Super Nintendo USB Controller

Created by Robert Dale Smith

https://learn.adafruit.com/super-nintendo-usb-controller

Last updated on 2023-10-11 11:01:38 AM EDT
# Table of Contents

## Overview
- Features
- Materials & Tools
- Parts

## How it Works

## Mod the Controller
- Solder Wires to KB2040

## Installing CircuitPython
- CircuitPython Quickstart
- Safe Mode
- Flash Resetting UF2

## Code the Controller
- Text Editor
- Download the Project Bundle
- How It Works

## Use the Controller
- Testing
- Reassemble
- Play and Tinker

## Bonus (GP2040-CE)
- What is GP2040-CE?
- Installing GP2040-CE
- Input Modes
- Web Config
- Fun Time
Overview

Many of us have fond memories growing up playing Super Nintendo. The controller is iconic and a true gaming classic. Rather than relegating these old controllers to the dust bin, they can be given new life.

In this guide, we'll show you how to easily convert any original Super Nintendo compatible controller into a fully functional modern USB gamepad that can be used just about anywhere.

We will be tack-soldering onto the PCB inside the controller but there's no cutting or component removal so it's super easy to convert it back to SNES-plug-compatibility if you decide to go back to your retro roots!

Features

- Fully re-programmable Super Nintendo USB controller
- Windows/MacOS/Linux Compatible USB HID Gamepad
- 100% reversible modification

Materials & Tools

- Super Nintendo Console Controller
- An Adafruit KB2040 microcontroller
- About 2ft of jumper wire (Thin 30 AWG wire wrap recommended)
• A soldering iron and solder
• Hot glue gun with glue stick
• Flush angled cutter pliers
• A multimeter (optional)
• Phillips screwdriver
• USB-A cable

Parts

SNES Controller
A third-party SNES (Super Nintendo/Famicom) controller. Works great with Fuzeboxen as well!
https://www.adafruit.com/product/131

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

Rainbow "Wire Wrap" Thin 30 AWG Prototyping & Repair Wire
This stuff is called "wire-wrap wire" because it used to be used for wire-wrapping high-speed digital circuits on a special kind of contact board. It's pretty rare to see...
https://www.adafruit.com/product/4730
USB A/Micro Cable - 2m
This is your standard USB A-Plug to Micro-USB cable. It's 2 meters long so you'll have plenty of cord to work with for those longer extensions.
https://www.adafruit.com/product/2185

If needed/desired:

Flush diagonal cutters
These are the best diagonal cutters, large super-comfortable grip to use and have strong nippers for perfect trimming of wires and leads. I've used my pair every day for years.
https://www.adafruit.com/product/152

Digital Multimeter - Model 9205B+
This massive multimeter has everything but the kitchen sink included. It's a great addition to any workbench or toolbox. It's low cost, simple to use, and has a big clear...
https://www.adafruit.com/product/2034

How it Works

The Super Nintendo controller has a relatively simple protocol. It uses two 8-bit shift registers in parallel, totaling 16-bits. 12 button state bits followed by 4 device ID bits.

The shift registers can be read by signaling a latch pulse to start the read cycle, followed by 16 clock pulses for each of the 16 data bits to be read in.
A latch pulse tells the controller that the console is about to send clock pulses and read in the button data. Each time the clock signal goes low the next button data bit is read in.

For example, the first bit is \( B \), second bit is \( Y \), third is Select and so on. In the image above you can see what it looks like when \( Y \) and Start buttons are held and released together, over and over.
Mod the Controller

Disassemble

Start by removing the five Phillips head screws on the rear of the controller.

Carefully remove the PCB from the shell and disconnect the cable.

Your circuit board may have one or two chips, but both variants will have the same five output pins on the cable connector.
Clean (optional)

With these controllers being over three decades old and likely never been cleaned, this is a good opportunity to clean it up.

The rubber pads and plastic parts can all be cleaned with a simple mixture of dish soap, water, and a bit of gentle scrubbing.
Solder USB Cable

Next we will need to prepare and solder a USB-A cable directly to the KB2040 microcontroller board. Any old spare USB-A cable will work for this project.

