UMEÅ UNIVERSITET

DESIGN OF AUXILIARY COMMUNICATION FOR AUDIO BETWEEN COMPUTERS AND DSPS

Programming and optimization of computational resources

DESIGN AV LJUDKOMMUNIKTION MELLAN DATOR OCH SIGNALPROCESSOR

Programmering och optimering av beräkningsresurser

Oscar Eriksson Janze

Examensarbete, 15 HP
Högskoleingenjörsprogrammet i Elektronik och datorteknik, 180 HP
Vt 2023

Preface

First, I want to thank ”Hörselskadades Riksförbund (HRF), Hörselforksningsfonden,” for making this thesis project a reality. This has been a great thesis project where I could encapsulate my knowledge and evolve it.

I want to thank Amin Saremi for giving me the chance to work with Umeå University on this research project. I want to thank my colleagues Eric, Pamela, Joel, and Omar for being good colleagues, giving input and making the days always something different.

Abstract

This thesis report is about designing a prototype and establishing audio communication between a computer and Digital Signal Processor (DSP) using two preamp circuits using both auxiliary and USB connection. The paper gives the reader an overview on how audio is transmitted from a computer, through the system and to the desired output. The reader should also get a better understanding of how an AD-converter samples the incoming signal to the Discrete plane and how an AUX or phone connector works. This information can be used for designing preamp circuits to communicate between a computer and the DSP. The DSP circuit uses an STM32 processor to control the incoming and outgoing signals with the use of ADC and DAC conversion. The DSP also uses microphones to capture surrounding sound. An addition is to make a prototype on how to use these microphones to send the signal upstream to the computer. The microphone is then benchmarked with the use of Matlab, calculating Total Harmonic Distortion. Management and optimization of code structure and resources is done in the source files of the project. Using imperative C programing, large functions are broken down into smaller functions to ease readability and control flow. The result is a prototype circuit that can communicate audio signals with both audio jack and USB between computers input and output to the DSP. Using CubeMX in conjunction with CubeIDE to add additional ADC channels to be able to incorporate an automatic source control when the audio jack or USB is connected.

Sammanfattning

Denna avhandling handlar om att designa en prototyp och etablera ljudkommunikation mellan en dator och en digital signalprocessor (DSP) med hjälp av två förstärkarkretsar genom både aux- och USB-anslutning. Rapporten ger läsaren en översikt över hur ljud skickas från en dator, genom systemet och till önskad utgång. Läsaren bör också få en bättre förståelse för hur en AD-omvandlare fungerar för att sampla den inkommande signalen till tidsdiskreta planet och hur en AUX- eller telekabel fungerar. Målet är att använda denna information för att skapa förstärkare som kan kommunicera mellan en dator och DSP:en. DSP-kretsen använder en STM32-processor för att hantera de inkommande och utgående signalerna med användning av ADC- och DAC-omvandling. DSP:en använder också mikrofoner för att fånga omgivande ljud. Ett tillägg är att skapa en prototyp för hur man kan använda en av mikrofonerna för att skicka signalen till datorn. Mikrofonen jämförs sedan med hjälp av Matlab genom att beräkna total harmonisk distorsion. Hantering och optimering av kodstruktur och resurser görs i projektets källkodsfiler. Genom att använda imperativ C-programmering bryts stora funktioner ned i mindre funktioner för att underlätta läsbarheten och styrningen av flödet. Resultatet är en prototypkrets som kan kommunicera ljudsignaler med både ljuduttag och USB mellan datorns in- och utgång och DSP:en. Genom att använda CubeMX tillsammans med CubeIDE läggs ytterligare ADC-kanaler för att möjliggöra automatisk källkontroll när ljuduttaget eller USB-anslutningen är ansluten.

Contents

1. Introduction

In 2019, one in five people worldwide had some form of hearing impairment. In Sweden, about 18% of the population has some type of hearing loss, which can be caused by factors such as age, work-related injuries, and other ear disorders.

Hearing aids are a way to combat hearing loss, but they may not provide the best sound quality when listening to music, attending meetings, or using the phone. On the other hand, it can cover more area and allow for larger components, which can improve the quality of hearing aids. More sophisticated algorithms can be used to process audio signals from different devices or in real-time.

A device that can manage audio signals is a Digital Signal Processor (DSP) unit. DSP units are specifically designed to manage specific tasks such as Fast Fourier Transform, finite impulse response (FIR) filtering, and other functions commonly used in signal processing. With this device, we can create several different algorithms that can enhance a person's audio profile, such as filter banks and noise reduction.

Analog-to-digital converters (ADCs) are used in most digital circuits and allow analog signals to be sampled for signal processing. The frequency, amplitude, and resolution of the ADC are important for collecting analog information accurately. For example, in the case of speech, a sampling frequency of 16 kHz or higher is desired, while music requires more than 32 kHz. The resolution of the ADC provides a more accurate reading of the signal.

Project FA21-0017 [1], from Hörselforksningsfonden is described as “smart headphones that can adapt to the user’s unique hearing profile”, involves a DSP-platform which includes microphones, amplifiers, filters, a microprocessor with DSP, Bluetooth, and I2S. The platform is powered by a USB cable but will be powered by a battery. The microprocessor of the platform is the STM32F7/STM32F4, which has a DSP unit and a floating-point unit (FPU) built in. The STM32 is based on a 32-bit ARM architecture, which is a reduced instruction set computing (RISC) architecture. The advantage of working with a RISC architecture is that the processor has a smaller instruction set but more general-purpose registers that can perform several instructions per cycle. This means that the processor is a cheaper and more energy-efficient solution compared to the x86 instruction set.

The current task of the project is to develop a prototype that takes the signal from a computer's sound card output and implements it in the system with ADC conversion. The prototype also includes designing a microphone circuit for a computer's audio input. The processor should be able to detect the different devices and prioritize them accordingly. The prototype has primary signal processing algorithms with filter banks programmed as FIR filters. However, there is a delay in the algorithm, which causes the sound the person hears to be delayed. The algorithm's goal is that it must be optimized for a delay of fewer than twenty milliseconds. This will involve investigating code closer to the hardware, managing, and optimizing a new code for filter banks, and reducing the delay in the algorithm. This part will be developed in parallel with Eric [10], who will implement a new infinite impulse response (IIR) filter algorithm.

Several problems arise, such as how we use this device between various sources such as music, calls, meetings, or hearing aids. How do we take a computer's audio output to an external circuit? How do we connect a microphone to a computer? How do we configure the ADC channels on the STM32 and find more sources?

2. Theory

Signal processing is a subfield in electrical engineering with focus on analyzing, modifying, and incorporating a sample into something applicable. In signal processing, the use of applied mathematics, samples, and sampling is used to break down a signal in various categories. Such categories being analog, digital, non-linear, continuous time signal or discrete time signal. In analog, continuous, and nonlinear signal processing, electronic circuits are commonly used for filters and mixers. The purpose of filters is to isolate frequencies that contain noise and refine the desired ones. Digital signal processing and discrete time signal processing is used when a signal is sampled within a discrete time-domain. Continuous time signals such as sinusoidal waves functions contain infinite data for a set amount of time. When combining signal processing and computers, we are restricted by the number of samples, resolution, and the size we can do in a discrete-time period.

2.1. Analog to digital converter Interface

Analog to digital signal converters or commonly known as AD-converter or ADC is a physical approximation of the ideal continuous-time period to discrete-time period [2]. By using a combination of voltage dividers, comparators, logic-gates, and decoder the circuit converts the analog sample to a digital sample. A clock signal is applied to the logic-gates to update the current value of the analog signal. If the process is updated in an interval, it is called sampling rate. This can then be used by the computer to read the actual value of the signal.

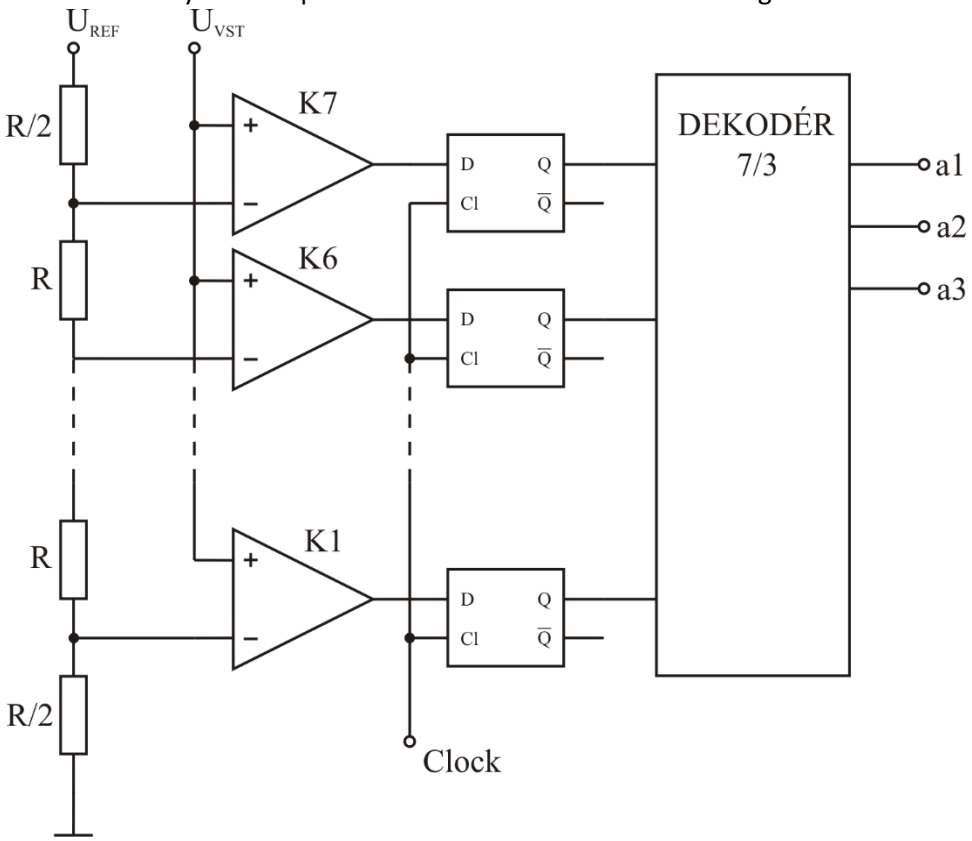

Figure 1 - AD-converter Wikimedia Commons.

Figure 1 shows an AD-converter with the use of previous mentioned components. In many cases of AD-conversion of 12-bit resolution is sufficient for reproducing an original audio signal. To set the desired sampling rate on an STM32 processor calculation (1) is needed:

$$\frac{Clock\ peripheral}{Desired\ frequency} = Counter\ period - 1 \qquad (1)$$

When digitalizing a signal, the error Quantization noise as a function of bit depth is also a common issue in AD-conversion. The error comes from an issue of rounding between analog input and digitalized values. The equation of this error is formulated in equation (2):

$$SQNR = 20 * log_{10}(2^Q) \approx 6.02 * Q\ dB \qquad (2)$$

Where Q is the number of quantized bits. The STM32F4 and STM32F7 both come with 3x12-bit AD-converters equipped on the processor itself. This gives us SQNR of $6.02 * 12 \approx 72.24\ dB$ which means that the noise floor is 72.24 dB lower than the signal. Given that our audio chain can produce a 90 dB SPL at maximum then the noise floor in this scenario will be 90 dB SPL – 72.24 dB SPL = 18 dB which is barely noticeable. The number of samples or frequency samples is also an important part of sampling. When sampling a signal, the frequency needs to be at least two times the original signal to reproduce it in memory [2]. Having less samples than the original signal will create aliasing and other distortions.

2.2. Computer Communications

Computers communicate with digital signals between internal devices using addresses, controllers, and data buses. See figure 2 for an overview on how the bus system works inside a computer.

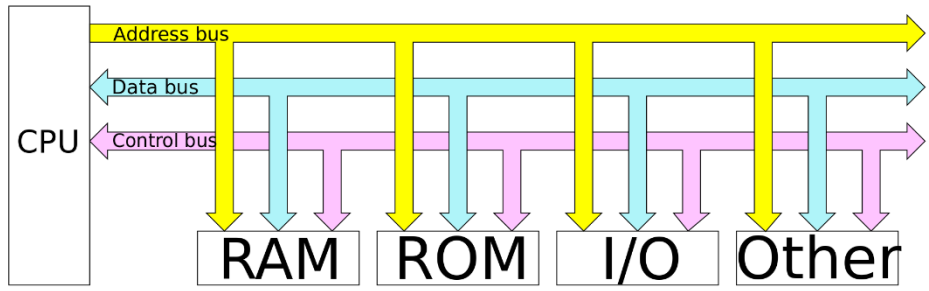

Figure 2 - Computer bus system Wikimedia commons.

On regular desktop computers, if the computer is used to play music or prompt a sound, then the CPU sends that data to the audio device. When computers need to communicate with the outside world, they need both Digital-to-analog conversion (DAC) and ADC to send and receive signals. On a computer, the memory controller reads and stores information in the system memory. If a sound is prompted, the memory controller unit sends that information to the audio controller. Communication is then transmitted on the system bus or PCI bus to and from the audio controller. Pulse-code modulation, Wavetable synthesis and Frequency Modulation synthesis were used by older computers to create sound, while modern computers use AC'97 codec or Intel's High-Definition Audio Controller. The audio controller, with help from direct memory access (DMA) talks to encoders and decoders from the users I/O interface.

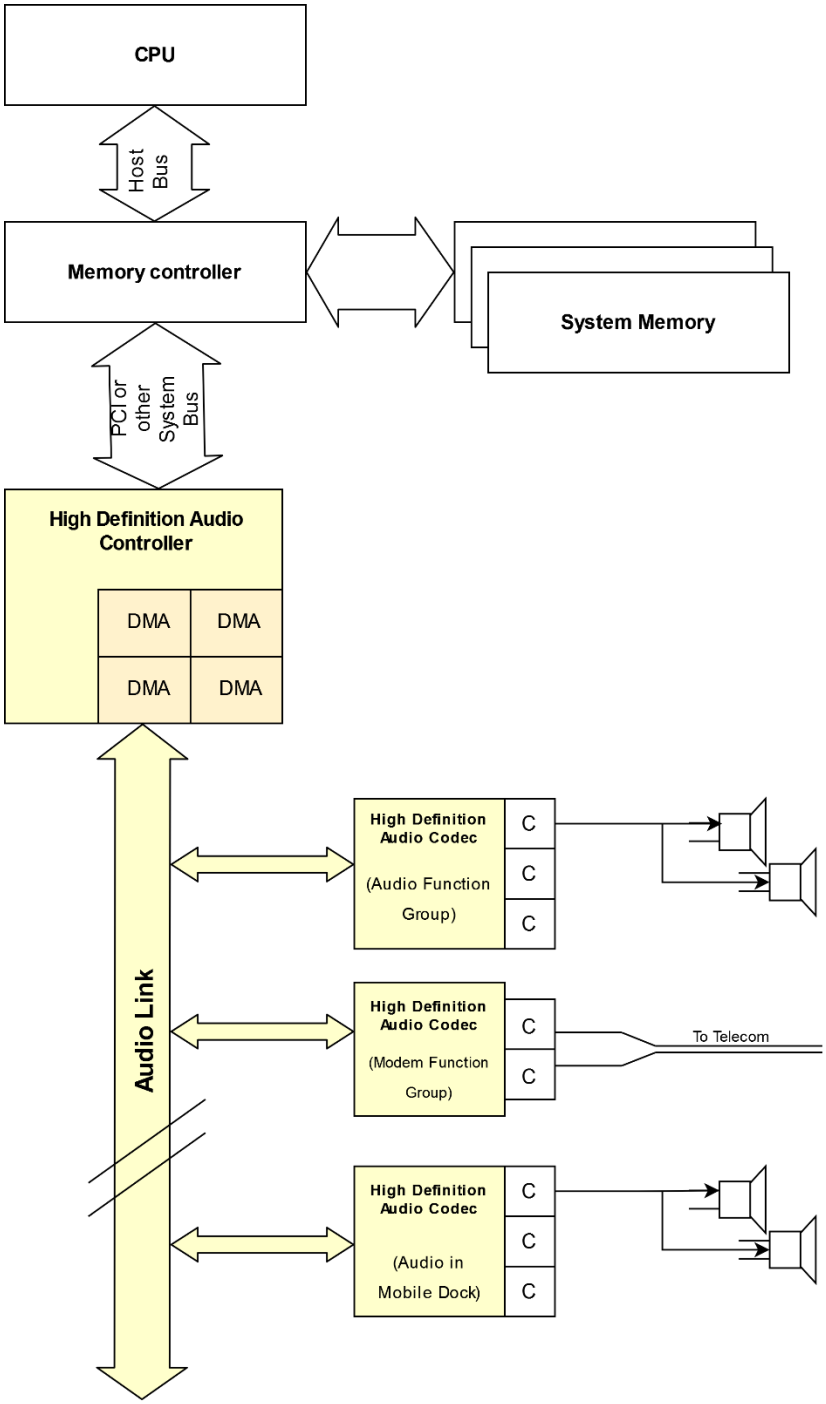

Figure 3 - Intel HD audio hardware system overview. Reproduced from datasheet [3].

