = 0;
int leftButtonLastState = 0;
int rightButtonLastState = 0;
int cap0State = 0;
int cap0LastState = 0;
int cap1State = 0;
int cap1LastState = 0;
int cap2State = 0;
int cap2LastState = 0;
int cap3State = 0;
int cap3LastState = 0;
int cap12State = 0;
int cap12LastState = 0;
int cap6State = 0;
int cap6LastState = 0;
int cap9State = 0;
int cap9LastState = 0;
int cap10State = 0;
int cap10LastState = 0;
bool cap1ON = false; //prevents off command from being spewed
bool cap12ON = false; //prevents off command from being spewed
bool cap6ON = false; //prevents off command from being spewed

const int CAPMIN = 25; //lowest value that's considered an intentional touch to
minimize crosstalk
const int CAPSLOP = 0; //value of capacitive pad variance that's considered noise


////////////////////////////////////////////////////////////////////////
//mic_meter code
// To keep the display 'lively,' an upper and lower range of volume
// levels are dynamically adjusted based on recent audio history, and
// the graph is fit into this range.
#define  FRAMES 8
uint16_t lvl[FRAMES],           // Audio level for the prior #FRAMES frames
         avgLo  = 6,            // Audio volume lower end of range
         avgHi  = 512,          // Audio volume upper end of range
         sum    = 256 * FRAMES; // Sum of lvl[] array
uint8_t  lvlIdx = 0;            // Counter into lvl[] array
int16_t  peak   = 0;            // Falling dot shows recent max
int8_t   peakV  = 0;            // Velocity of peak dot

////////////////////////////////////////////////////////////////////////
void setup() {
  CircuitPlayground.begin();
  CircuitPlayground.setBrightness(30); //make brighter for performance, up to 255
  CircuitPlayground.clearPixels();
  for(uint8_t i=0; i&lt;FRAMES; i++) lvl[i] = 256;

}
```

```
////////////////////////////////////////////////////////////////////////
// COLOR TABLES for animation ------------------------------------------

const uint8_t PROGMEM
  reds[]   = { 0x9A, 0x75, 0x00, 0x00, 0x00, 0x65, 0x84, 0x9A, 0xAD, 0xAD },
  greens[] = { 0x00, 0x00, 0x00, 0x87, 0xB1, 0x9E, 0x87, 0x66, 0x00, 0x00 },
  blues[]  = { 0x95, 0xD5, 0xFF, 0xC3, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
  gamma8[] = { // Gamma correction improves the appearance of midrange colors
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
    0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x03, 0x03, 0x03, 0x03,
    0x03, 0x03, 0x04, 0x04, 0x04, 0x04, 0x05, 0x05, 0x05, 0x05, 0x05, 0x06,
    0x06, 0x06, 0x06, 0x07, 0x07, 0x07, 0x08, 0x08, 0x08, 0x09, 0x09, 0x09,
    0x0A, 0x0A, 0x0A, 0x0B, 0x0B, 0x0B, 0x0C, 0x0C, 0x0D, 0x0D, 0x0D, 0x0E,
    0x0E, 0x0F, 0x0F, 0x10, 0x10, 0x11, 0x11, 0x12, 0x12, 0x13, 0x13, 0x14,
    0x14, 0x15, 0x15, 0x16, 0x16, 0x17, 0x18, 0x18, 0x19, 0x19, 0x1A, 0x1B,
    0x1B, 0x1C, 0x1D, 0x1D, 0x1E, 0x1F, 0x1F, 0x20, 0x21, 0x22, 0x22, 0x23,
    0x24, 0x25, 0x26, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2A, 0x2B, 0x2C, 0x2D,
    0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
    0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x40, 0x41, 0x42, 0x44, 0x45, 0x46,
    0x47, 0x48, 0x49, 0x4B, 0x4C, 0x4D, 0x4E, 0x50, 0x51, 0x52, 0x54, 0x55,
    0x56, 0x58, 0x59, 0x5A, 0x5C, 0x5D, 0x5E, 0x60, 0x61, 0x63, 0x64, 0x66,
    0x67, 0x69, 0x6A, 0x6C, 0x6D, 0x6F, 0x70, 0x72, 0x73, 0x75, 0x77, 0x78,
    0x7A, 0x7C, 0x7D, 0x7F, 0x81, 0x82, 0x84, 0x86, 0x88, 0x89, 0x8B, 0x8D,
    0x8F, 0x91, 0x92, 0x94, 0x96, 0x98, 0x9A, 0x9C, 0x9E, 0xA0, 0xA2, 0xA4,
    0xA6, 0xA8, 0xAA, 0xAC, 0xAE, 0xB0, 0xB2, 0xB4, 0xB6, 0xB8, 0xBA, 0xBC,
    0xBF, 0xC1, 0xC3, 0xC5, 0xC7, 0xCA, 0xCC, 0xCE, 0xD1, 0xD3, 0xD5, 0xD7,
    0xDA, 0xDC, 0xDF, 0xE1, 0xE3, 0xE6, 0xE8, 0xEB, 0xED, 0xF0, 0xF2, 0xF5,
    0xF7, 0xFA, 0xFC, 0xFF };
////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////
void loop() {

