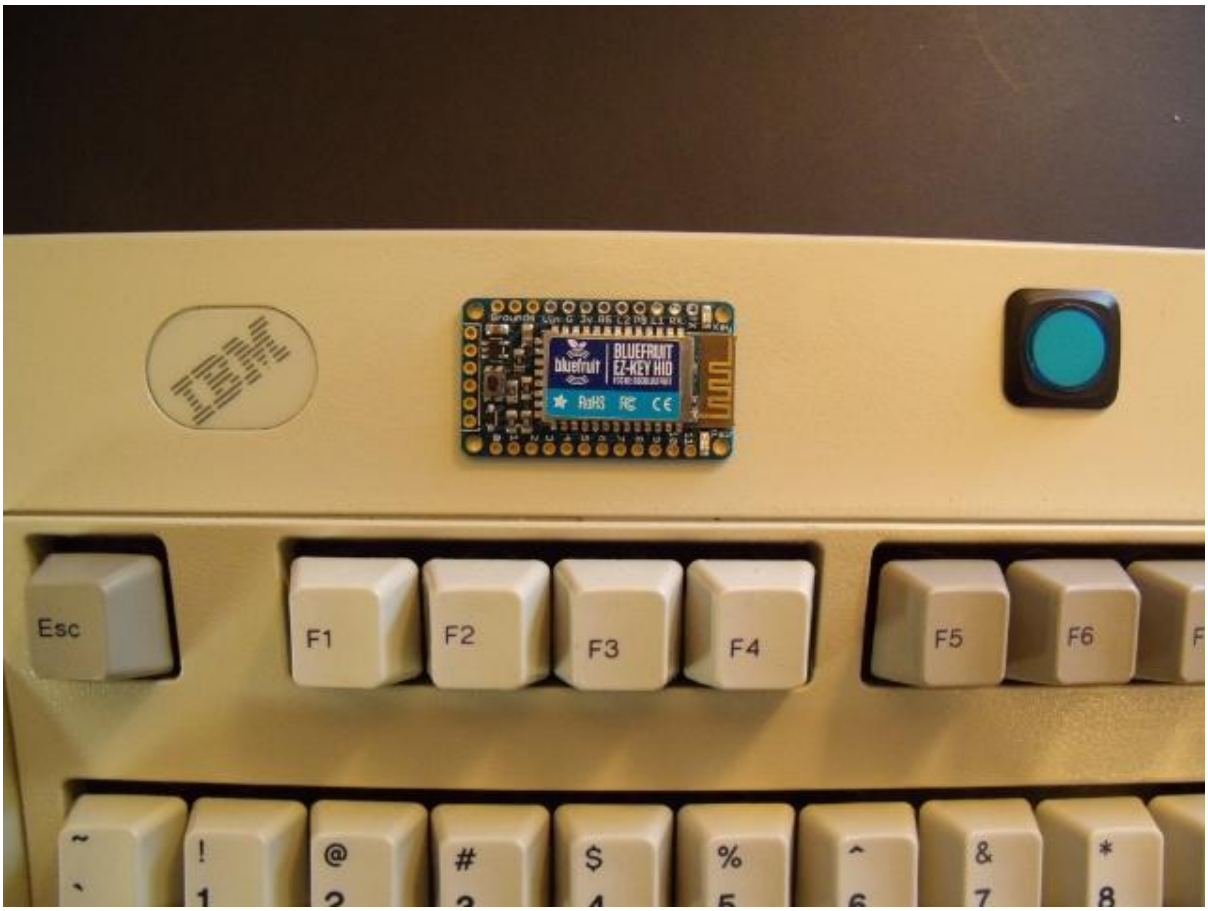
Convert your Model M Keyboard to Bluetooth with Bluefruit EZ-Key HID

Created by Benjamin Gould



https://learn.adafruit.com/convert-your-model-m-keyboard-to-bluetooth-with-bluefruit-ez-key-hid

Last updated on 2023-08-29 02:27:59 PM EDT

Table of Contents

Overview 3

Research 4

- History
- Variants
- Connectors
- Resources

Tools 9

Parts 12

Prepare the Keyboard 16

- Get your hands on a Model M

How to Clean your Model M Keyboard 18

Model M Keyboard Disassembly 22

Code 25

- Getting Started

Configuring the Firmware 27

Integrating the Firmware 29

Defining the Keymaps 31

Interacting with Bluefruit 33

Circuit 37

Assembly 42

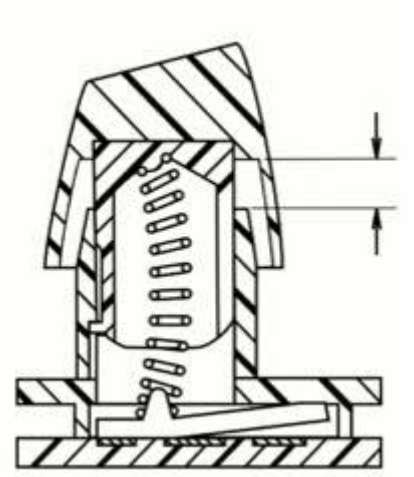
Use It! 50

- Pairing the Keyboard
- Using the Keyboard
- Conclusion

Overview

IBM's venerable Model M keyboard is a true classic in the world of computing. Created in an era when personal computers were regarded as big ticket items and their manufacture was a highly competitive industry, it is not surprising that keyboards from the 1980's were the beneficiaries of extensive engineering and high quality materials, especially since at the time they were the only input devices for many systems (except maybe a joystick once in a while).

The Model M keyboard boasts a legendary "buckling spring" type of key switch that was developed by IBM to emulate the experience of typing on a typewriter, which many of their potential users might find familiar and approachable. The buckling spring mechanism provides excellent audible and tactile feedback, and many users claim that they help improve your typing because they provide two forms of feedback at the exact moment of switch actuation. Keyboards today, if a computer even comes with one at all, are undoubtedly geared toward low cost and have almost no distinguishing features, so its no surprise that the Model M and other keyboards from that era are still popular among discerning users.



When I first saw the Bluefruit EZ-key HID module, I knew that I wanted to make a wireless Model M keyboard. There are plenty of examples open source keyboard controllers and converters, and since Bluefruit supports sending up to six keys at once based on input from the serial connection, I thought this should be possible. Keep reading to see the results of my experiment...

Research

History

Every Model M keyboard uses the PS/2 keyboard protocol - the Model M is in fact the original PS/2 keyboard. Even though they all use the same protocol, there are different variants, and in order to properly interface with a Model M, it is germane to understand their history.

From computer-engineering.org ():

IBM introduced a new keyboard with each of its major desktop computer models. The original IBM PC, and later the IBM XT, used what we call the "XT keyboard." These are obsolete and differ significantly from modern keyboards; the XT keyboard is not covered in this article. Next came the IBM AT system and later the IBM PS/2. They introduced the keyboards we use today, and are the topic of this article. AT keyboards and PS/2 keyboards were very similar devices, but the PS/2 device used a smaller connector and supported a few additional features. Nonetheless, it remained backward compatible with AT systems and few of the additional features ever caught on (since software also wanted to remain backward compatible.) Below is a summary of IBM's three major keyboards.

IBM PC/XT Keyboard (1981):

- 83 keys
- 5-pin DIN connector
- Simple uni-directional serial protocol
- Uses what we now refer to as scan code set 1
- No host-to-keyboard commands

IBM AT Keyboard (1984) - Not backward compatible with XT systems.

- 84 -101 keys
- 5-pin DIN connector
- Bi-directional serial protocol
- Uses what we now refer to as scan code set 2
- Eight host-to-keyboard commands

IBM PS/2 Keyboard (1987) - Compatible with AT systems, not compatible with XT systems.

- 84 - 101 keys
- 6-pin mini-DIN connector
- Bi-direction serial protocol
- Offers optional scan code set 3
- 17 host-to-keyboard commands

Variants

The very first Model M's were of the AT variety; they had 101 keys and a detachable cable with a 5 pin DIN connector. The layout of this keyboard is the basis for what most people would consider to be a "standard" layout for a full size keyboard. Built for use with the IBM PC/AT (model 5170) system, the matrix scan codes ("scan code set 2") as well as the connector were compatible with the 84-key AT Model F keyboard. When IBM released its line of PS/2 computers, the Model M was updated to feature a new connector (6 pin mini-DIN) and included support for an alternate set of scan codes ("scan code set 3"), although the default was to utilize the same set of scan codes as the PC/AT.

Scan code set 3 was developed for use with IBM terminals and emulators; Model M keyboards that utilize this code set by default have either 122 or 101 keys, and are generally incompatible with Windows computers. I suspect that if you had an adapter to be able to plug one in to a PS/2 port, it may work on more robust operating system like Mac OS X or Linux. These keyboards are relatively easy to interface with due to

the internal connector which is comprised of 3x2 header pins with .1" spacing.

Every Model M has a "birth certificate" on the back that tells its date of manufacture, as well as the model and part number. Wikipedia has a useful guide of Model M part numbers () if you're not sure what you're looking looking at.

Digging around I discovered that basically all Model M keyboards support scan code set 3, although most (for the most part any of them with an AT or PS/2 connector) utilize code set 2 by default. The PS/2 protocol has a host-to-keyboard command for requesting a specific code set. In order to support all Model M's, upon initialization our code can request code set 3; terminal keyboards will ignore this command (they always use code set 3) while all other Model M's will honor it and we can assume that code set 3 is in use. Therefore we don't have to worry about separate key maps for the different scan codes.

Not all PS/2 keyboards will honor this host-to-keyboard command to switch codesets! In fact, I suspect that most do not. In that case this converter won't work, but it shouldn't be hard for an intrepid maker to make some small changes to the code to get it working for codeset 2!

Connectors

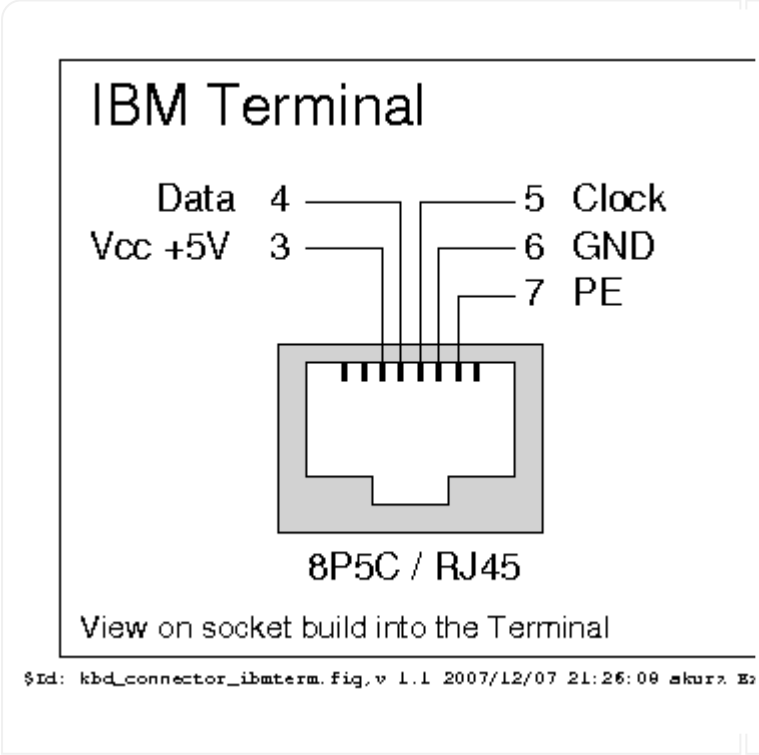
Since our Bluetooth keyboard is going to be wireless, we don't really care about the connectors, right? That's partially true... in our case it doesn't matter if the connector is a 6 pin mini-DIN (PS2), a 5 pin DIN (AT) or RJ45. We do however need to know how to connect our converter to the existing controller, which we can fairly easily ascertain by examining how the existing cable/connector is attached to the controller board.

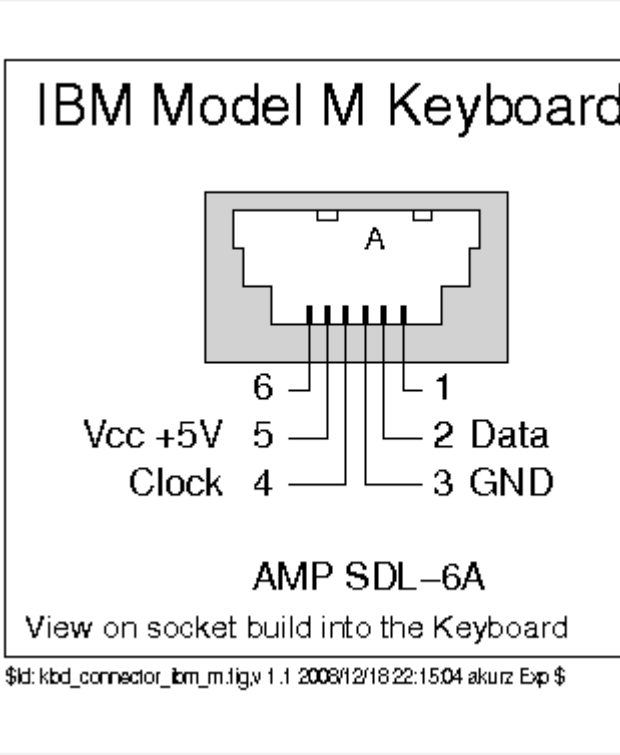
A PS/2 keyboard interface has 4 lines:

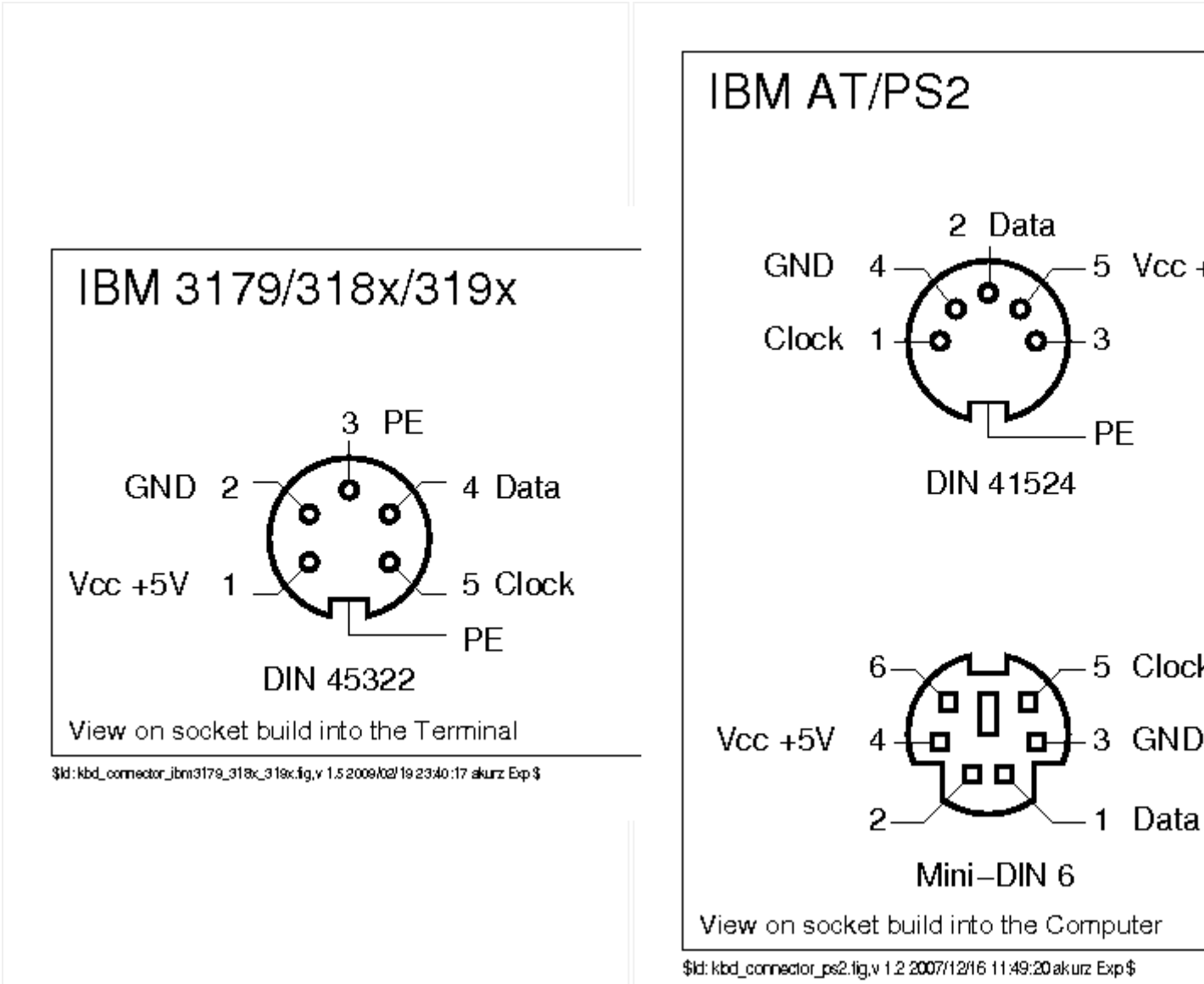
1. +5V
2. Clock - this line is used by the keyboard to signal the timing of data being transmitted. The host relies on this signal both to send and receive data, and can inhibit the keyboard by pulling this line low
3. Data - This line is used for bi-directional data transmission. Depending on the direction of communication, reading the state of this line on either the rising or falling edge of the clock signal provides one bit of data being transmitted.
4. GND

We will need to connect to all four of these lines; our battery circuit will provide the input voltage and a connection to ground. Also, we will use 2 digital pins on the Arduino to bitbang the Clock and Data lines according to the protocol.

