Hand-Wired Keyboard Guide

By Ewen Cluney

Introduction

This is a short guide to building handwired mechanical keyboards, which I put together as part of an effort to teach myself and hopefully create a clear guide for others. Most keyboards are built around a PCB (printed circuit board) specially made for that purpose.[1] Although this makes the building process considerably easier, handwired keyboards kick the level of customization that's possible up another notch, particularly if you have access to a 3D printer or laser cutter.[2] Although it can be more time-consuming, handwiring can produce basically any style of keyboard that PCBs can, from tiny macro pads to full 100% keyboards. You may feel the limitations of a 3D printer's build plate size or the tedium of wiring over 100 diodes, but it's mostly a matter of how patient you're willing to be. Once you know what you're doing, bending and soldering diodes can become oddly relaxing.

For the purposes of this guide, I'll be focusing on using a Pro Micro with QMK firmware. It's possible to use other microcontrollers (Teensy, QT Py, etc.) and other kinds of firmware (such as ZMK, Circuit Python, etc.), but I'm doing what I know.

Additional Handwiring Guides

- https://beta.docs.qmk.fm/using-qaamk/guides/keyboard-building/hand_wire
- https://geekhack.org/index.php?topic=87689.0
- https://sachee.medium.com/building-my-first-keyboard-and-you-can-too-512c0f8a4c5f
- https://blog.roastpotatoes.co/guide/2015/11/04/how-to-handwire-a-planck/
- https://www.masterzen.fr/2018/12/16/handwired-keyboard-build-log-part-1/
- https://deskthority.net/viewtopic.php?f=7&t=6050
- https://blog.jfedor.org/2020/11/dactyl-manuform-build-log.html
- https://johannes-jansson.github.io/projects/2018/07/23/hand-wiring-lets-split.html

[1] If you can get/make a PCB design, you can get them made pretty cheaply from companies like JLCPCB (`jlcpcb.com`), provided you're okay with a turnaround time of a few weeks for manufacturing and shipping from China.
[2] 3D printers are what I'm familiar with personally, owing in large part to the fact that they're substantially cheaper than laser cutters. People have done some truly gorgeous keyboards using layers of laser-cut acrylic though.

Materials

Apart from the lack of a PCB, the materials you need are pretty similar to other mechanical keyboards. There are *some* items you can get from your average local electronics parts shop, but for the most part you'll have to order them online.

- A **Pro Micro** or other compatible microcontroller (Elite-C, BIT-C, etc.). The Pro Micro typically goes for around $7, while alternatives with USB-C are more like $20 or so. Regardless, the microcontroller will have a USB connector, and you'll need a cable to go from that to your computer. For split keyboards (p. 17) you'll need two microcontrollers. Unlike with PCB-based keyboards, for handwired ones you generally don't need any header pins.
- **Mechanical key switches**; MX switches are overwhelmingly the most common, but Alps and Choc switches also work just fine as long as the case/plate you're using can accommodate them.[3]
- **Keycaps:** You will of course need keycaps that are compatible with the type of switches you're using. It's possible to 3D print keycaps, though SLA (resin) printing is better than filament/FDM printing for that purpose.
- **1N4148 Diodes**, specifically the through-hole version. These have a glass body with a reddish-brown interior and a black stripe on one end, with leads coming out of either end. Smaller macropads may not need diodes, but usually you'll need as many diodes as there are keys. These are among the cheapest components that go into keyboards.
- **Wire:** For our purposes this should be a thin gauge of wire, around 22-26 AWG. Having the insulation made of silicon or something else that's reasonably flexible is helpful since you'll have a fair number of wires jammed in there together. You don't need anything special in the way of wire, though using a ribbon cable will let you make the wiring to the microcontroller a little neater.[4]
- Some kind of **case/plate**. On p. 4 there are some examples of 3D printable designs you can download for free, and on p. 5 there are tools you can use to generate a case. Basic PLA filament should be fine, and as long as the print doesn't have any issues, basic settings with infill at 15-20% will work just fine. You can also just get a switch tester and turn it into a functional keyboard.
- **Hardware** (screws, nuts, standoffs) as needed. Keyboards typically use M2 or M3 screws. A few designs also use heat set nuts, which you embed into the plastic by putting each one on the tip of your soldering iron and letting it melt into place.
- **Adhesive rubber feet** for the bottom of the keyboard.

[3] Other momentary switches can work as well, such as arcade-style buttons (note that arcade buttons come 24mm and 30mm sizes), foot switches, EC11 encoders, etc.
[4] I recommend using an Xacto Knife to separate the wires in ribbon cables.

Some optional but potentially helpful stuff includes:

- If the design you're working with puts exposed wires in too close proximity, you may want some **heat shrink tubing** (or **electrical tape**). Encoder switches in particular have contacts that are close together and can benefit from the extra insulation.
- **Superglue** can also come in handy if you need to attach 3D printed parts together permanently. However, superglue is surprisingly watery before it dries, and it can easily have capillary action that leaves an ugly whitish discoloration. I recommend letting the superglue dry completely, in the orientation you want for the final product, before you add any other parts in.
- **Double-sided tape** is incredibly useful for easily securing the microcontroller when the design lacks sufficient built-in mounting.
- Depending on the design, you may also need/want stabilizers, OLED displays, TRRS connectors, EC11 encoder switches, etc.

Tools

These are the major tools you'll need for a handwiring project. I won't explain soldering from the ground up, since there are plenty of tutorials on soldering online. This process doesn't require any surface mount soldering at all, so it's relatively easy as soldering projects go.

- **Soldering Iron:** You can get a basic soldering iron for $10 or so, but if you're going to do soldering with any regularity, I recommend going for a soldering station in the $50-100 range. Regardless, for this work you want a relatively thin tip for your soldering iron.
- **Solder:** Go for a thin solder without lead. Even lead-free solder still has some potentially toxic chemicals, so a fan or filter for your work area is a good idea.
- **Wire cutters**, for (of course) cutting wires and trimming diode leads.
- **Needle-nose pliers** and/or **tweezers**, for bending the diode leads. Larger pliers can be handy for removing support material from 3D prints.
- **Wire strippers**
- **Screwdrivers**, of whatever type you need for any screws and such you'll be using.

Optional items include:

- A **multimeter** can be handy for checking connections and testing switches.
- A **helping hands** (a thing with a magnifying glass and two alligator clips) is generally useful for soldering projects.

Your work area should be well-ventilated, with a surface that's high enough for you to work comfortably. Working over some kind of tray that can catch all the bits of insulation and such doesn't hurt.

Existing Case Designs

There are several 3D printable designs available on sites like Thingiverse and Github. Here are some that I recommend. Some of them lack clear build guides or even firmware, but with the tools in this book you should still be able to turn them into working keyboards. The VOID9 in particular has a detailed guide and a good design overall, and makes an excellent board to tackle for learning purposes.

