Split Ortho Keyboard with TCA8418 Matrix Expanders

Created by John Park

https://learn.adafruit.com/split-ortho-keyboard

Last updated on 2023-08-29 04:39:23 PM EDT
Table of Contents

Overview
- Parts

Build the Split Ortho Keyboard
- Setup
- Wire Columns and Rows
- QT Py Prep
- Case Build
- 3D Files
- 3D/2D Mix Files
- Frame Prep
- Base Prep
- Switch Plate
- Attach Keys To PCBs
- Right Side Mounting
- STEMMA QT Cable Right Side
- QT Py Mount
- STEMMA QT Connection
- Case Top
- Connect the Halves
- USB Cable

CircuitPython
- CircuitPython Quickstart
- Safe Mode
- Flash Resetting UF2

Code the Split Ortho Keyboard
- Text Editor
- Download the Project Bundle
- How It Works
- Libraries
- HID Setup
- I2C Setup
- Matrix Pins
- Main Loop
- keymaps.py
Overview

Forget about that staggered key layout -- a holdover from mechanical typewriters -- and build your own custom ortholinear keyboard! Honor the angle of your wrists with the split design.

This 60% layout is fast and efficient (once you practice and get used to a new way of working) -- keyboard "layers" allow you to access any key you need by "lowering" or "raising" to whole new sets of keys.
Pick your favorite MX compatible keyswitches and press them into the socketed 6x5 NeoKey PCB. Build a 3D printed or 3D printed/laser cut acrylic combo case.

Two TCA8418 matrix expanders handle reading the diode key matrices, and a Kee Boar KB2040 running CircuitPython does the rest, acting as a USB HID keyboard device.

Create and edit your own custom layouts with the keymaps.py config file. The layout I used was inspired by this keyboard project.

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
2 x Adafruit TCA8418 Keypad Matrix and GPIO Expander Breakout https://www.adafruit.com/product/4918

2 x NeoKey 5x6 Ortho Snap-Apart Mechanical Key Switch PCB NeoKey 5x6 Ortho Snap-Apart Mechanical Key Switches https://www.adafruit.com/product/5157

6 x Kailh Mechanical Key Switches Linear Black 10 pack https://www.adafruit.com/product/4953

You'll want to pick some keycaps. If you are highly elite and need no legends, try one of the choices below. Otherwise, search online for "MX compatible ortholinear keycap set". The ones used in this guide are the Drop MT3 profile Dancer ortho kit.

6 x Blue DSA Keycaps 10 pack https://www.adafruit.com/product/5005

6 x Dark Blue DSA Keycaps 10 pack https://www.adafruit.com/product/5016

or

12 x Cyan MA Keycaps 5 pack https://www.adafruit.com/product/5174

STEMMA QT / Qwiic JST SH 4-Pin Cable - 400mm long
This 4-wire cable is a little over 400mm / 15.7" 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/5385
Brass Heat-Set Inserts for Plastic - M3 x 4mm - 50 pack
Wanna improve the connection strength between your project's 3D-printed parts, and also have nice clean surfaces? Instead of gluing bits together, or screwing plastic screws...
https://www.adafruit.com/product/4255

Black Nylon Machine Screw and Stand-off Set – M3 Thread
Totaling 420 pieces, this M3 Screw Set is a must-have for your workstation. You’ll have enough screws, nuts, and hex standoffs to fuel...
https://www.adafruit.com/product/4685

Black Nylon Machine Screw and Stand-off Set – M2.5 Thread
Totaling 380 pieces, this M2.5 Screw Set is a must-have for your workstation. You’ll have enough screws, nuts, and hex standoffs to fuel your maker...
https://www.adafruit.com/product/3299

2 x Little Rubber Bumper Feet
Pack of 4
https://www.adafruit.com/product/550
Build the Split Ortho Keyboard

Setup

Both sides of the Split Ortho Keyboard contain their own 6x5 diode matrix keyswitch PCB wired to the column and row pins of a TCA8418 keypad matrix expander.

