Instrumentation Design Report

Ashcharya Kela [01841948], Eleftheria Safarika [01873841]

Contents

1 Introduction

The report discusses the design, implementation, and testing of a complex impedance instrument to measure the impedance of an unknown Device Under Test (DUT).

There are several common methods utilised to perform this operation, such as the I-V, Auto-Balancing Bridge (ABB), Bridge, and RF methods. Based on the specifications outlined in Appendix A, the RF method is incompatible with the required frequency ranges as it is commonly implemented for frequencies above 1MHz. Although the Bridge method has high accuracy for a wide frequency spectrum, it has to be manually balanced. Hence, our instrument was designed through combining the benefits of the I-V and ABB method as it provides high accuracy, wide frequency spectrum and simplicity.

1.1 Member Roles

Both team members contributed equally on the project. We both worked together for the design of the PCB's architecture, schematic, and layout. During the second phase of the project, Eleftheria focused more on the final choice of components, component ordering and hardware soldering, and Ashcharya mainly focused on the digital signal processing code. We both collaborated on the PCB testing phase.

2 Measurements of Impedance

2.1 Theory

This method exploits the ability of an operational amplifier in negative feedback to keep the voltage at its negative terminal at the same potential as the bias voltage at its positive terminal (place equation here). Given that the input impedance to the op-amp is very high, there is almost no current going into the op-amp, hence all the current through the DUT will be flowing through the feedback path. Placing a resistor of well-known value there, such that the output of the op-amp doesn't saturate at the supply rails, we can use the idealised Equation (1) to calculate the expected impedance of the DUT:

$$-\frac{V_{out}}{V_{DUT}} = \frac{Z_{feedback}}{Z_{DUT}} \Rightarrow Z_{DUT} = -\frac{V_{DUT}}{V_{out}} Z_{feedback} \tag{1}$$

However, the op-amp, as is known, is not ideal, and it is limited by factors such as the Gain-Bandwidth Product. Depending on whether the DUT is resistive, capacitive, or inductive, the measured impedance becomes:

$$\frac{R_1 + R_2}{A_0} + R_1 + j\frac{\omega}{\omega_0}\left(\frac{R_1 + R_2}{A_0}\right) \tag{2}$$

$$\frac{R}{A_0\omega_0}\frac{\omega_N}{Q} + \frac{R}{A_0\omega_0}j\omega + \frac{R\omega_N^2}{A_0\omega_0}\frac{1}{j\omega} \tag{3}$$

$$-\frac{L}{A_0\omega_0}\omega^2 + \frac{R}{A_0} + j\omega\left(\frac{R}{A_0\omega_0} + L\left(1 + \frac{1}{A_0}\right)\right) \tag{4}$$

where A_0 is the DC gain, ω_N is the resonant frequency caused by the inductive behavior of the op-amp at high frequencies, ω_0 is the dominant pole frequency, internal to the op-amp, and $Q = \frac{\sqrt{\omega_0(A_0+1)RC}}{1+\omega_0 RC}$

2.2 Differentiating Series and Parallel Impedance

Circuits can have various different arrangements. It is not so useful to be able to distinguish if capacitors are in series or in parallel or whether resistors are in series or parallel since for most applications, knowing the lumped resistive and capacitive behaviour is desired. However, it is quite useful to distinguish whether a reactive component is in series or parallel with a resistive component.

To achieve this, complex impedance must be measured at a wide range of frequencies. The chosen frequencies were 1 kHz, 10 kHz and 100 kHz as suggested in the specifications.

Table 1 below shows impedance equations for different arrangements and their respective approximations for low and high frequencies.

Arrangement	Impedance Equation	Low Frequency	High Frequency
RC Parallel	$Z = \frac{1}{\frac{1}{R}+j\omega C}$	$Z \approx R$	$Z \approx 0$
RC Series	$Z = R + \frac{1}{j\omega C}$	$Z \approx \frac{1}{j\omega C}$	$Z \approx R$
RL Parallel	$Z = \frac{j\omega RL}{R+j\omega L}$	$Z \approx 0$	$Z \approx R$
RL Series	$Z = R + j\omega L$	$Z \approx R$	$Z \approx j\omega L$
LC Parallel	$Z = \frac{j\omega L}{1+(j\omega)^2 LC}$	$Z \approx 0$	$Z \approx \infty$
LC Series	$Z = j\omega L + \frac{1}{j\omega C}$	$Z \approx -j\infty$	$Z \approx j\infty$

Table 1: Impedance Equations and Approximations for different circuit arrangements

The approximations in Table 1 are based on the assumption that the value of $R \ll C$. Based on the equations provided in Table 1, the impedance of RC and RL for different circuit arrangement varies with frequency. Hence, by measuring the impedance at different frequencies, it can be deduced what arrangement is present by the following:

1. The real-part of impedance determines the resistance value. It is expected to remain constant for an ideal resistor assuming the ABB amplifier GBP is high enough such that the single-order low-pass pole (due to compensation capacitor) is not observed in the output of the ABB amplifier.

2. The reactance of the circuit is determined by the imaginary component of impedance. A positive reactance suggests inductive behaviour and a negative reactance suggests capacitive behaviour.

3. With increasing frequency, if the impedance approaches 0 Ω, the circuit is likely to be represented as RC in parallel. However, if at low frequency the impedance is 0 Ω, and at high frequency the impedance is approximately R, it suggests a parallel RL arrangement. This is because inductance behaves as 'open-circuit' in parallel with resistor at high frequency and thus, impedance only seems resistive. Similarly, series arrangement can be determined through the method.

The functional algorithm created to implement this can be seen in Appendix **??**.

3 PCB Design

3.1 PCB Schematic and Design Decisions

The PCB schematic in Appendix 7 shows the connections of our circuit as diagrammatically shown in Figure 1.

Despite the DAC from the microcontroller having an internal buffer, an external buffer was added to ensure that the loading impacts on current created by the ABB opamp doesn't impact the integrity of the sinusoid signal output from the DAC.

The series resistance placed at the output of the buffer was to limit the current required by the DAC since it had a supply limit of 15mA.

Instrumentation Amplifiers - This component was utilised due to its high common-mode rejection ratio (CMRR) in addition to having high input impedance. The selection of AD8220BRMZ was made due to its low input bias current (25pA) as well as the operating range for input and output voltages. Since our DAC and ADC can only take a maximum of 3.3V, all our signals were centered around 1.65V to allow maximum head and leg room. It was quite common for the instrumentation amplifiers to have the input and output voltage range restricted to VCC and VSS ± a value. The input operating range for AD8220 was between -0.1V and 3V whilst the output was between 0.15V to 4.85V; Since the DUT nor the range resistors will have a voltage amplitude greater than 1.3V centered around 1.65V, no DC level shifting was required, making the circuitry simpler. Moreover, to avoid a secondary gain stage, the input signal from the DAC was designed to provide 1.3V across the DUT for all frequencies. Hence, unity gain stability was an important factor to avoid ringing effects as the gain desired from the inamp was 1 (achieved

