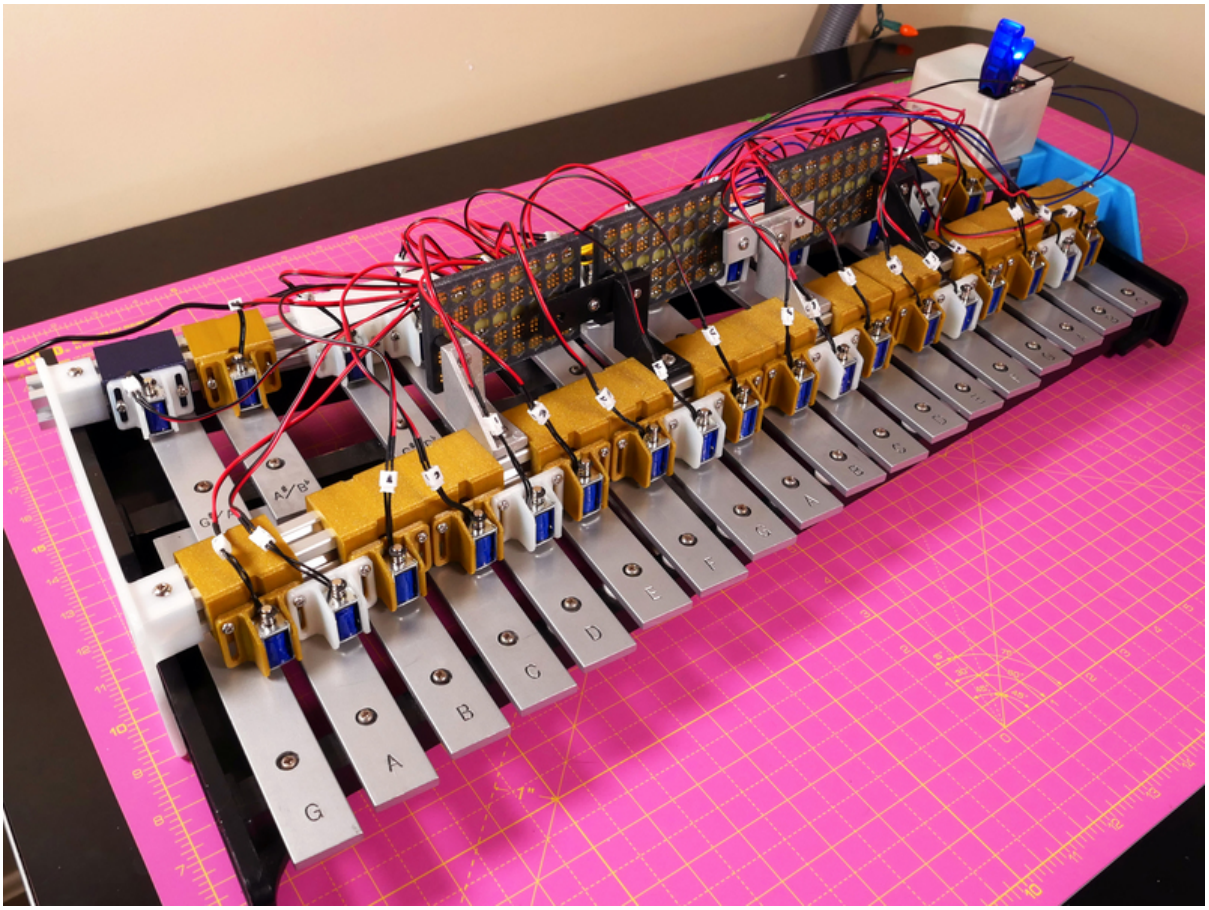
Wireless BLE MIDI Robot Xylophone

Created by Liz Clark



https://learn.adafruit.com/wireless-ble-midi-robot-xylophone

Last updated on 2024-06-03 03:07:54 PM EDT

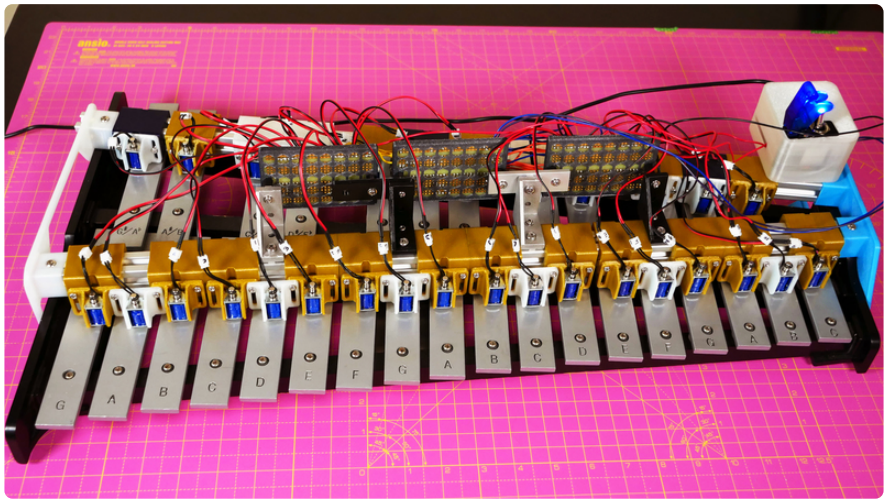
Table of Contents

Overview

This project converts a xylophone* into a MIDI instrument, but not just any MIDI instrument: a Bluetooth Low Energy (BLE) MIDI instrument. This means that it can be setup within range of a Bluetooth connection and receive MIDI data without being tied down with a bunch of cables (besides power of course).

30 solenoids sit comfortably over the xylophone keys, waiting to strike up melodies and chords either live with a MIDI keyboard or playing along with MIDI data from a digital audio workstation (DAW). It has a lot of potential as either a live instrument or for some extra fun in the studio.

*Yes, technically this is a bell kit, or glockenspiel, and not a xylophone. However, most people will recognize instruments in the mallet family colloquially as xylophones. Xylophone is also comparatively easier to say and more commonly heard than glockenspiel or bell kit.

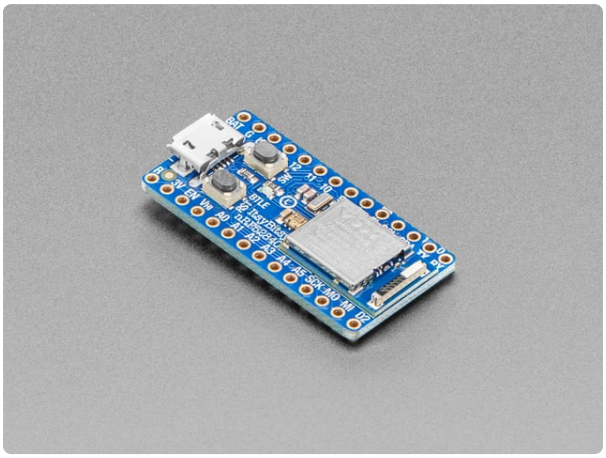


The Xylophone/Glockenspiel

The model instrument used for this project is fairly common in the United States, as it is often distributed to students in school who begin taking percussion lessons in music programs. You can find the identical model by searching for "**Ludwig Educational Bell Kit**". There's quite a few available new and used on Ebay, Amazon, Reverb, Guitar Center or perhaps your cousin's attic.

Of course, you can also modify the mounting system to fit your chosen mallet instrument, or any other instrument. As long as it has metal bars it will sound nicely with the solenoids. Any other material bar should have some sort of dampener attached to the solenoid since they could damage softer materials, such as wood.

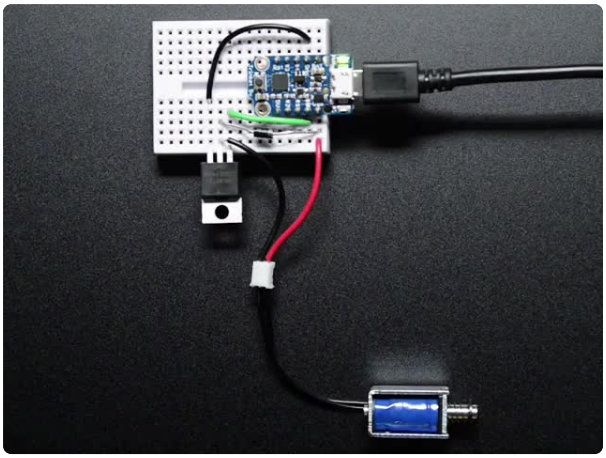
Parts



Adafruit ItsyBitsy nRF52840 Express - Bluetooth LE

What's smaller than a Feather but larger than a Trinket? It's an Adafruit ItsyBitsy nRF52840 Express featuring the Nordic nRF52840 Bluetooth LE...

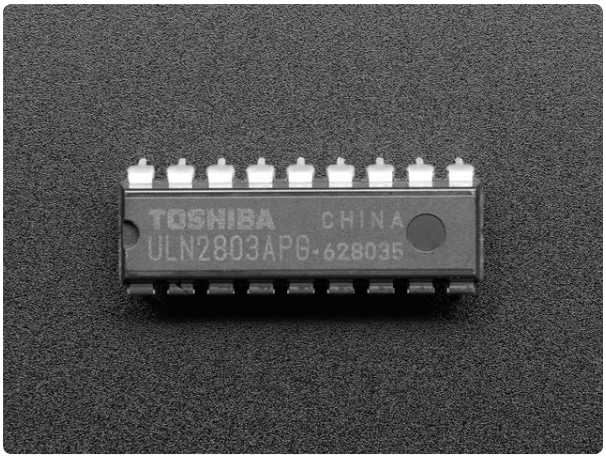
https://www.adafruit.com/product/4481



Mini Push-Pull Solenoid - 5V

Solenoids are basically electromagnets: they are made of a coil of copper wire with an armature (a slug of metal) in the middle. When the coil is energized, the slug is pulled into the...

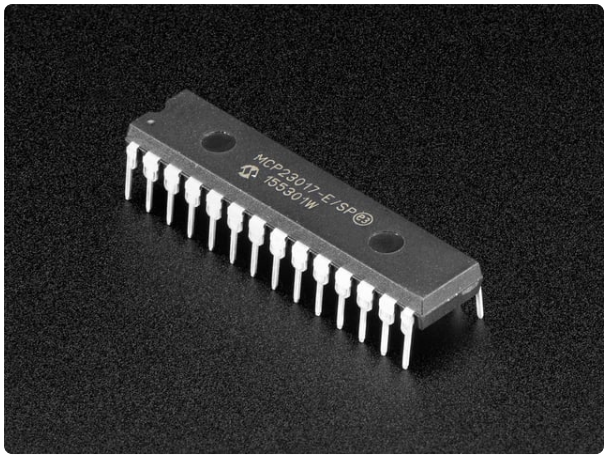
https://www.adafruit.com/product/2776



ULN2803: 8 Channel Darlington Driver (Solenoid/Unipolar Stepper)

Bring in some muscle to your output pins with 8 mighty Darlingtons! This DIP chip contains 8 drivers that can sink 500mA from a 50V supply and has kickback diodes included inside for...

https://www.adafruit.com/product/970



MCP23017 - i2c 16 input/output port expander

Add another 16 pins to your microcontroller using an MCP23017 port expander. The MCP23017 uses two i2c pins (these can be shared with other i2c devices), and in exchange gives you 16...
https://www.adafruit.com/product/732

Illuminated Toggle Switch with Cover - Blue

Fire up your engines because these are the hot-rods of toggle switches! Equipped with an aerodynamic blue protective casing, and a diffused blue LED at the tip of the switch, this...
https://www.adafruit.com/product/3306

1 x 1/2 Perma-Proto Breadboards - 3 Pack — https://www.adafruit.com/product/2782
Adafruit Perma-Proto Half-sized Breadboard PCB - 3 Pack

---

1 x Perma-Proto Quarter-sized Breadboard — https://www.adafruit.com/product/1608
Adafruit Perma-Proto Quarter-sized Breadboard PCB - Single

---

1 x Through-Hole Resistors - 2.2K ohm 5% 1/4W — https://www.adafruit.com/product/2782
2.2K ohm resistors - Pack of 25

---

1 x CONN IC DIP SOCKET 28POS — https://www.digikey.com/product-detail/en/on-shore-technology-inc/ED281DT/ED3050-5-ND/4147600
28 Pin DIP socket - DigiKey

---

2 x CONN IC DIP SOCKET 18POS — https://www.digikey.com/product-detail/en/on-shore-technology-inc/ED18DT/ED3047-5-ND/4147597
18 Pin DIP socket - DigiKey

---

1 x Hook-up Wire — https://www.adafruit.com/product/3174
Hook-up Wire Spool Set - 22AWG Solid Core - 10 x 25ft

JST-PH Battery Extension Cable - 500mm

30 x JST-PH Extension Cable — https://www.adafruit.com/product/1131

JST-PH Battery Extension Cable - 500mm

---

2 x 20x20 Aluminum Extrusion — https://www.adafruit.com/product/1221

Slotted Aluminum Extrusion - 20mm x 20mm - 610mm long

---

2 x Coupling Plate — https://www.adafruit.com/product/1216

Coupling Plate - 3 Holes - 20x20 Aluminum Extrusion

---

4 x Double Corner Brace — https://www.adafruit.com/product/1259

Aluminum Extrusion Double Corner Brace Support (for 20x20)

