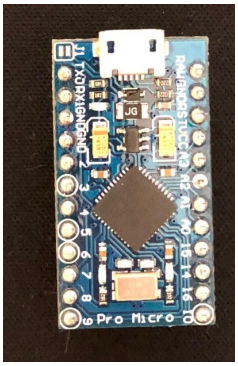
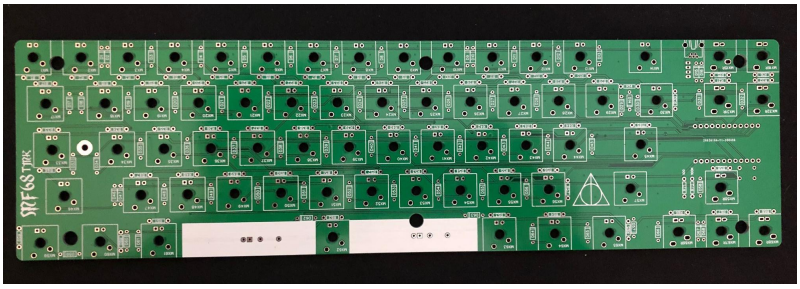
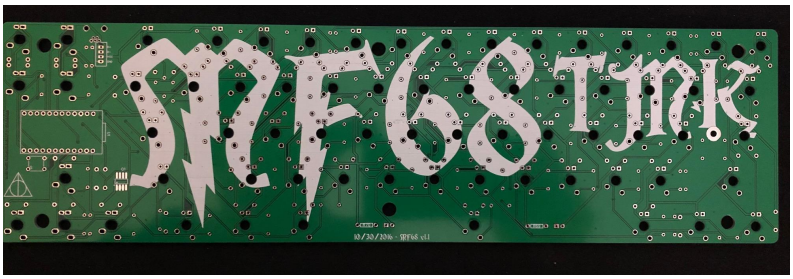
Jeremy Ahn
4/8/2020
Microprocessors
Lab Project Final Report

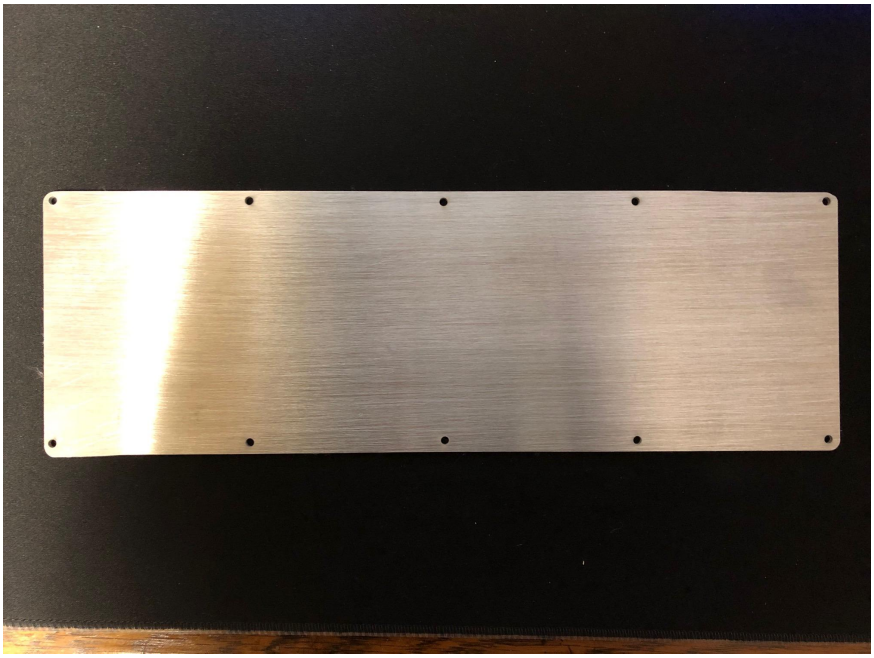
68 key Custom Mechanical Keyboard Final Report

Goal: To build a 68-key custom mechanical keyboard with an ‘Open Sandwich’ custom aluminum case using the Arduino Pro Micro as the main microcontroller.

Size - 68%
Materials

Arduino Pro Micro	
pcb	Easyeda, $29.01  
keyswitches	Gateron brown $34.99 https://www.pcgamingrace.com/products/gateron-switches?_pos=3&_sid=68301667d&_ss=r&variant=20340749059

Stabilisers	$15 https://flashquark.com/product/genuine-cherry-plate-mounted-stabilizers/?attribute_size=6.25u
Aluminum case (called)	Total: $44.07 Order sheets from McMaster-Carr: Multipurpose 6061 Aluminum Sheet, 0.05” Thick, 6”x24” - $13.50 Multipurpose 6061 Aluminum Sheet, 0.09” Thick, 6”x24” - $18.99 Cut by Steve Coan, Create Cut Invent/Closed Circuit Innovations Inc. (Free) 

	
Keycaps	$26 https://www.amazon.com/gp/product/B07JN4YK82/ref=ppx_od_dt_b_asin_title_s00?ie=UTF8&psc=1 
M3 screws	$5.55 - 100 pcs https://www.amazon.com/gp/product/B01B1OD62C/ref=ppx_yo_dt_b_asin_title_o04_s01?ie=UTF8&psc=1

	
M3 standoffs	$4.44 - 30 pcs https://www.amazon.com/gp/product/B00XBFYHSC/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1 
Push button	10 pcs - $2.23 https://www.ebay.com/itm/10Pcs-6x6x5mm-PCB-Momentary-Tactile-Tact-Push-Button-Switch-4-Pin-DIP-Micro-Mini/323481083510?_trkparms=ispr%3D1&hash=item4b50f99e76:g:7Y0AAOSw5dhbtR7F&enc=AQAEAAACUBPxNw%2BVj6nta7CKEs3N0qW6EqBSB679PCaQJqd2xs3dtRupC%2FU%2FjCstjhq9dGPppkRQIFlKltbCI1yzSicoszc1TeXWULqJQxnY3hJbk1g1EIluZKR8bsocWV594ExTM8EUMSkS68psBkq04W%2BALNPQN7PXXUJebJS%2BUxfomHQq%2BtCtfO1zLm5FM7XvCQhCIlqCAOzqgxDe15BkAmIXqpuTuphugUx2Uj%2FOUQSZ06DQtMOuWucDlRS21FIJFj4vSzG33kRfcMKlS%2B6E1iyS%2B9uznJCQyX9ZCmHkuIc1xp8hdQJtqTFWB5qQwXanP1rvayKA59cW3UUAg4avgdlpMh9b2y%2Fh4PYEugPG0s0DpKWNu%2BP50x84kXKpcfX5g7LOp%2FBTI6U9vRZNjhNd59wkrBVxsocLTO%2BCzTr8boyMewgagnQCX3R0XJq6jI%2BDpik7QLd0ftmtbphOntz1kAgqzkOZbNqomruOSuR65xNeQUw%2B9n%2BC1OlwVKkzcJXBGOfQjlrQJlN4938V%2BDrItwuAg4Eb9CCfCpYPHeQHnw5Ny0upwXcXq2YRbGqyZb9tdf23RtVUMj1taEfPdq93nl%2FB%2BpR2ReSeB09tt7AsgmiN6d7U%2FOU49DPQ6B6sNVvHH6b

