# Table of Contents

## Overview
- Parts 4

## Build the Blaster Circuit
- NeoPixels 9
- And More NeoPixels 10
- Connect the Strands 11
- Quad Board 12
- Add Amps 13
- Connect Speakers & Switches 15
- NeoPixels, USB, & Battery Power 17

## CircuitPython on Feather M4 Express
- Set up CircuitPython Quick Start! 19
- Further Information 21

## CircuitPython Libraries
- The Adafruit CircuitPython Library Bundle 22
- Downloading the Adafruit CircuitPython Library Bundle 23
- The CircuitPython Community Library Bundle 23
- Downloading the CircuitPython Community Library Bundle 24
- Understanding the Bundle 24
- Example Files 25
- Copying Libraries to Your Board 25
- Understanding Which Libraries to Install 26
- Example: ImportError Due to Missing Library 28
- Library Install on Non-Express Boards 29
- Updating CircuitPython Libraries and Examples 30

## Code the Blaster
- Libraries 30
- Text Editor 30
- Audio Samples 31
- Make Your Own 31
- Code.py 31
- What It Does 34

## Original Lucio Blaster Construction 35
Overview

It's time for a modernized prop blaster! This guide is an overhaul of the Lucio Blaster project from 2017 (The original was a three part guide [here](https://adafru.it/t9B), [here](https://adafru.it/tFw), and [here](https://adafru.it/MaK).) You can also use this as a leaping off point for any prop or cosplay element that requires lights, music, and triggered sound effects.

A lot has changed in three years, and the project has been greatly simplified, thanks to advances in hardware and software! We'll now use the Feather M4 Express running CircuitPython and a Prop-Maker FeatherWing.

Thanks to dual-stream, native .mp3 playback in CircuitPython, we can eliminate the need for multiple pieces of hardware previously required to play music and sound effects at the same time, and the power, amplification, and NeoPixel driving circuits have been similarly simplified.

And speaking of NeoPixels -- the LED animation library makes it astonishingly simpler to create highly awesome lighting effects!

This guide shows how to connect all the electronics you need to make your own Sonic Amplifier, or modify them to any prop need.

A future update will show all of the guts fit into a new 808-themed 3D printed blaster! Download the [model files here](https://adafru.it/MaL) to make your own.
Here's some demo video of the original version to get a feel for the project:

Parts

**Adafruit Feather M4 Express - Featuring ATSAMD51**
It's what you've been waiting for, the Feather M4 Express featuring ATSAMD51. This Feather is fast like a swift, smart like an owl, strong like a ox-bird (it's half ox,...
https://www.adafruit.com/product/3857

**Adafruit Prop-Maker FeatherWing**
The Adafruit Feather series gives you lots of options for a small, portable, rechargeable microcontroller board. Perfect for fitting into your next prop build! This FeatherWing will...
https://www.adafruit.com/product/3988
Adafruit NeoPixel Digital RGBW LED Strip - Black PCB 144 LED/m
What is better than smart RGB LEDs? Smart RGB+White LEDs! These NeoPixels now have 4 LEDs in them (red, green, blue and white) for excellent lighting effects. These LED...
https://www.adafruit.com/product/2848

Adafruit Mono 2.5W Class D Audio Amplifier - PAM8302
This super small mono amplifier is surprisingly powerful - able to deliver up to 2.5 Watts into 4-8 ohm impedance speakers. Inside the miniature chip is a class D controller, able to...
https://www.adafruit.com/product/2130

Adafruit Quad Side-By-Side FeatherWing Kit with Headers
This is the FeatherWing Quad Side-by-Side - a prototyping add-on and more for all Feather boards. This is similar to our
https://www.adafruit.com/product/4254

