SNES Mouse to USB HID with CircuitPython

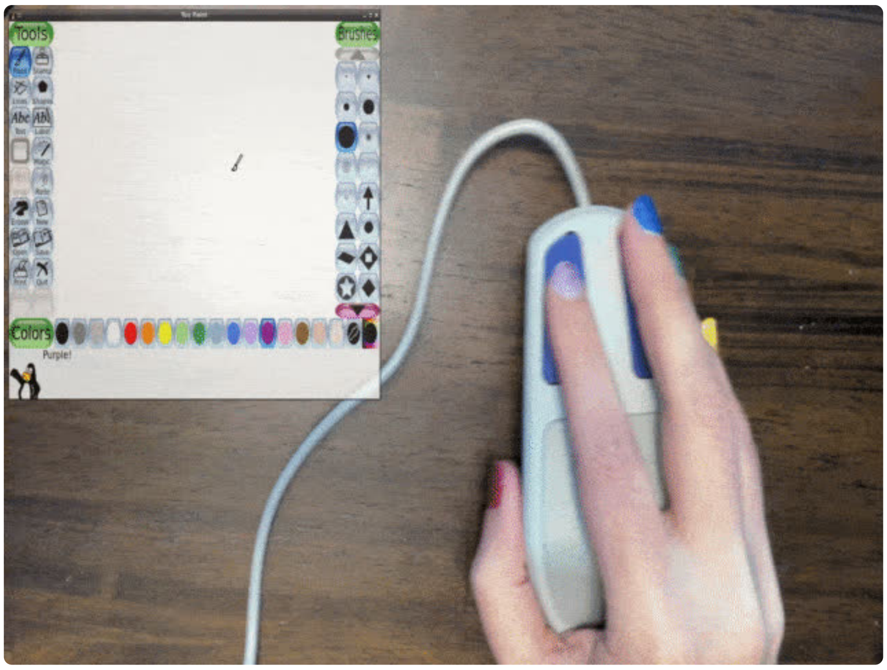
Created by Jeff Epler

https://learn.adafruit.com/snes-mouse-to-usb-hid-with-circuitpython

Last updated on 2024-06-03 03:47:21 PM EDT

Table of Contents

Overview



The Super Nintendo aka SNES was an iconic console of the 16-bit era. The Nintendo Mouse was originally developed as a bundled accessory for Mario Paint. It has two buttons and an SNES connector. Later, third party games would also use this mouse accessory.

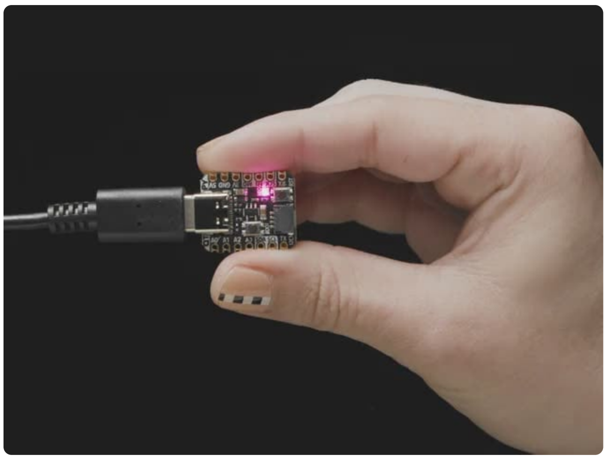
In this guide, I'll show how to connect the SNES mouse (actually, a modern third-party version known as the the Hyper Click Retro Style Mouse (https://adafru.it/18A6)) to a PC or Mac using CircuitPython to convert the original protocol into USB HID. As far as I know, this will also work with the original mouse, but as I don't own one, I haven't tested it.

Instead of using a difficult-to-source connector and custom PCB, this guide takes the approach of using a SNES controller extension cable which is soldered to the microcontroller board.

While this guide is written for the QT Py RP2040, the controller communicates using the standard SPI protocol, which means it's possible to adapt the code to just about any CircuitPython board simply by changing the pins that are used.

Once you've got the project wired up, it's also a great platform for connecting any other SNES compatible controller — just modify the source code according to the data the controller sends.

