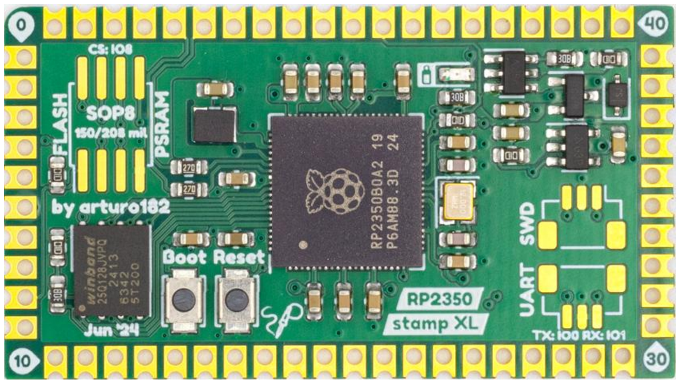
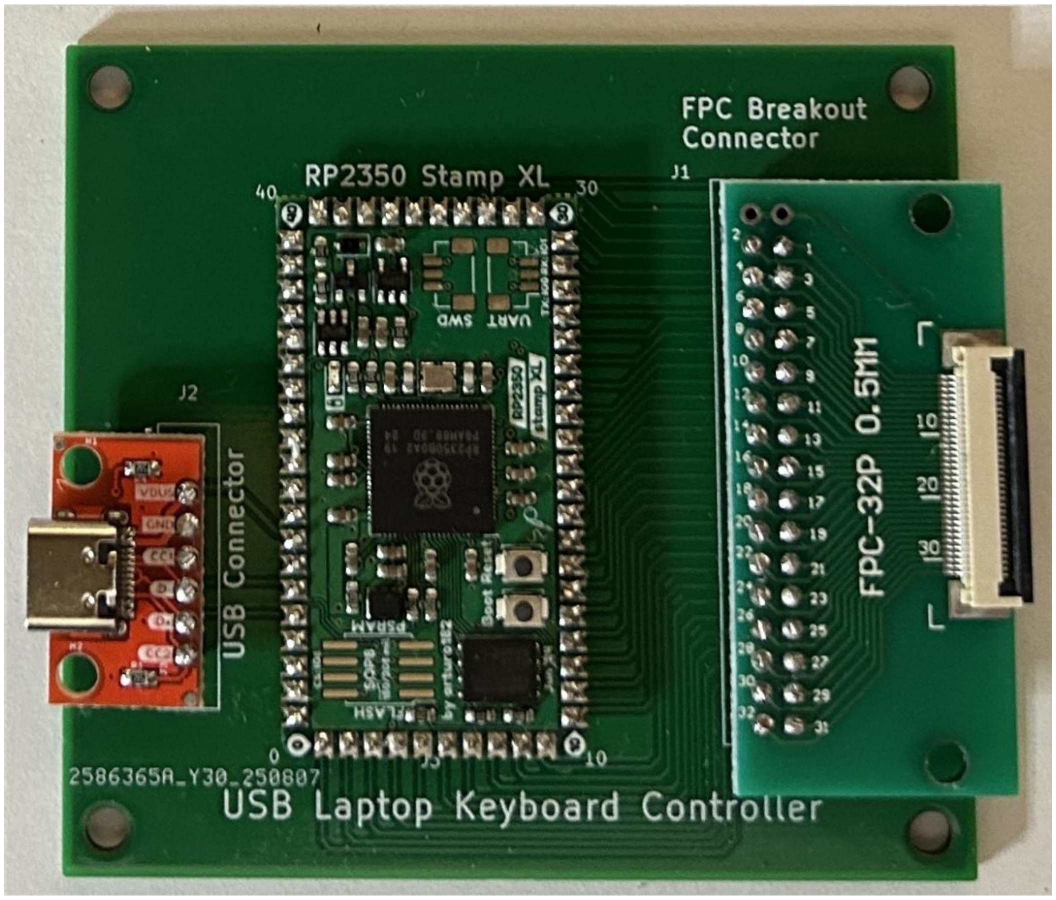
RP2350 Stamp XL 34 Pin Laptop Keyboard Controller

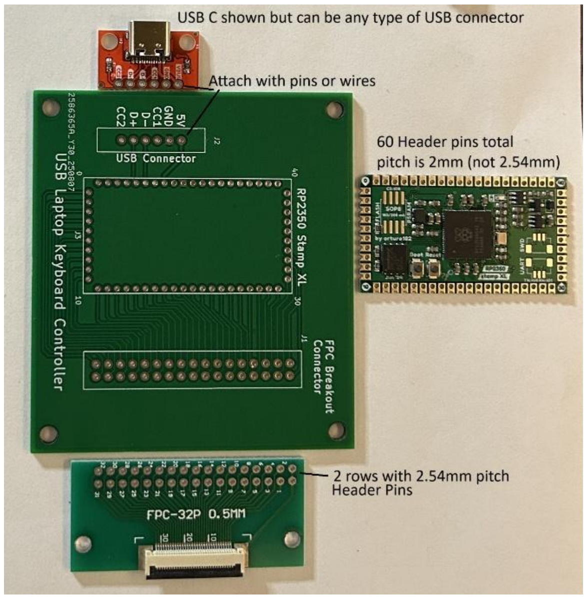
This document will describe how to make a USB controller for a laptop keyboard with up to 34 FPC pins using the RP2350 Stamp XL from Solder Party shown below.



