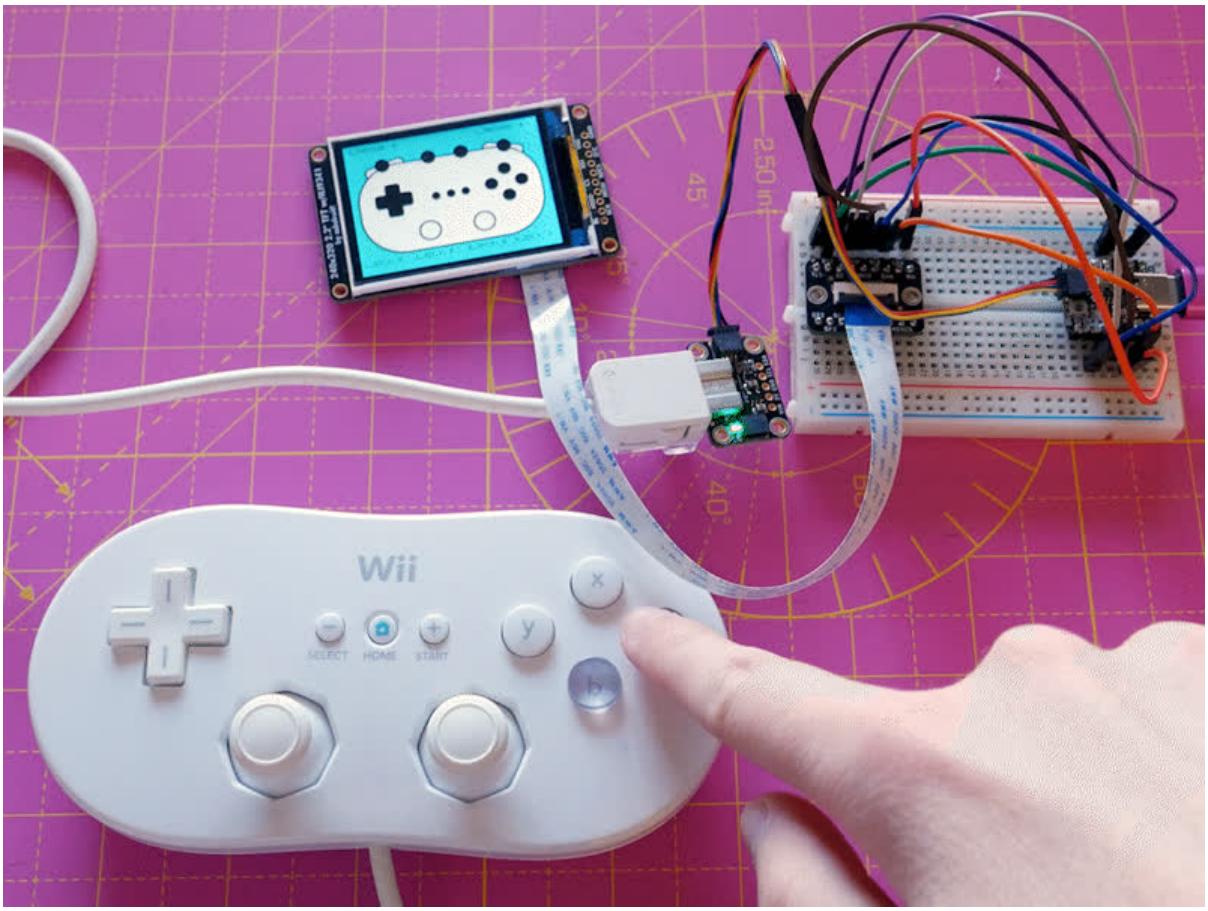
Adafruit CircuitPython Wii Classic Controller Library

Created by Liz Clark



https://learn.adafruit.com/adafruit-circuitpython-wii-classic-controller-library

Last updated on 2024-06-03 03:49:12 PM EDT

Table of Contents

Overview

The Adafruit CircuitPython Wii Classic Controller library (https://adafru.it/18Cq) will let you connect a Wii Classic (https://adafru.it/18Cr) compatible controller to an Adafruit Wii Nunchuck Breakout Adapter (http://adafru.it/4836) over STEMMA QT I2C to read incoming inputs from the controller. This means you could use these controllers in your next CircuitPython project to drive your robot, control your synth or run your NeoPixel light show.

