LED Trampoline
Created by Ruiz Brothers

https://learn.adafruit.com/led-trampoline

Last updated on 2023-08-29 03:41:40 PM EDT
# Table of Contents

**Overview**  
- Interactive NeoPixels  
- Trampoline  
- Reel of NeoPixel LEDs  
- Prerequisite Guides  
- Electronic Components  
- Supplies  
- Cool Tools!

**Circuit Diagram**  
- Circuit Diagram  
- Wired Connections

**Software**  
- Setup  
- Helper Functions  
- Color Setup  
- try and except  
- Main Loop  
- Flash and Fade Mode  
- Sparkle Mode  
- Color Chase Mode  
- MicroUSB Dock for Dev Boards

**3D Printing**  
- What If I Don't Have A 3D Printer?  
- Project Enclosure Parts  
- Slice Settings  
- 3D Printed Parts  
- Design Source Files

**Vibration Switch**  
- Wires for Vibration Sensor  
- Prep Wires  
- Heat Shrink Tubing  
- Prevent Shorts  
- Connect Wires to Vibration Sensor  
- Insulate Exposed Leads  
- More Heat Shrink

**Button**  
- Wires for Button  
- Wire Up Push Button  
- Heat Shrink

**Perma-Proto**  
- Perma-Proto PCB  
- Wire Perma-Proto PCB

**JST Plugs**  
- 3-Pin JST Plug & Receptacle
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NeoPixels</td>
<td>29</td>
</tr>
<tr>
<td>• Prep Wires</td>
<td></td>
</tr>
<tr>
<td>• Extended JST Plugs</td>
<td></td>
</tr>
<tr>
<td>• Female JST Receptacles</td>
<td></td>
</tr>
<tr>
<td>• Count and Cut NeoPixel Strip</td>
<td></td>
</tr>
<tr>
<td>• Wire JST to NeoPixel Strip</td>
<td></td>
</tr>
<tr>
<td>ItsyBitsy M0</td>
<td>31</td>
</tr>
<tr>
<td>• Connectors</td>
<td></td>
</tr>
<tr>
<td>• Wire Perma-Proto</td>
<td></td>
</tr>
<tr>
<td>• Wire ItsyBitsy</td>
<td></td>
</tr>
<tr>
<td>• Final Wiring</td>
<td></td>
</tr>
<tr>
<td>Case Assembly</td>
<td>34</td>
</tr>
<tr>
<td>• Install Case</td>
<td></td>
</tr>
<tr>
<td>• Install Button and Cover</td>
<td></td>
</tr>
<tr>
<td>• Test NeoPixel Strips</td>
<td></td>
</tr>
<tr>
<td>Trampoline Setup</td>
<td>36</td>
</tr>
<tr>
<td>• Remove Padding</td>
<td></td>
</tr>
<tr>
<td>• Installing NeoPixel Strips</td>
<td></td>
</tr>
<tr>
<td>• Threading NeoPixel Strips</td>
<td></td>
</tr>
<tr>
<td>• Attaching the Vibration Sensor</td>
<td></td>
</tr>
<tr>
<td>• Clip Case to Trampoline</td>
<td></td>
</tr>
</tbody>
</table>
Overview

Interactive NeoPixels

In this project we'll build an LED Trampoline. This is a really fun project that lights up when you jump on it! These trampolines are meant for exercising and jumping on this thing for just a few minutes feels like a working out so perfect for tiring out kids, just be sure to supervise them. It works really well and looks amazing at night, especially around a fire pit with a fog machine.

Trampoline

I found this trampoline at my local Walmart near the sporting goods isle. It was about $25. It has 6 detachable legs which is convenient for storing. It's meant for cardio workouts and features elastic straps instead of metal springs, which is pretty nice.

1 x 36" Trampoline
Calm 36-Inch Trampoline Cardio

https://www.walmart.com/ip/Calm-36-Inch-Trampoline-Cardio-Blue/265966558
Reel of NeoPixel LEDs

The trampoline is about 36 inches in diameter, which is fairly large so I used a 3 meter long LED strip (1) to wrap around the whole framing. There's a little bit extra here, about 10 pixels, but this should give us plenty of LEDs to work with. The Low density strip contains 30 LEDs per meter, giving us a total of 90 RBG LEDs. You can choose to use a medium/high density, but be aware there are hardware limitations in Circuit Python.