	IT5rt9co%2F3QaNCxXuhavU%2B0dBsq9C3Gw7fWbFyFQAZ1hambkO09yQMtSbWfNotrxvkxO1RSH7U0u9C7eeajZDg2TUHJVyh6tM%2BtkM74YbR4vUGPGOmmh3Ao%2B%2FL9Sxh9xkKJSLAGNx7018dSP4G2m2kG4%3D&checksum=323481083510bc2e3251c3a742c2a570bda4ec7fee09 
diodes	$4.99 - 100 pcs https://www.amazon.com/gp/product/B06XB1R2NK/ref=ppx_yo_dt_b_asin_title_o03_s00?ie=UTF8&psc=1 
Micro usb to usb cable	$8.99 - 2 pack, 6.6 ft https://www.amazon.com/gp/product/B078QHT2KY/ref=ppx_od_dt_b_asin_image_s00?ie=UTF8&psc=1

	
Soldering iron	Already have
software	QMK firmware ● Avr-gcc ● Avrdude ● CrossPack by ObDev

Total cost: $175.27

https://68keys.io/

For this project, I wanted to build a keyboard with my knowledge of microcontrollers and the Arduino Pro Micro. Since I have already had experience using the Pro Micro, I wanted to find a way to use that specific microcontroller so I wouldn't have to worry about learning a new system. Luckily, I found a guide that uses the Arduino Pro Micro to make a custom 68-key keyboard. While it is not among the common class sizes of keyboards, it uses the Arduino Pro Micro as the foundation of the keyboard, which is why I decided to use it as the basis for my project. The website [1] 68keys.io goes through a fairly detailed guide on how to use the Arduino Pro Micro to make a custom keyboard that uses a custom built aluminum case and PCB.

The aluminum case was one of the aspects of this guide that appealed to me because of its simplicity. The guide calls for an open sandwich aluminum case, which essentially uses only two plates that are separated by screws and spacers, as shown in Figure 1.

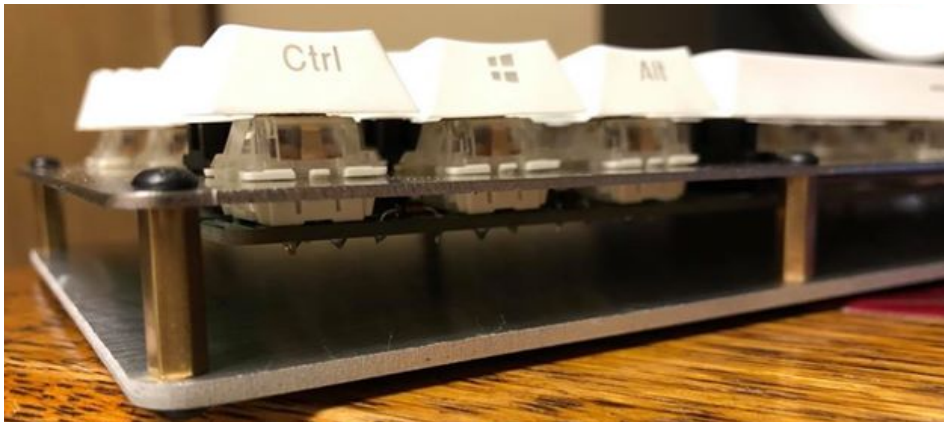


Figure 1 - Open sandwich case. Top plate holds switches. Bottom plate adds support

The top plate holds the switches, while the bottom plate adds support. To make these custom plates, I had to use a specific keyboard layout that used the amount of keys I was going for. The editor and raw data are shown in Figure 2.

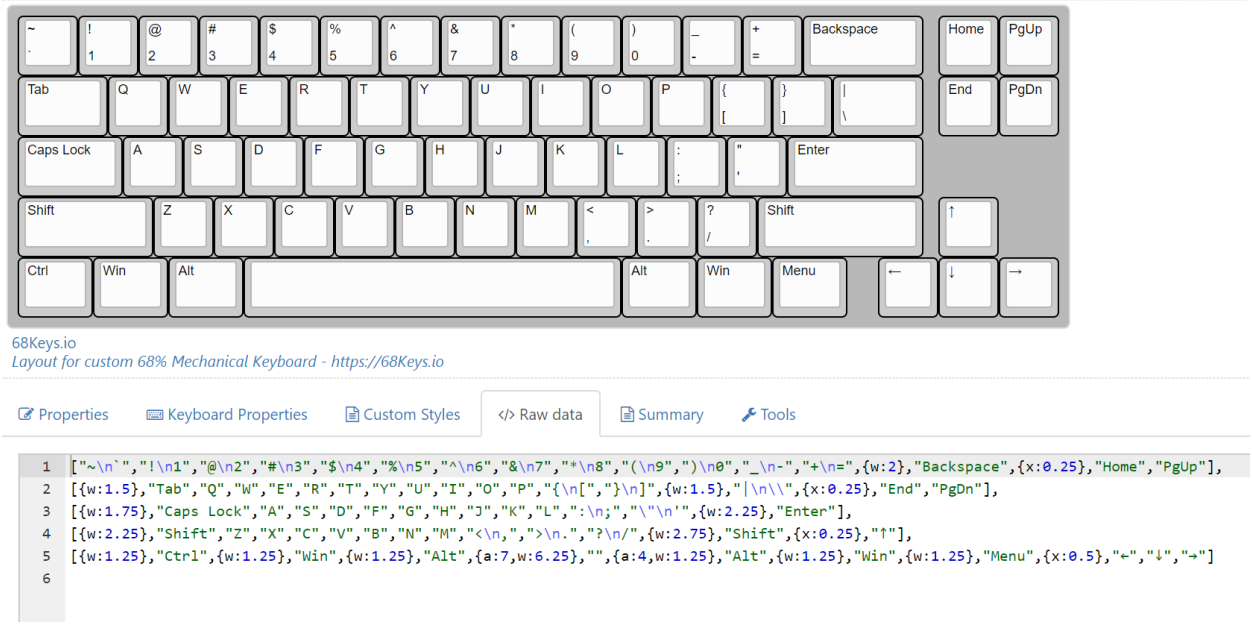


Figure 2 - Keyboard layout editor and raw data. Keyboard Layout Editor [2]

After obtaining the raw data from this editor, I was able to use another custom website to make the plate and case cutout files [Figure 2] that I would be able to send to company or service for them to cut out the designs from aluminum.