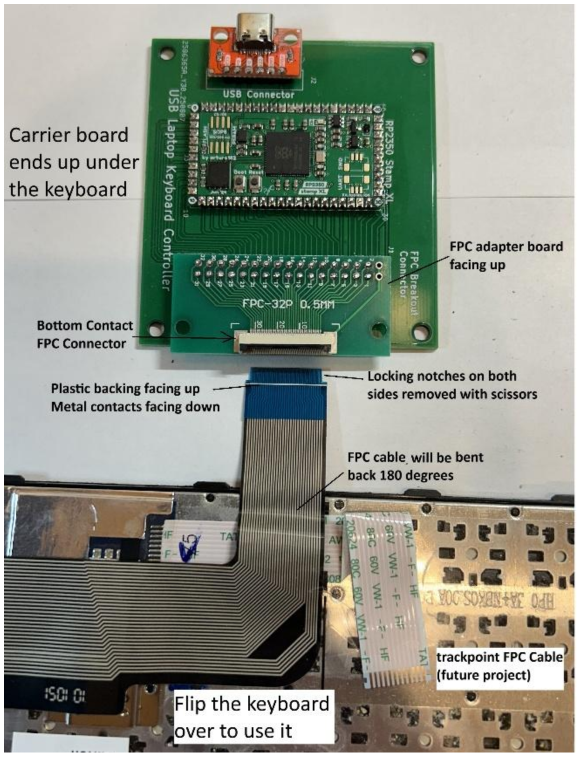
The Stamp XL uses a Raspberry Pi 2350B controller chip with 48 GPIO's, compared with the 26 GPIO's available on a Raspberry Pi Pico. I designed the circuit board shown below to connect the RP2350 Stamp XL with an FPC adapter board and USB breakout board. The surface mount soldering has already been done, leaving you to solder the through-hole pads with header pins or wires. All associated files are at my GitHub repository.

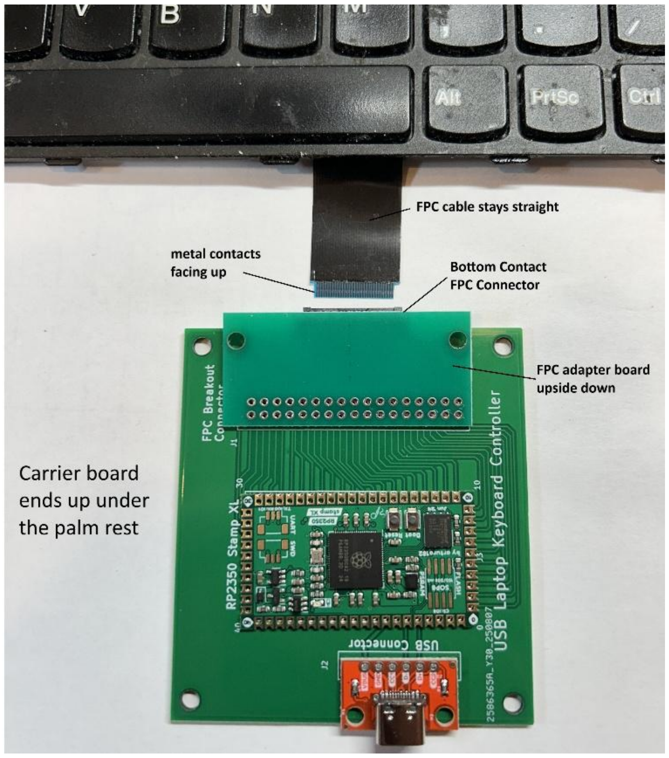


The carrier board is designed in KiCad (instead of Eagle) and measures 2.82 x 2.68 inches (71.7 x 68.0mm). The KiCad board file "2350B_Keybd_Cntrl.kicad_pcb" can be sent to fab houses like OSHPark.com ($38/3 boards) and Eurocircuits.com (€47/1 board). The zipped Gerber file "2350B_Keybd_Cntrl.zip" can be sent to fab houses like JLCPCB.com. With slow economy shipping to the US, the cost is $10 for 10 boards. The above files are available for download from my repo. The 3 components that attach to my carrier board are shown below prior to soldering.


USB C shown but can be any type of USB connector
Attach with pins or wires
60 Header pins total
pitch is 2mm (not 2.54mm)
2 rows with 2.54mm pitch
Header Pins
USB Connector
USB Laptop Keyboard Controller
RP2350 Stamp XL
FPC Breakout Connector
FPC-32P 0.5MM

The FPC breakout boards are very common and can be found at Amazon, EBay, and AliExpress among other sellers. Pick the FPC breakout board pitch and pin count that matches your keyboard FPC cable. These boards will have 2 rows of 2.54 mm pitch through-hole pads. The 32 pin FPC breakout board shown above was needed for a Lenovo E550 keyboard. It's a little different than most breakout boards and has 2 extra pads that are tied together and go to the mounting tab of the FPC connector as a ground/shield. I did not install pins in these two locations because they're not needed and would tie two GPIO pins together. The E550 FPC cable had locking notches that had to be trimmed off with scissors in order to use the FPC adapter board. Avoid the boards that come with the header pins already soldered because this limits how they can be mounted. The typical breakout board uses FPC connectors with bottom contacts. Determine if your FPC cable will end up with its contacts facing up or down when routed to the carrier board. It's common for the FPC cable to be flipped back 180 degrees when the FPC connector is underneath the keyboard. The pictures below show when to solder the FPC breakout board facing up or facing down so it can handle bottom contacts or top contacts on the cable. The left picture shows the cable will be flipped over so the carrier board ends up mounted underneath the keyboard. The right picture shows the FPC cable is not flipped and the carrier board will be mounted under the palm rest.