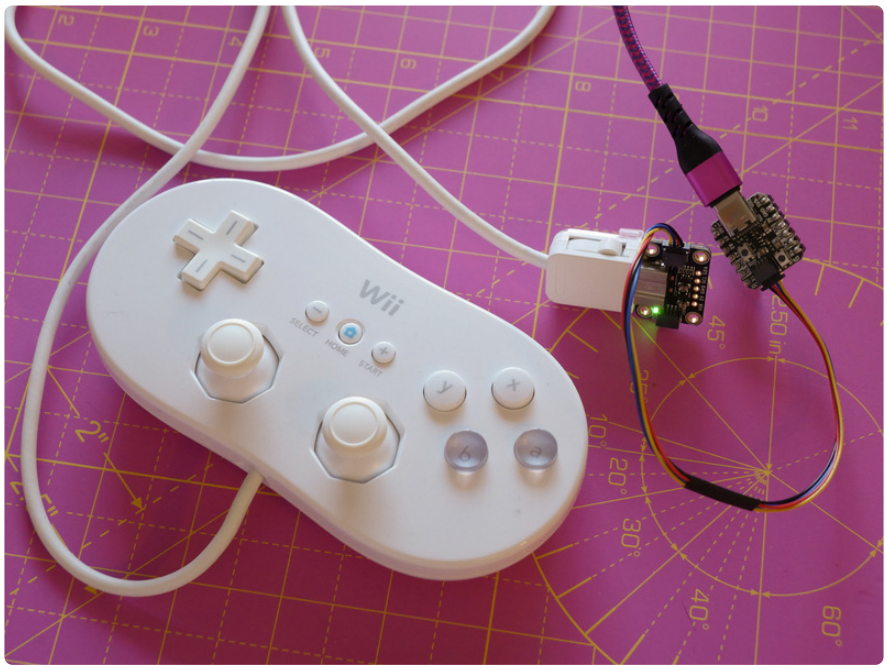
You can use this CircuitPython driver with gamepads that have the notched plug connector. In addition to the original Wii Classic Controller, these include the NES and SNES-style controllers that were released with the mini classic edition systems (https://adafru.it/18Cs). All of these controllers use the same I2C address (**0x52**) and button mappings. For example, pressing the A button on the Wii Classic controller or the SNES Classic controller will both be recognized by the driver as an A button input.

Adafruit Wii Nunchuck Breakout Adapter
Dig out that old Wii controller and use it as a sleek controller for your next robot if you like. The Adafruit Adafruit Wii Nunchuck Breakout Adapter fits snugly into the Wii connector...
https://www.adafruit.com/product/4836

Community Bundle Library

There is a library in the CircuitPython Community Bundle (https://adafru.it/18Ct), the CircuitPython_WiiChuck library (https://adafru.it/18Cu), that has drivers for a variety of additional Wii Remote accessories. Check it out if you're experimenting with some of the quirkier hardware that was released during the Wii console's reign.

Library Syntax



The following page will walk you through the syntax for interfacing with the driver in CircuitPython to read your controller button presses. Note that depending on your controller, all of the functionality may not be available. For example, NES-style controllers will usually only have a D-Pad and buttons for A, B, Start and Select.

Instantiate Over I2C

The I2C address for the controllers is **0x52** and it cannot be changed.

```
import board
import adafruit_wii_classic

i2c = board.STEMMA_I2C()
ctrl_pad = adafruit_wii_classic.Wii_Classic(i2c)
```

Depending on your controller, all of the functionality may not be available.

D-Pad

- **Up button** - `ctrl_pad.d_pad.UP`
- **Down button** - `ctrl_pad.d_pad.DOWN`
- **Left button** - `ctrl_pad.d_pad.LEFT`
- **Right button** - `ctrl_pad.d_pad.RIGHT`

Buttons

- **A button** - `ctrl_pad.buttons.A`
- **B button** - `ctrl_pad.buttons.B`
- **X button** - `ctrl_pad.buttons.X`
- **Y button** - `ctrl_pad.buttons.Y`
- **Start button** - `ctrl_pad.buttons.START`
- **Select button** - `ctrl_pad.buttons.SELECT`
- **Home button** - `ctrl_pad.buttons.HOME`
- **Left shoulder button** - `ctrl_pad.buttons.L`
- **Right shoulder button** - `ctrl_pad.buttons.R`
- **Z left button** - `ctrl_pad.buttons.ZL`
- **Z right button** - `ctrl_pad.buttons.ZR`

Analog Joy Sticks

The Wii Classic-style controllers have two analog joysticks. Each joystick outputs an `x` and `y` axis value.

