Cyber Cat MIDI Keyboard

Created by John Park

https://learn.adafruit.com/midi-cyber-cat-keyboard

Last updated on 2024-03-08 04:12:19 PM EST

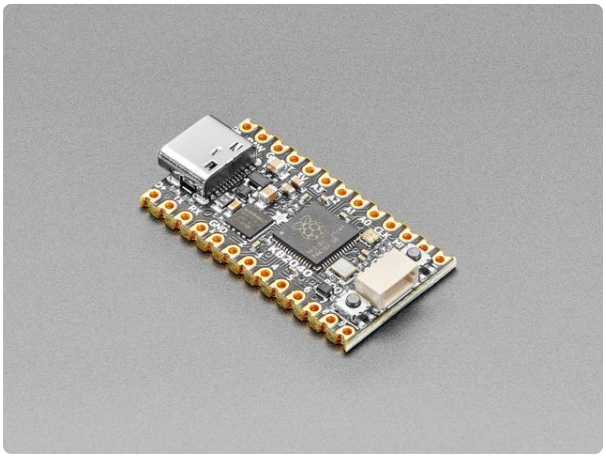
Table of Contents

Overview

You can transform a toy keyboard into a highly customized MIDI controller, perfect for laying down synthwave pads on your favorite hardware or software synthesizer.

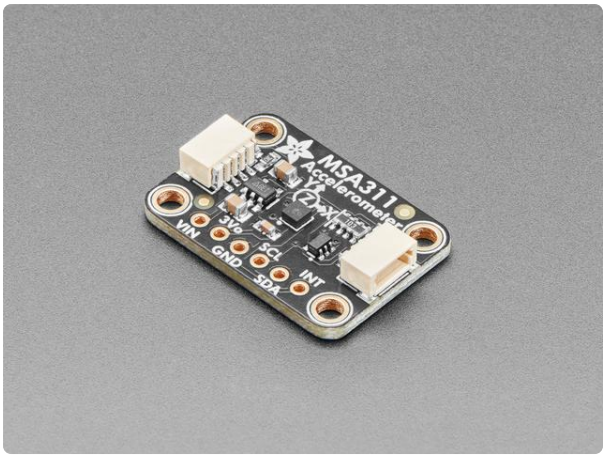
All it takes is an Adafruit Kee Boar KB2040 running CircuitPython, a Meowsic cat piano, and the desire to be awesome. Plus some soldering and a bit of paint.

Parts



Adafruit KB2040 - RP2040 Kee Boar Driver

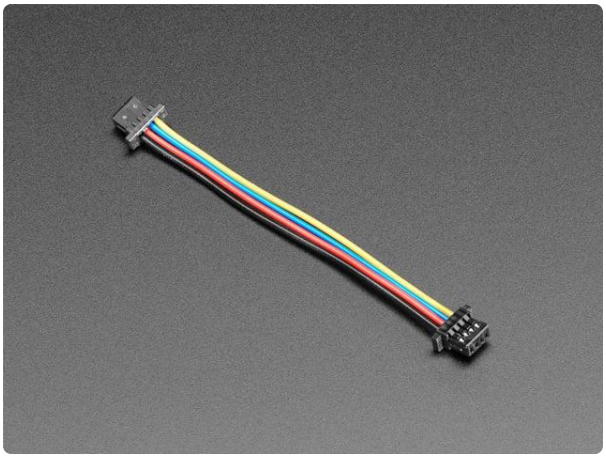
A wild Kee Boar appears! It's a shiny KB2040! An Arduino Pro Micro-shaped board for Keebs with RP2040. (#keeblife 4 evah) A lot of folks like using Adafruit... https://www.adafruit.com/product/5302



Adafruit MSA311 Triple Axis Accelerometer - STEMMA QT / Qwiic

The MSA311 is a super small and low-cost triple-axis accelerometer. It's inexpensive, but has just about every 'extra' you'd want in an...

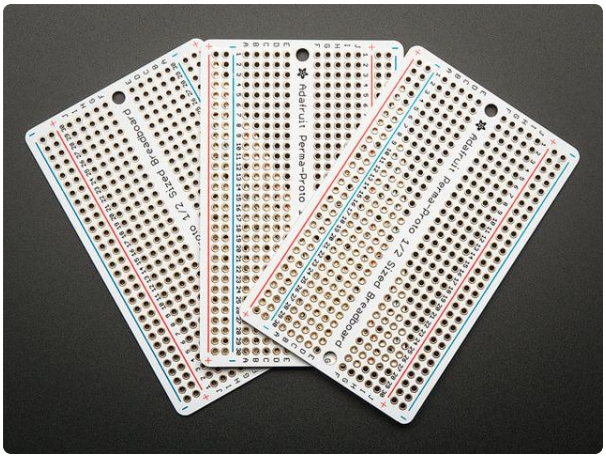
https://www.adafruit.com/product/5309



STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long

This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...

https://www.adafruit.com/product/4399



Adafruit Perma-Proto Half-sized Breadboard PCB - 3 Pack!

Customers have asked us to carry basic perf-board, but we never liked the look of most basic perf: its always crummy quality, with pads that flake off and no labeling. Then we thought...

https://www.adafruit.com/product/571

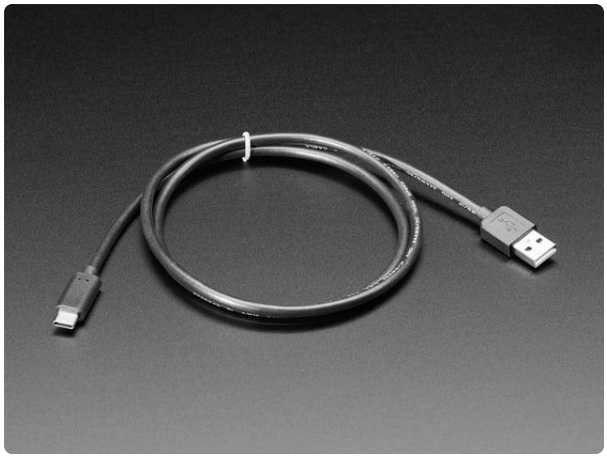
Panel Mount 1/8" / 3.5mm TRS Audio Jack Connector

What is this TRS bit? Turtle Rock Studios? Transmission Raman...

https://www.adafruit.com/product/3692

[3.5mm Stereo Male/Male Cables - Black Metal - 1 meter long]
Here is a gorgeous metal-covered audio cable straight from Blade Runner/ cyberpunk heaven. And we have them in a couple different colors! They're a step up from plain rubber/ABS...
https://www.adafruit.com/product/4069

[USB C Round Panel Mount Extension Cable]
If you need to add a panel-mount connection but don't have the time or ability to cut a custom oval or square hole, this USB C Round Panel Mount Extension...
https://www.adafruit.com/product/4218



[USB Type A to Type C Cable - approx 1 meter / 3 ft long]
As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...
https://www.adafruit.com/product/4474

[iOS Lightning to USB OTG Cable]
Your iOS phone or tablet may not have a USB port on the bottom but that doesn't mean you can't use it to connect USB devices. Secretly known as a 'Camera Connector' or...
https://www.adafruit.com/product/3940

Silicone Cover Stranded-Core Ribbon Cable - 10 Wire 1 Meter Long

For those who are fans of our silicone-covered wires, but are always looking to up their wiring game. We now have Silicone Cover Ribbon cables! These may look...

https://www.adafruit.com/product/3890

Optional

You can get some spray primer, paint, and clear coat at your local hardware or art supply store if you want to paint the keyboard.

Synthesizer

You can use any hardware synth that accepts DIN-5 MIDI (using an adapter cable from TRS-A) or TRS MIDI (may require TRS-A to TRS-B cable or adapter). You can control software synths that can be a USB MIDI host, such as an iOS device or computer.

A great free synth to use on iOS is AudioKit Synth One (https://adafru.it/C-6).