---

1 x Slim T-Nuts — https://www.adafruit.com/product/1157

Aluminum Extrusion Slim T-Nut for 20x20 - M4 Thread - pack of 50

---

1 x M4 Button Hex Screws — https://www.adafruit.com/product/1160

Button Hex Machine Screw - M4 thread - 8mm long - pack of 50

---

1 x M3 Standoffs - Variety pack — https://www.amazon.com/Litorange-Standoff-Threaded-Motherboard-Assortment/dp/B07D7828LC/

M3 standoffs

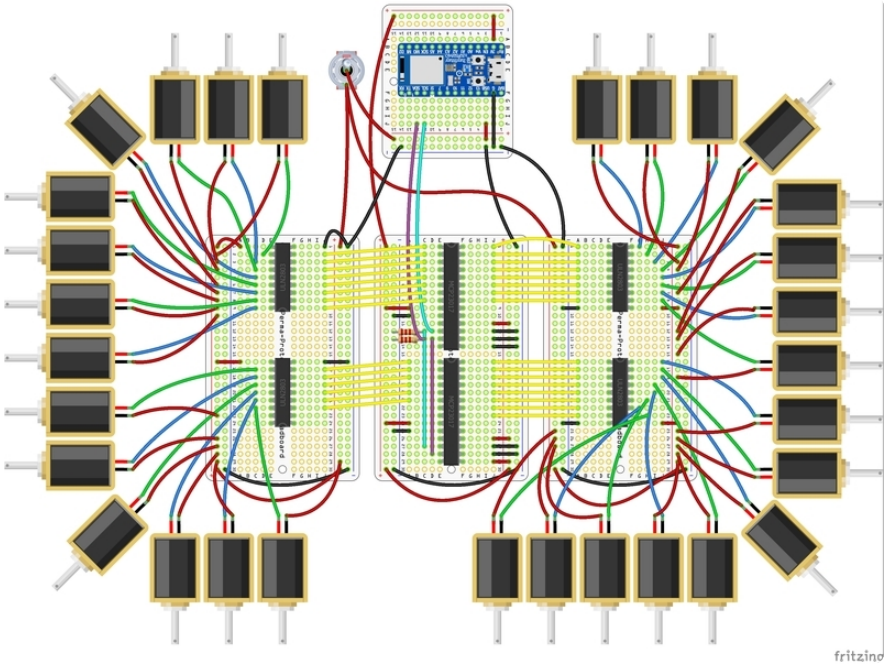
---

1 x M3 Screws - Variety pack — https://www.amazon.com/gp/product/B07YXXK4HB/

M3 screws

Electronics

Circuit Diagram



How It Works

The ItsyBitsy nRF52840 is running CircuitPython code that allows for MIDI to be received over BLE. Additionally, it's communicating over I2C between two MCP23017's.

The MCP23017 is a multiplexer that expands the number of digital inputs and outputs that a microcontroller can communicate with. In this case, they're allowing for 30 additional outputs for the solenoid motors.

Before the solenoids can be triggered through, the MCP23017's output their signal to a ULN2803. The ULN2803 is a darlington motor driver, which is great for driving solenoids and they can take in 3.3V or 5V logic.

For more info on the MCP23017, check out this Learn Guide

https://adafru.it/Lkc

Datasheet for the ULN2308

https://adafru.it/Lkd

The CircuitPython code on the IstyBitsy nRF52840 sends on and off signals over I2C to the MCP23017's depending on the MIDI note that it receives over BLE. The MCP23017 then sends an on signal, followed quickly by an off signal, to a ULN2308 driver. Finally, the ULN2308 triggers the solenoid to hit a note on the xylophone.

Although there's quite a bit of wiring between the IC's and solenoids, the ItsyBitsy is only using 5 pins, and that's counting power and ground. The use of I2C keeps things nice and compact.

Wiring

Power:

- **3V** from the ItsyBitsy to **VDD** and **RESET** on both MCP23017's
- **3V** from the ItsyBitsy to **A0** on the second MCP23017
- **USB** from the ItsyBitsy to the switch
- **COMMON** on all four ULN2308's to **USB** from the switch

Ground:

- **GND** from the ItsyBitsy to **VSS**, **A2** and **A1** on both MCP23017's
- **GND** from the ItsyBitsy to **A0** in the first MCP23017
- **GND** from the ItsyBitsy to **GND** on all four ULN2308's

I2C:

- **2.2K** pull-up resistor on both **SCL** and **SDA** on the first MCP23017
- **SCL** from the ItsyBitsy to **SCL** on both MCP23017's
- **SDA** from the ItsyBitsy to **SDA** on both MCP23017's

Solenoids:

- The solenoid's **wire 1** to the output pins (**O1-O8**) on all four ULN2803's
- The solenoid's **wire 2** to **USB** from the switch

A0, A1 and A2 on the MCP23017 determine the I2C address, which is why the wiring differs between the two.

Data Signal Flow

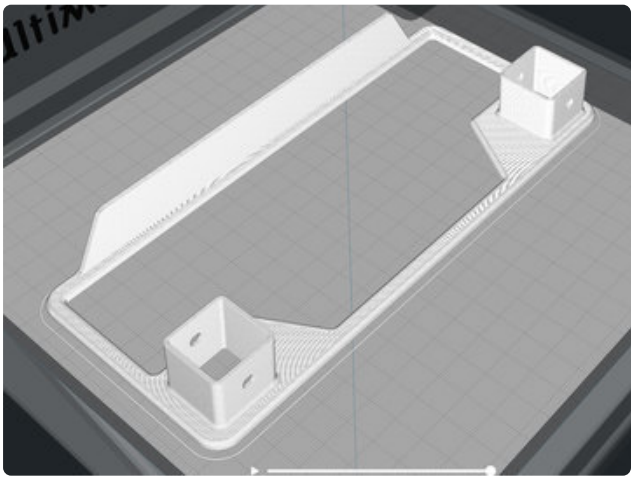
The chart below explains the pinouts for the multiplexers to the drivers and what MIDI note they correspond with.

Note (Name and MIDI #)	MUX (#, Pin Name, Pin #)	Driver (#, Pin Name, Pin Number)
G2 (55)	MUX1, **GPA0**, Pin 21	Driver 1, **I1**, Pin 1
G#2 (56)	MUX1, **GPA1**, Pin 22	Driver 1, **I2**, Pin 2
A2 (57)	MUX1, **GPA2**, Pin 23	Driver 1, **I3**, Pin 3
A#2 (58)	MUX1, **GPA3**, Pin 24	Driver 1, **I4**, Pin 4
B2 (59)	MUX1, **GPA4**, Pin 25	Driver 1, **I5**, Pin 5
C3 (60)	MUX1, **GPA5**, Pin 26	Driver 1, **I6**, Pin 6
C#3 (61)	MUX1, **GPA6**, Pin 27	Driver 1, **I7**, Pin 7
D3 (62)	MUX1, **GPA7**, Pin 28	Driver 1, **I8**, Pin 8
D#3 (63)	MUX1, **GPB0**, Pin 1	Driver 2, **I1**, Pin 1
E3 (64)	MUX1, **GPB1**, Pin 2	Driver 2, **I2**, Pin 2

F3 (65)	MUX1, **GPB2**, Pin 3	Driver 2, **I3**, Pin 3
F#3 (66)	MUX1, **GPB3**, Pin 4	Driver 2, **I4**, Pin 4
G3 (67)	MUX1, **GPB4**, Pin 5	Driver 2, **I5**, Pin 5
G#3 (68)	MUX1, **GPB5**, Pin 6	Driver 2, **I6**, Pin 6
A3 (69)	MUX1, **GPB6**, Pin 7	Driver 2, **I7**, Pin 7
A#3 (70)	MUX1, **GPB7**, Pin 8	Driver 2, **I8**, Pin 8
B3 (71)	MUX2, **GPA0**, Pin 21	Driver 3, **I1**, Pin 1
C4 (72)	MUX2, **GPA1**, Pin 22	Driver 3, **I2**, Pin 2
C#4 (73)	MUX2, **GPA2**, Pin 23	Driver 3, **I3**, Pin 3
D4 (74)	MUX2, **GPA3**, Pin 24	Driver 3, **I4**, Pin 4
D#4 (75)	MUX2, **GPA4**, Pin 25	Driver 3, **I5**, Pin 5
E4 (76)	MUX2, **GPA5**, Pin 26	Driver 3, **I6**, Pin 6
F4 (77)	MUX2, **GPA6**, Pin 27	Driver 3, **I7**, Pin 7
F#4 (78)	MUX2, **GPA7**, Pin 28	Driver 3, **I8**, Pin 8

G4 (79)	MUX2, **GPB0**, Pin 1	Driver 4, **I1**, Pin 1
G#4 (80)	MUX2, **GPB1**, Pin 2	Driver 4, **I2**, Pin 2
A4 (81)	MUX2, **GPB2**, Pin 3	Driver 4, **I3**, Pin 3
A#4 (82)	MUX2, **GPB3**, Pin 4	Driver 4, **I4**, Pin 4
B4 (83)	MUX2, **GPB4**, Pin 5	Driver 4, **I5**, Pin 5
C5 (84)	MUX2, **GPB5**, Pin 6	Driver 4, **I6**, Pin 6

3D Printing



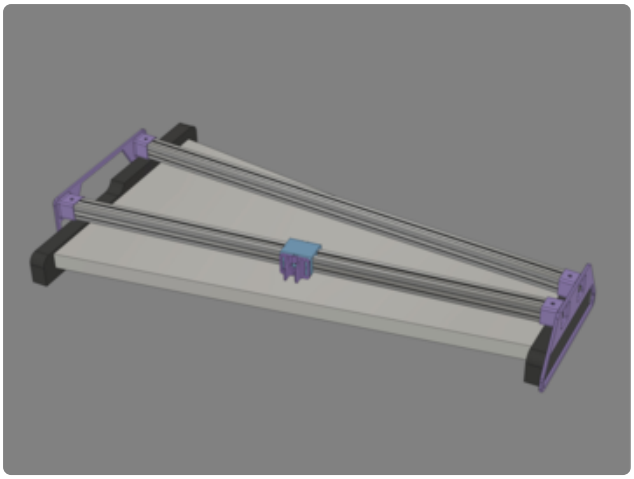
Parts List

STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material. Original design source may be downloaded using the links below.

base-foot-left.stl
base-foot-right.stl
rail-mount.stl
noid-mount.stl

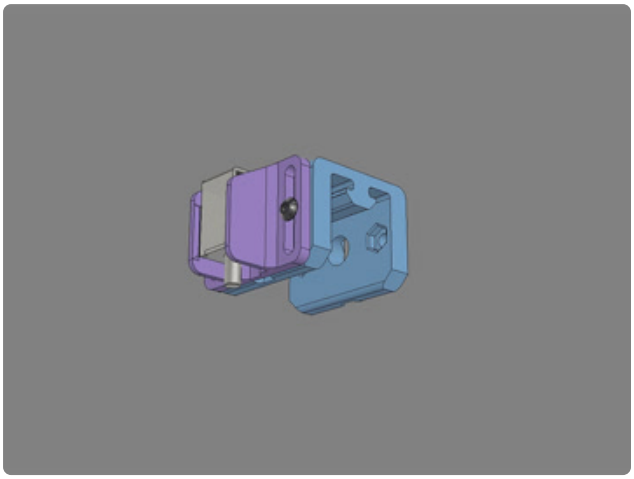
Download CAD from Fusion 360

https://adafru.it/Lla



CAD Assembly

The left and right base feet are secured to the two pieces of 2020 aluminum stock using M4 hardware screws and nuts. The rail mount is fitted into the slotted profiles. The solenoid mount is secured to the rail mount using two M3 x 10mm screws. The mini 5V solenoid is press fitted into the solenoid mount.



Solenoid Mounts

The rail and solenoid mounts feature recesses for M3 hex nuts. The slotted tabs on the solenoid mount allow for z-height adjustment. Holes in the center allow for wire and cable pass-through.

