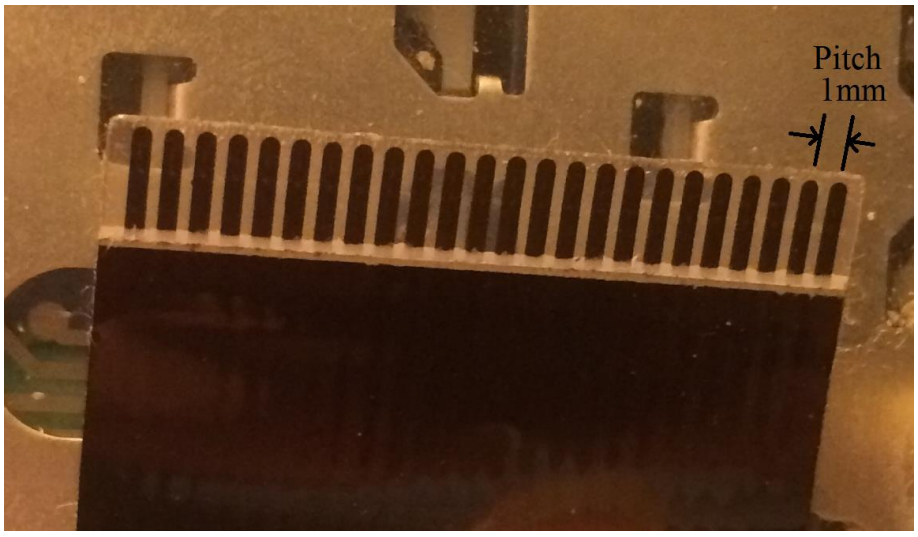
Instructable - Laptop Keyboard Conversion to USB

This Instructable will provide a step by step procedure for building a USB laptop keyboard controller. I created this guide and video to hopefully make it easier for people to re-purpose an old laptop. A typical laptop keyboard relies on the motherboard for the scanning circuitry. I use a Teensy microcontroller mounted on a connector board to take over this function. Teensies are often used by the mechanical keyboard enthusiasts at Geekhack and Deskthority and the TMK software is the most popular controller code. The TMK code is a bit of an overkill if you just want a simple USB keyboard but it will certainly provide all the features you could ever need. If you would rather write your own keyboard software using Arduino, the Teensyduino functions give you total USB control. Whatever software you decide to use, it will require a key matrix that maps out how your keyboard is wired. One approach, (that I never want to do again) is to exhaustively check every connector pin combination with an ohm meter while holding down each key. I did this when I converted a Sony Vaio into a Raspberry Pi laptop.
An Instructable from alpinedelta disassembles the keyboard in order for the connections to each switch to be visually traced back to the connector. Instead of taking the keyboard apart or using an ohm meter, this Instructable will load the Teensy with an automated continuity tester. The Teensy will report over USB, the two pin numbers that are connected when you press a key. After every key has been pressed, the results can be transferred to a row-column matrix and used by the TMK keyboard controller software or a home-brew Teensyduino routine.

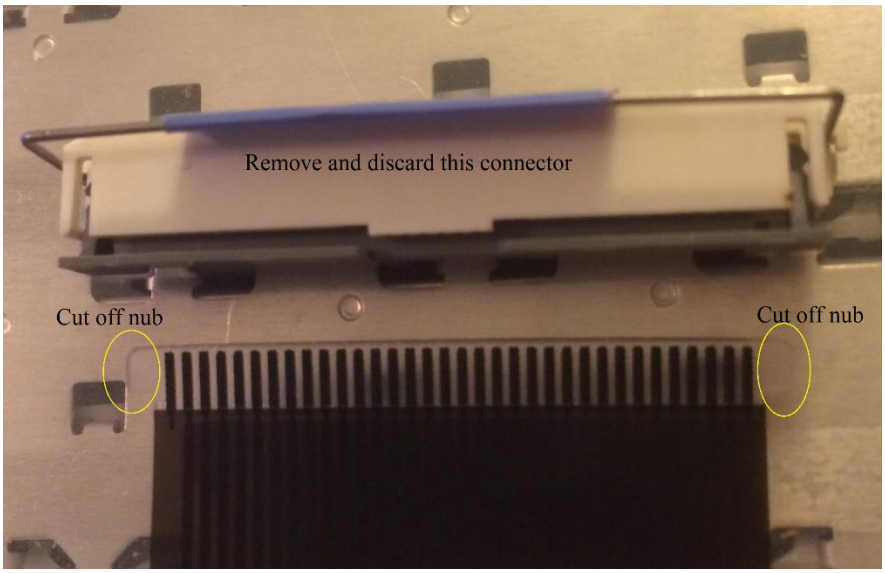
Laptop keyboards use a flexible printed circuit (FPC) that connects all the key switches in an array of rows and columns. There are many different FPC to motherboard connection methods but a large number of laptops use a set of traces that end with exposed metal on one side and a plastic backing on the other side. The plastic backing plus the FPC material measure about 0.33 mm thick on a standard cable. A typical keyboard without a number pad has 24 or 25 signal traces with a 1 mm pitch. If there is a number pad, then it's common to have 26 traces with a 1 mm pitch. Most of the FPC cables I tested will fit in a generic connector with no modifications. Connectors for this style of keyboard FPC cable are readily available from companies like Aliexpress or Digikey. The number of signal traces and the pitch are the parameters you will need when ordering. The picture below shows a standard 24 pin FPC Cable with a 1mm trace pitch.



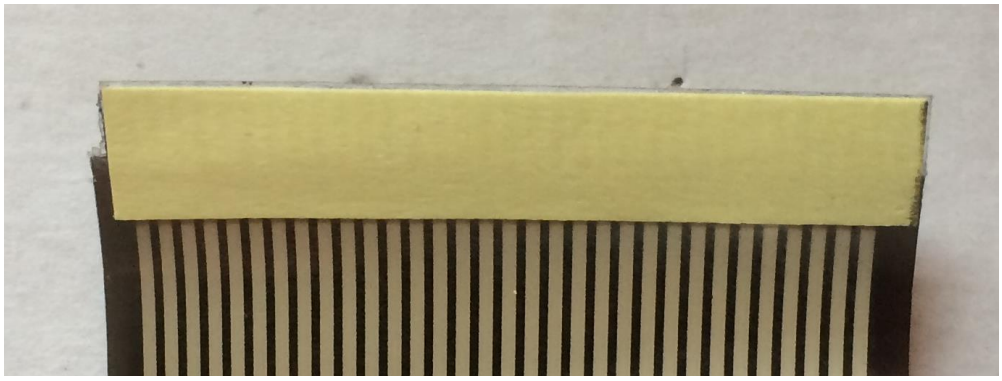
Some FPC cables need to be modified to fit in a generic connector. Locking nubs on the side of the cable are easy to remove with wire cutters.

If the FPC traces don't line up with the connector pins, use an X-ACTO knife to trim along the side of the cable.

The Dell Latitude D630 keyboard needed the most modifications. It had a solder-less connector on the end of the FPC cable that was easily removed. Then I trimmed off the side nubs and made a notch to align the contacts.