Carrier board ends up under the keyboard
FPC adapter board facing up
Bottom Contact FPC Connector
Plastic backing facing up Metal contacts facing down
Locking notches on both sides removed with scissors
FPC cable will be bent back 180 degrees
trackpoint FPC Cable (future project)
Flip the keyboard over to use it


FPC cable stays straight
metal contacts facing up
Bottom Contact FPC Connector
FPC adapter board upside down
Carrier board ends up under the palm rest

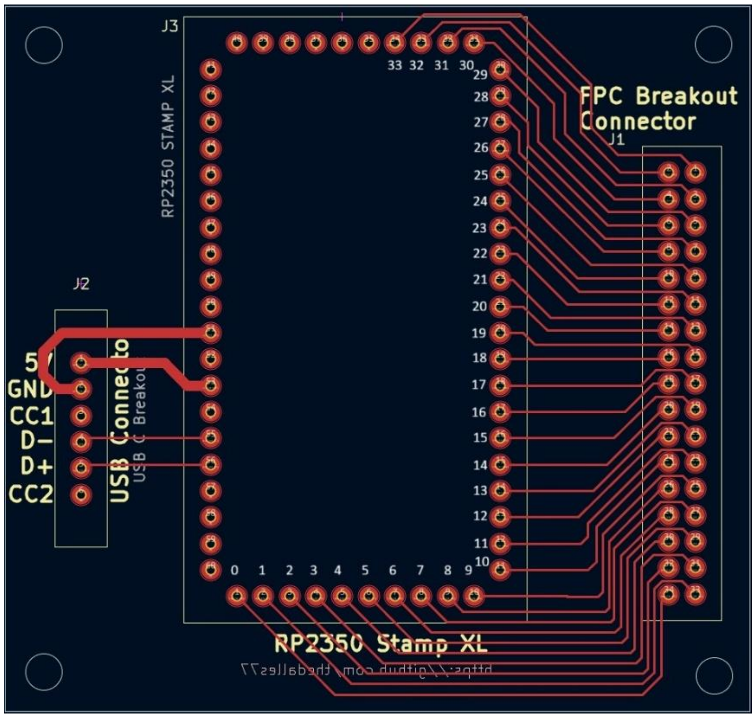
The Type C USB connector board from Amazon, Sparkfun, and AliExpress can be installed directly on the carrier board with 6 header pins or with 4 wires that allow the USB connector to be at the back or side of your keyboard enclosure. Only 4 wires (VBUS, GND, D+, D-) are needed because CC1 and CC2 are connected to 5.1K resistors on the USB board and are not used by the Stamp XL. You can wire to a USB Type A, Mini-B, or microB breakout connector if you don't want to use USB C.

Solder Party designed the RP2350 Stamp XL with the Raspberry Pi 2350B chip which has 48 GPIO's. I could have designed the carrier board for a very large FPC pin count but I seldom see more than 34 pins on an FPC keyboard cable. I wanted to keep the carrier board size as small as possible to keep the cost down.

I'm new to KiCad and couldn't figure out how to make pads with plated holes that are large enough to allow the RP2350 Stamp XL to be soldered directly to the board so you'll have to use header pins.

Very Important Note: Solder Party designed the Stamp XL with 2 mm pitch header pins (not the standard 2.54 mm pitch). Be sure to buy the correct pitch pins.

This is how I routed the signals in KiCad. GPIO’s 0 thru 33 are connected to the 2 rows of pads for the FPC breakout board. If your FPC connector has less than 34 pins, solder it any way you want. The software will figure out which GPIO signals are connected to key switches and which are no connects. Unlike my Teensy decoder code, this code only deals with GPIO numbers, never pin numbers. This eliminates the problem of defining which end of the connector is pin 1.



I've chosen to use Circuit Python instead of Arduino to program the Stamp XL board. A good starting point for learning how to write Python code is at circuitpython.org. Select the “Get Started” box. The following is a synopsis of the steps that they provide.

To enter into loader mode, hold the boot push button down while plugging in the USB cable. A new folder will show as a USB drive, named RP2350.

If things are acting weird or you want to start with a clean slate, copy the flash_nuke.uf2 file (downloaded here) over to the USB drive folder. The drive will blink out and then return all clean. I did this because the Stamp XL arrived with an old version of Python installed.

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

Keyboard library – Go to circuitpython.org/libraries and download/unzip the library bundle that matches the version of Python you are using. The bundle includes lots of libraries but you only need to copy the Adafruit_HID folder into the lib folder on the CIRCUITPY drive.

As you write and debug your Python code, you will need an editor. I prefer Thonny and it can be downloaded at thonny.org/. 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 RP2350.