Fortunately, all of the Model M connectors you might come across are well known. A quick jaunt over to the "Connectors" page at kbdbabel.org () yields the following diagrams:





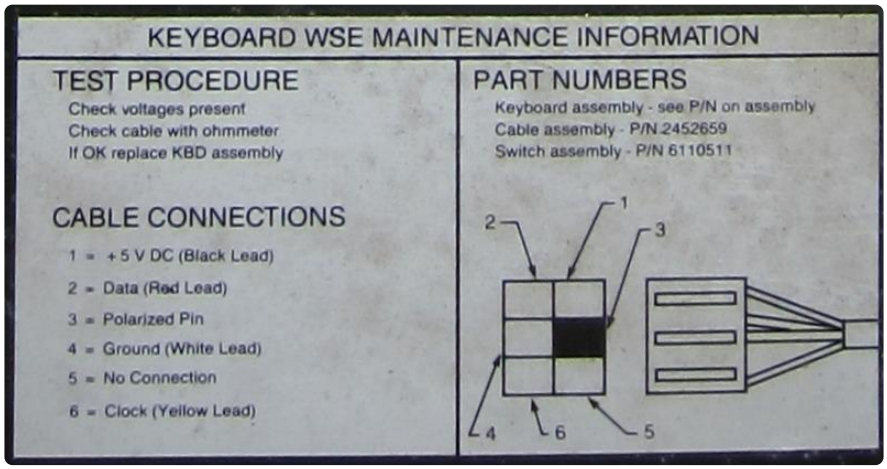


All of these are connectors were actually used for different variants the Model M keyboard! You can compare the diagrams to the connector on any Model M you have to understand what each pin on does. If necessary use your multimeter to test continuity and determine where each pin connects to the controller.

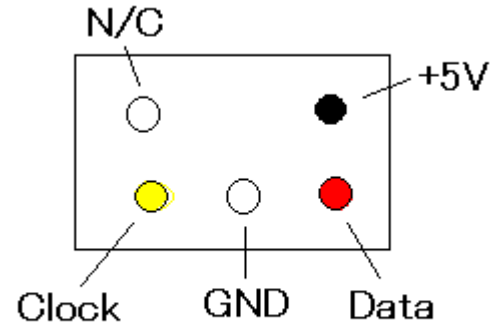
The connector in the upper right is a female "SDL" connector and is the six-pin connector that a Model M's detachable cable plugs into, if it has one. If you have a keyboard with this connector, you may have desolder it first in order to complete this project; although some boards have extra solder points for these lines that you can utilize (YMMV).

Since I'm using a terminal Model M in this tutorial, and they all have a 3x2 male header with .1" spacing, we can use our female jumper wires of the same spacing to easily hook up to our circuit without soldering to the controller. Here is a pinout from a label inside a 122-key variant of the Model M, but it is valid for the 101-key version as

well (thanks Soarer!):



Here is another illustration of the same information (source ()):



Facing to a connector on the PCB

Resources

- IBM 1390876 Keyboard () - nice explanation of the different scan code sets for IBM keyboards
- Wikipedia, IBM Model M Keyboard Features by Part Number () - a nice compilation of a large number of Model M part numbers along with their distinguishing characteristics (click the "show" link to display the table)
- Trivia, Model M & F () - Tons of great info with lots of pictures of different variants of Model M and Model F keyboards

---

Tools

You will need some basic electronics tools in order to complete this project.

If don't already have these things, you might want to take a quick trip over to the Adafruit Store and pick up Ladyada's Electronics Toolkit (http://adafru.it/136) which will

get you going with just about all the tools that you'll need. You'll still need to get your hands on a 7/32 hex socket though.

If you already have tools, go ahead and make sure you have the following available:

7/32 Hex Socket

The shell of the Model M is held together using 4 7/32"
hex bolts. You should either use a screwdriver with a hex driver head attached, or a ratchet with an elongated socket that can reach a few centimeters deep. If you only have metric sockets 6mm may work but be careful not to round over the corners of the hex.

Soldering iron

Any entry level 'all-in-one' soldering iron that you might find at your local hardware store should work. As with most things in life, you get what you pay for.

Upgrading to a higher end soldering iron setup, like the Hakko FX-888 that we stock in our store (http://adafru.it/180), will make soldering fun and easy.

Click here to buy our entry level adjustable 30W 110V soldering iron (http://adafru.it/180).

Click here to upgrade to a Genuine Hakko FX-888 adjustable temperature soldering iron. (http://adafru.it/303)

Flush Diagonal Cutters

You will need flush diagonal cutters to trim the wires and leads off of components once you have soldered them in place.

Click here to buy our favorite cutters (http://adafru.it/152).

Multimeter

For this project you might not absolutely NEED a multimeter, but it will be very useful to check continuity as you assemble the parts. Even the most basic multimeter can check continuity, so I recommend picking one up if you don't have one already.

Click here to buy a basic multimeter. (http://adafru.it/71)

Click here to buy a top of the line multimeter. (http://adafru.it/308)

Click here to buy a pocket multimeter. (http://adafru.it/850)

Solder Sucker

Strangely enough, that's the technical term for this desoldering vacuum tool. Useful in cleaning up mistakes, every electrical engineer has one of these on their desk.

Click here to buy a one (http://adafru.it/148).

Solid Core Hook-up Wire

If you don't already have a plentiful stock of hook-up wire on your bench, you might want to pick some up. This set has a nice variety of colors with a convenient set up for dispensing lengths of wire.

Click here to purchase this from the Adafruit store (http://adafru.it/266)

Parts

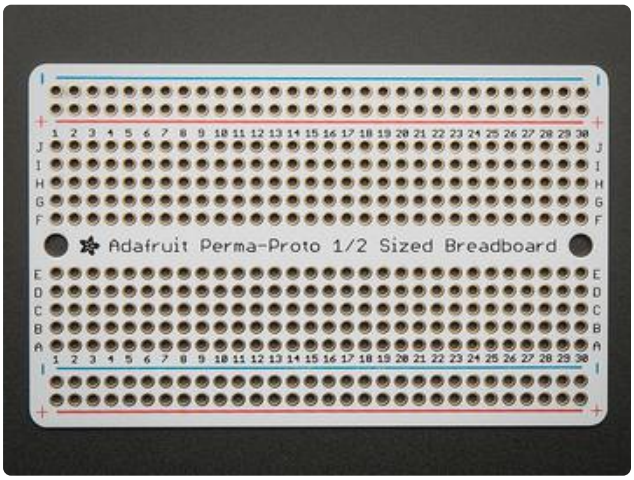
Model M Keyboard

First and foremost, you will need a Model M keyboard of course! For this project I prefer a "terminal" version of the Model M - these were typically produced for use with "green screens" and usually have an attached cable with a RJ45 style connector (it looks like a typical LAN network cable connector). These come in 101 key and 122 key versions, either one will work fine. Part #1392595 is a common part number for a 101-key terminal Model M.

I'll only be focusing on the terminal Model M keyboard for this tutorial - they are generally cheaper and easier to work with - but in case you'd like to try another one I did verify that the code works ok with a "normal" PS/2 Model M as well.

These are readily available on eBay if you are looking to purchase one (try searching for "model m keyboard rj45" or "model m terminal keyboard").

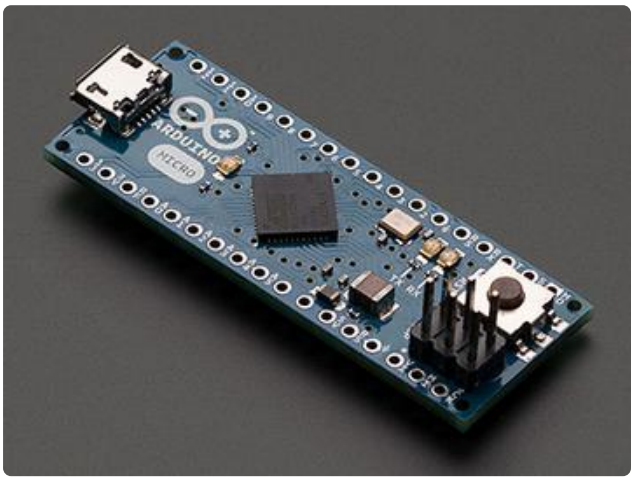
Perma-Proto Half-sized PCB



I'm going to mount the Arduino Micro and the Bluefruit used in this project onto a Perma-Proto PCB to make sure that everything stays in place and all of our connections remain sound inside of the keyboard. This is optional, you could definitely wire all of the connections point-to-point if you prefer. As it happens, there are standoffs injection molded into the shell of the Model M keyboard that are the perfect size for this cute little PCB.

Click here to purchase this from the Adafruit store (http://adafru.it/571)

Arduino Micro

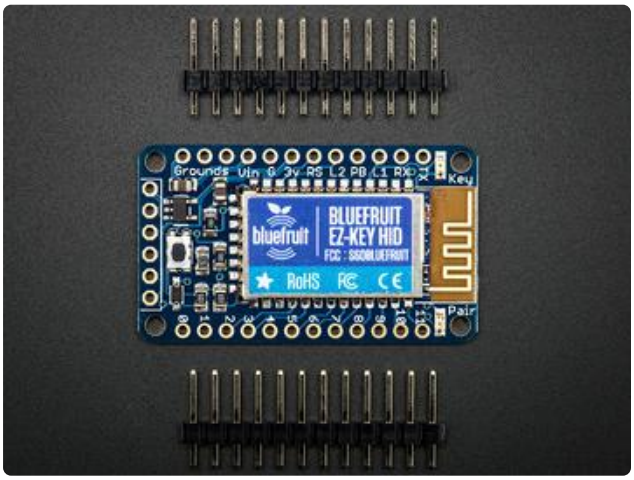


The brains of this project will be an Arduino Micro. In the Adafruit store there is one that has headers already soldered on, and one that does not come with headers at all. If, like me, you are going to mount your components onto a Perma-Proto PCB pick up the version with headers, or make sure you have some strips of header that you can solder on yourself. If you will directly wire your components together then grab the version without headers.

Click here to purchase WITH headers (http://adafru.it/1086)

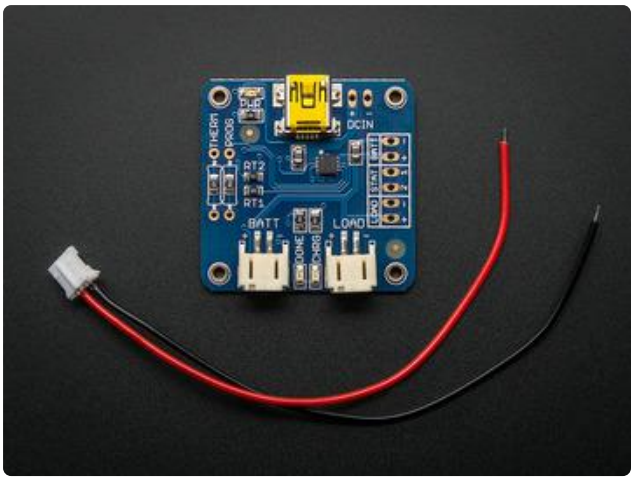
Click here to purchase WITHOUT headers (http://adafru.it/1315)

Bluefruit EZ-Key HID



Bluetooth capabilities will be handled by the Bluefruit EZ-Key HID module. Since version 1.1 this module can accept raw HID reports as input over its serial connection, which allows for the fine-grained representation of the keyboard's state that is necessary for this project. Bluefruit 1.0 will not work, and Bluefruit 1.2 is required if you would like to implement "consumer" keys (such as volume and media player controls).

Click here to purchase this from the Adafruit store (http://adafru.it/1535)



USB LiIon/LiPoly charger - v1.2

This USB charger will suit our purpose well; it is a good size, has a default charge rate of 500mA (same as most USB ports on your computer) and has an onboard JST connector and matching cable included that will come in handy.

Click here to purchase this from the Adafruit store (http://adafru.it/259)

Lithium Ion Polymer Battery - 3.7v 2500mAh

I've chosen a 2500mAh battery for this project, but you could substitute a smaller battery if you'd like as well. However the Model M has plenty of space inside to fit a decently sized battery, and the 30 year old electronics draw a respectable amount of current so the extra capacity is nice to have.

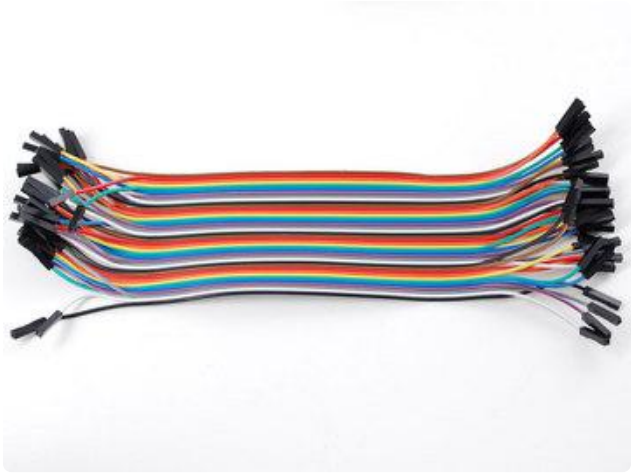
I've found that this size battery allows for approximately 24 hours of operation on a full charge.

Click here to purchase this from the Adafruit store (http://adafru.it/328)

16mm Illuminated Pushbutton - Blue Latching On/Off Switch

This nifty switch is perfect as a power switch for our project; the LED is driven independently from the latching toggle switch, so we can use the switch as a power button and the LED can be driven by the state pairing LED on the Bluefruit module.