  //Sound activated lights
  uint8_t  i, r, g, b;
    uint16_t minLvl, maxLvl, a, scaled;

    a           = CircuitPlayground.mic.peak(10); // 10 ms of audio
    sum        -= lvl[lvlIdx];
    lvl[lvlIdx] = a;
    sum        += a;                              // Sum of lvl[] array
    minLvl = maxLvl = lvl[0];                     // Calc min, max of lvl[]...
    for(i=1; i&lt;FRAMES; i++) {
      if(lvl[i] &lt; minLvl)      minLvl = lvl[i];
      else if(lvl[i] &gt; maxLvl) maxLvl = lvl[i];
    }

    // Keep some minimum distance between min &amp; max levels,
    // else the display gets "jumpy."
    if((maxLvl - minLvl) &lt; 40) {
      maxLvl = (minLvl &lt; (512-40)) ? minLvl + 40 : 512;
    }
    avgLo = (avgLo * 7 + minLvl + 2) / 8; // Dampen min/max levels
    avgHi = (maxLvl &gt;= avgHi) ?           // (fake rolling averages)
      (avgHi *  3 + maxLvl + 1) /  4 :    // Fast rise
      (avgHi * 31 + maxLvl + 8) / 32;     // Slow decay

    a = sum / FRAMES; // Average of lvl[] array
    if(a &lt;= avgLo) {  // Below min?
      scaled = 0;     // Bargraph = zero
    } else {          // Else scale to fixed-point coordspace 0-2560
      scaled = 2560L * (a - avgLo) / (avgHi - avgLo);
      if(scaled &gt; 2560) scaled = 2560;
    }
    if(scaled &gt;= peak) {            // New peak
```

```
      peakV = (scaled - peak) / 4;  // Give it an upward nudge
      peak  = scaled;
    }

    uint8_t  whole  = scaled / 256,    // Full-brightness pixels (0-10)
             frac   = scaled &amp; 255;    // Brightness of fractional pixel
    int      whole2 = peak / 256,      // Index of peak pixel
             frac2  = peak &amp; 255;      // Between-pixels position of peak
    uint16_t a1, a2;                   // Scaling factors for color blending

    for(i=0; i&lt;10; i++) {              // For each NeoPixel...
      if(i &lt;= whole) {                // In currently-lit area?
        r = pgm_read_byte(&amp;reds[i]),   // Look up pixel color
        g = pgm_read_byte(&amp;greens[i]),
        b = pgm_read_byte(&amp;blues[i]);
        if(i == whole) {               // Fraction pixel at top of range?
          a1 = (uint16_t)frac + 1;     // Fade toward black
          r  = (r * a1) &gt;&gt; 8;
          g  = (g * a1) &gt;&gt; 8;
          b  = (b * a1) &gt;&gt; 8;
        }
      } else {
        r = g = b = 0;                 // In unlit area
      }
      // Composite the peak pixel atop whatever else is happening...
      if(i == whole2) {                // Peak pixel?
        a1 = 256 - frac2;              // Existing pixel blend factor 1-256
        a2 = frac2 + 1;                // Peak pixel blend factor 1-256
        r  = ((r * a1) + (0x84 * a2)) &gt;&gt; 8; // Will
        g  = ((g * a1) + (0x87 * a2)) &gt;&gt; 8; // it
        b  = ((b * a1) + (0xC3 * a2)) &gt;&gt; 8; // blend?
      } else if(i == (whole2-1)) {     // Just below peak pixel
        a1 = frac2 + 1;                // Opposite blend ratios to above,
        a2 = 256 - frac2;              // but same idea
        r  = ((r * a1) + (0x84 * a2)) &gt;&gt; 8;
        g  = ((g * a1) + (0x87 * a2)) &gt;&gt; 8;
        b  = ((b * a1) + (0xC3 * a2)) &gt;&gt; 8;
      }
      CircuitPlayground.strip.setPixelColor(i,
        pgm_read_byte(&amp;gamma8[r]),
        pgm_read_byte(&amp;gamma8[g]),
        pgm_read_byte(&amp;gamma8[b]));
    }
    CircuitPlayground.strip.show();