- **akufu32:** A 32-key split keyboard design from Japan by NGM Design (@ngmdesign on Twitter), with a nice rounded case. There's also a 36-key variant, the akufu36. There are versions for MX and Alps switches.
- **Aplx2:** An extremely simple 2-key board intended for playing *Osu!*, though of course you can tweak the firmware to use it for whatever you want.
- **Banana:** An 8-key macropad in the shape of a banana. It is extremely silly, but it's also an easy project and quite a conversation piece.
- **Bento:** The Bento is a cute little macropad that has 5 keys and an EC11 rotary encoder switch.
- **Crocodox:** This is a compact, split keyboard, similar to a Corne. The STL files are available in a few different variations, including ones for Alps and Kailh Choc switches.
- **Dactyl:** The Dactyl is a handwired keyboard with a uniquely ergonomic design that would be difficult to implement via conventional PCB-based designs. There are several variations with different numbers of keys, and even an online tool (`dactyl.siskam.link`) to generate a Dactyl design to your specifications.
- **Etch-A Mouse:** This kind of silly device uses a pair of EC11 encoders to create a simple substitute for a mouse.
- **Lemon Keypad:** Adafruit offers a whimsical 6-key design that's easy to 3D print and assemble. It's made for either their QT Py RP2040 or Feather controllers, but would be easy to build with a Pro Micro if you prefer.
- **Matcha59:** A 3D printable ortholinear keyboard that incorporates an encoder knob.
- **Mechagodzilla:** This is a more conventional 80% keyboard, with the basic staggered QWERTY setup and a navigation cluster. If that still isn't enough keyboard for you (and somehow still isn't enough soldering), the "Motivation" is a 100% keyboard.
- **SICK-68:** As the name suggests, this is a 68-key keyboard. It has a pretty conventional staggered design, and the STL files are split for easier printing.
- **SICK-PAD:** If you want a numpad as well, this is a pretty straightforward 17-key one.

- **TouhouPad:** This 9-key board provides all the controls needed to play the Touhou series of scrolling shooters. Note that while the original is a simple handwire, the TouhouPad v2 uses a PCB.
- **VOID9:** A simple 9-key macropad that will teach you how to assemble a small matrix. The STL files include tilted and non-tilted bases and square and bezel plates, and designer Victor Lucachi has a detailed build guide.
- **VOID30/40:** If you want to get a little more ambitious without going all the way to a Dactyl, the VOID30 and 40 are ortholinear 30% and 40% keyboards. While they'll be a lot of work to put together, their ortholinear layouts make them more straightforward.
- **ZigZag32:** Also from NGM Design, this 32-key keyboard has a single body but keys in clusters like a split keyboard. It also uses Kailh Choc switches.

Generated Designs

If you prefer, you can put the design together yourself. If (like me) you're not equipped to do designs from scratch, there are some online tools you can use to create a layout and generate case designs.

The **Keyboard Layout Editor** (keyboard-layout-editor.com) lets you create a keyboard layout and then export a JSON file that you can then use in other tools.[5] For everything else in this section, this is your starting point. Its interface isn't the most efficient—you have to create keys and then change the numbers specifying their position rather than just dragging them—but it's fairly self-explanatory, assuming you know basic keyboard lingo.

The **Keyboard Firmware Builder** (kbfirmware.com) lets you take that JSON file and generate QMK firmware files and a wiring diagram. By default it throws in some stuff for macros and such, and you may want to go through the files and take out the parts you don't need.

Secret Bonus Pins

For whatever reason the firmware builder often tries to use pin B0. It along with D5 are the Secret Bonus Pins (lol). On a stock microcontroller there are built-in LEDs taking up those pins, and if you want to use them, you'll need to desolder those (extremely small) LEDs. There's a guide to doing that here: https://golem.hu/guide/pro-micro-upgrade/ If you aren't doing that, change the pin assignments to make sure they aren't using B0 or D5. You most likely won't need those pins unless you're making a unibody keyboard with more than 80 keys.

[5] If you're willing to order PCBs, there's also a program that can take a KLE JSON file and generate a PCB design. https://github.com/jeroen94704/klepcbgen

Laser-Cut Cases

The **Plate & Case Builder** (`builder.swillkb.com`) can take the JSON file from the KLE and generate SVG, DXF, and EPS files suitable for laser cutting. The tool has several settings that you should tweak and experiment with to get the results you want.

Ponoko (`ponoko.com`) is one of the most popular laser cutting services, but there are many others (you may even have one near you), and in fact the builder site can automatically send the files to Lasergist.

It's possible to take DXF files and convert them into STL files for 3D printing, though the process is finnicky and annoying. This article explains in more detail: `https://all3dp.com/2/dxf-to-stl-convert-file/`

3D Printed "PCBs" and Cases

A designer who goes by 50an6xy06r6n created a Hotswap PCB Generator, an OpenSCAD script that can produce a 3D printable hotswap "PCB," plate, and case from a JSON file. OpenSCAD (`openscad.org`) is a free 3D CAD program, and you'll need it to finish this process.

I put "PCB" in quotes because that part of the design is more of a plastic holder for diodes and wires. It's really neat, but it's not a PCB. The Github repository has a detailed build guide for these; the process is a little different from a conventional handwire. On the other hand, the cases it generates will work fine without the PCB part, and if you prefer you can just print the case and backplate and build it as you would normally.

To get the script to work you need to take a few steps:

- Download and unzip the contents of the above repository.
- Download and install node.js (`nodejs.org`).
- Open a command line (Windows) or terminal (Linux/macOS) window and navigate to the `scripts` directory.
- Enter the command "`npm install`".
- You may need to also navigate to `scripts/node_modules/@ijprest/kle-serial` and run "`npm install`" again to ensure it installs all the necessary dependencies.

Once that's done, you should be able to run the script. In the scripts directory again, run the command "`npm start -- [JSON filename]`". (It's easiest to move your KLE JSON file into the scripts directory.) Once the script runs, you can go to the scad folder to open the OpenSCAD files and tweak them to get the whole thing set up to your liking.

The files **parameters.scad** and **layout.scad** contain the settings you can modify. The readme.md file in the `scad` directory explains each setting in detail. While you can open these in OpenSCAD and make changes in the Customizer, the Customizer is a bit buggy, so it's better to just open them in a text editor. In particular, the `tent_angle_x` and `tent_angle_y` values default to 5, which adds an angle to the backplate. This creates a wedge-shaped backplate, and if you'd prefer it to be flat, you should change those values to zero. (And the aforementioned bugginess means that the Customizer just ignores you if you try to change the values that way.)

The default settings also don't create room for a Pro Micro or an opening for its USB plug. If you want to include one, you'll need to add values for `base_mcu_layout` in the layout.scad file.

pcb.scad, case.scad, plate.scad, and backplate.scad have the actual parts, which you can then export as STL files for printing. (keyboard.scad meanwhile shows you the elements of the keyboard put together, just to give you an idea what it looks like.) If you make changes to the parameters and layout and save them, OpenSCAD will update the other files automatically. Also, depending on how you're planning to put your keyboard together, you can skip printing some of the parts. The case will be able to seat the switches by itself, so you won't actually need the plate unless you're going for a sandwich case.

