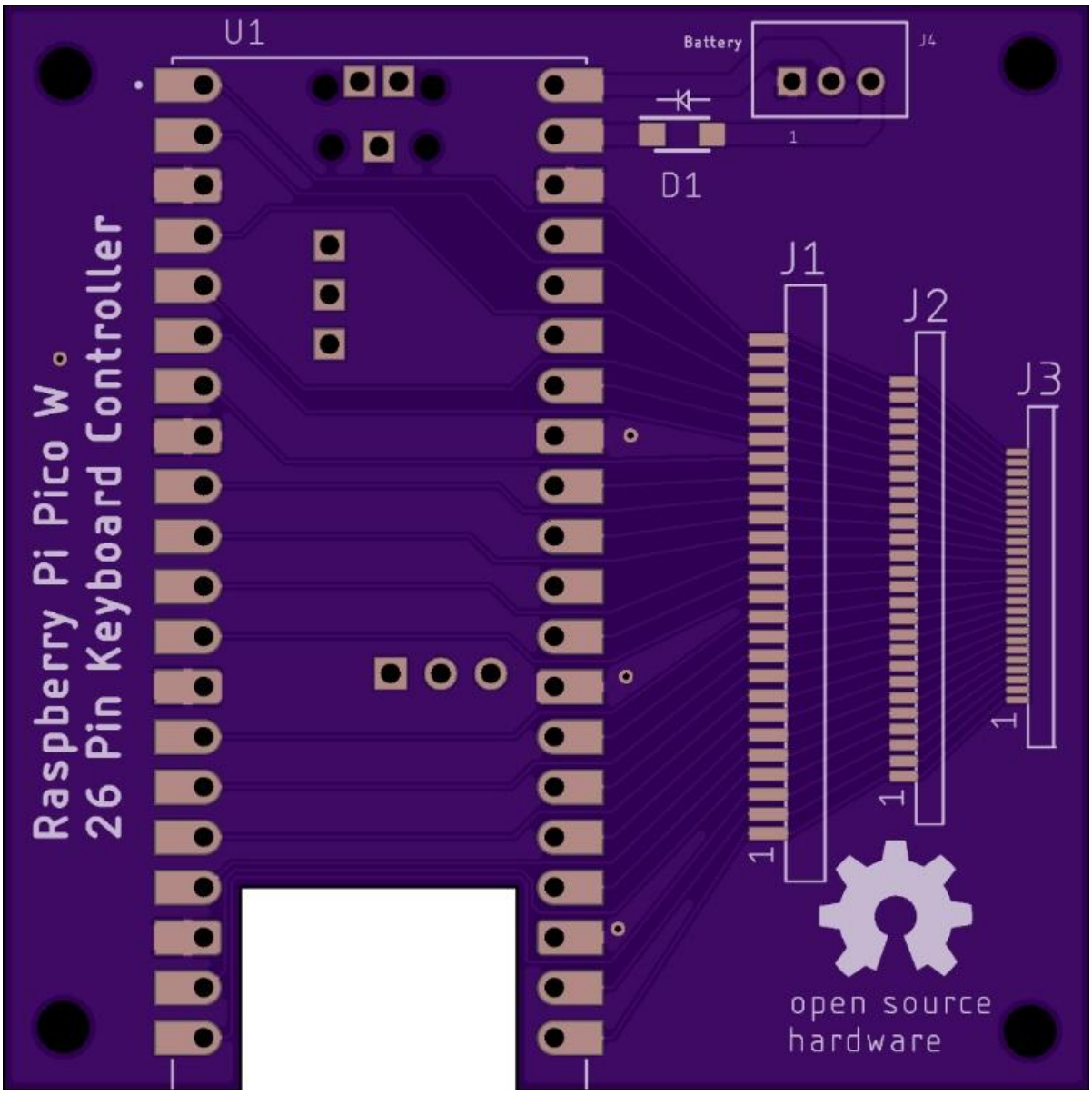
Raspberry Pi Pico W 26 Pin Laptop Keyboard Controller

