

Master's Thesis
"Hybrid Electronic ELF Signal
Processing System"
Postgraduate Studies-MET

GEORGE MYSTRIDIS

Physics Department

University of Ioannina



Postgraduate studies in Modern Electronic Technologies
Laboratory of Electronics, Telecommunications and Applications

Physics Department

School of Sciences

University of Ioannina

Master's Thesis

Hybrid Electronic ELF Signal Processing System

Georgios Mystridis

Identification number: 803

Thesis advisor: Vasileios Christofilakis

Ioannina

March 2024

Acknowledgements

With the completion of my master's thesis, I would like to express my gratitude to all those who have contributed to it.

I would like to thank Assistant Professor Vasileios Christofilakis, my master's thesis supervisor, for his continuous support and guidance during its development. I extend my gratitude to the technician Georgios Baldoumas for his invaluable help and to Assistant Professor Ioannis Papadopoulos and Professor Konstantinos Fountas for being part of the three-member committee reviewing this dissertation.

I owe thanks to my friends Spyridon Lontos and Alexandros Sakkas for their interest and help during various challenges of this dissertation.

Last but definitely not least, I want to thank my parents, Eleni Petgazle and Panagiotis Mystridis, for their crucial support over all these years, without which this work would not have been possible.

Contents

| | |
|--|-----------|
| Acknowledgements..... | iii |
| Περίληψη | vi |
| Abstract | vii |
| List of Figures | viii |
| List of Tables..... | x |
| List of Abbreviations | xi |
| Chapter 1 Introduction | 1 |
| 1.1 Ohm's Law | 1 |
| 1.2 Kirchhoff's Laws..... | 2 |
| 1.3 Filters | 3 |
| 1.3.1 Filter specifications | 4 |
| 1.3.2 Classification of filters..... | 4 |
| Some More Key Terms | 5 |
| 1.4 Transfer Function And Transforms | 6 |
| 1.5 Digital Filters..... | 9 |
| 1.5.1 Impulse Response Invariant Design..... | 12 |
| 1.5.2 Step Response Invariant Design | 12 |
| 1.5.3 Bilinear Transform Design | 14 |
| 1.6 Modeling Electronic Circuits..... | 15 |
| 1.6.1 Poles, Zeros and System Response | 17 |
| Transient Response | 18 |
| Chapter 2 Designing the System | 24 |
| 2.1 Designing the Hardware and Firmware..... | 24 |
| 2.1.1 Custom PCB..... | 24 |
| Digital to Analog Converter..... | 24 |
| Preamplifier..... | 27 |
| Notch Filter | 28 |
| COMPLETE PCB | 31 |
| 2.1.2 Arduino Firmware | 34 |
| 2.2 Developing the Software Application..... | 36 |
| 2.2.1 The interface between the Hardware and the Software..... | 36 |
| 2.2.2 Lowpass Butterworth Filter | 40 |

| | |
|---|--------|
| 2.2.3 The software | 43 |
| Coding the Fourier Transform | 47 |
| Communication Throughout the application..... | 50 |
| Chapter 3 Final Measurements and Conclusions | 54 |
| 3.1 Transient Responses and Stability | 54 |
| 3.1.1 Digital Filter..... | 54 |
| 3.1.2 Notch Filter | 56 |
| 3.2 Frequency Responses | 57 |
| 3.2.1 Notch Filter's Response | 57 |
| Using The PicoScope as an analog signal source | 57 |
| Using a Digital Signal Source | 61 |
| 3.2.2 Butterworth Digital Filter's Response | 64 |
| 3.3 Conclusions..... | 69 |
| 3.4 Future Improvements..... | 69 |
| Bibliography | 71 |
| Appendix | - 1 - |
| A.1 PCB SCHEMATIC | - 1 - |
| A.2 Decoupling Capacitors | - 2 - |
| A.3 DERIVATION OF THE TRANSFER FUNCTION OF THE NOTCH FILTER..... | - 7 - |
| A.4 Deriving the difference equation for a 2 nd order Butterworth Lowpass Filter ... | - 12 - |
| A.5 ARDUINO CODE..... | - 13 - |
| A.6 ATmega Firmware..... | - 15 - |
| A.7 ATmega CODE..... | - 28 - |
| A.8 MATLAB CODE | - 32 - |

Περίληψη

Σκοπός αυτής της μεταπτυχιακής εργασίας είναι η δημιουργία ενός υβριδικού συστήματος, που θα μπορεί να δέχεται αναλογικά σήματα στην ELF περιοχή του φάσματος ($< 30\text{Hz}$), να τα ενισχύει, να τα φιλτράρει και να αποθηκεύει τα δεδομένα σε ηλεκτρονικό υπολογιστή. Για την επίτευξη του σκοπού αυτού, σχεδιάστηκε μια εξατομικευμένη πλακέτα (PCB) με τα απαραίτητα ηλεκτρονικά υλικά (ICs, κλπ.), αναπτύχθηκε μία εφαρμογή (standalone software) με την χρήση του MATLAB, καθώς και ένα firmware για Arduino board με σκοπό την σωστή επικοινωνία των παραπάνω.

Ξεκινώντας με το PCB, φτιάχτηκε με δύο πιθανές εισόδους. Η μία προέρχεται απευθείας από μία αναλογική πηγή τάσης, ενώ η δεύτερη αρχικά δέχεται ψηφιακό σήμα, έτσι ώστε ο χρήστης να μπορεί να δουλέψει είτε με ψηφιακό είτε με αναλογικό αρχικό σήμα εισόδου. Στην πρώτη περίπτωση, το σήμα αρχικά περνά από ένα προενισχυτή ($\times 20$) και καταλήγει σε αναλογικό φίλτρο εγκοπής (Notch) που φιλτράρει τα 50Hz του θορύβου που προκαλεί το ρεύμα. Στην συνέχεια, το πλέον φιλτραρισμένο σήμα είναι έτοιμο να σταλθεί σε μια πλακέτα *Arduino Uno R3*, η οποία δρα ως μεσολαβητής για την επικοινωνία PCB-MATLAB/Υπολογιστή. Στην δεύτερη περίπτωση, το σήμα (8-bit ψηφιακό) αρχικά περνά από DAC ενσωματωμένο στο PCB, και μετά όντας πλέον αναλογικό, ακολουθεί την ίδια πορεία με την πρώτη περίπτωση. Το Arduino, στην συνέχεια, παίρνει δείγματα κάθε 2ms από το σήμα εξόδου του PCB, τα μετατρέπει σε κλίμακα 0 – 5 (σύμφωνα με την βιβλιογραφία) και μέσω της θύρας USB τα στέλνει στον υπολογιστή. Εκεί, η εφαρμογή στον υπολογιστή, μετά την εισαγωγή μερικών απαραίτητων παραμέτρων από τον χρήστη, ξεκινάει να λαμβάνει τα δείγματα του σήματος από το Arduino. Αφού τελειώσει η λήψη, ακολουθεί μια διαδικασία μετασχηματισμού Fourier, με σκοπό την «μεταφορά» του ληφθέντος σήματος στο χώρο της συχνότητας. Έπειτα, μπορεί να φιλτραριστεί το σήμα περεταίρω (Low Pass Butterworth Filter) μέσω ενσωματωμένου εργαλείου ψηφιακού φιλτραρίσματος (DSP). Μάλιστα, μπορούν να επιλεγθούν η τάξη του φίλτρου (1 – 10) καθώς και η συχνότητα αποκοπής (1-100Hz καθώς προορίζεται για ανάλυση ELF σημάτων). Τέλος, δίνεται η δυνατότητα αποθήκευσης σε αρχείο τριών διαφορετικών μορφών (.mat, .txt και .xlsx).

Abstract

The goal of this postgraduate dissertation is the development of a hybrid system, able to receive analog signals in the ELF part of the spectrum ($< 30 \text{ Hz}$), amplify and filter them and lastly save the data on a computer. To achieve this, a PCB with all the necessary hardware (ICs, etc.) was designed, and a standalone application, using MATLAB, was developed, as well as Arduino firmware, so the PCB could interface with the computer.

Starting with the PCB - it is made with two different inputs in mind. One is to be connected directly to an analog voltage source and the other one is to receive a digital input so that the user can work either with a digital or an analog input/initial signal. In the first case, the signal initially goes through a preamplifier ($\times 20$) and then through an analog Notch filter, so that the 50 Hz noise of the powerline supply is filtered out. The, now filtered, signal is ready to be transferred to an Arduino board, which acts as the interface, for the communication between the PCB and the computer. In the second case, the input signal (8-bit digital) goes through an onboard DAC, and then, as an analog signal follows the same path as in the first case. The Arduino board takes samples every 2 ms from the PCB output pin, scales it between $0 - 5$, and then through the onboard USB port, sends it to the computer. There, the standalone application, after the user selects the value of some necessary parameters, starts receiving samples of the signal from the Arduino. After it's done receiving data, it automatically begins a process of Fourier transform, so that the signal is mapped to the frequency domain. Following that, one can filter the signal further through a built-in digital filtering tool (Low Pass Butterworth Filter). Additionally, the order of the filter ($1 - 10$) and the cutoff frequency ($1 - 100 \text{ Hz}$ because it is made with ELF signals in mind) can be selected (DSP). Lastly, the option of saving the data is given to the user in three different format files (.mat, .xlsx, .txt).

List of Figures

| | |
|--|----|
| Figure 1.1: Simple circuit depicting the 3 fundamental units in electronics. | 1 |
| Figure 1.2: Kirchhoff's Current Law. | 2 |
| Figure 1.3: Kirchhoff's Voltage Law. | 3 |
| Figure 1.4: A basic depiction of the four major filter types. | 5 |
| Figure 1.5: A Two-Port Network..... | 6 |
| Figure 1.6: Block diagrams of an FIR (a) and an IIR (b) filter. | 10 |
| Figure 1.7: Visualization of group delay. | 11 |
| Figure 1.8: Impulse Response Invariant Design steps..... | 12 |
| Figure 1.9: Step Response Invariant Design steps. | 13 |
| Figure 1.10: Step Response example. | 13 |
| Figure 1.11: Block Diagram representation of a system. | 15 |
| Figure 1.12: Block diagram of a transfer function. | 16 |
| Figure 1.13: Voltage-current, voltage-charge, current-voltage, and impedance relationships for capacitors, inductors and resistors. | 16 |
| Figure 1.14: The left (stable) and right (unstable) half planes. | 18 |
| Figure 1.15: Plot of a system's Transient Response. | 19 |
| Figure 1.16: Effect of zeros on the transient response with a step function as input. | 19 |
| Figure 1.17: Example with Rise, Peak and Settling time. | 20 |
| Figure 1.18: The 4 forms of a transient response. | 21 |
| Figure 1.19: Example of a system G_s with a unit step input ($1s$). (a) Block diagram of the system, showing its input and output; (b) pole-zero plot of the system; (c) evolution of the system's step response. | 22 |
| Figure 2.1: Simplified Block Diagram of the System. | 24 |
| Figure 2.2: Block Diagram of the Hardware part of the system. | 24 |
| Figure 2.3: DAC circuit and Pre-Amplifier Schematics. | 25 |
| Figure 2.4: Current-to-Voltage converter circuit..... | 25 |
| Figure 2.5: Pin connections (left) and internal block diagram (right) of the DAC0808..... | 26 |
| Figure 2.6: Pin connections for the operational Amplifier TS922 (for SO8 and TSSOP8 formats). | 27 |
| Figure 2.7: The op amp configuration used for the preamplifier; the non-inverting configuration..... | 27 |
| Figure 2.8: Bootstrapped Twin-T Notch Filter. | 28 |
| Figure 2.9: The notch filter schematic. | 29 |
| Figure 2.10: Effect of Quality factor on the notch. | 30 |
| Figure 2.11: Impedances between subcircuits. | 31 |
| Figure 2.12: Photo of the Board during testing of the DAC. | 31 |
| Figure 2.13: Front side of the PCB on KiCad 3D Viewer. | 32 |
| Figure 2.14: Back side of the PCB on KiCad 3D Viewer. | 32 |
| Figure 2.15: PCB Schematic of the front of the board. | 33 |
| Figure 2.16: PCB Schematic of the back of the board..... | 33 |
| Figure 2.17: Arduino Uno R3 SMD version. | 34 |
| Figure 2.18: Flow chart of the Arduino Firmware..... | 35 |