Download my Matrix_Decoder_RP2350B.py code. In Thonny, select File, Open, This Computer and navigate to the Matrix Decoder code you downloaded. This will bring it up in the editor and you can read through the comments to get an idea of how it works. In Thonny, select the "Stop" icon to halt the "Hello World" program. Next select File, Save As, CircuitPython Device. Overwrite the "Hello World" code.py file with your Thonny code (renamed to code.py). With the keyboard not connected, hit the "Stop" icon and then "Run" icon to see if any key presses are given in the Thonny console window. If they are, you have a solder short that must be fixed. Disconnect the USB cable each time you need to power down and debug problems. When ready, re-attach the USB cable to power back up. The code will start running as soon as power is applied but it's best to hit the "Stop" icon and then the "Run" icon so you can watch the Thonny console window. Once your shorts are cleared, you can prove the program will send numbers to an editor by shorting two GPIO's together with a wire. Now that the program and board are ready, disconnect the USB cable and hook up the keyboard FPC cable, then connect the USB cable. When you hit the "Stop" icon and then the "Run" icon, if the program sends two numbers to an editor before any keys are pressed, these are probably grounds or LEDs in the keyboard that must be skipped by removing them from the I_O array starting on line 33 in the code. Once you have no numbers reported when you hit "Run", you're ready to make a key matrix.

Bring up an editor like Notepad++ and load the blank keyboard pin list text file that has all the possible keyboard keys. Place the cursor far to the right of the first key in the list. This is where the Python program will send the GPIO numbers when you push a key. After each key press and release, the program will send a down arrow to position the cursor for the next key. Once you have GPIO numbers for all keys, you can use the manual procedure described in step 12 of my Instructable to determine the row and column GPIO's. Next you will build a matrix table, as described in step 13 with the key names at the row/column intersection. The translation described in step 14 is not needed since FPC pin numbers are not used, only GPIO numbers. Instead of the manual procedure, a semi-automated program can be used that will determine the rows and columns. It will also build the key matrix with some editing as described in this PDF and in Appendix B below.

The Lenovo E550 key list is given as Appendix A at the bottom of this document. It was created using the Matrix_Decoder_RP2350B.py routine. The manual procedure described in Step 12 of the Instructable starts with the modifier keys to find many of the column GPIO's. The E550 control keys show 2 and 8 are columns. The Shift keys show 0 and 2 are columns (we already knew 2). The Alt keys show 1 and 11 are columns and GUI uses 0 which we already knew is a column. Lenovo keyboards typically give the Fn key its own two signals so 28 and 29 will be treated separately. So far, the columns we know are 0, 1, 2, 8, 11. We know from the modifier keys that 17, 3, 18, and 19 are rows so scan down the list and find when these are used and if they reveal a new column. 19 is used with 5 for the F12 key so 5 is a column. Next start with "A", looking for numbers not yet known to be rows or columns. "A" uses 6 and 4 so look at other keys that use either 6 or 4. "Z" uses 6 and 2 but we already knew 2 is a column so 6 must be a row. Going back to "A", if 6 is a row, 4 is a column. Now the known columns are 0, 1, 2, 4, 5, 8, 11. Typically there are 8 columns so there's 1 more to find. Going down the list, "E" uses 10 and 7 but "C" tells us that 10 is a row so 7 must be the last column. To confirm all columns are identified, I put all the keys into a matrix. I picked 29 as a column for Fn but 28 would also have worked. The Lenovo E550 Key matrix is given below.

GPIO	0	1	2	4	5	7	8	11	29
3	LSHIFT		RSHIFT						
6	TAB		Z	A	1	Q	GRAVE	ESC	
9	Y	N	M	J	7	U	6	H	
10	F3		C	D	3	E	F2	F4	
12	CAPS		X	S	2	W	F1		
13	T	B	V	F	4	R	5	G	
14	F7		PERIOD	L	9	O	F8		
15	LBRC	SLASH		SCOLON	0	P	MINUS	QUOTE	
16	RBRC		COMMA	K	8	I	EQUAL	F6	
17			RCTRL				LCTRL		
18		RALT						LALT	
19	LGUI	RIGHT			F12				
20		LEFT	KPDOT		END			UP	
21		DOWN			F11		HOME		
22	BKSPC	SPACE	ENTER	BSLASH	F10		F9	F5	
23		PGDOWN	PGUP	PSCREEN	INS		DELETE		
28									FN
30	KP/	KP+	KP9	KP7	KP8	KP*	KP-	NUMLK	
31	KP5	KP0	KP_ENT	KP2	KP3	KP6	KP1	KP4	

To turn the matrix table into a USB keyboard routine, I used KMK (instead of Arduino). It's open-source firmware normally used for mechanical keyboards but it also works for laptop keyboards (once you have the matrix). KMK uses CircuitPython which is why I also wrote the 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 example keyboard code with your key matrix information. Getting started with KMK is described at their KMK 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. It has many folders and files. Copy the KMK folder and boot.py to the CIRCUITPY drive on the Stamp.

For step 4, use code_Dell1545.py as a starting point for keyboards without a number pad. If your keyboard has a number pad, use code_LenovoE550.py so you can see how the number pad key names are done. Load the code into Thonny and edit as described below:

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 per the table you created. The basic keycode names that KMK uses are given here plus you can see the names I used in the E550 code example. Keypad keys must start with KC.P followed by the name or number from the keycode basic list. Each keycode name (except FN) must be preceded with KC. Put KC.NO in each matrix location that has no key at that position.