Parts



Adafruit QT Py RP2040

What a cutie pie! Or is it... a QT Py? This diminutive dev board comes with one of our new favorite chip, the RP2040. It's been made famous in the new

https://www.adafruit.com/product/4900

Pink and Purple Woven USB A to USB C Cable - 1 meter long

This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also made for USB C for our modernized breakout boards, Feathers, and...

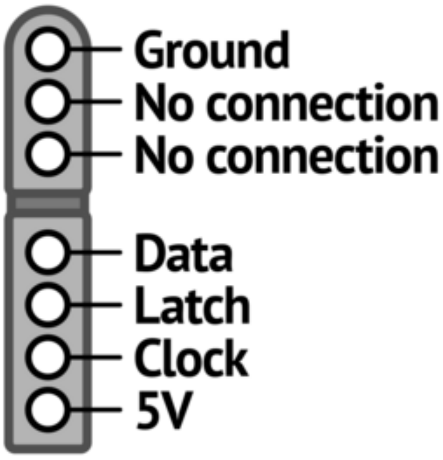
https://www.adafruit.com/product/5153

1 x SNES-compatible mouse Hyperkin Hyper Click Retro Style Mouse for Super NES	https://www.amazon.com/Hyperkin-Hyper-Click-Retro-super-nintendo/dp/B0734BCWS1
1 x SNES Controller Extension Cable 2Pack 6 ft/1.8m SNES Controller Extension Cable for Retro-Bit SNES Super Nintendo Entertainment System Console	https://www.amazon.com/gp/product/B075K4R86R

Assembly & Wiring

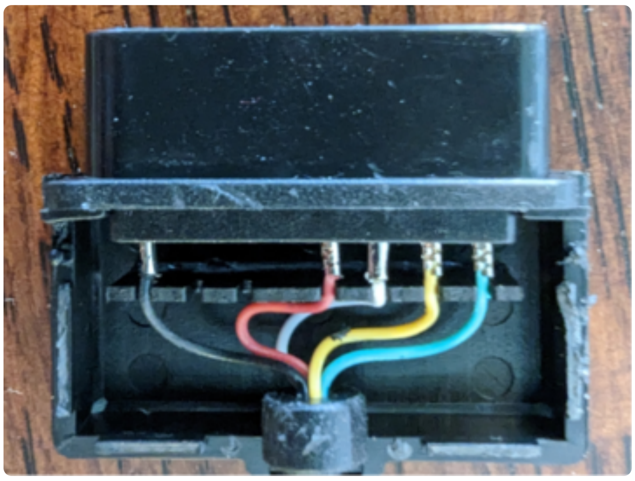
Take a SNES controller extension cable. Find the end of the cable that would plug into the console (it's the one that looks the same as the connector on the mouse). Using a spudger or other tool, pry the shell off the connector to access the wires (you can be destructive if you like, the shell does not need to be re-used).

Be careful prying the cover off - use of gloves and eye protection is recommended.



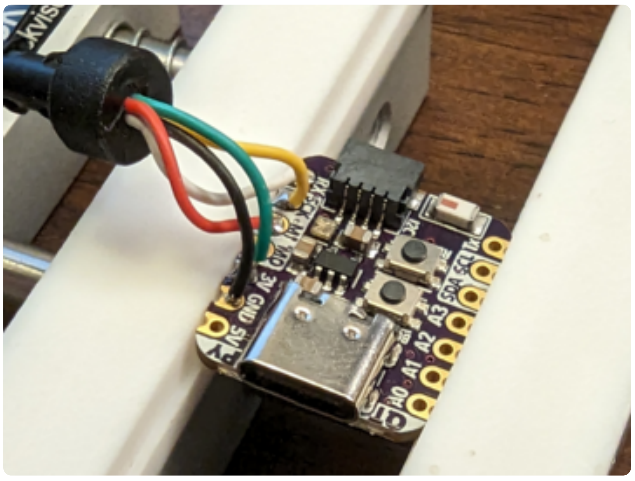
Note how each wire is connected, because there's not a standard color code. You'll need to note the color or wire marking for:

Ground (GND)
Data
Latch
Clock
VCC (5V)



Non-standard wire colors inside the author's SNES controller extension cable.

Round end of the connector is to the left in this particular view.



Now, clip the wires, strip them, stick them through the pads of the QT Py, and solder in place:

VCC to 3V
GND to GND
Clock to SCK
Data to MISO
Latch to RX
After soldering, clip any excess wire that remains.