Prerequisite Guides

If you're new to Circuit Python, NeoPixels and Soldering, take a moment to walk through the following guides to get you started.

- Welcome to Circuit Python (1)
- Adafruit NeoPixel Uber Guide (1)
- Adafruit's Guide to Excellent Soldering (1)

Electronic Components

The Adafruit ItsyBitsy M0 and NeoPixel strip are the main electronic components used in this project.

1 x Adafruit ItsyBitsy M0
<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x</td>
<td>Vibration Sensor Switch</td>
<td>Medium sensitivity non-directional vibration</td>
</tr>
<tr>
<td>1 x</td>
<td>Adafruit NeoPixel Digital RGB LED Strip</td>
<td>3 meters - White 60 LEDs per meter</td>
</tr>
<tr>
<td>1 x</td>
<td>Soft Tactile Button</td>
<td>8mm x 8mm pack of 10</td>
</tr>
<tr>
<td>1 x</td>
<td>USB Battery Pack</td>
<td>4000mAh - 5V @ 1A</td>
</tr>
<tr>
<td>1 x</td>
<td>26AWG Wire</td>
<td>Silicone Cover Stranded-Core Wire - 25ft</td>
</tr>
<tr>
<td>1 x</td>
<td>Solder Wire</td>
<td>Solder Spool - 1/4 lb SAC305 RoHS lead-free / 0.031&quot; rosin-core - 0.25 lb / 100 g</td>
</tr>
</tbody>
</table>

### Supplies

Wires and some handy supplies.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x</td>
<td>Wire Strippers</td>
<td>Hakko Professsional Quality 20-30 AWG Wire Strippers - CSP-30-1</td>
</tr>
<tr>
<td>1 x</td>
<td>Wire Cutters</td>
<td>Flush diagonal cutters - CHP170</td>
</tr>
</tbody>
</table>

### Cool Tools!

These help make the project a smooth building experience. You don't need them all of them, but I recommend them.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x</td>
<td>Panavise</td>
<td></td>
</tr>
<tr>
<td>Item</td>
<td>Quantity</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Panavise Jr. - PV-201</td>
<td>1 x</td>
<td></td>
</tr>
<tr>
<td>Ultimaker 2+</td>
<td>1 x</td>
<td>3D Printer</td>
</tr>
</tbody>
</table>

Circuit Diagram

This provides a visual reference for wiring of the components. They aren't true to scale, just meant to be used as reference. The vibration switch wasn't available in Fritzing so instead we have a tilt switch – These are basically the same pin layout.
Wired Connections

Ground will be shared by the neopixel strip, vibration switch and the button. The circuit will be powered via USB battery bank.

- Pin 10 from ItsyBitsy M0 to DIN on NeoPixel Strip
- Pin 9 from ItsyBitsy M0 to Vibration Switch
- Pin 7 from ItsyBitsy M0 to Button Switch
- Pin USB from ItsyBitsy M0 to +5V on NeoPixel Strip
- Ground from ItsyBitsy M0 to NeoPixel strip, vibration switch and button

Software

Now we're going to take a look at the code behind our LED trampoline light show!

Copy the following file to your Itsy Bitsy CIRCUITPY drive, and rename it to code.py. Or copy and paste it into your current code.py, replacing its current contents. Save your file, and you're ready to go!

```python
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
# # SPDX-License-Identifier: MIT
import random
```
import time
import board
import digitalio
import neopixel

pixel_pin = board.D10  # The pin the NeoPixels are connected to
button_switch_pin = board.D9  # Pin button is attached to
vibration_switch_pin = board.D7  # Pin vibration switch is attached to.
pixel_count = 40  # Number of pixels in your strip
chase_color_duration = 3  # Seconds each color lasts in the color chase mode