CircuitPython

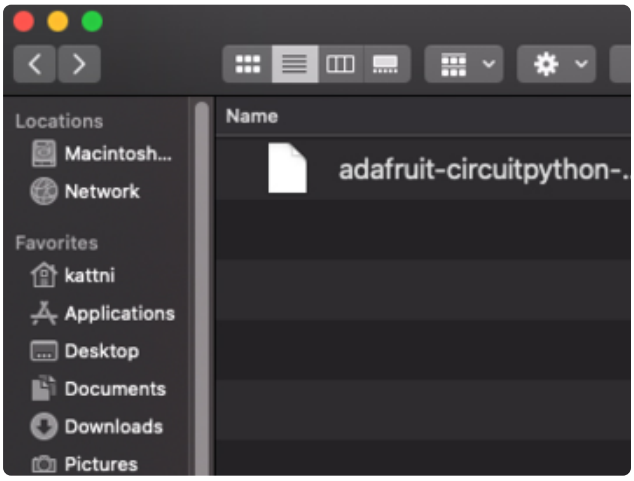
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.

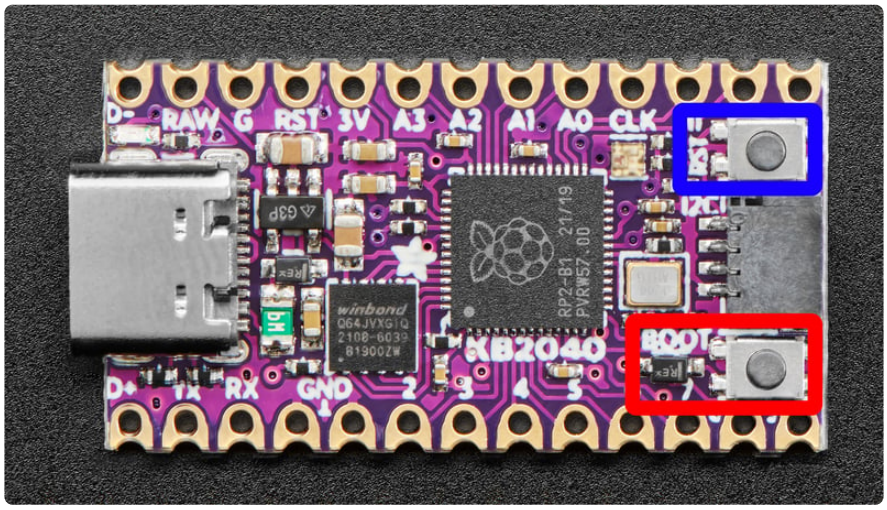
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/Xdr



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.

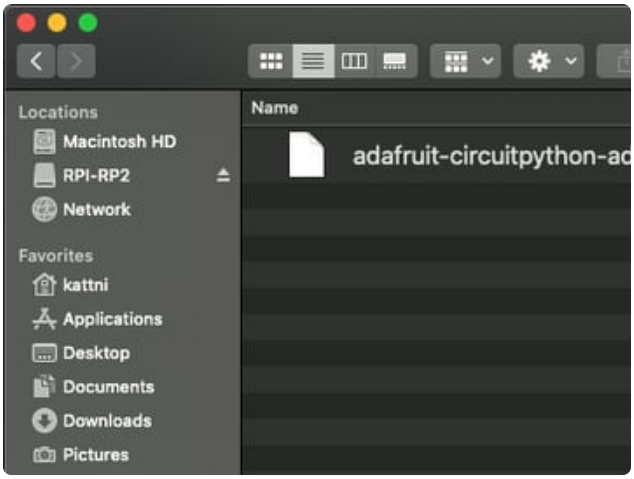


To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

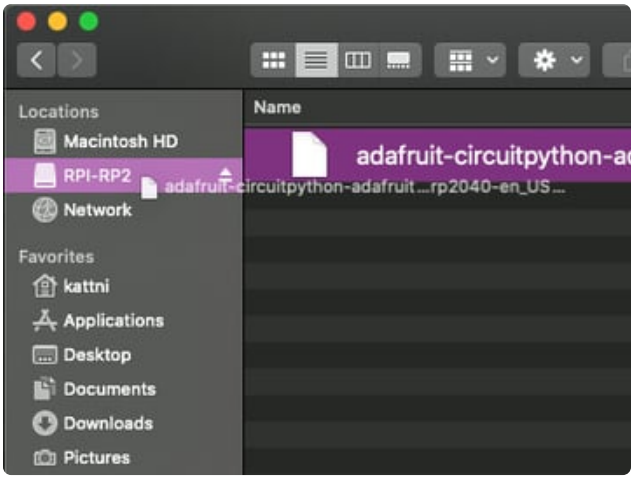
If the drive does not appear, release all the buttons, and then repeat the process above.

You can also start with your board unplugged from USB, press and hold the BOOTSEL button (highlighted in red above), continue to hold it while plugging it into USB, and wait for the drive to appear before releasing the button.

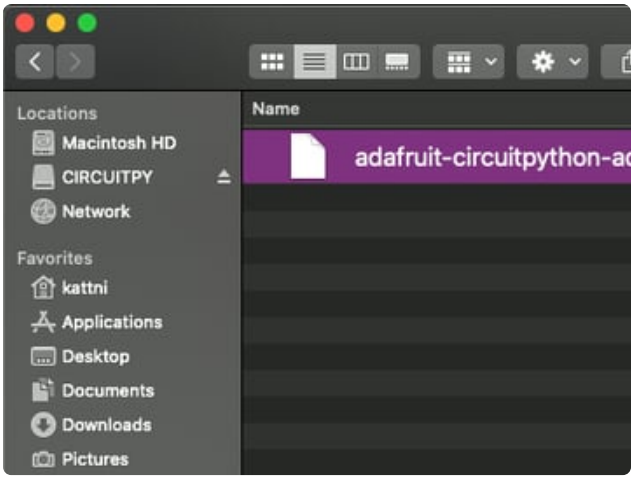
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **RPI-RP2**.



Drag the **adafruit_circuitpython_etc.uf2** file to **RPI-RP2.**



The **RPI-RP2** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

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

Safe Mode

You want to edit your **code.py** or modify the files on your **CIRCUITPY** drive, but find that you can't. Perhaps your board has gotten into a state where **CIRCUITPY** is read-only. You may have turned off the **CIRCUITPY** drive altogether. Whatever the reason, safe mode can help.

Safe mode in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode bypasses any code in **boot.py** (where you can set **CIRCUITPY** read-only or turn it off completely). Second, it does not run the code in **code.py**. And finally, it does not automatically soft-reload when data is written to the **CIRCUITPY** drive.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the **CIRCUITPY** drive.

Entering Safe Mode

To enter safe mode when using CircuitPython, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

If you successfully enter safe mode on CircuitPython, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

```
Auto-reload is off.
Running in safe mode! Not running saved code.

CircuitPython is in safe mode because you pressed the reset button during boot.
Press again to exit safe mode.

Press any key to enter the REPL. Use CTRL-D to reload.
```

You can now edit the contents of the **CIRCUITPY** drive. Remember, your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.

Flash Resetting UF2

If your board ever gets into a really weird state and doesn't even show up as a disk drive when installing CircuitPython, try loading this 'nuke' UF2 which will do a 'deep

clean' on your Flash Memory. **You will lose all the files on the board**, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

Download flash erasing "nuke" UF2

https://adafru.it/RLE

Code the Cyber Cat MIDI Keyboard

Text Editor

Adafruit recommends using the **Mu** editor for editing your CircuitPython code. You can get more info in this guide (https://adafru.it/ANO).

