Aalto-yliopisto, Sähkötekniikan korkeakoulu

ELEC-D0301 Protopaja

2024

Loppuraportti/Final report

Project #05

Savox noise data collector

Title: Laita tähän se otsikko jonka ilmoititte projektisuunnitelmassa. Ei pelkkä yrityksen nimi.

Valitse joko väliraportti tai loppuraportti, ja poista toinen.

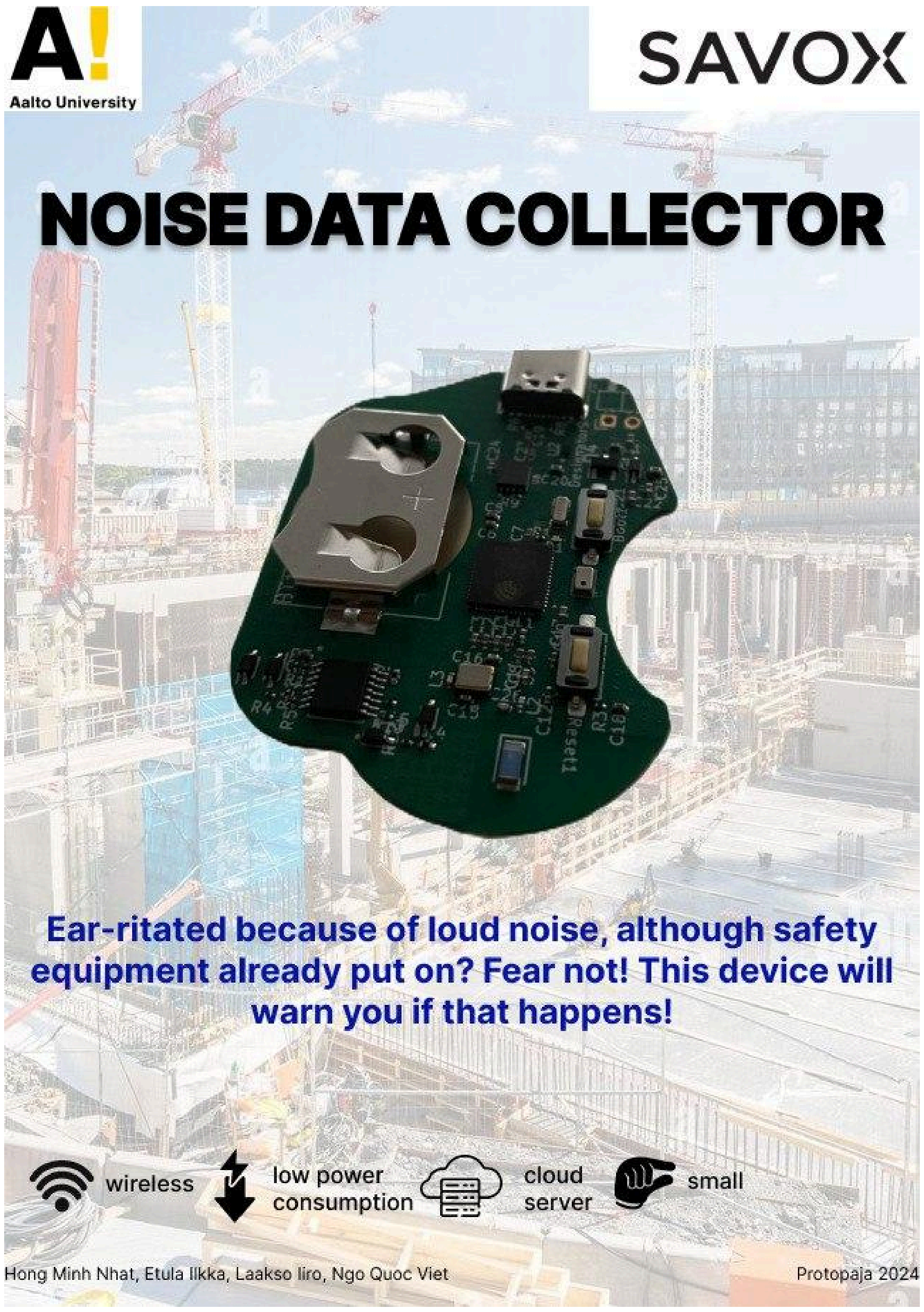

Aalto University
SAVOX
NOISE DATA COLLECTOR
Ear-ritated because of loud noise, although safety equipment already put on? Fear not! This device will warn you if that happens!
wireless
low power consumption
cloud server
small
Hong Minh Nhat, Etula Ilkka, Laakso Iiro, Ngo Quoc Viet
Protopaja 2024

Figure 1: group's poster

Date: 29.8.2024

Hong Minh Nhat

Ilkka Etula

Ngo Quoc Viet

Iiro Laakso

Information page

<u>Students</u>

Hong Minh Nhat mail:nhat.hong@aalto.fi

Quoc Viet Ngo mail:viet.ngo@aalto.fi

Iiro Laakso

Ilkka Etula

<u>Project manager</u>

Nhat Hong

Sponsoring Company

Savox

Starting date

3.6.2024

Submitted date

29.8.2024

Tiivistelmä

Projekti sisältää esineiden internetin kehittämistä. Kompakti ääni datan keräämistä varten kehitetty laite, joka sopisi kuulosuojaimiin, jotka on suunnitellut Savox, yhtiö joka spesialisoituu kommunikaatio ja turvallisuus ratkaisuihin. Tämä innovatiivinen kone pyrkii seuraamaan ja tallentamaan taustamelun voimakkuutta samalla kun se on mukavasti käytössä. Kerätty data, joka sisältää monenlaisia äänen mittareita, lähetetään ajoittain netti serverille, varastointia ja analyysia varten.

laite hyödyntää ESP32 microcontrolleria, johon kuuluu korkeatasoinen analogia-digitaalimuunnin tarkan äänen tason mittaamista varten. Se muodostaa yhteyden Internetiin Wi-Fi-yhteyden kautta tietojen siirtämiseksi palvelimelle. Kerätty data on järjestetty jäsenneltyihin äänitallennuksiin, mukaan lukien aikaleimat ja äänen amplitudi lukemat. Nämä tiedot lähetetään sitten yhdessä JSON-paketissa palvelimelle, mikä varmistaa tehokkaan tiedonsiirron ja minimaalisen viiveen.

Palvelinpuolella verkkosovellus käsittelee ja tallentaa saapuvat tiedot. Sovellus näyttää reaaliaikaiset melutasot ja historialliset tiedot, jotka tarjoavat arvokasta tietoa ympäristön meluolosuhteista. Tämän asetuksen avulla käyttäjät voivat seurata melualtistustasoja ajan mittaan ja tarjota sekä välitöntä palautetta että pitkän aikavälin analyysejä. Tämän laitteen integrointi

kuulosuojaimiin parantaa käyttökokemusta tarjoamalla käyttökelpoisia melu tietoja saumattomasti ja käyttäjäystävälliseksi.

Ryhmän jäsenten roolit:

-Hong Minh Nhat: fyysinen rakenne, piirrosmerkkikaavion suunnittelu, ryhmänjohtaja

-Ilkka Etula: Sulautettu rakenne, energiankulutuksen laskeminen

-Ngo Quoc Viet: PCBn ohjelmoiminen

-Iiro Laakso: palvelimen ohjelmointi

Summary

The project involves developing an IOT, compact noise data collection device designed to be integrated into a headband product from Savox, a company specializing in communication and safety solutions. This innovative device aims to monitor and record ambient noise levels while being worn comfortably. The collected data, which includes various noise metrics, is periodically sent to a web server for storage and analysis.

The device leverages the ESP32 microcontroller, which is equipped with a high-resolution ADC to measure noise levels accurately. It connects to the internet via Wi-Fi to transmit data to the server. The collected data is organized into structured records, including timestamps and noise amplitude readings. This data is then transmitted in a single JSON package to the server, ensuring efficient data transfer and minimal latency.

On the server side, a web application processes and stores the incoming data. The application displays real-time noise levels and historical data, providing valuable insights into environmental noise conditions. This setup enables users to monitor noise exposure levels over time, offering both immediate feedback and long-term analysis. The integration of this device into the headband product will enhance user experience by providing actionable noise data in a seamless and user-friendly manner.

The role of each member of our teams are as following:

-Hong Minh Nhat: physical design, schematic design, group lead

-Ilkka Etula: embedded design, power consumption calculations

-Ngo Quoc Viet: PCB programming

-Iiro Laakso: full stack (front end and back end) server developer

Table of Contents / Sisällysluettelo

1. Introduction / Johdanto

Construction workers have to work at construction sites all the time. As such, the working condition is dangerous in multiple aspects. One of such is hearing, as the machines are usually very loud. Thus, the workers have to wear ear mufflers to protect their ears against deafening, as sounds that are greater than 85dB are harmful for humans. However, the muffler is not an almighty protection against sound pollution. There might be sound leakage into the muffler that is caused by factors such as the workers' facial hair or them wearing glasses. As such, it can be dangerous for the user. Therefore, our group aims to develop a device that can record the sound data within the muffler for muffler developing purposes, and also warns the user that the sound level is beyond a certain safe threshold.

2. Objective / Tavoite

The mission of the project is to create a device which can measure the intensity of the sound and transmit that data to a server which will display it to its user. The device has to be small enough to fit inside the hearing protector and be able to work for a full work day (approx. 8 h). Device has to be durable enough so that things like sweat or small drizzle don't damage the machine while it is in use. Machine also has to fit into the hearing protector in a way that it does not touch or irritate the ear so that it would become uncomfortable to wear during a workday.

3. Code

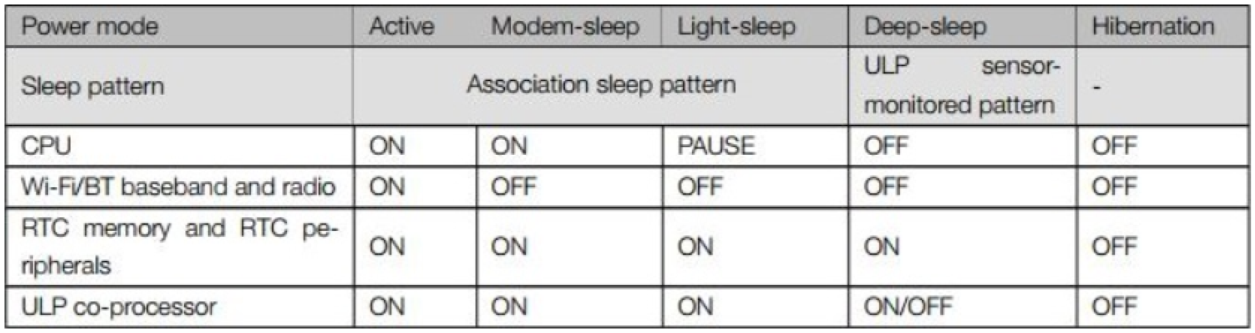
3.1. Arduino IDE

The Arduino IDE (Arduino Ide) is used to code the Esp32 for its useful tools and library that support the microcontroller.

Some drawbacks with this IDE is that it does not provide a debugging function, and the compiling speed can be long. But, we still decide to use it because of our past experience with it and the easy and simple interface that it provides.

