ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI INTÉZET

FINAL ESSAY

OE-KVK
2023.

Konstantaras Manolis
T009289/FI12904/K

Reading board for blind and visually impaired people

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN FACULTY OF ELECTRICAL ENGINEERING
ELECTRONIC AND COMMUNICATION SYSTEMS INSTITUTE
INSTRUMENTATION AND AUTOMATION DEPARTMENT

STUDENT'S DECLARATION

I the undersigned student hereby declare that this degree project / thesis is the result of my own work; I have disclosed the references and tools used in an identifiable manner. The results indicated in the degree project / thesis completed may be used by the university and the institution announcing the task to their own purposes free of charge, subject to any restrictions regarding confidentiality.

Dated: Budapest, ... 12.06. 2023

.....

student's signature

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbuda University
Kandó Kálmán Faculty of Electrical Engineering

Institute of Instrumentation and Automation

THESIS ASSIGNMENT

Student's surname, forename (s): Manolis Konstantaras
Thesis number: SZD2309040923444780[redacted]
Registration number: T009289/FI12904/K **Neptun code:** [redacted]

Branch of study:
Specialization: Instrumentation and automation specialisation

Thesis title: Reading board for blind and visually impaired people
Task description: Plan the digitization and recognition of the text! Write the compiler! Design the board's hardware and control software!

Institutional consultant's name: József Zsolt Molnár

The limitation period of the theme issued: 31.12.2025.
Deadline for submission of Thesis: 15.12.2023.

The thesis is: Not secreted.

Issued: Budapest, 25.10.2023.

Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar Elektronikai és Kommunikációs Rendszerek Intézet

P.H.

.....
Director of Institute

The Thesis is suitable for submission:

.....
Institutional consultant

.....
External consultant

Table of contents:

1. Introduction

1.1 The history of Braille writing

The visually impaired people didn't have the chance to read, until the French inventor Louis Braille developed the Braille code in 1824, at the age of fifteen, a type of writing which was an improvement on the night writing (an early form of tactile writing). Braille characters are made up of a 3x2 dot matrix which is called the Braille cell. A Braille cell can have multiple combinations of different raised dots, these combinations make up letters or abbreviations. There are 3 main types of Braille writing:

- Uncontracted braille – the traditional letter-by-letter transcription of basic letters
- Contracted braille – the addition of abbreviations and other contractions for space saving
- Grade 3 – various non-standardized personal stenography that is less commonly used [9]

As technology evolved with time, new methods of braille writing arose, such as the braille typewriter, braille printers, or nowadays the refreshable braille displays. These refreshable braille displays are rather useful to the visually impaired, because they provide them with the ability to read contents of e-mails, messages, articles, books, etc. from their phones or computers, by connecting these displays to their devices. Refreshable braille displays are also widely used by unsighted programmers. Although these devices are very helpful and revolutionary, they do require quite a significant sum of money. The most common cause for this phenomenon is that in these commercially produced refreshable braille displays, rather expensive piezoelectric actuators are used, instead of other methods such as their relatively cheaper electromechanical actuator counterparts. The main reason behind this is that as of now, piezoelectric actuators are more reliable in terms of prediction of motion and precision, despite that, they also require continuous control and precise regulation of the voltage levels to ensure that the actuator doesn't get damaged besides operating as intended. Instead by using electromechanical actuators, that can move a Braille pin with the help of a magnetized electromagnetic core and some small magnets, we could cut down the costs significantly in a project like this. This way we could not only reduce the cost of such a device, but also be able to the reduce the size and the weight of it. Although precision is lacking a bit in such an actuator, it is still a viable option for operating a braille cell as the magnetized pin generates

enough force to resist the pressure applied by one's fingertip, while reading the dots. Furthermore, besides precision, another drawback of the mechanism is also evident, which manifests itself in the way of magnetic crosstalk (not just between neighboring electromagnets inside the braille cell, but also external magnetic interference) that could interfere with the magnetized core used in the design, which then would influence the magnets inside the actuator housing, and move the braille dot from one state to another without any control signals. This would be rather inconvenient, nevertheless fortunately this can almost be completely suppressed by the usage of a magnetic shielding around each braille dot's actuator mechanism and maybe even around each braille cell.

1.2 The scope of the project

This project's scope is to create a device that can help the visually impaired people, to read bodies of text they can find on their own, or on other people's computers, furthermore, to ease the ability for blind people to communicate with the mute/deaf individuals. The goal is to create a rather compact device that is not too big, thus being portable, and is able to fulfill its goal of allowing its user to get information from a screen, in a situation where listening to the text is either not possible or is not the best approach.

Its purpose is to read a text, which is then fed to the microcontroller, is converted into a text ready for encoding, and is divided up into words or smaller segments, which is then finally displayed as on the tactile display board as Braille text through 3x2 matrix modules. Reading text files written by others is not its only purpose, however it serves that one perfectly. The text file which is read by the microcontroller on the press of a button, could be edited to contain anything to the user's desire, from text messages to articles on the web. However, this function would require further development of the product.
The developed device also features buttons to navigate through the text string with buttons which behave like the operation of turning a page of a book.

I am aware of the advantages and disadvantages of this device, or rather the principle of reading Braille characters, not being the fastest or most comfortable option. While it is a must for the visually impaired to be able to read in Braille writing, which is widely used, most commonly found on elevator buttons or ATM machines' buttons, they often instead rely on their hearing, when dealing with situations regarding using any technology. For example, the usage of a smartphone is possible for blind people through the usage of the narrator that the

modern phones utilize, to read text out loud from the screen so that the user can navigate the phone.

In other words, some blind people rather rely on audio instead of the slower procedure of reading, however in some cases it would be advantageous to read instead of listening. This is where the refreshable braille display comes to place. In the following section I will expand upon the possible usages and applications of this device, complete with its advantages and disadvantages.

2. Expanding upon the problem

The question is, why would a board, that lets a blind person to read be better than listening to that information instead? For us to understand the question we will have to ask ourselves: Why would we prefer reading a book instead of listening to it? We could answer that with: Reading is often faster than listening to a slow recording. However, we could always make the recording's playback speed faster, yet we still cling to reading instead of listening most of the time.

Let's suppose that we are travelling though the city using the crowded public transport, and we get a message that contains private information. Information that only concerns us, the one who received it, let it be something about business, private information, or an intimate message. We wouldn't want anyone else to hear/read that information, so we won't risk listening to it, even using headphones, as someone sitting next to us might overhear it, we will carefully read it. In the case of a blind person there is another problem, which is they might not know if the next message/email contains any sensitive information, so they will not risk listening to it, this way reading Braille characters is much more evident for them. Not to mention that the usage of headphones, earbuds for the visually impaired can be highly disadvantageous as they heavily rely on their hearing, due to the lack of vision.

This is the reason why reading for the blind can be advantageous over listening to audio narration.

3. Bibliographical research

I searched for several similar projects, and I found quite a few that grabbed my attention. There are several types of these projects ranging from just six actuators that make up one braille characters, to entire boards with multiple rows of braille characters. I also found one which was just a board that gives haptic feedback to the user, being able to “draw” whatever the user wants on the board by activating the correct actuators.[1]

I have also found another small project on YouTube using commercially made piezoelectric actuators manufactured by a German company called Metec Ag., instead of electromagnetic actuators.[2]

There was also a project called MagTics[3]:
MagTics is a wearable haptic interface, which was made for the purpose of receiving notifications from one’s phone, reading the time, or for the purpose of achieving other such tasks. It functions similarly to a smart watch. It uses rigid, small latching electromagnetic actuators, with a rather flexible 3D printed casing. It also uses a lightweight shielding on each actuator to eliminate interference through the electromagnetic actuator arrays. It is also made in a way which allows the user to wrap it around its hand, or along its finger for example.

The shielded magnet is formed by a neodymium permanent magnet embedded within a cylindrical shell made of low-carbon steel, a soft magnetic material that can be easily magnetized or demagnetized, keeping both poles of the permanent magnet unshielded. As a result, the magnetic field generated by the permanent magnet is focused into the top and bottom faces, i.e., in the region of the actuation coils and the latching plates, while the lateral magnetic flux in the central region is confined within the shielding material.[3]

This device is also suitable for usage with virtual reality (VR) or haptic feedback that increases the sense of presence and immersion by rendering collisions, shapes, and forces between the user and virtual objects through augmented reality (AR).[3]

In the next section I will mention two modules that I have found during my research, with different working principles, that use different technology.

4. Planning of the Braille character

4.1 Braille cell examples

For the first step in achieving a working Braille display, I need to design a character, made up of a mechanism that can push out and retract 6 pins, that provide haptic feedback to the user. I have seen several solutions to the problem in terms of different mechanisms used, and there's at least two solutions that I would like to mention. Both these solutions have their respective advantages and disadvantages that I am going to list below. The first one, which is already mentioned above, is a prebuilt Braille character composed of piezoelectric actuators. The company that designed this particular one, METEC[4] has three Braille designs available. The one I chose for illustrative purposes is the P20 model.[4]
The second one is a custom-made Braille cell which was designed by a contender in a funding project, in order to reduce cost of reading boards, compared to commercially available Braille reading boards that utilize piezoelectrically actuated Braille dots. That one uses electromagnetic actuators instead.

Braille cell P20 [4]

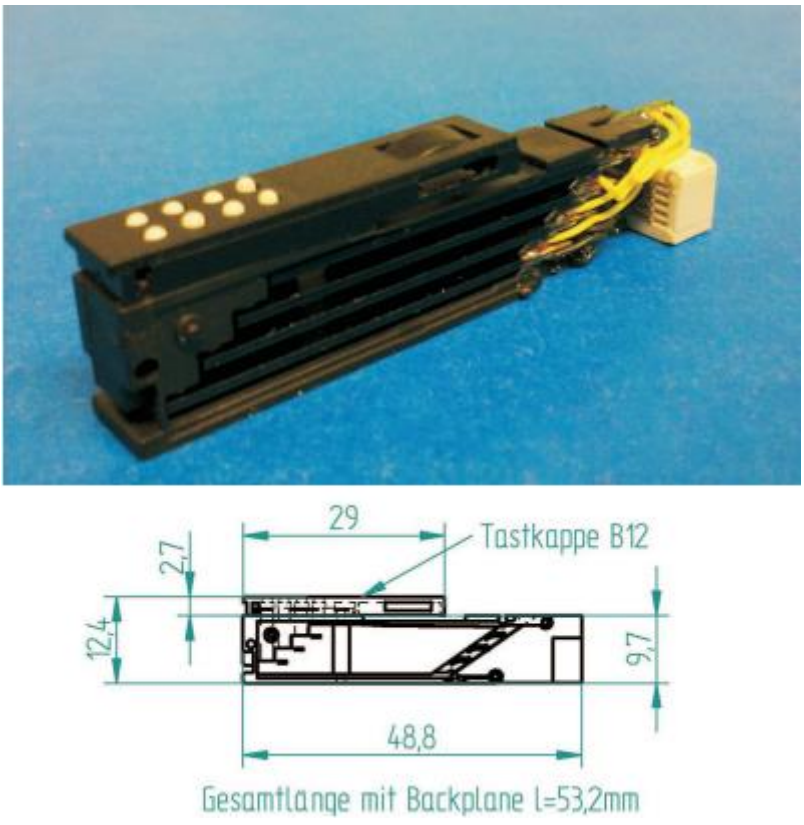


Fig. 1 METEC Modul P20 [4]

This design features a Braille-cell with 8 dots, driven by piezo-actuators (bending type). Active Back panel for 6 or 8 cells each, stackable. Flat or concave tactile surface without or with one interaction button.[4]

This type of Braille cell utilizes bending type piezo actuators, which work on the principle of piezoelectricity, or rather the reverse piezoelectric effect. The piezoelectric effect is the electrical charge that is generated by some crystalline materials, during the process of applying mechanical stress to them. These materials also exhibit the reverse of this effect, meaning that by applying an electric field to them they will show signs of deformation, even if not severe.
The piezoelectric effect results from the linear electromechanical interaction between the mechanical and electrical states in crystalline materials with no inversion symmetry.[5]
This bending type piezoelectric actuator relies on this principle as well.

These type of Braille cells can be found all over the world, as this is the most common solution in terms of Braille cells, or as the standard components for Braille reading boards, however this mechanism has more drawbacks than advantages, with the most obvious, noticeable at first sight, being its size.

Advantages	Disadvantages
• Narrow design, due to the bending actuators, placed beneath each other. • Driver circuit integrated into the cell's back panel.	• Huge thickness and length dimensions, due to the nature and placement of the piezo actuators. • More expensive compared to other solutions. • Lack of possibility to create a compact version that can be placed in not just rows but columns. • Requires constant electric charge for the piezoelectric actuators to stay bent upright.