Figure 3 shows how the HD audio specification communicates with the memory controller and CPU of the computer. The signal is transmitted via the system bus to the audio controller and creates an environment with fast communication between the computer's memory and the outside world. Connecting to and from the audio I/O interface is standardized to auxiliary cables between the computer and the external device being either a microphone or speaker system. Most common is the TRS setup for both microphone and speaker setup. In the High-Definition Audio Specification we can find the Audio Function Group of the so-called widgets [3]. The Audio input converter widget supply a voltage reference for microphone support.

Microphones are transducers that convert sound to an electrical signal. Inside microphones, a membrane or diaphragm is moved depending on the mechanical properties of a soundwave. This movement is based on the frequency of the sound. Microphones come in different forms such as condenser, ribbon, electret and more recently MEMS. Although they share the same principle of capturing sound waves they differ in their internal structure. With the evolution of technology in both sound and mobile devices, the requirement for smaller components is therefore needed. In the industry of mobile devices and audio it is more common to see either electret or MEMS microphone for their smaller sizes.

2.3. Phone Connectors and audio transmission

Phone connector, TRS, audio jack or AUX consists of an auxiliary cable with contact points in tip, ring, and sleeve. This format was invented in the 19th century for telephone switchboards but has evolved into many diverse types of connectors. The most common on digital platforms today is the 3.5mm audio jack, found commonly on phones and computers. The cable is commonly used to send either composite video, simple analog signals, or custom signals. Depending on the cable connection we can create either a mono or stereo transmission. In mono transmission we only send the signal as it was intended to come from one source such as a microphone. In stereo we send signals from two sources being left and right. The sibling of the TRS connector is the TRRS connector. TRRS follows the same principles with an added ring. TRRS is commonly found on headsets with an added microphone. There are two versions of TRRS, where the last two rings become either a signal for ground or the microphone called CTIA or OMTP. These two standards emerged in times of different manufacturing needs from both Apple and Nokia but have since been standardized to different devices on the market. In later years, Bluetooth has been a leading technology when communicating with other devices and seeks to replace this standard. Though the TRRS phone connector standard is slowly being replaced in favor of Bluetooth, it is still possible to find manufacturers keeping, removing, or adding the port after market needs.

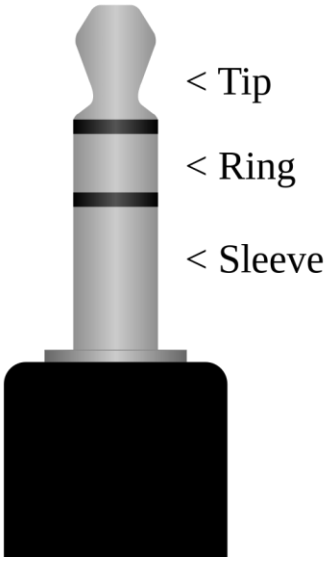

Figure 4 – Tip, Ring, Sleeve, plug Wikimedia commons.

Figure 4 shows a phone connector (TRS) which is most common for computer auxiliary communication.

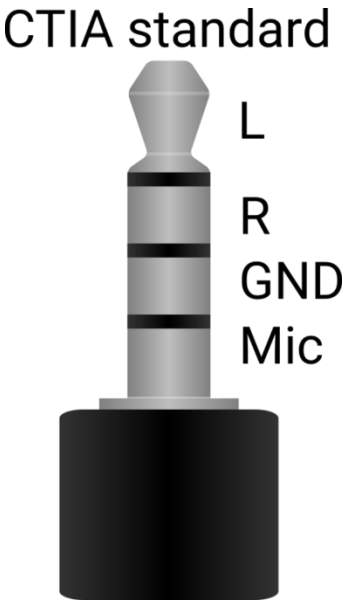

Figure 5 – Tip, Ring, Ring, Sleeve plug (CTIA) Wikimedia commons.

Figure 5 shows a TRRS phone connector with a stereo interface and microphone setup.
In audio processing it is common to keep signals balanced with a negative and a positive signal. In the case of computers, to reduce the number of cables needed, the signal needs to be balanced on a single line at a bias point with a maximum bias voltage of +2.5V and a max of 5V. The output voltage and reference voltage may vary between different soundcard devices. What needs to be done here is to find a way to take the output and convert it to the DSP-circuit using ADC. Using the same theory, we can also use one of the DSP-circuit microphones and connect it to a computer's input. The platform uses a 3.3 voltage common collector. Voltage bias for signals used will be half of that, being +1.65V.

An alternative route for audio signals is USB. USB, however, uses a driver stack to communicate with the system bus or PCI bus. USB has its own advantages by being able to supply external devices with power but also comes with the disadvantage of needing specifications on what the device is supposed to do and delay. The USB protocol uses a differential pair of negative and positive signals to communicate with other devices. Therefore, a device is needed that can communicate with the USB stack. This could be done on the STM32 processor but would then reserve necessary space and algorithmic time. The solution is to use a stereo audio codec that can communicate with the computer and be recognized as a sound device.

3. Method

This project will contain two solutions for audio signal processing and transmission to the DSP-unit with both TRS/TRRS and USB-type C. Initial circuitry is dimensioned and assessed on breadboard before implementation of PCB design.

To start with the project, circuits and components were searched on the internet to get a broad understanding of how different preamplifiers were built. Since much of sound design is about containing frequencies and amplifying signals in different frequency bands, this design will only need to amplify the computer's output since most of the filtering is done by the DSP. Although the amplifier still needs to remove certain frequency components from the upper frequency bands and the extreme low frequencies. Limits on what frequencies are useful for a human span from 15 Hz to 20 kHz. Often in audio design filter frequency limits are closer to 10 kHz since much of the higher frequencies are not audible.

3.1. Frequency response

The platform is using oversampling which means a closer inspection of frequency response is needed. The STM32 processor has a nominal sampling rate of 88 kHz which is four times higher than the actual audio sample rate of 22 kHz. When designing anti-aliasing filter, we need to design filters that adequately attenuate the aliasing components. The following equations are for the frequency response:

$$H(j\omega) = \frac{1}{1+j(RC\omega)} \quad (3)$$

$$|H(j\omega)| = \frac{1}{\sqrt{1+R^2C^2\omega^2}} \quad (4)$$

Equations (3) is the frequency response and equation (4) give the amplitude of the frequency response. R and C should be chosen to have a cutoff frequency around 10 kHz. This means that the first alias will be 78 kHz (i.e., 88 kHz – 10 kHz). Therefore, we chose R to 180 ohms and C to 100 nF which gives an attenuation of 0.1 at 78 kHz. This means that our RC filter dampens the first alias by 20 dB.

When using an AD-converter it is also common to find anti-aliasing filters to remove aliasing. This filter should be designed to sufficiently attenuate the frequencies higher than the Nyquist rate and the theoretical aliasing. The work for the circuit design was broken down into two sections. Using the TRS/TRRS connector and signal amplifiers, the following implementations were created.

1. TRS implementation and TRRS implementation
    a) Microphone
    b) Audio preamplifier
2. USB-type C Implementation

The designs carry the same theory between them. An analog signal comes from the computer's output and sampled by the DSP with the use of ADC. USB communication as mentioned earlier needs an audio stereo codec which can communicate with the USB stack. The codec needs to have both ADC and DAC for it to work in this design. Also, the microphone from the DSP-circuit needs to be able to send a signal to the computer. These are often weak signals and need to be amplified, therefore operational amplifiers are needed take the weak signal and amplify it by a desired gain. The gain is of the signal needs to reach maximum peak-to-peak value when the computers output is at max for a full range of sound. When designing amplifiers with filters, calculations are needed for

the cutoff frequency and the gain of the signal. For the non-inverting amplifier, the equations used are (5), (6), and (7):

$$Gain = 1 + \frac{R2}{R1} \quad (5)$$

$$Fc = \frac{1}{2\pi RC} \quad (6)$$

$$Vref = Vcc * \frac{R2}{R1+R2} \quad (7)$$

The lowpass filter with the given cutoff frequency is used for antialiasing purposes. The type of amplifier used was a frequency dependent operational amplifier circuit using the following schematic (non-inverting) [4]. All schematics was made using Altium Designer for future work.

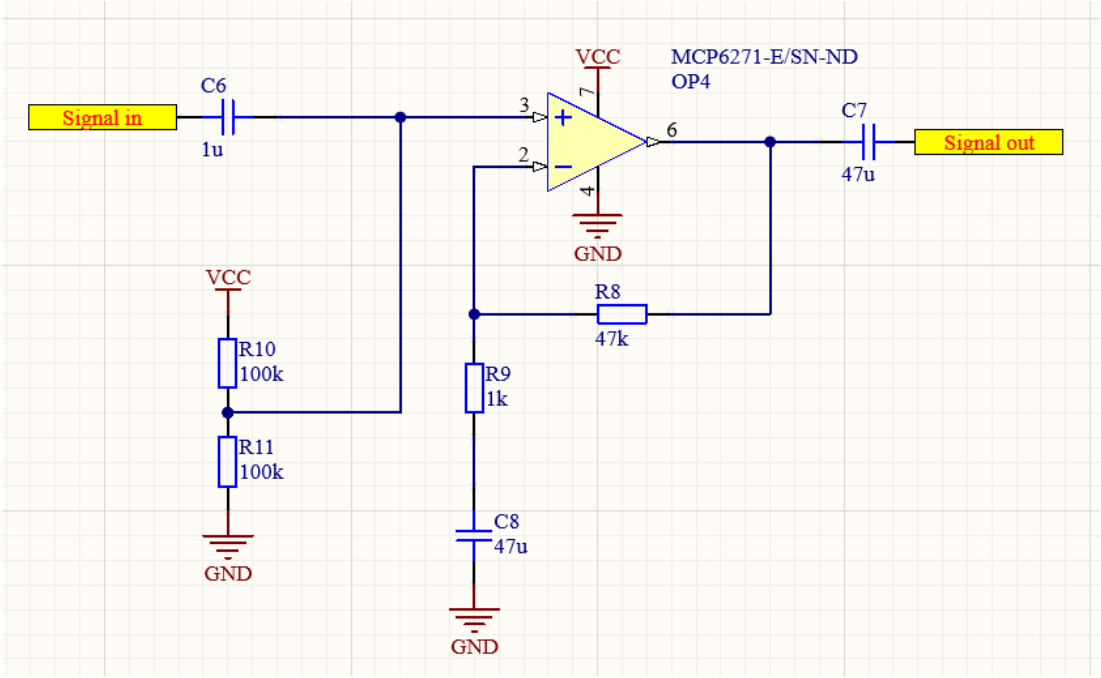

Figure 6 – Frequency dependent operational amplifier.

Figure 6 shows a frequency dependent operational amplifier circuit. The circuit describes an operational amplifier with a DC-block capacitor and the potential raised to half of the common collector on the incoming signal. The signal is then amplified to the desired amount with R1 and R2 and lastly a DC-block at the end. This lets the AC-signal with its voltage and frequency to be uninterrupted. If a capacitor is set parallel to the R8 resistance, then the circuit will also have a high pass filter, which cuts off the higher frequency band and reduces the gain to zero. When using this amplifier for audio signals, the R10 and R11 resistances should at least have a tolerance of ±1% to not induce noise in the signal. This amplifier circuit will be used as reference in all the implementations but will have small adjustments depending on the application.

3.2. Audio jack, TRS Implementation and TRRS implementation

Design of the signal transmission for computer audio is designed for a 3.3V system in mind. Computers audio signals are frequency dependent signals which leads to the use of an amplifier that blocks the DC-signal but amplifies the AC-signal. The same goes for the microphone.

a.) Microphone

The microphone used on the DSP platform is a prebuild circuit from Mikroe called Mic Click [5]. On the circuit itself we can find a MEMS microphone from Knowles Acoustics called SPQ0410HR5H-B [6]. This microphone circuit contains two signal amplifiers with filters for better noise cancellation and antialiasing. From the microphone a DC-block and a bias is applied to the signal with a lowpass filter with a cutoff of 408 Hz. The transduced signal is amplified 169 times and passed to a series resistance with a value of 100k ohms.

What is needed to be done is to design the microphone with an amplifier and filter and pass it on the already biased signal from the computer. The filter dimensions should be specified human-audible frequencies, which means that the desired frequency is between 0 and 10 000 Hz. In the datasheet of the SPQ0410HR5H-B [6], the frequency response curve shows the degradation of the sensitivity when reaching high frequencies of 10kHz. Using again the frequency dependent signal amplifier is then sufficient with a cutoff above 10kHz frequency.

The circuit is later benchmarked with the procedure of playing a test signal from a phone speaker at maximum volume at 5 cm distance, recording the signal on computer software, and using Matlab to calculate total harmonic distortion (THD). The choice of distance is based on the theorized final mounting position of the microphone to the user's mouth in a headset application. The calculation follows Fast-Fourier Transformation, absolute value of the peaks of the test signals recorded harmonics.

$$THD_F = \sqrt{\frac{V_2^2+V_3^2+V_4^2+\cdots+V_n^2}{V_1}} \qquad (8)$$

Equation 8 is used to estimate distortions in an electrical component. The sum of the n-nth harmonics squared, then divided by the fundamental frequency V_1. In this case, the test signal is a 1kHz signal from a phone's speaker. This is later compared with the auditory distortion from the circuit and the datasheet of the MEMS microphone. A tone generator from a mobile phone was used to generate the 1kHz tone. The industry standard of a 1 kHz tone from a mobile speaker is around 90dB using highly accurate measurements. The test signal is then recorded with software such as Audacity and exported as a Wav file. Then the file is imported into Matlab for further analysis.

Matlab code to measure THD:

```
%% Read input and plot
[x,Fs_read]=audioread('lasttest_noamp.wav');
input=x(:,1);

X=fft(input);
le=length(X);
frekvenser=(0:(le/2)-1).*(Fs_read/le);

plot(frekvenser,abs(X(1:length(frekvenser))));
title('Frequency spectrum')
figure,
plot(abs(X(1:length(frekvenser))));
title('Index spectrum')

set_indexes1 = 10000;
set_indexes2 = 16000;
set_indexes3 = 23000;
set_indexes4 = 31000;
set_indexes5 = 39000;
set_indexes6 = 44000;
set_indexes7 = 50000;
set_indexes8 = 60000;

%% Find indexes and calculate the total harmonic distortion
 [V0, index]=max(abs(X));

 [V1,index_1]=max(abs(X(set_indexes1:set_indexes1+3000)));
 index_1=index_1+set_indexes1

 [V2,index_2]=max(abs(X(set_indexes2:set_indexes2+3000)));
index_2=index_2+set_indexes2

 [V3,index_3]=max(abs(X(set_indexes3:set_indexes3+3000)));
index_3=index_3+set_indexes3

 [V4,index_4]=max(abs(X(set_indexes4:set_indexes4+3000)));
index_4=index_4+set_indexes4

 [V5,index_5]=max(abs(X(set_indexes5:set_indexes5+3000)));
index_5=index_5+set_indexes5

 [V6,index_6]=max(abs(X(set_indexes6:set_indexes6+3000)));
index_6=index_6+set_indexes6

 [V7,index_7]=max(abs(X(set_indexes7:set_indexes7+3000)));
index_7=index_7+set_indexes7

 [V8,index_8]=max(abs(X(set_indexes8:set_indexes8+3000)));
index_8=index_8+set_indexes8

%% Total harmonic distortion (THD)
THD=sqrt((V1^2)+(V2^2)+(V3^2)+(V4^2)+(V5^2)+(V6^2)+(V7^2)+(V8^2))/V0
```

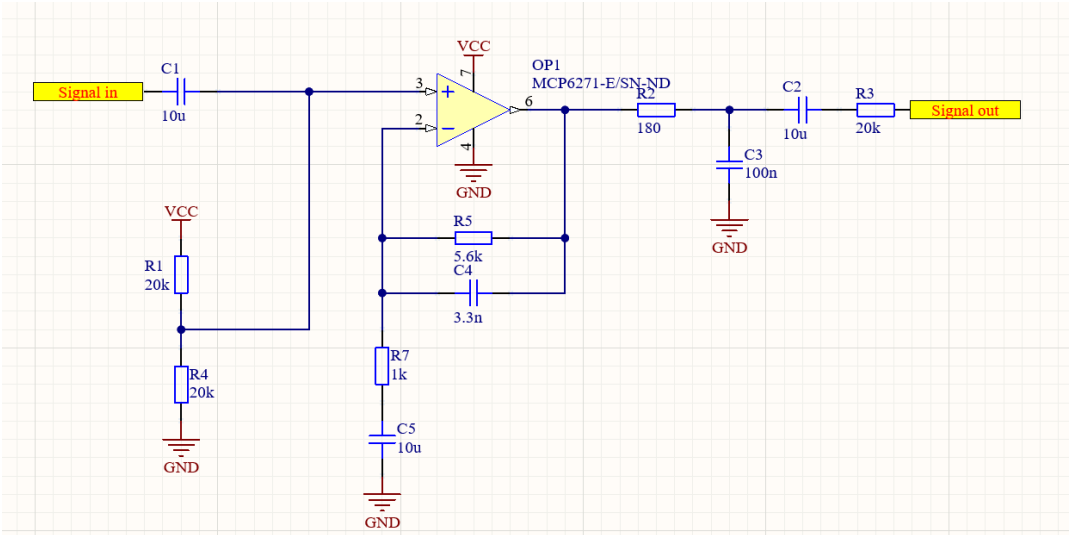

Figure 7 – Amplifier circuit for microphone signal to computer.