    peak += peakV;
    if(peak &lt;= 0) {
      peak  = 0;
      peakV = 0;
    } else if(peakV &gt;= -126) {
      peakV -= 2;
    }

    if(++lvlIdx &gt;= FRAMES) lvlIdx = 0;

////////////////////////////////////////////////////////////////////////////
//BUTTONS
//
//read the buttons
  leftButtonState =  (CircuitPlayground.leftButton());
  //not pressed = 0, pressed = 1
  rightButtonState = (CircuitPlayground.rightButton());

//Left Button
//compare current button states to previous button states
  if(leftButtonState != leftButtonLastState){          //the state has changed
    if(leftButtonState == 1){                         //went from off to on, it's
pressed
      usbMIDI.sendControlChange(LEFT_BUTTON_CC, 1, CHANNEL); //send MIDI note ON
```

```
command
      CircuitPlayground.redLED(HIGH); //turn on the red LED
    }
    else{                   //the button went from on to off, it isn't pressed
      usbMIDI.sendControlChange(LEFT_BUTTON_CC, 0, CHANNEL);  //send MIDI note OFF
      CircuitPlayground.redLED(LOW);                    //turn off the red LED
    }
    leftButtonLastState = leftButtonState;       //toggle
  }


//Right Button
  if(rightButtonState != rightButtonLastState){
    if(rightButtonState == 1){
      usbMIDI.sendControlChange(RIGHT_BUTTON_CC, 1, CHANNEL);
      CircuitPlayground.redLED(HIGH);
    }
    else{
      usbMIDI.sendControlChange(RIGHT_BUTTON_CC, 0, CHANNEL);
      CircuitPlayground.redLED(LOW);
    }
    rightButtonLastState = rightButtonState;
  }
////////////////////////////////////////////////////////////////////////////
//Cap sense pads
//
//read the capacative pads
  int cap1  = CircuitPlayground.readCap(1);
  int cap0  = CircuitPlayground.readCap(0);
  int cap2  = CircuitPlayground.readCap(2);
  int cap3  = CircuitPlayground.readCap(3);
  int cap12 = CircuitPlayground.readCap(12);
  int cap6  = CircuitPlayground.readCap(6);
  int cap9  = CircuitPlayground.readCap(9);
  int cap10 = CircuitPlayground.readCap(10);



  //cap1 Crossfader Deck B

  if(cap1 &gt; CAPMIN){
       cap1State=1;
    }
    else{
       cap1State=0;
    }
    if(cap1State != cap1LastState){
      if(cap1State==1){
        usbMIDI.sendControlChange(CAP1_CONTROL, 127, CHANNEL);
        CircuitPlayground.redLED(HIGH);
      }
      else {
        usbMIDI.sendControlChange(CAP1_CONTROL, 0, CHANNEL);
        CircuitPlayground.redLED(LOW);
      }
      cap1LastState = cap1State;
    }

  //cap0
    if(cap0 &gt; CAPMIN){
       cap0State=1;
    }
    else{
       cap0State=0;
    }
    if(cap0State != cap0LastState){
      if(cap0State==1){
        usbMIDI.sendControlChange(CAP0_CONTROL, 1, CHANNEL);
        CircuitPlayground.redLED(HIGH);
```

