Wireless ESP32-S2 Touch Screen Controller for Pure Data

Created by Liz Clark

https://learn.adafruit.com/wireless-esp32-s2-controller-for-pure-data

Last updated on 2023-09-20 03:34:05 PM EDT
# Table of Contents

**Overview** 3
- Parts

**Circuit Diagram** 6

**3D Printing** 7

**Install Pure Data** 8

**CircuitPython** 9
- CircuitPython Quickstart

**Coding the ESP32-S2 Controller** 11
- Upload the Code and Libraries to the QT Py ESP32-S2
- secrets.py
- Open the Pure Data File

**How the CircuitPython Code Works** 16
- Socket Connection
- The Loop
- Reading the Touch Screen
- Reading the ADXL343
- Comparing Values
- Sending Data Over the Socket

**How the Pure Data Script Works** 19
- Reading Data from the QT Py ESP32-S2
- The Sequencer
- Start the Synth
- Loop Through the Sequencer
- The Sawtooth Synth
- VCF Filter and Audio Output

**Assembly** 22
- Solder the ESP32-S2 and Lipoly Charger BFF

**Connect and Mount the STEMMA Boards** 24
- Mount the Boards

**Mount the Touch Screen** 27

**Assemble the Case** 28

**Usage** 30
- Start the Synth
- Sequence Notes
- Adjust Filters
- Start and Stop the Synth

©Adafruit Industries
Overview

This guide will show you how to use a QT Py ESP32-S2 running CircuitPython to control a Pure Data patch over WiFi. Pure Data () is an open source programming language that can be used to create musical applications. It's a visual programming language that lets you drag and drop code blocks into the software interface to create the patches.

The QT Py ESP32-S2 is soldered to a charger BFF. This allows the project to run on a LiPo battery.

An ADXL343 accelerometer controls the synthesizer patch's filters. A TSC2007 and resistive touch screen are used to send notes to the synth patch.
The build is housed in a 3D printed case that can be held in your hand.

Parts

Adafruit QT Py ESP32-S2 WiFi Dev Board with STEMMA QT
What has your favorite Espressif WiFi microcontroller, comes with our favorite connector - the STEMMA QT, a chainable I2C port, and has...
https://www.adafruit.com/product/5325

Adafruit LiIon or LiPoly Charger BFF Add-On for QT Py
Is your QT Py all alone, lacking a friend to travel the wide world with? When you were a kid you may have learned...
https://www.adafruit.com/product/5397
1 x **USB Cable**
USB A to C cable, 1 meter, purple
https://www.adafruit.com/product/5153

2 x **STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long**
STEMMA QT cable
https://www.adafruit.com/product/4399

1 x **Lithium Ion Polymer Battery Ideal For Feathers - 3.7V 400mAh**
3.7V 400mAh LiPo
https://www.adafruit.com/product/3898

1 x **High Temperature Polyimide Tape - 1cm wide x**
Adafruit TSC2007 I2C Resistive Touch Screen Controller
Getting touchy performance with your screen's touch screen? Resistive touch screens are incredibly popular as overlays to TFT and LCD displays. Only problem is they require a bunch...
https://www.adafruit.com/product/5423

Resistive Touch screen - 3.7" Diagonal
Want to poke at your projects? This resistive touch screen can be used with a stylus or fingertip and is easy to use with a microcontroller. You can put it over a paper overlay for a...
https://www.adafruit.com/product/333

ADXL343 - Triple-Axis Accelerometer (+-2g/4g/8g/16g) w/ I2C/SPI
Analog Devices has followed up on their popular classic, the ADXL345, with this near-drop-in-replacement, the ADXL343. Like the original, this is a triple-axis accelerometer with...
https://www.adafruit.com/product/4097
33 meter roll
High Temperature Polyimide Tape

1 x Black Nylon Machine Screw and Stand-off Set – M2.5 Thread
M2.5 screws and stand-offs

https://www.adafruit.com/product/3057

https://www.adafruit.com/product/3299

Circuit Diagram

This project consists of STEMMA boards that are connected to each other using STEMMA QT cables.

- TSC2007
  - SCL (yellow wire) to ADXL343 SCL
  - SDA (blue wire) to ADXL343 SDA
  - VIN (red wire) to ADXL343 VIN
  - GND (black wire) to ADXL343 GND

- ADXL343
  - SCL (yellow wire) to QT Py SCL1
  - SDA (blue wire) to QT Py SDA1
  - VIN (red wire) to QT Py 3.3V
  - GND (black wire) to QT Py GND
The controller may be housed in a 3D printed case, described below. The case consists of four parts: a top lid, main body, bottom lid and handle. All parts print with no supports.