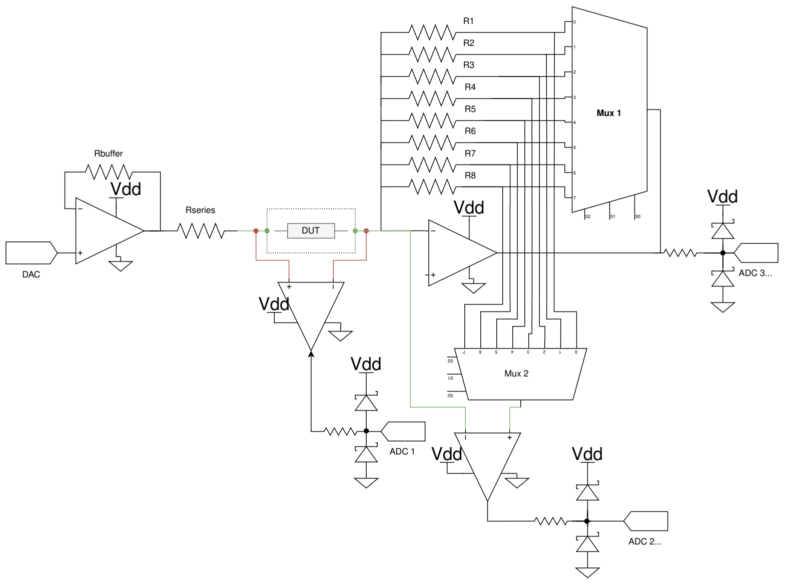


Figure 1: Diagram of our instrument design

by making R_g an open-circuit). Since, inamp gain was fixed by hardware (not variable by software) and that the frequency specifications required measurements up to 1MHz, the minimum gain bandwidth product desired was 1MHz. The AD8220 had a GBP of 1.5MHz. Finally, at high frequencies, the slew rate of the opamp becomes important as a low slew rate means that the output of the opamp cannot change as rapidly to the changing inputs, thereby creating distortions. Since the slew rate was $2\mu s$, the output voltage of the inamp could change by 2V at a frequency of 1MHz.

Auto-balancing Bridge Opamp - The component chosen has a gain bandwidth product of 30MHz, much higher than the maximum frequency of 1MHz at which the DUT will be tested at. This is due to the compensation capacitor within the opamp that is placed for stability purposes. At high frequencies, the opamp acts as a low-pass filter, restricting the gain available in addition to causing an inductive behaviour at the input of the opamp. This inductive behaviour is observed as the phase changes at high frequencies. Since phase changes a decade before the corner frequency, an opamp with a bandwidth of minimum 10MHz was required to accurately measure a 1MHz signal else the internal inductive behaviour of the opamp can make DUT's behave more inductive at high frequencies than what they actually are. Finally, having a low input bias current of 2pA was significant especially for DUT's that were high impedance. For a DUT of $1M\Omega$, the current through it for 1.3V amplitude sinusoid would be $1.3\mu A$. If the input bias current was in the nA, a large portion of current would be fed into the ABB amplifier instead of the range resistor, causing the virtual ground (with DC offset of 1.65V) to become unstable.

Analogue Multiplexer - Analogue MUX was utilised to perform the switching operation of the range resistors. A secondary MUX was used to obtain the voltage on the right-hand side of the range resistor and input it into the inamp. This provided a more accurate measurement of voltage across range resistor and thus, the current flowing in the feedback loop because it didn't include MUX1, whose on-resistance is ill-defined and can vary with various factors such as temperature and frequency. The on-resistance of MUXes being ill-defined becomes an issue for low DUT impedance measurements as it can cause the gain to be greater than 1 and possibly saturating the output of the ABB.

Decoupling Capacitors - These were added as close to the power supplies as possible in layout to remove any transients that may occur, ensuring an almost DC voltage. This is particularly important in opamps as large transients can cause an oscillatory motion for the opamp.

Schottky Diodes - Schottky diodes were used as protection for the ADC measurements. Although the amplitude of the input sine-wave is 1.3V peak-to-peak, a wrong range resistor being switched on can cause a high gain, saturating the output of the ABB at 5V. Due to the ADC having a limit of 3.3V, this can break the ADC. Hence, Schottky diodes were used to prevent voltages greater than 3.6V and voltages lower than -0.3 from entering the ADC. Ideally, Schottky diodes should be used along with transistors to control the amount of current as when the Schottky diodes are forward biased, it can supply too much current to the power supply, creating unwanted behaviours. A Schottky diode with low forward voltage and low capacitive behaviour was chosen. The lower the forward voltage of the diode, the closer the voltage protection to the power supplies value.

3.2 PCB Layout

The layout of the PCB was design to have a shield for the STM32F446RE such that it can be connected to the PCB itself through header pins. This helps minimise noise that would have been present with wires and coupling effects. The traces that connected the 'current' banana plugs (that externally connected the DUT) to the ABB amplifier were ensured to be of equal lengths and as short as possible to avoid phase offsets and noise. This can be seen in Appendix Figure 8. The same concept was applied to 'Voltage' banana plugs connecting to the inamps. The banana plugs were arranged for the the connection of the four Kelvin measurement wires.

Both the top and bottom layer of the PCB was a ground plane to which signals were routed. The top-place consisted of all the analogue and digital signals of interest where the bottom plane was predominantly used for routing power signals and switch-select lines for the MUXes. Vias have been placed in appropriate places to prevent cross-talk between signals in addition to providing the power from the ground plane. A **further improvement** to our design could add a stitching of ground vias.

The trace width was intentionally enlarged for connections to the 'current' banana plugs as large currents could flow for low impedances. The rest of the trace width was done such that they matched the pads of the 0603 resistors to avoid changing widths constantly which can add noise and phase. Trace bends were minimised in angles to 45 degrees as well. The pins used by the microcontroller were chosen such that the distances of the traces from them were minimised and allowed for a cleaner design.

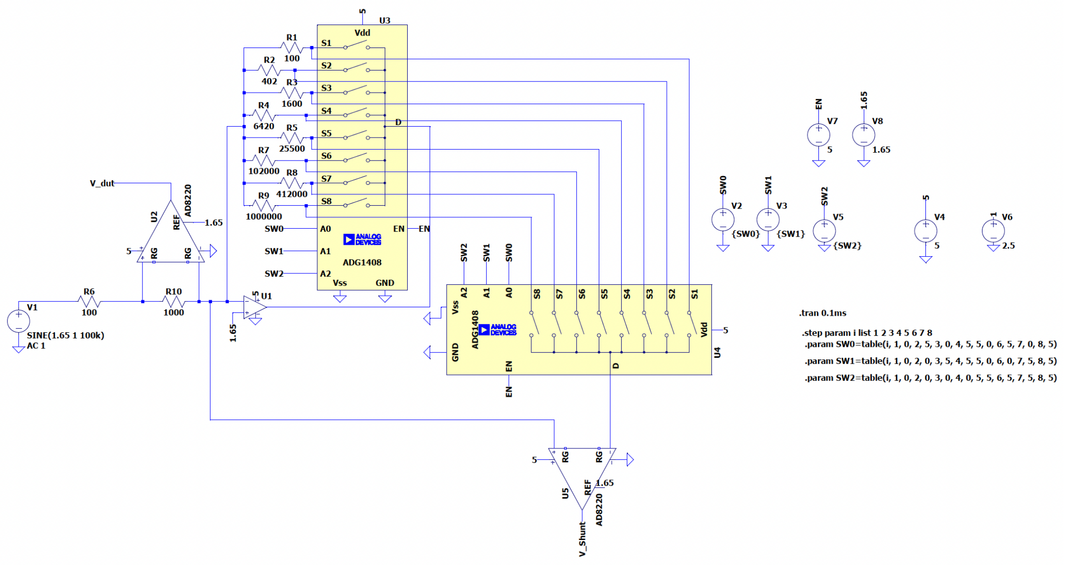
The layout of the PCB can be found in Appendix Figures 8 and 9.

4 Bill of Materials

The Bill of Materials can be found in Appendix E.