The main problem of the device is caused by the working principle piezoelectric actuators. For the actuator plates to exhibit the sufficient amount of deformation they need to be long, as the longer they are the more intense the piezoelectric effect is. For example, the Modul P20[4] is 53,2mm long, along with its backplane, according to its documentation.[4] Therefore, there

is no method to make these cells compact, so we cannot use them to form multiple rows of braille characters.

The power requirement for the dots is:

200V +- 5%: absolute max. rating: 215V

1,2 µA typ. per dot

The power requirement for the back panel is:

3,3 – 5V +- 5%: 25µA typ. per cell (with static driven signals, no key pressed)

200V +- 5%: absolute max. rating 215V

5µA typ. per dot

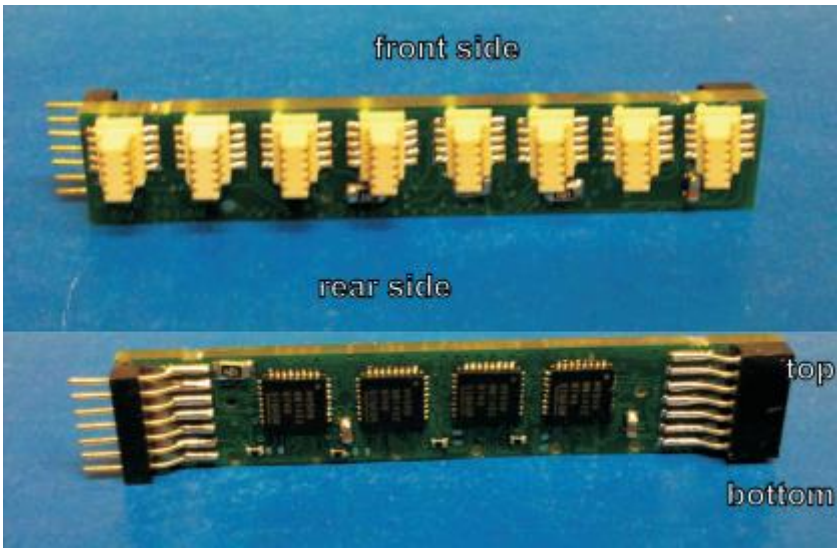


Fig. 2: Modul P20 Backpanel[4]

Electromechanical Refreshable Braille Module [6]

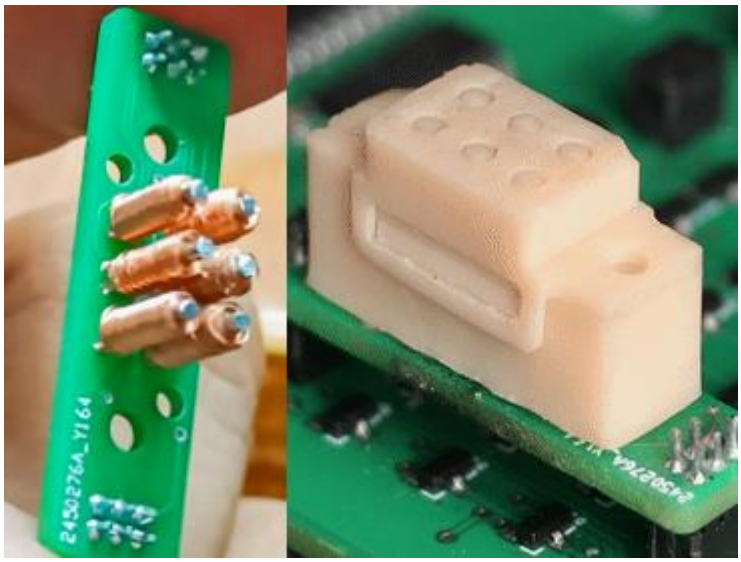


Fig. 3: The inside, and the casing of the module [6]

This is a Braille cell with 6 dots. The mechanism of the module is actuated by an electromagnet, which raises the dots and lowers them. This specific design uses a cylindrical axis, on which a round cylindrical-like shaped cam is found. This cam is hollow, and on the inside a small magnet can be found. This part rotates around along with the axle, if the electromagnet (coil) is activated, and lifts the Braille dot up or down.

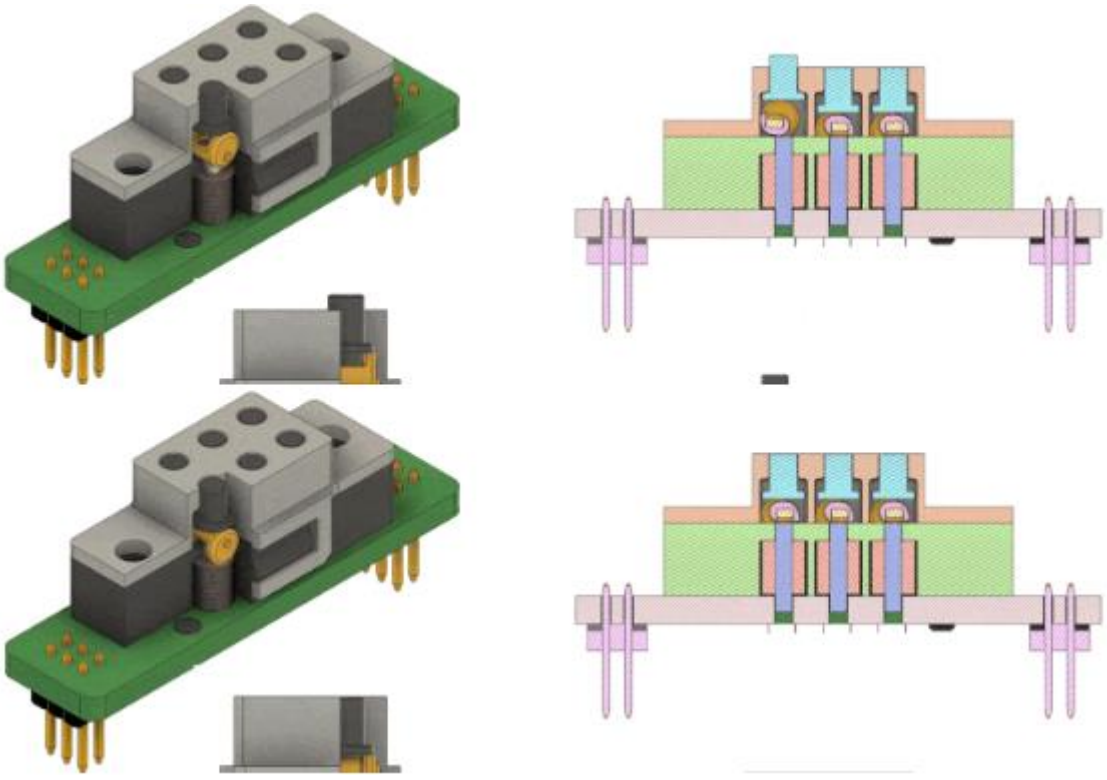


Fig. 4: The lifting mechanism of the Braille dots: lifted state(top pictures) and lowered state(bottom pictures) [6]

The electromagnet is made up of a 50-micron enamel-insulated copper wire wound around a ferrite core.[6]

The braille cells made of the ferrite core were found to be:

- Running cooler & with lesser power requirement [6]
- Having faster maximum refresh rate, from 200ms to 50ms [6]
- Overcoming the issue when the cam magnet is too close to the core, the natural attraction overcomes the opposing magnetic flux, and the cam refuses to flip. the natural magnetic attraction to a ferrite core vs the steel core felt like it was less than half of it.[6]

The module is designed in a way that the Braille dot latches, after toggling the coil with either polarity in the current. This happens because the coil is wrapped around a Ferrite core, which gets magnetized due to the electromagnetic field the coil generates, and even after the current is cut off, the ferrite core keeps its magnetic properties because it is ferrimagnetic. Ferrimagnetic substances are attracted by magnets and can be magnetized to make permanent magnets. [7]

After the Braille dot is lifted to its upper position, and the cam turns into its stable position, it cannot be pushed back by the touch of a finger (so by reading the dots), which means it not only works, but also reduces power consumption, since no power is used after the dots' position is set.

Advantages	Disadvantages
• Small, compact size and lighter build, meaning it can also be used for boards that have more lines of characters. • Less power consumption since the mechanism magnetically latches. • Generally cheaper technology	• Possibility of crosstalk between neighboring pins if magnetic field is too intense → might need magnetic shielding

The only apparent downside to this design is the possibility of crosstalk interference between adjacent magnetic fields, also between different Braille cells. If the magnetic field is too

intense it might affect the neighboring ferrimagnetic bodies, or even the small magnets. If this happens the braille display might become distorted, thus unreadable, which is of course not desirable. What is the solution for this problem? Using a thin wall, pipe shaped magnetic shielding around each coil, for example low carbon steel. Low carbon steel has less magnetic insulation, than other magnetic insulators, but in this case it would be more than enough, for magnets this size.

4.2 Selecting, and designing the Braille module

Due to the advantages that an electromechanical actuator offers, I am going to use one for the task that relies on the same principle, so an electromagnetically actuated braille module.

One braille character board is going to consist of six electromechanical pins on a board, with magnetic shielding around each of them. The small board itself will not be equipped with the driver circuit needed to drive the actuators, rather only the two polarities of each coil as outputs. Thus, one character board will have 12 pins, 6 on both ends of the board similarly to the example character cell described above. The reason behind this is that if I were to design the character cell in a way that it would have the driver circuit on board, that would mean 6 H-bridges per character cell, and it would possibly make the character cell board significantly bulkier, as I would have to resort to designing a cell that has more PCBs stacked on top of each other. And because I want to make this cell as compact as possible, I am going with this design.

As far as the mechanics of the cell goes, I want a design that keeps the inner magnet in place after the coil has been activated and deactivated. In order to achieve this, I need the coil to magnetize a body of ferrimagnetic material (Ferrite), which will help the magnet inside the mechanism latched to either end, even while pressing the Braille dot down while reading it. The coil will be wound around this material so that the magnetization process is optimal.

I have thought of two ways to do this, either have the ferrimagnetic body sort of wrap around the actuator, so that either end of the inner magnet can get as close to the electromagnet as possible or use two ferrimagnetic bodies at each end of the mechanism that have similar magnetic properties to the ferrimagnetic material.

‘C’ shaped ferrimagnetic material case:

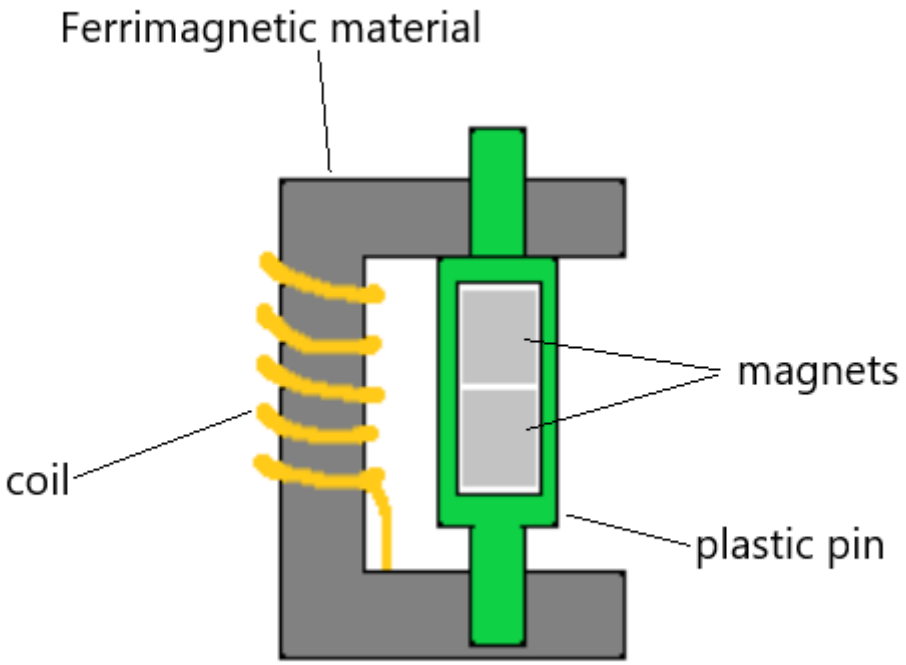


Fig. 5: The ‘C’ shaped Ferrite:
As the coil magnetizes the material, each of its ends becomes one of the two poles of the magnet, which then pulls and pushes the inner magnet attached to the Braile pin.