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Component</th>
<th>Description</th>
<th>Product URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 x</td>
<td>NeoPixel 1/4 60 Ring</td>
<td>5050 RGBW LED w/ Integrated Drivers</td>
<td><a href="https://www.adafruit.com/product/2875">https://www.adafruit.com/product/2875</a></td>
</tr>
<tr>
<td>3 x</td>
<td>NeoPixel Ring - 16</td>
<td>5050 RGBW LEDs w/ Integrated Drivers</td>
<td><a href="https://www.adafruit.com/product/2856">https://www.adafruit.com/product/2856</a></td>
</tr>
<tr>
<td>1 x</td>
<td>NeoPixel Jewel</td>
<td>7 x 5050 RGBW LED w/ Integrated Drivers</td>
<td><a href="https://www.adafruit.com/product/2860">https://www.adafruit.com/product/2860</a></td>
</tr>
<tr>
<td>Quantity</td>
<td>Item Description</td>
<td>Link</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
<td>-------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>1 x</td>
<td>Colorful Round Tactile Button Switch Assortment</td>
<td><a href="https://www.adafruit.com/product/1009">https://www.adafruit.com/product/1009</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15 pack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 x</td>
<td>Mini Panel Mount SPDT Toggle Switch</td>
<td><a href="https://www.adafruit.com/product/3221">https://www.adafruit.com/product/3221</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ON/OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 x</td>
<td>Terminal Block kit for Feather</td>
<td><a href="https://www.adafruit.com/product/3173">https://www.adafruit.com/product/3173</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.1&quot; Pitch</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.7V 6600mAh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 x</td>
<td>Thin Plastic Speaker w/Wires</td>
<td><a href="https://www.adafruit.com/product/1891">https://www.adafruit.com/product/1891</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 ohm 0.25W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 x</td>
<td>JST PH 3-Pin to Male Header Cable</td>
<td><a href="https://www.adafruit.com/product/4046">https://www.adafruit.com/product/4046</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>200mm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 x</td>
<td>JST PH 3-Pin to Socket Cable</td>
<td><a href="https://www.adafruit.com/product/4046">https://www.adafruit.com/product/4046</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>200mm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 x</td>
<td>Molex Pico Blade 2-pin Cable</td>
<td><a href="https://www.adafruit.com/product/3922">https://www.adafruit.com/product/3922</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>200mm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Silicone Cover Stranded-Core Wire - 26AWG in Various Colors**

Silicone-sheathing wire is super-flexible and soft, and it's also strong! Able to handle up to 200°C and up to 600V, it will do when PVC covered wire wimps out. We like this...

Build the Blaster Circuit

We'll build out the circuit starting with the NeoPixels. These will plug into the JST port on the Propmaker FeatherWing which can drive them with more power than a bare microcontroller on its own.

A pair of toggle switches will connect the Propmaker for powering the project on and off via the Enable pin, and for switching song/color modes.

To drive the four speakers we'll connect one to the amp built into the Propmaker and then add three 2.5W class D amps for the remaining speakers.

We'll plug the Feather and Propmaker into a quad FeatherWing kit which gives us plenty of prototyping area to connect the amps and the trigger buttons. Adding Feather terminal headers make it easy to connect the buttons.

Here's a Fritzing diagram of the complete circuit:
NeoPixels
We'll create a single circuit of all the NeoPixel strands, rings, and jewel so that they share a single data line, ground, and power.

First, solder together the four quarter segments into a single ring. When doing so, be sure to solder all of the power and ground pads, and three of the four Data In/Data Out pairs.

On the un-soldered Data In segment, solder a JST wiring interconnect cable so that the ring can be connected to the rest of the NeoPixels -- this will be the last set of NeoPixels so it doesn't need an out cable.
And More NeoPixels

Solder a pair of in/out interconnect cables to each ring, jewel, and strand so they may all be joined.

It's important to be consistent with polarity of interconnects so all of the Data In sides can only be connected with Data Out sides.
Connect the Strands

I decided to wire the two VU meter strips directly to each other, and use the interconnects at either end for connecting to the Propmaker FeatherWing and the rest of the circuit.

Plug them all together and they’re ready for use!

You may choose to wire each segment directly to the next, which will work just fine. Interconnects are a bit of extra effort, but give you added freedom during integration with your prop blaster parts.
Quad Board
Solder female headers into the top two positions of the Quad board.

Solder male headers to the underside of the Feather and the Propmaker so they can be attached to the Quad.