5 LTSpice Simulation

Figure 2 shows the schematic for the instrument on LTSpice. Different models for multiplexers, ABB opamp, and instrumentation amplifiers are used with modified specifications to reflect our component choices.



```
.tran 0.1ms
.step param i list 1 2 3 4 5 6 7 8
.param SW0=table(i, 1, 0, 2, 5, 3, 0, 4, 5, 5, 0, 6, 5, 7, 0, 8, 5)
.param SW1=table(i, 1, 0, 2, 0, 3, 5, 4, 5, 5, 0, 6, 0, 7, 5, 8, 5)
.param SW2=table(i, 1, 0, 2, 0, 3, 0, 4, 0, 5, 5, 6, 5, 7, 5, 8, 5)
```

Figure 2: Auto-balancing Bridge Instrumentation Amplifier Schematic

The input sine wave was varied in amplitude and frequency, to investigate how the output's amplitude and shape was affected, while simultaneously sweeping through the feedback resistors. The .param commands seen on the schematic perform the function of toggling the control switches of the two MUXes. Hence, for an example value of DUT impedance, some switches will result in the gain of the ABB being less than one, and some in higher than one. We ideally want the DUT impedance at any frequency value to be equal to or higher than the value of the feedback resistor, so as to avoid the signal exceeding 3.3V when inputting max amplitude sine wave into the circuit. The figure below shows how different range resistors produce different voltages, as they change the gain of the ABB. Having 8 different range resistors allows us to measure a wide range of impedances.

The range resistors were chosen in a logarithmic manner of multiples of four. This is because with an amplitude of 1.3V, the lowest voltage drop across the DUT would be $\frac{1}{4}$ the value across the range resistor (i.e. the lowest voltage across the DUT is about 0.3V. Having a voltage drop across the DUT below this would result in poor SNR performances. Moreover, low voltages up until 0.3V were perfectly measurable by the instrumentation amplifier and the ADC since the ADC's resolution is 1 bit per 0.8mV.

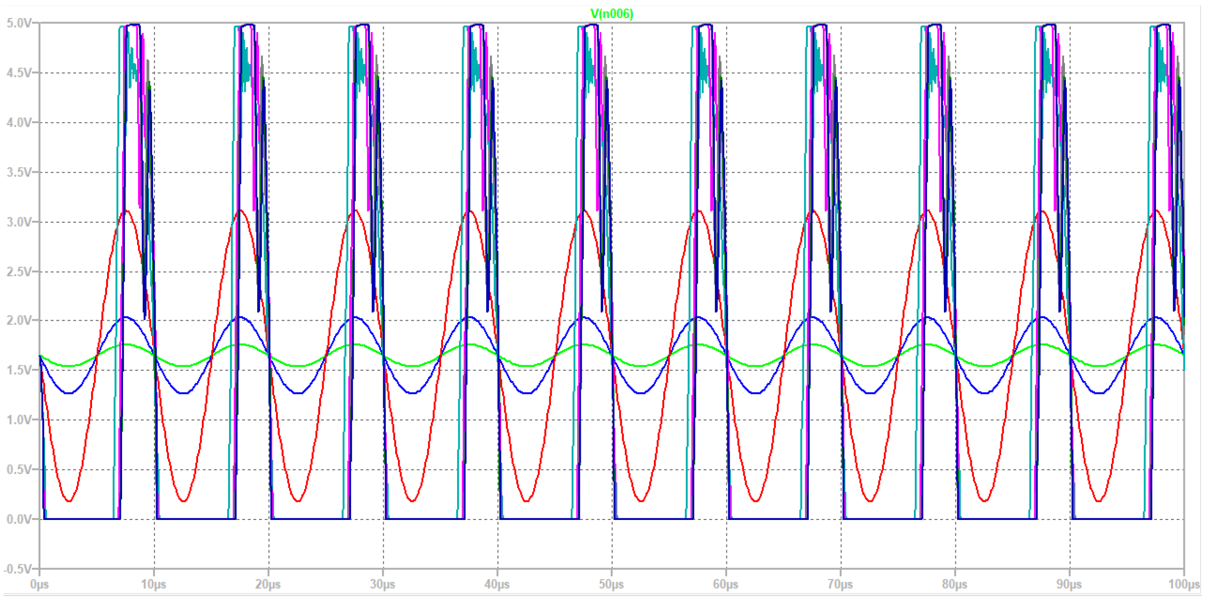


Figure 3: Output voltage of ABB with different range resistors

Figure 3 shows the output of the ABB for the different range resistors selected every time. We see that only a few of the switch configurations are giving an output waveform which is not distorted and which is within a usable range.

6 Testing

The assembled instrument is shown in Figure 4.

6.1 Measurements

The complex impedance instrument created was able to successfully measure the impedance of a resistor for frequencies between 1 kHz to 100 kHz with an accuracy of ±2%. Moreover, the instrumentation was also calibrated to detect the impedance of capacitance with a phase error of ±15°. Figure 5a and 5b show the output of the ADC for voltages measured across both the DUT and the desired range resistor. Serial plotter on Arduino was used to display this.

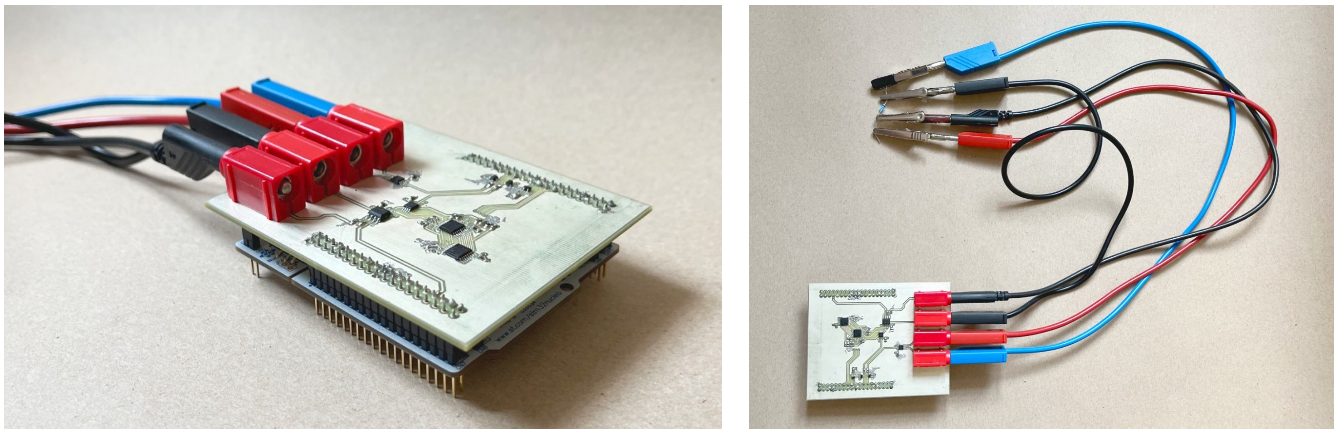
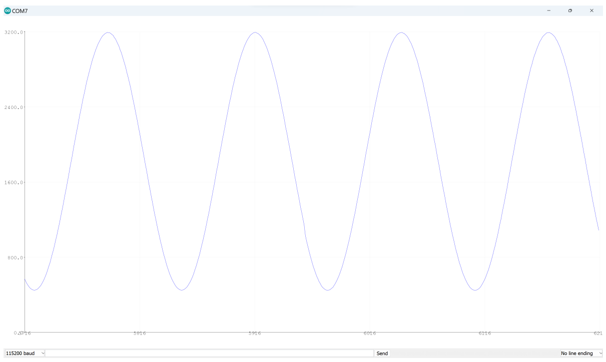
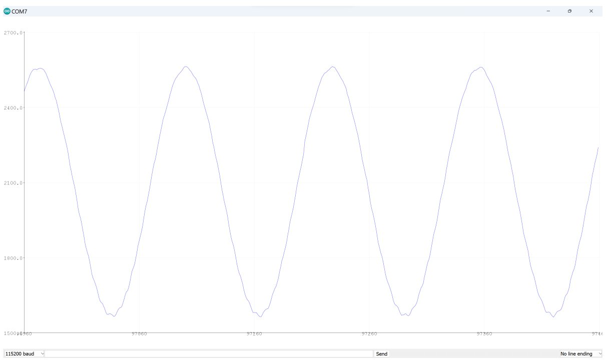


