USB Game Controller with SNES-like Layout

Created by Tim C

https://learn.adafruit.com/usb-game-controller-with-snes-like-layout

Last updated on 2025-04-22 09:29:21 AM EDT

Table of Contents

Overview

This is a generic USB game controller which provides a two-handed gaming experience for retro gaming, or really any game you want to use a handheld rather than keyboard controller! Use it with your Raspberry Pi or desktop computer while playing emulated games, or as a controller to your CNC machine. It appears as a gamepad device (not a keyboard or mouse) which most games recognize.

The controller functions include a D-Pad, 2 Shoulder buttons, 2 Menu (Select / Start) and 4 Action (Classic X,Y,A,B) Buttons. It's powered through a USB-A Type connector.

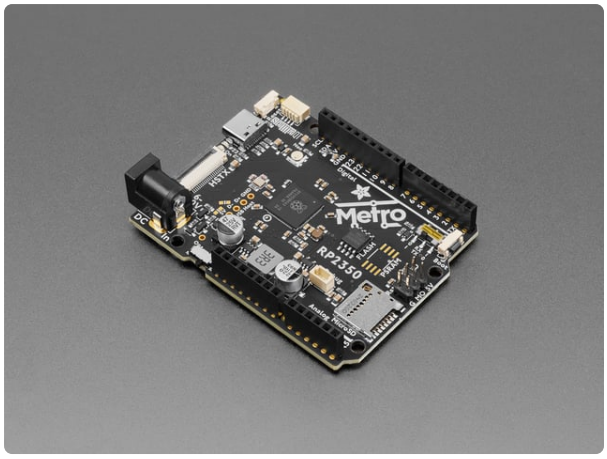
This guide will demonstrate how to use CircuitPython and Arduino to read data from the controller and determine when buttons have been pressed.

Parts

USB Game Controller with SNES-like Layout

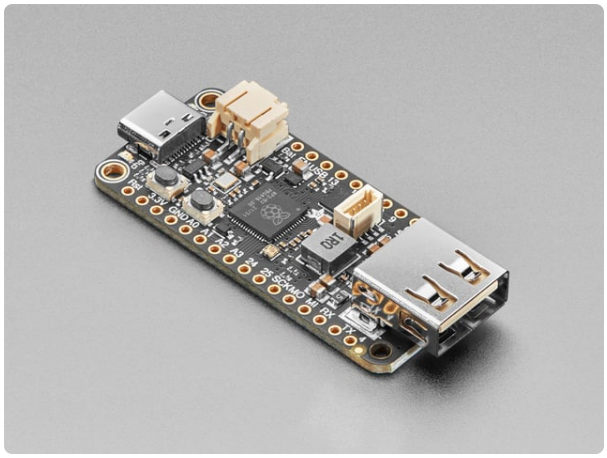
This is a generic USB game controller, which plugs into to provide a two-handed gaming experience for retro gaming, or really any game you want to use a handheld rather than...

https://www.adafruit.com/product/6285



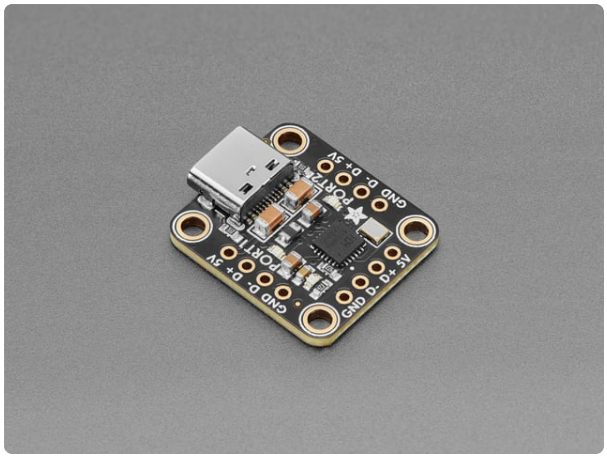
Adafruit Metro RP2350

Choo! Choo! This is the RP2350 Metro Line, making all station stops at "Dual Cortex M33 mountain", "528K RAM round-about" and "16 Megabytes of Flash...
https://www.adafruit.com/product/6003