Start by cutting off the unused end of the cable opposite of the USB-A connector. Then trim back the outer insulation to reveal the 4 USB wires. (Power, Ground, D+ and D-)

Many USB cables will also include a 5th uninsulated wire that is connected to the shielding of the USB connector. It’s recommended to connect this to ground, but you are also fine to just cut it off.
The wire colors are common but may not always match. Verify with a volt meter.

Black: Ground
Green: D+
White: D-
Red: Power

Following the diagram below, solder the four wires to the KB2040. (and the shielding wire to ground if present)

Solder Wires to Controller

Cut 5 jumper wires about 4 inches long (10cm). Then strip a small bit of insulation off of the wire tips.
On the front side of the controller PCB, solder a wire to each of the five controller cable pins.
Keeping track of the pin number wire order, carefully flip over the board and place it within the front half of the controller shell.

Mount KB2040

Using a healthy dab of hot glue, the KB2040 can be mounted directly to the controller PCB.

Solder Wires to KB2040

The SNES controller shift registers are capable of operating at 3.3V, the safe operating range of the KB2040's GPIO. So we'll connect the SNES power to the KB2040's 3.3V pin.
Solder the remaining wires to the KB2040. Ground to ground. Then clock, latch, and data pins to D5, D6, and D7.

SNES → KB2040
1: Ground → GND
2: Data → D7
3: Latch → D6
4: Clock → D5
5: Power → 3.3V
Program the KB2040

Before reassembling the controller, put the KB2040 in boot mode.

Holding the boot button while connecting the USB cable to a computer will mount a storage device called RPI-RP2.

Installing CircuitPython

CircuitPython is a derivative of MicroPython 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

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

---

**Code the Controller**

**Text Editor**

Adafruit recommends using the Mu editor for editing your CircuitPython code. You can get more info in [this guide](#).

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.

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. The CIRCUITPY drive appears when you plug the KB2040 into the computer via USB.

```
# SPDX-FileCopyrightText: 2023 Robert Dale Smith for Adafruit Industries
#
# SPDX-License-Identifier: MIT
# USB HID descriptor for generic DirectInput compatible gamepad.

import usb_hid

# This is only one example of a gamepad descriptor, and may not suit your needs.
GAMEPAD_REPORT_DESCRIPTOR = bytes(
    0x05, 0x01,  # USAGE_PAGE (Generic Desktop)
    0x09, 0x05,  # USAGE (Gamepad)
    0xa1, 0x01,  # COLLECTION (Application)
    0x15, 0x00,  #   LOGICAL_MINIMUM (0)
    0x25, 0x01,  #   LOGICAL_MAXIMUM (1)
    0x35, 0x00,  #   PHYSICAL_MINIMUM (0)
    0x45, 0x01,  #   PHYSICAL_MAXIMUM (1)
    0x75, 0x01,  #   REPORT_SIZE (1)
    0x95, 0xe0,  #   REPORT_COUNT (14)
    0x05, 0x09,  #     USAGE_PAGE (Button)
    0x09, 0x01,  #     USAGE (Button 1)
    0x09, 0x02,  #     USAGE (Button 14)
    0x09, 0x02,  #     INPUT (Data,Var,Abs)
    0x05, 0x09,  #     USAGE_PAGE (Button)
    0x09, 0x01,  #     USAGE (Button 1)
    0x09, 0x02,  #     USAGE (Button 14)
    0x09, 0x02,  #     INPUT (Data,Var,Abs)
)
```