3.2. Esp32 sleep mode

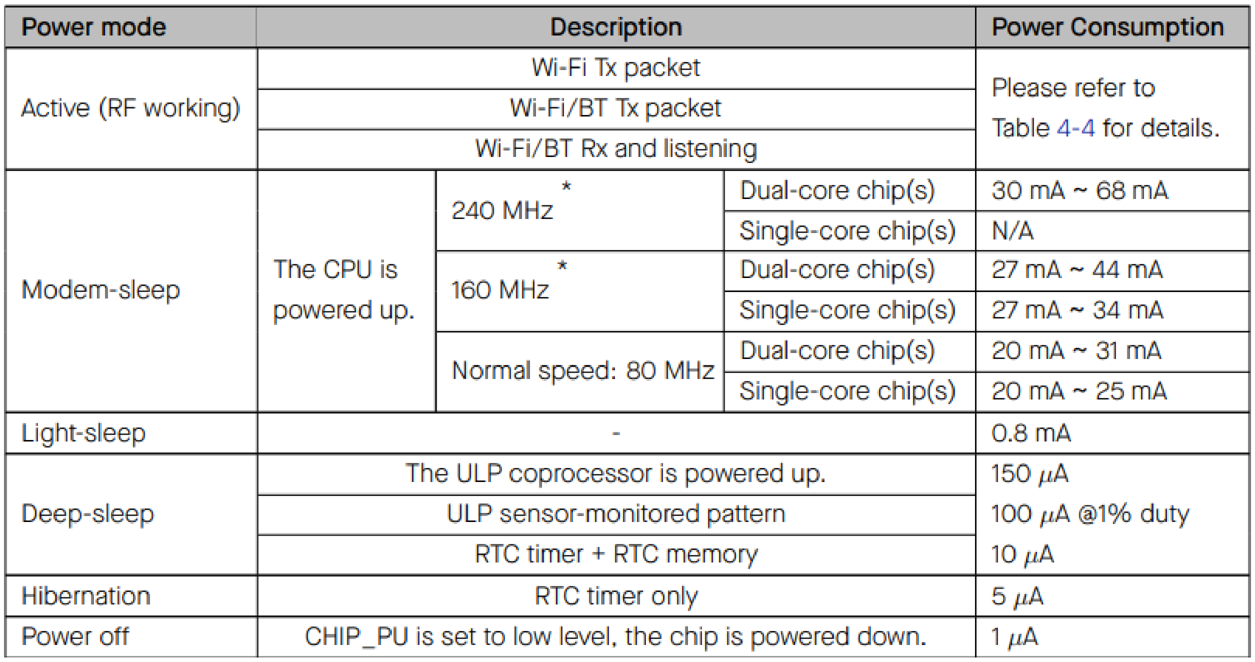
The main key for a low energy IoT device is sleep mode, a state where the device powers down and turns off some of its components. The table below shows the different power mode of the esp32 and the state of its components.



Power mode	Active	Modem-sleep	Light-sleep	Deep-sleep	Hibernation
Sleep pattern	Association sleep pattern			ULP sensor-monitored pattern	-
CPU	ON	ON	PAUSE	OFF	OFF
Wi-Fi/BT baseband and radio	ON	OFF	OFF	OFF	OFF
RTC memory and RTC peripherals	ON	ON	ON	ON	OFF
ULP co-processor	ON	ON	ON	ON/OFF	OFF

Figure 2: The table illustrates the different power modes and components state respectively(esp32_datasheet_en)

The data sheet also provides the comparison of the power consumption between different modes.



Power mode	Description			Power Consumption
Active (RF working)	Wi-Fi Tx packet			Please refer to Table 4-4 for details.
	Wi-Fi/BT Tx packet			
	Wi-Fi/BT Rx and listening			
Modem-sleep	The CPU is powered up.	240 MHz *	Dual-core chip(s)	30 mA ~ 68 mA
			Single-core chip(s)	N/A
		160 MHz *	Dual-core chip(s)	27 mA ~ 44 mA
			Single-core chip(s)	27 mA ~ 34 mA
		Normal speed: 80 MHz	Dual-core chip(s)	20 mA ~ 31 mA
			Single-core chip(s)	20 mA ~ 25 mA
Light-sleep	-			0.8 mA
Deep-sleep	The ULP coprocessor is powered up.			150 μA
	ULP sensor-monitored pattern			100 μA @1% duty
	RTC timer + RTC memory			10 μA
Hibernation	RTC timer only			5 μA
Power off	CHIP_PU is set to low level, the chip is powered down.			1 μA

Figure 3: The table illustrates the power consumption of each component in different power mode (esp32_datasheet_en)

If the esp32 is active all the time, the power consumption would be 95 to 250 mA, which is not ideal for the device's intended purposes. That is why the sleep mode is needed to power down the device when necessary. The team has chosen the deep sleep mode after consideration, further clarification is in 3.3 Code flow section.

3.3 ULP (ultra low power coprocessor)

The **ULP (Ultra Low Power) coprocessor** in the ESP32 is a specialized processor designed to handle low-power tasks while the main CPU (the main Xtensa cores) is in a deep sleep mode.

It is another feature of the esp32 that is helpful in reducing the power consumption (typically 20 uA). The ulp can be utilized to collect the data instead of using the main cpu.

The ULP has its own RAM, which is 8kB, and the data in the esp will be stored as uint16_t which after approximately 1900 data points (this include timestamps of the data).

3.4. Project code flow and microphone calibration

Most of the libraries used in the code are already available in the Arduino ide default library. However, for the purpose of this project, some additional library need to be used:

#include <WiFi.h> to help with wifi connection (Wi-fi library)

#include <HTTPClient.h>   to help with sending data to the server (HTTPClient Library)

#include <ArduinoJson.h>   to configure the data sent (ArduinoJson Library)

#include <esp_sleep.h> to initialize sleep mode settings (ESP Sleep Library)

Alongside some math library for the data processed.

Below is the code flow simplified:

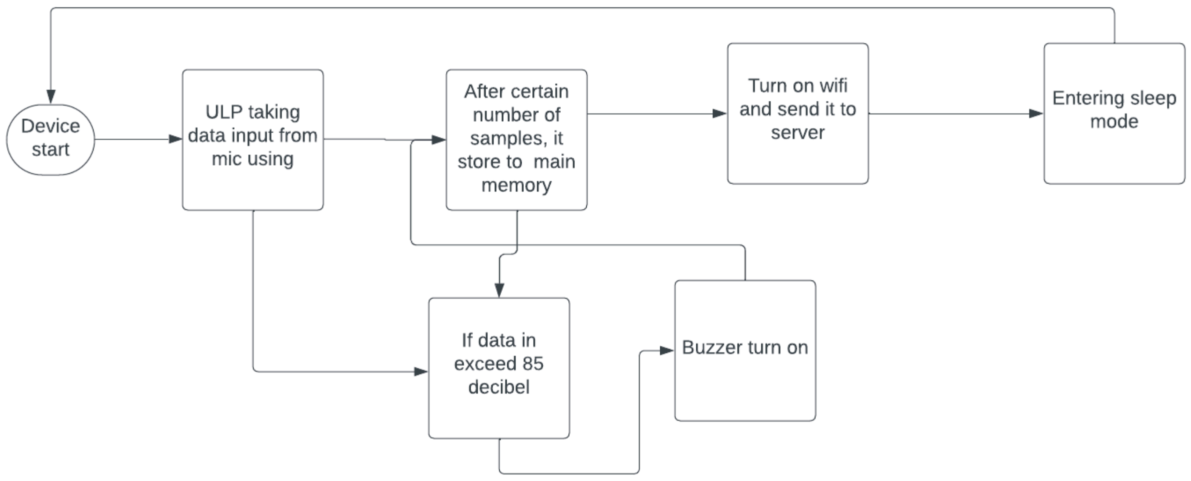

Device start
ULP taking data input from mic using
After certain number of samples, it store to main memory
Turn on wifi and send it to server
Entering sleep mode
If data in exceed 85 decibel
Buzzer turn on

Figure 4:code flow

Microphone calibration is needed because the raw data read with the ulp from the microphone is not meaningful. So I have utilized a logarithm gain function log10(raw_data/ref_signal)*100+30. This will give us the gain in signal, and for the microphone we use, the ref_signal would be 50. For example, if the signal input is 70, which is 1.4 times the reference signal of 50 when there is no sound (around 25 to 30 decibels).

4. Server

4.1. The frontend code

4.1.1. drawing a graph

I had from the start a vision of how I would display the data my code would receive from the device. I wanted it to be a line graph that would display the noise level and how it changed during the course of somebody's workday. I also wanted there to be an easily readable display that depicted time spent over certain decibel thresholds so that potentially harmful long exposure to loud sounds would be very easy to see by looking at the time the graph spent over certain thresholds. In addition to this I also wanted the code to be able to show history of previous graphs in it, have simple customisation options and allow you to download the data for further study and recordkeeping.

I started by learning how javascript code works and what you can do with it. My first idea was to make an object that draws the graph onto the grid line when it receives the data. I managed to get an object that constantly moves to the right and resets back to the left side at the end. It also draws a line that represents the amount of decibels the microphone receives at any given moment.

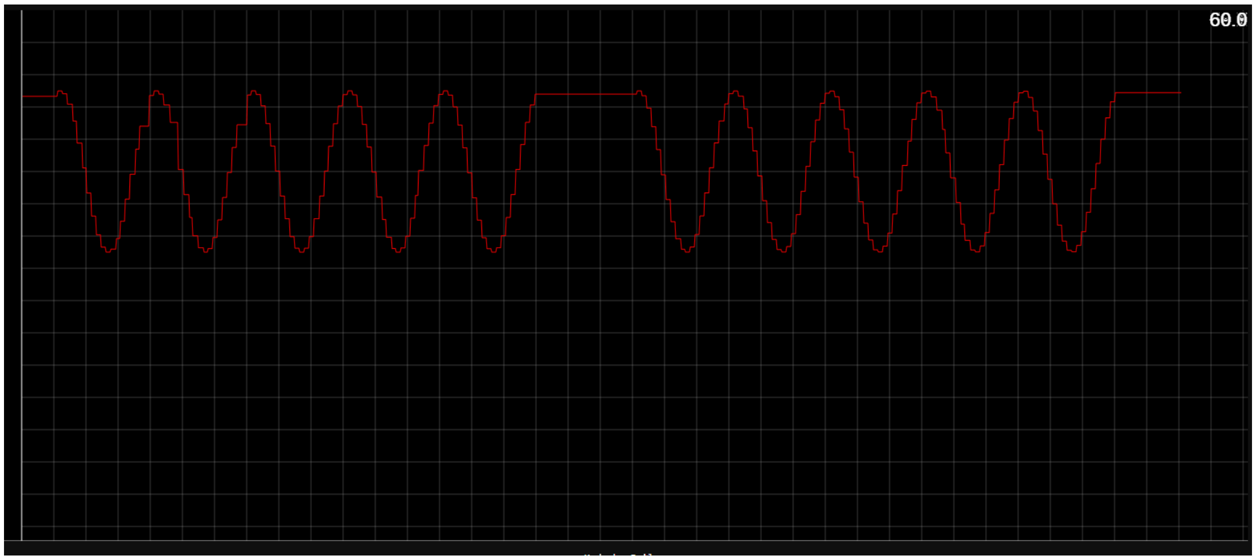


Figure 5: This is a picture of that codes output

After that I decided to scrap this method and instead decided to utilize a chart.js library on making the graph. Main reasons for this change were the ease of use, ability to depict very large sets of data and precision. Chart.js is a lot more precise and showcases data points better than just a drawn shape could. It also has many customisation options and works a lot easier when the amount of data is very large. I first tested this code with a collatz sequence so see how the graph would be displayed.