```
      }
      else {
        usbMIDI.sendControlChange(CAP0_CONTROL, 0, CHANNEL);
        CircuitPlayground.redLED(LOW);
      }
      cap0LastState = cap0State;
    }

//cap2
      if(cap2 &gt; CAPMIN){
         cap2State=1;
      }
      else{
         cap2State=0;
      }
      if(cap2State != cap2LastState){
        if(cap2State==1){
          usbMIDI.sendControlChange(CAP2_CONTROL, 1, CHANNEL);
          CircuitPlayground.redLED(HIGH);
        }
        else {
          usbMIDI.sendControlChange(CAP2_CONTROL, 0, CHANNEL);
          CircuitPlayground.redLED(LOW);
        }
      cap2LastState = cap2State;
      }

//cap3
      if(cap3 &gt; CAPMIN){
         cap3State=1;
      }
      else{
         cap3State=0;
      }
      if(cap3State != cap3LastState){
        if(cap3State==1){
          usbMIDI.sendControlChange(CAP3_CONTROL, 1, CHANNEL);
          CircuitPlayground.redLED(HIGH);
        }
        else {
          usbMIDI.sendControlChange(CAP3_CONTROL, 0, CHANNEL);
          CircuitPlayground.redLED(LOW);
        }
      cap3LastState = cap3State;
      }

//cap12 Crossfader Deck B

  if(cap12 &gt; CAPMIN){
       cap12State=1;
    }
    else{
       cap12State=0;
    }
    if(cap12State != cap12LastState){
      if(cap12State==1){
        usbMIDI.sendControlChange(CAP12_CONTROL, 127, CHANNEL);
        CircuitPlayground.redLED(HIGH);
      }
      else {
        usbMIDI.sendControlChange(CAP12_CONTROL, 0, CHANNEL);
        CircuitPlayground.redLED(LOW);
      }
      cap12LastState = cap12State;
    }

//cap6
  if (cap6&gt;CAPMIN){
      int cap6NewVal = map(CircuitPlayground.readCap(6),0,200,0,127);
```

```
      if (abs(cap6 - cap6NewVal&gt;CAPSLOP)) {
          cap6 = cap6NewVal;
          usbMIDI.sendControlChange(CAP6_CONTROL, cap6, CHANNEL);
          CircuitPlayground.redLED(HIGH);
          cap6ON = true;
      }
      else{
        if (cap6ON==true){
           usbMIDI.sendControlChange(CAP6_CONTROL, 0, CHANNEL); //send a 0
          CircuitPlayground.redLED(LOW);
           cap6ON=false;
        }
      }
    }
//cap9
      if(cap9 &gt; CAPMIN){
         cap9State=1;
      }
      else{
         cap9State=0;
      }
      if(cap9State != cap9LastState){
        if(cap9State==1){
          usbMIDI.sendControlChange(CAP9_CONTROL, 1, CHANNEL);
          CircuitPlayground.redLED(HIGH);
        }
        else {
          usbMIDI.sendControlChange(CAP9_CONTROL, 0, CHANNEL);
          CircuitPlayground.redLED(LOW);
        }
      cap9LastState = cap9State;
      }

//cap10
      if(cap10 &gt; CAPMIN){
         cap10State=1;
      }
      else{
         cap10State=0;
      }
      if(cap10State != cap10LastState){
        if(cap10State==1){
          usbMIDI.sendControlChange(CAP10_CONTROL, 1, CHANNEL);
          CircuitPlayground.redLED(HIGH);
        }
        else {
          usbMIDI.sendControlChange(CAP10_CONTROL, 0, CHANNEL);
          CircuitPlayground.redLED(LOW);
        }
      cap10LastState = cap10State;
      }