©Adafruit Industries
0x81, 0x01,      # INPUT (Cnt, Ary, Abs)
0x05, 0x01,      # USAGE_PAGE (Generic Desktop)
0x25, 0x07,      # LOGICAL_MAXIMUM (7)
0x46, 0x3b, 0x01, # PHYSICAL_MAXIMUM (315)
0x75, 0x04,      # REPORT_SIZE (4)
0x95, 0x01,      # REPORT_COUNT (1)
0x65, 0x14,      # UNIT (Eng Rot: Angular Pos)
0x09, 0x39,      # USAGE (Hat switch)
0x81, 0x42,      # INPUT (Data, Var, Abs, Null)
0x65, 0x00,      # UNIT (None)
0x95, 0x01,      # REPORT_COUNT (1)
0x81, 0x01,      # INPUT (Cnt, Ary, Abs)
0x26, 0xff, 0x00, # LOGICAL_MAXIMUM (255)
0x46, 0xff, 0x00, # PHYSICAL_MAXIMUM (255)
0x09, 0x30,      # USAGE (X)
0x09, 0x31,      # USAGE (Y)
0x09, 0x32,      # USAGE (Z)
0x09, 0x35,      # USAGE (Rz)
0x75, 0x08,      # REPORT_SIZE (8)
0x95, 0x04,      # REPORT_COUNT (6)
0x81, 0x02,      # INPUT (Data, Var, Abs)
0x06, 0x00, 0xff, # USAGE_PAGE (Vendor Specific)
0x09, 0x20,      # Unknown
0x09, 0x21,      # Unknown
0x09, 0x22,      # Unknown
0x09, 0x23,      # Unknown
0x09, 0x24,      # Unknown
0x09, 0x25,      # Unknown
0x09, 0x26,      # Unknown
0x09, 0x27,      # Unknown
0x09, 0x28,      # Unknown
0x09, 0x29,      # Unknown
0x09, 0x2a,      # Unknown
0x09, 0x2b,      # Unknown
0x09, 0x2c,      # REPORT_COUNT (12)
0x81, 0x02,      # INPUT (Data, Var, Abs)
0x0a, 0x21, 0x26, # Unknown
0x95, 0x08,      # REPORT_COUNT (8)
0xb1, 0x02,      # FEATURE (Data, Var, Abs)
0xc0            # END_COLLECTION
)

gamepad = usb_hid.Device(
    report_descriptor=GAMEPAD_REPORT_DESCRIPTOR,
    usage_page=0x01,           # Generic Desktop Control
    usage=0x05,                # Gamepad
    report_ids=(0,),           # Descriptor uses report ID 0.
    in_report_lengths=(19,),   # This gamepad sends 19 bytes in its report.
    out_report_lengths=(0,),   # It does not receive any reports.
)

usb_hid.enable((gamepad,))

# SPDX-FileCopyrightText: 2023 Robert Dale Smith for Adafruit Industries
#
# Simple Super Nintendo controller to standard USB HID gamepad with DirectInput
# button mapping.
# Tested on KB2040

import time
import board
import digitalio
import usb_hid

# Update the SNES Controller pins based on your input
LATCH_PIN = board.D6
CLOCK_PIN = board.D5
DATA_PIN = board.D7

# Set up pins for SNES Controller
latch = digitalio.DigitalInOut(LATCH_PIN)
latch.direction = digitalio.Direction.OUTPUT
clock = digitalio.DigitalInOut(CLOCK_PIN)
clock.direction = digitalio.Direction.OUTPUT
data = digitalio.DigitalInOut(DATA_PIN)
data.direction = digitalio.Direction.INPUT
data.pull = digitalio.Pull.UP  # pull-up as a default

# SNES to USB button mapping
buttonmap = {
    "B": (0, 0, 0x2),       # Button 1
    "Y": (1, 0, 0x1),       # Button 3
    "Select": (2, 1, 0x01), # Button 9
    "Start": (3, 1, 0x02),  # Button 10
    "Up": (4, 0, 0),        # D-pad North
    "Down": (5, 0, 0),      # D-pad South
    "Left": (6, 0, 0),      # D-pad East
    "Right": (7, 0, 0),     # D-pad West
    "A": (8, 0, 0x4),       # Button 2
    "X": (9, 0, 0x8),       # Button 4
    "L": (10, 0, 0x10),     # Button 5
    "R": (11, 0, 0x20)      # Button 6
}