To export to STL, open each file in OpenSCAD, press F6 to render (or pick it from the Design menu) and then F7 to export as STL (or click File > Export > Export as STL…). These are generally simple, flat parts, and you most likely won't need a raft or any supports when printing.

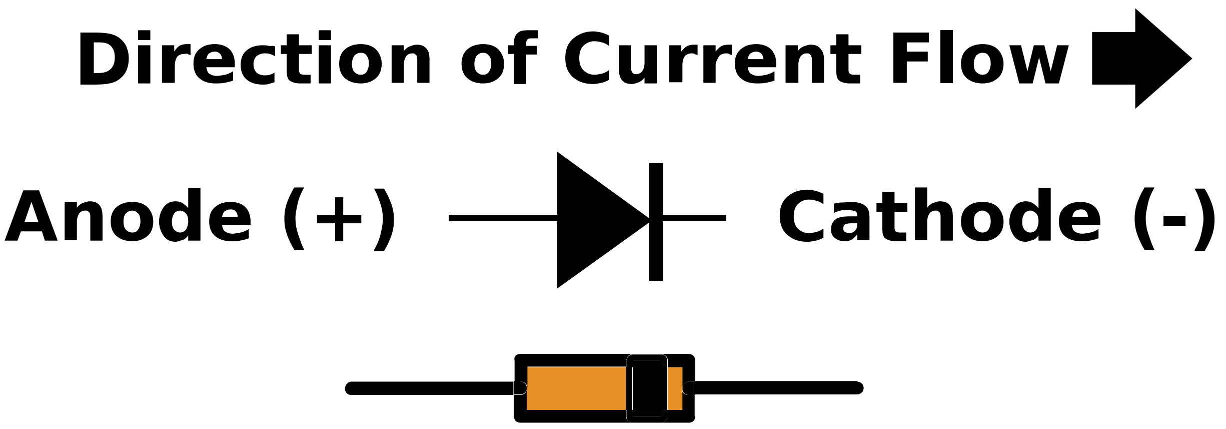
Wire the Matrix

First, take your plate/case and seat all your switches into it. While it isn't an absolute requirement, I recommend orienting them all the same way to make wiring a little easier. I've found that some designs are made for a particular orientation, and if the switches aren't seating properly, you should try rotating them 90 degrees.

A Pro Micro has 24 pins total. While for a small macropad it's possible to simply wire up each switch individually, if you get beyond 18 or so keys, you're going to have to take a more efficient approach. A matrix essentially creates a set of rows and columns, so that the controller can still figure out which key you're pressing while using fewer contacts on the microcontroller. The microcontroller then scans through the columns hundreds or thousands of times per second to pick up keypresses.

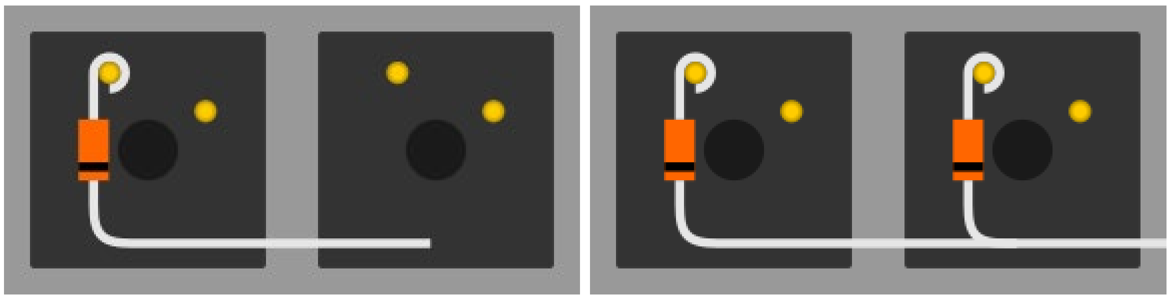
In order for a matrix to function without the current looping around, you'll need diodes. The purpose of diodes is to only allow current to flow in one direction, so it's important to make sure they're pointing the right way. Current only flows from the anode (positive) side to the cathode (negative) side, and the cathode side is the one with the black stripe.[6]

[6] Think of it as a minus sign. Also, my dad (who was a test engineer) used the mnemonic "cats are negative."



There are a few different ways to go about doing a handwire, but it’s very common to use the diode leads as part of the wiring. Think of one switch lead (left or right) as being for diodes and the other as being for wires. Think of either the rows or columns as being the diode axis and the other as the wire axis. I recommend having the diode axis be whichever is longer (which would be the rows on most keyboards), since doing longer wires for the wire axis gets to be more annoying.

You can create a loop in the anode (non-striped side) lead close to the body of the diode and then bend the other lead at a right angle.[7] The loop goes around one of the switch’s leads (trim the rest of that lead), which you then solder in place. The other lead goes to the matching lead of the next switch’s diode.



For the “wire axis,” you’re going to use just wires to create a line along the non-diode leads of each switch on each row or column.

During this process, think about the lengths and the overall volume of the wires. The wires at the ends of the rows and columns will need to be long enough to reach to the microcontroller,[8] but the overall volume of wire needs to be small enough to fit into the case in the first place. If your design needs a lot of wires, it’s generally better to use thinner and more flexible ones so you don’t have to squish them into the case so much.

[7] Diodes often come connected together with paper strips, and you can save a little time by bending the leads on the cathode side all at once.

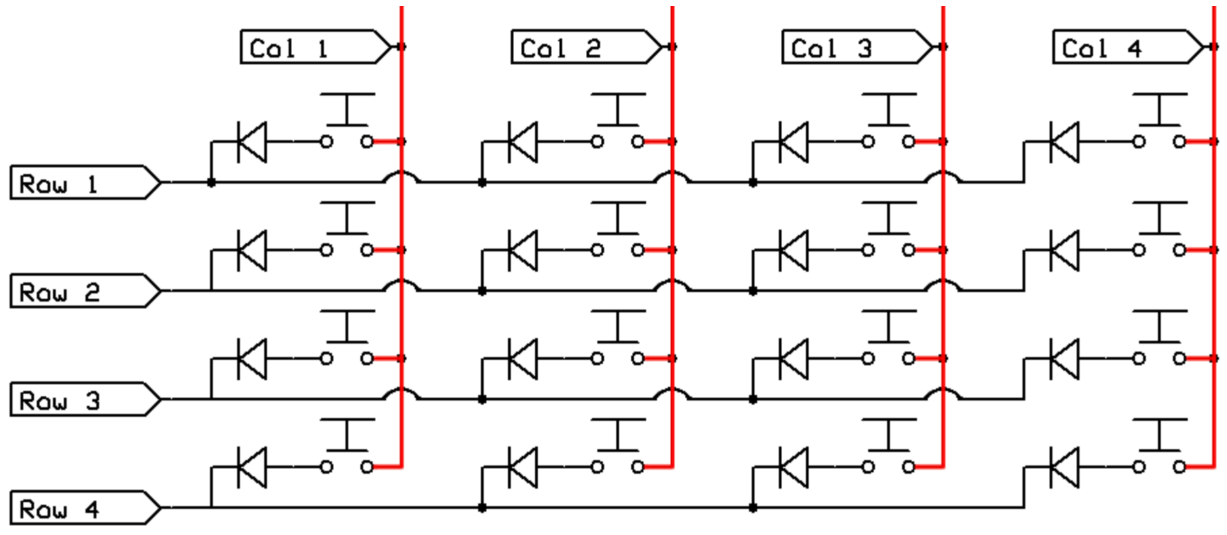
[8] If a wire is too short you can splice some additional wire to lengthen it (in which case you should add insulation to the spot where you solder the pieces together), but obviously it’s better to get the length right the first time if you can.