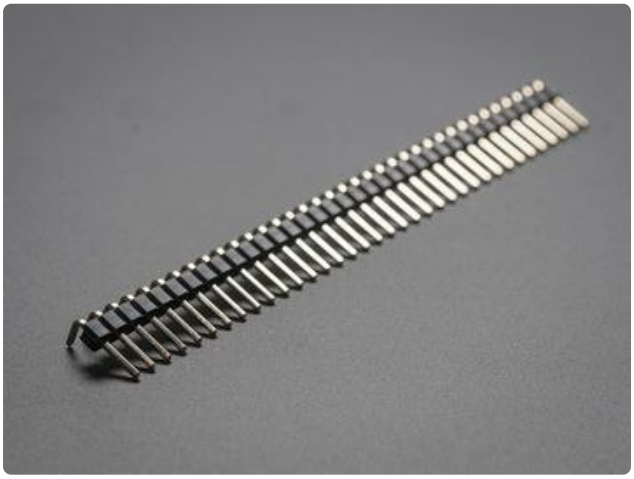
Click here to purchase this from the Adafruit store (http://adafru.it/1476)



Premium Female/Female Jumper Wires

These .1" jumper wires are perfect for connecting our circuit to the existing electronics inside the terminal Model M keyboard. The PS/2 connections that we will be utilize are conveniently available on male header pins with this spacing, so with these we won't have to solder directly to the original parts. You will only need 4 of the individual wires.

Click here to purchase this from the Adafruit store (http://adafru.it/266)



Break-away 0.1" right-angle male header

Space is kind of tight inside the keyboard, and the strips of straight header stand up just a little too tall for our purpose. You could just bend some straight header yourself if you'd like, but everything seems to stay together a lot nicer when you have the right part for the job.

Click here to purchase this from the Adafruit store (http://adafru.it/1540)

Prepare the Keyboard

Before getting started, you will need to make sure that your keyboard is prepped to be modified. There are many different varieties of the Model M, so first you will want to make sure that you have one that you will be able to work with easily. Next you will want to get everything cleaned up properly and then disassemble the case so that you can install the new hardware inside.

Get your hands on a Model M

Not many pieces of computer hardware are still in high demand after 30 years, but the Model M is a notable exception when it comes to this. The clicky tactile switch that they are known for is still highly sought after, and in general even used Model M's in somewhat rough condition can fetch a good price. You do have options though:

You can purchase a new Model M keyboard - IBM sold its keyboard manufacturing factory in the US to Lexmark in the mid-nineties, and later Lexmark sold the patents and tooling to some of the employees who formed a new company, Unicomp (). Today you can purchase a new "Model M" from Unicomp, as well as replacement parts for your old ones if necessary. The model that most approximates the example shown in this tutorial is the Classic 101 Buckling Spring PS/2 (). I'm not sure what the controller looks like inside of these or how easily it will be to wire up the PS/2 terminals to the Arduino, so caveat emptor.

You can go dumpster diving - The Model M was a very prolific piece of equipment, so if you are adept at pulling electronics back from the abyss you might be able to find one. Businesses upgrading there systems or schools/universities that are cleaning out their closets may be good options. Also, try checking Craigslist for free or inexpensive listings of old IBM PC systems, or check your local thrift stores.

You can purchase one in used condition - I would definitely recommend going this route for this project, as you will be able to save some money versus a new or refurbished keyboard, and also you can make sure that you get one that is easy to work with. If you are buying a keyboard off of eBay, you might want to verify the condition and make sure that the pictures they provide are of the actual keyboard you are getting. I'll be working with the rather rough looking specimen show below, which is definitely in need of some TLC.

Since this keyboard is so dirty I will take the opportunity to give you some tips on cleaning up a Model M in case you find one that is in rough shape as well. They actually clean up quite nicely - in the next section I'll show you how to get it done.

How to Clean your Model M Keyboard

If you buy a used Model M keyboard off of ebay or from an electronics recycler, or even if you just pull one out of a corner in your garage after it has been forgotten and unused for a few years, you'll probably find that it could benefit from a good cleaning before being put back into use. The materials used in the construction of the Model M were quite durable and the overall design was quite good, so most of the time a thorough cleaning is all that is needed to make one serviceable again.

Following are the steps that you can take without disassembling the keyboard. We will be opening up the shell of the keyboard anyhow for our project, but its a good idea to clean what we can first so that debris does not make its way into parts of the keyboard that previous were protected. These steps are also good to perform every so often to take of any grime from normal use.

Start by removing the key covers. Most Model Ms have two-part key caps that consist of a cover and a stem. The cover pops off pretty easily just by pulling on it with your fingers. If yours has two-part keycaps, remove just the covers and place them in a bowl or jar for cleaning.

Some have one-piece integrated key caps - mostly later models, models that were branded for other companies (such as Lexmark and Dell), and the "quiet touch" (rubber dome) models. If your Model M has one-piece keys, you might find it easier to remove them by prying them off as shown in the next step.

No matter what model you are working with, the keys that are 2x width or greater (left shift, right shift, backspace, etc) are one-piece keys that are easiest to remove if you pry them off as in the picture. Remove these keys and add them to the jar with the key covers for cleaning.

The spacebar is pried off just like the 2X keys, the only difference is that there is a metal stabilizer bar. Just pop off the stabilizer and put the spacebar in the cleaning jar.

Once all of the covers and large keys have been removed, proceed to remove the remaining key stems, but don't add them to the cleaning jar unless they are particularly dirty. It is very easy for water to find a home inside of the key stems and you have to be careful to make sure they are completely dry if you soak them with the rest of the keys. The stems don't usually get dirty enough to make it worth the extra effort, so keeping them separate can save you some time.

At this point all of the keys should be removed from your keyboard. The springs should not fall out even if you turn the keyboard upside down. Try not to let anything fall into the "barrels" as any debris in there could start to cause problems after repeated actuations.

If you look closely at the photo, the spring that was under the "grave" key (`) is caught on the barrel. That's not uncommon, so take a moment to look for anything like that and straighten it out.

This particular keyboard has a lot of crud on the barrel frame so I'm going to go over it quickly with a vacuum before I do anything else.

Fill the cleaning jar with warm water and add a small amount of liquid laundry detergent - the cheapest, mildest stuff you have is probably best. You can use the use the spacebar to gently stir and agitate the keys until the detergent is evenly distributed. Let it soak for about 30 minutes, and give a quick stir every 10 minutes or so. The detergent will do its work and after 30 minutes the keys will be surprisingly clean.

The Model M's keys are made of durable PBT plastic and the legends are applied via dye sublimation, so cleaning them like this should not cause any damage or fading.

Remove the keys from the wash, giving each one a quick wipe with a clean, slightly wet rag to remove any residue. Give the key a quick rinse in some clean water and let it sit out to air dry. The integrated keys with the small crevices can take 24 hours or more to completely air dry. It is important that the keys be completely dry before putting them back on the keyboard.

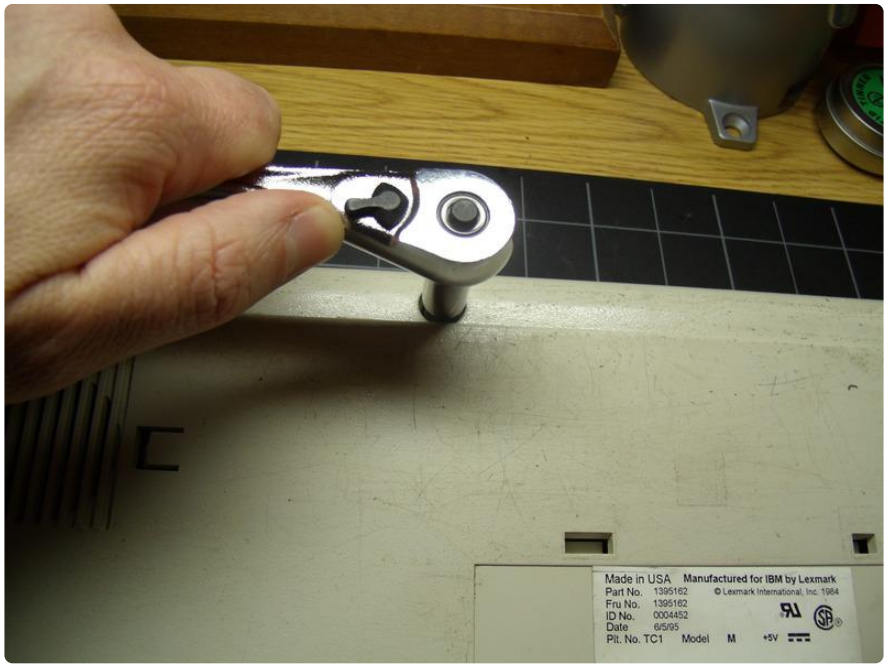
Pro tip: Apparently the previous owner didn't like the Page Up key or something, and put an X on it with permanent marker that didn't come off in the wash. I was able to remove it by applying some alcohol-based hand sanitizer and letting it soak for a while, followed by light scrubbing.

At this point, if I wasn't planning on disassembling the board I would just clean the exterior of the shell and the barrel frame, put the keys back on, and be finished. You can clean the exterior with mild soap and water, and stubborn grime can usually be removed with rubbing alcohol or hand sanitizer gel if necessary. To clean the barrel frame without disassembling the board, cotton swabs dipped in rubbing alcohol work well.

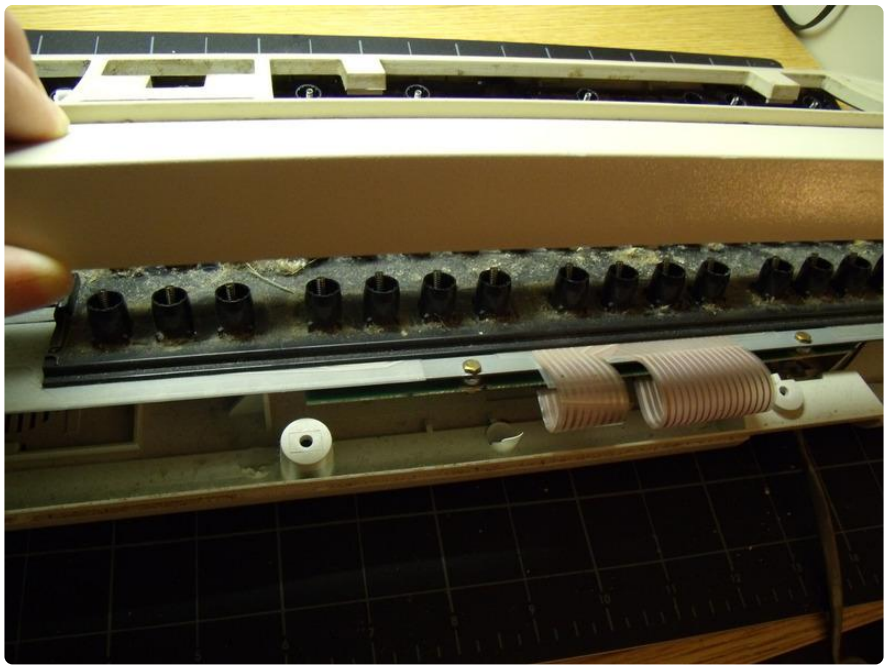
In order to remove the keyboard's shell, continue to the next page about disassembly of the Model M. It is a bit easier clean the shell and the barrel frame when the keyboard is taken apart, so if you are going to take it apart anyhow you might as well wait until it is in pieces before you do that.

Model M Keyboard Disassembly

In order to disassemble your Model M, start with a 7/32" nut driver or elongated hex socket. There are four bolts, all this size, along the top edge of the keyboard that must be removed. They should come off fairly easily. If they get stuck at all when loosening or tightening don't force anything - the threads for these bolts are machined right into the plastic, so they will strip easily if you crank down too hard.

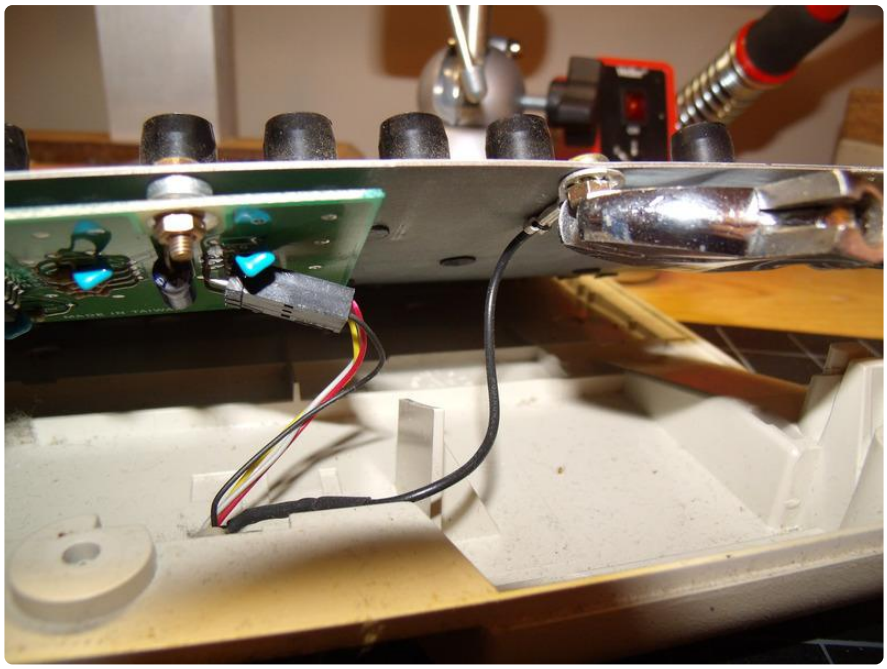


Once all of the bolts have been removed, the clamshell should open right up as pictured.



With the top removed, you will able to lift the steel plate that supports the keyboard's components. Tip it up slowly from the top - but you won't be able to completely remove it until you remove the grounding cable from the screw as pictured. Also, you will need to disconnect the cable from the controller board, which should expose a 3x2 matrix of male headers with .1" spacing. The middle pin from one of the rows will be missing, that is normal.

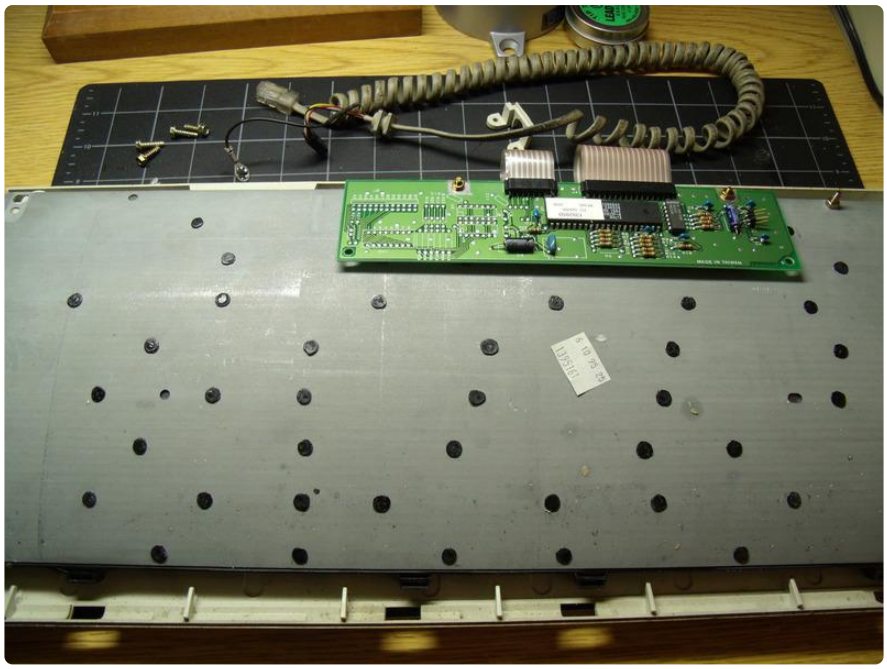
While every "terminal" Model M I've seen has a controller of this variety - attached to the steel plate and making a 3x2 array of header available - the circuitry on a "normal" Model M (that is, one with a "regular PS/2" connector) will most likely look different! Most likely there will not be header pins for you to utilize and it will probably be seated in the lower half of the clamshell.