Copy the layer 0: Base Layer over to the layer 1: Fn Media Layer. Now look at your keyboard's media keys to see which (if any) you want to make work. The media keycode names are given here. For the Lenovo E550, I wanted the Mute, Volume Down, Volume Up, Brightness Down, and Brightness Up keys to work when Fn is pressed. These media keycode names replaced the corresponding Function key names at F1, F2, F3, F5, and F6 in the matrix.

Click the "Stop" icon in Thonny just to make sure it's ready for new code. Save your code to your computer and then save it to the Stamp's CIRCUITPY drive with the name "code.py" (this will overwrite your previous code.py that was the matrix decoder).

Now click the “Run” icon in Thonny and see if you have any typos reported in the console window. Once all typos are cleared, you should be able to type on the keyboard to test if 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. The Stamp will start running the code as soon as USB power is applied.

Appendix A – Key Connections for a Lenovo E550 Keyboard

KC.LCTRL	17	8
KC.RCTRL	17	2
KC.LSHIFT	3	0
KC.RSHIFT	3	2
KC.LALT	18	11
KC.RALT	18	1
KC.LGUI	19	0
KC.RGUI		
FN	29	28
KC.A	6	4
KC.B	13	1
KC.C	10	2
KC.D	10	4
KC.E	10	7
KC.F	13	4
KC.G	13	11
KC.H	11	9
KC.I	16	7
KC.J	9	4
KC.K	16	4
KC.L	14	4
KC.M	9	2
KC.N	9	1
KC.O	14	7
KC.P	15	7
KC.Q	7	6
KC.R	13	7
KC.S	12	4
KC.T	13	0
KC.U	9	7
KC.V	13	2
KC.W	12	7

KC.X	12	2
KC.Y	9	0
KC.Z	6	2
KC.GRAVE	8	6
KC.N1	6	5
KC.N2	12	5
KC.N3	10	5
KC.N4	13	5
KC.N5	13	8
KC.N6	9	8
KC.N7	9	5
KC.N8	16	5
KC.N9	14	5
KC.N0	15	5
KC.MINUS	15	8
KC.EQUAL	16	8
KC.BSPACE	22	0
KC.ESC	11	6
KC.F1	12	8
KC.F2	10	8
KC.F3	10	0
KC.F4	11	10
KC.F5	22	11
KC.F6	16	11
KC.F7	14	0
KC.F8	14	8
KC.F9	22	8
KC.F10	22	5
KC.F11	21	5
KC.F12	19	5
KC.INSERT	23	5
KC.DELETE	23	8
KC.RIGHT	19	1
KC.LEFT	20	1

KC.UP	20	11
KC.DOWN	21	1
KC.SLASH	15	1
KC.DOT	14	2
KC.COMMA	16	2
KC.SCOLON	15	4
KC.QUOTE	15	11
KC.ENTER	22	2
KC.LBRC	15	0
KC.RBRC	16	0
KC.BSLASH	22	4
KC.CAPS	12	0
KC.TAB	6	0
KC.SPACE	22	1
KC.HOME	21	8
KC.END	20	5
KC.PGUP	23	2
KC.PGDOWN	23	1
KC.PSCREEN	23	4
KC.SCROLLLOCK		
KC.NUM_LOCK	30	11
KC.PAUSE	**11**	**28 (see Appendix B)**
KC.PSLS	30	0
KC.PAST	30	7
KC.PMNS	30	8
KC.PPLS	30	1
KC.PENT	31	2
KC.PDOT	20	2
KC.P0	31	1
KC.P1	31	8
KC.P2	31	4
KC.P3	31	5
KC.P4	31	11
KC.P5	31	0

KC.P6		31	7
KC.P7		30	4
KC.P8		30	5
KC.P9		30	2
KC.AUDIO_MUTE	Func	12	8
KC.AUDIO_VOL_UP	Func	10	0
KC.AUDIO_VOL_DOWN	Func	10	8
KC.BRIGHTNESS_UP	Func	16	11
KC.BRIGHTNESS_DOWN	Func	22	11
KC.MEDIA_NEXT_TRACK	Func		
KC.MEDIA_PREV_TRACK	Func		
KC.MEDIA_STOP	Func		
KC.MEDIA_PLAY_PAUSE	Func		
KC.MEDIA_EJECT	Func		
KC.MEDIA_FAST_FORWARD	Func		
KC.MEDIA_REWIND	Func		

Appendix B - Semi-Automated Matrix Generator Procedure

This procedure assumes you have already used the matrix decoder program to create a pin connection list for all the keys. The manual procedure to determine the row and column pins can be time consuming so Marcel Hillesheim wrote a Python 3 program for the Teensy LC and 3.2 that eliminates most of the manual labor. I have updated his program over the years to add more Teensy controllers and have now modified it for KMK on the Raspberry Pi Pico or Solder Party 2350 XL .

Download the KMK_matrixgenerator.py Python 3 program to your PC. Load it into Thonny and in the lower right corner, select Local Python 3 so it will run on your PC.

Move the completed keyboard pin list text file into the same directory as the Python program so it's easy to find. You don't need to edit the connection list to remove keys that you don't have. The Python program skips the unused keys automatically.

