Tandy 1000 Keyboard to USB with CircuitPython

Created by Jeff Epler


Last updated on 2023-08-29 04:39:17 PM EDT
# Table of Contents

## Overview
- Homework
- Jeff's Verdict
- Parts

## 3D Printing & Wiring
- Wiring & Assembly
- Solder the DIN connector
- Install the connector & QT Py
- Solder the QT Py

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

## Coding the Keyboard
- Text Editor
- Download the Project Bundle
- How it works

## Gallery
Overview

In this guide, you'll see how to adapt a Tandy 1000 keyboard for use on modern computers using CircuitPython and an Adafruit QT Py RP2040 microcontroller board.

The techniques in this guide may also be helpful in converting other classic keyboards that use a clocked serial protocol. The RP2040's "pio" peripheral is perfect for this kind of low level I/O task, and even provides a buffer for up to 8 key events so no keypresses get lost.

Homework

I received this keyboard with no documentation. Happily, computers of this era were pretty well documented. The [technical reference manual is on archive.org](https://archive.org/) and I was quickly able to discover that the keyboard outputs a clock signal and a data signal, and each event is exactly 8 bits long.

Thanks to Tandy perservationists, I also found [a list of the scan codes](https://example.com) which are needed to convert the keycodes into USB HID codes, and [pinout of the keyboard's DIN-8 connector](https://example.com). I also found a [really cool video that showed the process of restoring one of these machines](https://example.com). In the [Part II video you can see the wild keyswitch mechanism up close](https://example.com).
That's enough information to get started wiring an adapter and writing code!

Jeff's Verdict

With almost every key from a standard "104-key keyboard", you really could use this as an everyday input device, especially if you have fond memories of a Tandy computer.

Pros:

- The keys have a nice feel, with a pleasant clack when the keyswitches bottom out
- Neat design of keyboard feet lets you select a tilted or flat keyboard angle
- The build is solid, while not as thoroughly over-built as early IBM keyboards
- Almost a full set of keys compared to a 101-key keyboard
- The keyboard's own controller has good anti-ghosting built in and supports SOME but not all 4KRO combinations. All of W+A+S+D+shift+control+alt can be pressed simultaneously

Cons:

- Unusual location of some keys, especially alt
- Caps Lock & Num Lock keys cannot be used as general keys
- Missing windows/menu keys (suggest making "hold" the Windows key)
- Doesn't support full NKRO
- Rare & expensive with fragile and unique keyswitches
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

Pink and Purple Woven USB A to USB C Cable - 2 meters long
This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also made for USB C for our modernized breakout boards, Feathers and more.
https://www.adafruit.com/product/5044

USB C to USB C Cable - USB 3.1 Gen 4 with E-Mark - 1 meter long
As technology changes and adapts, so does Adafruit! Rather than the regular USB A, this cable has USB C to USB C plugs!USB C is the latest...
https://www.adafruit.com/product/4199
Silicone Cover Stranded-Core Wire - 50ft 30AWG Black
Silicone-sheathing wire is super-flexible and soft, and its also strong! Able to handle up to 200°C and up to 600V, it will do when PVC covered wire wimps out. We like this wire...
https://www.adafruit.com/product/3164

Silicone Cover Stranded-Core Wire - 50ft 30AWG Red
Silicone-sheathing wire is super-flexible and soft, and its also strong! Able to handle up to 200°C and up to 600V, it will do when PVC covered wire wimps out. We like this wire...
https://www.adafruit.com/product/3165

Silicone Cover Stranded-Core Wire - 50ft 30AWG Blue
Silicone-sheathing wire is super-flexible and soft, and its also strong! Able to handle up to 200°C and up to 600V, it will do when PVC covered wire wimps out. We like this wire...
https://www.adafruit.com/product/3166

Silicone Cover Stranded-Core Wire - 50ft 30AWG Yellow
Silicone-sheathing wire is super-flexible and...
https://www.adafruit.com/product/3167
1 x SD-80LS
Female 8-pin DIN connector, solder eyelets, panel mount

Silicone Cover Stranded-Core Wire - 50ft 30AWG Green
Silicone-sheathing wire is super-flexible and...
https://www.adafruit.com/product/3168

Silicone Cover Stranded-Core Wire - 50ft 30AWG White
Silicone-sheathing wire is super-flexible and...
https://www.adafruit.com/product/3169

3D Printing & Wiring
I made a simple 3D printed bracket for the QT Py and keyboard connector and printed it using standard settings. If you want to modify the bracket, you can pick up the design files which are in OpenSCAD format.