Figure 7 is an amplifier circuit used to send microphone input to the computer. The incoming signal from the Mic click microphone is passed onto the circuit. The signal is amplified with a gain of 6.6 and cuts off high frequencies above 8612 Hz. Removing the R5 resistance, replacing it with a cable will result in a non-amplified circuit. An aliasing filter with a cutoff at 8841 Hz, DC-block, and output impedance. The signal is then passed onto the phone connector in the following configuration.

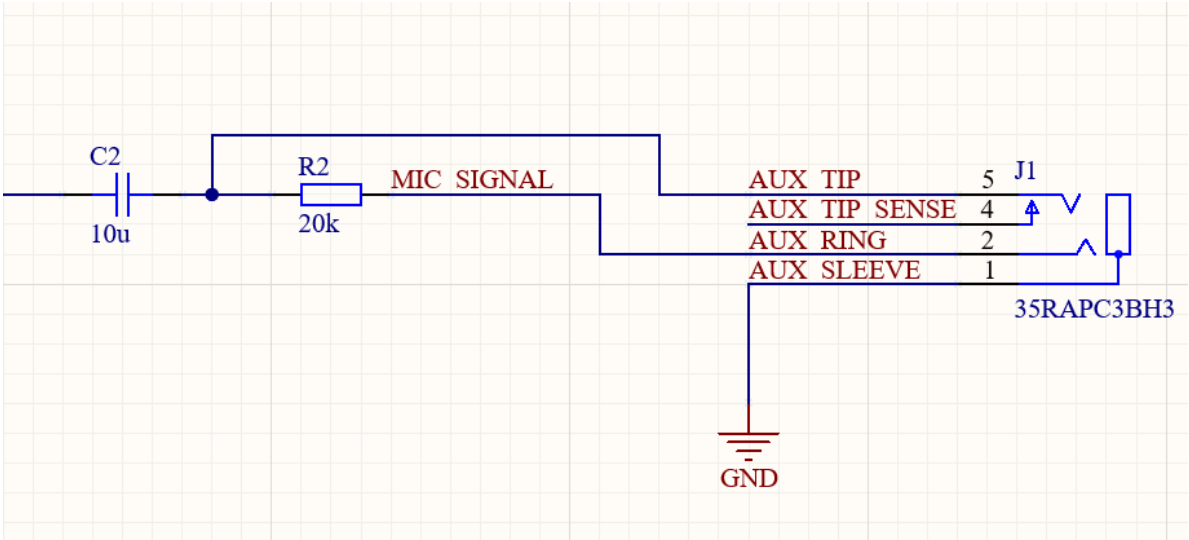

Figure 8 – Phone connector, TRS setup for microphone.

Figure 8 shows the microphone signal is being sent through a DC-blocking capacitor and biasing the tip signal. The ring connector on the aux provides a low voltage that can be either used as voltage bias or be grounded.

b) Audio preamplifier

Using the same amplifier as in Figure 6, the audio output from the computer can also be processed into the DSP. A slight modification when working with audio is to use a potentiometer to control the volume.

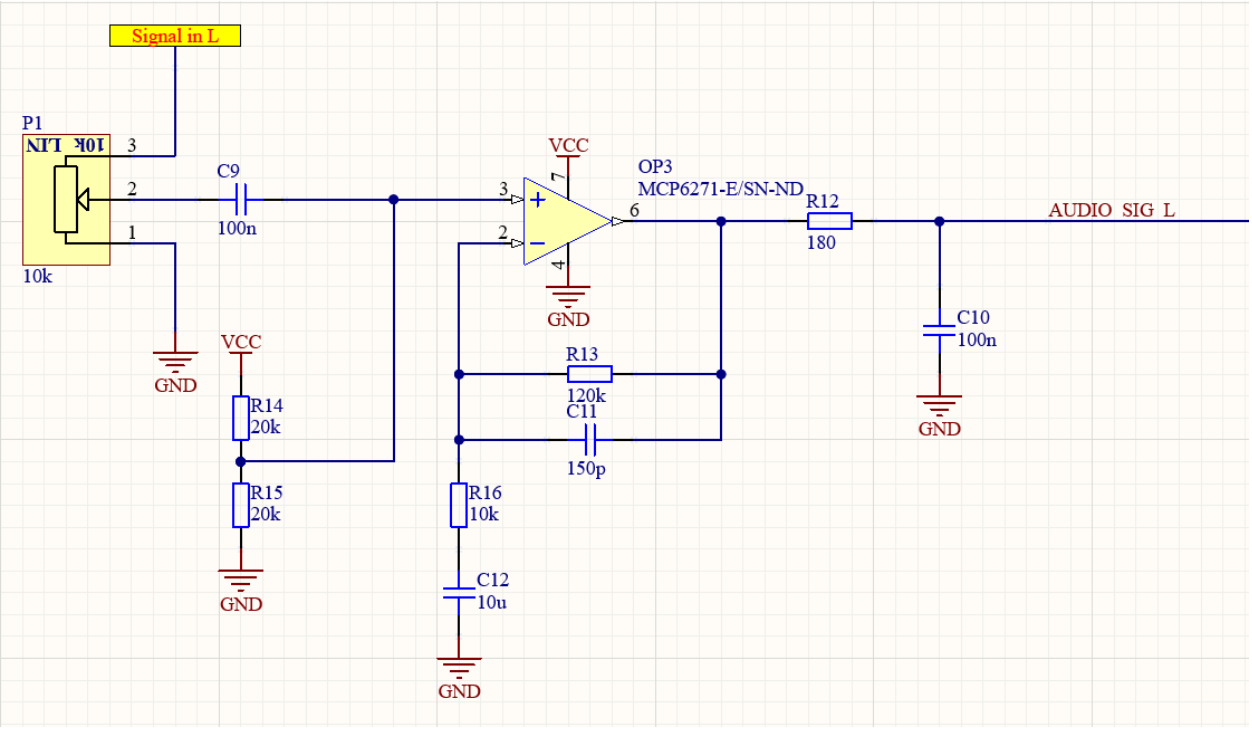

Figure 9 – Preamplifier circuit from computer to DSP

Figure 9 is a preamplifier circuit designed to process the computers output to the DSP. The circuit for the preamplifier, takes the incoming signal from the computer's soundcard, amplifies the signal by twelve, cuts off frequencies higher than 8841 Hz, and an antialiasing filter with a cutoff frequency of 8841Hz before the output which is then sampled by the STM32 ADC.

The connection AUX_TIP_SENSE in figure 8 is a mechanical switch used to tell an IC that there is a connection to the plug. In this prototype where the required component is missing the following comparator circuit is used to simulate the response.

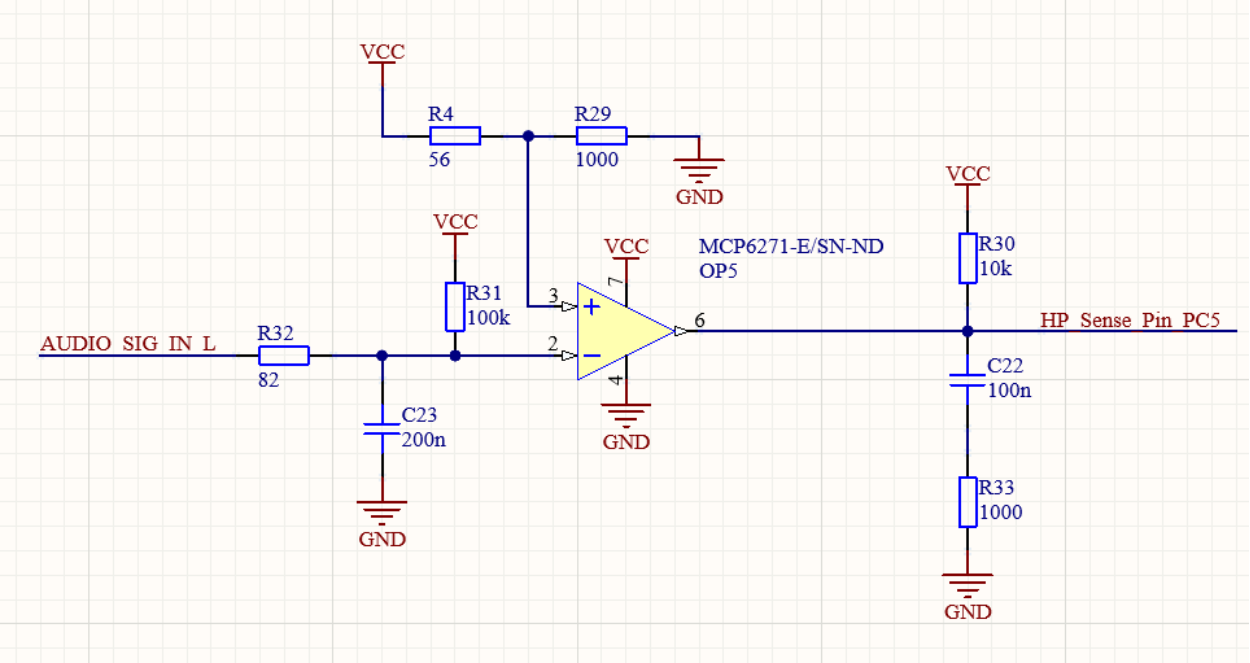

Figure 10 - Comparator sense circuit

Figure 10 is a comparator circuit meant to simulate the response of figure 8's AUX_TIP_SENSE. The circuit divides the audio signal, sending a small part into the comparator and compares it to 3.125V. This is a temporary solution on breadboard and will be replaced with the following configuration in figure 11 on PCB.

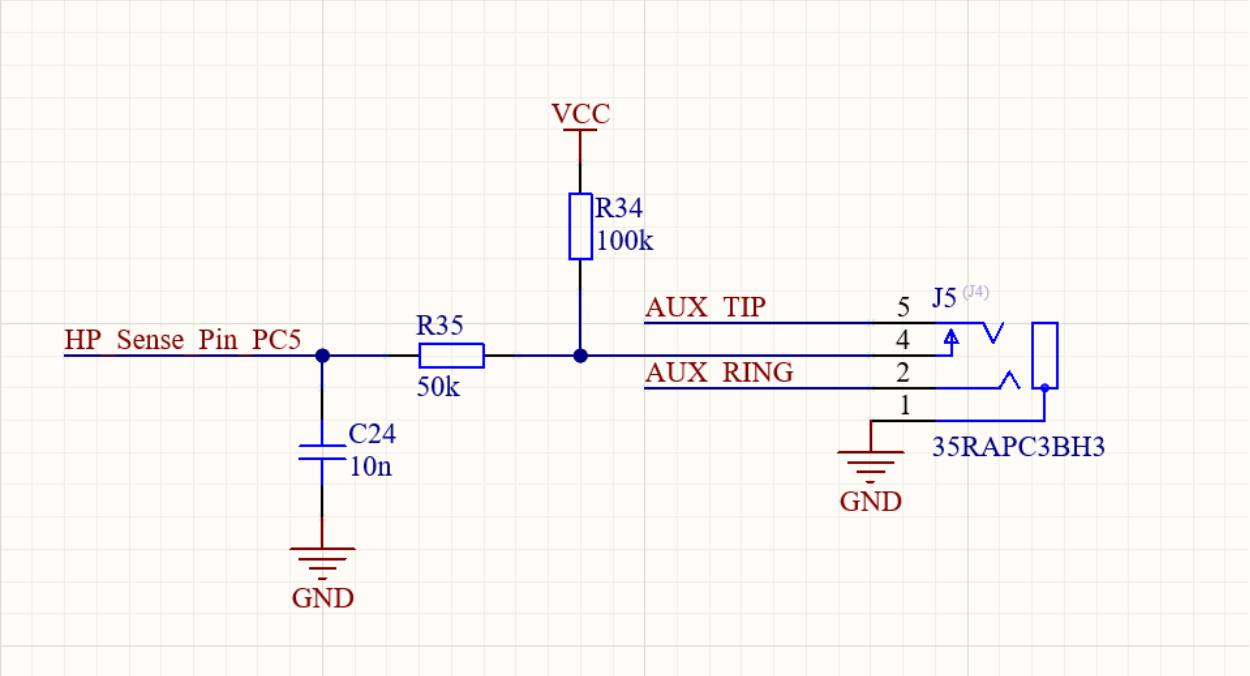

Figure 11 - Jack sensing circuit

3.2. USB-type C Implementation

Since the platform needs both power and the ability to communicate with the computer, a circuit is needed to manage the information from the USB protocol. USB uses a differential pair configuration with two data lines, being positive data signal and a negative data signal to communicate with other devices. The STM32 processor could manage the USB protocol but would have to be initiated with the device setup process then having a playback delay would take a chunk of the memory and processing power, slowing down the DSP. Therefor it is better to let another circuit manage the USB protocol while being a source of voltage. What is required then is a circuit with USB-codec, ADC, DAC, and I2S. The circuit needs to be small and have the desired outputs. There are other microcontrollers for all four tasks, but they are either expensive or need to be programed before usage. What is needed is a circuit with a modular approach that does not require the use of third-party software. What was found is the PCM29xx series and later settled on the PCM2904 from Texas Instruments [7]. The PCM2904 is a single-chip USB stereo audio codec with USB-compliant full-speed protocol controller. On the chip we can find both ADC and DAC ports for both left and right audio. The circuit does the setup of the USB-protocol at startup and has a playback delay of 1 ms.

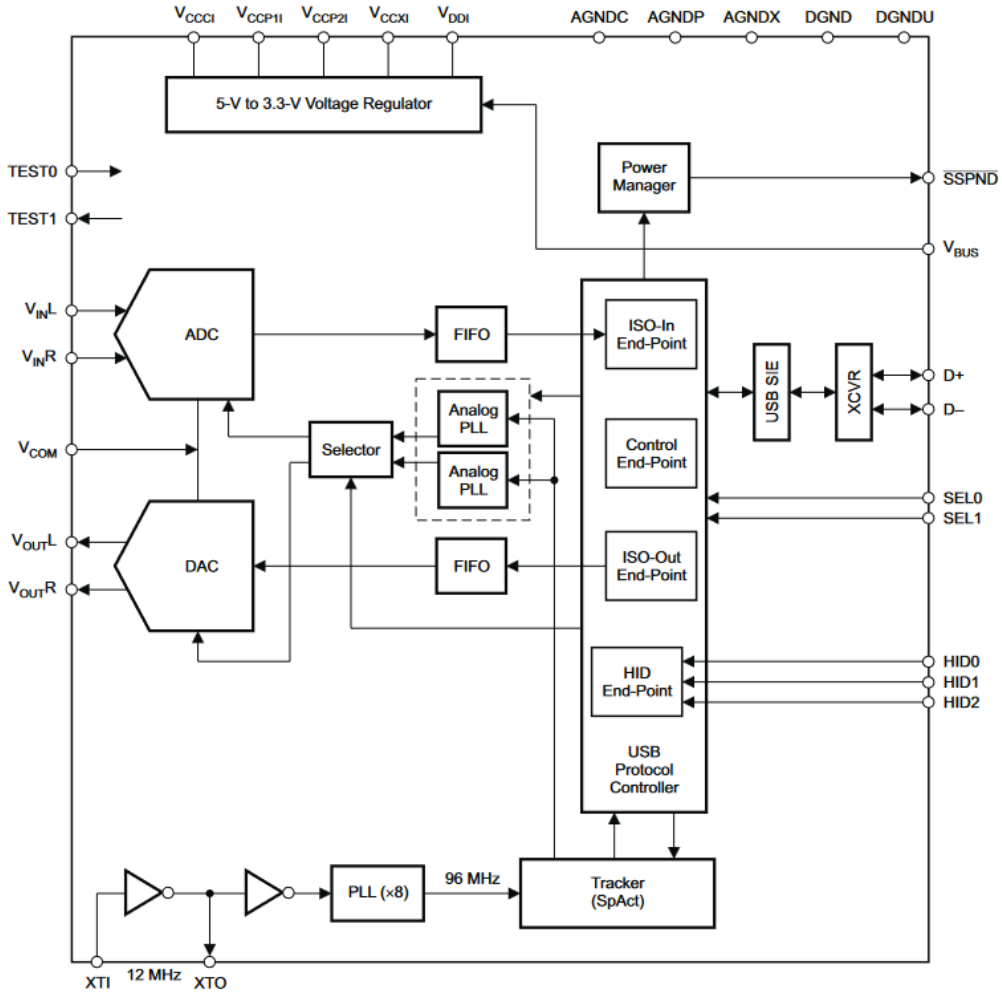

Figure 12 – PCM2904 Functional block diagram. Figure cropped from datasheet [7].