The STL files can be downloaded directly here or from Thingiverse.

ESP32_Controller_Case.zip

Thingiverse download

The main portion of the case snap fits together. The handle attaches to the main body with two M2.5 screws.
The bottom lid has mounting holes for the STEMMA boards and a slot for the QT Py ESP32-S2. The top lid has a slot for the touch screen's ribbon cable.

Install Pure Data

Pure Data can run on computers running Windows, Mac OS or Linux.

Open a browser and navigate to the Pure Data downloads page. Select the installer for your computer and download it.

After installing, launch Pure Data. A blank Pure Data window will open. This window acts as a console log when you are running a patch. You can print messages to it and get real time feedback.

Pure Data is open source and has a lot of community support for documentation. There is a getting started guide available on the Pure Data website that will be helpful if you are new to Pure Data
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.
Plug your board into your computer, using a known-good data-sync cable, directly, or via an adapter if needed.

Click the reset button once (highlighted in red above), and then click it again when you see the RGB status LED(s) (highlighted in green above) turn purple (approximately half a second later). Sometimes it helps to think of it as a "slow double-click" of the reset button.

On some very old versions of the UF2 bootloader, the status LED turns red instead of purple.

For this board, tap reset and wait for the LED to turn purple, and as soon as it turns purple, tap reset again. The second tap needs to happen while the LED is still purple.

Once successful, you will see the RGB status LED(s) turn green (highlighted in green above). If you see red, try another port, or if you’re using an adapter or hub, try without the hub, or different adapter or hub.

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

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 QTPYS2BOOT.

Drag the adafruit_circuitpython_etc.uf2 file to QTPYS2BOOT.

The BOOT drive will disappear and a new disk drive called CIRCUITPY will appear.

That's it!

Coding the ESP32-S2 Controller

Once you've finished setting up your QT Py ESP32-S2 with CircuitPython, you can access the code and necessary libraries by downloading the Project Bundle.

To do this, click on the Download Project Bundle button in the window below. It will download as a zipped folder.

```python
# SPDX-FileCopyrightText: 2022 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import ipaddress
import wifi
import socketpool
import board
import simpleio
import adafruit_tsc2007
```
import adafruit_adxl34x

# Get wifi details and host IP from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# I2C setup for STEMMA port
i2c = board.STEMMA_I2C()

# touchscreen setup for TSC2007
irq_dio = None
tsc = adafruit_tsc2007.TSC2007(i2c, irq=irq_dio)

# accelerometer setup
accelerometer = adafruit_adxl34x.ADXL343(i2c)
accelerometer.enable_tap_detection()

# MIDI notes - 2 octaves of Cmaj7 triad
notes = [48, 52, 55, 59, 60, 64, 67, 71]

# reads touch input
point = tsc.touch

# accelerometer x coordinate
acc_x = 0

# accelerometer y coordinate
acc_y = 0

# last accelerometer x coordinate
last_accX = 0

# last accelerometer y coordinate
last_accY = 0

# mapped value for touchscreen x coordinate
x_map = 0

# mapped value for touchscreen y coordinate
y_map = 0

# last mapped value for touchscreen x coordinate
last_x = 0

# last mapped value for touchscreen y coordinate
last_y = 0

# state for whether synth is running
run = 0

# tap detection state
last_tap = False

# new value detection state
new_val = False

# URLs to fetch from
HOST = secrets["host"]
PORT = 12345
TIMEOUT = 5
INTERVAL = 5
MAXBUF = 256

# connect to WIFI
print("Connecting to %s\n%ssesssstys["ssid"]")
wifi.radio.connect(secrets["ssid"], secrets["password"])
print("Connected to %s")

pool = socketpool.SocketPool(wifi.radio)
ipv4 = ipaddress.ip_address(pool.getaddrinfo(HOST, PORT)[0][4][0])

buf = bytearray(MAXBUF)

print("Create TCP Client Socket")
s = pool.socket(pool.AF_INET, pool.SOCK_STREAM)

print("Connecting")
s.connect((HOST, PORT))

while True:
    # tap detection
    # if tap is detected and the synth is not running...
    if accelerometer.events["tap"] and not last_tap and not run:
        # run is updated to 1
        run = 1
        # last_tap is reset
        last_tap = True
        print("running")
        # message is sent to Pd to start the synth
        # all Pd messages need to end with a ";"
        size = s.send(str.encode(' '.join(["run", str(run), ";"])))
    # if tap is detected and the synth is running...
    if accelerometer.events["tap"] and not last_tap and run:
        # run is updated to 0
        run = 0
        # last_tap is reset
        last_tap = True
        print("not running")
        # message is sent to Pd to stop the synth
        # all Pd messages need to end with a ";"
        size = s.send(str.encode(' '.join(["run", str(run), ";"])))