Two ferrimagnetic cores case:

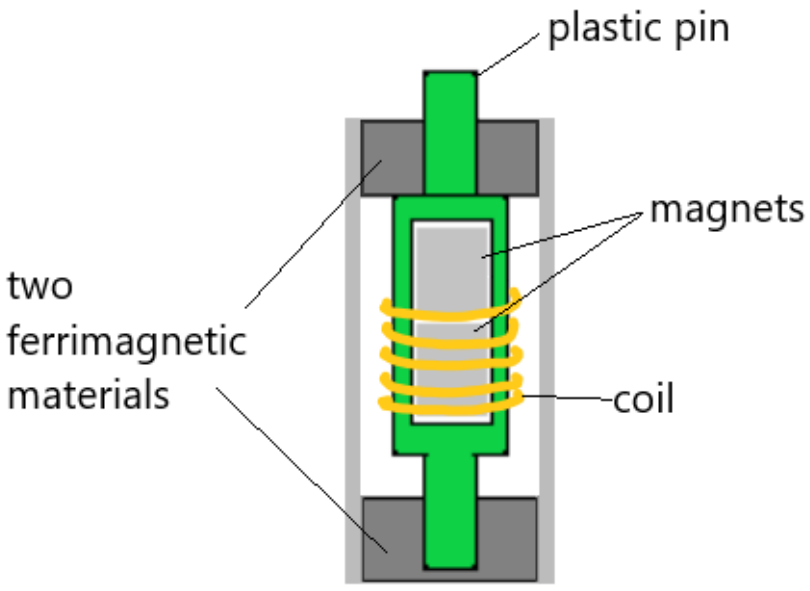


Fig. 6: Two magnets case: The coil magnetizes the two ferrimagnetic materials located at each end of the mechanism, which then pulls to, and keeps the inside magnet at their position.

I settled with using the ‘C’ shaped design, as it only needs one body of Ferrimagnetic material.

Components (1 Braille character):

- Coil: simple copper(Cu) wire (x6)
- Magnet: 1.5mm diameter, 2mm high neodymium disks (x12)
- Pin / magnet casing: 3D printed ABS (x6)
- Ferrimagnetic material: C shaped Ferrite core (x6)
- Cylindrical shaped cover: 3D printed ABS (x6)
- Half pipe shaped magnetic shielding: Low-Carbon Steel (x6)
- Board: Printed Circuit Board with 12 pins, 6 on both ends (x1)

I have designed the entirety of the Braille cell module in SolidWorks to provide a visual preview of the assembly and its operation. Here are the screenshots of the 3D model, and their descriptions:

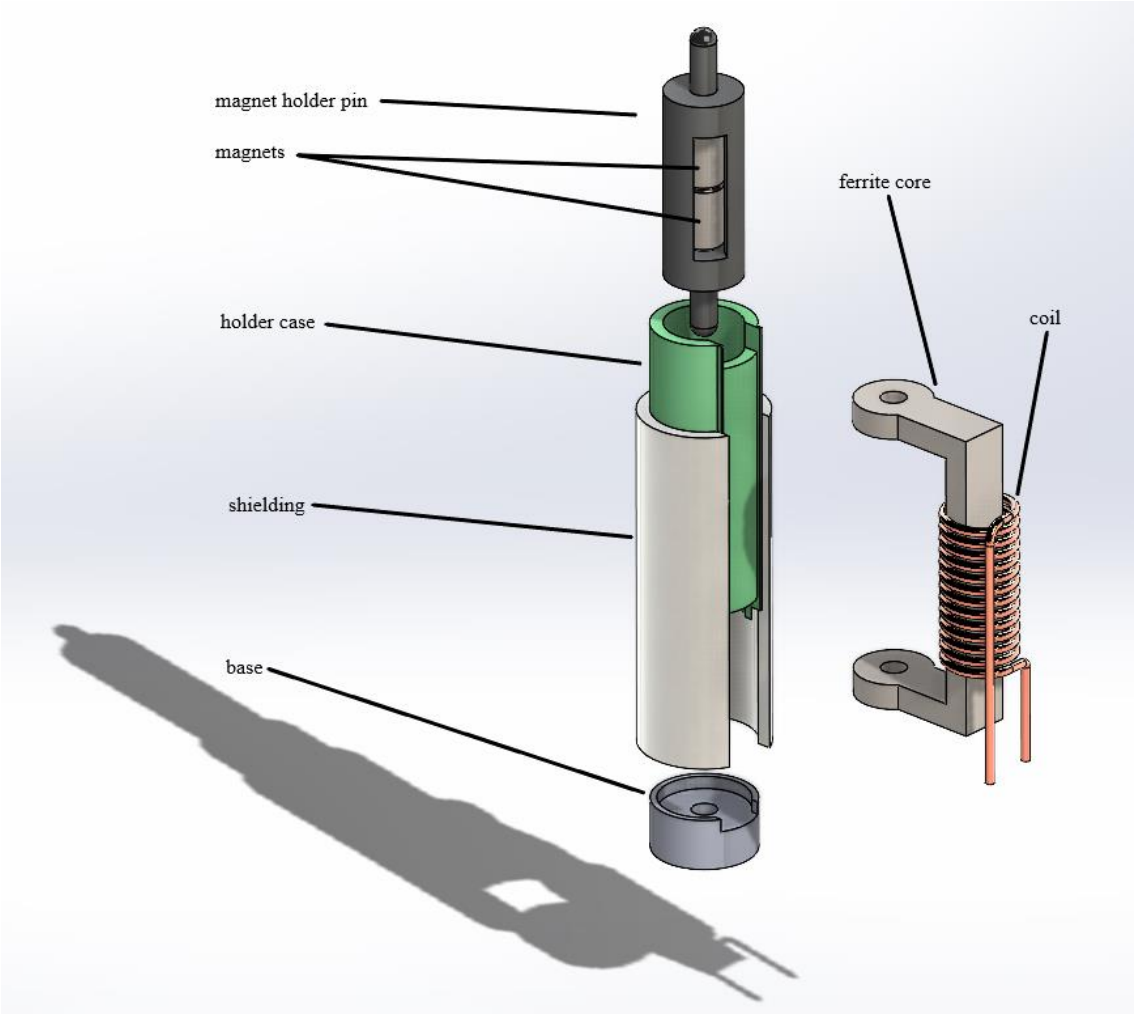


Fig. 7: The parts and assembly of one Braille pin

Here we can see the parts that one pin is made up of (Fig.7.). I'm going to go through the components from top to bottom. On the top we can see a 3D printed hollow body that holds the magnets and serves as the pin which will provide haptic feedback to the user, it can hold barrel shaped magnets stacked on top of each other of different possible heights as long as they fit inside. This pin fits inside the cylindrically shaped 3D printed holder case (green on the figure), which holds and restrains the pin from moving sideways. On the right we can notice the electromagnet, the ferrite core and the copper wire coil wrapped around it. The core fits into the two ends of the green holder case, this way the electromagnet can get close to the inside magnets to provide magnetic latching ability. Around the green holder we can find the magnetic shielding made of Low-Carbon Steel. The shielding's purpose here is to protect the magnets inside of the 3D printed pin from interference from neighboring ferrimagnetic cores' magnetic fields, hence it is shielding only the magnets, not the ferrite core. This magnetic shielding is held in place thanks two the two rail like lips that are machined onto the side of

the green holder case. On the bottom we can see the base in which the green ABS body along with the ferrite core fits into and blocks the rotation of the whole mechanism. This base is fastened onto the PCB, while the two polarities of the coil is soldered onto the circuit board.

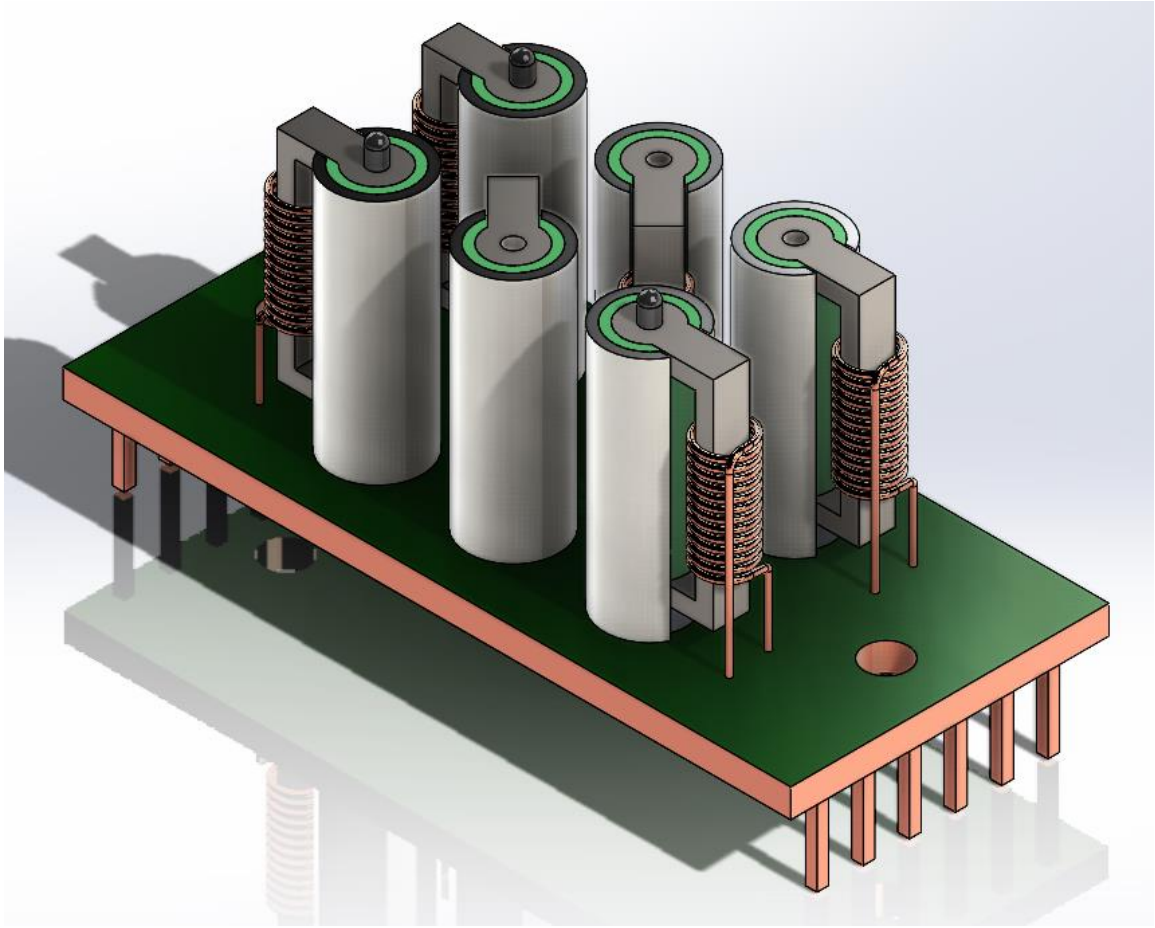


Fig. 8: The entire Braille cell without its cover.

On this picture (Fig.8.) we can see the entirety of the Braille character cell. The layout of the character cell is made so that the magnetic interference is kept to the minimum, not only inside the board, but also if we were to put extra boards next to it. On this 3D model, the printed circuit board is just a clean copper board with no circuitry, and no vias, etc. The 12 pins on the bottom will be connected to each coil's two polarities. Bear in mind that the pins are placed in a way so that they can fit into a stock breadboard or female sockets (2.54mm away from each other), however the length of the board may have to be adjusted.