Figure 4: Images of the final physical instrument from different angles



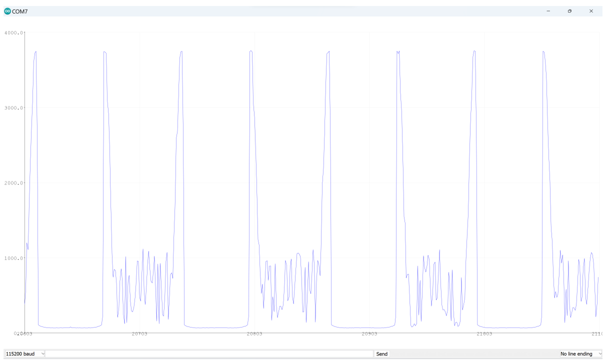
(a) Voltage across DUT



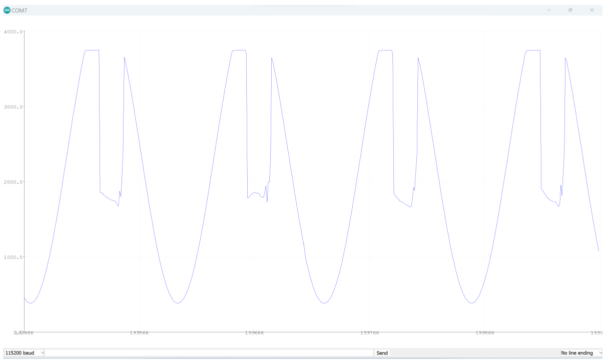
(b) Voltage across range resistor

Figure 5: Voltage across the DUT and across the range resistor respectfully

The figures below show how incorrect selection of range resistor has on the voltage across the feedback resistor; The waves become distorted and saturate.



(a) Too large range resistor



(b) Slightly large range resistor

Figure 6: Inadequate feedback resistor choices

7 Computation of Impedances

7.1 Auto-ranging Resistors

8 Range resistors were chosen that varied logarithmically by four. The values of resistors were chosen such that the minimum gain that any DUT impedance experienced was $\frac{1}{4}$ when the right range resistor was chosen.

The algorithm was implemented such that the lower value resistors would switch on first and then increment. This is to ensure that our gain isn't beyond 1.5 as this can lead the ABB output to saturate at 5V and potentially damage our ADC that have a limit of 3.3V (despite Schottky diodes acting as a safety mechanism).

The algorithm was based on finding the voltage of the peak of the amplitude and comparing it to $0.25 * 1.3 = 0.325$. Since we have a DC offset of 1.65V, wee want to compare the voltage across the range resistor to $1.65 + 0.325 = 1.925$. If the peak is less than 1.925V, then the range resistor selection is too low and we need to increment. Since we are incrementing from low range resistor to high, the first selection that provides a peak voltage of less than or equal to 1.925 is desired. The code for this can be found in Appendix G.

7.2 Goertzel Algorithm

The Goertzel Algorithm was used to find the magnitude and phase of the voltage across DUT and range resistor. The Goertzel Algorithm undergoes a process of filtering as it is implemented as an FIR filter and this helps to remove the sampling frequency distortions that may be present. Moreover, to improve the Signal-to-Noise ratio of the coefficients of the Goertzel, Hamming window was applied to the signal which is particularly beneficial to remove any startup noise. The formula for Hamming Window that was applied is:

$$Hamming = 0.54 - 0.46cos(\frac{2\pi n}{N-1}) \tag{5}$$

where N = number of samples and n is the discrete time index. Refer to Appendix B and Appendix F for the code.

8 Digital Signal Processing code and STM32

Two ADC and DAC were utilised in this design. DAC1 was utilised to create the input sine wave going into the DUT through a LUT whilst DAC2 was utilised to generate a constant bias offset voltage of 1.65V. ADC1 was utilised to encode voltage across the DUT into digital values whilst ADC2 encoded voltage across the range resistor. It is important to note that since both ADC were used simultaneously, that their clocks were synchronised in order for the phase measurements to be accurate else, a resistor could be denoted as a reactive component and vice versa. Both the ADCs and the DAC1 were triggered by the positive edge of timer 2 whilst the DAC2 had no trigger as its purpose was simply a DC offset.

Moreover, the data was transmitted serially through the UART interface asynchronously with a baudrate of 115200. This was utilised to view the waveforms as well as transmit the output impedance result into Serial Monitor/Plotter on arduino IDE.

The instantiation of the ADCs, DACs and UART interface in code are shown in H.

8.1 Pin Configuration and the STM32 IDE

Pin Channel	Pin Number	Pin Name	Pin Type	Signal	Direction
CN10	13	PA6	Signal	ADC1	Input
CN10	15	PA7	Signal	ADC2	Input
CN7	32	PA4	Signal	DAC1	Output
CN10	11	PA5	Reference Signal	DAC2 (1.65V)	Output
CN10	25	PB10	Signal (Digital)	SW0	Output
CN10	23	PA8	Signal (Digital)	SW1	Output
CN10	21	PA9	Signal (Digital)	SW2	Output
CN10	24	PB1	Signal	ABB_ADC	Input
CN10	20	GND	Power	GND	N/A
CN10	9	GND	Power	GND	N/A
CN7	8	GND	Power	GND	N/A
CN7	20	GND	Power	GND	N/A
CN7	19	GND	Power	GND	N/A
CN7	22	GND	Power	GND	N/A
CN10	32	AGND	Power	GND	N/A
CN7	18	+5V	Power	5V	N/A
CN7	16	+3V3	Power	3.3V	N/A

Table 2: List of pin assignments performed on the microcontroller, pin and signal names

where:

- ADC1: output of the instrumentation amplifier performing the voltage measurement
- ADC2: output of the instrumentation amplifier performing the current measurement
- DAC1: generates the sine wave input signal of varying frequency, which gets buffered and then enters the DUT
- DAC2: generates a reference signal of 1.65V for biasing our signals around it, allowing for maximum output voltage swing with equal leg and head room (between 0 and 3.3V)
- SW0, SW1, SW2: MUX switching signals for the 8-switch MUXes, where SW0 is the least significant digit and SW2 is the msot significant digit
- ABB_ADC: the output of the ABB amplifier, which is available for signal processing and comparison between its value and the voltage and current measurements by the instrumentation amplifiers

9 Appendix

A Specifications Required

1. **Frequency Coverage** - 100 Hz to 100kHz: internal op-amp compensation and in-amp effects (such as bias currents) should not influence the response of our instrument to an input sine wave.

2. **Impedance Range** - 10Ω to 1MΩ: our instrument must correctly identify impedances within this range for the frequency range stated above, which may contain resistive, capacitive, as well as inductive components. Moreover, the instrument must be able to correctly identify the arrangement of the different resistive and reactive components that form the impedance.