    # tap detection debounce
    if not accelerometer.events["tap"] and last_tap:
        last_tap = False

    # if the touchscreen is touched...
    if tsc.touched:
        # point holds touch data
        point = tsc.touch
        # x coordinate is remapped to 0 - 8
        x_map = simpleio.map_range(point["x"], 0, 4095, 0, 8)
        # y coordinate is remapped to 0 - 8
        y_map = simpleio.map_range(point["y"], 0, 4095, 0, 8)

        # accelerometer x value is remapped for synth filter
        acc_x = simpleio.map_range(accelerometer.acceleration[0], -10, 10, 450, 1200)
        # accelerometer y value is remapped for synth filter
        acc_y = simpleio.map_range(accelerometer.acceleration[1], -10, 10, 250, 750)

        # if any of the values are different from the last value...
        if x_map != last_x:
            # last value is updated
            last_x = x_map
            # new value is detected
            new_val = True
        if y_map != last_y:
            last_y = y_map
            new_val = True
        if int(acc_x) != last_accX:
            last_accX = int(acc_x)
            new_val = True
        if int(acc_y) != last_accY:
            last_accY = int(acc_y)
            new_val = True

        # if a new value is detected...
        if new_val:
            # note index is updated to y coordinate on touch screen
            note = notes[int(y_map)]
            # message with updated values is sent via socket to Pd
            # all Pd messages need to end with a ";"
            size = s.send(str.encode(' '.join(["x", str(x_map), ";","y", str(y_map), ";","aX", str(acc_x), ";","aY", str(acc_y), ";","n", str(note), ";"])))
# new_val is reset
new_val = False

Upload the Code and Libraries to the QT Py ESP32-S2

After downloading the Project Bundle, plug your QT Py ESP32-S2 into the computer's USB port with a known good USB data+power cable. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called CIRCUITYP. Unzip the folder and copy the following items to the QT Py ESP32-S2's CIRCUITYP drive.

- lib folder
- code.py

Your QT Py ESP32-S2 CIRCUITYP drive should look like this after copying the lib folder and the code.py file.

![CIRCUITYP Drive Contents]

secrets.py

You will need to create and add a secrets.py file to your CIRCUITYP drive. Your secrets.py file will need to include the following information:

```python
secrets = {
    'ssid': 'YOUR WIFI NETWORK NAME',
    'password': 'YOUR WIFI NETWORK PASSWORD',
    'host': 'YOUR COMPUTER IP ADDRESS',
    'timezone': "America/New_York", # http://worldtimeapi.org/timezones
}
```

'host' will hold the IP address of the computer running Pure Data that you are connecting to. The QT Py ESP32-S2 will be connecting to that computer with a socket connection.
To find your IP address on a Windows machine, refer to this documentation from Microsoft.

To find your IP address on a Mac, refer to this documentation from Apple.

To find your IP address on Linux, enter `ip addr` in a terminal.

Open the Pure Data File

Locate the Pure Data script file in the Project Bundle folder:

- pureDataWithESP32.pd

Launch the Pure Data application that you installed on your computer and open the file.
How the CircuitPython Code Works

First, the `STEMMA_I2C` port is setup for the I2C devices. Then the TSC2007 and ADXL343 are setup.

```python
# I2C setup for STEMMA port
i2c = board.STEMMA_I2C()

# touchscreen setup for TSC2007
irq_dio = None
tsc = adafruit_tsc2007.TSC2007(i2c, irq=irq_dio)

# accelerometer setup
accelerometer = adafruit_adxl34x.ADXL343(i2c)
accelerometer.enable_tap_detection()
```

A number of variables and states are declared. Their function is commented in the code.

Most importantly, the `notes` array holds the MIDI note numbers that are sent to the Pure Data script and played through the synth. If you want to play different notes, you'll want to edit that array.

```python
# MIDI notes - 2 octaves of Cmaj7 triad
notes = [48, 52, 55, 59, 60, 64, 67, 71]

# reads touch input
point = tsc.touch

# accelerometer x coordinate
acc_x = 0

# accelerometer y coordinate
acc_y = 0

# last accelerometer x coordinate
last_accX = 0

# last accelerometer y coordinate
last_accY = 0

# mapped value for touchscreen x coordinate
x_map = 0

# mapped value for touchscreen y coordinate
y_map = 0

# last mapped value for touchscreen x coordinate
last_x = 0

# last mapped value for touchscreen y coordinate
last_y = 0

# state for whether synth is running
run = 0

# tap detection state
last_tap = False

# new value detection state
new_val = False
```
Socket Connection

After the QT Py ESP32-S2 connects to your WiFi network, it connects to your computer with a socket connection. The socket needs to know your computer's IP address, which is stored in `host` in the secrets.py file. The port is defined as `12345`.