  // MIDI Controllers should discard incoming MIDI messages.
    while (usbMIDI.read()) {
  }
}
```

You can download the code with the button below. Once downloaded, unzip it and place the directory in your Arduino sketches directory.

**PZ1PizzaBoxDJ.zip**

https://adafru.it/rBK

# Customize the Code

You can see in the top of the sketch there are variables for the different MIDI control change commands. These numbers can be changed, but they correspond to the **Circuit Playground** pins for convenience. These are the MIDI control change numbers you'll need to know when mapping the **PZ-1** to your DJ software. The comments show how I'm mapping them in Traktor.

The sketch also contains the **Circuit Playground** example sketch **mic_meter** to handle the sweet sound-reactive NeoPixel lighting effects.

# Map It

Once you've uploaded the code, verify it is sending the commands by opening your MIDI monitoring software. When you choose the **Circuit Playground** as MIDI input and press the buttons and pads, you'll see the number, value, and channel data streaming in the utility.

Launch your DJ software and head to the MIDI controller configuration area (usually in the preferences). Here you can refer to the help for particular software you're using to assign each MIDI control change signal from the **PZ-1** to the desired software command.


Here is a sample mapping that pairs the **Circuit Playground PZ-1** with **Traktor Pro 2** from **Native Instruments**.

PZ1_Traktor_mapping.tsi.zip

https://adafru.it/rBL

# Conductive Painting

# Cardboard Canvas

Corrugated cardboard makes a pretty good surface for this project. You can usually get a clean, empty box from a local pizza place if you ask nicely, or buy them in bulk online.

A 12" pizza box is a good size for this controller.

## Other Materials

Mylar stencil blanks are available online or at art/graphics supply stores. They come in a convenient 12" x 12" size, an 7 mil is a good thickness for this.

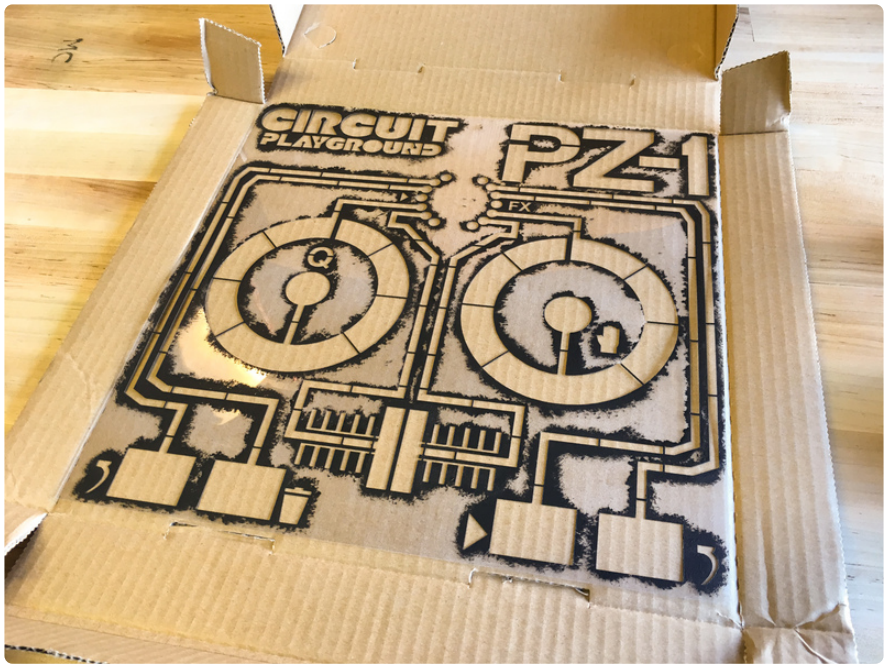
You'll also want a hobby knife, masking tape, and some spray adhesive to keep the stencil mask from lifting up while painting.

# Stencil Design

Your Circuit Playground's capacative pads need to touch the circuit, so that's a good starting point for your design. If you'd like to create the PZ-1 exactly as seen here, download the file below and either print it on regular printer paper to trace, or cut it on a CNC or laser cutter.

**PZ-1_DJ_Stencil.svg**

https://adafru.it/rBM

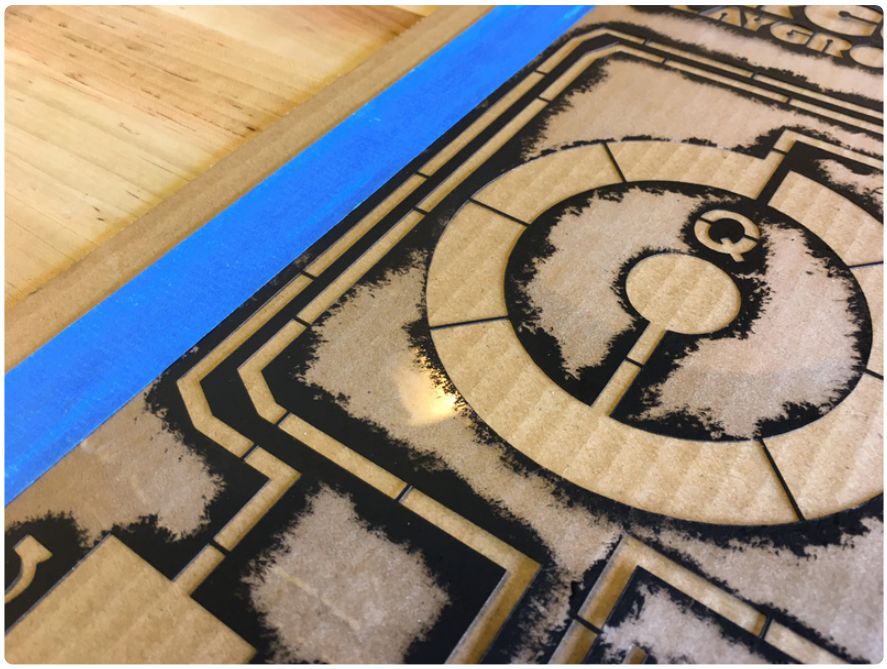