If you’re using stranded wire, you should twist and tin the ends before soldering. Once you’ve stripped some insulation, use your fingers to twist the strands so they stay together better, and then use your soldering iron to “soak” some solder into the exposed wire. Be careful, because this can put enough heat into the wire to make it painful to hold, especially if it’s shorter.

You can use either short pieces of wire with the ends stripped or longer pieces where you also strip them at points in the middle. [9] Short pieces are a little easier to do individually, but take significantly longer overall. For a longer piece, you want to strip a relatively long portion of the insulation at one end and then apply the strippers at points in the middle to push the insulation down and create exposed areas.[10] Any functional wire strippers will work for this, but the vise style where the mechanism also pulls the separated sections of insulation apart for you are especially helpful here. Also solid core wire makes it significantly easier to slide the insulation back and forth on the wire, which makes it substantially easier move it where it’s needed as you attach it to switch pins.

For the end of the wire that goes onto the lead of a switch or another post, I recommend having the exposed portion be a little longer, and using tweezers to make a little hook. You should be able to do this after tinning the tip of the wire, and if you can’t bend it, you probably put a little too much solder on there. For the portions in the middle, making a loop will let you secure the wire better, but either splitting strands around a pin or just laying it next to it and applying enough solder to attach it will work too.

The schematic of a 4x4 macropad (minus the microcontroller and everything it includes) would look like this:



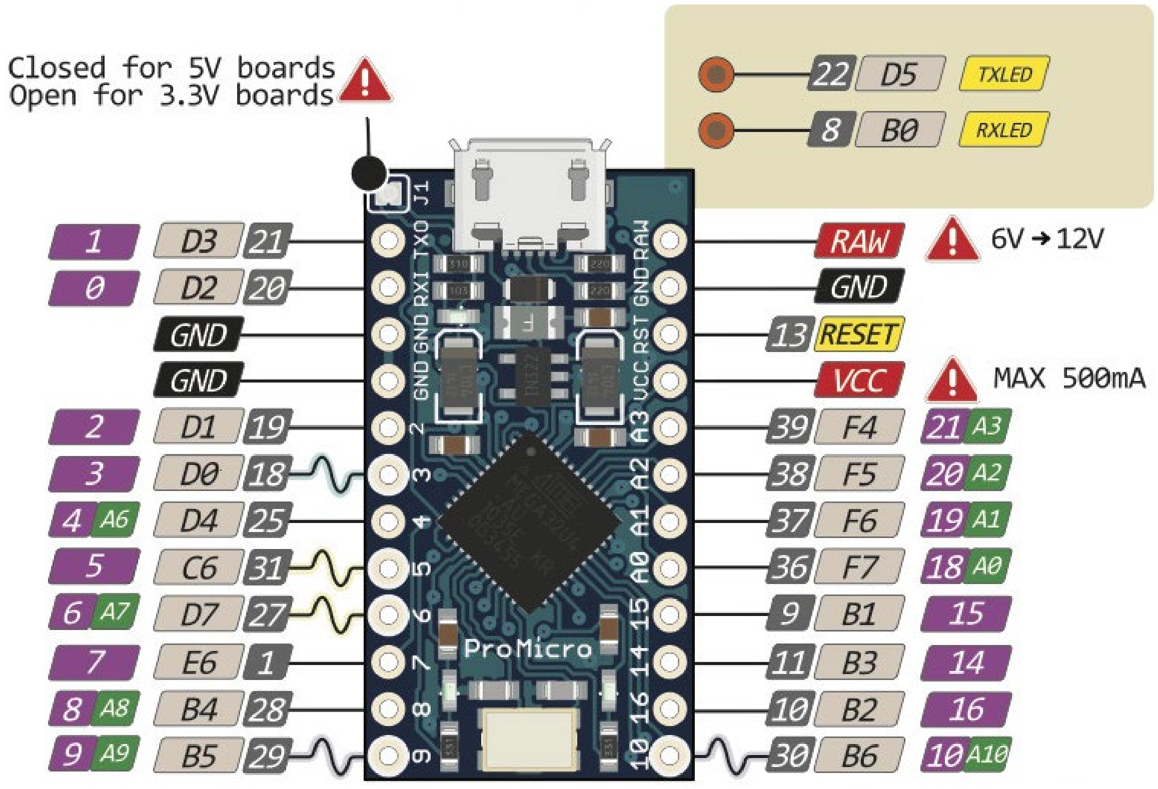
[9] It’s also possible to use wires without insulation, provided you take steps to ensure they don’t make contact where they shouldn’t, say with electrical or Kapton tape.
[10] Another option is to use an Xacto knife to carefully cut sections of insulation off.

Wire to the Microcontroller

Once you've finished wiring the rows and columns, it's time to wire them into the microcontroller. The GND, VCC, and RST pins have specific, special uses, but the firmware can read any of the remaining 18 pins.[11] If you're following an existing design that gives clear instructions, it's easier to connect them per the build guide, but if you're figuring it out for yourself, the set of pins you use is essentially arbitrary. The important thing is that the pin specifications in the firmware match.[12] If you use the firmware generator, it will spit out a list of which rows and columns go to which pins, though it does let you customize that if you prefer.

You can get more elaborate, but the simplest way to attach the rows and columns to the microcontroller is to run a wire from one end of the row/column (on the cathode end of the diode axis), put an exposed end of the wire through the appropriate hole, and solder it in place. Desoldering and resoldering a misplaced wire here is kind of annoying, so try to get the right holes.