While the original SNES controllers were designed to operate at 5V, the author's mouse was verified to work properly at 3.3V. This simplifies the circuit, because otherwise level shifting converters would be needed.

Install CircuitPython

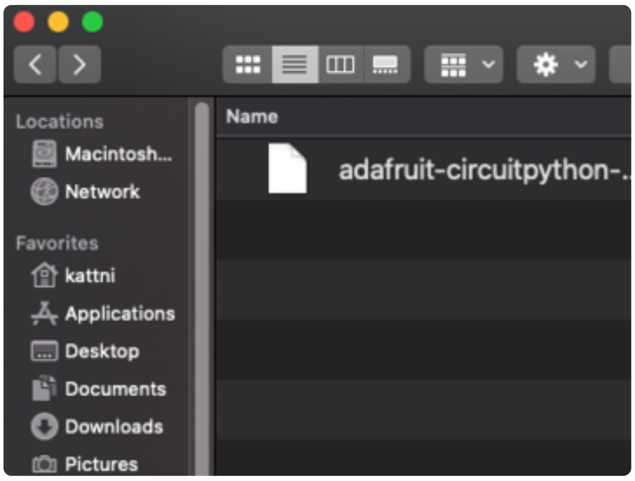
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

CircuitPython Quickstart

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

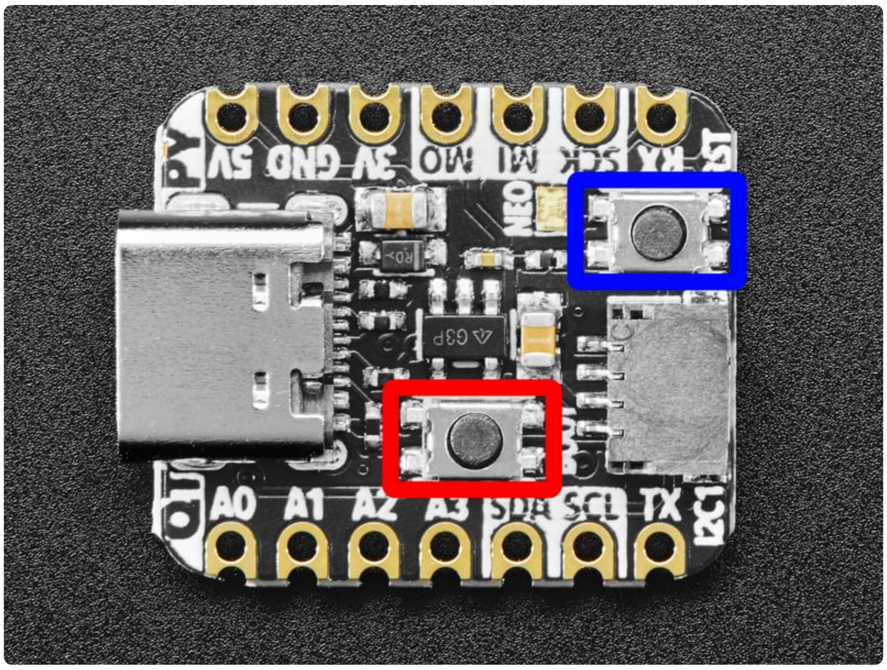
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/RLD



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

Save it wherever is convenient for you.