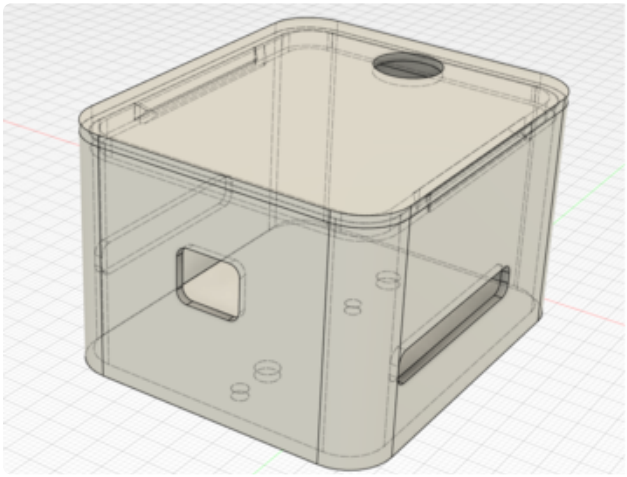
Additional Parts

In addition to the solenoid mounts and legs, there are few other 3D printed parts available:

- itsyBitsyXyloCaseBody.stl
- itsyBitsyXyloCaseTop.stl
- permaProtoBackPlate.stl
- doubleBrace.stl
- 3-holeCouplingPlate.stl
- singleBrace.stl

robot_xylophone_STLs.zip

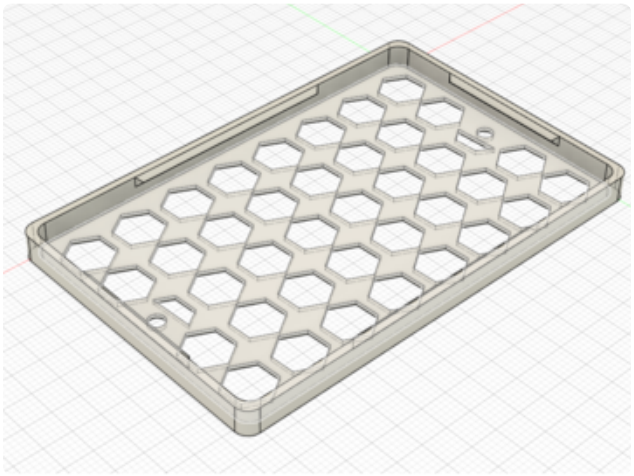
https://adafru.it/-ye



ItsyBitsy Case

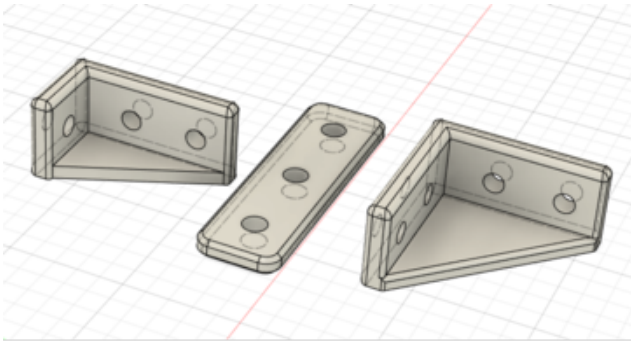
The ItsyBitsy case is a snap fit case and the design is parametric. It allows for the ItsyBitsy's 1/4 Perma-Proto to be mounted at the bottom with stand-offs. Additionally, there is a hole for a USB power cable and slots on both sides for the power and I2C wires to run from the ItsyBitsy to the other Perma-Proto boards on the xylophone. Finally, a power switch is mounted at the top and the lid has a hole for the switch to mount to.

The case mounts to the 20x20 aluminum extrusion using M4 screws and T-Nuts.



Perma-Proto Back Plate

The Perma-Proto back plate snap fits onto the 1/2 Perma-Proto boards. It allows for the solder joints to be protected from any damage or shorts. The Perma-Proto's mounting holes are included on the back plate so that they can mount to the xylophone.



Double Corner Brace, Single Corner Brace and 3 Hole Coupling Plate

These parts are CAD recreations of the original aluminum versions, with the exception of the single corner brace which is custom. You can use either the 3D printed versions or the aluminum versions.

The single corner brace is used to help support the ItsyBitsy case.

CircuitPython for ItsyBitsy nRF52840 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.

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

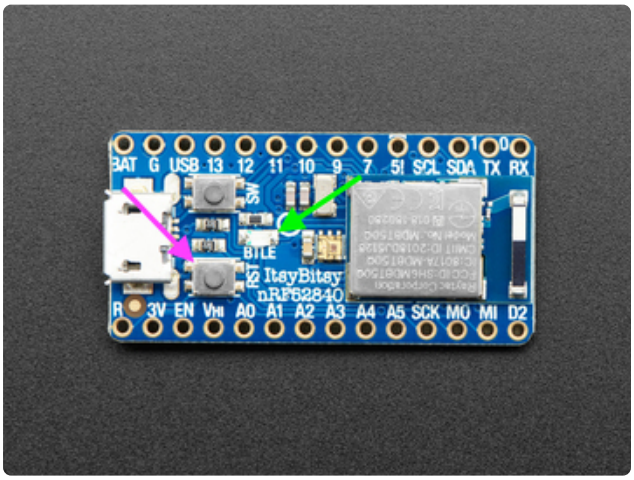
https://adafru.it/Ie7

Further Information

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

Click the link above and download the latest UF2 file.

Download and save it to your desktop (or wherever is handy).



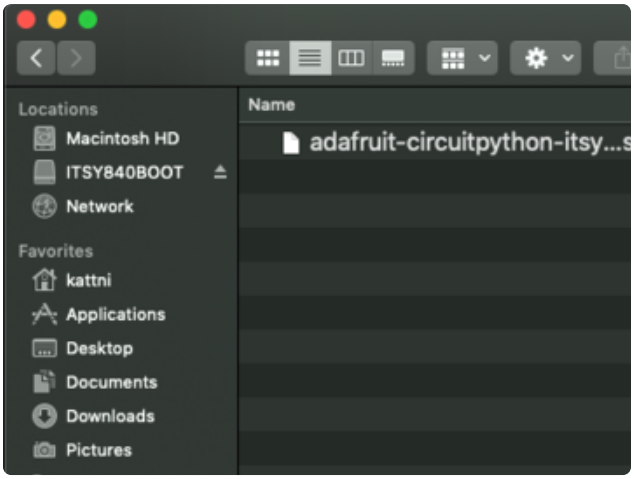
Plug your Itsy nRF52840 into your computer using a known-good USB cable.

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.

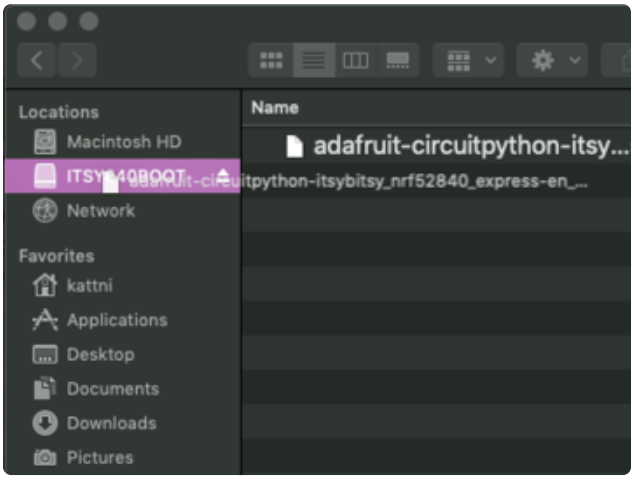
In the image, the **Reset** button is indicated by the magenta arrow, and the **BTLE status LED** is indicated by the green arrow.

Double-click the **Reset** button on your board (magenta arrow), and you will see the **BTLE LED** (green arrow) will pulse quickly then slowly blue. If the DotStar LED turns red, check the USB cable, try another USB port, etc.

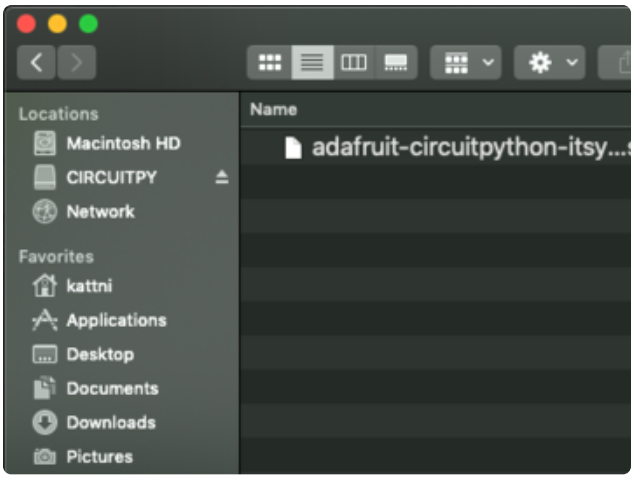
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



You will see a new disk drive appear called **ITSY840BOOT**.



Drag the **adafruit_circuitpython_etc.uf2** file to **ITSY840BOOT.**



The LED will flash. Then, the **ITSY840BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

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

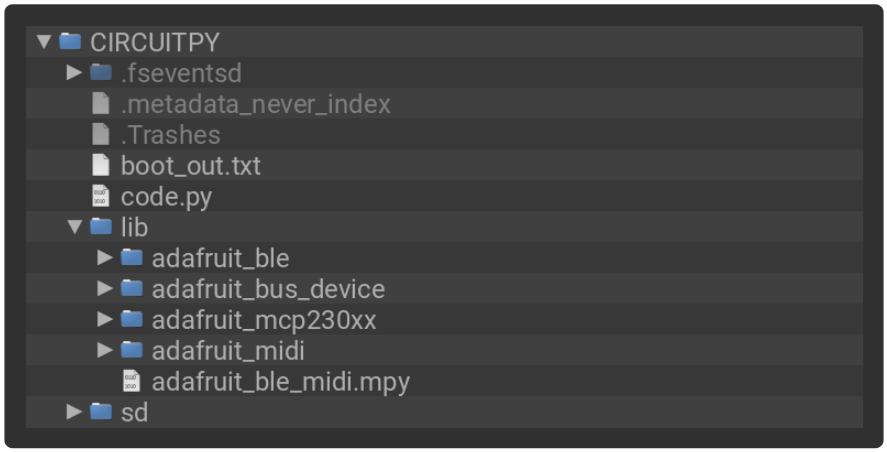
Coding the BLE MIDI Robot Xylophone

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory

BLE_MIDI_ROBOT_Xylophone/ and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:




```
# SPDX-FileCopyrightText: 2020 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import busio
from adafruit_mcp230xx.mcp23017 import MCP23017
from digitalio import Direction
import adafruit_ble
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
import adafruit_ble_midi

# These import auto-register the message type with the MIDI machinery.
# pylint: disable=unused-import
import adafruit_midi
from adafruit_midi.control_change import ControlChange
from adafruit_midi.midi_message import MIDIUnknownEvent
from adafruit_midi.note_off import NoteOff
from adafruit_midi.note_on import NoteOn
from adafruit_midi.pitch_bend import PitchBend

#  i2c setup
i2c = busio.I2C(board.SCL, board.SDA)

#  i2c addresses for muxes
mcp1 = MCP23017(i2c, address=0x20)
mcp2 = MCP23017(i2c, address=0x21)

#  1st solenoid array, corresponds with 1st mux
noids0 = []

for pin in range(16):
    noids0.append(mcp1.get_pin(pin))
for n in noids0:
    n.direction = Direction.OUTPUT

#  2nd solenoid array, corresponds with 2nd mux
noids1 = []

for pin in range(16):
    noids1.append(mcp2.get_pin(pin))
```

```
for n in noids1:
    n.direction = Direction.OUTPUT

#  MIDI note arrays. notes0 = noids0; notes1 = noids1
notes0 = [55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70]
notes1 = [71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86]

#  setup MIDI BLE service
midi_service = adafruit_ble_midi.MIDIService()
advertisement = ProvideServicesAdvertisement(midi_service)

#  BLE connection setup
ble = adafruit_ble.BLERadio()
if ble.connected:
    for c in ble.connections:
        c.disconnect()

#  MIDI in setup
midi = adafruit_midi.MIDI(midi_in=midi_service, in_channel=0)

#  start BLE advertising
print("advertising")
ble.start_advertising(advertisement)

#  delay for solenoids
speed = 0.01

while True:

    #  waiting for BLE connection
    print("Waiting for connection")
    while not ble.connected:
        pass
    print("Connected")
    #  delay after connection established
    time.sleep(1.0)

    while ble.connected:

        #  msg holds MIDI messages
        msg = midi.receive()

        for i in range(16):
            #  states for solenoid on/off
            #  noid0 = mux1
            #  noid1 = mux2
            noid0_output = noids0[i]
            noid1_output = noids1[i]

            #  states for MIDI note recieved
            #  notes0 = mux1
            #  notes1 = mux2
            notes0_played = notes0[i]
            notes1_played = notes1[i]

            #  if NoteOn msg comes in and the MIDI note # matches with predefined
notes:
            if isinstance(msg, NoteOn) and msg.note is notes0_played:
                print(time.monotonic(), msg.note)

                #  solenoid is triggered
                noid0_output.value = True
                #  quick delay
                time.sleep(speed)
                #  solenoid retracts
                noid0_output.value = False

            #  identical to above if statement but for mux2
            if isinstance(msg, NoteOn) and msg.note is notes1_played:
```

```
                print(time.monotonic(), msg.note)

                noid1_output.value = True

                time.sleep(speed)

                noid1_output.value = False

    #  if BLE disconnects try reconnecting
    print("Disconnected")
    print()
    ble.start_advertising(advertisement)
```

CircuitPython Code Walkthrough

Libraries

First, the CircuitPython libraries are imported.

```
import time
import board
import busio
from adafruit_mcp230xx.mcp23017 import MCP23017
from digitalio import Direction
import adafruit_ble
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
import adafruit_ble_midi

# These import auto-register the message type with the MIDI machinery.
# pylint: disable=unused-import
import adafruit_midi
from adafruit_midi.control_change import ControlChange
from adafruit_midi.midi_message import MIDIUnknownEvent
from adafruit_midi.note_off import NoteOff
from adafruit_midi.note_on import NoteOn
from adafruit_midi.pitch_bend import PitchBend
```