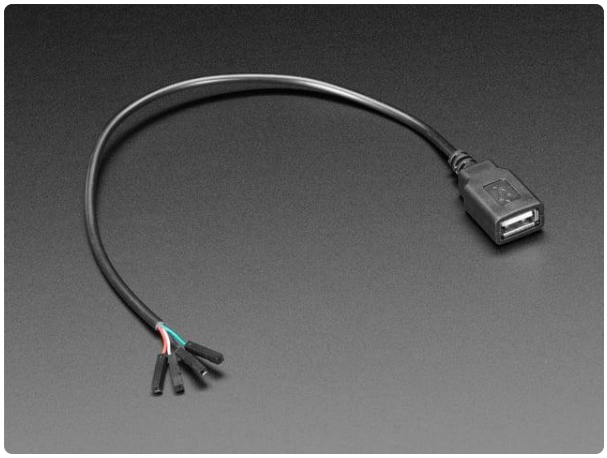
Adafruit Feather RP2040 with USB Type A Host

You're probably really used to microcontroller boards with USB, but what about a dev board with two? Two is more than one, so that makes it twice as good! And...
https://www.adafruit.com/product/5723



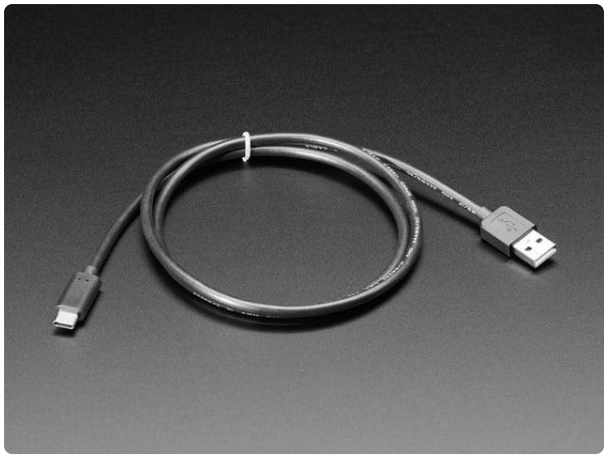
Adafruit CH334F Mini 2-Port USB Hub Breakout

Sometimes, you've got something with a USB host, like an embedded Linux board, and you want to connect more than one thing. Or maybe you want to turn something like a keyboard into...
https://www.adafruit.com/product/5999



USB Type A Jack Breakout Cable with Premium Female Jumpers

If you'd like to connect a USB-host-capable chip to your USB peripheral, this cable will make the task very simple. There is no converter chip in this...
https://www.adafruit.com/product/4449



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...

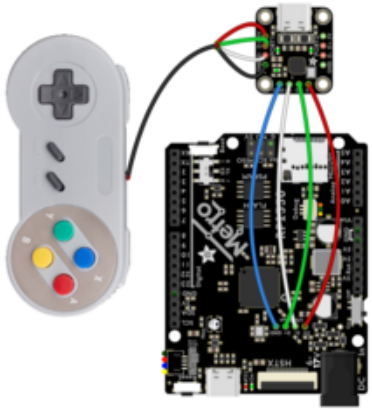
https://www.adafruit.com/product/4474

CircuitPython

The Feather RP2040 USB Host and Metro RP2350 devices both support USB Host and can be used with this game controller.

Wiring Feather RP2040 USB Host

Connecting to the Feather RP2040 USB Host requires plugging in a USB C cable to the Feather's Host port and connecting the other end to a USB Hub breakout such as the CH334F (http://adafru.it/5999).



Wiring Metro RP2350

Connecting to the Metro RP2350 USB Host port requires soldering pins to the broken out USB Host connections as shown in this guide page (https://adafru.it/1ahU). Make the following connections between the Metro USB Host pins and CH334F host connection opposite the USB C connector.

GND to **GND** with **Black** or **Blue**
D+ to **D+** with **Green**
D- to **D-** with **White**
5V to **5V** with **Red**

Note that the data pins are swapped on the Metro compared to the CH334F breakout. On the Metro **D-** is next to **5V**, whereas on the CH334F **D-** is next to **GND**.