3. **Phase Accuracy** - less than 5°: a complex impedance will cause a phase shift between the current and the voltage through our DUT; our instrument should be able to identify this phase shift correct to less than 5°

B Goertzel's Algorithm Implementation Proof

On our microcontroller we have a sampled time signal on which we can compute the Discrete Fourier Transform (DFT), with formula:

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{-\frac{2\pi kn}{N} j} \tag{6}$$

(check whether there is anything else missing)

where:

- k is the frequency domain bin number, which corresponds to the frequency investigated at every step
- $X(k)$ gives the magnitude at the k^{th} frequency - we usually have an infinite number of frequencies to compute the DFT for
- N is the number of samples in the time domain

If we multiply Equation (1) by $e^{2\pi k \frac{N}{N} j} = 1$, since k is an integer between 0 and $N-1$, we get:

$$X[k] = e^{2\pi k \frac{N}{N} j} \sum_{n=0}^{N-1} x[n] e^{-\frac{2\pi kn}{N} j} \tag{7}$$

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi k(\frac{n-N}{N})} \tag{8}$$

Since we are dealing with discrete frequencies, the z-transform applies, which has the form:

$$Z\{x[n]\} = \sum_{n=0}^{\infty} x[n] z^{-n}, \qquad and \qquad Z^{-1}\{1\} = \delta(n) \tag{9}$$

where $\delta(n)$ is the Dirac-Delta function in the discrete time domain.

Since $e^{2\pi k \frac{N}{N} j} = 1$, and knowing that multiplication in the frequency domain is equivalent to convolution in the time domain:

$$x[n]\delta[n] \Leftrightarrow X(k) \times e^{2\pi k \frac{N}{N} j} \tag{10}$$

Given that $\delta[n]$ is the impulse response of the system, we can also denote it as $h[n]$. From Equation (3) we can say:

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{\left(-j\frac{2\pi kn}{N} + j\frac{2\pi kN}{N}\right)} = \sum_{n=0}^{N-1} x[n] e^{-\left(j\frac{2\pi kn}{N} - j\frac{2\pi kN}{N}\right)}$$

Assuming $x[n] = x[n]$ and $h[n] = e^{j\frac{2\pi kn}{N}u[n]}$, as $Z\{u[n]\} = \sum_{n=0}^{N-1} e^{-j\frac{2\pi kN}{N}} = \sum_{n=0}^{N-1} e^{-j2\pi k}$. Hence:

$$X(k) = \sum_{n=0}^{N-1} x[n]h_k[n] \tag{11}$$

which implies that to find the magnitude of the k^{th} frequency we need to sum up all known samples n. Since $x[n]h[n-m] = \sum_{m=-\infty}^{+\infty} x[m]h[n-M] = y_k[n]$, we can express Equation (4) as:

$$y_k[n] = \sum_{m=-\infty}^{+\infty} x[m]h[n-m] = \sum_{m=0}^{N-1} x[m]e^{j2\pi k\left(\frac{n-m}{N}\right)}u(n-m) \tag{12}$$

Comparing Equation (3) and Equation (6), we can see that $X(k) = y_k[N]$, when $n = M$, which is the N^{th} sample of the convolution. When $n = N$, Equation (6) becomes:

$$y_k[N] = \sum_{m=0}^{N-1} x[m]e^{j2\pi k\frac{N-m}{N}}u(N-m) = \sum_{m=0}^{N-1} x[m]e^{-j2\pi k\frac{m-N}{N}}u(N-m)$$

$$X(k) = \sum_{n=0}^{N-1} x[n]e^{-2j\pi k\left(\frac{n-N}{N}\right)}$$

We can see, therefore, that $y_k[n]$ and $X(k)$ are equivalent. Moreover, $u(N-m)$ is first a shift N samples to the left and then a reflection about the y-axis. Hence, $X(k) = y_k[n = N]$ for $k = 0, 1, ..., N-1$, and we can take the output sample in time $n = N$ of an IIR system with impulse response $h_k[n]$. To find the transfer function for $h_k[n]$:

$$H_k(z) = \sum_{n=-\infty}^{\infty} h_k[n]z^{-n} = \sum_{n=-\infty}^{\infty} e^{j\frac{2\pi nk}{N}}u[n]z^{-n} = \sum_{n=0}^{\infty} e^{j\frac{2\pi kn}{N}}z^{-n} = \sum_{n=0}^{\infty}\left(e^{\frac{2\pi kj}{N}}z^{-1}\right)^n$$

The final form is a geometric series with common factor $e^{\frac{2\pi kj}{N}}z^{-1}$. For $|r| < 1$, the sum of the geometric series is $\frac{u_1}{1-r}$, where $u_1 = \left(e^{j\frac{2\pi k}{N}}z^{-1}\right) = 1$. Hence, $H_k(z) = \frac{1}{1-e^{j\frac{2\pi k}{N}}z^{-1}}$, where the Goertzel algorithm only converges for $|z| > 1$. Overall:

$$Y(z) = H_k(z)X(z) = \frac{X(z)}{1 - e^{\frac{2\pi kj}{N}}z^{-1}}$$

This leads to the difference equation $Y(z) - Y(z)e^{\frac{2\pi kj}{N}}z^{-1} = X(z) \Rightarrow Y(z) = Y(z)e^{\frac{2\pi kj}{N}}z^{-1} + X(z)$. Taking the inverse z-transform and noting that $Z^{-1}\{A(z)z^{-1}\} = a[n-1]$, we get: $y[n] = x[n] + y[n-1]e^{\frac{2\pi kj}{N}}$. This is complex but can be simplified:

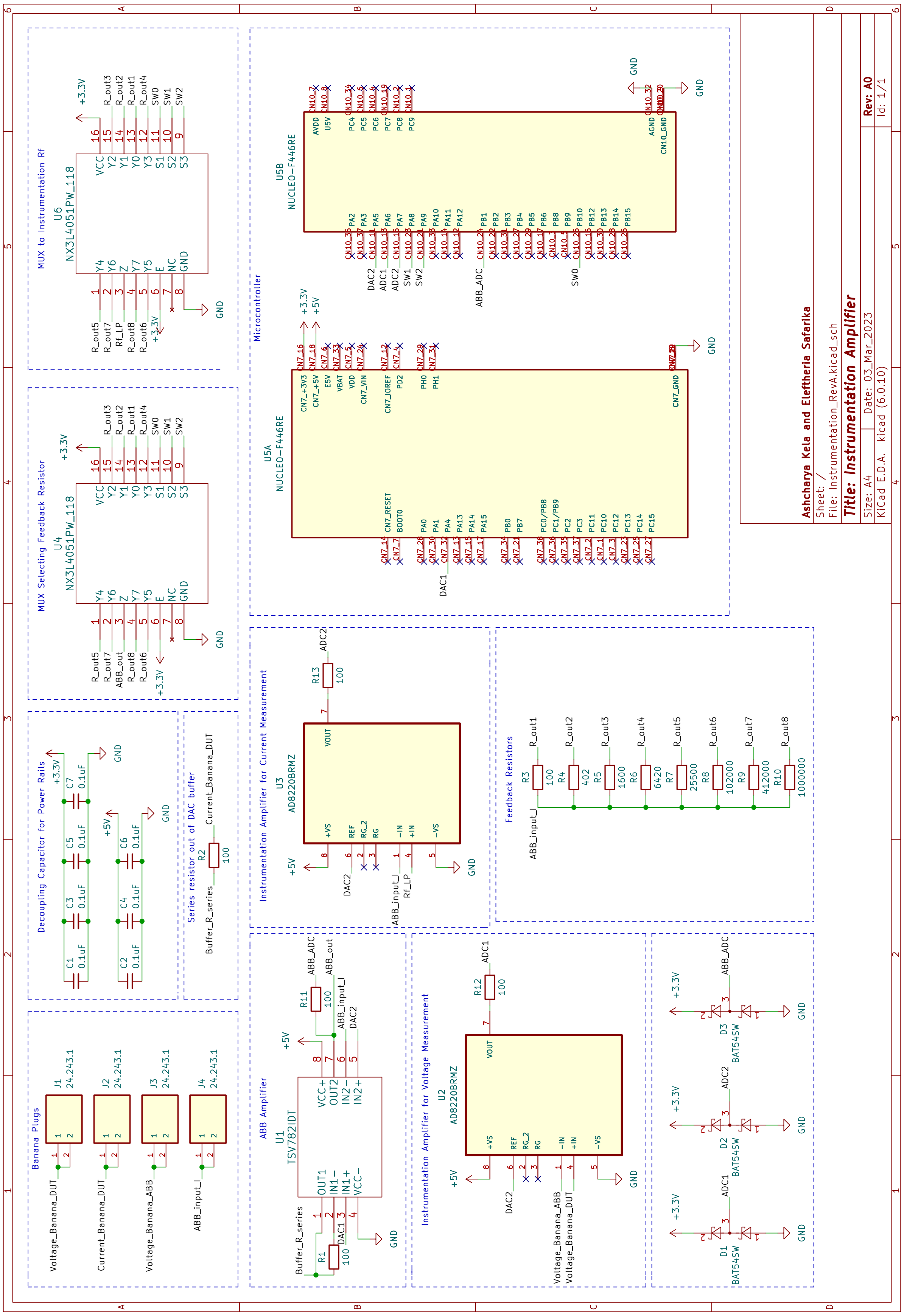
$$H_k(z) = \frac{1}{1 - e^{j\frac{2\pi k}{N}}z^{-1}} \times \frac{1 - e^{-j\frac{2\pi k}{N}}z^{-1}}{1 - e^{-j\frac{2\pi k}{N}}z^{-1}} = \frac{1 - e^{-j\frac{2\pi k}{N}}z^{-1}}{1 - 2\cos\left(\frac{2\pi k}{N}\right)z^{-1} + z^{-2}}$$