I2C Setup

Next, I2C and the two MCP23017's are setup.

```
#  i2c setup
i2c = busio.I2C(board.SCL, board.SDA)

#  i2c addresses for muxes
mcp1 = MCP23017(i2c, address=0x20)
mcp2 = MCP23017(i2c, address=0x21)
```

Solenoids

This is followed by the arrays (`noids0` and `noids1`) that will hold the solenoids that are connected to the two multiplexers. This allows them to be accessed as digital outputs.

```
#  1st solenoid array, corresponds with 1st mux
noids0 = []

for pin in range(16):
    noids0.append(mcp1.get_pin(pin))
for n in noids0:
    n.direction = Direction.OUTPUT

#  2nd solenoid array, corresponds with 2nd mux
noids1 = []

for pin in range(16):
    noids1.append(mcp2.get_pin(pin))
for n in noids1:
    n.direction = Direction.OUTPUT
```

MIDI Note Numbers

Following the solenoid arrays are two arrays for the MIDI note numbers. `notes0` will correspond with `noids0` and `notes1` will correspond with `noids1`. Later in the loop, these arrays will be used to match against incoming `NoteOn` MIDI messages.

```
#  MIDI note arrays. notes0 = noids0; notes1 = noids1
notes0 = [55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70]
notes1 = [71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86]
```

BLE Setup

The BLE MIDI service is setup, followed by the BLE connection.

```
#  setup MIDI BLE service
midi_service = adafruit_ble_midi.MIDIService()
advertisement = ProvideServicesAdvertisement(midi_service)

#  BLE connection setup
ble = adafruit_ble.BLERadio()
if ble.connected:
    for c in ble.connections:
        c.disconnect()
```

For more info on the BLE MIDI library, check it out on GitHub

https://adafru.it/Lke

MIDI Setup

`midi` is setup to be a MIDI-in device on MIDI channel 1. Channel 1 is defined as `0` in the CircuitPython MIDI library. A MIDI-in device is able to receive MIDI output from a digital audio workstation (DAW) or other MIDI communication method.

```
#  MIDI in setup
midi = adafruit_midi.MIDI(midi_in=midi_service, in_channel=0)
```

Begin Advertising

With everything setup, BLE can begin advertising for a connection.

```
#  start BLE advertising
print("advertising")
ble.start_advertising(advertisement)
```

Solenoid Delay

The last step before the loop is setting up `speed` to hold the delay between solenoids triggering and retracting. You can adjust this depending on your preferences.

```
#  delay for solenoids
speed = 0.01
```

The Loop

The loop begins with the initial BLE connection. Once a connection is established, "`Connected`" will print to the REPL. This is followed by a delay that helps to stabilize the BLE MIDI communications before everything begins.

```
while True:

    #  waiting for BLE connection
    print("Waiting for connection")
    while not ble.connected:
        pass
    print("Connected")
    #  delay after connection established
    time.sleep(1.0)
```

MIDI Messages

Once BLE is connected, `msg` is setup to hold the incoming MIDI messages.

```
while ble.connected:

        # msg holds MIDI messages
        msg = midi.receive()
```

Arrays of Arrays

The `for` statement allows for the ItsyBitsy to identify if an individual MIDI note number has been received and control individual solenoids attached to the multiplexers. `notes0_played` and `notes1_played` will handle the MIDI note numbers. `noid0_output` and `noid1_output` will handle the multiplexer outputs.

```
for i in range(16):
            # states for solenoid on/off
            # noid0 = mux1
            # noid1 = mux2
            noid0_output = noids0[i]
            noid1_output = noids1[i]

            # states for MIDI note recieved
            # notes0 = mux1
            # notes1 = mux2
            notes0_played = notes0[i]
            notes1_played = notes1[i]
```

Let the Solenoids Play!

The following `if` statements, nested in the previous `for` statement, are where the action is. The code is looking for a `NoteOn` message that contains one of the MIDI notes in either `notes0_played` or `notes1_played`. If there's a match, then the solenoid in the corresponding array index will trigger and retract with the predefined speed acting as the delay.

```
# if NoteOn msg comes in and the MIDI note
    # matches with predefined notes:
    if isinstance(msg, NoteOn) and msg.note is notes0_played:
        print(time.monotonic(), msg.note)

        # solenoid is triggered
        noid0_output.value = True
        # quick delay
        time.sleep(speed)
        # solenoid retracts
        noid0_output.value = False

    # identical to above if statement but for mux2
    if isinstance(msg, NoteOn) and msg.note is notes1_played:
```

```
        print(time.monotonic(), msg.note)
        noid1_output.value = True
        time.sleep(speed)
        noid1_output.value = False
```

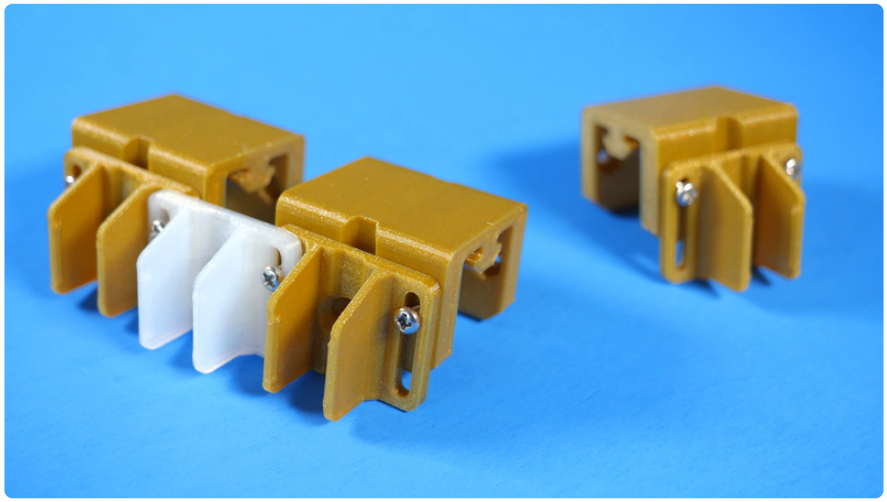
Why is it only looking for a `NoteOn` message? The way that mallet instruments work is that they have to be struck quickly in order for the note to resonate properly. If the code were waiting for a `NoteOff` message, then for longer note values the solenoid may not retract quickly enough to sound the note. That's why everything is relying on that initial `NoteOn` message.

Reconnect BLE

The code ends by checking if BLE disconnects. If it does, then BLE will begin advertising again to reconnect and print to the REPL that it has disconnected.

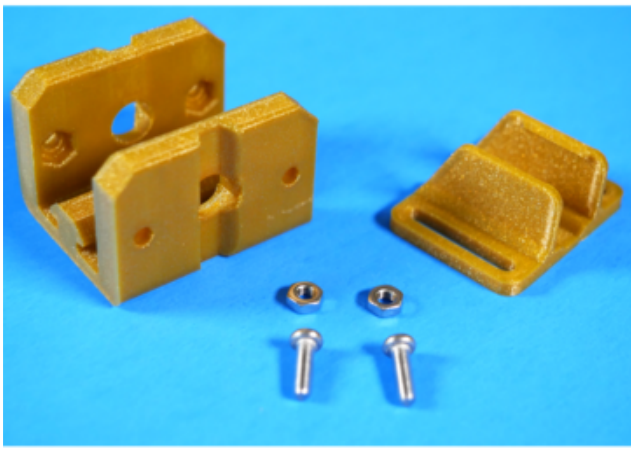
```
#  if BLE disconnects try reconnecting
    print("Disconnected")
    print()
    ble.start_advertising(advertisement)
```

Solenoid Mount Assembly



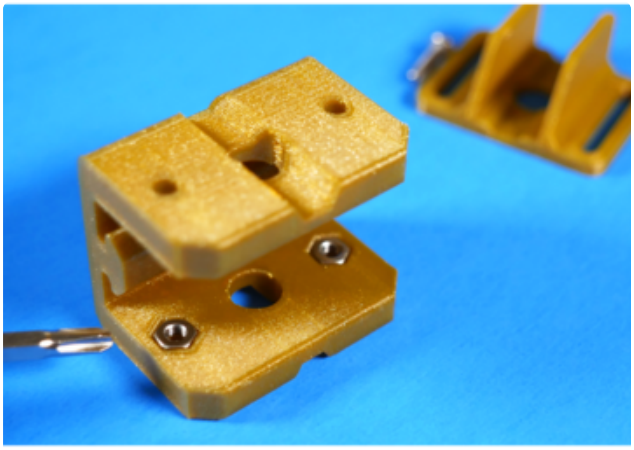
The solenoids sit snugly in mounts that slide onto the 20x20 aluminum extrusion pieces. They are assembled using M3 screws and nuts. The sharps and flat notes, at the top of the xylophone, will have a single mount for each note. The natural notes, at the bottom of the xylophone, will use two rail mounts and three solenoid mounts together for proper spacing.

Mount Assembly for Sharps/Flats

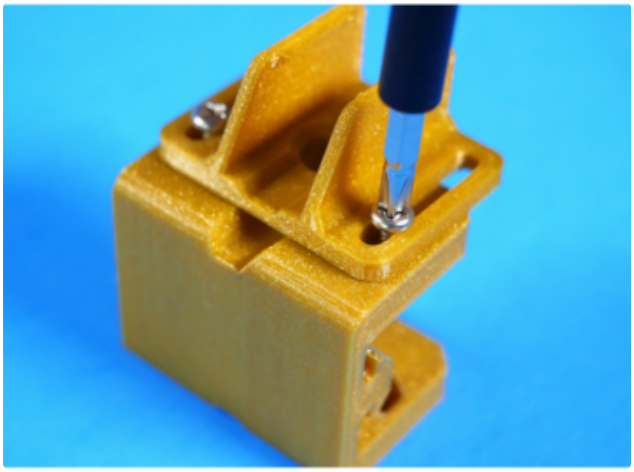


You'll need:

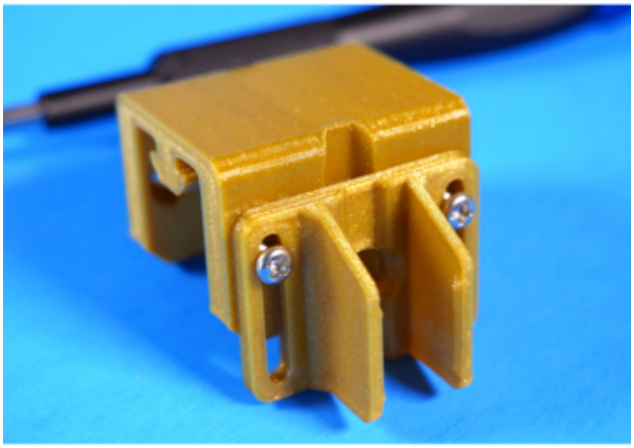
One rail mount
One solenoid mount
Two M3 nuts
Two M3x10 screws



Turn the rail mount on its side and place the two M3 nuts into their slots.

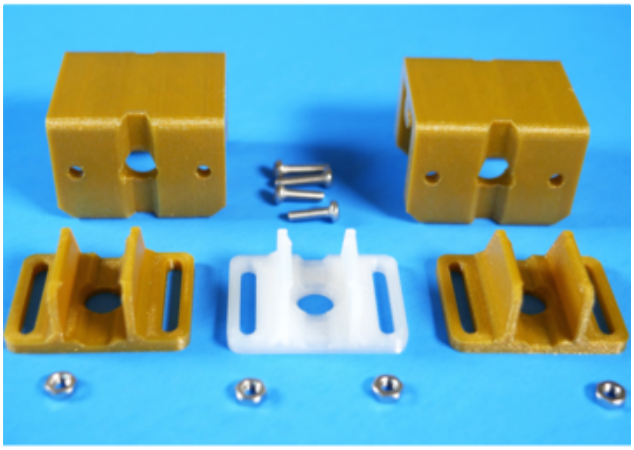


Attach the solenoid mount to the rail mount by screwing the two M3 screws into the slotted nuts.



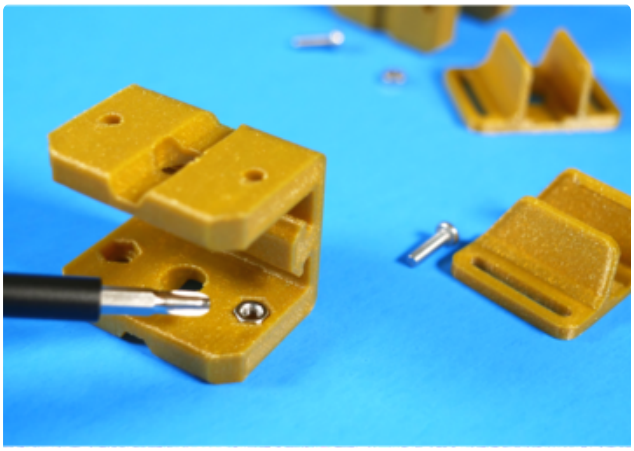
The fully assembled single mount.
Assemble 12 total.