pixels = neopixel.NeoPixel(pixel_pin, pixel_count,
                           brightness=.4, auto_write=False)
button_switch = digitalio.DigitalInOut(button_switch_pin)
button_switch.direction = digitalio.Direction.INPUT
button_switch.pull = digitalio.Pull.UP

vibration_switch = digitalio.DigitalInOut(vibration_switch_pin)
vibration_switch.direction = digitalio.Direction.INPUT
vibration_switch.pull = digitalio.Pull.UP

led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT

# Colors:
RED = (255, 0, 0)
YELLOW = (255, 150, 0)
ORANGE = (255, 40, 0)
GREEN = (0, 255, 0)
TEAL = (0, 255, 120)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)
MAGENTA = (255, 0, 20)
WHITE = (255, 255, 255)

# Sparkle colors:
GOLD = (255, 222, 30)
PINK = (242, 90, 255)
AQUA = (50, 255, 255)
JADE = (0, 255, 40)
AMBER = (255, 100, 0)

def cycle_sequence(seq):
    while True:
        yield from seq

def fade_control():
    brightness_value = iter([r / 15 for r in range(15, -1, -1)])
    while True:
        # pyLint: disable=stop-iteration-return
        pixels.brightness = next(brightness_value)
        pixels.show()
        yield

def sparkle_code(color_values):
    (red_value, green_value, blue_value) = color_values
    p = random.randint(0, (pixel_count - 2))
    pixels[p] = (red_value, green_value, blue_value)
    pixels.show()
    pixels[p] = (red_value // 2, green_value // 2, blue_value // 2)
    pixels.show()
    pixels[p + 1] = (red_value // 10, green_value // 10, blue_value // 10)
    pixels.show()
fade = fade_control()
flash_color = cycle_sequence([RED, YELLOW, ORANGE, GREEN, TEAL, CYAN, BLUE, PURPLE, MAGENTA, WHITE])

sparkle_color_list = (MAGENTA, PINK, GOLD, AQUA, JADE, AMBER)
sparkle_color_index = 0

chase_color_list = (RED, ORANGE, YELLOW, GREEN, BLUE, PURPLE)
chase_color_index = 0
chase_color_cycle = chase_color_list[chase_color_index]
offset = 0

chase_last_color = time.monotonic()
chase_next_color = chase_last_color + chase_color_duration

button_state = None

mode = 0
print("Mode: ", mode)

initial_time = time.monotonic()

while True:
    try:
        now = time.monotonic()
        if not button_switch.value and button_state is None:
            button_state = "pressed"
        if button_switch.value and button_state == "pressed":
            print("Mode Change")
            led.value = True
            pixels.fill((0, 0, 0))
            mode += 1
            button_state = None
            if mode > 2:
                mode = 0
                print("Mode: ", mode)
        else:
            led.value = False
        if mode == 0:
            try:
                if not vibration_switch.value:
                    print("Flash and fade mode activate!")
                    fade = fade_control()
                    pixels.fill(next(flash_color))
                    pixels.show()
                except StopIteration:
                    pass
            except StopIteration:
                pass
        if mode == 1 and not vibration_switch.value:
            print("Sparkle mode activate!")
            pixels.brightness = 1
            sparkle_color_index = (sparkle_color_index + 1) % len(sparkle_color_list)
            sparkle_code(sparkle_color_list[sparkle_color_index])
        if mode == 2 and not vibration_switch.value:
            pixels.brightness = 1
            for i in range(0, pixel_count):
                c = 0
                if ((offset + i) % 8) < 4:
                    c = chase_color_cycle
                pixels[i] = c
                pixels[(pixel_count - 1) - i] = c
            pixels.show()
            offset += 1
        if now >= chase_next_color:
chase_color_index = (chase_color_index + 1) % len(chase_color_list)
chase_color_cycle = chase_color_list[chase_color_index]
pixels.fill((0, 0, 0))
chase_last_color = now
chase_next_color = chase_last_color + chase_color_duration
except MemoryError:
    pass

Setup

The first thing we do after importing the libraries we'll be using is set up the pins so our code knows where to look for the switches and LEDs.