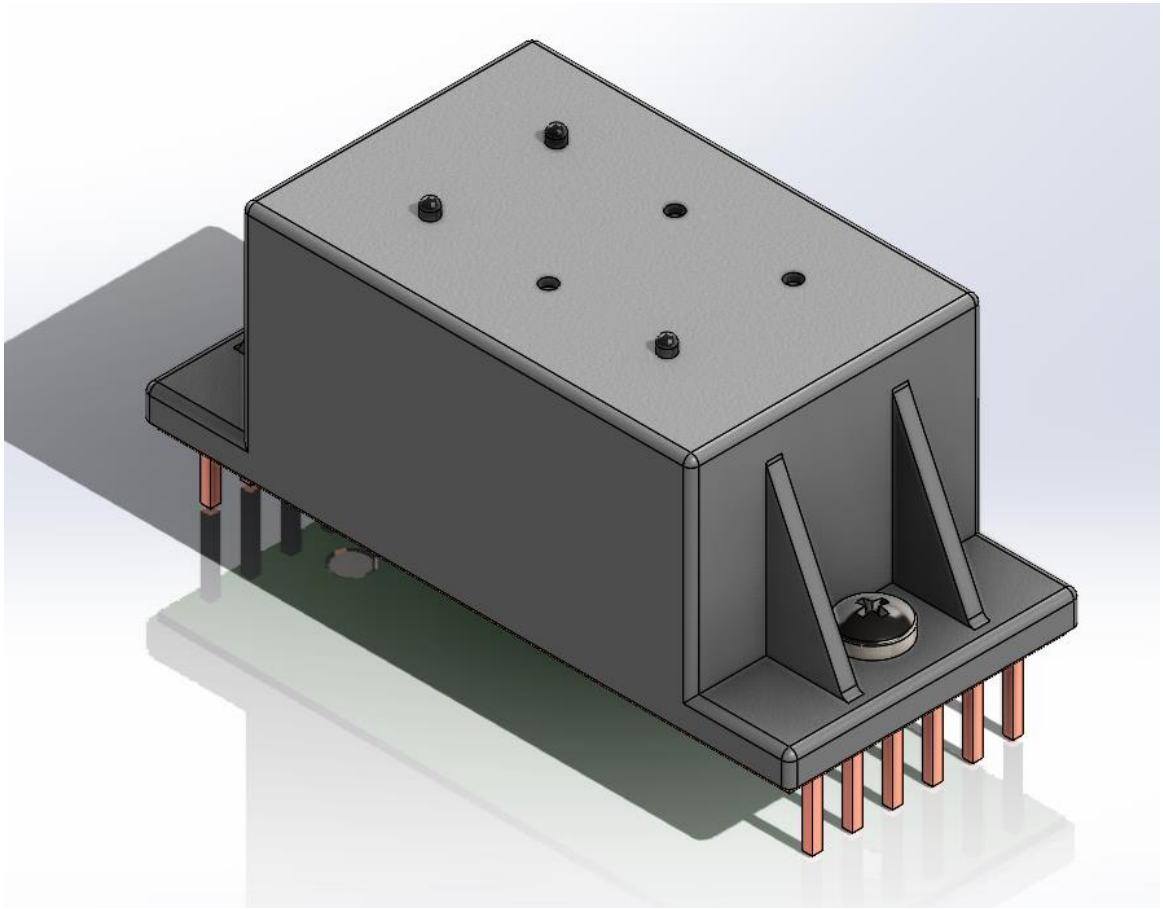


Fig. 9: The entire Braille cell

Here on Figure 9. We can see the entire design of the Braille cell, complete with its cover. The cover is made of 3D printed ABS, just as the other parts. The design is minimal, features 4 ribs for stability and is attached to the circuit board with two M2 cross-head screws.

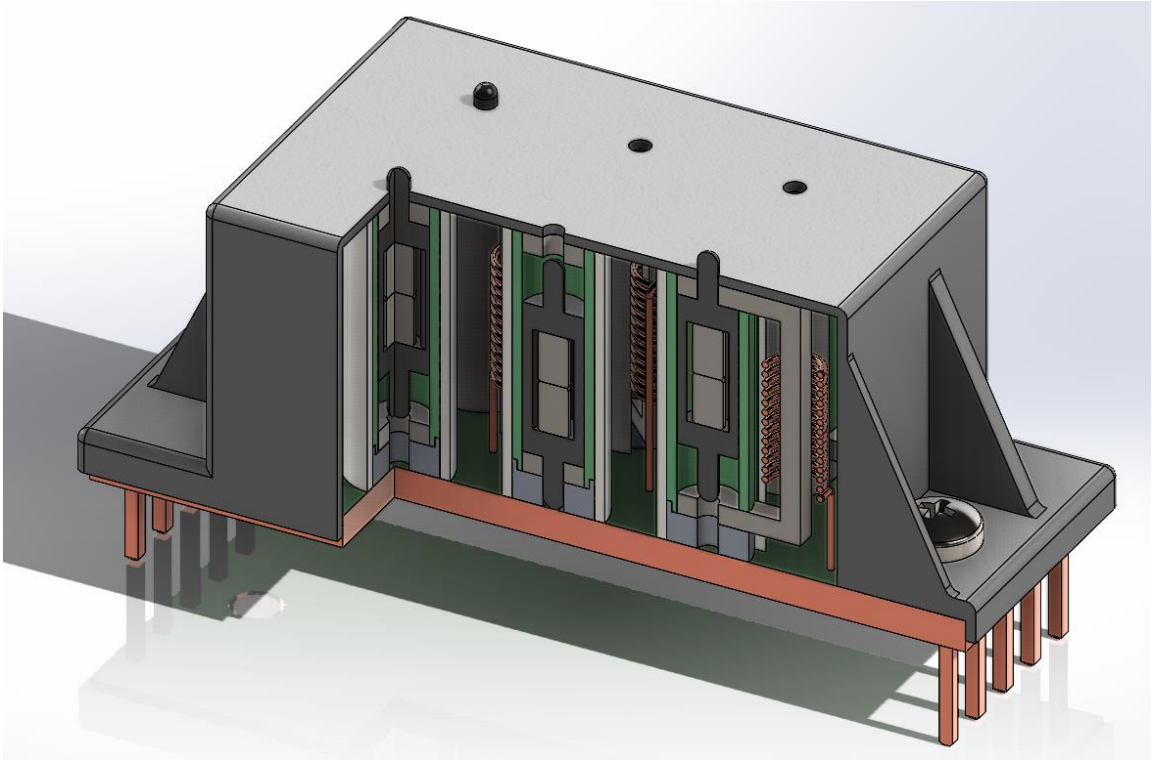


Fig. 10: The Braille cell with a cross section.

On this Figure (Fig.10.) we can see the cross section of the Braille cell board. Thanks to this view we can clearly see the moving pins on the inside in both up and down positions.

This is a compact design, meaning that it will not feature the driver circuit inside the cell board, if it were designed in that way the cell would be much bulkier, also this way anyone that might ever use this design could use any type of driver circuit they like.

Let's calculate the magnetic flux and the magnetic field strength of the coil.
At the center of the coil the magnetic field strength is: $B = (\mu 0/2)(NI/r)$

$\mu 0 = 4\pi \times 10^{-7}$
N is the number of revolutions in the coil: N = 13
I is the current: for example with a 10Ohm load it is about 0.5A
r is the radius of the loop: r = 1.325mm = 0.001325m

So the magnetic field strength in the middle of the coil is:

$B = (4\pi \times 10^{-7}/2) * (13 * \frac{0.5}{0.001325}) = 3.08 * 10^{3}$ T (Tesla) or 3.08mT

Ferrites used in transformers and other electrical appliances, are called soft ferrites, because they have low coercivity, which means they are easy to magnetize and demagnetize.

The sizes of the Braille character module can be seen on the following sheet:

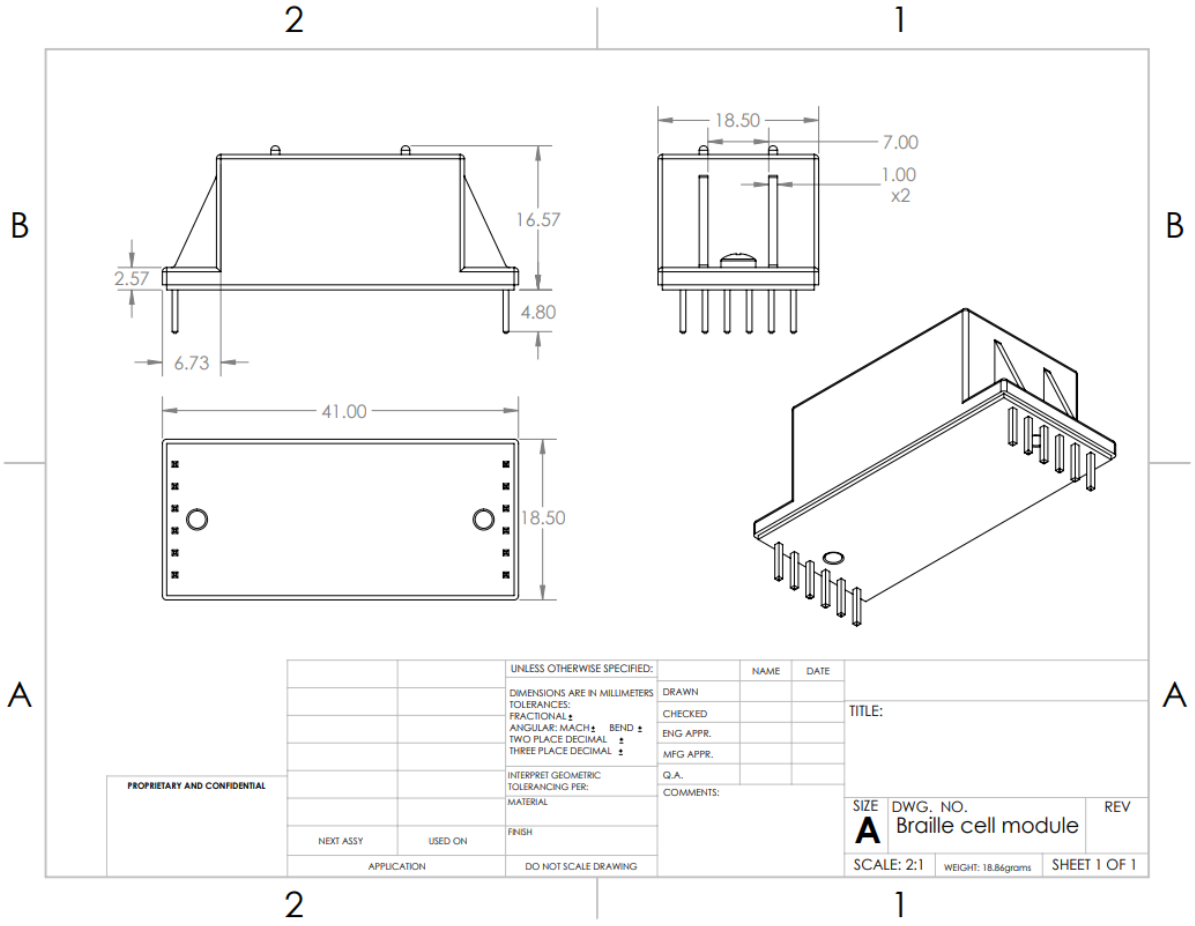


Fig. 11: The dimensions of the braille cell

4.3 Designing the circuit board of the Braille module

I have designed the following circuit board for the Braille module in EAGLE.

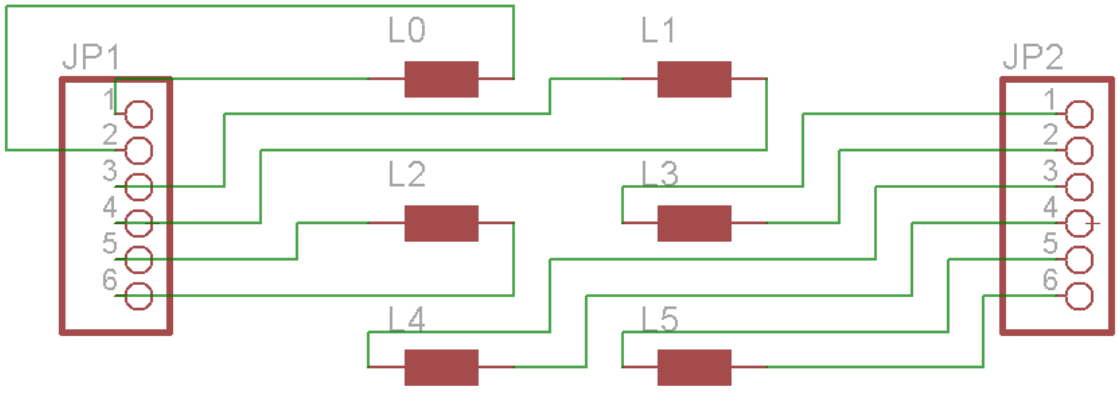


Fig. 12: The wiring diagram of the design.

On Fig.12. we can see the circuit of the Braille module's board, the layout how the pins are connected is what's important here. I have connected the first three (L0, L1, L2) coils to legs JP1 1-6, and coils L3, L4 and L5 to legs JP2 1-6.

On Fig. 13. below we can see how the circuit board's layout would look, however in this case the inductors take up much more space than they would in practice, still size of the design is similar to the one that I modelled in SolidWorks. The bottom side connections are the blue ones, although a little faint color, they are still noticeable.