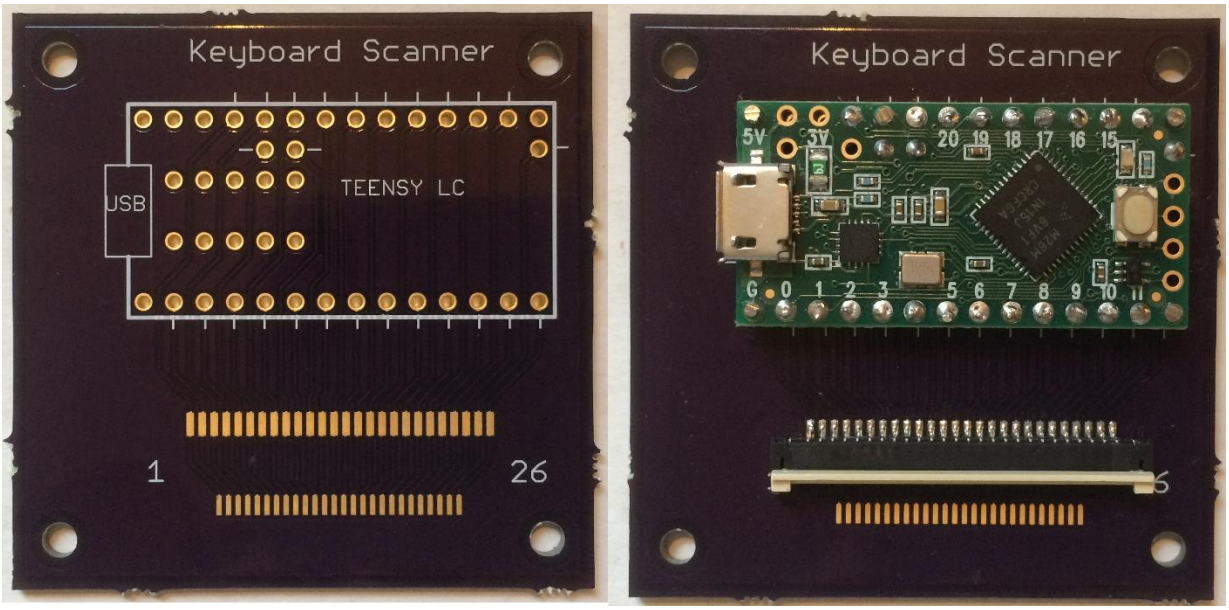


The final mod was to remove the extra thick plastic backing on the end of the cable and replace it with 2 pieces of a paper to bring the thickness back to normal for a generic FPC connector.



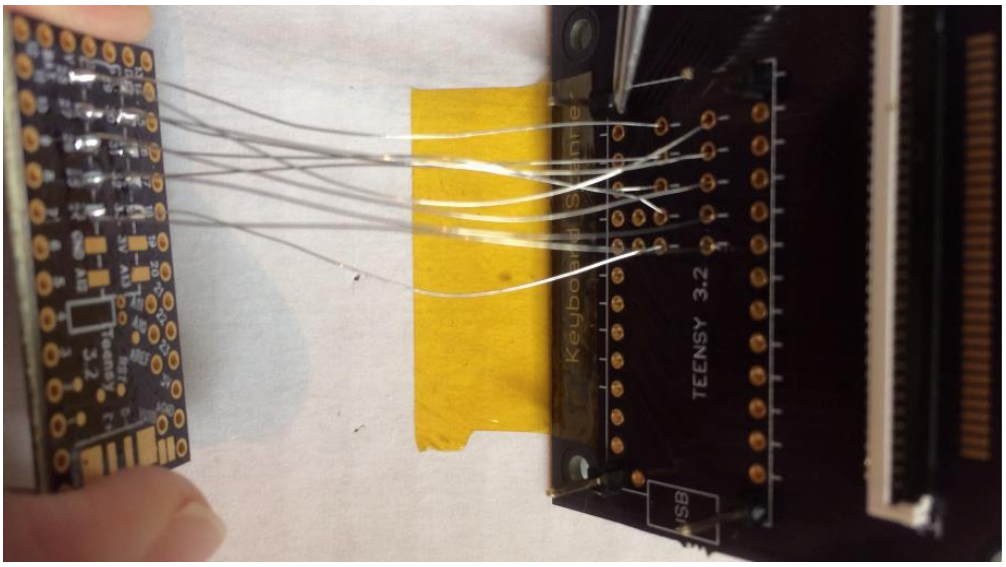
I designed a circuit board using Eagle for the Teensy LC that routes its 26 I/O pins to 26 surface mount pads for an FPC connector with a 1mm pitch or 0.8mm pitch. A 24, 25, or 26 pin FPC connector can be soldered to this board based on your needs. I avoided using the 27th Teensy LC output because it's connected to an LED and 27 pin FPC connectors are rare.

After soldering the FPC connector to the board, I soldered 4 header posts to the board to support the corners of the Teensy and then I soldered the Teensy to the header posts. The last step was to connect the rest of the Teensy I/O signals to the board with 30 gauge wire. I used wire instead of header posts to make it easy to cut the Teensy off the board for future projects. The Teensy pads that must be connected to the board are marked with a small line. The bare and assembled LC board is shown below.

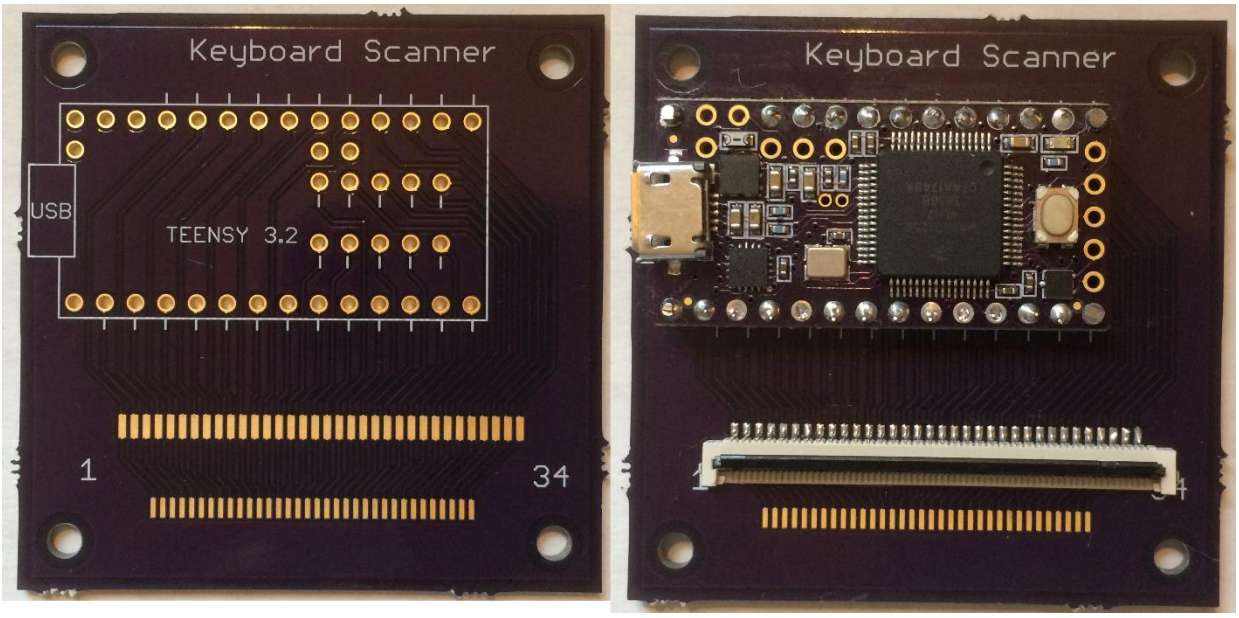


All of the signals for the Teensy LC are routed on one side of my board so I designed the other side for a Teensy 3.2 with 34 I/O signals and an FPC connector with a 1 mm or 0.8 mm pitch. If you need all 34 pins, you must unsolder the LED on the Teensy 3.2 to free it up for use by the keyboard.