# Use the Stencil

Once you've cut your stencil, you can spray the back side lightly with spray adhesive to help prevent it from lifting up from the cardboard while painting. You'll also want to use masking tape around the edges.

Paint the circuit by loading your stencil brush with conductive paint and then tapping it from above. This vertical tapping helps keep the paint inside the lines, vs strokes which tend to leak underneath as the bristles find the gaps.

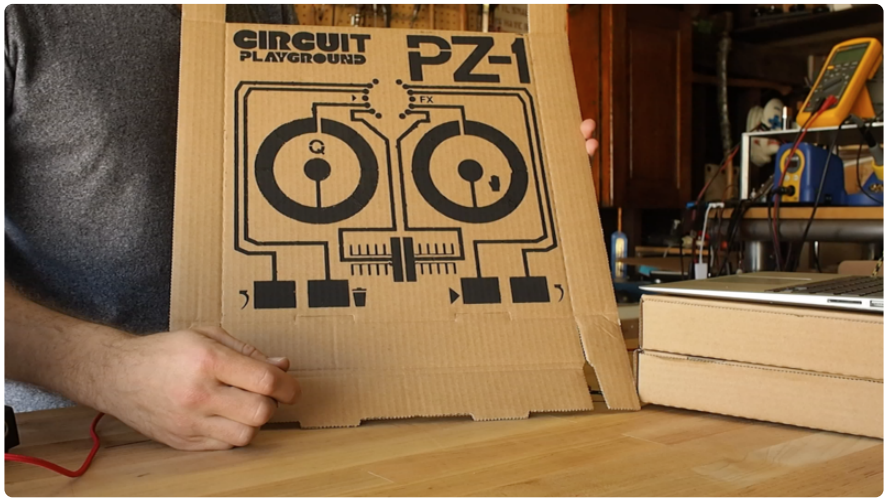
Work your way across the entire surface.




FX


Once you've finished, savor the exciting reveal! Lift of your stencil and admire the beautiful results!

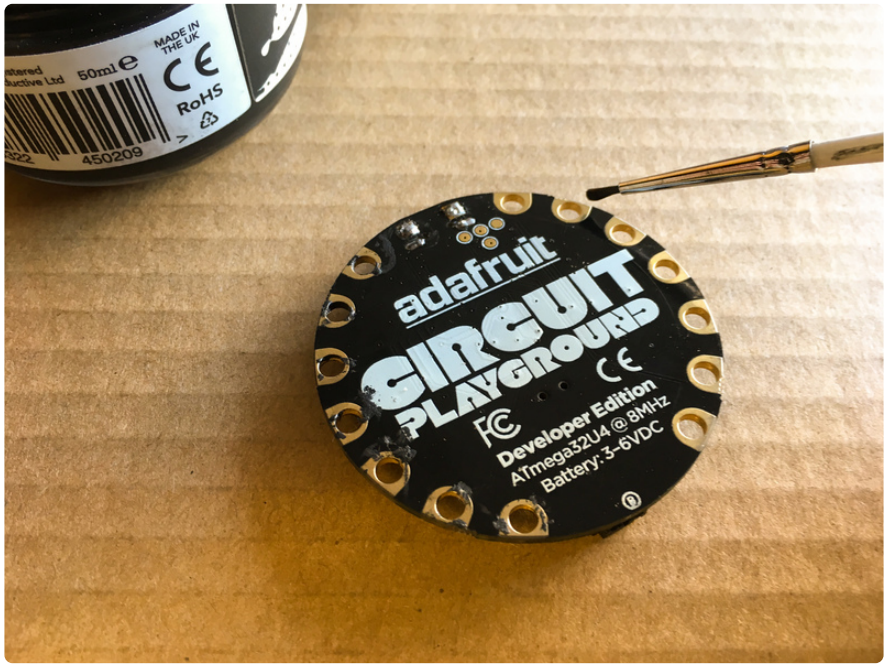
To prevent the stencil from flopping around too much, I designed small tabs to hold parts together. These break the flow of the circuits, however, so it's necessary to repair the breaks with small dabs of paint after you've removed the Mylar stencil mask.

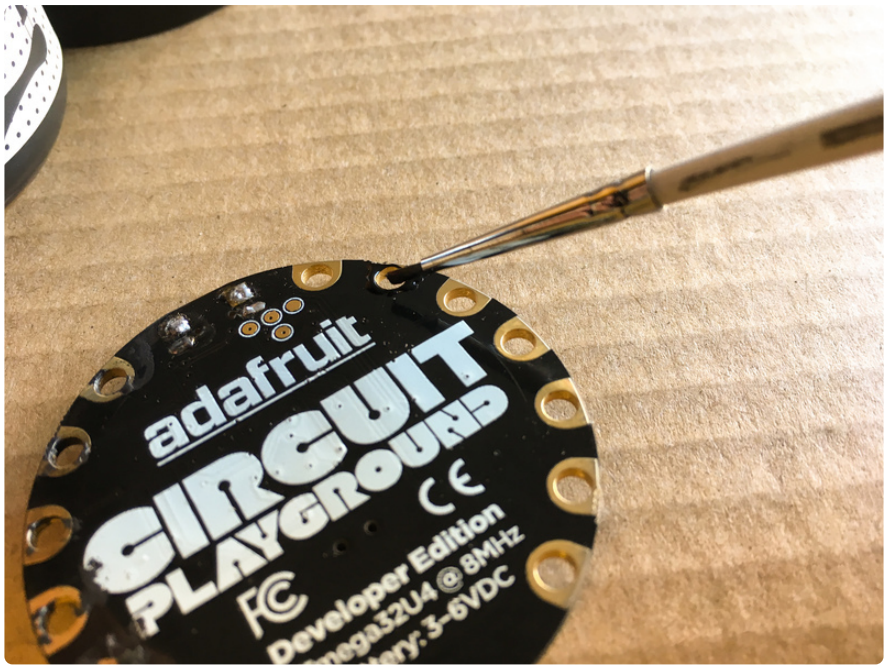

CIRCUIT
PLAYGROUND
PZ-1
FX

# Mount the Circuit Playground

Adding your Circuit Playground to the controller is simple. You'll dab a bit of conductive paint on each of the copper touch pads on the bottom side of the Circuit Playground, then carefully set