With the cable detached you can remove it and set it aside, we will not be needing it anymore.

At this point, I recommend taking the steel plate and flipping it over, both to examine the controller and to check the plastic rivets. Model M's have plastic rivets that are melted to hold the key assembly tight against the steel plate. These tend to deteriorate over time, so after 20 or 30 years it is not uncommon to have a few or many of these broken off. If too many are missing in any given section, it can result in inconsistent tactility or even failed actuation. If you have a lot of these rivets missing you might want to double check how your keyboard feels to see if any repairs are in order - if so you will probably have to do a bolt mod to tighten things up ( ). Avoid this is if you can, because once you start there is no turning back, and tightening 80 or so tiny bolts can be the opposite of fun. However if your keyboard is not working properly this step can be well worth it.

As it happens, as dirty as this board was, all of the rivets are in perfect condition.



Now, with everything disassembled we can install our components and button everything back up.

Code

The PS/2 protocol is fairly straightforward, and the Arduino Micro is easily capable of running a bitbang implementation of the host side. We could also implement this with interrupts if we wanted to, but it is not necessary for this project.

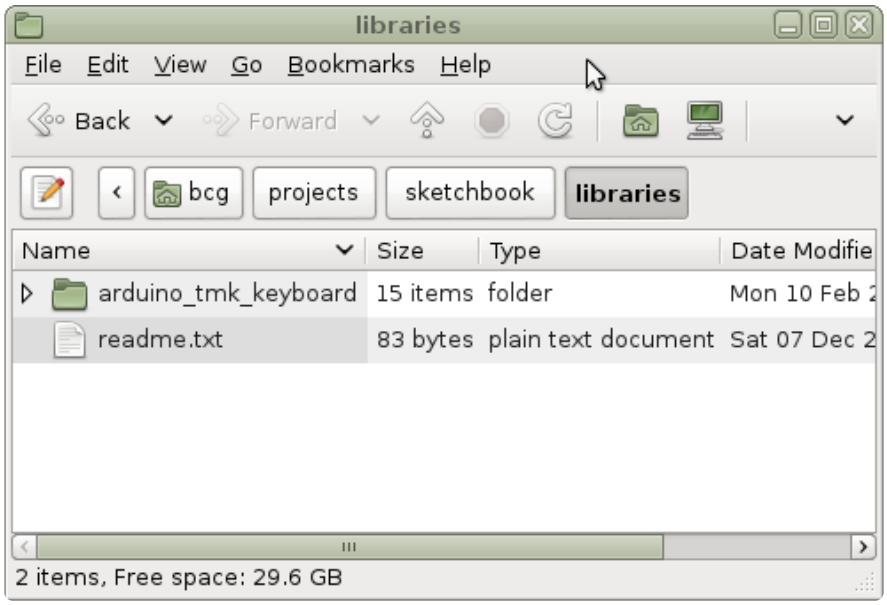
For this project I decided to lean on the excellent TMK Keyboard Firmware (). It has a ton a great features - multi-layer key maps, mouse keys, consumer keys, function callbacks for key presses, and more. Only one "problem" - it is written in AVR C with a Makefile based build, and I wanted to use the Arduino IDE to build and load the firmware. So I ported the tmk_keyboard project to the Arduino platform as a library, and creatively named it arduino_tmk_keyboard ().

Click the button below to download the library as a zip file.

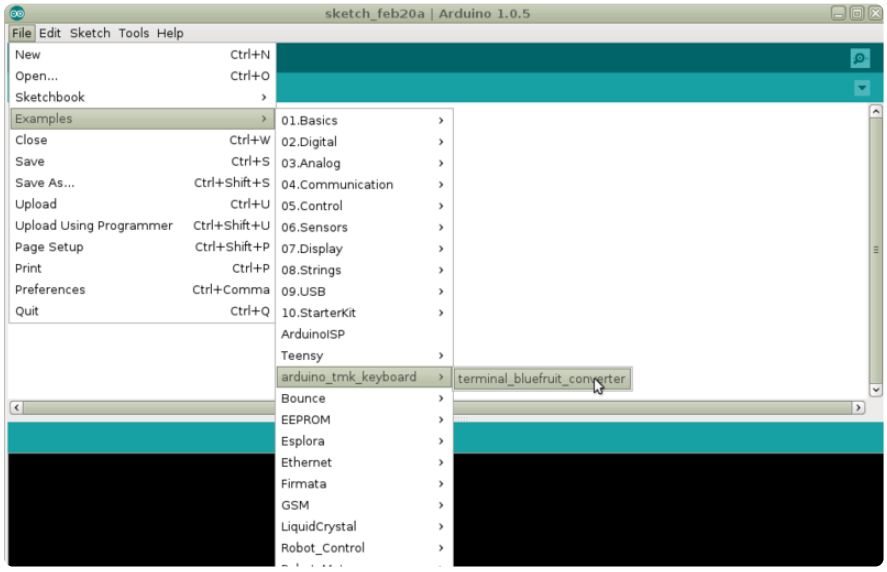
Download the Arduino library as a zip file

Getting Started

First, hit the button above to download a zip of the Arduino library. Unzip it into your sketchbook's "libraries" folder:



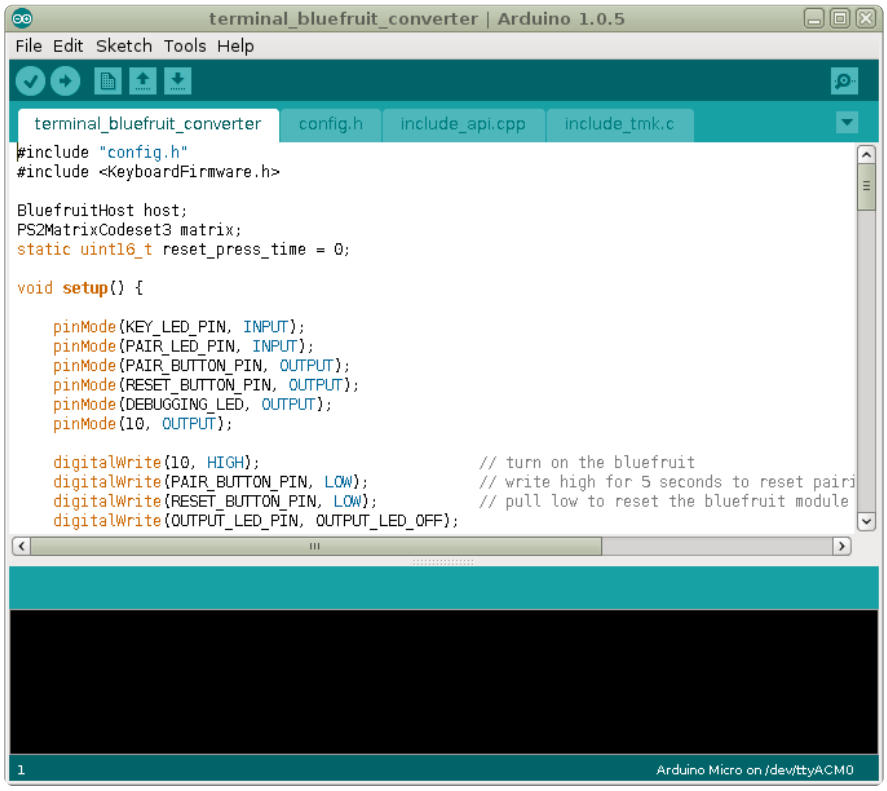
Next, fire up the Arduino IDE and open the 'terminal_bluefruit_converter' example sketch.



The example will open and you'll see four tabs:

1. terminal_bluefruit_converter - this is your main sketch, and probably the only place where you'll need to make any actual code changes. The bootstrapping of the firmware and any custom functions are defined here, and your keymaps are configured in this file as well.
2. config.h - this is the main configuration file; it is included at the top all other code files so it is a great place to define macros, as well as quickly enable and disable features including debug messages
3. include_api.cpp - this is a hack to properly pull in the library's API classes. Don't touch this file.
4. include_tmk_.c - this is another hack to pull in the C code from the TMK firmware. Don't touch this either.

Switch the board you're targetting to "Arduino Micro" and make sure you have the right serial port connected. Bonk that checkmark to verify you have things compiling correctly, then lets take a quick look at the code.



Configuring the Firmware

With the example sketch loaded, lets take look at the config.h file first. Here is where we can set up the constants and configuration options that the firmware uses. For instance I'm defining constants for the pins on the Micro that we'll be using here so that if you want to try this on a different Arduino it shouldn't be a monumental effort.

```
#ifndef KEYBOARDFIRMWARE_CONFIG_H
#define KEYBOARDFIRMWARE_CONFIG_H 1

#define RESET_BUTTON_PIN 8
#define PAIR_LED_PIN 7
#define PAIR_BUTTON_PIN 6
#define OUTPUT_LED_PIN 5
#define KEY_LED_PIN 4
#define OUTPUT_LED_ON HIGH
#define OUTPUT_LED_OFF LOW
#define DEBUGGING_LED 13
#define DEBUGGING_LED_ON HIGH
#define DEBUGGING_LED_OFF LOW

#define DEBUG_ENABLE true

#define EXTRAKEY_ENABLE 1
#define MOUSEKEY_ENABLE 1

#define MATRIX_ROWS 17
#define MATRIX_COLS 8
```

```
#define PS2_CLOCK_PORT  PORTD
#define PS2_CLOCK_PIN   PIND
#define PS2_CLOCK_DDR   DDRD
#define PS2_CLOCK_BIT   1

#define PS2_DATA_PORT   PORTD
#define PS2_DATA_PIN    PIND
#define PS2_DATA_DDR    DDRD
#define PS2_DATA_BIT    0

#define PS2_USE_BUSYWAIT true

#ifndef PS2_MATRIX_HAS_GHOSTING
#define PS2_MATRIX_HAS_GHOSTING false
#endif

#endif
```

I assume that the pin definitions at the top of the file do not require much explanation... for the most part they line up with the functionality of the Bluefruit. There are also a couple macros that define whether or not we need to set the pins low or high when we're dealing with LEDs. Beyond that, here are just some quick notes about this file:

- DEBUG_ENABLE - this is a useful feature when you're building your project, as it will output a lot of great information over the Arduino serial monitor as you type. Also debugging information from the Bluefruit module is forwarded to the serial monitor as well. To be sure this is available however on the Arduino Micro (and Leonardo) you have to loop and keep checking to see if it is ready... this will cause your keyboard to hang when it is turned on if you don't have anything connected and listening on the Micro's USB port. Therefore if you use this feature during testing be sure to disable before closing your keyboard up!

- EXTRAKEY_ENABLE and MOUSEKEY_ENABLE - these features enable and disable the "consumer keys" (or "media keys", like volume and music playback) as well as the "mouse keys" feature (which lets you control the mouse cursor using your keyboard). If you don't want these features, you can comment out these lines and your sketch will be a bit smaller. Don't forget to add them back in if you decide to use them - you won't get any errors if you map these keys while they're disabled, they just won't work!

- MATRIX_ROWS and MATRIX_COLS - these values are needed throughout the firmware and they don't typically change so its useful to define them here. Unless you're implementing this on a different keyboard you won't need or want to change these.

- PS2_USE_BUSYWAIT and the PS2 register and and pin definitions - this tells the framework that we are using the bitbang implementation for PS/2, and sets it up

on pins 2 and 3. TMK Firmware uses the regular AVR C definitions for the registers as opposed to the Arduino pin numbers; these values should be fine on the Micro, but you might need to double check on a different Arduino. These are also interrupt pins on the Micro so you don't have to change these if you decide to switch to the interrupt based PS/2 driver instead.

- PS2_MATRIX_HAS_GHOSTING - You won't need to do anything with this for this project... if we were implementing a keyboard matrix directly instead of making a converter, we might have to account for cases where multiple keys being pressed could create alternate paths for current to flow on the matrix and create "ghost" keystrokes. The Model M already accounts for ghosting on its matrix so we don't have to worry about it.

Integrating the Firmware

Using the Arduino library wrappers for the TMK firmware, its pretty simple to integrate hte keyboard's functionality into our sketch.

First, let's take a look at the setup() function for our sketch:

```
BluefruitHost host;
PS2MatrixCodeset3 matrix;
static uint16_t reset_press_time = 0;

void setup() {

    pinMode(KEY_LED_PIN, INPUT);
    pinMode(PAIR_LED_PIN, INPUT);
    pinMode(PAIR_BUTTON_PIN, OUTPUT);
    pinMode(RESET_BUTTON_PIN, OUTPUT);
    pinMode(DEBUGGING_LED, OUTPUT);

    digitalWrite(PAIR_BUTTON_PIN, LOW);            // write high for 5 seconds to
reset pairing
    digitalWrite(RESET_BUTTON_PIN, LOW);           // pull low to reset the
bluefruit module
    digitalWrite(OUTPUT_LED_PIN, OUTPUT_LED_OFF);

#if DEBUG_ENABLE
    debug_enable = true;
    while (!Serial) ;
#endif

    print_set_sendchar(arduino_tmk_sendchar);
    dprint("started logging\n");

    digitalWrite(DEBUGGING_LED, DEBUGGING_LED_ON);
    KeyboardFirmware.begin(host, matrix);
    digitalWrite(DEBUGGING_LED, DEBUGGING_LED_OFF);

    digitalWrite(RESET_BUTTON_PIN, HIGH);  // turn on bluefruit
}
```

We have some straightforward code configuring the pins to interact with the Bluefruit module; we're connecting the pair and reset buttons to the Arduino because we'll map those to keys on the keyboard later. After that comes some setup for the debugging features of the TMK framework, and finally the call that makes the magic happen:

KeyboardFirmware.begin(host, matrix);

This function call will send the PS2 commands to handshake with the keyboard and it will initialize the firmware. The "host" and "matrix" were defined at the top of the above snippet; in this case the "host" is the Bluefruit module, so we're using a BluefruitHost object. On a different keyboard this might be called USBHost or PS2Host (although I don't have wrappers written for those yet). Back on the Research page I discussed the PS2 protocol and how it has different codesets; we put that to use here, as we determined that if we can properly implement a PS2 host for a codeset 3 device, we can support all of the Model M keyboards that were produced by IBM.

Next lets take a look at our loop():

```
void loop() {

    // this the main hook into the tmk firmware that handles all the heavy lifting
    KeyboardFirmware.runTask();

    // Now sync the pair button light with the output pin
    if (digitalRead(PAIR_LED_PIN) == HIGH) {
        digitalWrite(OUTPUT_LED_PIN, OUTPUT_LED_ON);
    } else {
        digitalWrite(OUTPUT_LED_PIN, OUTPUT_LED_OFF);
    }

    // receive any messages from Bluefruit and output them if necessary
    unsigned char c;
    while (Serial1.available()) {
        c = (unsigned char) Serial1.read();
        if (debug_enable) Serial.write(c);
    }

    // next we'll check if the reset key was held down for 5 seconds
    if (reset_press_time &amp;amp;&amp;amp; timer_elapsed(reset_press_time) &gt; 5000) {
        dprintf("= setting reset low\n");
        digitalWrite(RESET_BUTTON_PIN, LOW);
        delay(10);
        digitalWrite(RESET_BUTTON_PIN, HIGH);
        dprintf("= restored reset high\n");
        reset_press_time = 0;
    }

}
```

This is very simple; the runTask() call is perhaps the most important call in the whole program, as that is the function that reads the state of the matrix, maps the matrix state to keycodes and/or macros, and then sends the state to the host.