Then we set the `pixel_count = 40`. This project uses 80 pixels total. However, there were hardware limitations with running 80 pixels. So, we've cut the strip into two, and attached the two strips of 40 to the same pin. Since they're attached to the same pin, the code doesn't know the difference and sends the same data to both strips. That's why the total is set to 40!

We also set the `chase_color_duration = 3` which determines how long in seconds each color lasts in the color chase mode. You can change this number if you'd like your colors to last a longer or shorter time.

Next we setup our switches, and the red LED which we will be using as a status light for mode changes.

Next up is a list of color variables. We have a lot of color in this project! Since we reuse the colors multiple times, we've created a variable for each color so we can call it by a name instead of the `(R, G, B)` color tuple.

Helper Functions

Then we have a few functions that we'll be using. `cycle_sequence` is a helper function we use with generators. You can learn about how generators work in the Generate Your Colors section of the Hacking Ikea Lamps with CPX guide (). In the LED Trampoline project, we're using a generator with our Flash and Fade mode. The fade takes a certain amount of time, and it's possible you could try to jump faster than the...
fade animation takes. In this case, the generator allows for the fade animation to be interrupted and a new color fade to begin.

We have `fade_control` which is what handles our flash and fade. Each time we call it, it starts with the brightness at 100 percent and then steps down 15 times to 0. This is how the fade works!

The `sparkle_code` handles the Sparkle mode. It sets a random pixel to a color and then sets one next to it to a faded version of that color. That is how we achieve a sparkle effect. This continues and once they're all full, will continue to randomly overwrite lit pixels. The sparkle code is super fast! This means that each time you jump, you'll trigger more than one sparkle at a time. It looks amazing!

We set `fade = fade_control()` so we can later call `fade_control` using `fade`.

Next we setup the list of colors we're going to use in Flash and Fade. `flash_color` uses the `cycle_sequence` generator to go through the list of colors and start over endlessly. This way you can jump as long as you want on the Flash and Fade mode and the colors keep right on changing!

### Color Setup

We set the colors for the Sparkle mode using `sparkle_color_list` and the six special colors chosen for the Sparkle mode. They're slightly different than the main color list because these colors look especially great with the way the `sparkle_code` works. You can feel free to change these to any of the other colors or add more if you like!

`sparkle_color_index` is used to tell the code to start with the first color on the list. Remember, in CircuitPython, counting begins with 0, so 0 represents the first color in the list!

We set the colors for the Color Chase mode using `chase_color_list` and six of our color variables. You could add to this and change which colors are used if you like!

`chase_color_index` is the same as the sparkle version - it tells the code to start with the first color on the list.

The next few lines setup the chase color changing. We use `time.monotonic()` because we'd like to be able to set the amount of time the chase mode is a certain color, but we want the code to continue waiting in case we decide to jump or change...
modes. `time.monotonic()` is called non-blocking, meaning the code is not stopped by it. Learn more about non-blocking code in the Passing Time section of the Hacking Ikea Lamps with CPX guide.

We set `mode = 0`. This tells the code to begin with the first mode. We'll start with Flash and Fade! We `print("Mode: ", mode)` which is "Mode:" and the current mode, which is `0` before we start. So if you are connected to the serial console during testing, you'll see `Mode: 0`.

To use `time.monotonic()`, you always set a start time. That's the last thing we do before we start our main loop!

```
try and except
```

```
while True: says to run the code indented beneath it forever.
```

The first and last thing inside our `while True:` loop, are `try:` and `except` `MemoryError: pass`. This is due to a very rare situation where the code returns a memory error and stops.

```
try: says to first try running all of the code indented under the try. If there is an error, the code moves to the except: and completes the code underneath. In this case it will pass which says to just keep right on going and start again with the try.
```

Since this project is meant to be attached to a trampoline and continue to run while you have fun jumping, we've included this try and except code so if it does run into that problem, it will keep going. This way you can keep on jumping even if the code runs into this problem! Otherwise, you'd have to reset the board. And what's the fun in that!