Once a socket connection is established with the Pure Data script, the loop begins to run.

```python
HOST = secrets["host"]
PORT = 12345
TIMEOUT = 5
INTERVAL = 5
MAXBUF = 256

# connect to WIFI
print("Connecting to %s\n"%secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print("Connected to %s\n"%secrets["ssid"])

pool = socketpool.SocketPool(wifi.radio)
ipv4 = ipaddress.ip_address(pool.getaddrinfo(HOST, PORT)[0][4][0])
buf = bytearray(MAXBUF)

print("Create TCP Client Socket")
s = pool.socket(pool.AF_INET, pool.SOCK_STREAM)

print("Connecting")
s.connect((HOST, PORT))
```

The Loop

The loop begins by checking for tap detection from the accelerometer. If the ADXL343 detects a tap, it will either start or stop the Pure Data synth from playing.

```python
# tap detection
# if tap is detected and the synth is not running...
if accelerometer.events["tap"] and not last_tap and not run:
    # run is updated to 1
    run = 1
    # last_tap is reset
    last_tap = True
    print("running")
    # message is sent to Pd to start the synth
    # all Pd messages need to end with a ";"
    size = s.send(str.encode(' '.join(["run", str(run), ";"])))

# if tap is detected and the synth is running...
if accelerometer.events["tap"] and not last_tap and run:
    # run is updated to 0
    run = 0
    # last_tap is reset
    last_tap = True
    print("not running")
    # message is sent to Pd to stop the synth
```
# all Pd messages need to end with a ";"
size = s.send(str.encode(' '.join(["run", str(run), ";"])))
# tap detection debounce
if not accelerometer.events["tap"] and last_tap:
    last_tap = False

Reading the Touch Screen

If a touch input is detected with the TSC2007, the x and y coordinates are mapped from their 12-bit value to a range of 0 to 8. This mapping matches with the sequencer array in the Pure Data script.

```python
# if the touchscreen is touched...
if tsc.touched:
    # point holds touch data
    point = tsc.touch
    # x coordinate is remapped to 0 - 8
    x_map = simpleio.map_range(point["x"], 0, 4095, 0, 8)
    # y coordinate is remapped to 0 - 8
    y_map = simpleio.map_range(point["y"], 0, 4095, 0, 8)
```

Reading the ADXL343

The x and y values from the ADXL343 are mapped to match values for the synth's filters in the Pure Data script.

```python
# accelerometer x value is remapped for synth filter
acc_x = simpleio.map_range(accelerometer.acceleration[0], -10, 10, 450, 1200)
# accelerometer y value is remapped for synth filter
acc_y = simpleio.map_range(accelerometer.acceleration[1], -10, 10, 250, 750)
```

Comparing Values

The current and previous values from the TSC2007 and ADXL343 are compared using if statements. If the values do not match, then the previous value is updated and the state of new_val is set to True. This indicates that a new value has been received and should be sent to the Pure Data script.

```python
# if any of the values are different from the last value...
if x_map != last_x:
    # last value is updated
    last_x = x_map
    # new value is detected
    new_val = True
if y_map != last_y:
    last_y = y_map
    new_val = True
if int(acc_x) != last_accX:
    last_accX = int(acc_x)
    new_val = True
if int(acc_y) != last_accY:
    last_accY = int(acc_y)
    new_val = True
```
last_accY = int(acc_y)
new_val = True

Sending Data Over the Socket

If a new value is received from one of the sensors, then a message is sent over the socket with the new readings. Each piece of data is preceded by a variable name that Pure Data will be listening for.

A new note is sent to the sequencer in the Pure Data patch with the `note` variable. The `y_map` coordinate on the touch screen is used as an index marker in the `notes` array.

It's important to note that Pure Data uses a networking protocol called FUDI (). As a result, each message needs to end with a semicolon (;
).

```python
# if a new value is detected...
if new_val:
    # note index is updated to y coordinate on touch screen
    note = notes[int(y_map)]
    # message with updated values is sent via socket to Pd
    # all Pd messages need to end with a ";"
    size = s.send(str.encode(' '.join(["x", str(x_map), ";",
    "y", str(y_map), ";",
    "aX", str(acc_x), ";",
    "aY", str(acc_y), ";",
    "n", str(note), ";"]))

    # new_val is reset
    new_val = False
```

How the Pure Data Script Works

Pure Data uses programming blocks and lines to build a functional script.
Reading Data from the QT Py ESP32-S2

At the top of the script, Pure Data begins listening to port 12345 and receiving data over the port using `netrecv`. The incoming data is sent from the QT Py ESP32-S2 with the socket message. This message is parsed using `route`. The message is divided by the variable names sent before each sensor value.