Then, solder the terminal blocks to the bottom quad area.
Add Amps

Use three sections of male header pin rows to solder the three amps to the prototyping area as shown, pushing the long ends of the pins down into the board so they can be used as posts for soldering the wires.

We'll wire the circuit connections as they're shown in the Frizing diagram, except we'll do it underneath the board.

Solder a set of wires from the Quad board GND to the GND posts of the two bottom amps as shown in the first wiring picture here.

This is followed by running a circuit from the +V rail to the appropriate posts.

Then, run the A0 pin to the A+ (audio in) pin on both bottom amps and the A1 pin to the A+ of the top amp.
Connect Speakers & Switches

Connect two of the small speakers and the large front speaker to their respective amps as shown using the terminal blocks.

The third small speaker will connect to the speaker output on the Propmaker. Do this by grafting in a Molex "Pico-Blade" connector (https://adafru.it/CVi).

You'll use the Feather terminal headers to connect the trigger buttons to ground, A5 and A6.

The toggle switches are connected to the En & GND pins on the Propmaker wing and the SW & GND pins, also on the Propmaker. You can solder these directly or add a couple of 0.1" terminal blocks first.
Neopixels, USB, & Battery Power

You can now plug in the LiPoly battery as well as the USB panel mount to extend the Feather's USB port to make it accessible from the exterior of the prop. The USB will be used both for programming the Feather as well as charging the battery.

Plug the NeoPixel mega strand into the JST socket on the Propmaker FeatherWing and the blaster is ready to go!
CircuitPython on Feather M4 Express

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It
makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the CIRCUITPY drive to iterate.

The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

**Set up CircuitPython Quick Start!**

Follow this quick step-by-step for super-fast Python power :)

Download the latest version of CircuitPython for this board via CircuitPython.org

https://adafruit.it/Emh

Click the link above and download the latest UF2 file.

Download and save it to your desktop (or wherever is handy).
Plug your Feather M4 into your computer using a known-good USB cable.

A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

Double-click the Reset button next to the USB connector on your board, and you will see the NeoPixel RGB LED turn green. If it turns red, check the USB cable, try another USB port, etc. Note: The little red LED next to the USB connector will pulse red. That's ok!

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

You will see a new disk drive appear called FEATHERBOOT.

Drag the adafruit_circuitpython_etc.uf2 file to FEATHERBOOT.
The LED will flash. Then, the FEATHERBOOT drive will disappear and a new disk drive called CIRCUITPY will appear.

That's it, you're done! :)

Further Information

For more detailed info on installing CircuitPython, check out Installing CircuitPython (https://adafruit.it/Amd).

CircuitPython Libraries

As CircuitPython development continues and there are new releases, Adafruit will stop supporting older releases. Visit https://circuitpython.org/downloads to download the latest version of CircuitPython for your board. You must download the CircuitPython Library Bundle that matches your version of CircuitPython. Please update CircuitPython and then visit https://circuitpython.org/libraries to download the latest Library Bundle.

Each CircuitPython program you run needs to have a lot of information to work. The reason CircuitPython is so simple to use is that most of that information is stored in other files and works in the background. These files are called libraries. Some of them are built into CircuitPython. Others are stored on your CIRCUITPY drive in a folder called lib. Part of what makes CircuitPython so great is its ability to store code separately from the firmware itself. Storing code separately from the firmware makes it easier to update both the code you write and the libraries you depend.

Your board may ship with a lib folder already, it's in the base directory of the drive. If not, simply create the folder yourself. When you first install CircuitPython, an empty lib directory will be created for you.
CircuitPython libraries work in the same way as regular Python modules so the Python docs (https://adafruit.io/rar) are an excellent reference for how it all should work. In Python terms, you can place our library files in the lib directory because it’s part of the Python path by default.

One downside of this approach of separate libraries is that they are not built in. To use them, one needs to copy them to the CIRCUITPY drive before they can be used. Fortunately, there is a library bundle.

The bundle and the library releases on GitHub also feature optimized versions of the libraries with the .mpy file extension. These files take less space on the drive and have a smaller memory footprint as they are loaded.