$$y_k[n] = x[n] - x[n-1]e^{-j\frac{2\pi k}{N}} + 2\cos\left(\frac{2\pi k}{N}\right)y_k[n-1] - y_k[n-2]$$

at $n = 0$, $x[-1] = 0$ and $y[-2] = 0$, $y[-1] = 0$. The whole system can be represented as states:

$$s[n] = x[n] + 2\cos\left(\frac{2\pi k}{N}\right)s[n-1] - s[n-2], \quad y_k[n] = s[n] - e^{-j\frac{2\pi k}{N}}s[n-1], \quad s[-1] = 0, \quad s[-2] = 0$$

C Schematic Diagram of the Instrument



MUX to Instrumentation Rf
U6
NX3L4051PW_118
MUX Selecting Feedback Resistor
U4
NX3L4051PW_118
Microcontroller
U5B
NUCLEO-F446RE
U5A
NUCLEO-F446RE
Decoupling Capacitor for Power Rails
Series resistor out of DAC buffer
Instrumentation Amplifier for Current Measurement
U3
AD8220BRMZ
Feedback Resistors
Banana Plugs
ABB Amplifier
U1
TSV782IDT
Instrumentation Amplifier for Voltage Measurement
U2
AD8220BRMZ
D1 BAT54SW
D2 BAT54SW
D3 BAT54SW
Ashcharya Kela and Eleftheria Safarika
Sheet: /
File: Instrumentation_RevA.kicad_sch
Title: Instrumentation Amplifier
Size: A4
Date: 03_Mar_2023
Rev: A0
KiCad E.D.A. kicad (6.0.10)
Id: 1/1

Figure 7: Schematic of the impedance measurement - made on KiCad

D Layout of the Instrument

D.1 Front-Plane Copper

All analogue and digital signals were flowing through here.

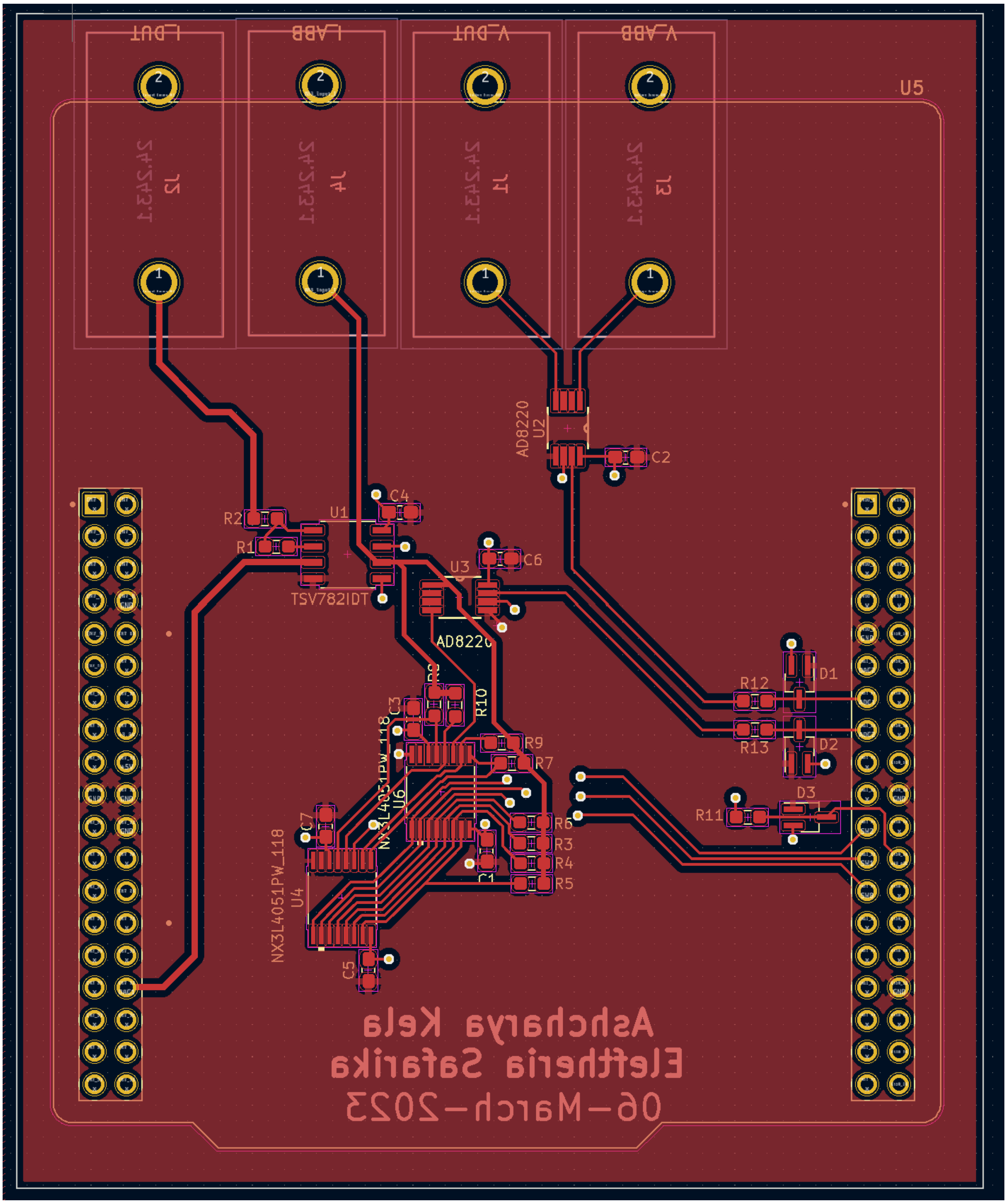


Figure 8: Front Copper Layout of impedance instrument - made on KiCad

D.2 Bottom/Ground Plane Layout of impedance instrument - made on KiCad

Mainly used for the power signals.