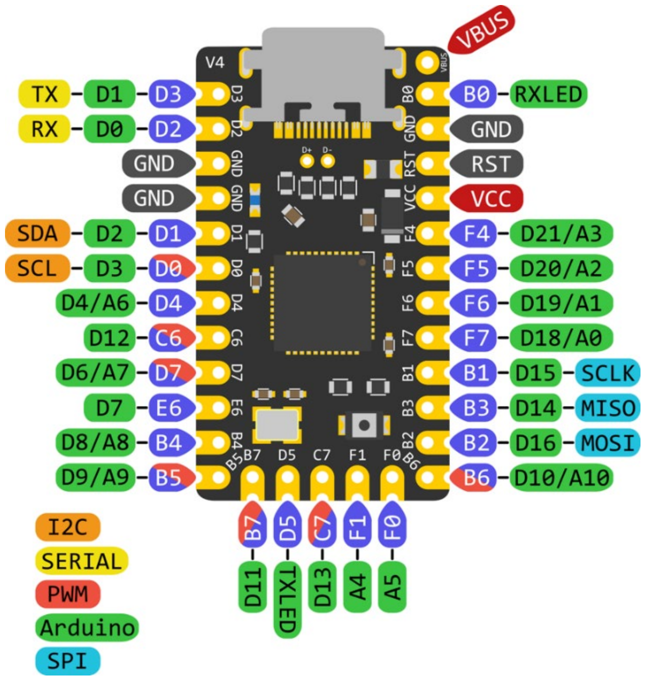
Remember that the numbering of the columns is set up assuming you're looking at the keyboard from the top, but you'll be wiring it from the bottom, so the numbers will be in descending order from your perspective.



[11] See p. 5 for info about how to use the Secret Bonus Pins.
[12] In the event that you're building an existing design that has firmware but not a clear explanation of how to attach it to the microcontroller, you can find that out from the firmware's config.h file, explained later in this book.

Get Elite

The Elite-C is a Pro Micro compatible microcontroller that costs around $20. In addition to a mid-mount USB-C port replacing the relatively fragile micro USB port of the Pro Micro, it puts turns the Secret Bonus Pins into regular pins, and adds a few more besides. (Also it has the port/bit numbers instead of the Arduino numbering, so you won’t have to convert them.) While the USB-C connector is nice to have regardless, the extra pins let you potentially have more keys or do other bells and whistles that might not be possible with a basic Pro Micro.

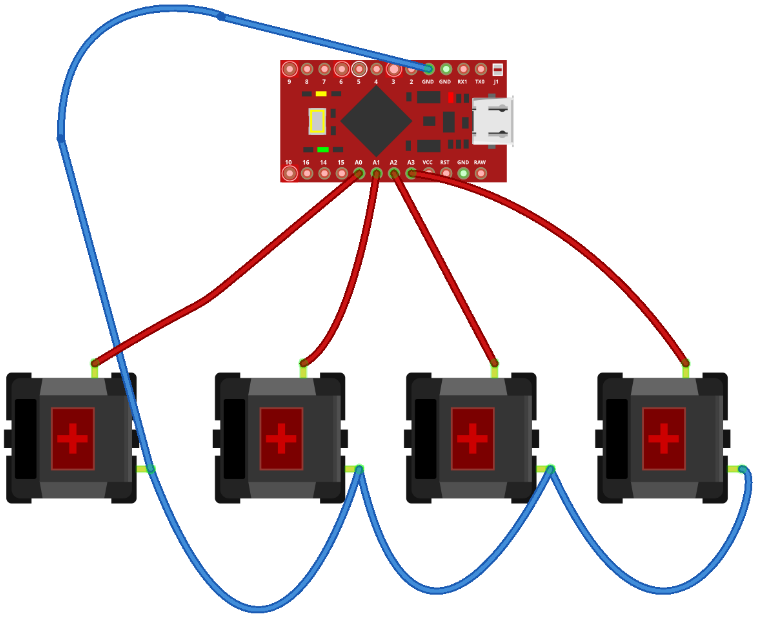


There are several other Pro Micro compatible boards, like the QMK Proton C, BIT-C, Puchi-C, and so on. They open up some interesting possibilities, though I recommend waiting until you’re fairly confident in your abilities before spending the money. Some projects aren’t going to work out, and it kinda sucks when one takes a $20 board with it.

Direct Wiring

If your design uses 18 or fewer keys, you have the option to use direct wiring instead of creating a matrix. Matrices can be a still more efficient for smaller designs (a matrix-based 3x3 macropad would only need 6 wires going to the microcontroller for example), but direct wiring is simpler and doesn't require diodes.

Direct wired designs will have one pin of each key switch connected together and then to Ground (GND) on the microcontroller, and then the other pin of each switch connected to a separate microcontroller pin. The image below shows how you might wire up a 4-key macropad.[13]



Instead of the usual matrix definition, you'll need to put `#define DIRECT_PINS` into the config.h file.

For the above macropad design, it would look something like this:

```
#define DIRECT_PINS
{
      { F7, F6, F5, F4 }
}
```

If you're not arranging them in a straight line, you can create multiple rows like this:

```
#define DIRECT_PINS
{
      { F1, F0, B0, C7 },
      { F4, F5, F6, F7 }
}
```

[13] I created it using a circuit diagramming tool called Fritzing (fritzing.org). Handwired keyboards are often simple enough that you don't really need to do that, but it's a good choice if you do need it or you want to create a really clear build guide.

Firmware

For the purposes of this guide, we're going to use QMK firmware. If you don't have it set up on your computer already, you should follow the setup instructions on the QMK site. Basically, in Windows you should install the special QMK build of the MSYS terminal, while for macOS and Linux you'll install and use it in your OS's terminal.

Established Options

If you're using a design that already has firmware in the QMK Github repository, you can use the QMK Configurator to tweak and compile the firmware from a web browser. The firmware will also be in the `keyboards` folder of your QMK installation, and you can modify and compile it from there. If the design has firmware but isn't in the repository, you can download the files and put them in the `keyboards` folder of your local installation.

Something New

If you use the firmware generator, you can just have it compile the .hex file for you and use the QMK Toolbox to flash it to the microcontroller and be done with it. It can also produce a zip file of source code, which you can then customize to your liking, tweaking settings that the generator doesn't cover.

While it's possible to assemble your firmware files by hand, I recommend using either the generator or an existing keyboard's firmware profile[14] as a starting point. If you prefer, QMK has a way to create a set of files for a default firmware setup in the command line, explained here.

Assigning Pins

In the firmware's config.h file, you'll define the pins for the rows and columns using these functions:

```
#define MATRIX_ROWS 5

#define MATRIX_COLS 15

#define MATRIX_ROW_PINS { D0, D5, B5, B6 }

#define MATRIX_COL_PINS { F1, F0, B0, C7, F4, F5, F6, F7, D4, D6, B4, D7 }
```

MATRIX_ROWS and MATRIX_COLS simply tell the firmware how many positions there are in each, while MATRIX_ROW_PINS and MATRIX_COL_PINS define the specific pins each uses, in order. The labels for the pins that you use for firmware purposes are based on the pins of the ATMEGA32U4 chip, which are different from the Arduino numbers that are actually printed on a Pro Micro, so here's a handy conversion chart:

[14] I use a Gergoplex as my daily driver, and I duplicated its layout on a Microdox. When I started working on a 36-key Dactyl variant, I used my customized Microdox keymap as a starting point, which made the process surprisingly quick and easy. Unless you have a really novel layout, chances are there's already something close in QMK's keyboards directory already.