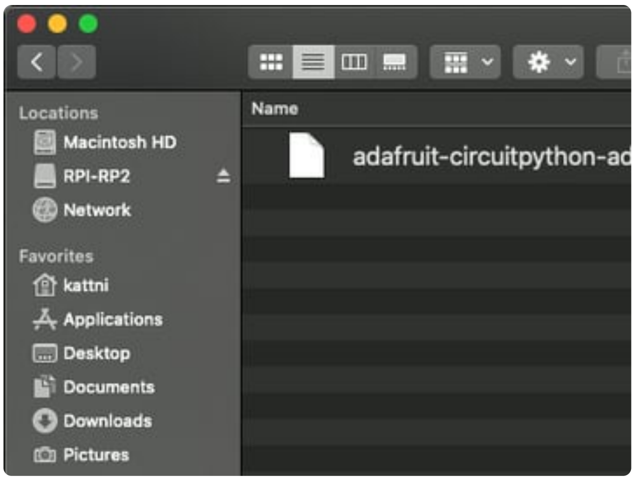


To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

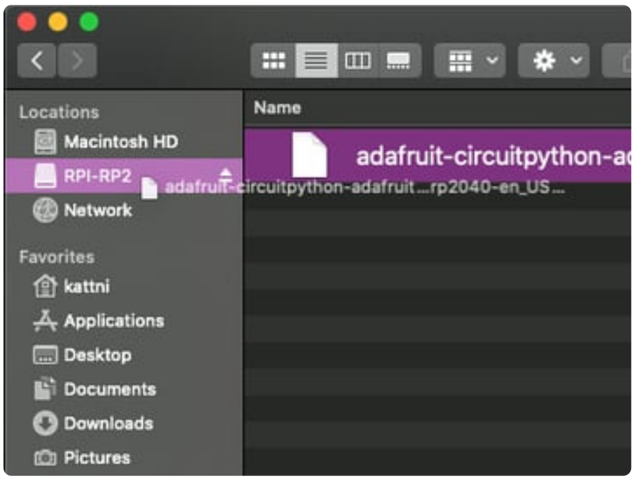
If the drive does not appear, release all the buttons, and then repeat the process above.

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.

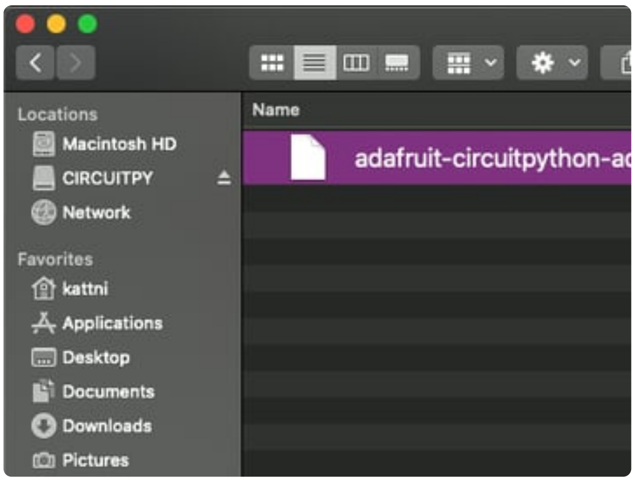
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



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 CIRCUITPY doesn't show up as a disk drive after installing CircuitPython, try loading this 'nuke' UF2 to RPI-RP2. 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

Coding the Mouse with CircuitPython

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.

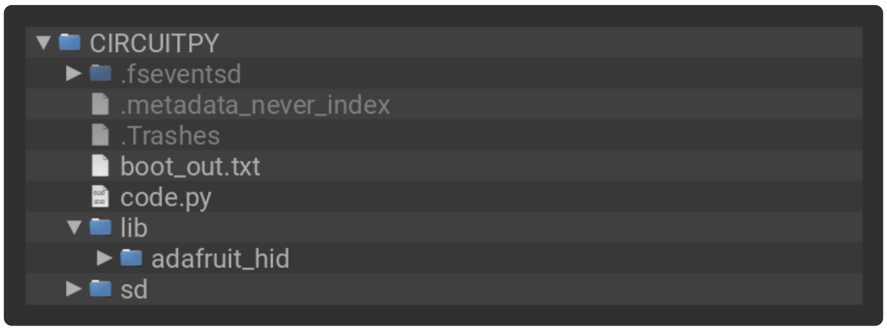
Download the Project Bundle

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.

Hook your QT Py/board to your computer via a known good USB data+power cable. It should show up as a thumb drive named **CIRCUITPY**.

Using File Explorer/Finder (depending on your Operating System), drag the contents of the uncompressed bundle directory onto your board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