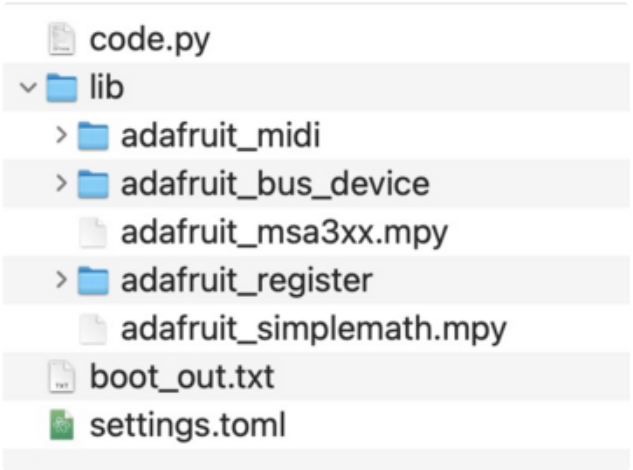
Alternatively, you can use any text editor that saves simple text files

Download the Project Bundle

Your project will use a specific set of CircuitPython libraries, and the **code.py** file. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Plug your Kee Boar board into your computer with a known good USB cable with both data and power wires. It should show up as a thumb drive in your File Explorer or Finder (depending on your operating system) named **CIRCUITPY**.

Drag the contents of the uncompressed bundle directory onto your Key Boar board **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.



Upload the Code and Libraries to the KB RP2040

You should have the following files on your CIRCUITPY drive:

lib folder
code.py

```
# SPDX-FileCopyrightText: 2023 John Park for Adafruit
#
# SPDX-License-Identifier: MIT
# Cyber Cat MIDI Keyboard conversion for Meowsic Cat Piano

#  Functions:
#    --28 keys
#    --left five toe buttons: patches
#    --right five toe buttons: picking CC number for ice cream cone control
#    --volume arrows: octave up/down
#    --tempo arrows: pitchbend up/down
#    --on switch: reset
#    --nose button: midi panic
#    --record button: ice cream cone CC enable/disable (led indicator)
#    --play button: start stop arp or sequence in soft synth via cc 16 0/127
#    --treble clef button: hold notes (use nose to turn off all notes)
#    --face button: momentary CC 0/127 on CC number 17

import keypad
import board
import busio
import supervisor
import digitalio
from adafruit_simplemath import map_range
from adafruit_msa3xx import MSA311
import usb_midi
import adafruit_midi
from adafruit_midi.note_on import NoteOn
from adafruit_midi.note_off import NoteOff
from adafruit_midi.control_change import ControlChange
from adafruit_midi.program_change import ProgramChange
from adafruit_midi.pitch_bend import PitchBend

supervisor.runtime.autoreload = True  # set False to prevent unwanted restarts due
to OS weirdness

ledpin = digitalio.DigitalInOut(board.A3)
ledpin.direction = digitalio.Direction.OUTPUT
ledpin.value = True

i2c = board.STEMMA_I2C()
msa = MSA311(i2c)

key_matrix = keypad.KeyMatrix(
    column_pins=(board.D2, board.D3, board.D4, board.D5, board.D6, board.D7,
board.D8, board.D9),
    row_pins=(board.D10, board.MOSI, board.MISO, board.CLK, board.A0, board.A1)
)

midi_uart = busio.UART(board.TX, None, baudrate=31250, timeout=0.001)

midi_usb_channel = 1
midi_usb = adafruit_midi.MIDI(midi_out=usb_midi.ports[1],
out_channel=midi_usb_channel-1)
midi_serial_channel = 1
midi_serial = adafruit_midi.MIDI(midi_out=midi_uart,
out_channel=midi_serial_channel-1)

octave = 4
note_offset = 9  # first note on keyboard is an A, first key in keypad matrix is 0

def send_note_on(note, octv):
    note = ((note+note_offset)+(12*octv))
    midi_usb.send(NoteOn(note, 120))
    midi_serial.send(NoteOn(note, 120))

def send_note_off(note, octv):
    note = ((note+note_offset)+(12*octv))
```

```
    midi_usb.send(NoteOff(note, 0))
    midi_serial.send(NoteOff(note, 0))

def send_cc(number, val):
    midi_usb.send(ControlChange(number, val))
    midi_serial.send(ControlChange(number, val))

def send_pc(bank, folder, patch):
    send_cc(0, bank)
    send_cc(32, folder)
    midi_usb.send(ProgramChange(patch))
    midi_serial.send(ProgramChange(patch))

def send_bend(bend_start, bend_val, rate, bend_dir):
    b = bend_start
    if bend_dir == 0:
        while b > bend_val + rate:
            print(b)
            b = b - rate
            midi_usb.send(PitchBend(b))
            midi_serial.send(PitchBend(b))
    if bend_dir == 1:
        while b < bend_val - rate:
            print(b)
            b = b + rate
            midi_usb.send(PitchBend(b))
            midi_serial.send(PitchBend(b))

def send_midi_panic():
    for x in range(128):
        midi_usb.send(NoteOff(x, 0))
        midi_serial.send(NoteOff(x, 0))

# key ranges
piano_keys = range(0, 28)  # 'range()' excludes last value, so add one
patch_toes = list(range(28, 33))
cc_toes = list(range(35, 40))
clef_button = 33
nose_button = 47
face_button = 34
record_button = 44
play_button = 45
vol_down_button = 43
vol_up_button = 42
tempo_down_button = 41
tempo_up_button = 40

# patch assigments
patch_list = (
                (0, 0, 0),  # bank 0, folder 0, patch 0
                (1, 0, 0),
                (1, 0, 1),
                (2, 0, 0),
                (3, 0, 0),
)

pb_max = 16383  # bend up value
pb_default = 8192  # bend center value
pb_min = 0  # bend down value
pb_change_rate = 100  # interval for pitch bend, lower number is slower
pb_return_rate = 100  # interval for pitch bend release

# accelerometer filtering variables
slop = 0.2  # threshold for accelerometer send
filter_percent = 0.5  # ranges from 0.0 to 1.0
accel_data_y = msa.acceleration[1]
last_accel_data_y = msa.acceleration[1]

# midi cc variables
```

```
cc_enable = True
cc_numbers = (1, 43, 44, 14, 15)  # mod wheel, filter cutoff, resonance, user, user
cc_current = 0
cc_play = 16
cc_face_number = 17

started = False  # state of arp/seq play
note_hold = False

print("Cyber Cat MIDI Keyboard")


while True:
    if cc_enable:
        new_data_y = msa.acceleration[1]
        accel_data_y = ((new_data_y * filter_percent) + (1-filter_percent) *
accel_data_y)  # smooth
        if abs(accel_data_y - last_accel_data_y) > slop:
            modulation = int(map_range(accel_data_y, 9, -9, 0, 127))
            send_cc(cc_numbers[cc_current], modulation)
            last_accel_data_y = accel_data_y

    event = key_matrix.events.get()
    if event:
        if event.pressed:
            key = event.key_number

            # Note keys
            if key in piano_keys:
                send_note_on(key, octave)

            # Volume buttons
            if key is vol_down_button:
                octave = min(max((octave - 1), 0), 7)
            if key is vol_up_button:
                octave = min(max((octave + 1), 0), 7)

            # Tempo buttons
            if key is tempo_down_button:
                send_bend(pb_default, pb_min, pb_change_rate, 0)
            if key is tempo_up_button:
                send_bend(pb_default, pb_max, pb_change_rate, 1)

            # Patch buttons (left cat toes)
            if key in patch_toes:
                pc_key = patch_toes.index(key)  # remove offset for patch list
indexing
                send_pc(patch_list[pc_key][0], patch_list[pc_key][1],
patch_list[pc_key][2])

            # cc buttons (right cat toes)
            if key in cc_toes:
                cc_current = cc_toes.index(key)  # remove offset for cc list
indexing

            # Play key -- use MIDI learn to have arp/seq start or stop with this
            if key is play_button:
                if not started:
                    send_cc(cc_play, 127)  # map to seq/arp on/off Synth One, e.g.
                    started = True
                else:
                    send_cc(cc_play, 0)
                    started = False

            # Record key -- enable icecream cone
            if key is record_button:
                if cc_enable is True:
                    cc_enable = False
                    ledpin.value = False
```

```
                elif cc_enable is False:
                    send_cc(cc_numbers[cc_current], 0)  # zero it
                    cc_enable = True
                    ledpin.value = True

            # Clef
            if key is clef_button:  # hold
                note_hold = not note_hold

            # Face
            if key is face_button:  # momentary cc
                send_cc(cc_face_number, 127)

            # Nose
            if key is nose_button:
                send_midi_panic()  # all notes off

        if event.released:
            key = event.key_number
            if key in piano_keys:
                if not note_hold:
                    send_note_off(key, octave)
                if note_hold:
                    pass

            if key is face_button:  # momentary cc release
                send_cc(cc_face_number, 0)

            if key is tempo_down_button:
                send_bend(pb_min, pb_default, pb_return_rate, 1)

            if key is tempo_up_button:
                send_bend(pb_max, pb_default, pb_return_rate, 0)
```

How It Works

The main functions of the code are to turn key and button presses into MIDI messages, and to read the accelerometer and turn it's values into MIDI CC messages.