Figure 12 shows inputs and outputs of the PCM2904. The USB-protocol requires a highly accurate clock to work. The chip itself will not work without a crystal oscillator at pins XTI and XTO. The oscillator pins need to also be grounded with capacitors and have a high impedance resistor between the pins of the oscillator. It should also be noticed that when working with circuits that

requires external oscillators, the oscillator should be as close to the circuit as possible without any over-hanging resistance. The information of what components required can be found in the datasheet under clock and reset.

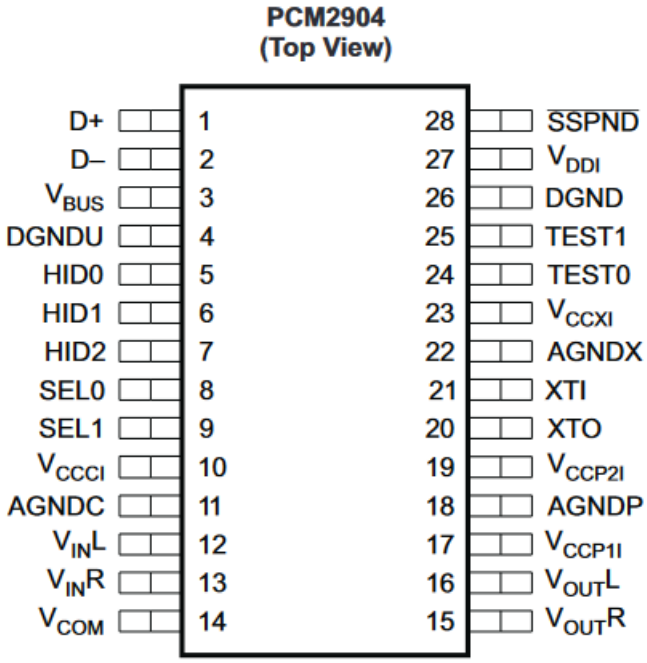

Figure 13 – PCM2904 Pinout Configuration. Figure cropped from datasheet [7].

Figure 13 is the pinout configuration of the PCM2904. The chip comes in SSOP-28 Pin configuration and is used in combination with previous amplifiers to make a complete circuit. Though one problem emerges with this circuit and that it needs both 5V supply voltage and 3.3V for the internal AD-converter. In the datasheet under typical application, the circuit is then complimented with a Low drop-out voltage regulator (LDO), USB-type C connector configured to USB 2.0, and the oscillator [7].

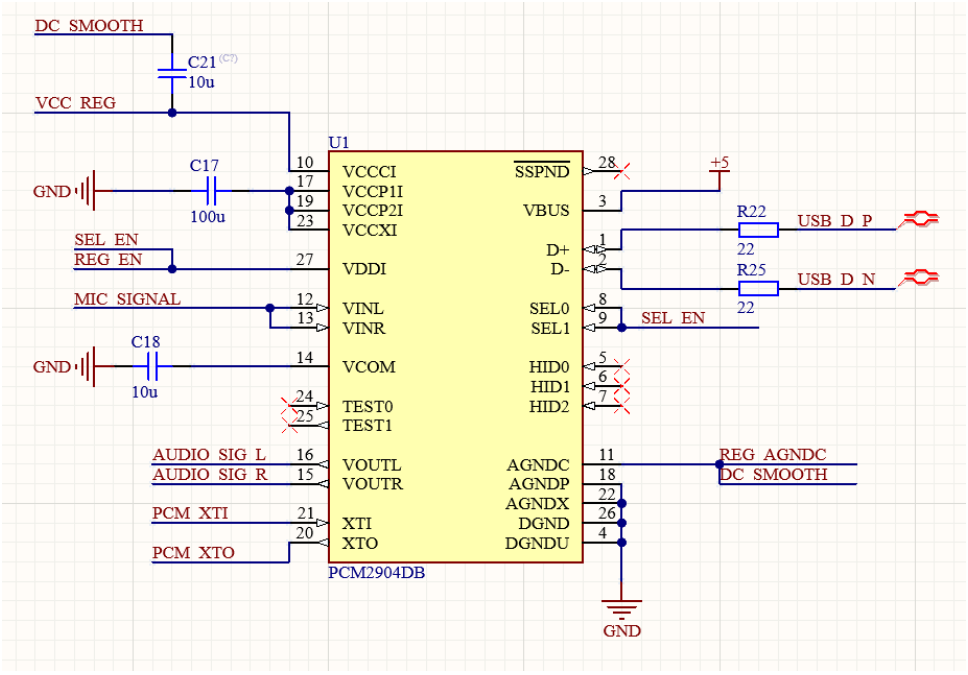

Figure 14 – PCM2904 single mic setup, audio L/R.

Figure 14 shows the typical connection for a PCM290x series using the datasheet for reference. Audio signals for left and right passed onto the amplifier in figure 9. Since this design is only for one microphone in mind, the left and right microphone signals are combined, giving the same signal on left and right audio to the computer.

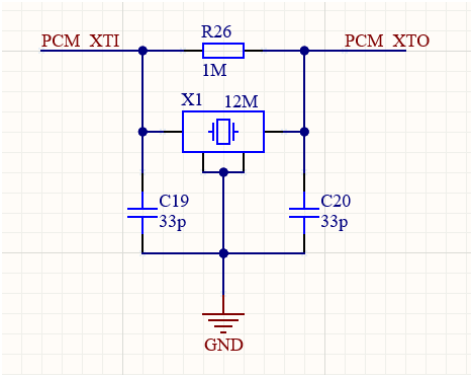

Figure 15 – Crystal oscillator 12MHz.

Figure 15 shows the typical application for the crystal oscillator input and output. Depending on the oscillator the values of C19 and C20 differ.

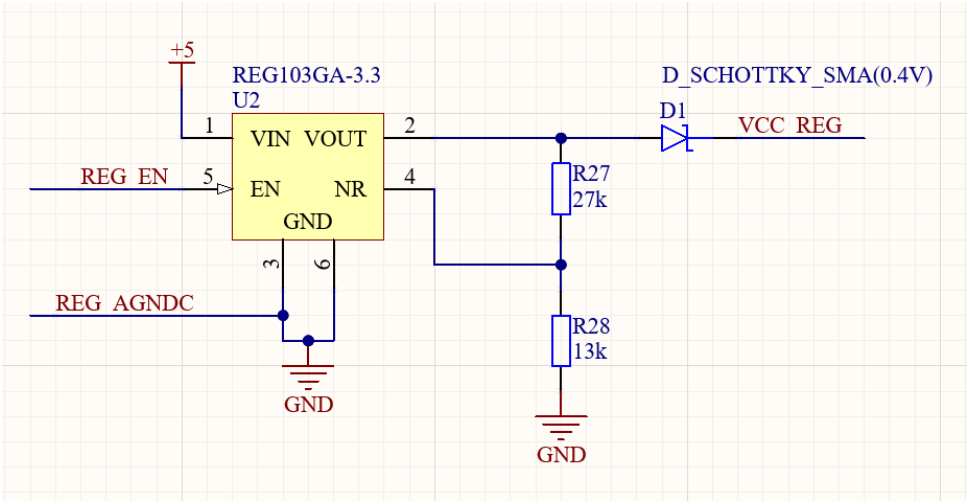

Figure 16 – REG103xA 3.3V Low drop-out regulator.

Figure 16 is an LDO regulator is connected to PCM circuit to improve the AD-converter and supply the rest of the circuit with power.

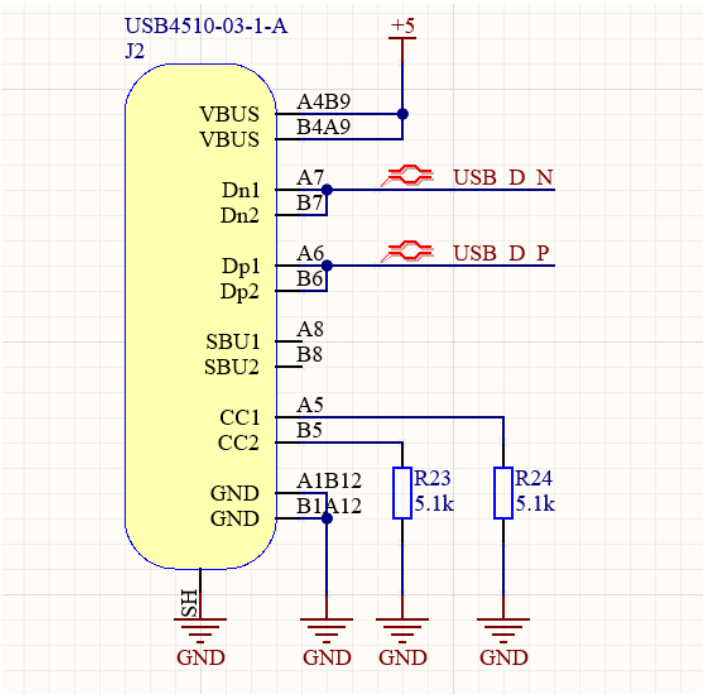

Figure 17 – USB-type C 2.0 configuration.

Figure 17 shows the USB-C connection with a USB 2.0 configuration with CC1 and CC2 connected to ground via resistances R23 and R24.

In mixed signal design, ground signals should not be separated but analog and digital signals should be far apart from each other [8]. Low analog signals should also be separated from High analog signals. These components soldered onto a SOIC and SSOP-adapters, respectively. Exception is the USB-type C connector circuit when it is replaced with the breakout board 2585 from Pololu [9]. With all components on place the output signals of the circuitry are applied onto the DSP-platform.

The DSP platform is built and programed on a STM32-processor with the use of ST's own interactive development environment named STM32CubeIDE in combination with ST's code generation program called STM32CubeMX. CubeIDE is an eclipse-based development environment with support for languages such as C, C++, Pascal, and Java. The DSP platform has a codebase which is written in C for low-level control, adaptability, and speed. ST provides a Hardware Abstraction Layer for embedded software with drivers for both high-level and low-level control. With the use of CMSIS DSP libraries, the processor can use a multiple of common signal processing functions. CubeMX provides a graphical interface for ease of generating functionality on specific pins. This project will include code and configuration of both Cortex M4(STM32f429) and Cortex M7(STM32F767). These processors share the same amount of pins and have similar configurations but have different FPU's and the M7 comes equipped with an adaptive real-time memory accelerator. The platform was programed using an older version of CubeIDE, the firmware and libraries are old and porting the codebase to latest firmware and upgrading the toolchain becomes a high priority. What needs to be done is to add additional AD-conversion, pin configuration, and merge the code Eric's IIR-filter functions [10]. Since merging code can be tedious process, version control is required when developing on the platform.

When working with a bigger codebase its common to use version control such as Git to exchange code and track code. GitHub is a website with version which hosts repositories for users and companies. When using git, developers can stage changes and send it to the repository. Other developers can later pull, fetch, merge, branch from the codebase. These commands are used in specific situations when working in the repository. When working with git it is inevitable that conflicts will emerge when developers work on the same file. A conflict in git is a difference in two commits meaning difference in rows and columns. Solving a conflict can be done by various way but in this project when only two people are working on the program, a linear approach of using add, commit, and push with thorough communication is enough.

During the development process of the circuitry, minor changes was made to fit a model. The model being that sounds should not reach a point where it causes damages to the ear of the user. The amplified signal is supposed clip a small amount in its state of least resistance. Too much gain in a signal, however, causes distortions, and can be damaging to the ear long term [11]. Sound above 85 dB is the limit where noises can cause damages to the ear. Soundcard circuitry will differ on most computers and the ability for the user to change the gain of the signal is necessary.

4. Result

The complete circuit.

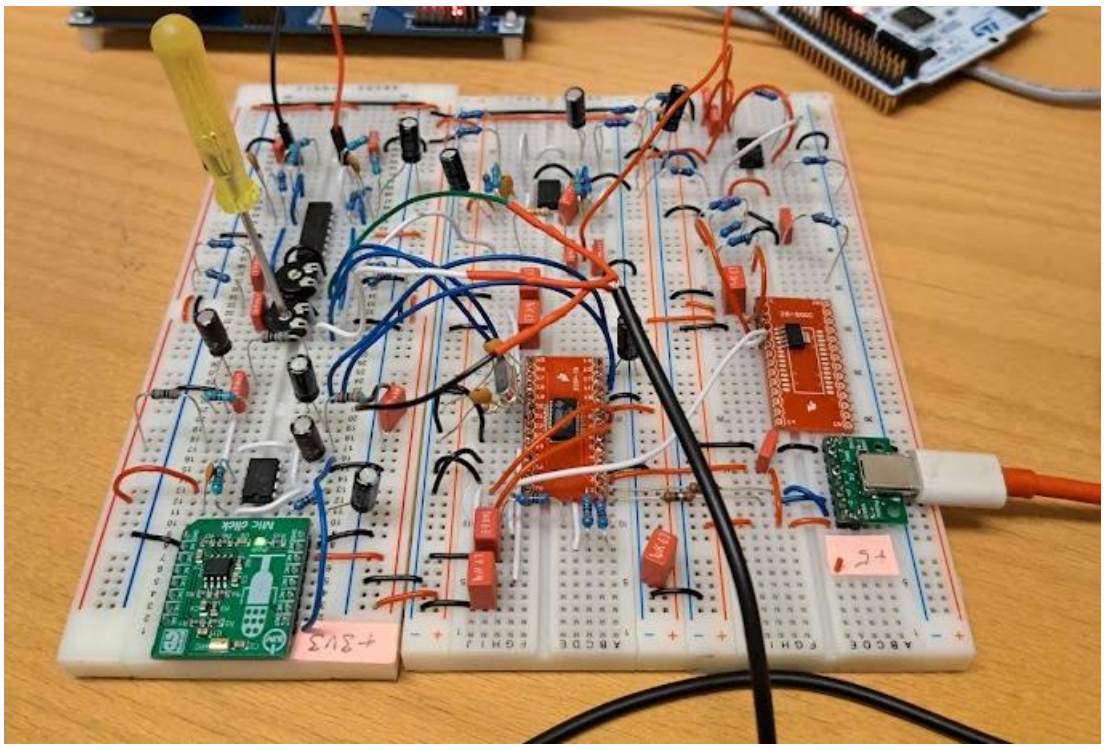

Figure 18 - Final circuit breadboard design

Figure 18 is a combination of figure's 7-10 and figure's is 14-17. The comparator from figure 10 senses if the left audio channel is in this configuration.

4.1. Audio jack, TRS Implementation and TRRS implementation

a) Microphone

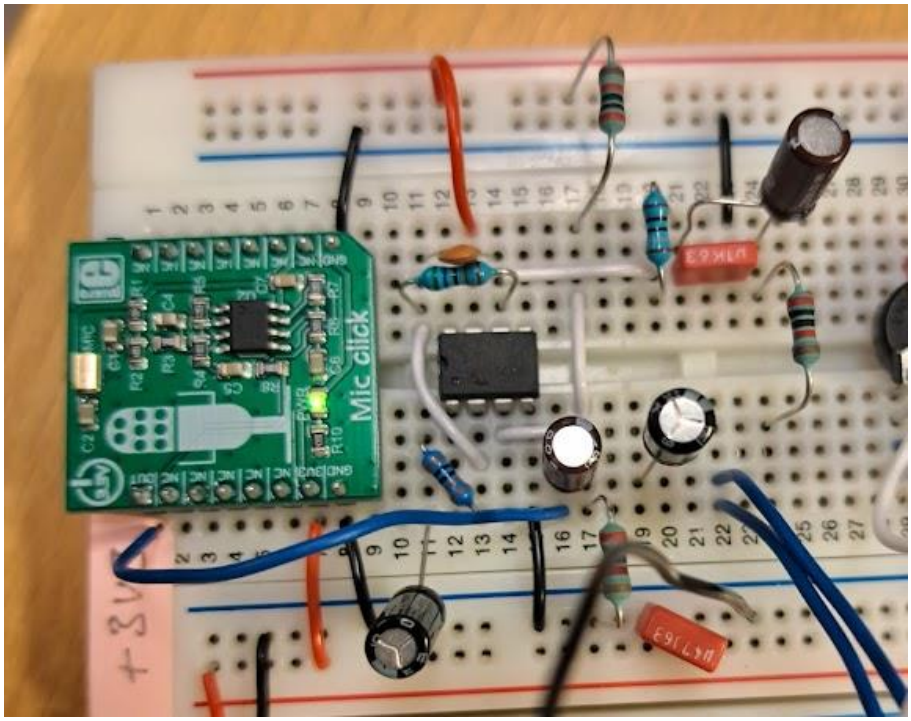

Figure 19 - Microphone amplifier.

Figure 19 shows the signal from the Mic click circuit is passed onto the amplifier while keeping the signal at reference voltage of 1.65 V. Then the signal is amplified by 6.6 times and put into the configuration of figure 7. Playing a 1000 kHz test into the microphone from a 5 cm distance gives the following results.