In Thonny, select the Run icon. The console window will show all the text files it sees in the directory. Here is an example of the Thonny console window with comments in *italics*:

```
index      file name

  1   Keyboard_pin_list_KMK_1545.txt

  2   Keyboard_pin_list_KMK_E550.txt

----------------------------------------------------

Enter the index number of the *.txt file you want.

OR: enter your own filepath:
```

I've got 2 text files for 2 keyboards in the directory. For this example, I'll type 2 and Enter to use the E550 keyboard pin list:

```
1

----------------------------------------------------

index    device

  1      2350

----------------------------------------------------

Please enter the index number of your device:
```

Currently there is only 1 device so type 1 and Enter to get the following:

1

initial output pin: 17   *the program looks at Control Left and Control Right to get its bearings*

Results:

GPIO PINS:

8 input column GPIO's:

[0, 1, 2, 4, 5, 7, 8, 11]

18 output row GPIO's:

[3, 6, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 30, 31]

It was at this point that I learned that Marcel's Python program had left out GPIO's 28 and 29 for the FN key. Looking at the KEY matrix confirmed that FN is missing. Apparently, a key switch that doesn't share its row or column with any other key will be skipped by the program. I could have manually added GPIO's 28 and 29 to the list and added a new row and column to the KEY matrix but this seems like a lot of work. An easier solution was to go back to the pin connection text file and pick an unused key like Pause and give it numbers like 28 and 11. 11 was chosen because it's already known to be a column and 28 is one of the FN GPIO's so it's no longer all alone. The following results are after making this edit to the pin connection text file.

GPIO PINS:

9 input column GPIO's:

[0, 1, 2, 4, 5, 7, 8, 11, 29]   *Now GPIO 29 is listed*

19 output row GPIO's:

[3, 6, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 28, 30, 31]   *Now GPIO 28 is listed*

KEY T*his is the base layer key matrix with the correct names needed by KMK. I'll remove KC.PAUSE when I copy it over to the KMK code.py file.*

```
{
{KC.LSHIFT,KC.NO,KC.RSHIFT,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.TAB,KC.NO,KC.Z,KC.A,KC.N1,KC.Q,KC.GRAVE,KC.ESC,KC.NO},
{KC.Y,KC.N,KC.M,KC.J,KC.N7,KC.U,KC.N6,KC.H,KC.NO},
{KC.F3,KC.NO,KC.C,KC.D,KC.N3,KC.E,KC.F2,KC.F4,KC.NO},
{KC.CAPS,KC.NO,KC.X,KC.S,KC.N2,KC.W,KC.F1,KC.NO,KC.NO},
{KC.T,KC.B,KC.V,KC.F,KC.N4,KC.R,KC.N5,KC.G,KC.NO},
{KC.F7,KC.NO,KC.DOT,KC.L,KC.N9,KC.O,KC.F8,KC.NO,KC.NO},
{KC.LBRC,KC.SLASH,KC.NO,KC.SCOLON,KC.N0,KC.P,KC.MINUS,KC.QUOTE,KC.NO},
{KC.RBRC,KC.NO,KC.COMMA,KC.K,KC.N8,KC.I,KC.EQUAL,KC.F6,KC.NO},
{KC.NO,KC.NO,KC.RCTRL,KC.NO,KC.NO,KC.NO,KC.LCTRL,KC.NO,KC.NO},
{KC.NO,KC.RALT,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.LALT,KC.NO},
{KC.LGUI,KC.RIGHT,KC.NO,KC.NO,KC.F12,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.LEFT,KC.PDOT,KC.NO,KC.END,KC.NO,KC.NO,KC.UP,KC.NO},
{KC.NO,KC.DOWN,KC.NO,KC.NO,KC.F11,KC.NO,KC.HOME,KC.NO,KC.NO},
{KC.BSPACE,KC.SPACE,KC.ENTER,KC.BSLASH,KC.F10,KC.NO,KC.F9,KC.F5,KC.NO},
{KC.NO,KC.PGDOWN,KC.PGUP,KC.PSCREEN,KC.INSERT,KC.NO,KC.DELETE,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.PAUSE,FN},
{KC.PSLS,KC.PPLS,KC.P9,KC.P7,KC.P8,KC.PAST,KC.PMNS,KC.NUM_LOCK,KC.NO},
{KC.P5,KC.P0,KC.PENT,KC.P2,KC.P3,KC.P6,KC.P1,KC.P4,KC.NO},
}
```

Func *this shows the Fn-Media key locations but it needs further editing*

```
{
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.AUDIO_VOL_UP,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.AUDIO_VOL_DOWN,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.AUDIO_MUTE,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.BRIGHTNESS_UP,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.BRIGHTNESS_DOWN,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
}
```

```
----------------------------------------------------
MODIFIER            this matrix is not used for KMK. It’s left over from the Teensy code
----------------------------------------------------
{
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
                                          :
                                          :
{KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO,KC.NO},
}
----------------------------------------------------
ONE                this matrix is not used for KMK. It’s left over from the Teensy code
----------------------------------------------------
{
{1,1,1,1,1,1,1,1,1},
           :
           :
{1,1,1,1,1,1,1,1,1},
}
----------------------------------------------------
Finished
```

Next, copy and edit the above information into your code.py KMK routine (using the E550 example code).