The rest of the code in that block makes sure that the LED button on the keyboard stays in sync with the Pair LED of the Bluefruit, and also we are forwarding any output from Bluefruit's serial connection to the Arduino serial monitor for debugging purposes. Finally, we check the state of the "reset key" that we'll define on the keyboard, and if it has been held for 5 seconds we pull the reset pin on Bluefruit low for 10ms, which effectively reboots the module (without affecting the pairing).

Most of this code is pretty boilerplate, and relatively boring. In the next section you can be a bit more creative though...

Defining the Keymaps

This is where we get to have a little fun! Well, maybe not quite "drink-a-twelve-pack-of-root-beer-and-reinstall-Linux" fun, but at least it will allow you to get creative using the medium of expressive scancode remapping.

You'll find the example keymaps at the bottom of the sketch:

```
START_KEYMAPS

KEYMAPS = {
    /* 0: default
     * ,---.   ,---------------. ,---------------. ,---------------. ,-----------.
     * |Esc|   |F1 |F2 |F3 |F4 | |F5 |F6 |F7 |F8 | |F9 |F10|F11|F12| |PrS|ScL|Pau|
     * `---'   `---------------' `---------------' `---------------' `-----------'

* ,-----------------------------------------------------------. ,-----------. ,---------------
     * |  `|  1|  2|  3|  4|  5|  6|  7|  8|  9|  0|  -|  =|  \|BS | |Ins|Hom|PgU| |
NmL|  /|  *|  -|
     * |-----------------------------------------------------------| |-----------| |---------------|
     * |Tab  |  Q|  W|  E|  R|  T|  Y|  U|  I|  O|  P|  [|  ]|    \| |Del|End|PgD|
|  7|  8|  9|   |
     * |-----------------------------------------------------------| `-----------' |-----------|  +|
     * |CapsLo|  A|  S|  D|  F|  G|  H|  J|  K|  L|  ;|  '|  #|Retu|
|  4|  5|  6|   |
     * |-----------------------------------------------------------|     ,---.     |---------------|
     * |Shif|  \|  Z|  X|  C|  V|  B|  N|  M|  ,|  ,|  /|Shift     |     |Up |     |  1|  2|  3|   |
     * |-----------------------------------------------------------| ,-----------. |-----------|Ent|
     * |Ctrl|    |Alt |          Space              |Alt |    |Ctrl| |Lef|Dow|Rig| |      0|  .|   |
     * `----'    `-------------------------------------'    `----' `-----------' `---------------'
     */
    PS2_CODESET3_KEYMAP(
                    FN2, F14, F15, F16, F17, F18, F19, F20, F21, F22, PSCR,SLCK,
                    F1,  F2,  F3,  F4,  F5,  F6,  F7,  F8,  F9,  F10, F11, F12,

    PSCR,ESC,   GRV, 1,   2,   3,   4,   5,   6,   7,   8,   9,   0,   MINS,EQL,
JYEN,BSPC,  INS, HOME,PGUP,  NLCK,PSLS,PAST,PMNS,
    SLCK,INT4,  TAB, Q,   W,   E,   R,   T,   Y,   U,   I,   O,   P,
LBRC,RBRC,     BSLS,  DEL, END, PGDN,  P7,  P8,  P9,  PPLS,
```

```
    PAUS,INT5,  LCTL,A,   S,   D,   F,   G,   H,   J,   K,   L,   SCLN,QUOT,
NUHS,ENT,        UP,         P4,  P5,  P6,  PCMM,
    APP, INT6,  LSFT,NUBS,Z,   X,   C,   V,   B,   N,   M,   COMM,DOT, SLSH,
RO,  RSFT,  LEFT,PAUS,RGHT,  P1,  P2,  P3,  PENT,
    RGUI,LGUI,  FN0,      LALT,               SPC,
RALT,     RCTL,       DOWN,       NO,  P0,  PDOT,NO
    ),
    PS2_CODESET3_KEYMAP(
                     TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,
                     TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,

    TRNS,TRNS,
TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,  FN1,
LGUI,VOLU,  TRNS,TRNS,TRNS,TRNS,
    TRNS,TRNS,
TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,     TRNS,  FN2,
APP, VOLD,  BTN1,MS_U,BTN2,TRNS,
    TRNS,TRNS,  TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,
TRNS,TRNS,       TRNS,       MS_L,MS_D,MS_R,TRNS,
    TRNS,TRNS,  TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,TRNS,
TRNS,TRNS,  TRNS,TRNS,TRNS,  TRNS,TRNS,TRNS,TRNS,
    TRNS,TRNS,  TRNS,     TRNS,               TRNS,
LGUI,     APP,        TRNS,       TRNS,TRNS,TRNS,TRNS
    ),
};

FN_ACTIONS = {
    ACTION_FUNCTION(0),
    ACTION_FUNCTION(1),
    ACTION_FUNCTION(2),
};

KEYMAPS_FINISHED
```

Basically, these are a bunch of macros that implement the steps necessary to create mapping arrays for the converter's scancodes to the values expected by the host device. The PS2_CODESET3_KEYMAP macro maps the default PS2 scan codes into an order that allows us to define our key layers laid out like the actual keyboard. Don't get confused by the layout however... that is what the 122-key Model M looks like.

The keymaps above however work well on my 101-key Model M, and all of the scan codes are in the same spot on both keyboards except for three: Print Screen (F23), Scroll Lock (F24), and Pause (at the center of the arrow keys on the 122). If you're interested you can find complete diagrams of the scancodes here: http://www.seasip.info/VintagePC/ibm_1390876.html ().

Here I've defined 2 layers; the first is the pretty standard layout for a Model M, except that I got rid of the Caps Lock key (which I consider to be useless) and moved the Left

Control key up in its place (where it belongs). The Left Control key then becomes a function key (FN1) that switches the keyboard to Layer 2.

The second key layer is mostly transparent ("TRNS") which means that whatever is on the layer below is used for that key. You'll find the layer 2 keys near Insert and Delete, as well as on the number pad. Insert becomes a function key that when held down for 5 seconds resets the pairing of the Bluefruit, and Delete sends the Bluefruit a reset signal. On the number pad are "mouse key" controls - if you hold down FN1 and use those keys, you can move the mouse cursor and use BTN1 and BTN2 to left and right click, just like you would with any other pointing device.

Have fun and play around with this part until you have things set up the way you like them... you can find more documentation on how to do key mapping in TMK's documentation:

- https://github.com/tmk/tmk_keyboard/blob/master/doc/keymap.md ()
- https://github.com/tmk/tmk_keyboard/blob/master/doc/keycode.txt ()

(not all of the key mapping features are supported by the Arduino framework... for instance oneshot and key tapping are not supported at the time of this writing).

Interacting with Bluefruit

The last thing we need to do to complete our sketch is to implement the function that will allow us to control the Bluefruit module. In the keymap, we defined 3 function keys. For each of these function keys, we defined an associated action in the "FN_ACTIONS" array:

```
FN_ACTIONS = {
    ACTION_FUNCTION(0),
    ACTION_FUNCTION(1),
    ACTION_FUNCTION(2),
};
```

The ACTION_FUNCTION macro tells the TMK framework that there should be a function called action_function defined in the sketch, and that when the given FN key is pressed or released the corresponding index should be passed into that function, along with metadata about the event (such as if the key was pressed or released).

Therefore, we can define our action function like this:

```
void action_function(keyrecord_t *record, uint8_t id, uint8_t opt)
{
    bool pressed = record-&gt;event.pressed;
    dprint("== action function called"); dprintln();
    dprint("=  id:      "); debug_dec(id); dprintln();
```

```
    dprint("=  pressed: "); debug_dec(record-&gt;event.pressed); dprintln();
    if (id == 0) {
        if (pressed) {
            layer_on(1);
        } else {
            layer_clear();
        }
    } else if (id == 1) {  // bluefruit pair button
        if (pressed) {
            dprintf("= setting pair button HIGH\n");
            digitalWrite(PAIR_BUTTON_PIN, HIGH);
        } else {
            dprintf("= setting pair button LOW\n");
            digitalWrite(PAIR_BUTTON_PIN, LOW);
        }
    } else if (id == 2) {
        if (pressed) {
            if (reset_press_time == 0) {
                reset_press_time = timer_read();
            }
        } else {
            reset_press_time = 0;
        }
    }
    dprint("== end of action function\n");
}
```

The logic here is pretty simple; the id variable that is passed represents the index that was defined for the given function key in FN_ACTIONS, so we just have an if-else structure to react to that appropriately. Pressing or releasing FN0 toggles the second layer, holding down FN1 for 5 seconds resets the pairing of the Bluefruit, and holding down FN2 for 5 seconds resets the module. You can implement any sort of arbitrary logic in this function, just try to make sure it returns quickly so that your keyboard is nice and responsive!

Since we're on the subject of interacting with Bluefruit, lets take a quick look at the BluefruitHost object:

```
void BluefruitHost::begin()
{
    Serial1.begin(9600);
}

uint8_t BluefruitHost::getLEDs()
{
    // not implemented on Bluefruit; method is virtual so feel free to override
    return 0;
}

void BluefruitHost::sendKeyboard(KeyboardReport &amp;report)
{
    bluefruit_trace_header();
    dprintf("(keyboard) ");
    _serial_send(0xFD);
    _serial_send(report.getModifiers());
    _serial_send(report.getReserved());
    for (short i = 0; i &lt; REPORT_SIZE; i++)
    {
        _serial_send(report.getKey(i));
    }
    bluefruit_trace_footer();
```

```
}

void BluefruitHost::sendMouse(MouseReport &amp;report)
{
    bluefruit_trace_header();
    dprintf("(mouse) ");
    _serial_send(0xFD);
    _serial_send(0x00);
    _serial_send(0x03);
    _serial_send(report.getButtons());
    _serial_send(report.getX());
    _serial_send(report.getY());
    _serial_send(report.getV()); // TODO: determine if bluefruit
    _serial_send(report.getH()); //       supports mouse wheel - BCG
    _serial_send(0x00);
    bluefruit_trace_footer();
};
```

Here we see that TMK is sending a keyboard and mouse report to the host object. As it happens, the format of the reports is basically exactly the same information that Bluefruit can consume over its serial connection. Starting with Bluefruit 1.1, you can send "raw HID" report to the Bluefruit module, enabling you to send state for up to 6 keys + modifiers, as well as mouse reports - that's exactly what the above code is doing.

In Bluefruit 1.2, support for "consumer keys" was added. You can also send this information over serial, allow it is a little trickier. The data that Bluefruit expects is a 16 bit map to denote which key is pressed. Some of the consumer key functions do not map to existing TMK keycodes, so I've ignored those; similarly some of the defined TMK consumer keys are not supported by Bluefruit, so I've ignored those as well. Feel free to change them to suit your purposes.

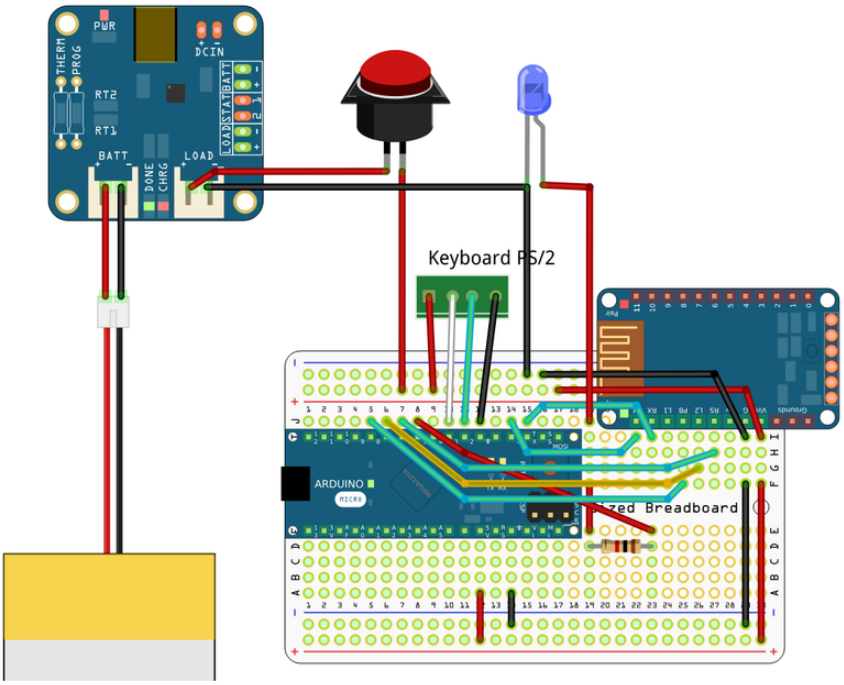
```
/*
+-----------------+-------------------+-------+
| Consumer Key    | Bit Map           | Hex   |
+-----------------+-------------------+-------+
| Home            | 00000001 00000000 | 01 00 |
| KeyboardLayout  | 00000010 00000000 | 02 00 |
| Search          | 00000100 00000000 | 04 00 |
| Snapshot        | 00001000 00000000 | 08 00 |
| VolumeUp        | 00010000 00000000 | 10 00 |
| VolumeDown      | 00100000 00000000 | 20 00 |
| Play/Pause      | 01000000 00000000 | 40 00 |
| Fast Forward    | 10000000 00000000 | 80 00 |
| Rewind          | 00000000 00000001 | 00 01 |
| Scan Next Track | 00000000 00000010 | 00 02 |
| Scan Prev Track | 00000000 00000100 | 00 04 |
| Random Play     | 00000000 00001000 | 00 08 |
| Stop            | 00000000 00010000 | 00 10 |
+-----------------------------------+-------+
*/
#define CONSUMER2BLUEFRUIT(usage) \
    (usage == AUDIO_MUTE           ? 0x0000  : \
    (usage == AUDIO_VOL_UP         ? 0x1000  : \
    (usage == AUDIO_VOL_DOWN       ? 0x2000  : \
    (usage == TRANSPORT_NEXT_TRACK ? 0x0002  : \
    (usage == TRANSPORT_PREV_TRACK ? 0x0004  : \
    (usage == TRANSPORT_STOP       ? 0x0010  : \
    (usage == TRANSPORT_STOP_EJECT ? 0x0000  : \
```

```
    (usage == TRANSPORT_PLAY_PAUSE ? 0x4000  : \
    (usage == AL_CC_CONFIG         ? 0x0000  : \
    (usage == AL_EMAIL             ? 0x0000  : \
    (usage == AL_CALCULATOR        ? 0x0000  : \
    (usage == AL_LOCAL_BROWSER     ? 0x0000  : \
    (usage == AC_SEARCH            ? 0x0400  : \
    (usage == AC_HOME              ? 0x0100  : \
    (usage == AC_BACK              ? 0x0000  : \
    (usage == AC_FORWARD           ? 0x0000  : \
    (usage == AC_STOP              ? 0x0000  : \
    (usage == AC_REFRESH           ? 0x0000  : \
    (usage == AC_BOOKMARKS         ? 0x0000  : 0))))))))))))))))
void BluefruitHost::sendConsumer(uint16_t data)
{
    if (data == _last_consumer_data) return;
    _last_consumer_data = data;

    uint16_t bitmap = CONSUMER2BLUEFRUIT(data);
    bluefruit_trace_header();
    dprintf("(consumer) ");
    _serial_send(0xFD);
    _serial_send(0x00);
    _serial_send(0x02);
    _serial_send((bitmap&gt;&gt;8)&amp;0xFF);
    _serial_send(bitmap&amp;0xFF);
    _serial_send(0x00);
    _serial_send(0x00);
    _serial_send(0x00);
    _serial_send(0x00);
    bluefruit_trace_footer();
};

void BluefruitHost::sendSystem(uint16_t data)
{
    // not implemented in Bluefruit
}
```

The sendSystem() function of the host normally would be used to send "system" commands, for example to put to the computer to sleep or to wake it up from suspend... however Bluefruit does not support this type of HID profile so it is ignored in our code.