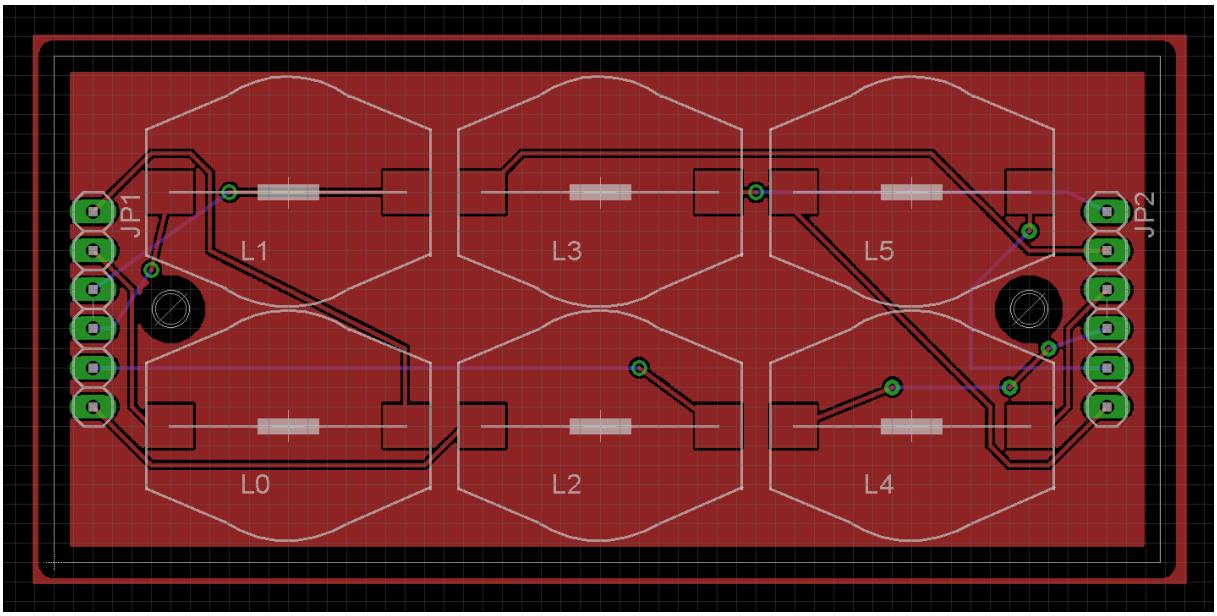


Fig. 13: The layout of the Braille module's printed circuit board

5. Planning the Driver circuit

After testing several methods and thinking the operation of the circuit through, I have settled on using an H-bridge per Braille pin to drive the circuit. Each H-Bridge will have an output pin connected to it, that sends the Braille pin upwards, and all of the H-Bridges will be connected to one common output pin which will clear the Braille display, by applying current to the negative polarity of the coils. This way an 8-character display can be easily controlled with a microcontroller which has enough output pins. The H-Bridges will be put on a control

PCB, but they can also be visualized on a breadboard. I will need 6 H-bridges per Braille module. The driver circuit will also need a controller. For that purpose, I chose the Arduino UNO R3, mainly because this microcontroller is available in Autodesk Tinkercad for simulation.

The Arduino UNO R3 is a microcontroller based on the **ATmega328P (16MHz)**. It has 14 digital input/output pins (of which 6 can be used for PWM outputs), 6 analog inputs, a USB connection, a power jack, an ICSP header and a reset button. The microchip is also replaceable as it is not soldered onto the Arduino board.[8]

5.1 Designing the driver circuit

For the next step I am going to be designing the Driver circuit that controls the Braille modules. For this simulation I am going to be making a circuit board that has four Braille module slots. Each module is driven by 6 H-bridges, which are soldered onto the driver circuit board. Each H-bridge is made up of a pair of N-channel and a pair of P-channel MOSFETS, with flyback diodes loop for the purpose of protection from Voltage spikes. These voltage spikes can be caused by the collapsing electromagnetic field from the coil, when the load is switched off, however most MOSFETS have built in flyback diodes. In this simulation, instead of building each H-bridge separately, I will be using integrated H-bridge driver ICs for the purpose of simplifying the circuit and reducing the size of it. This circuit board will have 11 input pins plus a GND pin, and on the output will be the 4 Braille modules, with 12 input pins each. Although they have 12 pins, since the negative polarity of each coil through the H-bridges will be connected, only 6 pins will be used actively for control purposes on each Braille module, the other 6 will be only used for resetting the Braille pins. The 11 input pins of the driver circuit will be controlled by the Arduino's digital Input/Output pins, 6 of which will be responsible for the movement of the six dots in each Braille cell. 4 Digital I/O pins will be controlling which Braille cell is currently active to be set by the program code, and the last pin will be responsible for setting all the dots in all cells to the lower position, effectively resetting the braille cells, to be ready to display the next four characters in the string, which is inputted into the program code, or to reset the whole program entirely.

In the braille alphabet, in some cases there can be special characters. (excluding the abbreviations and special words such as "& or and") These special characters mostly come to play when a change in textual environment is evident, such as when a change from letters to numbers* or vice-versa happens. *For example, in this case before the numbers come, a

special character is inserted into the text, which means: “numbers incoming” and then the numbers follow. This is needed because of the limitation in the 6 dots’ combinations, as there aren’t enough combinations to fit all the letters and numbers into the combinations without mixing them together, meaning that for example two dots below each other in the top left corner stands for the letter “b”, or the number “2”, lets suppose that these two dots would be in another position rather than the upper left, still the visually impaired wouldn’t or at least would hardly be able to tell the difference as they only get haptic feedback through their fingertips and the only thing they feel is two dots atop each other, in contrary to people with vision, who can clearly tell the difference just by looking. This is why in braille the numbers from 1 to 9,0 are denoted by the same dot combinations as a-j (“a” is the same as “1”, while “j” is the same as “0”). There is also a special character that designates that letters are coming next, as well as a character the denotes capital letters.(which is not always necessary or useful but can be, for example in case of alphanumeric codes with uppercase letters)
It is also possible for two of these special characters to be inserted into a text, for example when after a number(without a space) a capital letter follows, in this case there will be a “letters incoming” sign and a “capital letter” sign after each other. I’m describing this in order to clear up any misconceptions about the braille alphabet. This also means that under the condition of using four braille cell modules on the driver board, even after such an extreme case, at minimum two characters will be displayed in addition to the two special characters. This is why I decided on using 4 braille modules. Using only one module could also be a viable option for reading, but in this case an automatic clock cycle would be preferable, with a “start/stop” button instead of a “switch to next string” button, and it would have to be finetuned by the user themselves to make reading one character at a time comfortable.

The “next” and the “reset” buttons will be placed separately from the driver circuit board, and each will be connected to one Input pin on the Arduino.

Here I have created a block diagram for the entire Project:

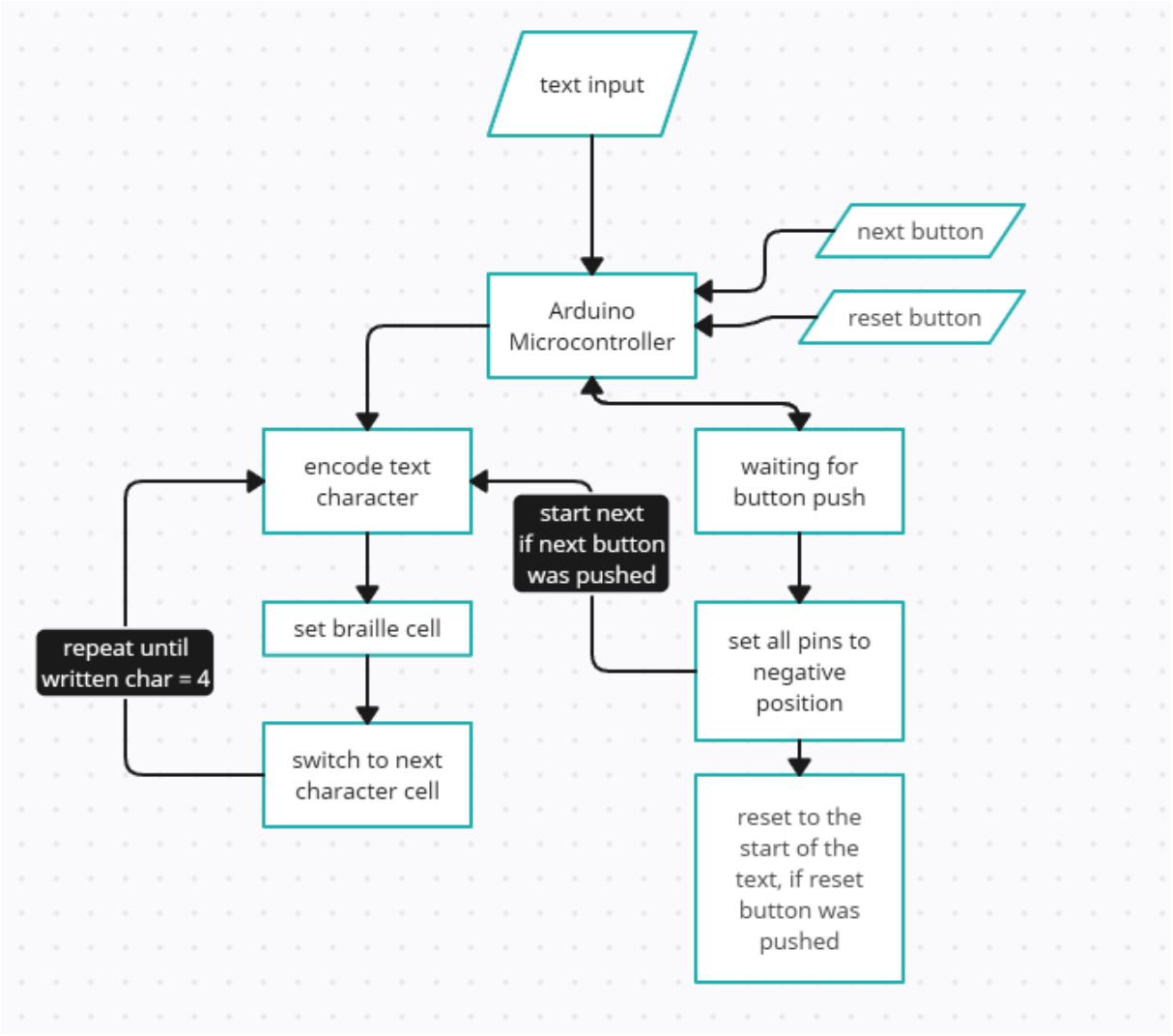


Fig. 12: Block diagram of the project[17]

The block diagram above describes the working process of the refreshable braille reading board, which is the following:

At first a string of text is read by the program running on the Arduino, after which the encoder function starts encoding each letter to what each letter stands for in terms of which pins have to be in up and down positions. After that the program sends out the first information to match the first braille cell, the first few characters of the encoded string (in case of a needed special character it writes the special character then moves on as if it wrote a letter) which gets "written" on the first cell (set braille cell). After that the encoder function continues where it left off and writes the next letters in the encoded string, which gets written out to the next braille cell module (switch to next character cell). This loop continues until the number of written letters/symbols reaches 4, at which point the program stops until it receives an asynchronous impulse from the next button or the reset button.

If the next button is pressed then the program clears the reading table, and starts from “encode text character” again until the written number of letters/symbols reaches 4 again. If the reset button is pushed, then after the reading table is cleared, the program hops back to the start of the string.

5.2 Designing the circuit board of the driver circuit

This step involves the design of the driver circuit board. This circuit which, is controlled by the Arduino Uno R3 microcontroller, has 6 H-bridges per braille cell slot, input sockets for the Arduino, and output sockets for the braille cell modules.