The Teensy 3.2 uses surface mount pads for 10 of the I/O signals so it's a little more work to solder them to the board. After soldering the FPC connector to the 3.2 side of the board, solder 10 "flying leads" to the surface mount pads on the Teensy 3.2 and pass each wire thru the corresponding pad on the board.



Finish the assembly by soldering 30 gauge solid wire to the remaining I/O signals marked with a small line. The bare and assembled 3.2 board is shown below.



The Parts List is given below.

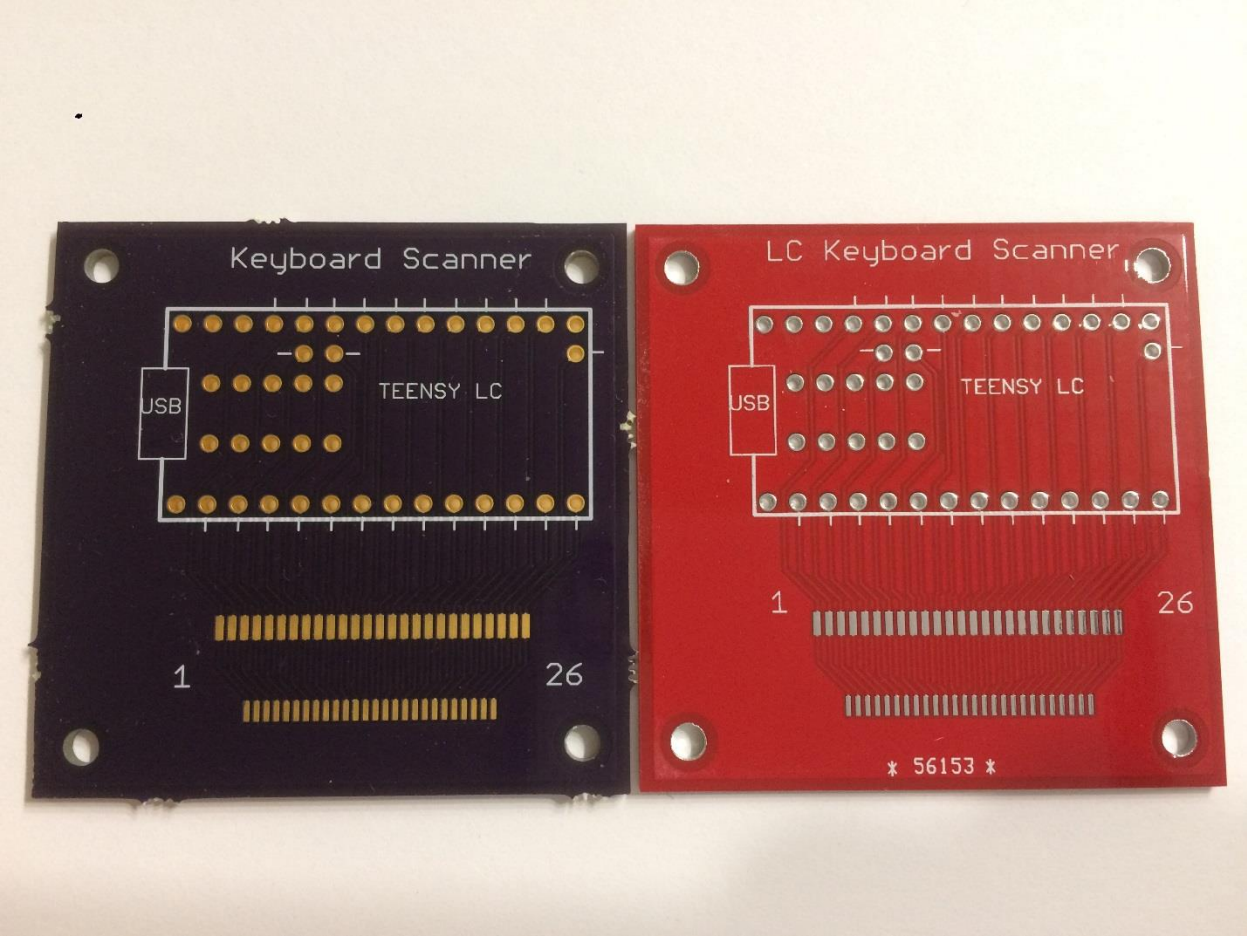
1. The Teensy LC is $15.53 or the Teensy 3.2 is $23.49 from Amazon. You can also order direct from PJRC.

2. The FPC connectors from AliExpress are about $5 for a lot of 5. An example search on their website would be “laptop keyboard connector 1.0 spacing 24 pin”. Digikey is another possible source for connectors.

3. Fabrication of the circuit board costs $18 from OSH Park for a lot of 3 (purple board shown below), or $14 from DirtyPCBs for their "protopack" lot of about 10 (red board shown below). OSH Park fabricates the boards in the United States and my order took 12 days to arrive in Tacoma, Washington. DirtyPCBs are fabricated in China and it took 28 days for the boards to be delivered. If you don’t need to make any modifications, then you can send the Eagle board file directly to either vendor.

Other Items needed include header pins, wire, solder, flux, and USB cable.

Notes about the DirtyPCBs protopack: I received 10 flawless boards and a couple extra that had some minor flaws so I used them for solder testing. When ordering, I used the default thickness of 1.2mm instead of the 1.6mm OSH Park thickness. The other difference was the selection of HASL for the surface finish instead of ENIG which costs $15 more. OSH Park boards have perfectly smooth SMD pads due to their ENIG finish but the HASL finish on the DirtyPCBs boards have slightly uneven SMD pads. The different PCB surface finishes are explained in this Optimum Design Associates post. I've soldered 1mm pitch and 0.8mm pitch SMD connectors to the DirtyPCBs board with no problems. The HASL finish might start to be a problem with finer pitch connectors but for this project, it works fine and keeps the cost low.



Once you have the Teensy and FPC connector soldered to the board, follow these steps to decode your keyboard matrix.

Load the Continuity Tester into the Teensy per the following procedure.

- Follow the PJRC link for installing Arduino and Teensyduino on your computer.
- Download the Arduino code from the Pin_Continuity_Tester folder at my GitHub repo. Use file Matrix_Decoder_LC.ino for a Teensy LC or Matrix_Decoder_3p2.ino for a Teensy 3.2.
- Load the Matrix_Decoder code into the Arduino integrated development environment (IDE).
- Connect a USB cable from the Teensy to the computer. Your computer should automatically load the necessary USB drivers.
- In the Arduino IDE, under “tools”, select board: Teensy LC or Teensy 3.2/3.1 depending on what you’re using. Also under “tools”, select USB type: Keyboard.
- Compile and load the Matrix_Decoder code into the Teensy. If it’s your very first time loading the Teensy, you’ll have to push the button on the Teensy to enable the loader.
- Disconnect the USB cable from the Teensy.

