# Project of Electrical and Electronic Technology

Project name:

**Open-Source Ergonomic Keyboard**

School name: Mechanical Engineering

Student name: Radiv Sarwar

Student ID: 2019380138

2024.06.28

# Abstract

In modern day-to-day life, ergonomics plays a crucial role in enhancing comfort, health, and efficiency across various activities. From the office to home environments, ergonomic principles inform the design of furniture, tools, and devices to better fit human capabilities and limitations. Ergonomic office chairs and desks promote proper posture and reduce the risk of back and neck pain, while ergonomic keyboards and mice prevent repetitive strain injuries like carpal tunnel syndrome. In the home, ergonomic kitchen tools and furniture improve ease of use and reduce physical strain during daily tasks. Even in digital interactions, ergonomic considerations ensure user-friendly interfaces on computers and smartphones, accommodating diverse needs and promoting accessibility. By integrating ergonomic design into everyday products and environments, individuals can enjoy enhanced well-being, productivity, and quality of life. Among all the tools used in our day-to-day life the keyboard is one of the most overlooked tools that is utilized. In the modern life most work is done using computers and keyboards are one of the 2 primary input sources for computers. Unfortunately, the design of most common keyboards is conducive to wrist problems after heavy usage. To solve this problem, we need to introduce cheap ways to produce keyboards with an ergonomic benefit.

Ergonomics in keyboard design is crucial for minimizing wrist pain and promoting overall musculoskeletal health. A well-designed ergonomic keyboard encourages a neutral wrist position, reducing strain on tendons and muscles that can lead to conditions like carpal tunnel syndrome and tendonitis. Features such as split keyboards, adjustable tilt angles, and wrist rests help maintain proper alignment of the wrists, elbows, and shoulders, preventing repetitive strain injuries. Additionally, ergonomic keyboards consider factors like key layout and spacing to optimize comfort and typing efficiency, ultimately supporting long-term comfort and productivity for users. Taking these ergonomic principles into account can significantly alleviate wrist pain and enhance the overall ergonomic experience of using keyboards.

We will be exploring an opensource implementation of an ergonomic keyboard that can be assembled by anyone using the opensource documentation.

**Key words:** Ergonomics, Keyboard, Opensource;

# 1. Contents

# ❖ Introduction

## 1. Content and introduction of the project

### 1.1 Purpose and the Keyboard

Wrist pain can develop due to various factors related to repetitive movements, poor ergonomics, or underlying conditions. Repetitive strain injuries (RSIs) are common culprits, where frequent and repetitive motions, such as typing or using a mouse, can strain the muscles, tendons, and ligaments in the wrist and forearm. When these activities are performed with improper posture or without adequate breaks, it can lead to inflammation and irritation of the tendons (tendonitis) or compression of the median nerve (carpal tunnel syndrome), causing pain and discomfort.

Ergonomics plays a crucial role in preventing wrist pain. Poor ergonomics, such as using a keyboard or mouse that is too high or too low, or not properly supporting the wrists, can contribute to strain and discomfort. This is where the concepts of pronation and supination come into play. Pronation refers to the inward rotation of the forearm and hand, while supination is the outward rotation. Improper positioning, such as excessive pronation (where the forearm and hand are rotated inward excessively), can strain the muscles and tendons that support the wrist joint. This strain can exacerbate over time, leading to chronic pain and stiffness.

Similarly, excessive supination can also strain the wrist and forearm muscles, particularly if sustained for prolonged periods. Ergonomic keyboards and accessories are designed to mitigate these issues by promoting a more natural alignment of the hands and wrists. For instance, ergonomic keyboards often have split or curved designs that encourage a more neutral wrist position, reducing the strain caused by pronation or supination. By adopting ergonomic principles and equipment, individuals can minimize the risk of developing wrist pain associated with repetitive strain and poor positioning, thereby promoting comfort and long-term musculoskeletal health.

One would want to use an ergonomic keyboard primarily to enhance comfort and reduce the risk of repetitive strain injuries. These keyboards are designed with features such as split layouts, curved keys, and adjustable tilt angles, which encourage a more natural hand and wrist position during typing. By promoting better ergonomic alignment, they can help prevent conditions like carpal tunnel syndrome and tendonitis that often result from prolonged, repetitive keyboard use. Additionally, ergonomic keyboards can improve overall typing efficiency and comfort, making them a valuable choice for anyone who spends significant time at a computer.

The keyboard we will be selecting today is called the dactyl-manuform keyboard. This keyboard is an opensource keyboard that is focused on the ergonomic designs of keyboards.
Originally the dactyl keyboard was introduced in the open-source community by Matt Adareth on 2015 at a closure conference. He developed an algorithm with the closure programming language to form a dactyl keyboard in the open-SCAD program that matches the size of his hands. This program was open source and hence many programmers flocked to the project to try and 3D print their own version of the keyboard.

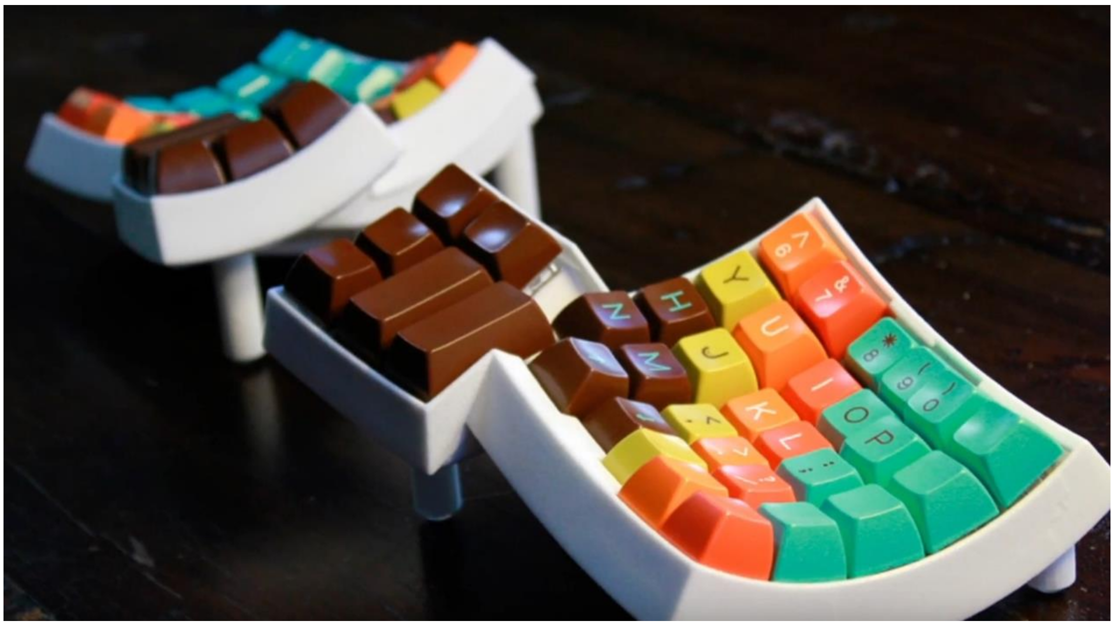


**Figure 0-1: A dactyl keyboard**

The main features of this keyboard is that it has no staggered layout when it comes to the rows. Normally in any modern keyboard we will see the staggered rows. This means the rows are not lined up with each other and each subsequent key in one column is situated at a diagonal direction rather than directly under. This design choice stems from the layouts of typewriters. Because of the mechanical components and jamming issues, typewriters had to have such staggered rows. But for modern mechanical keyboards we have no such obstacles to keep the rows staggered. This was still adopted to the design of keyboards because of the convention of the time and the skill already attained by typers using a typewriter.
This staggered layout is actually not useful for writing. Our fingers can easily move upside downside and left and right. But diagonal movements are not intuitive in the same way. This staggered layout introduces unnecessary movement by our hands to type. This contributes to long term fatigue and problems in our wrists.

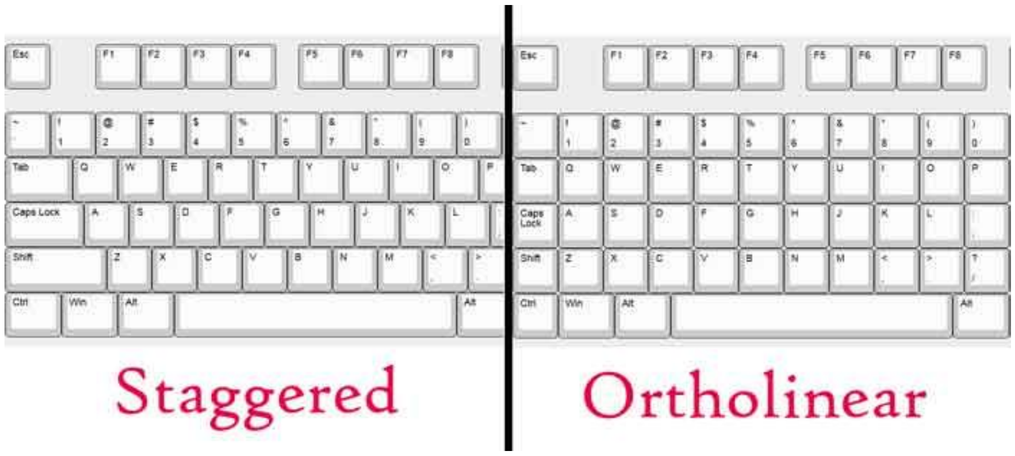


Figure 0-2: staggered vs ortholinear keyboards

The Dactyl-manuform is a fork of this open-source project. Here fork means that it is an offshoot of an original project taking its own form. The dactyl-manuform is a split keyboard that utilizes the unique concave shape along with an ortholinear layout that addresses most if not all of the issues with modern keyboards that contribute to bad ergonomic practices.

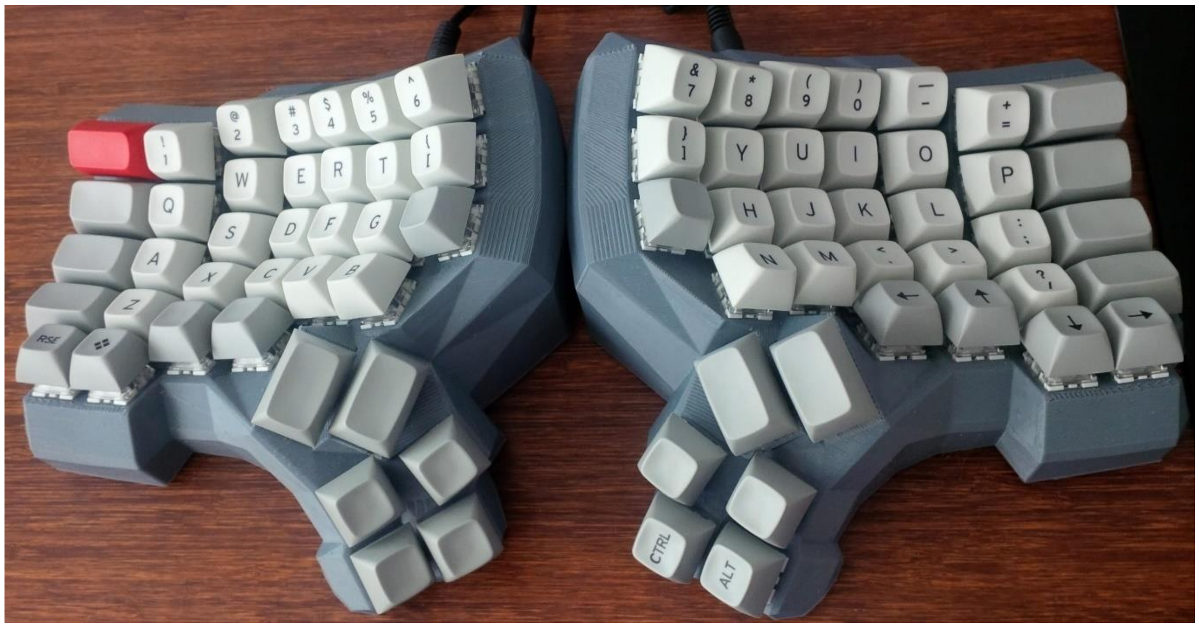


Figure 0-3 A dactyl manuform built by a community member

The concave shape of the keyboard provides a very comfortable place to keep your hands that does not make your hand too supinated or pronated. Each column of this keyboard is shaped to groove under each digit of your hand. As a result this design promotes proper typing etiquette and all the fingers are used for typing. Due to the ortholinear nature of this keyboard your fingers can naturally find the keys by moving in straight directions.

In the nature of being open-source, we will also be using an open-source firmware called “qmk” to flash and use this keyboard.

### 1.2 Requirement

1. The keyboard built should be ergonomic in structure. This means that in rest position, the hands should take a very comforting position and during typing the fingers cannot travel too far from the home row (a,s,d,f for the left hand and j,k,l,; for the right hand).
2. The build material and process have to be approachable to anyone. The case will be 3D printed according to the opensource stl files.
3. Pronation and supination for the hands have to be minimum while using the keyboard. In the long term this will prevent wrist damage and carpal tunnel syndrome.

## 2. Knowledge required in the project

1. Understanding how to create the circuit for a keyboard. Generally, keyboards have very simple PCB designs. However, for this project we will be hand wiring the keyboard manually instead of developing a PCB.
2. Understanding how to compile and flash a firmware on the microchip used to operate this keyboard. For this keyboard we will be using an Arduino pro micro UTAtmega32.
3. Basic soldering skills.

## 3. Expected results

1. After 3D printing, we will be clipping in the switches to the body.
2. We will be soldering a matrix circuit with the switches along the body.
3. After flashing the firmware, we will be able to use the keyboard.

## 4. Progress and time-line

- *Jun.15th, 2024*

I ordered the 3D printed shell from a 3D printing company

- *Jun.20th, 2024*

The shell arrived with most of the parts. I prepared the schematic and chose the variant of the keyboard I want to build.

- *Jun.21st,2024~Jun.25th,2024*

I assembled and soldered all the required parts that needs to be connected in the device.

- *Jun.25th,2024~Jun.27th,2024*

After the assembly there were several problems that required trouble-shooting. I spent all this time fixing all the issues and problems with the build.

# 2. Mechanical design & Model based on animation

## 1.1 Design

Due to the open-source nature of this project we can obtain the stl files for this keyboard. It should also be noted that there are various versions of this keyboard. Since this project allows people to design the board to suit themselves it is very common to add a row/column or delete entire rows/columns according to the needs of each individual. For this project I have chosen the 5x7 layout of the dactyl keyboard. This refers to the version of the board that contains 5 rows and 7 columns on the keyboard. All the alphanumeric keys, arrow keys, home, end, alt, cntrl, pgup, pgdown and the number row will be available on the keyboard. Through the firmware we will also be able to code in a secondary layer that will act as a numpad and turn the number keys to function keys. Here is the model for the 5x7 version of the dactyl keyboard that I will be building:

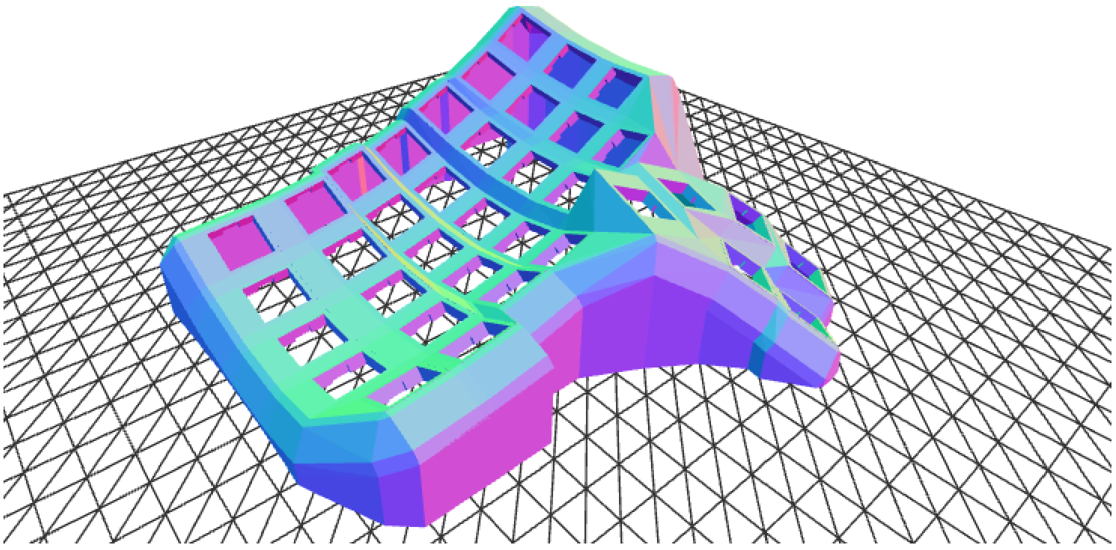


**Figure 2-1 : left part of the split keyboard**

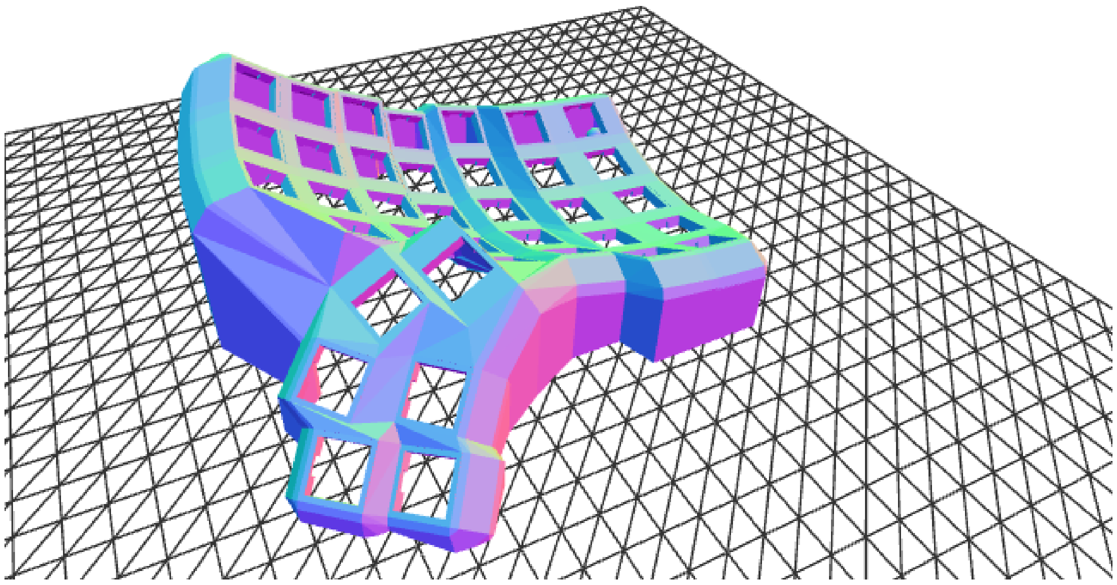


Figure 2-2: right side of the split keyboard

This version is taken from the open-source fork that can be found in the references section.[1]

## 1.2 Structure Components

**Introduction to the main component:**

**The shell:**

The shell is a 3D printed shell according to the stl files found from the open SCAD program to develop dactyl-manuform shapes. This is shell acts as the outer case and the plate to hold the switches at the same time.

**Introduction to auxiliary components:**

**Keyboard switches：**

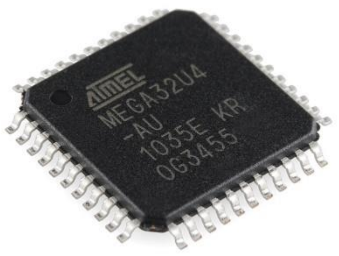
They are the units of the keyboard. These switches are mechanical in nature. They are called Gateron Yellow and they are linear. Due to the design of the keyboard shell the switches are snapped into place and stay very stationary without any wobble.

**Programmable control board：**

I will be using the Arduino pro-micro that houses the ATmega32U4 chip to operate this open-source keyboard. I will not be using the Arduino program however. I will be using the open-source firmware QMK to compile a firmware and flash it into the control board.

# 3. Electrical modules and elements

## 2.1 ATmega32U4

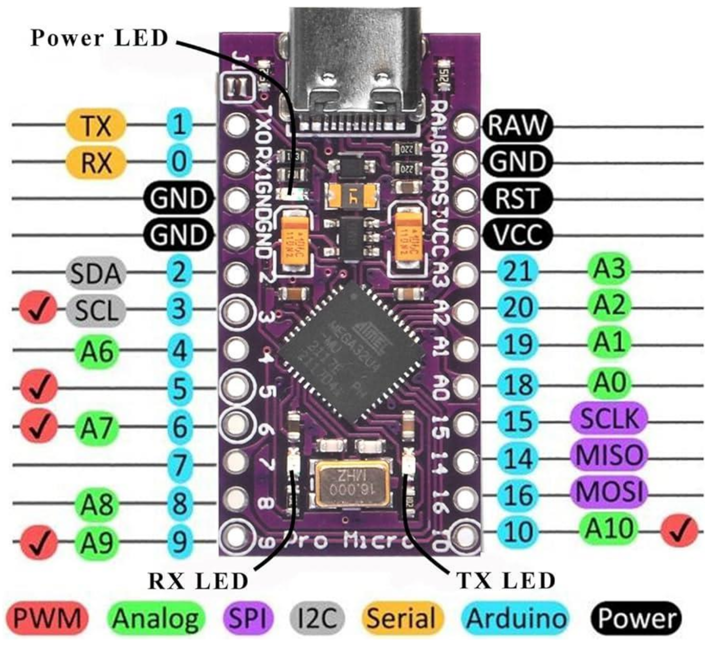


The ATmega32U4 is a microcontroller from Microchip Technology (formerly Atmel), known for its versatile applications in various embedded systems. Here are its specifications:

1. **Architecture**: 8-bit AVR microcontroller.
2. **CPU Speed**: Operates at up to 16 MHz
3. **Flash Memory**: 32 KB for storing program code.
4. **SRAM**: 2.5 KB for temporary data storage.
5. **EEPROM**: 1 KB for non-volatile data storage.
6. **I/O Pins**: 26 general-purpose I/O pins.
7. **Timers**: 3 timers/counters for timing and PWM (Pulse Width Modulation) applications.
8. **Analog-to-Digital Converter (ADC)**: 10-bit ADC with up to 12 channels.

9. **Communication Interfaces**: Includes UART (Universal Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface), and I2C (Inter-Integrated Circuit).
10. **USB Interface**: Integrated USB 2.0 full-speed device interface, which is a notable feature compared to many other AVR microcontrollers.

We will be using the Arduino pro micro for the chip and its programmability. This board has all the required pins for our columns and rows. We will also need VCC channels to use a



## 2.2 1N4148 Diodes:

The 1N4148 is a widely used small signal switching diode known for its fast-switching speed and low forward voltage drop of approximately 0.7V at 10 mA. It features a reverse voltage rating of 100V and boasts a very fast reverse recovery time of around 4 ns, making it suitable for high-frequency applications where rapid switching is essential. This diode is available in various package types including DO-35 (glass axial leaded), SOD-80 (mini-MELF), and SOD-123 (SMD package), catering to different circuit design needs. Common applications include signal and logic circuit switching, voltage spike protection, and signal rectification in small power supplies.

Specifications of 1N4148 Diode:

1. Type: Small signal switching diode
2. Forward Voltage Drop: 0.7V at 10 mA
3. Reverse Voltage Rating: 100V
4. Reverse Recovery Time: 4 ns (typical)
5. Package Types: DO-35 (glass axial leaded), SOD-80 (mini-MELF), SOD-123 (SMD)

## 2.3 Gateron Yellow Keyboard switches:

Gateron Yellow keyboard switches are a popular choice among mechanical keyboard enthusiasts for their smooth linear feel and reliability. Known for their light actuation force and consistent keystroke response, Gateron Yellow switches offer a satisfying typing experience ideal for both gaming and typing tasks. They are prized for their affordability compared to other premium mechanical switches while maintaining a high standard of quality and durability.

Specifications of Gateron Yellow switches:

- Type: Mechanical
- Switch type: Linear
- Actuation force: 50g
- Actuation point: 2.0mm
- Total travel distance: 4.0mm
- Keycap stem: MX-compatible
- Lifespan: 50 million keystrokes

## 2.4 3.5mm AUX cable:

A 3.5mm AUX cable with 3 contact points, commonly known as a TRRS (Tip-Ring-Ring-Sleeve) cable, is designed primarily for audio devices that support microphone input in addition to stereo audio output. This type of cable is widely used for connecting headphones or headsets with integrated microphones to smartphones, tablets, laptops, and other audio equipment. The additional contact point allows for both audio playback and microphone communication through a single cable, making it convenient for hands-free calling or voice recording applications. The specifications of a typical TRRS 3.5mm AUX cable are as follows:

Specifications:

- Connector type: 3.5mm TRRS (Tip-Ring-Ring-Sleeve)
- Cable length: Variable (common lengths include 1m, 1.5m, 2m, etc.)
- Material: Typically copper conductors with a PVC or TPE outer jacket
- Compatibility: Compatible with devices that have a 3.5mm audio jack supporting TRRS functionality
- Usage: Suitable for connecting headphones/headsets with microphone to audio devices for both listening to audio and using the microphone.

## 2.5 TRRS audio jack:

A TRRS (Tip-Ring-Ring-Sleeve) audio jack is a type of 3.5mm audio connector that includes four conductive segments. It is commonly found on smartphones, tablets, laptops, and other audio devices that support both stereo audio output and microphone input through a single jack. The TRRS jack allows for the integration of headphones or headsets equipped with a microphone, enabling users to listen to audio output while also using the microphone for tasks such as voice calls, voice recordings, or gaming communications. This multi-functional capability makes TRRS jacks versatile for various audio applications where both listening and speaking functionalities are required in a compact, single-port design.

Specifications of a TRRS audio jack:
- Connector type: 3.5mm TRRS (Tip-Ring-Ring-Sleeve)
- Number of conductive segments: 4 (Tip, Ring1, Ring2, Sleeve)
- Compatibility: Compatible with devices that support TRRS functionality
- Typical use: Used for connecting headphones or headsets with integrated microphones to audio devices
- Application: Suitable for hands-free communication, gaming, voice recording, and multimedia playback.

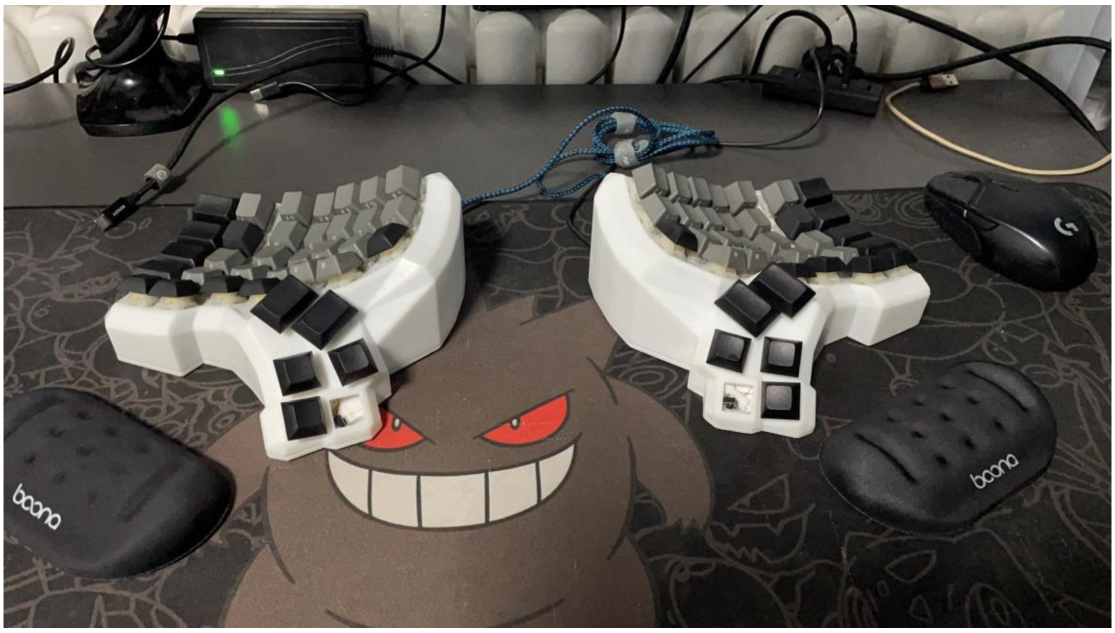
### 2.5 Microswitch:

The 4 Pin 6x6 Tact Push Button Switch, often referred to simply as a micro switch or tact switch, is a small electronic component commonly used in electronic devices for momentary contact applications. These switches will be used for reset buttons for both microcontrollers of each side.

# 4. Construction of logic and programming software

## 4.1 Basic assembly steps

After I get the shell, I fit all the keys inside the sockets for the switches. The sockets for the switches are already made tight fit. The switches sit flush and snap into the place. Then we fit the keycaps on top of the switches to finish all the builds that’s required on the top side.



This is how the top side will look without all the wiring being done. Now I will be soldering on the wires.

Except the internal wiring of the circuit the top construction is done.

## 4.2 Circuit design and hand wiring

The wiring of a keyboard is a kind of matrix circuits. A typical keyboard employs a matrix circuit to register key presses efficiently. For example, in this keyboard we will have about 72 keys (36 on each half). The microcontroller we are using does not have 72 pins to account for all 72 keys. For cases like this we employ a matrix like circuit.

Imagine a grid where each row corresponds to a set of keys and each column represents another set. When you press a key, it completes a circuit between its corresponding row and column. This intersection uniquely identifies the pressed key based on its row and column coordinates.

To manage this, the keyboard controller scans each row of the matrix sequentially. It checks the status of each column to detect any completed circuits (key presses). This scanning happens rapidly, usually faster than human perception, ensuring real-time responsiveness.

When a key press is detected, the controller identifies which row and column are involved, thereby determining which key was pressed. This information is then sent to the computer as an input signal.

Keyboards typically use diodes to prevent ghosting and ensure accurate key detection, allowing multiple keys to be pressed simultaneously without confusion. This matrix arrangement efficiently reduces the number of wires needed, making keyboards compact while maintaining functionality. For the diodes in this build we are using the 1N4148 diodes.

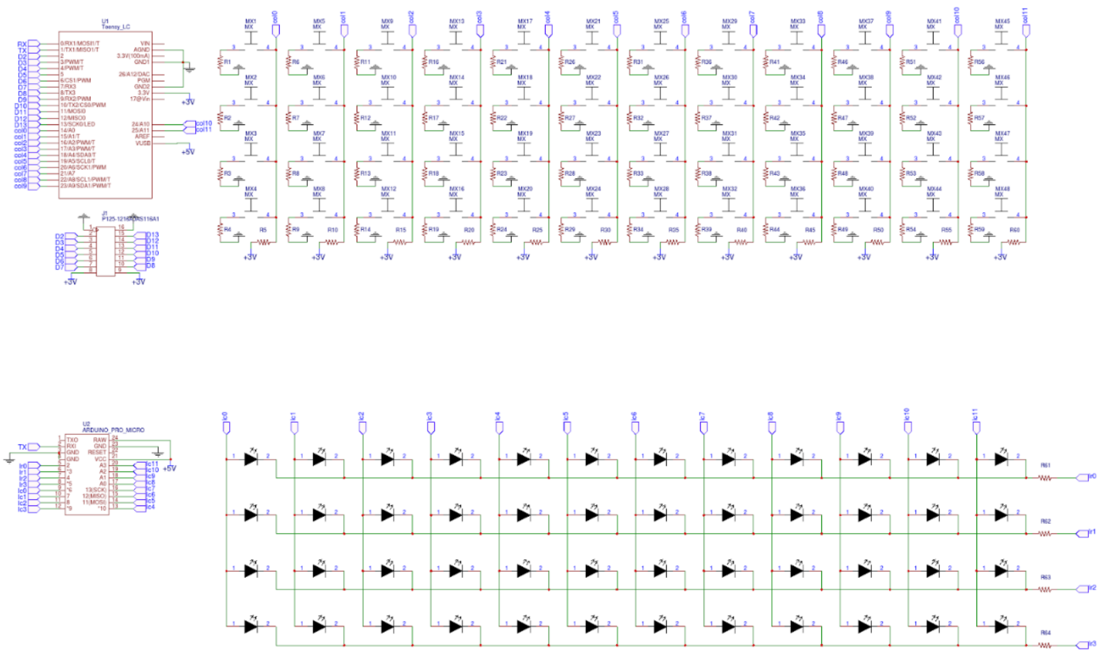


**Figure 4-1 typical wiring circuit of a keyboard**

The circuit diagram we will be using for this build will be shown below:

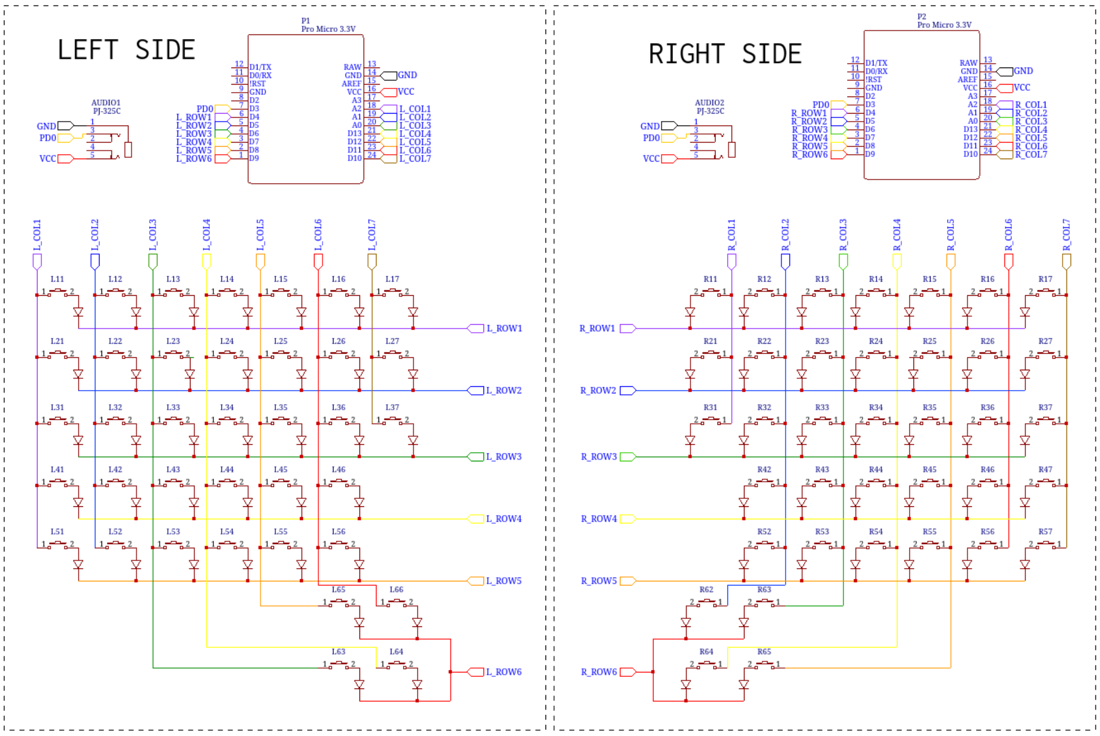


**Figure 4-2 The circuit diagram for what we will be making**

From the figure 3-2 we can see that the number of columns on each side is 7 and the number of rows on each side is 5. Hence this is known as the 5x7 variant of dactyl-manuform. Between each of the column connections between each key we will be soldering a diode to ensure that the flow of electricity is one-way to avoid ghosting issues.
Here we can also observe the inner working of the thumb cluster. From the outside one would assume that the thumb cluster of the keyboard is a separate column and row structure in the circuit. However, for simplifying the circuit we can see that the column 3,4,5 and 6 are continued to make up for the columns and the rows 6 covers all the thumb keys. Even though here we can see that the row 6 adds another row of keys it is not counted when giving the conventional name to this variant of the keyboard.

Human hands are very versatile and it helped our species to make and create tools that led to the current day civilization. The thumb of our hands is one of the strongest muscles but unfortunately it does not shine while typing on a normal keyboard. To utilize this very strong point of our hands ergonomic keyboards have a very wide selection of keys that are made for the thumbs.

The reason for hand wiring this keyboard instead of designing and printing a PCB is the structure of the shell of the keyboard. The shell is concave. Thus, it is not feasible to house a flat PCB inside this shell. Very talented community members have already made flexible PCB boards to house the switches in this shell but I did not want to risk handling a very flexible and nimble PCB. I have experience soldering hence I went for the hand wiring route.

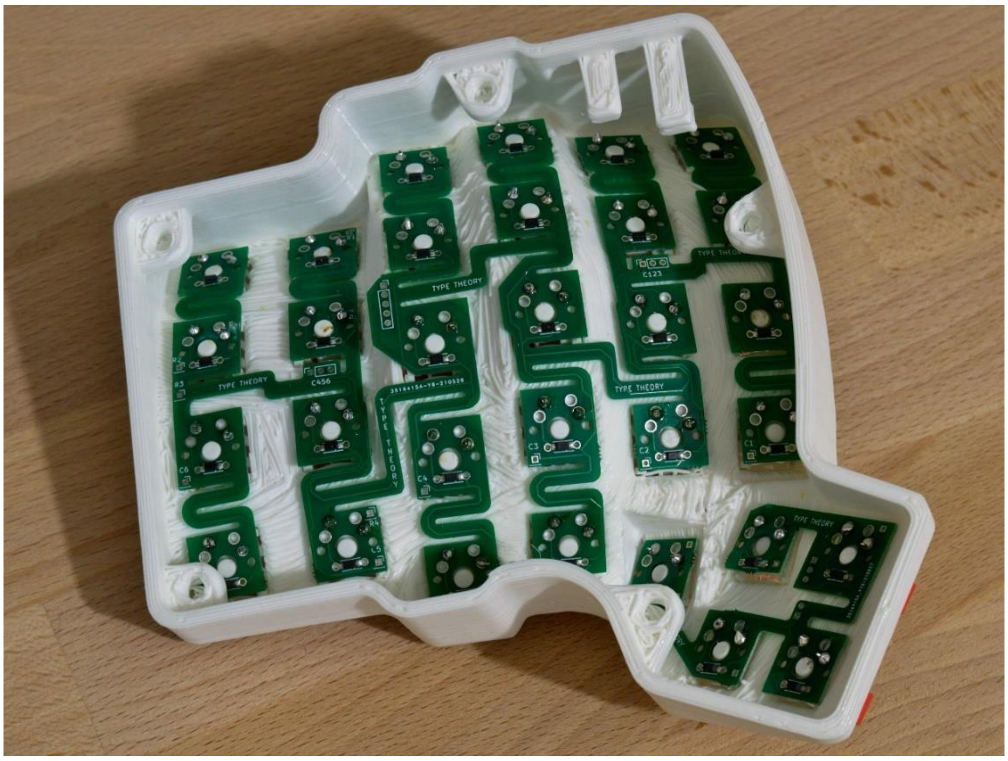


**Figure 4-3 An example of flexible PCBs inside a dactyl-manuform keyboard**[2]

In our build I will be only building with 70 keys. The keys L64 and R64 will be left hollow to house the reset switches. The function of the reset switch is to allow the user to reset the microcontrollers on both sides reset when we will be attempting to flash the firmware. I am a user who experiments with different keyboard layouts like Dvorak and Coleman other than QWERTY. Having access to the reset switches so easily helps to quickly flash the firmware to the board with a new keymap of our liking. If something is wrong or I want to make different keymaps with different layers I will be able to compile a new firmware easily and flash the firmware very quickly and easily without taking the keyboard apart.
Keymaps are files that define the mapping of keys on a keyboard. While compiling we provide a keymap for the firmware. This keymap is then followed to compile the firmware.

Human hands are very versatile and it helped our species to make and create tools that led to the current day civilization. The thumb of our hands is one of the strongest muscles but unfortunately it does not shine while typing on a normal keyboard. To utilize this very strong point of our hands ergonomic keyboards have a very wide selection of keys that are made for the thumbs.

# 5. Model building and analysis of problems

## 4.1 Testing the microcontroller and switches before assembly

Before fitting and soldering we connect the microcontrollers to the computer to check if its okay. I also test out the switches on my other keyboard to make sure I have no faulty ones.

Figure 5-1 Testing out the switches on another keyboard

There are no other components in this build that requires testing. We can proceed with the build if everything is okay.

## 4.2 Wiring the keyboard

According to the circuit design I started soldering the underside of the shell. Each switch has 2 pins for connection. We use one pin for rows and the other for columns.

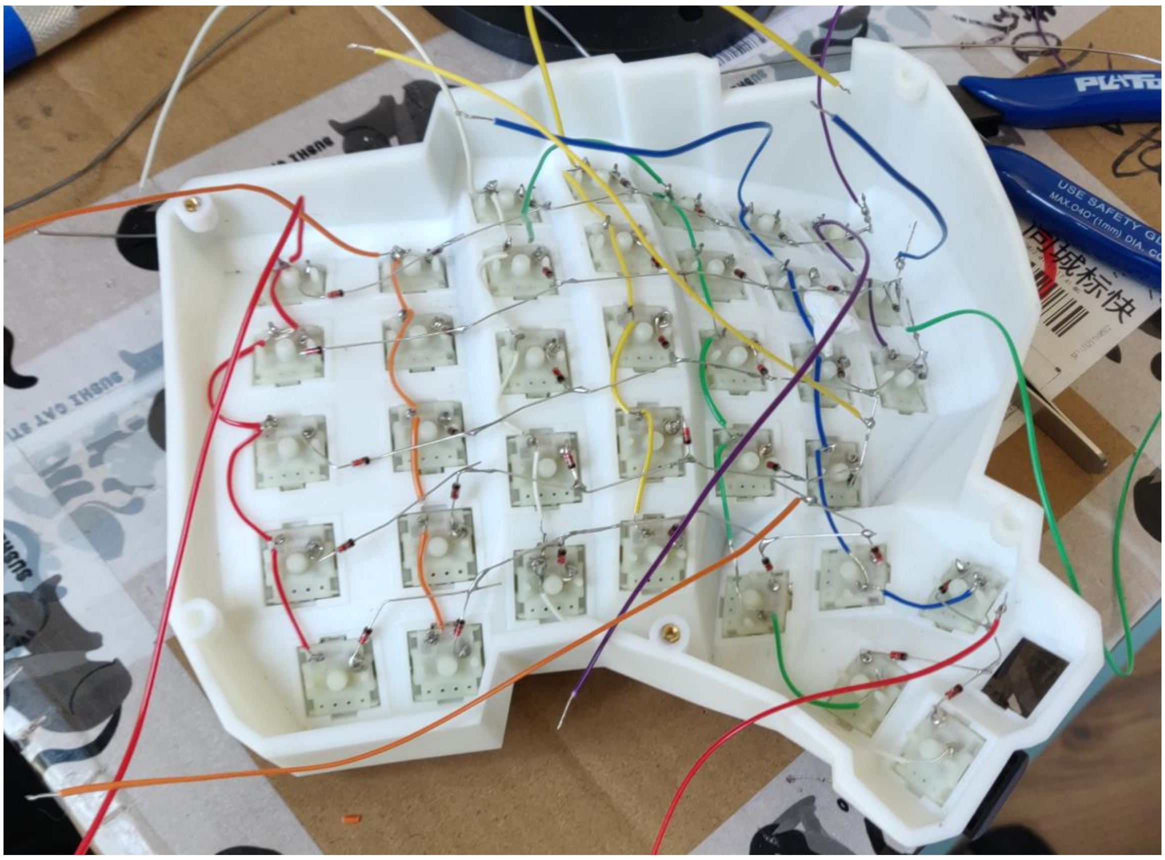


**Figure 5-2 The internal wiring of the left side after soldering the switches.**

As we can see from the figure 4-2, the top pin of each switch was used to solder on the columns and the bottom pin of the switches were used to connect all the rows of the keyboard. For the rows I used the diodes to form the circuit connection between each switch. This allows for no wire use for rows. This helped me differentiate the rows from the columns. Different colors were used to wire the columns to differentiate the matrix connection to the pins of the microcontroller.

For the columns,

| | |
|---|---|
| Red | 1st column |
| Orange | 2nd column |
| White | 3rd column |
| Yellow | 4th column |
| Green | 5th column |
| Blue | 6th column |
| Purple | 7th column |

Extra wires at the end of each connection are also made to solder on to the pins of the microcontroller. The same process is repeated on the other side.

Now I solder the end connections of all the rows and columns to the pins of the microcontroller.

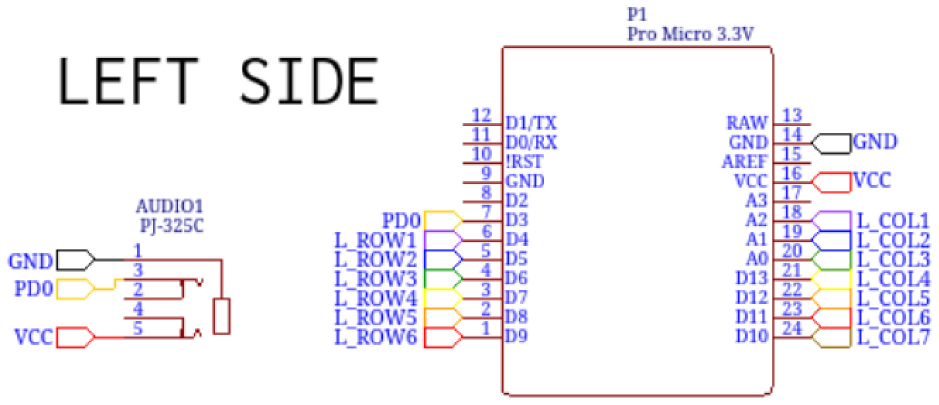


Figure 5-3: The pin connections of the microcontroller

As we can see from figure 4-3, rows 1 through 6 will be soldered to the pin 6 through 1 respectively. Similarly on the other side we will solder the columns 1 to 7 to pin 18 to 24 respectively in that order. The other two components we will be connecting is the reset switch which is not mandatory and the audio jack port. The audio jack port in this build will facilitate communication between both sides of the split keyboard. Depending of the use case and preference, only one microcontroller will be connected to the computer. That means all the inputs from both sides will have to go through that one connection. Both of the sides of the split keyboard talk to each other through the audio jack we will be connecting. The pin layout of the audio jack is given in figure 3-5. According to the figure we connect the ground pin to one of the ground pins of the microcontroller. The PD0 pin is connected to pin 7 and VCC pin is connected to the VCC pin on the microcontroller. Pin 10 of the micro controller is the reset pin. We connect one of the pins of the reset switch to the reset pin and another end to the ground pin in the microcontroller. That is all the connections we make with the microcontroller.

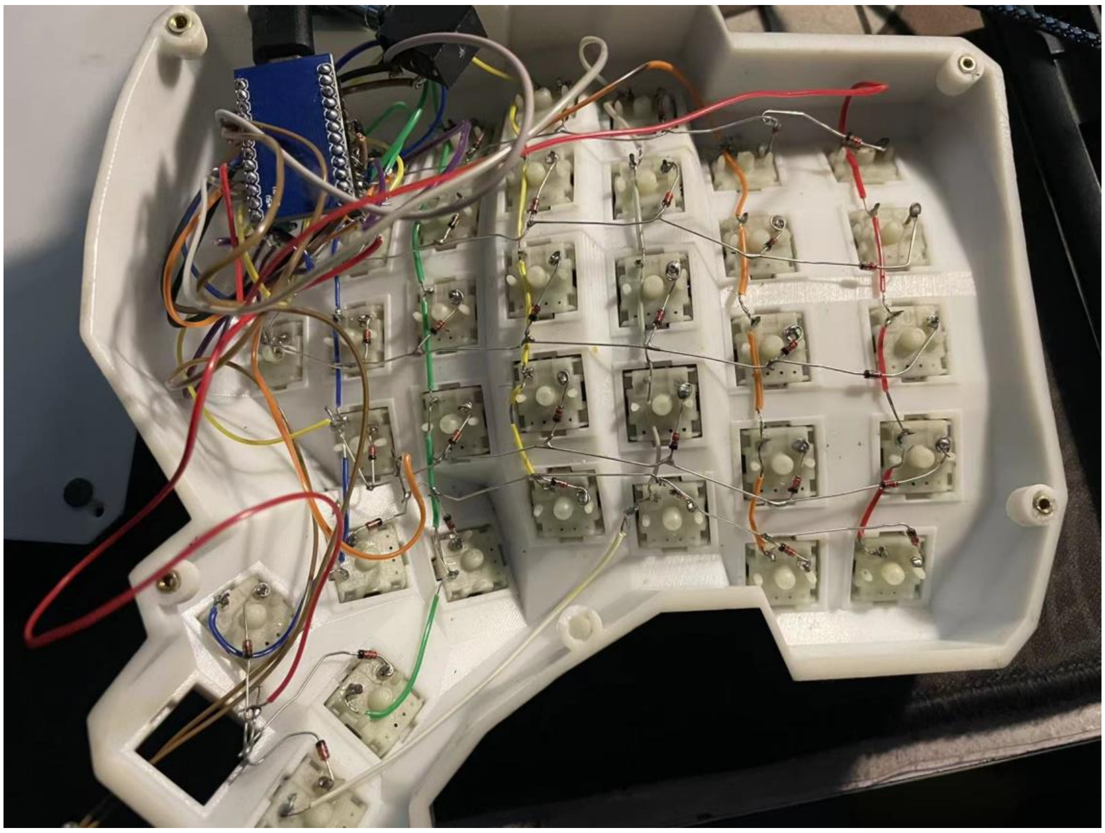


**Figure 5-4: The right side after soldering the microcontroller with the proper connections.**

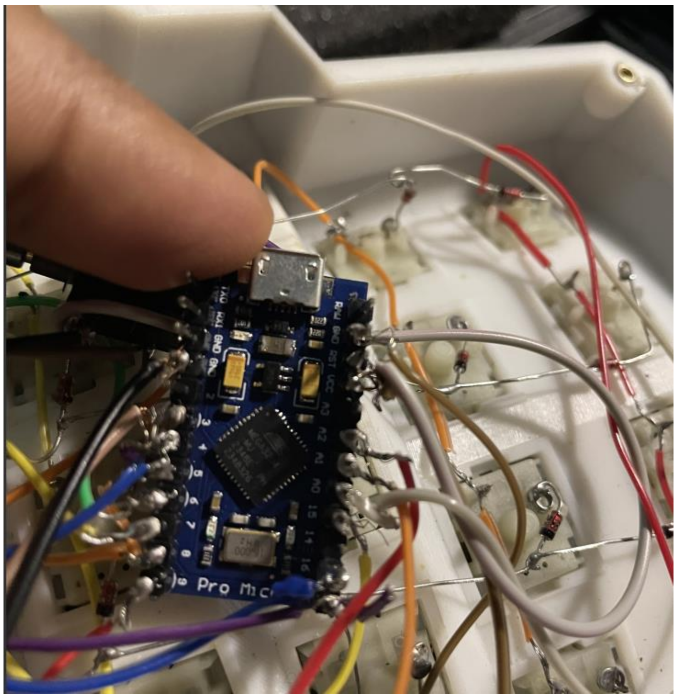


**Figure 5-5: Closer look at the microcontroller after soldering all the connections.**

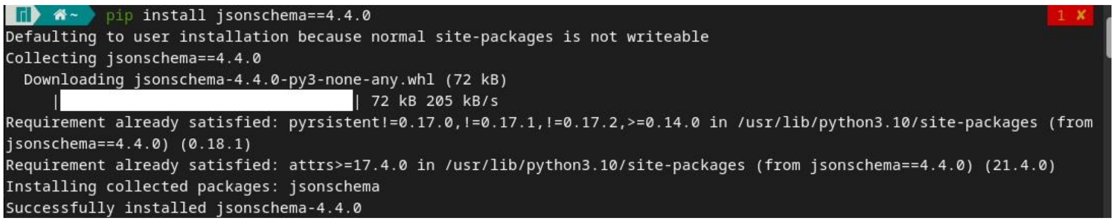
## 4.3 Installing QMK on the computer

For this project we will be using QMK open-source firmware to flash the keyboard. QMK Firmware, an open-source project primarily for mechanical keyboards, traces its origins back to 2013 when it began as a fork of the TMK (Tiny Matrix Keyboard) project. Initially designed for Atmel AVR microcontrollers, QMK has evolved into a versatile firmware platform supporting numerous keyboard layouts and customizations. It provides advanced features such as key remapping, macros, and layers, catering to both enthusiasts and professional users seeking tailored keyboard functionality. QMK's community-driven development model encourages contributions and adaptations, fostering a robust ecosystem of compatible keyboards and add-ons. Over the years, QMK has expanded its support to include ARM-based microcontrollers and diverse hardware configurations, ensuring its relevance in the evolving landscape of mechanical keyboards and DIY keyboard enthusiasts.

There are 2 ways of flashing the firmware on the keyboard. I have tried both. I will explain both ways to do it here.

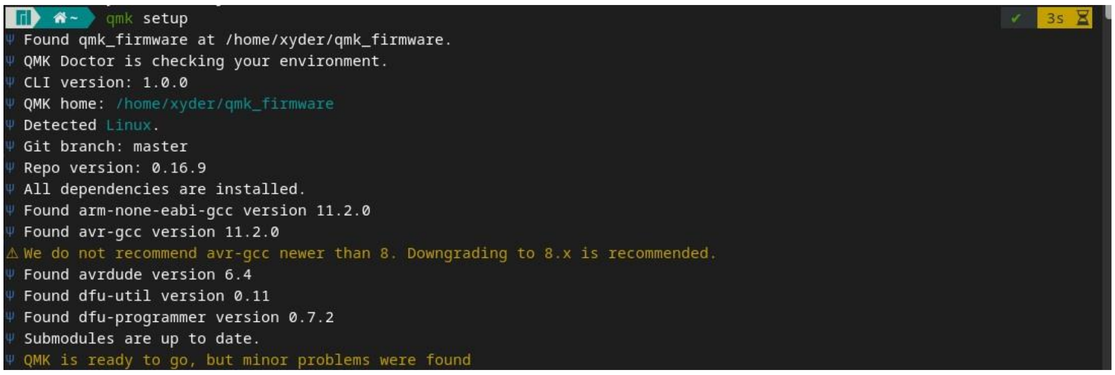
The first way is to compile the firmware on the device offline. For this we will be needing to install different programs depending on the operating system. QMK has an amazing documentation that explains all of this in further detail.[3]

I have tried this method on Linux (Manjaro). We first make sure that we install pip from the package manager.



```
 ~  pip install jsonschema==4.4.0                                                                    1 ✘
Defaulting to user installation because normal site-packages is not writeable
Collecting jsonschema==4.4.0
  Downloading jsonschema-4.4.0-py3-none-any.whl (72 kB)
     |████████████████████████████████| 72 kB 205 kB/s
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in /usr/lib/python3.10/site-packages (from
jsonschema==4.4.0) (0.18.1)
Requirement already satisfied: attrs>=17.4.0 in /usr/lib/python3.10/site-packages (from jsonschema==4.4.0) (21.4.0)
Installing collected packages: jsonschema
Successfully installed jsonschema-4.4.0
```

Figure 5-6: installing pip



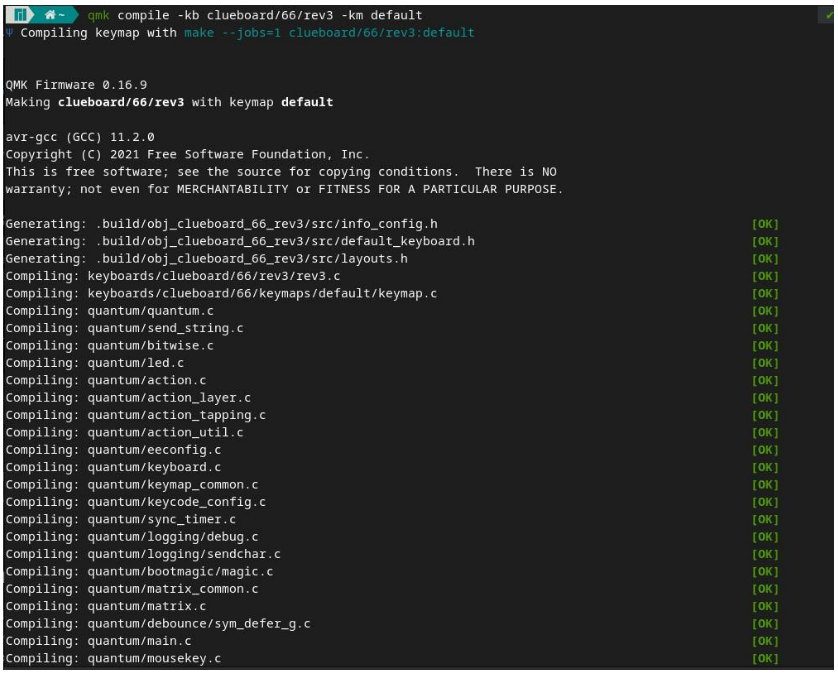
```
qmk setup                                                                    ✔  3s ⌛
Ψ Found qmk_firmware at /home/xyder/qmk_firmware.
Ψ QMK Doctor is checking your environment.
Ψ CLI version: 1.0.0
Ψ QMK home: /home/xyder/qmk_firmware
Ψ Detected Linux.
Ψ Git branch: master
Ψ Repo version: 0.16.9
Ψ All dependencies are installed.
Ψ Found arm-none-eabi-gcc version 11.2.0
Ψ Found avr-gcc version 11.2.0
⚠ We do not recommend avr-gcc newer than 8. Downgrading to 8.x is recommended.
Ψ Found avrdude version 6.4
Ψ Found dfu-util version 0.11
Ψ Found dfu-programmer version 0.7.2
Ψ Submodules are up to date.
Ψ QMK is ready to go, but minor problems were found
```

Figure 5-7: setting up QMK

Then download the firmware files for QMK from their official github repository.[4] We can do it by using the command “`yay -S qmk-git`”. This will download all the necessary files we will be needing for the QMK firmware compilation. We finally use the command “`qmk setup`” to install all the necessary files from the files that we downloaded.

This open-source project already comes with default resources required to compile most if not all open-source keyboards. For our purpose we will find our files in the path qmk-firmware/keyboards/handwired/dactyl-manuform/5x7.

This will be helping us compile the firmware we will be needing for the keyboard.



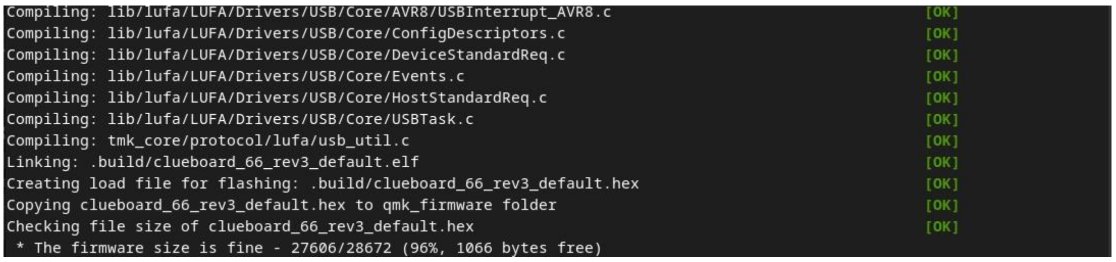
```
qmk compile -kb clueboard/66/rev3 -km default
Ψ Compiling keymap with make --jobs=1 clueboard/66/rev3:default


QMK Firmware 0.16.9
Making clueboard/66/rev3 with keymap default

avr-gcc (GCC) 11.2.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Generating: .build/obj_clueboard_66_rev3/src/info_config.h                  [OK]
Generating: .build/obj_clueboard_66_rev3/src/default_keyboard.h             [OK]
Generating: .build/obj_clueboard_66_rev3/src/layouts.h                      [OK]
Compiling: keyboards/clueboard/66/rev3/rev3.c                               [OK]
Compiling: keyboards/clueboard/66/keymaps/default/keymap.c                  [OK]
Compiling: quantum/quantum.c                                                [OK]
Compiling: quantum/send_string.c                                            [OK]
Compiling: quantum/bitwise.c                                                [OK]
Compiling: quantum/led.c                                                    [OK]
Compiling: quantum/action.c                                                 [OK]
Compiling: quantum/action_layer.c                                           [OK]
Compiling: quantum/action_tapping.c                                         [OK]
Compiling: quantum/action_util.c                                            [OK]
Compiling: quantum/eeconfig.c                                               [OK]
Compiling: quantum/keyboard.c                                               [OK]
Compiling: quantum/keymap_common.c                                          [OK]
Compiling: quantum/keycode_config.c                                         [OK]
Compiling: quantum/sync_timer.c                                             [OK]
Compiling: quantum/logging/debug.c                                          [OK]
Compiling: quantum/logging/sendchar.c                                       [OK]
Compiling: quantum/bootmagic/magic.c                                        [OK]
Compiling: quantum/matrix_common.c                                          [OK]
Compiling: quantum/matrix.c                                                 [OK]
Compiling: quantum/debounce/sym_defer_g.c                                   [OK]
Compiling: quantum/main.c                                                   [OK]
Compiling: quantum/mousekey.c                                               [OK]
```

Figure 5-8: An example of using the resources already provided to compile the firmware of a keyboard on the list

For a full list of all the available keyboards we can use the command “qmk list-keyboards”.
We can compile the firmware for our keyboard by using the command
“`qmk compile -kb handwired/dactyl-manuform/5x7 -km default`”

Here in this command the -km and its argument default is signifying the default keymap for this keyboard. Keymap is the config file to fix what key goes where on our keyboard. QMK already comes with default keymaps so all we have to do is compile the firmware with the default keymap. If we need to change the keymap or make a new one we can use the command:
“`qmk new-keymap -kb <keyboard_name>`”

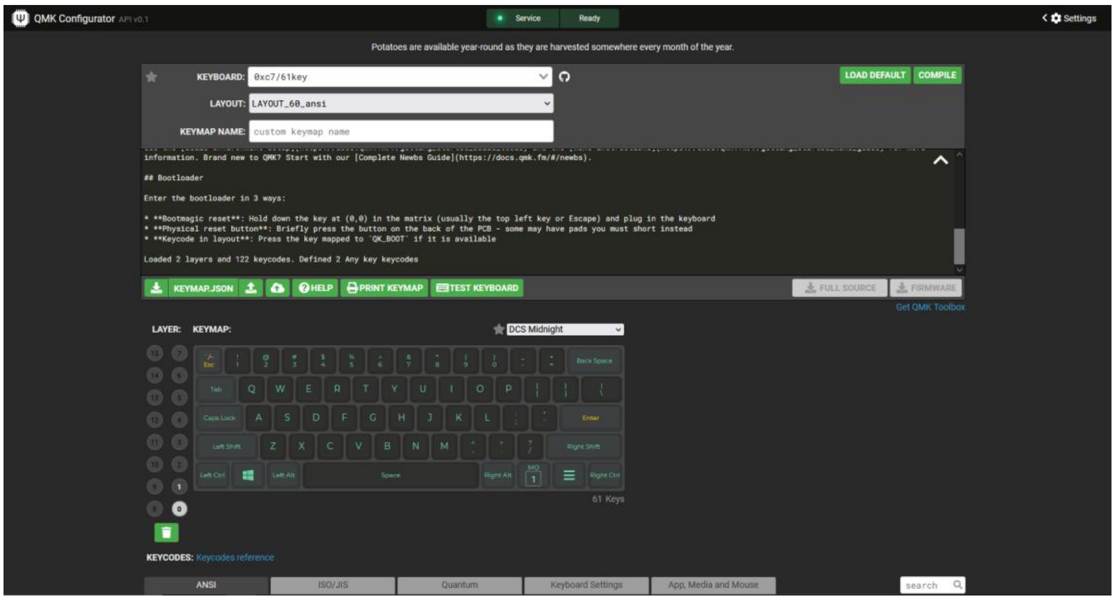
Now since we compiled the firmware, we will get a message after the compilation like so:



```
Compiling: lib/lufa/LUFA/Drivers/USB/Core/AVR8/USBInterrupt_AVR8.c                          [OK]
Compiling: lib/lufa/LUFA/Drivers/USB/Core/ConfigDescriptors.c                               [OK]
Compiling: lib/lufa/LUFA/Drivers/USB/Core/DeviceStandardReq.c                               [OK]
Compiling: lib/lufa/LUFA/Drivers/USB/Core/Events.c                                          [OK]
Compiling: lib/lufa/LUFA/Drivers/USB/Core/HostStandardReq.c                                 [OK]
Compiling: lib/lufa/LUFA/Drivers/USB/Core/USBTask.c                                         [OK]
Compiling: tmk_core/protocol/lufa/usb_util.c                                                [OK]
Linking: .build/clueboard_66_rev3_default.elf                                               [OK]
Creating load file for flashing: .build/clueboard_66_rev3_default.hex                       [OK]
Copying clueboard_66_rev3_default.hex to qmk_firmware folder                                [OK]
Checking file size of clueboard_66_rev3_default.hex                                         [OK]
 * The firmware size is fine - 27606/28672 (96%, 1066 bytes free)
```

**Figure 5-9: Compilation being done by QMK**

Now we demonstrate another method which is more accessible and approachable for someone who is not comfortable with the command line prompt. This time we will be using windows.
Instead of using a command line to setup and compiling a firmware and providing a keymap we can just use a website by QMK to do it all for us. This website is:
[https://config.qmk.fm](https://config.qmk.fm)



Potatoes are available year-round as they are harvested somewhere every month of the year.

KEYBOARD: 0xc7/61key

LAYOUT: LAYOUT_60_ansi

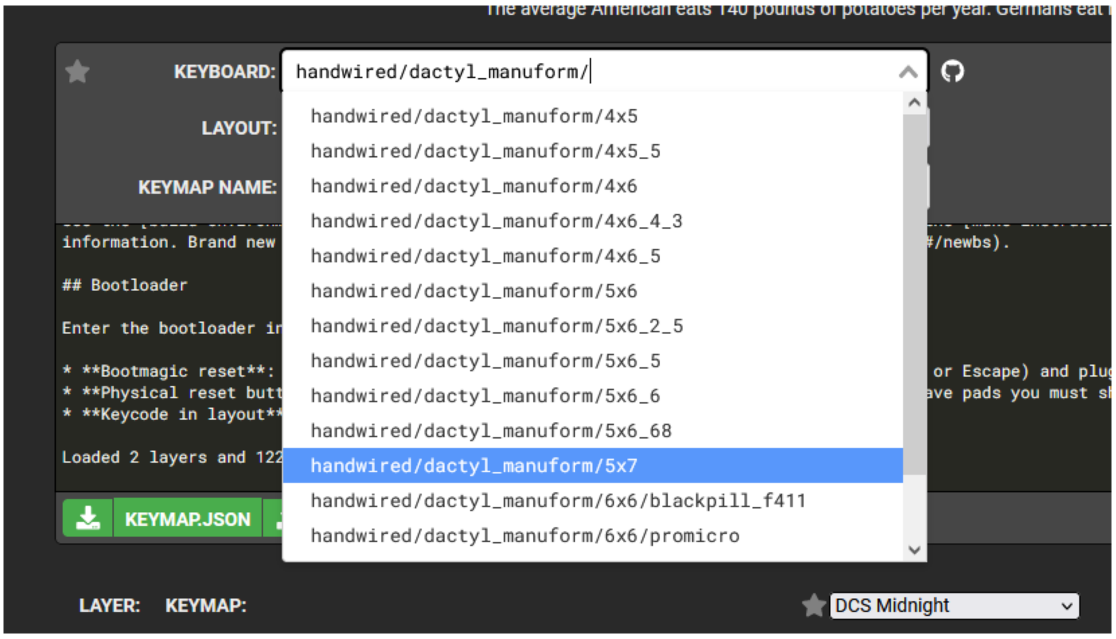
KEYMAP NAME: custom keymap name

```
information. Brand new to QMK? Start with our [Complete Newbs Guide](https://docs.qmk.fm/#/newbs).

## Bootloader

Enter the bootloader in 3 ways:

* **Bootmagic reset**: Hold down the key at (0,0) in the matrix (usually the top left key or Escape) and plug in the keyboard
* **Physical reset button**: Briefly press the button on the back of the PCB - some may have pads you must short instead
* **Keycode in layout**: Press the key mapped to `QK_BOOT` if it is available

Loaded 2 layers and 122 keycodes. Defined 2 Any key keycodes
```

DCS Midnight

Figure 5-10: When you first enter the QMK configurator website

We select the type of keyboard we are building at the first drop-down menu:



KEYBOARD: handwired/dactyl_manuform/

```
handwired/dactyl_manuform/4x5
handwired/dactyl_manuform/4x5_5
handwired/dactyl_manuform/4x6
handwired/dactyl_manuform/4x6_4_3
handwired/dactyl_manuform/4x6_5
handwired/dactyl_manuform/5x6
handwired/dactyl_manuform/5x6_2_5
handwired/dactyl_manuform/5x6_5
handwired/dactyl_manuform/5x6_6
handwired/dactyl_manuform/5x6_68
handwired/dactyl_manuform/5x7
handwired/dactyl_manuform/6x6/blackpill_f411
handwired/dactyl_manuform/6x6/promicro
```

Figure 5-11: choosing our keyboard model

Then we change the keymap given in the interactive window below if we wish to. For now, I want to compile the firmware using the default keymap. So, we press compile and wait for the file to be compiled. After compilation we press the firmware button to download the firmware. The firmware will be a .hex file in our case.

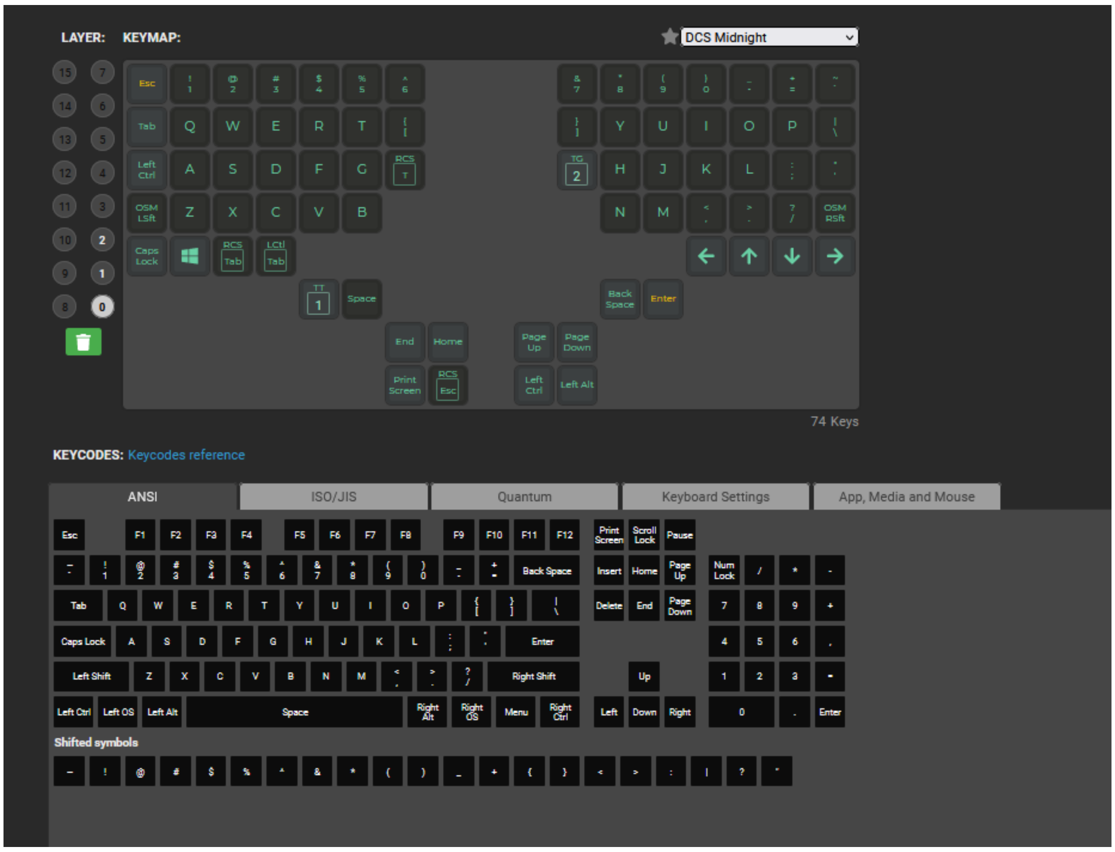
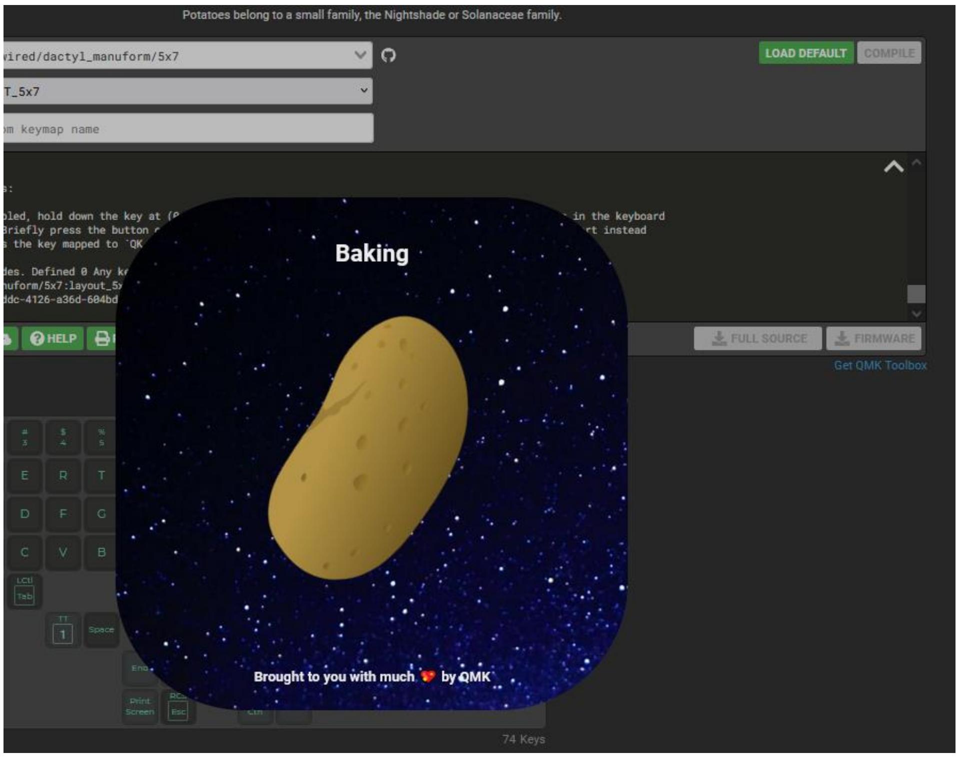


Figure 5-12: We can edit the keymap here


Potatoes belong to a small family, the Nightshade or Solanaceae family.

```
ired/dactyl_manuform/5x7
T_5x7
```

Figure 5-13: After pressing compile we wait for the process to end

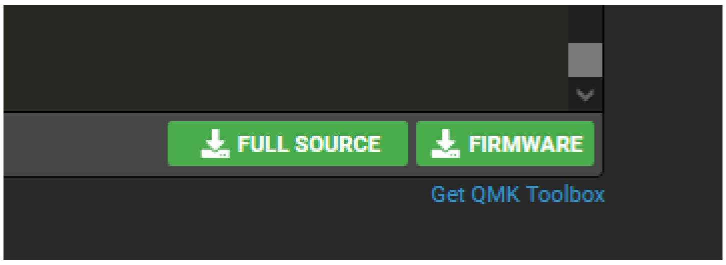


Figure 5-14: we press on firmware to download the compiled firmware

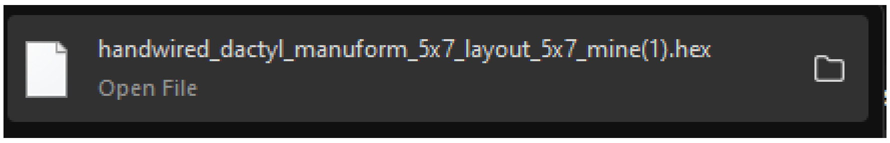
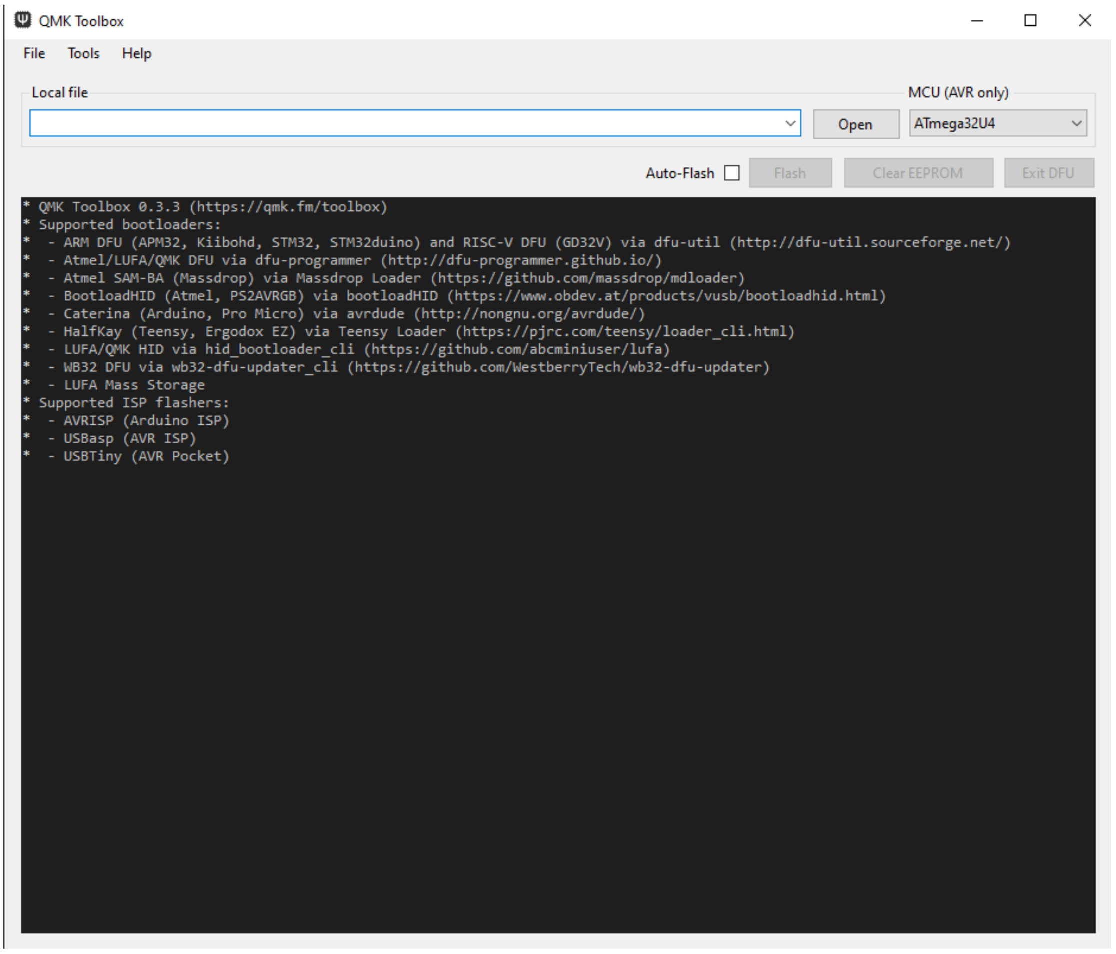


Figure 5-15: we download a .hex file which is our firmware

We now download another QMK tool specifically used to flash the firmware into our keyboard. This tool is called the “qmk_toolbox” can be found on the official qmk_toolbox github repository.[5]

When we install this tool, we will be greeted with the following program:

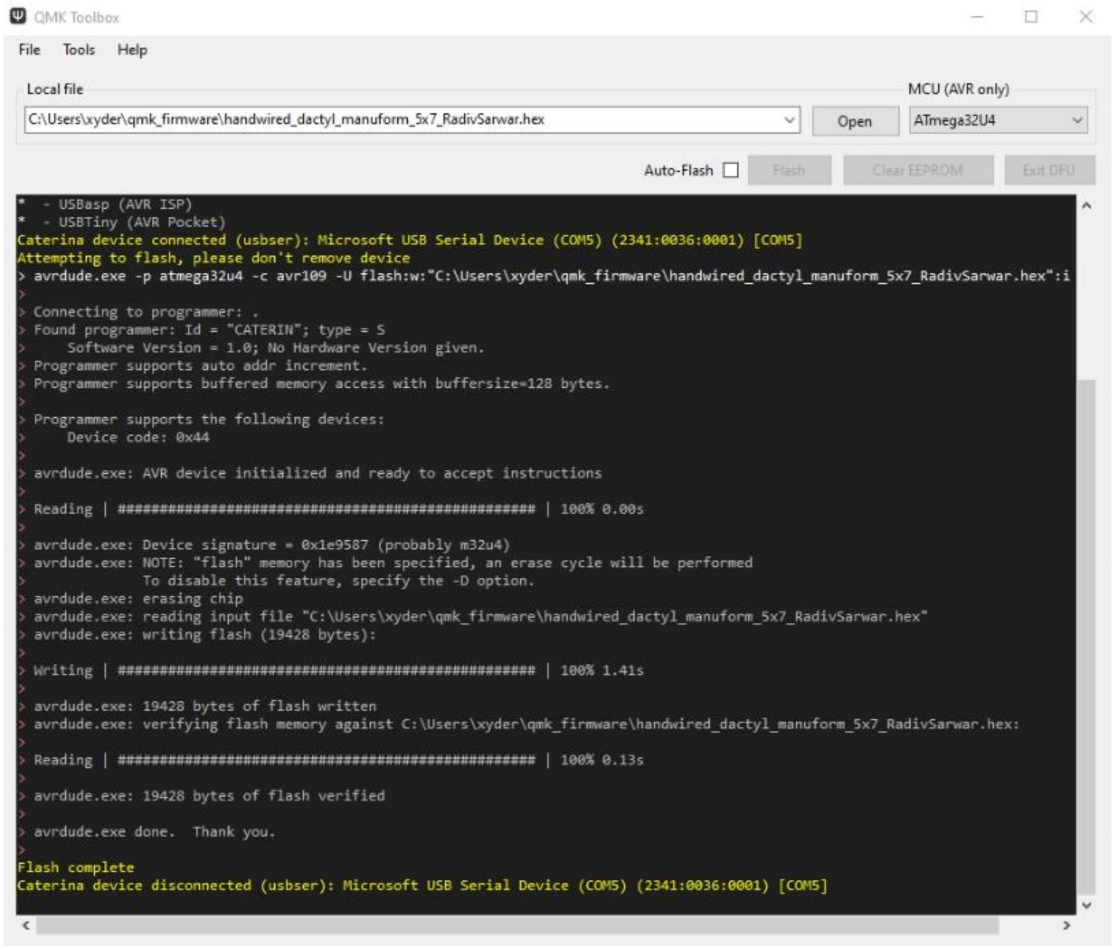


MCU (AVR only): ATmega32U4

```
* QMK Toolbox 0.3.3 (https://qmk.fm/toolbox)
* Supported bootloaders:
*  - ARM DFU (APM32, Kiibohd, STM32, STM32duino) and RISC-V DFU (GD32V) via dfu-util (http://dfu-util.sourceforge.net/)
*  - Atmel/LUFA/QMK DFU via dfu-programmer (http://dfu-programmer.github.io/)
*  - Atmel SAM-BA (Massdrop) via Massdrop Loader (https://github.com/massdrop/mdloader)
*  - BootloadHID (Atmel, PS2AVRGB) via bootloadHID (https://www.obdev.at/products/vusb/bootloadhid.html)
*  - Caterina (Arduino, Pro Micro) via avrdude (http://nongnu.org/avrdude/)
*  - HalfKay (Teensy, Ergodox EZ) via Teensy Loader (https://pjrc.com/teensy/loader_cli.html)
*  - LUFA/QMK HID via hid_bootloader_cli (https://github.com/abcminiuser/lufa)
*  - WB32 DFU via wb32-dfu-updater_cli (https://github.com/WestberryTech/wb32-dfu-updater)
*  - LUFA Mass Storage
* Supported ISP flashers:
*  - AVRISP (Arduino ISP)
*  - USBasp (AVR ISP)
*  - USBTiny (AVR Pocket)
```

Figure 5-16: qmk_toolbox

We select ATmega32U4 in the MCU tab and locate the .hex file we just downloaded. For my build I already have the 2 reset switches ready when I plug in the keyboard. When I plug in the keyboard and press the reset switches immediately the software detects both of the microcontrollers and prepares to flash the firmware we provided. We then press flash and the flashing process begins.


QMK Toolbox

File Tools Help

Local file: C:\Users\xyder\qmk_firmware\handwired_dactyl_manuform_5x7_RadivSarwar.hex — Open

MCU (AVR only): ATmega32U4

Auto-Flash ☐ Flash Clear EEPROM Exit DFU

```
*  - USBasp (AVR ISP)
*  - USBTiny (AVR Pocket)
Caterina device connected (usbser): Microsoft USB Serial Device (COM5) (2341:0036:0001) [COM5]
Attempting to flash, please don't remove device
> avrdude.exe -p atmega32u4 -c avr109 -U flash:w:"C:\Users\xyder\qmk_firmware\handwired_dactyl_manuform_5x7_RadivSarwar.hex":i
>
> Connecting to programmer: .
> Found programmer: Id = "CATERIN"; type = S
>     Software Version = 1.0; No Hardware Version given.
> Programmer supports auto addr increment.
> Programmer supports buffered memory access with buffersize=128 bytes.
>
> Programmer supports the following devices:
>     Device code: 0x44
>
> avrdude.exe: AVR device initialized and ready to accept instructions
>
> Reading | ################################################## | 100% 0.00s
>
> avrdude.exe: Device signature = 0x1e9587 (probably m32u4)
> avrdude.exe: NOTE: "flash" memory has been specified, an erase cycle will be performed
>              To disable this feature, specify the -D option.
> avrdude.exe: erasing chip
> avrdude.exe: reading input file "C:\Users\xyder\qmk_firmware\handwired_dactyl_manuform_5x7_RadivSarwar.hex"
> avrdude.exe: writing flash (19428 bytes):
>
> Writing | ################################################## | 100% 1.41s
>
> avrdude.exe: 19428 bytes of flash written
> avrdude.exe: verifying flash memory against C:\Users\xyder\qmk_firmware\handwired_dactyl_manuform_5x7_RadivSarwar.hex:
>
> Reading | ################################################## | 100% 0.13s
>
> avrdude.exe: 19428 bytes of flash verified
>
> avrdude.exe done.  Thank you.
>
Flash complete
Caterina device disconnected (usbser): Microsoft USB Serial Device (COM5) (2341:0036:0001) [COM5]
```

**Figure 5-17: Flashing the firmware on the microcontrollers using qmk_toolbox**

The software will notify us that the flashing is done and we can now test our keyboard. The process is finished.

## 4.4: Painting

We take some time to paint the keyboard after it is assembled. Originally I did not want to paint the keyboard. But using it, I was not happy with the white color of the PLA used for 3D printing. I wanted a unique color.

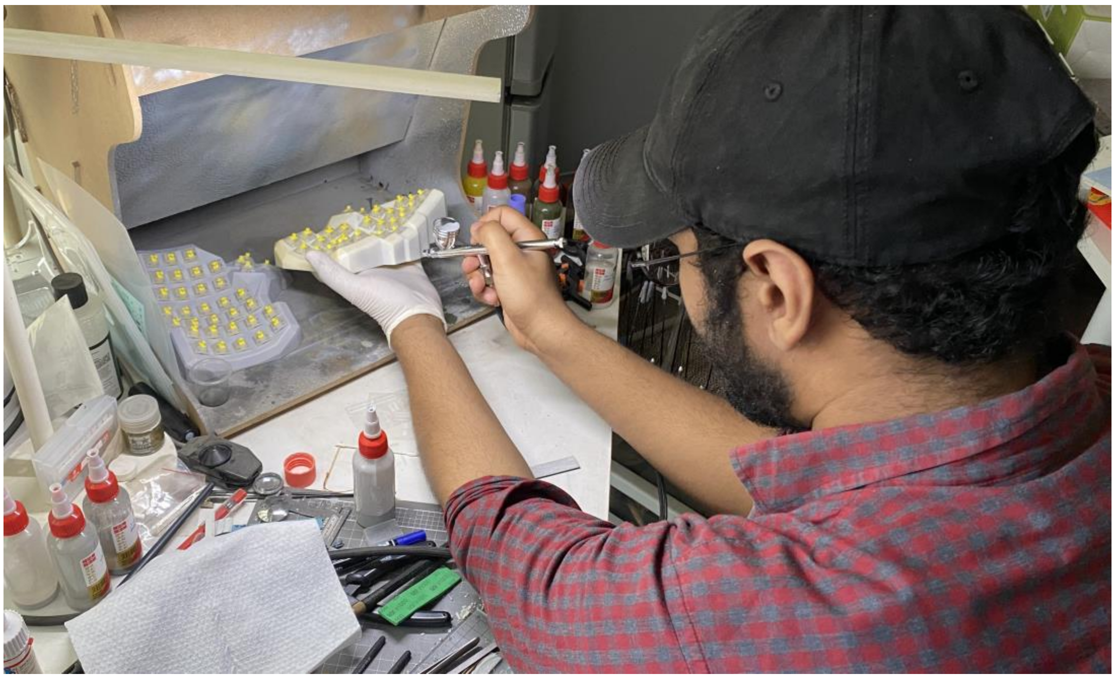


**Figure 5-18: We Prime the board with a neutral gray first.**

I start by priming the keyboard with a neutral gray with some paint thinner and an airbrush. We then let it sit for about an hour and a half to dry.
Then I use the color “meteoric iron” to paint layers on the keyboard using the airbrush.

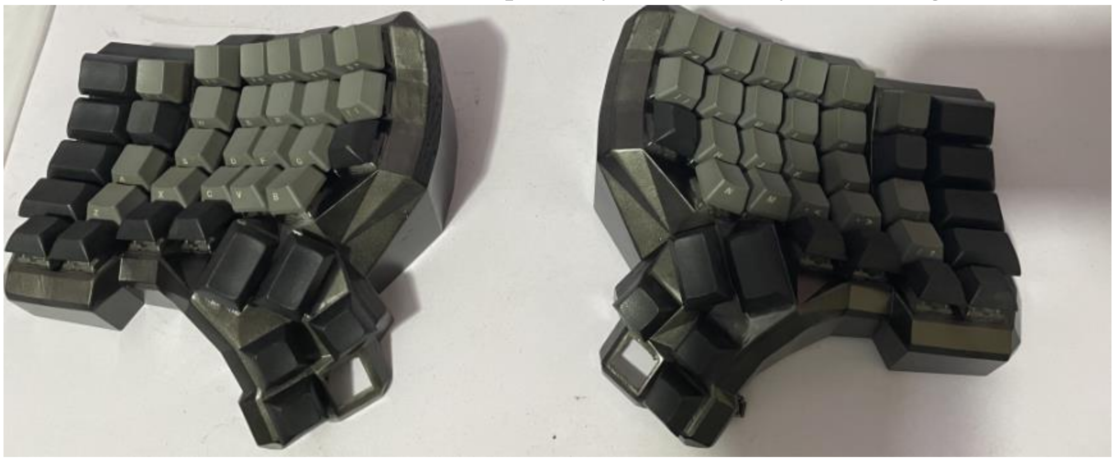


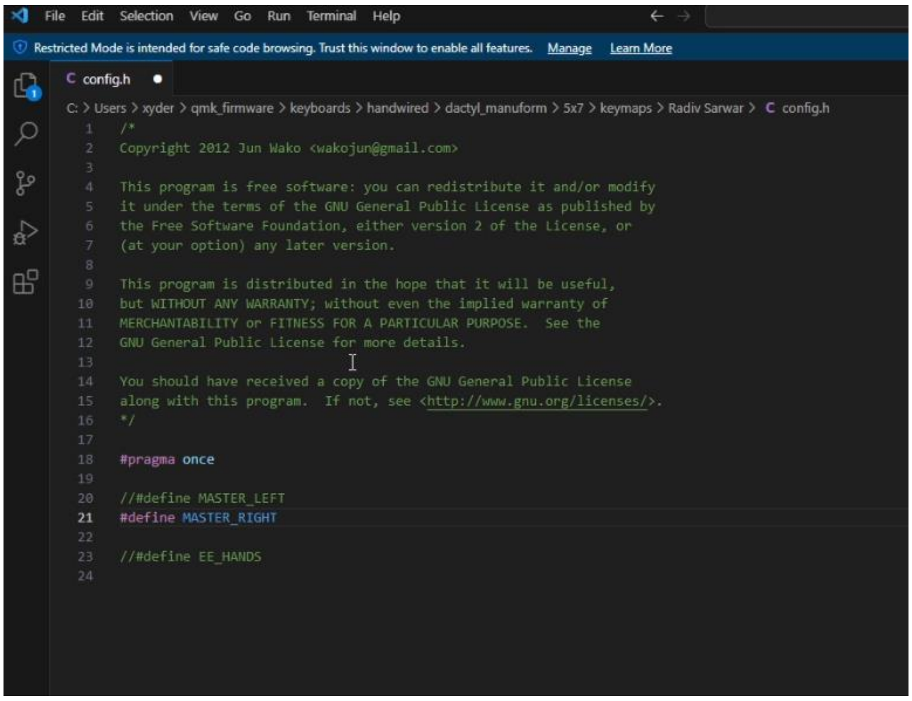
**Figure 5-19: Final result after Painting**

This is what the final results look like. Overall, I am happy with how it turned out.

## 4.5 Issues that arises after completing the process

There were several issues that followed the flashing of the firmware on the keyboard. They will be listed and explained below:

1. Loose Audio cable:
   The first time I flashed the microcontroller, the right side did not get flashed at all. Only the left side was working. After a little bit of troubleshooting, I realized that the audio cable used between the keyboards was loose. After fixing the issue the flash method was done again.
2. Wrong wiring:
   The wiring of the keyboard was wrong. I made a mistake while wiring 2 columns and it switched. One row was not working at all and the other row was showing the wrong keys when pressed. The mistake was fixed after resoldering them to their correct pinouts.
3. The connector of the microcontroller snapped:
   The microcontroller is notorious for snapping under pressure. While using the keyboard the controller connector snapped. As a result, the left micro controller was not possible to connect to the computer. So now I have to use the right microcontroller. But when I connect the left microcontroller, the left side are mapped and act like the right-side keys and the right-side act like left side keys when pressed. This happens because the firmware that was flashed determines the connected controller as the left-side controller and the other side as the right-side controller. To fix this issue I had to manually go to the config folder found in the qmk firmware folder and change the master side to right as shown in the picture and comment out the master side is left to change the settings.



Restricted Mode is intended for safe code browsing. Trust this window to enable all features.

C: > Users > xyder > qmk_firmware > keyboards > handwired > dactyl_manuform > 5x7 > keymaps > Radiv Sarwar > config.h

```
/*
Copyright 2012 Jun Wako <wakojun@gmail.com>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#pragma once

//#define MASTER_LEFT
#define MASTER_RIGHT

//#define EE_HANDS

```

Figure 5-20: Changing the settings in the config file

Using this new config file and the default keymap I compiled the firmware once again. Then the new corrected firmware was flashed to the keyboard. This fixed the issue.

4. The screws used to fit the bottom plates to the shell came loose. The fitting of the screw holes is too loose to hold the screws. This was fixed by using superglue to fit the screw housings properly to the screw holes.

# 6. Personal Summary

This project is a personal project of mine. I was always interested in the keyboard scene. I also developed some wrist pain because of using the computer a lot. Both my interest and the wrist pain led me to a project like this.

I learned about ergonomics and its importance to the health of my body. Using this keyboard for a few weeks helped me quite a lot regarding my wrist pain. Using this keyboard feels very comfortable. The ortholinear layout of the keyboard minimizes finger movement. It takes less time than expected to get used to the layout and typing starts to feel very natural. The learning curve for touch typing is not as steep as one would expect if they know how to touch type already.

This keyboard also forces you to type with all of your fingers. Since each column is engraved in a concave manner to house each digit of the hand it is very natural to use all the fingers to type. This promotes proper typing etiquette and helps build healthy muscle memory.

The most challenging part of this project was to solder and hand wire all the connections between the switches. I was already accustomed to soldering but this entire process tested my patience and endurance. Especially when it came to the miswiring with the two columns. Diagnosing that problem was the most challenging problem in the project. After 2 days of trying everything, I finally found out that 2 columns were mis-wired and that was causing the issues. Even though the mistake caused a lot of headaches it made me study the circuit more deeply and helped me understand the innerworkings of the device. This made me appreciate the simple yet functional design of the device. Solving the problem gave me a feeling of satisfaction that matches very few experiences in my life.

This project also made me appreciate the open-source community. There are a lot of cool open-source projects that look very interesting. The projects with good documentation help people like me to begin our journey into a community that is very welcoming. I want to work hard contribute to this open-source community one day as a thank you to people who work for free and provide excellent software and education through great design and stellar documentation.

# 7. Outlook

In this electronic internship, I chose to design an open-source keyboard that focuses on ergonomics and improves upon the concept of modern keyboard design.

First of all, this internship made me realize how important ergonomics is in day-to-day life. This cannot help one in a single day. But good habits and proper posture adds up in the long term to ensure we live a healthy life doing what we love doing.

Secondly, this internship gives me a deeper understanding of electrical circuits and their efficient design. Accounting for 70 keys with just a handful of pins on a microcontroller inspires a sense of respect for engineers who design efficient and robust circuits and PCBs.

In addition, this internship also exercised my endurance when it came to soldering 70 switches with each other and my problem solving and trouble shooting skills when it comes to electrical design. This helped me learn the common pitfalls when following a circuit design and what to do to prevent such mistakes in the future. This helped me not be afraid when I see a circuit schematic and break it down into pieces that are easier to understand.

Finally, this internship made me understand the importance not giving up when facing bugs and issues that seems impossible to solve. I gained more confidence making this project and have faith in myself when it comes to figuring things out and approaching concepts slowly and deeply to understand them innately.

To sum up, this electronic internship is a valuable opportunity for me to learn electrical knowledge and software command line. This taught me to read and follow proper documentation and it also taught me to recognize good documentation and how important it is when it comes to open-source projects.

# 8. Reference

[1] https://github.com/markdhooper/CMD-dactyl-manuform
[2] https://typetheory.co.uk/
[3] https://docs.qmk.fm/
[4] https://github.com/qmk/qmk_firmware
[5] https://github.com/qmk/qmk_toolbox/releases

# 9. Acknowledgements

First of all, I would like to express our sincere thanks to my tutor, Mr. Yuan Xiaoqing, for his guidance and most importantly his patience when it comes to students like me. When it comes to international students, we don’t always expect to find an understanding person who goes the extra mile to help students in any way shape or form. But Mr. Yuan Xiaoqing never once not helped me whenever I needed his help. He is one of the professors who will always greet you with a smile an hear your problems and do his best to solve them. This inspires me to be better. I want to say a heartfelt thank you to the Professor for his kindness and his hard work.

At the same time, I also want to thank the teachers of the School of Mechanical Engineering for providing me with the basic knowledge and experience when it comes to soldering and other components. This helped me to complete this project taking proper safety precautions. I am very grateful for the education imparted by my school.