**Main Loop**

First, we make it so the button must be pressed and released before the code switches modes. This way you cannot accidentally skip modes by holding down the button too long.

Each time the mode is switched, it turns off the LEDs so you're starting your new mode with a clean slate. It blinks the little red LED, which can't be seen once the project is assembled, but is super handy when you're testing the project while still attached to your computer. Each time the button is pressed and released, it increases
the mode by 1. Once that number is greater than 2 (remember we start counting at 0, so 2 is the third mode) it resets to 0. This way we cycle through each of the three modes and then begin again with the first! Each time it prints the mode to the serial output, which you can see if you're connected while testing the project on your computer.

Flash and Fade Mode

If the mode is 0, we start the Flash and Fade code. Here we have another try and except. This is due to the way that the flash_control generator works. flash_control begins at 100 percent and ends at 0, and would normally simply stop here as it has iterated through all possible values. We want it to keep going so we can jump again, flash again and fade again. So, we've included except StopIteration: pass to keep the Flash and Fade mode moving.

Inside our try, we have if not vibration_switch.value: . The vibration switch defaults to "True" when not triggered, and becomes "False" when triggered. So, if we're looking to do a thing when the switch is triggered, we want to refer to when it is False which is done by using if not. This way, when we jump, we trigger the next set of events to happen. We print that the mode is activated! We call fade, we fill the pixels with the next color from our flash_color list, and we call pixels.show() which tells the pixels to light up with whatever we just set. In this case, it will flash the color and fade to black. Then we call next(fade) which tells the fade_control to start again.

Sparkle Mode

If the mode is 1 and we've triggered the vibration switch, then we start the Sparkle code. We print the mode is activated!

When you change modes in the middle of the Flash and Fade code, the brightness stays at the level it was at the exact moment that you switched modes. This means you could switch modes when the brightness is set to 0. If that happened, you'd get no sparkles at all! That's no good. So, we set pixel.brightness = 1 to set it back to 100%, so you get all the sparkle goodness regardless of where the brightness was when you switched.

The next line tells the sparkle code that sparkle_color_list should cycle through all the colors. And the last line says run the sparkle code repeatedly, cycling through the list of colors as you jump, and to keep doing this until you change modes!
Color Chase Mode

If the mode is 2 and we've triggered the vibration switch, we start the Color Chase code. We print the mode is activated! And, for the same reason as in Sparkle mode, we set the brightness to 100%.

The next bit is the core of the Chase mode. This is the chase code. This tells the strips to light up every 4 LEDs with 4 off between. Then it says, each time the switch is triggered, move the groups of lit LEDs towards the middle. In our case, since we've split our LED strips into 2, the chase will happen in two parts, one on each half, moving towards the middle of that half. They meet and meld in the middle!

The last section is what tells the chase code to change colors. We told it at the beginning to change colors every 3 seconds. So, every time 3 seconds passes, chase mode changes color. It forever cycles through the list of colors we provided during color setup. This happens whether or not you're jumping, because it's entirely based on time!

And that's it!

MicroUSB Dock for Dev Boards

Like the dock in this photo? You can 3D Print and build your own!

3D Printed MicroUSB Dock
3D Printing

What If I Don't Have A 3D Printer?

Not to worry! You can use a 3D printing service such as 3DHubs or MakeXYZ to have a local 3D printer operator 3D print and ship you parts to you. This is a great way to get your parts 3D printed by local makers. You could also try checking out your local Library or search for a Maker Space.

Project Enclosure Parts

The case and cover snap fit together. The ItsyBitsy board clips into the case with the microUSB facing the smaller opening. The push button is fitted over the post. The 3D printed button cover is fitted over the rubber part of the push button. The clip is press fitted into the back of the case.

<table>
<thead>
<tr>
<th>trampoline-case.stl</th>
<th>Fits the ItsyBitsy and Push Button</th>
<th>Prints with no support</th>
</tr>
</thead>
<tbody>
<tr>
<td>trampoline-cover.stl</td>
<td>Snap fits on top of the case</td>
<td>Prints with no support</td>
</tr>
</tbody>
</table>
Slice Settings

These parts have been tested and 3D printed on an Ultimaker 2+ and 3 using PLA filament. The parts were sliced using CURA 3.x with the following slice settings.