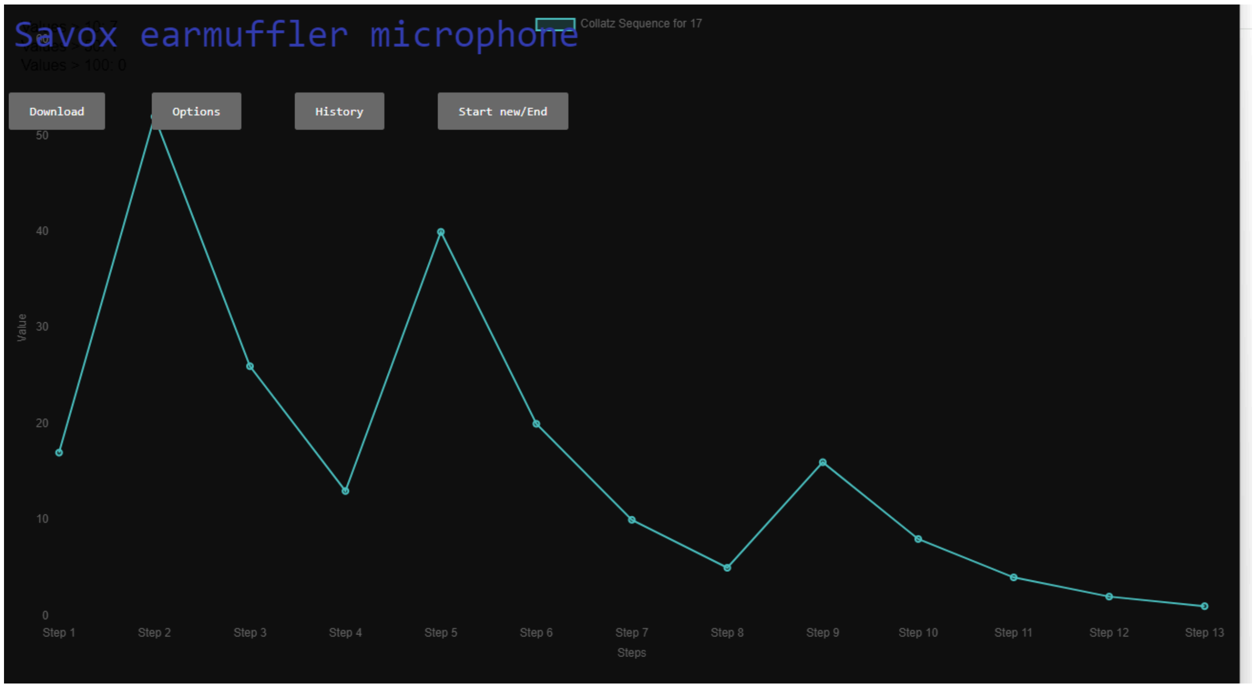
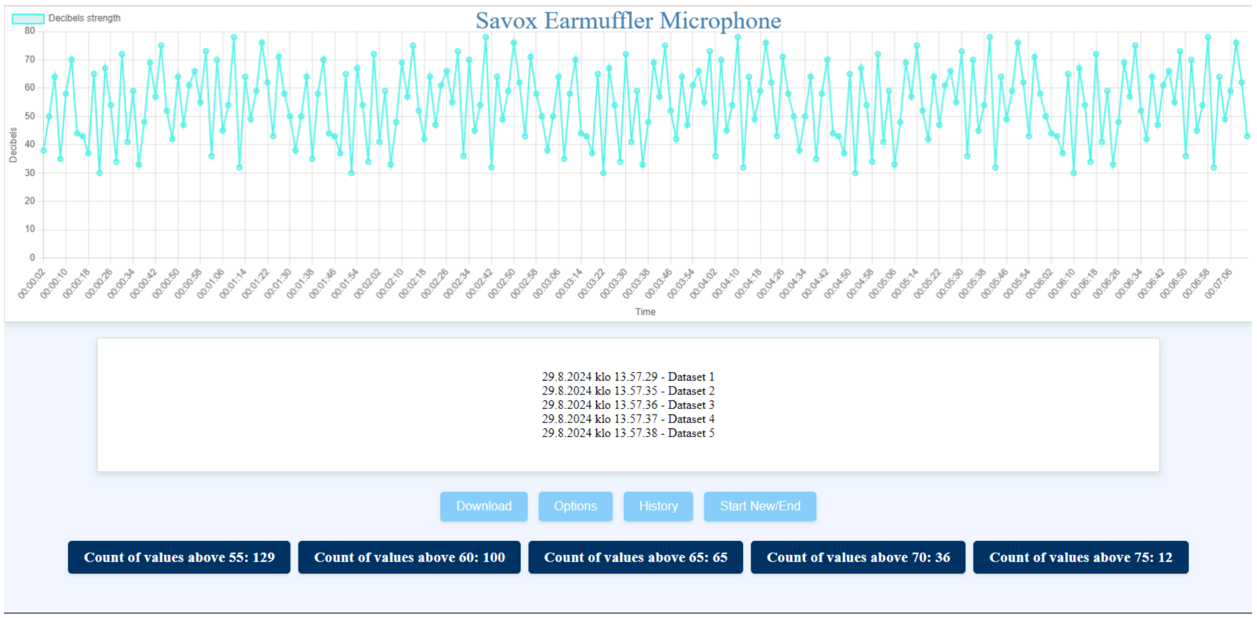


Figure 6: Here is a picture of how the graph would look with the help of chart.js

After this I added the basic functionality to some of the buttons. I made it possible to start seeing the chart by clicking the start new/End button and I added a possibility to download the data with the Download button. After that I cleaned up the appearance of the server and made it visually more appealing. I also added the text that showcases time spent over certain thresholds. I also added the possibility to use the history button to move back to previous noise data send to it.

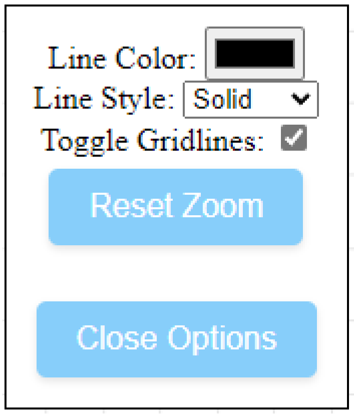


29.8.2024 klo 13.57.29 - Dataset 1
29.8.2024 klo 13.57.35 - Dataset 2
29.8.2024 klo 13.57.36 - Dataset 3
29.8.2024 klo 13.57.37 - Dataset 4
29.8.2024 klo 13.57.38 - Dataset 5

Count of values above 55: 129
Count of values above 60: 100
Count of values above 65: 65
Count of values above 70: 36
Count of values above 75: 12

Figure 7: Picture of the final state of the server's appearance.

I also changed the outlook of the server to a different color. I chose light blue as a color, since I consider lightly colored websites very pleasing and blue is a color that is displayed very prominently in the Savoxs own website. Lastly I added customisation options for the graph and an zooming functionality that makes it easier to read very large datasets.



Line Color:
Line Style: Solid
Toggle Gridlines: ✓

Figure 8: Picture of options menu.

4.2. The backend code and Server

I started by learning about how the backend code works and how servers work. I learned from the Aalto university wiki pages mini project about how you work with the server. I added code from my computer to that server's directory. That code is able to create an address which it then listens to until something sends to it an json file that it can then store so that my frontend code could receive it and display it at the website. This is done by parsing any message that it receives and turning it into a json file which it then stores. If received data cannot be turned into a json file it will send an error message about this issue.

5. Embedded

5.1. The planning

Because of the small size needed for the device, we found it necessary to design our own PCB. Initially we intended to use AA or AAA batteries, but we found coin cell batteries to be a better option even before we could start designing the PCB. After that we were thinking about using a CR2477 battery, but finally we chose to use a smaller CR2032 battery. For the microcontroller, the TAs suggested for us to use an Esp32 for its low power use. Later we had to switch to a different version of it, because the version we were planning on using had no internal memory. The final Esp32 model we used was Esp32-S3FN8. Our initial plan was to use the deep sleep mode of the Esp32 and make a peak detector circuit to save on power usage of our design. We also have to set the initial settings like clearances and trace thicknesses like how it is instructed on JLCPCB manufacturing page.