- **Left joystick** - `ctrl_pad.joystick_l`
- **Right joystick** - `ctrl_pad.joystick_r`

You can assign two variables to each joystick object to read the `x` and `y` coordinates:

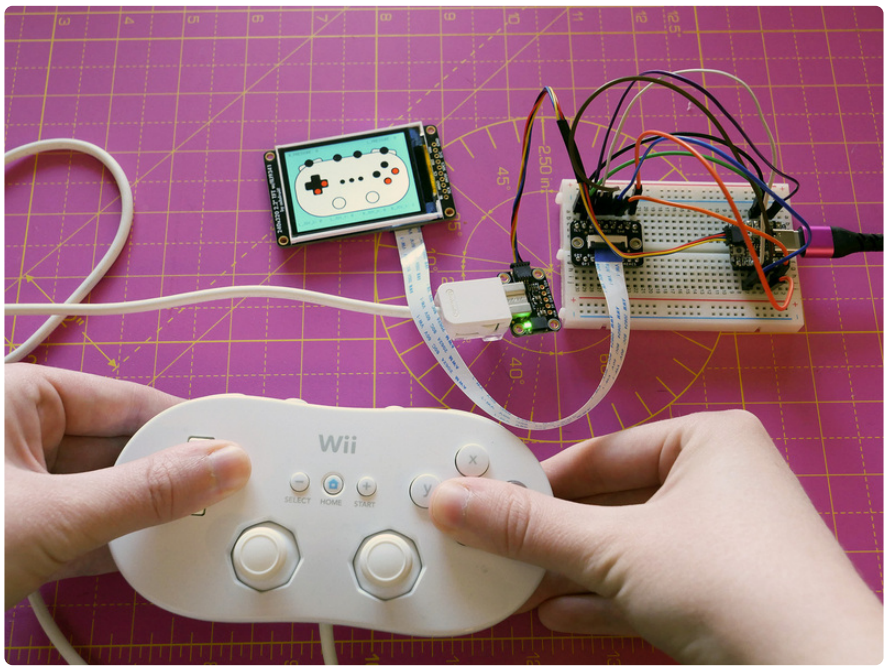
```
left_x, left_y = ctrl_pad.joystick_l
```

Analog Shoulder Button Force

The official Wii Classic controllers have analog sensors in the left and right shoulder buttons, which let you read the force being applied to the button before it is fully pressed.

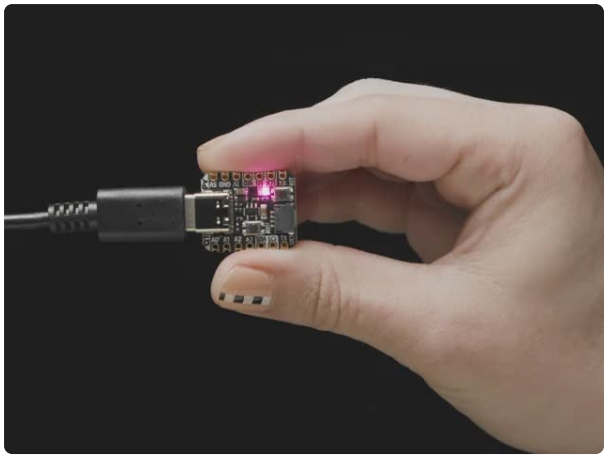
- **Left shoulder pressure** - `ctrl_pad.l_shoulder.LEFT_FORCE`
- **Right shoulder pressure** - `ctrl_pad.r_shoulder.RIGHT_FORCE`

Example Code



There are two examples included in the Wii Classic Controller library (https://adafru.it/18Cq): a simple test and a displayio visualizer. The simple test code listens for a few of the possible inputs from the controller and prints when they've been pressed to the REPL. The displayio visualizer displays a vector image of a controller to show when a button is pressed, as well as the x and y coordinates of the joysticks.

Both examples use a QT Py RP2040.



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

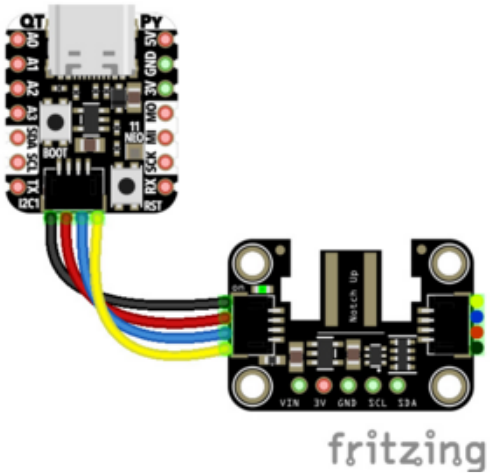
STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long

This 4-wire cable is a little over 100mm / 4" long and fitted with JST-SH female 4-pin connectors on both ends. Compared with the chunkier JST-PH these are 1mm pitch instead of... https://www.adafruit.com/product/4210

Adafruit Wii Nunchuck Breakout Adapter

Dig out that old Wii controller and use it as a sleek controller for your next robot if you like. The Adafruit Adafruit Wii Nunchuck Breakout Adapter fits snugly into the Wii connector... https://www.adafruit.com/product/4836

Simple Test Wiring



Plug the Nunchuck breakout into the STEMMA QT port on the QT Py RP2040.