The circuit features the following components:

- 1x11 single row horizontal female standard pin socket for output signals from microcontroller (x1)
- 1x1 single vertical female standard pin socket for GND connection (x1)
- 1x6 single row vertical female standard pin socket for output to braille cells (x8)
- H-bridge built with 2 pairs of transistors with built-in flyback diode for voltage protection (x24)

The operation of the circuit:

When a braille pin gets toggled on, its respective H-bridge opens one of its transistor pairs to let current flow through in one direction. For example, if pin 1 is turned on (upper left h-bridge) then each braille cell’s upper left h-bridge opens in one direction. However, that would cause all the braille cells to have the same information written on them, which is why there is 4 control signals, one for each braille cell, that controls if the cell is active or not. This means that even if all upper left h-bridges are on, only on one load will current flow through, which is the one cell whose control signal is toggled on, since the control signal acts as the power supply for the H-bridges. If only one of the four control signals is turned on at a time, then we can control which braille cell we want to write on, and even after we switch to the next one that we want to write, the previous one will stay latched thanks to the magnetized ferrite. The last pin is for opening all the H-bridges in the other direction, so that the coil connected to each will reverse its magnetic field thus toggling every single magnetic pin to the lower position. Of course, all 4 of the control signals are toggled on in this case, to ensure

that the entire board can be refreshed. The pin layout for the 1x11 socket from right to left is the following (or on the board view from top to bottom):

- 1st braille pin socket
- 2nd braille pin socket
- 3rd braille pin socket
- 4th braille pin socket
- 5th braille pin socket
- 6th braille pin socket
- Control signal 1's socket
- Control signal 2's socket
- Control signal 3's socket
- Control signal 4's socket
- Refresh socket

Fig. 13: input sockets of driver circuit board

The 1x6 output sockets are wired in the same layout as on the braille cell module board, so that the pins can fit into the sockets. The pin layout for the 1x6 sockets from top to bottom on the schematic (or from right to left on the board view) are the following:

The upper sockets:

- 1st H-bridge socket 1
- 1st H-bridge socket 2
- 2nd H-bridge socket1
- 2nd H-bridge socket2
- 3rd H-bridge socket1
- 3rd H-bridge socket2

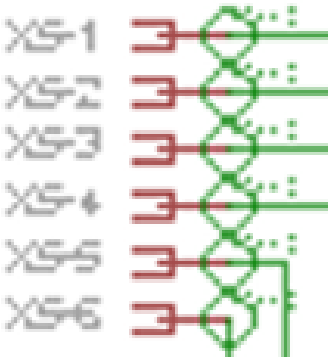


Fig. 14: upper sockets for braille cell module

The lower sockets:

- 4th H-bridge socket1
- 4th H-bridge socket2
- 5th H-bridge socket1
- 5th H-bridge socket2
- 6th H-bridge socket1
- 6th H-bridge socket2

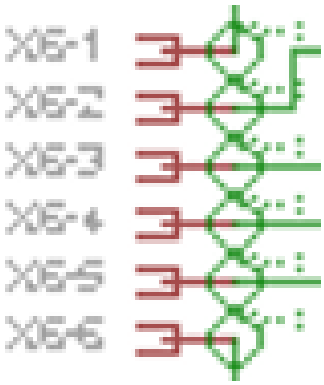


Fig. 15: lower sockets for braille cell module

The following figure is the wiring diagram schematic of the designed printed circuit board.

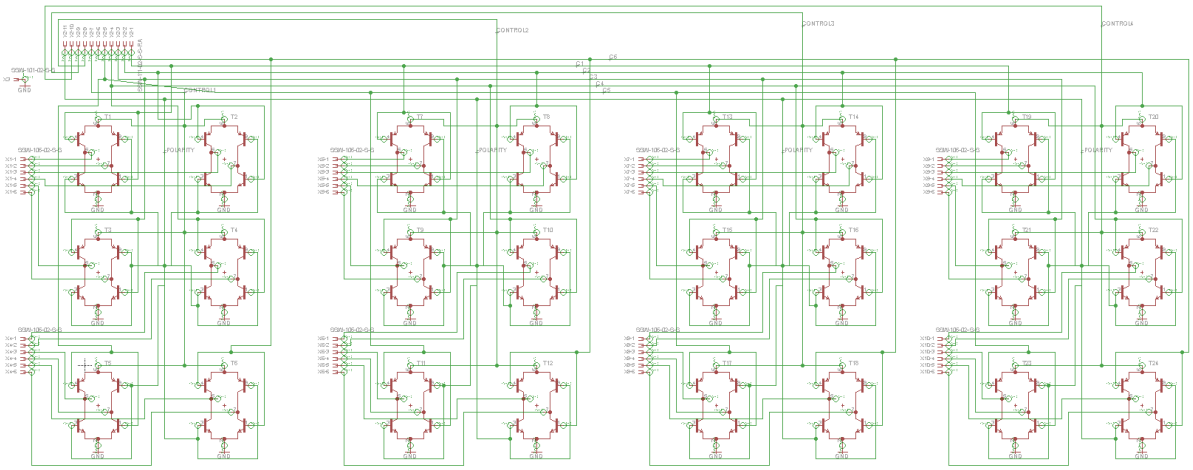


Fig. 16: Wiring diagram of driver circuit

Here we can see the entire wiring diagram of the circuit. Each group of 6 H-bridges are connected to the circuit in a similar way to help review. Also I didn't use a bus line to ensure visibility of where each wire goes. This particular circuit was made for the purpose of driving 4 braille cell modules, more precisely 4 times 6 coils which serve as the load. Through the usage of the microcontroller, specific H-bridges get activated, and by setting which module is activated through the control signal, this circuit alternates between which braille cells are getting written.

On the next two figures (Fig. 17, Fig. 18) we can observe the top and bottom view of the designed driver circuit board. On the top and bottom of the figures, we can see the 1x6 sockets in which the braille cell modules fit into. On the left side are the input sockets, that are looking outwards, and at the bottom left corner is the GND socket for connecting the ground

of the circuit and the microcontroller. The board is based on a double-sided PCB, with both sides being used for connection of the components. The connection between two sides for a wire is made possible with through vias. Both sides of the board have a layer applied to them. The following layout of the components is beneficial to the design because it is arranged in a way to save space by placing all the components underneath each braille cell module's supposed slot. This arrangement is convenient because this way, the size of the driver circuit board is significantly reduced. The size of this board is about 4.5cm x 10cm. Furthermore, for convenience and clarity in design, each h-bridge is placed below their respective actuator.

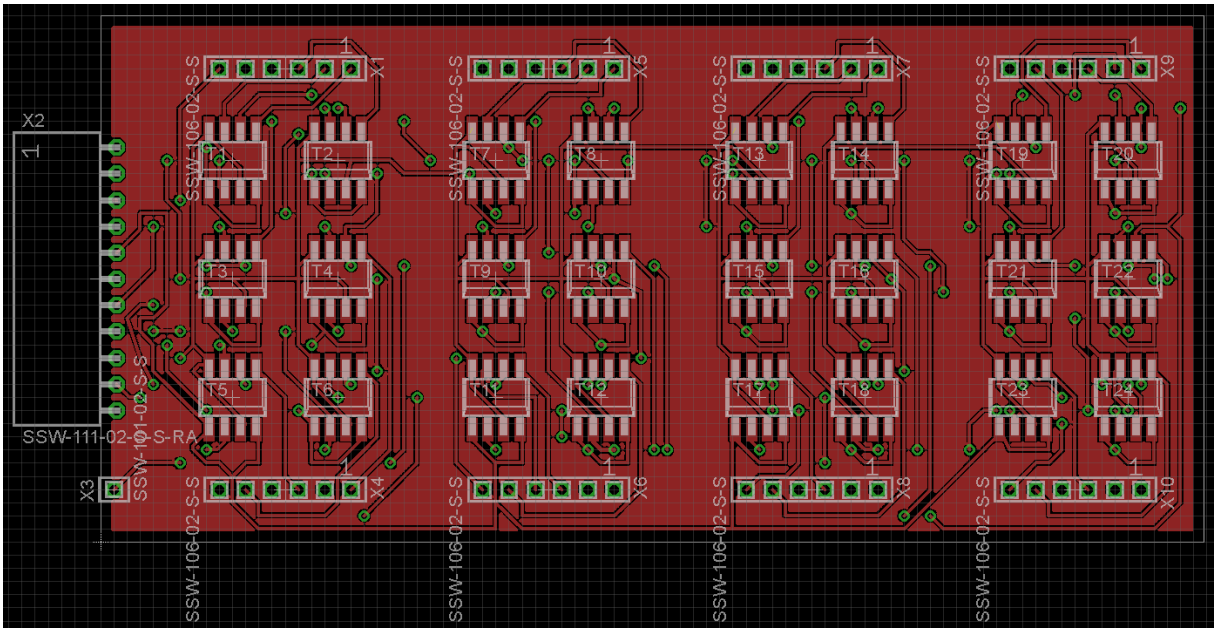


Fig. 17: Top view of the driver circuit board

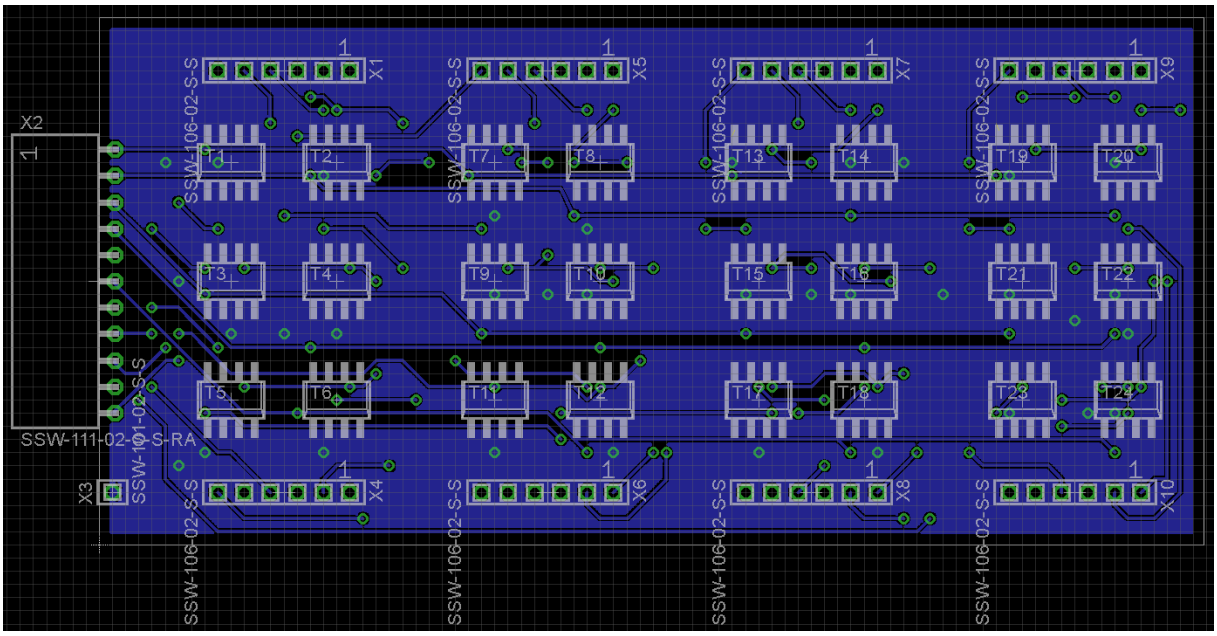
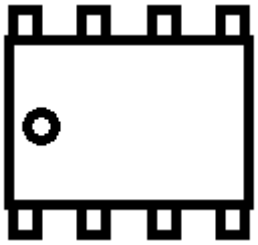


Fig. 18: Bottom view of the driver circuit board

The bottom (blue) side is the GND plane, each ground pin of each h-bridge IC is connected to the ground plane with a through via. All of which is connected to the GND pin of the circuit board. Managing the GND pins with a copper plane helps with organizing the wires, as less wires are needed in the circuit, this way the wiring becomes less cramped, also it makes connecting all of the ground pins on each IC less of a hassle, as this copper ground layer connects almost everything and I only need to connect the isolated ground pins on each IC to the common ground layer.

This is the pinout for each H-bridge IC as can be seen on the circuit board:

Upper pins from left to right: **open 2nd pair of transistors (reset) | right load pin | Vcc | left load pin**



Lower pins from left to right:
open 1st pair of transistors (+) | GND | open 2nd pair of transistors (reset) | open 1st pair of transistors (+)

Fig. 19: H-bridge integrated circuit pinout

6. The Program Code and simulation

6.1 Program code development

The program code was written in C++ programming language, in Autodesk Tinkercad's Arduino environment. The first instance of the code was written to control LED's for simulation purposes instead of actuators. I know that the fact that I used LEDs to simulate a project that is about a reading table for the visually impaired might seem a little questionable, but the reality is that this was the easiest way to optimally test such a program, considering the limitations of Tinkercad, and its limited supply of components. This simulation included the development and testing of the braille encoder function, since I wanted to see that it transforms each character correctly, and if the program is able to display a braille character on a 3x2 matrix of LEDs. Next, I tested displaying multiple characters with delays between each other on the same one matrix, and upgraded the encoder function to include the special braille characters which indicate different changes in the textual environment. These processes were tested on the LED equivalent of one braille cell module. After deeming the first simulation successful, I have moved onto the next step.