![qt-tandy.stl](qt-tandy.stl)

Design files (Open SCAD)

The QT Py clip in design is taken from the guide "[QT Py Snap Fit Case ()](https://www.adafruit.com/blog/post/9269/)".

Wiring & Assembly

First, wire the DIN connector and install it in the bracket. Then, install the QT Py in the bracket. Finally, wire the QT Py.

For all wiring, follow the wiring table below, taking care about the orientation and peculiar numbering of the DIN connector.

The photos show six different colors of wire for clarity. You don't necessarily need to use six different colors but if you use fewer you'll need to take extra care to get the wiring correct.

The keyboard was originally designed to be powered from 5V, but during testing powering it with 3.3V worked fine, and it removes any concern about voltage level conversion.

<table>
<thead>
<tr>
<th>Color Code</th>
<th>DIN Position</th>
<th>QT Py Position</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>3</td>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>Red</td>
<td>5</td>
<td>3.3V</td>
<td>VCC (3.3v is OK)</td>
</tr>
<tr>
<td>Yellow</td>
<td>2</td>
<td>MOSI</td>
<td>Busy (active low)</td>
</tr>
<tr>
<td>Green</td>
<td>4</td>
<td>SCK</td>
<td>Clock Out</td>
</tr>
</tbody>
</table>
The following image from Wikipedia shows the DIN connector numbering seen from the front side on the socket (female). When you work on the solder side of the socket, the pins are mirrored or reversed so that if the keying notch is on the bottom, 1 will be on your right and 3 will be on your left.

In the following Fritzing diagram, the connector image has been pre-flipped, so pin #3 is at the left, #2 is at the top, and the keying notch is at the bottom. This is how you will view the connector while soldering it.

<table>
<thead>
<tr>
<th>Color</th>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>6</td>
<td>MISO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reset (active low)</td>
</tr>
<tr>
<td>White</td>
<td>1</td>
<td>RX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data Out</td>
</tr>
</tbody>
</table>

DIN 45326 / IEC/DIN EN 60130-9 types
IEC-20 and IEC-21: eight-pin, 45°, 270°
Solder the DIN connector

Take six lengths of wire, 100mm (4in) each is sufficient. It's easiest if you use six different colors as shown in the photos, but at the risk of getting your wires literally crossed you can use fewer colors. Strip one end of each wire and solder them to the DIN connector according to the wiring table. It may help if you twist the strands of solder together and then bend it part way back, to make a little hook that grabs the eyelet.

Install the connector & QT Py

Take the included nut off the DIN connector. Stick the DIN connector through the large circular hole in the print, then put the nut in from the back side. Orient the keying notch up and tighten the nut.

Take the QT Py. Point the USB connector "outwards" from the bracket, and press it firmly into the clip until it snaps firmly into place.

Solder the QT Py

Starting with RX and working outwards to GND, shorten each wire to a more appropriate length and then strip a very small amount (around 2mm).
Insert the wire into the top of the QT Py and while holding it in place use a small amount of solder to bond it firmly to the pad.

Repeat until all wires are soldered. Once you've double checked your wiring according to the table and images above, you can proceed to coding the Tandy 1000 Keyboard, stopping off at the Installing CircuitPython page if your QT Py doesn't have CircuitPython installed yet, or it's not up to date.

Install 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 link](https://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

---

**Coding the 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 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.

```python
# SPDX-FileCopyrightText: 2022 Jeff Epler for Adafruit Industries
# SPDX-License-Identifier: MIT
import time
import array
import board
import digitalio
import rp2pio
import usb_hid
import adafruit_pioasm
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode as K

KBD_NRESET = board.MISO
KBD_DATA = board.RX
KBD_CLOCK = board.SCK # Note that KBD_CLOCK must be 1 GPIO# above KBD_DATA
KBD_NBUSY = board.MOSI

tandy1000_keycodes = [
    None, K.ESCAPE, K.ONE, K.TWO, K.THREE, K.FOUR, K.FIVE, K.SIX, K.SEVEN,
    K.EIGHT, K.NINE, K.ZERO, K.MINUS, K.EQUALS, K.BACKSPACE, K.TAB, K.Q, K.W,
```
LOCK_KEYS = (K.CAPS_LOCK, K.KEYPAD_NUMLOCK)
LOCK_STATE = {
    K.CAPS_LOCK: False,
    K.KEYPAD_NUMLOCK: False,
}