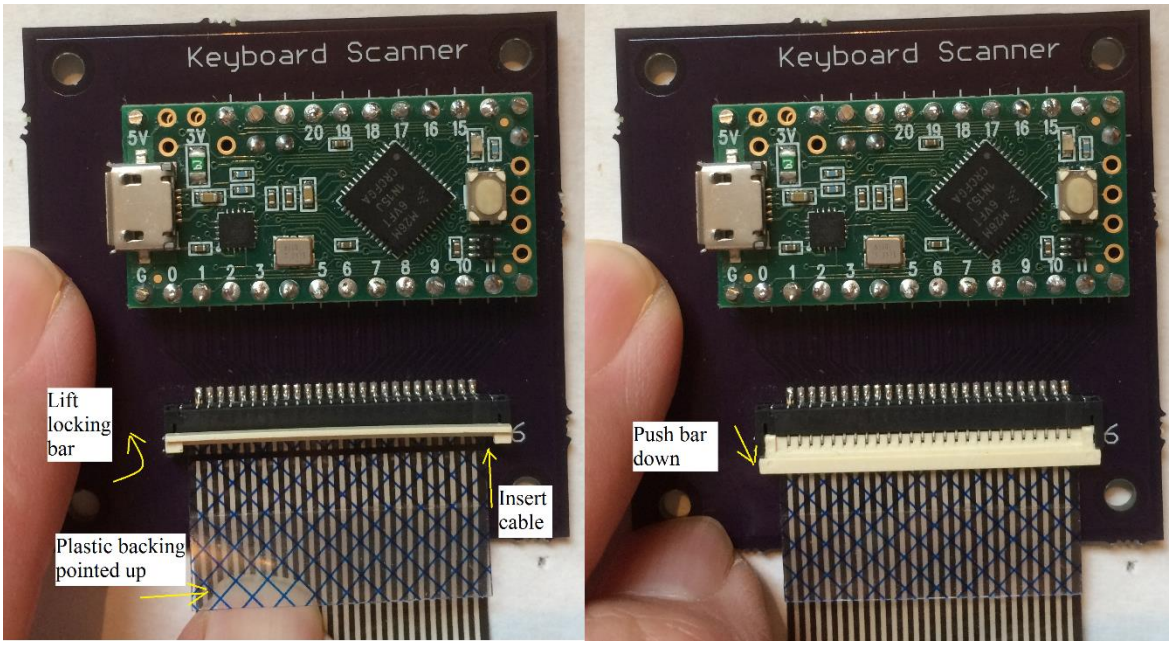
Open a text editor on your computer. I like to use Notepad++ on Windows or Geany on the Pi because they have column editing. There are two “key-list” text files you can download at my repo, one for a keyboard with a number pad and one for no number pad. The “key-list” file lists each key that you will push followed by tabs to make the results more readable and easy to copy into a spreadsheet. You will

probably need to modify the file slightly to match your keyboard's keys. A non-US keyboard can still use this routine, just make a list of your keys and the program will report pin connections. The GUI key in my list is either the "windows key" from a PC or the "clover key" from a Mac. Place the cursor to the right of the very first key in the list as shown below. This will determine where the Teensy begins to display the pin numbers as you push each key.

```
Cntrl-L
Cntrl-R
Shift-L
Shift-R
Alt-L
Alt-R
GUI
Fn
A
B
C
```

Put Cursor Here

Use your finger nail to gently lift the locking bar on the FPC board connector to the open position. Slide the FPC cable into the connector with the bare metal contacts pointed down (closest to the board) and the plastic backing strip pointed up. Lock the cable to the connector by gently pushing the bar down. Connect a USB cable from the Teensy to the computer and wait 20 seconds for the Teensy to be recognized as a USB keyboard. This delay is in the code to make sure your computer is ready to receive numbers from the Teensy. If numbers are reported on the screen before any keys are pressed, these pins are shorted together and must be fixed. If you have an FPC cable with more than 26 signals, it may use some of the extra traces for grounds, back-lighting, or a track-pad. This may cause the test routine to register two pins as shorted. If this happens, you'll need to do some code modifications to exclude these pins. The code normally scans all pins starting from pin 1 and ending at the last pin but you can adjust these variables to avoid shorts.



Press each key, one by one on the test keyboard as listed on the editor screen. The Teensy will send two pin numbers over USB that were connected when the key was pressed. The Teensy will then send a down arrow to position the cursor for the next key. If your keyboard contains diodes on the switches, the first pin is the cathode side and the second pin is the anode side.

```
Cntrl-L         19  20
Cntrl-R         20  22
Shift-L         21  23
Shift-R         23  25
Alt-L           7   24
Alt-R           7   15
GUI             9   26
Fn              12  18
A               16  22
B               13  15
C               14  21
```

After pressing every key on the keyboard, save the finished file for analysis. To determine the Input and output pins, follow a few simple rules based on the Modifier keys; Control, Alt, Shift, GUI, and Fn. As a general rule, 8 of the keyboard pins will be inputs to the Teensy and the remainder will be outputs. The Modifier keys usually have an output row all to themselves which allows these keys to be held down while other keys are pressed. This avoids a sneak path which would cause ghosting. These "rules" are not always followed (especially by the Fn key) so you may need to do some trial and error as you build the matrix. I have lots of keyboard examples at my Github repo to help you out.

Control-Left and Control-Right will have a common pin between them. Example:

Cntrl-L 19 20

Cntrl-R 20 22

The common pin, Pin 20 in this example, will be a Teensy output, and 19 & 22 will be inputs.

Similarly Alt-Left and Alt-Right will have a common pin between them, just as Shift-Left and Shift-Right will also have a common pin. Example:

Alt-L 7 24

Alt-R 7 15

Shift-L 21 23

Shift-R 23 25

The Alt common pin will be a Teensy output, and 15 & 24 will be inputs.

The Shift common pin will be a Teensy output, and 21 & 25 will be inputs.

The GUI key is usually a single key as in this example;

GUI 9 26

Search all the other pins in the list to see if 9 or 26 are used on other keys. In this example, pin 9 was not used for any other key so it will be a Teensy output and 26 will be an input. Sometimes both pins are used for other keys but one of the pins is used for common keys like letters and numbers and the other pin is used for less-common keys like page-up. In this case the pin used for common keys will be a Teensy input and the other pin will be an output. Note that the GUI key will still work if you swap the pins.

The Fn key is also a single key as in this example;

Fn 12 18

Using the same approach as the GUI key, search all the other pins to see if 12 or 18 are used on other keys. In this example, pin 12 was not used for any other key so it will be an output and 18 will be an input. If both pins are used on other keys, follow the same rules as the GUI example. Sometimes both pins are used by common keys which means you can pick either pin as an input and the other as an output.

The eight input pins for the HP DV9000 example keyboard have been identified as; 15, 18, 19, 21, 22, 24, 25, and 26. All other pins will be Teensy outputs. Make a keyboard matrix table like the one shown below with the 8 input pins across the top in ascending order and all the other pins as outputs on the side, also in ascending order.

Inputs to the Teensy

Outputs from the Teensy

FPC Connector Pin Number	15	18	19	21	22	24	25	26
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
16								
17								
20								
23								

The orientation of the keyboard matrix is just my personal preference. You can swap the rows/columns and inputs/outputs if you want. Swapping pins may be necessary if you have a rare laptop keyboard that has diodes on its switches.