Libraries

First, we import libraries. `keypad` allows us to read the key/button matrix, `board` gives us pin definitions, `busio` is used for I2C, `digitalio` for LED, `simplemath` `map_range` is used to turn accelerometer values into useable CC values, `msa311` is the accelerometer board, and the remaining libraries are for `MIDI`.

```
import keypad
import board
import busio
import supervisor
import digitalio
from adafruit_simplemath import map_range
from adafruit_msa3xx import MSA311
import usb_midi
import adafruit_midi
from adafruit_midi.note_on import NoteOn
from adafruit_midi.note_off import NoteOff
```

```
from adafruit_midi.control_change import ControlChange
from adafruit_midi.program_change import ProgramChange
from adafruit_midi.pitch_bend import PitchBend
```

Setup

Next comes setup.

The LED is set up as a digitalio output and turned on.

```
ledpin = digitalio.DigitalInOut(board.A3)
ledpin.direction = digitalio.Direction.OUTPUT
ledpin.value = True
```

Accelerometer

The accelerometer is set up on I2C next.

```
i2c = board.STEMMA_I2C()
msa = MSA311(i2c)
```

Key Matrix

The key matrix is defined by the eight column pins and six row pins.

```
key_matrix = keypad.KeyMatrix(
    column_pins=(board.D2, board.D3, board.D4, board.D5, board.D6, board.D7,
board.D8, board.D9),
    row_pins=(board.D10, board.MOSI, board.MISO, board.CLK, board.A0, board.A1)
)
```

You'll set things up so octaves can shifted with one of the arrow button pairs. The `octave` variable will keep track of this, and the `note_offset` accounts for the lowest key (0 in the matrix) being an A (9 for the lowest MIDI A).

```
octave = 4
note_offset = 9
```

MIDI

MIDI is set up on both the serial UART and over USB. You can change which channels to use here if you like.

```
midi_uart = busio.UART(board.TX, None, baudrate=31250)
```

```
midi_usb_channel = 1
midi_usb = adafruit_midi.MIDI(midi_out=usb_midi.ports[1],
out_channel=midi_usb_channel-1)
midi_serial_channel = 1
midi_serial = adafruit_midi.MIDI(midi_out=midi_uart,
out_channel=midi_serial_channel-1)
```

These functions are set up to send all of the required MIDI messages:

```
def send_note_on(note, octv):
    note = ((note+note_offset)+(12*octv))
    midi_usb.send(NoteOn(note, 120))
    midi_serial.send(NoteOn(note, 120))

def send_note_off(note, octv):
    note = ((note+note_offset)+(12*octv))
    midi_usb.send(NoteOff(note, 0))
    midi_serial.send(NoteOff(note, 0))

def send_cc(number, val):
    midi_usb.send(ControlChange(number, val))
    midi_serial.send(ControlChange(number, val))

def send_pc(bank, folder, patch):
    send_cc(0, bank)
    send_cc(32, folder)
    midi_usb.send(ProgramChange(patch))
    midi_serial.send(ProgramChange(patch))

def send_bend(bend_start, bend_val, rate, bend_dir):
    b = bend_start
    if bend_dir == 0:
        while b > bend_val + rate:
            print(b)
            b = b - rate
            midi_usb.send(PitchBend(b))
            midi_serial.send(PitchBend(b))
    if bend_dir == 1:
        while b < bend_val - rate:
            print(b)
            b = b + rate
            midi_usb.send(PitchBend(b))
            midi_serial.send(PitchBend(b))

def send_midi_panic():
    for x in range(128):
        midi_usb.send(NoteOff(x, 0))
        midi_serial.send(NoteOff(x, 0))
```

Key/Button Assignments

All of the keys and buttons will show up as a key matrix event when pressed or released, these are their correlations to the physical buttons:

```
piano_keys = range(0, 28)  # 'range()' excludes last value, so add one
patch_toes = list(range(28, 33))
cc_toes = list(range(35, 40))
clef_button = 33
nose_button = 47
face_button = 34
record_button = 44
play_button = 45
```

```
vol_down_button = 43
vol_up_button = 42
tempo_down_button = 41
tempo_up_button = 40
```

Patches

The five left paw toes send MIDI bank/folder/patch messages which are defined in this list:

```
patch_list = (
                (0, 0, 0),  # bank 0, folder 0, patch 0
                (1, 0, 0),
                (1, 0, 1),
                (2, 0, 0),
                (3, 0, 0),
)
```

Pitch Bend Setup

These variables set the maximum, default, and minimum values for pitch bend messages, as well as the increment size to sweep through them.

```
pb_max = 16383  # bend up value
pb_default = 8192  # bend center value
pb_min = 0  # bend down value
pb_change_rate = 100  # interval for pitch bend, lower number is slower
pb_return_rate = 100  # interval for pitch bend release
```

Accelerometer Filtering

These values are used to filter the raw accelerometer readings into something more useable.

```
# accelerometer filtering variables
slop = 0.2  # threshold for accelerometer send
filter_percent = 0.5  # ranges from 0.0 to 1.0
accel_data_y = msa.acceleration[1]
last_accel_data_y = msa.acceleration[1]
```

MIDI CC Variables

These variables are used to track the ice cream cone CC enable state, set the CC numbers used by the right paw toes, play button, and face button.

The `started` and `note_hold` variables track state of the arpeggiator/sequencer enable switch (on the software/hardware synth, there is not a built in arpeggiator in the code), and the state of the `note_hold` clef button.

```
cc_enable = True
cc_numbers = (1, 43, 44, 14, 15)  # mod wheel, filter cutoff, resonance, user, user
cc_current = 0
cc_play = 16
cc_face_number = 17
started = False  # state of arp/seq play
note_hold = False
```

Main Loop

The main loop checks for accelerometer changes and button events.

If the accelerometer is enabled, it will send CC messages on the chosen number.

```
while True:
    if cc_enable:
        new_data_y = msa.acceleration[1]
        accel_data_y = ((new_data_y * filter_percent) + (1-filter_percent) *
accel_data_y)  # smooth
        if abs(accel_data_y - last_accel_data_y) > slop:
            modulation = int(map_range(accel_data_y, 9, -9, 0, 127))
            send_cc(cc_numbers[cc_current], modulation)
            last_accel_data_y = accel_data_y
```

Key Matrix Events

This line checks to see if any keys or buttons have been pressed or released:

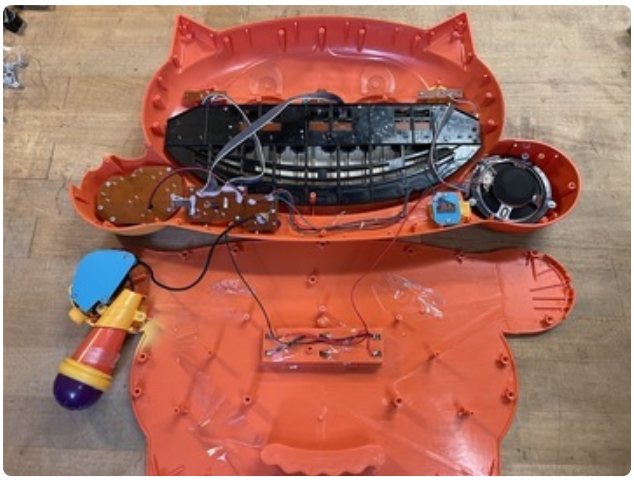
`event = key_matrix.events.get()`

If pressed or released, they all run their corresponding functions as defined earlier:

```
if event:
        if event.pressed:
            key = event.key_number

            # Note keys
            if key in piano_keys:
                send_note_on(key, octave)

            # Volume buttons
            if key is vol_down_button:
                octave = min(max((octave - 1), 0), 7)
            if key is vol_up_button:
                octave = min(max((octave + 1), 0), 7)

            # Tempo buttons
            if key is tempo_down_button:
                send_bend(pb_default, pb_min, pb_change_rate, 0)
```

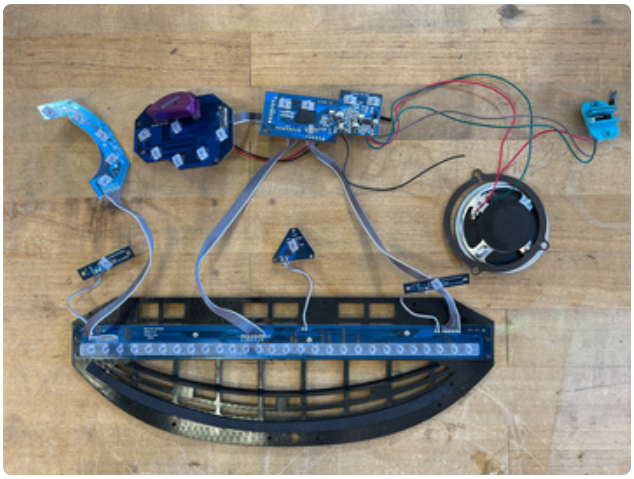
```
            if key is tempo_up_button:
                send_bend(pb_default, pb_max, pb_change_rate, 1)

            # Patch buttons (left cat toes)
            if key in patch_toes:
                pc_key = patch_toes.index(key)  # remove offset for patch list
indexing
                send_pc(patch_list[pc_key][0], patch_list[pc_key][1],
patch_list[pc_key][2])

            # cc buttons (right cat toes)
            if key in cc_toes:
                cc_current = cc_toes.index(key)  # remove offset for cc list
indexing

            # Play key -- use MIDI learn to have arp/seq start or stop with this
            if key is play_button:
                if not started:
                    send_cc(cc_play, 127)  # map to seq/arp on/off Synth One, e.g.
                    started = True
                else:
                    send_cc(cc_play, 0)
                    started = False

            # Record key -- enable icecream cone
            if key is record_button:
                if cc_enable is True:
                    cc_enable = False
                    ledpin.value = False

                elif cc_enable is False:
                    send_cc(cc_numbers[cc_current], 0)  # zero it
                    cc_enable = True
                    ledpin.value = True

            # Clef
            if key is clef_button:  # hold
                note_hold = not note_hold

            # Face
            if key is face_button:  # momentary cc
                send_cc(cc_face_number, 127)

            # Nose
            if key is nose_button:
                send_midi_panic()  # all notes off

        if event.released:
            key = event.key_number
            if key in piano_keys:
                if not note_hold:
                    send_note_off(key, octave)
                if note_hold:
                    pass

            if key is face_button:  # momentary cc release
                send_cc(cc_face_number, 0)

            if key is tempo_down_button:
                send_bend(pb_min, pb_default, pb_return_rate, 1)

            if key is tempo_up_button:
                send_bend(pb_max, pb_default, pb_return_rate, 0)
```

Build the Cyber Cat MIDI Keyboard



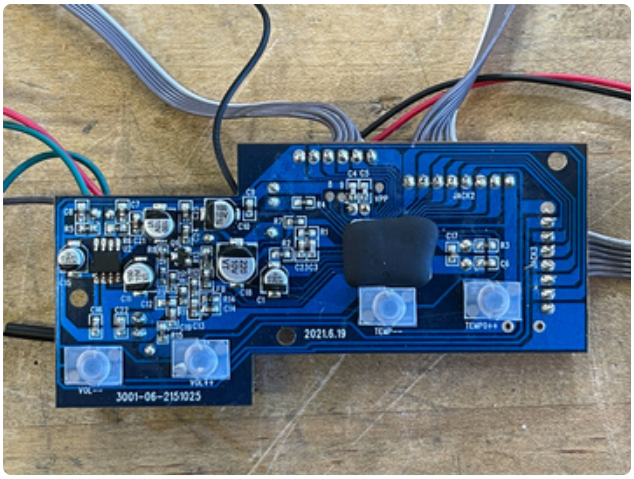
Disassembly

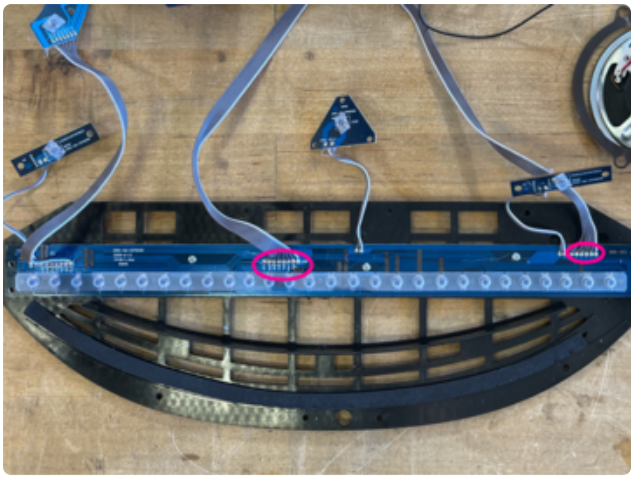
First, flip over the keyboard and remove all of the screws, then pop it open.



Innards

Unscrew all of the assemblies and remove them from the case -- this will make it easier to solder the wiring to the PermaProto boards, and allow you to paint the case should you so choose.





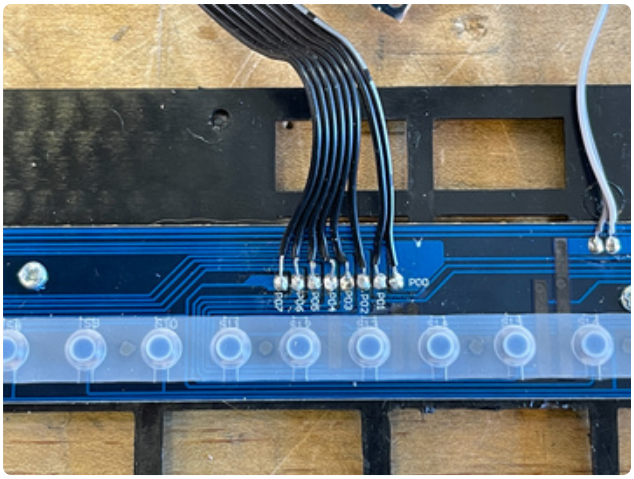
Matrix Wiring

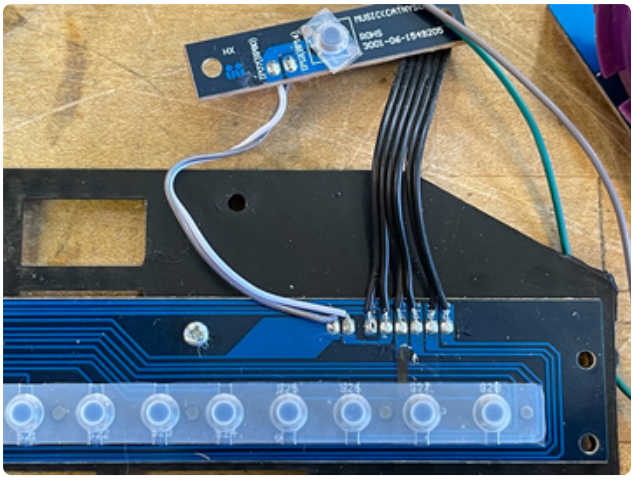
All of the keys and buttons are part of a 8x6 matrix decoded by the epoxied on-board microcontroller. We'll desolder some of the existing ribbon wires and solder our own from the keyboard to a PermaProto breadboard. This way, we can decode the matrix with the Kee Boar KB2040 running CircuitPython.