KEYPAD_NUMLOCK_LOOKUP = [
    {
        K.KEYPAD_PLUS: K.INSERT,
        K.KEYPAD_MINUS: K.DELETE,
        K.KEYPAD_SEVEN: K.BACKSLASH,
        K.KEYPAD_EIGHT: (K.LEFT_SHIFT, K.GRAVE_ACCENT),
        K.KEYPAD_NINE: K.PAGE_UP,
        K.KEYPAD_FOUR: (K.LEFT_SHIFT, K.BACKSLASH),
        K.KEYPAD_FIVE: K.END,
        K.KEYPAD_SIX: K.ONE,
        K.KEYPAD_ZERO: K.ZERO,
        K.KEYPAD_PERIOD: K.PERIOD,
    },
    {
        K.KEYPAD_PLUS: (K.LEFT_SHIFT, K.EQUALS),
        K.KEYPAD_MINUS: K.MINUS,
        K.KEYPAD_SEVEN: K.SEVEN,
        K.KEYPAD_EIGHT: K.EIGHT,
        K.KEYPAD_NINE: K.NINE,
        K.KEYPAD_FOUR: K.FOUR,
        K.KEYPAD_FIVE: K.FIVE,
        K.KEYPAD_SIX: K.SIX,
        K.KEYPAD_ONE: K.ONE,
        K.KEYPAD_TW0: K.TWO,
        K.KEYPAD_THREE: K.THREE,
        K.KEYPAD_ZERO: K.ZERO,
        K.KEYPAD_PERIOD: K.PERIOD,
    }
]

# Assert busy
busy_out = digitalio.DigitalInOut(KBD_NBUSY)
busy_out.switch_to_output(False, digitalio.DriveMode.OPEN_DRAIN)

# Reset the keyboard
reset_out = digitalio.DigitalInOut(KBD_NRESET)
reset_out.switch_to_output(False, digitalio.DriveMode.OPEN_DRAIN)
time.sleep(.1)
reset_out.value = True

program = adafruit_pioasm.Program(""
    wait 1 pin 1
    in pins, 1
""
```python
wait 0 pin 1
```
How it works

Preliminaries

After the required `import` lines, there are lines to set the I/O pins used, and to map from Tandy keycodes to USB keycodes:

```python
import time
import array
import board
import digitalio
import rp2pio
import usb_hid
import adafruit_pioasm
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode as K

KBD_NRESET = board.MISO
KBD_DATA = board.RX
KBD_CLOCK = board.SCK # Note that KBD_CLOCK must be 1 GPIO# above KBD_DATA
KBD_NBUSY = board.MOSI

tandy1000_keycodes = [...]
```

Locking keys

The keyboard handles the caps lock and num lock key states internally. For instance, when caps lock is first pressed, the keyboard turns the light on and sends a "caps lock is pressed" message to CircuitPython. When caps lock is pressed again, the keyboard turns the light off and sends a "caps lock is released" message.

There's no way for CircuitPython to set the state of the caps lock or num lock LEDs. So, instead, the CircuitPython code tracks whether caps lock or num lock are active, and then modifies the events it sends back to the computer accordingly.

These lines configure which keys are locking keys, and the behavior of keypad keys depending on the numlock state:

```python
LOCK_KEYS = (K.CAPS_LOCK, K.KEYPAD_NUMLOCK)
LOCK_STATE = {
    K.CAPS_LOCK: False,
    K.KEYPAD_NUMLOCK: False,
}
KEYPAD_NUMLOCK_LOOKUP = [...]
```
Initialize the keyboard

To initialize the keyboard, first assert the "busy" and "reset" signals. Then, release the "reset" signal to allow the keyboard to initialize itself. The "busy" signal will remain asserted until CircuitPython is ready to receive key events.