Sometimes only 7 pins can be identified as inputs because two modifier keys share the same input pin. If this happens, you'll have to make an educated guess for the 8th input. For some keyboards, the input pins are grouped together (i.e., 17 thru 24) which makes it easy to fill in the missing input pin. Other

keyboards have no grouping of inputs so you'll have to begin filling out the matrix with only 7 inputs. The missing input pin will reveal itself when some of the keys can't be placed in the matrix.

Place each key name at the row/column intersection of the pins as shown in the HP DV9000 keyboard example shown below. The modifier keys are in bold so you can see that they have a row all to themselves. This keyboard followed the "rules" exactly.

Inputs to the Teensy

Outputs from the Teensy

FPC Connector Pin Number	15	18	19	21	22	24	25	26
1	Pad 7	Pad +	Pad 9	Pad 2	Pad 3	Pad 1	Pad Enter	Pad 8
2	Pad /	Pad 6	Pad -	Pad 5	Pad .	Pad 4	Pad 0	Pad *
3	Del	Page-Up	Home	End	Page-Down	Arrow-Right	Arrow-Left	Insert
4	Equal	Minus	[	/	;	0	P	Quote
5	F12	F9	F10	Period	L	9	O	F11
6		\	Num-Lock	Space	Enter	Arrow-Down	Arrow-Up	Back-Space
7	**Alt-R**					**Alt-L**		
8		Menu						
9								**GUI**
10			]	Comma	K	8	I	
11	N	6	Y	M	J	7	U	H
12		**Fn**						
13	B	5	T	V	F	4	R	G
14	F1	F2	F3	C	D	3	E	F4
16	Caps-Lock	Back-Tick	Tab	Z	A	1	Q	Esc
17	F8	F5	F6	X	S	2	W	F7
20			**Cntrl-L**		**Cntrl-R**			
23				**Shift-L**			**Shift-R**	

You will need to translate the row and column FPC pin numbers to the Teensy I/O numbers for your keyboard controller code. The two tables below show how the FPC connector pins are routed to the Teensy LC I/O's and the Teensy 3.2 I/O's.

Teensy LC

FPC Pin #	Teensy LC I/O #
1	23
2	0
3	22
4	1
5	24
6	2
7	21
8	3
9	25
10	4
11	20
12	5
13	19
14	6
15	18
16	7
17	17
18	8
19	16
20	9
21	15
22	10
23	14
24	11
25	26
26	12

Teensy 3.2

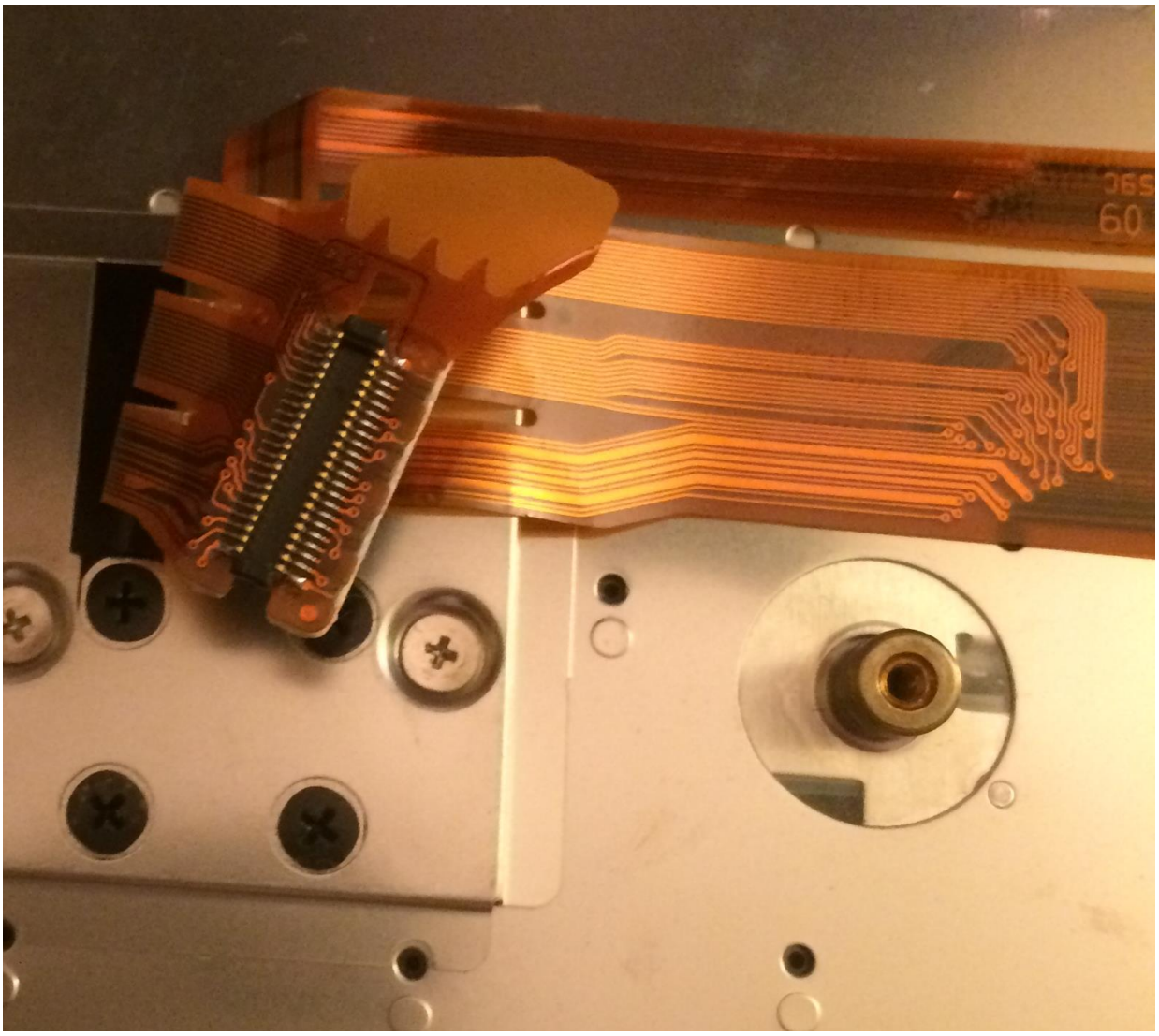
FPC Pin #	Teensy 3.2 I/O #
1	23
2	0
3	22
4	1
5	21
6	2
7	20
8	3
9	19
10	4
11	18
12	5
13	17
14	6
15	24
16	7
17	25
18	8
19	33
20	9
21	26
22	10
23	27
24	11
25	28
26	12
27	32
28	31
29	30
30	29
31	16
32	15
33	14
34	13

A Deskthority post from "flabbergast" describes using the ChibiOS development environment to configure TMK for ARM based processors like those used on the Teensy LC and 3.2. A toolchain such as the GNU ARM Embedded Toolchain is used to compile the code for the Teensy LC or 3.2. You will need to install the ChibiOS development environment per these instructions.
The teensy_lc_onekey example details the steps to create a working TMK build. The QMK keyboard routine is based on TMK and it also has ChibiOS support for the Teensy LC and 3.2.

The TMK/QMK keyboard software is very powerful with tons of features but it can be confusing, (at least to me). As an alternative, I wrote an Arduino USB keyboard routine using the Teensyduino "Micro-Manager" functions. There's just 1 file to load using the Arduino IDE and it's only about 375 lines with lots of comments. I'm a hardware guy so expect ugly code but it provides a basic keyboard controller with 6 key rollover that you can modify to suite your needs. There are detailed instructions on how to modify my LC code and 3.2 code to work with your matrix. Every keyboard listed below has a folder containing a pin connect list, key matrix, and a Teensyduino USB keyboard routine giving you lots of examples to follow.