Triple Mount for Natural Notes

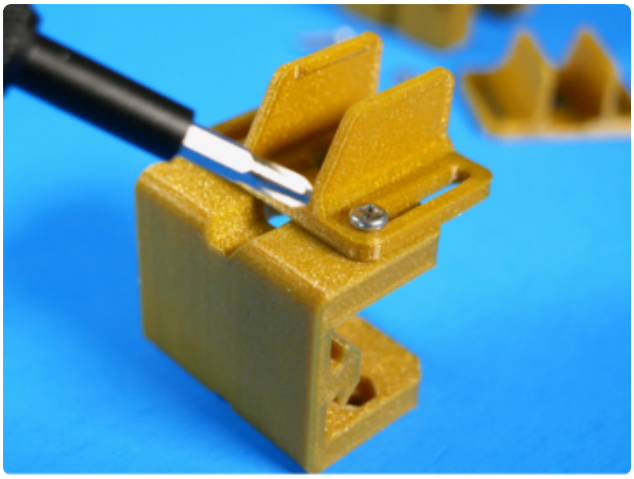


You'll need:

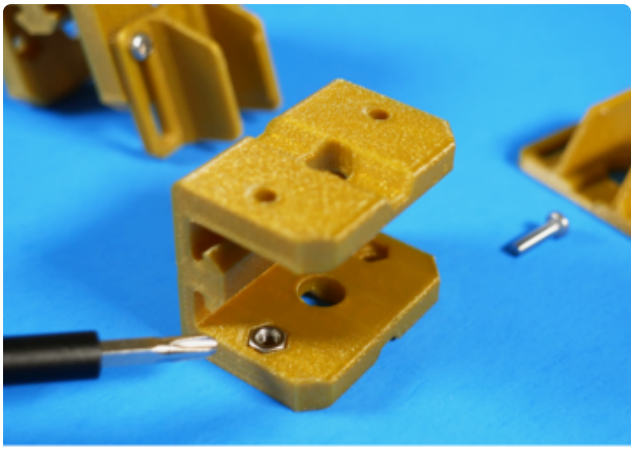
Two rail mounts
Three solenoid mounts
Four M3 nuts
Four M3x10 screws



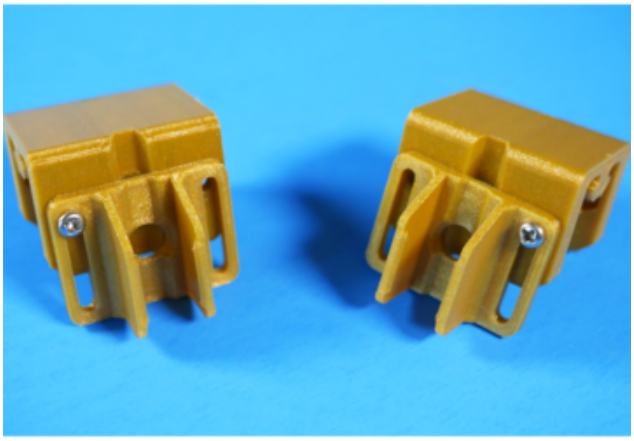
Take the first rail mount and place one M3 nut in the mount's slot on the right.



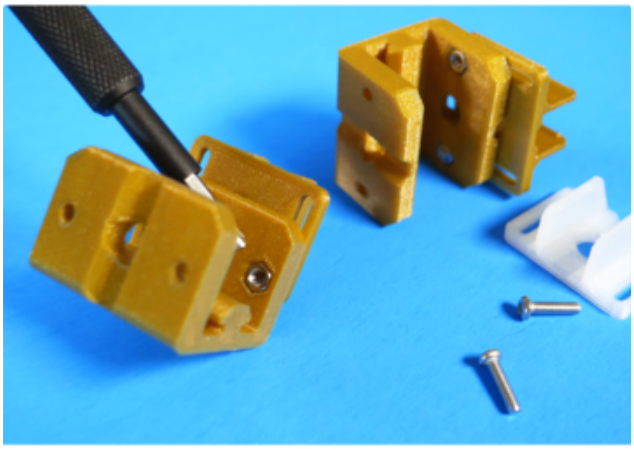
Attach one solenoid mount to the first rail mount using one M3 screw.



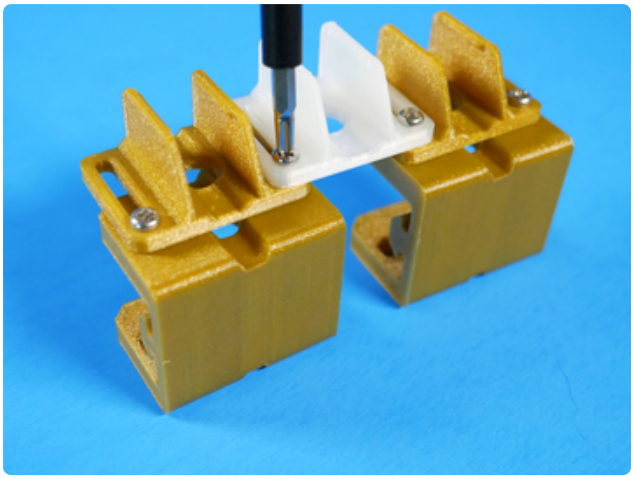
With the second rail mount, place one M3 nut in the mount's left-hand slot.



Attach one solenoid mount to the second rail mount using one M3 screw. This will leave an empty space for the third solenoid mount to be attached.

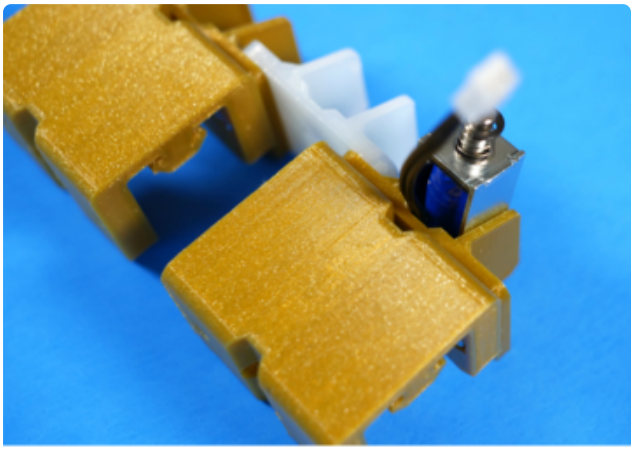


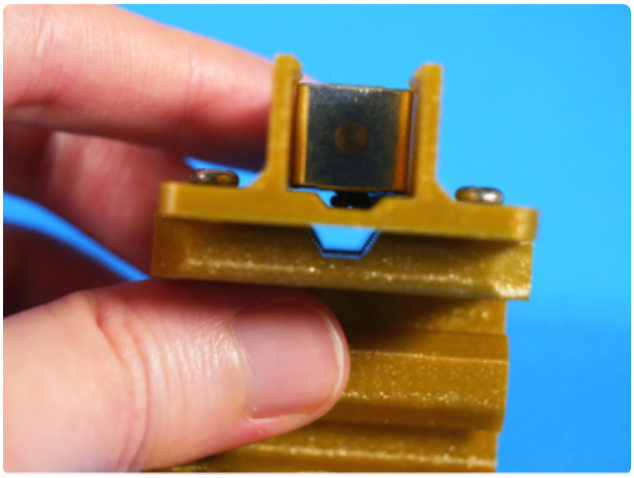
Insert M3 nuts into the two remaining slots on both rail mounts.



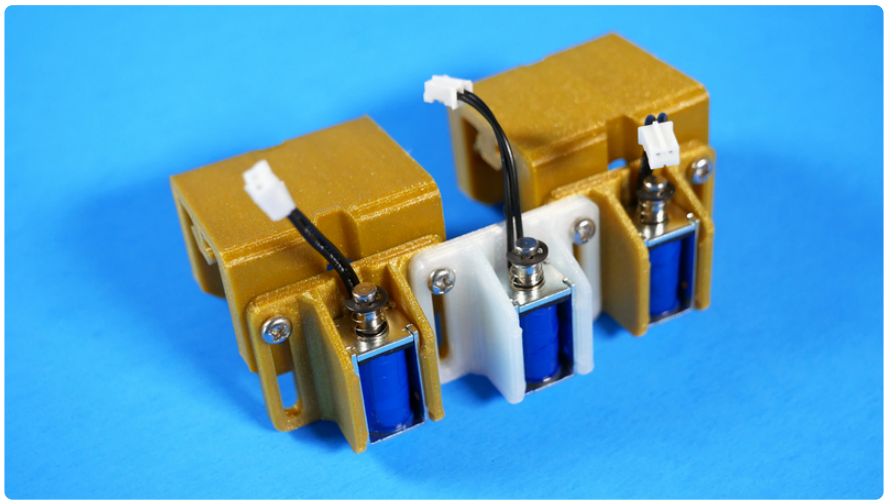
Attach the third solenoid mount with M3 screws so that it's sitting on top of the original solenoid mounts. Assemble 6 total.

Insert the Solenoids



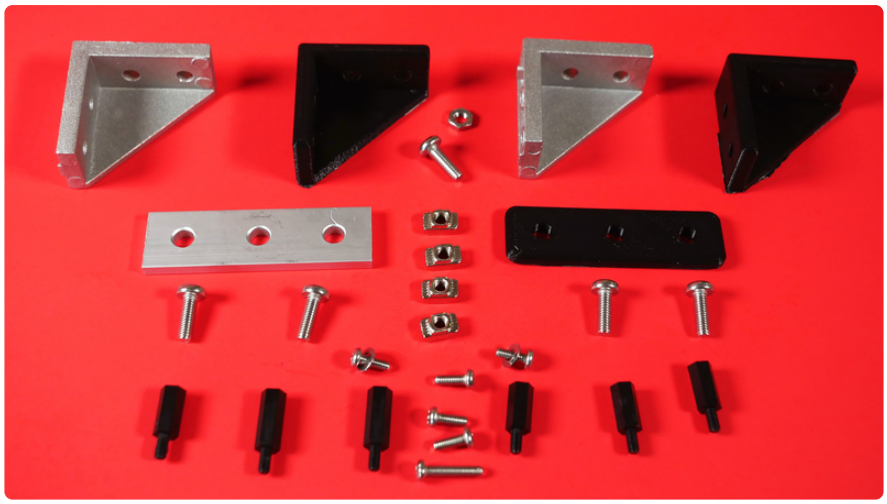


Insert the solenoids into the solenoid mounts so that their wires are running up the cable channel on the mount.



Perma-Proto Board Mount Assembly

The circuit for the xylophone lives primarily on three proto boards. These boards are mounted vertically on the 20x20 aluminum extrusions with braces and coupling plates. You can use either aluminum versions of these pieces or 3D print them.

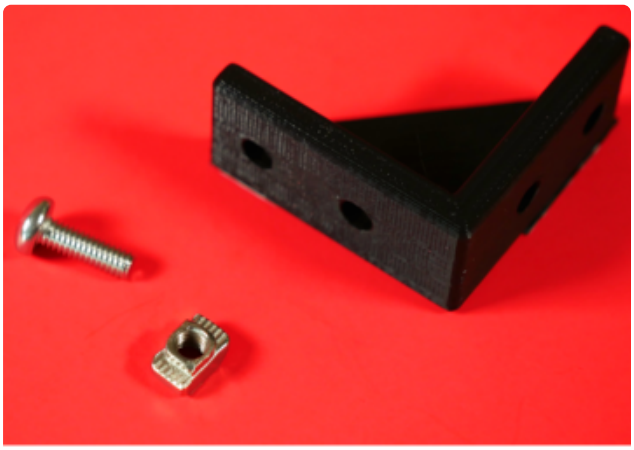


You'll need:

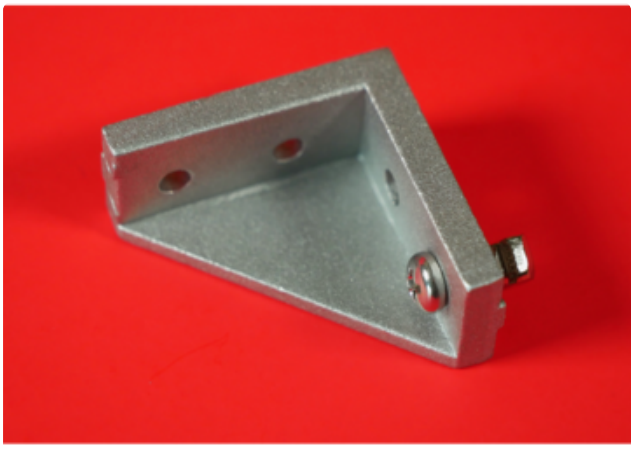
- Four double corner braces
- Two 3 hole coupling plates
- Five M4 screws
- Five M3x10 screws
- One M3x16 screw
- One M4 nut
- Four slim T-Nuts
- Six M3 standoffs
- Six M3 washers (optional)

The washers are helpful if you're using the aluminum versions of the parts, since the holes can be slightly over-sized for the M3 hardware.