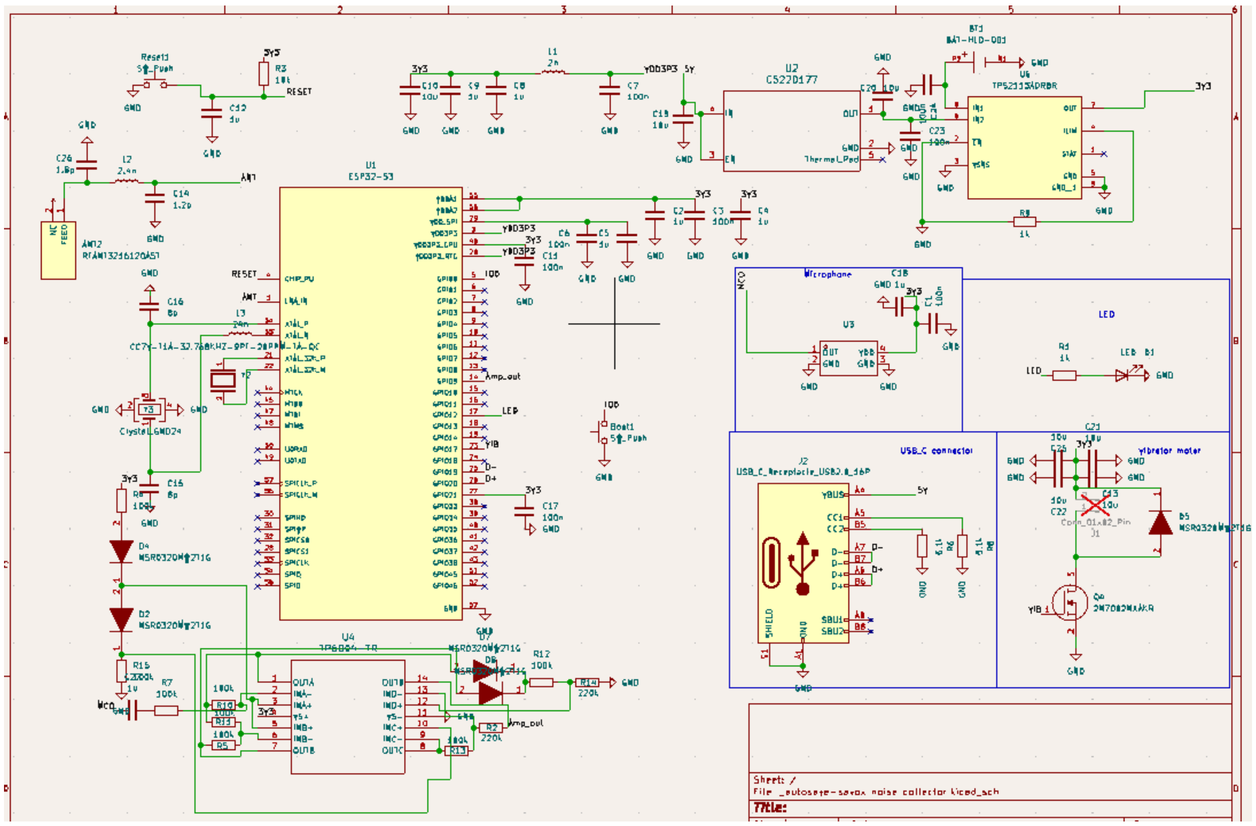


Figure 9: overall schematic of the project

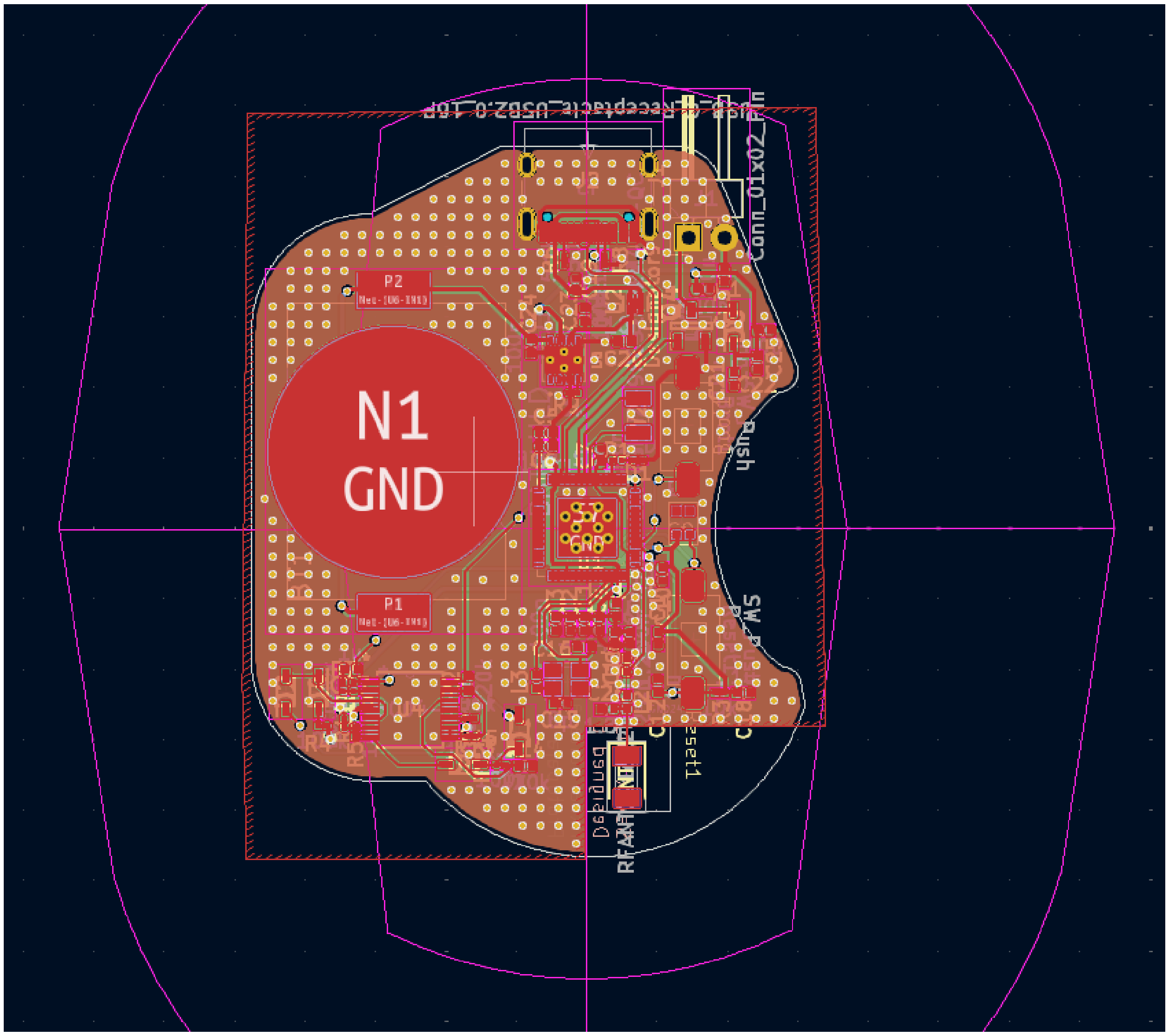


Figure 10: overall appearance of the PCB as an end product. The purple lines are an estimate of how big the muffler would be.

5.2. Power source

We still had to make circuitry to handle any microphone output signals that had negative voltage. We had a few minor issues with the USB ports power output. Our voltage regulator's output was at 3,3V and it had to be swapped for one that outputs at 3V to match our battery. We also had to add an OR-gate to disconnect the battery when the USB provides the power.

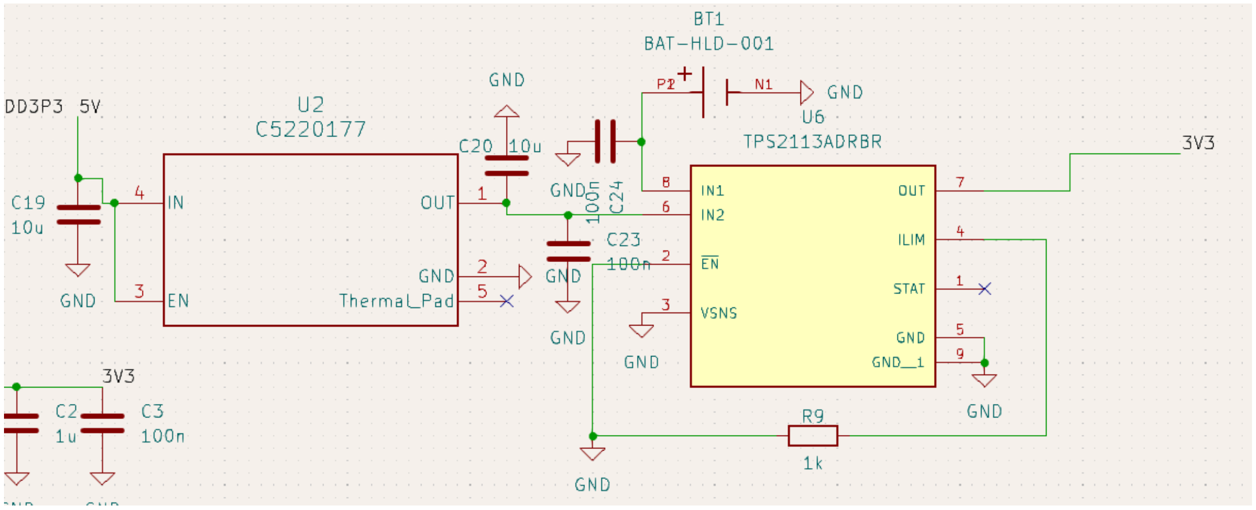


Figure 11: circuit responsible for distributing power from different sources

Circuitry handling the power input from USB (5V in the picture) and CR2032 battery (BT1 in the picture). U2 is the voltage regulator and U6 is the OR-gate. We used an OR-gate here to make sure that if both the battery and the USB are online, it will not fry the circuit, because it will only take in one power-source instead of both of them. We wanted to maximize usage for as long as possible, so we decided to run in deep sleep mode. However, that proved hard to pick up signals collected from the microphone. We proceeded with Ultra Low Power mode(ULP), so that it saves energy the best that it can.

5.3. Microphone

At first, we decided to process the data from hardware by using a peak detection circuit. The purpose of it is to detect when the signal is highest, hold it in that position, and reset it whenever we want.

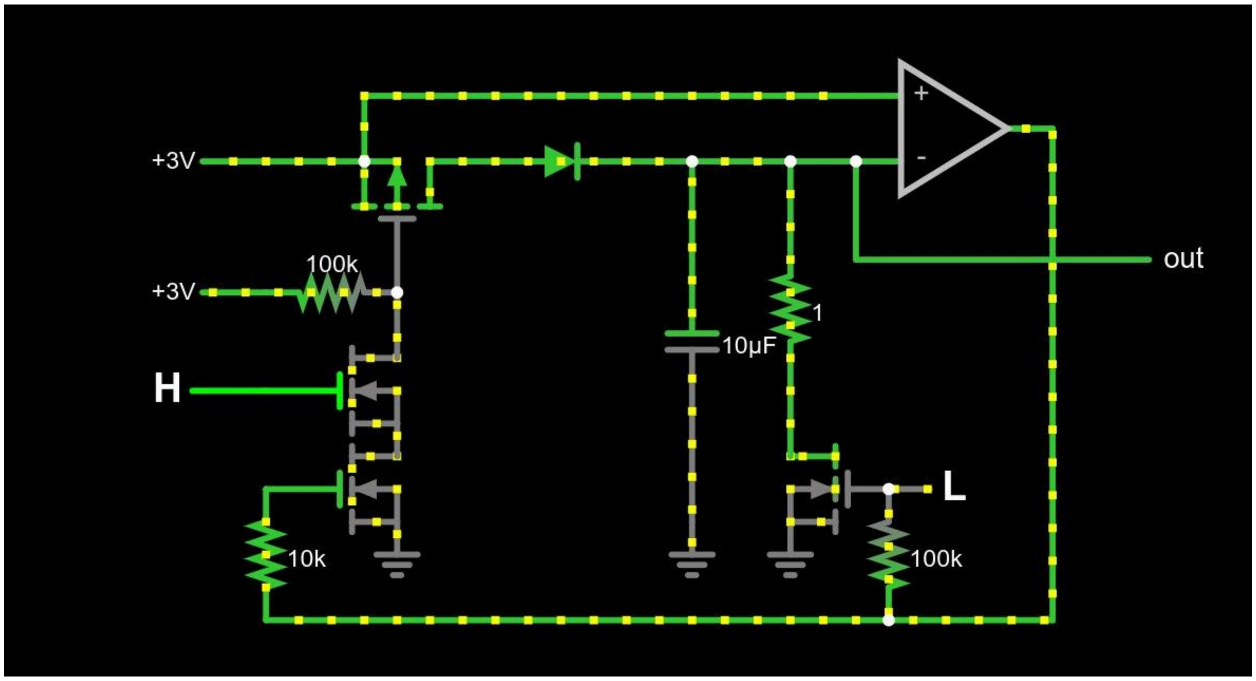


Figure 12: a circuit design that intends to capture the highest signal

At first, we tried to use an IC(integrated circuit) peak detector, but it turns out they do not use an ideal diode(a diode where it is hooked with an operational amplifier), thus causing voltage drop. Therefore, we have to design a peak detecting circuit ourselves.

We soon encountered issues with how complex the detection circuits are, more specifically the peak detector's signal either decays too fast, or could not adapt to sudden change. In both cases, we lost collected data. Thus, we chose to handle those functions mathematically with the Esp32 in ULP mode.

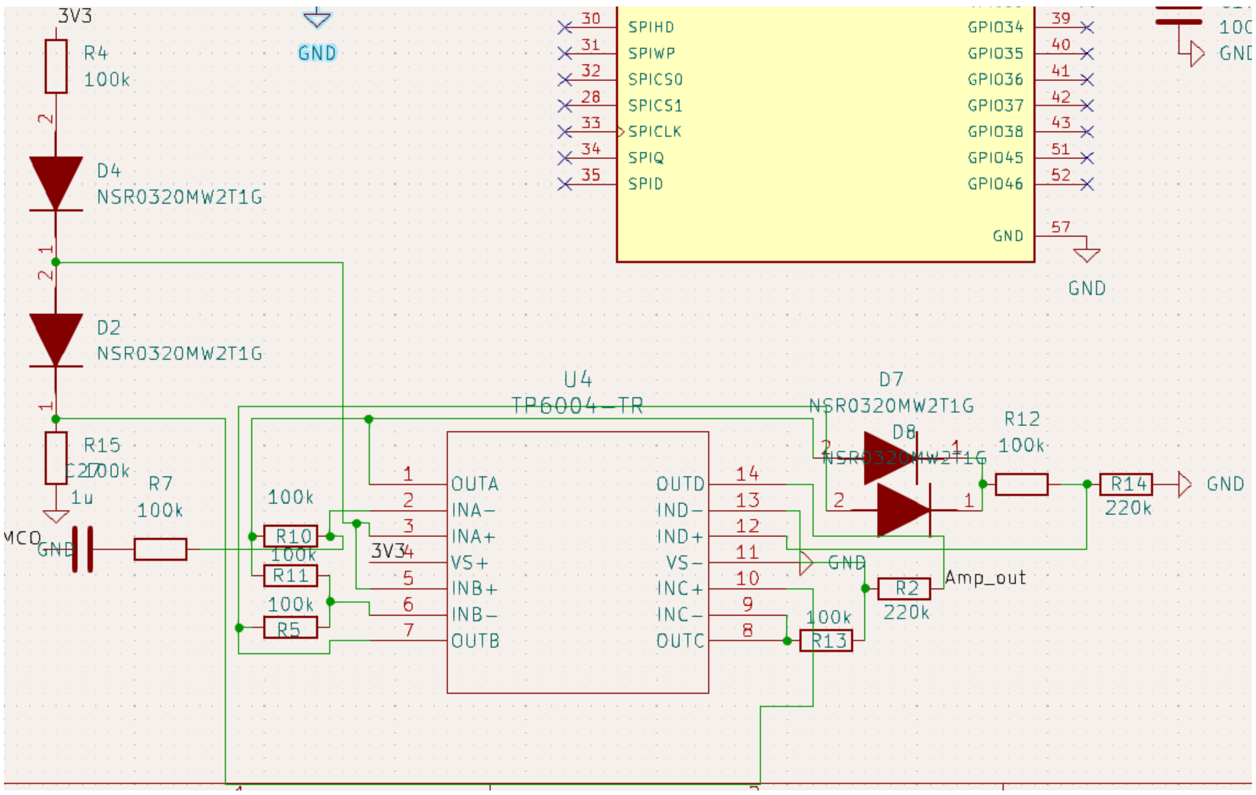


Figure 13: circuit that makes sure no negative voltage input from microphone is received

This is a circuit that is used to handle microphone signals (MCO in the picture) and output a signal we can use (Amp_out in the picture). We are not sure if the microphone will generate negative voltage or not, and that will ruin our ESP. As such, we used a processing circuit above to ensure we do not receive any negative signals while maintaining the signal integrity. We also used a MEMs analog microphone to record sounds.