- Dell Inspiron 1525 - Keyboard Part Number D9K01
- Dell Latitude 131L - Keyboard Part Number V-0511BIAS1-US
- Dell Latitude X1 - Keyboard Part Number 0M6607
- Dell Latitude D630 - Keyboard Part Number DP/N 0DR160
- HP Compaq Presario 2100 - Keyboard Part Number AEKT1TPU011
- HP Compaq Presario V4000 - Keyboard Part Number NSK-H3L01
- HP Pavilion DV9000 - Keyboard Part number AEAT5U00110
- Sony Vaio PCG-K25 - Keyboard Part Number KFRMBA151B
- Sony Vaio VPCCW - Keyboard Part Number 148754321
- Sony Vaio VPCEA - Keyboard part number A-1765-621-A
- Sony Vaio VPCEB4 – Keyboard part number A-1766-425-A
- Lenovo ThinkPad T61 – Keyboard part number 42T3177

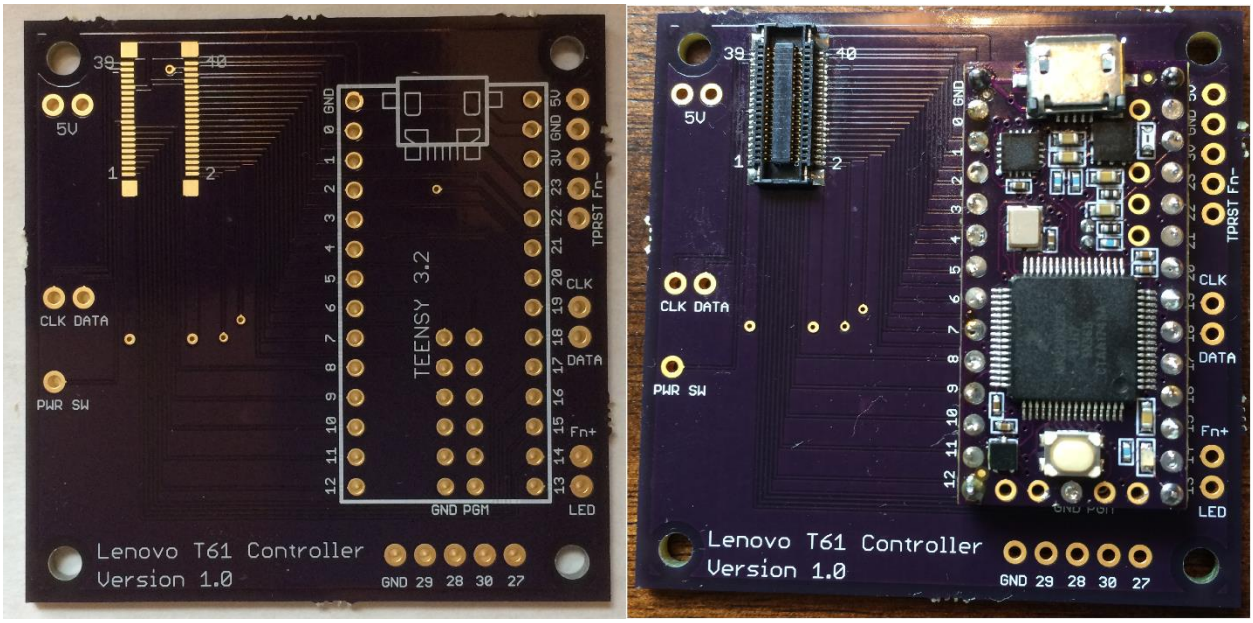
If your keyboard has a non-standard FPC cable connector like the 44 pin plug shown below, the task becomes more challenging.



If your keyboard has a non-standard FPC cable connector like the one shown above, the task becomes more challenging. If you can't find a mating connector at Aliexpress or any other site, your only alternative is to remove the connector on the laptop motherboard. A common method is to put flux and low melt solder on all the joints, then use a hot air rework station and tweezers as shown in this video. You will need to make a board layout that routes the Teensy I/O signals to your keyboard connector. I like to do a preliminary layout on paper first in order to place the parts and route the signals with the fewest via's. It's easy to assign the Teensy I/O pins in software based on whatever pin order makes the layout work best. It's tempting to make the layout next, but do the schematic first so your layout will have air-wires showing you how to route each trace. I didn't make a schematic for the keyboard scanner board because of the confusing front side LC/back side 3.2 routing. The downside of not having a schematic was the lack of any verification that the layout was electrically correct. I had to triple check everything before sending the file out for fab.

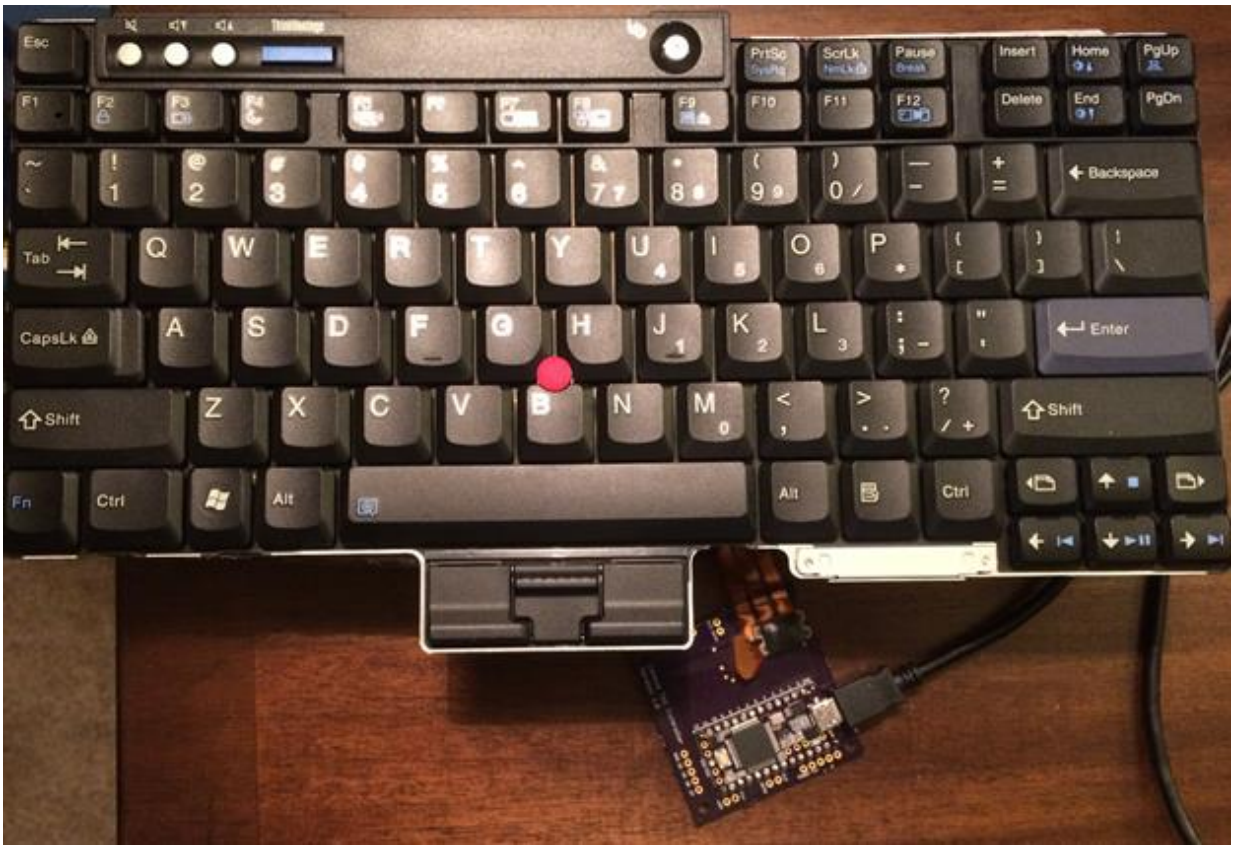
I've chosen to use Eagle but KiCad and other layout tools are available. If you want to learn Eagle, download the freeware version of Eagle software and then go thru the Sparkfun tutorials for Eagle schematic and layout. Also look at the Adafruit tutorial on creating parts in Eagle because you'll need to make a package and symbol for your connector. After you get your layout fabricated, you'll need to change the Matrix_Decoder software to work with the new I/O pin-out.