Pro Micro	ATMEGA32U4	Pro Micro	ATMEGA32U4
TX0	D3	RAW	
RX1	D2	GND	
GND		RST	
GND		VCC	
2	D1	A3	F4
3	D0	A2	F5
4	D4	A1	F6
5	C6	A0	F7
6	D7	15	B1
7	E6	14	B3
8	B4	16	B2
9	B5	10	B6

It will also need `#define DIODE_DIRECTION` followed by either `COL2ROW` or `ROW2COL`. `COL2ROW` means the stripes on your diodes are facing the rows, and the rows are the diode axis.

```
#define DIRECT_PINS { { F1, F0, B0, C7 }, { F4, F5, F6, F7 } }
```

In the firmware’s [keyboardname].h file, there will be a definition of a C macro that looks something like this:

```
#define LAYOUT( \
    k00, k01, k02, k03, k04, k05, k06, k07, k08, k09, \
    k10, k11, k12, k13, k14, k15, k16, k17, k18, k19, \
    k20, k21, k22, k23, k24, k25, k26, k27, k28, k29 \
) { \
    { k00, k01, k02, k03, k04, k05, k06, k07, k08, k09 }, \
    { k10, k11, k12, k13, k14, k15, k16, k17, k18, k19 }, \
    { k20, k21, k22, k23, k24, k25, k26, k27, k28, k29 }, \
}
```

If your matrix doesn’t have keys in every location, you can put KC_NO in the blanks to make things a little easier, like this:

```
#define LAYOUT( \
    k00, k01, k02, k03, \
    k10, k11, k12, k13, \
    k20, k21, k22, \
    k30, k31, k32, k33, \
    k40,      k42 \
) { \
    { k00, k01, k02, k03, }, \
    { k10, k11, k12, k13, }, \
    { k20, k21, k22, KC_NO, }, \
    { k30, k31, k32, k33, }, \
    { k40, KC_NO, k42, KC_NO } \
}
```

In the actual keymaps you'll then reference the LAYOUT macro and populate it with keycodes, like this:

const uint16_t PROGMEM keymaps[][MATRIX_ROWS][MATRIX_COLS] = {

```
[0] = LAYOUT(
  KC_NLCK, KC_PSLS, KC_PAST, KC_PMNS, \
  KC_P7,   KC_P8,   KC_P9,   KC_PPLS, \
  KC_P4,   KC_P5,   KC_P6, \
  KC_P1,   KC_P2,   KC_P3,   KC_PENT, \
  KC_P0,            KC_PDOT)
}
```

Compiling

To compile the firmware, put it in its own subfolder in the `keyboards` folder of your QMK installation, then open your terminal (or MSYS if you're on Windows) and navigate to the `qmk_firmware` folder (usually by typing `cd qmk_firmware`). Type "`make <keyboardname>:<keymapname>`"[15] and press Enter, and if everything goes well, it will compile and produce a .hex file in the QMK directory. If it has errors, you'll have to go back and figure out where the problem is, and I've found that googling the error will usually turn up something useful. Sometimes it's something complex and obscure, but a lot of the time it's just a stray comma.

Flashing the Firmware

Once you have the firmware compiled, you can flash it to the microcontroller using either the QMK toolbox or in the command line. For a basic Pro Micro you can ready it for flashing by bridging the GND and RST contacts, but note that some other microcontrollers ignore this convention and will just flash automatically when you tell your computer to flash an microcontroller.

- In the QMK Toolbox, use the Open command to load your .hex file. You can then either pick the Auto-Flash option to have it flash the first readied microcontroller it detects or get it ready and click the Flash button.
- With the command line, use the `make` command like you did when compiling, but add `:flash` onto the end.

[15] If the firmware is in a subfolder of the keyboards folder, you'll have to include that as well, with a forward slash. For example, if you wanted to compile default Dactyl firmware, you'd have to type `make handwired/dactyl:default`.

Test the Keyboard

Plug your new keyboard in and see if it works right. `keyboardtester.com` provides a simple but effective way to go through and ensure that each key is working properly. You may need to touch up your soldering or tweak your firmware to get it working properly.

If there are problems, it’s likely someone made a mistake somewhere. Don’t get discouraged! It takes some practice to get good at this, and even then, mistakes will still slip through now and then.

What you get when you plug the keyboard in and press the keys can give you some idea what’s going on.

- If a single key isn’t responding at all, check that (1) the diode is oriented correctly and (2) the wires are in fact properly soldered to it. You can also use a multimeter or continuity tester to see if the switch itself is defective or damaged, but that’s rare.
- If an entire row/column isn’t responding at all, there’s probably an issue with the wire going from that row/column to the microcontroller.
- If on a direct-wire keyboard a portion of the keys don’t work, there may be an issue with the soldering of the ground pins.
- If all the keys from two rows or columns are reversed, the wires for those rows/columns are probably reversed as well.
- If a special feature isn’t working, make sure you have it enabled in the rules.mk file.

Finishing Touches

Finally, put whatever finishing touches you want to complete the project.

- Make refinements to your firmware if desired.
- Fit the parts of the case together with screws and other hardware as needed.[16]
- Attach your keycaps to the switches.
- Put rubber feet on the bottom of the case.
- Add cool stickers.
- Post pics on social media.

[16] I originally bought a pack of heat set nuts because I was building a particular keyboard whose design called for them, but I’ve found they’re generally handy for finishing up 3D printed keyboard cases. Designs don’t always have good screw holes, and when they do, your 3D printer may not print them as well as would be ideal. To install these, I recommend changing to a blunter soldering iron tip (so it doesn’t poke through the bottom of the nut and melt extra plastic), turning the soldering iron’s heat setting down fairly low, and having tweezers or needle nose pliers handy in case the nut gets stuck on the soldering iron tip.

Cool Options

This final section is a grab bag of features that you can add to a keyboard and how to implement them in a handwired QMK-powered keyboard.

Reset Switch

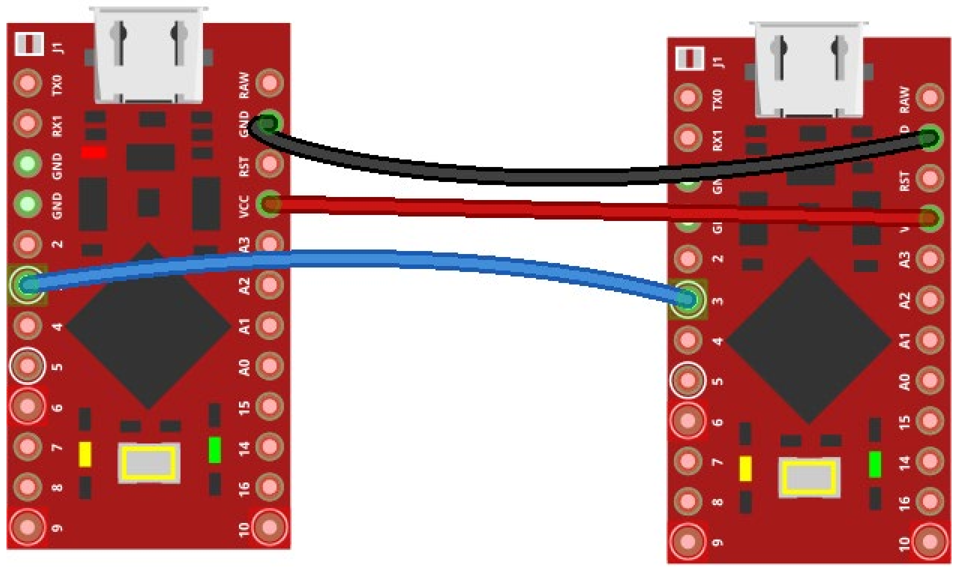
If you expect to be flashing the firmware more than once in a blue moon, you may want to add a dedicated reset switch. For this you just need to add a momentary switch attached to the GND and RST contacts on the microcontroller, preferably one you can stick somewhere where you won't press it accidentally.