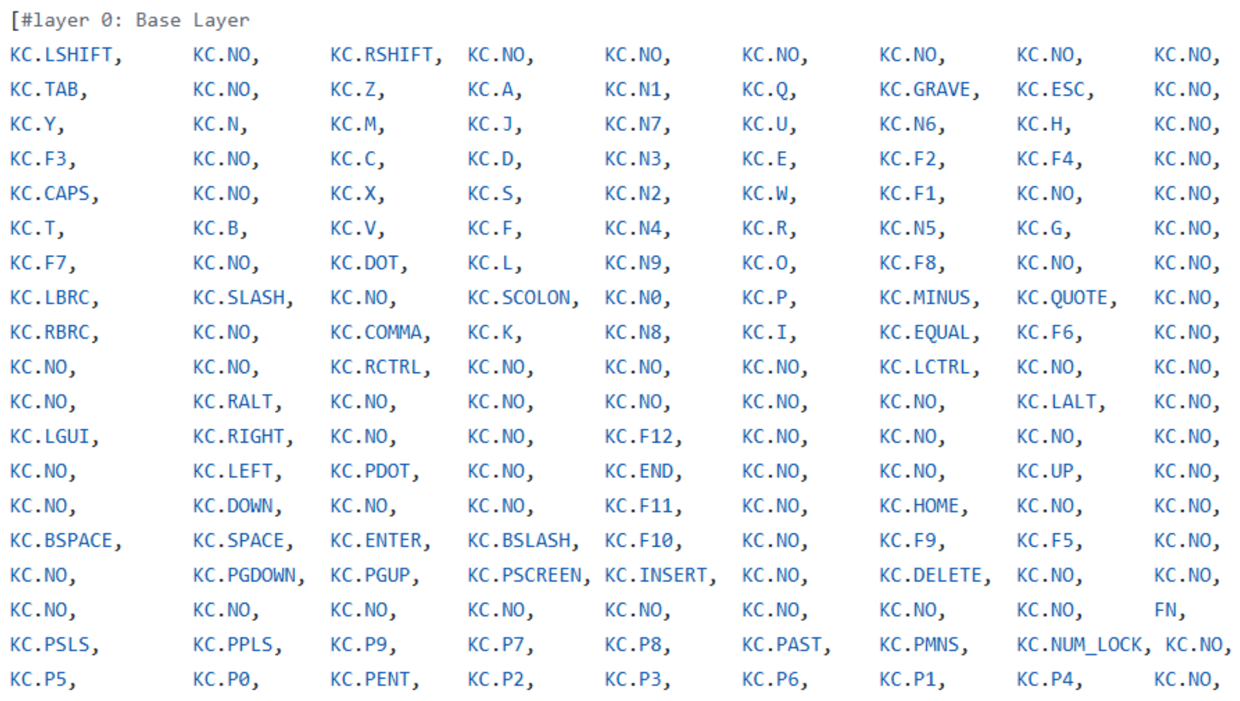
The 9 input column GPIO's are [0, 1, 2, 4, 5, 7, 8, 11, 29]. These are listed on line 27 of code.py as the keyboard column pins, adding board.GP in front of each number:

keyboard.col_pins = (board.GP0, board.GP1, board.GP2, board.GP4, board.GP5, board.GP7, board.GP8, board.GP11, board.GP29)

The 19 output row GPIO's are [3, 6, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 28, 30, 31]. These are listed on lines 28 and 29 as the keyboard row pins, adding board.GP in front of each number:

keyboard.row_pins = (board.GP3, board.GP6, board.GP9, board.GP10, board.GP12, board.GP13, board.GP14, board.GP15, board.GP16, board.GP17, board.GP18, board.GP19, board.GP20, board.GP21, board.GP22, board.GP23, board.GP28, board.GP30, board.GP31)