# D-pad buttons to hat-switch value
dpad_state = {
    "Up": 0,
    "Down": 0,
    "Left": 0,
    "Right": 0,
}

hat_map = {
    (0, 0, 0, 0): 0x08,  # Released
    (1, 0, 0, 0): 0x00,  # N
    (1, 1, 0, 0): 0x01,  # NE
    (0, 1, 0, 0): 0x02,  # E
    (0, 1, 0, 1): 0x03,  # SE
    (0, 0, 0, 1): 0x04,  # S
    (0, 0, 1, 1): 0x05,  # SW
    (0, 0, 1, 0): 0x06,  # W
    (1, 0, 1, 0): 0x07,  # NW
}

def read_snes_controller():
    data_bits = []
latch.value = True
time.sleep(0.000012)  # 12µs
latch.value = False

do
for _ in range(16):
    time.sleep(0.000006)  # Wait 6µs
data_bits.append(data.value)

    clock.value = True
time.sleep(0.000006)  # 6µs
clock.value = False
time.sleep(0.000006)  # 6µs

    return data_bits

# Find the gamepad device in the usb_hid devices
gamepad_device = None
for device in usb_hid.devices:
    if device.usage_page == 0x01 and device.usage == 0x05:
        gamepad_device = device
        break

if gamepad_device is not None:
    print("Gamepad device found!")
else:
    print("Gamepad device not found.")

report = bytearray(19)
report[2] = 0x08  # default released hat switch value
report[3] = 127  # default x center value
report[4] = 127  # default y center value
report[5] = 127  # default z center value
report[6] = 127  # default rz center value
prev_report = bytearray(report)

while True:
    button_state = read_snes_controller()
    all_buttons = list(buttonmap.keys())

    for idx, button in enumerate(all_buttons):
        index, byte_index, button_value = buttonmap[button]
        is_pressed = not button_state[index]  # True if button is pressed

        if button in dpad_state:  # If it's a direction button
            dpad_state[button] = 1 if is_pressed else 0
        else:
            if is_pressed:
                report[byte_index] |= button_value
            else:  # not pressed, reset button state
                report[byte_index] &= ~button_value

    # SOCD (up priority and neutral horizontal)
    if (dpad_state["Up"] and dpad_state["Down"])�
        dpad_state["Down"] = 0
    if (dpad_state["Left"] and dpad_state["Right"])�
        dpad_state["Left"] = 0
        dpad_state["Right"] = 0

    # Extract the dpad_state to a tuple and get the corresponding hat value
    dpad_tuple = (dpad_state["Up"], dpad_state["Right"], dpad_state["Left"], dpad_state["Down"])

    if prev_report != report:
        gamepad_device.send_report(report)
        prev_report[:] = report

    time.sleep(0.1)
How It Works

The code uses the `digitalio` library to send the SNES console latch/clock signals and to read the controller button data wire. Then the `usb_hid` library is used to send standard USB HID Gamepad reports when buttons are pressed or released.

USB HID Gamepad Descriptor

First, the boot.py file is needed to predefine the USB descriptor to appear to the computer as gamepad at boot time. Within this file the `usb_hid` library is imported and a custom USB HID gamepad descriptor is defined. This descriptor follows a common pattern of DirectInput compatible USB controllers, making it easily compatible with Windows, Mac, and Linux operating systems.

The USB Gamepad descriptor defined has a size of 19 bytes.

- 1st and 2nd bytes contain the 14 button bits. (1 = pressed, 0 = released)
- 3rd byte contains a 4-bit hat switch value (d-pad)
- 4th-7th bytes are the 8-bit analog axis values X, Y, Z, Rz (left/right analog stick)
- The remaining bytes contain compatibility feature flags and vendor specific values.

Libraries

The meat of our program lives within the code.py file. It starts with the importing the necessary libraries.

```python
import time
import board
import digitalio
import usb_hid
```

Digital In Out

Next, to define the controller's Latch, Clock, and Data GPIO pin numbers. Then to create an instance of the `DigitalInOut` class for each pin and set the pin directions.