A perfect example of a non-standard FPC cable is the 44 pin connector used on the Lenovo Thinkpad T61 laptop. There are at least three web sites that detail how to make a USB controller for a Lenovo keyboard. An Instructable from rampadc uses a connector board with some glue logic and wires to an Arduino. A later Instructable from rampadc uses a single board with an MSP430 microcontroller. Mark Furland from Tome uses a connector board with wires to an Arduino. Mark's web site states that a Digikey WM6787CT-ND connector will work with the keyboard FPC cable. This saved me from having to unsolder the connector from the motherboard. It was pretty easy to search online and find a schematic for this laptop due to its popularity. Without the schematic or the info from rampadc, I would have been doing a lot of probing with an ohm meter to determine the ground pins and narrow down which pins needed to be scanned for the key matrix. I really like the feel of this keyboard which made it worth the effort to design the Teensy 3.2 circuit board shown below.



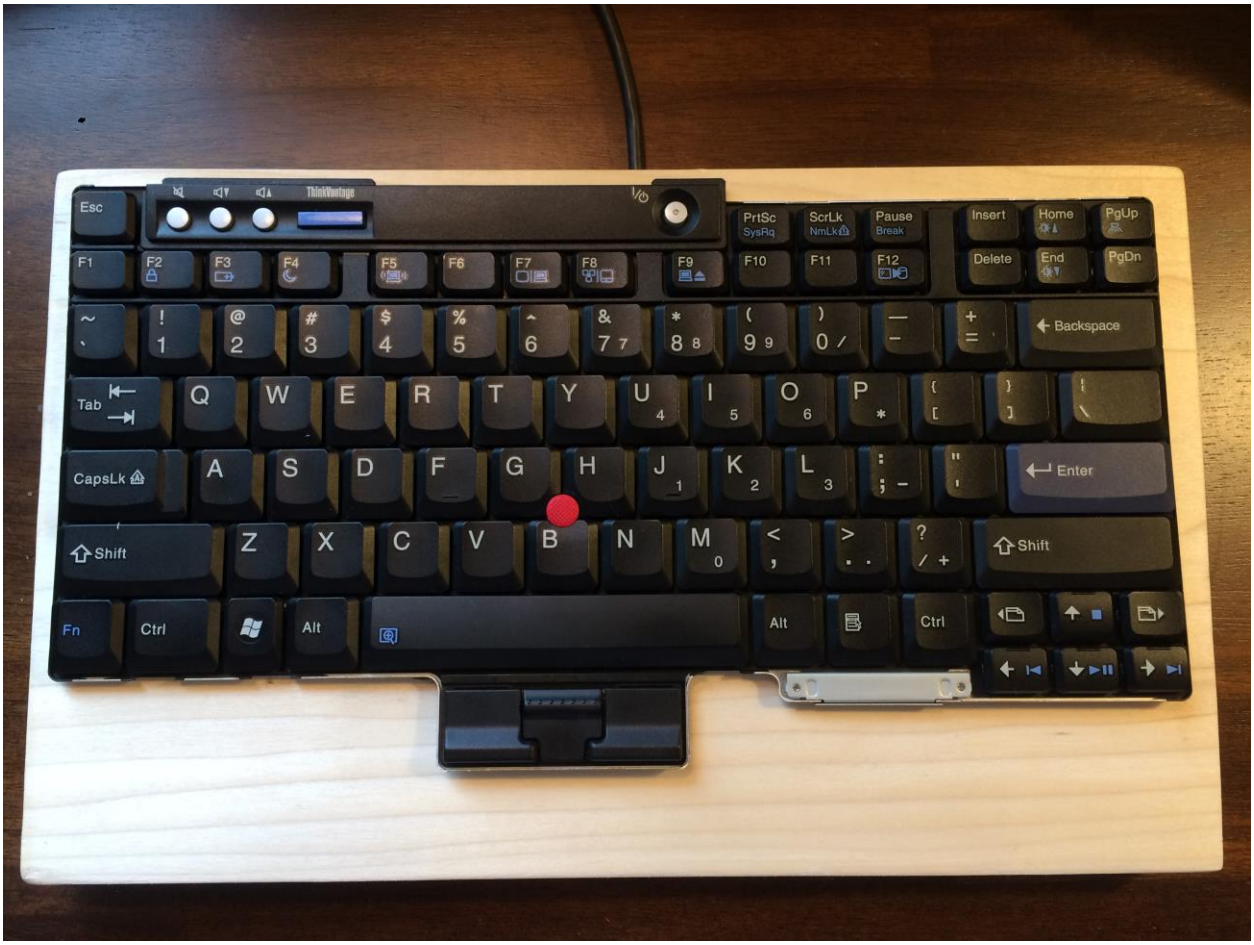
I modified the Matrix_Decoder scanning software to only scan the 8 input pins and 16 output pins from the matrix. You can use this scanner code as a guide if you have a keyboard with lots of grounds and more pins than your Teensy can handle. The scanning software produced a connection list that was turned into a key matrix table using the same steps described earlier in this Instructable. The Fn switch has its own two pins on the connector that are scanned separately from the key matrix. The Trackpoint on the keyboard needs PS/2 clock and data signals from the Teensy plus a reset signal when power is applied. The Teensy 3.2 is 5 volt tolerant so it can directly drive these signals. The T61 USB keyboard and Trackpoint routine and a detailed project description for the Teensy 3.2 controller are at my repo.

The keyboard and Teensy 3.2 USB controller card is shown below.