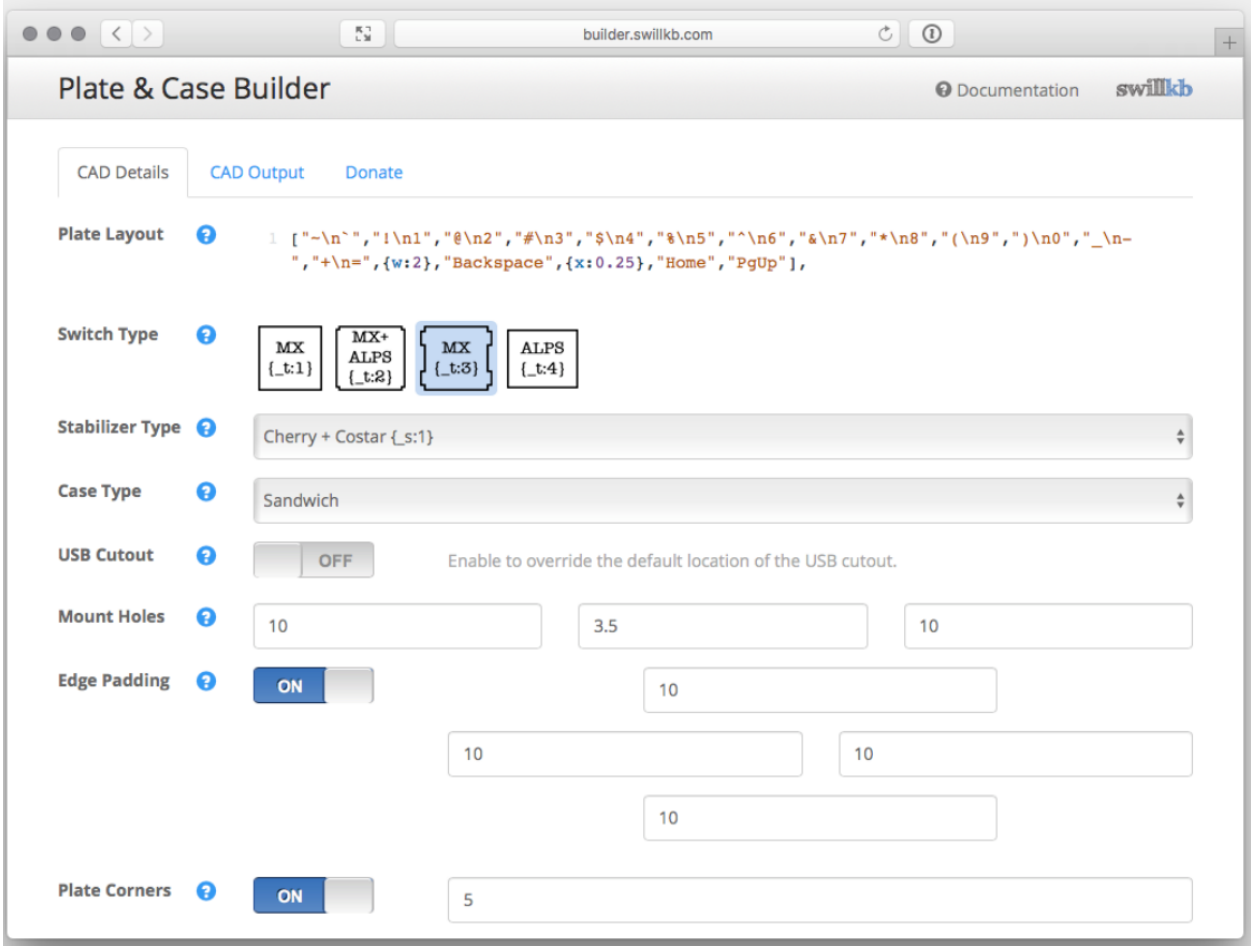


Figure 2 - Plate and Case Builder. Provided dxf files for CNC cutouts. Plate & Case Builder [3]

Using the Plate and Case builder, I was able to get dxf files that I would then be able to send to a company or service where they would be able to cut out the designs from aluminum. I was not too worried about using the given data and files that was provided on the guide because designing a case or layout was not the primary goal for this project. I wanted to focus more on the electronics of the keyboard and therefore welcomed the pre-made designs.

The dxf files looked like the following Figures 3 and 4.

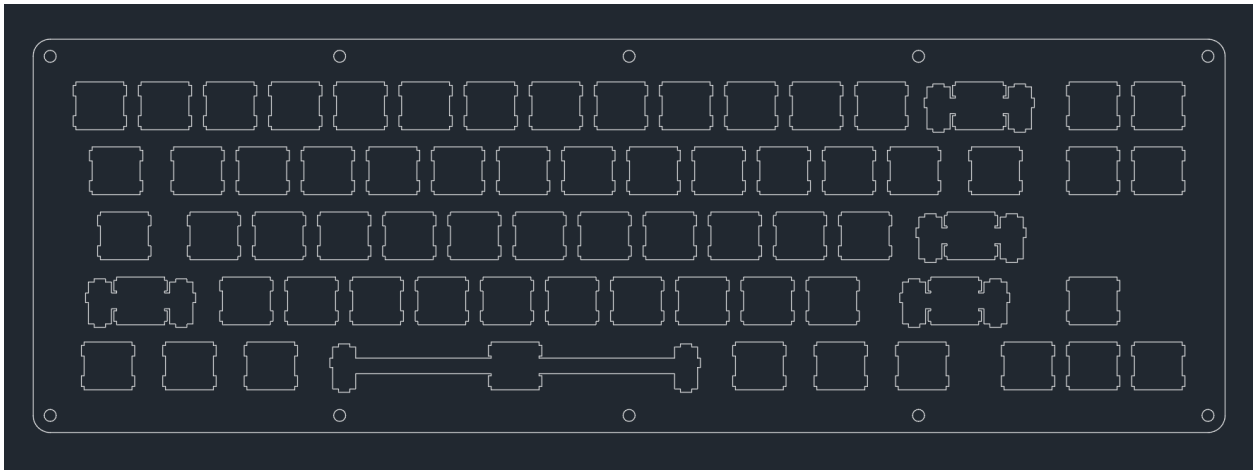


Figure 3 - Switch layer

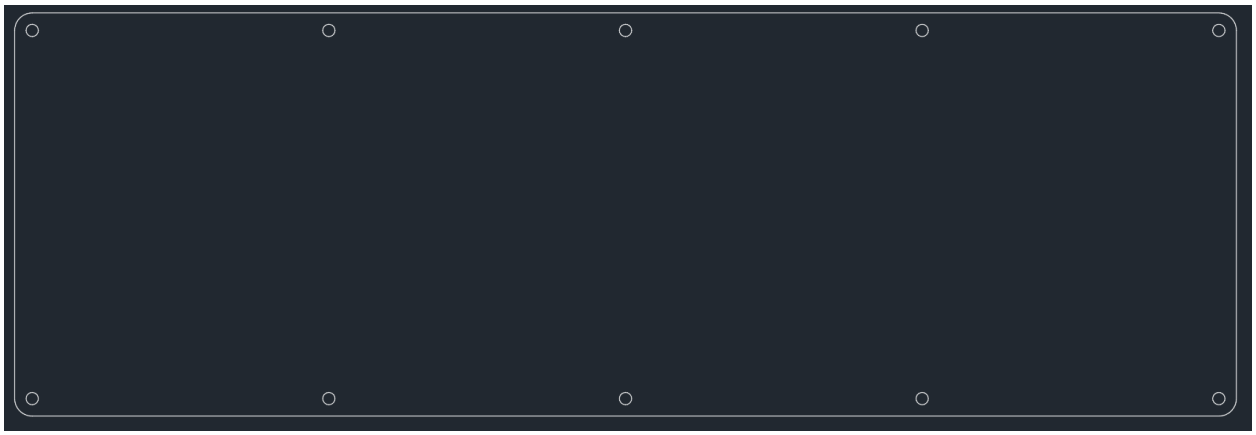


Figure 4 - Bottom Layer

The person I found to cut out the sheets for me was a local mechanic who would cut out the sheets for a flat rate of $100, given I provided the original aluminum sheets [4 website: createcutinvent.com]. I found aluminum sheets on Mcmaster-carr [5] for fairly cheap rates, and was able to send the sheets and .dxf files and receive the finished layers fairly quickly. I knew aluminum keyboard covers would be expensive, so I was intrigued by the cheapness of the open sandwich case that used a minimal amount of aluminum but still provided a metal frame. The aluminum sheets themselves cost about $40 in total, but amazingly the man I asked to cut out the sheets very kindly did them for free. [1]When I got my case layers, I was even told that on the top switch layer, some of the smaller pieces had broken when he was cleaning, so he used a stainless steel sheet to cut out a new part and gave that to me again for free. Essentially, I was able to get a stainless steel switch layer and aluminum bottom layer for my custom keyboard case [Figures 5 and 6]

[1] His name is Steve Coan, and his website is https://www.createcutinvent.com/about.html. I am extremely grateful for his services and kindness, especially during these times of pandemic. I would highly recommend anyone to check him out for any sort of laser cutting services.

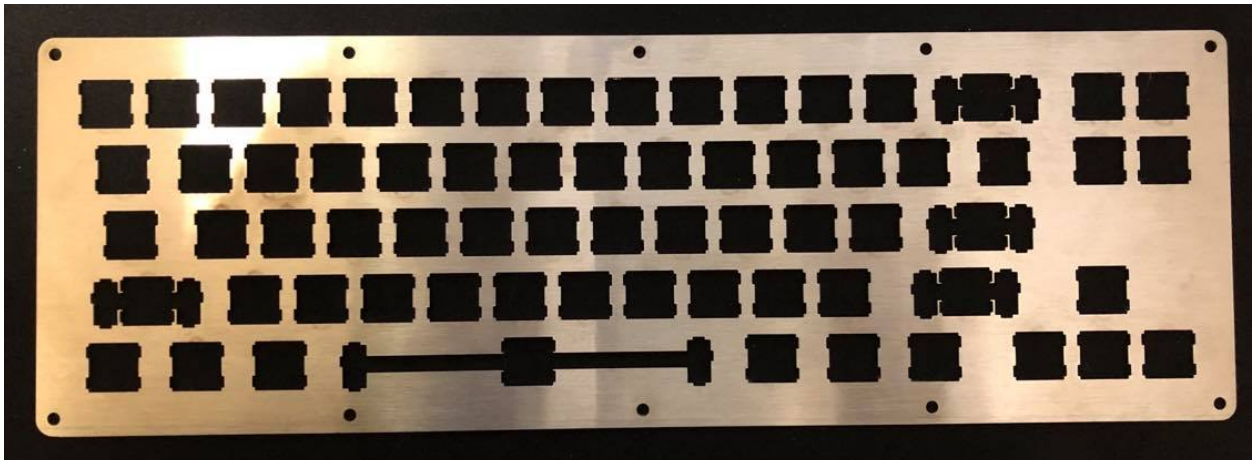


Figure 5 - Top switch layer, stainless steel

Figure 6 - Bottom layer, aluminum 6061

The only issue when getting the parts for the case was that the dxf files were originally in mm and I had to resize them to inches in AutoCAD. After that, however, the only requirements for assembling the case itself were M3 screws, 8mm and M3 standoffs, 15mm. For switches and stabilizers, I got Gateron brown switches, and cherry mx plate-mounted stabilizers. Figure 7 shows the case with the switches and stabilizers installed.

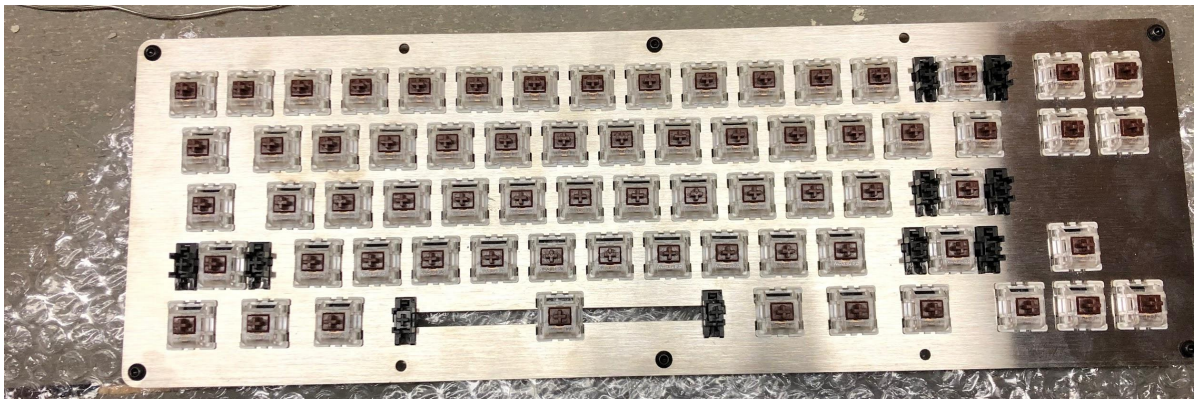


Figure 7 - Top switch plate with switches and stabilizers.

For the PCB, I wanted to customize it as much as I could and make my own schematics. However, the files on the website guide were pre-made and not that easy to customize. I especially didn't want to ruin anything for the final PCB board, and there was also the question of time and resources being home because of COVID-19. So, I used the pre-made custom PCB design from the website and ordered a set of 5 (minimum amount) from EasyEDA.com [6]. The PCB itself looked like Figures 8 and 9.

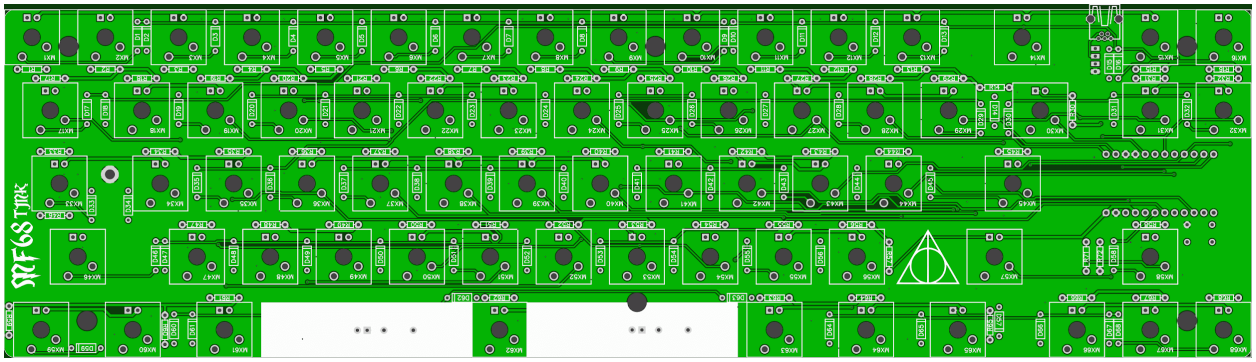


Figure 8 - PCB side 1

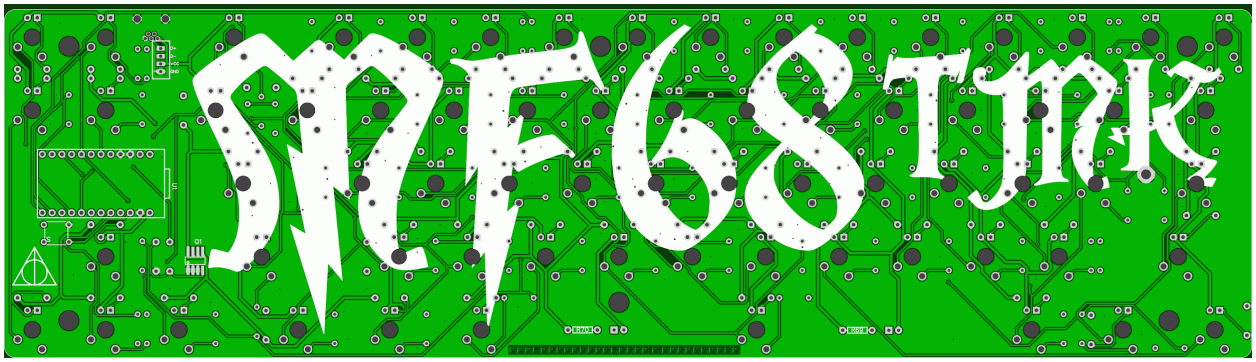


Figure 9 - PCB side 2

These images and design were taken from the github user di0ib's database [7]. They have the words MF68 because it was originally designed to be a replacement PCB board for the Magicforce 68 keyboard. The mapping of the PCB board and schematic can be found in the following table [Table 1]

Atmega	Arduino pin
Columns	
D3	Digital Pin 1
D2	Digital Pin 0
D1	Digital Pin 2
D0	Digital Pin 3

C6	Digital pin 5
D7	Digital pin 6
E6	Digital pin 7
B4	Digital pin 8
row	
B6	Digital pin 10
B2	MOSI (Leonardo: 11; Pro Micro: 16)
B3	MISO (Leonardo: 12; Pro Micro: 14)
B1	SCK (Leonardo: 13; Pro Micro: 15)
F7	Analog 0
F6	Analog 1
F5	Analog 2
F4	Analog 3

Table 1 - pin mapping of PCB to Arduino
col: D3 D2 D1 D0 D4 C6 D7 E6 B4
row: B6 B2 B3 B1 F7 F6 F5 F4

The accompanying circuit diagram [Figure 10] gives a schematic for the Arduino wiring (shown is the Arduino Leonardo, equivalent microprocessor for the Arduino Pro Micro), as well as an example of the wiring that is involved in the keyswitches.

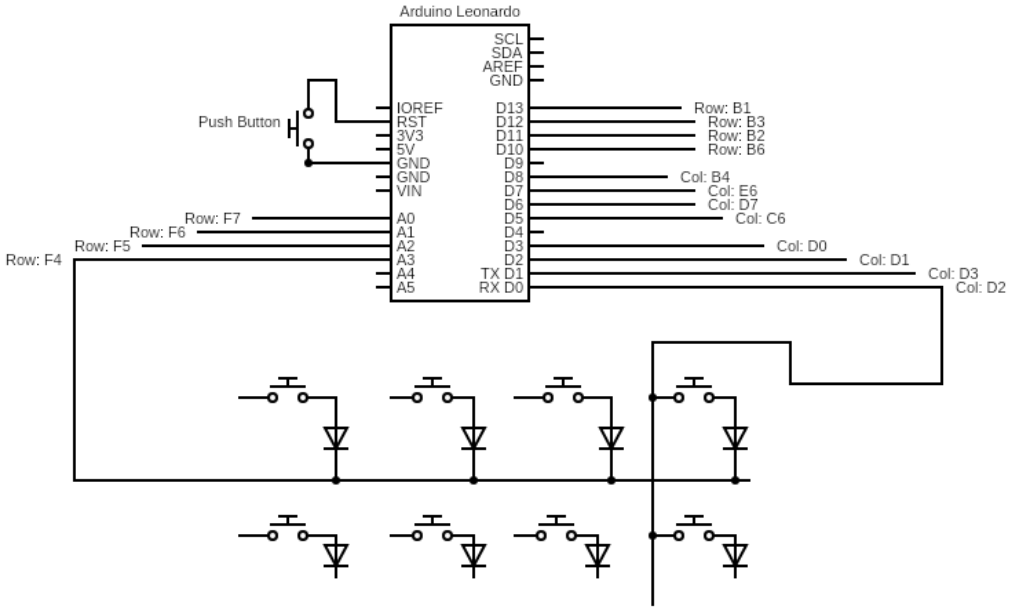


Figure 10 - Circuit diagram of Arduino

Essentially, the keyboard is like a larger keypad, where the microprocessor will check every column and every row to see which one has been pressed and carry out instructions based on the information gathered. As the Arduino checks each row and column, either the row or column is raised HIGH or LOW, and the combination of a pressed key switch will send a signal through to the microprocessor to let it know which key has been pressed.

```
+ #ifndef MF68_H
+ #define MF68_H
+
+ #include "quantum.h"
+
+ #define KEYMAP( \
+     K00, K01, K02, K03, K04, K05, K06, K07, K08, K10, K11, K12, K13, K14,     K15, K16, \
+     K17, K18, K20, K21, K22, K23, K24, K25, K26, K27, K28, K30, K31, K32,     K33, K34, \
+     K35,  K36, K37, K38, K40, K41, K42, K43, K44, K45, K46, K47,     K48, \
+     K50,   K51, K52, K53, K54, K55, K56, K57, K58, K60, K61,    K62,          K63, \
+     K64,  K65,  K66,                 K67,              K68,  K70,  K71,  K72, K73, K74 \
+ ) { \
+     { K00, K01, K02, K03, K04, K05, K06, K07, K08 }, \
+     { K10, K11, K12, K13, K14, K15, K16, K17, K18 }, \
+     { K20, K21, K22, K23, K24, K25, K26, K27, K28 }, \
+     { K30, K31, K32, K33, K34, K35, K36, K37, K38 }, \
+     { K40, K41, K42, K43, K44, K45, K46, K47, K48 }, \
+     { K50, K51, K52, K53, K54, K55, K56, K57, K58 }, \
+     { K60, K61, K62, K63, K64, K65, K66, K67, K68 }, \
+     { K70, K71, K72, K73, K74 } \
+ }
+
+ #define KC_KEYMAP( \
+     K00, K01, K02, K03, K04, K05, K06, K07, K08, K10, K11, K12, K13, K14,     K15, K16, \
+     K17, K18, K20, K21, K22, K23, K24, K25, K26, K27, K28, K30, K31, K32,     K33, K34, \
+     K35,  K36, K37, K38, K40, K41, K42, K43, K44, K45, K46, K47,     K48, \
+     K50,   K51, K52, K53, K54, K55, K56, K57, K58, K60, K61,    K62,          K63, \
+     K64,  K65,  K66,                 K67,              K68,  K70,  K71,  K72, K73, K74 \
+ ) KEYMAP( \
+     KC_##K00, KC_##K01, KC_##K02, KC_##K03, KC_##K04, KC_##K05, KC_##K06, KC_##K07, KC_##K08, \
+     KC_##K10, KC_##K11, KC_##K12, KC_##K13, KC_##K14, KC_##K15, KC_##K16, KC_##K17, KC_##K18, \
+     KC_##K20, KC_##K21, KC_##K22, KC_##K23, KC_##K24, KC_##K25, KC_##K26, KC_##K27, KC_##K28, \
+     KC_##K30, KC_##K31, KC_##K32, KC_##K33, KC_##K34, KC_##K35, KC_##K36, KC_##K37, KC_##K38, \
+     KC_##K40, KC_##K41, KC_##K42, KC_##K43, KC_##K44, KC_##K45, KC_##K46, KC_##K47, KC_##K48, \
+     KC_##K50, KC_##K51, KC_##K52, KC_##K53, KC_##K54, KC_##K55, KC_##K56, KC_##K57, KC_##K58, \
+     KC_##K60, KC_##K61, KC_##K62, KC_##K63, KC_##K64, KC_##K65, KC_##K66, KC_##K67, KC_##K68, \
+     KC_##K70, KC_##K71, KC_##K72, KC_##K73, KC_##K74 \
+ )
+
+ #endif
```

https://github.com/qmk/qmk_firmware/pull/1698/files

Once I had gotten all the parts and PCB, I was able to begin soldering. Each key required a 1N4148 diode, and the Arduino also needed a push button to reset it when I uploaded my code. After I finished soldering, I was ready for the code.

Finally, for the code, I used QMK firmware to program the arduino. For much of the coding, I used the Terminal application on my Macbook Pro, and had to first install homebrew [8]. Once that was done, I found guides on basic qmk setup [9], building keymaps[10], and customizing keymap settings with keymap layers [11], basic keycodes [12], and flashing the qmk to the arduino [13]. Once I had figured all of that out and set up a basic qmk environment for me to customize and flash a keymap setting onto my Arduino, I set about customizing a keymap setting that I wanted my keyboard to have. The code for the default, basic, single-layer keymap

setting from Github [14] looked as follows [Figure 11]

```
#include QMK_KEYBOARD_H

const uint16_t PROGMEM keymaps[][MATRIX_ROWS][MATRIX_COLS] = {
  [0] = LAYOUT_68_ansi(
    KC_GRV, KC_1, KC_2, KC_3, KC_4, KC_5, KC_6, KC_7, KC_8, KC_9, KC_0, KC_MINS, KC_EQL, KC_BSPC, KC_ESC, KC_PGUP,
    KC_TAB, KC_Q, KC_W, KC_E, KC_R, KC_T, KC_Y, KC_U, KC_I, KC_O, KC_P, KC_LBRC, KC_RBRC, KC_BSLS, KC_DEL, KC_PGDN,
    KC_CAPSLOCK, KC_A, KC_S, KC_D, KC_F, KC_G, KC_H, KC_J, KC_K, KC_L, KC_SCLN, KC_QUOT, KC_ENT,
    KC_LSFT, KC_Z, KC_X, KC_C, KC_V, KC_B, KC_N, KC_M, KC_COMM, KC_DOT, KC_SLSH, KC_RSFT, KC_UP,
    KC_LCTL, KC_LALT, KC_LGUI, KC_SPC, KC_RGUI, KC_RALT, KC_RCTL, KC_LEFT, KC_DOWN, KC_RGHT
  )
};
```

Figure 11 - Basic keymap settings for 68keys keyboard.

The code itself was fairly easy to understand, because the names for what key you want each switch to trigger was intuitive. For example, KC_1 relates to the 1 key, and KC_ENT is "Enter", and KC_LSFT is "Left shift". The github database also came with configuration settings [Figure 12].

```
#pragma once

/* USB Device descriptor parameter */
#undef VENDOR_ID
#undef PRODUCT_ID
#undef DEVICE_VER
#undef MANUFACTURER
#undef PRODUCT
#undef DESCRIPTION
#define VENDOR_ID 0xFEED
#define PRODUCT_ID 0x0A0C
#define DEVICE_VER 0x0068
#define MANUFACTURER 68Keys.io
#define PRODUCT The 68Keys.io Keyboard
#define DESCRIPTION A 68 keys mechanical keyboard
```

Figure 12 - 68keys configuration settings

The guide on 68keys.io essentially switched the "Esc" button to be on the right side, but I wanted it to be on the left side. To do this among other customizations, I wrote the following code in Figure 13.

```
#include QMK_KEYBOARD_H

const uint16_t PROGMEM keymaps[][MATRIX_ROWS][MATRIX_COLS] = {
  [0] = LAYOUT_68_ansi(
    KC_GESC, KC_1, KC_2, KC_3, KC_4, KC_5, KC_6, KC_7, KC_8, KC_9, KC_0, KC_MINS, KC_EQL, KC_BSPC, KC_HOME, KC_PGUP,
    KC_TAB, KC_Q, KC_W, KC_E, KC_R, KC_T, KC_Y, KC_U, KC_I, KC_O, KC_P, KC_LBRC, KC_RBRC, KC_BSLS, KC_DEL, KC_PGDN,
    KC_CAPSLOCK, KC_A, KC_S, KC_D, KC_F, KC_G, KC_H, KC_J, KC_K, KC_L, KC_SCLN, KC_QUOT, KC_ENT,
    KC_LSFT, KC_Z, KC_X, KC_C, KC_V, KC_B, KC_N, KC_M, KC_COMM, KC_DOT, KC_SLSH, KC_RSFT, KC_UP,
    KC_LCTL, KC_LGUI, KC_LALT, KC_SPC, KC_RALT, MO(1), KC_RCTL, KC_LEFT, KC_DOWN, KC_RGHT),
  [1] = LAYOUT_68_ansi(
    KC_TRNS, KC_F1, KC_F2, KC_F3, KC_F4, KC_F5, KC_F6, KC_F7, KC_F8, KC_F9, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS,
    KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS,
    KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS,
    KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS,
    KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, MO(1), KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS),

};
```

Figure 13 - Customized keymapping

I actually found a command that is called KC_GESC, which essentially sets the most upper left-hand switch on my keyboard to be a special key that when normally pressed sends "Esc". However, when pressed with Shift or the windows key, it will become the " ` " or "~" key and send those accordingly. I also wanted to have a second layer for the keymap where I could

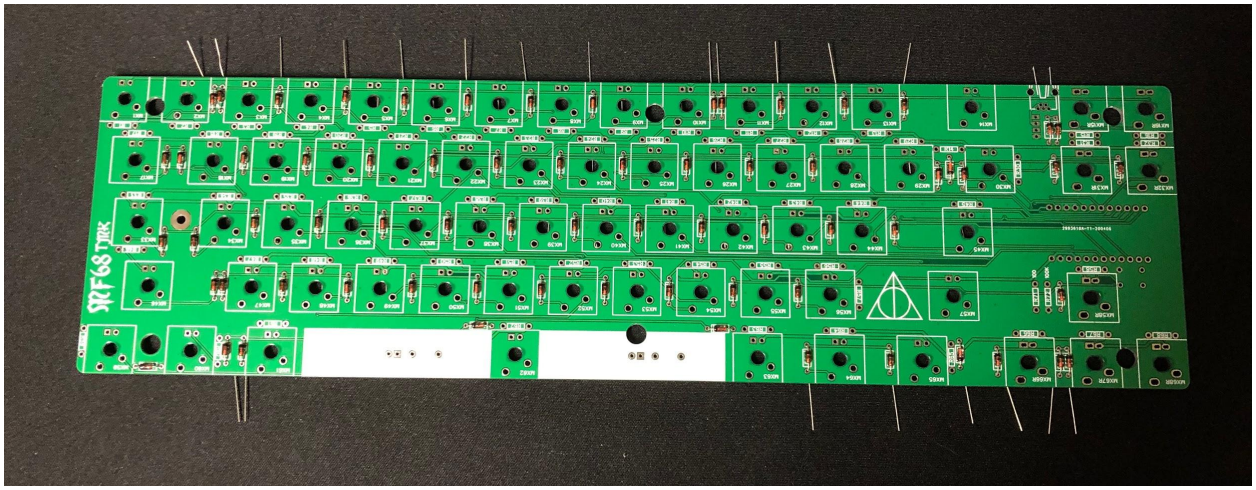
further customize the keymapping. To do this, I set the key between the right Alt and right Ctrl buttons as an FN key, so that when pressed and held the number keys would buffer F1, F2, etc. To do this in the code, that switch had to be set as MO(1), which ‘momentarily’ turns on the layer within the parenthesis, in this case Layer 1. In this new layer, the KC_1, KC_2, etc. keys are turned to KC_F1, KC_F2, etc. All the rest of the keys are replaced with KC_TRNS, which just causes the code to fall back to the layer before when pressed.

After many iterations of trial and error getting my code to work with the MacOS Terminal, I was finally able to upload and flash my custom qmk firmware onto my Arduino. Thankfully, I didn’t destroy or burn anything while soldering, so my keyboard worked on the first try. The switches worked as they should, the layered keymapping with the FN also worked, and the sound of the switches was extremely satisfying. The final product looked like this: [Figure 14 and 15]

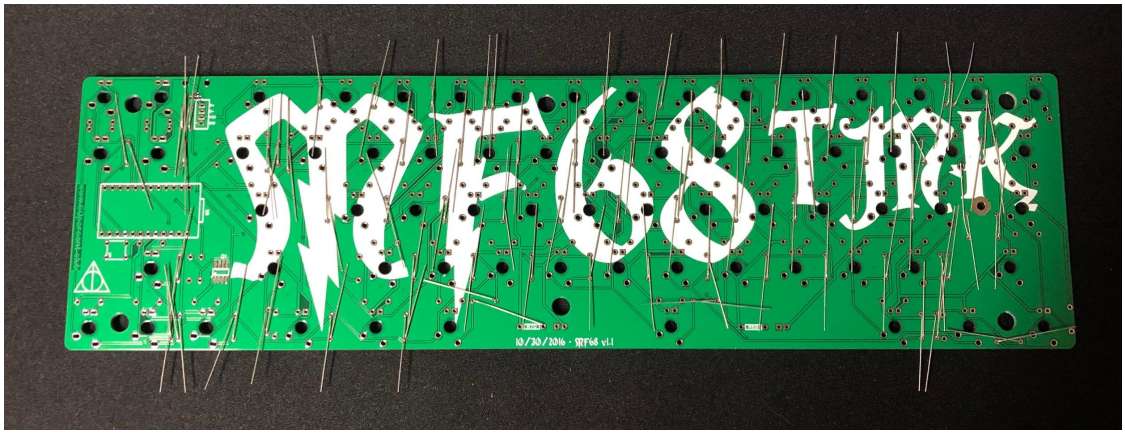
Figure 14 - Final working keyboard with white keycaps

Figure 15 - Final working keyboard, side view.

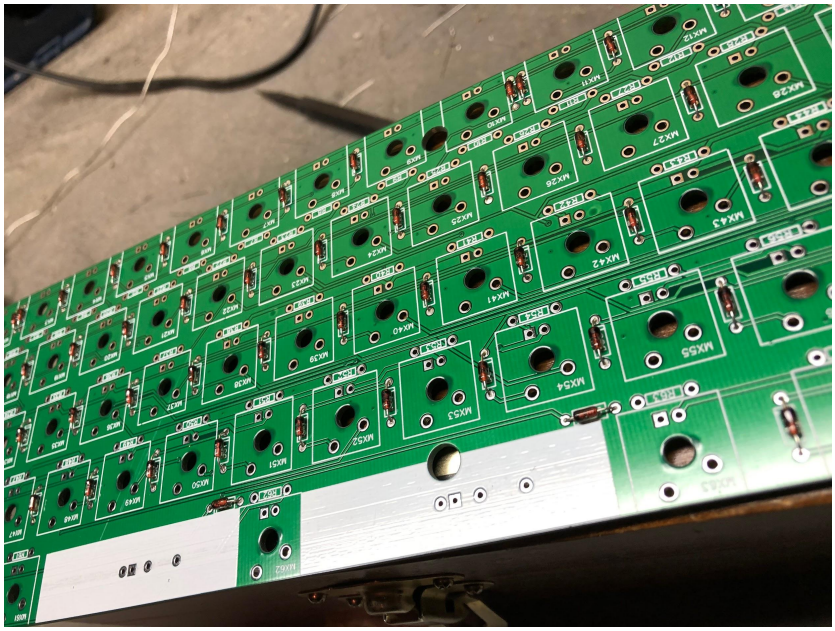
Additional progress pictures

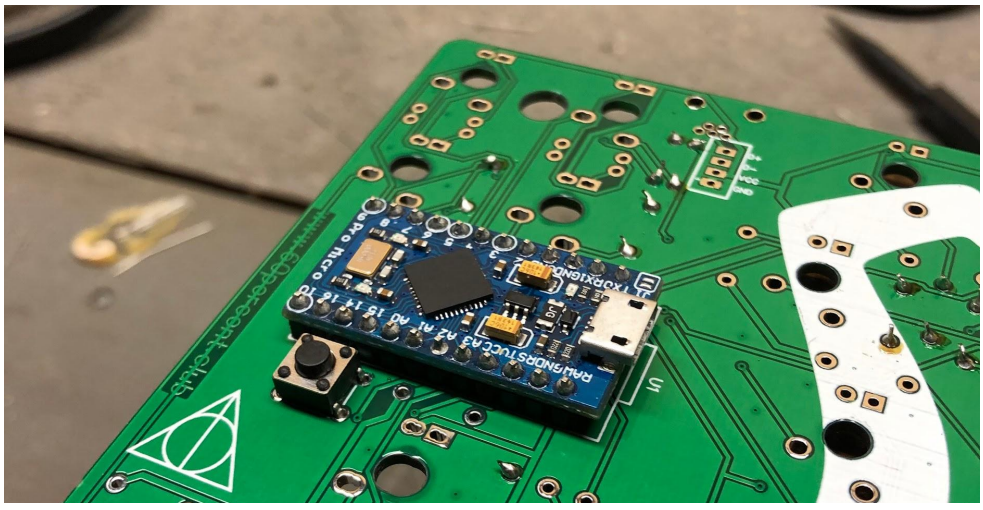


Diodes installed, not yet soldered

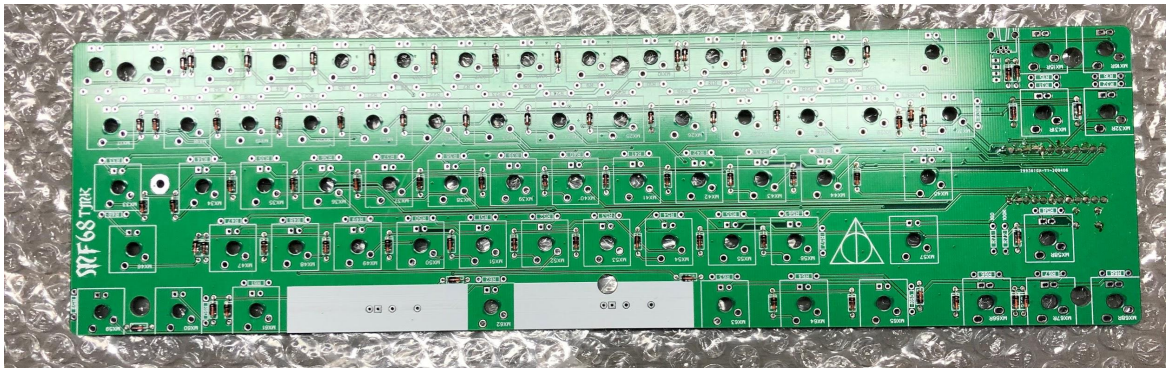


Diodes installed, not yet soldered, back view

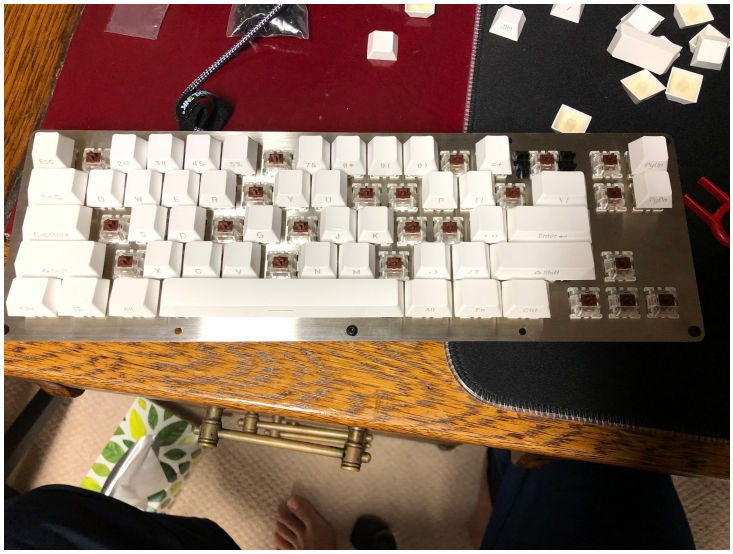




Arduino and button soldered in



All diodes soldered



Almost done

Sources

[1] Sbstjn, “Custom 68% Mechanical Keyboard DIY Guide,” *68Keys.io*. [Online]. Available: https://68keys.io/. [Accessed: 01-Apr-2020].

[2] http://www.keyboard-layout-editor.com/##@_backcolor=%23b8b8b8b8&name=68Keys.io&author=Layout%20for%20custom%2068%25%20Mechanical%20Keyboard%20-%20https%2F:%2F%2F%2F%2F68Keys.io&switchMount=cherry&plate:true%3B&@=~%0A%60&=!%0A1&=%2F@%0A2&=%23%0A3&=$%0A4&=%25%0A5&=%5E%0A6&=%2F&%0A7&=*%0A8&=(%0A9&=)%0A0&=%2F_%0A-&=+%0A%2F=&_w:2%3B&=Backspace&_x:0.25%3B&=Home&=PgUp%3B&@_w:1.5%3B&=Tab&=Q&=W&=E&=R&=T&=Y&=U&=I&=O&=P&=%7B%0A%5B&=%7D%0A%5D&_w:1.5%3B&=%7C%0A%5C&_x:0.25%3B&=End&=PgDn%3B&@_w:1.75%3B&=Caps%20Lock&=A&=S&=D&=F&=G&=H&=J&=K&=L&=%2F:%0A%2F%3B&=%22%0A'&_w:2.25%3B&=Enter%3B&@_w:2.25%3B&=Shift&=Z&=X&=C&=V&=B&=N&=M&=%3C%0A,&=%3E%0A.&=%

3F%0A%2F%2F&_w:2.75%3B&=Shift&_x:0.25%3B&=%E2%86%91%3B&@_w:1.25%3B&=Ctrl&_w:1.25%3B&=Win&_w:1.25%3B&=Alt&_a:7&w:6.25%3B&=&_a:4&w:1.25%3B&=Alt&_w:1.25%3B&=Win&_w:1.25%3B&=Menu&_x:0.5%3B&=%E2%86%90&=%E2%86%93&=%E2%86%92
[3] http://builder.swillkb.com/
[4] https://www.createcutinvent.com/home.html
[5] https://www.mcmaster.com/standard-aluminum-sheets/
[6] https://easyeda.com/
[7] https://github.com/di0ib/tmk_keyboard/tree/master/keyboard/mf68
[8] Homebrew - https://brew.sh/
[9] Set up qmk - https://docs.qmk.fm/#/newbs_getting_started?id=_1-download-software
[10] Building keymap and customizing qmk - https://docs.qmk.fm/#/newbs_building_firmware
[11] Keymap layers - https://docs.qmk.fm/#/keymap?id=layers-and-keymaps
[12] Basic keycodes - https://docs.qmk.fm/#/keycodes?id=grave-escape
[13] Flashing qmk to arduino - https://docs.qmk.fm/#/newbs_flashing
[14] Github basic 68keys keymap code - https://github.com/qmk/qmk_firmware/tree/master/keyboards/40percentclub/mf68/keymaps/68keys

Additional sources
[ ]“An Easier and PowerfulOnline PCB Design Tool,” *EasyEDA*. [Online]. Available: https://easyeda.com/. [Accessed: 01-Apr-2020]. - PCB and schematic design
[] http://www.keyboard-layout-editor.com/##@_backcolor=%23b8b8b8&name=68Keys.io&author=Layout%20for%20custom%2068%25%20Mechanical%20Keyboard%20-%20https%2F:%2F%2F%2F%2F68Keys.io&switchMount=cherry&plate:true%3B&@=~%0A%60&=!%0A1&=%2F@%0A2&=%23%0A3&=$%0A4&=%25%0A5&=%5E%0A6&=%2F&%0A7&=*%0A8&=(%0A9&=)%0A0&=%2F_%0A-&=+%0A%2F=&_w:2%3B&=Backspace&_x:0.25%3B&=Home&=PgUp%3B&@_w:1.5%3B&=Tab&=Q&=W&=E&=R&=T&=Y&=U&=I&=O&=P&=%7B%0A%5B&=%7D%0A%5D&_w:1.5%3B&=%7C%0A%5C&_x:0.25%3B&=End&=PgDn%3B&@_w:1.75%3B&=Caps%20Lock&=A&=S&=D&=F&=G&=H&=J&=K&=L&=%2F:%0A%2F%3B&=%22%0A'&_w:2.25%3B&=Enter%3B&@_w:2.25%3B&=Shift&=Z&=X&=C&=V&=B&=N&=M&=%3C%0A,&=%3E%0A.&=%3F%0A%2F%2F&_w:2.75%3B&=Shift&_x:0.25%3B&=%E2%86%91%3B&@_w:1.25%3B&=Ctrl&_w:1.25%3B&=Win&_w:1.25%3B&=Alt&_a:7&w:6.25%3B&=&_a:4&w:1.25%3B&=Alt&_w:1.25%3B&=Win&_w:1.25%3B&=Menu&_x:0.5%3B&=%E2%86%90&=%E2%86%93&=%E2%86%92 - keyboard layout editor
[] http://blog.komar.be/how-to-make-a-keyboard-the-matrix/
[] http://pcbheaven.com/wikipages/How_Key_Matrices_Works/