Due to the regular updates and space constraints, Adafruit does not ship boards with the entire bundle. Therefore, you will need to load the libraries you need when you begin working with your board. You can find example code in the guides for your board that depends on external libraries.

Either way, as you start to explore CircuitPython, you'll want to know how to get libraries on board.

### The Adafruit CircuitPython Library Bundle

Adafruit provides CircuitPython libraries for much of the hardware they provide, including sensors, breakouts and more. To eliminate the need for searching for each library individually, the libraries are available together in the Adafruit CircuitPython Library Bundle. The bundle contains all the files needed to use each library.
Downloading the Adafruit CircuitPython Library Bundle

You can download the latest Adafruit CircuitPython Library Bundle release by clicking the button below. The libraries are being constantly updated and improved, so you'll always want to download the latest bundle.

Match up the bundle version with the version of CircuitPython you are running. For example, you would download the 6.x library bundle if you're running any version of CircuitPython 6, or the 7.x library bundle if you're running any version of CircuitPython 7, etc. If you mix libraries with major CircuitPython versions, you will get incompatible mpy errors due to changes in library interfaces possible during major version changes.

Click to visit circuitpython.org for the latest Adafruit CircuitPython Library Bundle
https://adafruit.it/ENC

Download the bundle version that matches your CircuitPython firmware version. If you don't know the version, check the version info in boot_out.txt file on the CIRCUITPY drive, or the initial prompt in the CircuitPython REPL. For example, if you're running v7.0.0, download the 7.x library bundle.

There's also a py bundle which contains the uncompressed python files, you probably don't want that unless you are doing advanced work on libraries.

The CircuitPython Community Library Bundle

The CircuitPython Community Library Bundle is made up of libraries written and provided by members of the CircuitPython community. These libraries are often written when community members encountered hardware not supported in the Adafruit Bundle, or to support a personal project. The authors all chose to submit these libraries to the Community Bundle make them available to the community.

These libraries are maintained by their authors and are not supported by Adafruit. As you would with any library, if you run into problems, feel free to file an issue on the GitHub repo for the library. Bear in mind, though, that most of these libraries are supported by a single person and you should be patient about receiving a response. Remember, these folks are not paid by Adafruit, and are volunteering their personal time when possible to provide support.
Downloading the CircuitPython Community Library Bundle

You can download the latest CircuitPython Community Library Bundle release by clicking the button below. The libraries are being constantly updated and improved, so you'll always want to download the latest bundle.

[Click for the latest CircuitPython Community Library Bundle release](https://adafru.it/VCn)

The link takes you to the latest release of the CircuitPython Community Library Bundle on GitHub. There are multiple versions of the bundle available. Download the bundle version that matches your CircuitPython firmware version. If you don't know the version, check the version info in boot_out.txt file on the CIRCUITPY drive, or the initial prompt in the CircuitPython REPL. For example, if you're running v7.0.0, download the 7.x library bundle.

Understanding the Bundle

After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX, it places the file in the same directory as the zip.

Open the bundle folder. Inside you'll find two information files, and two folders. One folder is the lib bundle, and the other folder is the examples bundle.

Now open the lib folder. When you open the folder, you'll see a large number of .mpy files, and folders.
Example Files

All example files from each library are now included in the bundles in an examples directory (as seen above), as well as an examples-only bundle. These are included for two main reasons:

- Allow for quick testing of devices.
- Provide an example base of code, that is easily built upon for individualized purposes.

Copying Libraries to Your Board

First open the lib folder on your CIRCUITPY drive. Then, open the lib folder you extracted from the downloaded zip. Inside you'll find a number of folders and .mpy files. Find the library you'd like to use, and copy it to the lib folder on CIRCUITPY.

If the library is a directory with multiple .mpy files in it, be sure to copy the entire folder to CIRCUITPY/lib.

This also applies to example files. Open the examples folder you extracted from the downloaded zip, and copy the applicable file to your CIRCUITPY drive. Then, rename it to code.py to run it.