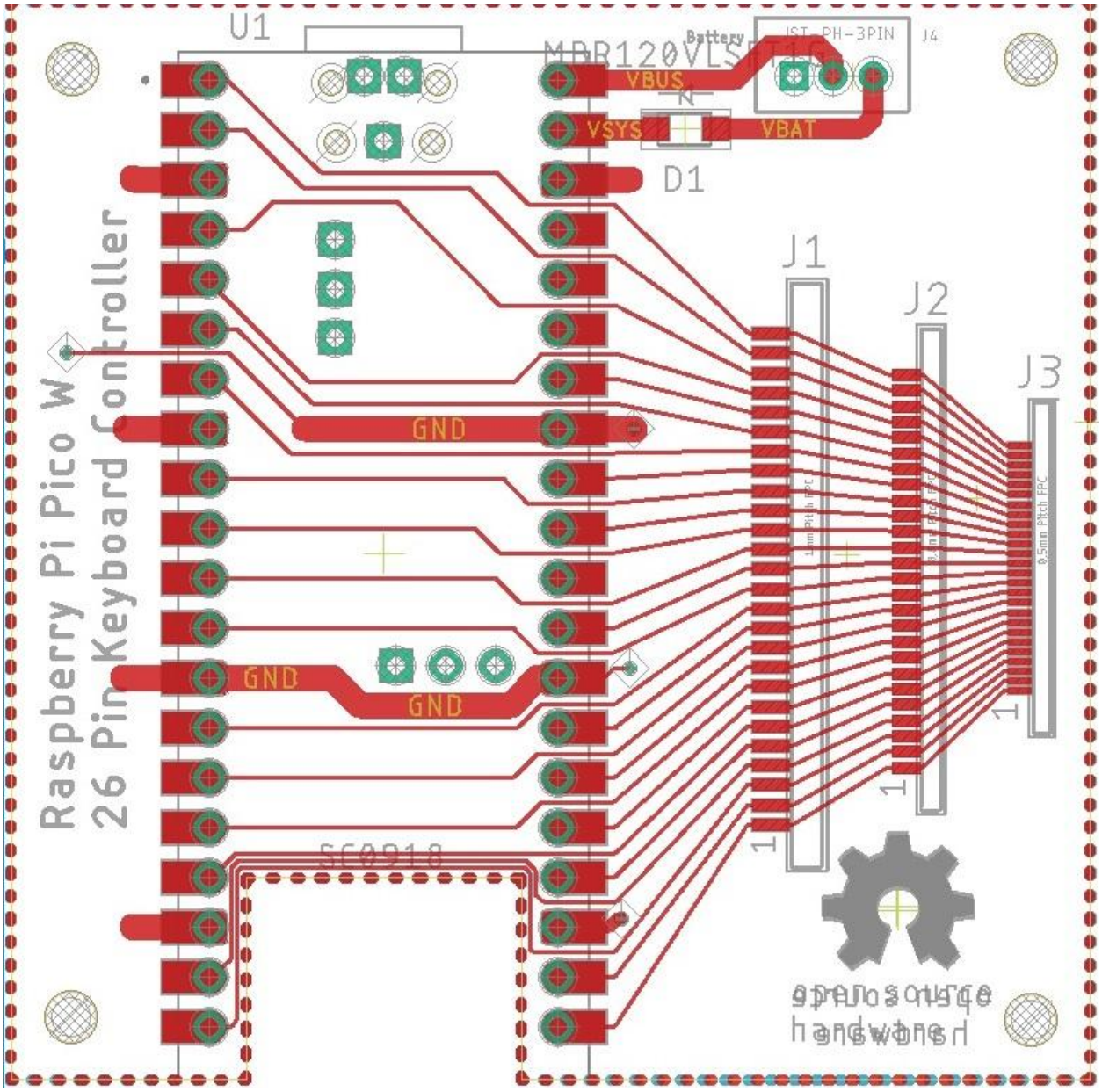
This document will describe how to make a USB and Bluetooth controller for a laptop keyboard with 26 or less FPC pins using a Raspberry Pi Pico W. If your keyboard has more than 26 pins, switch to using a RP2350B, documented at my repo.

The Pi Pico circuit board shown below has pads to solder either a 1.0mm, 0.8mm, or 0.5mm pitch FPC connector with up to 26 pins. If your connector has less than 26 pins, solder it starting at pin 1, leaving the pads at the other end empty. All associated files are at my Github repository.



The cutout in the board gives better reception for the Bluetooth antenna on the Pico. The Pico can be mounted with header pins or soldered directly to the board for a lower profile. If the keyboard will not be used for Bluetooth, it will always have USB power so the Schottky diode (D1) and 3 pin JST connector (J4) are not needed. These components are only used for battery operation with Bluetooth.