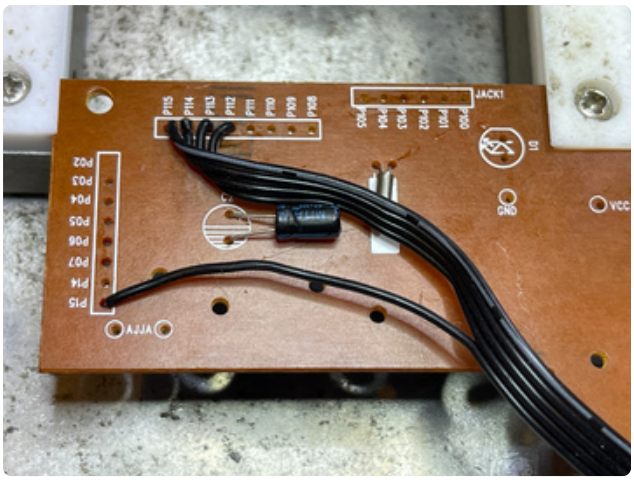
Desolder all of the ribbon cables running to the Meowsic PCB.

Next, desolder two of the three sets of ribbon cable from the keyboard PCB as shown in the photo.

If your Meowsic keyboard has different pin labels (these things get revised periodically) you may need to use a continuity tester to determine which pins act as which parts of the matrix.





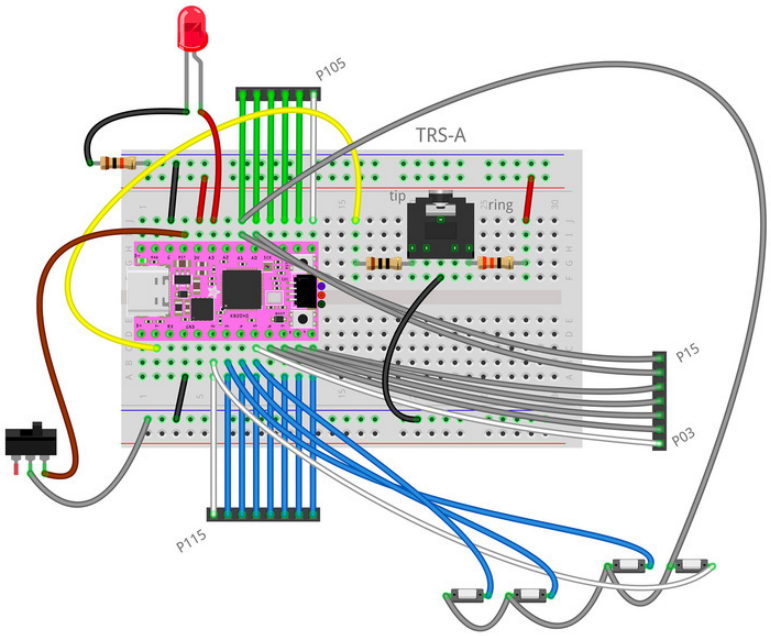


New Ribbons

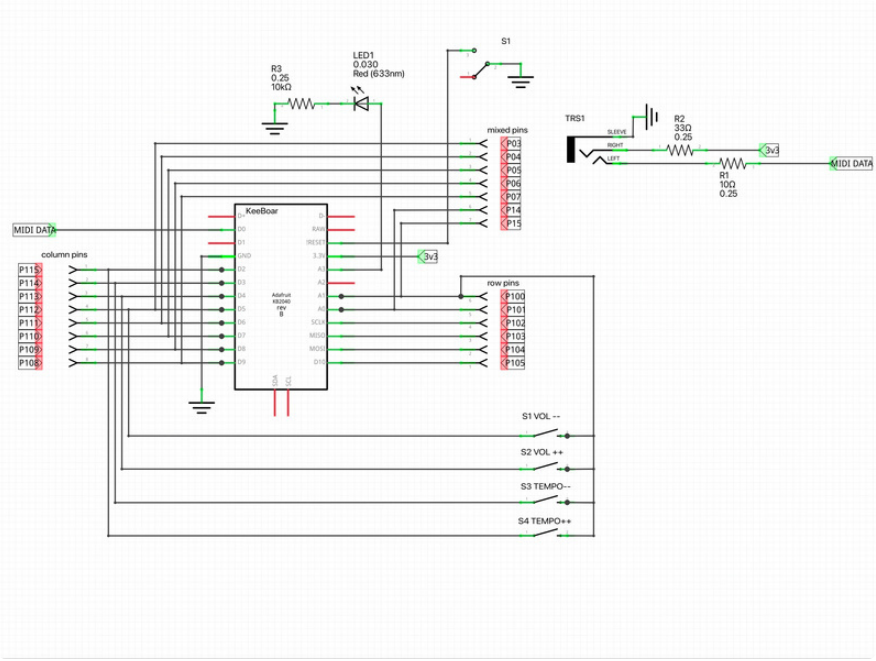
Solder silicone ribbon cables from the two keyboard matrix positions as shown. These will connect the keyboard (an satellite button pads) to the KB2040 as shown in the Fritzing diagram.

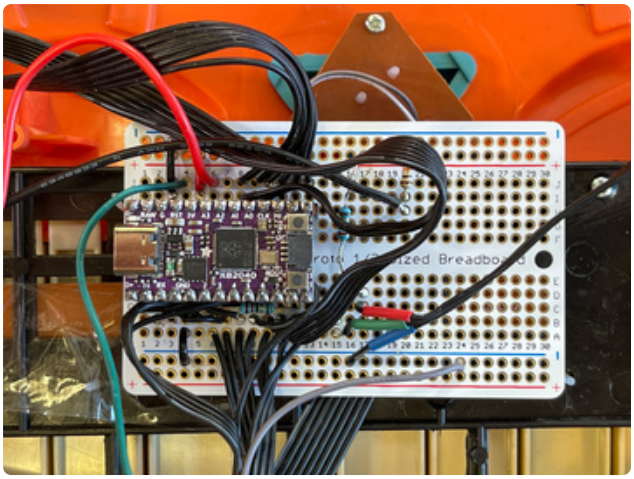
Solder a third ribbon wire to the original PCB as shown in order to use the four arrow buttons.