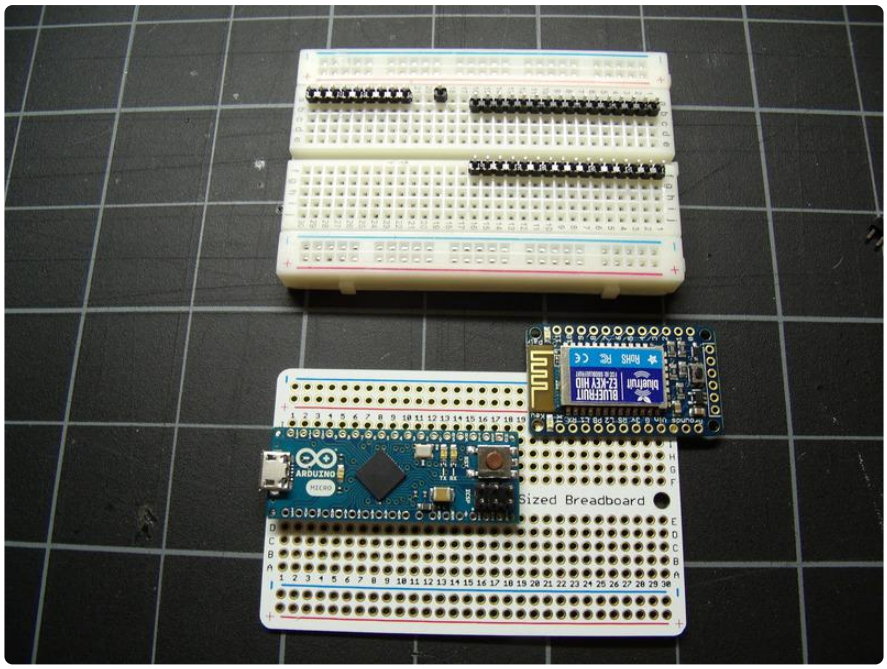
At this point, we've touched on all the major aspects of the code for this project, so after you've updated the keymaps and whatever else you'd like to customize, go ahead and compile it and load it on to your Arduino Micro to prepare to assemble and test the keyboard.

Circuit

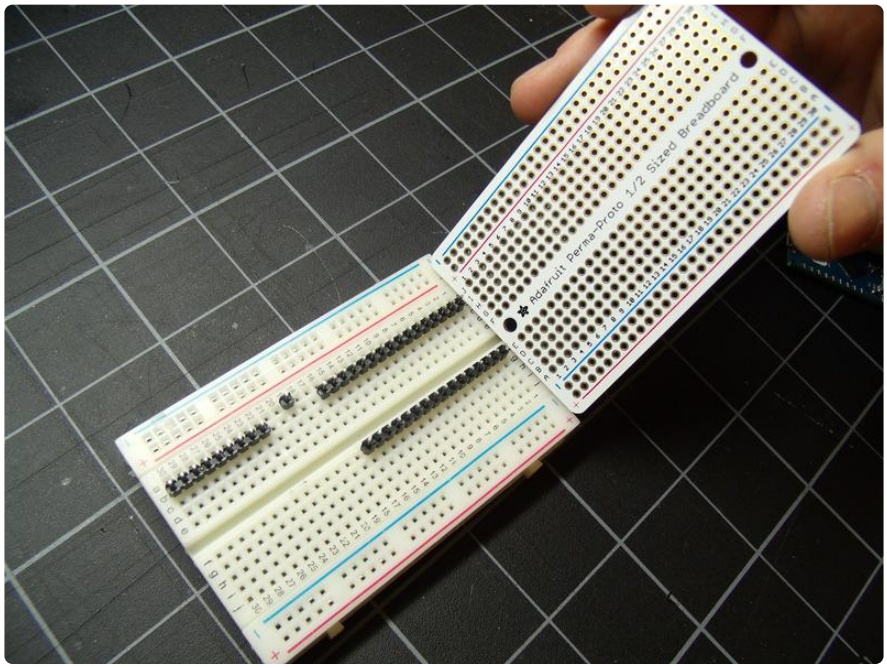


The first thing I want to do is lay out my components and get some header tacked in place on the PCB where they will go. I'm going to get the header onto the PCB before I solder to the components themselves, because I'm perpetually blessed to always make a mistake and I'd rather do the rework before I've soldered onto my more expensive components if I have to.

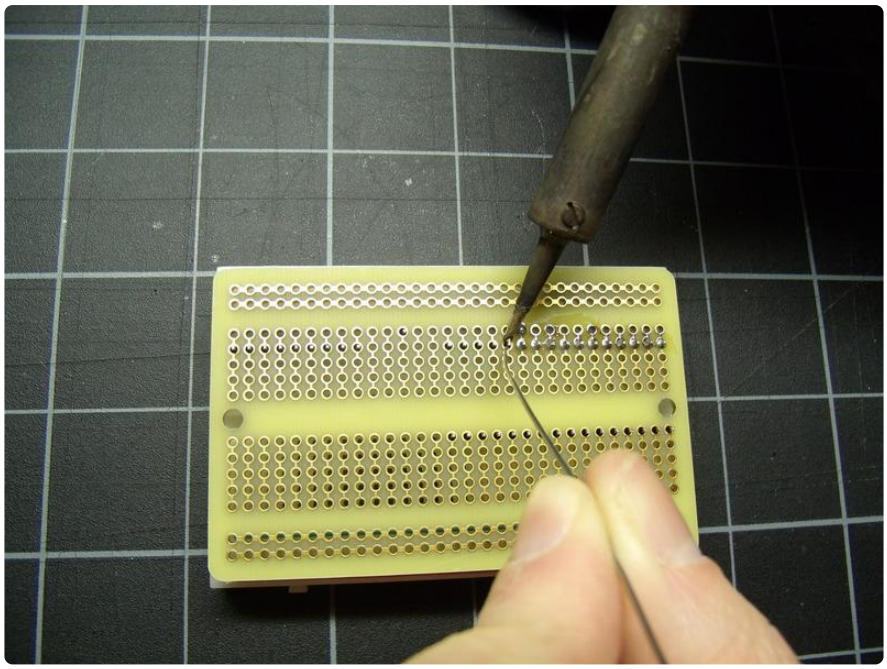
Start by laying out the components on the perma-proto where you'll want them to go. The perma-proto board is numbered the same as a half-sized breadboard, so I can create a mirror image of the PCB by simply rotating the breadboard 180 degrees so that the number 1 is on the right side instead of the left. I'll then put some strips of header into the breadboard in the same numbered columns that I used to lay out the components.



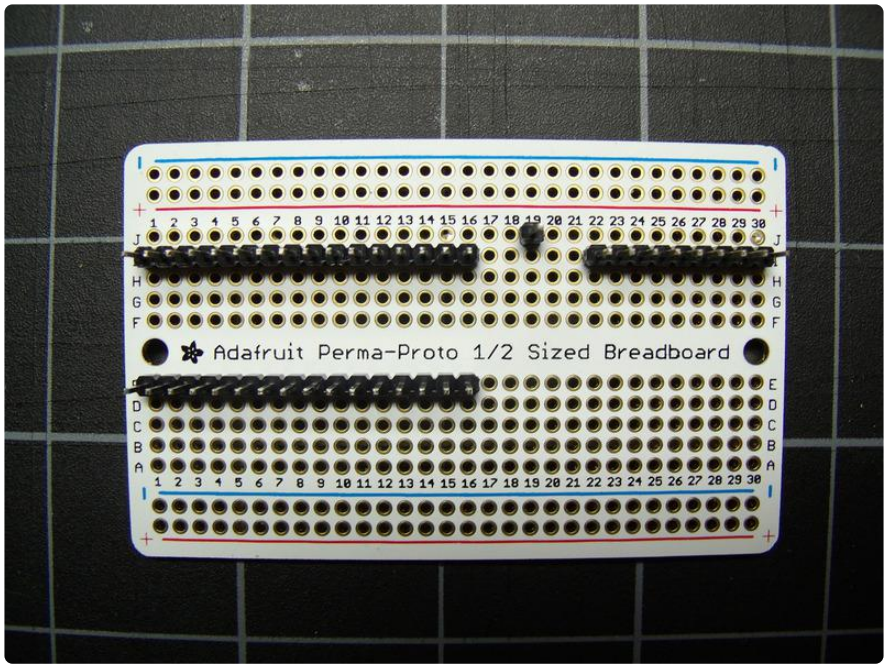
Next, I'll take the perma-proto and line up the "1" columns and place the PCB on top of the breadboard so that the white silk screened side is facing the breadboard.



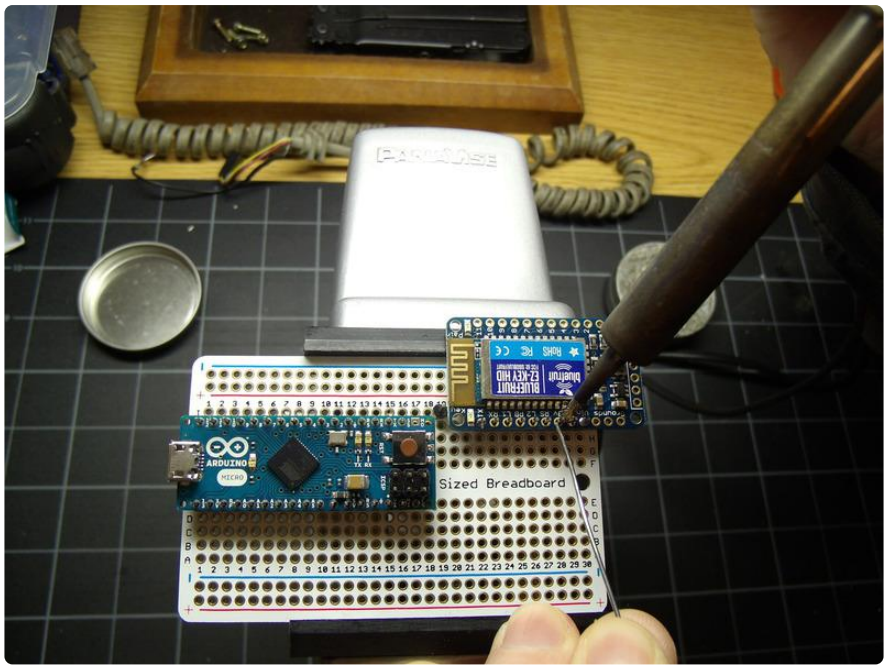
Now I'll solder the header wherever it is sticking through.



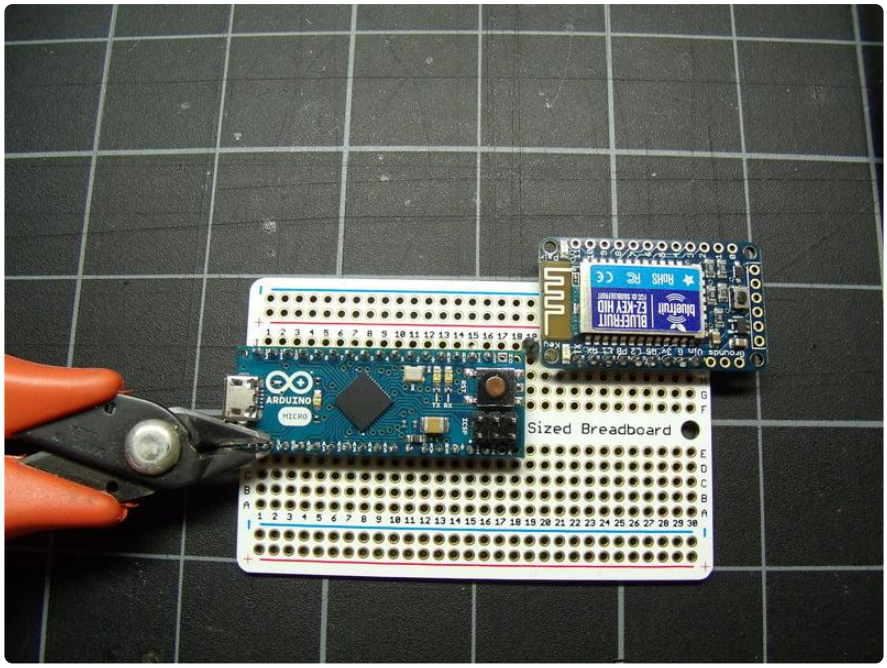
When I remove the PCB, I should have header neatly soldered in the same locations that I laid out my components. Take a moment to do a "dry fit" here and make sure that everything is where you'd like it to be. I actually did end up doing some rework on this project, so some of the subsequent pictures might look slightly different, but this layout should work fine:



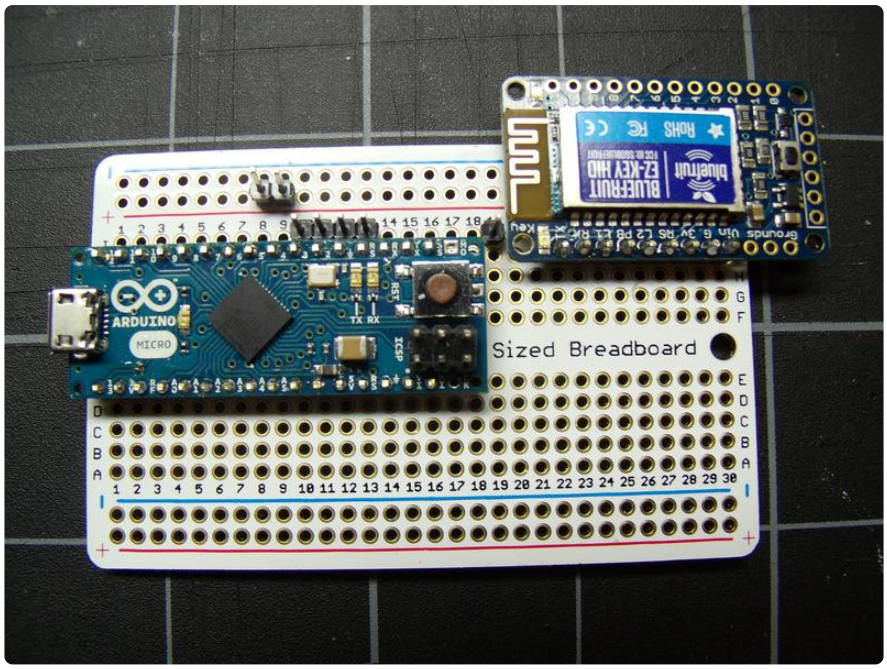
After test fitting your components, go ahead and solder them on too.



With your flush cutters, snip off the long ends of the header pins.