If a library has multiple .mpy files contained in a folder, be sure to copy the entire folder to CIRCUITPY/lib.
Understanding Which Libraries to Install

You now know how to load libraries on to your CircuitPython-compatible microcontroller board. You may now be wondering, how do you know which libraries you need to install? Unfortunately, it's not always straightforward. Fortunately, there is an obvious place to start, and a relatively simple way to figure out the rest. First up: the best place to start.

When you look at most CircuitPython examples, you'll see they begin with one or more `import` statements. These typically look like the following:

```
• import library_or_module
```

However, `import` statements can also sometimes look like the following:

```
• from library_or_module import name
• from library_or_module.subpackage import name
• from library_or_module import name as local_name
```

They can also have more complicated formats, such as including a `try / except` block, etc.

The important thing to know is that an `import` statement will always include the name of the module or library that you're importing.

Therefore, the best place to start is by reading through the `import` statements.

Here is an example import list for you to work with in this section. There is no setup or other code shown here, as the purpose of this section involves only the import list.

```
import time
import board
import neopixel
import adafruit_lis3dh
import usb_hid
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode
```

Keep in mind, not all imported items are libraries. Some of them are almost always built-in CircuitPython modules. How do you know the difference? Time to visit the REPL.
In the Interacting with the REPL section on The REPL page in this guide, the `help("modules")` command is discussed. This command provides a list of all of the built-in modules available in CircuitPython for your board. So, if you connect to the serial console on your board, and enter the REPL, you can run `help("modules")` to see what modules are available for your board. Then, as you read through the `import` statements, you can, for the purposes of figuring out which libraries to load, ignore the statement that import modules.

The following is the list of modules built into CircuitPython for the Feather RP2040. Your list may look similar or be anything down to a significant subset of this list for smaller boards.

```
>>> help("modules")
__main__ board micropython storage
_bleio builtin msgpack struct
adafruit_bus_device collections busio neopixel_write supervisor
adafruit_pixelbuf countio cpio os
analogio digitalio displayio nokia5110 sys
array errc ntp runio
atexit fontio qrio traceback
audiobusio framebufferio rainbowio ubl
audiocore gc random usb_cdc
audioceiver getpass re usb_hid
audiomp3 imagencapt imgpio rgbmatrix ush_midi
audiooplib io json rtc
binascii json keypad rtc
bitbangio keypad rtc
bitmaptools math sdcardio
https microcontroller sharpdisplay
```

Now that you know what you’re looking for, it’s time to read through the import statements. The first two, `time` and `board`, are on the modules list above, so they’re built-in.

The next one, `neopixel`, is not on the module list. That means it’s your first library! So, you would head over to the bundle zip you downloaded, and search for neopixel. There is a neopixel.mpy file in the bundle zip. Copy it over to the lib folder on your CI RCUITPY drive. The following one, `adafruit_lis3dh`, is also not on the module list. Follow the same process for adafruit_lis3dh, where you'll find adafruit_lis3dh.mpy, and copy that over.

The fifth one is `usb_hid`, and it is in the modules list, so it is built in. Often all of the built-in modules come first in the import list, but sometimes they don’t! Don't assume that everything after the first library is also a library, and verify each import with the modules list to be sure. Otherwise, you'll search the bundle and come up empty!

The final two imports are not as clear. Remember, when `import` statements are formatted like this, the first thing after the `from` is the library name. In this case, the library name is `adafruit_hid`. A search of the bundle will find an adafruit_hid folder.
When a library is a folder, you must copy the entire folder and its contents as it is in the bundle to the lib folder on your CIRCUITPY drive. In this case, you would copy the entire adafruit_hid folder to your CIRCUITPY/lib folder.

Notice that there are two imports that begin with \texttt{adafruit\_hid}. Sometimes you will need to import more than one thing from the same library. Regardless of how many times you import the same library, you only need to load the library by copying over the adafruit_hid folder once.

That is how you can use your example code to figure out what libraries to load on your CircuitPython-compatible board!