Split Keyboards

A "split" keyboard is divided into two distinct parts, which can be better ergonomically and is generally nifty. With QMK the standard way to make a keyboard split it to have a microcontroller in each half and connect them with a TRRS cable. TRRS is like a 1/8" headphone jack with an extra conductor.[17] This is a short guide that omits several options; see the QMK documentation page for full details: https://beta.docs.qmk.fm/using-qmk/hardware-features/feature_split_keyboard

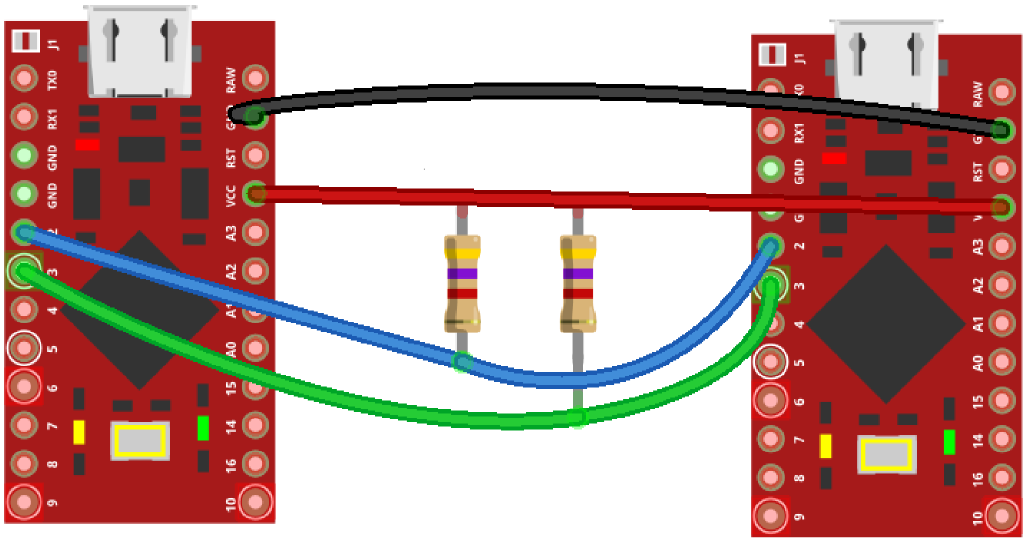
QMK supports both serial and I2C methods of connecting the halves, the biggest difference for our purposes being that serial needs 3 conductors while I2C needs 4. Regardless of what kind of connector you use, note that you shouldn't plug/unplug them while the keyboard is plugged into your computer.

For a **serial** connection, the three wires need to go between GND, VCC, and D0/D1/D2/D3 (whichever you choose to specify with the `SOFT_SERIAL_PIN` value). For example, if you opt for D0, it would look like this:



[17] TRRS means "Tip Ring Ring Sleeve," where a normal headphone jack is a "TRS." Other cables with the right number of conductors will work fine though.

For an I2C connection, they need to connect the GND, VCC, 3, and 2 pins. Each of the data pins needs to have a resistor between it and the VCC line on one side. The resistors should be between 2.2k and 10k Ohms, with 4.7k being preferred.



For the firmware:

- The rules.mk file needs to include `SPLIT_KEYBOARD = yes`
- You will need to specify which half is the "master," the one you plug the USB cable into. There are several methods for this (described in detail here), but the simplest is to include `#define MASTER_RIGHT` or `#define MASTER_LEFT` in the rules.mk file.
- In the config.h file, specify which communication method you're using, either `#define USE_I2C` or `#define SOFT_SERIAL_PIN D0` (or whichever valid pin you intend to use).
- For the hardware configuration, you'll need to specify separate pins, like this (and again with `LEFT` for the left half).
    - `#define MATRIX_ROW_PINS_RIGHT { <row pins> }`
    - `#define MATRIX_COL_PINS_RIGHT { <col pins> }`
- In the rare case where you're doing direct wiring on a split keyboard, you'll instead use:
    - `#define DIRECT_PINS_RIGHT { <pins> }`
- If you mirror the matrices between the two halves, you'll need to include `#define FLIP_HALF` in config.h.

Hotswap!

Normally when you build a keyboard, you solder directly to the pins on the key switches, meaning that changing out switches requires desoldering them. With a PCB that's annoying, and with a handwired board it's kind of nightmarish. Raymond Yang comes to the rescue with a 3D printed hotswap socket, which creates contacts when you place diodes and (solid core) wire into it. While this isn't *necessary* to build a handwired keyboard by any means, it can make a really nice addition and generally make changing switches vastly easier. You can find the STL file for these on Thingiverse:

https://www.thingiverse.com/thing:4791318

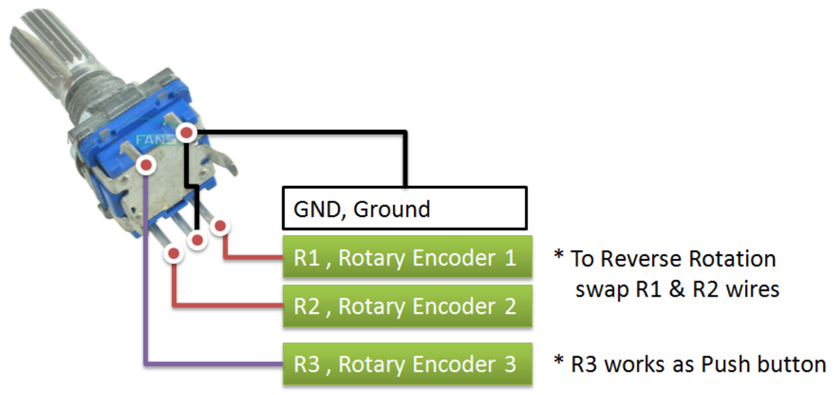
Rotary Encoder Switches

A fair number of hobbyist mechanical keyboards use EC11 rotary encoder switches to add a knob. These can register turning left or right, plus being pushed down,[18] and you can assign whatever keystrokes you want to those three.[19] They're often sold with knobs you can easily attach, though the shaft on the EC11 also fits guitar knobs.[20] There are also plenty of 3D printable knob designs out there.