220C extruder temp
65c bed temp
0.2 layer height
0.38 line width
2 Wall Line Count – 0.4 nozzle
20% infill
70mm/s print speed

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Support Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>trampoline-case-clip.stl</td>
<td>Snap fits onto the back of the case.</td>
<td>No support</td>
</tr>
<tr>
<td>trampoline-strip-clip.stl</td>
<td>Attaches to trampoline frame and hangs LEDs strip</td>
<td>Print 15 of these.</td>
</tr>
<tr>
<td>trampoline-button-cover.stl</td>
<td>Fits over rubber part of the push button.</td>
<td>No support</td>
</tr>
</tbody>
</table>
3D Printed Parts

All of the parts are 3D printed with FDM type 3D printers using various colored filaments. All of the parts are separated into pieces to make 3D printing easier. Assembly is pretty easy and straight forward. Use the links below to download the STLs files.

- Download STLs on Thingiverse
- Download STLs on Pinshape
- Download STLs on Youmagine

Design Source Files

The enclosure assembly was designed in Fusion 360. This can be downloaded in different formats like STEP, SAT and more. Electronic components like the board, displays, connectors and more can be downloaded from our Fusion 360 CAD parts github repo.

- Trampoline Fusion 360 Design
- ItsyBitsy Case Fusion 360 Design
- Adafruit CAD Parts Github
Vibration Switch

Wires for Vibration Sensor
We'll need two long wires for connecting the vibration sensor to the ItsyBitsy board. Mine were about 23cm (9in) in length. I suggest using different colored wires to help tell the connections apart.

Prep Wires

Using wire strippers, remove a bit of insulation from the tips of each wire. Since I'm using 30AWG silicone covered wires, they're stranded wires so I suggest tinning the tips with a bit of solder. This will help prevent them from fraying when attaching them to pins and other wires. I used a third helping hand to keep the wires steady, which is super helpful when soldering.
Heat Shrink Tubing
I used a couple of pieces of heat shrink tubing to keep the two wires bundled together. Since this project will use a quite a few wired connections, keeping them nice and tidy is super useful.
Prevent Shorts
Before soldering the wires to the leads of the vibration sensor, add a piece of heat shrink tubing to each wire. This will prevent the two leads from shorting when assembling.

Connect Wires to Vibration Sensor
The two leads on the vibration sensor are long so we can cut them shorter. To make it easier to attach the wires to the leads, add a bit of solder. Be sure to add pieces of heat shrink tubing before soldering the wires to the leads. I secured the vibration sensor to a set of third helping hands to keep it steady while soldering the connections.
Insulate Exposed Leads

Those extra pieces of heat shrink tubing can slide over the exposed leads and cover them up so they don't accidentally short. I like to use a lighter to heat up the tubing.
More Heat Shrink
While prototyping the project, I flexed the leads so much that they ripped out of the capsule. So, to avoid damaging the leads I suggest heat shrinking the whole sensor. This way, pulling on the wires won't stress the leads. If you don't have the right size of tubing, a dab of hot glue will be suffice.

Button

Wires for Button
We'll need two pieces of wire to connect the button the ItsyBitsy board. These can be a bit short since the button will be mounted close to the ItsyBitsy. I made mine about 2cm (0.7") in length. I used 30AWG silicone covered wires for this because they're thinner and will need to be flexed quite a bit to install into the case.