There are cases, however, where libraries require other libraries internally. The internally required library is called a dependency. In the event of library dependencies, the easiest way to figure out what other libraries are required is to connect to the serial console and follow along with the \texttt{ImportError} printed there. The following is a very simple example of an \texttt{ImportError}, but the concept is the same for any missing library.

**Example: ImportError Due to Missing Library**

If you choose to load libraries as you need them, or you're starting fresh with an existing example, you may end up with code that tries to use a library you haven't yet loaded. This section will demonstrate what happens when you try to utilise a library that you don't have loaded on your board, and cover the steps required to resolve the issue.

This demonstration will only return an error if you do not have the required library loaded into the lib folder on your CIRCUITPY drive.

Let's use a modified version of the Blink example.

```python
import board
import time
import simpleio

led = simpleio.DigitalOut(board.LED)

while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```
Save this file. Nothing happens to your board. Let's check the serial console to see what's going on.

![Serial Console Output]

You have an `ImportError`. It says there is no module named 'simpleio'. That's the one you just included in your code!

Click the link above to download the correct bundle. Extract the lib folder from the downloaded bundle file. Scroll down to find simpleio.mpy. This is the library file you're looking for! Follow the steps above to load an individual library file.

The LED starts blinking again! Let's check the serial console.

![Serial Console Output]

No errors! Excellent. You've successfully resolved an `ImportError`!

If you run into this error in the future, follow along with the steps above and choose the library that matches the one you're missing.

**Library Install on Non-Express Boards**

If you have an M0 non-Express board such as Trinket M0, Gemma M0, QT Py M0, or one of the M0 Trinkeys, you'll want to follow the same steps in the example above to install libraries as you need them. Remember, you don't need to wait for an `ImportError` if you know what library you added to your code. Open the library bundle you downloaded, find the library you need, and drag it to the lib folder on your CIRCUITPY drive.

You can still end up running out of space on your M0 non-Express board even if you only load libraries as you need them. There are a number of steps you can use to try
to resolve this issue. You'll find suggestions on the [Troubleshooting page](https://adafru.it/Den).

**Updating CircuitPython Libraries and Examples**

Libraries and examples are updated from time to time, and it's important to update the files you have on your CIRCUITPY drive.

To update a single library or example, follow the same steps above. When you drag the library file to your lib folder, it will ask if you want to replace it. Say yes. That's it!

A new library bundle is released every time there's an update to a library. Updates include things like bug fixes and new features. It's important to check in every so often to see if the libraries you're using have been updated.

---

**Code the Blaster**

Libraries

Once your Feather M4 Express is set up with CircuitPython 6.0.0-alpha.1 or greater, you'll also need to add some libraries. Follow this page [here](https://adafru.it/ABU) for information on how to download and add libraries to your Feather.

From the library bundle you downloaded in that guide page, transfer the following libraries onto the Feather's /lib directory:

- adafruit_bus_device
- adafruitLedAnimation
- adafruit_lis3dh_animation
- neopixel.mpy

---

**Text Editor**

Adafruit recommends using the Mu editor for using your CircuitPython code with the Feather boards. You can get more info in [this guide](https://adafru.it/ANO).

Alternatively, you can use any text editor that saves files.
Audio Samples

You can get started by using the Lucio music and sound effects .mp3 files. (Of course, you're free to use any files that you like, see below for more info on that.)

Download the .zip file linked below and then uncompress the file. Drag the /lucio directory onto the root level of your Feather's CIRCUITPY drive.

[ lucio.zip ]
https://adafru.it/MaI

Make Your Own

To make your own .mp3, use audio software such as Audacity (https://adafru.it/Lee) to save them with these settings:

- bit rate: anywhere from 16 to 320 kb/s (lower will be smaller, higher is better quality)
- sample rate: 22050Hz or 44100Hz are recommended, although 16kHz, 24kHz, and 48kHz should also work
- channels: mono or stereo
- must be DRM free

Code.py

Copy the code below and paste it into Mu. Then, save it to your Feather as code.py.