The “KEY” matrix from the matrix generator output could be used as-is (with Pause removed) for the layer 0 Base Layer found at lines 44 – 62 of code.py but it’s more readable if tabs are added as shown below.



```
[#layer 0: Base Layer
KC.LSHIFT,      KC.NO,      KC.RSHIFT,  KC.NO,      KC.NO,      KC.NO,      KC.NO,      KC.NO,      KC.NO,
KC.TAB,         KC.NO,      KC.Z,       KC.A,       KC.N1,      KC.Q,       KC.GRAVE,   KC.ESC,     KC.NO,
KC.Y,           KC.N,       KC.M,       KC.J,       KC.N7,      KC.U,       KC.N6,      KC.H,       KC.NO,
KC.F3,          KC.NO,      KC.C,       KC.D,       KC.N3,      KC.E,       KC.F2,      KC.F4,      KC.NO,
KC.CAPS,        KC.NO,      KC.X,       KC.S,       KC.N2,      KC.W,       KC.F1,      KC.NO,      KC.NO,
KC.T,           KC.B,       KC.V,       KC.F,       KC.N4,      KC.R,       KC.N5,      KC.G,       KC.NO,
KC.F7,          KC.NO,      KC.DOT,     KC.L,       KC.N9,      KC.O,       KC.F8,      KC.NO,      KC.NO,
KC.LBRC,        KC.SLASH,   KC.NO,      KC.SCOLON,  KC.N0,      KC.P,       KC.MINUS,   KC.QUOTE,   KC.NO,
KC.RBRC,        KC.NO,      KC.COMMA,   KC.K,       KC.N8,      KC.I,       KC.EQUAL,   KC.F6,      KC.NO,
KC.NO,          KC.NO,      KC.RCTRL,   KC.NO,      KC.NO,      KC.NO,      KC.LCTRL,   KC.NO,      KC.NO,
KC.NO,          KC.RALT,    KC.NO,      KC.NO,      KC.NO,      KC.NO,      KC.NO,      KC.LALT,    KC.NO,
KC.LGUI,        KC.RIGHT,   KC.NO,      KC.NO,      KC.F12,     KC.NO,      KC.NO,      KC.NO,      KC.NO,
KC.NO,          KC.LEFT,    KC.PDOT,    KC.NO,      KC.END,     KC.NO,      KC.NO,      KC.UP,      KC.NO,
KC.NO,          KC.DOWN,    KC.NO,      KC.NO,      KC.F11,     KC.NO,      KC.HOME,    KC.NO,      KC.NO,
KC.BSPACE,      KC.SPACE,   KC.ENTER,   KC.BSLASH,  KC.F10,     KC.NO,      KC.F9,      KC.F5,      KC.NO,
KC.NO,          KC.PGDOWN,  KC.PGUP,    KC.PSCREEN, KC.INSERT,  KC.NO,      KC.DELETE,  KC.NO,      KC.NO,
KC.NO,          KC.NO,      KC.NO,      KC.NO,      KC.NO,      KC.NO,      KC.NO,      KC.NO,      FN,
KC.PSLS,        KC.PPLS,    KC.P9,      KC.P7,      KC.P8,      KC.PAST,    KC.PMNS,    KC.NUM_LOCK, KC.NO,
KC.P5,          KC.P0,      KC.PENT,    KC.P2,      KC.P3,      KC.P6,      KC.P1,      KC.P4,      KC.NO,
```

The Layer 1 Fn Media layer on lines 67 – 85 is created by copying the base layer and then replacing the 5 media specific keys at the locations shown in the “Func” matrix.

```
[#layer 1: Fn Media Layer
 KC.NO,         KC.NO,      KC.PGDOWN, KC.LGUI,    KC.NO,      KC.NO,     KC.NO,      KC.PGUP,
 KC.NO,         KC.RCTRL,   KC.NO,     KC.NO,      KC.NO,      KC.NO,     KC.LCTRL,   KC.NO,
 KC.RALT,       KC.NO,      KC.NO,     KC.PSCREEN, KC.NO,      KC.LALT,   KC.NO,      KC.EJCT,
 KC.NO,         KC.Z,       KC.A,      KC.N1,      KC.TAB,     KC.ESC,    KC.GRAVE,   KC.Q,
 KC.NO,         KC.C,       KC.D,      KC.N3,      KC.F3,      KC.BRID,   KC.F2,      KC.E,
 KC.SPACE,      KC.ENTER,   KC.BSLASH, KC.MPRV,    KC.BSPACE,  KC.BRIU,   KC.VOLU,    KC.NO,
 KC.NO,         KC.COMMA,   KC.K,      KC.N8,      KC.RBRC,    KC.F6,     KC.EQUAL,   KC.I,
 KC.NO,         KC.DOT,     KC.L,      KC.N9,      KC.MUTE,    KC.NO,     KC.VOLD,    KC.O,
 KC.LEFT,       KC.NO,      KC.NO,     KC.END,     KC.NO,      KC.UP,     KC.HOME,    KC.NO,
 KC.RIGHT,      KC.NO,      KC.NO,     KC.MNXT,    KC.NO,      KC.NO,     KC.INSERT,  KC.NO,
 KC.DOWN,       KC.NO,      KC.NO,     KC.MPLY,    KC.NO,      KC.NO,     KC.DELETE,  KC.NO,
 KC.SLASH,      KC.NO,      KC.SCOLON, KC.N0,      KC.LBRC,    KC.QUOTE,  KC.MINUS,   KC.P,
 KC.N,          KC.M,       KC.J,      KC.N7,      KC.Y,       KC.H,      KC.N6,      KC.U,
 KC.B,          KC.V,       KC.F,      KC.N4,      KC.T,       KC.G,      KC.N5,      KC.R,
 KC.NO,         KC.X,       KC.S,      KC.N2,      KC.CAPS,    KC.NO,     KC.F1,      KC.W,
 KC.NO,         KC.RSHIFT,  KC.NO,     KC.NO,      KC.LSHIFT,  KC.NO,     KC.NO,      KC.NO,
 KC.NO,         KC.NO,      FN,        KC.NO,      KC.NO,      KC.NO,     KC.NO,      KC.NO,
```

Final Notes:

Most of the KMK Keycodes are obvious but some are hard to understand, especially the short “alias” names. Here are the strange ones:

KC.N0 – KC.N9 are the number keys 0 – 9.

KC.LBRC is the left brace and KC.RBRC is the right brace.

KC.GRAVE is ` (to the left of the 1 key)

KC.PSLS is Pad slash

KC.PAST is Pad asterisk

KC.PMNS is Pad minus

KC.PPLS is Pad plus

KC.PENT is Pad enter

KC.PDOT is Pad dot (period)

KC.P0 – KC.P9 are Pad numbers 0 - 9