Wire Up Push Button

Now we can attached the wires to the leads on the button. The wires will need to be soldered to the correct pins, so take note of which side they're connected. The polarity doesn't important so it won't matter which wires goes to ground or input on the ItsyBitsy board. There's four pins and a set of two have linked continuity. The two pins on the side of the body are normally open, while the further apart ones are linked. If you're not sure which leads to use, use a multimeter set to the continuity mode to see which leads are electrically linked.
Heat Shrink

I added a piece of heat shrink tubing to keep the two wires together. I didn't need to insulate the exposed pins because they are held mechanically and won't intersect when installing into the case.
Perma-Proto PCB

There's several wires in the build that will need to connect to the USB (voltage) and ground (G) pins on the ItsyBitsy board. I used a piece perma-proto PCB to extend the voltage and ground pins on the ItsyBitsy board. There's three voltage wires and five ground wires needed. I used a 2x6 piece, a little extra for good measure. To cut the PCB, I used a Dremel rotary tool with a thin cut wheel. I suggest cutting in a well ventilated area and use a dust mask and safety glasses. I used a Panavise (\(\)) to hold the PCB steady.

Wire Perma-Proto PCB

One set of voltage and ground pins will need to be connected to the ItsyBitsy board. We'll need two pieces of wire. I used 26AWG silicone covered wires (\(\)) here to provide the NeoPixel strips with sufficient current. To keep the PCB sturdy while soldering, I used a pair of third helping hands.
JST Plugs

3-Pin JST Plug & Receptacle
I used these 3-pin JST cables () to connect the NeoPixels to the ItsyBitsy board. These made it easy to disconnect the board from the strips. I found this really useful because the NeoPixel strips will be attached to the trampoline. This allows me to remove the ItsyBitsy board from the assembly when I needed to change or update the code.
Prep Wires

I started by trimming the wires of the male connectors short because they're a bit longer than needed. Then, I used wire strippers to remove a bit of insulation for each wire. I tinned the exposed strands of wire with a bit of solder. I found these wires to be pretty stiff. They're 22AWG sized and difficult to attach to the pins on the ItsyBitsy board. To give them a bit more flex, I extended the wires with 26AWG silicone covered wires. The different colors are nice and help tell the connections apart.
Extended JST Plugs
I soldered the pieces of 26AWG wire and insulated the exposed connections with heat shrink tubing. I made sure to keep the colored wires consistent across the two male connectors. It's very important to keep track of the connections to avoid shorting or damaging the neopixel strips.

Female JST Receptacles
The female receptacles will be soldered directly to the pads on the NeoPixel strips and does not need to be extended with flexible wires. I found the length of wires a bit too long so I cut them a bit shorter. Keep each wire tinned with a bit of solder, it'll make it easier to attach them to the pads on the LED strips later. I also added some bits of heat shrink tubing!

NeoPixels
Count and Cut NeoPixel Strip
The 3 meter long NeoPixel strip includes 90 LEDs in total. I wrapped the full length around the trampoline (36" in diameter) and found that I only need 80 LEDs. The remaining ten pixels can be cut off using a pair of scissors. Because of the hardware limitations with CircuitPython, we found running the code with 90 NeoPixels was slow and delayed. To combat this, we can cut the strip in half, so each contains just 40 pixels each. Then, in the code we reduced the pixel count to just 40.
Wire JST to NeoPixel Strip

With our two strips of 40 NeoPixels (80 in total) we can solder the JST receptacles to the pads. It’s very important to note which wire will be attached to the 5V, GND and DIN pads. I connected the male JST plug to the female receptacles and followed the colored wires, going red to 5V, black to GND and yellow to DIN. This way we can keep the connections across the two consistent. Double check the wiring before soldering the wires to the pads on the NeoPixel strips. I found tinning the pads with a bit of solder first makes it easier to attach the wires.
Connectors
Before we solder all of the wires to the ItsyBitsy board, it's a good idea double check all of our wiring. Planning ahead allows you to catch any mistakes we might encounter. Take this opportunity to add more heat shrink!