Then, attach your controller to the Nunchuck breakout with the notch facing up towards the front of the breakout.

Library Installation

To use with CircuitPython, you need to first install the Wii Classic controller library, and its dependencies, into the **lib** folder onto your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

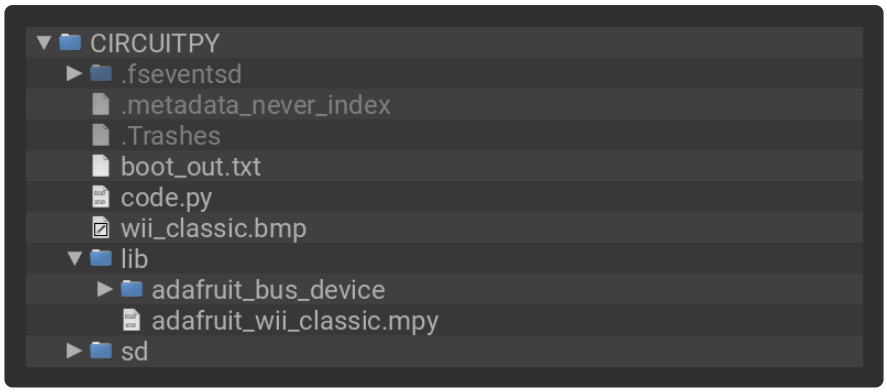
Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file.

Plug your CircuitPython compatible microcontroller into your computer via a known good USB cable. Be sure the cable has data+power wires. The board should show up as a new flash drive named **CIRCUITPY** in your computer File Explorer or Finder (depending on your operating system).

Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py** file to your **CIRCUITPY** drive.

Your **CIRCUITPY/lib** folder should contain the following folder and file:

- **adafruit_bus_device/**
- **adafruit_wii_classic.mpy**

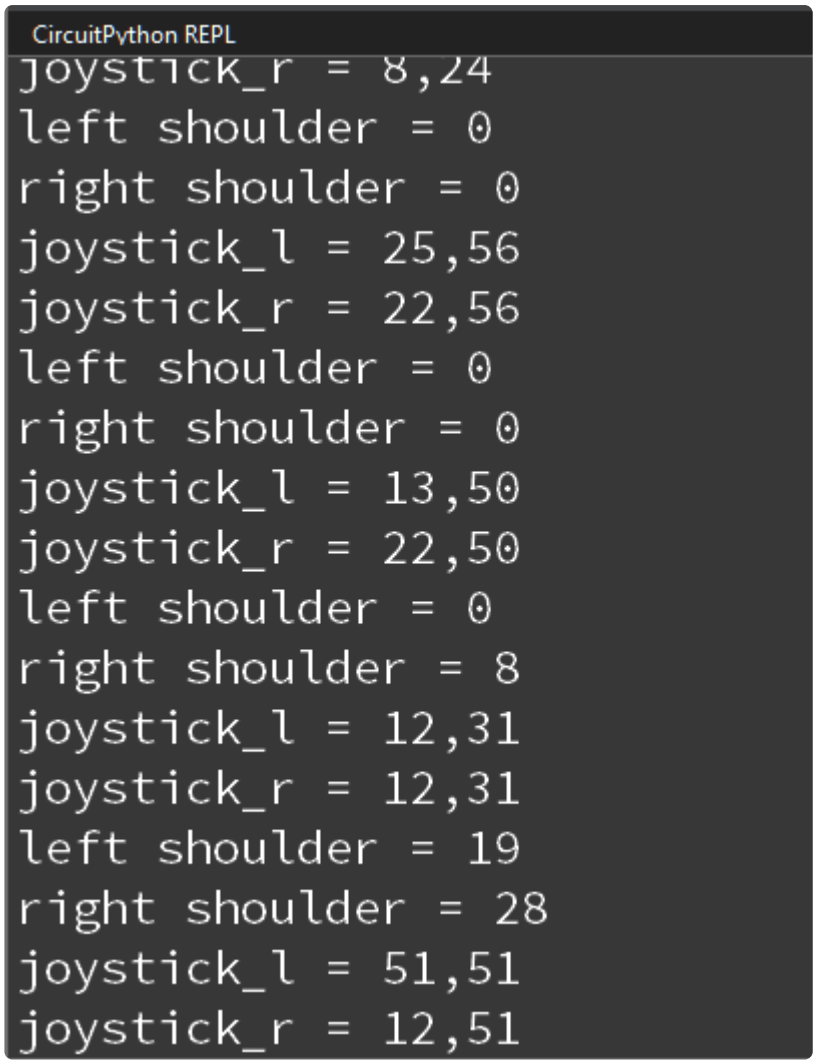


Simple Test Example

Once everything is saved to the **CIRCUITPY** drive, connect to the serial console (https://adafru.it/Bec) to see the data printed out!

