RootKB

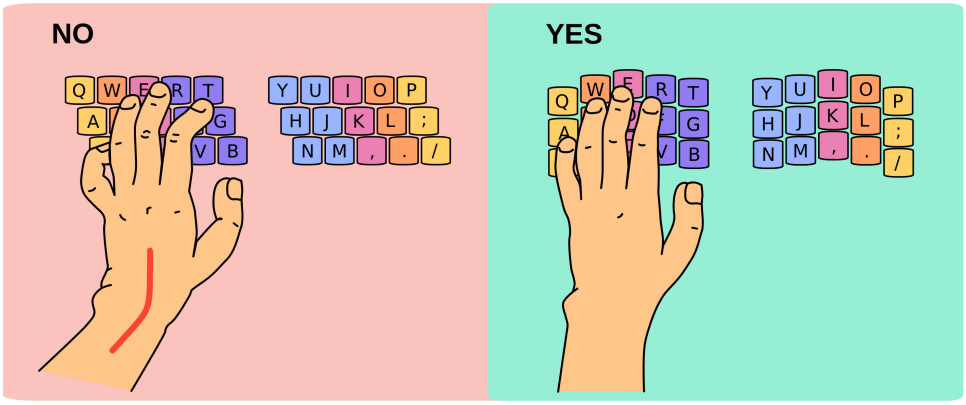
Name: Matei-Ștefan Ionescu
Group: 333CA

Introduction

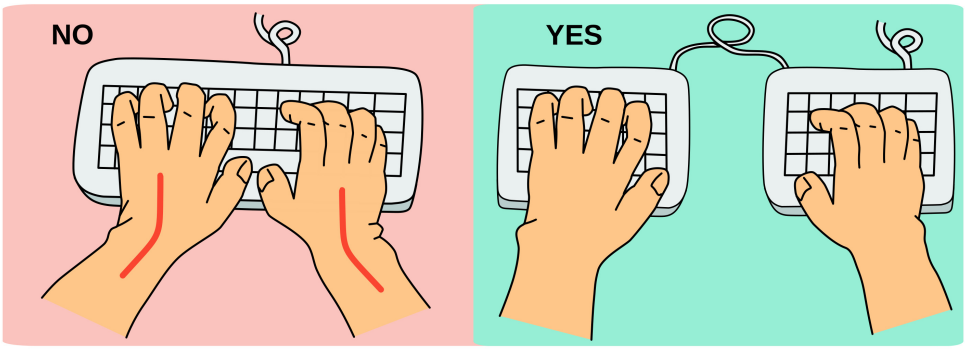
RootKB is a 44 key split mechanical keyboard, featuring hot-swappable switches, a columnar layout with an ergonomically placed 4 key thumb cluster, per-key RGB lighting and an OLED screen on each side.

As a computer science student, I am spending a big part of my day typing on my keyboard. So, I was always interested in how I could make this activity more enjoyable and efficient. I started researching keyboard ergonomics and I found that there are many problems with regular keyboards.

Firstly, we are using the same layout designed in the early 1870s for typewriters. At that time, having staggered rows allowed the typebars to align more efficiently, reducing the likelihood of jams. Now, since computer keyboards don't suffer from the same issue, having staggered rows is just inefficient since it requires your fingers to travel in an unnatural and uneven way. In order to fix this, I am using a column staggered design for my keyboard. This way the fingers will rest in a more natural position and travel less, while keeping an even travel distance between the fingers of my left hand and right hand.



A second problem with standard keyboards is the fact that your hands are placed too close together on the keyboard. This puts your wrists in an unnatural position, potentially leading to discomfort or repetitive strain injuries like carpal tunnel syndrome. This issue is fixed by splitting the keyboard, allowing each half to be placed in a more natural position for your hands.



The third problem is that your strongest fingers, the thumbs are only used to press one key, the space bar. To make my keyboard more efficient and comfortable I will be using a 4 key thumb cluster for each side. So, the thumbs will be pressing 8 keys instead of 1.

A fourth inefficiency that arises from the regular keyboard design is stretching our fingers a long way in order to press certain keys. To fix this, we will use 6 columns and 3 rows aside from the thumb cluster, so our fingers will easily reach each key. At this point you might be wondering where the functions and numbers rows will go, and how you will be able to access special characters. The answer is layers. You might not know it, but you are already familiar with layers. Every time you use the Shift key you are changing the layer of your keyboard to access different symbols, like uppercase characters. There will be buttons in the thumb clusters that when held down will change the layer of the keyboard. So, you will be able to access all the characters you need by holding down one of these buttons.

Discovering all these problems with standard keyboards and inspired by other open-source projects like the Corne, Lily58 and Piantor, my goal is to design and build my own ergonomic keyboard.

Aside from the ergonomic design, the keyboard will feature per-key RGB lighting, allowing users to choose their favorite colors. Additionally, because the switch type preferences differ from person to person, the keyboard is hot-swappable, allowing users to use their favorite switches for the best typing experience. An OLED screen will also be placed on each side of the board, allowing users to display an image or info about the board, like the RGB profile used and the current board layer.

General Description

Each half of the board will have its own Arduino ProMicro. One of them will be the master communicating with the PC using the USB standard. The firmware for communication will be implemented using the LUFA library. The two Arduino microcontrollers will communicate with each other using UART, through a TRRS cable, sharing a common VCC and GND. Each microcontroller will connect to its own display module through an I2C connection. A switch matrix with 6 columns and 4 rows will be created. Each switch will have its own diode in order to prevent ghosting and allow NKRO (N-Key Rollover). Each column and each row will be connected to the microcontroller. The matrix will be scanned by sending a signal through one column at a time and checking the signal on the rows. If we got the signal on a row that means that the key where that row and column intersects was pressed.

Hardware Design

Components

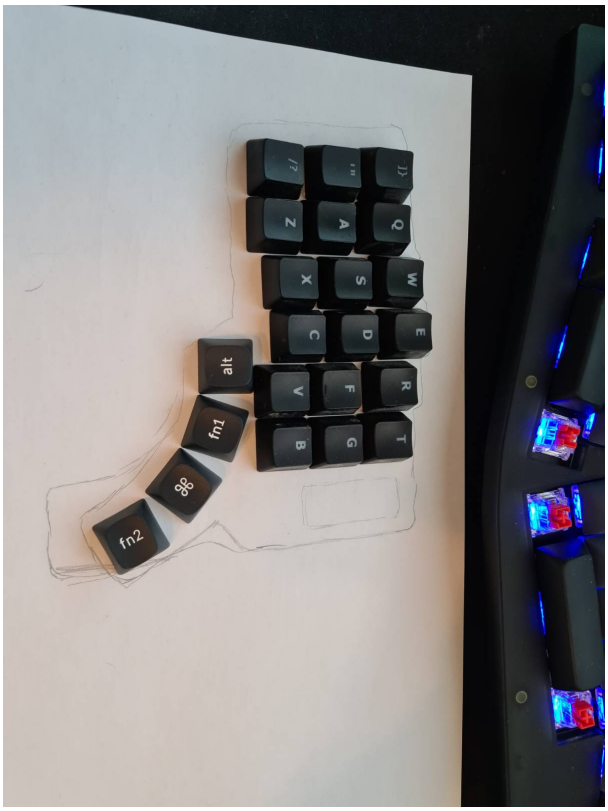
The following components are required to build the keyboard:

- 2x Custom PCBs
- 2x 3D Printed Cases
- 2x Arduino ProMicro (Type-C)

- 2x OLED Display Module 0.91” 128×32
- 2x PJ-320A TRRS Socket
- 44x Kailh MX Hot-swap Sockets
- 44x SK6812 MINI-E RGB LED Modules
- 44x 1N4148W SMD Diodes
- 4x 12-pin sockets for microcontrollers
- 2x 4-pin sockets for OLED Modules
- 2x Acrylic Display Covers
- 16x M2 Standoffs
- 32x M2 Screws
- 44x MX Switches
- 1x Keycaps Set
- 1x TRRS Cable
- 1x USB Type-C Cable

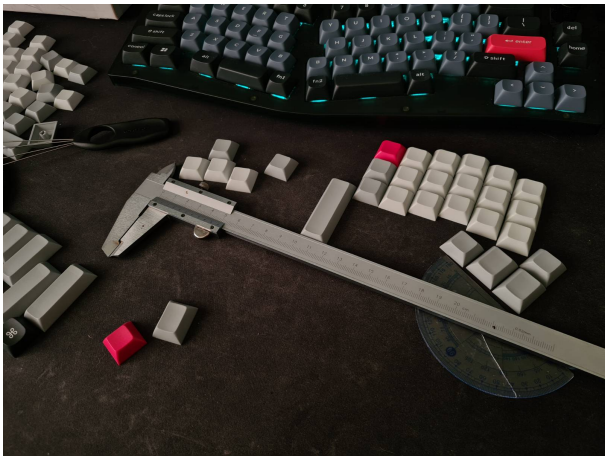
Layout Design

I started designing the layout by removing keycaps from some of my keyboards and placing them in the positions I wanted. Early versions looked like this:



After some trial-and-error arranging the keys, I settled with the following layout that I found very comfortable:

The next step was to recreate it on the computer to have a reference for creating the PCB and case. I chose to use Ergogen for this task. Ergogen is an open-source project aiming to provide a common configuration format to describe ergonomic 2D layouts. First I measured the dimensions of my keycaps and the angle of my thumb cluster and started learning how to write a config file using Ergogen by following tutorials and the Ergogen documentation.



The config file that I have written is this:

ergogen_config.txt

```
units:
  spacing: U
  key_size: 18.2
  big_key_height: 23


points:
  key.height: key_size
  key.width: key_size
  key.spread: spacing
  key.padding: spacing
  zones:
    matrix:
      key:
      columns:
        outer:
        pinky:
        ring:
          key.stagger: spacing/4
        middle:
          key.stagger: spacing/4
        index:
          key.stagger: -spacing/4
        inner:
          key.stagger: -spacing/4
      rows:
        bottom:
        top:
        home:
    thumbfan:
      anchor:
        ref: matrix_index_bottom
        shift: [-spacing/2.5, -spacing*1.2]
        rotate: -29
      columns:
        near:
          rows.thumb_extra.skip: true
        home:
          key.height: big_key_height
          key.stagger: (big_key_height - key_size) / 2
          rows.thumb_extra.skip: true
        far:
          rows.thumb.height: big_key_height
          key.padding: big_key_height / 2 + key_size / 2 + 0.8
      rows:
        thumb:
        thumb_extra:


outlines:
```

```
keys:
  - what: rectangle
    where: true
    operation: stack
    size: [U, U]

plate:
  - what: rectangle
    where: true
    operation: stack
    size: [14, 14]
```

The config file generates this layout:

PCB Design

The next step was to design the PCBs for my project. After studying which CAD software would fit my needs better, I chose to use KiCAD because it is the most popular software for keyboard PCB design and there are libraries you can find on Github containing the footprints of the parts I chose to use.

I started learning about how keyboards work, what components are needed and how are they connected to each other. After spending a lot of time reading documentation, researching other projects and watching tutorial I created my PCB schematic for the left side of the keyboard.

Using the Ergogen layout and the schematic, I began working on the PCB itself. After carefully placing all of the components in their position and creating mounting holes I started routing the PCB. The first problem occurred when I received my components from the store. The initial project featured a reset button, but because the dimensions of the Arduino ProMicro were bigger than the ones written on the store page, I had to remove it and readjust the PCB, and the footprint of the microcontroller. After a lot of work all that was left to be done was branding my PCB with the RootKB logo on the silkscreen. The final result looks like this:

Then I had to repeat the whole process for the right half.

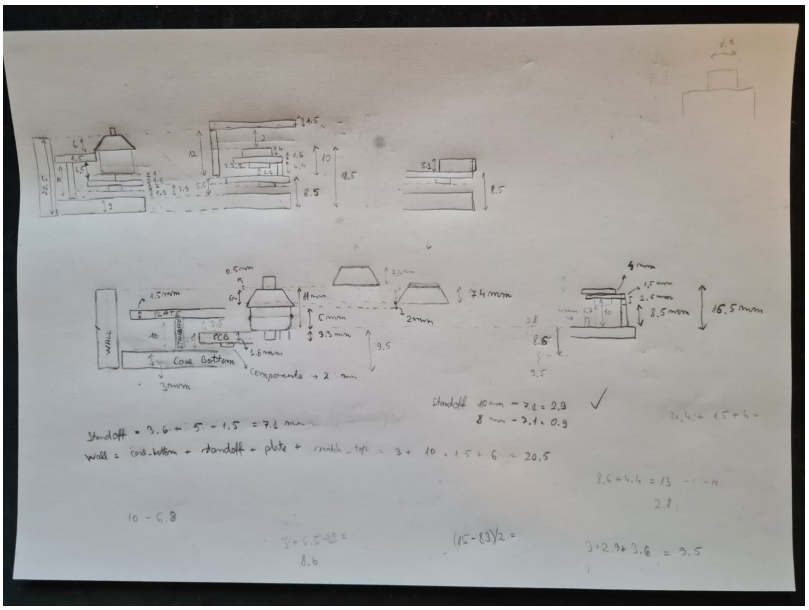
Next, I ordered the PCBs and prayed that they work as envisioned.

Plate Design

With the PCB ready, I imported the .dxf files to Fusion360 in order to create the plate. After researching the required dimensions that will allow the switches to click it, I created my first design meant to be made out of FR-4 or aluminum with a 1.5mm thickness. I also created annotated 2D drawings in AutoCAD for production. This ended up too expensive and would take too long to be made. So I redesigned the plate to be 3D printed with a 3mm thickness for more strength.

Case Design

In order to create the case I had to carefully calculate the dimensions. At this point I found out that the sockets I originally planned to use were to tall and the microcontroller would rise above the case (which would be an unacceptable design flow in my opinion), so I switched the socketing option:



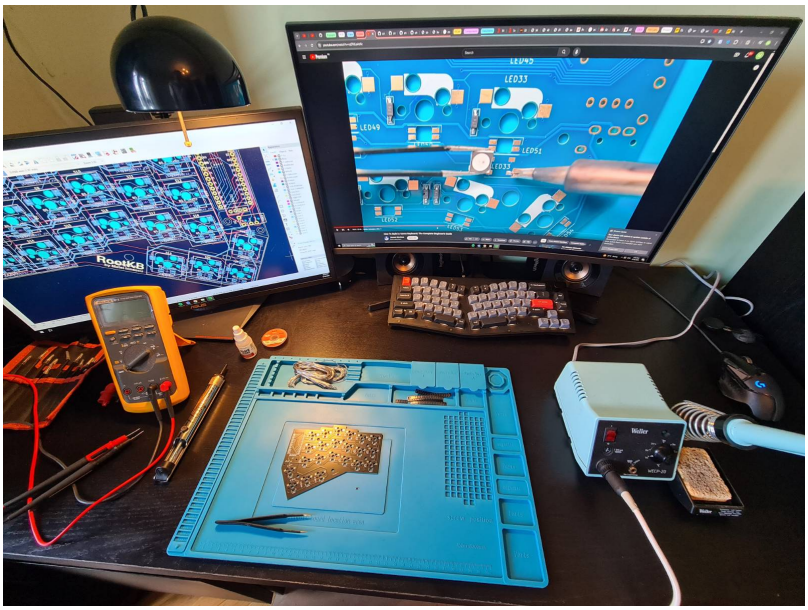
The designed cases:

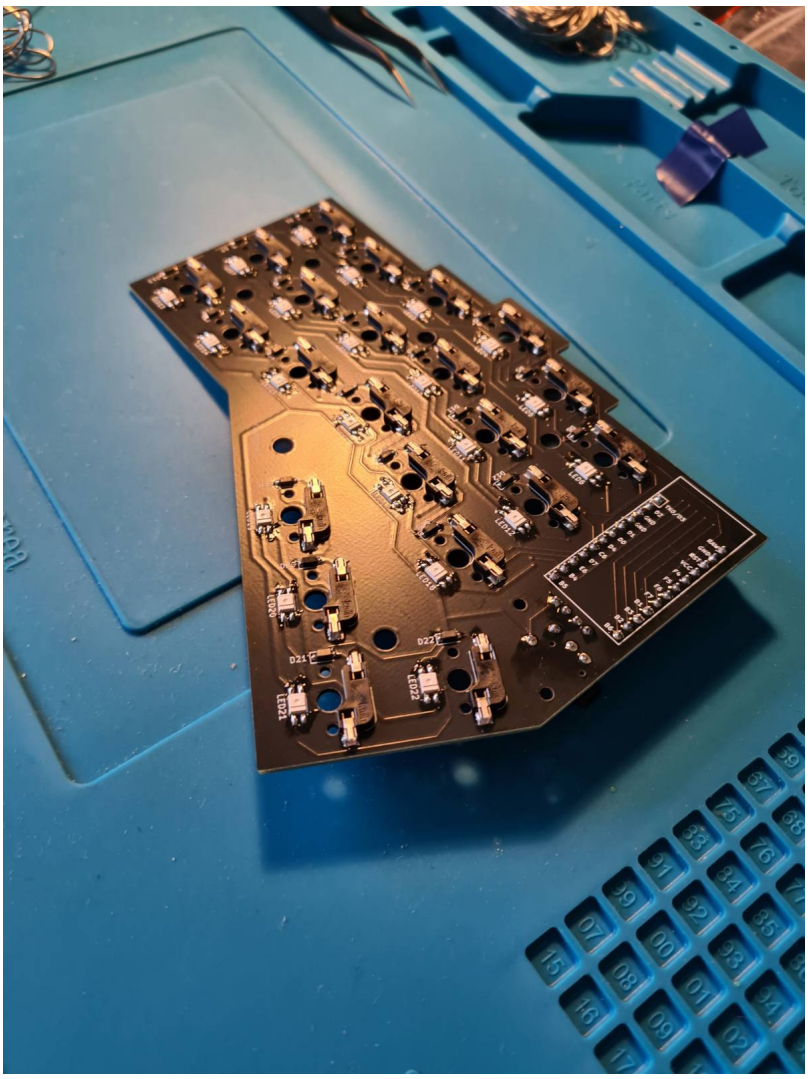
And putting it all together:

Building The Keyboard

I finally got all the components.

It was time to build the keyboard. I soldered all the 44 diodes, 44 switch sockets and 44 LEDs. For socketing the microcontrollers I used resistor legs as pins in order to fit the case as mentioned above.





After I bulit the first half, I wrote some quick scripts to test that everything works fine.

And after another day of soldering I had two working keyboard halfs.

Software

The original plan was to use the LUFA library, but due to lack of time I ended us using the Arduino Framework.

Matrix Scanning

Scanning the matrix is the task that runs most of the time on the microcotroller. To store the state of the matrix I am using bit operations. Each half of the matrix is represented as a uint64_t number. Each half of the keyboard reads the state of the matrix representing the keys that are pressed. Then the right half of the matrix sends using UART its matrix state uint64_t number to the left half. The state of all keys on the keyboard is then obtained by using the bitwise OR operation.

Sending Keys to PC

The layers of the keyboard are represented in the keys.h file. According to which keys are pressed in the matrix state, the according key will be sent to the PC. When a layer changes all keys from that layer will be released.

The layouts of the keyboard look like this:

RGB & OLED

I have also implemented brightness and color control of the RGB lights on the keyboard. When the the brightness is modified the change can be seen on the display, and after a short period of time it goes back to displaying the logo. When changes happen, the Left half will send the new color to the right half over UART.

Conclusion

This was a challenging project, that took me a couple of months to design, build and program, but the result is awesome and I am happy I'm writing this part of the article on the keyboard I built from scratch.

GitHub

All the files for this project are available at https://github.com/Matei77/RootKB, including the case design, PCB and software.

Journal