Wire Perma-Proto

I suggest starting with getting all of the power and ground connections soldered to the piece of perma-proto PCB. I secured it to panavise to keep it in place while soldering. The board can get a bit cluttered so take your time and orient the PCB so your hands don't get in the way while soldering. You'll want to double check your work and ensure the connections are in the right places. The perma-proto PCB has high quality through-holes so it's ok if you need to rework any of the pins.
Wire ItsyBitsy

Now it's time to wire up the ItsyBitsy board. I started with the most difficult connections being the Data Input for the NeoPixel strips. Because these connections have a thicker 26AWG wires, it's hard to share a single pin. I used a pair of flush snips to thin out the wires so that I could insert them both into pin #10. You'll want to double check your work to ensure both connections are solid. Next, we can solder the button to Pin #9 and the vibration sensor to Pin #7.
Final Wiring

The last wiring we need to do is the ground and voltage wires from the Perma-Proto PCB. The black wire goes into the G labeled pin (it's ground) and the red wire goes to the USB pin (this is voltage from USB power like a battery or USB hub.). Once everything is wired, you'll want to double check your work, again making sure the connections are solid and in the right pins. The final circuit looks a bit cluttered so we can orient the wires so we can handle the board. This will make it more manageable once we fit everything into the 3D printed case.
Case Assembly

Install Case

Orient the ItsyBitsy PCB so the microUSB connector is facing the small opening on the side of the 3D printed case. Place the board over the case and press it down into it into the case. Three walls should clip onto the edges of the PCB. Depending on how tight the tolerances are, you may need to firmly press down on the PCB.

The push button can be fitted into the square post with the leads on the sides. It should have a snug fit and hold in place. Orient the wires so they're routed along the outside of the post. Be careful not to bend the leads.

You'll want to position the wiring along the inside of the case so they're neatly routed. The Perma-Proto PCB should have enough room to fit in between the ItsyBitsy board
and side walls of the case. The two JST plugs and vibration sensor wiring should be fitted over the opening.

Install Button and Cover

Place the 3D printed button actuator over the rubber piece of the switch. Orient the cover so the button cutout is lined up with button. Then, bring the two parts together and firmly press them together to snap it shut. You should get a nice click. If you can't get the cover to fully close, you may need to adjust the PCB or wiring.
Test NeoPixel Strips
Now we can test out the NeoPixel strips. You'll want to double check the wires are soldered to the right connections before powering on the circuit. Pin #10 ought to be wired to the DATA INPUT, not the output, as indicated on the strip itself with a little arrow icon pointing away from the pads. If everything checks out, plug in a USB battery to the ItsyBitsy board. You can shake the vibration sensor to trigger the NeoPixel animations. Use the button to cycle between 3 different modes.

Trampoline Setup

Remove Padding
The trampoline has a protective padding that covers the framing and elastic bands. We'll need to remove it in order to attach the clips and hang the NeoPixel strips. Start by stretching the elastic lining and uncover the padding. You'll notice holes in various spots that accommodate for the threaded posts. These are for attaching the legs to the framing.

Installing NeoPixel Strips
Before we install the strips, we need to plan where the connections are going to be. Ideally, we want them to be closed to one of the legs of the trampoline because that's where we're going to attach the enclosure. We have two LED strips to install, so we'll need each to go in the opposite direction.
Threading NeoPixel Strips

The NeoPixel strips come with a protective silicone sheathing that is grippy. This introduces friction when the strips rub against the framing. Start by threading the end of the strip through the elastic strap and slowly inch it along the framing. Using both hands, guide both ends of the strip through the elastic straps until the whole strip is covering the framing. Repeat the process for the second strip, but going in the opposite direction. Once both strips are in place, attach the clips across the framing and hang the strip making sure not to cover up the LEDs.
Attaching the Vibration Sensor

Finding the best location for the vibration sensor requires a bit of play testing. For me, I mounted the sensor inside one of the elastic bands. This way it'll easily get triggered whenever the bed is slightly vibrated. I found the sensor to come out after a handful of jumps so I added some mounting tack to hold it in place. I could have also used zip ties, gaffers tape, hot glue or even sewed it onto the strap. I originally embedded the sensor in the enclosure but found it difficult to trigger. It might also work mounted to the framing or one of the legs. Experiment with different spots and methods of securing it to the trampoline.

Clip Case to Trampoline

The clip attached to the leg closest to the LED strip connectors. I oriented the case so that the microUSB is facing the bottom – This way the LED wiring isn't twisting or bending. The clip should have a tight fit and grasp the leg snugly.