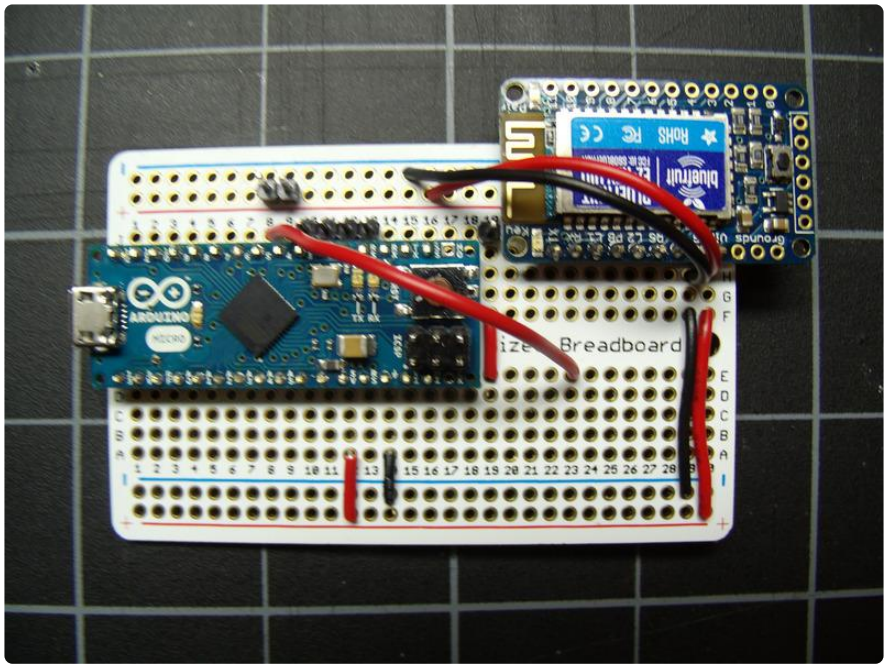


Your PCB should look something like this at this point:

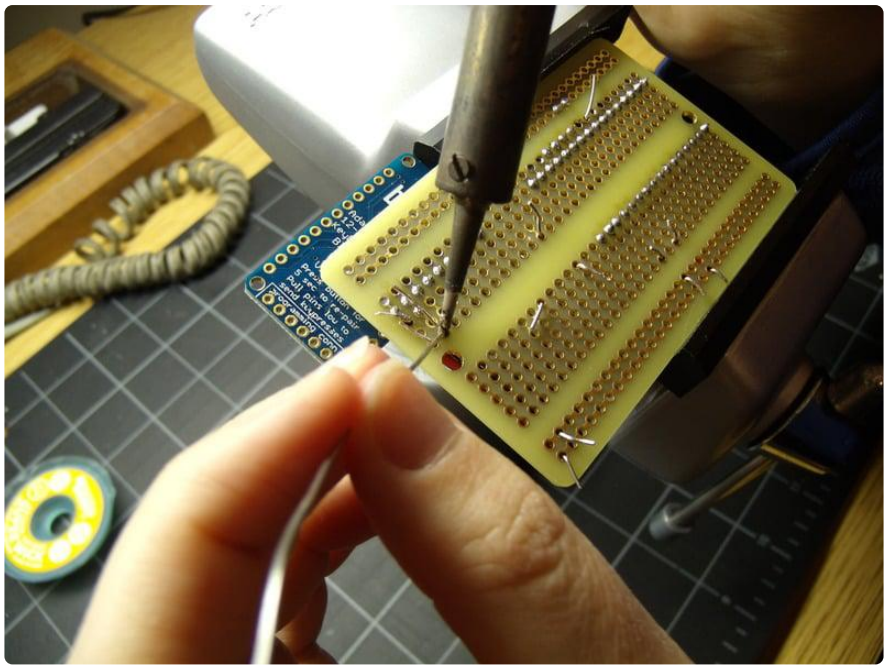


One change I made later on, I didn't use the straight header pins in the power rail or right above the Arduino; I switched to right angle header instead to make it fit inside the keyboard case.

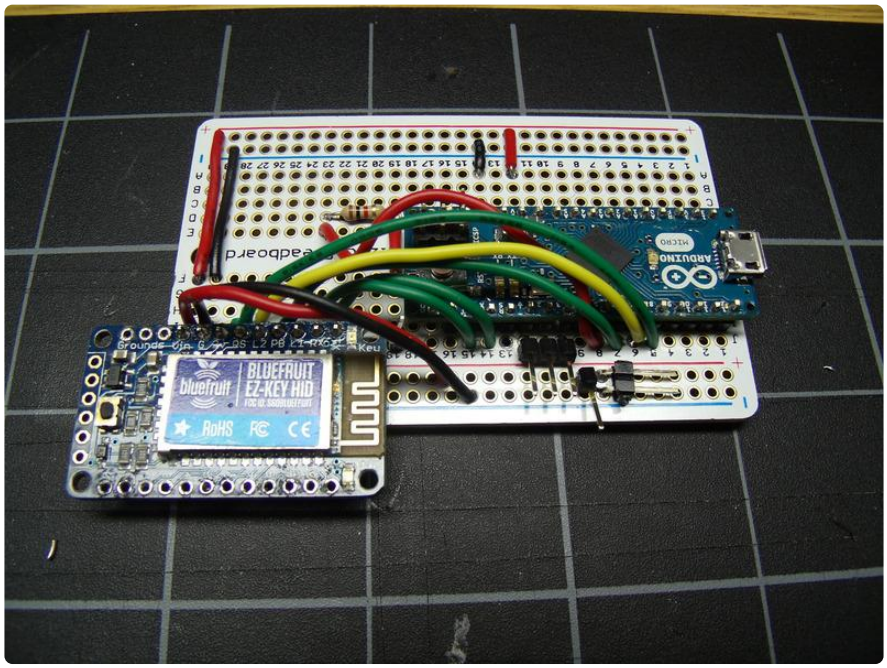
Lay out your wiring now, strip the ends and put the leads through the holes, bending the leads over to keep everything in place.



Now flip the PCB and solder up the connections. See the Fritzing diagram to see how to do your wiring... and don't forget the 1K resistor for the LED button we'll be using.



After soldering and clipping the leads, the converter portion of your circuit is fully assembled:



Notice the place of the right angle header on the power rails and near pins 2 and 3 on the Arduino... make sure you don't forgot to solder these, you'll need them to connect to the keyboard and power circuitry!

Assembly

I'd like to be able to remove the parts for this project individually if I ever need to disassemble it, so I'd like to avoid soldering everything together with permanent

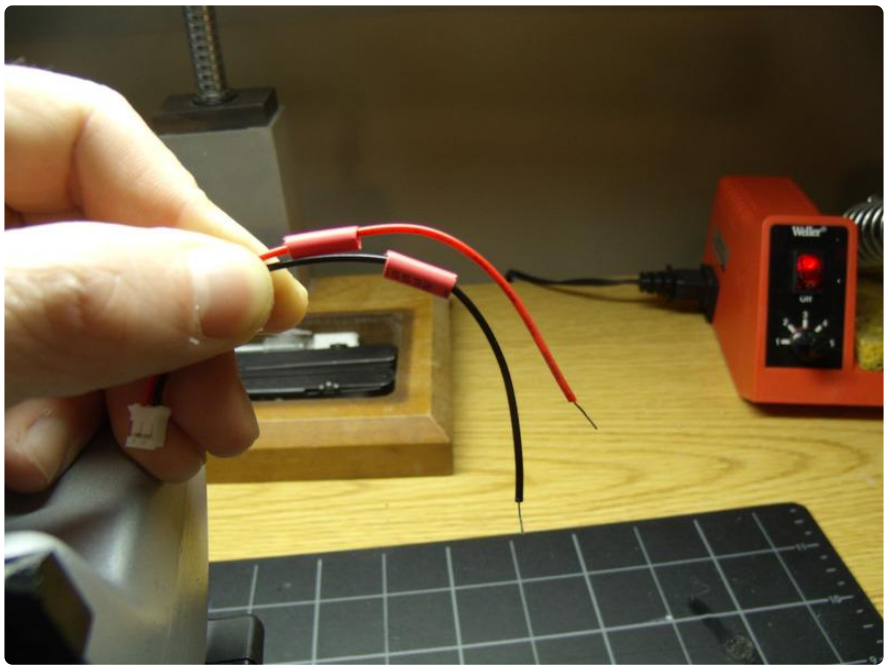
joints. Therefore I'm utilizing female jumper wires to allow for connections that can be easily unmade and remade when necessary. This is especially useful for the LED button, since I'd like for the top of the clamshell to be completely removable for future hacking.

Let's start by cutting one of our jumper wires in half; we'll solder each half to a different terminal on the button.

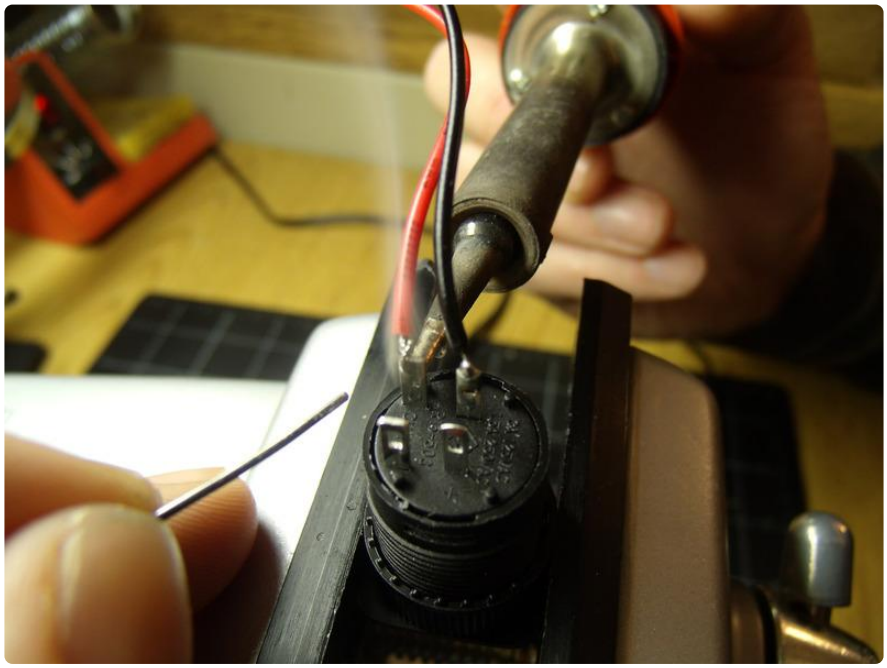
This is stranded wire so I'm just going to tin the ends a bit.

This is optional, but I'm adding a couple small pieces of heat shrink to the JST cable that came with the battery. This cable will get soldered to the LED button. The black

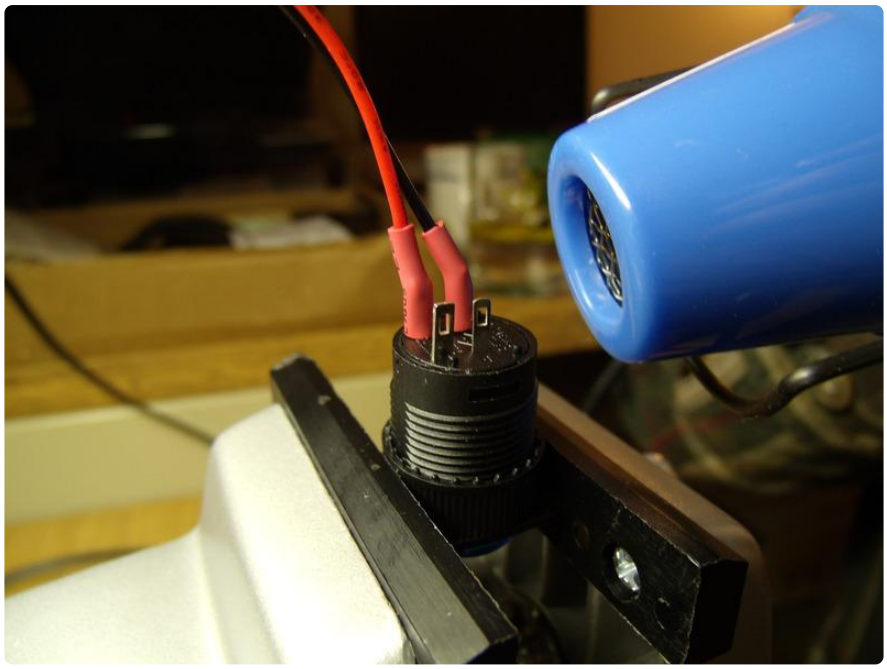
wire will be soldered to the anode (-) of the LED. The red wire will be soldered to one side of the switch. One half of the jumper wire that we cut will be soldered to the other side of the switch, and the remaining half will connect to the cathode (+) of the LED.



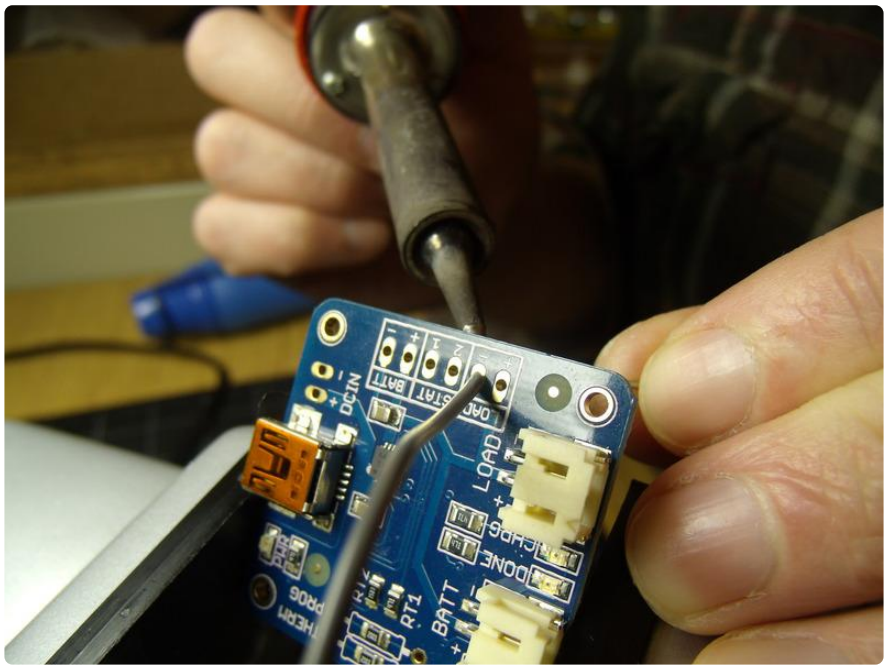
Put the leads of the wires you are soldering to the button through the holes in the terminal and bend them over to hold things in place temporarily. A small touch of solder will then keep everything nice and secure.



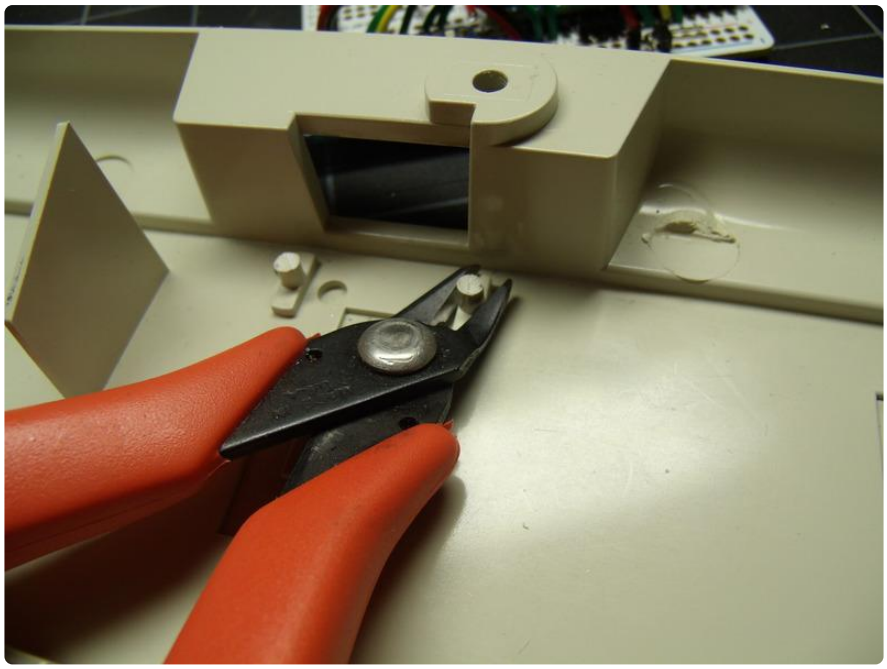
Slide your heat shrink down and blast it with a heat gun to make everything snug.