Pulsing the Latch wire output high will signal to the controller the start of the next read cycle. Then the controller will wait for a sequence of Clock output pulses. Upon each clock high/low, the controller will respond with the next button bit that we will read from the data input. (0 = pressed, 1 = released)
LATCH_PIN = board.D6
CLOCK_PIN = board.D5
DATA_PIN = board.D7

# Set up pins for SNES Controller
latch = digitalio.DigitalInOut(LATCH_PIN)
latch.direction = digitalio.Direction.OUTPUT

clock = digitalio.DigitalInOut(CLOCK_PIN)
clock.direction = digitalio.Direction.OUTPUT

data = digitalio.DigitalInOut(DATA_PIN)
data.direction = digitalio.Direction.INPUT
data.pull = digitalio.Pull.UP  # pull-up as a default

SNES to USB Mapping

Next, to define the buttonmap object that will be used to translate from the SNES controller button data read in to the USB HID report output byte and bit positions.

The buttons are defined in the order they are read in from the SNES controller. For example, SNES bit 0 is "B", bit 1 is "Y", bit 2 is "Select", etc.

"ButtonName": (SNES bit index, HID byte index, HID bit index)

```
# SNES to USB button mapping
buttonmap = {
    "B": (0, 0, 0x2),       # Button 1
    "Y": (1, 0, 0x1),       # Button 3
    "Select": (2, 1, 0x01), # Button 9
    "Start": (3, 1, 0x02),  # Button 10
    "Up": (4, 0, 0),        # D-pad North
    "Down": (5, 0, 0),      # D-pad South
    "Left": (6, 0, 0),      # D-pad West
    "Right": (7, 0, 0),     # D-pad East
    "A": (8, 0, 0x4),       # Button 2
    "X": (9, 0, 0x8),       # Button 4
    "L": (10, 0, 0x10),     # Button 5
    "R": (11, 0, 0x20)      # Button 6
}
```

D-pad to Hat-switch State

Standard USB HID gamepads use a hat-switch value for directional pad button inputs. We will use the dpad_state object to store the current d-pad directions. Then the hat_map is used to convert that state directly to the corresponding hat-switch directional value.

```
# D-pad buttons to hat-switch value
dpad_state = {
    "Up": 0,
    "Down": 0,
    "Left": 0,
    "Right": 0,
```
hat_map = {
    (0, 0, 0, 0): 0x08,  # Released
    (1, 0, 0, 0): 0x00,  # N
    (1, 1, 0, 0): 0x01,  # NE
    (0, 1, 0, 0): 0x02,  # E
    (0, 1, 0, 1): 0x03,  # SE
    (0, 0, 0, 1): 0x04,  # S
    (0, 0, 1, 1): 0x05,  # SW
    (0, 0, 1, 0): 0x06,  # W
    (1, 0, 1, 0): 0x07,  # NW
}

SNES Controller Reading

Next, we'll define the `read_snes_controller()` function that will be used to poll the controller by pulsing the latch and clock outputs while reading the data input.

The function will read and return 16-bits from the controller. The first 12-bits each represent a button, while the remaining 4-bits are the controller ID (all high).

```python
def read_snes_controller():
    data_bits = []
    latch.value = True
    time.sleep(0.000012)  # 12µs
    latch.value = False
    for _ in range(16):
        time.sleep(0.000006)  # Wait 6µs
        data_bits.append(data.value)
        clock.value = True
        time.sleep(0.000006)  # 6µs
        clock.value = False
        time.sleep(0.000006)  # 6µs
    return data_bits
```

USB HID Gamepad Device

Next, we'll define `gamepad_device` with the usb_hid device predefined at boot within the boot.py file.

```python
# Find the gamepad device in the usb_hid devices
gamepad_device = None
for device in usb_hid.devices:
    if device.usage_page == 0x01 and device.usage == 0x05:
        gamepad_device = device
        break
```
Finally, we'll need to create an array of 19 bytes called `report`. Set the hat-switch byte to its default value of `0x08`. Also set the unused analog values to the default center position of `127`.