Continue below the program listing for a breakdown of how the program functions, section by section.



```
▼ CIRCUITPY
  ▶ .fseventsd
    .metadata_never_index
    .Trashes
    boot_out.txt
    code.py
  ▼ lib
    ▶ adafruit_hid
  ▶ sd
```

```
# SPDX-FileCopyrightText: 2023 Jeff Epler for Adafruit Industries
# SPDX-License-Identifier: MIT
import time
import board
import digitalio
from adafruit_hid.mouse import Mouse
from usb_hid import devices

SCALE = 3

def delta(value): # Convert 8 bit value to signed number of distance steps
    value = ~value # All value is inverted on the bus
    if value & 0x80:  # and is in sign-magnitude format
        return -(value & 0x7f)
    else:
        return value & 0x7f

mouse = Mouse(devices)

spi = board.SPI()
strobe = digitalio.DigitalInOut(board.RX)
strobe.switch_to_output(False)

if not spi.try_lock():
    raise SystemExit("could not lock SPI bus")

spi.configure(baudrate=100_000, polarity=1)

# Wait for the mouse to be ready at power-on
time.sleep(2)

# A buffer to fetch mouse data into
data = bytearray(4)

print("Mouse is ready!")
while True:
    # Request fresh data
    strobe.value = True
    strobe.value = False

    # Must do read in 2 pieces with delay in between
    spi.readinto(data, end=2)
    spi.readinto(data, start=2)
    lmb = bool(~data[1] & 0x40)  # data is inverted on the bus
    rmb = bool(~data[1] & 0x80)
    dy = delta(data[2])
    dx = delta(data[3])


    # Pressing both buttons together emulates the wheel
    wheel = lmb and rmb

    # Compute the new button state
    old_buttons = mouse.report[0]
    mouse.report[0] = (
        0 if wheel else
        mouse.LEFT_BUTTON if lmb else
        mouse.RIGHT_BUTTON if rmb else
        0)

    # If there's any movement, send a move event
    if dx or dy:
        if wheel:
            mouse.move(dx * SCALE, 0, wheel=-dy)
        else:
            mouse.move(dx * SCALE, dy * SCALE)
    elif old_buttons != mouse.report[0]:
        mouse.press(0) # Send buttons previously set via report[0]
```

How it works

Preliminaries

After the required import lines, there's a line where you can customize the amount of movement per tick. You can also check your operating system settings for mouse speed & acceleration settings.

```
# SPDX-FileCopyrightText: 2023 Jeff Epler for Adafruit Industries
# SPDX-License-Identifier: MIT
import time
import board
import digitalio
from adafruit_hid.mouse import Mouse
from usb_hid import devices

SCALE = 3
```

The SNES mouse sends movement data in a specific way: one bit of direction (+ or -) and 7 bits of distance. But all the bits are inverted, too! This function converts the received value into a value from -127 to +127:

```
def delta(value): # Convert 8 bit value to signed number of distance steps
    value = ~value # All value is inverted on the bus
    if value & 0x80:  # and is in sign-magnitude format
        return -(value & 0x7f)
    else:
        return value & 0x7f
```

The mouse communicates using a protocol that is compatible with SPI, though there's also a strobe output that has to be pulsed HIGH (True) anytime new data is needed.

I was unable to determine why the 2-second delay was necessary, but without it the mouse didn't function properly.

```
spi = board.SPI()
strobe = digitalio.DigitalInOut(board.RX)
strobe.switch_to_output(False)

if not spi.try_lock():
    raise SystemExit("could not lock SPI bus")

spi.configure(baudrate=100_000, polarity=1)

# Wait for the mouse to be ready at power-on
time.sleep(2)
```

In the forever-loop, repeatedly fetch data from the mouse in its own format, then convert it to the necessary format for USB HID.

In this implementation, holding **BOTH** buttons at the same time starts mouse wheel emulation mode: moving the mouse up/down sends wheel events instead of normal movement events.

```
while True:
    # Request fresh data
    strobe.value = True
    strobe.value = False

    # Must do read in 2 pieces with delay in between
    spi.readinto(data, end=2)
    spi.readinto(data, start=2)
    lmb = bool(~data[1] & 0x40)  # data is inverted on the bus
    rmb = bool(~data[1] & 0x80)
    dy = delta(data[2])
    dx = delta(data[3])


    # Pressing both buttons together emulates the wheel
    wheel = lmb and rmb

    # Compute the new button state
    old_buttons = mouse.report[0]
    mouse.report[0] = (
        0 if wheel else
        mouse.LEFT_BUTTON if lmb else
        mouse.RIGHT_BUTTON if rmb else
        0)

    # If there's any movement, send a move event
    if dx or dy:
        if wheel:
            mouse.move(dx * SCALE, 0, wheel=-dy)
        else:
            mouse.move(dx * SCALE, dy * SCALE)
    elif old_buttons != mouse.report[0]:
        mouse.press(0) # Send buttons previously set via report[0]
```