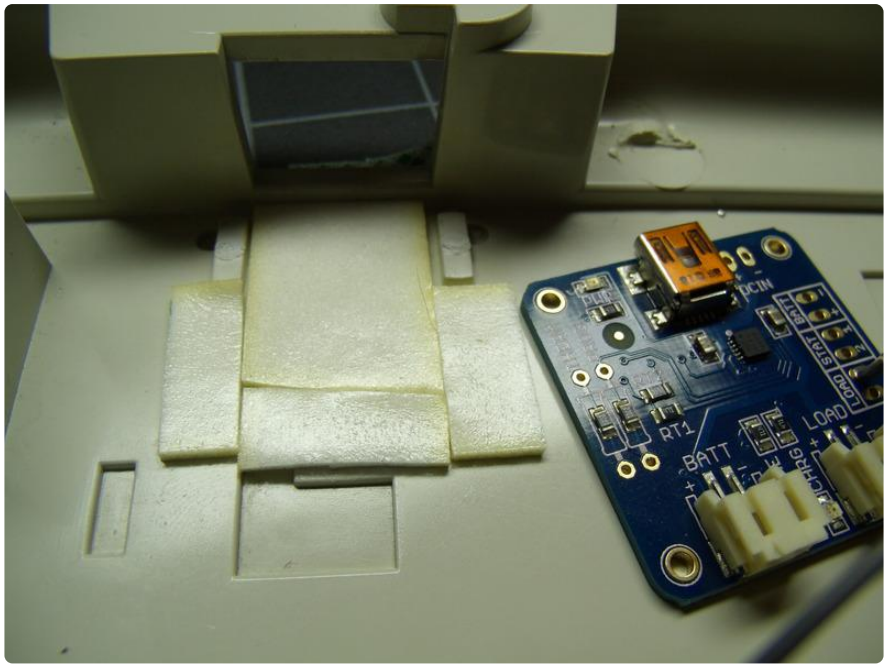
After soldering on all of the wires to the button, solder another jumper wire to the load ground terminal on the battery charger. This will connect to the ground rail on the converter PCB.



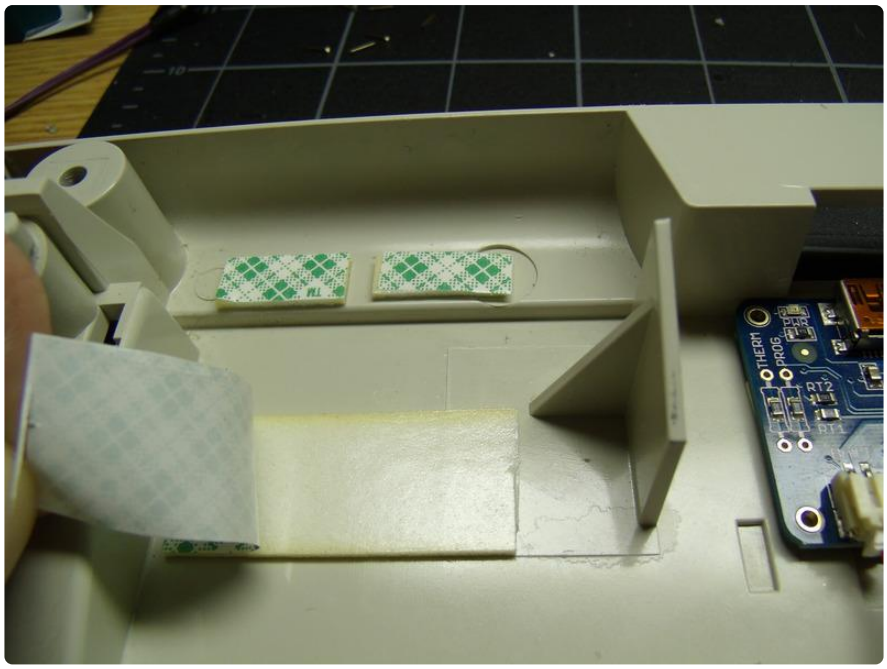
In the bottom of your Model M's clamshell is an opening where the cable used to be, and there should be 2 small posts. I cut these off since I will be putting the battery charger here, and they get in the way.



Put down some double sided foam tape and set the charger in place.

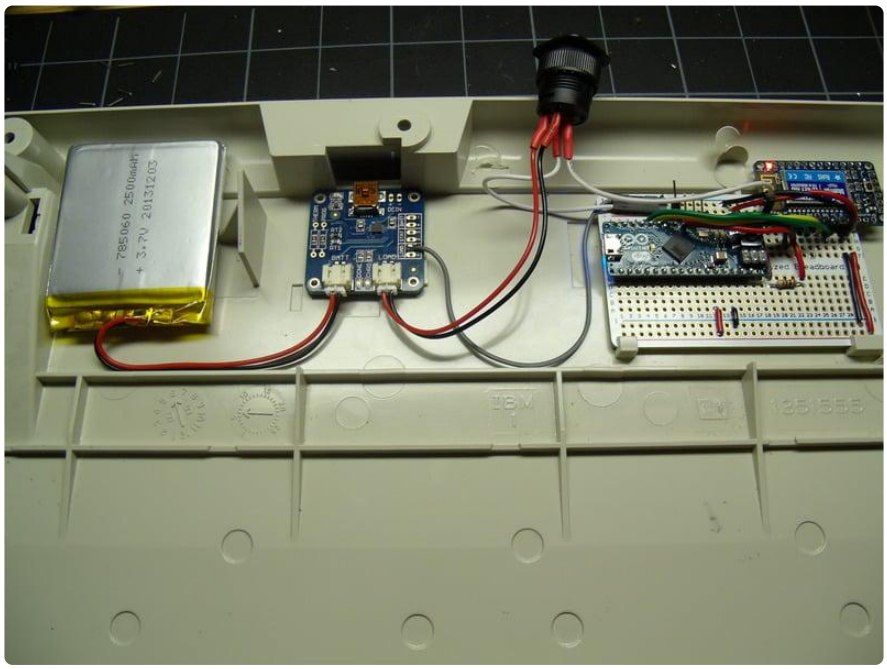


Next, I test fit the battery and put down foam tape where it sits in the case.

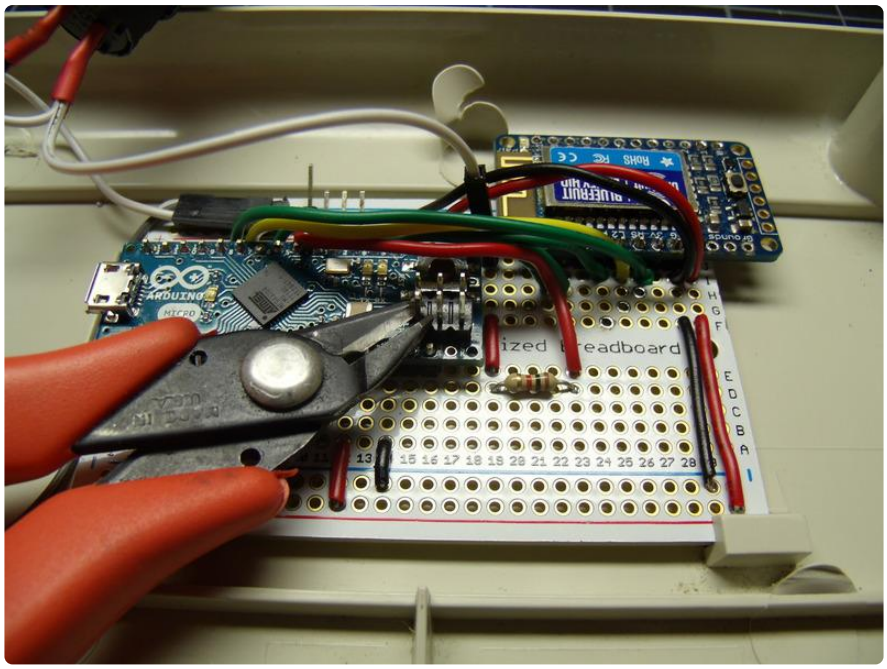


After setting the battery in place, I slid the converter PCB into the existing standoffs in the Model M case. Coincidentally, it fit perfectly! Almost as if it was designed that way...

Next I attached the button - plugged in the JST connector to the LOAD socket, and connected the jumper on other side of the switch to the PCB's power rail, and the cathode of the LED to the header pin sticking up in column 19 of the PCB.

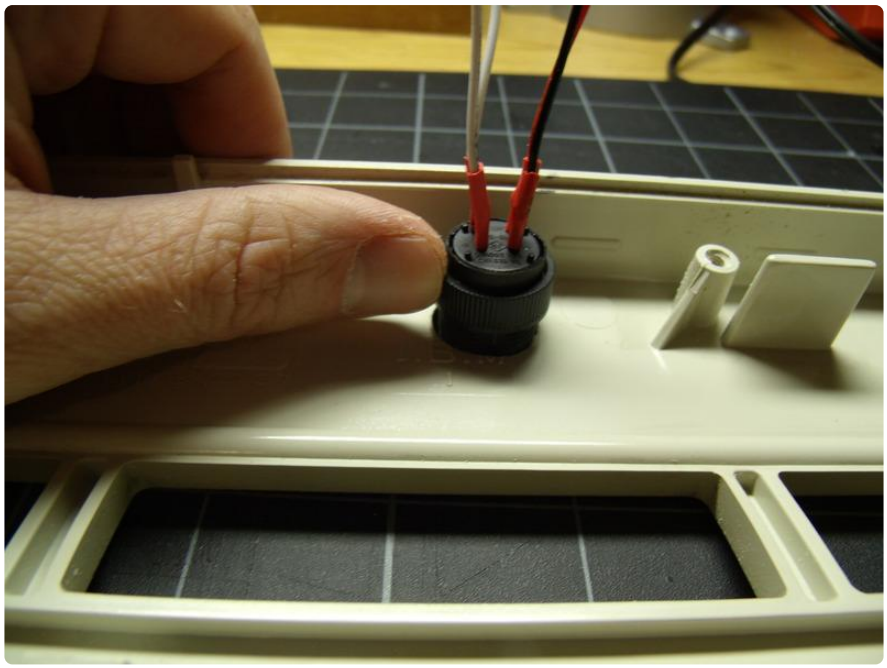


After test fitting the keyboard assembly with all of the extra parts installed, I found that the programming header on the Arduino Micro was being shorted by the keyboard's existing controller, so I cut it off.

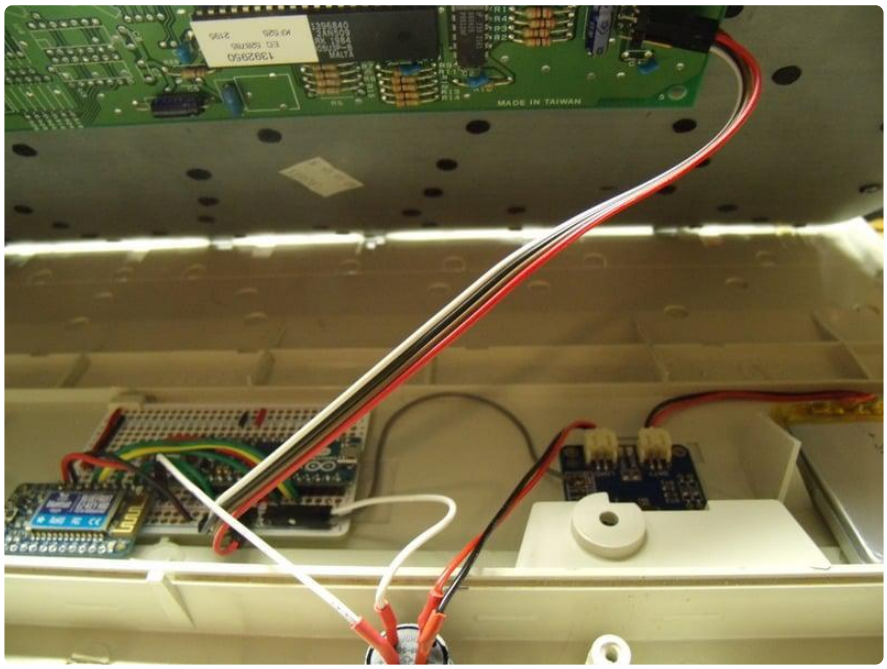


Finally, I have a good test fit... almost ready to put everything back together! One mistake I made was the placement of the LED button... it slightly interferes with the FPC cable as shown in the picture, but I was able to carefully fold the FPC under and it seems to work fine.

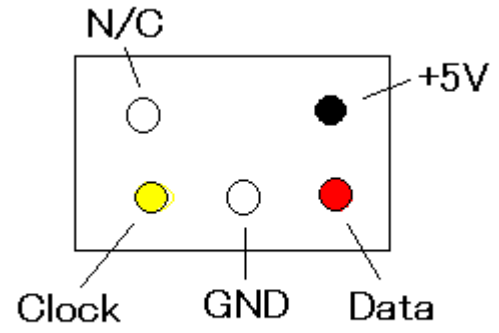
I made note that the button is roughly aligned with the F6 key, and used that as a reference of where to put the mounting hole. A couple minutes with a Dremel tool and I was able to mount the button to the top of the clamshell.



FInally we need to connect the converter the controller of the keyboard. I used 4 jumper wires, still attached to each other, to go from the PS2 connector on the keyboard controller to the appropriate pins on the converter PCB.



For quick reference, here is the pinout of the Model M controller's connector:



Facing to a connector on the PCB

With everything in place, double check your connections and then carefully close everything back up. I recommend only using one of the four bolts to keep everything in place until the keyboard is tested. Other than that, your keyboard is just about finished. Take this opportunity to make sure the charger and battery are working correctly and give the battery a good charge before you test the keyboard.

Use It!

Pairing the Keyboard

Make sure you have a PC or other bluetooth enabled device that supports Bluetooth 2.1. Turn on the keyboard by pressing the latching button down, and wait for the Arduino's bootloader to tick off 8 seconds before it boots up.

The Bluefruit module should show up on your PC as a keyboard, and you can click through the dialogs to complete the pairing. The Arduino sketch keeps the button's LED in sync with the pairing LED on the Bluefruit, so you can use its flashing as an indicator of the pairing status. Check out this guide for more detailed steps on pairing Bluefruit with your devices ().

Once you have successfully paired the keyboard, the LED indicator should be flashing slowly.

Using the Keyboard

Once the Model M is paired to your device, you're ready to go! I recommend that before you tightly button everything back up, you should take this opportunity to test out all the keys and make sure they do what you intend them to do. If you need to do any last minute remapping it is a lot easier to do if you've only put one of the bolts

back in so far!

Conclusion

I found this project to be very fulfilling and educational, and I was extremely impressed with how easy was to use the Bluefruit HID module. There are definitely some improvements that I'd like to make in the future, such as replacing the entire stock keyboard controller and just wire the FPC cables from the Model M's matrix to the Arduino to reduce the power consumption. I found that the battery used in this project gives about 24 hours of use on a single charge... those 30 year old electronics really suck down a lot of power! Also at some point I'd like to add a thumb joystick (http://adafru.it/512) to control the mouse cursor instead of using mouse keys.

If you enjoy keyboard hacking, or you have questions about the Model M or other keyboards, definitely check out the maker sections on keyboard forums such as geekhack.org () or deskthority.net ()!

Happy Hacking!