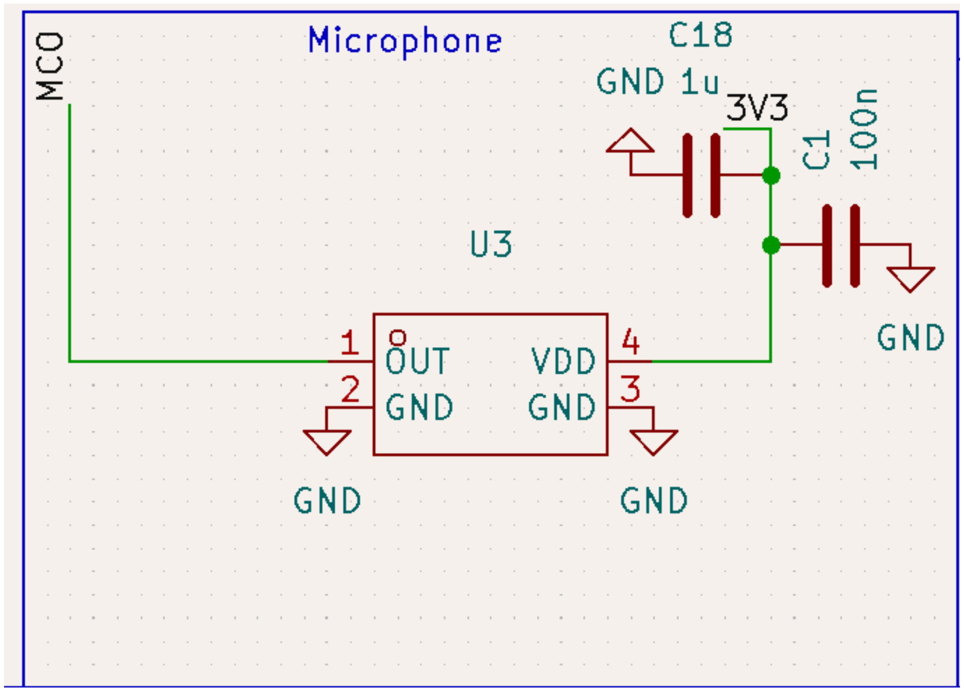


Figure 14: the microphone wiring

We also had to manually draw our own microphone footprints, as we could not find any equivalent or suitable footprints online.

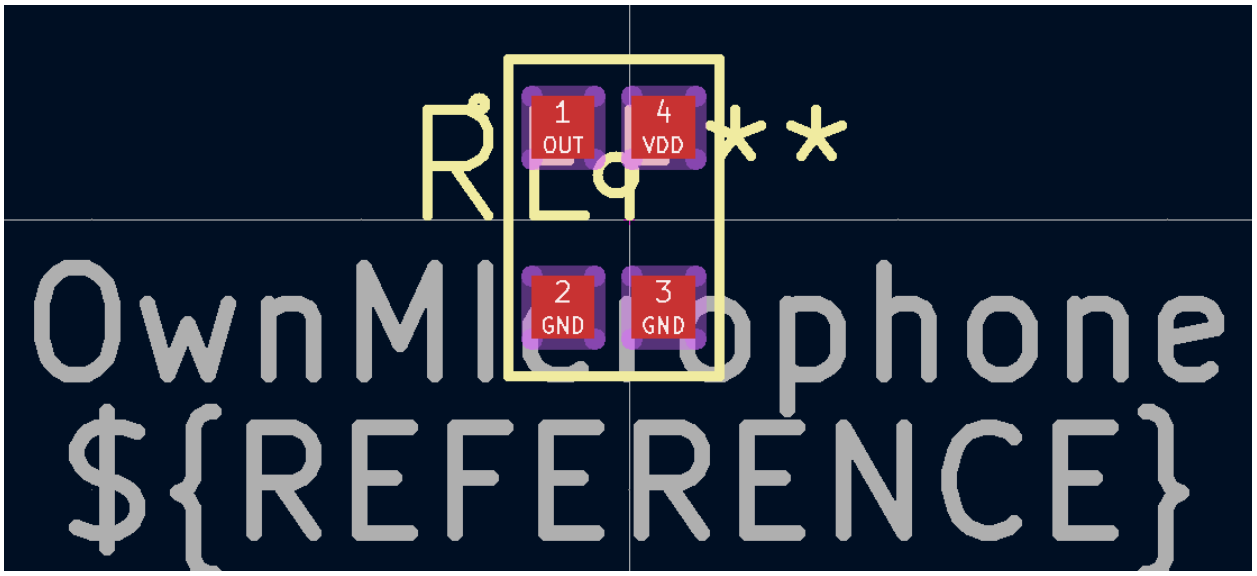


Figure 15: self-made footprint of a microphone

5.4. Vibrator

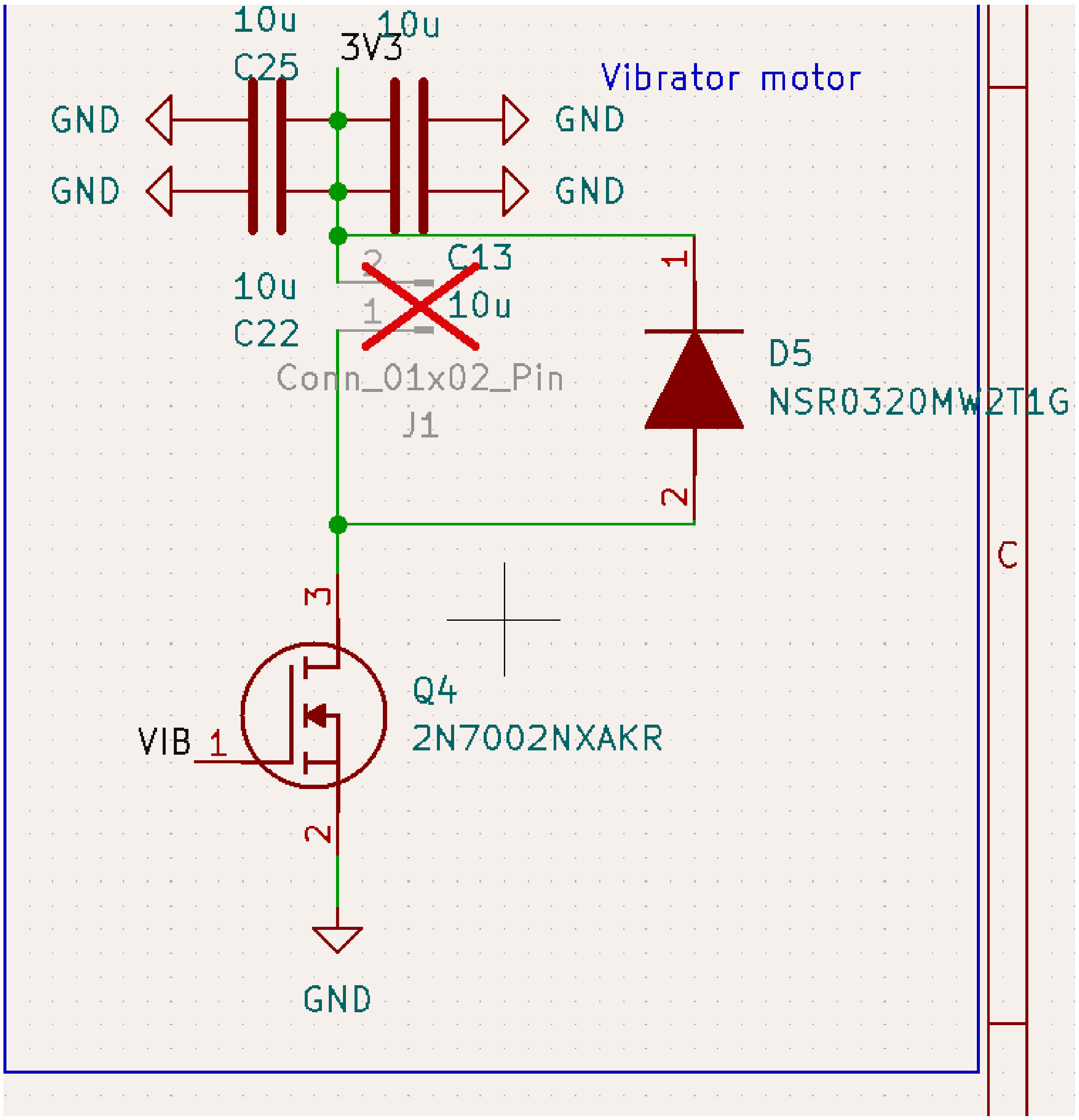


Figure 16: the vibrator motor circuit

At first, we planned that in order to notify the user that the sound level is too dangerous, we wanted to use both a small speaker and a vibrator. We wanted to warn the user with either tactile or audio, because it currently lies within the user's ear. The vibrator consumes a lot of energy, so we were considering a small speaker. But then, the speaker, although consumes less energy, might feed back noise signals into the microphone again. Therefore, we went with a vibrator.

5.5. Microcontroller

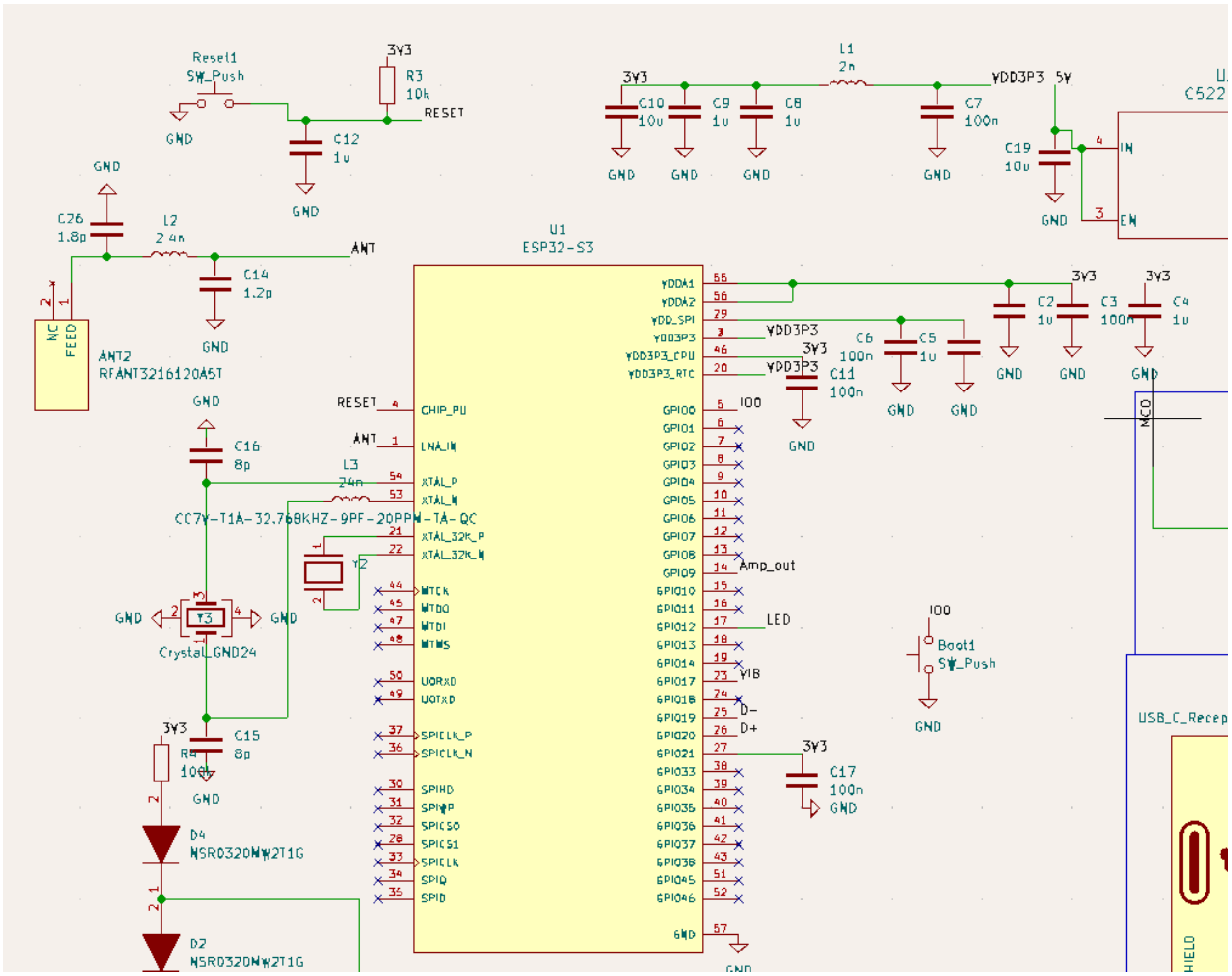


Figure 17: the microcontroller schematic

The ESP32S3FN8 was chosen as it has a flash memory installed within it already. The only main issue with the microcontroller was the crystal/oscillator that acts as an internal clock for the ESP. We need a 32.768KHz crystal for the antenna, and 40MHz crystal for the ESP. We also decided to use a microcontroller instead of a module because it is considerably a lot smaller. We also were not able to find any suitable crystal footprints online so we had to make one on our own.