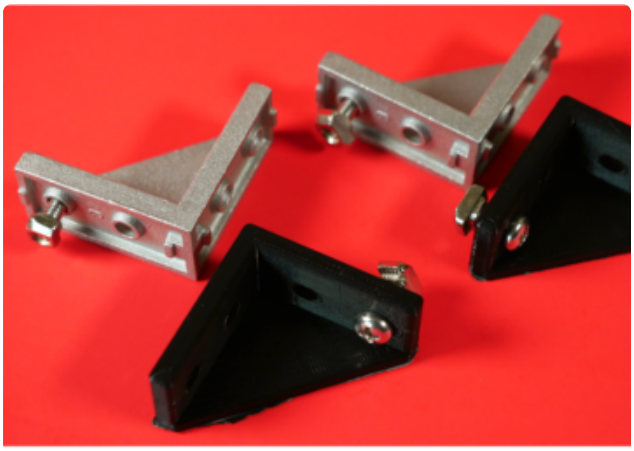
Mount Prep



Take the four double corner braces, four M4 screws and four slim T-Nuts.



Attach a t-nut to one of the far holes on each double corner brace with an M4 screw.

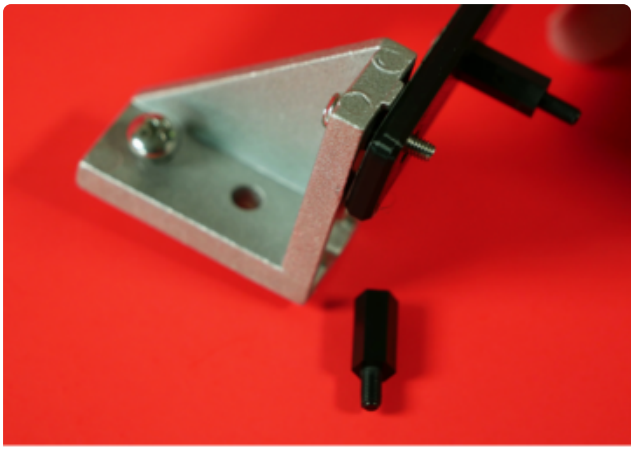


These will eventually mount to the 20x20 extrusions.

Mount 1 Assembly

Attach an M3 standoff in the center hole of the coupling plate using an M3 screw.

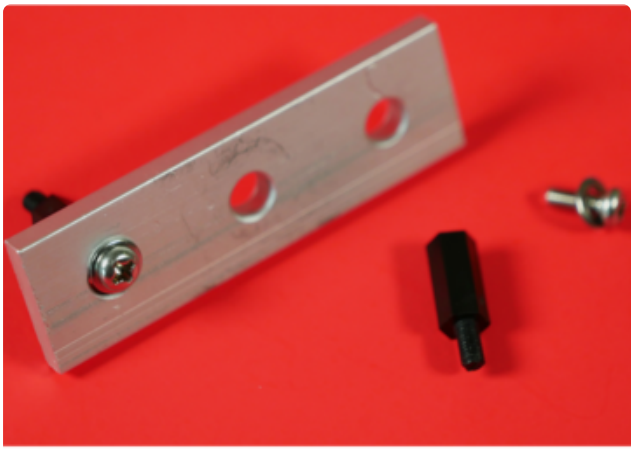
Bring in one of the assembled double corner braces, one M3 standoff and the M3x16 screw.



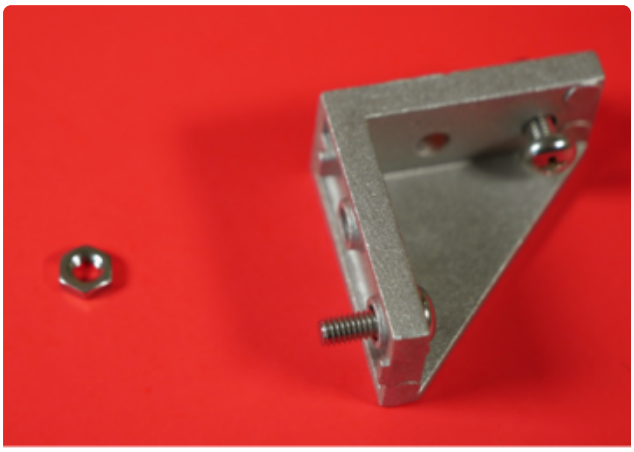
Using the opposite far hole on the double corner brace, attach the 3 hole coupling plate with the M3x16 through the coupling plate's left hole. Attach the standoff to the screw.

The completed assembly.

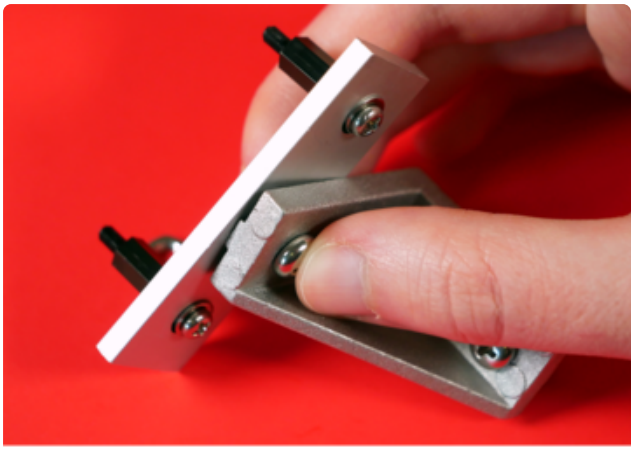
Mount 2 Assembly

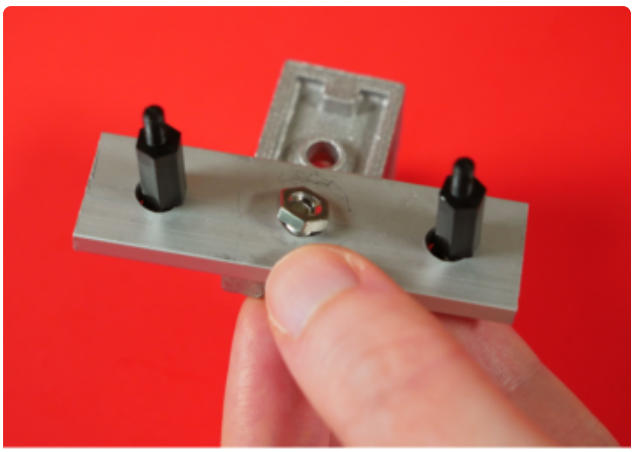


Attach M3 standoffs to the 3 hole coupling plate's left and right holes with M3 screws.



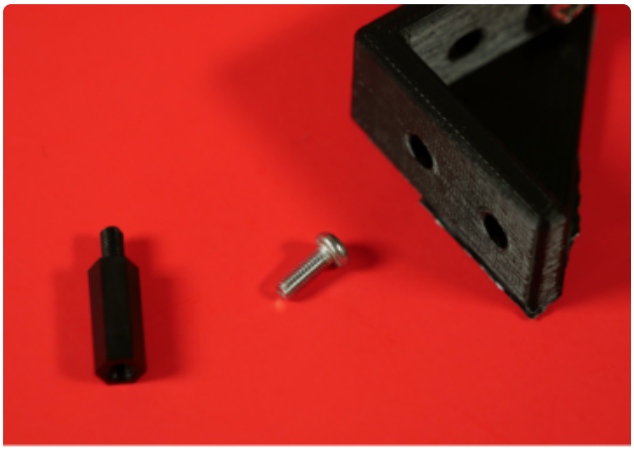
Using one of the assembled double corner braces, insert an M4 screw into the open far hole.





Attach the coupling plate to the corner brace using the M4 screw and the coupling plate's center hole. Secure it with an M4 nut.

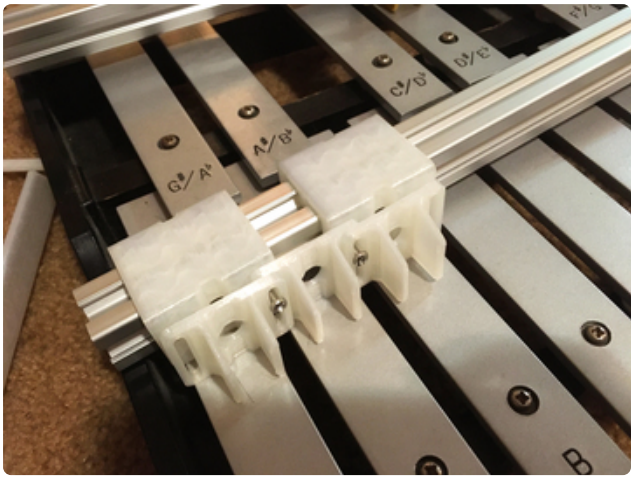
Mount 3 and 4 Assembly



With the two remaining assembled corner braces, attach an M3 standoff with an M3 screw in the open far hole.

When you're finished, the four mounts should look like the four pictured below.

Mounting to the Aluminum Extrusions



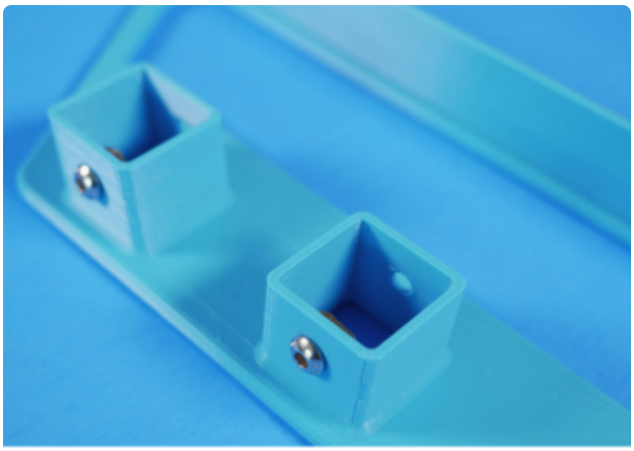
Adding the Solenoid Mounts

The 20x20 aluminum extrusions will sit towards the back of each set of notes so that the solenoids will strike towards the middle of the keys for the best possible tone. Slide the assembled solenoid mounts onto the rails.

Extrusion Feet

The two 3D printed feet allow for the extrusions to sit at the correct angle across the keys and at the right height as well.

To assemble, take two slim T-Nuts and two M4 screws. Screw them loosely together through the top holes on both legs.

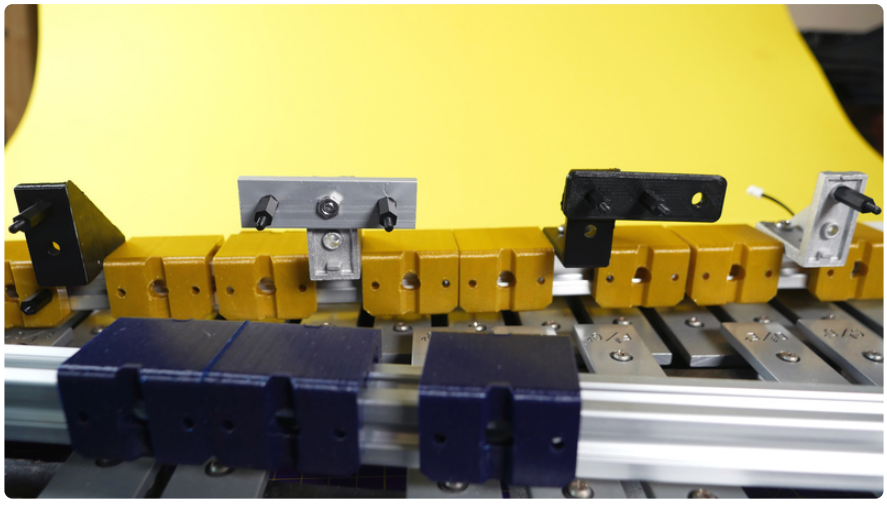


After the solenoid mounts are on the aluminum extrusions, you can slide the legs on either side of the extrusions and tighten the M4 screws so that they're secured using the T-Nuts.

Attach the Proto Board Mounts

In the center of the 2nd, 3rd, 4th and 5th triple solenoid mounts, place the slim T-Nuts from proto board mounts. Turn them so that they're locked in.

Attach each mount with the M4 screw to the T-Nut.

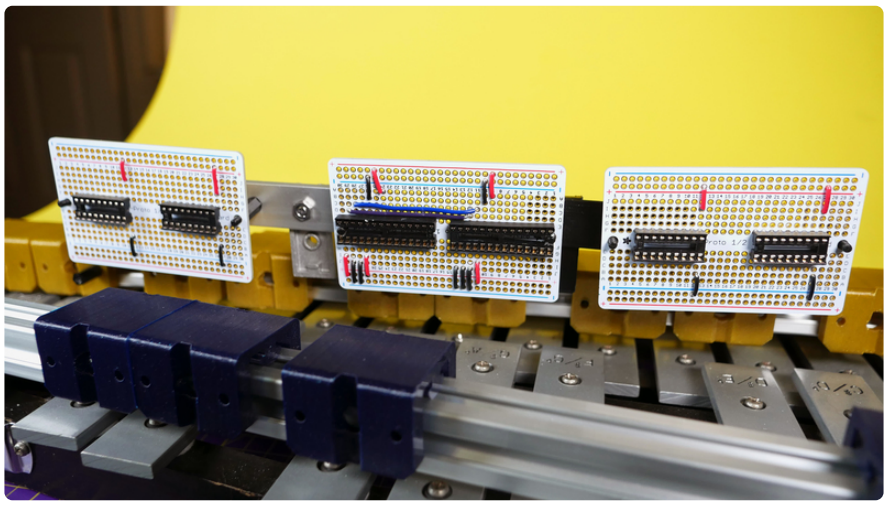


Wiring

The electronics for the xylophone are setup identically to the Fritzing circuit diagram on the Electronics (https://adafru.it/Lkf) page. The ItsyBitsy will live on a 1/4 Perma-Proto and then three 1/2 Perma-Proto boards will be used for the MCP23017's and the four ULN2308's.