Each TCA8418 takes care of handling keypress decoding and sends the event queue to the QT Py RP2040 over I2C.

Since only one address is possible on the TCA8418 you'll use the two independent I2C buses of the QT Py RP2040.
Wire Columns and Rows

Solder a wire to each column and row pad on the 6x5 PCBs.

Run these to their corresponding column and row pins on the TCA8418.

Repeat this for the second side.
QT Py Prep
In order to plug the STEMMA QT cable into the second I2C bus, solder the STEMMA QT/Qwiic breakout to the SDA, SCL, 3V, and GND pins as shown here.

Note, both sockets on the breakout run to the same on-board I2C bus, so one TCA8418 will plug into the breakout, while the other will plug into the QT Py's on-board STEMMA QT connector.

Case Build
You have two options for the case build -- fully 3D printed or mixed 3D printed with laser cut/CNC acrylic (or other materials).

Download the files from the link below.

3D and 2D fabrication files

3D Files
If you go with the fully 3D build, print the following files in PLA at ~0.2mm layer height and ~10% infill:

- sok_plate x2
- sok_top x2
- sok_base_L
- sok_baseR
3D/2D Mix Files

If you opt for a mix of 3D printed and 2D, here are the files you'll use:

3D Printed

- sok_qt
- sok_mid x2

2D Laser Cut/CNC Milled

- sok_curves x2
Frame Prep

Use your soldering iron to press a threaded M3 brass heat set insert into each of the six mounting holes of the base walls. You use these to screw on the top plate later.

If doing the 3D + laser cut frame, insert a second set of inserts on the back side as well so you can screw on the top plate and the back plate.
Base Prep
Screw in M2.5 stand-offs to support the PCB and breakout board as shown.

You can also add rubber bumper feet.

Then, screw the back plate on using M3 screws (only necessary for the hybrid case build).
Switch Plate

To stabilize the keyswitches, snap them into the switch plate. Be sure their header pins are all oriented in the same direction as shown here.
Attach Keys To PCBs
Carefully align all the pins to the sockets and gently, slowly press the keys into place.

This takes patience and massaging, so maybe put on some soothing music and light a good candle first.

Then add your keycaps. (Ignore the photo that shows the caps already in place during the first step.)
Right Side Mounting

On the right side, mount just the TCA8418 and the keyswitch PCB as shown.
If you don't think you want to fully disconnect the halves, plug in the STEMMA QT cable as shown. This provides a neater appearance and allows you to push the cable in to adjust the slack.

Click the QT Py into its mount, facing the base. This will allow access to the reset and boot buttons through the case bottom's holes.
STEMMA QT Connection

Connect the long STEMMA QT cable for the right half keyboard to the STEMMA QT/Qwiic breakout, using the standoff hardware as routing/strain relief.

Plug the short STEMMA QT cable from the QT Py to the TCA8418.
Case Top
Add the case top and screw it down using M3 screws into the threaded inserts.
Connect the Halves
If you didn't connect the right half STEMMA QT cable before closing the case, do so at this time.
USB Cable
Plug in a USB-C cable and you’re ready to code and use the keyboard.

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 Split Ortho Keyboard

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, the code.py file and the keymaps.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.
import time
import board
from adafruit_tca8418 import TCA8418
import usb_hid
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode
from keymaps import layer_keymaps # keymaps are saved in keymaps.py file

num_layers = len(layer_keymaps)
current_layer = 1

for tca in tcas:
    for pin in KEYPADPINS:
        tca.keypad_mode[pin] = True
        tca.enable_int[pin] = True
        tca.event_mode_fifo[pin] = True
        tca.key_intenable = True

print("Ortho Split Keyboard")

while True:
    for i in range(len(tcas)):
        tca = tcas[i] # get the TCA we're working with
        keymap = layer_keymaps[current_layer][i] # get the corresponding keymap