Our microcontroller's design is based on a module adafruit py ESP32S3.

After we got the PCBs we noticed they didn't work. Because of time constraints, we couldn't order any new PCBs so we had to try to do any necessary fixes manually on the PCBs we had. At first, we figured the problem might lie within the crystal, so we tried to solder a parallel capacitor to match it with the crystal. We had to solder the capacitors connected to the 40 MHz crystal for ones with larger capacitance and swap the resistor connected to the OR-gate (R9 in picture of power circuitry (still at its original value of 1 kΩ)) for one with less resistance. But then it still did not function. We then replaced the crystal we installed with one from other broken ESP32 S3s by blowing hot air at 340 degrees for about 2 minutes, because apparently the one we used has 20ppm, when the PCB only functions with 10ppm. After the change, it worked. We also noticed that 3V was not enough to power the device and we had to try connecting a 3,7V LiPo battery. The voltage drop before it goes to another component is 0.2V, so when we tried to use the 3V battery one, the voltage drop brought it down to 2.7 or 2.8V, which is too low for the microcontroller to run. Therefore, we decided to power it with a Lipo battery with 3.7V.

The ESP also has 2 buttons: Reboot and Reset. The buttons are placed close to each other for ease of identification. A pin of the microcontroller is also connected to a LED for debugging purposes, because if it is hooked into the source, it would be wasting power.

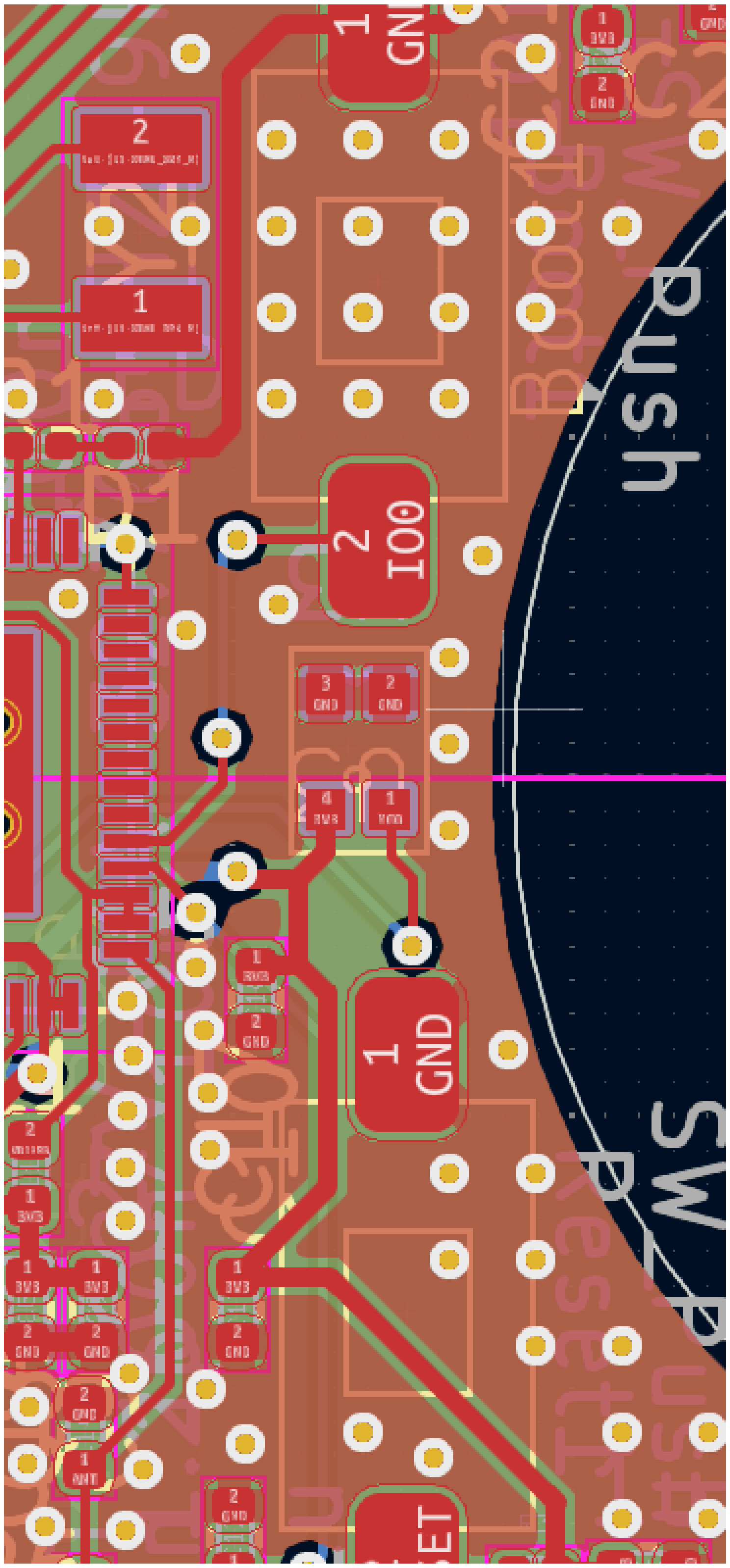


Figure 18: two buttons are positioned close to each other

The capacitors that are connected to the PCB are done as guidelines for ESP32 S3 online. The link will be within references. To program the microcontroller, the two pins GPIO19 and GPIO20 are connected to D- and D+ respectively of a USB connector. We decided to use a Type-C connector.

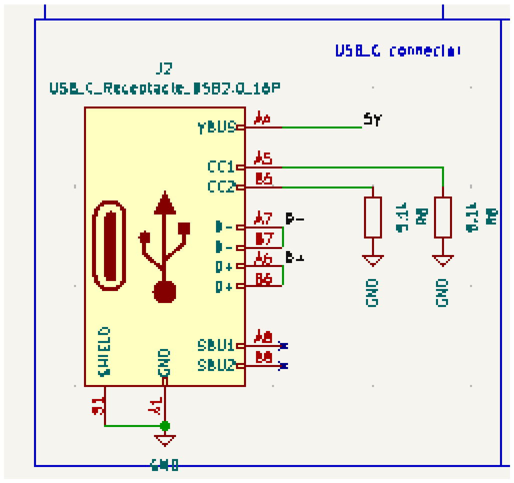


Figure 19: USB connector

As in the datasheet noted,GPIO0 should not be left floated, thus we connected it to the reboot switch.

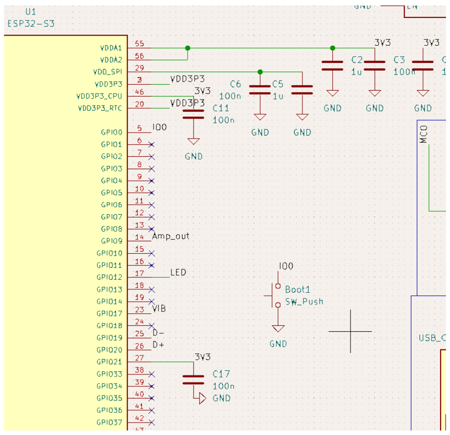
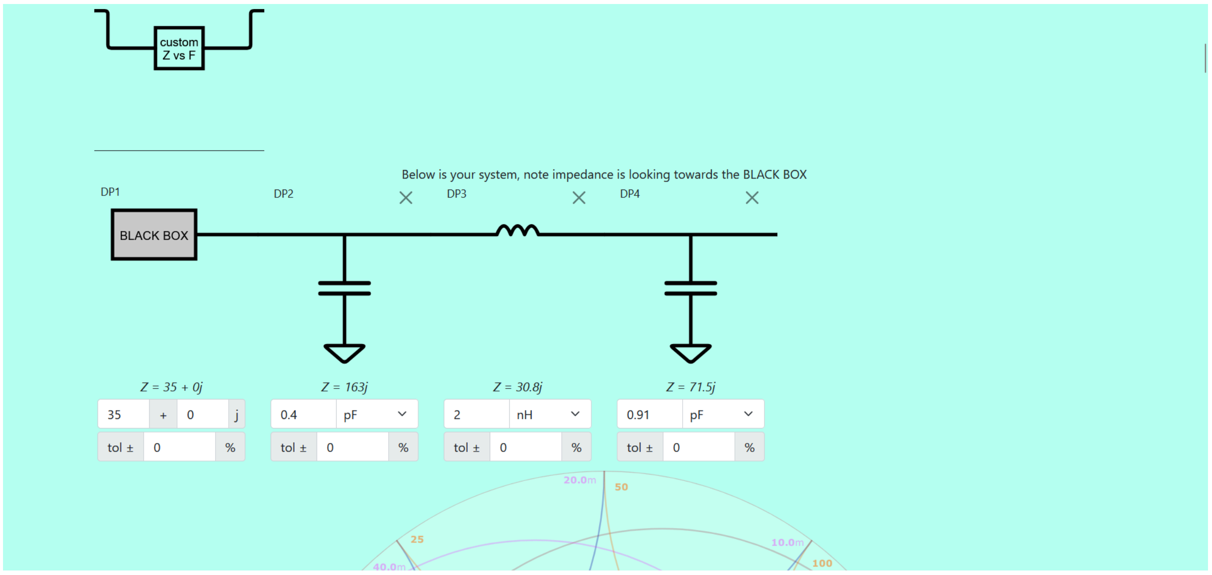


Figure 20: GPIO0, GPIO19 and GPIO20 connections.

5.6. Antenna

We spent a lot of time matching the impedance with the ESP chip. If we do not do that, the signal effectiveness will be too great of a loss, which affects its transmission. Therefore, we use a Smith chart to figure out what circuit, as well as what component would be suitable to match it. The impedance from the LNA_IN pin is 35+0j, whereas the impedance of the antenna is 50+0j.