One board will have the two MCP32017's and then the two remaining boards will have two ULN2308's each. To begin wiring, solder the IC sockets onto the boards. Then, run the wires for power, ground and I2C.

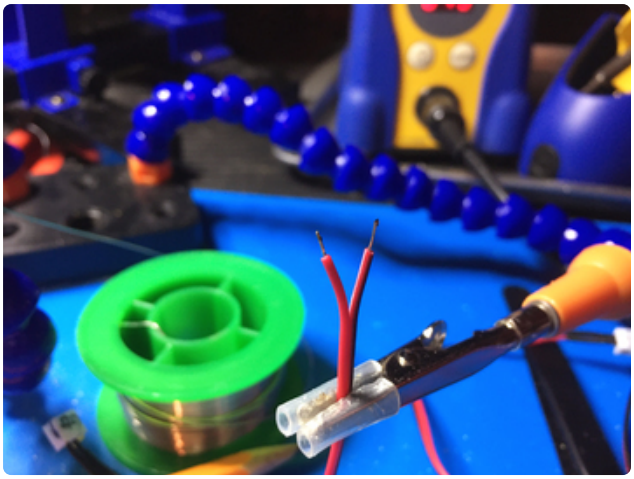
Mount the boards to the xylophone. The screw ends of the M3 stand-offs are facing out so that they can slot into the boards' mounting holes and then be secured with some M3 nuts. By doing this, you'll know exactly how long the wires have to be between the multiplexers and drivers, followed by the driver boards and the solenoids.

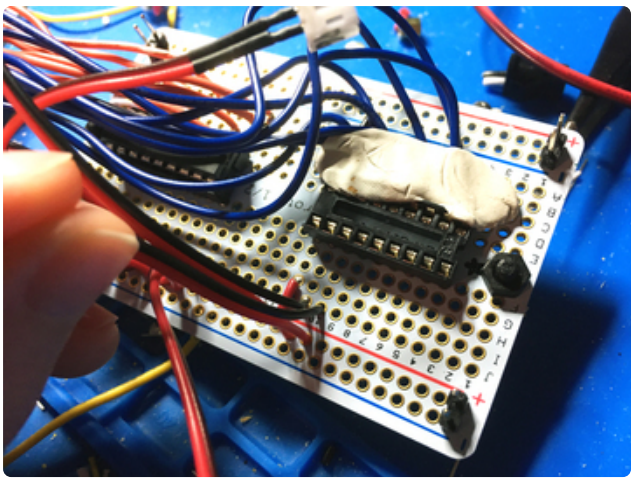


All of the outputs from the multiplexers will connect to the inputs of the drivers, which should be oriented to face towards the bottom of the xylophone. This way, the outputs of the drivers will be at the top and easier to access for the solenoids.

The solenoids will connect using a JST extension cable. Cut them to length based on the location of the solenoid and the driver output.

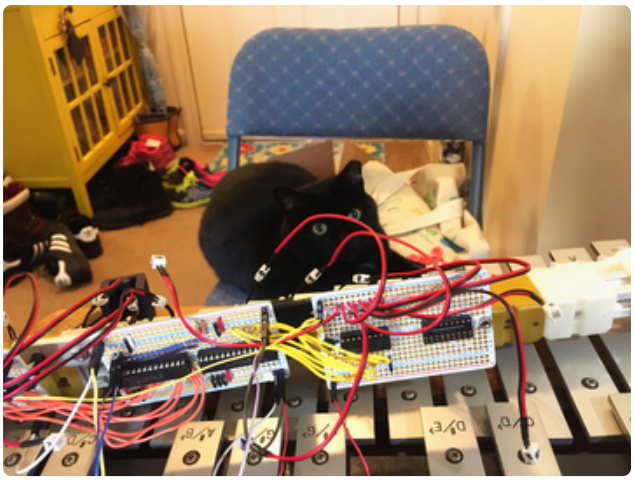
Don't get rid of the other connection end though! They'll come in handy for future projects. You can never have too many JST connectors.





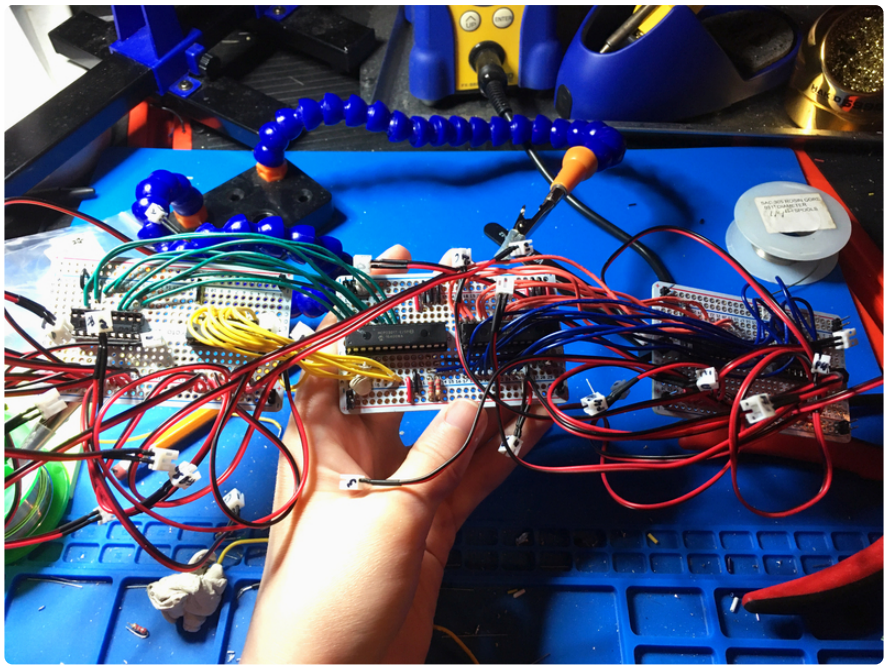
Strip back a few millimeters of wire on the JST extension and then tin them for soldering. Solder one wire to the driver's output and the second wire to the power rail, which will carry 5V. Solenoids do not have polarity, so it doesn't matter in this case which wire is connected to either soldering point.

Continue this for all of the solenoids.

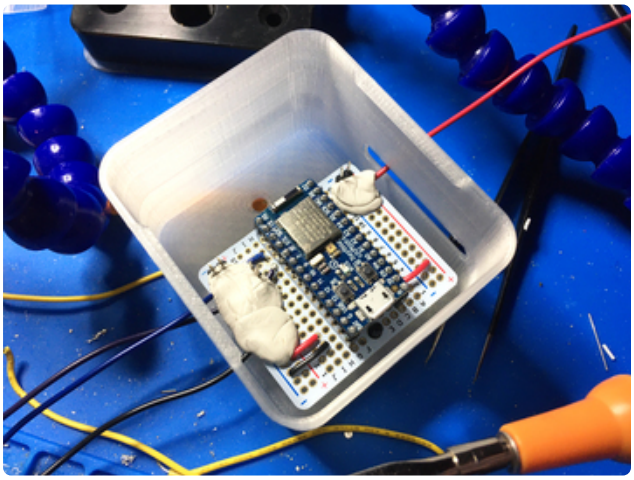


The process of wiring up the multiplexers to the drivers can be a little tedious. To avoid any major mistakes you can wire up one driver to each multiplexer to make sure that I2C is working and that both driver boards are outputting to the solenoids as expected.

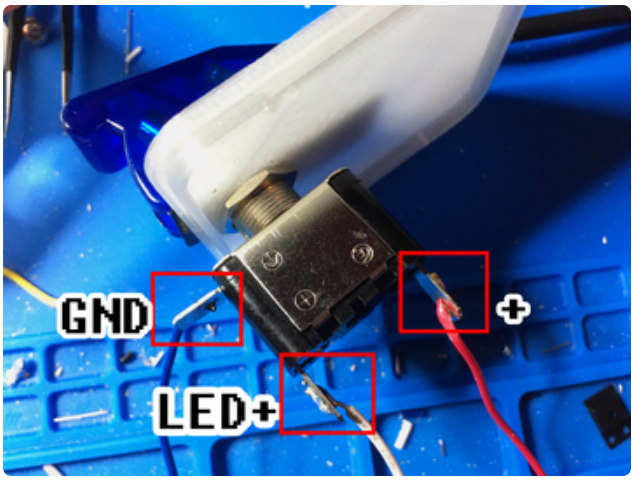
It also helps to have a cat inspect the circuit's progress.



Wiring the ItsyBitsy



The ItsyBitsy is attached to the 1/4 Perma-Proto with socket headers. Power, ground and I2C signals are run via long wires to the other Perma-Protos through the case's wiring slots at the bottom.



A toggle switch is in-line with the USB power signal from the ItsyBitsy that supplies the ULN2308's. This allows you to have control over the solenoids receiving power in case of a problem.

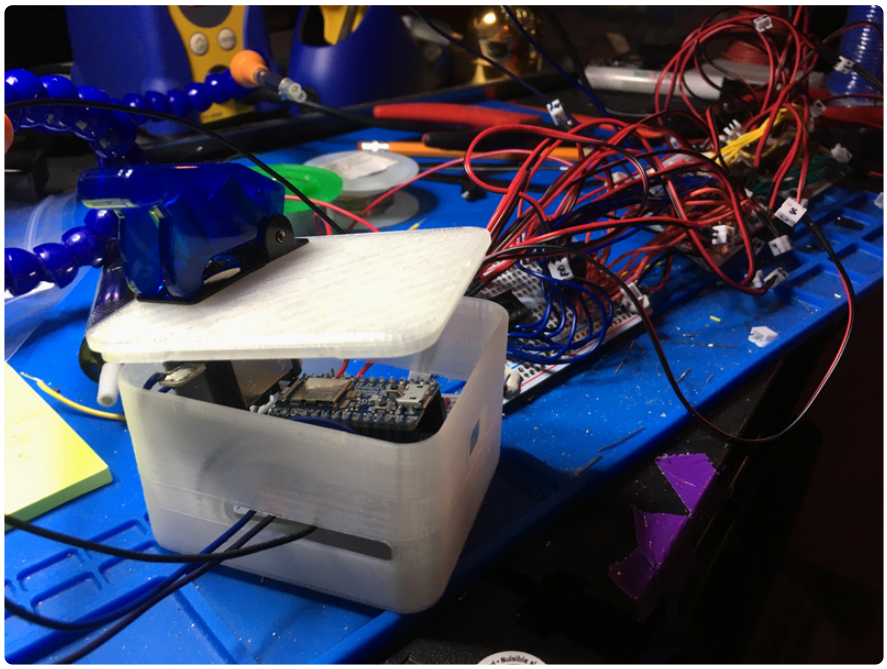
This particular toggle switch also has an LED at the top that lights up when power is engaged.

The toggle switch's **GND** is connected to the ItsyBitsy's **GND**
The toggle switch's + pin is connected to the ItsyBitsy's **USB** pin
The toggle switch's **headlamp** pin is connected to the ItsyBitsy's **USB** pin

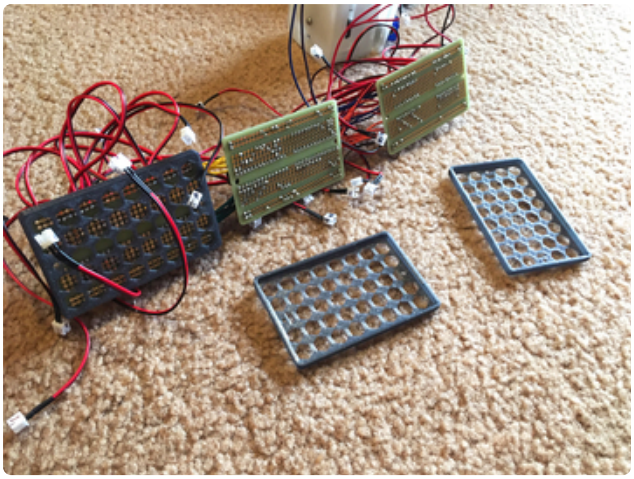
For more info on these toggle switches and their wiring options, check out this product page.

http://adafru.it/3306

With everything wired up, it's time for the final assembly.