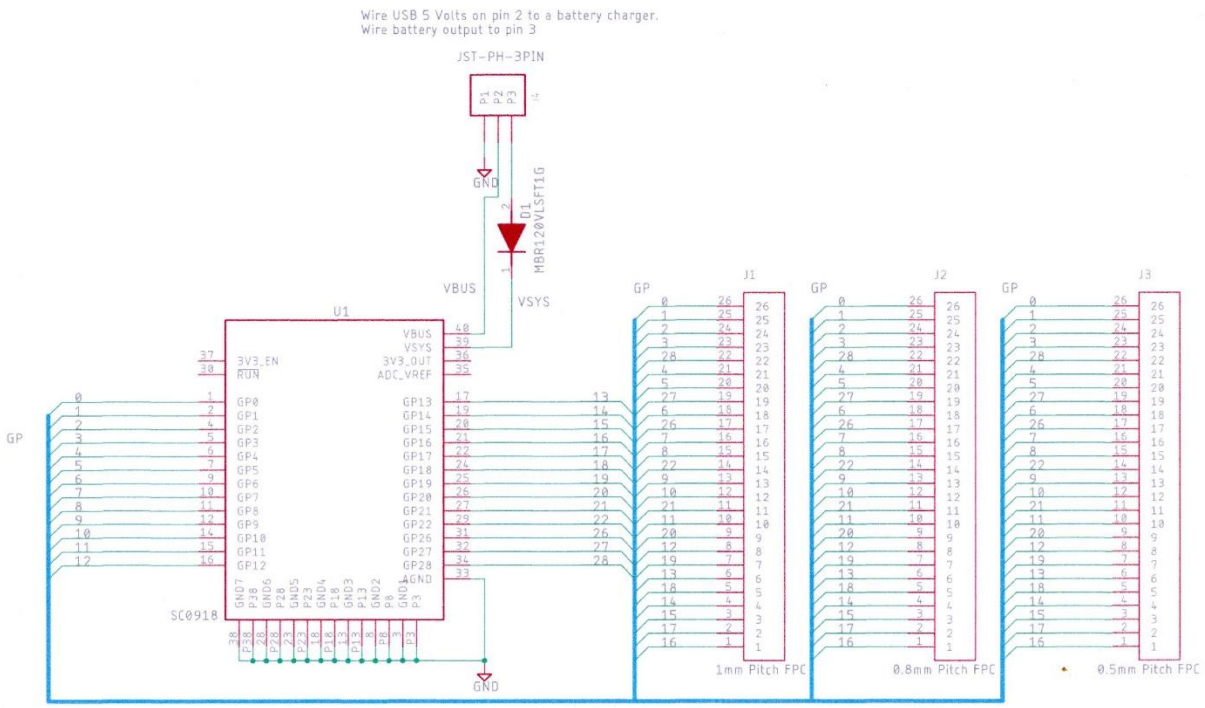
The “Pico_26Pin_Keyboard.brd” Eagle file and “Pico_26Pin_Keyboard.zip” Gerber file at my repo can be fabricated by OSH Park or other fab houses like JLCPCB. The Eagle layout (without area fill) is shown below so you can see the signal traces from the GPIO pins to the 3 FPC connector pads.



The circuit board measures 55mm x 55mm (2.17” x 2.17”).

If you plan to use Bluetooth, the controller will be running from a lithium battery. To charge the battery, plug in a USB cable to the Pico. USB 5 volts on the Pico VBUS pin is provided on J4 pin 2 (this is a 3 pin JST connector). This should be cabled along with ground to a battery charging circuit. The nominal 3.7 volt battery output should be cabled to the JST connector J4 pin 3. This feeds an MBR120VLSFT1G Schottky diode (D1) that “or-ties” the battery voltage to the Pico’s VSYS pin. There is another Schottky diode in the Pico that brings USB power to VSYS when the USB cable is attached.