The next step consisted of using multiple LED matrixes to develop a program code, which can switch the output to multiple matrixes, one-after-one. This simulation was also a success, and I didn't have to bother with making the LED's latching since the final product's actuators have magnetic latching capabilities. At this point I have realized that the special characters that indicate changes in the text could be an obstacle later on for the limited amount (4) of braille cells available. However shortly after, I thought that this can be avoided by using a second counter (the first counter being the one that keeps track of which character the program is at in the string) which counts the number of actual characters that were written out to the Braille cell modules. This seemed okay at first, but then I ran into more complications, which were caused by the textual environment in some instances creating cases where the special character would be written to the board, but due to the limited character length the actual character (letter or number) that was indicated by the special character was not written, and was lost in the process, with the program jumping to the next segment instead of the remaining character(s). This was caused by the way I wrote the program code. At first the encoder was a long function with the different cases that could happen in the text. So to say the function directly converted the raw input string into the 6 bit braille writing, the main problem of this approach, which I didn't think of initially, is that not just each character had to

be made into different cases, but through that also each case had to be made, where the program would need to use the different special characters. This caused one letter to sometimes in reality, be 3 characters written out to the board, which wouldn't have been a problem if I could carry the remaining characters over to the next segment that were cut off. At the end I have settled on redeveloping the program code, to first take the raw text, convert it into a text with special characters added to represent the special characters in braille, then encode the already converted text into information to the braille cells, with the encoder function. I have also dropped the idea about letting two special characters appear before a character, as that would be just too much information for the reader in comparison to actual useful information that is the text, also if there is an "uppercase character" we can already assume that the incoming character is a letter, and not a number as there isn't another meaning to that character

So for example, lets assume the following raw text is in the input string:

"HelloWorld123a Foo456X bBar"

The program will convert it to the following:

"$Hello$World#123_a $Foo#456$X b$Bar"

'$' means Uppercase letter incoming ('$' is used before every capital letter except when more than one occurs to be strung together)
'#' means number incoming ('#' is used before every string of numbers)
'_' means letter incoming ('_' is used if the letter is immediately after a number, else it is automatically assumed that after a space letters approach)

6.2 The simulation of the test circuit

Later I used this program code and added some extra additions to it, thereby transforming it into the final product's code.

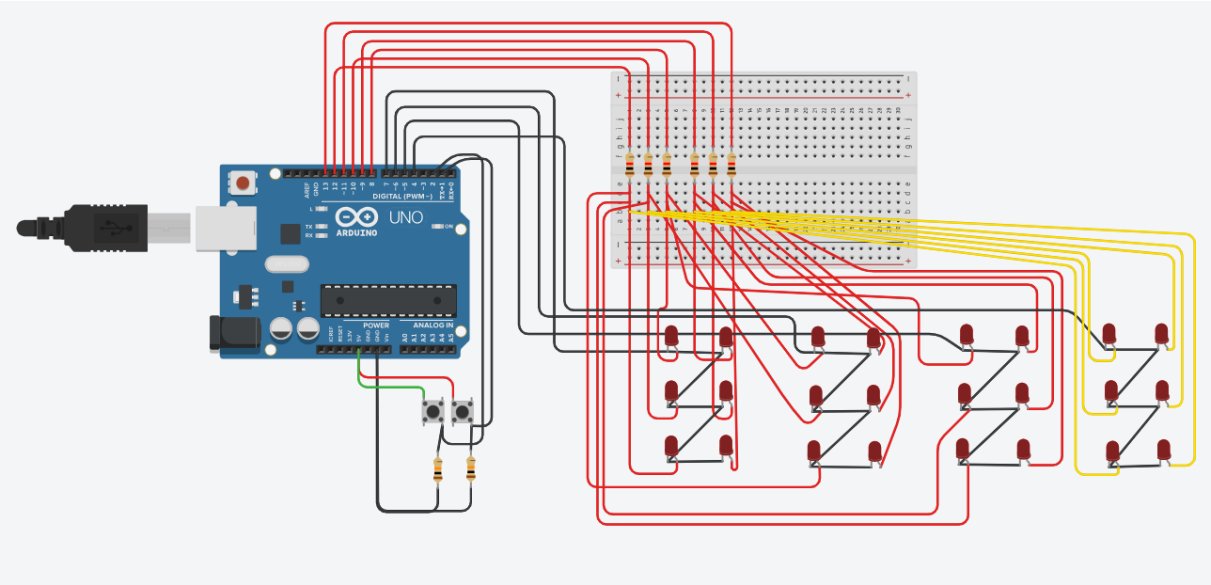


Fig. 20: The test circuit built in Tinkercad

This is the final phase of the test circuit that I built. Each LED in each of the 4 matrices is controlled by a digital output pin, so 6 digital output pins control the LEDs. Furthermore, each matrix is toggled on or off by 4 additional digital output pins, these are the control signals. Each matrix is controlled separately by the control signals. The two buttons can be found before the Arduino with the left one being the next button, and the right one being the reset button. The next button lets the writing function in the code pass to the next four characters, thus writing out the next part of the input string onto the braille cell modules. The reset button resets/clears the display by telling the program to jump back to the start of the string, and by making all the counters equal zero.

6.3 Program Code Breakdown

In this section I'm going to break down the program code that I have written for the final design. I'm only going to include only fractions of the important parts, and skip the rest, so that the code doesn't take up much space.

The first part is **declaring** the different variables such as declaring the pin names as integers with the pin number value which will later be assigned to be set to an actual digital pin, and not just a variable. Also this contains the setting of the raw string that is going to be read.:

```
int BRAILLE0 = 8;
int BRAILLE1 = 9;
int BRAILLE2 = 10;
int BRAILLE3 = 11;
```

```
int BRAILLE4 = 12;
int BRAILLE5 = 13;
int CONTROL1 = 7;
int CONTROL2 = 6;
int CONTROL3 = 5;
int CONTROL4 = 4;
int RESET = 3;
int SWITCH = 2;
int SW_RESET = 1;
int count = 0;
int written = 0;
char string1[] = "HelloWorld123aFoo456bBar";
```

Then we proceed with the **setup function**, to set the pins into their respective modes. (input or output):

```
void setup()    {
        pinMode(LED_BUILTIN, OUTPUT);
        pinMode(CONTROL1, OUTPUT);
        pinMode(CONTROL2, OUTPUT);
        pinMode(CONTROL3, OUTPUT);
        pinMode(CONTROL4, OUTPUT);
        pinMode(BRAILLE0, OUTPUT);
        pinMode(BRAILLE1, OUTPUT);
        pinMode(BRAILLE2, OUTPUT);
        pinMode(BRAILLE3, OUTPUT);
        pinMode(BRAILLE4, OUTPUT);
        pinMode(BRAILLE5, OUTPUT);
        pinMode(RESET, OUTPUT);
        pinMode(SWITCH, INPUT);
        pinMode(SW_RESET, INPUT);
        //Serial.begin(9600);
}
```

Next I declare the **stepper function**. The stepper function takes care of switching the correct braille cell module's power on or off, to ensure that the information in the converted string is displayed correctly.:

```
void stepper() {
	if (count == 0){
		digitalWrite(CONTROL1, LOW);
		digitalWrite(CONTROL2, HIGH);
		digitalWrite(CONTROL3, LOW);
		digitalWrite(CONTROL4, LOW);
		count++;
	}
	else if (count == 1){
		digitalWrite(CONTROL1, LOW);
		digitalWrite(CONTROL2, LOW);
		digitalWrite(CONTROL3, HIGH);
		digitalWrite(CONTROL4, LOW);
		count++;
	}
	else if (count == 2){
		digitalWrite(CONTROL1, LOW);
		digitalWrite(CONTROL2, LOW);
		digitalWrite(CONTROL3, LOW);
		digitalWrite(CONTROL4, HIGH);
		count++;
	}
	else if (count == 3){
		digitalWrite(CONTROL1, HIGH);
		digitalWrite(CONTROL2, LOW);
		digitalWrite(CONTROL3, LOW);
		digitalWrite(CONTROL4, LOW);
		count++;
	}
```

```
}
```

The program starts off with CONTROL1 set to high and the rest zero, this is set in the start of the main program loop, with the count variable being declared 0 above, after each iteration the stepper function steps into the next variation, thus setting the active braille cell module beforehand.

We continue with the **resetBoard** function. This function is getting called after the next or the reset button was pushed. It is responsible for refreshing/clearing the entire braille reading board, by setting all CONTROL pins to HIGH (this activates all braille cell modules) and outputting a HIGH on the RESET pin, which then applies reverse polarity current on each and every load. (reverse polarity is from the viewpoint of the coils, with the default being the instance when the pin is pushed outward):

```
void resetBoard() {
        digitalWrite(CONTROL1, HIGH);
        digitalWrite(CONTROL2, HIGH);
        digitalWrite(CONTROL3, HIGH);
        digitalWrite(CONTROL4, HIGH);
        delay(50);
        digitalWrite(RESET, HIGH);
        delay(50);
        digitalWrite(RESET, LOW);
        digitalWrite(CONTROL1, HIGH);
        digitalWrite(CONTROL2, LOW);
        digitalWrite(CONTROL3, LOW);
        digitalWrite(CONTROL4, LOW);
}
```

Now comes the longest part of the program, which manifests itself in the encoder function: **processLetter()**

This function is the heart of the program code, as this “translator” is the part which encodes each character into the HIGH/LOW states on each BRAILLE pin, from the converted string.

It utilizes the switch() function, and contains a case for each different character.
Here are some instances of the switch function, I'm not going to include the whole function, as it takes up about 250 lines.
This is the case for 'A' and 'a':

```
void processLetter(char c) {
	switch (c) {
		case 'A':
		case 'a':
			digitalWrite(BRAILLE0, 1);
			break;
	//other cases
}
```

These are the cases for special braille characters "uppercase letter incoming" ($ in converted string) and "numbers next"or # (# in the converted string as well):

```
void processLetter(char c) {
	switch (c) {
	//other cases
		case '$':
			digitalWrite(BRAILLE5, 1);
			break;
		case '#':
			digitalWrite(BRAILLE1, 1);
			digitalWrite(BRAILLE3, 1);
			digitalWrite(BRAILLE4, 1);
			digitalWrite(BRAILLE5, 1);
			break;
	//other cases
}
```

And for the last part we have the **main loop** part of the program, which utilizes the previous functions and executes the process. In the start of the loop is the for loop which generates the converted text from the raw input string, and stores it in the encodedString variable. This

segment of code declares the three variables needed for its operation, and then inside the for loop uses functions isupper(), isdigit() and islower() to check each character in the raw string, and the previous character before them. With the usage of if statements, it can decide if the character needs a special character before them or not. The rest of the string gets moved back one character each time a special character is inserted.

```
void loop()
{
char encodedString[100];
char prev = '\0';
int encodedIndex = 0;

for (int i = 0; string1[i] != '\0'; ++i) {
        if (isupper(string1[i])) {
                if (!isupper(prev)) {
                encodedString[encodedIndex++] = '$';
                }
        }
        else if (isdigit(string1[i])) {
                if (!isdigit(prev)) {
                encodedString[encodedIndex++] = '#';
                }
        }
        else if (islower(string1[i])) {
                if (isdigit(prev)) {
                encodedString[encodedIndex++] = '_';
                }
        }
        encodedString[encodedIndex++] = string1[i];
        prev = string1[i];
}
encodedString[encodedIndex] = '\0';
//Serial.println(encodedString);
```

```
//rest of the code
}
```

This is the rest of the main **loop**, also the rest of the whole program. It firstly sets the CONTROL pins into their default states, then stores the states of each push button in a variable and updates them with each loop. The next two if statements are made for the two buttons. What actually decides if the button resets or continues the string writing, is the "written" variable which keeps actual track of where the program is in the string and stores it, unlike the "i" index variable which only keeps track for the duration of one loop. If the second if is TRUE, so if the next button is pushed the program starts, and the first part of the string gets written out to the braille reading board. The for loop inside the if statement is responsible for this operation. First it sets 'i' 's size to be smaller than the length than the encoded string, then it makes the currentChar variable equal the i^{th} element of the encoded string. After that it calls the encoder function and runs the current character through it. The encoder function writes the data out onto the braille cell module, and then the code resumes by calling the stepper() function and increases the value of "written" by 1. An if statement makes sure that the loop quits when the count variable (which gets incremented inside the stepper function) reaches the value of 4, this means that the loop ran through 4 times and that the 4 braille cells are displaying characters now, until the board gets cleared with either of the two buttons.

```
void loop()
{
//rest of the code
        digitalWrite(CONTROL1, HIGH);
        digitalWrite(CONTROL2, LOW);
        digitalWrite(CONTROL3, LOW);
        digitalWrite(CONTROL4, LOW);
        int switchValue = digitalRead(SWITCH);
        int resetValue = digitalRead(SW_RESET);
        int i = 0;
        count = 0;
        if (resetValue == HIGH){
                written = 0;
                resetBoard();
```

```
        }
        if (switchValue == HIGH){
                i = written;
                resetBoard();
                for (i; i < strlen(encodedString); i++) {
                        char currentChar = encodedString[i];
                        processLetter(currentChar);
                        delay(200);
                        stepper();
                        digitalWrite(BRAILLE0, 0);
                        digitalWrite(BRAILLE1, 0);
                        digitalWrite(BRAILLE2, 0);
                        digitalWrite(BRAILLE3, 0);
                        digitalWrite(BRAILLE4, 0);
                        digitalWrite(BRAILLE5, 0);
                        written++;
                        if (count >= 4) break;
                }
        }
}
```

When I tested the LED circuit the only problem that I stumbled into was caused by the serial port. There is 14 digital I/O pins on the Arduino Uno R3 and, two of those, pin 0 and pin 1, are also responsible for the serial communication. At first I didn't notice this, and used pin 1 for one of the buttons' input. As I had "Serial.begin(9600);" in the **setup** function, and used the "Serial.println();" function to debug the program every time I had a problem, this meant that the serial communication was initialized, thus conflicting with the button that was connected to digital pin 0. The button and also the whole program started glitching out, so after I realized the problem I commented out the lines of code relating to the serial communication.