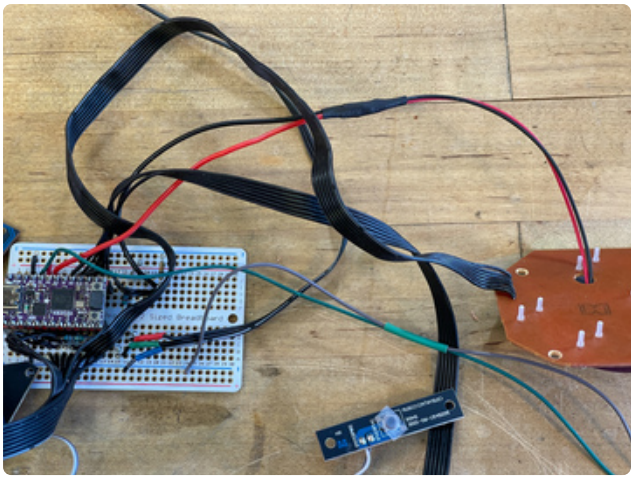
Circuit Diagram



Schematic







Circuit

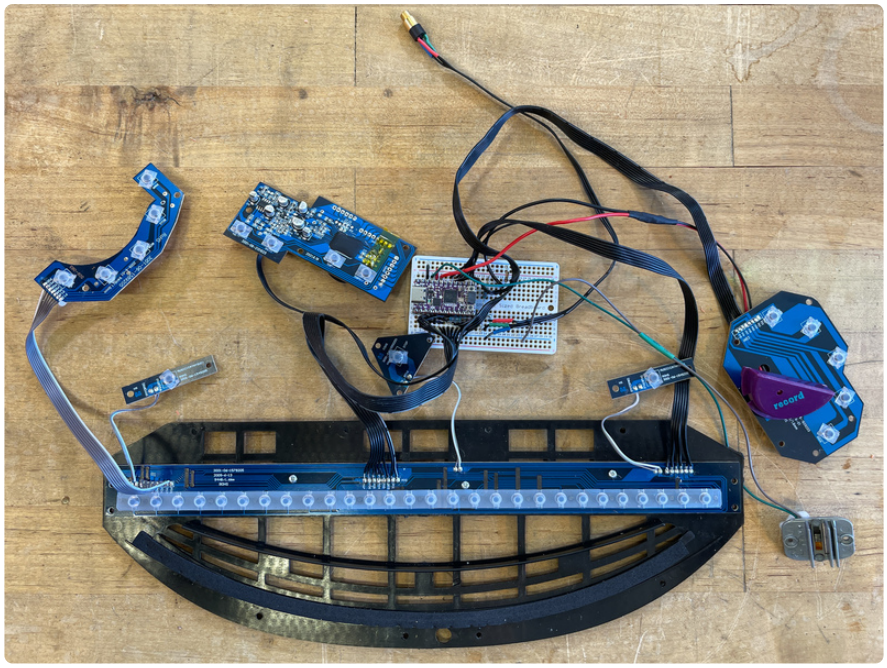
Build the circuit shown in the Fritzing diagram and schematic above onto a PermaProto.

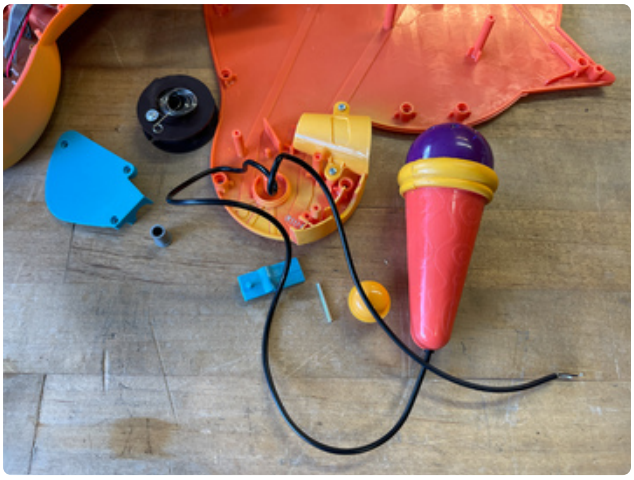
The LED is built into the keyboard's record button but you'll add a 10KΩ resistor to one leg.

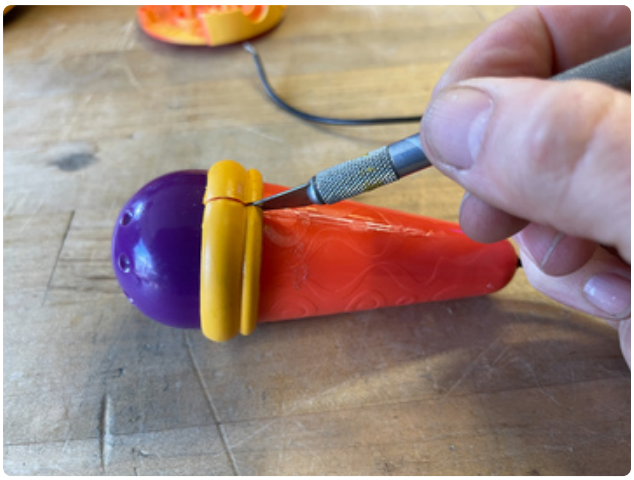
You'll want to extend the TRS-A jack to the case with a panel-mount version.

You may omit the TRS-A midi plug if you want to just use USB MIDI.

You'll wire the common and right contacts of the slide switch to **GND** and **Enable** so it can be repurposed as a reset button.







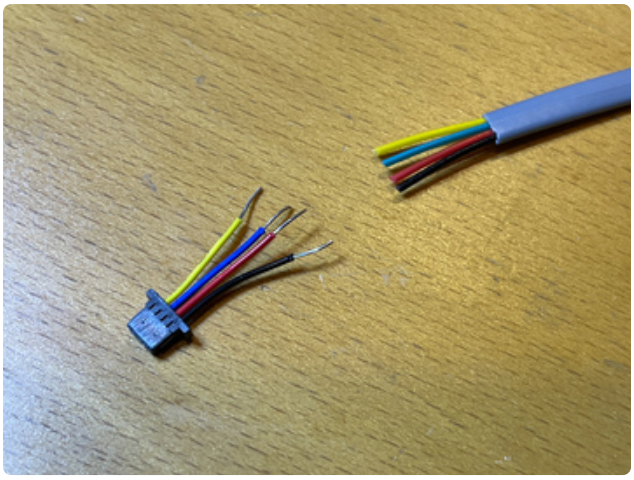
Ice Cream Cone Modulation

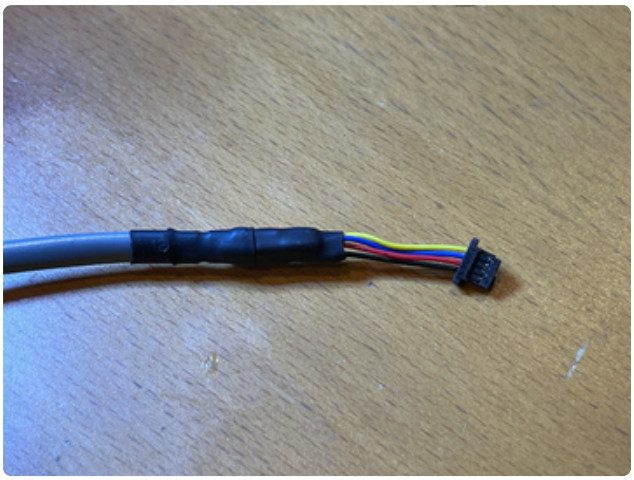
The ice cream cone houses a microphone, but we'll repurpose that as a gesture controller for MIDI CC such as filter depth by inserting an accelerometer.

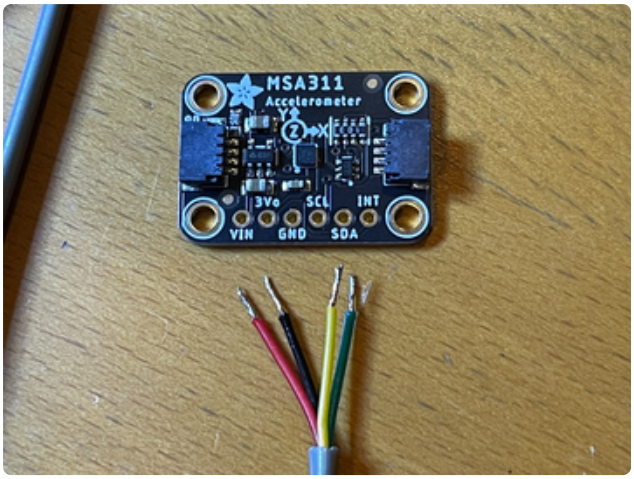
Disassemble the cone mechanism from the base.