TSC2007 x coordinate: `x`
TSC2007 y coordinate: `y`
ADXL343 x value: `aX`
ADXL343 y value: `aY`
Run state: `run`
MIDI note number: `n`

Values are sent around a Pure Data script using `s` or `send`. Underneath `route`, you can see each individual value being sent with `s`.

The Sequencer

The `seq` array holds the received MIDI notes and are played in a loop. You can adjust the array's parameters using the `bounds`, `const` and `resize` messages.

Values are received as inputs using `receive` or `r`. `n` and `x` are received by the array using `r`. Those values are written to the array using `tabwrite`.

©Adafruit Industries
Start the Synth

The `run` affects whether the synth patch is playing or not. When a new `run` message is received from the socket, the patch will either begin playing or stop. Additionally, when it stops, a 0 is sent to all of the parameters so that the synth is muted using `s off`.

Loop Through the Sequencer

The `+ 1` and `% 8` blocks act as a for loop, advancing by 1 with a limit of 8. This lets the script iterate through the notes in the `seq` array like a sequencer. The `tabread` reads the MIDI notes in the `seq` array. Those note numbers are converted to frequencies using the MIDI to Frequency block (`mtof`). The frequency is sent to the synth.

The Sawtooth Synth

The synth is built using a sawtooth waveform, created with the `phasor~` block. It has a series of filters that are affected by the accelerometer.
VCF Filter and Audio Output

The final stop is a VCF filter, created with the `vcf~` block. The VCF's value is affected by the note playing in the `seq` array.

The `output~` block controls the audio output from the patch. You can change the volume by adjusting the `dB` block and mute all audio by clicking next to `mute`.

Assembly
Solder the ESP32-S2 and Lipoly Charger BFF

Solder the short end of the headers to the LiPo BFF.

Place the QT Py ESP32-S2 onto the headers on the LiPo BFF. Make sure that the pins are matched. The QT Py's USB port should be on the other side of the LiPo BFF's LiPo port.

Solder the QT Py to the headers.
Your QT Py ESP32-S2 is ready to be battery-powered!

Connect and Mount the STEMMA Boards

The STEMMA boards are connected to each other and the QT Py ESP32-S2 with 50mm QT to QT cables.

Plug the TSC2007’s right socket into the ADXL343’s right socket.
Plug the ADXL343’s left socket into the QT Py ESP32-S2’s STEMMA socket.

Mount the Boards

Insert M2.5 screws into the eight mounting holes in the bottom lid of the case.

Attach M2.5 stand-offs to each of the eight M2.5 screws.
Slide the QT Py ESP32-S2 into the slot on the bottom lid of the case. The QT Py's USB port should be facing out towards the open area of the lid.

Attach the ADXL343 to the four stand-offs next to the QT Py's slot with four M2.5 screws.

Attach the TSC2007 to the four remaining stand-offs with M2.5 screws.
Mount the Touch Screen

The top lid of the 3D printed case is sized to attach the touch screen to it.

On the back of the touch screen, place a mounting sticker in each corner. If you do not have mounting stickers, you can use pieces of looped tape.

Carefully run the touch screen’s ribbon cable through the slot on the top lid. Then, attach the touch screen to the top lid with the mounting stickers.
Assemble the Case

Attach the handle to the main body of the case with two M2.5 screws and two M2.5 nuts.
Attach the bottom lid of the case to the main body.

Plug the touch screen's ribbon cable into the TSC2007's socket.
Plug a LiPo battery into the LiPo BFF. Attach the battery with electrical tape or kapton tape so that it is secured in the case.

Now you're ready to rock!

Usage

Start the Synth

Turn on the QT Py ESP32-S2 controller with the on/off switch on the LiPo BFF.
Open the Pure Data script. The QT Py ESP32-S2 will automatically connect and begin sending data from the sensors.

Sequence Notes

Touch different spots on the touch screen to send different combinations of notes to the Pure Data patch. You'll see them logged in the `seq` array.

Adjust Filters
Twist and turn the controller to send accelerometer data to affect the filters on the synth patch.

**Start and Stop the Synth**

The ADXL343 has tap detection. Tap the side of the enclosure to start and stop the synth patch.