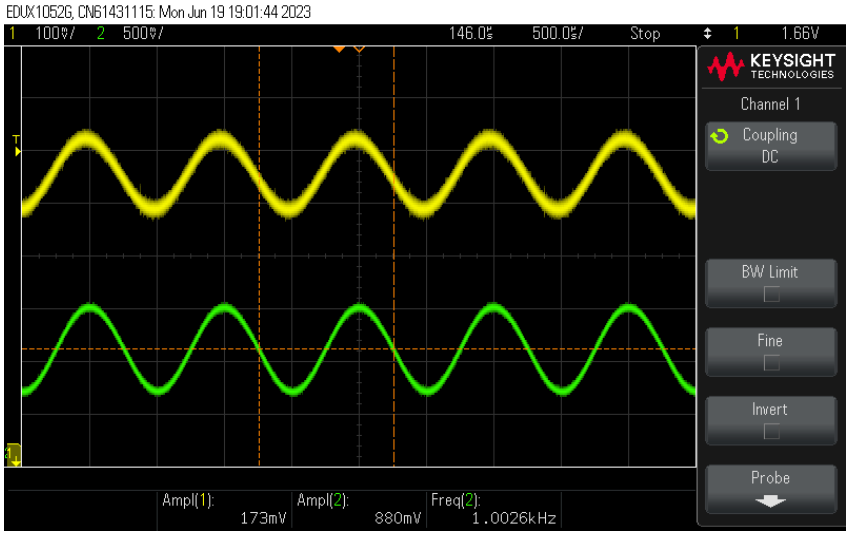

Figure 20 - Microphone signal amplified and biased by aux (Green). Original microphone signal (Yellow).

Figure 20 compares the original signal from the Mic click circuit (Yellow), and the amplified and biased signal which is sent to the computer (Green).

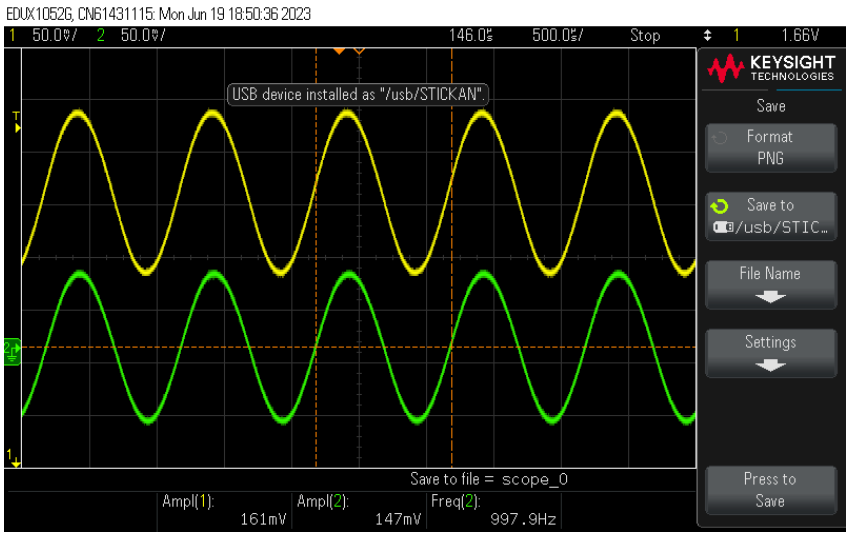

Figure 21 - Microphone signal no amplification and biased by aux (Green). Original microphone signal (Yellow).

Figure 21 shows the microphone signal passed through the amplifier without amplification, shorting the R5 resistance in figure 7.

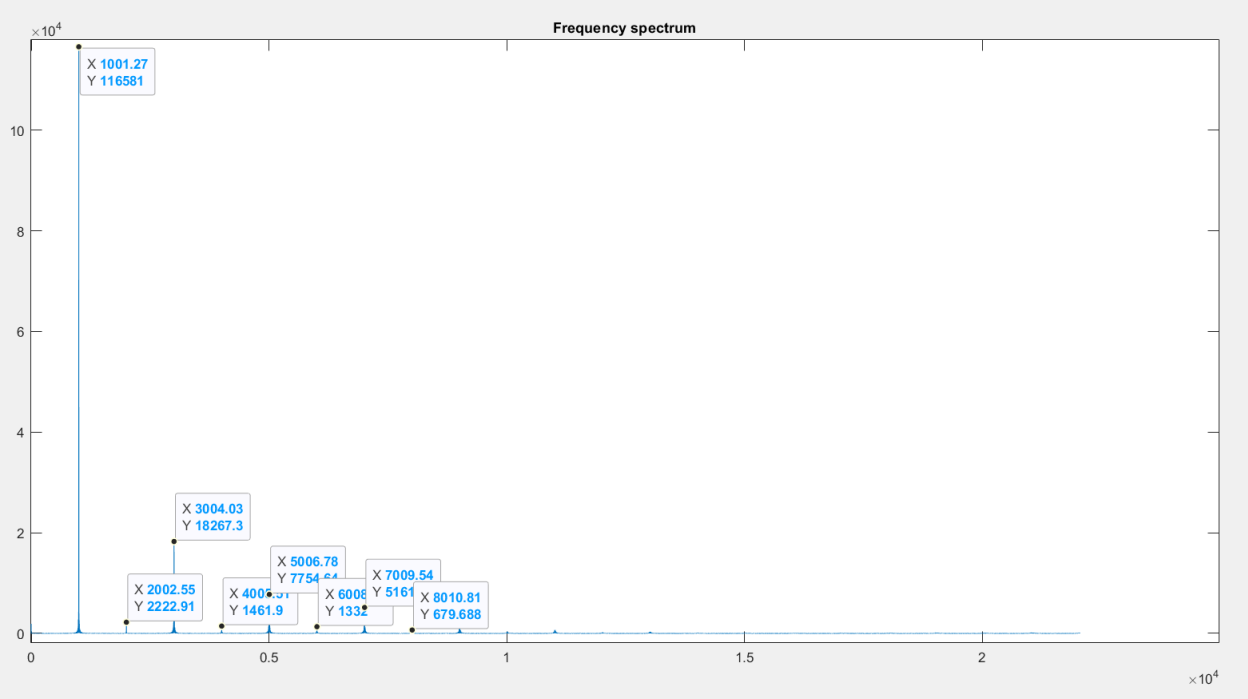

Figure 22 - Matlab frequency spectrum plot. Amplified microphone circuit. 1 kHz test signal.

Figure 22 shows the FFT frequency plot generated by Matlab showing the harmonics of the recorded 1 kHz signal.

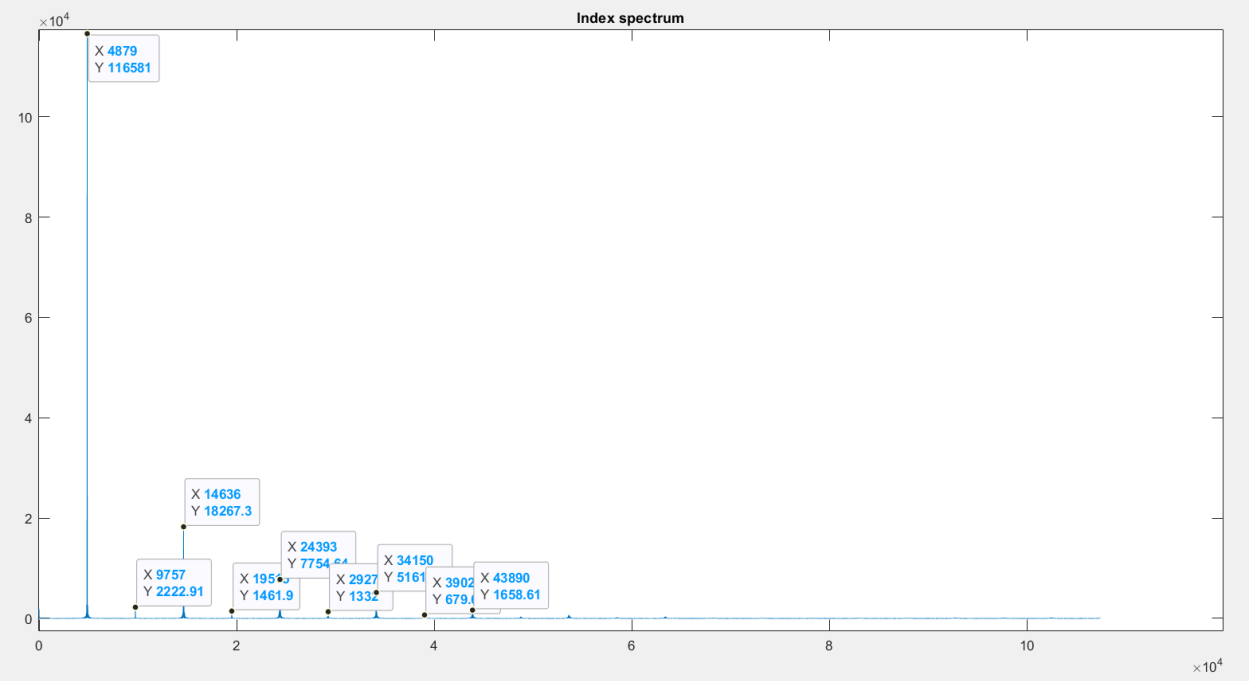

Figure 23 - Matlab index spectrum plot. Amplified microphone circuit. 1 kHz test signal.

Figure 23 shows the index spectrum of the recorded 1 kHz test signal used to calculate THD. Total harmonic distortion of the amplified microphone circuit is calculated in Matlab to 0.1784 or 17.84 %.

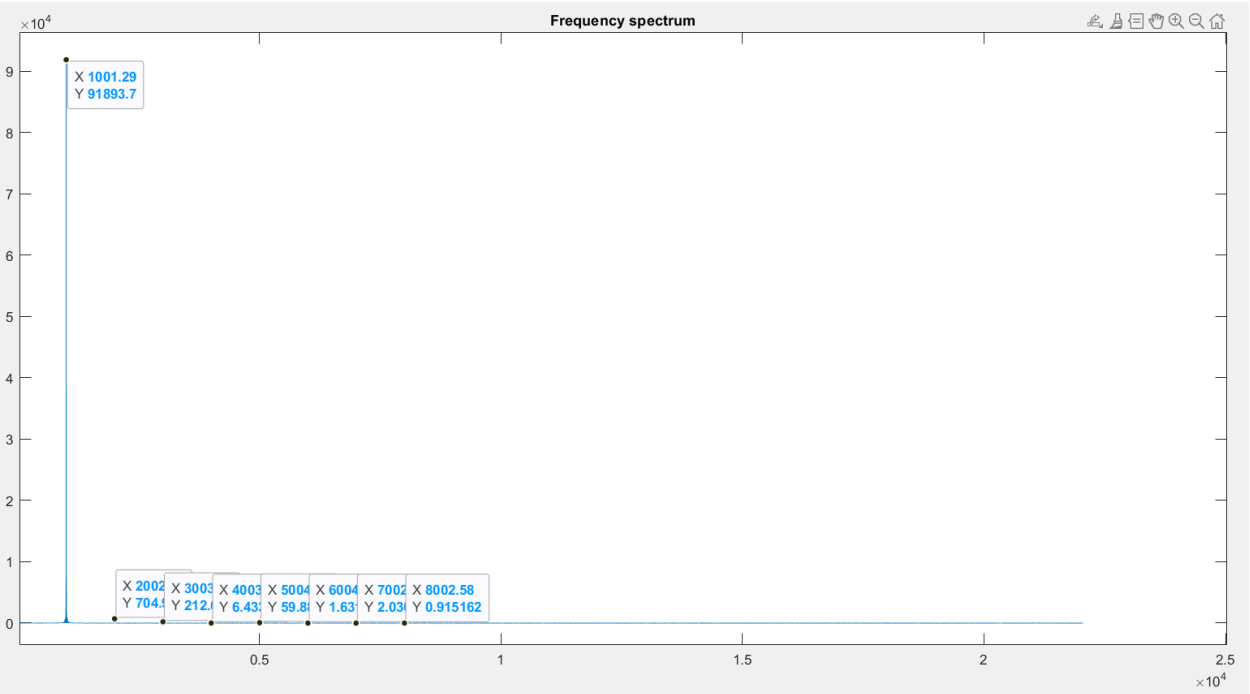

Figure 24 - Matlab frequency spectrum plot. No amplification microphone circuit. 1 kHz test signal.

Figure 24 shows the FFT frequency spectrum in Matlab using a 1 kHz recorded test signal, no amplification.

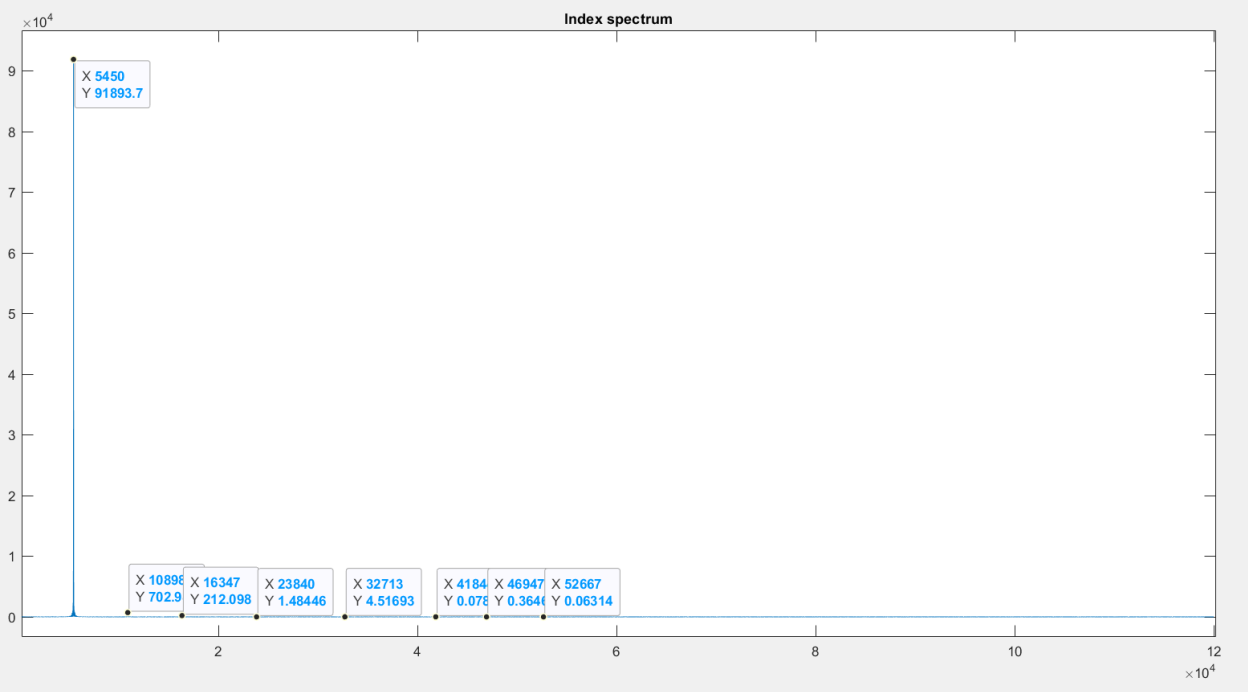

Figure 25 – Matlab index spectrum plot. No amplification microphone circuit. 1 kHz test signal.

Figure 25 shows the index spectrum of the recorded 1 kHz test signal used to calculate THD. Total harmonic distortion of no amplification on the microphone is calculated in Matlab to 0.0080 or 0.8 %

b) *Audio preamplifier*

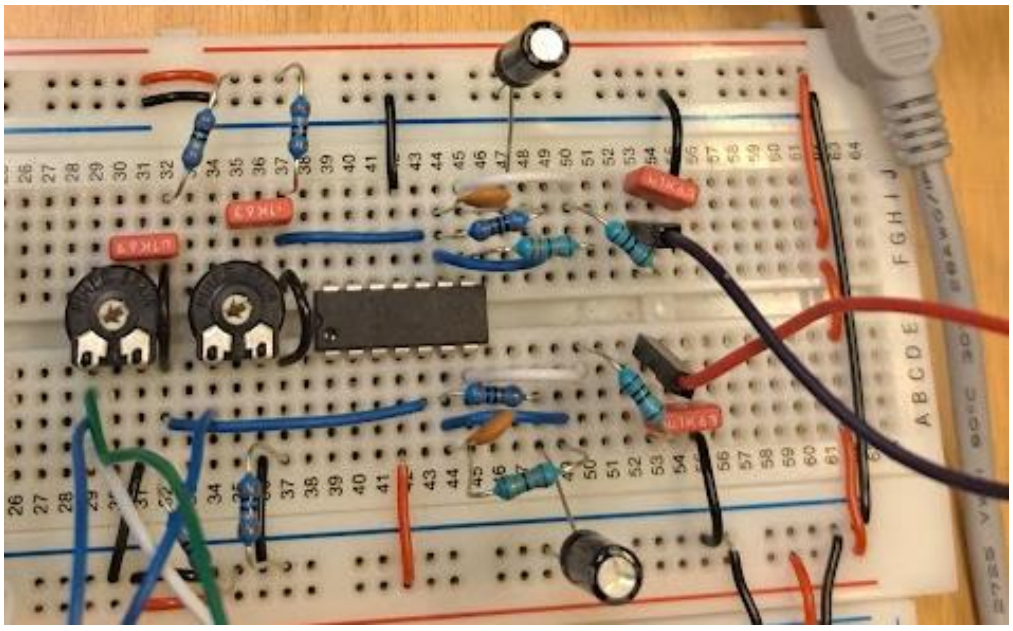

Figure 26 – Preamplifier circuit from PC aux and PCM2904 to STM32

Figure 26 shows the preamplifier circuit used to amplify to the signal sent from computers output or from the PCM2904.