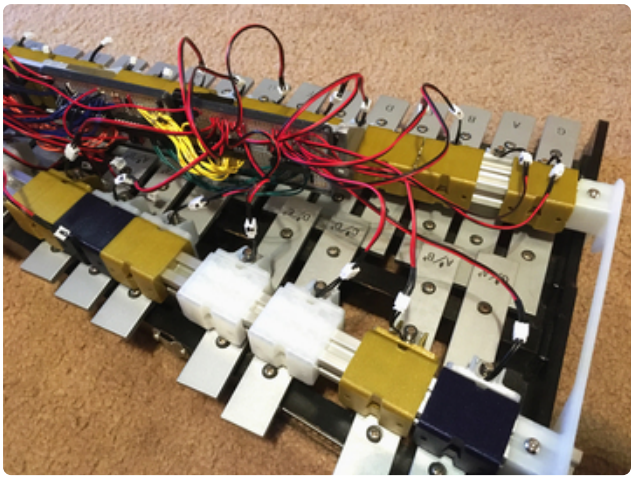


Final Assembly



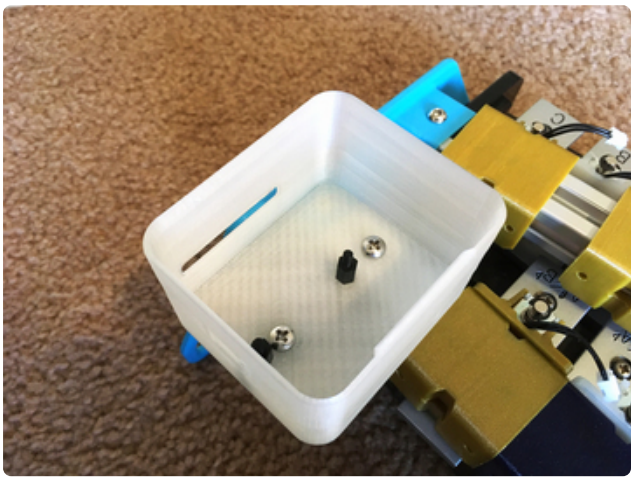
Perma-Proto Back Plate

Before mounting the Perma-Proto boards, snap on their 3D printed back plates. These will help to protect the solder points from damage and shorts.

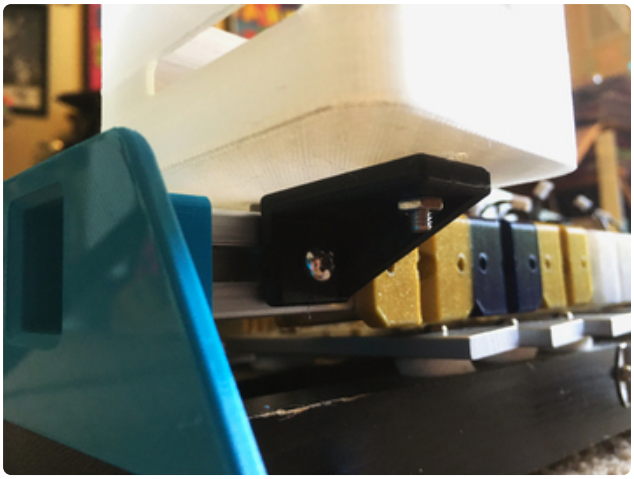


Mount the Perma-Proto boards and plug in the solenoid motors to their respective cables.

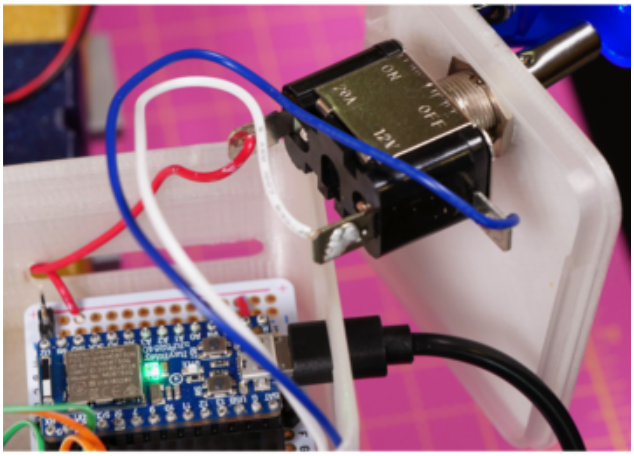
Mounting the ItsyBitsy's Case



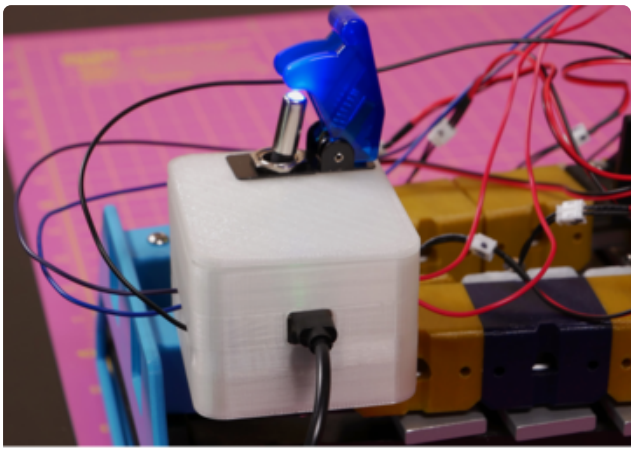
Attach two M3 stand-offs with two M3 screws into two of the holes in the bottom of the case as pictured. Then, with one M4 screw and a T-Nut, mount the case to the aluminum extrusion.



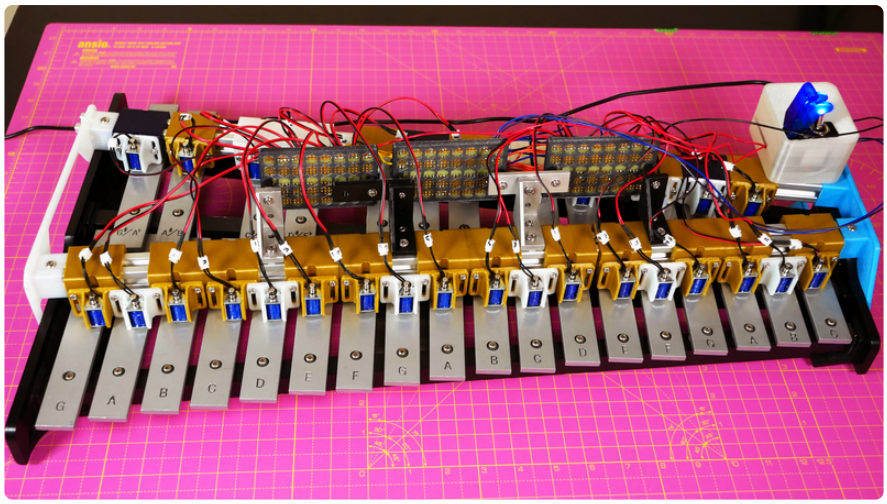
A 3D printed single corner brace is secured to the side of the aluminum extrusion using an M4 screw and a T-Nut. The remaining hole from the case is secured to the corner brace with an M4 screw and M4 nut.



The ItsyBitsy's 1/4 Perma-Proto can then be mounted to the M3 stand-offs with M3 nuts.



The ItsyBitsy is ready to be powered-up to run the BLE MIDI xylophone! For best results, use an external 5V USB power supply rated for at least 2A.

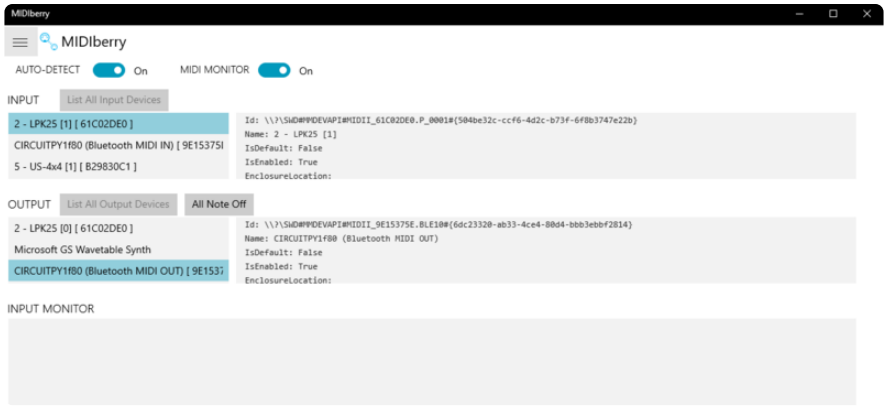


MIDI Setup and Usage

The process for using this xylophone with your DAW or MIDI controller will depending on your computer's operating system. BLE MIDI is supported natively in macOS. Apple has great instructions on how to easily connect your BLE MIDI peripherals here (https://adafru.it/Ksa).

Windows BLE MIDI Setup

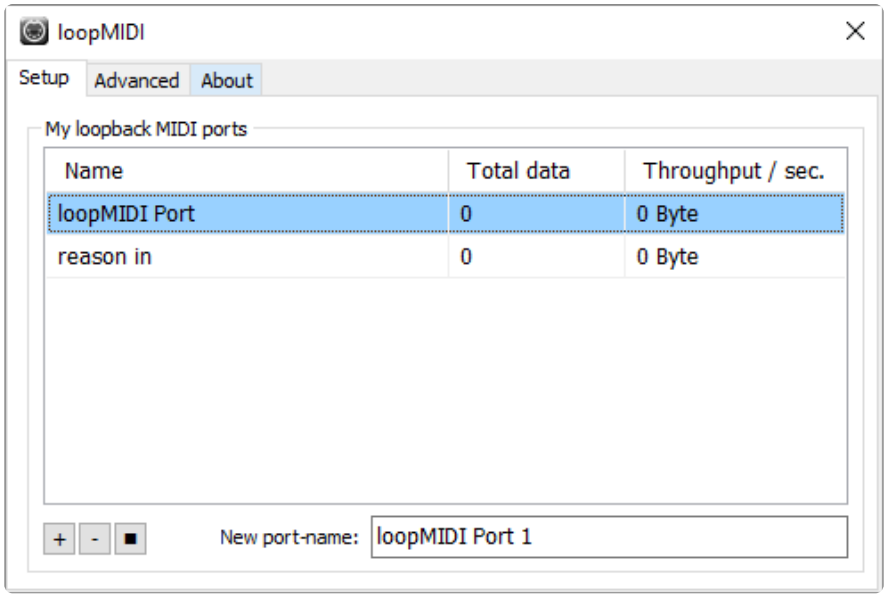
On Windows, there is a little more setup involved. In order to have your DAW or USB MIDI controller connect, you'll need a software bridge to connect the ItsyBitsy to your desired MIDI input. There are a few available, but MIDIberry (https://adafru.it/LkB) is a freeware option that is available through the Microsoft store. It allows you to select the input, output and monitor the MIDI messages being sent.

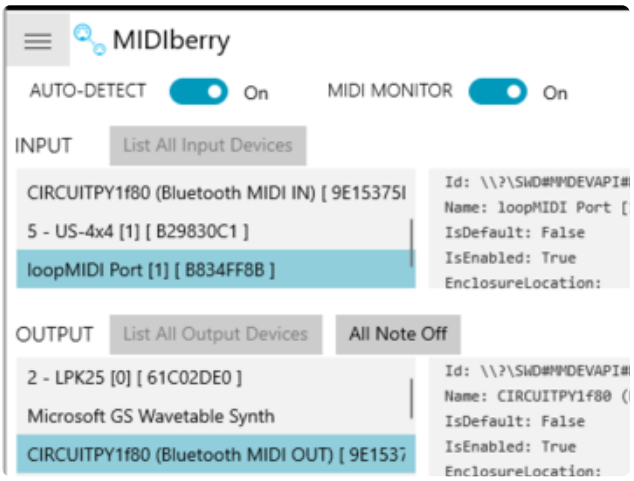


If you just want to play your xylophone live, then you can use MIDIberry on its own to control it. However, if you want to send MIDI out from a DAW for the xylophone to play for recording or live purposes, you'll want to setup a virtual MIDI loopback.

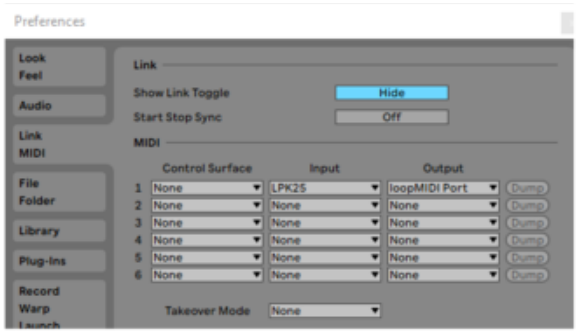
Most DAW's look for a MIDI hardware connection to assign MIDI input and outputs. By using a virtual MIDI loopback, you can trick the DAW into thinking you have different MIDI hardware devices plugged into your computer or even give your MIDI hardware extra functionality.

One of the most common solutions for this is the loopMIDI (https://adafru.it/LkC) project by Tobias Erichsen. It allows you to easily add virtual MIDI loopbacks on the fly.





In MIDIberry, you can define the virtual loopback as an input with the ItsyBitsy as an output. This means that there is a virtual cable connecting the two together, allowing the ItsyBitsy to receive MIDI data from a DAW over BLE.



After setting up your virtual loopback, you can define it as a MIDI out in your preferred DAW.

You can setup a MIDI track to send MIDI data out to the virtual loopback and your xylophone should begin playing along.

Playing the xylophone live with a MIDI keyboard is very fun and can allow you to play pieces that previously would've been impossible when limited to four mallets; especially on a bell kit/glockenspiel.

Taking in MIDI data from a DAW is really where things get interesting though, since you can record music live that previously would've been through a software synth. It could also make be a great addition to a live music setup whether it's for a traditional duet or for electro-acoustic music.