it down on the corresponding painted trace pads on your cardboard. You can add a bit more paint inside the pads running up to the topside and back over to the outside of the board -- this will not only act as a conductive trace, but also keep the board "glued" down to the controller.

Don't blob on excessive amounts of paint under the Circuit Playground or it may squish out and short your circuit!

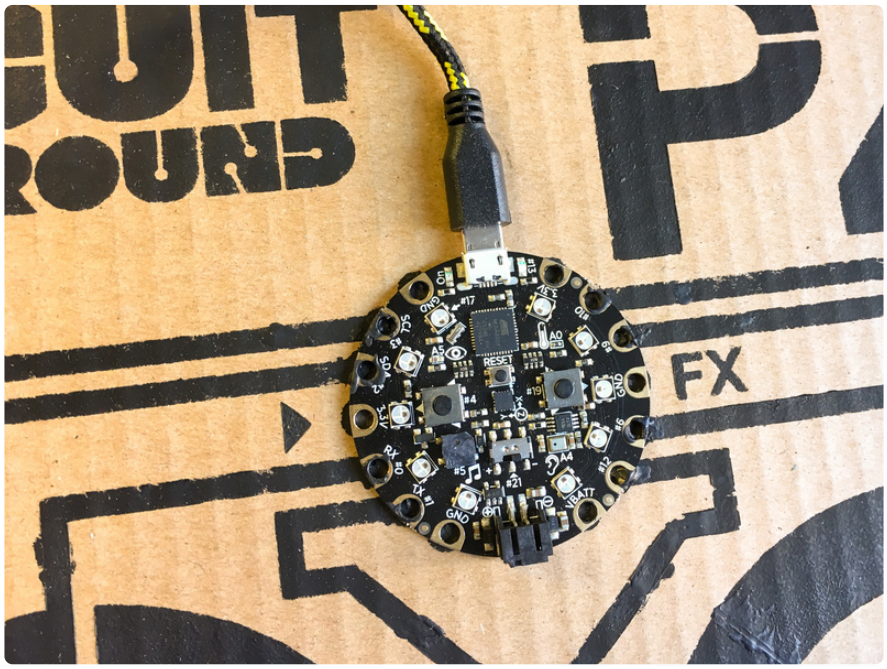



adafruit
CIRCUIT
PLAYGROUND
Developer Edition


FX

# Rock the Crowd


You're ready to let the PZ-1 take control of your DJ software. Plug the controller into your laptop with the USB cable.

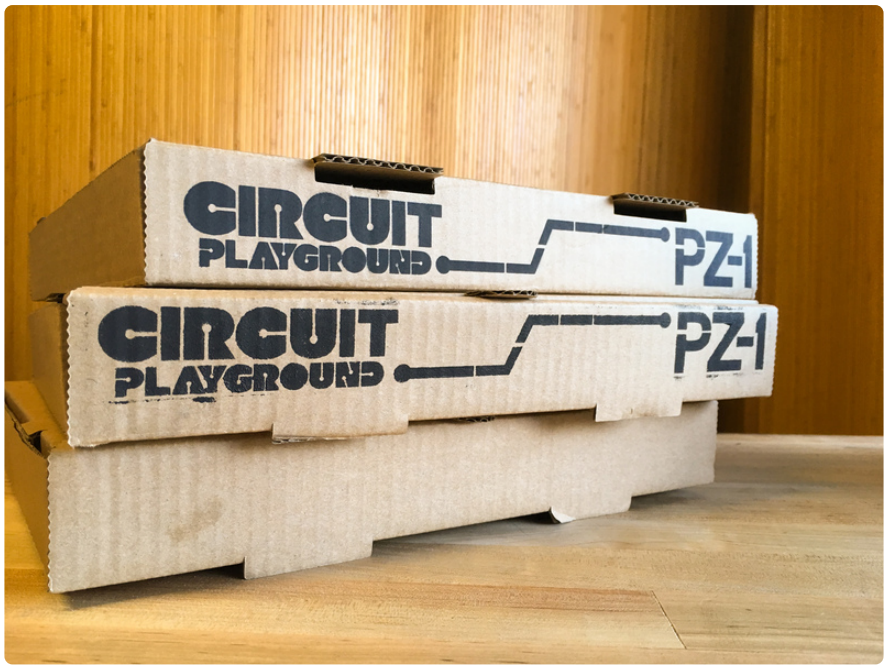



You can keep the box unfolded and flat, or fold it up.


The conductive paint is water-based, so it can smudge and smear from regular use, or from the spilling of beverages. You can use a coat of fixative spray or some Mod Podge to seal it. It'll still work just fine!

What's the time? It is time to DJ.

Once you've rocked the crowd all the way out and back, and everyone has gone home for a good night's rest, fold up your PZ-1 and go deliver some delicious tracks at the next party! (Yes, sometimes the next party is in your workshop.)


CIRCUIT
PLAYGROUND
PZ-1
CIRCUIT
PLAYGROUND
PZ-1