Figure 27-33 shows measurements of preamplifier circuit using two different computer output configurations. Using the output from both standard headphone jack and external soundcard. The potentiometer is set in ranges from 0 to 100 % where 0 means lowest impedance for the incoming

signal and maximum volume. Maximum resistance at 100 % of the potentiometer results in lowest volume

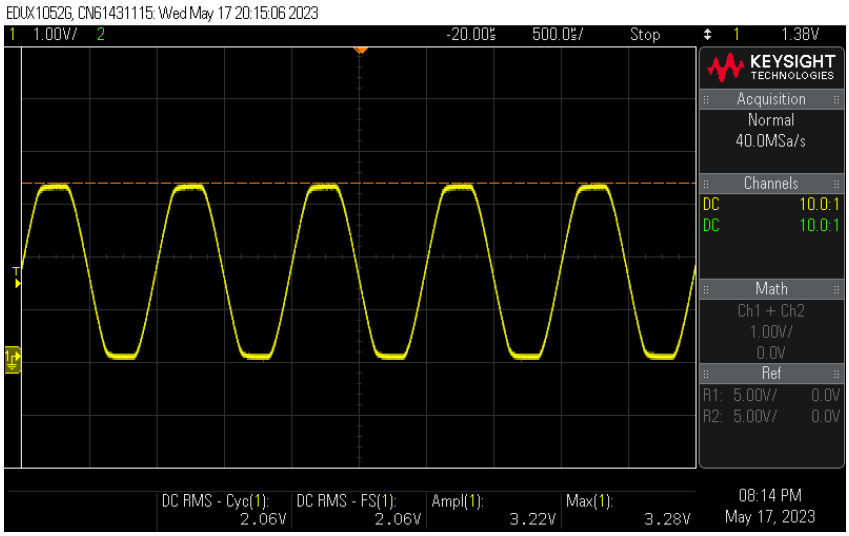

Figure 27 - Aux computer, test signal 1 kHz potentiometer lowest resistance

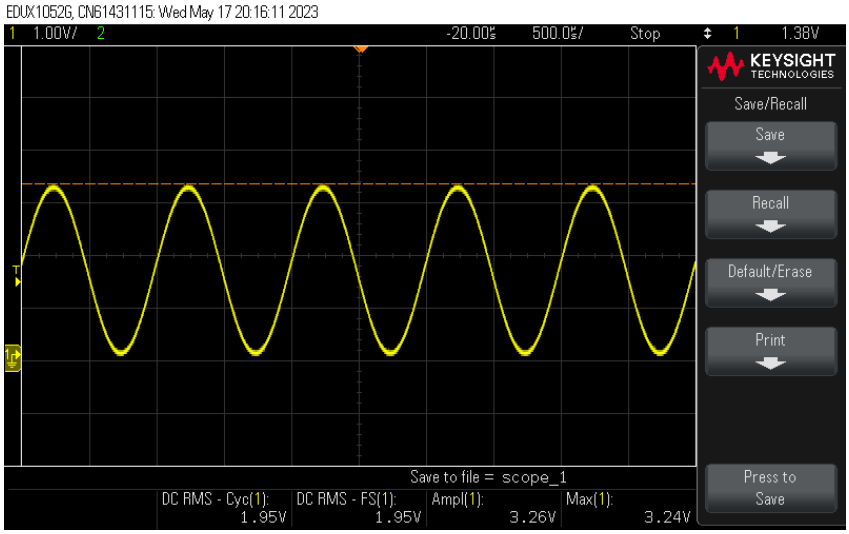

Figure 28 - Aux computer test signal 1 kHz adjusted.

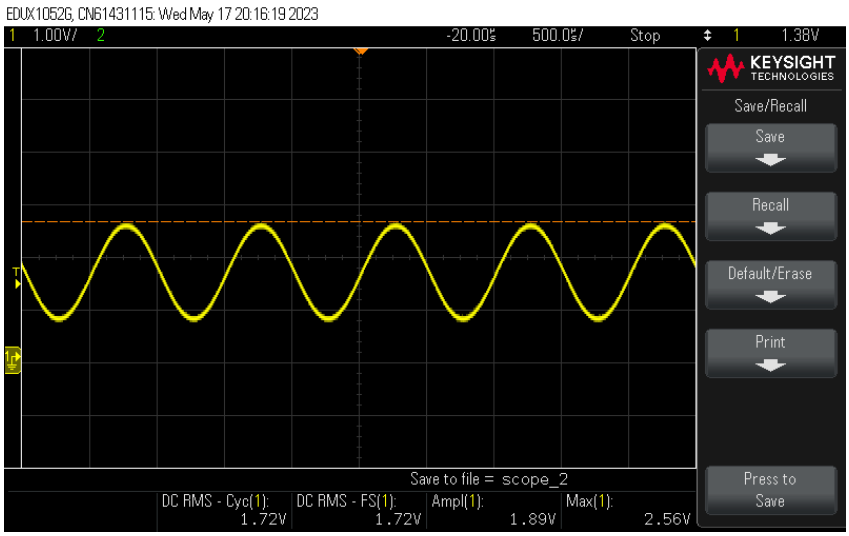

Figure 29 - Aux computer test signal 1 kHz potentiometer 50 % resistance

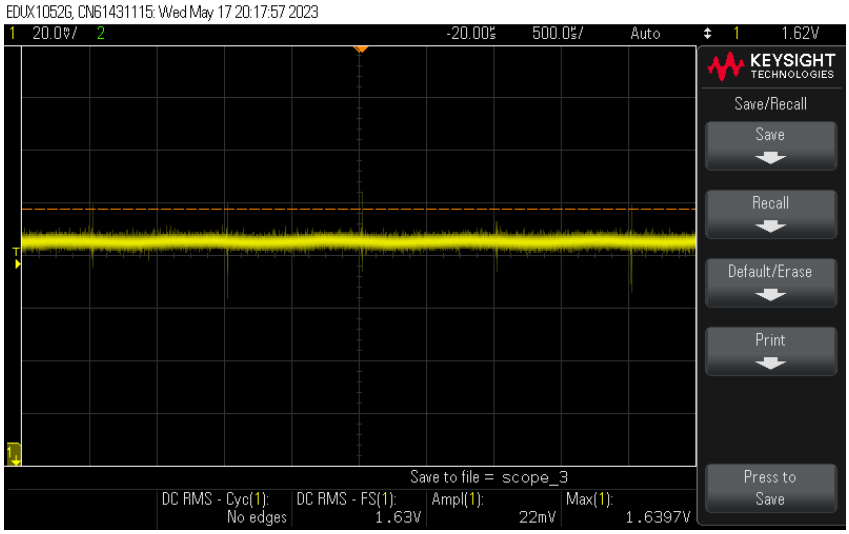

Figure 30 - Aux computer test signal 1 kHz potentiometer maximum resistance

Measurements from an external soundcard device:

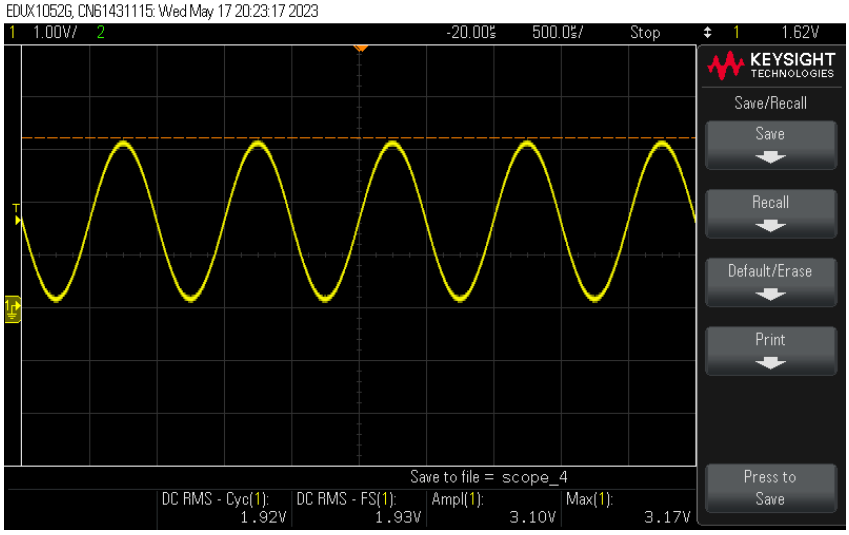

Figure 31- USB soundcard from computer, test signal 1 kHz potentiometer lowest resistance

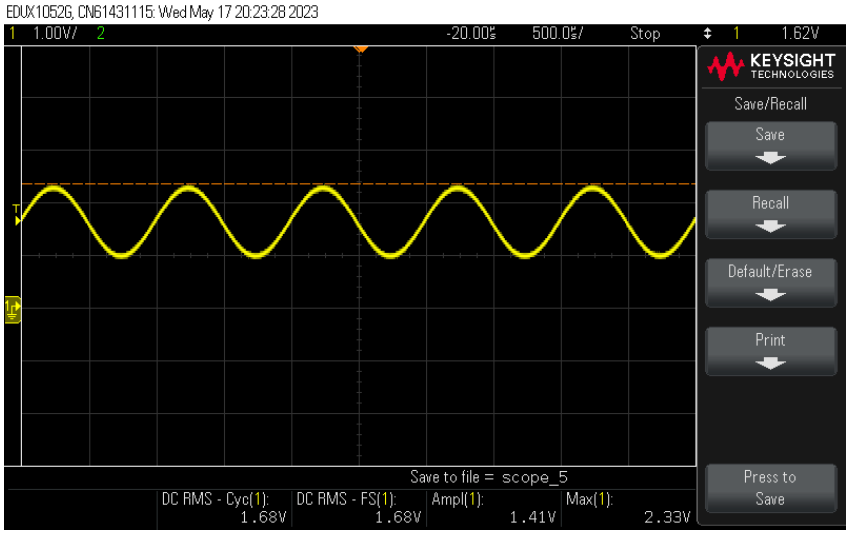

Figure 32- USB soundcard from computer, test signal 1 kHz potentiometer 50 % resistance

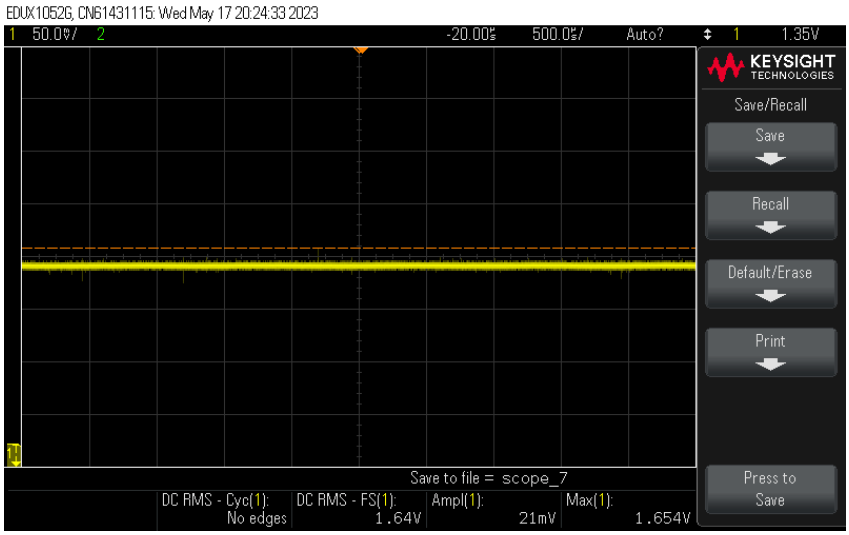

Figure 33 - USB soundcard from computer, test signal 1 kHz potentiometer maximum resistance

4.2. USB-type C implementation

a) Microphone

Connecting PCM2904 to pc:

```
jaanze@HP:~/exjobb/examensarbete$ sudo dmesg -C
jaanze@HP:~/exjobb/examensarbete$ sudo dmesg -w
[43713.762797] usb 1-4.3: new full-speed USB device number 117 using xhci_hcd
[43713.872182] usb 1-4.3: New USB device found, idVendor=08bb, idProduct=2904, bcdDevice= 1.00
[43713.872201] usb 1-4.3: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[43713.872208] usb 1-4.3: Product: USB Audio CODEC
[43713.872214] usb 1-4.3: Manufacturer: Burr-Brown from TI
[43713.904453] input: Burr-Brown from TI              USB Audio CODEC  as /devices/pci0000:00/0000:00:14.0/usb1/1-4/1-4.3/1-4.3:1.3/0003:08BB:2904.0099/input/input170
[43713.966978] hid-generic 0003:08BB:2904.0099: input,hidraw1: USB HID v1.00 Device [Burr-Brown from TI              USB Audio CODEC ] on usb-0000:00:14.0-4.3/input3
```

Figure 34 - Linux terminal running dmesg to examine kernel ring buffer.

Figure 34 shows a Linux terminal running dmesg, reading the kernel ring buffer and recognizing the PCM2904 as an audio codec.

Measurements of the microphone connected to PCM2904:

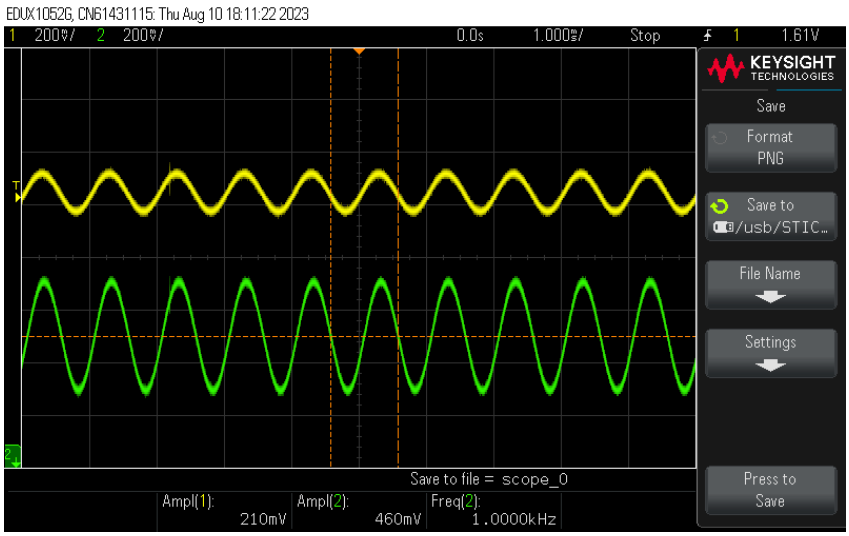

Figure 35 - Microphone signal amplified and biased by PCM2904 (Green). Original microphone signal (Yellow).

Figure 35 shows the PCM2904 with the amplified microphone circuit recording a 1 kHz test signal.

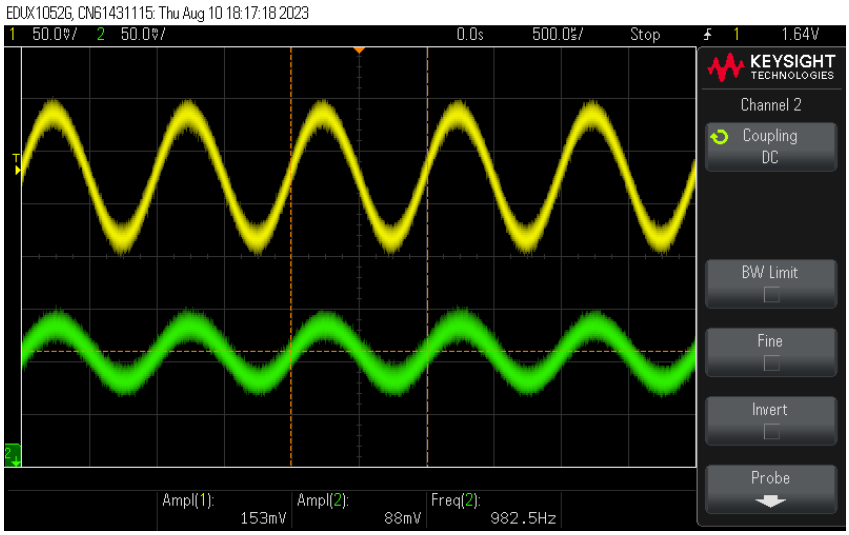

Figure 36 - Microphone signal no amplification and biased by PCM2904 (Green). Original microphone signal (Yellow).

Figure 36 shows the microphone signal sent to the ADC on the PCM2904 with a 1 kHz test signal without amplification.