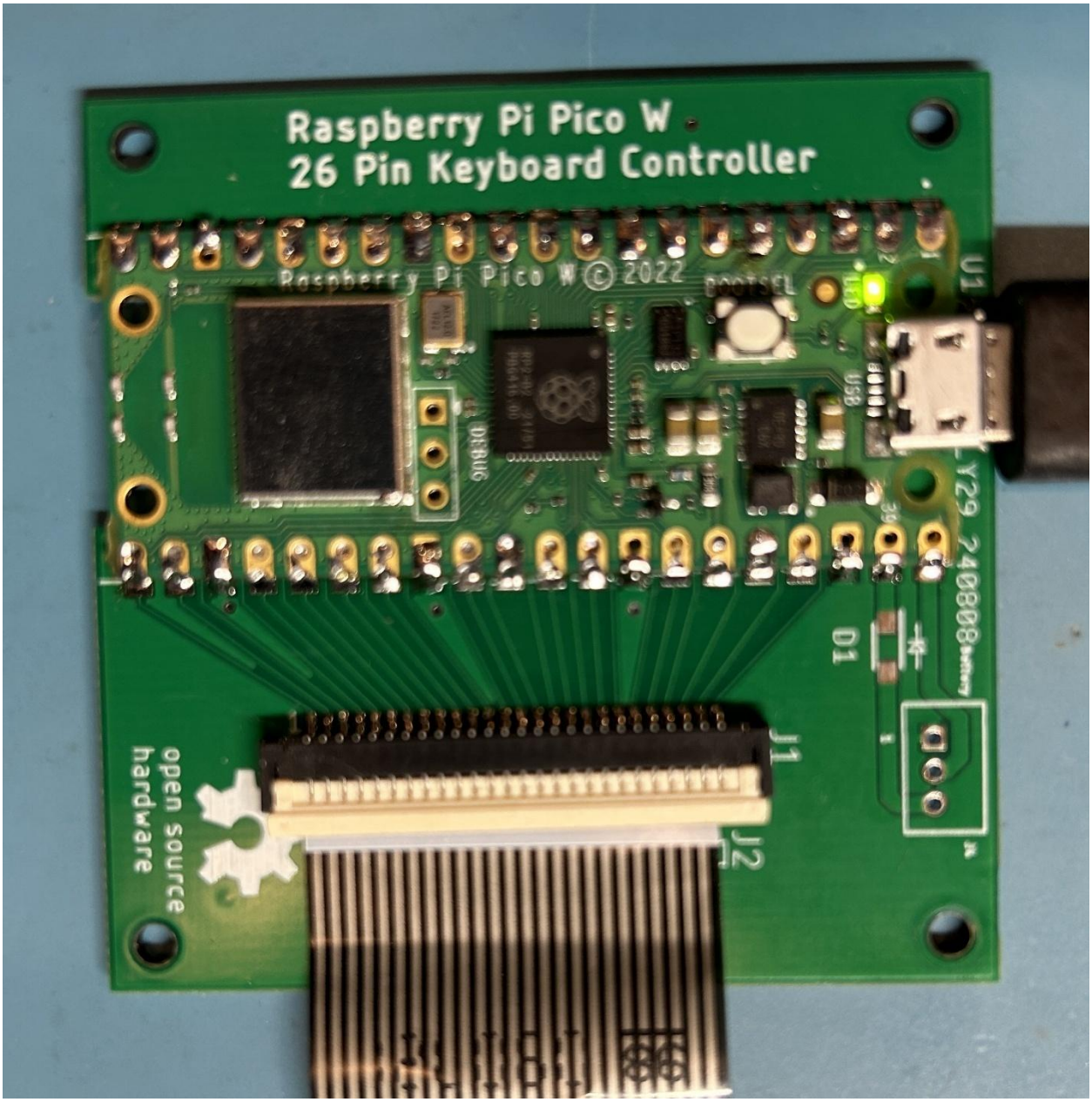
The Eagle schematic “Pico_26Pin_Keyboard.sch” at my repo is shown below.



The keyboard connections to the Pico GP I/O are shown below.

FPC Connector pin number	Pico GP I/O number
1	16
2	17
3	15
4	14
5	18
6	13
7	19
8	12
9	20
10	11
11	21
12	10
13	9
14	22
15	8
16	7
17	26
18	6
19	27
20	5
21	4
22	28
23	3
24	2
25	1
26	0

The assembled circuit board with a Raspberry Pi Pico W and a 26 pin 1mm pitch FPC connector is shown below. The FPC cable from a Dell Inspiron 1545 keyboard is attached. The procedure to make this example keyboard operational over USB will be described in the following pages.



I soldered this Pico directly to the connector board for a low profile but it could also be soldered with header pins. The diode at D1 and the JST connector at J4 are not yet installed for Bluetooth battery operation.

A good starting point for learning how to write Python code for the Pico is at circuitpython.org/. Select the “Get Started” box. The following is a synopsis of the steps that they provide.