These signals are "active low" and "open drain", which means that to assert the signal, the correct value to send to the pin is False.

```python
# Assert busy
busy_out = digitalio.DigitalInOut(KBD_NBUSY)
busy_out.switch_to_output(False, digitalio.DriveMode.OPEN_DRAIN)

# Reset the keyboard
reset_out = digitalio.DigitalInOut(KBD_NRESET)
reset_out.switch_to_output(False, digitalio.DriveMode.OPEN_DRAIN)
time.sleep(.1)
reset_out.value = True
```

Configure the PIO peripheral

These lines configure a program within the PIO peripheral. The program waits for the clock input to become true, then copies the data value, and waits for the clock input to become false.

The peripheral is configured to wait for 8 bits of data before making all 8 bits available to CircuitPython in a buffer.

```python
program = adafruit_pioasm.Program(""
    wait 1 pin 1
    in pins, 1
    wait 0 pin 1
"
)

sm = rp2pio.StateMachine(program.assembled,
    first_in_pin = KBD_DATA,
    in_pin_count = 2,
    pull_in_pin_up = 0b11,
    auto_push=True,
    push_threshold=8,
    in_shift_right=True,
    frequency=8_000_000,
    **program.pio_kwargs)
```

Main loop

Finally, the program is ready to receive key events from the keyboard and translate them to USB. First, read one value from the PIO peripheral and turn it into a boolean 'pressed' value and a Tandy key number value.
If the key number is not valid, something has gone wrong. Reset the keyboard and try again. Otherwise, event processing continues below.

```python
# Now ready to get keystrokes
kbd = Keyboard(usb_hid.devices)
baby_out.value = True
while True:
    sm.readinto(buf, swap=False)
    val = buf[0]
    pressed = (val & 0x80) == 0
    key_number = val & 0x7f

    if key_number > len(tandy1000_keycodes):
        # invalid keycode -- reset the keyboard
        reset_out.switch_to_output(False, digitalio.DriveMode.OPEN_DRAIN)
        time.sleep(.1)
        reset_out.value = True
        continue

    keycode = tandy1000_keycodes[key_number]
    if keycode is None:
        continue

    Next, translate the keycode based on the caps lock state. Then, if the event is a press, process it one of several ways:

    • If it's a lock key, just update the state of the lock key.
    • If caps lock is on, and the keycode is alphabetic, then send it shifted.
    • If the key code is a tuple, send it as multiple events. This is used, for instance, to send the backslash key when num lock is off and numpad "7" is pressed.
    • Otherwise send a simple USB key press event.

    Otherwise, if the event is a release, it's processed in one of the following ways:

    • If it's a lock key, just update the state of the lock key.
    • If it's a tuple, there's nothing to do.
    • Otherwise send a simple USB key release event.

... 
keycode = KEYPAD_NUMLOCK_LOOKUP[LOCK_STATE[K.KEYPAD_NUMLOCK]].get(keycode, keycode)
if pressed:
    if keycode in LOCK_KEYS:
        LOCK_STATE[keycode] = True
    elif LOCK_STATE[K.CAPS_LOCK] and K.A <= keycode <= K.Z:
        old_report_modifier = kbd.report_modifier[0]
        kbd.report_modifier[0] = (old_report_modifier & ~MASK_RIGHT_SHIFT) ^ MASK_LEFT_SHIFT
        kbd.press(keycode)
        kbd.release_all()
        kbd.report_modifier[0] = old_report_modifier
        continue
    elif isinstance(keycode, tuple):
        old_report_modifier = kbd.report_modifier[0]
        ...  
```

©Adafruit Industries
kbd.report_modifier[0] = 0
kbd.press(*keycode)
kbd.release_all()
kbd.report_modifier[0] = old_report_modifier
else:
    kbd.press(keycode)
else:
    if keycode in LOCK_KEYS:
        LOCK_STATE[keycode] = False
    elif isinstance(keycode, tuple):
        pass
    else:
        kbd.release(keycode)

---

Gallery

When I got the keyboard the exterior was in pretty rough shape after decades of storage.
While the keyboard was disassembled for cleaning, I took some photos of the interior. Happily, none of the keyswitches was the slightest bit damaged.

The keyboard runs off an 8048 microcontroller with built in storage, probably "mask programmed" at the factory with the final code.
With just soap and water on the plastic case & keyswitches, it cleaned up beautifully!