```python
# Lucio 2020
# Feather M4 + Propmaker + amps + lots of neopixels
import board
import busio
from digitalio import DigitalInOut, Direction, Pull
import audioio
import audiomixer
import audiomp3
import adafruit_lis3dh
import neopixel
from adafruit_led_animation.animation.solid import Solid
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.pulse import Pulse
from adafruit_led_animation.helper import PixelSubset
from adafruit_led_animation.group import AnimationGroup
from adafruit_led_animation.color import RED, ORANGE, WHITE

ORANGE_DIM = 0x801400  # half value version
RED_DIM = 0x800000
```

©Adafruit Industries
# ---Set Volume Max Here---
VOLUME_MULT = 0.65  # 1 = full volume, 0.1 is very quiet, 0 is muted

# ---SWITCH/BUTTON SETUP---
mode_switch = DigitalInOut(board.D9)
mode_switch.switch_to_input(pull=Pull.UP)
mode_state = mode_switch.value
trig_button = DigitalInOut(board.A4)
trig_button.switch_to_input(pull=Pull.UP)
alt_button = DigitalInOut(board.A5)
alt_button.switch_to_input(pull=Pull.UP)

# ---ACCELEROMETER SETUP---
# Set up accelerometer on I2C bus, 4G range:
i2c = busio.I2C(board.SCL, board.SDA)
int1 = DigitalInOut(board.D6)
accel = adafruit_lis3dh.LIS3DH_I2C(i2c, int1)

# ---SPEAKER SETUP---
enable = DigitalInOut(board.D10)
enable.direction = Direction.OUTPUT
enable.value = True
# Set up speakers and mixer. Stereo files, where music has empty right channel, FX
# empty left
speaker = audioio.AudioOut(board.A0, right_channel=board.A1)
mixer = audiomixer.Mixer(channel_count=2, buffer_size=2304, sample_rate=22050)

# ---NEOPIXEL SETUP---
pixel_pin = board.D5
pixel_num = 154
pixels = neopixel.NeoPixel(
    pixel_pin, pixel_num, brightness=0.6, auto_write=False,
    pixel_order=neopixel.GRBW
)
# ^ change pixel_order depending on RGB vs. RGBW pixels

# ---Pixel Map---
# this is the physical order in which the strips are plugged
pixel_stripA = PixelSubset(pixels, 0, 18)  # 18 pixel strip
pixel_stripB = PixelSubset(pixels, 18, 36)  # 18 pixel strip
pixel_jewel = PixelSubset(pixels, 36, 43)  # 7 pixel jewel
pixel_ringsAll = PixelSubset(pixels, 43, 151)  # all of the rings
# or use rings individually:
# pixel_ringA = PixelSubset(pixels, 43, 59)  # 16 pixel ring
# pixel_ringB = PixelSubset(pixels, 59, 75)  # 16 pixel ring
# pixel_ringC = PixelSubset(pixels, 75, 91)  # 16 pixel ring
# pixel_ringD = PixelSubset(pixels, 91, 151)  # 60 pixel ring

# ---BPM---
BPM = 128
BEAT = 60 / BPM  # quarter note beat
b16TH = BEAT / 4  # 16TH note
b64TH = BEAT / 16  # sixty-fourth

# ---Anim Setup---
# heal color mode
# Pulse 'speed' = smoothness
pulse_rings_m0 = Pulse(pixel_ringsAll, speed=0.01, color=ORANGE, period=BEAT)
pulse_jewel_m0 = Pulse(pixel_jewel, speed=0.01, color=ORANGE, period=BEAT)
comet_stripA_m0 = Comet(
    pixel_stripA, speed=b64TH, color=ORANGE, tail_length=9, bounce=False
)
comet_stripB_m0 = Comet(
    pixel_stripB, speed=b64TH, color=ORANGE, tail_length=9, bounce=False
)
# speed color mode
pulse_rings_m1 = Pulse(pixel_ringsAll, speed=0.02, color=RED, period=BEAT / 2)
pulse_jewel_m1 = Pulse(pixel_jewel, speed=0.02, color=RED, period=BEAT / 2)
comet_stripA_m1 = Comet(  
    pixel_stripA, speed=b64TH, color=RED, tail_length=9, bounce=False
)
comet_stripB_m1 = Comet(  
    pixel_stripB, speed=b64TH, color=RED, tail_length=9, bounce=False
)
solid_white = Solid(pixel_ringsAll, color=WHITE)