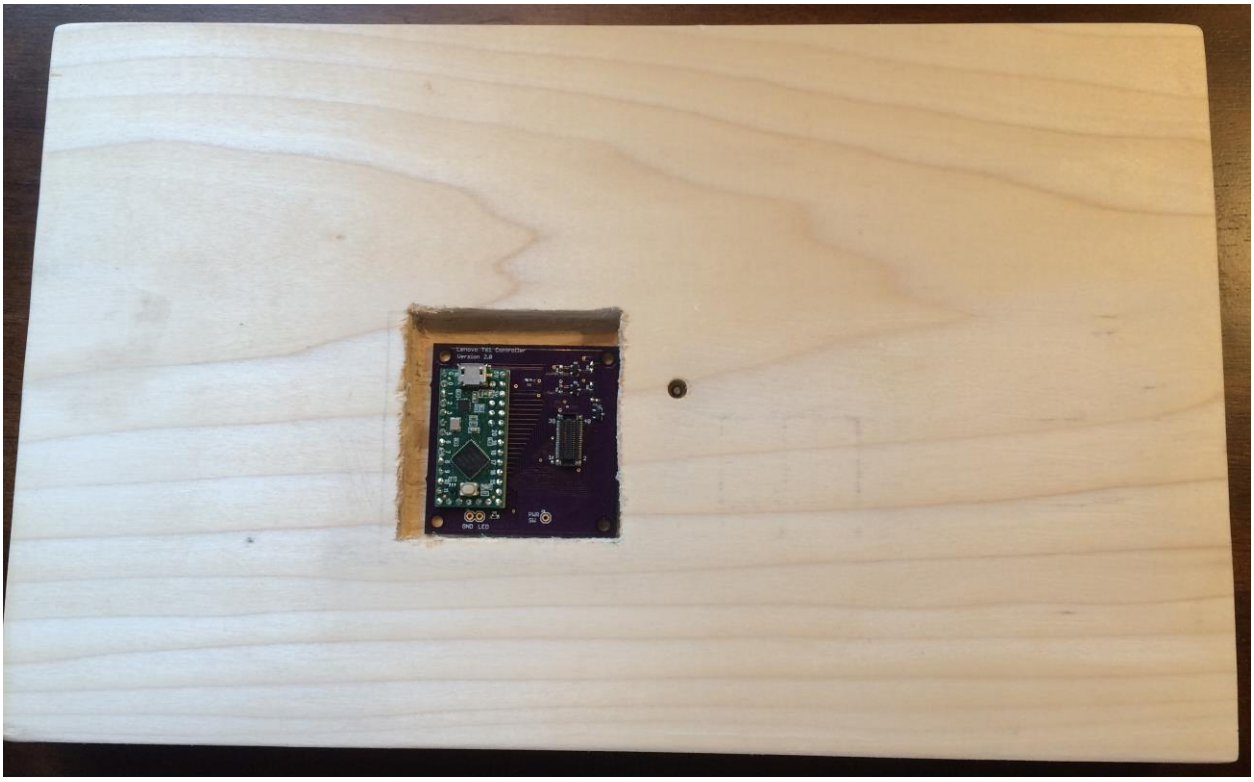


I wanted to build a standalone T61 Keyboard on a block of wood but the 3.2 circuit board needed the connector and Teensy re-positioned so the board would be hidden underneath the keyboard. I figured while I'm at it, I should change over to the LC and save some money. The Teensy LC has fewer I/O signals and they are not 5 volt tolerant so I needed to make some design changes. I adding a TLV810 to generate a reset for the trackpoint plus a couple BSS138 FETs as level translators for the trackpoint clock and data. To save an I/O pin, I wired the Fn switch into an empty cell in the matrix so it can be scanned with all the other keys. There was one Teensy I/O pin left to drive the Caps Lock LED. The T61 USB keyboard and Trackpoint routine and a detailed project description for the Teensy LC controller (shown below) are at my repo.

If you're not going to use the original laptop base for your USB keyboard, you can build a wood base like the one shown below.



I used a chisel to pound out a cavity for the Teensy LC controller board that is hidden by the keyboard.

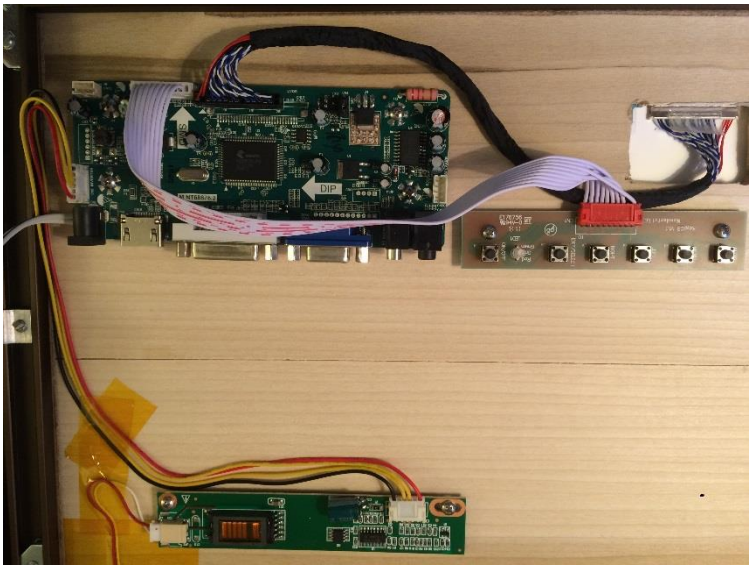


You'll have to drill a hole into the cavity for the USB cable and glue rubber feet on the backside.

If you have a wood router, you can inset the entire keyboard down into the wood.

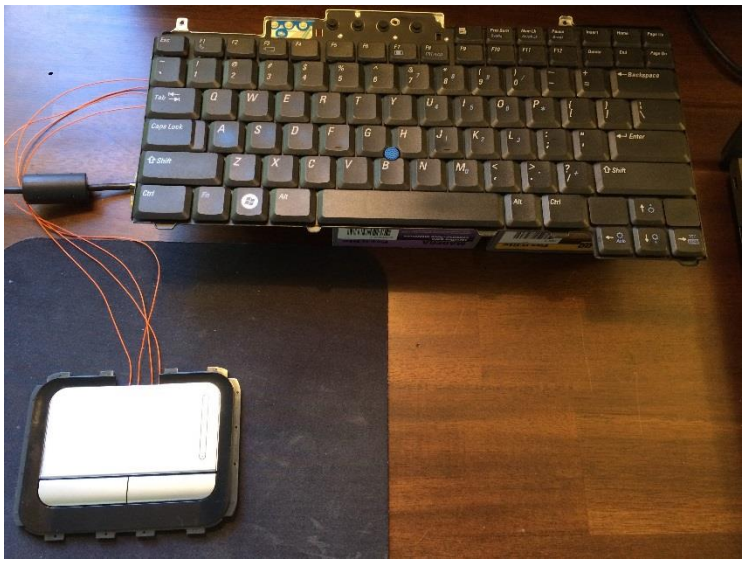
I hope this Instructable will help you find new uses for your old broken laptop. The two bins of laptop keyboards shown below are from Re-PC and about 75% have FPC cables that are compatible with the scanner board from this Instructable.

Once you have the keyboard working, you can use an M.NT68676 video converter card to drive the LCD. I'm using this converter to drive the HP DV9000 laptop LCD in a picture frame as shown below.



You can even make the touchpad work if your Teensy has two I/O pins left. The TMK code has a touchpad to USB option that can be enabled during the build process. The Teensy LC will need a level

translator but the Teensy 3.2 has 5 volt tolerant I/O and can directly drive the touchpad. If you are using Teensyduino, I have modified the touchpad code from Playground Arduino so it outputs over USB. The code and documentation are at my Github repo. I merged the touchpad code with the Dell Latitude D630 keyboard code so you have an example to follow. The Touchpad in the picture below is from an HP DV9000 and the test results are documented here.



Send me an email at thedalles77@gmail.com if you have any questions, comments, or corrections.

Good Luck

Frank Adams