Be sure to connect **D-** on the breakout to **D-** on the Metro, and **D+** on the breakout to **D+** on the Metro.

USB Host is under active development. At the time of writing, the SNES-like controller only works when connected through a USB hub such as the CH334F.

Demo Code

Connect the USB game controller while the microcontroller is unplugged from power. Only power up the microcontroller once the USB host connections are made securely.

Press the 'Download Project Bundle' button below to download a zip file containing the demo code. Connect your computer to your Feather or Metro via a known good data+power USB cable. Copy **code.py** and the required libraries to the **CIRCUITPY** drive on your device which appears when the board is connected to your computer via good USB cable.

The code will scan for connected USB devices, printing out information about each one it finds.

Then it will start a loop reading from the first device it found and printing messages when each of the gamepad buttons are pressed. Check the serial console to see the output.

```
# SPDX-FileCopyrightText: Copyright (c) 2025 Tim Cocks for Adafruit Industries
#
# SPDX-License-Identifier: MIT
import array
import time
import usb.core
import adafruit_usb_host_descriptors

# Set to true to print detailed information about all devices found
VERBOSE_SCAN = True

BTN_DPAD_UPDOWN_INDEX = 1
BTN_DPAD_RIGHTLEFT_INDEX = 0
BTN_ABXY_INDEX = 5
BTN_OTHER_INDEX = 6

DIR_IN = 0x80
controller = None

if VERBOSE_SCAN:
    for device in usb.core.find(find_all=True):
        controller = device
        print("pid", hex(device.idProduct))
        print("vid", hex(device.idVendor))
        print("man", device.manufacturer)
        print("product", device.product)
        print("serial", device.serial_number)
        print("config[0]:")
        config_descriptor =
adafruit_usb_host_descriptors.get_configuration_descriptor(
            device, 0
        )

        i = 0
        while i < len(config_descriptor):
            descriptor_len = config_descriptor[i]
            descriptor_type = config_descriptor[i + 1]
            if descriptor_type == adafruit_usb_host_descriptors.DESC_CONFIGURATION:
                config_value = config_descriptor[i + 5]
                print(f" value {config_value:d}")
            elif descriptor_type == adafruit_usb_host_descriptors.DESC_INTERFACE:
                interface_number = config_descriptor[i + 2]
                interface_class = config_descriptor[i + 5]
                interface_subclass = config_descriptor[i + 6]
                print(f" interface[{interface_number:d}]")
                print(
                    f"  class {interface_class:02x} subclass {interface_subclass:
02x}"
                )
            elif descriptor_type == adafruit_usb_host_descriptors.DESC_ENDPOINT:
                endpoint_address = config_descriptor[i + 2]
                if endpoint_address & DIR_IN:
                    print(f"  IN {endpoint_address:02x}")
                else:
                    print(f"  OUT {endpoint_address:02x}")
            i += descriptor_len

# get the first device found
device = None
while device is None:
    for d in usb.core.find(find_all=True):
        device = d
```

```
        break
    time.sleep(0.1)

# set configuration so we can read data from it
device.set_configuration()
print(
    f"configuration set for {device.manufacturer}, {device.product},
{device.serial_number}"
)

# Test to see if the kernel is using the device and detach it.
if device.is_kernel_driver_active(0):
    device.detach_kernel_driver(0)

# buffer to hold 64 bytes
buf = array.array("B", [0] * 64)


def print_array(arr, max_index=None, fmt="hex"):
    """
    Print the values of an array
    :param arr: The array to print
    :param max_index: The maximum index to print. None means print all.
    :param fmt: The format to use, either "hex" or "bin"
    :return: None
    """
    out_str = ""
    if max_index is None or max_index >= len(arr):
        length = len(arr)
    else:
        length = max_index

    for _ in range(length):
        if fmt == "hex":
            out_str += f"{int(arr[_]):02x} "
        elif fmt == "bin":
            out_str += f"{int(arr[_]):08b} "
    print(out_str)


def reports_equal(report_a, report_b, check_length=None):
    """
    Test if two reports are equal. If check_length is provided then
    check for equality in only the first check_length number of bytes.

    :param report_a: First report data
    :param report_b: Second report data
    :param check_length: How many bytes to check
    :return: True if the reports are equal, otherwise False.
    """
    if (
        report_a is None
        and report_b is not None
        or report_b is None
        and report_a is not None
    ):
        return False

    length = len(report_a) if check_length is None else check_length
    for _ in range(length):
        if report_a[_] != report_b[_]:
            return False
    return True


idle_state = None
prev_state = None

while True:
```

```
    try:
        count = device.read(0x81, buf)
        # print(f"read size: {count}")
    except usb.core.USBTimeoutError:
        continue

    if idle_state is None:
        idle_state = buf[:]
        print("Idle state:")
        print_array(idle_state[:8], max_index=count)
        print()

    if not reports_equal(buf, prev_state, 8) and not reports_equal(buf, idle_state,
8):
        if buf[BTN_DPAD_UPDOWN_INDEX] == 0x0:
            print("D-Pad UP pressed")
        elif buf[BTN_DPAD_UPDOWN_INDEX] == 0xFF:
            print("D-Pad DOWN pressed")

        if buf[BTN_DPAD_RIGHTLEFT_INDEX] == 0:
            print("D-Pad LEFT pressed")
        elif buf[BTN_DPAD_RIGHTLEFT_INDEX] == 0xFF:
            print("D-Pad RIGHT pressed")

        if buf[BTN_ABXY_INDEX] == 0x2F:
            print("A pressed")
        elif buf[BTN_ABXY_INDEX] == 0x4F:
            print("B pressed")
        elif buf[BTN_ABXY_INDEX] == 0x1F:
            print("X pressed")
        elif buf[BTN_ABXY_INDEX] == 0x8F:
            print("Y pressed")

        if buf[BTN_OTHER_INDEX] == 0x01:
            print("L shoulder pressed")
        elif buf[BTN_OTHER_INDEX] == 0x02:
            print("R shoulder pressed")
        elif buf[BTN_OTHER_INDEX] == 0x10:
            print("SELECT pressed")
        elif buf[BTN_OTHER_INDEX] == 0x20:
            print("START pressed")

        # print_array(buf[:8])

    prev_state = buf[:]
```