# ---Anim Modes---
uv_strip_animations_mode0 = AnimationGroup(comet_stripA_m0, comet_stripB_m0, sync=True)
uv_strip_animations_mode1 = AnimationGroup(comet_stripA_m1, comet_stripB_m1, sync=True)

# ---Audio Setup---
if mode_state:
    BGM = "/lucio/bgmheal.mp3"
else:
    BGM = "/lucio/bgmspeed.mp3"
sample0 = audiomp3.MP3Decoder(open(BGM, "rb"))
FX = "/lucio/shoot.mp3"
sample1 = audiomp3.MP3Decoder(open(FX, "rb"))
speaker.play(mixer)
mixer.voice[0].play(sample0, loop=True)
mixer.voice[0].level = 0.3 * VOLUME_MULT
mixer.voice[1].level = 0.7 * VOLUME_MULT

while True:
    if mode_state:  # heal mode on startup
        vu_strip_animations_mode0.animate()
pulse_rings_m0.animate()
pulse_jewel_m0.animate()
    else:  # speed mode on startup
        vu_strip_animations_mode1animate()
pulse_rings_m1.animate()
pulse_jewel_m1animate()

    # Change modes
    if mode_switch.value:
        if mode_state == 0:  # state has changed, toggle it
            BGM = "/lucio/bgmheal.mp3"
sample0.file = open(BGM, "rb")
mixer.voice[0].play(sample0, loop=True)
vu_strip_animations_mode0.animate()
pulse_rings_m0.animate()
pulse_jewel_m0.animate()
        mode_state = 1
    else:
        if mode_state == 1:
            BGM = "/lucio/bgmspeed.mp3"
sample0.file = open(BGM, "rb")
mixer.voice[0].play(sample0, loop=True)
vu_strip_animations_mode1animate()
pulse_rings_m1.animate()
pulse_jewel_m1.animate()
        mode_state = 0

    x, _, _ = accel.acceleration  # get accelerometer values

    if not mixer.voice[1].playing:
        if not trig_button.value:  # trigger squeezed
            FX_sample = "/lucio/shoot.mp3"
sample1.file = open(FX_sample, "rb")
mixer.voice[1].play(sample1)
if mode_state:
solid_white.animate()

else:
    solid_white.animate()

if not alt_button.value:
    # alt trigger squeezed
    FX_sample = "/lucio/alt_shoot.mp3"
    sample1_file = open(FX_sample, "rb")
    mixer.voice[1].play(sample1)
    if mode_state:
        solid_white.animate()
    else:
        solid_white.animate()

if accel.acceleration.x > 8:
    # reload
    FX_sample = "/lucio/reload.mp3"
    sample1_file = open(FX_sample, "rb")
    mixer.voice[1].play(sample1)
    if mode_state:
        solid_white.animate()
    else:
        solid_white.animate()

if accel.acceleration.x < -8:
    # Ultimate
    FX_sample = "/lucio/ultimate.mp3"
    sample1_file = open(FX_sample, "rb")
    mixer.voice[1].play(sample1)
    if mode_state:
        solid_white.animate()
    else:
        solid_white.animate()

What It Does

These are the main features of the code:

- Play one of two background music song files depending on the mode switch position
- Play NeoPixel LED animation to the beat, the base color changed depending on mode switch position
- Trigger button plays shooting sound effect file and flashes all NeoPixels white
- Alt trigger button plays alt shooting sound effect file and flashes all NeoPixels white
- Tilt "up" of the unit (Propmaker FeatherWing's accelerometer) plays reload sound effect
- Tilt "down" of the unit plays "ultimate" sound effect

Look through the code and you'll see places to tweak and modify the behavior. You can use the led_animation library's PixelSubset code to send entirely different animation and color to the different logical groupings of pixels, for example.
Original Lucio Blaster Construction

Original Lucio Blaster Construction (https://adafruit.it/MaK)