To enter into boot loader mode, hold the Pico push button down while plugging in the Pico’s USB cable. The new folder will show as a drive named RPI-RPI2.

If the Pico is acting weird or you want to start with a clean slate, copy the flash_nuke.uf2 file (downloaded here) over to the Pico’s RPI-PPI2 folder. The drive will blink out and then return all clean.

Python - Start with the circuitpython.org/downloads page, then select your Pico model. Download and copy the latest Adafruit Circuit Python uf2 over to the Pico’s RPI-PPI2 folder. The drive will blink out and return as CIRCUITPY with an empty lib folder. The code.py file is a one-line program that prints “Hello world”. You will be replacing this file later.

Keyboard library – Go to circuitpython.org/libraries and download the library bundle that matches the version of Python you are using. The bundle includes so many libraries that they can’t all fit in the Pico’s memory. Only copy the Adafruit_HID folder into the lib folder on the Pico.

As you write and debug your Python code you will need an editor. Adafruit likes the Mu editor and they give links for installation in their tutorial. I prefer Thonny and it can be downloaded at thonny.org/. My descriptions below are based on Thonny.

A fresh Thonny install defaults to running the Python code locally on your PC, not on the Pi Pico. It will error because it can't find the "board" library that's embedded in Circuitpython. Go to the bottom right corner of Thonny and click Local Python 3 and then "configure interpreter". In the first drop down, select CircuitPython (generic). Under Port, leave it at <Try to detect port automatically>. Leave everything else checked and then select OK. Now you can select stop and then run in Thonny and it runs the code in the Pico.

Download my Matrix_Decoder.py code, then start Thonny. Select File, Open, This Computer and navigate to the Matrix_Decoder.py code. This will bring it up in the editor and you can read through the comments to get an idea of how it works. On Thonny, select the “Stop” icon to halt any program that is running on the Pico. Next select File, Save As, CircuitPython Device. Overwrite the existing code.py file with your Thonny code. Select the “Run” icon so the Pico will run the code.py program. As you debug your code, you will need to hit the Stop icon in order to save your changes to the Pico. Be sure to also save your updated code to your computer.

Bring up another editor like Notepad++ and load a blank key_list text file that lists all the possible keyboard keys. Place the cursor to the far right of the first key in the list. This is where the Python program will send the Pico GPIO numbers when you push a key. Then the program will send a down arrow to position the cursor for the next key. The pin connections for an example Dell Inspiron 1545 keyboard (model NSK-D9301) are given in Appendix A.

Dell 1545 Example Keyboard Connected to the Pico

The manual procedure to determine which GPIO's are rows and which are columns, uses the Modifier keys; Control, Alt, Shift, GUI, and Fn. Usually there will be 8 GPIO's that will be columns and programmed as Pico inputs with pull ups. The remaining GPIO's will be rows and driven low one at a time by the Pico. A semi-automated program to determine the rows, columns, and key matrix is also available and described here.

In the Dell example, Control-Left and Control-Right have GPIO 8 in common which will be a row. GPIO's 2 and 27 will be columns. Shift-Left and Shift-Right have GPIO 22 in common (row) and GPIO's 2 and 5 are columns (we already knew 2 was a column). Alt-Left and Alt-Right have GPIO 9 in common (row) and GPIO's 1 and 6 are columns. So far, we know 1, 2, 5, 6, and 27 are columns. GPIO 4 on the GUI key would make sense as a column based on looking down the list. Likewise, GPIO 3 for the Fn key makes sense as a column when looking down the list. That gives 7 columns and to find the 8th, look down the list to find a key that doesn't use 1, 2, 3, 4, 5, 6, or 27. The E key shows 28 is the last column. All other GPIO's will be rows. Build a key matrix like the one shown below with the column GPIO's listed across the top and

all the remaining GPIO's listed as rows down the side. Now place each key name at the appropriate row/column intersection. The KMK keycode names are given here.