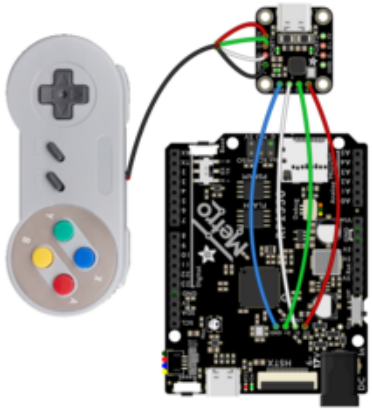
```
code.py output:
pid 0xe401
vid 0x81f
man None
product USB gamepad
serial None
config[0]:
 value 1
 interface[0]
  class 03 subclass 00
  IN 81
configuration set for None, USB gamepad          , None
Idle state:
7f 7f 00 80 80 0f 00 00

D-Pad UP pressed
D-Pad DOWN pressed
D-Pad LEFT pressed
D-Pad RIGHT pressed
A pressed
B pressed
X pressed
Y pressed
SELECT pressed
START pressed
R shoulder pressed
L shoulder pressed
```

Arduino

The Feather RP2040 USB Host and Metro RP2350 devices both support USB Host and can be used with this game controller.

Wiring Feather RP2040 USB Host

Connecting to the Feather RP2040 USB Host requires plugging in a USB C cable to the Feather's Host port and connecting the other end to a USB Hub breakout such as the CH334F (http://adafru.it/5999).



Wiring Metro RP2350

Connecting to the Metro RP2350 USB Host port requires soldering pins to the broken out USB Host connections as shown in this guide page (https://adafru.it/1ahU). Make the following connections between the Metro USB Host pins and CH334F host connection opposite the USB C connector.