7. Finalized Product

The previous circuit works very similarly to the designed final version of the project with the electromagnetic actuators, however this circuit has only LEDs meaning, that this circuit doesn't need bi-directional current for the loads to function as imagined. The most significant way that his circuit is different from the final design, apart from the lack of a driver circuit with h-bridge ICs, is the missing reverse current input to reset all the braille cells. This is because LEDs don't need bi-directional current control to work as intended, so that's why nothing is connected to digital I/O pin number 3. The final design isn't that much different from this one. These are the steps to create the final build:

First, I would have to upload the program code to the Arduino Uno R3 microcontroller via serial communication with a PC.
Secondly, I would have to connect the digital I/O pins to their respective female sockets on the driver circuit board (see Fig.13., and its legend) and also connect the ground of the microcontroller with the common ground on the driver circuit.
For the next step I would have to insert the 4 braille cell modules next to each other, by connecting their pins into their respective sockets on the driver circuit board.
For the finishing step, I would then connect the two pushbuttons, with their resistor pair(10kOhm) to two of the microcontroller's Digital I/O, pins and to the ground in serial connection with the resistors, as can be seen on Fig.20.
That would conclude the assembly process of the now complete and functional Refreshable Braille Display.

8. Cost analysis, production options

In this section I will describe the possibilities for development for the refreshable reading board, and add a price estimation to each component.

The first step is the microcontroller, the Arduino Uno R3, but really, any similar microcontroller would do perfectly, as long as it has enough available output pins.

Arduino Uno Rev3 – about 24€ from Arduino's official site[8] (shipping/tax not included)

The next step would be buying the components.

Coils:

For the coils I would use a standard copper wire with 0.5mm diameter and manually wind it around the ferrite core. Its price is about 5€ for 50 meter long, so it is negligible, as I only need about 1 - 1.5meters.

Ferrite cores:

I need a C-shaped ferrite core with about the right size for this, I have found one online, that could fit into the assembly. If the ferrite core is not the exact match, it can be machined. Although a bit hard because the ferrite material is rigid like ceramic, it can be cut, and shaped.

C shaped ferrite core for transformer – about 4.5$ / piece, so 24 pieces = 108$[10]
according to current middle exchange rate: 99.2€

H-bridges:

For the H-bridge IC's I'm going to use the following component:
L9110H H-Bridge Motor Driver for DC Motors - 8 DIP - 2.5V-12V 800mA[11]
The price is 1.5$ or 1.37€ / piece, so 24 piece = 32.88€

For the next step I will share an estimation of the PCBs' production cost.
I used a PCB calculator to estimate the price for a 90x100mm circuit board, because manufacturing one larger board costs less then making several smaller ones. All of the needed boards (the driver circuit's, and the four braille cell modules') can be cut out of the calculated board.
This board is a IPC 6012 Class 2 standard double layered board with through all vias. The material is Tg140 FR4 (the most common dielectric material that is used in the fabrication of circuit boards).
The estimated price is 120.20$ or 110€. [12]

Alternatively, I could order standard PCBs instead of advanced ones but that wouldn't meet any PCB standard, and the minimum piece order there is 5, so in that case I would need to order 10 PCBs totaling to: 25.29$ (driver circuit board) + 25.29$ (braille cell boards) = 50,58$ or 46,46€. [12]

For the pushbuttons, I chose the following component:
PANASONIC ESE20C321 Pushbutton Switch [13]
The price of the component is 573,781 Ft which is 1.51€
for two pieces: 3.02€

Magnetic shielding:

I chose the MuMETAL® Round Bar MUBAR-250 with 6.35mm diameter, however I couldn't find the price, as it gets cut up to custom pieces, and gets machined per request.[14]

Other braille cell actuator parts:

The rest of the actuator mechanism would be made of 3D printed ABS.
Assuming that I would own a 3D printer, 1kg of ABS filament can be bought for about 6200Ft or 16.50€. However since I don't own one the cost of printing these components might be different.

For an alternative, on a 3D printing service website from what I could find I would estimate the printing of all the components together to about the following:
1 actuator case = 3 parts that if we put together is about 5-6mm diameter and 15mm height.
A filled 4x4x4cm cube takes about one hour to print. These components together need less than a $4cm^3$ of material, so I would estimate total printing time to about 50 minutes maximum. The base setup price for the 3D printing is 3500Ft. AN hour of printing starts from 800Ft. So in total the whole operation would be 4300Ft, or 11.3€. [15]

Magnet disks:

Pack of 100 neodymium magnet disks, 2mm diameter x 1mm height
7$ on Amazon (with 12$ shipping), so 6.4€.

The last step would be to assemble each braille pin, and fasten them to each braille cell board. Next, connect the braille cell boards to the driver circuit board, and connect everything, including the pushbuttons to the Arduino Uno R3 microcontroller.

Cost analysis:

Arduino Uno Rev3	24€
Coils	~2€
C shaped ferrite cores	99.2€
H-bridges	32.88€
PCBs	46.46€
Pushbuttons	3.02€
Magnetic shielding	? (~20€)

3D Printed parts	11.3€
Magnet disks	6.4€
Total	**245,26€**

9. Development Options

The design of the circuit boards allows for expansion, what I mean by that, is that a larger driver circuit board could be designed with using the same connections, layout, etc., thus enabling the easy design of a larger/longer braille display with more characters. The only thing needed for a larger display is a port extender IC, to get more available digital output pins. Of course, even after creating a larger circuit board, the braille cell modules' design wouldn't have to be changed. The braille cell modules were designed to be able to be utilized for braille displays with more rows as well, so making something along those lines would also be a possibility.

How could the design be upgraded or polished?

Firstly by adding a plastic cover, to make the device's top flat, so that the visually impaired can easily find the buttons, and the haptic display with no disruption. A cover would also be helpful to hide the exposed electronics of the circuit, and to hide the microcontroller.

Upgrading the way that the text is inputted and read could also be a reasonable attempt, for example instead of changing the text in the program itself, the Arduino could constantly "listen" to the connected device through its serial port, and this way wait for a text to be output onto the tactile display. This might make the use of the circuit more streamlined.

10. Additional Thoughts / Summary

Looking back on the project, and the project's scope, the main goal, which was to create a refreshable braille display what is affordable, has been met. The average price for refreshable braille displays that are commercially available, range from 600€ to several thousands of euros, this makes them rather expensive. The production cost of this project is much lower than these products. Even if we look at this as a prototype, after adding a plastic cover, and some finishing touches by integrating the buttons and the microcontroller into the cover, I believe it would not cost more than 300€.

Final thoughts

In my opinion, this project was an interesting challenge, especially to improve a version of something that already exists but reducing the cost and the size of the product. The best thing in the project is probably the braille cell module, which was designed in mind to make it as small-sized as possible with a minimalistic design, but to also make it interchangeable, with other similar, potentially improved modules. The fact that the braille cell modules are not soldered onto the driver circuit means that they can be hotswapped from their slots with other similar designs, which only incorporate the load of one coil, or faulty modules, makes it easier to modify or repair the circuit without completely breaking-down the circuit.

The final size of the board is also commendable in my opinion, with the driver board taking up a 4.5x10cm space, and the braille display part consisting of 4 braille characters taking up only 4.1x7.4cm space, making it rather compact. Multiple driver boards could also be connected to one microcontroller, given enough digital I/O pins are available, and could be placed next to each other to make a longer haptic display. Of course, in case of any sort of extension, the code would have to be slightly modified.

The most challenging part of the project which I spent most time with all-in-all is probably the code. As I had to rewrite the code multiple times, not only to be able to test a similar but different build in Tinkercad, but also to remake the code for the final product without being able to test it.

11. References:

[1] Prof. H. Shea – Haptic displays (2022) -> 12x16 Latching electromagnetic haptic display:
Source: https://www.epfl.ch/labs/lmts/lmts-research/haptics/

[2] Piezoelectric actuator based braille character display – METEC AG:
Source: https://www.metec-ag.de/downloads/braille-line-flat-20-elektronik.pdf

[3] 2019/02 - MagTics: Flexible and Thin Form Factor Magnetic Actuators for Dynamic and Wearable Haptic Feedback:
Source: https://www.epfl.ch/labs/lmts/wp-content/uploads/2019/02/mag-tics-uist-lowres.pdf

[4] METEC – MODUL P20
METEC AG – Hasenberg Str. 31. – 70178 Stuttgart – Germany
https://www.metec-ag.de/downloads/p20.pdf

[5] Gautschi, G. (2002). *Piezoelectric Sensorics: Force, Strain, Pressure, Acceleration and Acoustic Emission Sensors, Materials and Amplifiers*. Springer. doi:10.1007/978-3-662-04732-3. ISBN 978-3-662-04732-3.

[6] Vijay (2023) - Electromechanical Refreshable Braille Module: Lowering the cost of Refreshable Braille Cells by using Electromagnetic Cam Actuators & 3D Printing
Source: https://hackaday.io/project/191181-electromechanical-refreshable-braille-module

[7] Wikipedia – Ferrimagnetism (10. 02. 2007)(last edited 31. 10. 2023)
Source: https://en.wikipedia.org/wiki/Ferrimagnetism

[8] Arduino UNO R3 documentation/website (last edited 09. 12. 2023)
Source: https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf

[9]Wikipedia – Braille (18. 03. 2005) (last edited 08. 12. 2023)
Source: https://en.wikipedia.org/wiki/Braille

[10] Ferrite Core C Shape Core for Transformer
Source: https://evergrowrs.en.made-in-china.com/product/pxErmATVTjkS/China-Ferrite-Core-C-Shape-Core-for-Transformer.html

[11] Adafruit L9110H H-Bridge Motor Driver for DC Motors - 8 DIP - 2.5V-12V 800mA
Source: https://cdn-shop.adafruit.com/product-files/4489/4489_datasheet-l9110.pdf

[12] Advanced PCB Manufacturing Cost Calculator
Source: https://www.pcbway.com/HighQualityOrderOnline.aspx

[13] PANASONIC ESE20C321 Pushbutton Switch (October 2012)
Source: https://www.farnell.com/datasheets/1790644.pdf

[14] Custom cut MuMETAL® Round Bar MUBAR-250 (broschure: 2012)
Magnetic Shield Corporation, 740 N. Thomas Drive, Bensenville, IL 60106 USA
Source: https://store-w4rbnih33r.mybigcommerce.com/content/mu-2.pdf

[15] 3D Bérnyomtatás – Budapest
Source: https://rocket3d.hu/

[16] Amazon – JUNAN Mini Neodymium Magnet 2mm x 1mm Tiny Rare Earth Disc Magnets - 0.06kg Pull (Pack of 100)
Source: https://www.amazon.com/JUNAN-Neodymium-Magnet-Earth-Magnets/dp/B09V14FGQF/ref=sr_1_1?crid=2G3N4RWOHA6IP&keywords=neodymium+1x2mm&qid=1701529675&sprefix=neodymium+1x2m%2Caps%2C188&sr=8-1

[17] Created with: https://creately.com/lp/block-diagram-maker/