Dell 1545 Keyboard Matrix

	GP1	GP2	GP3	GP4	GP5	GP6	GP27	GP28
GP7			PGDOWN	LGUI				PGUP
GP8		RCTRL					LCTRL	
GP9	RALT			PSCREEN		LALT		EJCT
GP10		Z	A	N1	TAB	ESC	GRAVE	Q
GP11		C	D	N3	F3	F4	F2	E
GP12	SPACE	ENTER	BSLASH	F10	BSPACE	F5	F9	
GP13		COMMA	K	N8	RBRC	F6	EQUAL	I
GP14		DOT	L	N9	F7		F8	O
GP15	LEFT			END		UP	HOME	
GP16	RIGHT			F12			INSERT	
GP17	DOWN			F11			DELETE	
GP18	SLASH		SCOLON	N0	LBRC	QUOTE	MINUS	P
GP19	N	M	J	N7	Y	H	N6	U
GP20	B	V	F	N4	T	G	N5	R
GP21		X	S	N2	CAPS		F1	W
GP22		RSHIFT			LSHIFT			
GP26			FN					

The table below is the same as above except the media keys replace certain function keys. These are sent when Fn is held down. The table of KMK Media keys can be found here.

	GP1	GP2	GP3	GP4	GP5	GP6	GP27	GP28
GP7			PGDOWN	LGUI				PGUP
GP8		RCTRL					LCTRL	
GP9	RALT			PSCREEN		LALT		EJCT
GP10		Z	A	N1	TAB	ESC	GRAVE	Q
GP11		C	D	N3	F3	BRID	F2	E
GP12	SPACE	ENTER	BSLASH	MPRV	BSPACE	BRIU	VOLU	
GP13		COMMA	K	N8	RBRC	F6	EQUAL	I
GP14		DOT	L	N9	MUTE		VOLD	O
GP15	LEFT			END		UP	HOME	
GP16	RIGHT			MNXT			INSERT	
GP17	DOWN			MPLY			DELETE	
GP18	SLASH		SCOLON	N0	LBRC	QUOTE	MINUS	P
GP19	N	M	J	N7	Y	H	N6	U
GP20	B	V	F	N4	T	G	N5	R
GP21		X	S	N2	CAPS		F1	W
GP22		RSHIFT			LSHIFT			
GP26			FN					

KMK is open-source firmware normally used for mechanical keyboards but it also works for laptop keyboards, as long as you know how the switches are wired (see above tables). KMK uses CircuitPython which is why I also wrote my matrix decoder code in CircuitPython. All of the hard work of scanning key switches and communicating over USB is embedded in KMK. All you need to do is edit my KMK Dell example code with your key matrix information. Getting started with KMK is described at their GitHub repo which I will detail below.

For step 1, you already have CircuitPython installed.

For step 2, click on “get an up-to-date copy of KMK”.

For step 3, unzip the kmk_firmware-main folder. The top folder is kmk_firmware-main. It has many folders and files. Copy the KMK folder and boot.py to the CIRCUITPY drive on the Pico.

For step 4, use my code_Dell1545.py as a starting point. Load it into Thonny and edit it as follows:

Change keyboard.col_pins to list the GPIO column pins left to right across the top of your matrix.

Change keyboard.row_pins to list the GPIO row pins top to bottom along the side of your matrix.

Change the keyboard.keymap matrix for the base layer and the Fn media layer per the tables you created (see above). Each keycode name (except FN) must be preceded with KC. Put KC.NO if there is no key at a matrix location. After saving the code to your computer, save it to the Pico’s CIRCUITPY drive with the name code.py (this will overwrite your previous code.py that was the matrix decoder).

With the code.py still In Thonny, click the “Stop” icon, then click the “Run” icon. You should now be able to bring up a blank editor like Notepad++ and type on the keyboard to test if all the keys work. If you have any typo’s, they will be reported in the Thonny console window. Use the keyboardchecker.com website to verify all the keys work.

For normal USB keyboard operation (without Thonny), you can just plug in the USB cable. This will bring up a CIRCUITPY drive folder which you can close.

Appendix A – Pin Connections for a Dell 1545 example keyboard

KMK Keycode Name	GPIO#	GPIO#
KC.LCTRL	27	8
KC.RCTRL	8	2
KC.LSHIFT	22	5
KC.RSHIFT	22	2
KC.LALT	9	6
KC.RALT	9	1
KC.LGUI	7	4
KC.RGUI		
FN	26	3
KC.A	10	3
KC.B	20	1
KC.C	11	2
KC.D	11	3
KC.E	28	11
KC.F	20	3
KC.G	20	6
KC.H	19	6
KC.I	28	13
KC.J	19	3
KC.K	13	3
KC.L	14	3
KC.M	19	2
KC.N	19	1
KC.O	28	14
KC.P	28	18
KC.Q	28	10
KC.R	28	20
KC.S	21	3
KC.T	20	5
KC.U	28	19
KC.V	20	2
KC.W	28	21