35 + 0 j	tol ± 0 %	
0.4 pF	tol ± 0 %	
2 nH	tol ± 0 %	
0.91 pF	tol ± 0 %	

Figure 21: circuit simulation

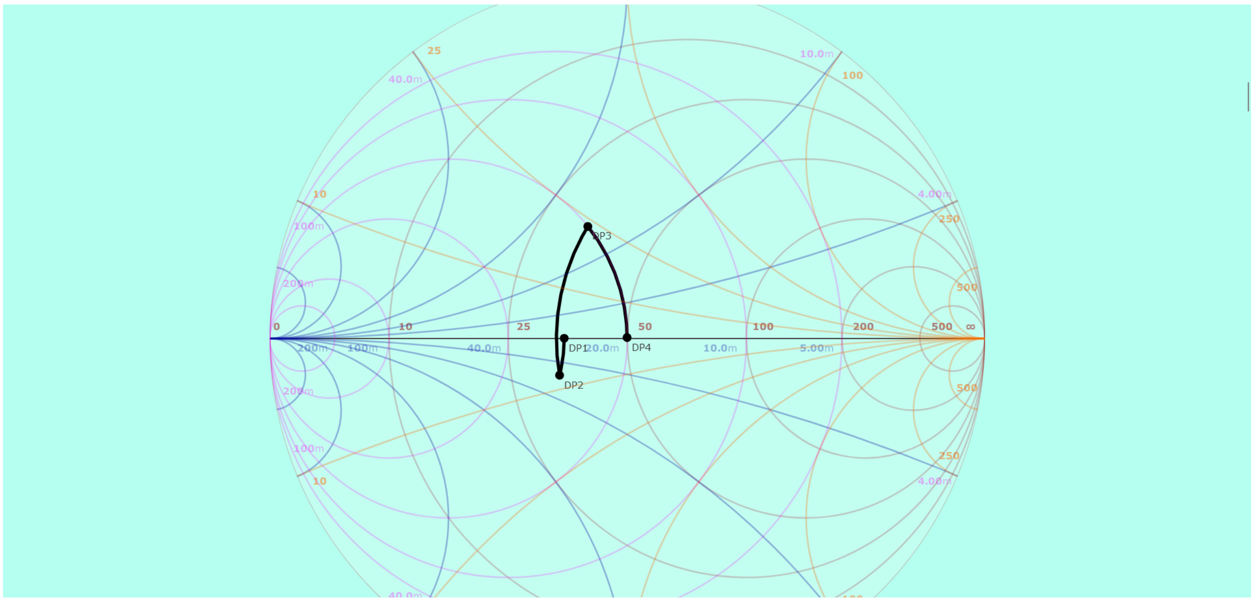


Figure 22: impedance changing through different components

The aim of a Smith chart is to reach the middle part(50 +0j) so that the impedance matches. We then added as much via holes as possible surrounding the wiring of the antenna. We also made sure that none of the components nearby affected the signal.

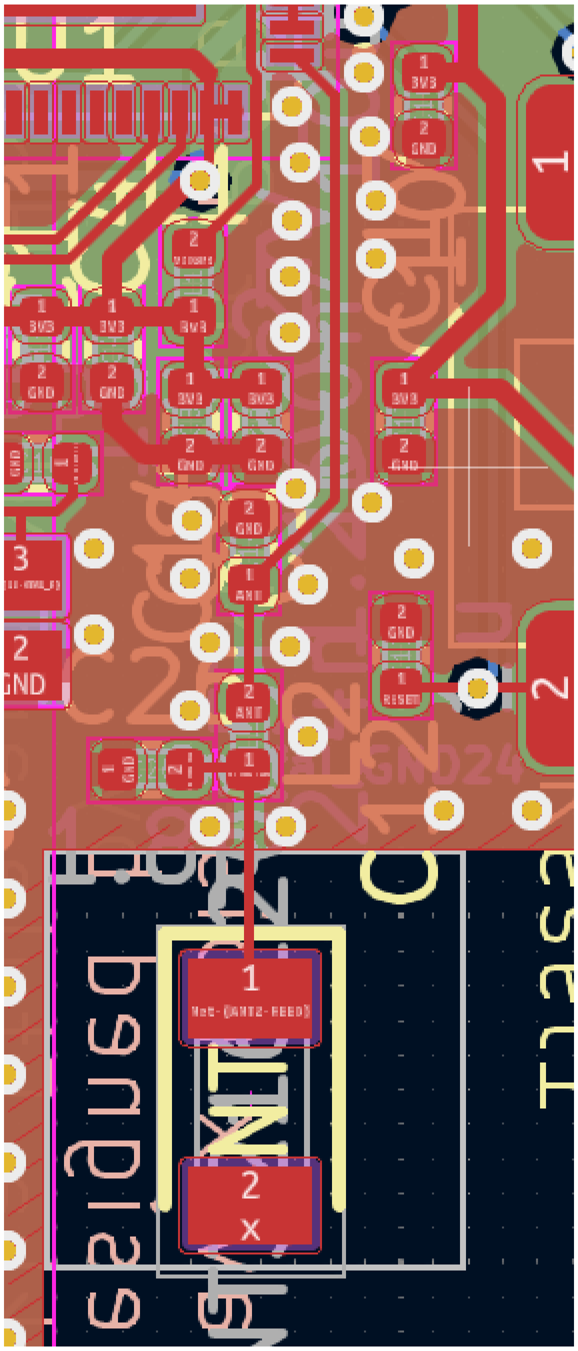
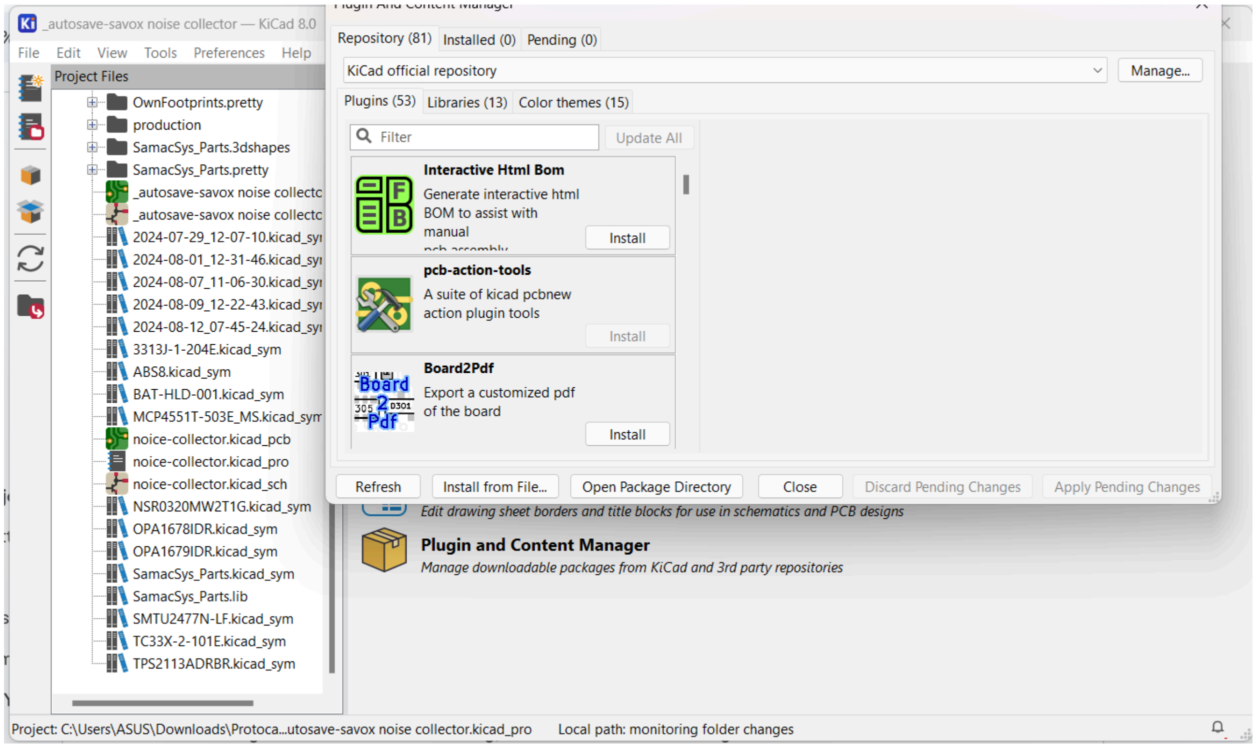


Figure 23: via holes surrounding the wire to the antenna to ensure no effects on signal transmitting

Antennas should be placed on the edge of a PCB to ensure the least amount of interference caused by other components.

5.7. Ordering

Ordering the PCB took a lot of effort. We decided to order our PCB from JLCPCB, as they will assemble all the components for us. Although they have components and footprint, we could not extract the footprints from EasyEDA into KiCad. The file .elibz that contains both symbol and footprint, when transferred into KiCad(only version 8.0 or above can read it), transports only the symbol. The footprint cannot be extracted. Therefore, we usually have to look for the components in other sites such as Digikey or Mouser and then see if JLCPCB has it or not. Most of the crystals we found were available on other pages but not JLCPCB. If we manage to find one available in other pages, JLCPCB is out of stock for it. For microphones, it was even more frustrating. What we wanted is a MEMs analog microphone, but the search bar does not separate them. In fact, they barely show which one is digital, which one is analog until we open the information page of that component. When I found a good MEMs analog microphone on Digikey, JLCPCB did not have it. This combined with the crystal finding is a battle on two different fronts.
After we finished trying to find the suitable crystals and microphone on JLCPCB, we gave up on trying to find the exact footprint that is available online (mostly on SnapEDA), and made one on our own.
We then figured out since we are going to order a rigid PCB, but we were having flex PCB settings, we had to change the settings. Lucky for us the requirements for rigid PCB was not as strict as flex PCB, we did not have to reposition anything much.
Then, we downloaded the JLCPCB plugin for BOM file generator (GND via hole generator as well so that the antenna will not be affected much), and got ready to order it.



OwnFootprints.pretty
production
SamacSys_Parts.3dshapes
SamacSys_Parts.pretty
_autosave-savox noise collecto
_autosave-savox noise collecto
2024-07-29_12-07-10.kicad_syı
2024-08-01_12-31-46.kicad_syı
2024-08-07_11-06-30.kicad_syı
2024-08-09_12-22-43.kicad_syı
2024-08-12_07-45-24.kicad_syı
3313J-1-204E.kicad_sym
ABS8.kicad_sym
BAT-HLD-001.kicad_sym
MCP4551T-503E_MS.kicad_sym
noice-collector.kicad_pcb
noice-collector.kicad_pro
noice-collector.kicad_sch
NSR0320MW2T1G.kicad_sym
OPA1678IDR.kicad_sym
OPA1679IDR.kicad_sym
SamacSys_Parts.kicad_sym
SamacSys_Parts.lib
SMTU2477N-LF.kicad_sym
TC33X-2-101E.kicad_sym
TPS2113ADRBR.kicad_sym

Interactive Html Bom
Generate interactive html BOM to assist with manual

pcb-action-tools
A suite of kicad pcbnew action plugin tools

Board2Pdf
Export a customized pdf of the board

Edit drawing sheet borders and title blocks for use in schematics and PCB designs

Plugin and Content Manager
Manage downloadable packages from KiCad and 3rd party repositories

Project: C:\Users\ASUS\Downloads\Protoca...utosave-savox noise collector.kicad_pro   Local path: monitoring folder changes

Figure 24: Plugin download on KiCad

However, the final page required fixing the position file, as some components are rotated the wrong way, or in the wrong position. Since we have to order it as fast as possible, and changing the position of the component without being able to tell where it is relative to other components, it was a heavy mental burden for us. We also found some capacitors that were out of stock with that amount of capacitance, so we had to change the footprint in the PCB, and we had to tweak some other components as well, but we managed to pull it through and submitted it.

6. Physical

We thought we needed to maximize the amount of space as much as possible, so we were intending to use flex PCB. After wiring, the PCB is so small that rigid PCB would be fine. All that needed to be finished is the casing.

The battery size was a major issue, as it affects whether the device fits in the headgear or not. We had our eyes on CR2477 as it has the best power density, but it was too big. We later decided to go with CR2032, as it is a lot smaller.

The casing is needed so that it would be waterproof from the sweat of the users.