| | |
|--|----|
| Figure 2.19: MATLAB communication with Windows for port availability. | 36 |
| Figure 2.20: Window for the user to select the port. | 37 |
| Figure 2.21: Prompt for the user to input some key for the sampling variable values. | 38 |
| Figure 2.22: Setting the port and defining the terminator to read and write. | 38 |
| Figure 2.23: Flow chart of the interfacing process on the MATLAB side. | 39 |
| Figure 2.24: Code snippet handling the difference equation. | 42 |
| Figure 2.25: Filter Specifications Window. | 42 |
| Figure 2.26: Main window of the MATLAB app. | 43 |
| Figure 2.27: Progress bar. | 44 |
| Figure 2.28: Data and Key Variables window. | 44 |
| Figure 2.29: One of the error messages displayed, when the app is not used properly. | 45 |
| Figure 2.30: Graphs with plotted data. | 46 |
| Figure 2.31: View of a signal in the time and frequency domain. | 49 |
| Figure 2.32: The function that performs Fast Fourier Transform in the app. | 49 |
| Figure 2.33: The entirety of the source code for the application in a collapsed form. | 51 |
| Figure 2.34: Defining a class as a Handle class in MATLAB. | 52 |
| Figure 2.35: Flow chart of the custom MATLAB application. | 53 |
| Figure 3.1: Comparison between the Theoretical and the Filter's Step Response. Note: The notable shift at the beginning is because the System data used for this graph, take into account the delay filtering causes. | 55 |
| Figure 3.2: Comparison of Transient responses with a sine as an input. | 55 |
| Figure 3.3: Theoretical Step Response of the Notch Filter. | 56 |
| Figure 3.4: Theoretical Transient response of the Notch Filter with a sine as input. | 56 |
| Figure 3.5: The PicoScope 2204A Oscilloscope by © 2024 Pico Technology Ltd. | 57 |
| Figure 3.6: Frequency Response of the Notch filter. | 58 |
| Figure 3.7: Comparison of the Theoretical and actual filter's frequency responses using an analog signal source. | 59 |
| Figure 3.8: Op amp saturation and signal clipping. | 61 |
| Figure 3.9: Voltage Divider Values. | 62 |
| Figure 3.10: Notch filter's frequency response when using the DAC – a digital signal source as input. | 63 |
| Figure 3.11: Comparison of theoretical and actual data, when the input is from a digital signal source. | 63 |
| Figure 3.12: Sine sampled at Nyquist Frequency. Note: The amplitude of the input signal is 1, while the sampled is around 0,04. That is because the samples align at the zeros of the sine. The reason because they are not exactly zero is the accuracy of the calculations. | 64 |
| Figure 3.13: A zoom in of the above graph. | 65 |
| Figure 3.14: Sine sampled close to the Nyquist frequency. | 65 |
| Figure 3.15: Magnitude Response Comparison. | 66 |

List of Tables

| | |
|--|--------|
| Table 1: The four Fourier approaches. | 47 |
| Table 2: Custom PCB data while measuring input directly from a Voltage Source..... | 60 |
| Table 3: Measurement Data from the Notch while using ATmega as a signal source. | 62 |
| Table 4: Data taken while testing the digital filter..... | 67 |
| Table 5: Reset and Interrupt Vectors..... | - 20 - |

List of Abbreviations

| Abbreviation | Definition |
|---------------------|------------------------------------|
| ELF | Extremely Low Frequency |
| AC | Alternating Current |
| AF | Audio Frequency |
| BW | Bandwidth |
| DAC | Digital to Analog Converter |
| DC | Direct Current |
| FIR | Finite Impulse Response |
| IC | Integrated Circuit |
| IIR | Infinite Impulse Response |
| IIT | Impulse Invariant Transformation |
| KCL | Kirchhoff's Current Law |
| KVL | Kirchhoff's Voltage Law |
| LTl | Linear Time Invariant |
| USB | Universal Serial Bus |
| PCB | Printed Circuit Board |
| RF | Radio Frequency |
| TF | Transfer Function |
| MSB | Most Significant Bit |
| LSB | Less Significant Bit |
| SI | International System of Units |
| RHP | Right-Half plane |
| LHP | Left-Half plane |
| FFT | Fast Fourier Transform |
| DFT | Discrete Fourier Transform |
| DTFS | Discrete Time Fourier Series |
| DTFT | Discrete Time Fourier Transform |
| CTFS | Continuous Time Fourier Series |
| CTFT | Continuous Time Fourier Transform |
| VRM | Voltage Regulator Module |
| IDE | Integrated Development Environment |
| ISR | Interrupt Service Routine |

Chapter 1 Introduction

This thesis describes the development of a software-hardware/analog-digital hybrid system capable of amplifying and filtering signals within the ELF band of radio frequencies.

It opens with a brief introduction to fundamental laws in electronics, as well as to analog and digital low pass and notch filtering, and more.

1.1 Ohm's Law

Ohm's Law, unveiled by Georg Simon Ohm and presented in his 1827 paper titled "The Galvanic Circuit Investigated Mathematically," constitutes the first and arguably the most significant relationship among current, voltage, and resistance.

An electric circuit is created when a conductive path is established, enabling the uninterrupted flow of electric charge. This continuous movement of electric charge along the conductors in a circuit is known as current and it is commonly described in analogical terms, similar to the flow of liquid through a hollow pipe.

The force motivating these charge carriers to move is called Voltage. In more scientific terms, Voltage is a measure of potential energy between two points. When discussing the presence of Voltage in a circuit, we are addressing the measurement of the potential energy available to transport charge carriers between specific points in the circuit. The term "Voltage" has no meaning without specifying two distinct points of reference.

Current does not move freely through a conductor. It's met with opposition to that movement. This opposition to the motion is more properly called resistance. The amount of current is dependent on the voltage present and the amount of resistance in the circuit. Similar to voltage, resistance is relative between two points. Consequently, descriptions of voltage and resistance frequently specify being "between" or "across" two points within a circuit.

Ohm's fundamental discovery revealed that the electric current flowing through a metal conductor in a circuit is directly proportional to the Voltage applied across it for any given temperature.

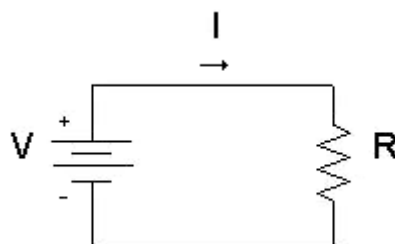


Figure 1.1: Simple circuit depicting the 3 fundamental units in electronics.

Kirchhoff's Laws

Ohm expressed his discovery in the form of a simple equation, effectively describing how current, voltage, and resistance interrelate [1, p. 4]:

$$I = \frac{V}{R} \quad 1.1$$

Where [1, p. 2]:

- I is the symbol of Current - measured in Amperes (S.I.)
- V symbolizes the Voltage across the conductor - measured in Volts (S.I.)
- R symbolizes the Resistance/opposition of the conductor to the flow of current – measured in Ohm (S.I.)

1.2 Kirchhoff's Laws

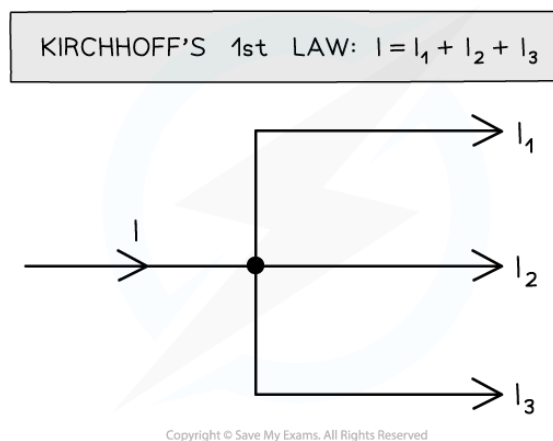
Born on 12 March 1824 in Prussia, a state within the German Empire, Gustav Robert Kirchhoff was a German physicist renowned for his contributions to electrical circuits, black body radiation, and spectroscopy.

In 1845, while still a student, Kirchhoff formulated his circuit laws, which are now ubiquitous in electrical engineering, as they are the fundamental/basic laws used in circuit analysis to solve complex problems. Originally conceived as a seminar exercise, his study later evolved into his doctoral dissertation.

Kirchhoff's first circuit law – also known as Kirchhoff's Current Law (KCL) states that the sum of all currents flowing into a node equals the sum of currents flowing out of the node. It can be written as [1, p. 2]:

$$\sum i_{IN} = \sum i_{OUT} \quad 1.2$$

Figure 1.2: Kirchhoff's Current Law.



Note: Kirchhoff's First Law (10.1.4) | CIE A Level Physics Revision Notes 2022. (n.d.). Save My Exams.
<https://www.savemyexams.com/a-level/physics/cie/22/revision-notes/10-d-c-circuits/10-1-dc-practical-circuits--kirchhoffs-laws/10-1-4-kirchhoffs-first-law/>

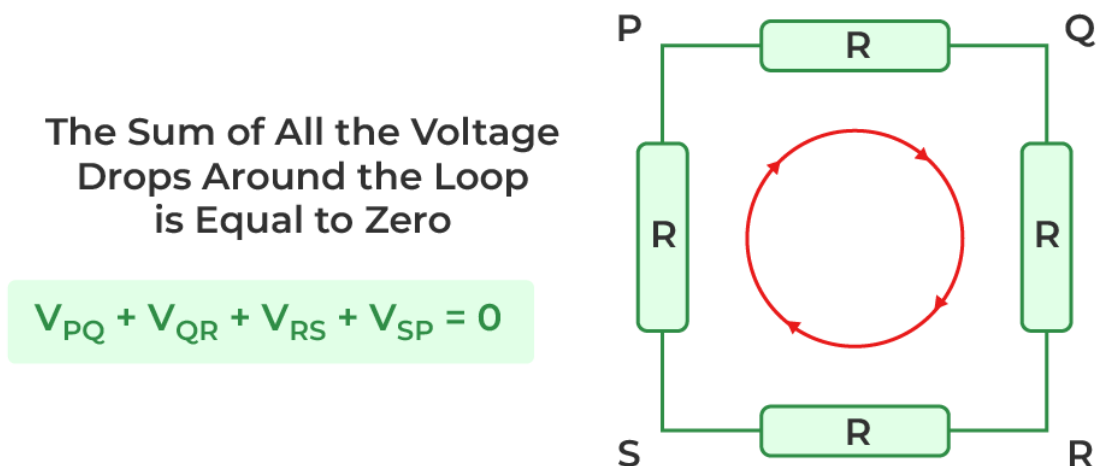
Filters

According to his second circuit law or KVL, the sum of voltages around a loop is zero. Mathematically, this is written as [1, p. 2]:

$$\sum_n V_n = 0 \quad 1.3$$

Where n is the number of element Voltages around the loop.

Figure 1.3: Kirchhoff's Voltage Law.



Note: GeeksforGeeks. (2023, June 15). Kirchhoff's Laws. <https://www.geeksforgeeks.org/kirchhoffs-laws/>

1.3 Filters

Filters, in general, are frequency-selective circuits capable of passing/amplifying or attenuating a signal depending on its frequency. Thus, using filters, one can minimize the effect of the noise (irrelevant or else undesirable frequencies) in a signal. Analog filters are a basic building block of signal processing and more generally, in electronics.

There are many practical applications for filters. Some of them being:

- In Radio communications, filters enable radio receivers to only "see" the desired signal while rejecting all other signals (assuming the other signals have different frequency content).
- In DC power supplies, filters are used to eliminate undesired high frequencies that are present on AC input lines. Additionally, filters are used on a power supply's output to reduce ripples.
- In audio electronics, there are the so-called crossover networks. They are networks of filters used to channel low-frequency audio to woofers, mid-range frequencies to midrange speakers, and high-frequency sounds to tweeters.

1.3.1 Filter specifications

Assuming the filter is a 2-port network (1 input – 1 output), then we can define the following terms:

- Passband
- Stopband
- Passband & stopband ripples

Passband is defined as the range of frequencies, that are passed seemingly “untouched”, or in scientific terms, with the minimum attenuation through the filter. While stopband is the band of frequencies that are heavily attenuated or blocked by the filter. Passband and stopband ripples are, as the name suggests, fluctuations (measured in dB) occurring in the passband or stopband of a filter’s frequency magnitude response curve.