GND to **GND** with **Black** or **Blue**
D+ to **D+** with **Green**
D- to **D-** with **White**
5V to **5V** with **Red**

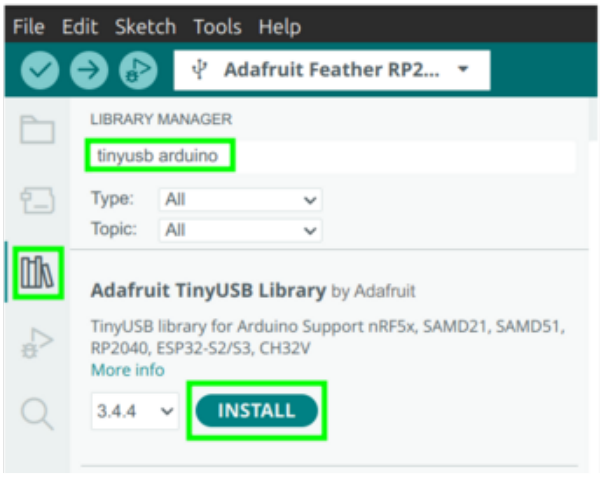
Note that the data pins are swapped on the Metro compared to the CH334F breakout. On the Metro **D-** is next to **5V**, whereas on the CH334F **D-** is next to **GND**.

Be sure to connect **D-** on the breakout to **D-** on the Metro, and **D+** on the breakout to **D+** on the Metro.

USB Host is under active development. At the time of writing this, the SNES-like controller only works when connected through a USB hub such as the CH334F.

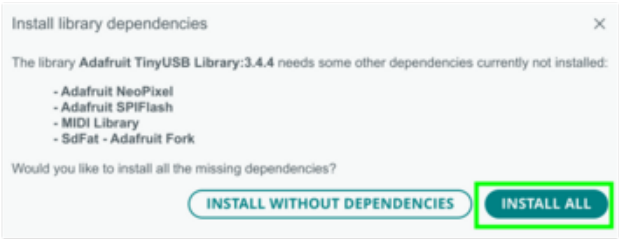
Install the Libraries

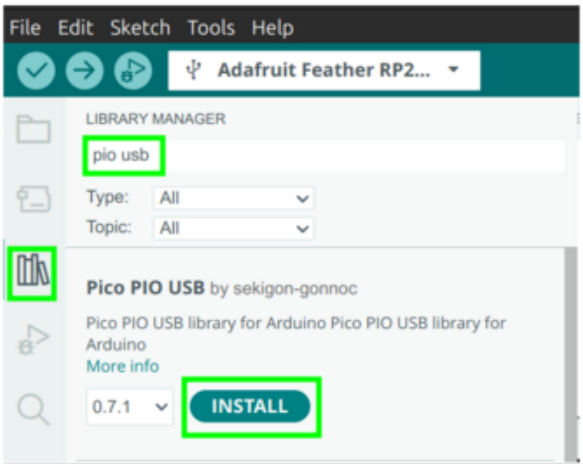
You can install the libraries for this project using the Library Manager in the Arduino IDE.



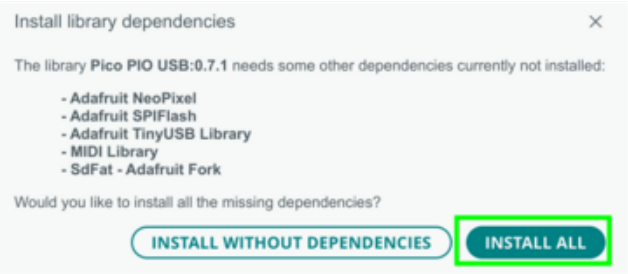
Click the Library Manager icon, search for Adafruit TinyUSB Arduino, and select the Adafruit TinyUSB Library.

If asked about dependencies click "Install All".





Then install the Pico PIO USB library. Click the **Library Manager** icon menu item again, search for **PIO USB**, and select the **Pico PIO USB** library by sekigon-gonnoc.