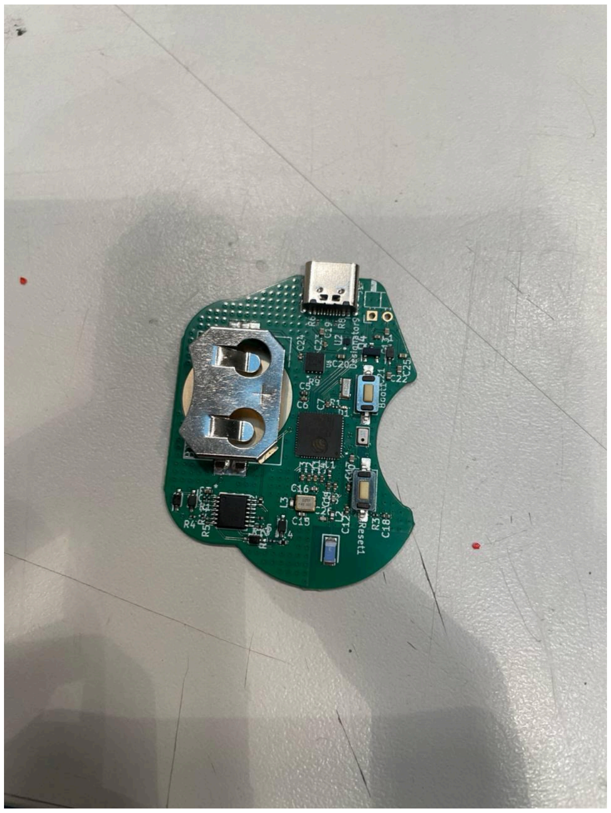
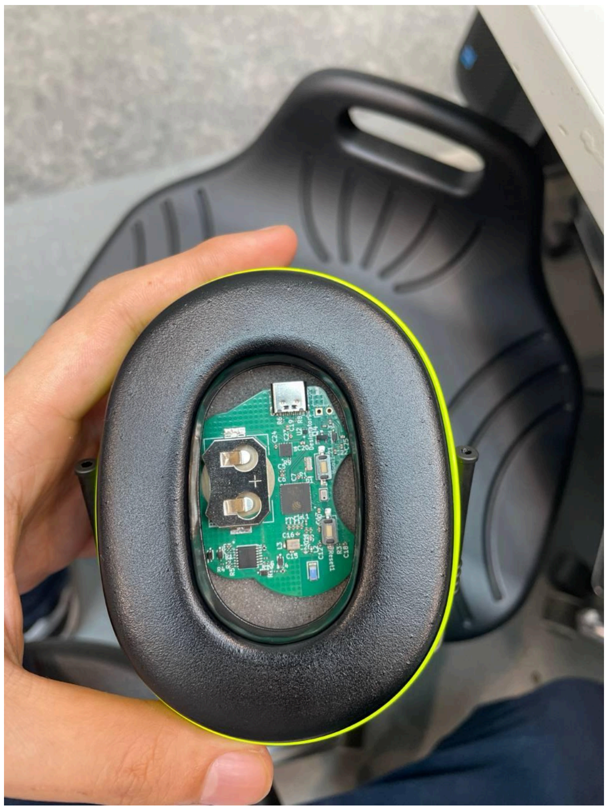



Figure 25,26: The PCB and how it would fit within the protective gear

The PCB overall look is unorthodox, but it is designed with intent. The concave part on the right is a signifier that the user can take it out that way. The edges are curved so that it will not cause any unwanted cuts. The top part is flattened as a USB-C connector would prefer good connection if the board itself is not curved at the top.

As one can see the size of this PCB, it is quite small compared to the muffler. Therefore, we designed a snap fit case because sliding is hard to design, whereas latches are too fragile. Threaded inserts are too long as well.

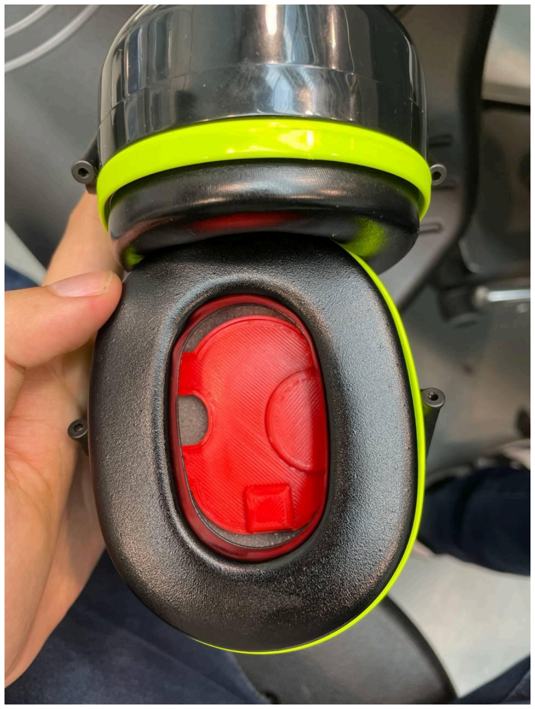


Figure 27:The PCB as well as the case fit inside the muffler

However, due to changes in battery usage, the overall design for the PCB changed dramatically. However, it was not that much of a change that would make the design not fit within the protection gear. So we just made extra space for the battery. Most of the design remained unchanged.

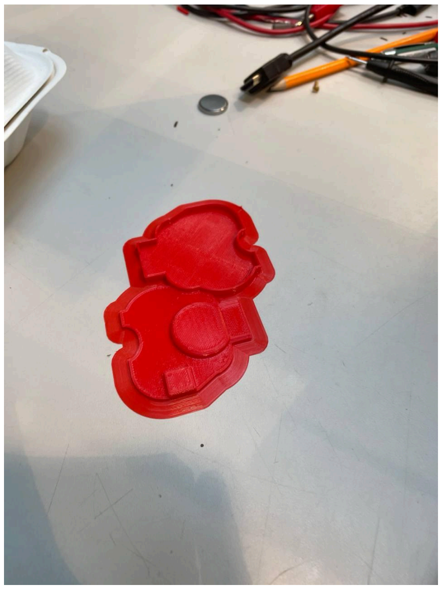


Figure 28: The final PCB design.

7. Reflection of the Project / Projektitoiminta

The phase of the project is followed until the prototyping phase, as we underestimated how long it would take to design the PCB. Therefore, the test phase had been pushed back a lot. We did not follow the work packages, since one of our team members decided to not follow through. It also turned out as we expected that the physical aspect of the device does not have much work to be done. Most of the work lies within the embedded part, and there are some changes in roles of who is responsible for that part.. We also underestimated the amount of coding that needs to be done. The amount of hours spent per week was quite close to estimation.

For all of us, it can be said that this project has given us a lot of experiences in aspects that we are not familiar with. This project feels like it pushes us out of our comfort zone to gain knowledge on elements that we would have known existed before. It definitely increased our skills, and made us more confident with how we turned out to be.

7.1. Reaching objective / Tavoitteet saavuttaminen

We had to change from using deep sleep mode to Ultra Low Power mode because deep sleep would require us to use a peak detector circuit, but the circuit does not keep the level of voltage long enough. In other words, the signal decays. If we changed the design of the circuit, then the detector cannot record signals that change too fast. Thus, we decided to not use deep sleep but ULP to record the data in low power instead. Besides that, most parts of the project are completed, and there were barely any adjustments in objectives. The expected outcome is mostly achieved as well, so it can be said that the project is completed.

7.2. Timetable / Aikataulu

The timeline we put in the plan template was poorly thought out, so we have expected that. We also expected that it is very hard to follow the plans and schedule 100% of the time, as there are too many sudden aspects that change the course of the plan. The workload for designing the physical appearance of the product is overestimated for 2 people, but is just right for one person.

The embedded workload was overwhelmingly underestimated. The server workload was also underestimated as well.

7.3. Risk analysis / Riskianalyysi

We did not foresee that one of our members would discontinue with the project. That means there would be more work needed to be done by 4 members. For financial risks there were not many. We overestimated that we might run out of budget. However, because of that, the risk of the final product not working, as there is no prototype, is very high. We had overlooked such a risk. The task difficulty underestimation was expected. The insufficient amount of subtask risk was quite unnecessary, as when a problem arises, we just solve it and move on to the next objective. Work distribution was something we did not expect, as some parts of the project needed extra time to take care of, whereas others aspects are easily accomplished. We also did not predict that communication would be such a large issue that we have to deal with. At the beginning, the lack of communication in our team was quite detrimental.

7.4. Further improvements / Riskianalyysi

We have issues most of the time with the battery. The project can be improved by having 3 1.5V cell batteries in series, or use a larger voltage coin battery. The size might be bigger but it is a suggestion worth considering. The crystal definitely needed to be swapped out.

8. Discussion and Conclusions / Yhteenveto ja johtopäätökset

Ilkka Etula:

I learned to use the KiCad software and that it takes very long to design a functional PCB with it. I also learned that I can not solder surface mount components very well. The latter one might be partially because the soldering of those components was in a hurry to fix the PCB and they had to be connected either off the solder pads (capacitors for crystal) or on a too small solder pad (resistor for power circuitry). I also learned more about working in a group.

Hong Minh Nhat:

I was reminded of how to use KiCad, read a lot of documents and datasheets, and learn deeper about electrical engineering, such as diode drops voltage, therefore there is something called an ideal diode, reminding me what nMOSFETs and operational amplifiers are, soldering surface mounts, etc. That makes me feel a bit more confident as an electrical engineer. I also learned a lot as a manager for the team, and tried my best to do the best and the most that i can. As a physical designer, I never had to design something that aimed to be as small as possible, so this is also a new experience for me too.

Iiro Laakso:

I learned how to use visual studio code and how to code using it. I learned the basics of how html, css and javascript code works and what kind of things you can do with it. I learned how to change visuals with css and how to give elements functionality in js. I learned how libraries work and how you can implement them to your code. I learned how to transport your code into the server to be read and how you can make your frontend code receive information that backend code receives. I also learned how to make js code able to save files in it to a user's computer and how to code specific visuals to servers buttons and text by using css.

Ngo Quoc Viet:

I learned how to configure the ADC of the ESP32 to read analog signals from a microphone. This involved setting up the correct ADC channel, adjusting the attenuation for better signal resolution, and using logarithmic scaling to process the raw data into decibel levels. By writing and fine-tuning the code for the ESP32, I was able to accurately capture the ambient sound levels and convert them into meaningful decibel readings that could be transmitted for further analysis. I explored how to set up a WebSocket server using Node.js to handle real-time data streaming from the ESP32. This allowed for continuous and efficient transmission of decibel readings to connected clients without the overhead of repeated HTTP requests.

List of Appendixes / Liitteet

List here Appendixes you attach to PDF (or as separate documents).

You may consider putting these as Appendixes

- Project plan (as is)
- User manual of your technics
- Large drawings

- …

References / Lähteet

Circuitjs:

https://www.falstad.com/circuit/circuitjs.html

Smith chart generator:

https://www.will-kelsey.com/smith_chart/

32.768kHz crystal:
https://jlcpcb.com/partdetail/TXCCorp-AH03200003/C337577
Antenna:
https://jlcpcb.com/partdetail/Walsin_TechCorp-RFANT3216120A5T/C127629
Microphone:
https://jlcpcb.com/partdetail/Linkmems-LMA2718T421_OA52/C7587901
Switch:
https://jlcpcb.com/partdetail/Bzcn-TSB008A2530A/C2888954
Chottky Diode:
https://www.mouser.fi/ProductDetail/onsemi/NSR0320MW2T1G?qs=ZXBb0xZ9WeD3Qiaxi5ZC9w%3D%3D
Or gate:
https://jlcpcb.com/partdetail/TexasInstruments-TPS2113ADRBR/C354512
Battery holder:
https://jlcpcb.com/partdetail/Myoung-MY_203208/C964831
40MHz crystal:
https://jlcpcb.com/partdetail/361523-S3D40000000B20F30T/C387435
Voltage regulator:
https://jlcpcb.com/partdetail/TexasInstruments-TPS7A0233DQNR/C5220177
Op amp:
https://jlcpcb.com/partdetail/3peak-TP6004TR/C248577

Adafruit Py ESP32S3 design:

https://learn.adafruit.com/adafruit-qt-py-esp32-s3?view=all#downloads

ESP32S3 datasheet:

https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf

PCB manufacturing settings:

https://jlcpcb.com/capabilities/pcb-capabilities

ESP32S3 guidelines:

https://docs.espressif.com/projects/esp-hardware-design-guidelines/en/latest/esp32s3/index.html

Peak detecting circuit design:

https://www.youtube.com/watch?v=jllsqRWhjGM&t=1088s

chart.js implementation:

Installation | Chart.js (chartjs.org)

deploying code to the server

Communication with the game controller and deployment - ELEC Prototyping Exercises - Aalto University Wiki

Surname, F. 2010. Developing gadget systems. Transactions of Gadgets Vol. 53(1), pp. 34-45.

Surname, F., Sukunimi, E. & Teekkari, T. 2010. Developing gadget systems. Transactions of Gadgets Vol. 53(1), pp. 34-45.