1.3.2 Classification of filters

There are mainly two ways of classifying a filter; depending on the components used in making the circuit and depending on its frequency magnitude response.

Using the former, there are two types of filters:

- Passive filters
- Active filters

Passive filters are built using – as the name suggests – passive components such as resistors, capacitors and inductors. On the other hand, an active filter is made using active elements (transistors, op-amps) in addition to resistors and capacitors.

Using the latter, they can be classified as [2, Ch. 1.1]:

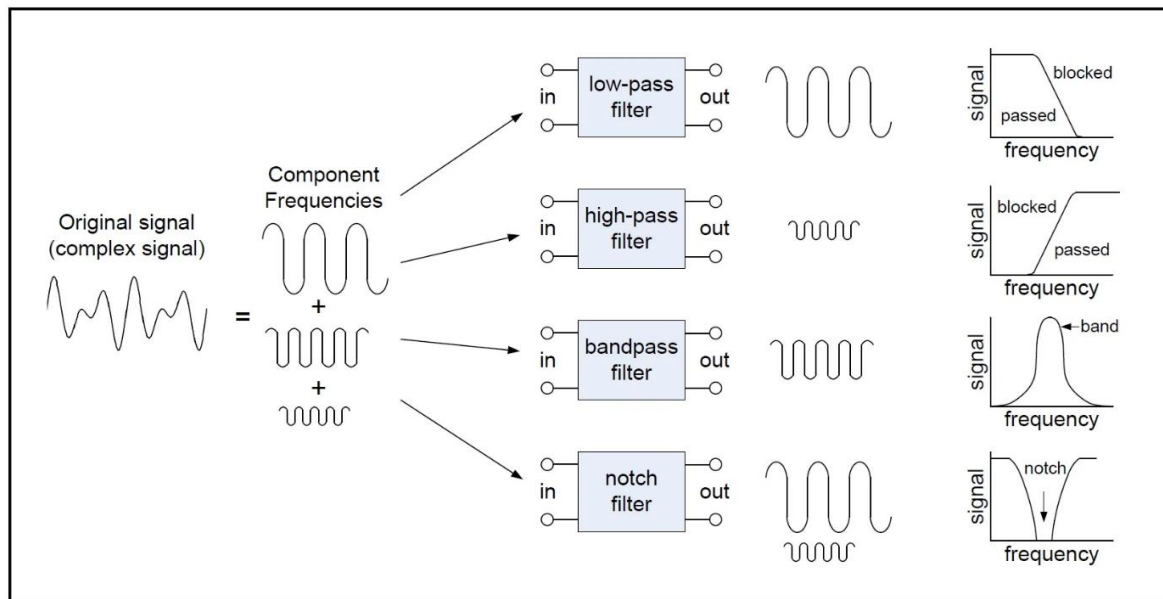
1. Low Pass Filters: Only signals with low frequencies – between 0Hz and f_c ¹ - are allowed to pass through the filter, while higher frequencies are blocked/attenuated.
2. High Pass Filters: Opposite of the low-pass, it only allows high-frequency signals while frequencies below f_c are blocked/attenuated.
3. Band Stop Filters: Blocks/Attenuates signals whose frequencies fall within a certain band set up between two points, while allowing the rest to pass through².
4. Band Pass Filters: Allows signals whose frequency falls within a certain range (between two points) to pass while blocking the rest of the frequencies on either side of said range.

Finally, it is also worth noting that depending on the operating frequency range, filters may be categorized as Audio Frequency (AF) or Radio Frequency (RF) filters.

¹ f_c : Cutoff Frequency. It’s explained later on.

² NOTE: A notch filter is a band-stop filter with a narrow band-stop bandwidth. Notch filters are used to attenuate a narrow range of frequencies, and it’s one of the two filters used in this project.

Figure 1.4: A basic depiction of the four major filter types.



Note: An Introduction to Filters. (2023, September 30). All About Circuits. <https://www.allaboutcircuits.com/technical-articles/an-introduction-to-filters/>

Some More Key Terms

- Response curves: they are graphs used to describe how a filter behaves. On the y-axis, we usually have attenuation or gain, while frequency goes on the x-axis. Attenuation/Gain, or Magnitude of the TF, is simply the ratio $\frac{V_{OUT}}{V_{IN}}$, often expressed in decibels using the following formula [3, p. 321]:

$$A(dB) = 20 \log_{10} \cdot \frac{V_{OUT}}{V_{IN}} \tag{1.4}$$

- Cutoff or $-3dB$ Frequency (f_c or f_{3dB}): Cutoff or “minus 3dB” frequency, is the frequency value that corresponds to a 3dB drop of the output signal’s amplitude relative to that of the input signal. That 3dB value means that the output power is reduced by one-half or that:

$$\frac{V_{OUT}}{V_{IN}} = \frac{1}{\sqrt{2}} \Rightarrow V_{OUT} = \frac{V_{IN}}{\sqrt{2}} \tag{1.5}$$

It’s worth mentioning that there is only one f_c for low-pass and high-pass filters, but for band-pass and band-stop filters there are two. They are normally referred to as f_1 and f_2 .

- Stopband Frequency (f_s): It’s the frequency value at which the attenuation reaches a specific value.
 - ❖ For band-pass and band-stop filters, two Stopband frequencies exist. The frequency range between them is referred to as the stopband.
 - ❖ For low-pass and high-pass filters, frequencies beyond or before -respectively- the f_s are referred to as the stopband.

Transfer Function And Transforms

- Center Frequency (f_0): is a central frequency that lies between the upper and lower cutoff frequencies (for band-pass and band-stop filters). It can be calculated by the geometric mean [4, p. 8.9-8.10] of F_H and F_L ³: $f_0 = \sqrt{F_H F_L}$
- Bandwidth (BW): is defined as the width of the passband. Mathematically, it's the difference between the frequencies where the response is -3dB from the maximum value [4, p. 8.9]: $BW = F_H - F_L$
- Quality factor (Q): It basically measures how close to perfect (ideal) a filter⁴ can be. Basically, the higher its value, the better the filter; the lower the losses. A general definition that applies to any system (and from which all other definitions are derived) is:

$$Q = 2\pi \frac{\text{energy stored}}{\text{energy dissipated per cycle}} \quad 1.6$$

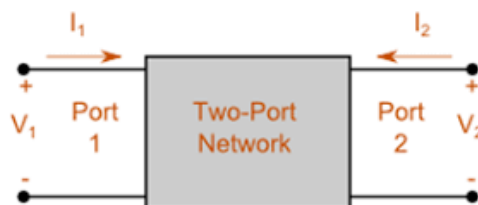
For band-pass and notch filters, the value of Q is given by the following equation [4, p. 8.9]:

$$Q = \frac{f_0}{BW} \quad 1.7$$

1.4 Transfer Function And Transforms

The filters used in this system - both analog and digital – are linear circuits⁵, meaning that they can be represented by a 2-port network as shown below:

Figure 1.5: A Two-Port Network.



Note: Characterization of Linear Time-Invariant Two-Port Networks | BengalStudents. (n.d.).
<https://www.bengalstudents.com/books/electrical-circuit-theory-and/characterization-linear-time>

This means that they can be described using what is called a **Transfer Function**. A Filter's Transfer Function- in the s domain- can be⁶ defined as the ratio of the output Voltage of the filter to its input voltage [5, p. 45]:

³ See next bullet point.

⁴ Or how close to perfect a component can be.

⁵ Explained in Chapter 1.6

⁶ In reality, the transfer function can be defined as a ratio between two chosen units of a system. For example, between the input current and the output voltage of a circuit.

$$H(s) = \frac{V_o(s)}{V_i(s)} \quad 1.8$$

In the time domain, however, it is different. It's defined as the convolution of the input:

$$v_{OUT}(t) = v_{IN}(t) * h(t) \quad 1.9$$

As noticed, the Transfer Function is expressed in terms of the variable "s". That variable comes from the Laplace Transform, and it represents complex frequency (angular frequency ω^7). The Laplace Transform is a mathematical technique that changes a function of time into a function of the (angular) frequency domain or else the s-domain. In electronics, it's used to more easily analyze a complex circuit. It's defined by⁸ [5, p. 35]:

$$\mathcal{L}\{f(t)\} = \int_{0^-}^{\infty} f(t)e^{-st} dt, \quad s = \sigma + j\omega^9 \quad 1.10$$

But if the Laplace Transform takes us from the time domain to the frequency domain, why don't we just use the (continuous time) Fourier Transform (definition below [6, p. 288])?

$$\mathcal{F}\{f(t)\} = \int_{-\infty}^{\infty} f(t)e^{-j\omega t} dt \quad 1.11$$

The answer is that the Laplace Transform is, in a way, a generalized Fourier Transform, or in a more mathematical term, it can be considered as a super-set for the Continuous Time Fourier Transform. That can be seen when in the Laplace Transform, $\sigma = 0$, we get the Fourier Transform but only for $t > 0$. But there are some differences; the Fourier Transform is exclusively used in bilateral form (see that the integral limits are from $t \rightarrow -\infty$ to $t \rightarrow \infty$), whereas the Laplace is not (it's more often than not used in unilateral form). That's why it is very well suited to problems involving initial conditions ($t = 0$), and it's used to calculate the transient response of a system.

The Fourier Transform is used in the Telecommunications Field, where almost all signals of interest are sinusoidal and complex exponential (which are the kind for which the Fourier Transform is well suited), and the steady-state response is the sought solution.

So, we can say, that the Transfer Function is a mathematical description of the filter's frequency (s) domain behavior. If a filter's Transfer Function is given, we can compute the specific magnitude and phase information of the circuit, by simply substituting $s = j\omega^{10}$ and evaluating the expression at the angular frequency's (ω) value of interest.

Given, that it is a complex function, it can be expressed as:

⁷ The conversion between these is simple using the well-known formula: $\omega \left(\frac{rad}{sec}\right) = 2 * \pi * f(Hz)$

⁸ This is the definition of the unilateral Laplace transform.

⁹ The notation for the lower limit means that even if $f(t)$ is discontinuous at $t = 0$, we can start the integration prior to the discontinuity, as long as the integral converges.

¹⁰ In the electronics field – for AC analysis, it's assumed $\sigma = 0$. The reason is that we are looking at the response of the system to periodic (and thus non-decaying) sinusoidal signals, whereby Laplace conveniently reduces to Fourier along the imaginary axis. The real axis in the Laplace domain represents exponential decay/growth factors that pure signals do not have, and which Fourier does not model.

$$T(j\omega) = |T(j\omega)|e^{j\phi(\omega)} \quad 1.12$$

where:

$$\phi(\omega) = \text{arg}(T(j\omega)) \quad 1.13$$

Usually, the magnitude of $T(j\omega)$ can be expressed in decibels, in terms of the Gain Function which is the unit used for Bode plots [2, p. 3]:

$$G(\omega) = 20\log_{10}(|T(j\omega)|) \quad 1.14$$

Alternatively, in terms of the Attenuation function [2, p. 3]:

$$A(\omega) = -20\log_{10}(|T(j\omega)|) \quad 1.15$$

In this use case, the transfer function is a ratio of voltages. That is why the conversion to decibels is 20 times the logarithm of the ratio. Had it been a ratio of powers, it would be converted to decibels by taking 10 times the logarithm of the ratio [3, p. 322].

When we want to see the output expressed in terms of time, all we need to do is take the Inverse Laplace of $V_{IN}(s) * T(s)$.

$$V_{OUT}(t) = \mathcal{L}^{-1}V_{IN}(s)T(s) \quad 1.16$$

But what happens when we need to work on the digital domain? How will we digitize the Transfer function? For linear systems, the Laplace transform is used to transform time domain characteristics to the frequency domain. For the discrete time (digital) systems, the Z-Transform will be used [6, p. 742]:

$$Z\{x(n)\} = X(z) = \sum_{n=-\infty}^{\infty} x(n) z^{-n} \quad 1.17$$

1.5 Digital Filters

Linear electronic analog filters are those that can be described with linear differential equations – because of the equations that govern the components like inductors and capacitors (derivatives with respect to time). Using those equations, and Kirchoff's and Ohm's Laws, we can derive the transfer function of a system fairly easily. This will later be referred to as "Analog Filter Theory".

Now, besides Analog filters, we can also create Digital ones that serve the same function. There is a variety of methods that can be used to design said type of filters; a common one being using the so-called "analog filter approximation functions". These are already developed functions, that have the same properties¹¹ as their analog filter's Transfer Functions counterpart – with Butterworth and Chebyshev being the most common. The reason they are used in Digital Filter Design is that they are more easily¹² translated for use in discrete-time systems.

Most of the filters designed that way, are recursive in nature, meaning that the output of the filter will depend on previous values of itself – as well as past and current values of the input¹³. These types of filters can theoretically have impulse responses that continue forever, and therefore, are commonly referred to as **Infinite Impulse Response filters (I.I.R.)**.

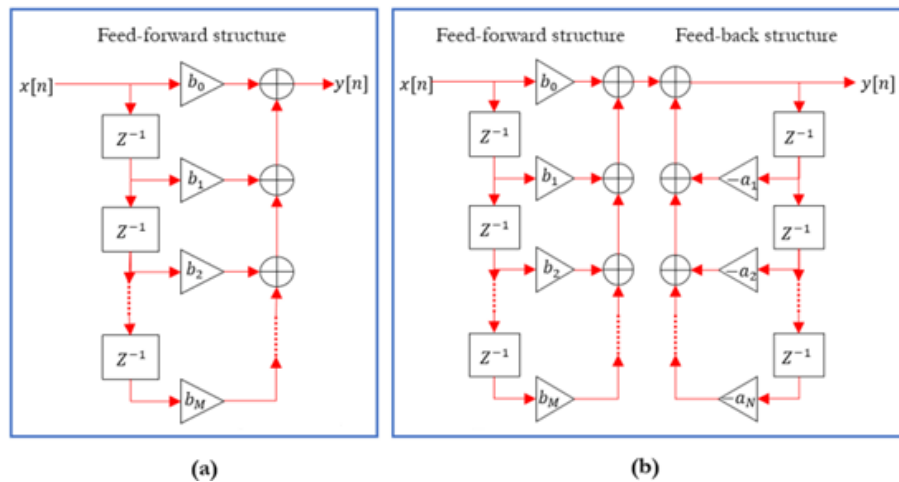
Another method of designing digital filters, which does not depend on analog filter theory, but rather uses the frequency response of the desired filter, to directly determine the digital filter coefficients. These types of filters generally have an impulse response containing only a finite number of values and thus are commonly called **Finite Impulse Response filters (F.I.R.)**.

¹¹ Or -more specifically - as close as possible, because there is no perfect equivalent to analog filters at all frequencies.

¹² Compared to Transfer functions derived from pure circuit analysis

¹³ The equation that describes this is called "difference equation" and it's the discrete time counterpart of a differential equation.

Figure 1.6: Block diagrams of an FIR (a) and an IIR (b) filter.



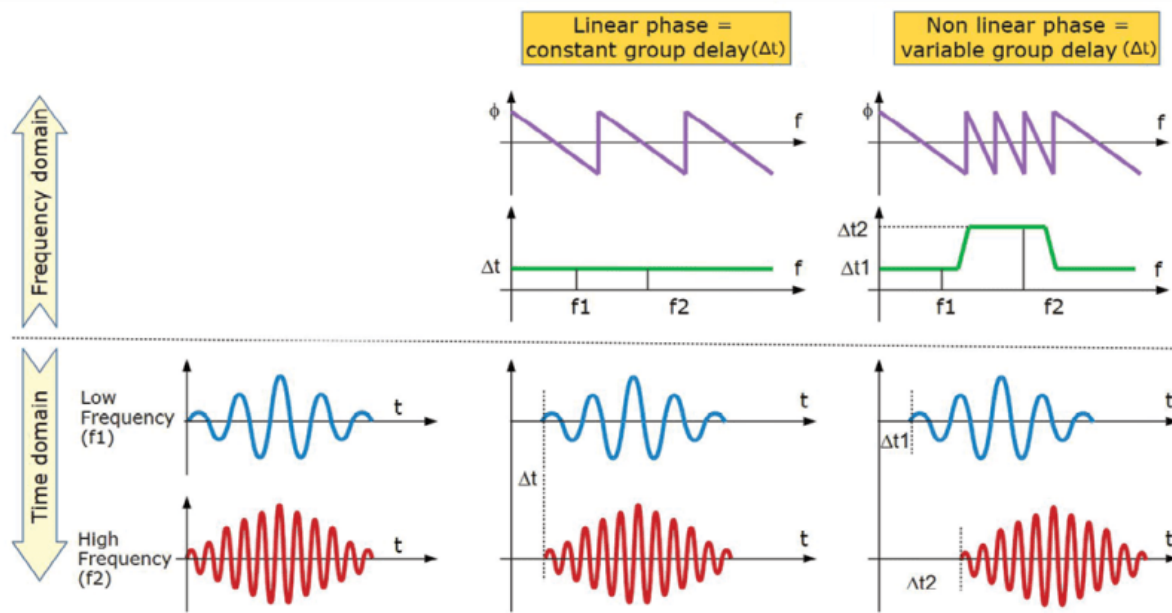
Note: Mathuranathan. (2021, May 15). Choosing FIR or IIR ? Understand design perspective - GaussianWaves. GaussianWaves. <https://www.gaussianwaves.com/2017/02/choosing-a-filter-fir-or-iir-understanding-the-design-perspective/>

The initial decision in designing a system containing a digital filter is determining whether to use an IIR or FIR filter. First, and foremost, the appropriate filter type must be selected based on the application's requirements.

IIR filters have the advantages of providing higher selectivity for a particular order and a closed-form design technique that doesn't require iteration. The design technique also provides for a rather precise solution to the specifications of gain and edge frequencies. However, IIR filters also have the disadvantages of nonlinear phase characteristics and possible instability due to poor implementation. FIR (non-recursive) filters, on the other hand, can provide a linear phase response (constant group delay¹⁴) that is important for data transmission and high-quality audio systems. Also, they are always stable because they are implemented using an all-zero transfer function. Since no poles can fall outside the unit circle, the filter will always be stable. But because of this, the order of the filter is much higher than the IIR filter, which has a comparable magnitude response. This higher order leads to longer processing times and larger memory requirements. In addition, FIR filters must be designed using an iterative method since the required filter length to satisfy a given filter specification can only be estimated [2, pp. 187–188].

¹⁴ In the field of signal processing, group delay and phase delay are two interconnected ways of describing how a signal's frequency components experience time delays while traversing through a LTI system (such as a microphone or a filter – analog or digital). While phase delay specifically characterizes the time shift of a sinusoidal component (a sine wave in steady state), group delay describes the time shift of the envelope of a wave "packet". This packet essentially is a cluster/group of oscillations centered around a single frequency, travelling collectively.

Figure 1.7: Visualization of group delay.



Note: Lacoste, R. (2021, January 29). Group Delay Basics - More Filter Fun - Circuit Cellar. Circuit Cellar. <https://circuitcellar.com/research-design-hub/group-delay-basics-more-filter-fun/>

Taking all this into account, the filter designer needs to consider the requirements imposed on the digital filter. If the magnitude response is crucial with less emphasis on phase response, opting for an IIR filter would be preferable. On the other hand, if phase response holds greater significance than magnitude response, choosing an FIR filter is warranted. In cases where both magnitude and phase response are equally important, one must also consider processing time constraints and memory requirements. If other options prove ineffective, it is possible to design both an FIR and IIR filter (with some phase correction) to meet the specifications. Subsequently, both filters can be tested to assess the results.

After settling on the filter type, several additional decisions must be made. For instance, will the system operate in real-time, or can it be a non-real-time system? A real-time system involves providing input samples to the digital filter and processing them to generate an output sample - all before the next input sample arrives. This imposes a very precise time constraint on the available processing time, with higher sampling frequencies providing less time for processing. On the other hand, some systems are afforded the luxury of operating in non-real-time. For example, signals can be recorded and processed at a later time¹⁵. In this case, extensive processing is possible due to the lack of a fixed time interval that marks the end of the processing time¹⁶.

As mentioned previously, when designing an IIR filter, one must select the right “analog filter approximation function”. The second step is finding a method of translating their analog

¹⁵ This is the way the digital filter designed for this system operates.

¹⁶ Given that for this project the IIR approach was chosen, more emphasis should be placed on how to proceed after deciding to design such a filter.

filter's characteristics¹⁷ into those of a digital one – or in other words, finding the corresponding discrete Transfer function for the digital filter. The three most common and widely known methods are the following:

- Impulse Response invariant design
- Step response invariant design
- Bilinear transform design

1.5.1 Impulse Response Invariant Design

Impulse Response Invariant Design, otherwise called impulse invariant transformation (IIT), is based on the idea of creating a digital filter with an impulse response¹⁸ that is a sampled version of the impulse response $h(t)$ of the analog filter [2, p. 142].

Let's start by taking the inverse Laplace transform of the analog filter's TF $H(s)$ and thus determining its continuous impulse response $h(t)$. Next, to determine the system's discrete-time impulse response $h(nT)$, all that is needed is to sample the $h(t)$, by effectively replacing t with nT . Lastly, the discrete TF $H(z)$ can be calculated by simply taking the Z-Transform of $h(nT)$ [2, Ch. 6.1].

Figure 1.8: Impulse Response Invariant Design steps.



1.5.2 Step Response Invariant Design

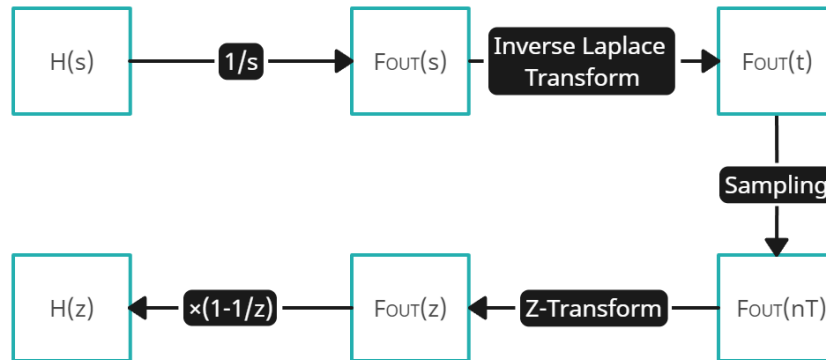
An alternative approach for digitizing an analog transfer function is to match the step response of both systems (analog and digital) [2, p. 146].

The step response of a system, given a specific initial state, consists of the time evolution of its outputs, when subjected to Heaviside step functions as inputs. In electronic engineering and control theory, the step response characterizes the behavior, with respect to time, of a system's output as its input transitions from zero to one in a brief interval. The process of the Step Response Invariant Design is quite similar to that of IIT, with the distinction that, in this case, instead of the impulse response $h(t)$, it's the step response of the analog filter that will be sampled and then transformed using the Z-transform [2, Ch. 6.2].

¹⁷ It's worth reminding that the analog filter approximation transfer function still describes/approximates an analog filter.

¹⁸ In the field of signal processing and control theory, the impulse response of a system is its output when it's subjected to a short input signal $\delta(t)$. Broadly speaking, the impulse response denotes how any dynamic system reacts to external alteration.

Figure 1.9: Step Response Invariant Design steps.



Finding the step response of a system is simple, given that we already have its TF $H(s)$:

$$\frac{F_{OUT}(s)}{F_{IN}(s)} = H(s) \Rightarrow F_{OUT}(s) = F_{IN}(s)H(s) \Rightarrow F_{OUT}(s) = \frac{1}{s}H(s) \quad 1.18$$

where $\frac{1}{s}$ is the Laplace Transform of the step/Heaviside function $u(t) = \begin{cases} 1, & t > 0 \\ 0, & t < 0 \end{cases}$ and $F_{OUT}(s)$ is the step response of the analog filter.

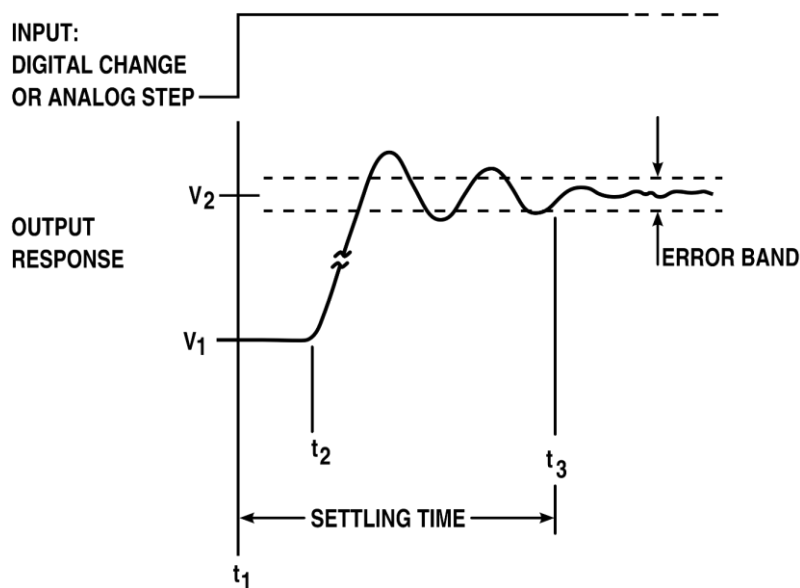
Now, to find the time-domain response $f_{OUT}(t)$ to the step input, all that is needed is an inverse Laplace transform:

$$f_{OUT}(t) = \mathcal{L}^{-1}\{F_{OUT}(s)\} \quad 1.19$$

Then, simply sample it to get the discrete-time version $f_{OUT}(nT)$. Next step is an Z-Transform [2, p. 147]:

$$F_{OUT}(z) = Zf_{OUT}(nT) = H(z) \cdot \frac{1}{1-z^{-1}} \Rightarrow \boxed{H(z) = F_{OUT}(z) \cdot (1-z^{-1})} \quad 1.20$$

Figure 1.10: Step Response example.



Note: Step response. (2023, October 27). In Wikipedia. https://en.wikipedia.org/wiki/Step_response

1.5.3 Bilinear Transform Design

Both the impulse invariant and step invariant design methods provide good approximations for lowpass and some bandpass analog filter responses. However, they cannot provide good matching of high-frequency responses, which makes it impossible to use them for highpass or bandstop filter design [2, p. 151]. That is why they fail to be the best approaches for matching analog filter responses when a precise match is needed throughout a wide range of frequencies. Additionally, distortion from aliasing can occur, if a careful selection of the sampling frequency and strict band-limiting¹⁹ don't take place.

Bilinear Transform Design attempts to tackle these problems – by aiming to make an adequate match over the entire filter frequency range. Certainly, this poses a challenge as the analog frequency range spans from zero to infinity, while the digital frequency only extends from zero to 2π ²⁰. Nevertheless, a transformation from the analog s -domain to the digital z -domain has been developed.

In this approach, the relationship between the complex variables s and z can be described by the following equation, with T representing the sampling period [2, p. 151]:

$$s = \frac{2}{T} \cdot \frac{z - 1}{z + 1} \quad 1.21$$

So, once the analog TF $H(s)$ has been determined, the bilinear transform substitution (1.21) can be used to simply derive the digital TF $H(z)$.

¹⁹ A bandlimited signal, strictly speaking, is a signal with zero energy beyond a specific frequency range. Practically, a signal is considered bandlimited if its energy outside a designated frequency range is sufficiently low to be deemed negligible for a specific application.

²⁰ The digital frequency range of 0 to 2π represents the normalized angular frequency, expressed in units of radians per sample: $\omega_{norm} = \frac{\omega_{analog}}{F_{sampling}}$

1.6 Modeling Electronic Circuits

Kirchhoff's and Newton's laws lead to mathematical models that describe the relationship between the input and output of dynamic systems. One such model is the linear, time-invariant²¹ differential equation [5, p. 16]:

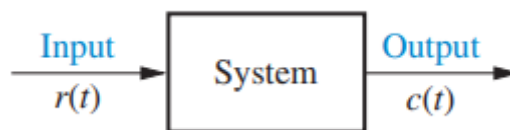
$$\frac{d^n c(t)}{dt^n} + d_{n-1} \frac{d^{n-1} c(t)}{dt^{n-1}} + \dots + d_0 c(t) = b_m \frac{d^m r(t)}{dt^m} + b_{m-1} \frac{d^{m-1} r(t)}{dt^{m-1}} + \dots + b_0 r(t) \quad 1.22$$

which relates the output, $c(t)$, to the input, $r(t)$, by way of the system parameters, a_i and b_j .

So, a system's input-output relationship can be described by a differential equation, where the formulation and coefficients of the equation serve as a description of the system. However, it is not a fully satisfying representation from a systems perspective. This is obvious, when looking at 1.22 – a general, n th-order, linear, time-invariant differential equation, we see that the system parameters, which are the coefficients, as well as the output and the input appear throughout the equation.

A mathematical representation, such as that shown in Figure 1.11, in which the input, output and the system are distinct and separate parts, would be preferred.

Figure 1.11: Block Diagram representation of a system.



Note: [5, p. 34]

Representing a system characterized by a differential equation in the form of a block diagram poses a challenge, and the Laplace Transform (1.10) serves as the mathematical tool for achieving this.

As mentioned earlier, the notation for the lower limit means that even if $f(t)$ is discontinuous at $t = 0$, we can start the integration prior to the discontinuity as long as the integral converges. This property has distinct advantages when applying the Laplace transform to the solution of differential equations, where the initial conditions are discontinuous at $t = 0$. For example, using differential equations, we have to solve for the initial conditions after the discontinuity knowing the initial conditions before said discontinuity. Whereas, using the Laplace Transform, we need only know the initial conditions before the discontinuity [5, p. 35].

Typically, a system that a LTI differential equation can represent, can be modeled as a transfer function. Starting with the general n th-order, linear, time-invariant differential equation that

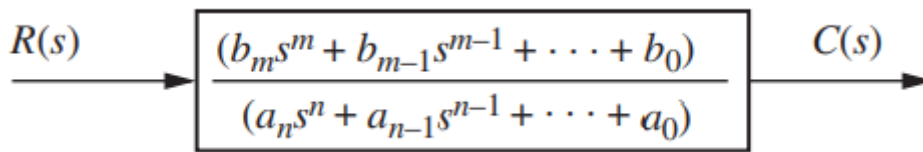
²¹ A circuit can be defined as linear when the relationship between its input and output is linear. Additionally, time invariance means that whether we apply an input to the system now or T seconds from now, the output will be identical except for a time delay of T seconds.

is 1.22, and after taking the Laplace Transform of both sides - assuming that all initial conditions are zero- we get:

$$\frac{C(s)}{R(s)} = H(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_0}{\alpha_n s^n + \alpha_{n-1} s^{n-1} + \dots + \alpha_0} \quad 1.23$$

Now, the input $R(s)$, the output $C(s)$ and the system – the ratio of polynomials in s – are separated. $H(s)$ is the transfer function (as defined in previous chapters) and is evaluated with zero initial conditions.




Figure 1.12: Block diagram of a transfer function.



Note: [5, p. 45]

Of course, the term “system” also includes electrical circuits/networks, meaning that they can also be modeled as a TF. In fact, resistors, inductors, and capacitors - the three basic building blocks of electronic (passive) components - can be individually modeled into the s -domain. Figure 1.13 summarizes the voltage-current-charge and impedance relationships for resistors, inductors, and capacitors, under zero initial conditions.

Figure 1.13: Voltage-current, voltage-charge, current-voltage, and impedance relationships for capacitors, inductors and resistors.

| Component | Voltage-current | Current-voltage | Voltage-charge | Impedance $Z(s) = V(s)/I(s)$ | Admittance $Y(s) = I(s)/V(s)$ |
|--|---|---|----------------------------------|---------------------------------|----------------------------------|
|  Capacitor | $v(t) = \frac{1}{C} \int_0^t i(\tau) d\tau$ | $i(t) = C \frac{dv(t)}{dt}$ | $v(t) = \frac{1}{C} q(t)$ | $\frac{1}{Cs}$ | Cs |
|  Resistor | $v(t) = Ri(t)$ | $i(t) = \frac{1}{R} v(t)$ | $v(t) = R \frac{dq(t)}{dt}$ | R | $\frac{1}{R} = G$ |
|  Inductor | $v(t) = L \frac{di(t)}{dt}$ | $i(t) = \frac{1}{L} \int_0^t v(\tau) d\tau$ | $v(t) = L \frac{d^2 q(t)}{dt^2}$ | Ls | $\frac{1}{Ls}$ |

Note: Nise, N. S. (2014). Control Systems Engineering. Wiley. p 47

Many electrical networks consist of multiple loops and nodes and that is why finding their TF can pose a challenge. But the process can be broken down into steps [5, p. 51]:

1. Using Figure 1.13, replace passive element values with their impedances.
2. Replace all sources and time variables with their Laplace transform.
3. Assume a transform current and a current direction in each mesh/loop.
4. Write Kirchhoff’s voltage law around each mesh/loop.
5. Solve the simultaneous equations for the output.
6. Using said equations, form the TF.

1.6.1 Poles, Zeros and System Response

Discussion of a system's time response is moot if the system does not have stability. But in order to explain what stability means in this context, we need to start from the fact that the output response of a system is the sum of two responses: the forced response and the natural response. The first one is also called steady-state response and it's defined as the behavior of the system after a long time period has passed since an external excitation and steady operating conditions have been reached. Natural response describes the way the system dissipates or acquires energy²². The form or nature of this response is dependent only on the system, not the input [5, p. 10]²³.

For a system to be useful, it's crucial for the natural response to eventually either diminish to zero, thus leaving only the forced response to remain present, or to oscillate. However, in certain systems, the natural response doesn't attenuate or oscillate but instead grows indefinitely. Over time, the escalating natural response surpasses the forced response significantly, resulting in a complete loss of control. This condition, called instability, poses a risk of potential self-destruction for the physical device unless design measures, such as limit stops, are implemented. Therefore, systems must be designed to be stable – that is, their natural response must decay to zero or oscillate as time approaches infinity.

Using these concepts, the following definitions of stability, instability, and marginal stability can be made [5, Ch. 6.1]:

- A linear, time-invariant system is stable if the natural response approaches zero as time approaches infinity.
- A linear, time-invariant system is unstable if the natural response grows without bound as time approaches infinity.
- A linear, time-invariant system is marginally stable if the natural response neither decays nor grows but remains constant or oscillates as time approaches infinity.

The above definitions rely on a description of the natural response. But because when one is looking at the total response, it can be difficult to differentiate between the natural and the forced response, an alternate definition of stability can be made²⁴:

- A system is stable if every bounded input yields a bounded output.
- A system is unstable if any bounded input yields an unbounded output.

While various techniques, such as solving differential equations or performing inverse Laplace transforms, will lead to the evaluation of the output response, they are time-intensive and laborious. Productivity is enhanced by analysis and design techniques that yield results swiftly. The use of poles and zeros and their relationship to the time response of a system is such a technique.

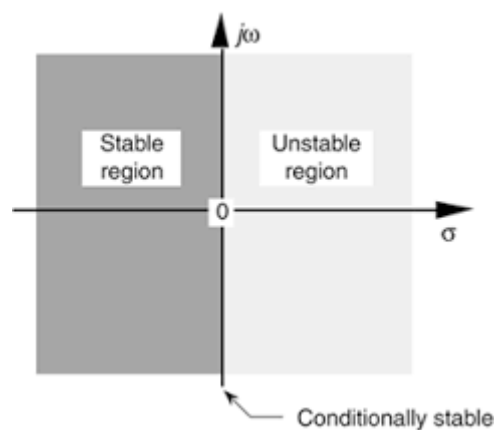
²² Simply put, the natural response is the system's response to initial conditions with all external forces set to zero, while the forced response is the system's response to an external stimulus with zero initial conditions.

²³ For reference, in linear differential equations these responses were referred to as the homogenous (natural response) and the particular (forced response) solutions.

²⁴ This is referred to as the Bounded-Input, bounded-output (BIBO) definitions of stability.

The poles of a transfer function are the values of the Laplace transform variable, s , that cause the transfer function to become infinite or any roots of the denominator of the transfer function that are common to roots of the numerator. Now, the zeros of a transfer function are the values of the Laplace transform variable, s , that cause the transfer function to become zero, or any roots of the numerator of the transfer function that are common to roots of the denominator [5, p. 159].

Figure 1.14: The left (stable) and right (unstable) half planes.



Note: THE LAPLACE TRANSFORM | Chapter Six. Infinite Impulse Response Filters. (n.d.).
https://flylib.com/books/en/2.729.1/the_laplace_transform.html

But how do poles and zeros relate to the stability of a system? By simply taking the inverse Laplace transform of the transfer function of a system²⁵ one can prove that poles in the left half-plane (lhp) yield either pure exponential decay or damped sinusoidal natural responses that decay to zero as time approaches infinity. On the other hand, poles in the right half-plane (rhp) yield either pure exponential or exponentially increasing sinusoidal natural responses. Finally, a system with poles on the imaginary axis ($\Re(s) = 0$) yields pure sinusoidal oscillations as a natural response²⁶.

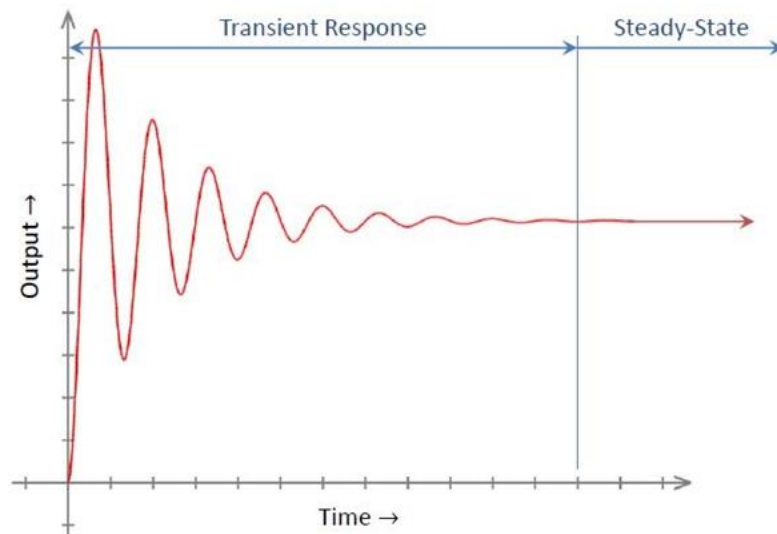
Transient Response

Zeros, on the other hand, don't directly affect the stability of an LTI system, and that's apparent when looking at the definitions of stability. What they do affect is the time - and consequently the transient - response of the system, which is defined as the system's initial response to a sudden change in its inputs. It refers to the behaviour of the system during the time period immediately after a change in its input and the arrival to the stable state.

²⁵ It should be clear that the inverse Laplace transform of the TF of a system yields its natural response, while the inverse Laplace transform of the input $R_{IN}(s)$ yields the forced response. A simple addition gives us the full response of a system – that is the inverse Laplace of the output $C_{OUT}(s)$. Visualized in Figure 1.19.

²⁶ If the pole is equal to zero, we still get a marginally stable system, but natural response is a constant value – not a sinusoid.

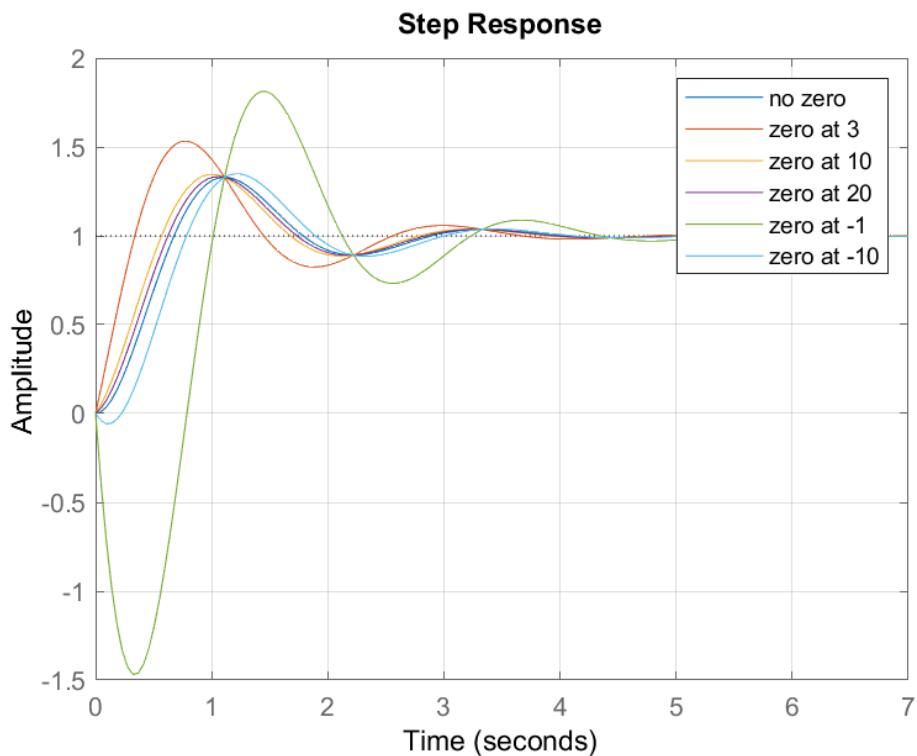
Figure 1.15: Plot of a system's Transient Response.



Note: Admin. (2014, April 9). ▷ Transient Response analysis of control systems. Electrical Equipment. <https://engineering.electrical-equipment.org/panel-building/transient-response-analysis-of-control-systems.html>

Adding a zero to a system leads to a new TF $H(s) \rightarrow (s + a) \cdot H(s)$, which consists of two parts: the derivative²⁷ of the original response ($s \cdot H(s)$) and a scaled version of the original response ($a \cdot H(s)$). How the value of the zero affects the response is depicted on Figure 1.16.

Figure 1.16: Effect of zeros on the transient response with a step function as input.

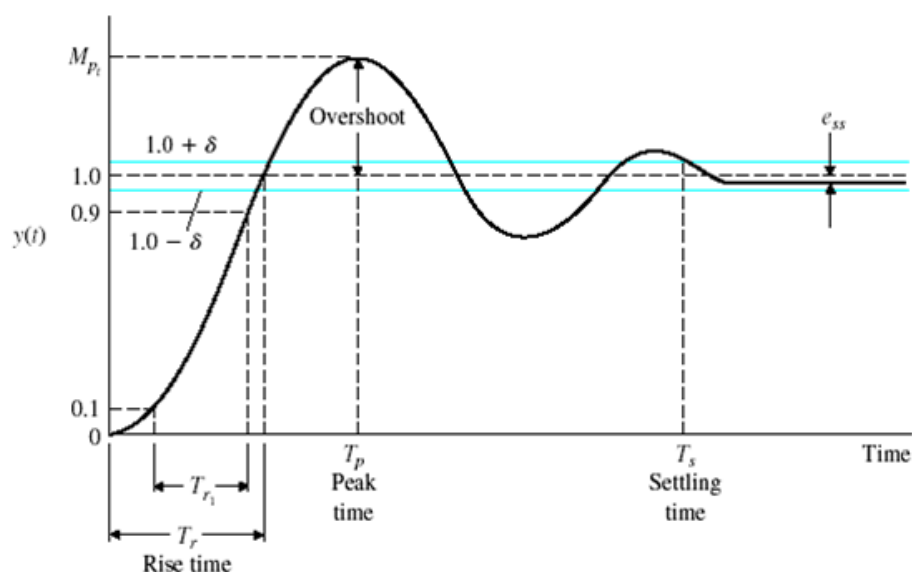


²⁷ Derivative in respect to time. According to easily available Laplace Transform tables: $\mathcal{L} \left[\frac{df}{dt} \right] = sF(s) - f(0^-)$ where $f(0^-) = 0$ because of the zero initial conditions.

The transient response can be further analyzed, depending on the order of the filter. Intuitively, the 1st order²⁸ systems are the simplest to analyze. Changing the parameters (such as the coefficients of the TF) only changes the speed of the response. Said speed – or more generally – performance can be described using these three terms [5, Ch. 4.3]:

- Time constant: It's the time it takes for the output to get to ~63% of the final value (steady state).
- Rise time (T_r): the time it takes for the output to go from 10% to 90% of the final value.
- Settling time (T_s): the time it takes for the output to stay within 2% of the final value.

Figure 1.17: Example with Rise, Peak and Settling time.



Note: Quality of the transient response for an arbitrary transfer function. (n.d.). Engineering Stack Exchange. <https://engineering.stackexchange.com/questions/45598/quality-of-the-transient-response-for-an-arbitrary-transfer-function>

While these terms remain useful when analyzing the transients of higher-order filters, it's not the most important aspect of the analysis. The reason why is that - now - changing the system's parameters can lead not only to a change in the speed of the response but also to a change in the form of the response [5, Ch. 4.4]. Those forms are shown in Figure 1.18:

1. Undamped Response:
 - a. Poles: Both²⁹ imaginary at $\pm j\omega_1$
 - b. Natural Response: Undamped sinusoid with radian frequency equal to ω_1
2. Underdamped Response:
 - a. Poles: Two complex at $-\sigma_d \pm j\omega_d$

²⁸ The order of a system can be defined using only the TF. More specifically, whichever of the order of the numerator and denominator is larger, that is the order of the system.

²⁹ There are only two poles, because it's defined for a 2nd order system – both for simplicity and because a 2nd order system will be used for this project.

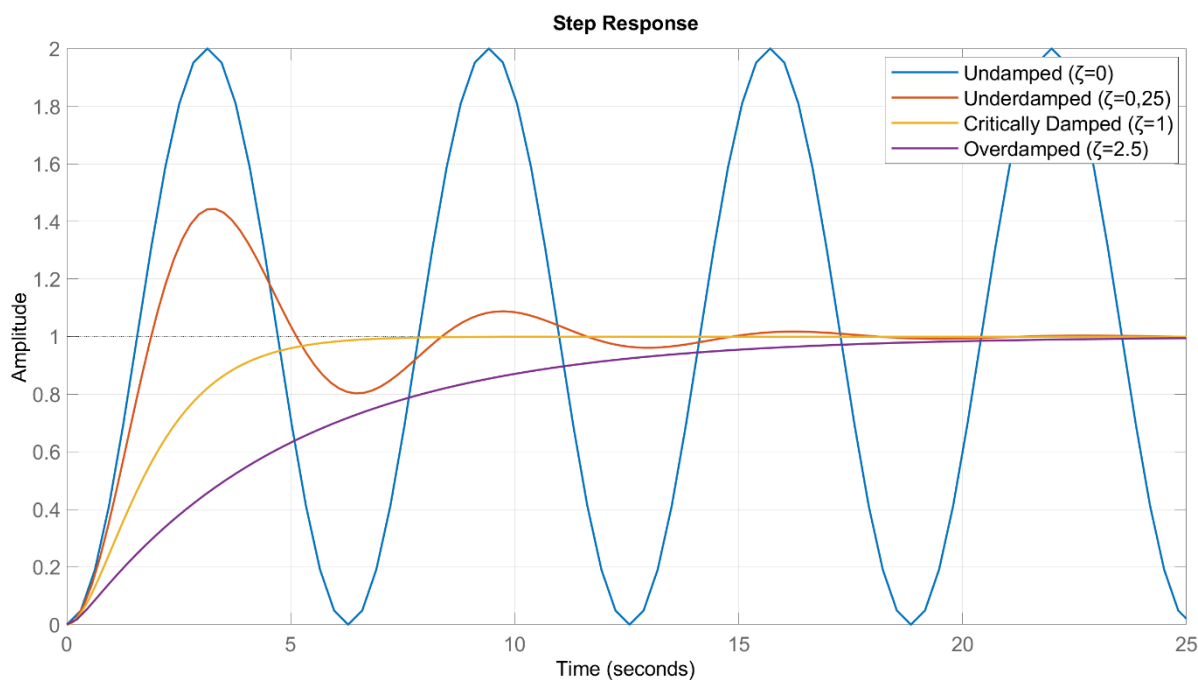
- b. Natural Response: Damped sinusoid described by: $c(t) = Ae^{(-\sigma_d)}\cos(\omega_d t - \phi)$
- 3. Critically Damped Response:
 - a. Poles: two real at $-\sigma_1$
 - b. Natural Response: Described by: $c(t) = K_1 e^{(-t \sigma_d)} + K_2 t e^{(-t \sigma_d)}$
- 4. Overdamped Response:
 - a. Poles: two real at σ_1, σ_2
 - b. Natural Response: Two exponentials with time constants equal to the reciprocal of the pole locations: $c(t) = K_1 e^{(-\sigma_1)} + K_2 e^{(-\sigma_2)}$

In Figure 1.18 we can also see a dimensionless variable ζ . This is called the damping ratio of a system and it's defined as [5, p. 170]:

$$\zeta = \frac{\text{Exponential Decay Frequency} \left(\frac{\text{rad}}{\text{sec}}\right)}{\text{Natural Frequency} \left(\frac{\text{rad}}{\text{sec}}\right)} \tag{1.24}$$

$$= \frac{1}{2\pi} \cdot \frac{\text{Natural Period}(\text{sec})}{\text{Exponential Time Constant}}$$

Figure 1.18: The 4 forms of a transient response.



Using ζ and the natural frequency we can derive a general 2nd order TF to quickly find the transient response of the system we are working on by simply comparing its TF to the general one. Starting with a general system:

$$G(s) = \frac{b}{s^2 + a \cdot s + b} \tag{1.25}$$

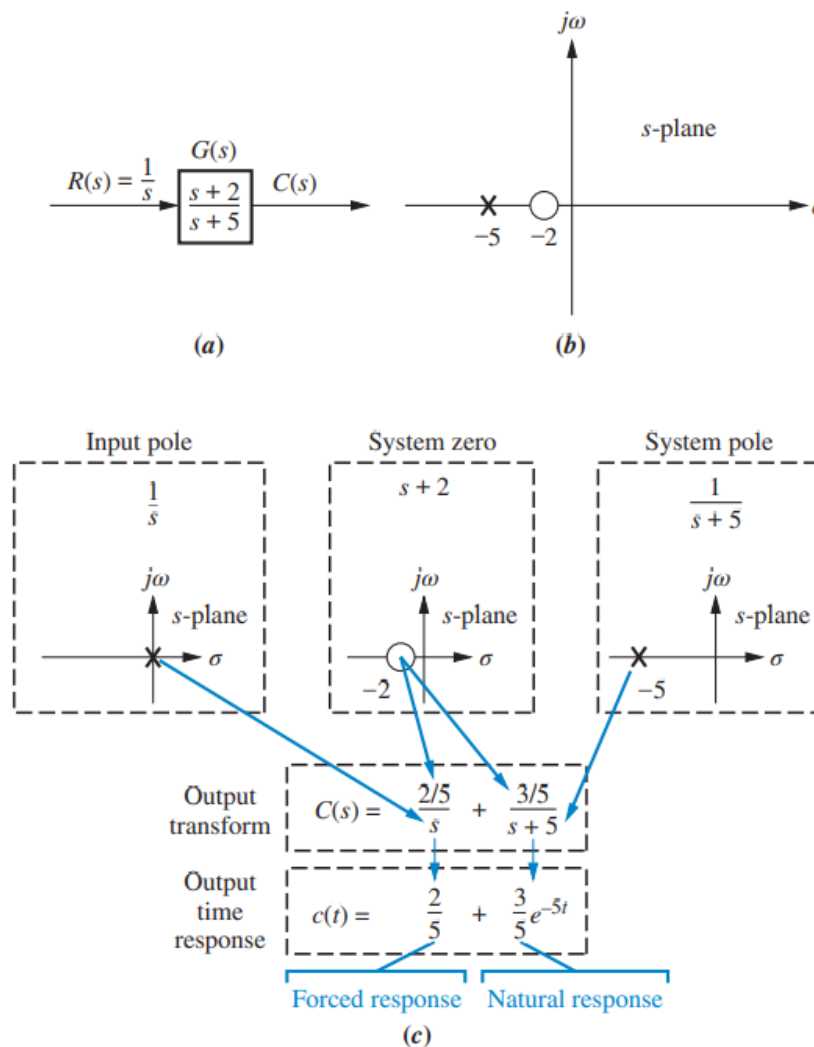
The natural frequency of a 2nd order system is defined as the frequency of oscillation of the system without damping [5, p. 169]. So, for the $G(s)$ to be undamped, it needs to have both its poles to be imaginary. Therefore $a = 0$ and $S_p = \pm j\sqrt{b} \rightarrow \omega_n = \sqrt{b}$.

Now, according to the definition 1.24 and assuming an underdamped system³⁰, the magnitude of the exponential decay is equal to $|\Re(S_p)| = \frac{a}{2}$. That means: $a = 2\zeta\omega_n$.

The general Transfer function looks like this:

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{1.26}$$

Figure 1.19: Example of a system $G(s)$ with a unit step input ($\frac{1}{s}$). (a) Block diagram of the system, showing its input and output; (b) pole-zero plot of the system; (c) evolution of the system's step response.



Note: Nise, N. S. (2014). Control Systems Engineering. Wiley. p.160

³⁰ This assumption is because an underdamped system's poles are complex – they have both a real and an imaginary part.

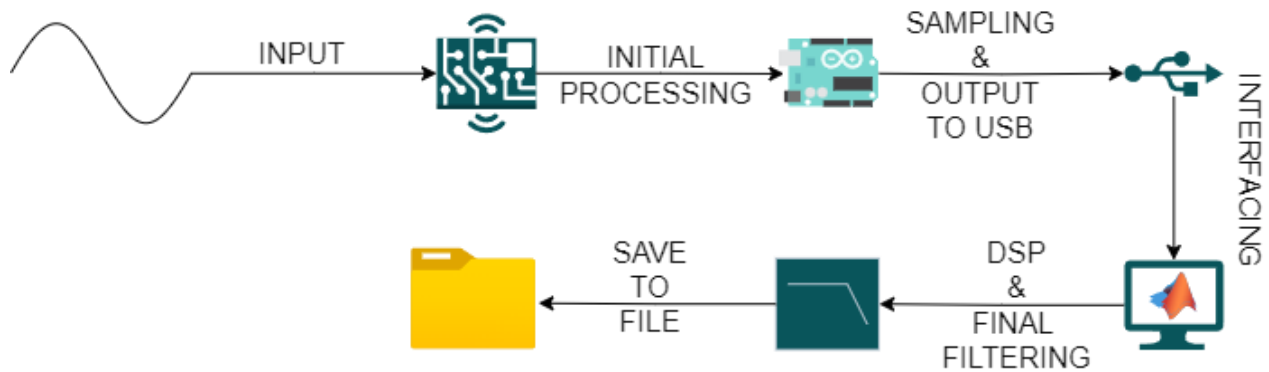
Now, using 1.26 and what was previously said about the 4 types of transient responses and their relationship to their poles, it can be concluded:

- If $\zeta = 0$ the system is Undamped ($S_p = \pm j\omega_n$)
- If $0 < \zeta < 1$ the system is Underdamped ($S_p = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1}$, where $\zeta^2 - 1 < 0$)
- If $\zeta = 1$ the system is Critically Damped ($S_p = -\zeta\omega_n$)
- If $\zeta > 1$ the system is Overdamped ($S_p = -\zeta\omega_n \pm \omega_n\sqrt{\zeta^2 - 1}$, where $\zeta^2 - 1 > 0$)

Chapter 2 Designing the System

As mentioned in the **Error! Reference source not found.**, the goal is the development of a hybrid system capable of receiving analog or digital signals and eventually - through software - filtering out the signals that are outside of the ELF band as shown in the following block diagram.

Figure 2.1: Simplified Block Diagram of the System.

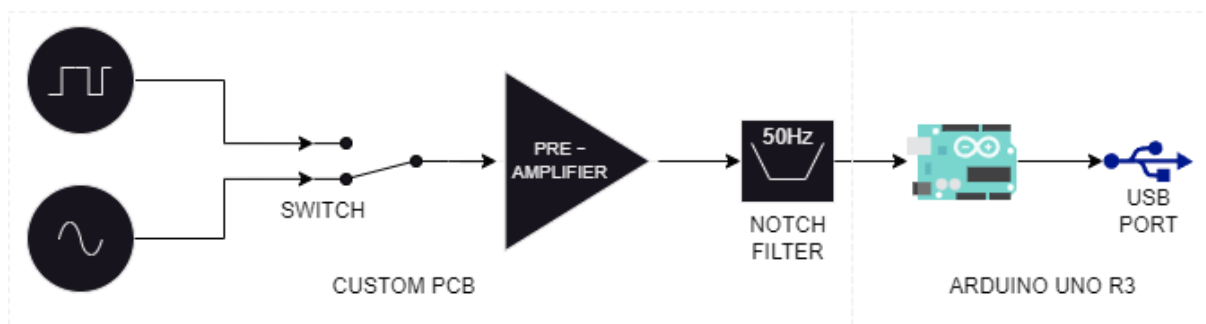


2.1 Designing the Hardware and Firmware

The hardware part of this system, after receiving a signal, is responsible for amplifying it, filtering out the power supply noise (50Hz), and finally sending it to a computer for the final processing (DSP).

As shown in the block diagram (Figure 2.2), the PCB will be handling the pre-amplification and the filtering, while the interfacing with the computer will be done through an Arduino board, more specifically Arduino Uno R3.

Figure 2.2: Block Diagram of the Hardware part of the system.



2.1.1 Custom PCB

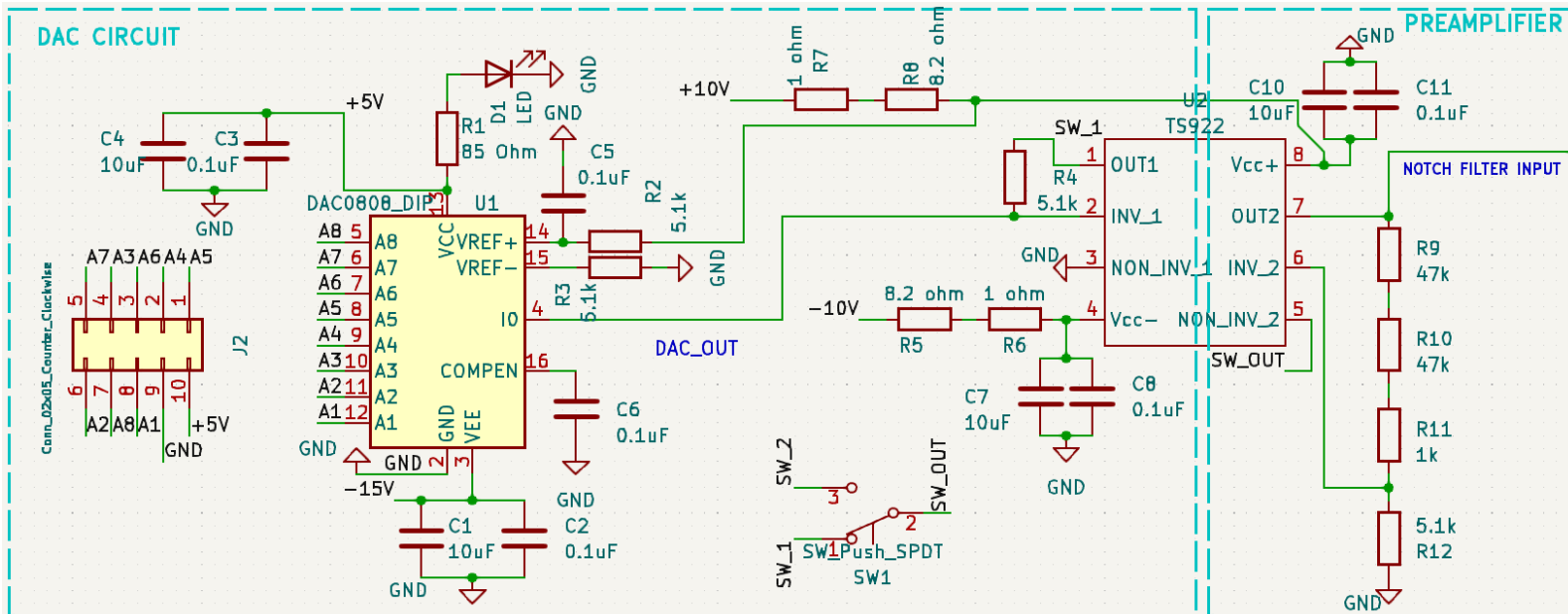
In this section, the specifications of the PCB along with its components will be examined.