Code Prep

The code consists of a main **.ino** program file and two header files. The header files store configuration for USB host on the RP2040 and HID device reports for your gamepad. You'll need all three of these files to properly compile and run the project. These files are available in the **.ZIP** folder below or on GitHub (https://adafru.it/1ahV).

snes_gamepad_simpletest.zip

https://adafru.it/1ahW

```
// SPDX-FileCopyrightText: 2025 Tim Cocks for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/*********************************************************************
 Adafruit invests time and resources providing this open source code,
 please support Adafruit and open-source hardware by purchasing
 products from Adafruit!

 MIT license, check LICENSE for more information
 Copyright (c) 2019 Ha Thach for Adafruit Industries
 All text above, and the splash screen below must be included in
 any redistribution
*********************************************************************/

/* This example demonstrates use of usb host with a SNES-like game controller
 * - Host depends on MCU:
```

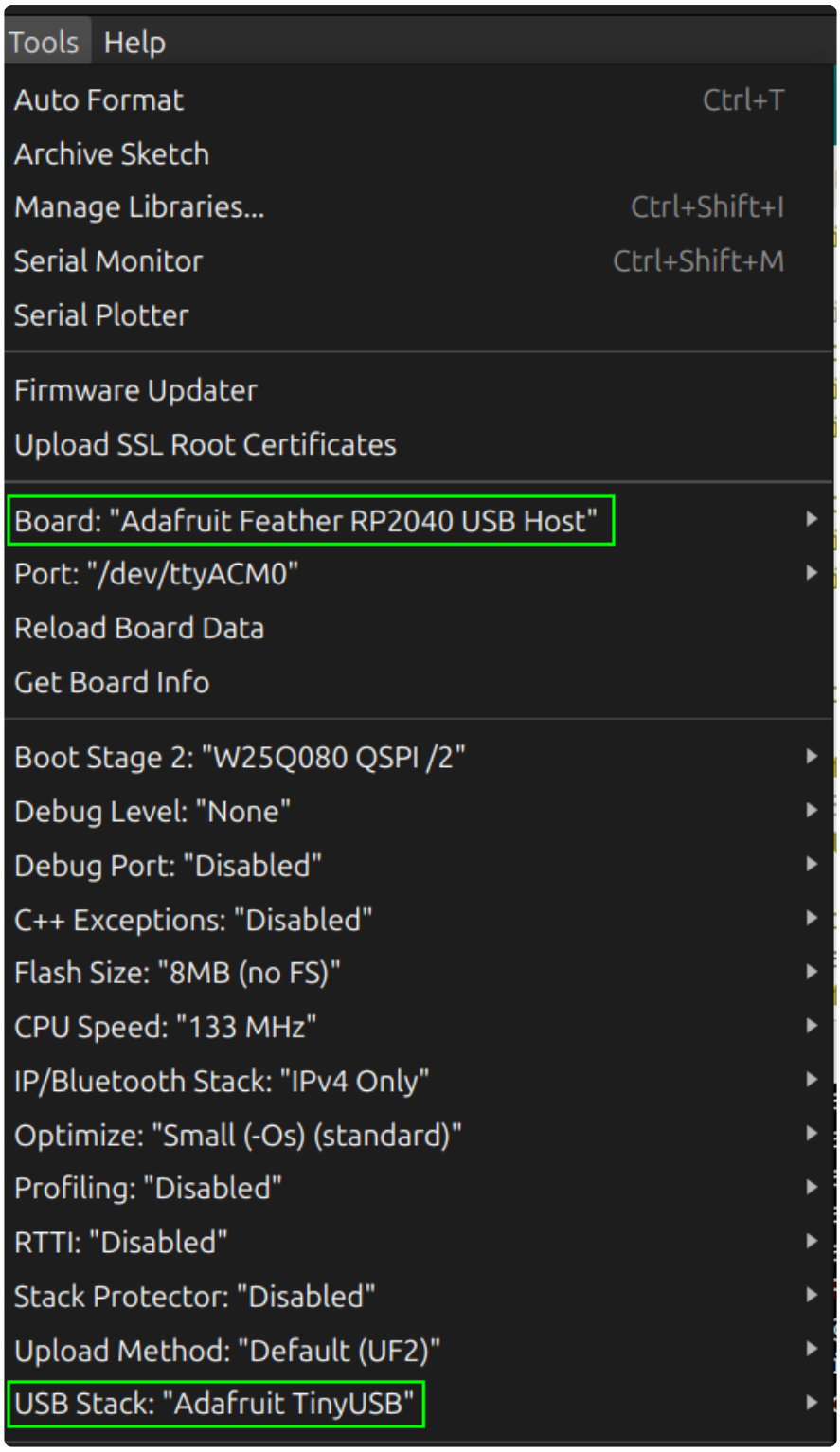
```
 *   - rp2040: bit-banging 2 GPIOs with Pico-PIO-USB library (roothub port1)
 *
 * Requirements:
 * - For rp2040:
 *   - Pico-PIO-USB library
 *   - 2 consecutive GPIOs: D+ is defined by PIN_USB_HOST_DP, D- = D+ +1
 *   - Provide VBus (5v) and GND for peripheral
 *   - CPU Speed must be either 120 or 240 MHz. Selected via "Menu -> CPU Speed"
 */

// USBHost is defined in usbh_helper.h
#include "usbh_helper.h"
#include "tusb.h"
#include "Adafruit_TinyUSB.h"
#include "gamepad_reports.h"

// HID report descriptor using TinyUSB's template
// Single Report (no ID) descriptor
uint8_t const desc_hid_report[] = {
    TUD_HID_REPORT_DESC_GAMEPAD()
};

// USB HID object
Adafruit_USBD_HID usb_hid;

// Report payload defined in src/class/hid/hid.h
// - For Gamepad Button Bit Mask see  hid_gamepad_button_bm_t
// - For Gamepad Hat    Bit Mask see  hid_gamepad_hat_t
hid_gamepad_report_t gp;

bool printed_blank = false;

void setup() {
  Serial.begin(115200);

  // configure pio-usb: defined in usbh_helper.h
  rp2040_configure_pio_usb();

  // run host stack on controller (rhport) 1
  USBHost.begin(1);
  delay(3000);
  Serial.print("USB D+ Pin:");
  Serial.println(PIN_USB_HOST_DP);
  Serial.print("USB 5V Pin:");
  Serial.println(PIN_5V_EN);
}

void loop() {
  USBHost.task();
  Serial.flush();

}

//--------------------------------------------------------------------+
// HID Host Callback Functions
//--------------------------------------------------------------------+