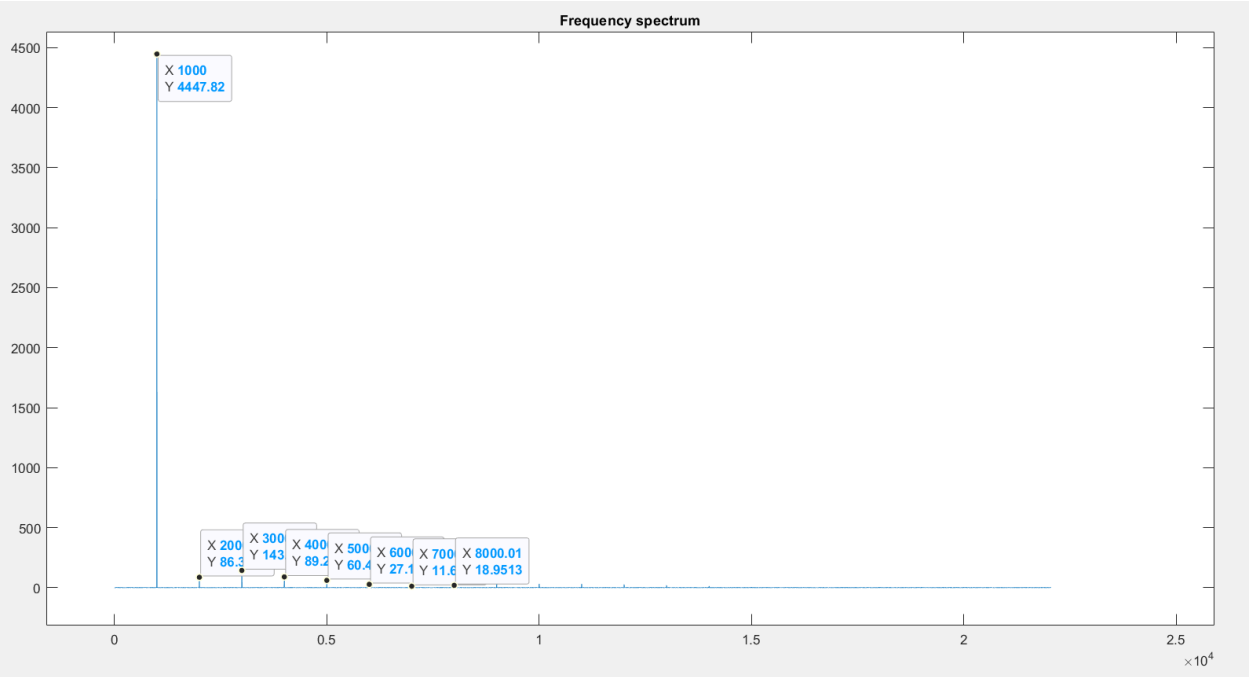

Figure 37 - Matlab THD frequency spectrum plot. Amplified microphone to PCM2904. 1 kHz test signal.

Figure 37 shows the FFT frequency spectrum of the recorded 1 kHz test signal with amplification in Matlab.

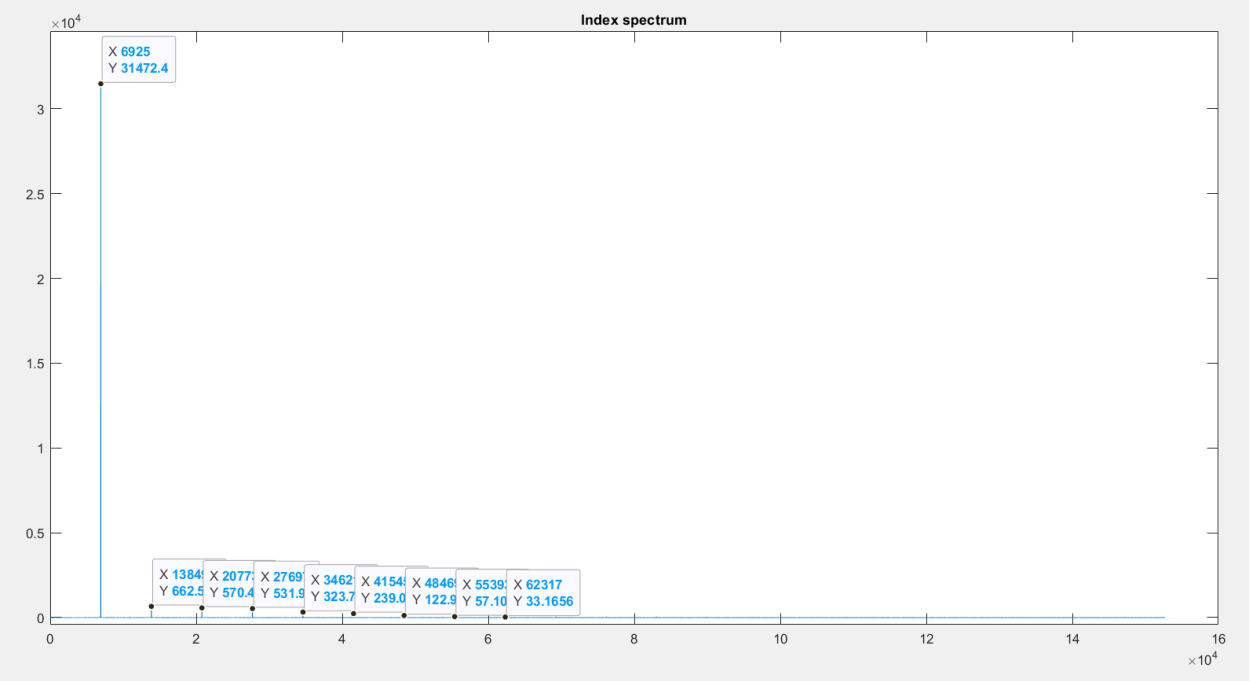

Figure 38 - Matlab THD index spectrum plot. Amplified microphone to PCM2904. 1 kHz test signal.

Figure 38 shows the index spectrum of the recorded 1 kHz test signal used to calculate THD. Total harmonic distortion of the amplified microphone to PCM2904 is calculated in Matlab calculated to 0.0458 or 4.58 %.

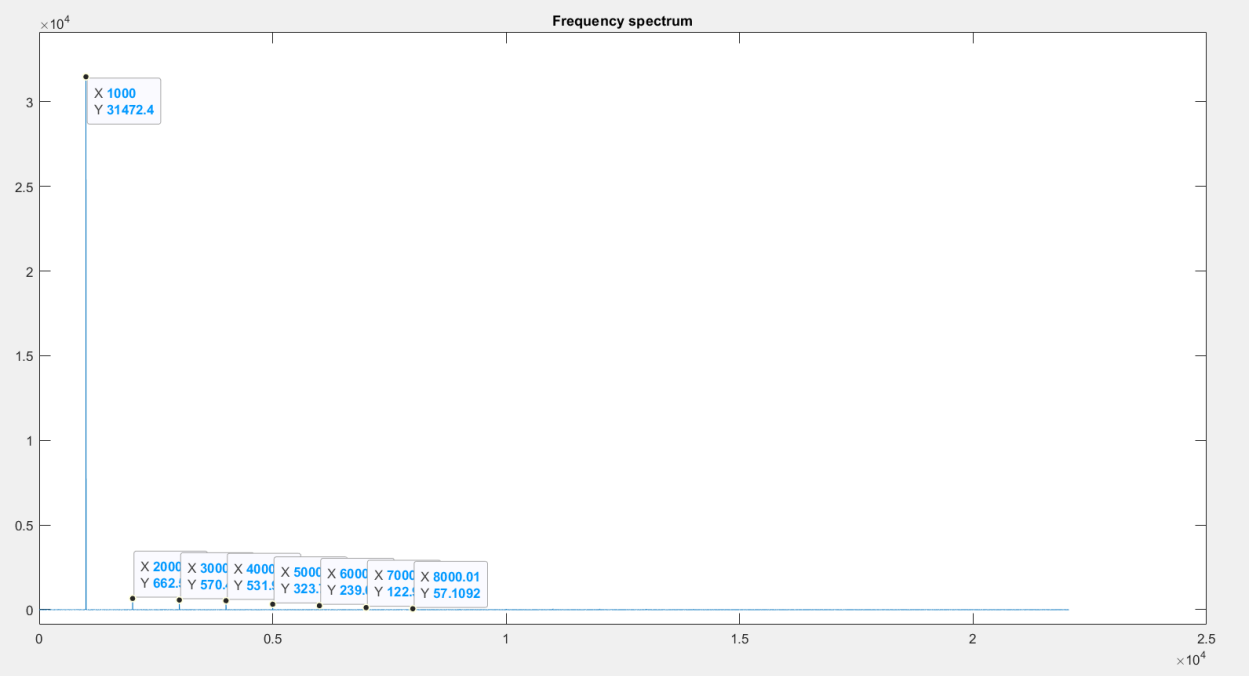

Figure 39 - Matlab THD frequency spectrum plot. No amplification, microphone to PCM2904. 1 kHz test signal.

Figure 39 shows the FFT frequency spectrum of the recorded 1 kHz test signal with no amplification in Matlab.

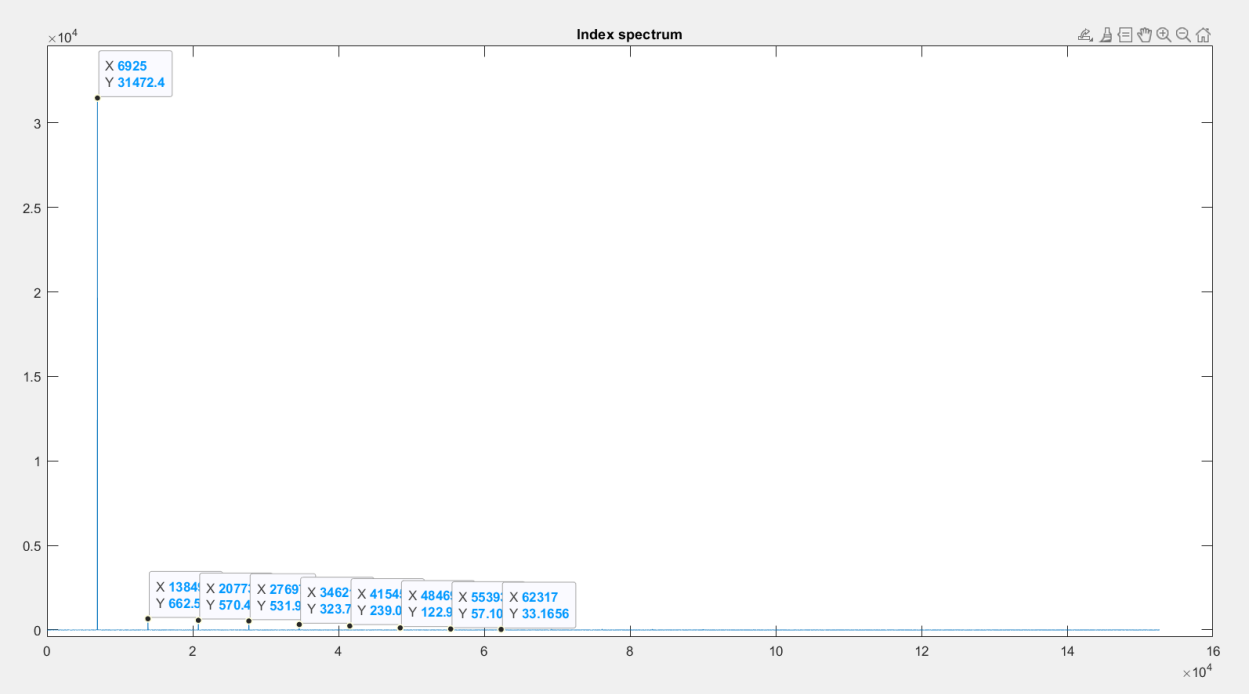

Figure 40 - Matlab THD index spectrum plot. No amplification, microphone to PCM2904. 1 kHz test signal.

Figure 40 shows the index spectrum of the recorded 1 kHz test signal used to calculate THD. Total harmonic distortion of no amplification, microphone to PCM2904 is calculated in Matlab calculated to 0.0352 or 3.52 %.

b) Audio preamplifier

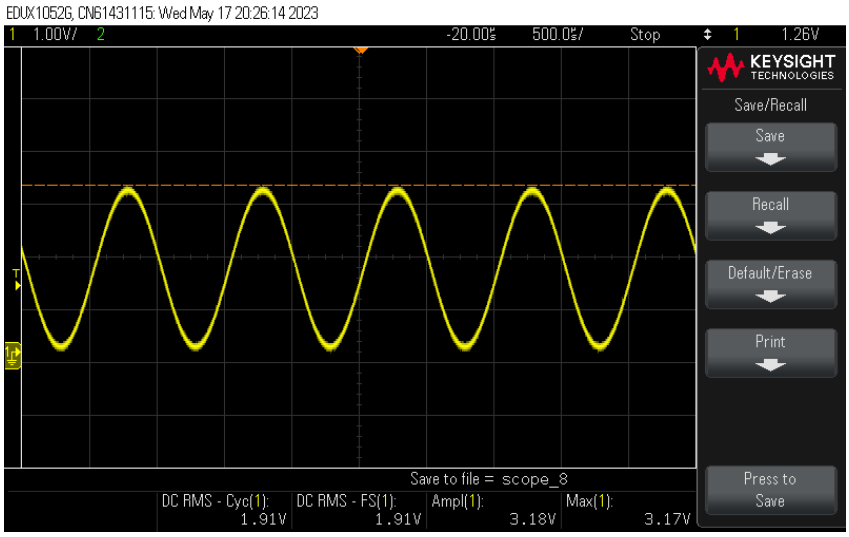

Figure 41 – PCM2904 test signal 1 kHz potentiometer lowest resistance

Figure 41 shows a 1 kHz test signal running through the preamplifier with the lowest amount of resistance.

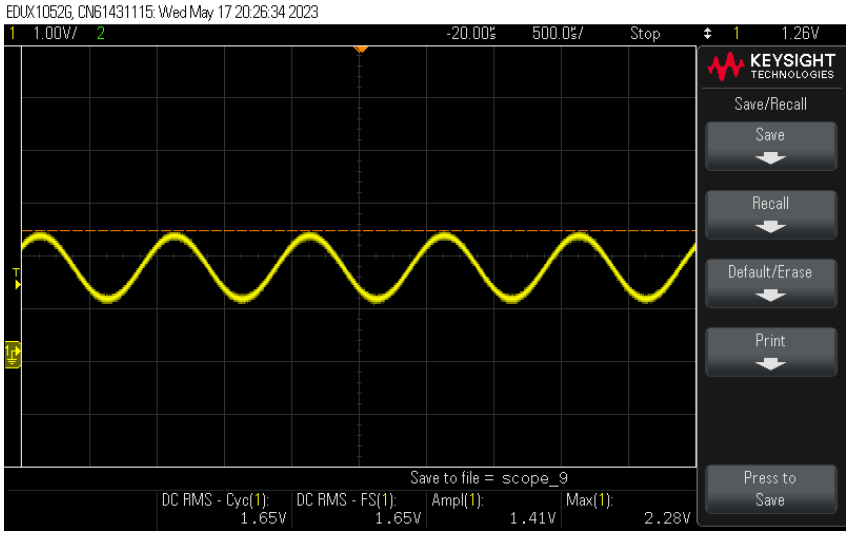

Figure 42 - PCM2904 test signal 1 kHz potentiometer 50 % resistance

Figure 42 shows a 1 kHz test signal running through the preamplifier with a 50 % resistance.

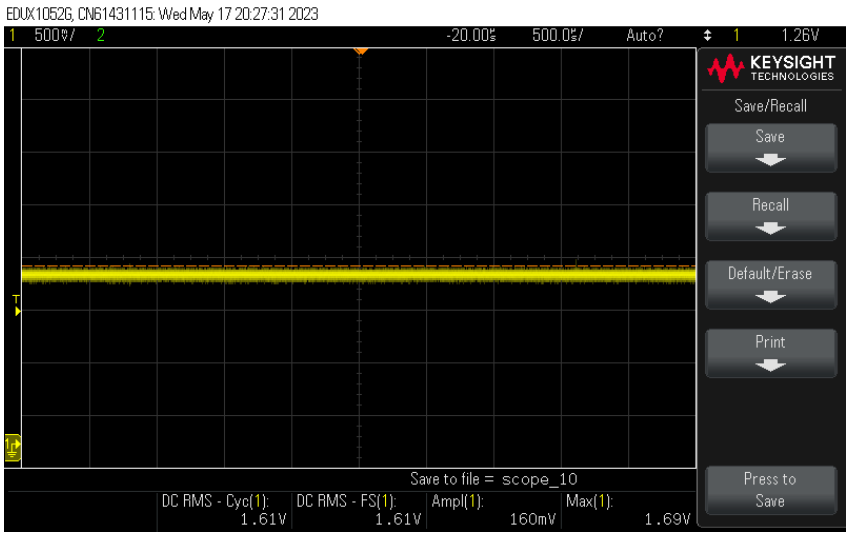

Figure 43 - PCM2904 test signal 1 kHz potentiometer maximum resistance

Figure 43 shows a 1 kHz test signal running through the preamplifier with the maximum resistance.