Generally the keyboard case needs to be designed to fit an encoder switch (as is the case with the Bento), but Kevin Eckert created a 3D printable adapter that lets you put one in a mounting for an MX key switch. Regardless, the switches usually come with a hex nut, which is invaluable for holding it in place.

The QMK site explains the various options (and how to implement multiple encoder switches) in the Encoders documentation, but this is a quick overview.

To set up the firmware, first you'll need to include `ENCODER_ENABLE = yes` in the rules.mk file.



[18] Note that the actuation force for pushing down is about 500gf, roughly 10 times that of a typical key switch, so you shouldn't assign it to anything you intend to use frequently.
[19] The push function will typically be set up as a key switch, but QMK handles the left and right turns differently, making it annoying to try to do anything differently with them on different layers.
[20] Though note that guitar knobs are typically larger, and not all keyboard designs have enough clearance for them to fit right.

The side of the encoder with two pins is for the push function, and keyboards usually integrate it into the keyboard layout, treating the GND and R3 pins like the pins of a key switch.

The side with 3 pins is for the rotation functions, and you set them up a little differently. The R1 and R2 pins on the switch will need to connect to data pins on the microcontroller, and the config.h file will need those pins defined, like this:

```
#define ENCODERS_PAD_A { D1 }
#define ENCODERS_PAD_B { D0 }
```

In this case you'd be attaching R1 to D1 (2) and R2 to D0 (3), though if you get them reversed you can just add `#define ENCODER_DIRECTION_FLIP` to rules.mk.

To define what an encoder does in the keymap.c file, you'll put code like this somewhere after the keymap definitions:

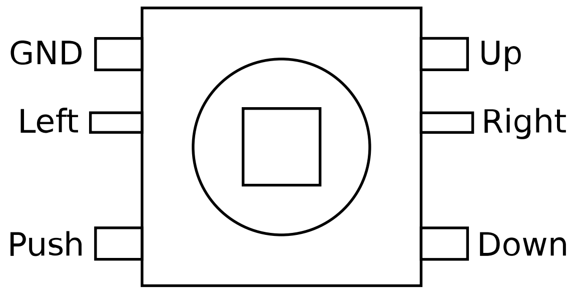
```
void encoder_update_user(uint8_t index, bool clockwise) {
    if (index == 0) {
        if (clockwise) {
            tap_code(KC_VOLU);
        } else {
            tap_code(KC_VOLD);
        }
    }
}
```

If you have multiple encoders, each will have an `index` value counting up from 0. And of course you can change `KC_VOLU and KC_VOLD` to whatever keycodes you want it to produce.

5-Way Switches

A newer and pretty nifty thing is the "5-way navigation switch," a little thing that has up/down/left/right and push functionality with roughly the footprint of an MX switch. You can get them from Adafruit (with a rubber cover available) and some other places, and they're pretty easy to wire up in place of more conventional key switches.

To my irritation I couldn't find a proper reference for what pins on these switches are what, so I put together the diagram you see below. The alignment of the switch is ultimately arbitrary, and this diagram assumes that the larger gap between the outer and middle pins goes towards the bottom, putting the Ground pin in the upper right.

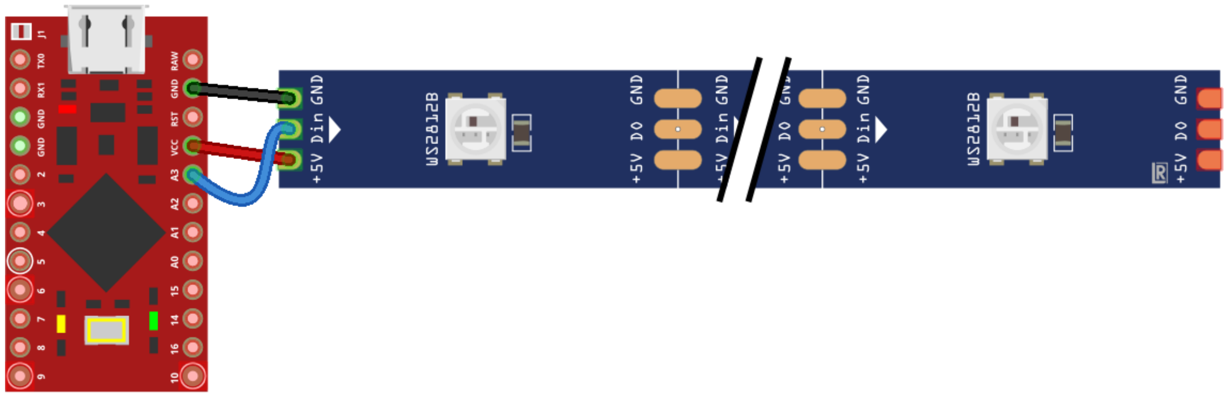


Kevin Eckert also made a 3D printable adapter, so you can slot one in place of an MX switch. It includes a design for a cover in the shape of an SA keycap.

Underglow

There are a few different ways to add lighting to a keyboard. “Underglow,” also known as “RBG lighting,” puts lights underneath the keyboard, and it’s one of the easier kinds of lighting you can do. One of the most popular ways to implement underglow is with a WS2812B LED strip, which is a series of SK6812 RGB LEDs on a flexible strip with contacts and resistors included.

The LED strip has three pins, labeled +5V, DIN, and GND. GND goes to the microcontroller’s GND, +5V goes to VCC, and DIN goes to a pin you define in the firmware. Note that the LED strip has DIN on one end and DO (Data Out) on the other, and it won't work if you solder it to DO instead of DIN.



- In rules.mk include `RGBLIGHT_ENABLE = yes`
- In config.h:
  - Define the pin that DIN connects to with `RGB_DI_PIN`. Unlike other pin assignments in config.h, this does not go in curly brackets.
  - Tell the firmware how many LEDs are attached with `RGBLED_NUM`
- In the keymap, you may want to add keycodes for controlling the lighting.

Weird Typing

I wanted to take the opportunity to mention that there are some nifty things you can do to type in exceptionally weird, interesting ways.

- **Artsey.io** is a system that lets you produce most keystrokes one-handed with a set of eight keys. It’s open source, and while it’s a little technical to set up, you can add it to most any QMK- or ZMK-compatible keyboard that has a 2x4 set of keys.
- **Steno** is a sophisticated chording system that uses 23 keys to produce whole words and even phrases at once. It’s what most court reporters use to transcribe speech in real time, and thanks to the Open Steno Project, you can now get the software to do it for free. You don’t have to spend years to get up to 225 WPM to benefit from it, and thanks to the Open Steno Project you don’t have to spend thousands of dollars to get the necessary hardware and software. QMK includes support for a couple of the standard steno protocols, making it pretty easy to implement steno keycodes the firmware.