Then create a `prev_report` array for keeping track of when the output report needs to be updated.

```python
report = bytearray(19)
report[2] = 0x08  # default released hat switch value
report[3] = 127  # default x center value
report[4] = 127  # default y center value
report[5] = 127  # default z center value
report[6] = 127  # default rz center value
prev_report = bytearray(report)
```

**Main Loop**

The main loop first reads in the SNES controller `button_state`. Then each possible button state bit is checked and the corresponding USB HID output report bits are toggled depending on whether it is pressed or released. (`report[0]` or `report[1]`)

Once the buttons have been processed, the `dpad_state` will be SOCD() cleaned. Meaning we will clean up opposing button direction presses to avoid them being output.

Then the `dpad_state` is translated to the hat-switch value and set to the third report byte. (`report[2]`)

Finally whenever the report data has changed, send the USB HID output report data with the `send_report(report)` function.

```python
while True:
    button_state = read_snes_controller()
    all_buttons = list(buttonmap.keys())
    for idx, button in enumerate(all_buttons):
        index, byte_index, button_value = buttonmap[button]
        is_pressed = not button_state[index]  # True if button is pressed
        if button in dpad_state:  # If it's a direction button
            dpad_state[button] = 1 if is_pressed else 0
        else:
            if is_pressed:
                report[byte_index] |= button_value
            else:  # not pressed, reset button state
                report[byte_index] &= ~button_value
```
# SOCD (up priority and neutral horizontal)
if (dpad_state["Up"] and dpad_state["Down"]):
    dpad_state["Down"] = 0
if (dpad_state["Left"] and dpad_state["Right"]):
    dpad_state["Left"] = 0
    dpad_state["Right"] = 0

# Extract the dpad_state to a tuple and get the corresponding hat value
dpad_tuple = (dpad_state["Up"], dpad_state["Right"], dpad_state["Left"],
              dpad_state["Down"])

if prev_report != report:
    gamepad_device.send_report(report)
    prev_report[:] = report

time.sleep(0.1)

---

Use the Controller

Testing

To quickly check your work, the website Gamepad-Tester.com is an easy way to test USB controllers on any computer with a browser.

After pressing a button the website will detect the controller and start showing button presses.

For compatibility, this project used a full gamepad USB HID descriptor with 14-buttons, d-pad, and two analog sticks. So only the D-pad and 8 buttons programmed will register button presses.

Reassemble

Congratulations! If the button presses are being detected, then you have successfully completed the Super Nintendo controller USB mod.

Lastly, put the shell back together and screw-in the five screws to the back shell.
Play and Tinker

Now you are ready to start gaming on Windows, Mac, or Linux operating systems.

For fun, the code can be easily modified to remap buttons or even to map the d-pad presses to analog stick outputs.

Bonus (GP2040-CE)

But wait.. there's more!

The great part about the KB2040 boards is the RP2040 chip and its ability to run the GP2040-CE firmware.

What is GP2040-CE?

GP2040-CE is a multiplatform, open source gamepad firmware that is compatible with PC, PS3 and PS4, Nintendo Switch, Steam Deck, MiSTer and Android.

Also a SNES Input add-on was recently added to GP2040-CE. So if you followed the KB2040 pinout in this guide, then you can use this pre-compiled build of GP2040-CE with the SNES Input add-on automatically enabled.
Installing GP2040-CE

To flash this firmware to the KB2040, simply hold the boot button while connecting the USB cable to a computer like was done with the CircuitPython install.

Then drag-n-drop this file onto the RPI-RP2 storage device that mounts.

Gamepad Testing

Using Gamepad-Tester.com again, you can verify GP2040-CE is working. By default the controller will be in X-input mode and should display as an Xbox 360 controller.

Input Modes

While connecting the controller's USB cable, you can hold A, B, X, or Y to switch the controllers input mode. For example, holding B button will make the controller compatible with Nintendo Switch.

Web Config

GP2040-CE even includes a built-in web configurator with a slew of advanced options. Simply hold the Start button while connecting the controller to put it in config mode. Then open a web browser and navigate to http://192.168.7.1/.

Fun Time

That's it! Now go play some games and have fun with your new Super Nintendo USB controller.