Comparator sense circuit:

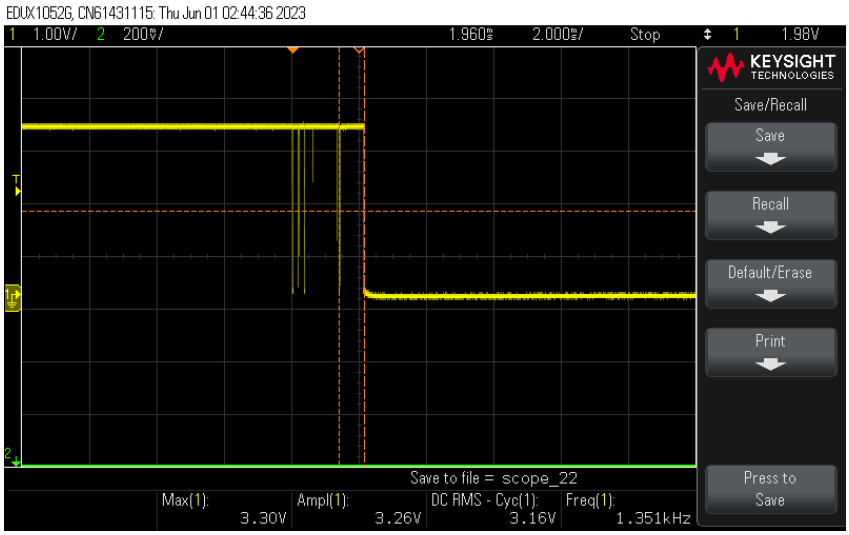

Figure 44 - Comparator sense circuit disconnection

Figure 44 shows the comparator circuit when disconnecting the phone connector from the computer.

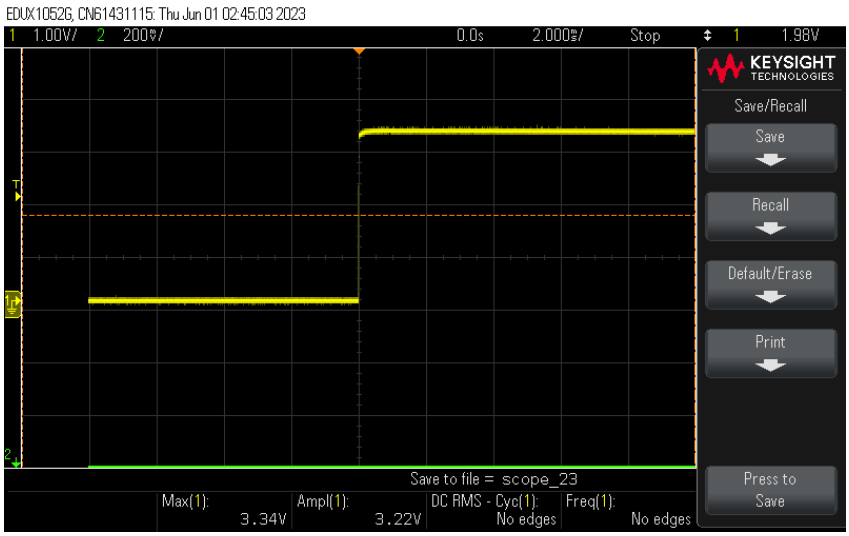

Figure 45 - Comparator sense circuit connection

Figure 45 shows the comparator circuit when connecting the phone connector to the computer.

4.3. Programing and configuration of STM32

The STM32f767's 12-bit ADC converter sample rate is controlled by a timer dependent on a divider of the internal clock frequency. Since the platform is already programed with over-sampling and filters in mind the timer is set to sample at a rate of 44 kHz. Which gives us a sample frequency of 22 kHz (Nyquist) [2].

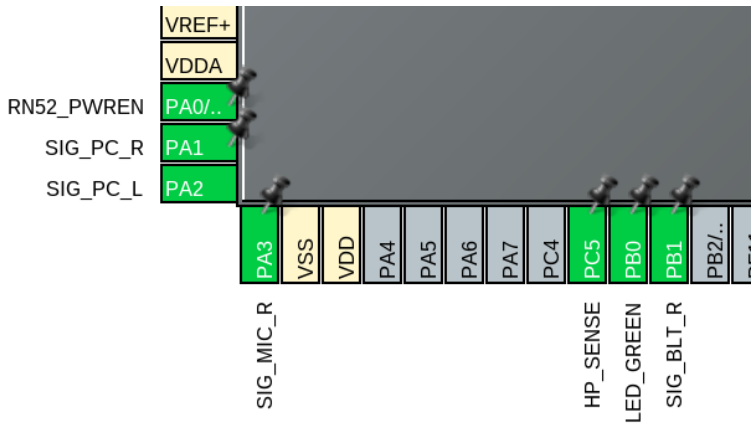

Figure 46 - Addition of ADC channels SIG_PC_R, SIG_PC_L and external sense pin in CubeMX

Figure 46 shows additions of the SIG_PC_L, SIG_PC_R and HP_SENSE signals to the DSP in CubeMX.

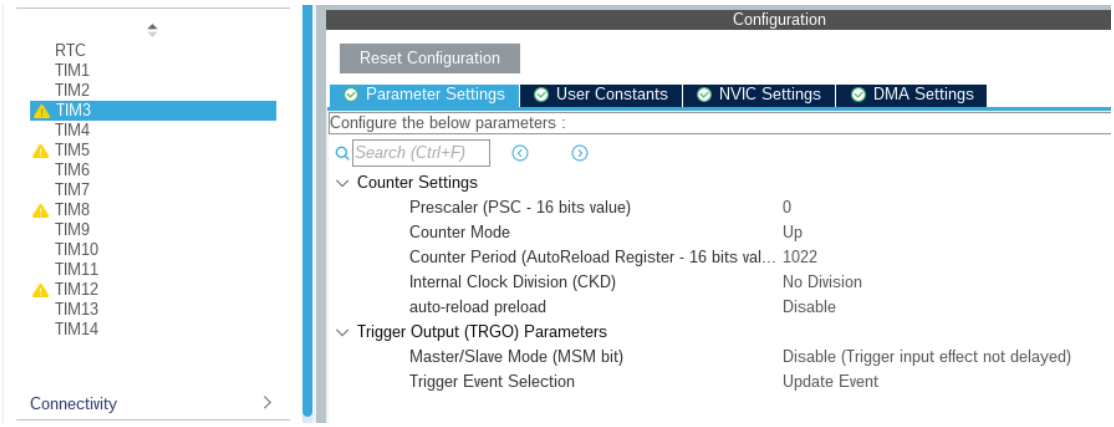

Figure 47 - CubeMX Timer configuration 88kHz sample rate ADC

Figure 47 shows configuration of timer for ADC conversion in CubeMX. Using equation (1):

$$Counter\ Period = \frac{90000000\ Hz}{88000\ Hz} \approx 1022$$

4.4. Changing ADC channels and sources on the DSP

```
audio_error_t audio_select_source(audio_source_t source){
    audio_input_source = source;

    LL_TIM_DisableCounter(AUDIO_ADC_TIMER_TRIG);

    …

    else if(source == AUDIO_SOURCE_PC){
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, 0);

        /** Configure Regular Channel*/
        LL_ADC_REG_SetSequencerRanks(ADC1, LL_ADC_REG_RANK_1, LL_ADC_CHANNEL_1); //RIGHT
        LL_ADC_SetChannelSamplingTime(ADC1, LL_ADC_CHANNEL_1, LL_ADC_SAMPLINGTIME_3CYCLES);

        LL_ADC_REG_SetSequencerRanks(ADC2, LL_ADC_REG_RANK_1, LL_ADC_CHANNEL_2); //LEFT
        LL_ADC_SetChannelSamplingTime(ADC2, LL_ADC_CHANNEL_2, LL_ADC_SAMPLINGTIME_3CYCLES);
    }

    LL_TIM_EnableCounter(AUDIO_ADC_TIMER_TRIG);

    return AUDIO_NO_ERROR;
}
```

Figure 48 – audio_select_source function selecting different channels.

The code takes the source parameter into the function and checks if the audio input source is the designated string. If the string compares with the corresponding if statement, the STM32 configures the ADC to the channel shown in figure 48.

Code to change source to PC on when external sense pin is activated:

```
void EXTI9_5_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI9_5_IRQn 0 */

  /* USER CODE END EXTI9_5_IRQn 0 */
  if (LL_EXTI_IsActiveFlag_0_31(LL_EXTI_LINE_5) != RESET)
  {
    LL_EXTI_ClearFlag_0_31(LL_EXTI_LINE_5);
    /* USER CODE BEGIN LL_EXTI_LINE_5 */
    if(LL_GPIO_IsInputPinSet(GPIOC, LL_GPIO_PIN_5)){
      audio_select_source(AUDIO_SOURCE_PC);
    }
    else if(!(LL_GPIO_IsInputPinSet(GPIOC, LL_GPIO_PIN_5))){
      audio_select_source(AUDIO_SOURCE_MICROPHONE);
    }
    /* USER CODE END LL_EXTI_LINE_5 */
  }
  /* USER CODE BEGIN EXTI9_5_IRQn 1 */

  /* USER CODE END EXTI9_5_IRQn 1 */
}
```

Figure 49 - External interrupt handler for HP_SENSE pin.

Figure 49 shows an external interrupt handler selecting between two sources if the HP_SENSE pin is either connected or disconnected. This configuration is for switching between microphone and audio jack. When the user connects to the platforms Bluetooth device the source is updated in a comparable way.

Code on startup selection of sources:

```
/**
  * @brief audio task run
  */
void audio_run(void){
    int i;
    int j;

    …

    //Error handling of these functions must be updated
    audio_init();
    audio_play();

    //Read pin for audio jack
    //If no cable is inserted, use microphone
    if(LL_GPIO_IsInputPinSet(GPIOC, LL_GPIO_PIN_5)){
        audio_select_source(AUDIO_SOURCE_PC);
    }
    if(!(LL_GPIO_IsInputPinSet(GPIOC, LL_GPIO_PIN_5))){
        audio_select_source(AUDIO_SOURCE_MICROPHONE);
    }


    while(1){
      #if(1)
            /*select source*/
            if(rn52_event_get() == 1){
                  update_source();
            }

            …


            …

      #endif
    }
}
```

Figure 50 - Main audio function

Figure 50 shows how sources are prioritized. If the audio jack cable is connected during startup process then the source is set to PC. If the cable is disconnected then it defaults to microphones on the platform. The function rn52_event_get() returns a 1 if the Bluetooth device receives music, phone call or connects with another device.

5. Discussion

Our first roadblock came when updating and porting the codebase using CubeMX and CubeIDE firmware. The codebase was old and needed an update, but it came at the cost of many hours of debugging the development environment. A function called LL_PWR_SetRegulVoltageScaling was initiated with a while-loop under main. This function blocked the serial-wire-debugger from connecting and reprograming the STM32 processor. This bug was due to CubeMX adding the check flag and has since been resolved on April 28th, with ST's update with STM32CubeMX 6.8.1 and STM32CubeIDE 1.12.1 [11]. When the code started to compile and run, we discovered that the code would not run as intended on the STM32F7 processor. The initialization of CMSIS in combination with the FPU unit did not work as intended and would in most cases slow down the processor. Later it was shown that the CMSIS libraries was not compatible with the processor at higher version. After 2 weeks of debugging, an update from an employee at ST gave us a guide on how the libraries are supposed to be configured. The implemented Arm-CMSIS library was settled with version 5.6.0 with a new catalog structure for the project. This solved the porting issue, then implementing new code was a number one priority.

When the firmware, software and chip finally would comply, changes to the code structure was easy. Using CubeMX made it easy to make changes and adding functionality to the chip. Adding the functionality of changing ports with an external interrupt was later declared in the program with the use of boilerplate code that already existed. ADC conversion is bound to a timer on the chip for ease when changing the sampling rate of the DSP. In later stages of the project the platform was changed from a sampling rate of 32 kHz to 22 kHz to save some resources. Since the STM32F7 processor was priority the STM32F4 came second. Although its weaker processor it could perform better in some areas. More research on this is needed to make a more conclusive comparison.

Circuitry design gave a couple of challenges when trying to find the specified components. In the case of audio signal processing, I2S is the best format for sending signals between digital devices. The PCM2904 is a temporary solution until it can be replaced with a device that covers that limitation. Another problem with the PCM2904, when using breadboard is the use of external crystal oscillators. These components are sensitive to all forms of disturbances and causes noise on the ground plane. Would the crystal oscillator be slightly out of position, the codec would not be recognized on the computer. It is a device which is more suited for PCB's and should be taken with great consideration when designing it.

When working with the microphone circuit, the first problem was to figure out what impedance was suitable for a computers input. Looking at the Mic click schematic [5], the circuit has a resistance R8 with a value of 100k ohms at the output. This resistance proves too much for a computer to drive, and when trying to record the sample it will not be audible. Therefore, the amplifier was needed to reset the resistance of the Mic Click circuit. Alternatively, the R8 resistance can be removed or replaced with lower value. Then the amplifier for the microphone would not be needed. Amplifying the microphone signal caused a higher THD. Although it picked up more sound, the harmonics of the signal cause a less accurate reproduction of the recorded signal. Therefor no amplifier and resolder the R8 resistance is the better approach for this application.

The limitation of jack sense not being available caused some issues as well. The incoming signal which would pass onto the amplifier becomes divided and degrades the amplitude. Not only was the change in quality audible but would cause noise in the signal. The issue will be resolved with the use of a connector with the sense pin. But even with the negatives comes the positives with instant change between source microphone and computers input.

Future work on the platform would then be implementation of a battery with a complete charge management controller. An example would be the MCP73831 from Texas Instruments. Also combining charging with a smaller and less dependent audio codec for PC audio. In later stages of the report the HS-100C from C-media was found, which comes with USB decoder, I2S, ADC and DAC but the components was not available at the time. Changing around the ports to a USB-C interface would make the device more modern and a direct volume control would increase usability of the platform. Oversampling is also an area that is needed to be explored to improve the resolution and SNR to avoid aliasing and phase distortions. Automatic source control is a desired feature which only work both ways in some cases. Switching from Bluetooth to PC and back is a feature still needing work. This with additions of turning off components when not used with use of RMS or volume threshold to activate and deactivate specified components.

6. Conclusion

To conclude this work, we must look at the goals set for this thesis. The goal of the thesis was to take the audio signals coming from a computer and communicate it to the DSP. This was done using signal amplifiers, auxiliary cables, USB, programming, and theory of signal processing. The task was accomplished with great effect with two audio circuits for both auxiliary communication and USB. Four microphone configurations was tested and analyzed. Programming and resource management was mainly done within the source code. Porting the codebase, configuration and enabling more ADC sources on the STM32 was accomplished with effect.

References

[1] Saremi, A. (2021) ”Smart headphones that can adapt to the user’s unique hearing profile”, Hörselforksningsfonden, project FA21-0017.

[2] J. H. McClellan, R. W. Schafer, M. A. Yoder, and M. A. (Mark A. . Yoder, DSP first, Second edition, Global edition. Harlow, Essex, England: Pearson, 2017.

[3] Intel Corporation, ”High Definition Audio Specification”, August 2023. [Online]. Available: https://www.intel.com/content/www/us/en/standards/high-definition-audio-specification.html

[4] B. Molin, Analog elektronik, Tredje upplagan. Lund: Studentlitteratur, 2020.

[5] Mikroe, “Mikroe Click Microphone”, August 2023. [Online]. Available: https://www.mikroe.com/mic-click

[6] Knowles Acoustics, “Slim UltraMini SiSonic Microphone Specification with Maximum RF Protection and Ultra Narrow Design”, August 2023. [Online]. Available: https://download.mikroe.com/documents/datasheets/spq0410hr5h-b.pdf

[7] Texas Instruments, “PCM2904 USB Audio Codec”, August 2023. [Online]. Available: https://www.ti.com/product/PCM2904

[8] Walt Jung, Walt Kester, Bill Chestnut Ethan Bordeaux, Johannes Horvath, “Mixed-Signal and DSP Design Techniques, Hardware Design”, August 2023. [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiBk4KctPr-AhWOv4sKHZpFDPcQFnoECD0QAQ&url=https%3A%2F%2Fwww.analog.com%2Fmedia%2Fen%2Ftraining-seminars%2Fdesign-handbooks%2Fmixedsignal_sect10.pdf&usg=AOvVaw104Phb7DsG5mey06AkIWr-

[9] Pololu, “USB 2.0 Type-C Connector”, August 2023. [Online]. Available: https://www.pololu.com/product/2585

[10] E. Nordén, ”Design & Implementation of infinite impulse response filters in headphones for hearing impaired listeners”, Bachelor thesis, Umeå University, 2023.

[11] Arbetsmiljöverket, ”Fördjupning om buller och ljud”, August 2023. [Online]. Available: https://www.av.se/halsa-och-sakerhet/buller/fordjupning-om-buller-och-ljud/

[12] STMicroelectronics Community, “STM32CubeMX 6.8.1 and STM32CubeIDE 1.12.1 released”, August 2023. [Online]. Available: https://community.st.com/s/question/0D53W00002F0Xf7SAF/stm32cubemx-681-and-stm32cubeide-1121-released