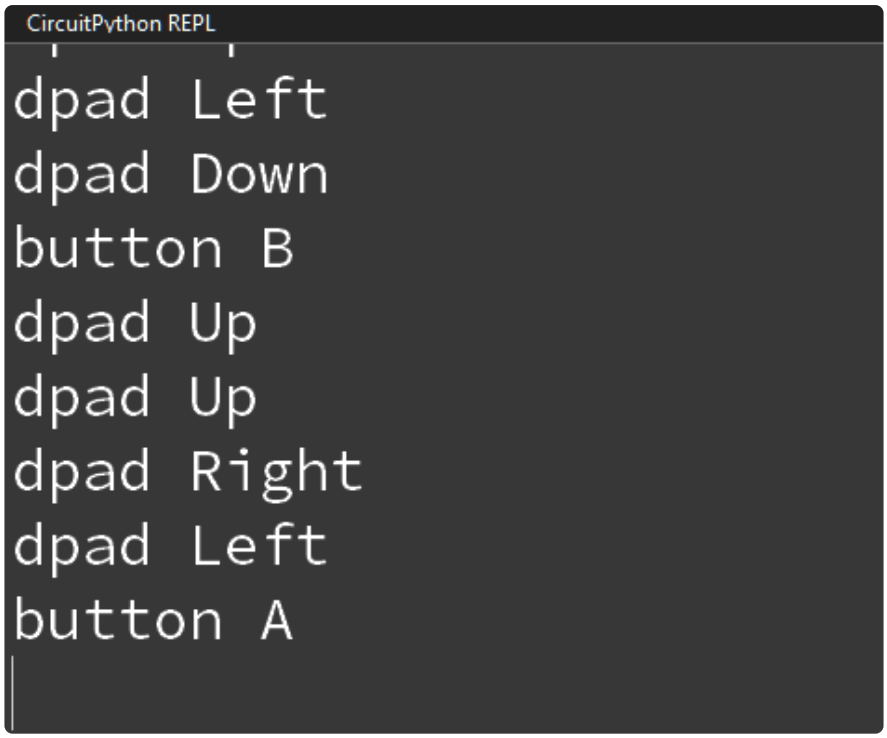
```
# SPDX-FileCopyrightText: Copyright (c) 2023 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import adafruit_wii_classic

i2c = board.STEMMA_I2C()
ctrl_pad = adafruit_wii_classic.Wii_Classic(i2c)

while True:
    left_x, left_y = ctrl_pad.joystick_l
    right_x, right_y = ctrl_pad.joystick_r
    left_pressure = ctrl_pad.l_shoulder.LEFT_FORCE
    right_pressure = ctrl_pad.r_shoulder.RIGHT_FORCE
    print("joystick_l = {},{}".format(left_x, left_y))
    print("joystick_r = {},{}".format(right_x, left_y))
    print("left shoulder = {}".format(left_pressure))
    print("right shoulder = {}".format(right_pressure))
    if ctrl_pad.buttons.A:
        print("button A")
    if ctrl_pad.buttons.B:
        print("button B")
    if ctrl_pad.d_pad.UP:
        print("dpad Up")
    if ctrl_pad.d_pad.DOWN:
        print("dpad Down")
    if ctrl_pad.d_pad.LEFT:
        print("dpad Left")
    if ctrl_pad.d_pad.RIGHT:
        print("dpad Right")
    time.sleep(0.5)
```

In the loop, the values of the two joysticks and the force readings from the two shoulder buttons are continuously printed to the REPL every 0.5 seconds. Please note that only the official Wii Classic controller has support for the shoulder button analog readings.

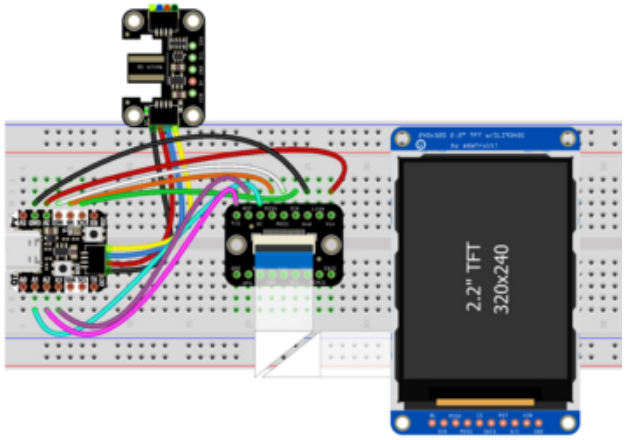


Please note that only the official Wii Classic controller has support for the shoulder button analog readings.