void tuh_hid_mount_cb(uint8_t dev_addr, uint8_t instance, uint8_t const*
desc_report, uint16_t desc_len)
{
  Serial.printf("HID device mounted (address %d, instance %d)\n", dev_addr,
instance);

  // Start receiving HID reports
  if (!tuh_hid_receive_report(dev_addr, instance))
  {
    Serial.printf("Error: cannot request to receive report\n");
  }
}
```

```
void tuh_hid_umount_cb(uint8_t dev_addr, uint8_t instance)
{
  Serial.printf("HID device unmounted (address %d, instance %d)\n", dev_addr,
instance);
}

void tuh_hid_report_received_cb(uint8_t dev_addr, uint8_t instance, uint8_t const*
report, uint16_t len) {

    if (report[BYTE_DPAD_LEFT_RIGHT] != DPAD_NEUTRAL ||
        report[BYTE_DPAD_UP_DOWN] != DPAD_NEUTRAL ||
        report[BYTE_ABXY_BUTTONS] != BUTTON_NEUTRAL ||
        report[BYTE_OTHER_BUTTONS] != BUTTON_MISC_NEUTRAL){

      printed_blank = false;

      //debug print report data
      // Serial.print("Report data: ");
      // for (int i = 0; i < len; i++) {
      //   Serial.print(report[i], HEX);
      //   Serial.print(" ");
      // }
      // Serial.println();

      if (report[BYTE_DPAD_LEFT_RIGHT] == DPAD_LEFT){
        Serial.print("Left ");
      }else  if (report[BYTE_DPAD_LEFT_RIGHT] == DPAD_RIGHT){
        Serial.print("Right ");
      }

      if (report[BYTE_DPAD_UP_DOWN] == DPAD_UP){
        Serial.print("Up ");
      }else  if (report[BYTE_DPAD_UP_DOWN] == DPAD_DOWN){
        Serial.print("Down ");
      }

      if ((report[BYTE_ABXY_BUTTONS] & BUTTON_A) == BUTTON_A){
        Serial.print("A ");
      }
      if ((report[BYTE_ABXY_BUTTONS] & BUTTON_B) == BUTTON_B){
        Serial.print("B ");
      }
      if ((report[BYTE_ABXY_BUTTONS] & BUTTON_X) == BUTTON_X){
        Serial.print("X ");
      }
      if ((report[BYTE_ABXY_BUTTONS] & BUTTON_Y) == BUTTON_Y){
        Serial.print("Y ");
      }

      if ((report[BYTE_OTHER_BUTTONS] & BUTTON_LEFT_SHOULDER) ==
BUTTON_LEFT_SHOULDER){
        Serial.print("Left Shoulder ");
      }
      if ((report[BYTE_OTHER_BUTTONS] & BUTTON_RIGHT_SHOULDER) ==
BUTTON_RIGHT_SHOULDER){
        Serial.print("Right Shoulder ");
      }
      if ((report[BYTE_OTHER_BUTTONS] & BUTTON_START) == BUTTON_START){
        Serial.print("Start ");
      }
      if ((report[BYTE_OTHER_BUTTONS] & BUTTON_SELECT) == BUTTON_SELECT){
        Serial.print("Select ");
      }
      Serial.println();
    } else {
      if (! printed_blank){
        Serial.println("NEUTRAL");
        printed_blank = true;
```

```
      }
    }

  // Continue to receive the next report
  if (!tuh_hid_receive_report(dev_addr, instance)) {
    Serial.println("Error: cannot request to receive report");
  }
}
```