        if tca.key_int:
            events = tca.events_count
            for _ in range(events):
                keyevent = tca.next_event
                keymap_number = (keyevent & 0x7F)
                (modifier, keycode) = keymap[keymap_number]  # get keycode & modifer from keymap
                print("Key event: 0x%02X - key #%d " % (keyevent, keyevent & 0x7F))
                if keycode is None:
                    pass
                else:
                    if keyevent & 0x80:  # if key is pressed
                        if modifier == 0:  # normal keypress
                            kbd.press(keycode)
                        elif modifier == 1:  # lower
                            current_layer = min(max((current_layer-1), 0), num_layers-1)
                        elif modifier == 2:  # raise
                            current_layer = min(max((current_layer+1), 0), num_layers-1)
                        kbd.release_all()
num_layers -1)

elif modifier == 7:  # cap mod
    kbd.press(Keycode.SHIFT, keycode)

else:  # key released
    if modifier == 7:  # capped shifted key requires special
        kbd.release(Keycode.SHIFT, keycode)
    else:
        kbd.release(keycode)

tca.key_int = True  # clear the IRQ by writing 1 to it
time.sleep(0.01)

How It Works

The keyboard uses two TCA8418 expanders to read the matrix columns and rows of the keyboard halves. The events are queued up and sent over I2C to the QT Py RP2040 which then correlates each keypress with a keycode from the keymaps.py file. These keypresses are then sent to the computer as USB HID keys.

Libraries

You'll import libraries to provide functionality in the code:

- time
- board
- adafruit_tca8418
- usb_hid
- adafruit_hid.keyboard
- adafruit_hid.keycode

The keymaps file is imported as well so that the layer_keymaps can be accessed inside of code.py

HID Setup

The USB HID keyboard is set up along with variables for the key layers.
I2C Setup

You'll use both I2C channels on the QT Py RP2040 to connect to the two TCA8418 boards. Since they share the same address which cannot be changed, they can't be on the same I2C bus.

```python
i2c_left = board.STEMMA_I2C()  # uses QT Py RP2040 STEMMA QT port
i2c_right = board.I2C()  # I2C channel on the QT Py RP2040 pads broken out on board
tca_left = TCA8418(i2c_left)
tca_right = TCA8418(i2c_right)
tcas = (tca_left, tca_right)  # put the TCA objects in a list for easy iteration later
```

Matrix Pins

The column and row pins of the TCA8418 are fixed, so we'll specify the ones we're using for the six columns and five rows of the keyboard halves.

Then, each pin is set to **keypad_mode** with **enable** and **fifo** (first in, first out) set.

```python
KEYPADPINS = (TCA8418.R0, TCA8418.R1, TCA8418.R2, TCA8418.R3, TCA8418.R4,
              TCA8418.C0, TCA8418.C1, TCA8418.C2, TCA8418.C3, TCA8418.C4,
              TCA8418.C5 )
for tca in tcas:
    for pin in KEYPADPINS:
        tca.keypad_mode[pin] = True
        tca.enable_int[pin] = True
        tca.event_mode_fifo[pin] = True
        tca.key_intenable = True
```

Main Loop

The main loop of the program checks each TCA8418 for events in the queue. If a key has been pressed or released it is checked against the keymap file to see which keycode to press or release. These can vary depending on modifiers and layers as well.

```python
while True:
    for i in range(len(tcas)):
        tca = tcas[i]  # get the TCA we're working with
        keymap = layer_keymaps[current_layer][i]  # get the corresponding keymap
        for it
```
if tca.key_int:
  events = tca.events_count
  for _ in range(events):
    keyevent = tca.next_event
    keymap_number = (keyevent & 0x7F)
    (modifier, keycode) = keymap[keymap_number]  # get keycode &
    # print("\tKey event: 0x%02X - key #%d " % (keyevent, keyevent &
    0x7F))
    if keycode is None:
      pass
    else:
      if keyevent & 0x80:  # if key is pressed
        if modifier == 0:  # normal keypress
          kbd.press(keycode)
        elif modifier == 1:  # lower
          current_layer = min(max((current_layer-1), 0), num_layers-1)
        elif modifier == 2:  # raise
          current_layer = min(max((current_layer+1), 0), num_layers-1)
        elif modifier == 7:  # cap mod
          kbd.press(Keycode.SHIFT, keycode)
      else:  # key released
        if modifier == 7:  # capped shifted key requires special
          handling
          kbd.release(Keycode.SHIFT, keycode)
        else:
          kbd.release(keycode)
  tca.key_int = True  # clear the IRQ by writing 1 to it
  time.sleep(0.01)