If you press any of the d-pad buttons or the A or B buttons, they will be printed to the REPL.



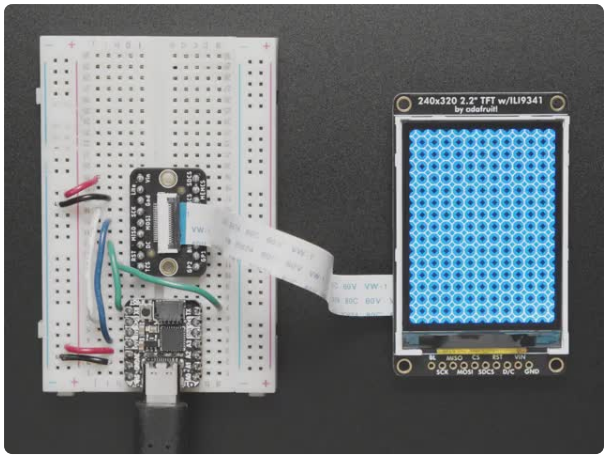
Displayio Visualizer Wiring



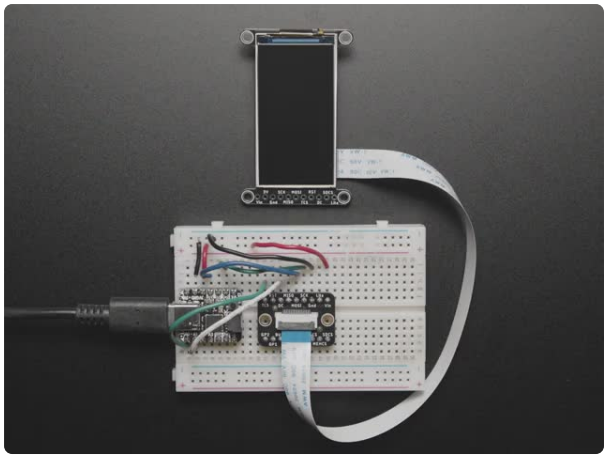
Connect the **Nunchuck breakout** to the **QT Py RP2040** via a **STEMMA QT cable**.

Then, connect the QT Py to the 2.2" TFT via an EYESPI breakout as follows.

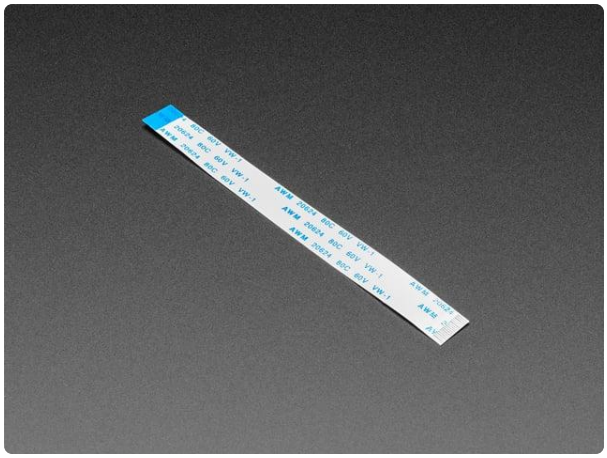
EYESPI Vin to **QT Py 3V (red wire)**
EYESPI GND to **QT Py GND (black wire)**
EYESPI SCK to **QT Py SCK (green wire)**
EYESPI MOSI to **QT Py MO (white wire)**
EYESPI MISO to **QT Py MI (orange wire)**
EYESPI DC to **QT Py A1 (cyan wire)**
EYESPI RST to **QT Py A3 (purple wire)**
EYESPI TCS to **QT Py A2 (pink wire)**



[2.2" 18-bit color TFT LCD display with microSD card breakout]
This lovely little display breakout is the best way to add a small, colorful, and bright display to any project. Since the display uses 4-wire SPI to communicate and has its own...
https://www.adafruit.com/product/1480



[Adafruit EYESPI Breakout Board - 18 Pin FPC Connector]
Our most recent display breakouts have come with a new feature: an 18-pin "EYE SPI" standard FPC...
https://www.adafruit.com/product/5613