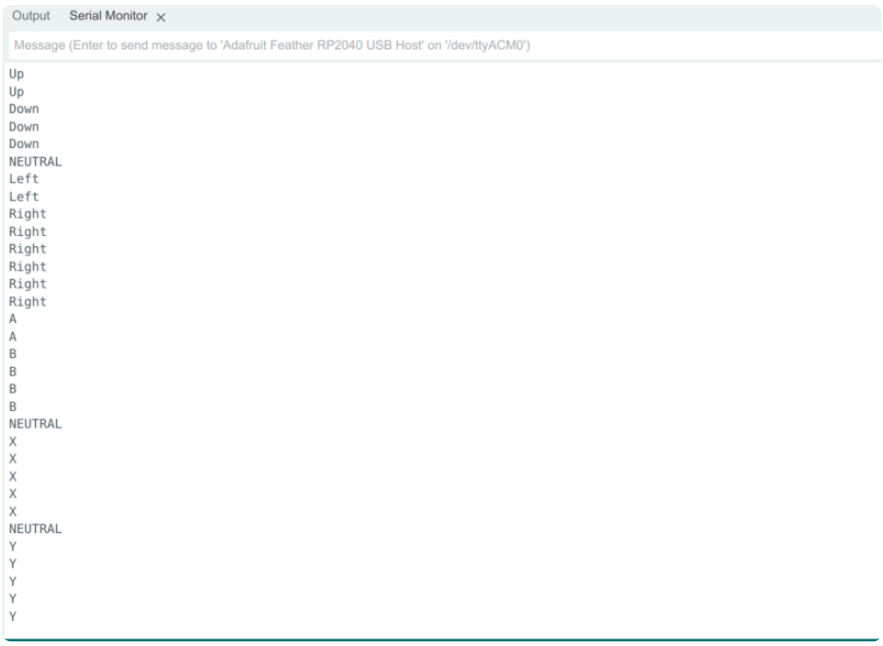
Upload and Test

Before uploading, you'll need to update some settings in the Boards menu. Select the appropriate board that you are using, either **Adafruit Feather RP2040 USB Host**, or **Adafruit Metro RP2350**. Under **USB Stack** select **Adafruit TinyUSB**. Then, upload the sketch to your board. You can use the Serial Monitor in the Arduino IDE for debugging any errors.



Serial Output

The code will connect to the SNES-like controller and read data coming from it. When it detects that buttons are pressed it will print which ones are down into the serial output.


Output
Serial Monitor
Message (Enter to send message to 'Adafruit Feather RP2040 USB Host' on '/dev/ttyACM0')
Up
Up
Down
Down
Down
NEUTRAL
Left
Left
Right
Right
Right
Right
Right
Right
A
A
B
B
B
B
NEUTRAL
X
X
X
X
X
NEUTRAL
Y
Y
Y
Y
Y