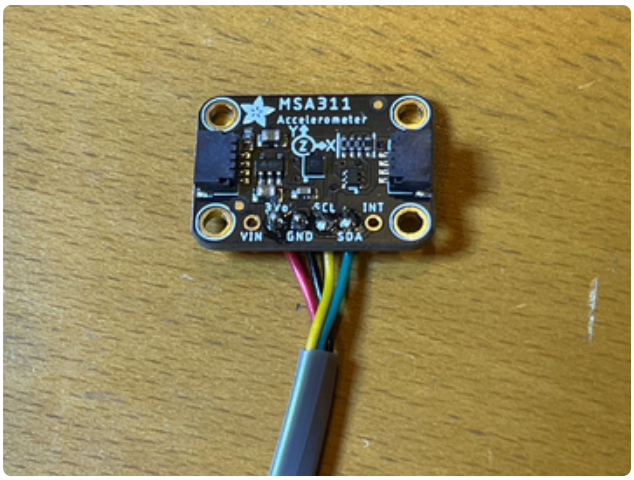
Cut open the cone carefully (very carefully) with a thin blade along the glued seams.

Use a four cable conductor, such as an RJ-14 phone cable, to create an extended STEMMA QT cable (or simply solder each end to the accelerometer board and PermaProto to keep it simple).






MSA311
Accelerometer
3Vo
SCL
INT
VIN
GND
SDA


MSA311
Accelerometer
3Vo
SCL
INT
VIN
GND
SDA

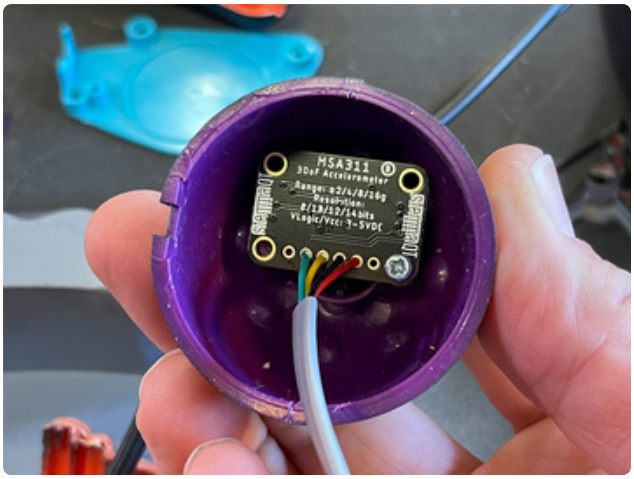
Prime the Case and Parts

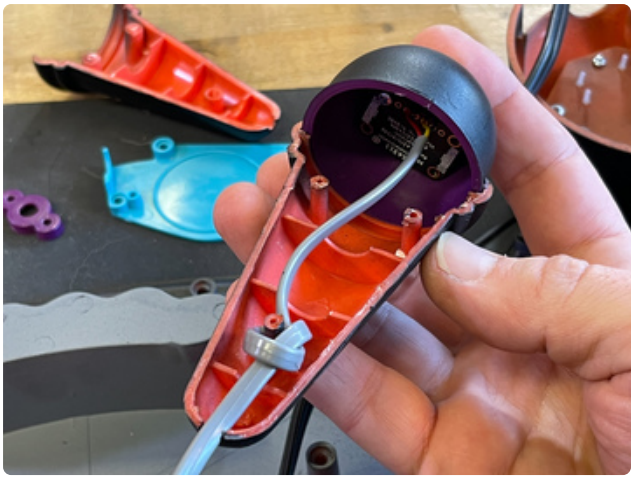
Since the keyboard is already pulled apart, this is a great time to give the Cyber Cat keyboard a new paint job!

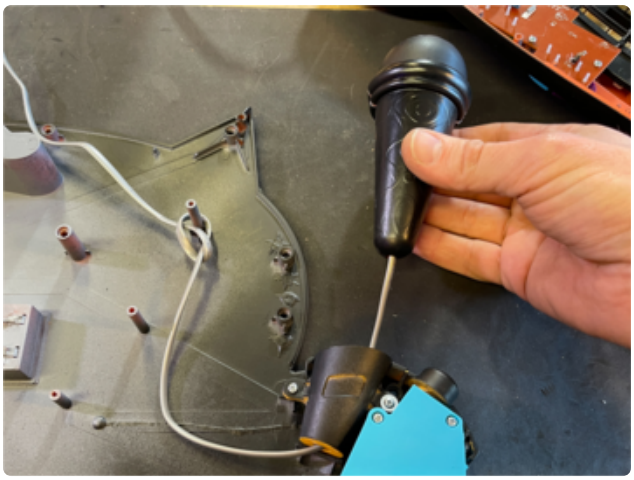
Use a few coats of primer, allowing proper drying time between coats.

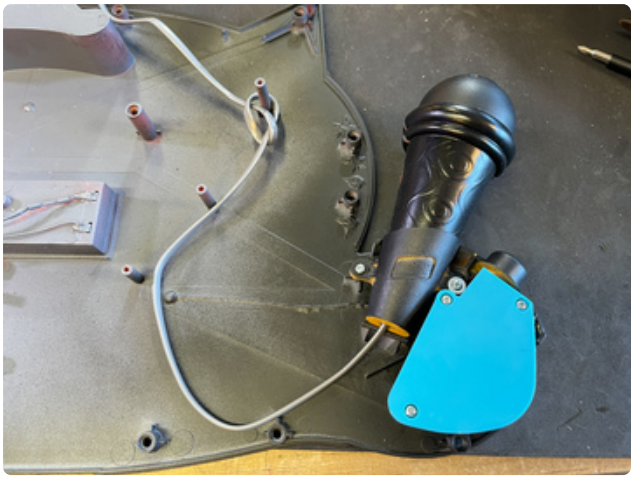
Paint

Then, paint each part to suit your taste. Use at least 3-5 coats with proper drying time between coats, and then use a matte or gloss sealant for surface protection.









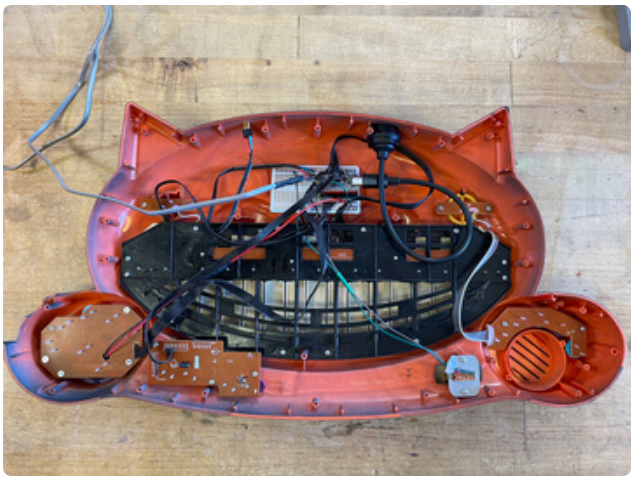
Cone Assembly

Screw the IMU board into the scoop's existing mounting post as shown.

Knot the cable around a cone mounting post for strain relief, feed the end through the cone holder and plug the STEMMA QT connector end into the KB2040.

Re-assemble the retraction mechanism if you like (it fills in some case gaps, but we won't be using it. Instead, knot the cable on a mounting post on the base so a short length of slack will allow the cone to be used and then placed back in its holder.

Close up the cone using a few dabs of CA glue.



Assembly

Place all of the buttons back in their holders.

Starting with the keybed and satellite button boards, screw in each part to re-assemble.

Carefully drill holes for the panel mount USB connector and TRS MIDI jack, then attach them.

Finally, press the back panel into place and re-fasten the screws.

Use the Cyber Cat MIDI Keyboard

Usage and specific settings will vary somewhat depending on the synthesizer you control. Hardware synths should recognize the Cyber Cat and "just work" however you can change the MIDI channel and CC#s in the **code.py** file if necessary.

Software synthesizers will usually recognize the Cyber Cat automatically, however if you need to pick the MIDI input device from a list, choose the one named **KB2040 CircuitPython usb_midi.ports[0]**

Some functions will work without adjustment, such as note on/off, octave switching, MIDI panic, and pitch bend. Others will require MIDI CC learning in software to pair a Cyber Cat button to a specific parameter of the synthesizer.

Here are the button functions for the Cyber Cat:

Here it is in action: