Department of
Mechatronic
Engineering

ERGONOMIC KEYBOARD

L8 BEng (Hons) Project on the research, development and design of an ergonomics focused Keyboard.

Project by
C. Krammel
S00241117

Ollscoil
Teicneolaíochta
an Atlantaigh

Atlantic
Technological
University

ATU.ie

Abstract

As technology and computers in particular have become a part of daily life from home use to the workplace an increased focus on ergonomics and injury prevention is required. While computers have evolved across many years, one of the primary ways of interaction with such devices in form of the keyboard has remained largely unchanged.

This project will focus on gathering relevant data from both existing research and users to create a keyboard with focus on ergonomics in mind. The relevant firmware will be developed using C Language for Arduino IDE with an optional goal to transition to the open-source QMK keyboard firmware project, using C, Java and Python. Relevant hardware will be prototyped alongside using Autodesk Fusion360 and 3D printing based on component choices made.

The prototype will be developed and improved upon using user feedback and learnings from research on keyboard ergonomics. Based on the requests or suggestions from users additional functions may be incorporated to aid with the ergonomic use of the keyboard as well as the workflow of the participating users

Declaration

I declare that I am the sole author of this thesis and that all the work presented in it, unless otherwise referenced, is my own. I also declare that this work has not been submitted, in whole or in part, to any other university or college for any degree or qualification.

Acknowledgements

I wish to thank my wife Louise, for supporting me through the entirety of this project and putting up with my being sat at the office computer or hunched over electronics in need to be soldered and tested for hours at end.
Further I wish to thank all participants of the survey for their active participation, suggestions and constructive feedback on the development of this project. A special thanks to Imelda, additionally providing continued support throughout the project.
Lastly I wish to direct a large thanks to the community behind the open-source QMK and VIA projects, as well as the reddit based r/olkb community. Without your support, guidance and efforts this project would likely have never seen the light of day.

Acronyms/Abbreviations

- ANSI: American National Standards Institute
- DMM: Digital Multimeter
- FDM: Fused Deposition Modeling, the most widely available method of 3D printing. Alternatively known as FFF (Fused Filament Fabrication). Using a thermoplastic filament to create a 3D model. A slicing software is used to most commonly break the Z axis of the model into singular 2D layers, that if stacked assemble the processed model. The sliced code of the processed model is then inputted to a compatible printer. The model will then be printed feeding the thermoplastic through a extrusion hot-end onto a print-bed layer by layer as a 2D print in the X & Y axis.
- GUI: Graphical User Interface, the visual elements of a software to enable users to interact with software and hardware through easy-to-understand buttons and menus rather than code.
- HID: Human Interface Device, common identifier used by PC operating systems for input devices like mice and keyboards
- I2C: Inter-Integrated Circuit, a two wire communication protocol using a serial data line (SDA) and a serial clock line (SCL) to allow connection of low speed peripherals and microcontrollers.
- IP: Intellectual Property
- ISO: International Organization for Standardization
- LED: Light emitting diode
- OS: Operating System
- PCB: Printed Circuit Board
- PDA: Personal Digital Assistant, a small handheld computer similar to, but existed before and eventually replaced by modern smartphones
- RGB: Red Green Blue, a common lighting control system, where color and brightness is controlled via the separate light-emitters for each of the three base colors being assigned values from 0 (off) to 255 (full brightness) to achieve the wanted result.
- RSI: Repetitive Strain Injury
- STL: File name acronym for stereolithography. A process in which a 3D model is broken down into triangulated surfaces and commonly used for 3D Printing.
- TKL: Ten Key Less
- TRRS: 4Pin 3.5mm Audio Jack, TRRS refers to the layout of connection-points Tip, Ring 1, Ring 2 & Sleeve
- U: Keyboard key-size unit. 1U = 18 x 18mm. Increased size, like for a 2U Backspace key, indicate an increase in the X axis, i.e. 2U = 36 x 18 mm

Table of Contents

List of Figures

Literature review: Ergonomic focused Computer Keyboard Design

Introduction: Keyboards as the human interface device to Technology

“Computers are something we take for granted nowadays; it is almost impossible to visualize an office without one. “– Anne D. Kroemer, Karl H.E. Kroemer in ‘Office Ergonomics: Ease and Efficiency at work, second Edition’ [1]

The above quote is an accurate representation how embedded computers in many forms are within the current everyday life, be it computers in offices at work or for a multitude of personal uses from business to entertainment at home. Our societies have adopted computers and their uses so much into our everyday lives, we even carry them in our pockets in the shape of smartphones everywhere we go.
As such we need an interface to interact with and allow for interaction with said devices. The oldest such interface that has prevailed from the inception of the computer till this day is the keyboard. The keyboard allows the user to directly translate letters and symbols into digital form, as such providing the interface to communicate, program and interact in many other ways with the computers of today.

Due to the dominance of these devices and their everyday use ergonomics is a major concern. A study in the American Journal of Preventive Medicine carried out between 1994 and 2006 showed an >700% increase of computer related injuries over the study period [2].
Similarly the research article ‘Computer Injuries and its Prevention in Scientific Ways’ by HM Arun Kumar from 2019 showed not only a 90% report rate of computer related injuries across the subjects of varying professions, but also a concerning lack of awareness of the seriousness and potential long-term impacts of computer related injuries [3].

To address these concerns many workplaces have adopted policies to educate their workforce on ergonomic best practices. This has shown to be an improvement in the awareness of the workers, as well as a reduction in reported injuries [4]. But along with the education must come also the access to ergonomic tools to support the users in realising an optimised and safe working environment, as such the keyboard, as an essential bridge between human and computer, is a critical point to be observed and closer analysed during this review for ergonomic best practices and design.

The Keyboard: A brief history

The inception of the keyboard can be attributed to the creation of the first commercially successful typewriter. In 1873 Christopher Latham Sholes and his backer James Densmore sold the manufacturing rights to their Sholes & Glidden Typewriter to the company E. Remington and Sons, with the aid of Remington’s Mechanics, the keyboard layout was optimised and the Ccmpany eventually arrived at the layout nowadays known as QWERTY (see Figure 1) [5].

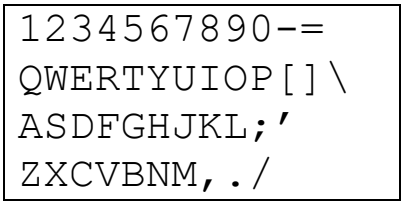


Figure 1 - Remington QWERTY Layout

The exact reasoning for the arrival on this Layout is lost to history, the most common theory is attributing this decision to limitations of the mechanics within the typewriters and the layout being designed to separate out common letter combinations to prevent a jam of the typewriter mechanism, when typing at higher pace [6] [8].

Others attribute the development of the Remington QWERTY layout to feedback from Morse code receivers. Explaining this assumption on the common confusion between the letter “Z” (… . in American Morse code) and the letters “SE” (… for S and . for E in American Morse code) especially at the beginning of a word, due to lack of context. As such the “Z”, “S” and “E” keys were positioned close together to allow for faster error correction [5].

The next development happened thanks to Charles Krum, whose efforts between 1907 to 1910 made the teletype system practical for everyday use. This system together with the telegraph, punch-cards and adding machines introduced changes to the QWERTY layout to incorporate communication technologies and faster entry of numbers.

This led to the incorporation of the popularized QWERTY layout into the early 1946 Eniac computer, which used a punch-card reader as its input and output as well as the 1948 Binac computer, that used electro-mechanically controlled typewriters to allow the inputting of data directly onto magnetic tape. Since then, the layout remained nearly unchanged as computers developed and became more common both in the workplace and for personal use, the QWERTY layout established itself as the default keyboard of choice due to its wide adoption [7].

Even with the modern development and popularisation of handheld computers like the Personal Digital Assistant (short: PDA) and Smartphones, the QWERTY keyboard prevailed in digital form, as such QWERTY remains the most common keyboard layout and nearly unchanged from its inception over a century ago, as shown in Figure 2 below.

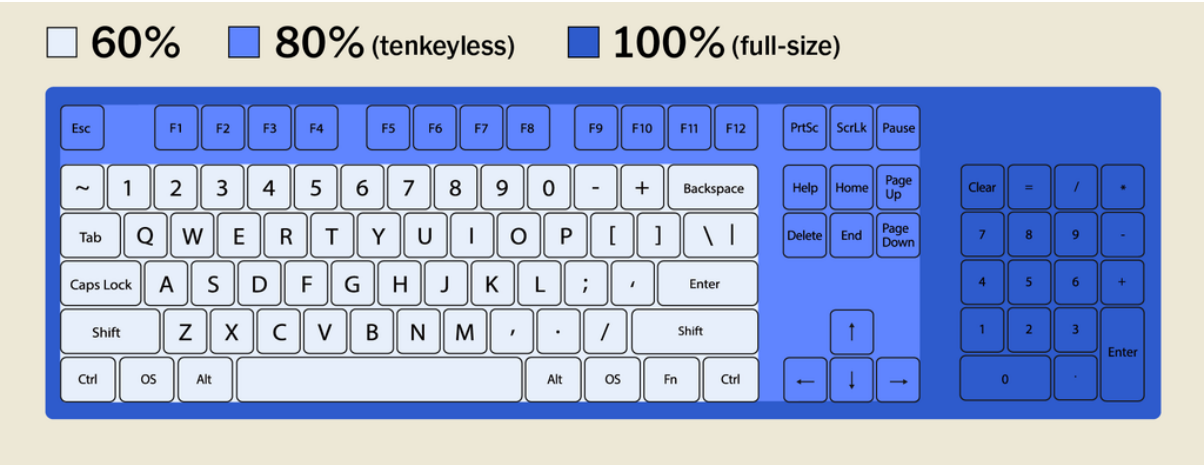


Figure 2 – common QWERTY Keyboard layouts [13]

Common injuries for typists

The most common injury type associated with keyboard use is repetitive strain injury (short RSI). These injuries most commonly are a direct result of prolonged deviations from the neutral wrist position, the below Figure 3 shows the types of deviation from neutral commonly associated with keyboard use and the maximum recommended healthy deviation in Degrees for prolonged operation [9].

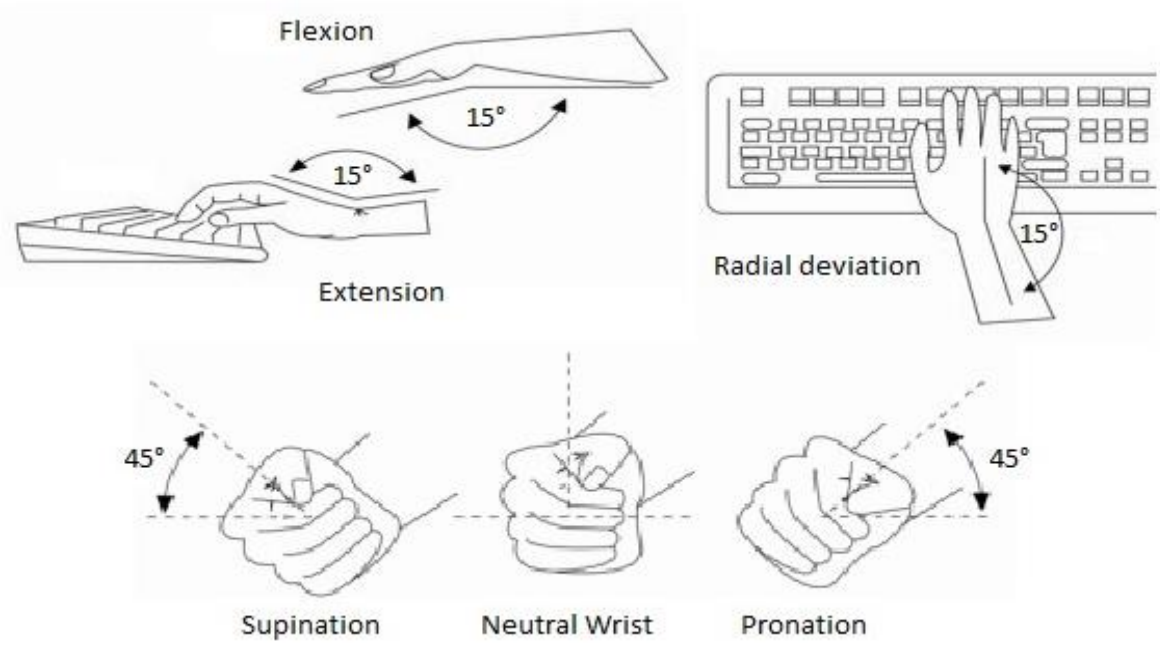


Figure 3 - Wrist extension types during keyboard use [9]

Prolonged use at overextended wrist position as well as over extension of the fingers can lead to a multitude of injuries within the RSI spectrum, including but not limited to passive wrist flexion, dorsiflexion impairment, forearm muscle pain, and ligamentous hypermobility of finger joints [10].

These types of injury are directly affecting the musculoskeletal or nervous system and the high risk associated with these afflictions is their manifestation in initial mild discomfort. It is thought these injuries can worsen due to their repetitive nature, while the afflicted person at a similar rate becomes accustomed to the discomforts. This poses a high risk of what initially was a discomfort and relatively easy to address, to turn into a long-term injury with severe consequences, such as the aforementioned limitation to wrist movement or the requirement of surgery [11] [12].

Typing Styles and their ergonomics

The use of the keyboard is a large contributing factor towards the ergonomics of a keyboard as well, as such the three most common typing styles are known as “Hunt and Peck”, “Touch Typing” and “Hybrid Typing”.

Hunt and Peck is the easiest to adapt and most common with inexperienced typists and users new to keyboards. The typist relies on the method of searching out each key by the printed-on letter usually the user will only use one or two fingers at any time to do the inputs. As a direct result of this process, the input speed is highly limited and errors are likely, as the user focuses the attention on searching out letters instead of focusing on the written text that is being inputted [16]. This method does not favor ergonomics due to the large amount of arm and hand movement required [17].

The Touch typing method of keyboard operation has the user focus on the inputted text as it is being typed, instead of observing the keyboard itself. This technique requires training and developing of muscle memory. The user not only must memorise the keyboard layout, but also be able to operate the keyboard without directly observing the keys. The typist will also use all fingers to operate the keyboard using this method, to aid this method the keys “F” and “J” can be usually found on a QWERTY style keyboard with either a different curvature to these particular keys or a small raised element to the key to allow the user to align their index fingers without the need to view the keyboard. The center row of the keyboard “A” to “ ‘ “ is known as the home-row (see Figure 4) within this typing style and used as the resting position for the users hand [18]. While this is a challenging method to adopt, the advantage is reduced risk of errors, as inputted text can be directly observed as it is being written [18], as well better ergonomics, as the movement of fingers is being reduced and over extension and flexion are being kept at a minimum [19].

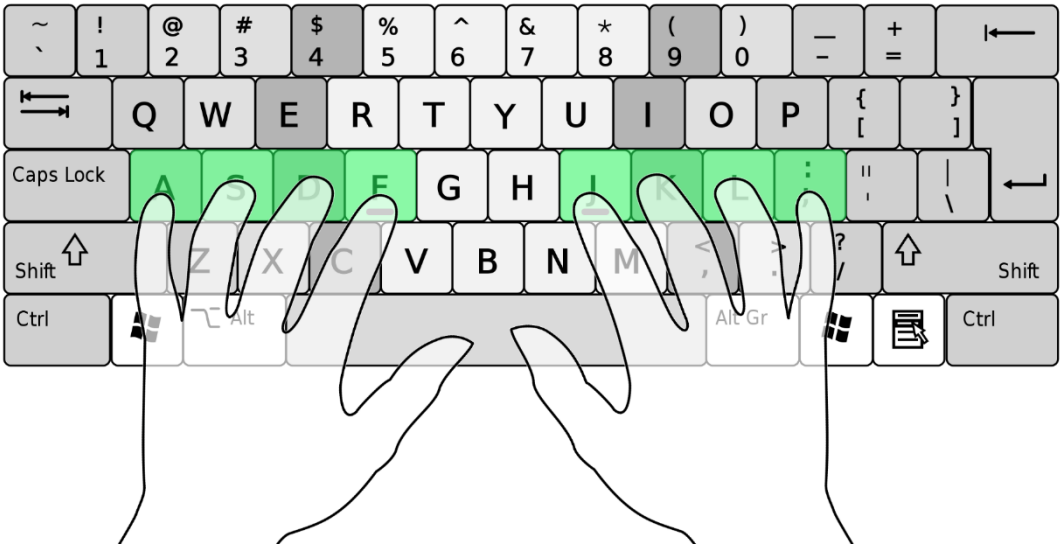


Figure 4 - Home Row on QWERTY Keyboard [21]

The hybrid typing method describes a broad range of input methods between the “hunt and peck” and the “touch typing” methods. Users may use multiple fingers for inputting, partially adhere to the home row resting position or type with only occasional glances at the keyboard. The Study on “How We Type […]” by Anna Maria Feit et al. [20] from May 2016 found that self-taught typists will adapt hybrid typing styles that function comparably to touch typing in both movement and efficiency, even if not using all fingers for typing.

Physical Keyboard Layouts and ergonomics

The default QWERTY Keyboard

Having prevailed for over a century in its current form and across multiple technologies the QWERTY keyboard has planted its roots well within technology. The layout is widely adopted across multiple professions and taught commonly in schools, further cementing its foothold, as such the layout benefits from the familiarity to the user as well as ease of access, however at the time of its inception ergonomics was not a design concern. Potentially some design influences may even have been to intentionally slow typists, in an attempt to deal with mechanical limitations of typewriters [8] [13]. The default QWERTY keyboard as shown in Figure 2 previously has some adaptations to access additional function keys (see Figure 2 80% variation) and a Num-Pad (see Figure 2 100% variation). There are also many variations based of the QWERTY layout to accommodate different Languages. Though the overall layout remains mostly the same, as can be seen in Figure 5 below, showing a German QWERTZ keyboard layout, the letter and number keys are of the same size of 18 x 18 mm within the layout and as such define the single unit key size (short 1U size). Function and modifier keys vary in size to aid in containing the layout within a rectangular shape, but may be modified, if for example space is at a premium like in portable laptop computers.

The layouts themselves most commonly are separated into two larger categories:

- The American National Standards Institute's (short ANSI) QWERTY, as identifiable in Figure 2 by the 2U length left shift key and single row 2U length enter key
- The QWERT Layout as per the International Organization for Standardization (short ISO), as seen in Figure 5 in the German QWERTZ variation, identifiable by the shortened 1.25U length left shift key and the 2 row enter key in shape of a 180° rotated "L"

Figure 5 - German QWERTZ layout [15]

The Curved and Split Keyboard

The default QWERTY layout forces the user to twist their wrist slightly outwards, this radial deviation is to compensate for the width of the users body with the keyboard placed center in front of the user. As a result the QWERTY keyboard has been changed to accommodate the natural human hand position via splitting the keyboard down the center [23]. While not a direct change to the functional layout of QWERTY, this allowed the left and right side to be angled inwards. This is to reduce the radial deviation occurring when typing on the default QWERTY layout (see Figure 3 & Figure 4) by angling the keyboard halves rather than have the user angle their wrists to suit the keyboard (see Figure 6). This split commonly allows for 2 types of ergonomics focused keyboards:

- The Curved Keyboard, that while split down the middle of the keys still is one unit but with a separated section, which is angled to accommodate the respective hand used to type on it.
- The Split Keyboard that differentiates from the curved model, completely separates the halves of the keyboard for right and left-hand use. The benefit of this approach lays in the ability of the user to position each half best to their needs. But as a result of the split extra cables may be required to connect the units with another.

The benefit of these keyboards is that a user already used to touch typing can adopt the slight change in layout with relative ease. Non touch typists on the other hand may not profit as much from this design and experience a longer adoption period [24]. The ergonomic benefits in the short term are not noticeable [23], but in the long term this design reducing wrist and hand pronation and radial deviation may aid in reducing the risk of RSI [24].

Figure 6 - From left to right: Curved Microsoft Ergo Keyboard, ErgoDox EZ Split Ortholinear Keyboard & Kinesis Gaming Split Ergo Keyboard [22]

Ortho Linear

The ortholinear Layout changes the default QWERTY layout from a staggered to a linear grid layout of the keys (see Figure 7). This is done to reduce the diagonal travel of each finger to the keys outside the home row, by keeping the distances uniform and in line. Additionally the modifier keys, which are usually of larger size than 1U size and hit with the pinky finger, which is the weakest, are redesigned to be the same 1U size as the Letter-Keys [26].

Figure 7 - Staggered default QWERTY (top) compared to Ortholinear QWERTY (bottom) [25]

Other ergonomic considerations during Keyboard design

Another factor of consideration during keyboard design is the choice of translating the keypress into a signal. In general this is achieved via the closure of a circuit at the point of the respective key, while there are many variants of electro mechanics developed to achieve this, the most common three are:

- Rubber Dome or Membrane
  On most Keyboards of this design a rubber material sits atop two thin membranes of printed circuitry with a dome shape at the point of each key. When the key is depressed, this design creates tactile feedback to the user, due to the collapse of the dome under the key, at the same time the fully collapsed dome presses the printed circuit lines beneath together and closes the circuit at the point of the keypress. This allows the keyboard to then translate the detection of the closed circuit, via a matrix, as the digital signal for the depressed key, on release of the key the dome will return to its original shape, lifting the key back to its resting position. Due to its lower cost this is the most common implementation on keyboards. Though the rubber dome leaves no option for adjustment to user preferences, requires full depression to register a key and limits user maintenance as membranes and rubber dome mats commonly are keyboard specific [29].

- Mechanical Switches
  Similar to the above rubber dome a matrix is used to scan for key activation via a closed circuit, though instead of a rubber sheet pressing together exposed circuitry, purpose made mechanical switches close the circuit instead. These switches commonly have a spring-operated stem in the center, which closes a metal leaf contact, when depressed. The benefit is that the feel, sound and even the force required to activate the switches can be

adjusted to the user's preferences. Keyboards of this design can also offer benefits like reparability as the keyboard can be designed to allow the user to easily replace defective switches or change switches based on preferences.

- Butterfly or Scissor-Switches [30].
  Primarily designed for low profile use in laptop keyboards, these switches combine a mechanical element to allow key movement and printed circuit sheets to register the keypress. Similar to rubber dome the user accessibility for maintenance is limited though the mechanical element of the key travel allows for a small level of customization of keys by the manufacturer [29].

A Study by Wimberly S. Hoyle et al. regarding user preference showed that the most preferred travel distance for key activation to register within the studied group was 1.6 to 2 mm [27]. The same study also highlighted that travel distances shorter than 0.4 mm decreased the accuracy of the typists. A separate study by Hans Brunner and Rose Mae Richardson further highlighted within their respective group of participants that users preferred a tactile feedback on key activation and disliked linear key activation with low resistance [28].

Conclusion

In conclusion there are many approaches to optimize a keyboard for human ergonomics. That being said, the dominance of the QWERTY layout, in particular in its international adaptations makes large changes difficult and faces low adoption rates due to the familiarity with QWERTY.
Given the dominance of keyboards in most if not all professions, as well as personal use, the importance of ergonomics cannot be denied. The long term ailments resulting from incorrect and prolonged over-use can have major consequences up to and including injuries requiring surgery. As such it is important that an ergonomic keyboard focuses on the aspect of both familiarities, by not straying too far from the default QWERTY layout, as well as ergonomics to reduce the risk of RSI by minimizing extensions of fingers and wrists beyond the neutral position as much as possible.

Given the findings in this review one may make the conclusion that a curved or split design keyboard may be the best compromise between ergonomics and familiarity in the QWERTY Layout. Further based on the studies done on user preferences the choice of mechanism, with which to register a key press should be a mechanical switch with a medium activation weight, tactile feedback and an activation point between 1.6 to 2 mm, additionally one may consider the integration of layered key functions similar to a laptop keyboard to minimise hand and finger travel and minimize the overall keyboard size.

The ortho linear layout could be an interesting consideration for ergonomics focused keyboard with minimal modification to the QWERTY layout. Though, additional research on user preferences would be required to gauge likelihood of adoption of this changed design.

Methodology

Research of existing products

Introduction

An analysis of existing products available on the open market and their functionality was required to gain a better understanding of the demand and to assist with the design process. Due to the large number of the products on the market the research was to focus on a small sample size of keyboards from large manufacturers, that were widely available and designed to address the ergonomic needs of the end-user. Part of this analysis was also involving the breakdown of technologies used, where possible and to outline their functionality. This was to allow an informed decision on the subsequent methodology and design process for the project of creating an ergonomic keyboard. Lastly a small survey was used to gather direct user feedback on different keyboard layouts for a gauging of end-user preferences.

Keyboard Technologies

Most if not all keyboards, independent of the type (Membrane, Mechanical …) use a Matrix system to register the keystrokes input by the user. This method allows a microcontroller to minimize the number of pins required to monitor multiple keys, as a pin is only assigned per column or row instead of a per key basis. When a key press occurs the corresponding row and column are connected and the Microcontroller can thereby detect the keypress.

Where the Types of Keyboards differ is in the physical implementation of this Matrix system.

Membrane Keyboards

Within a membrane keyboard the Key-Matrix is monitored via two membranes containing circuitry. Underneath each key the top and bottom circuit sheets will have an exposed contact, seperated only by an airgap created by the middle sheet, which has a hole in place of the contact (see Figure 8). When the key is pressed by the user, the pressure applied to the top sheet forces the contact to close with the bottom sheet, thereby closing the circuit at that point and allowing the microcontroller in use to detect the corresponding key.

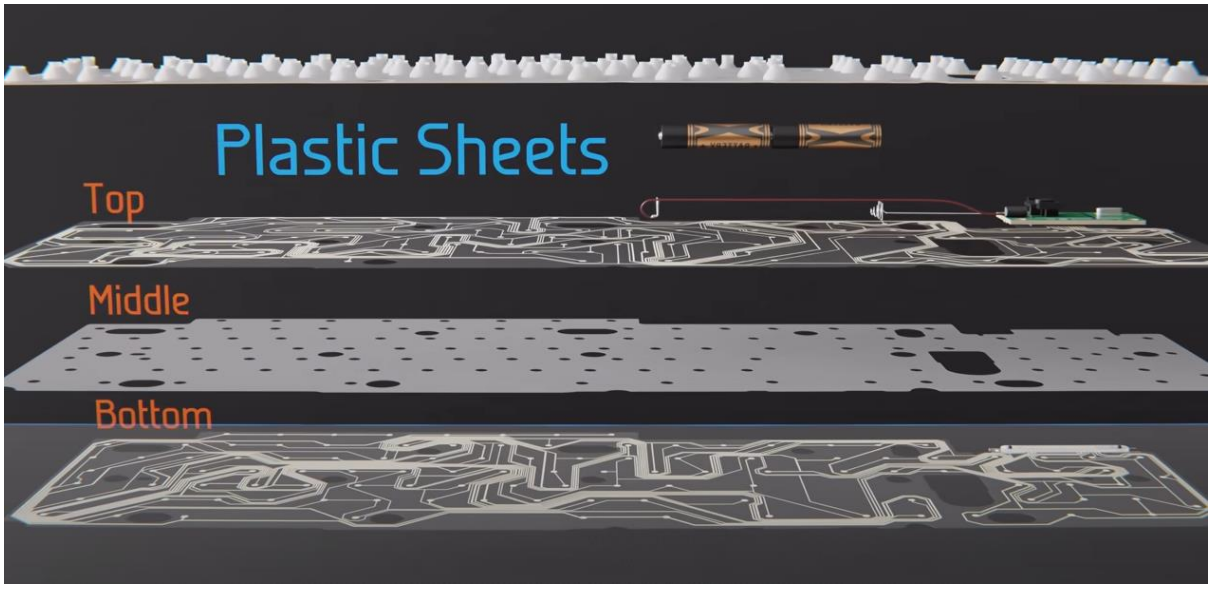


Figure 8 - Membrane Keyboard Circuitry [31]

To discern the keys 2 methods can be used:

1. A base voltage is applied to one of the two membranes representing the matrix columns and the other membrane representiong the rows is used for the microcontroller to poll for any detected inputs. When an input is detected a turn off is pulsed through the pins of the membrane carrying the base voltage. Thereby a keypress can be detected based on which column turning off causes the signal to turn off on the detecting row. The advantage of this method lays in lower energy consumption, but this is at a trade off of speed and limitation to how many keys can be detected at once.
2. Similar to method 1 the membranes are used to represent the rows and columns of the matrix respectively. But this time no permanent base voltage is applied, but rather a signal high voltage pulse is continously shifted column by column. The polling of the microcontroller on the rows can therfore clearly decern the input as only one column at a time carries the high signal voltage. While the dectetion is much faster using this method, the continues signal voltage shifting comes at a higher energy cost.

Mechanical Keyboards

Mechanical keyboards commonly use a matrix to accommodate the detection of Keypresses. This method allows minimising the required pins on a microcontroller by arranging the switches along a row and column based matrix, then polling a high signal through either rows or columns and detecting a closed circuit on the other. Often a diode alongside the mechanical key switch is used to prevent false keypresses being registered. The diode prevents feedback through pressed keys connected to the same row or column being polled, resulting in incorrect keystrokes being registered. A simple example of a three by three matrix is shown below in Figure 9 for a nine key setup, like numbers one to nine on a calculator.

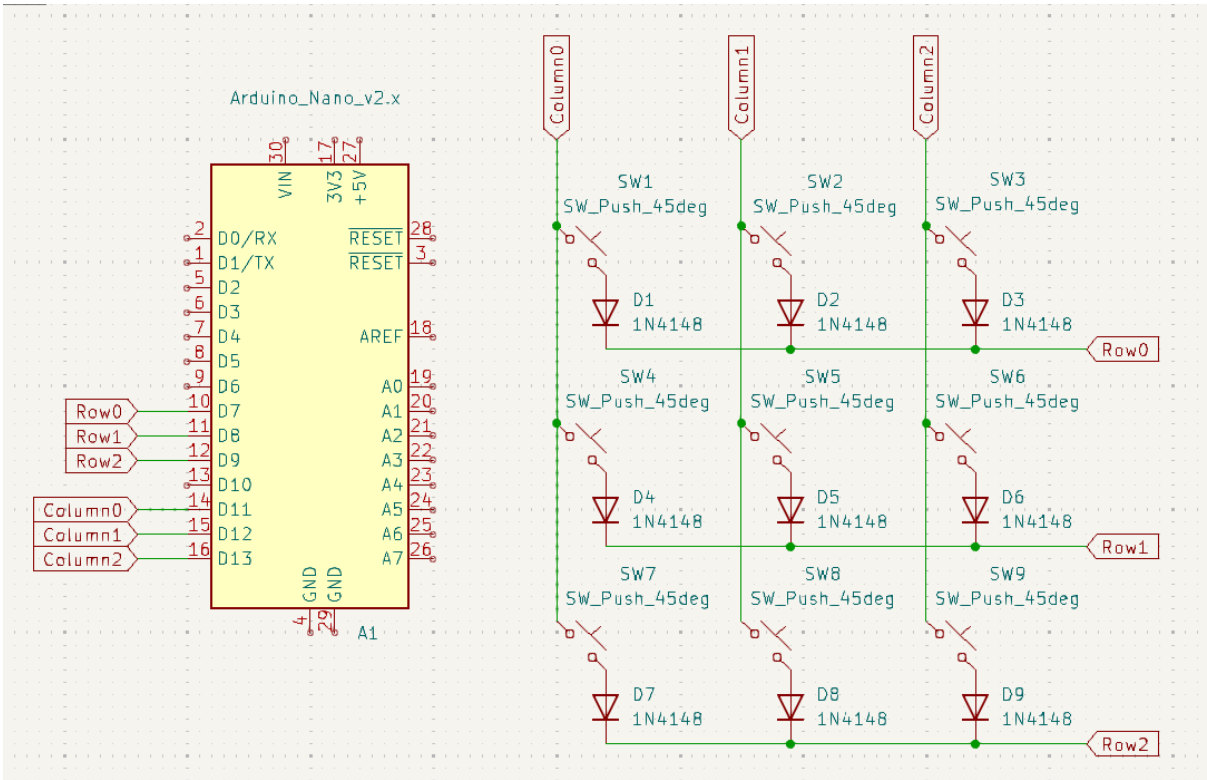


Figure 9 – Nine key Keypad 3x3 Matrix [32]

Microsoft Sculpt ergonomic Keyboard

Figure 10 - Microsoft Sculpt Keyboard [33]

Cost: €179.99 (amazon.de 15.Feb.2024)

The Microsoft Sculpt keyboard has been widely adopted as an ergonomics keyboard, alongside similar designs from other large manufacturers like Logitech and Kensington. The device features a built-in wrist-rest and offers a curved design, which physically separates the key-clusters assigned to each hand as for touch

typing use cases. Furthermore the overall shape features a gentle curvature upwards towards the split in the centre over the spacebar. This elevation aims to support the users' wrist by reducing the rotation usually required to accommodate the use of a flat keyboard. It should be also noted that many keys are modified in size and shape to fit the QWERTY layout within the changed keyboard design.

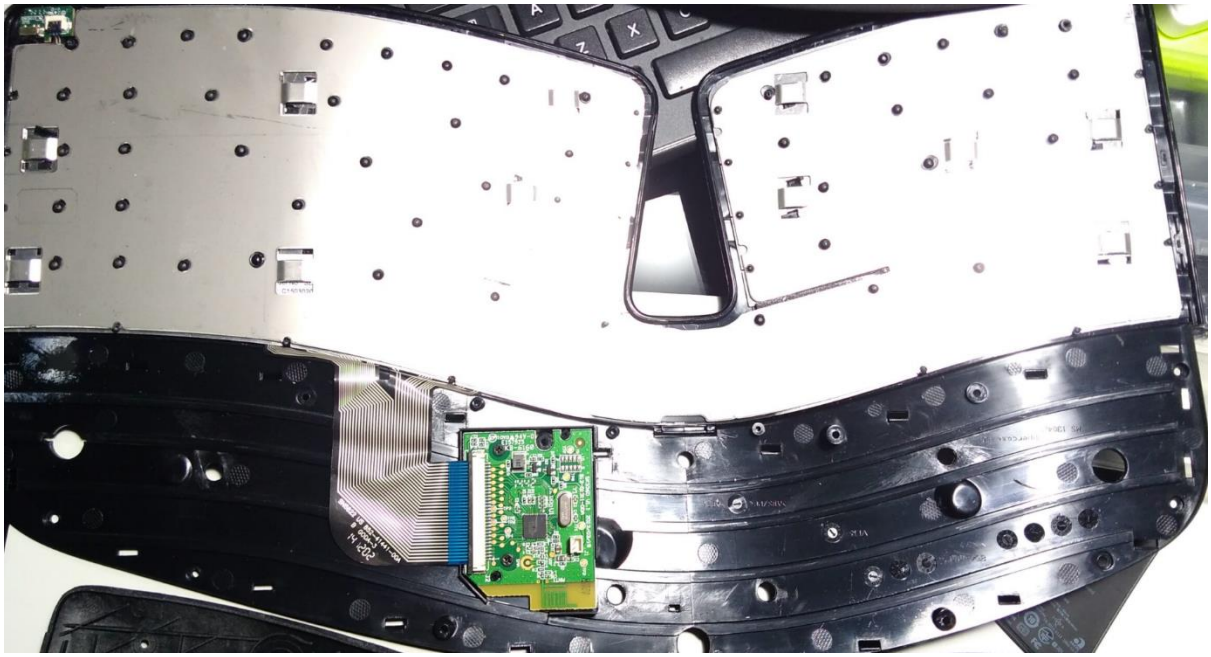


Figure 11 - Disassembled Microsoft Sculpt Keyboard [34]

The Keyboard is based on a scissor switch design that uses 2 membrane layers to close the contacts underneath each key, thereby registering the keypress. A PCB is built into the wrist-rest section of the keyboard housing to monitor the circuitry of the membranes. It can be seen that the PCB features a crystal oscillator and Microcontroller alongside other small components. The microcontroller in use based on its labeling appears to be from the nRF24LE1 family of Microcontrollers produced by Nordic Semiconductor, from the connector it can also be concluded that a total of 30 traces are used for the connection to the membranes. Assuming a logical split based on the keyboard design, this likely translates into a matrix of six rows by 18 columns, with the remaining traces being used for LED indicators for functions like Caps Lock.

Software wise the Microsoft Sculpt is limited to the provided drivers from Microsoft and all functions built into the keyboard are fixed. Reassignment of keys cannot be easily achieved to accommodate user preferences without third party software. Furthermore, the device operates as simple Human Interface Device within the operating system and is as such limited to the given functions a keyboard can have within the default drivers of the used operating system HID drivers.

ErgoDox EZ Split

Figure 12 - ErgoDox EZ Split Mechanical Keyboard [35]

Cost: $325 / €~305 (ergodox-ez.com 17.Feb.2024. Ergodox EZ Original with printed Keycaps, MX Brown Switches, Tilt/ Tent Legs and Wrist rest)

A more niche product developed with a more user focused approach as compared to the mass market product, like the previously discussed Microsoft Sculpt keyboard. ErgoDox developed this product with a multitude of features to allow users to make the keyboard suit them. The design itself is a heavily modified Split Keyboard Layout based on an ortholinear design. Legs on the sides of the Keyboard allow the user to position the Keyboard at an angle most comfortable to the individual person. Further, the split design allows for the positioning of each half at a more ideal position for each hand.

Figure 13 - Customized PCB designed by Dominic Beauchamp for ErgoDox EZ Split [36]

Another display of user focus is the use of Open-Source support for both hardware and software elements of the keyboard. As such users can design their own PCB to accommodate repairs and customisations to user-preferences, as can be seen in Figure 13 above with a user designed PCB to accommodate soldered mechanical switches. The keyboard is also based on a non-proprietary and widely available Teensy 2.0 microcontroller, as well as an IO-Expander to accommodate additional features like LED backlighting; due to the mechanical keyboard design the users can also customize the switches to their needs and liking.

On the Software side ErgoDox continues the user focus via integration of the Open-Source QMK Keyboard Firmware. This Open-Source Project allows users to easily modify the keyboard layout through a web based GUI., as such users can not just reassign keys and even layer functions on a single key through key combinations, but also completely change the layout, should the need arise to accommodate international layouts or user customized layouts.

User preference research regarding Layouts

Based on the previous findings, it was determined that the keyboard layout is largely influenced by user preference, a survey was created to determine the initial layout for the Prototype for this project.

Alongside the previously discussed keyboards additional layouts were created using the open-source software Keyboard Layout editor [37]. The various keyboard layouts were provided to the participants asking for feedback on comfort, ease of use and functionality. The participants were also asked to describe briefly their use of a keyboard within their workflow as well as their typing style.

The following figure depicts the various layouts provided as part of the survey in addition to the previously discussed ErgoDox EZ Split in its default setup and the Microsoft Sculpt Ergo keyboard.

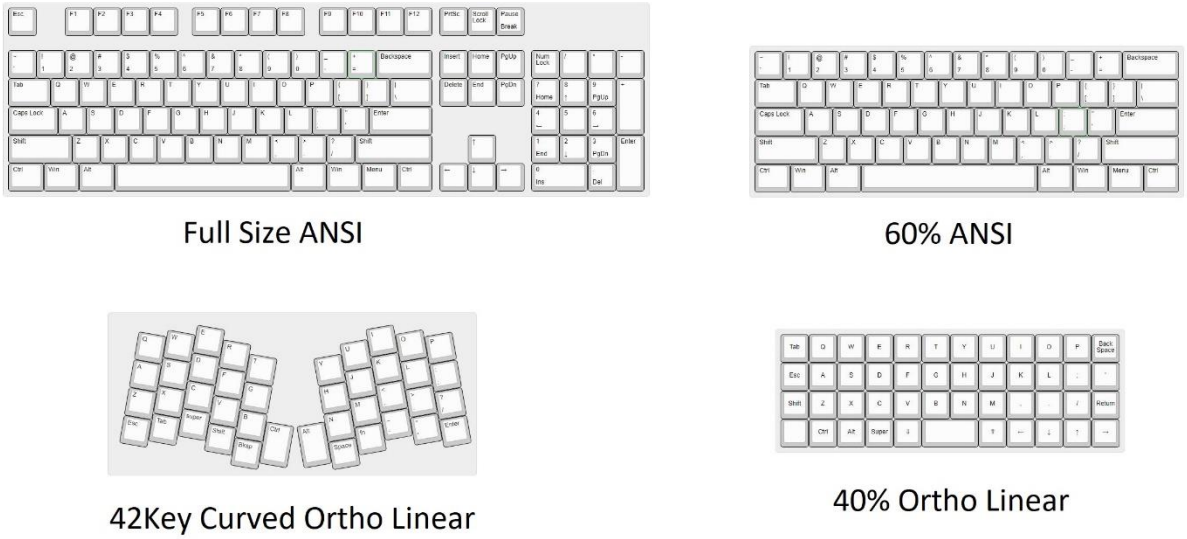


Figure 14 - Surveyed Layouts

The Survey

Thank you for your participation, please understand that this is an anonymous survey and no personally identifiable information will be collected. This survey aims to find user preferences in regards to keyboard layouts, with a focus on ergonomics and usability.

Please state your current Job Title and Typing style (Touch Typing, Hybrid or Hunt and Peck)

Please give a brief description of the uses of a keyboard within your current job and describe your preferences based on the provided layout samples.

Note that this was a verbal survey. Participant provided information and feedback was transcribed as it was given.

Participant Feedback

Participant 1:

Job Role: Head of IT

Typing Style: Touch Typing

Feedback: Within my role a keyboard is an essential part of the job, be it to write emails, reports or to interact with the multitude of systems used. It is often beneficial to have a portable device as frequent moving between locations makes larger layouts like the full size ANSI troublesome. The 60% is generally a good layout and common in smaller Laptops, that cannot accommodate a Num-Pad. The smaller layouts like 42Key and 40% can be beneficial in confined spaces, but the limitations in easily accessing frequently used key combinations and even simple keys can be an issue. A split layout like the ErgoDox is much appreciated as it cannot just accommodate space restrictions but also allow the spacing out of the hand position during prolonged typing. As such a 60% based layout in a split design is usually a preference of mine, curved keyboards can be beneficial too in a fixed location, but personally I often find myself making mistakes due to the oddly sized keys to accommodate the design.

Participant 2:

Job Role: Accountant

Typing Style: Touch typing

Feedback: The keyboard has heavy use in my workflow due to the amount of client emails I send, of course, it also is used for digital record keeping nowadays. The small size keyboards are, personally speaking, uncomfortable and I find it more than challenging to access frequently used keys that are often hidden behind key combinations to accommodate the smaller design. Due to the amount of numbers that have to be input as part of this job, having a Num-Pad is essential. The 60% keyboard seems interesting as it allows me to free up space on my desk which is great, though if I would have to have a separate Num-Pad that benefit is lost. I also assume that with the staggered layout it just wouldn't be the same if the Num-Pad would be built in as a secondary function.

Participant 3:

Job Role: Photographer/ Video & Photo Editor

Typing Style: Hunt and Peck

Feedback: I largely use the keyboard for emails and during editing with certain key combinations within the relevant editing programs to streamline my workflow. The small keyboards like the 42 Key and the 40% are completely unsuitable for this purpose as I can't easily access some of these key-combinations. The curved Microsoft keyboard is not to my liking as it takes me longer to find the keys I am looking for, the same goes for the ErgoDox. I will admit though the benefit of setting

up Macro keys and other functions on the ErgoDox seems like a large benefit. The 60% Keyboard seems interesting, though I am sorely missing my arrow keys that I use for navigation in my editing tools a lot. The full size keyboard is what I am used to, but I have to admit it takes up a lot of space and I rarely use the Num-Pad.

Participant 4:
Job Role: Manufacturing Equipment Technician

Typing Style: Hybrid

Feedback: Keyboards are used both for interacting with machines as well as all kinds of documentation within this job. I can't see the keyboards on the machines themselves to be changed due to how our processes work, but for documentation on our laptops we are usually provided with keyboards. As this is an international company there are so many different layouts of keyboards everywhere and it makes it pretty challenging at times to enter passwords. Having a keyboard to use with the laptop that doesn't change, no matter what keyboard on the laptop I am provided with would be nice, but it would need to be portable. So the full size one is out of the question. The split and curved one as well as the tiny ones also really don't suit me and slow down my typing. The 60% seems a good in-between and is kind of what is found on most of our laptops and machines anyway. But if I could get one that is like the ANSI layout on the machines, so the one on my laptop, which is UK layout, isn't different to those would be great.

Participant 5:
Job Role: Content creator and Streamer (YouTube)

Typing Style: Hunt and Peck

Feedback: I have tested a few different layouts already and settled for the 60% boards with an arrow cluster. When playing games on Stream I don't use the right half of the keyboard, as most games don't or rarely use the keys there, after all you are using the mouse with the other hand and there it is beneficial to have as much space for movement as possible, since I also edit my videos, the arrow cluster is a must have for me. The Split design of the ErgoDox is an interesting option, as I could use only one half when streaming for extra mouse-Space but for editing use both halves and have the ability to also map some keys around as needed for extra functions. Response time is of course also pretty important for games, especially competitive ones, but to be honest in testing many different keyboards, some now with even 8kHz polling, I don't see much of a difference in actual use.

Participant 6:
Job Role: Technical Support Engineer

Typing Style: Hybrid

Feedback: The role requires a lot of documentation and email communication with customers to assist them with troubleshooting. We are working from different locations frequently, be it the work from home days, twice a week office day or customer site as required. As such we carry our equipment around often, devices provided in shared spaces in the office and sometimes at client sites tend to not be looked after that well. I have seen too many keyboards with missing keys, so having your own is great. I personally like the heavily customized small keyboards like the 42Key and the 40% given here. The larger keyboards just take up a lot of space. I got to admit though that sometimes I miss some easier accessed functionality from the larger options. The Split ErgoDox is also a great option, not just because of its customization options but also because the two halves are even smaller in profile to transport than the 40% ortho Linear, which is already pretty small.

Participant 7:
Job Role: Small business owner

Typing Style: Touch Typing

Feedback: I use a keyboard for a multitude of applications, from emails to suppliers and customers alike, to maintaining my website and inventory management. During a previous career as head of an accounting department I have developed symptoms of RSI., as a direct result I abstain from using keyboards in the normal rectangular shape and have since over 5 years only used ergonomic curved keyboards like the Microsoft Sculpt to type at a better position for my wrists and arms. When working with numbers it is great to have a Num-Pad available, but I use a separate one for that case, as most curved keyboards do not feature one. The normal shaped keyboards from the list as well as the miniature ones are therefore not suitable for me. The Split ErgoDox would be a good alternative though, particularly since the layering of functions on keys via key combinations appears to allow for the Num-Pad as a secondary function, the keyboard half combination would better accommodate my hand position, but I may even be able to reduce the need for a separate Num-Pad.

Conclusion

The research showed that the market will accommodate user needs and requests, often at a high cost for lower volume products as seen in case of the ErgoDox Ez Split. Based on the technologies discussed earlier a mechanical keyboard was deemed the best option to prototype. A membrane keyboard would be too complex involving printed circuitry in laminated flexible sheets and preventing easy means of troubleshooting or repair. Taking the user feedback into consideration the 60% layout appeared to meet the preferences of most users. While participants able to touch type showed interest in curved and split Keyboards, those with other typing styles only showed some interest in Split designs and little to no liking for the curved layouts. Small layouts below 60% size were generally disregarded as uncomfortable and lacking functionality.

Design and Prototyping

Layout Design

With previous feedback from surveyed participants considered, a layout based on the 60% keyboard design would likely be most suitable for a broad range of users. Due to preferences the incorporation of the arrow keys would be a benefit as well to many. Lastly many users also stated that they use the Num-Pad for entering of numbers frequently, up to and including having separate Num-Pads for such purpose if the keyboard in use does not feature one.

Figure 15 - Prototype Layout 60% Ortholinear

With the provided feedback the above layout was derived using open-source software keyboard layout editor [37]. The layout features the user preferred 60% Design. The ortholinear key arrangement, reducing most function keys to 1U size also allows for the incorporation of the arrow keys. Furthermore the removal of the stagger in the Key arrangement would also accommodate the possible incorporation of a Num-Pad secondary function in a Num-Pad alike Key-Pattern. Lastly splitting the Layout as shown will accommodate users who prefer the default ANSI layout, but also allow users to experiment with repositioning of each half to their liking, as there was general user interest in split keyboard designs. Lastly the left half could be set to operate independently as inspired by survey feedback of Participant 5.

Component Choice

The primary components for the keyboard required were identified to be Mechanical Keyboard switches, a suitable Microcontroller and Diodes. To accommodate the split layout additionally a connector between the keyboard halves needed to be used and a set of LEDs for keyboard backlighting as an optional feature., as an additional target the overall cost should be kept to around €100 for all components combined

The implementation for the keyboard driver as a C-Code based driver was expected to take up less than 10kb per keyboard half, with the secondary/ right requiring less than the primary/ left, as the left also needed to handle the USB connection to the PC. Due to plans of alternative implementation via QMK for comparison and access to additional features, QMK was chosen to be a better indicator for required Memory size. The QMK documentation [38], as well as, a report on QMK feature size by T. Baart [40] was consulted, arriving at the finding that a program memory size of about 20 to 28kb of usable storage would be required for this projects intended features.

Microcontroller

Due to the possible integration with the QMK open-source software in the future, the repository for the QMK Project was consulted to determine compatible microcontrollers [39]. From this the most commonly available microcontrollers were from either the ARM, Raspberry Pi or Atmel AVR family, given the intended layout a minimum of 13 I/O Pins would be required to accommodate the right keyboard half with 8 Columns and 5 rows. Additionally the controller would need capability to communicate with another controller to accommodate the split design and have enough additional pins for such available, if possible one more pin would be required to drive the optional LED backlights, if implemented.

As such a minimum of 16 I/O Pins with at least one serial capable or two I2C Pins was required. Therefore the most suitable controllers from the provided list of compatible devices were identified as:

- Amtel AVR ATMega32U4
- ARM STM32F103
- Raspberry Pi RP2040

The ATMega32U4 chip [41] is featured on a multitude of Control-Boards like the Elite-C and the Arduino Pro-Micro. The Elite-C comes at a high cost of €27.95 per controller (mechboards.co.uk 08.Mar.2024). At a cost of €19.99 (amazon.de 08.Mar.2024) for a set of 3 the Pro-Micro is more preferable for this application. The ATMega32U4 features 32kB of Program Memory, which could be limiting further features in the future, but for the intended purpose this controller should be well suitable. The Pro-Micro Control board also features a total of 18 I/O Pins, with support for both Serial and I2C communication, another benefit is that the Pro-Micro Control board is available with USB-C, which is a common connector type for Keyboards.

The Raspberry Pi RP2040 chip [42] can be found on controllers like the Elite-Pi and customized iterations of the Pro-Micro control board. The chip is based on a Dual ARM Cortex M0+ design with up to 16MB of Program memory. The Elite-Pi control board features up to 25 I/O Pins and the Pro-Micro variant up to 27. The control boards would be well suitable for this application, also featuring a USB-C connector, dual I2C and Serial communications. Although at a cost of €23.95 for the Elite-Pi and €14.95 for the Pro-Micro variant (mechboards.co.uk 08.Mar.2024), these are some of the most expensive controllers in this list.

The only compatible ARM STM32F103 [43] Control Board listed by QMK is the Bluepill with the special stipulation of specifying the STM32duino Bootloader being used, with a total of 32 I/O Pins, the control board could easily accommodate the needs for this project. It also features Serial and I2C capabilities to allow communication between controllers and the C6 iteration of the STM32F103 has a total of 34kB programming memory available. Though the Bluepill Control Board features an outdated USB Micro B port for USB connections and its size will be a limiting factor for implementation in a smaller sized keyboard like this. The cost of €8.91 (gleanntronics.ie 08.Mar.2024) also puts this chip behind the ATMega32U4 based Pro-Micro controller

Based on these options the ATMega32U4 based Pro-Micro control board was decided on as the best suitable choice, both from a cost as well as from a required feature set perspective. As such the decision was made to proceed basing the keyboard implementation on this control board.

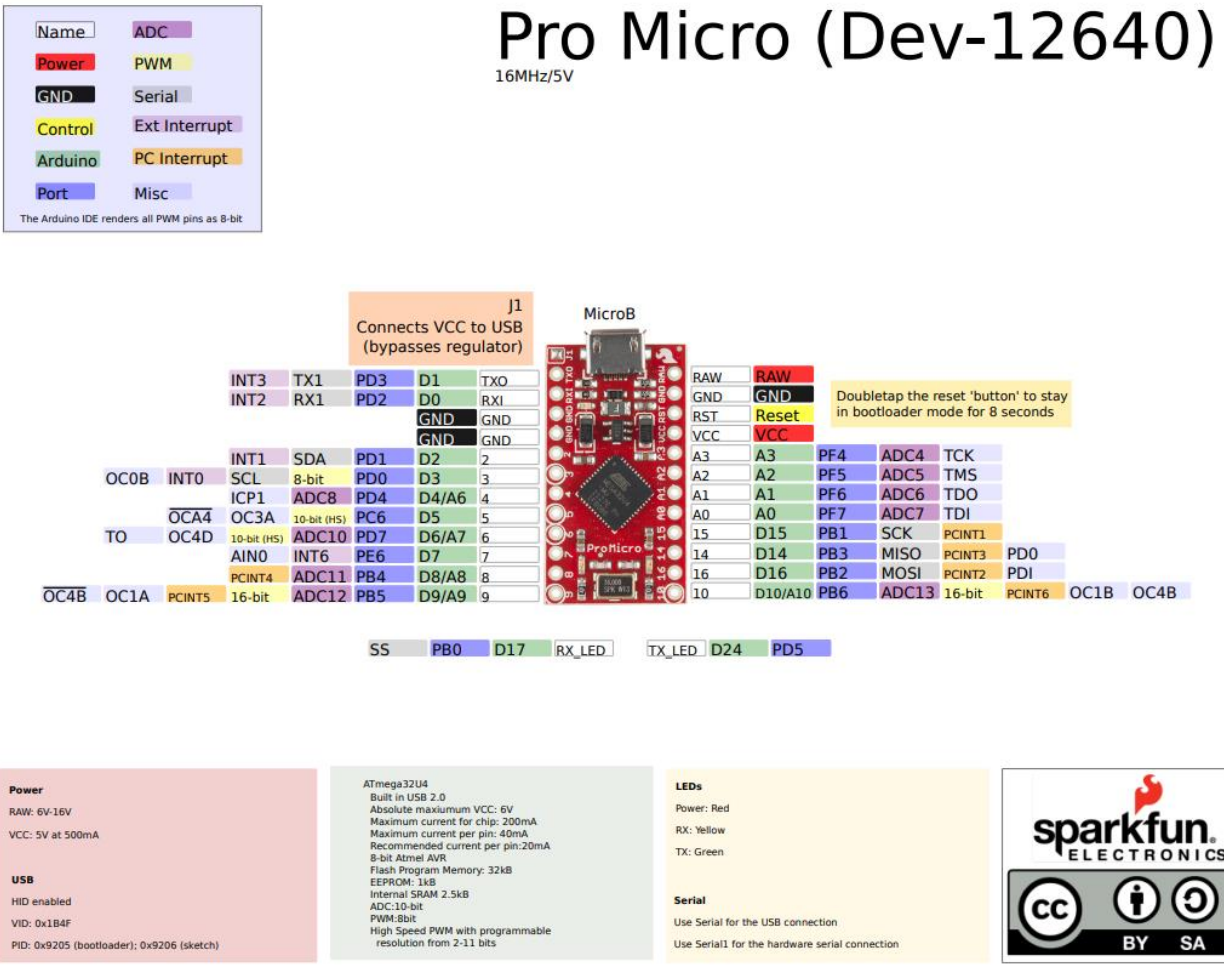


Figure 16 - Pro-Micro Control Board Datasheet [44]

Diodes

Based on the chosen microcontroller, it could be discerned from the ATMega32U4 Data sheet that a pin is detected as high if a voltage over 1.9V is applied and as low if the voltage applied is smaller than 0.9V. Additionally a fast response time would be advisable to accommodate high typing speeds. These criteria could easily be met by a 1N4148 Silicone Diode. These are rated, based on their datasheet [45], for a leakage below 0.54V which will still meet the criteria of a pin reading as low, but also offer an excellent response time at only two to four nanoseconds. Further the low cost of these diodes at only €0.031 if bought at a quantity over 10 (eu.mouser.com 13.Mar.2024) is beneficial to the overall cost target. As such the entire set of 67 diodes to outfit the Keyboard were available for just about €2.1.

Diotec Semiconductor

1N4148, 1N4150, 1N4151, 1N4448

Small Signal Switching Diodes
Kleinsignal-Schaltdioden

I_{FAV} = 150...200 mA V_{RRM} = 50...100 V
V_{F1} < 0.54 V I_{FSM} = 2000...4000 mA
T_{jmax} = 175°C t_{rr} < 2...4 ns

Version 2023-08-17

~DO-35
~SOD-27

SPICE Model & **STEP** File [1])

Marking
Type/Typ

HS Code 85411000

Typical Applications
Signal processing,
High-speed switching
Commercial /industrial grade [1])
Suffix -Q: AEC-Q101 compliant [1])
Suffix -AQ: AEC-Q101 qualified [1])

Typische Anwendungen
Signalverarbeitung,
Schnelles Schalten
Standardausführung [1])
Suffix -Q: AEC-Q101 konform [1])
Suffix -AQ: AEC-Q101 qualifiziert [1])

Features
Very high switching speed
Low junction capacitance
Low leakage current
Compliant to RoHS (w/o exemp.), REACH, Conflict Minerals [1])

Lead FREE

Besonderheiten
Extrem schnelles Schalten
Niedrige Sperrschichtkapazität
Niedriger Sperrstrom
Konform zu RoHS (ohne Ausn.), REACH, Konfliktmineralien [1])

Mechanical Data [1])

Taped in ammo pack	5000	Gegurtet in Ammo-Pack
Weight approx.	0.1 g	Gewicht ca.
Solder & assembly conditions	260°C/10s	Löt- und Einbaubedingungen
	MSL N/A	

Mechanische Daten [1])

Maximum ratings [2]) **Grenzwerte [2])**

Type Typ	Reverse voltage Sperrspannung V_R [V]	Repetitive peak reverse voltage Periodische Spitzensperrspannung V_{RRM} [V]
1N4148	75	100
1N4150	50	50
1N4151	50	75
1N4448	75	100

			1N4148 1N4448	1N4150	1N4151
Max. average forward current Dauergrenzstrom		I_{FAV}	200 mA [3])	300 mA [3])	200 mA [3])
Repetitive peak forward current Periodischer Spitzenstrom		I_{FRM}	500 mA [3])	600 mA [3])	500 mA [3])
Non-repetitive peak forward current Stoßstrom-Grenzwert	t_p = 1 µs T_j = 25°C	I_{FSM}	4000 mA	4000 mA	2000 mA
Max. power dissipation Max. Verlustleistung		P_{tot}	500 mW [3])		
Junction temperature – Sperrschichttemperatur Storage temperature – Lagerungstemperatur		T_j T_s	-50...+175°C -50...+175°C		

1 Please note the detailed information on our website or at the beginning of the data book
Bitte beachten Sie die detaillierten Hinweise auf unserer Internetseite bzw. am Anfang des Datenbuches
2 T_A = 25°C unless otherwise specified – T_A = 25°C wenn nicht anders angegeben
3 Valid, if leads are kept at ambient temperature at a distance of 5 mm from case
Gültig, wenn die Anschlussdrähte in 5 mm Abstand von Gehäuse auf Umgebungstemperatur gehalten werden

Figure 17 - N4148 datasheet [45]

Mechanical Keyboard Switches

The most commonly known and used keyboard switches are manufactured by Cherry GmbH Germany and find wide implementation across many large keyboard manufacturers like Logitech, Corsair, Kensington and many others. This has led to a form of standardization for mechanical keyboard switches based on Cherry's design. Due to this, even though other manufacturers like Gateron, Kailh and Durock may innovate on the internal designs of their switches, the external casing for the mechanical switches remains largely unchanged. This allows companies and users to easily replace switches for many reasons without the worry whether they will fit the pre-existing electronics.

As such the reference dimensions by Cherry were to be used during the design process and accommodate other manufacturers switches as well, based on learnings from the literature review it should be considered that users tended to prefer medium heavy switches with tactile feedback. Therefore, the MX RGB Ergo Clear by Cherry [46] would be a well suited switch. These are broadly available, feature a clear/ semi translucent housing to accommodate backlighting and feature a medium heavy actuation of 65g/ 40cn. These switches can be found for about €0.65 (amazon.de 13.Mar.2024 purchased as a set of 23 switches at €14.99).

Alternatives are also available from Gateron in the form of the Pro Brown style switches, though at a lighter actuation force, which may not be suitable based of the literature review findings. Due to the popularisation of customising keyboards in recent years many manufacturers have also started to offer specialised limited volume switches, focusing on certain customer requests and needs, from varying actuation force to intensity of the tactile feedback of the switch. For the purpose of this project the choice of MX Switches by Cherry is preferable both for the reason of broad availability as well as repeatability, due to the high level of quality control provided by Cherry.

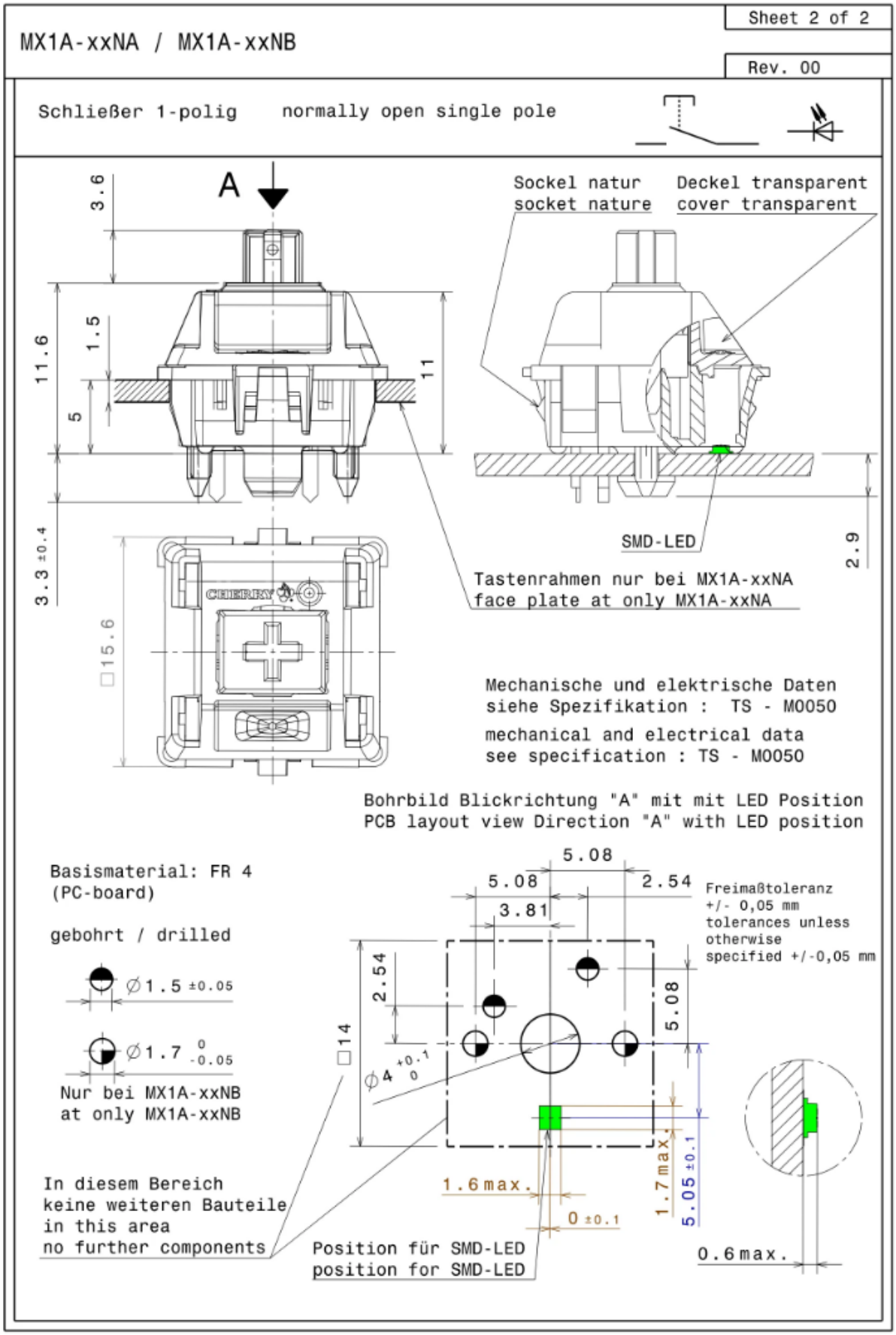
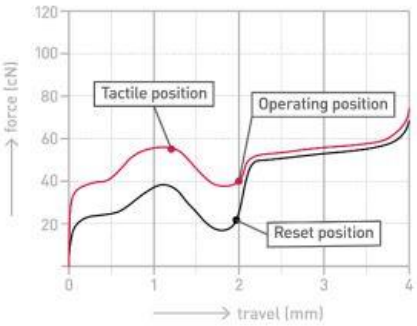


Figure 18 - Cherry MX Dimensions [46]

CHERRY MX RGB ERGO CLEAR

MX2A-H1NA *(factory lubricated)*



High-precision CHERRY MX switch with noticeable, tactile feeling

Original CHERRY MX is the world´s leading precision technology for mechanical key switches. The CHERRY Gold Crosspoint contact concept and the unprecedented production quality "Made in Germany" are unique. MX inside ensures unrivalled quality, precision and reliability.

Key benefits:

- Engineered and Made in Germany
- CHERRY MX RGB Ergo Clear switch: With tactile pressure point, key stroke with noticeable tactile feedback
- Factory-lubricated for optimal and smooth typing experience
- Safe and long-lasting reliability of switching performance and characteristics
- High-precision mechanical key switch based on the MX standard
- Clear stem and transparent housing
- World exclusive CHERRY Gold Crosspoint technology
- Short bounce time for high switching frequency (such as for fast typing)
- Self-cleaning contacts
- Switch resistant to dust and dirt
- Over 50 million keystrokes with no loss of quality

Technical Data:

• Switch Type:	MX Mechanical Switch
• Fastening:	Snap fastening in frame
• Protection class:	IP40
• Operation characteristics:	tactile
• Switching voltage:	5V
• Dielectric strength:	500V / 50Hz
• Durability:	> 50 million actuations
• Contact configuration:	Single-pole contact
• Actuator travel:	4 mm
• Pretravel:	2 mm
• Initial force:	35 cN
• Actuation force:	40 cN
• Pressure point force:	55 cN
• Bounce time:	typically < 1ms
• Minimum lead spacing:	16 mm
• Lighting:	SMD LED can be mounted directly on the PCB (not included in the module), opening for SMD LED in the base
• Insulation materials:	Thermoplastics
• Spring:	Stainless steel
• Contacts:	High-quality gold alloy

Figure 19 - Cherry MX RGB Ergo Clear Datasheet [46]

Connector

For the purpose of connecting the second keyboard half and allowing data transfer between two control boards a minimum of 3 pins is required if using serial to carry Ground, Power and Data. In the case of I2C an additional Pin is required to carry the second Data line. The 4 Pin 3.5mm TRRS Audio-Jack was found to be a good option, these connectors are designed for frequent plugging and un-plugging of audio devices, as well as low impedance, as to maintain audio clarity, thereby also providing a good option for low power serial data transfer. Lastly the connector has a small footprint, which would aid in the integration within the limited space of a Split keyboard. A set of 10 breakout boards with 3.5mm TRRS can be found for only €7.17 (amazon.de 13.Mar.2024), with a matching connector available at a unit cost of only €0.63 (digikey.ie 13.Mar.2024).

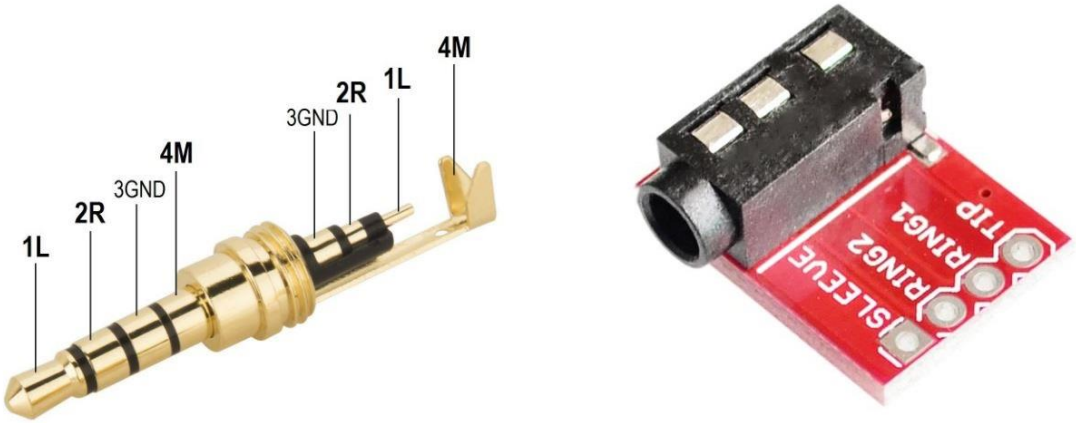


Figure 20 - TRRS 3.5mm connector (left) & TRRS 3.5mm Jack Breakout Board (right)

LEDs

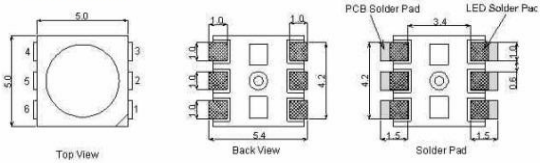
For the purpose of Backlight, a set of LEDs would be suitable to create an under-glow below the switches. Given the connection via USB, the LEDs needed to be able to be driven by the 5V USB power and not exceed the USB Standards Power limit of 500mA [47], which also would be shared with the Control Boards for the keyboard. Therefore, the commonly available variants of the WS2812 RGB LEDs, also known as NeoPixels, would be well suited due to their 5V power requirement and ability to be directly driven by many control boards like the ATMega32U4 Pro-Micro. The LEDs also have a max amp draw at full brightness of ca. 50mA which falls well within the power budget of the USB standard. It has to be considered though that if multiple LEDs were to be deployed, brightness may need to be limited to stay within the power-budget.

Worldsemi

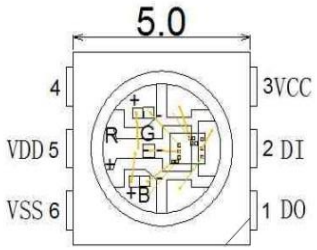
WS2812

Intelligent control LED integrated light source

Mechanical Dimensions



PIN configuration



PIN function

NO.	Symbol	Function description
1	DOUT	Control data signal output
2	DIN	Control data signal input
3	VCC	Power supply control circuit
4	NC	
5	VDD	Power supply LED
6	VSS	Ground

Absolute Maximum Ratings

Prameter	Symbol	Ratings	Unit
Power supply voltage	V_{CC}	+6.0~+7.0	V
Power supply voltage	V_{DD}	+6.0~+7.0	V
Input voltage	V_I	-0.5～VDD+0.5	V
Operation junction temperature	Topt	-25～+80	℃
Storage temperature range	Tstg	-55~+150	℃

Electrical Characteristics（T_A=-20～+70℃，V_{DD}=4.5～5.5V,V_{SS}=0V,unless otherwise specified）

Prameter	Smybol	conditions	Min	Tpy	Max	Unit
Low voltage output current	I_{OL}	ROUT	——	18.5	——	mA
	I_{dout}	Vo=0.4V，D_{OUT}	10	——	——	mA
Input current	I_I	V_I=V_{DD}/V_{SS}	——	——	±1	μA
Input voltage level	V_{IH}	D_{IN}，SET	0.7V_{DD}	——	——	V
	V_{IL}	D_{IN}，SET	——	——	0.3 V_{DD}	V
Hysteresis voltage	V_H	D_{IN}，SET	——	0.35	——	V

Switching characteristics（T_A=-20～+70℃，V_{DD}=4.5～5.5V,V_{SS}=0V,unless otherwise specified）

Prameter	Symbol	Condition	Min	Tpy	Max	Unit
Operation frequency	*Fosc*2	——	——	800	——	KHz
Transmission delay time	t_{PLZ}	CL=15pF,DIN→ DOUT,RL=10KΩ	——	——	300	ns
Fall time	t_{THZ}	CL=300pF,OUTR/OUTG/OUTB	——	——	120	μs
Data transmission rate	F_{MAX}	Duty ratio50%	400	——	——	Kbps
Input capcity	C_I	——	——	——	15	pF

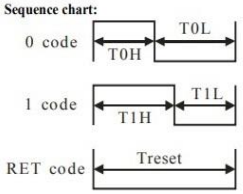
LED characteristic parameter

Emitting color	Wavelength(nm)	Luminous intensity(mcd)	Current(mA)	Voltage(V)
Red	620-630	550-700	20	1.8-2.2
Green	515-530	1100-1400	20	3.0-3.2
Blue	465-475	200-400	20	3.2-3.4

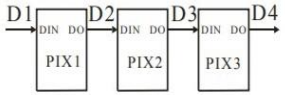
Data transfer time(TH+TL=1.25μs±600ns)

T0H	0 code ,high voltage time	0.35us	±150ns
T1H	1 code ,high voltage time	0.7us	±150ns
T0L	0 code , low voltage time	0.8us	±150ns
T1L	1 code ,low voltage time	0.6us	±150ns
RES	low voltage time	Above 50μs	

Sequence chart:



Cascade method:



http://www.world-semi.com
http://www.world-semi.com

Figure 21 - WS2812 Datasheet

Prototyping

With the component choices in place, the next step of developing a prototype could be taken. The chosen components and their spec sheets, as well as the prior keyboard layout design decisions were used as an outline for dimensions.

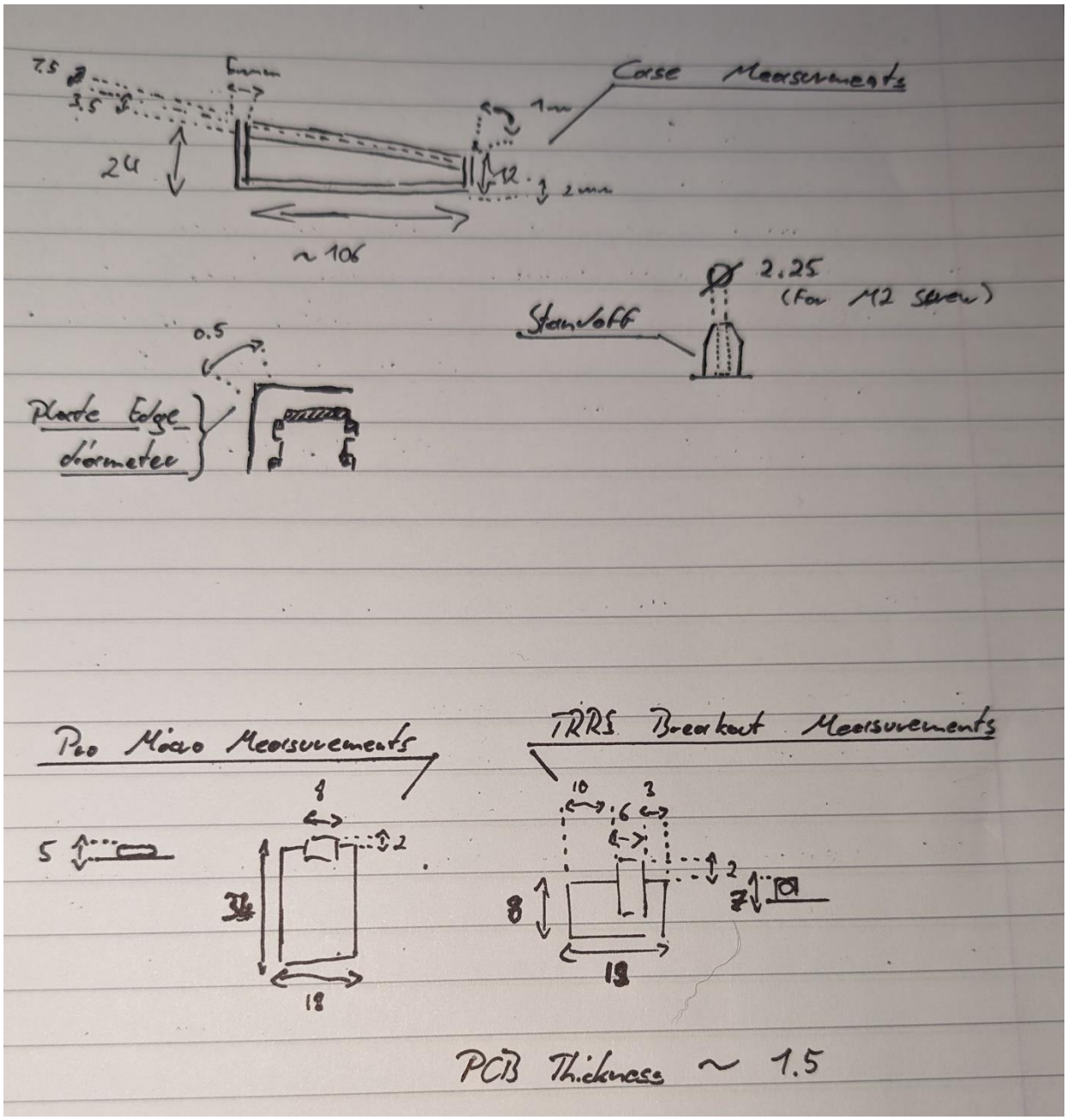


Figure 22 - Measurements and initial Prototyping

Of high importance were the dimensions taken for the plate, which would house the switches, as the design by Cherry intents for the switches to clip into a 1.5mm thickness plate with the correct cutouts. As FDM 3D Printing was intended to be used for the prototyping the 1.5mm plate thickness was considered too thin.

Not just was the printed result at 1.5mm flexible enough to cause concern about warping, but during prolonged use and exposure to key presses may cause stress fractures to occur, which means the plate was to be reinforced by 2.5mm, for a total thickness of 4mm. This was done while also creating a 1mm recess below the switch cutouts, to still accommodate the latch on the Cherry switches to correctly sit into the plate cutouts.

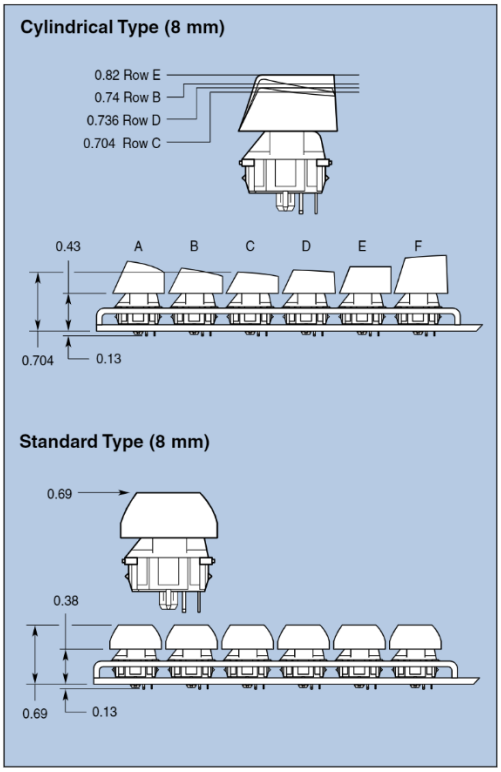
The case dimensions were based on a ca. 10mm deep open-top box shape to accommodate the electronics like TRRS breakout boards and control boards. This design was fitted with standoffs to support the plate for a better structural integrity and to minimise flexing. Additionally, the edges were raised to create a downward slope towards the front, the slope angle was influenced by the findings of the literature review, declaring a typing angle of over 15° to be considered wrist-extension. Therefore, a low slope of less than half of 15° at only six degrees was targeted and dimensions were adjusted accordingly. To further accommodate typing feel all exposed edges the user's hands may contact were to be rounded off.

With all above decisions and dimensions, the project was translated into a digital model using Autodesk Fuzion360. During this process additional standoffs without screw holes were added around the case perimeter to further accommodate the support of the switch-plate, cutouts for both connectors and PCBs were also added to the model, alongside screw holes for small 2M retaining screws for the PCBs, providing support when users connect or disconnect the relevant cables.

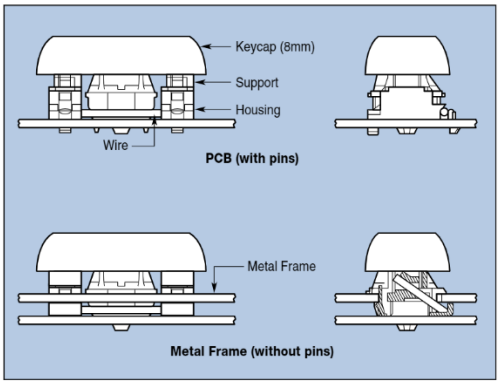
During this process it was also found that Cherry's datasheets [48] recommend a Stabiliser to be used on any keys of 2U size or larger. Since there were 3 of such keys, two for the spacebar on each half and one for the enter key, the plates were designed to accommodate clip-in stabilizers. These stabilizers are designed similar to the Cherry switch, with the intention of latching into a 1.5mm thick plate with the correct cut-outs. A suitable set of stabilisers was also sourced, due to inter-compatibility between manufacturers the stabilisers by Durock were sourced as available in single sets of 2U stabilisers at only €2.5 (keygem.com 18.Mar.2024). The function of the stabiliser is to use two sliders, that are connected with a steel wire, to transfer the linear motion of the switch-press evenly throughout the keycap. Therefore, the functionality of the switch is not restricted should the press occur off-centre.

Lastly small internal recesses were added to the space-bar-side on each keyboard half-case. These were to allow for the friction-fitting of 5mm cube magnets to fix the keyboard halves together, when used as a single-piece 60% keyboard instead of in a split configuration.

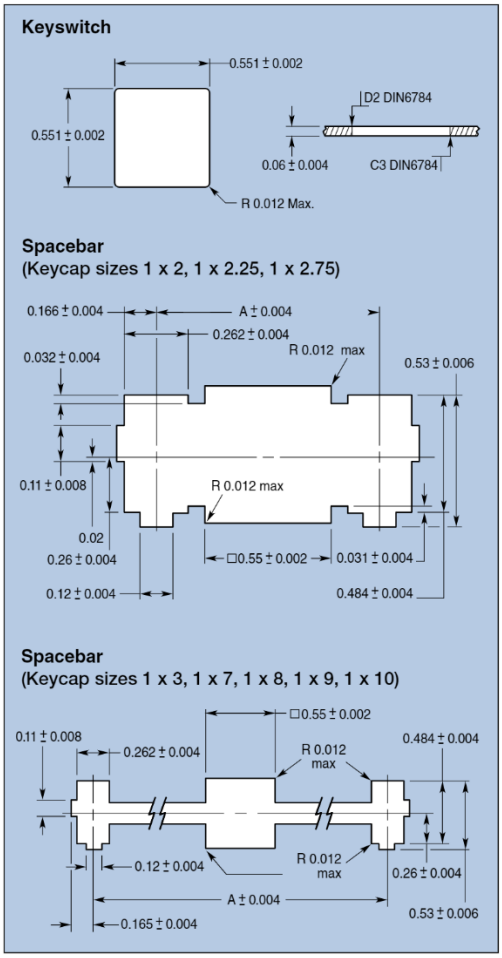
Keycap Dimensions



Spacebar Mounting Options



Metal Frame Cutout Dimensions



Leveling Mechanism Kits

Keycap Size	1 x 2 1 x 2, 2.5 1 x 2, 2.75	1 x 3	1 x 8	1 x 8 1 x 9 1 x 10	1 x 7
Keycap Type	Std & Cylin	Std & Cylin	Std	Cylin	Cylin
A (inches)	0.94	1.5	5.25	5.25	4.5
P/N w/frame	G99-0224	G99-0225	G99-0226	G99-0226	G99-0379
P/N w/o frame	G99-0742	G99-0743	G99-0744	G99-0744	G99-0745

10-4

Figure 23 - Cherry MX Datasheet for plate mounting switches and stabilizers [48]

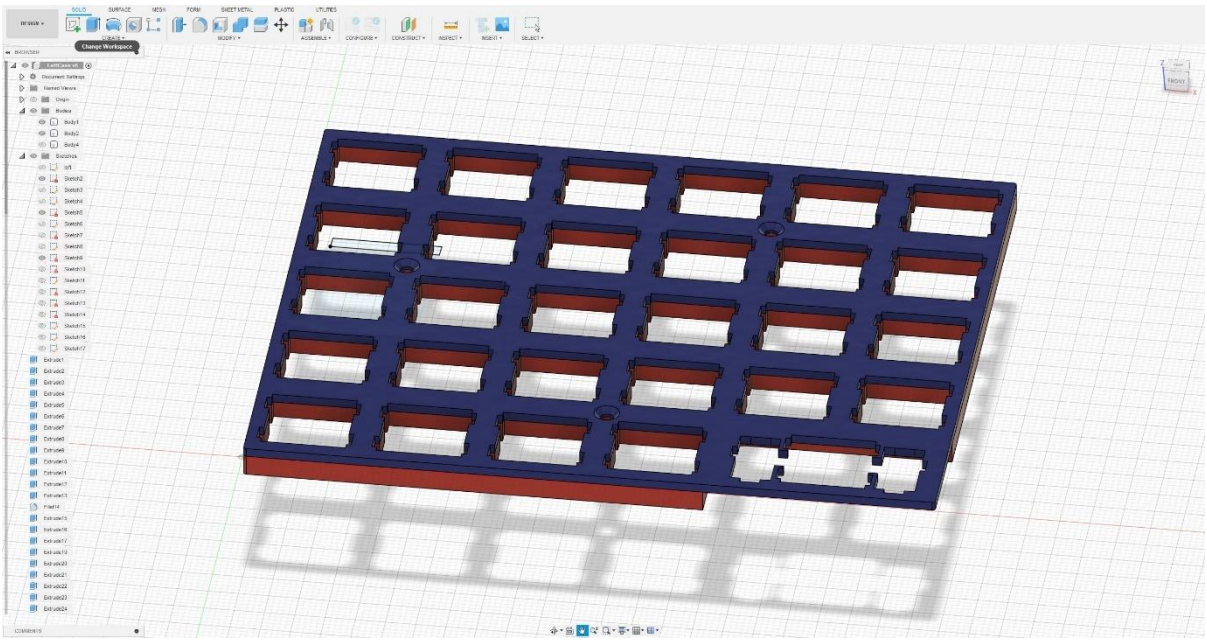


Figure 24 - Left switch plate Fuzion360 model

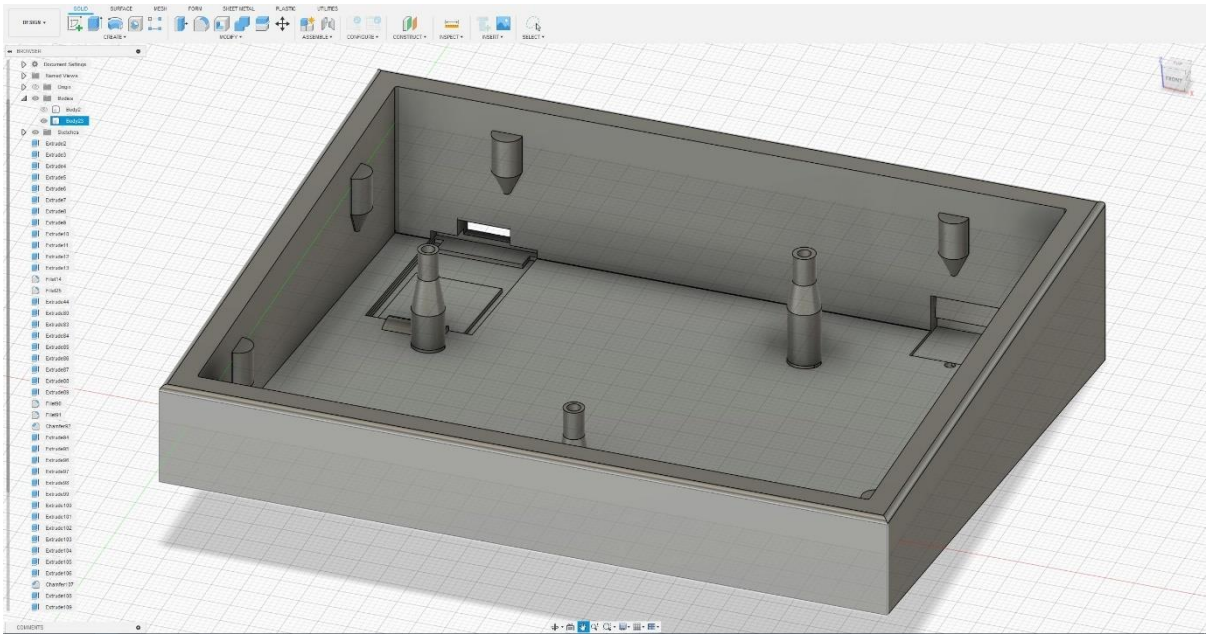


Figure 25- Left case Fuzion360 model

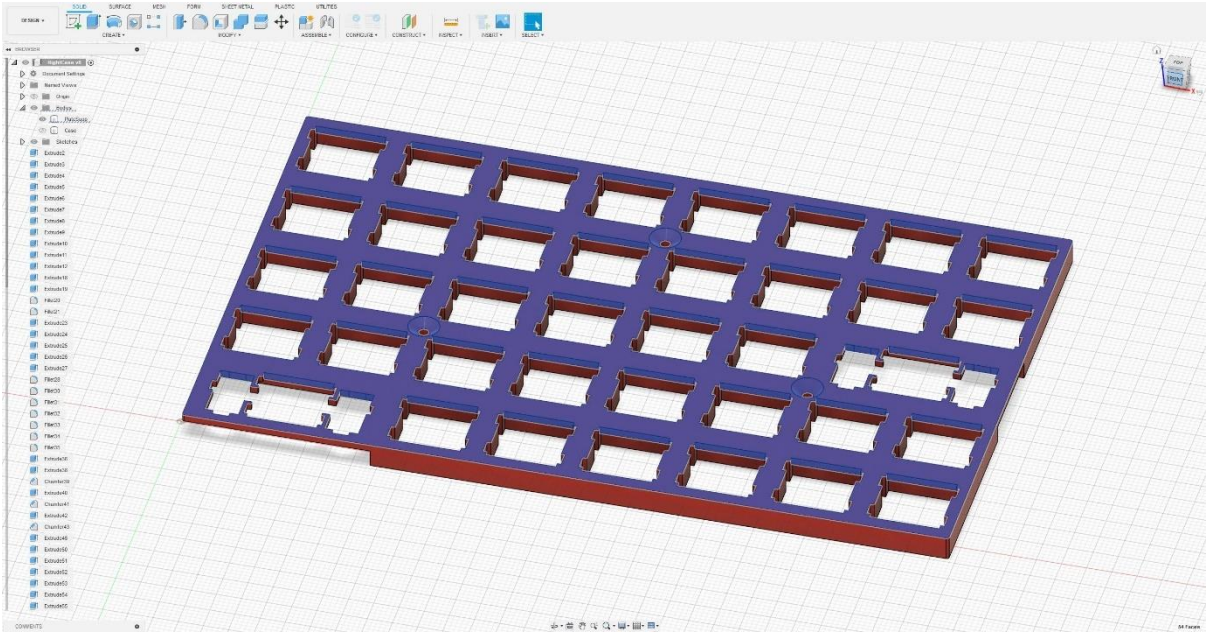


Figure 26 - Right switch plate Fuzion360 model

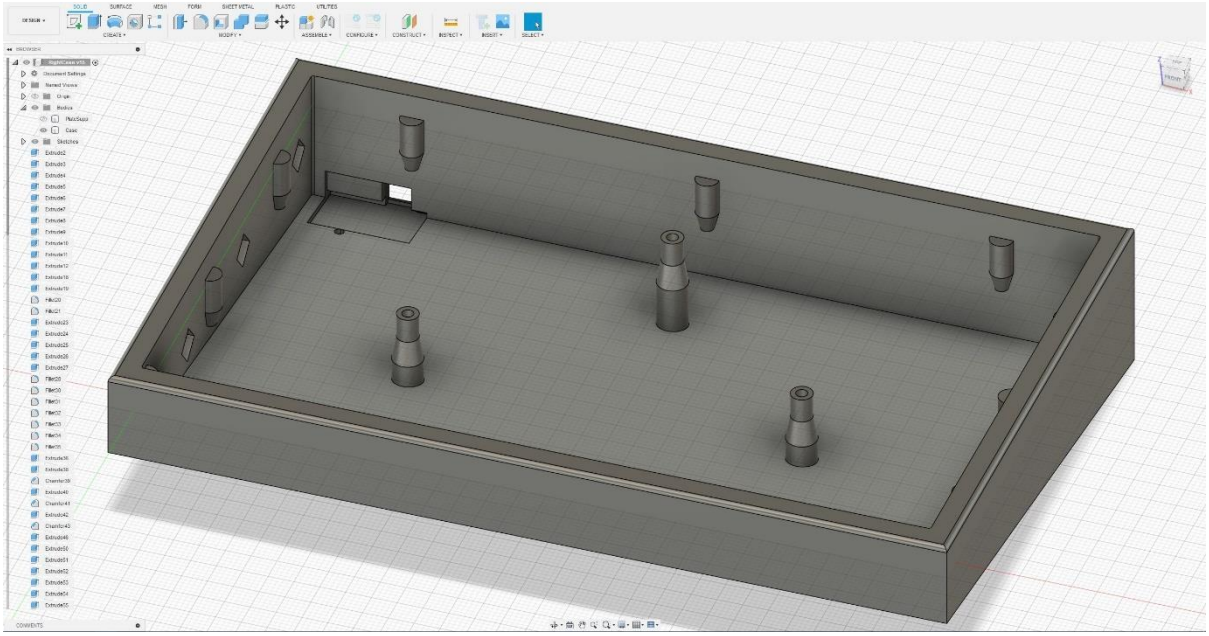


Figure 27 - Right case Fuzion360 model

Following the modeling process the Fuzion360 model was exported as a .stl file for 3D printing. These files was opened in the AnkerMake Slicer software and prepared for 3D printing. There were no requirements for additional structures like supports highlighted, due to the shapes of the plates and cases allowing for flat bed-alignment. The files were successfully translated by the slicing software into the relevant machine Code for use on an AnkerMake M5c FDM 3D printer and transferred to such machine for printing.

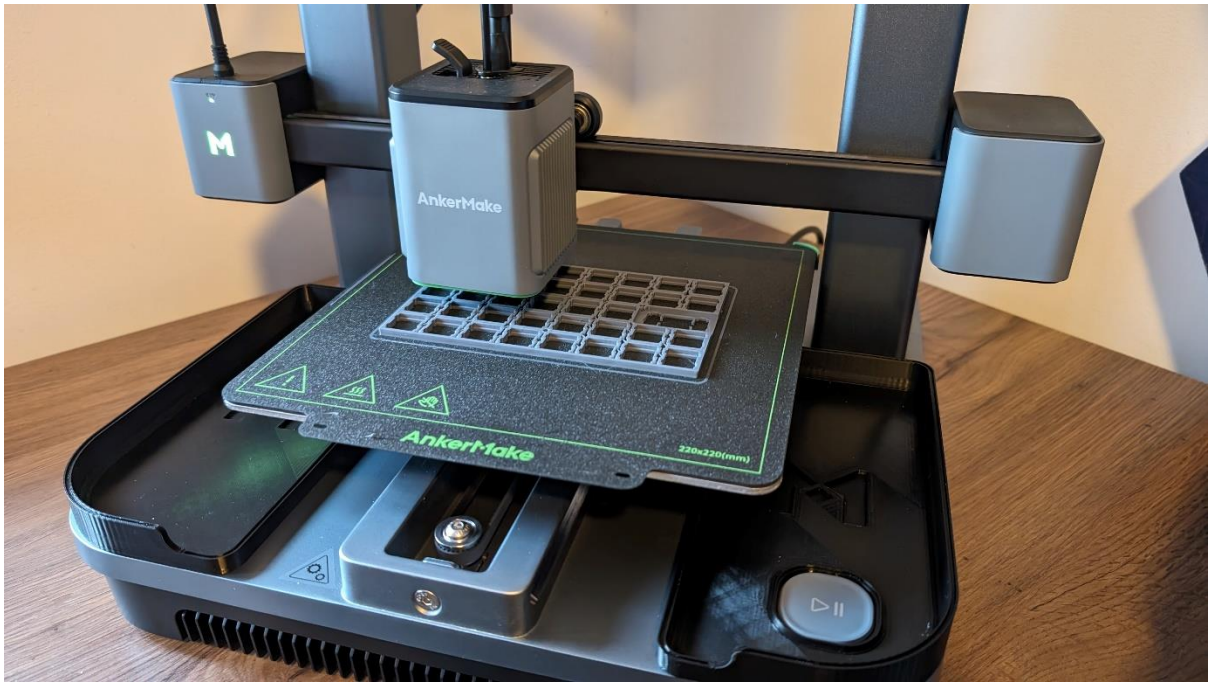


Figure 28 - 3D printing process of the right switch plate

After the physical design of the case for the keyboard prototype, the circuitry schematic was sketched using KiCad 8.0 (see Figure 29), due to the similar design between each keyboard half, a singular schematic was able to be drawn. Annotations were added to document the differences to be considered for each keyboard half. This included the switch matrix, incorporating a diode per switch to prevent unwanted feedback through pressed switches, which could cause false key-presses to be detected. The overall matrix to accommodate the designed layout was five rows by eight columns, with two less columns to be used on the slightly smaller left keyboard half. The TRRS 3.5mm audio jack was also mapped to the pinout as the connection point between boards for the I2C communication between control boards. Following the documentation on I2C it also is advised to implement 4.7kΩ pull-up resistors for the serial data lines [49], since these are not required to be implemented redundant, the left keyboard half would be the only side with these resistors fitted, to reduce component use. The WS2812B LED's were also mapped for the optional backlighting to be implemented using a free pin on the control board to directly drive the data line to the LEDs. Using a total of eight LEDs for the backlighting on the right and six LEDs for the left keyboard half, given the max power consumption of the chosen Pro-Micro controller at around 15mA the LEDs need to be limited in software later on to prevent the overall set of 14 LEDs at up to 50mA per LED not to exceed the 500mA power budget of the USB spec [47].

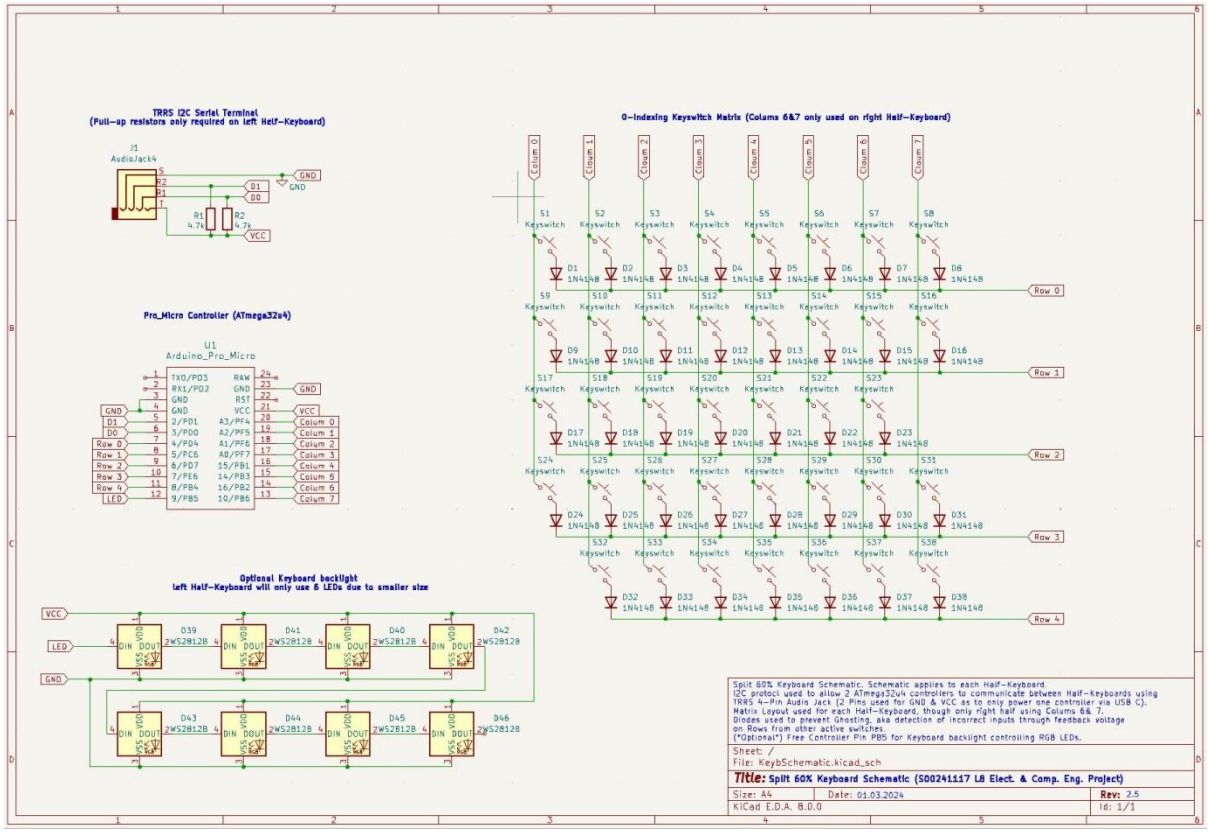


Figure 29 - KiCad 8.0 Circuitry Schematic

With the documentation in place and the case printed, the switches and stabilizers were fitted to the right and left keyboard plate. The fitment of the control board and the TRRS breakout board were also checked at this point (see Figure 30 below)

Figure 30 - Fitting components to case and plate

With the switches in place the diodes were soldered to each switch. At this point two options for the wiring process were tested. First using solid 1mm copper wire to provide rigidity and second using 26 AWG signal wires with color-coded silicone sheathing (see Figure 31 below).

During the process it was noted that the signal wire was pre-fitted with insulating sheathing and could be trimmed to size easily. Due to the low power requirements for a USB keyboard, running of the USB supplied max of 5V and 0.5A [47] the 26 AWG with an ampacity of 1.3A would be well suited for this application. Further the color-coding would aid with traceability and repairability.

The solid wire on the other hand would provide additional rigidity but would make tracing signals more difficult. Additionally clear shrink tubing was required to insulate the columns and rows. Given the 1mm diameter of the solid copper core, the wire would be comparable to a 18 AWG wire, which are rated for an ampacity of 10A and thereby well exceed the minimum requirements for cabling of a low power USB Keyboard. Lastly the copper wire would need to be cut and bent according to fitment required for each row and column, hindering replaceability.

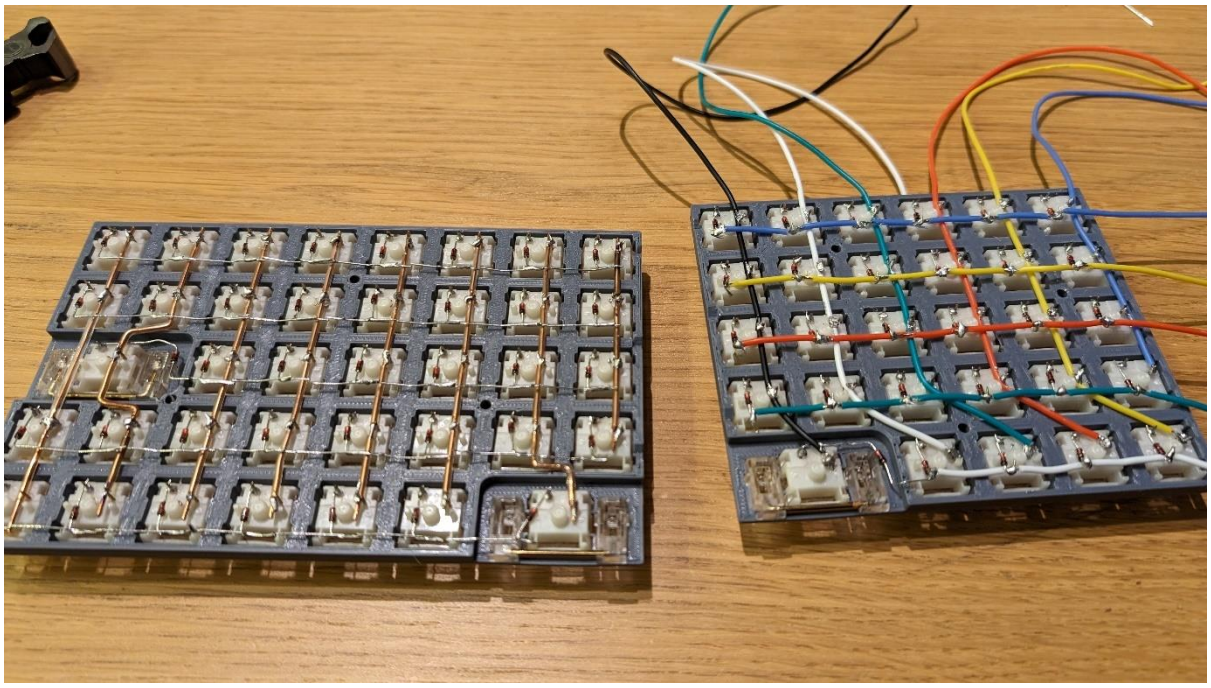


Figure 31 – Tested wiring options for switch matrix: Solid 1mm copper wire shown on the left, color-coded signal wire on the right

Based on the above findings the decision was made to continue the project using the 26 AWG signal wiring. This was also used to wire up the WS2812B LEDs as per the schematic, to reduce wiring the ground (black) and power (red) cables were wired across the top and bottom respectively of LEDs in sets of two, as can be seen in Figure 32. The signal wire (green) was fed into the first LED of the series of LEDs and then routed top to bottom in each set of two LEDs, while connecting the bottom LED's data-out to the next set of two top LED's data-in.

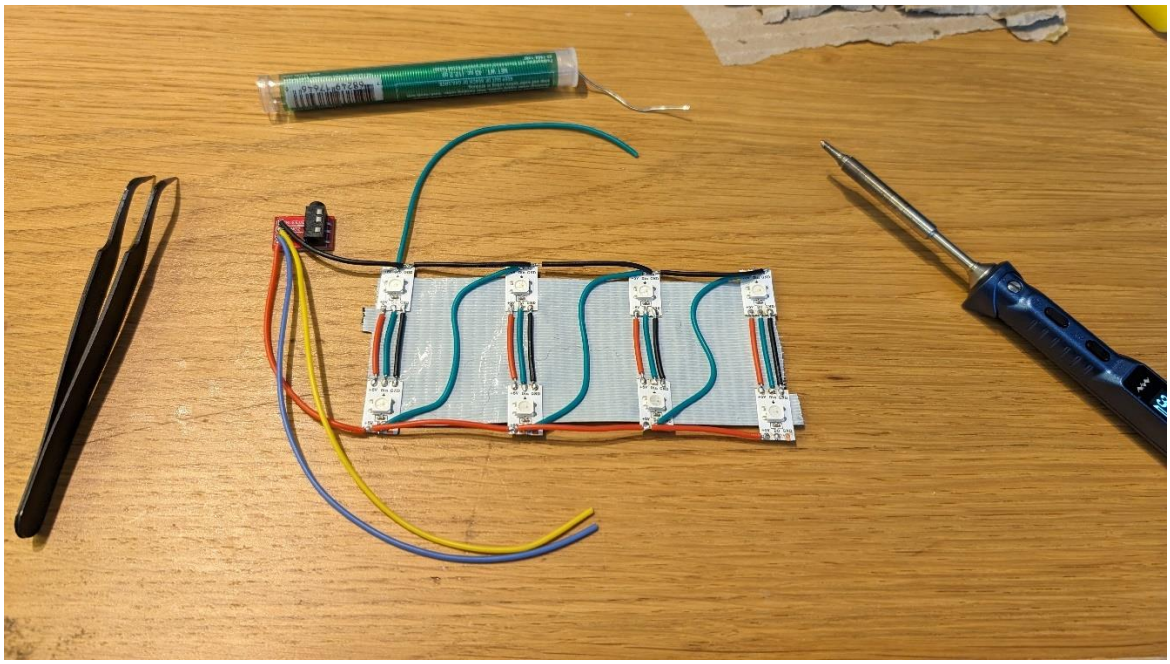


Figure 32 - WS2812B LED backlight wiring

Finally, all cables were wired to the Pro-Micro's Pins as per the earlier created KiCad schematic (see Figure 29). Using a DMM, in the diode-tester setting, the final matrix was checked. Each key was tested separately with the DMM connected to and moving through the rows and columns accordingly. With the functionality tested the components were fitted into the 3D printed case and the case was screwed closed after fitting the three 5mm cube magnets into their cutouts along the spacebar side of each keyboard half.

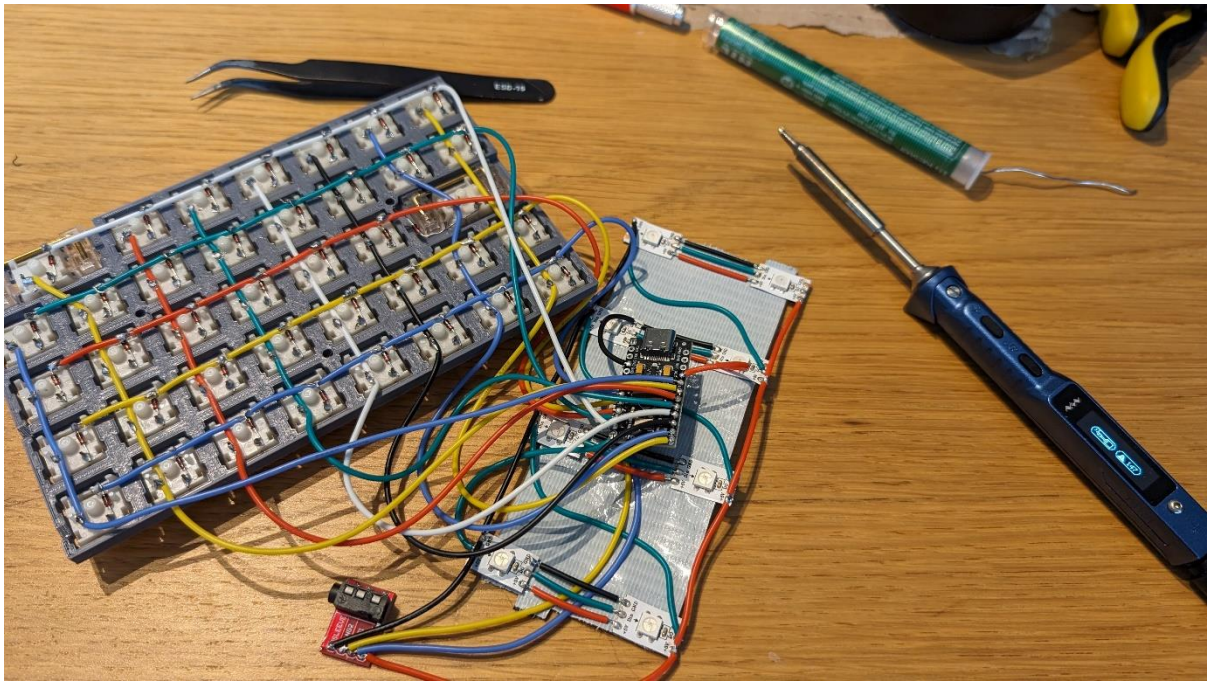


Figure 33 - Wired right switch matrix, control board, TRRS breakout board and LEDs

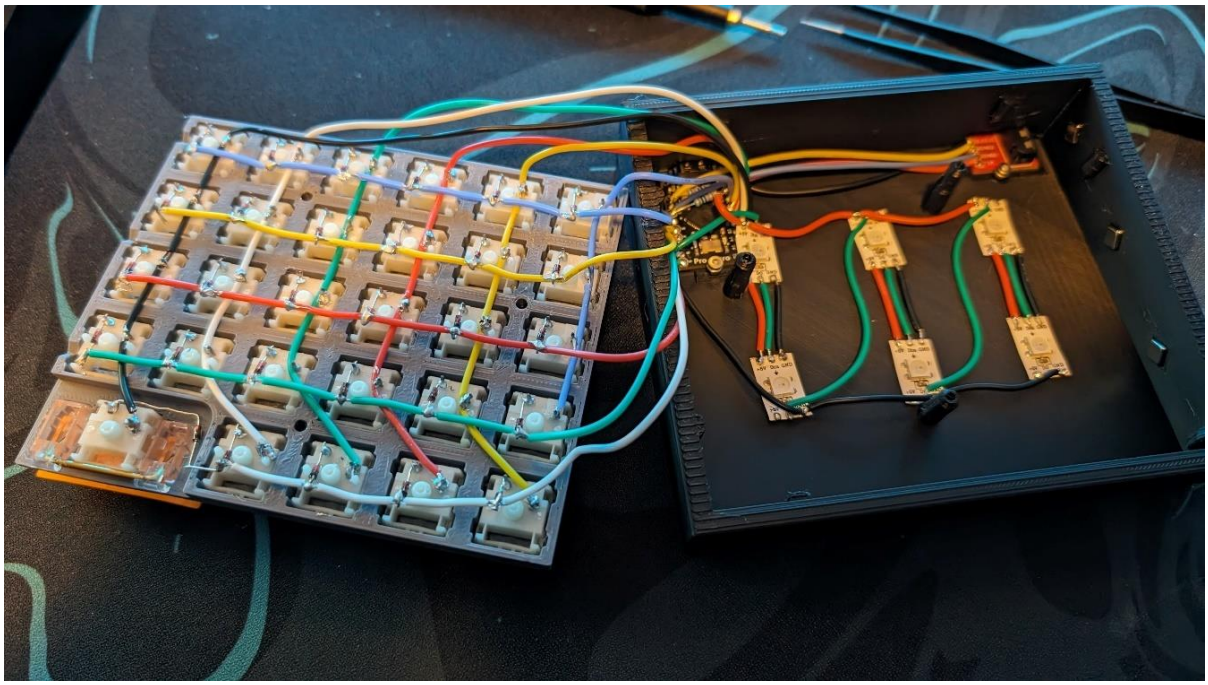


Figure 34 - Components and wiring fitted in left keyboard half case

Coding

At the same time of prototyping the code for the keyboard was researched and written. A software flow-chart was created to outline the initially expected functions to be programmed for the keyboard.

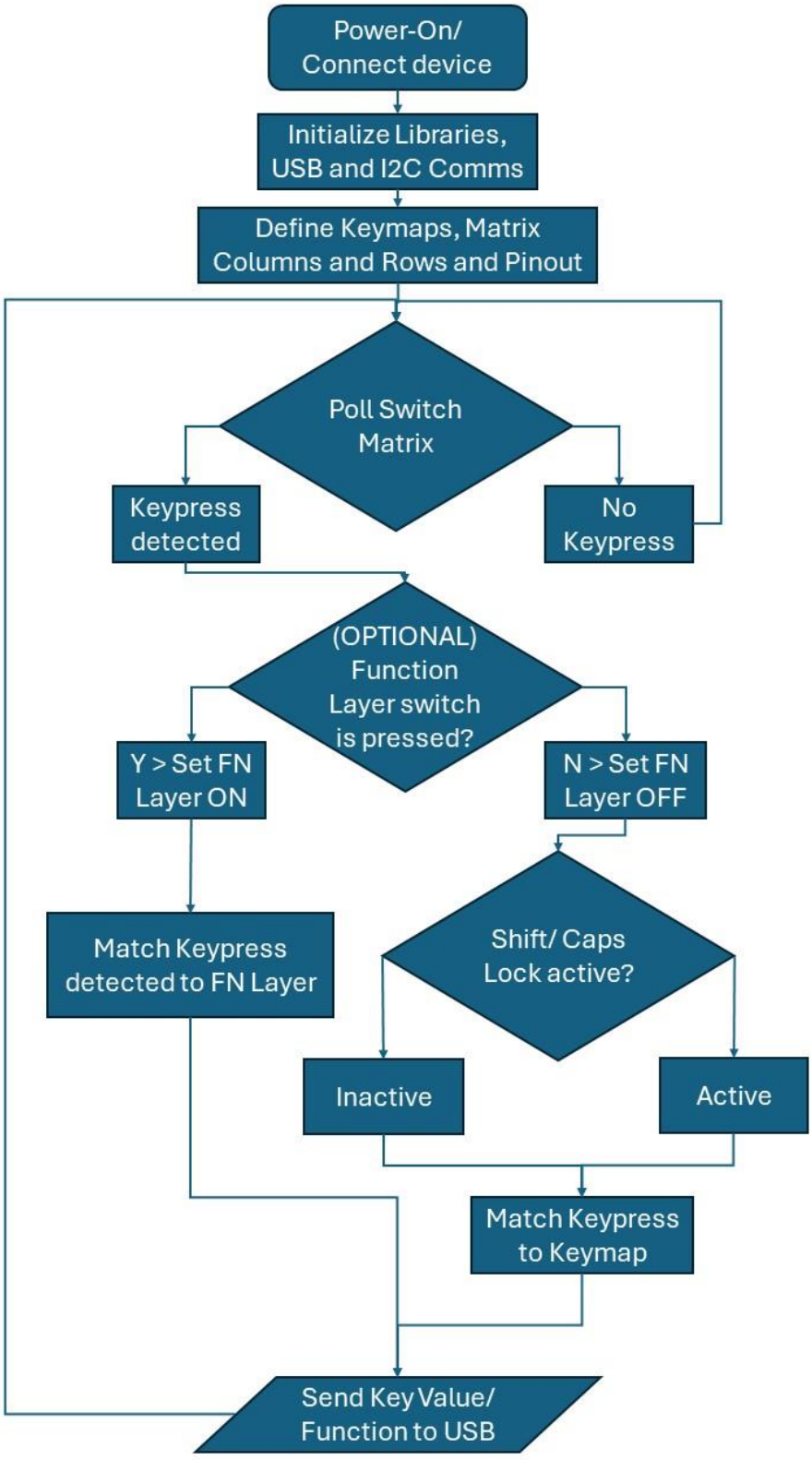


Figure 35 - Software flow-chart

Given the choice of Pro-Micro control board the Arduino IDE was used to generate a driver for each keyboard half. The Arduino library already includes header files for both keyboard functionality [50] as well as I2C protocol [51]. But for integration of the RGB LED function a new header would be required. Such was found and subsequently added to the local Arduino IDE installation, in the form of the Adafruit NeoPixel library [52]. With the required files in place and documentation available the subsequent code for each keyboard half was created (please reference comments for details):

Left/ primary keyboard Half

```
/* Title: LeftKeyboardHalf (Primary) - S00241117 Project
Code Details: Chip Driver for Pro Micro ATmega32U4 based Left Keyboard Half
Start Date: 15/01/2024

pin D1: N/A     -|   usbc   |-  RAW
pin D0: N/A     -|          |-  GND
        GND     -|          |-  RST
        GND     -|   top    |-  VCC
pin D2: I2C SDA -|          |-  pin A3:  col 0
pin D3: I2C SCL -|          |-  pin A2:  col 1
pin D4: row 0   -|          |-  pin A1:  col 2
pin D5: row 1   -|          |-  pin A0:  col 3
pin D6: row 2   -|          |-  pin D15: col 4
pin D7: row 3   -|          |-  pin D14: col 5
pin D8: row 4   -|          |-  pin D16: N/A
pin D9: LEDSTR  -|__________|-  pin D10: N/A
*/

///// Libraries /////
#include <Keyboard.h>           //Arduino Keyboard Library
#include <Wire.h>               //I2C handler
#include <Adafruit_NeoPixel.h>  //LED Strip Library

///// Global Variables /////
const int SCND_ADDY = 8;
const int ROWS = 5;
const int COLS = 6;
const int RIGHTCOLS = 8;
const int rowpin[ROWS] = {4, 5, 6, 7, 8};
const int colpin[COLS] = {A3, A2, A1, A0, 15, 14};
const int LEDPIN = 9;
const int NUMPIXELS = 6;

int byteCount = 0;                      //Counter to handle Secondary I2C
reception
int initialDelay = 60;                  //No. of Cycles to delay if key
held after first press, before repeating key input
int repeatDelay = 6;                    //No. of Cycles to delay if key
held continously, before repeating key input

byte bytes[5] = {0, 0, 0, 0, 0};        //Byte Array for Primary input
handling
byte bytesRight[5] = {0, 0, 0, 0, 0};   //Byte Array for Secondary input
handling

unsigned int keyHoldCount [ROWS][COLS]; //Tracker for held Keys on Primary
```

```
bool keyLongHold [ROWS][COLS];          //Tracker for continuously held
keys on Primary

unsigned int keyHoldCountRight [ROWS][RIGHTCOLS];   //Tracker for held Keys
on Secondary
bool keyLongHoldRight [ROWS][RIGHTCOLS];            //Tracker for
continously held keys on Secondary

////// Keymap for Primary & Secondary //////

char keyMap[ROWS][COLS] = {
  {177 , '1' , '2' , '3' , '4' , '5'},  //esc [...]
  {179,  'q' , 'w' , 'e' , 'r' , 't'},  //tab [...]
  {193,  'a' , 's' , 'd' , 'f' , 'g'},  //caps lock [...]
  {129,  'z' , 'x' , 'c' , 'v' , 'b'},  //shift [...]
  {128,  '`',  131,  130,  ' ' , '*'},  //ctrl, [...] win, alt, [...], not
used
};

char keyMapRight[ROWS][RIGHTCOLS] = {
  {'6' , '7' , '8' , '9' , '0' , '-' , '=', 178 }, //[...] backspace
  {'y' , 'u' , 'i' , 'o' , 'p' , '[' , ']', '\\'},
  {'h' , 'j' , 'k' , 'l' , ';' , '\'', 176, '*' }, //[...] enter, not used
  {'n', 'm' , ',' , '.' , '/' , 133 , 218, 212  }, //[...] shift, up, del
  {'*', ' ',  134,  237,  132,  216 , 217, 215  }, //not used [...] alt,
menu, ctrl, left, down, right
};

///// function prototypes /////
void keyPressed(int row, int col);
void keyReset(int row, int col);
void keyPressedRight(int row, int col);
void keyResetRight(int row, int col);
void printByteArray(const unsigned char* array, size_t length);
Adafruit_NeoPixel strip = Adafruit_NeoPixel (NUMPIXELS, LEDPIN, NEO_GRB +
NEO_KHZ800);

///// Arduino setup on power on /////
void setup() {
  Wire.begin();                             //startup I2C handler
  Serial.begin(9600);                       //Serial fucntion for debugging
  strip.begin();                            //startup LED handler
  Keyboard.begin();                         //startup Keyboard HID handler

  for (int r = 0; r < ROWS; r++) {          //Set all Rows as Output (HIGH)
    pinMode(rowpin[r], OUTPUT);
  }

  for (int c = 0; c < COLS; c++) {          //Set all Columns as Pull-Up
Input
    pinMode(colpin[c], INPUT_PULLUP);
  }

  strip.clear();                            //reset all LEDs

  for(int p=0; p<NUMPIXELS; p++) {          //repeat for each LED Pixel
    strip.setPixelColor(p, 200, 120, 0);    //set LED Pixel RGB color value
(orange)
  }
  strip.show();                             //Display RGB colors on current
LEDs
```

```
}

///// Arduino infinite loop function while powered on /////
void loop() {
  Wire.requestFrom(SCND_ADDY, 5);           //Send request to secondary for
5 bytes

  while (Wire.available()) {                //When receiving, loop through
received bytes
    bytesRight[byteCount] = Wire.read();    //Transpose received Byte to
bytesRight Array
    byteCount++;                            //move to next byte received
  }

  byteCount = 0;                            //Reset byteCount for next
request

  for (int r = 0; r < ROWS; r++) {          //repeat per Row

    digitalWrite(rowpin[r], LOW);           //Set current Row as LOW
    delayMicroseconds(100);                 //Delay to stabilize new state

    for (int c = 0; c < COLS; c++) {        //repeat per Colum
      if(digitalRead(colpin[c]) == LOW) {   //if LOW detected
        bytes[r] |= (1 << c);               //set bit in bytes array
      }
      else{                                 //otherwise
        bytes[r] &= ~(1 << c);              //clear byte, to handle key
releases
      }
    }
    digitalWrite(rowpin[r], HIGH);          //reset Row Pin to High
    delayMicroseconds(100);                 //delay to stabilize new state
  }

  for(int r = 0; r < ROWS; r++){            //repeat per Row
    for(int c = (COLS - 1); c >= 0; c--){   //repeat per Colum on Primary,
in reverse
      if((bytes[r] >> c) & 1){              //if position in bytes array is
1
        keyPressed(r,c);                    //call keyPressed with current
row and col
      }
      else{                                 //otherwise
        if(!((r == 3 && c == 0) || (r == 4 && c == 0) || (r == 4 && c == 2)
|| (r == 4 && c == 3))){
                                            //if NOT modifier key
          keyReset(r,c);                    //call keyReset ith current row
and col
        }
        else {                              //otherwise
          Keyboard.release(keyMap[r][c]);   //Keyboard Handler at current
row and col set key released
        }
      }
    }
  }

  /*Due to transposed bit array for Secondary, Cols are inverted and need
to be handled as such.
```

```
    Because of 0 indexing for 8 culums on the Secondary this will be
handled via 7 - c (current colum) */
  for(int r = 0; r < ROWS; r++){            //repeat per Row
    for(int c = (RIGHTCOLS - 1); c >= 0; c--){  //repeat per Colum on
Secondary, in reverse
      if((bytesRight[r] >> c) & 1){         //if position in bytesRight
array is 1
        keyPressedRight(r,(7-c));           //call keyPressedRight with
current row and col
      }
      else{                                 //otherwise
        if(!((r == 3 && c == (7-5)) || (r == 4 && c == (7-2)) || (r == 4 &&
c == (7-4)))){
                                            //if NOT modifier key
          keyResetRight(r,(7-c));           //call keyResetRight ith
current row and col
        }
        else{                               //otherwise
          Keyboard.release(keyMapRight[r][c]);  //Keyboard Handler at
current Secondary row and col set key released
        }
      }
    }
  }
}

///// Called Functions /////
void keyPressed(int row, int col) {         //Key Pressed on Primary at row
and col position
  if(!((row == 3 && col == 0) || (row == 4 && col == 0) || (row == 4 && col
== 2) || (row == 4 && col == 3))){
                                            //if NOT modifier key
     if(keyHoldCount[row][col] == 0){       //if first call for key
      Keyboard.write(keyMap[row][col]);     //print key matching Primary
keyMap
    }

    else if(keyLongHold[row][col] && keyHoldCount[row][col] > repeatDelay){
                                            //if key is held (detected
again without release)
                                            //check if long hold is true
and is Hold Count > repeatDelay
      Keyboard.write(keyMap[row][col]);     //print key matching Primary
keyMap
      keyHoldCount[row][col] = 1;           //Reset Hold Count for key at
current row and col
    }

    else if(keyHoldCount[row][col] > initialDelay){ //if key is held
(detected again without release)
                                            //following initial press,
check if Hold Count > initialDelay
      keyLongHold[row][col] = true;         //set long hold as true
    }

    keyHoldCount[row][col]++;               //increase Hold Count for key
at current row and col
  }
  else{                                     //otherwise (Key at current row
and col is modifier)
```

```
     Keyboard.press(keyMap[row][col]);      //register key matching Primary
keyMap as pressed in Keyboard Handler
  }

}


void keyPressedRight(int row, int col) {    //Key Pressed on Secondary at
row and col position
  if(!((row == 3 && col == 5) || (row == 4 && col == 2) || (row == 4 && col
== 4))){
                                            //if NOT modifier key
     if(keyHoldCountRight[row][col] == 0){  //if first call for key
      Keyboard.write(keyMapRight[row][col]);//print key matching Secondary
keyMap
      }

      else if(keyLongHoldRight[row][col] && keyHoldCountRight[row][col] >
repeatDelay){
                                            //if key is held (detected
again without release)
                                            //check if long hold is true
and is Hold Count > repeatDelay
        Keyboard.write(keyMapRight[row][col]);//print key matching
Secondary keyMap
        keyHoldCountRight[row][col] = 1;    //Reset Hold Count for key at
current row and col

      else if(keyHoldCountRight[row][col] > initialDelay){ //if key is held
(detected again without release)
                                            //following initial press,
check if Hold Count > initialDelay
        keyLongHoldRight[row][col] = true;  //set long hold as true
      }

      keyHoldCountRight[row][col]++;        //increase Hold Count for key
at current row and col

    }
    else{                                   //otherwise (Key at current row
and col is modifier)
      Keyboard.press(keyMapRight[row][col]);//register key matching
Secondary keyMap as pressed in Keyboard Handler
    }
}

void keyReset(int row, int col) {           //Key release handler at
current row and col for Primary Primary
  keyHoldCount[row][col] = 0;               //reset  current row and col
Hold Count
  keyLongHold[row][col] = false;            //reset  current row and col
long hold
}

void keyResetRight(int row, int col) {      //Key release handler at
current row and col for Secondary Primary
  keyHoldCountRight[row][col] = 0;          //reset  current row and col
Hold Count
  keyLongHoldRight[row][col] = false;       //reset  current row and col
long hold
}
```

Right/ secondary keyboard Half

```
/* Title: RightKeyboardHalf (Secondary) - S00241117 Project
Code Details: Chip Driver for Pro Micro ATmega32U4 based Right Keyboard
Half
Start Date: 15/01/2024

pin D1: N/A     -|   usbc   |-  RAW
pin D0: N/A     -|          |-  GND
        GND     -|          |-  RST
        GND     -|   top    |-  VCC
pin D2: I2C SDA -|          |-  pin A3:  col 0
pin D3: I2C SCL -|          |-  pin A2:  col 1
pin D4: row 0   -|          |-  pin A1:  col 2
pin D5: row 1   -|          |-  pin A0:  col 3
pin D6: row 2   -|          |-  pin D15: col 4
pin D7: row 3   -|          |-  pin D14: col 5
pin D8: row 4   -|          |-  pin D16: col 6
pin D9: LEDSTR  -|__________|-  pin D10: col 7
*/

///// Libraries /////
#include <Wire.h>                //I2C handler
#include <Adafruit_NeoPixel.h>  //LED Strip Library

///// Global Variables /////
const int SCND_ADDY = 8;
const int ROWS = 5;
const int COLS = 8;
const int LEDPIN = 9;
const int NUMPIXELS = 8;

const int rowpin[ROWS] = {4, 5, 6, 7, 8};
const int colpin[COLS] = {A3, A2, A1, A0, 15, 14, 16, 10};

byte keyStatus[ROWS] = {
  0b00000000,
  0b00000000,
  0b00000000,
  0b00000000,
  0b00000000
};

///// Keymap, not used on secondary (right) Keypad/////
/* char keyMap[ROWS][COLS] = {
    {'6' , '7' , '8' , '9' , '0' , '-' , '=', 178 }, //[...] backspace
    {'y' , 'u' , 'i' , 'o' , 'p' , '[' , ']', '\\'},
    {'h' , 'j' , 'k' , 'l' , ';' , '\'', 176, '*' }, //[...] enter, not
used
    {'n', 'm' , ',' , '.' , '/' , 133 , 218, 212  }, //[...] shift, up, del
    {'*', ' ',  134,  237,  132,  216 , 217, 215  }, //not used [...] alt,
menu, ctrl, left, down, right
};*/

///// function prototypes /////
void printByteArray(const unsigned char* array, size_t length);
Adafruit_NeoPixel strip = Adafruit_NeoPixel (NUMPIXELS, LEDPIN, NEO_GRB +
NEO_KHZ800);

///// Arduino setup on power on /////
void setup() {
  Wire.begin(SCND_ADDY);            //startup I2C handler
```

```
  strip.begin();                    //startup LED handler

  for (int r = 0; r < ROWS; r++) {  //Set all Rows as Output (HIGH)
    pinMode(rowpin[r], OUTPUT);
  }

  for (int c = 0; c < COLS; c++) {  //Set all Columns as Pull-Up Input
    pinMode(colpin[c], INPUT_PULLUP);
  }

  Wire.onRequest(requestEvent);     //define I2C request action

  strip.clear();                    //reset all LEDs

  for(int p=0; p<NUMPIXELS; p++) {  //repeat for each LED Pixel
    strip.setPixelColor(p, 200, 120, 0);    // set LED Pixel RGB color
value (orange)
  }
  strip.show();                     //Display RGB colors on current LEDs
}

///// Arduino infinite loop function while powered on /////

void loop() {

  for (int r = 0; r < ROWS; r++) {          //repeat per Row

    digitalWrite(rowpin[r], LOW);           //Set current Row as LOW
    delayMicroseconds(10);                  //Delay to stabilize new state

    for (int c = 0; c < COLS; c++) {        //repeat per Colum
      if(digitalRead(colpin[c]) == LOW) {   //if LOW detected
        keyStatus[r] |= (1 << c);           //set byte in Key status array
      }

      else {                                //otherwise
        keyStatus[r] &= ~(1 << c);          //clear byte, to handle key
releases
      }
    }


    digitalWrite(rowpin[i], HIGH);          //reset Row Pin to High
    delayMicroseconds(100);                 //delay to stabilize new state
  }

  delay(1);                                 //delay to stabilize system
befor next repeat

///// Function to handle I2C request Event /////

void requestEvent() {
  for(int i = 0; i<5; i++){
    Wire.write(keyStatus[i]);   //Send each Byte of the current KeyStatus
Array (up to 5)
  }
}
```

Code Explained

The libraries keyboard.h, wire.h and Adafruit_NeoPixel.h were included at the beginning of the code to allow the use of their functions. An ASCII schematic of the control board pinout also was added to aid with visualization of the Pro-Micro pin assignment.

On the left keyboard half, as the primary, the keymaps for both halves are defined. This is only required on the primary as only that half will be translating the registered keypress and communicate it to the connected computer, due to this the keyboard.h was also found to be not required for the secondary keyboard half.

Following the initialisation both sides are setup by enabling the I2C and LED handlers. Following this step the rows are looped through and setup as high outputs, followed by looping through the columns and setting them as pull-up inputs. All LEDs are being set to an orange hue. Power limitation for the LEDs was found to be not required in this case as only 2 out of 3 colours are being driven, not achieving full brightness either, to create an orange color hue. The secondary also incorporates the definition of the I2C request function in the setup.

The infinite loop for the primary first is set to request the current recorded keypresses from the secondary via I2C.

The keypress loop between the halves works the same way, a row is set to be low followed by a check per column if a low is detected. If detected this will be registered as a 1 in the keyStatus byte for the current row, otherwise this remains 0. Following the check of all available columns, the row is set back to high output and the loop iterates through the next row.

The primary half will use the variables tracking each keypress, both for itself as well as the one received from the right half, to translate the keypresses registered, cross-referencing the keymaps, into the wanted value or function and then feed this back to the connected computer. This also includes a tracking function for keys being pressed continuously. Here the initial press will be translated, followed by an initial delay before printing the keypress again to the Computer. If the key remains pressed a shorter delay is used following the initial delay before a repeat print of the keypress occurs, both delay loops reset as soon as the key is registered as released.

Figure 36 - Keyboard halves, running Arduino code

User Feedback

The functional prototype was provided to some of the earlier survey participants in an attempt to gather user feedback. Most users remarked that the layout appeared to be similar enough to a standard staggered ANSI keyboard, that a transition to the ortho linear layout should pose no problem. Some of the touch-typing users remarked a minor slow-down in their typing speed, while all hybrid and hunt and peck users noted negligible differences. Particularly hunt and peck users noted the ability to clip both halves together to reassemble the normal one-piece keyboard design was of great benefit in eliminating most issues with the differing new layout.

Some users noted the absence of the function keys F1 to F12 as well as the lack of availability of a Num-Pad, the later of which was criticized as a potential risk of slowing down work for some users. Additionally, the ability to configure key functions was suggested as a feature by some of the testing users. Many stated that changing functions through the Arduino IDE was too complex for them to use.

Based on the feedback the decision was made to transition away from Arduino IDE as the platform for the keyboard software and translate the code for use with the open source project QMK. Also integration with an easier to use, preferable graphical based, user interface for configuration of the keyboard was added to the optional goals for the project.

Transition to QMK

To aid with tracking the project, all relevant code and data was transitioned to a public GitHub repository. This would allow access to the in-process project from multiple locations and act as a backup with historical record of changes.

Consulting the QMK Firmware documentation [38] it was found that multiple files would need to be generated.

1. config.h – Configuration header file for list of #define features used by the keyboard, also incorporating the commented copyright information and licensing.
2. rules.mk – Makefile with list of QMK features to be en- or disabled by default.
3. info.json – Primary setup file containing keyboard identifiers, hardware feature definitions, like controller pin assignments, and hardware based keyboard matrix, with definition of key sizing and positions.
4. keymap.c – Virtual keymap assigning key functions and layered key assignments.

Optional

5. “Keyboard Name”.c – file to determine initial keyboard settings at power-on.

QMK would also allow the editing of the keymap file using a web based graphical user interface. A downside was noted, in that this will only create a keymap for keyboards that have been successfully committed to the QMK project’s GitHub and even with that will only generate a file, that the user will still have to flash to the keyboard using the text console based QMK Toolbox utility.

While researching the above requirements in the QMK documentation, another open-source project by the name of VIA was found. This project can be integrated with QMK keyboard firmware to enable life modifications of the keyboard settings using a simple web-based GUI [53]. Given the optional goal for an easy-to-use user interface and the well documented process of integrating VIA with QMK firmware [38] which only required two additional files to be created, the VIA functionality was to be incorporated with the to be developed QMK Firmware.

The requirements, as per the documentation on developing firmware withing the QMK standards highlighted the need for unique identification of the keyboard, including a project name. The keyboard was given the project name “HoneyLocust” based on the grey and yellow/ orange color scheme chosen for the keyboard matching the tree of the same name. For USB device identification the following values were chosen:

Vid: 0xCA18 (Based on the first three letters of the first name of the project author)

Pid:0x1118 (Based on the first two letters of the last name of the project author)

Using the research into QMK and the decisions above the required files were created as follows:

config.h

```
/* Copyright 2024  C. Krammel (c.krammel@gmx.de)

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                                //code reduction directive

#define USE_I2C                             //define inter controller
communications for split setup using I2C
#define MASTER_LEFT                         //define primary Keyboard Half
based on USB connection to PC (only left will ever be connected)
#define SPLIT_MAX_CONNECTION_ERRORS 10      //Timeout to try detect
secondary halft (right), if not detect assume single half operation
#define SPLIT_CONNECTION_CHECK_TIMEOUT 5000 //check every 5s (5k ms) for
reconnection of the Secondary Half (right)

#define LOCKING_SUPPORT_ENABLE              //Mechanical locking support.
Use KC_LCAP, KC_LNUM or KC_LSCR instead in keymap
#define LOCKING_RESYNC_ENABLE               //Locking modifier
resynchronize enfocement

/* mouse function config */
#define MOUSEKEY_INTERVAL       20
#define MOUSEKEY_DELAY          0
#define MOUSEKEY_TIME_TO_MAX    60
#define MOUSEKEY_MAX_SPEED      7
#define MOUSEKEY_WHEEL_DELAY 0
```

rules.mk

```
# Build Options
#   change yes to no to disable
#
BOOTMAGIC_ENABLE = no       # Enable Bootmagic Lite
MOUSEKEY_ENABLE = yes       # Mouse keys
EXTRAKEY_ENABLE = yes       # Audio control and System control
CONSOLE_ENABLE = no         # Console for debug
COMMAND_ENABLE = no         # Commands for debug and configuration
NKRO_ENABLE = no            # Enable N-Key Rollover
BACKLIGHT_ENABLE = no       # Enable keyboard backlight functionality
AUDIO_ENABLE = no           # Audio output
RGBLIGHT_ENABLE = yes       # Enable WS2812 RGB backlight.

SPLIT_KEYBOARD = yes

LTO_ENABLE = yes
```

info.json

```
{
    "keyboard_name": "HoneyLocust",
    "manufacturer": "Error42Generator",
    "url": "https://github.com/BismuthDrake/Honey-Locust-Ortho-Split-60-",
    "maintainer": "C. Krammel",
    "usb": {
        "vid": "0xCA18",
        "pid": "0x1118",
        "device_version": "0.0.1"
    },
    "rgblight": {
        "split": true,
        "led_count": 14,
        "split_count": [6, 8],
        "max_brightness": 200,
        "animations": {
            "breathing": true,
            "rainbow_mood": true,
            "rainbow_swirl": true,
            "snake": false,
            "knight": false,
            "christmas": false,
            "static_gradient": true,
            "rgb_test": false,
            "alternating": true,
            "twinkle": true
        }
    },
    "ws2812": {
        "pin": "B5"
    },
    "matrix_pins": {
        "cols": ["F4", "F5", "F6", "F7", "B1", "B3", "B2", "B6"],
        "rows": [ "D4", "C6", "D7", "E6", "B4"]
    },
    "diode_direction": "COL2ROW",
    "split": {
        "matrix_pins": {
            "right": {
                "cols": ["F4", "F5", "F6", "F7", "B1", "B3", "B2", "B6"],
                "rows": [ "D4", "C6", "D7", "E6", "B4"]
            }
        }
    },
    "processor": "atmega32u4",
    "bootloader": "caterina",
    "layouts": {
        "LAYOUT": {
            "layout": [
                {"matrix": [0, 0], "x": 0, "y": 0},
                {"matrix": [0, 1], "x": 1, "y": 0},
                {"matrix": [0, 2], "x": 2, "y": 0},
                {"matrix": [0, 3], "x": 3, "y": 0},
                {"matrix": [0, 4], "x": 4, "y": 0},
                {"matrix": [0, 5], "x": 4, "y": 0},

                {"matrix": [5, 0], "x": 7, "y": 0},
                {"matrix": [5, 1], "x": 8, "y": 0},
                {"matrix": [5, 2], "x": 9, "y": 0},
                {"matrix": [5, 3], "x": 10, "y": 0},
```

```
{"matrix": [5, 4], "x": 11, "y": 0},
{"matrix": [5, 5], "x": 12, "y": 0},
{"matrix": [5, 6], "x": 13, "y": 0},
{"matrix": [5, 7], "x": 14, "y": 0},

{"matrix": [1, 0], "x": 0, "y": 1},
{"matrix": [1, 1], "x": 1, "y": 1},
{"matrix": [1, 2], "x": 2, "y": 1},
{"matrix": [1, 3], "x": 3, "y": 1},
{"matrix": [1, 4], "x": 4, "y": 1},
{"matrix": [1, 5], "x": 5, "y": 1},

{"matrix": [6, 0], "x": 7, "y": 1},
{"matrix": [6, 1], "x": 8, "y": 1},
{"matrix": [6, 2], "x": 9, "y": 1},
{"matrix": [6, 3], "x": 10, "y": 1},
{"matrix": [6, 4], "x": 11, "y": 1},
{"matrix": [6, 5], "x": 12, "y": 1},
{"matrix": [6, 6], "x": 13, "y": 1},
{"matrix": [6, 7], "x": 14, "y": 1},

{"matrix": [2, 0], "x": 0, "y": 2},
{"matrix": [2, 1], "x": 1, "y": 2},
{"matrix": [2, 2], "x": 2, "y": 2},
{"matrix": [2, 3], "x": 3, "y": 2},
{"matrix": [2, 4], "x": 4, "y": 2},
{"matrix": [2, 5], "x": 5, "y": 2},

{"matrix": [7, 0], "x": 7, "y": 2},
{"matrix": [7, 1], "x": 8, "y": 2},
{"matrix": [7, 2], "x": 9, "y": 2},
{"matrix": [7, 3], "x": 10, "y": 2},
{"matrix": [7, 4], "x": 11, "y": 2},
{"matrix": [7, 5], "x": 12, "y": 2},
{"matrix": [7, 6], "x": 13, "y": 2,"w": 2},

{"matrix": [3, 0], "x": 0, "y": 3},
{"matrix": [3, 1], "x": 1, "y": 3},
{"matrix": [3, 2], "x": 2, "y": 3},
{"matrix": [3, 3], "x": 3, "y": 3},
{"matrix": [3, 4], "x": 4, "y": 3},
{"matrix": [3, 5], "x": 5, "y": 3},

{"matrix": [8, 0], "x": 7, "y": 3},
{"matrix": [8, 1], "x": 8, "y": 3},
{"matrix": [8, 2], "x": 9, "y": 3},
{"matrix": [8, 3], "x": 10, "y": 3},
{"matrix": [8, 4], "x": 11, "y": 3},
{"matrix": [8, 5], "x": 12, "y": 3},
{"matrix": [8, 6], "x": 13, "y": 3},
{"matrix": [8, 7], "x": 14, "y": 3},

{"matrix": [4, 0], "x": 0, "y": 4},
{"matrix": [4, 1], "x": 1, "y": 4},
{"matrix": [4, 2], "x": 2, "y": 4},
{"matrix": [4, 3], "x": 3, "y": 4},
{"matrix": [4, 4], "x": 4, "y": 4, "w": 2},

{"matrix": [9, 1], "x": 8, "y": 4, "w": 2},
{"matrix": [9, 2], "x": 10, "y": 4},
{"matrix": [9, 3], "x": 11, "y": 4},
```

```
                {"matrix": [9, 4], "x": 12, "y": 4},
                {"matrix": [9, 5], "x": 13, "y": 4},
                {"matrix": [9, 6], "x": 14, "y": 4},
                {"matrix": [9, 7], "x": 15, "y": 4}
            ]
        }
    }
}
```

keymap.c

```
#include QMK_KEYBOARD_H

enum custom_layers {
    _BASE,
    _FUNC,
    _NUM
};

const uint16_t PROGMEM keymaps[][MATRIX_ROWS][MATRIX_COLS] = {
  [_BASE] = LAYOUT(     // Deafualt Layer Keymap
  //|------+--------+--------+--------+--------+-------|      |--------+--
------+--------+--------+--------+--------+--------+--------|
     KC_ESC, 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_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_CAPS,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_DEL,
  //|------+--------+--------+--------+--------+-------|      |--------+--
------+--------+--------+--------+--------+--------+--------|
             KC_LCTL, KC_LGUI, KC_LALT, MO(2),   KC_SPC,
KC_SPC, MO(1),   KC_RALT, KC_APP,  KC_LEFT, KC_DOWN,  KC_RGHT
  //|------+--------+--------+--------+--------+-------|      |--------+--
------+--------+--------+--------+--------+--------+--------|
  ),

  [_FUNC] = LAYOUT(     // FN1 Layer Keymap
  //|------+--------+--------+--------+--------+-------|      |--------+--
------+--------+--------+--------+--------+--------+--------|
     KC_GRV,  KC_F1,   KC_F2,   KC_F3,   KC_F4,   KC_F5,         KC_F6,
KC_F7,   KC_F8,   KC_F9,   KC_F10,  KC_F11,  KC_F12,  KC_DEL,
  //|------+--------+--------+--------+--------+-------|      |--------+--
------+--------+--------+--------+--------+--------+--------|
     RGB_TOG, _______, RGB_VAI,_______, _______, _______,       _______,
_______, _______, _______, _______, _______, _______, _______,
  //|------+--------+--------+--------+--------+-------|      |--------+--
------+--------+--------+--------+--------+--------+--------|
     _______, RGB_RMOD,RGB_VAD,RGB_MOD, _______, _______,       KC_VOLU,
_______, _______, _______, _______, _______, _______,
  //|------+--------+--------+--------+--------+-------|      |--------+--
------+--------+--------+--------+--------+-----------------|
     _______, _______, _______,_______, _______, _______,       KC_VOLD,
KC_MUTE, KC_MRWD, KC_MFFD, _______, KC_BTN1, KC_MS_U, KC_BTN2,
```

```
  //|-------+--------+--------+--------+--------+-------|       |--------+--
------+--------+--------+--------+--------+--------+--------|
                 _______, _______,_______, _______, _______,
KC_MPLY, _______, _______, _______, KC_MS_L, KC_MS_D, KC_MS_R
  //|-------+--------+--------+--------+--------+-------|       |--------+--
------+--------+--------+--------+--------+--------+--------|
  ),

  [_NUM] = LAYOUT(      //FN2 Layer Keymap
  //|-------+--------+--------+--------+--------+-------|       |--------+--
------+--------+--------+--------+--------+--------+--------|
     _______, _______, _______,_______, _______, _______,        KC_EQL,
KC_SLSH, KC_ASTR, KC_MINS, _______, _______, _______, _______,
  //|-------+--------+--------+--------+--------+-------|       |--------+--
------+--------+--------+--------+--------+--------+--------|
     _______, _______, _______,_______, _______, _______,        KC_7,
KC_8,    KC_9,    KC_PLUS, _______, _______, _______, _______,
  //|-------+--------+--------+--------+--------+-------|       |--------+--
------+--------+--------+--------+--------+--------+--------|
     _______, _______, _______,_______, _______, _______,        KC_4,
KC_5,    KC_6,    KC_PLUS, _______, _______, _______,
  //|-------+--------+--------+--------+--------+-------|       |--------+--
------+--------+--------+--------+--------+-----------------|
     _______, _______, _______,_______, _______, _______,        KC_1,
KC_2,    KC_3,    KC_ENT,  _______, _______, _______, _______,
  //|-------+--------+--------+--------+--------+-------|       |--------+--
------+--------+--------+--------+--------+--------+--------|
                 _______, _______,_______, _______, _______,
KC_0,    KC_DOT,  KC_ENT,  _______, _______, _______, _______
  //|-------+--------+--------+--------+--------+-------|       |--------+--
------+--------+--------+--------+--------+--------+--------|
  ),
};
```

```
#include QMK_KEYBOARD_H

enum custom_layers {
    _BASE,
    _FUNC,
    _NUM
};

const uint16_t PROGMEM keymaps[][MATRIX_ROWS][MATRIX_COLS] = {
  [_BASE] = LAYOUT(     // Deafualt Layer Keymap
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
     KC_ESC, 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_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_CAPS,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_DEL,
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
            KC_LCTL, KC_LGUI, KC_LALT, MO(2),   KC_SPC,             KC_SPC, MO(1),   KC_RALT, KC_APP,  KC_LEFT, KC_DOWN,  KC_RGHT
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
  ),

  [_FUNC] = LAYOUT(     // FN1 Layer Keymap
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
     KC_GRV,  KC_F1,   KC_F2,   KC_F3,   KC_F4,   KC_F5,      KC_F6,   KC_F7,   KC_F8,   KC_F9,   KC_F10,  KC_F11,  KC_F12,  KC_DEL,
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
     RGB_TOG, _______, RGB_VAI,_______, _______, _______,     _______, _______, _______, _______, _______, _______, _______, _______,
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
     _______, RGB_RMOD,RGB_VAD,RGB_MOD, _______, _______,     KC_VOLU, _______, _______, _______, _______, _______,
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
     _______, _______, _______,_______, _______, _______,     KC_VOLD, KC_MUTE, KC_MRWD, KC_MFFD, _______, KC_BTN1, KC_MS_U, KC_BTN2,
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
              _______, _______,_______, _______, _______,              KC_MPLY, _______, _______, _______, KC_MS_L, KC_MS_D, KC_MS_R
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
  ),

  [_NUM] = LAYOUT(      //FN2 Layer Keymap
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
     _______, _______, _______,_______, _______, _______,     KC_EQL,  KC_SLSH, KC_ASTR, KC_MINS, _______, _______, _______, _______,
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
     _______, _______, _______,_______, _______, _______,     KC_7,    KC_8,    KC_9,    KC_PLUS, _______, _______, _______, _______,
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
     _______, _______, _______,_______, _______, _______,     KC_4,    KC_5,    KC_6,    KC_PLUS, _______, _______, _______,
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
     _______, _______, _______,_______, _______, _______,     KC_1,    KC_2,    KC_3,    KC_ENT,  _______, _______, _______, _______,
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
              _______, _______,_______, _______, _______,              KC_0,    KC_DOT,  KC_ENT,  _______, _______, _______, _______
  //|------+------+------+------+------+------|     |------+------+------+------+------+------+------+------|
  ),
};
```

Figure 37 - keymap.c formatting based on keyboard layout, as seen in coding tool (Notepad++)

HoneyLocust.c

```
#include "quantum.h"

void eeconfig_init_kb(void) {
#ifdef RGBLIGHT_ENABLE
    rgblight_enable();              // Enable RGB by default
    rgblight_sethsv(0, 200, 120);   // Set default HSV - orange hue, high
saturation, high brightness
#ifdef RGBLIGHT_EFFECT_RAINBOW_SWIRL
    rgblight_mode(RGBLIGHT_MODE_RAINBOW_SWIRL + 2); // RGB_RAINBOW_SWIRL to
be set as default on power on default
#endif
#endif

    eeconfig_update_kb(0);
    eeconfig_init_user();
}
```

To support integration with VIA the two additional files were required. rules.mk for enabling feature support within the VIA components of the QMK, as well as via.json to translate the to be displayed layout of keys on the GUI, the identifier of the keyboard for the web app to detect and the enabled extra features like backlighting.

via.json

```
{
    "name": "Honey Locust",
    "vendorId": "0xCA18",
    "productId": "0x1118",
    "matrix": {
        "rows": 10,
        "cols": 8
    },
    "layouts":{
        "keymap": [

["0,0","0,1","0,2","0,3","0,4","0,5",{"x":0.5},"5,0","5,1","5,2","5,3","5,4
","5,5","5,6","5,7"],

["1,0","1,1","1,2","1,3","1,4","1,5",{"x":0.5},"6,0","6,1","6,2","6,3","6,4
","6,5","6,6","6,7"],

["2,0","2,1","2,2","2,3","2,4","2,5",{"x":0.5},"7,0","7,1","7,2","7,3","7,4
","7,5",{"w":2},"7,6"],

["3,0","3,1","3,2","3,3","3,4","3,5",{"x":0.5},"8,0","8,1","8,2","8,3","8,4
","8,5","8,6","8,7"],

["4,1","4,2","4,3","4,4",{"w":2},"4,5",{"x":0.5,"w":2},"9,1","9,2","9,3","9
,4","9,5","9,6","9,7"]
        ]
    },
    "keycodes": ["qmk_lighting"],
    "menus": ["qmk_rgblight"]
}
```

rules.mk

```
VIA_ENABLE = yes        #Allows for VIA Tool integration
LTO_ENABLE = yes        #Link Time Optimization enabled in compiler to
reduce code size
```

The files were organized after creation as per the QMK documentation. Note the second rules.mk to support VIA needed was placed in the keymaps folder with the keymap.c and via.json.

rules.mk	09/04/2024 12:03	MK File	1 KB
info.json	09/04/2024 17:05	JSON File	5 KB
HoneyLocust.c	04/04/2024 23:34	C File	1 KB
config.h	09/04/2024 12:05	H File	2 KB
keymaps	09/04/2024 12:01	File folder	
└ via	09/04/2024 10:49	File folder	
├ keymap.c	09/04/2024 18:27	C File	5 KB
├ rules.mk	09/04/2024 12:36	MK File	1 KB
└ via.json	09/04/2024 17:06	JSON File	1 KB

Figure 38 - QMK firmware files folder structure

Using the compile commands within the application QMK MSYS the firmware files within the folder structure shown in Figure 38 were translated into a machine code .hex file. This file was then flashed onto the Pro-Micro control board using the QMK Toolbox application, for this the bootloader of the control board needed to be unlocked, which was temporarily achieved by bridging the reset and ground pins on the control board, while connected via USB. Following the flashing process QMK Toolbox showed as successful flashing of the new firmware onto the controller. To reduce the risk of damaging an already wired up controller within the prototype keyboard a spare control board was used for this process. Thanks to the inbuilt key-tester within QMK Toolbox functionality of the board could be confirmed by bridging contacts between the row and column assigned pins, which would highlight the assigned key in green in the testing window of the software.

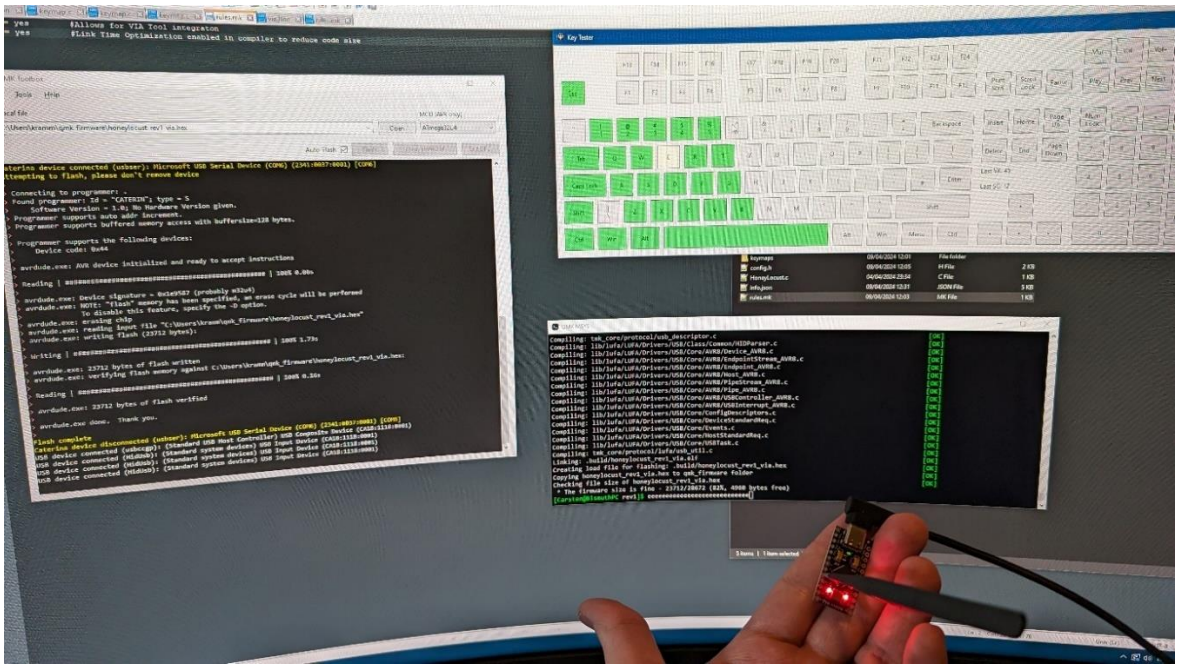


Figure 39 - QMK MSYS after compiling firmware on the left, QMK Toolbox after flashing Pro-Micro control board on the bottom and QMK Toolbox key tester on top, operated by bridging row and column pins to simulate keypresses

The firmware, following successful testing, was flashed to both halves of the keyboard prototype. The advantage noted here as compared to the earlier approach using Arduino IDE, was that the same firmware file was used for both control boards. The QMK Firmware as written would handle the identification of the primary and secondary keyboard half based on the USB connection to the computer, as the secondary would never be directly connected.

Figure 40 - Keyboard with VIA compatible QMK firmware and enabled RGB LED backlighting animations

Following a full functionality test confirming the correct operation of each key using the key test the integration with the VIA web app was tested. For this the app was accessed through https://usevia.app/ . Since the keyboard had not been committed to either QMK or VIA GitHub projects automated detection within the VIA app could not be supported. It was necessary to enable the “Show Design tab” in the settings of the VIA web application. Following this adjustment the design tab was accessed and using the “Load Draft” button the earlier created via.json could be uploaded to configure the app for detection of and use with the prototype keyboard. The correct upload could be confirmed as the VIA app previewed a blank version of the keyboard with no key assignments. Once this was done, the configure tab was accessed and the connected device prototype was authorised through the in-browser pop-up window.

Following this process the VIA web application now displayed the keyboard with current key assignments and allowed for easy re-assignment of functions using drag and drop. Within the lighting menu it was also possible to configure the lighting animations of the backlighting for intensity, colour and speed of animation as applicable to the respective animation setting chosen.

Any changes made would apply to the keyboard life and remain preserved even after disconnecting the keyboard from the computer. This would be a great benefit to users, now able to configure the keyboard easily to their liking on any computer with web access, but also preserve the changes if moving the keyboard between computers.

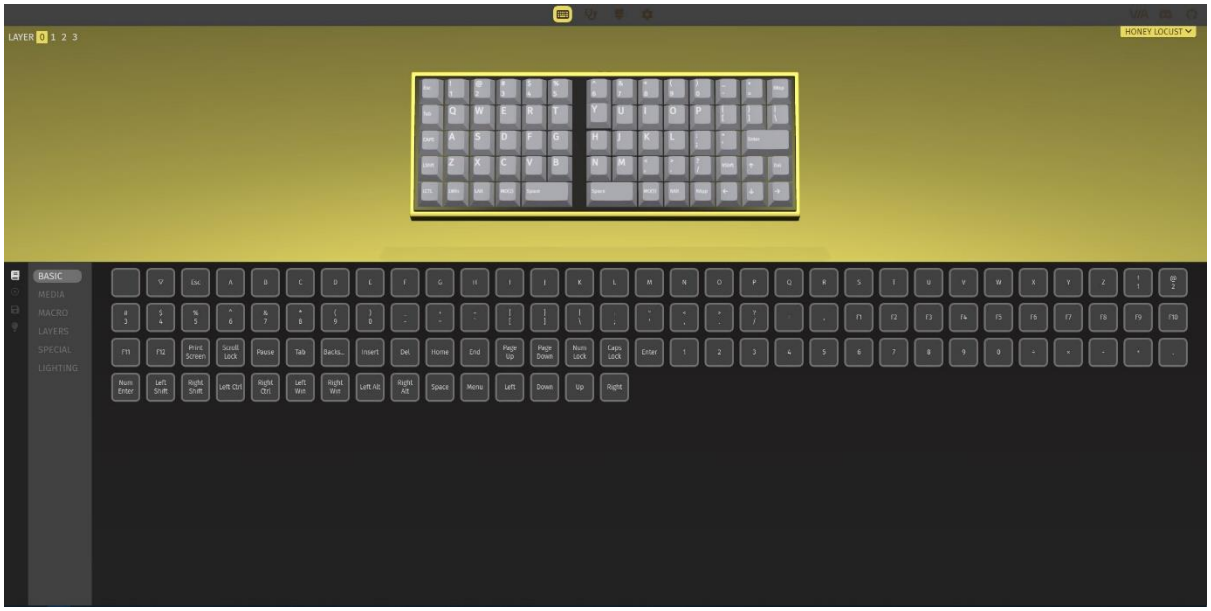


Figure 41 - VIA web application with connected and authorized HoneyLocust keyboard prototype

The final prototype of the HoneyLocust keyboard project incorporated controllable backlighting, integration with the VIA web application and re-configurable key assignments based on prior user feedback. The Num-Pad function can be accessed using the “layer 2” key beside the space bar on the left/ primary keyboard half. The Num-Pad keys are then overlayed in the same layout of a normal full—size keyboard’s Num-Pad with the keys of the right keyboard half using the spacebar itself as the “0” key. The “layer 1” key on the right/ secondary keyboard half, also located beside the spacebar, can be used to access function keys F1 to F12, media functions like play/ pause, next or previous track, lighting controls and even the functions of a computer mouse, using the arrow cluster for cursor movement and the keys above the right and left arrow as respective right and left mouse buttons.

With the ability to reassign keys freely and stagger up to 3 layers of functions per key, users may even choose to configure the primary keyboard half on its own as a fully operational one-handed keyboard. Given there are up to 4 functions per key in total, this can even be achieved without having to compromise on any functions or keys found on a full-size keyboard.

Conclusion

Given the research on ergonomics the biggest take-away was that an ergonomic focused keyboard needs to be able to adapt to the user, not the other way around. As such the final prototype of the HoneyLocust Keyboard can be used in either split configuration, as a single-piece keyboard or even configured for one handed use.

Furthermore the integration with the easy-to-use VIA web application and QMK Firmware allows for a broad variety of functions easily accessible to the user. This includes but is not limited to such functions like multiple layered functions per key, mouse button and movement functions, as well as configurable backlighting.

User feedback was gathered a final time from a limited amount of participants using the final prototype. The keyboard was well received, having incorporated many feature suggestions and requests from the first and second user feedback during the development process. The incorporation of a Num-Pad in a small size keyboard was highly appreciated by many users as well as the integration with the easy to use and understandable VIA web application.

The development process highlighted the benefits of a group developed and open-source project such as QMK Firmware and VIA. Not just do such projects allow for incorporation of a broad variety of features thanks to multiple contributors, but they also provide an open platform for new users to integrate new projects with. Therefore in the spirit of the two used projects, the decision was made to also publish this project as an open-source project on the GitHub platform under the GNU general public license and Creative Commons Zero v1.0 Universal licenses. Therefore all code and development files can be accessed under the following address:

https://github.com/BismuthDrake/Honey-Locust-Ortho-Split-60-

The freely available documentation as well as the choice of broadly available components also contributes to the ability of users to either modify or repair this keyboard project as applicable to their needs. Further supporting accessibility to users as well as sustainability is that the goal of developing the entire keyboard on an overall component cost of less than €100 was met. With the cost for 3D Printing filament used and cables considered, including the 3.5mm TRRS audio and USB-C to USB-A cables, the overall component cost arrived at a final value of ca. €75. This puts the price well below the cost of comparable ergonomics focused keyboard like the Microsoft Sculpt at ~€180 or the ErgoDox EZ Split at ~€305.

Sources and References

[1] A. D. Kroemer and K. H. E. Kroemer, *Office Ergonomics: Ease and Efficiency at Work, Second Edition*. Milton: Chapman and Hall/CRC, 2016. ISBN: 9781498747943

[2] A. Y. Douglas, T. J. Mehan, C. L. Collins, G. A. Smith, and L. B. McKenzie, "Acute computer-related injuries treated in U.S. emergency departments, 1994–2006," *American Journal of Preventive Medicine*, vol. 37, no. 1, pp. 24–28, 2009. doi:10.1016/j.amepre.2009.04.001

[3] H. A. Kumar, "Computer Injuries and its Prevention in Scientific Ways," *ACTA SCIENTIFIC PAEDIATRICS*, vol. 2, no. 5, pp. 3–7, Apr. 2019. Accessed: Nov. 2, 2023. [Online]. Available: https://actascientific.com/ASPE/pdf/ASPE-02-0071.pdf

[4] P. C. Bohr, *Journal of Occupational Rehabilitation*, vol. 10, no. 4, pp. 243–255, Dec. 2000. doi:10.1023/a:1009464315358

[5] K. Yasuoka and M. Yasuoka, "On the Prehistory of QWERTY," *Kyoto University Research Information Repository 320_Institute for Research in Humanities ZINBUN*, vol. 42, pp. 161–174, Mar. 2011. doi:10.14989/139379

[6] P. A. David, *Understanding the Economics of QWERTY: The Necessity of History*. New York and Oxford: Basil Blackwell, 1978.

[7] J. Noyes, "The QWERTY keyboard: A Review," International Journal of Man-Machine Studies, vol. 18, no. 3, pp. 265–281, 1983. doi:10.1016/s0020-7373(83)80010-8

[8] P. A. David, "'Clio and the Economics of QWERTY.,'" JSTOR, http://www.jstor.org/stable/1805621 (accessed Oct. 12, 2023).

[9] V. Ahlstrom and B. Kudrick, "Human Factors Criteria for the Design and Acquisition of Keyboards: A Revision to Chapter 9 of the Human Factors Design Standard ", National Technical Information Service, Springfield, Virginia, Report, 2004, FAA & DOT

[10] EF Pascarelli and JJ. Kella, "Soft-tissue injuries related to use of the computer keyboard. A clinical study of 53 severely injured persons." Journal of Occupational Medicine, May 1993, 35(5):522-32, PMID: 8515325.

[11] A. P. Verhagen *et al.*, "Conservative interventions for treating work-related complaints of the arm, neck or shoulder in adults," *Cochrane Database of Systematic Reviews*, 2013. doi:10.1002/14651858.cd008742.pub2

[12] K. Keller, J. Corbett, and D. Nichols, "Repetitive strain injury in computer keyboard users: Pathomechanics and treatment principles in individual and group intervention," *Journal of Hand Therapy*, vol. 11, no. 1, pp. 9–26, 1998. doi:10.1016/s0894-1130(98)80056-2

[13] A. Martin, “A new keyboard layout,” *Applied Ergonomics*, vol. 3, no. 1, pp. 48–51, 1972. doi:10.1016/0003-6870(72)90011-7

[14] K. Streams, “How to shop for a mechanical keyboard,” The New York Times, https://www.nytimes.com/wirecutter/blog/how-to-shop-for-a-mechanical-keyboard/ (accessed Oct. 13, 2023). Illustrations by Sarah MacReading & Dana Davis

[15] “Qwertz,” Wikipedia, https://en.wikipedia.org/wiki/QWERTZ (accessed Nov. 2, 2023).

[16] *Keyboarding Typing Skill: Step-by-Step Training Manual*. Kota Kinabalu: CP Forms, 2012. ISBN: 9789670241401

[17] Thompson, Mildred, “A Study of the Effect of Hunt-and-peck Habits on Typing Achievement.” N.p., Colorado State College of Education, Division of Education, 1944.

[18] M. van Weerdenburg, M. Tesselhof, and H. van der Meijden, “Touch-typing for Better Spelling and narrative-writing skills on the computer,” *Journal of Computer Assisted Learning*, vol. 35, no. 1, pp. 143–152, 2018. doi:10.1111/jcal.12323

[19] TheKnowledgeAcademy, “Benefits of touch typing: Explanation and defination,” , https://www.theknowledgeacademy.com/blog/benefits-of-touch-typing/ (accessed Nov. 21, 2023).

[20] A. M. Feit, D. Weir, and A. Oulasvirta, “How we type,” *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016. doi:10.1145/2858036.2858233

[21] “Touch typing,” Wikipedia, https://en.wikipedia.org/wiki/Touch_typing (accessed Nov. 24, 2023).

[22] M. Pinola and D. Gershgorn, “The best ergonomic keyboard,” The New York Times, https://www.nytimes.com/wirecutter/reviews/comfortable-ergo-keyboard/ (accessed Nov. 25, 2023).

[23] N. G. Swanson, T. L. Galinsky, L. L. Cole, C. S. Pan, and S. L. Sauter, “The impact of keyboard design on comfort and productivity in a text-entry task,” *Applied Ergonomics*, vol. 28, no. 1, pp. 9–16, Feb. 1997. doi:10.1016/s0003-6870(96)00052-x

[24] M. J. Smith *et al.*, “Effects of a split keyboard design and wrist rest on performance, posture, and comfort,” *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 40, no. 2, pp. 324–336, Oct. 1998. doi:10.1518/001872098779480451

[25] Tech-Fairy.com, “Staggered vs ortholinear keyboard, what are the differences?,” Tech Fairy, https://tech-fairy.com/staggered-vs-ortholinear-keyboard-what-are-the-differences/ (accessed Nov. 30, 2023).

[26] D. Rambo, “Building, Coding, Typing,” *COMPUTATIONAL CULTURE*, no. 8., ISSN 2047-2390

[27] W. S. Hoyle, M. C. Bartha, C. A. Harper, and S. C. Peres, “Low profile keyboard design,” *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 57, no. 1, pp. 1348–1352, Sep. 2013. doi:10.1177/1541931213571297

[28] H. Brunner and R. M. Richardson, “Effects of keyboard design and typing skill on user keyboard preferences and throughput performance,” *Proceedings of the Human Factors Society Annual Meeting*, vol. 28, no. 3, pp. 267–271, Oct. 1984. doi:10.1177/154193128402800316

[29] “Comparing mechanical, membrane and scissor-switch membrane keyboards [...],” https://www.ergopedia.ca/ergonomic_concepts/Mechanical_Keyswitches_Membrane_Keyswitches_and_Scissor_Switch_Membrane_Keyswitches_Ergonomic_Considerations.html (accessed Dec. 2, 2023).

[30] W. Antonelli, “A beginner’s guide to mechanical keyboards - how they work and compare to standard keyboards,” Business Insider, https://www.businessinsider.com/guides/tech/what-is-a-mechanical-keyboard?r=US&IR=T (accessed Dec. 3, 2023).

[31] T. Tablante, “How do computer keyboards work? ,” YouTube, https://www.youtube.com/watch?v=h-NM1xSSzHQ&ab_channel=BranchEducation (accessed Dec. 30, 2023).

[32] J. Scotto, “How a mechanical keyboard works (matrix and direct wiring),” YouTube, https://www.youtube.com/watch?v=7LyziNdFlew&list=WL&index=9&ab_channel=JoeScotto (accessed Dec. 30, 2023).

[33] S. Hossain, “Microsoft Sculpt Ergonomic Keyboard Review,” RTINGS.com, https://www.rtings.com/keyboard/reviews/microsoft/sculpt-ergonomic-keyboard (accessed Jan. 2, 2024).

[34] E. Contreras, “How to: Disassemble Microsoft sculpt ergonomic keyboard and make it wired,” Emmanuel Contreras, https://emmanuelcontreras.com/2017/06/07/how-to-disassemble-microsoft-sculpt-ergonomic-keyboard-and-make-it-wired/ (accessed Jan. 4, 2024).

[35] “ErgoDox EZ: An incredible mechanical ergonomic keyboard,” ErgoDox EZ: An Incredible Mechanical Ergonomic Keyboard | ErgoDox EZ, https://ergodox-ez.com/ (accessed Feb. 25, 2024).

[36] D. Beauchamp, “Ergodox PCB set,” Profet Keyboards, https://shop.profetkeyboards.com/product/ergodox-pcbs (accessed Feb. 27, 2024).

[37] I. Prest, “Keyboard Layout editor,” Keyboard Layout Editor, http://www.keyboard-layout-editor.com/ (accessed Feb. 29, 2024).

[38] J. Humbert, et al., “QMK firmware Documentation,” QMK Firmware, https://docs.qmk.fm/#/newbs (accessed Mar. 12, 2024).

[39] QMK Project “Qmk_firmware/docs/compatible_microcontrollers.MD at master · qmk/qmk_firmware,” GitHub, https://github.com/qmk/qmk_firmware/blob/master/docs/compatible_microcontrollers.md (accessed Mar. 2, 2024).

[40] T. Baart, “Reducing firmware size in QMK,” Thomas Baart, https://thomasbaart.nl/2018/12/01/reducing-firmware-size-in-qmk/ (accessed Mar. 5, 2024).

[41] Microchip Inc. “ATMEGA32U4 ,” microchip.com, https://www.microchip.com/en-us/product/ATmega32U4 (accessed Mar. 8, 2024).

[42] Raspberry Pi Ltd “Raspberry pi documentation - RP2040,” raspberrypi.com, https://www.raspberrypi.com/documentation/microcontrollers/rp2040.html (accessed Mar. 8, 2024).

[43] STMicroelectronics “STM32F103 Documentation,” STMicroelectronics, https://www.st.com/en/microcontrollers-microprocessors/stm32f103.html (accessed Apr. 2, 2024).

[44] SparkFun Electronics “Pro Micro DataSheet,” SparkFun, https://www.sparkfun.com/products/12640 (accessed Apr. 2, 2024).

[45] Mouser Electronics Inc. “1N4148 Diotec Semiconductor | mouser,” Mouser Electronics, https://www.mouser.com/ProductDetail/Diotec-Semiconductor/1N4148?qs=OlC7AqGiEDmWf1/PHy84oA%3D%3D (accessed Apr. 2, 2024).

[46] Cherry Europe GmbH “MX RGB ergo clear switch kit,” Cherry, https://www.cherry-world.com/mx-rgb-ergo-clear-switch-kit (accessed Mar. 14, 2024).

[47] USB Implementers Forum Inc. “USB4® Specification V2.0,” USB, https://www.usb.org/document-library/usb4r-specification-v20 (accessed Apr. 3, 2024).

[48] Cherry Europe GmbH “Cherry MX developer information: Individual MX keyboard,” Cherry, https://www.cherry-world.com/cherry-mx/developer (accessed Mar. 16, 2024).

[49] SparkFun Electronics, “I2C Explained,” I2C - SparkFun Learn, https://learn.sparkfun.com/tutorials/i2c (accessed Mar. 22, 2024).

[50] Arduino, “Keyboard,” Keyboard - Arduino Reference, https://www.arduino.cc/reference/en/language/functions/usb/keyboard/ (accessed Jan. 15, 2024).

[51] Arduino, “Wire,” Wire i2C - Arduino Reference, https://www.arduino.cc/reference/en/language/functions/communication/wire/ (accessed Jan. 15, 2024).

[52] Adafruit, “Adafruit/ADAFRUIT_NEOPIXEL: Arduino Library for controlling single-wire led pixels (Neopixel, WS2812, etc..),” GitHub, https://github.com/adafruit/Adafruit_NeoPixel (accessed Jan. 15, 2024).

[53] Via Team, “Via,” VIA, https://www.caniusevia.com/ (accessed Mar. 22, 2024).