keymaps.py

# SPDX-FileCopyrightText: Copyright (c) 2022 John Park & Tod Kurt for Adafruit
# Industries
#
# SPDX-License-Identifier: MIT
from adafruit_hid.keycode import Keycode
# https://docs.circuitpython.org/projects/hid/en/latest/api.html#adafruit-hid-
keycode-keycode
# keymap is keynumber, (modifier, keycode)
# lower keymap layer
km_lf_0 = {
  (1) : (0, Keycode.F11),
  (2) : (0, Keycode.F1),
  (3) : (0, Keycode.F2),
  (4) : (0, Keycode.F3),
  (5) : (0, Keycode.F4),
  (6) : (0, Keycode.F5),
  (11) : (0, None),
  (12) : (0, None),
  (13) : (0, None),
  (14) : (0, None),
  (15) : (0, None),
  (16) : (0, None),
  (21) : (0, None),
  (22) : (0, None),
  (23) : (0, None),
  (24) : (0, None),
  ...  # rest of the keymap
km_rt_0 = {
    1 : (0, Keycode.F6),
    2 : (0, Keycode.F7),
    3 : (0, Keycode.F8),
    4 : (0, Keycode.F9),
    5 : (0, Keycode.F10),
    6 : (0, Keycode.F12),
    11 : (0, Keycode.HOME),
    12 : (0, Keycode.PAGE_DOWN),
    13 : (0, Keycode.PAGE_UP),
    14 : (0, Keycode.END),
    15 : (0, Keycode.INSERT),
    16 : (0, Keycode.DELETE),
}

km_lf_1 = {
    1 : (0, Keycode.GRAVE_ACCENT),
    2 : (0, Keycode.ONE),
    3 : (0, Keycode.TWO),
    4 : (0, Keycode.THREE),
    5 : (0, Keycode.FOUR),
    6 : (0, Keycode.FIVE),
    11 : (0, Keycode.ESCAPE),
    12 : (0, Keycode.Q),
    13 : (0, Keycode.W),
    14 : (0, Keycode.E),
}

# main keymap layer
km_lf_1 = {
    1 : (0, Keycode.GRAVE_ACCENT),
    2 : (0, Keycode.ONE),
    3 : (0, Keycode.TWO),
    4 : (0, Keycode.THREE),
    5 : (0, Keycode.FOUR),
    6 : (0, Keycode.FIVE),
    11 : (0, Keycode.ESCAPE),
    12 : (0, Keycode.Q),
    13 : (0, Keycode.W),
    14 : (0, Keycode.E),
}
km_rt_1 = {
  (1) : (0, Keycode.SIX),
  (2) : (0, Keycode.SEVEN),
  (3) : (0, Keycode.EIGHT),
  (4) : (0, Keycode.NINE),
  (5) : (0, Keycode.ZERO),
  (6) : (0, Keycode.BACKSPACE),
  (11) : (0, Keycode.Y),
  (12) : (0, Keycode.U),
  (13) : (0, Keycode.I),
  (14) : (0, Keycode.O),
  (15) : (0, Keycode.P),
  (16) : (0, Keycode.BACKSLASH),
  (21) : (0, Keycode.H),
  (22) : (0, Keycode.J),
  (23) : (0, Keycode.K),
  (24) : (0, Keycode.L),
  (25) : (0, Keycode.SEMICOLON),
  (26) : (0, Keycode.QUOTE),
  (31) : (0, Keycode.N),
  (32) : (0, Keycode.M),
  (33) : (0, Keycode.COMMA),
  (34) : (0, Keycode.PERIOD),
  (35) : (0, Keycode.FORWARD_SLASH),
  (36) : (0, Keycode.ENTER),
  (41) : (0, Keycode.SPACE),
  (42) : (2, Keycode.R),  # raise
  (43) : (0, Keycode.LEFT_ARROW),
  (44) : (0, Keycode.DOWN_ARROW),
  (45) : (0, Keycode.UP_ARROW),
  (46) : (0, Keycode.RIGHT_ARROW)
}