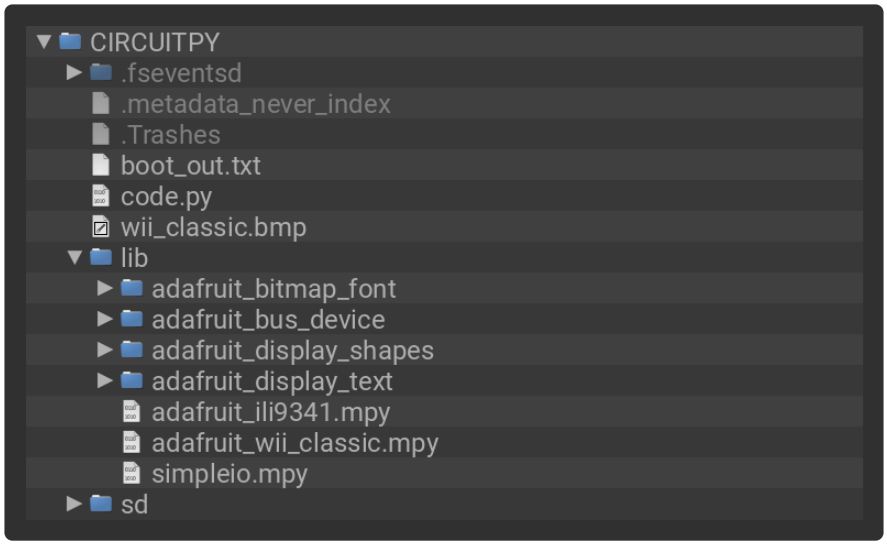
[EYESPI Cable - 18 Pin 100mm long Flex PCB (FPC) A-B type]
Connect this to that when a 18-pin FPC connector is needed. This 25 cm long cable is made of a flexible PCB. It's A-B style which means that pin one on one side will match...
https://www.adafruit.com/product/5239

Displayio Visualizer Example

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, and copy the **entire lib folder**, the **code.py** file and the **wii_classic.bmp** bitmap file to your **CIRCUITPY** drive.

Your **CIRCUITPY/lib** folder should contain the following files and folders:

- **adafruit_bitmap_font/**
- **adafruit_bus_device/**
- **adafruit_display_shapes/**
- **adafruit_display_text/**
- **adafruit_ili9341.mpy**
- **adafruit_wii_classic.mpy**
- **simpleio.mpy**




```
# SPDX-FileCopyrightText: 2023 Liz Clark for Adafruit Industries
# SPDX-License-Identifier: MIT

import board
import terminalio
import displayio

# Support 8.x.x and 9.x.x. Can be simplified after 8.x.x is discontinued as a
stable release.
try:
    from fourwire import FourWire
except ImportError:
    from displayio import FourWire

import simpleio
from adafruit_display_text import label
from adafruit_display_shapes.circle import Circle
import adafruit_ili9341
import adafruit_wii_classic

displayio.release_displays()

spi = board.SPI()
tft_cs = board.A2
tft_dc = board.A1

display_bus = FourWire(spi, command=tft_dc, chip_select=tft_cs, reset=board.A3)
display = adafruit_ili9341.ILI9341(display_bus, width=320, height=240)

bg = displayio.OnDiskBitmap("/wii_classic.bmp")
```

```
bg_tilegrid = displayio.TileGrid(bg, pixel_shader=bg.pixel_shader)

# Make the display context
splash = displayio.Group()
splash.append(bg_tilegrid)
display.root_group = splash

i2c = board.STEMMA_I2C()
ctrl_pad = adafruit_wii_classic.Wii_Classic(i2c)

RED = 0xFF0000
BLACK = 0x000000

button_spots = [
    {"label": "dpad_up", "pos": (68, 92), "size": 7, "color": RED},
    {"label": "dpad_down", "pos": (68, 132), "size": 7, "color": RED},
    {"label": "dpad_left", "pos": (48, 112), "size": 7, "color": RED},
    {"label": "dpad_right", "pos": (88, 112), "size": 7, "color": RED},
    {"label": "button_a", "pos": (277, 111), "size": 7, "color": RED},
    {"label": "button_b", "pos": (253, 137), "size": 7, "color": RED},
    {"label": "button_x", "pos": (252, 86), "size": 7, "color": RED},
    {"label": "button_y", "pos": (227, 111), "size": 7, "color": RED},
    {"label": "button_select", "pos": (136, 116), "size": 4, "color": RED},
    {"label": "button_home", "pos": (160, 116), "size": 4, "color": RED},
    {"label": "button_start", "pos": (184, 116), "size": 4, "color": RED},
    {"label": "button_zl", "pos": (134, 42), "size": 12, "color": RED},
    {"label": "button_zr", "pos": (188, 42), "size": 12, "color": RED},
    {"label": "button_lshoulder", "pos": (58, 44), "size": 12, "color": RED},
    {"label": "button_rshoulder", "pos": (259, 44), "size": 12, "color": RED},
]

overlays = []
for spot in button_spots:
    b = Circle(x0=spot["pos"][0], y0=spot["pos"][1], r=spot["size"],
fill=spot["color"])
    splash.append(b)
    overlays.append(b)

texts = [
    {"label": "l_x_text", "text": "L_JOY_X: 00", "x": 6, "y": 220, "color": BLACK},
    {"label": "l_y_text", "text": "L_JOY_Y: 00", "x": 88, "y": 220, "color": BLACK},
    {"label": "r_x_text", "text": "R_JOY_X: 00", "x": 173, "y": 220, "color":
BLACK},
    {"label": "r_y_text", "text": "R_JOY_X: 00", "x": 248, "y": 220, "color":
BLACK},
    {"label": "ls_text", "text": "L_PRESSURE: 00", "x": 10, "y": 11, "color":
BLACK},
    {"label": "rs_text", "text": "R_PRESSURE: 00", "x": 220, "y": 11, "color":
BLACK},
]

analog_text = []
for text in texts:
    t = label.Label(
        terminalio.FONT,
        text=text["text"],
        color=text["color"],
        x=text["x"],
        y=text["y"],
    )
    splash.append(t)
    analog_text.append(t)

last_l_x = 0
last_r_x = 0
last_l_y = 0
last_r_y = 0
last_r_press = 0
last_l_press = 0
```

```
while True:
    l_x, l_y = ctrl_pad.joystick_l
    mapped_l_x = simpleio.map_range(l_x, 7, 58, -50, 50)
    mapped_l_y = simpleio.map_range(l_y, 7, 58, -50, 50)
    r_x, r_y = ctrl_pad.joystick_r
    mapped_r_x = simpleio.map_range(r_x, 2, 26, -50, 50)
    mapped_r_y = simpleio.map_range(r_y, 3, 28, -50, 50)
    left_pressure = ctrl_pad.l_shoulder.LEFT_FORCE
    right_pressure = ctrl_pad.r_shoulder.RIGHT_FORCE
    if last_l_x != mapped_l_x:
        analog_text[0].text = "L_JOY_X: %d" % mapped_l_x
        last_l_x = mapped_l_x
    if last_l_y != mapped_l_y:
        analog_text[1].text = "L_JOY_Y: %d" % mapped_l_y
        last_l_y = mapped_l_y
    if last_r_x != mapped_r_x:
        analog_text[2].text = "R_JOY_X: %d" % mapped_r_x
        last_r_x = mapped_r_x
    if last_r_y != mapped_r_y:
        analog_text[3].text = "R_JOY_Y: %d" % mapped_r_y
        last_r_y = mapped_r_y
    if last_r_press != right_pressure:
        analog_text[4].text = "R_PRESSURE: %d" % right_pressure
        last_r_press = right_pressure
    if last_l_press != left_pressure:
        analog_text[5].text = "L_PRESSURE: %d" % left_pressure
        last_l_press = left_pressure
    if ctrl_pad.d_pad.UP:
        overlays[0].fill = RED
    else:
        overlays[0].fill = BLACK
    if ctrl_pad.d_pad.DOWN:
        overlays[1].fill = RED
    else:
        overlays[1].fill = BLACK
    if ctrl_pad.d_pad.LEFT:
        overlays[2].fill = RED
    else:
        overlays[2].fill = BLACK
    if ctrl_pad.d_pad.RIGHT:
        overlays[3].fill = RED
    else:
        overlays[3].fill = BLACK
    if ctrl_pad.buttons.A:
        overlays[4].fill = RED
    else:
        overlays[4].fill = BLACK
    if ctrl_pad.buttons.B:
        overlays[5].fill = RED
    else:
        overlays[5].fill = BLACK
    if ctrl_pad.buttons.X:
        overlays[6].fill = RED
    else:
        overlays[6].fill = BLACK
    if ctrl_pad.buttons.Y:
        overlays[7].fill = RED
    else:
        overlays[7].fill = BLACK
    if ctrl_pad.buttons.SELECT:
        overlays[8].fill = RED
    else:
        overlays[8].fill = BLACK
    if ctrl_pad.buttons.HOME:
        overlays[9].fill = RED
    else:
        overlays[9].fill = BLACK
    if ctrl_pad.buttons.START:
```

```
        overlays[10].fill = RED
    else:
        overlays[10].fill = BLACK
    if ctrl_pad.buttons.ZL:
        overlays[11].fill = RED
    else:
        overlays[11].fill = BLACK
    if ctrl_pad.buttons.ZR:
        overlays[12].fill = RED
    else:
        overlays[12].fill = BLACK
    if ctrl_pad.buttons.L:
        overlays[13].fill = RED
    else:
        overlays[13].fill = BLACK
    if ctrl_pad.buttons.R:
        overlays[14].fill = RED
    else:
        overlays[14].fill = BLACK
```

As you press the buttons on the controller, you'll see red circles appear on the picture of the controller. The `x` and `y` coordinates for each of the joysticks, as well as the force reading from the shoulder buttons (if applicable) are also displayed as text on the screen.

Python Docs

Python Docs (https://adafru.it/18Cv)