KC.X	21	2
KC.Y	19	5
KC.Z	10	2
KC.GRAVE	27	10
KC.N1	10	4
KC.N2	21	4
KC.N3	11	4
KC.N4	20	4
KC.N5	27	20
KC.N6	27	19
KC.N7	19	4
KC.N8	13	4
KC.N9	14	4
KC.N0	18	4
KC.MINUS	27	18
KC.EQUAL	27	13
KC.BSPACE	12	5
KC.ESC	10	6
KC.F1	27	21
KC.F2	27	11
KC.F3	11	5
KC.F4	11	6
KC.F5	12	6
KC.F6	13	6
KC.F7	14	5
KC.F8	27	14
KC.F9	27	12
KC.F10	12	4
KC.F11	17	4
KC.F12	16	4
KC.INSERT	27	16
KC.DELETE	27	17
KC.RIGHT	16	1
KC.LEFT	15	1
KC.UP	15	6

KC.DOWN	17	1
KC.SLASH	18	1
KC.DOT	14	2
KC.COMMA	13	2
KC.SCOLON	18	3
KC.QUOTE	18	6
KC.ENTER	12	2
KC.LBRC	18	5
KC.RBRC	13	5
KC.BSLASH	12	3
KC.CAPS	21	5
KC.TAB	10	5
KC.SPACE	12	1
KC.HOME	27	15
KC.END	15	4
KC.PGUP	28	7
KC.PGDOWN	7	3
KC.PSCREEN	9	4
KC.SCROLLLOCK		
KC.NUM_LOCK		
KC.PAUSE		
KC.PSLS		
KC.PAST		
KC.PMNS		
KC.PPLS		
KC.PENT		
KC.PDOT		
KC.P0		
KC.P1		
KC.P2		
KC.P3		
KC.P4		
KC.P5		
KC.P6		
KC.P7		

KC.P8

KC.P9

KC.AUDIO_MUTE	Func		14	5
KC.AUDIO_VOL_UP	Func		27	12
KC.AUDIO_VOL_DOWN	Func		27	14
KC.BRIGHTNESS_UP	Func		12	6
KC.BRIGHTNESS_DOWN	Func		11	6
KC.MEDIA_NEXT_TRACK	Func		16	4
KC.MEDIA_PREV_TRACK	Func		12	4
KC.MEDIA_STOP	Func			
KC.MEDIA_PLAY_PAUSE	Func		17	4
KC.MEDIA_EJECT	Func		28	9
KC.MEDIA_FAST_FORWARD	Func			
KC.MEDIA_REWIND	Func			

Appendix B – Bluetooth

I have not got Bluetooth working yet but here are my notes:

KMK BLE HID on GitHub

This requires the adafruit_ble library from Adafruit to be copied to the lib folder on the Pico. It is part of the Adafruit CircuitPython Bundle found here. Same can be found from the Adafruit library already downloaded.

To enable BLE hid, change the keyboard.go(). By default, the advertised name will be the name of the "flash drive". By default, this is CIRCUITPY.

```
from kmk.hid import HIDModes

if __name__ == '__main__':

    keyboard.go(hid_type=HIDModes.BLE)
```

There are two ways to change the advertising name. The first would be to change the name of the drive. The second would be to change the keyboard.go() like this.

```
if __name__ == '__main__':

    keyboard.go(hid_type=HIDModes.BLE, ble_name='KMKeyboard')
```

---

1. Enable BLE in KMK:
    - In your code.py (or similar KMK configuration file), you'll need to import HIDModes from kmk.hid.
    - When calling keyboard.go(), specify secondary_hid_type=HIDModes.BLE to enable BLE functionality alongside your primary HID type (e.g., USB).

Python

```
from kmk.hid import HIDModes
# ... other imports

keyboard.go(hid_type=HIDModes.USB, secondary_hid_type=HIDModes.BLE)
```

1. Switching between USB and BLE:
    - You can assign a key in your keymap to switch between the primary and secondary HID modes (e.g., USB and BLE) using KC.HID.

Python

```
from kmk.keys import KC
# ... other setup
```

```
keyboard.keymap = [
    [KC.A, KC.B, KC.HID_TO_BLE, KC.C], # Example: KC.HID_TO_BLE to switch to BLE
    # ... other layers
]
```