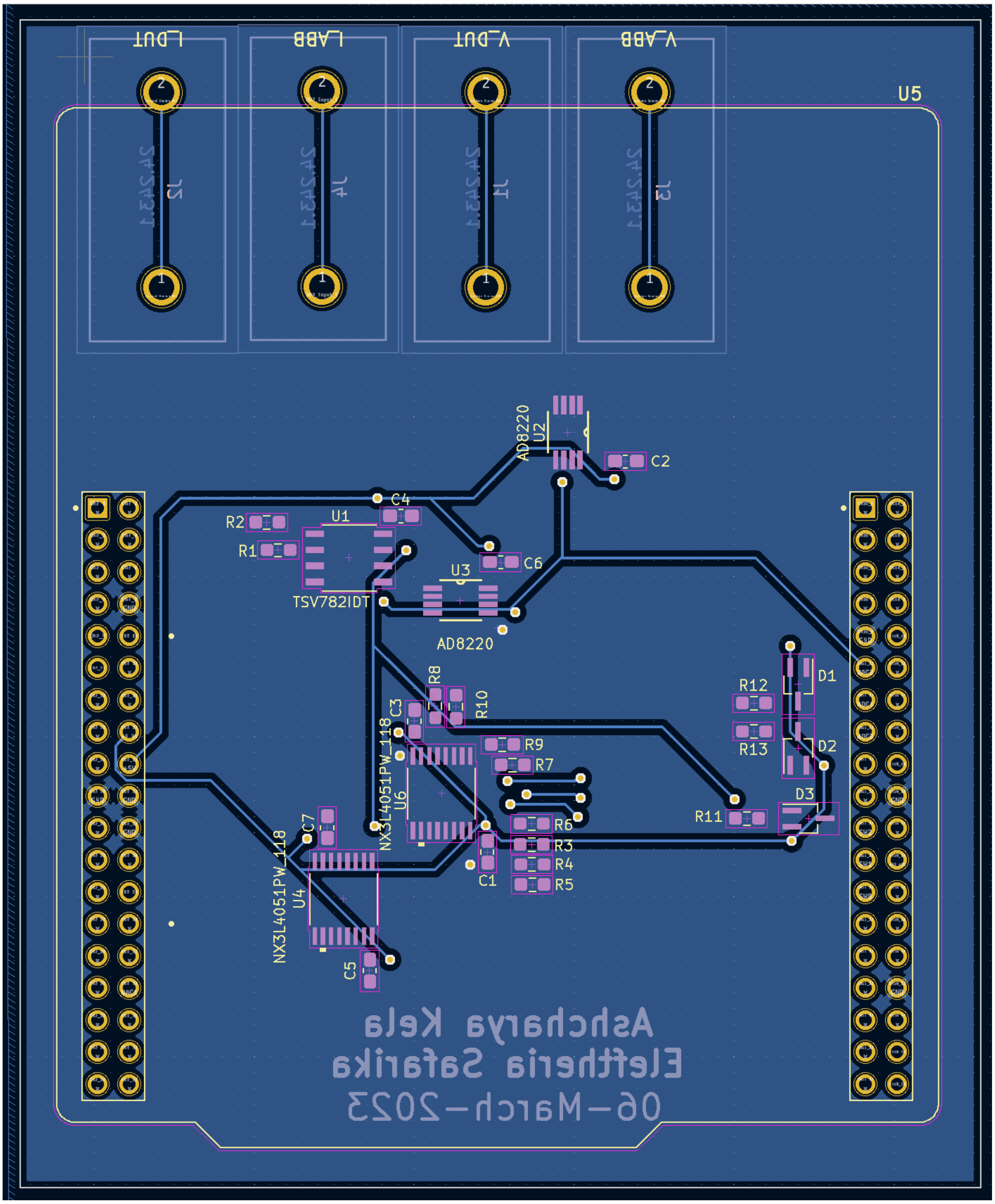


Figure 9: The layout of our instrument made on KiCad completed with the routings

E Bill of Materials

Component Name	Component Type	Quantity	Manufacturer	Supplier	Price per quantity	Total component price	Supplier Part Number
AD8220	Instrumentation Amplifier	2	Analog Devices	Mouser	9.39	18.78	584-AD8220BRMZ
TSV782IDT	Operational Amplifier	2	STMicroelectronics	Mouser	2.60	5.2	511-TSV782IDT
NX3L4051PW-Q100J	MUX	4	Nexperia	Mouser	1.31	5.24	771-NX3L4051PW-Q100J
RT0603BRD10100RL	100Ω Resistor	4	YAGEO	Mouser	0.273	1.092	603-RT0603BRD10100RL
RT0603BRD07402RL	402Ω Resistor	2	YAGEO	Mouser	0.282	0.564	603-RT0603BRD07402RL
RT0603BRD071K6L	1.6kΩ Resistor	2	YAGEO	Mouser	0.314	0.628	603-RT0603BRD071K6L
RT0603BRD076K42L	6.42kΩ Resistor	2	YAGEO	Mouser	0.273	0.546	603-RT0603BRD076K42L
RT0603BRD0725K5L	25.5kΩ Resistor	2	YAGEO	Mouser	0.273	0.546	603-RT0603BRD0725K5L
RT0603BRD07102KL	102kΩ Resistor	2	YAGEO	Mouser	0.265	0.53	603-RT0603BRD07102KL
RT0603BRD07412KL	412kΩ Resistor	2	YAGEO	Mouser	0.248	0.496	603-RT0603BRD07412KL
RT0603BRD071ML	1MΩ Resistor	2	YAGEO	Mouser	0.248	0.496	603-RT0603BRD071ML
UMK107ABJ104KAHT	Decoupling Capacitor	8	TAIYO YUDEN	Mouser	0.091	0.728	963-UMK107BJ104KAHT
BAT54SWF	Schottky Diode	6	Nexperia	Mouser	0.149	0.894	771-BAT54SWF
24.243.1/2	Banana plug sockets	4	Multicomp	Farnell	0.732	2.928	1698982/3
929852-01-19-10	Header pins	2	3M Electronics	Mouser	4.85	9.7	517-929852-01-19-10

F Goetzel Algorithm Software application

```
// Implementing the Goetzel Theorem
```

```c
float sample_rate_FREQ = 3*SINE_FREQ;
#define PI 3.14159
#define SAMPLING_RATE   sample_rate_FREQ
#define TARGET_FREQUENCY  SINE_FREQ
#define N 256 //Size of ADC. Number of samples.

//Global Variables
float s_n, s_n_1, s_n_2, omega, cos_omega, twocos_omega;
float real_part1, imaginary_part1;
float phase1, magnitudeSquared1, magnitude1;
int k;

// Defining arrays
uint32_t data1[ADC_BUFFER_SIZE];

void reset_Goetzel() {
  s_n_1 = 0;
  s_n_2 = 0;
  s_n = 0;

}

void Geotzel_Algorithm(uint32_t adc_buffer_x[]) {

  // initialize variables
  k = 0.5 + (N * TARGET_FREQUENCY) / SAMPLING_RATE;
  omega = (2 * PI * k) / N;
  cos_omega = cos(omega);
  twocos_omega = 2*cos_omega;

  //Reset previous values
  reset_Goetzel();

  // Implement hamming window
  float hamm_window;
  for(int i=0; i<ADC_BUFFER_SIZE; i++){
    hamm_window = 0.54 - (0.46 * cos((2*PI*i)/(ADC_BUFFER_SIZE-1)));
    data1[i] = adc_buffer_x[i]*hamm_window;
  }

  // For loop steps through all the ADC values in the array to compute s_N
  for (int i = 0; i < 256; i++) {
    s_n = data1[i] + (twocos_omega * s_n_1) - s_n_2;
    s_n_2 = s_n_1;  // s_n_2 is now the delayed version by 1 sample of s_n_1
    s_n_1 = s_n;
  }

  real_part1 = (cos_omega*s_n_1) - s_n_2;
  imaginary_part1= (sin(omega) * s_n_1);

  // Deriving results from the Goetzel theorem
  phase1 = atan2(imaginary_part1, real_part1);
  magnitudeSquared1 = (real_part1 * real_part1) + (imaginary_part1 * imaginary_part1);
  magnitude1 = sqrt(magnitudeSquared1);
}
```

G Auto-ranging Range Resistors Code

```c
int switch_resistor =0;
int max_val;
int selection;

```

```
void initial_MUX(){
  switch_resistor =0;
  max_val = 0.00;
}
void selecting_MUX(){
  selection = 1;
  while(selection == 1){
    switch_resistor++;

    switch (switch_resistor){
              case 1:
              HAL_GPIO_WritePin(GPIOA, SW2_Pin, GPIO_PIN_RESET);
              HAL_GPIO_WritePin(GPIOA, SW1_Pin, GPIO_PIN_RESET);
              HAL_GPIO_WritePin(SW0_GPIO_Port, SW0_Pin, GPIO_PIN_RESET);
            break;

              case 4:
              HAL_GPIO_WritePin(GPIOA, SW2_Pin, GPIO_PIN_RESET);
              HAL_GPIO_WritePin(GPIOA, SW1_Pin, GPIO_PIN_RESET);
              HAL_GPIO_WritePin(SW0_GPIO_Port, SW0_Pin, GPIO_PIN_SET);
            break;

              case 3:
              HAL_GPIO_WritePin(GPIOA, SW2_Pin, GPIO_PIN_RESET);
              HAL_GPIO_WritePin(GPIOA, SW1_Pin, GPIO_PIN_SET);
              HAL_GPIO_WritePin(SW0_GPIO_Port, SW0_Pin, GPIO_PIN_RESET);
            break;

            case 2:
              HAL_GPIO_WritePin(GPIOA, SW2_Pin, GPIO_PIN_RESET);
              HAL_GPIO_WritePin(GPIOA, SW1_Pin, GPIO_PIN_SET);
              HAL_GPIO_WritePin(SW0_GPIO_Port, SW0_Pin, GPIO_PIN_SET);
            break;

            case 5:
              HAL_GPIO_WritePin(GPIOA, SW2_Pin, GPIO_PIN_SET);
              HAL_GPIO_WritePin(GPIOA, SW1_Pin, GPIO_PIN_RESET);
              HAL_GPIO_WritePin(SW0_GPIO_Port, SW0_Pin, GPIO_PIN_RESET);
            break;

            case 6:
              HAL_GPIO_WritePin(GPIOA, SW2_Pin, GPIO_PIN_SET);
              HAL_GPIO_WritePin(GPIOA, SW1_Pin, GPIO_PIN_RESET);
              HAL_GPIO_WritePin(SW0_GPIO_Port, SW0_Pin, GPIO_PIN_SET);
            break;


            case 7:
              HAL_GPIO_WritePin(GPIOA, SW2_Pin, GPIO_PIN_SET);
              HAL_GPIO_WritePin(GPIOA, SW1_Pin, GPIO_PIN_SET);
              HAL_GPIO_WritePin(SW0_GPIO_Port, SW0_Pin, GPIO_PIN_RESET);
            break;

            case 8:
              HAL_GPIO_WritePin(GPIOA, SW2_Pin, GPIO_PIN_SET);
              HAL_GPIO_WritePin(GPIOA, SW1_Pin, GPIO_PIN_SET);
              HAL_GPIO_WritePin(SW0_GPIO_Port, SW0_Pin, GPIO_PIN_SET);
            break;

            default :
              HAL_GPIO_WritePin(GPIOA, SW2_Pin, GPIO_PIN_RESET);
              HAL_GPIO_WritePin(GPIOA, SW1_Pin, GPIO_PIN_RESET);
              HAL_GPIO_WritePin(SW0_GPIO_Port, SW0_Pin, GPIO_PIN_RESET);
    } // For switch resistor
    sync_sample_blocking(&adc_buffer_A[0], &adc_buffer_B[0], ADC_BUFFER_SIZE);

     for (int k=0; k<256; k++){
      if (adc_buffer_B[k]>3000){
        selection = 0;
      }
      }
    if(switch_resistor == 8){
        selection =0;
    }
  // End of while loop
  }
// End of function
}
```

H Instantiating ADC and different frequencies

```
    void sync_sample_blocking(uint32_t *buffer, uint32_t *bufferB, int buffer_size) {

//Set the correct sampling rate for DAC and ADC and start DMA for DAC1 to create Sinewave
  if (SINE_FREQ == 1000){
    //Set correct sampling rate
    My_TIM2_Init(SINE_1K);
    //Start the DAC via DMA
    HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1, (uint32_t*) sine_wave_array, SINE_1K_LUT_SIZE,
    DAC_ALIGN_12B_R);
  }
  else if (SINE_FREQ == 10000){
    My_TIM2_Init(SINE_10K);
    HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1, (uint32_t*) sine_wave_array, SINE_10K_LUT_SIZE,
    DAC_ALIGN_12B_R);
  }
  else if (SINE_FREQ == 100000){
    My_TIM2_Init(SINE_100K);
    HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1, (uint32_t*) sine_wave_array, SINE_100K_LUT_SIZE,
    DAC_ALIGN_12B_R);
  }

//Setup DAC2
    HAL_DAC_Start(&hdac, DAC_CHANNEL_2);
  HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, 2048);

// Setup ADC
//Must disable the DMA first otherwise, it will not let you change the destination pointer as you
    intend!
  HAL_ADC_Stop_DMA(&hadc1);
  buffer_is_full = 0;
  HAL_ADC_Start_DMA(&hadc1, buffer, buffer_size);

  // For the voltage across range resistor
  HAL_ADC_Stop_DMA(&hadc2);
  buffer_is_full = 0;
  HAL_ADC_Start_DMA(&hadc2, bufferB, buffer_size);


//Setup common sync timer
  HAL_TIM_Base_Stop(&htim2);
  HAL_TIM_Base_Start(&htim2);
  while (buffer_is_full == 0);
  HAL_TIM_Base_Start(&htim2);
  HAL_TIM_Base_Stop(&htim2);
}
```

```c


int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_USART2_UART_Init();
  MX_ADC1_Init();
  MX_ADC2_Init();
  MX_DAC_Init();
  MX_TIM2_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1) {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    // Always generates 1.65V from DAC2 ------------------------------
    HAL_DAC_Start(&hdac, DAC_CHANNEL_2);
    HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, 2048);

    sync_sample_blocking(&adc_buffer_A[0], &adc_buffer_B[0], ADC_BUFFER_SIZE);
   selecting_MUX();

    int magnitudeA, magnitudeB, phaseA, phaseB, magnitudef, phasef;
    Geotzel_Algorithm(adc_buffer_B);
    magnitudeA = magnitude1;
        phaseA = phase1;
    Geotzel_Algorithm(adc_buffer_B);
    magnitudeB = magnitude1;
        phaseB = phase1;

        magnitudef = magnitudeA/magnitudeB;
        phasef = phaseA-phaseB;

      HAL_UART_Transmit(&huart2,"Start\n",6,1);
        for(int i=0;i<ADC_BUFFER_SIZE;i++){
          sprintf( str, "%d",(uint16_t)(adc_buffer_B[i]& 0x00000FFF));
          HAL_UART_Transmit(&huart2,(uint8_t*)str,50,1);
          HAL_UART_Transmit(&huart2,(uint8_t*)"\n",1,1);
        }
      HAL_UART_Transmit(&huart2,(uint8_t*)"End\n",4,1);
```