# upper keymap layer
km_lf_2 = {
  (1) : (0, None),
  (2) : (0, None),
  (3) : (0, None),
  (4) : (0, None),
  (5) : (0, None),
  (15) : (0, Keycode.R),
  (16) : (0, Keycode.T),
  (21) : (0, Keycode.TAB),
  (22) : (0, Keycode.A),
  (23) : (0, Keycode.S),
  (24) : (0, Keycode.D),
  (25) : (0, Keycode.F),
  (26) : (0, Keycode.G),
  (31) : (0, Keycode.SHIFT),
  (32) : (0, Keycode.Z),
  (33) : (0, Keycode.X),
  (34) : (0, Keycode.C),
  (35) : (0, Keycode.V),
  (36) : (0, Keycode.B),
  (41) : (0, Keycode.CONTROL),
  (42) : (0, Keycode.GUI),
  (43) : (0, Keycode.ALT),
  (44) : (0, Keycode.GUI),
  (45) : (1, Keycode.L),  # lower
  (46) : (0, Keycode.SPACE)
}
km_rt_2 = {
    (1) : (0, None),
    (2) : (0, None),
    (3) : (0, None),
    (4) : (0, None),
    (5) : (0, None),
    (6) : (0, Keycode.BACKSPACE),

    (11) : (0, None),
    (12) : (0, None),
    (13) : (0, None),
    (14) : (0, None),
    (15) : (0, None),
    (16) : (0, Keycode.BACKSLASH),

    (21) : (0, None),
    (22) : (0, None),
    (23) : (0, None),
    (24) : (0, Keycode.LEFT_BRACKET),
    (25) : (0, Keycode.RIGHT_BRACKET),
    (26) : (0, Keycode.QUOTE),

    (31) : (0, None),
    (32) : (7, Keycode.LEFT_BRACKET),
    (33) : (7, Keycode.RIGHT_BRACKET),
    (34) : (0, None),
    (35) : (0, Keycode.BACKSPACE),
    (36) : (0, Keycode.ENTER),

    (41) : (0, Keycode.SPACE),
    (42) : (2, Keycode.R),  # raise
    (43) : (0, Keycode.LEFT_ARROW),
    (44) : (0, Keycode.DOWN_ARROW),
    (45) : (0, Keycode.UP_ARROW),
    (46) : (0, Keycode.RIGHT_ARROW)
}
The keymaps.py file first imports the `adafruit_hid.keycode` library.

```python
from adafruit_hid.keycode import Keycode
```

The file is organized by keymap sides (left/right), layers (0, 1, 2), and row clusters (1-6, 11-16, 21-26, etc.) These are dictionaries which have a "key" which corresponds to the key number as reported by the TCA8418, and a value tuple which tells the code two things: which modifier, and which keycode to use.

The modifier is usually 0 which means the keycode is sent as a normal keypress/release.

Modifier 1 means the "lower" key has been pressed, so a different keymap set is to be used until the "raise" key is pressed.

Modifier 2 means "raise"

Modifier 7 means a `Keycode.SHIFT` + the keycode are pressed at the same time. Normally you use the shift key to get this effect, but in some cases you may want to assign a key that is normally only invoked via shift. For example, a `{` key is usually shift + `Keycode.LEFT_BRACKET`, but if you want a key to press `{` all on its own, use modifier 7.