Digital to Analog Converter

The block diagram in Figure 2.2 is omitting a specific part of the PCB design. If the digital input of the board is to be used, the signal will have to be converted to analog,

before it goes through the pre-amplifier and the analog notch filter. To do that, a specific IC called DAC (Digital to Analog Converter) is needed, and as expected, just the IC is not enough, but rather a whole subcircuit must be designed.

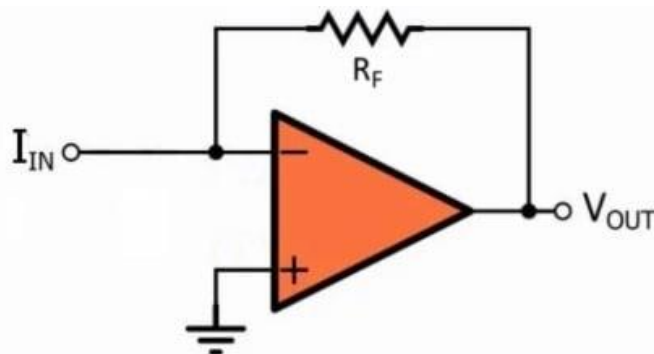
Figure 2.3: DAC circuit and Pre-Amplifier Schematics.



Note: TS922 is the Amplifier IC used. It's 'cut' in half in the picture because it is comprised of two identical operational amplifiers - one used for the DAC circuit and one as an Amplifier.

The specific name of the DAC is “DAC0808” and it’s an 8-bit one, meaning that its digital input is comprised of 8 parallel bits – 1 pin for each bit. As shown in Figure 2.5 the input pins 5-12 are the digital-in pins, with A1 being the MSB and A8 the LSB. 5VDC powers DAC0808 through the V_{CC} pin. V_{REF+} and V_{REF-} pins are necessary reference voltages for transistor logic. The V_{EE} has similar use, as it’s mainly directly connected to the emitter of NPN and PNP transistor within the IC according to its schematics. It can also be regarded as a negative power supply pin, as it is usually connected to negative volts. The COMP pin is used for voltage stability and lastly, I_o is the output pin, from which the converted signal comes out.

Figure 2.4: Current-to-Voltage converter circuit.



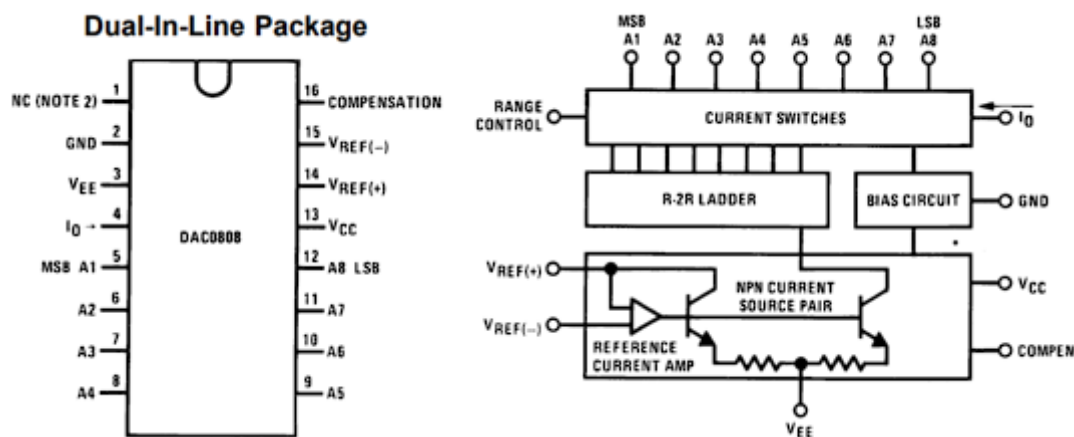
Note: Keim, R. (2020, September 27). Transimpedance Amplifier: Op-Amp-Based Current-to-Voltage signal converter. Video Tutorial. <https://www.allaboutcircuits.com/video-tutorials/op-amp-applications-current-to-voltage-converter/>

The output of DAC0808, as seen in Figure 2.3³¹, isn't connected directly to the next stage of the PCB, but first, it is going through an operational amplifier³². The reason for that is because the I_0 pin is outputting current – not voltage [7]. The DAC is designed like this because a smooth continuous output current is more desirable, as it does not contain the DAC noise that is created during its operation. Many systems use the DAC output current directly, but in this case, a voltage output is needed. The op-amp, in this specific configuration (more clearly depicted in Figure 2.4), functions as a Current-to-Voltage converter.

Because on (ideal) op amps, the inverting and non-inverting inputs have very high input impedance, no current can go through. So, it must go through R_F , and for that to happen, V_{OUT} needs to be at a lower voltage in comparison to the voltage of the non-inverting input. Due to the operational principles of op-amps since the inverting input is directly connected to the ground ($V = 0$), the voltage at the non-inverting input will also be zero³³. This is the reason the input is connected to the non-inverting input, so it would be possible for V_{OUT} to be at lower potential, and for the current to flow as intended.

While it is not its current use case, an amplification of the input still takes place. The current-to-voltage amplifier can be described as having a gain, but since the output and input signals have different units, and therefore cannot be directly compared, is more difficult to know its gain. According to the datasheet [7], the voltage on pin 4 (the DAC's output), restricted to a range of -0.55 to $0.4V$ when $V_{EE} = -5V$. However, the output voltage compliance is extended to $-5V$ when $V_{EE} < -10V$. This created a problem, since after the DAC subcircuit, the signal is routed through a unskippable $\times 20$ amplifier. More details and how this issue was resolved are provided on chapter 3.2.1.

Figure 2.5: Pin connections (left) and internal block diagram (right) of the DAC0808.



³¹ In the schematic, one can see pairs of capacitors connected to power input pins. These are called “decoupling capacitors” and will be analyzed in the appendix: A.2.

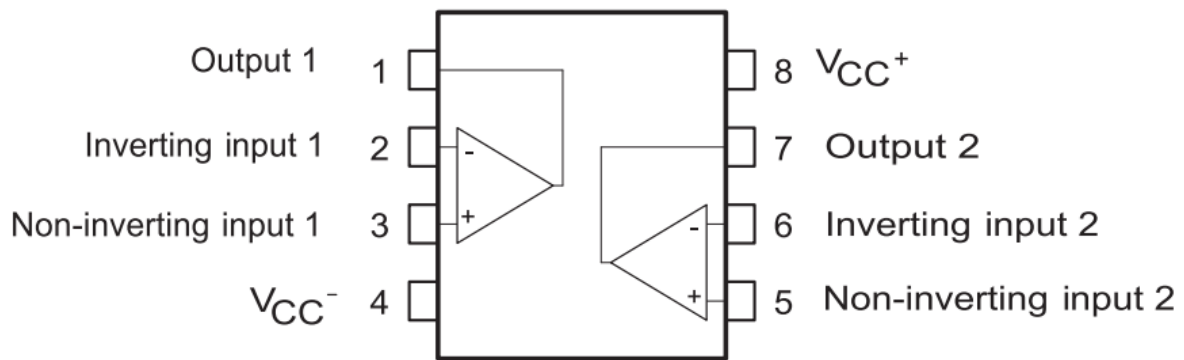
³² The DAC subcircuit shown in the schematic Figure 2.3, is the one proposed by the manufacturer [7, p. 4] for a typical DAC application.

³³ This is called virtual grounding.

Preamplifier

In the schematic Figure 2.3 is also shown the Preamplifier part of the PCB. It's vital, so that low level signals are amplified to a "standard" operating level. The IC used here (and every time an op amp is needed for this PCB) is the TS922 and it consists of 2 separate op-amps. It is shown in Figure 2.6 along with the IC's pin layout.

Figure 2.6: Pin connections for the operational Amplifier TS922 (for SO8 and TSSOP8 formats).



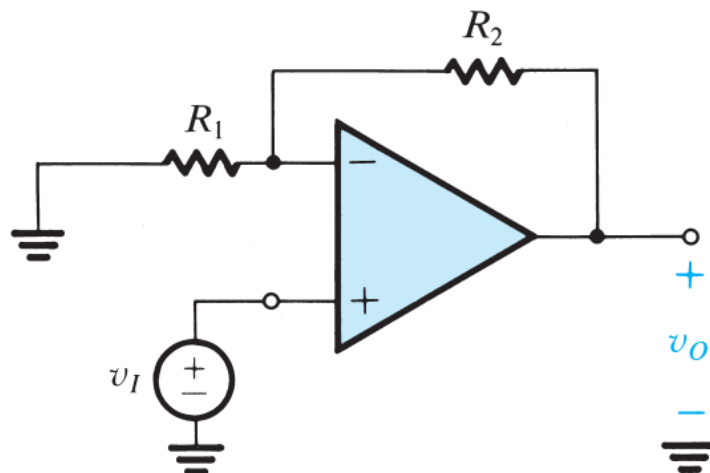
The op amp is in the noninverting configuration, and the resistor values are selected so that the gain is $\times 20$. Shown below is the equation that applies to this configuration and the "standalone" circuit [8, p. 73]:

$$Gain = \frac{V_o}{V_i} \Rightarrow Gain = 1 + \frac{R_2}{R_1} \quad 2.1$$

Substituting the values for the resistors used in the PCB results in:

$$Gain = 1 + \frac{95k\Omega}{5.1k\Omega} = 19.627451 \quad 2.2$$

Figure 2.7: The op amp configuration used for the preamplifier; the non-inverting configuration.

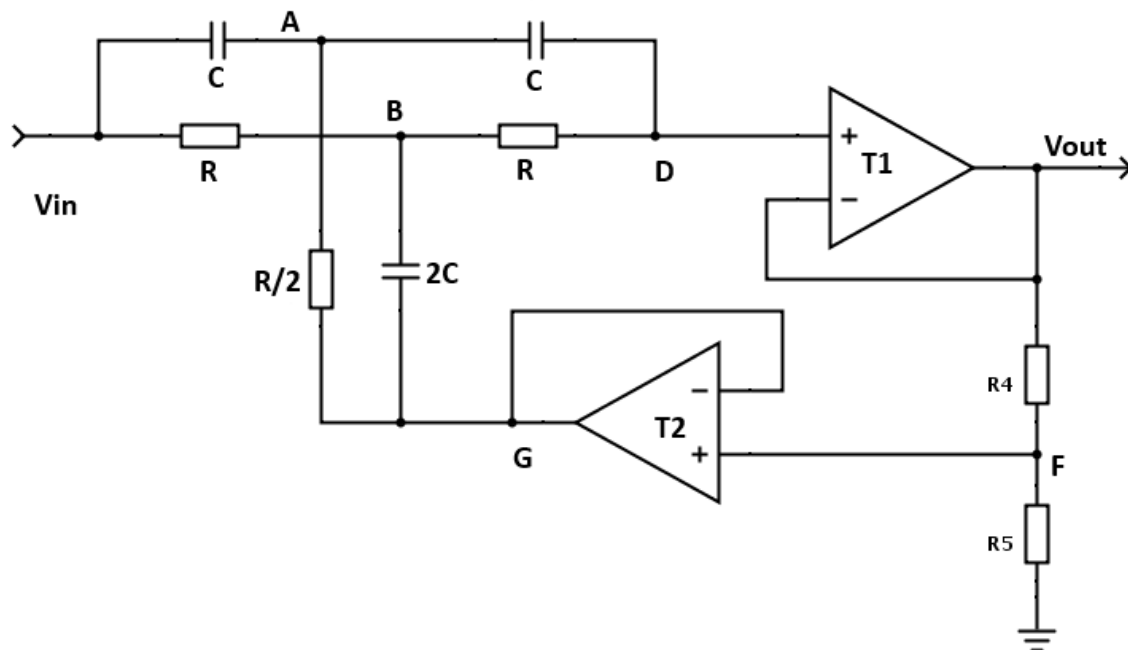


Note: [8, p. 74]

Notch Filter

Lastly, the part of the PCB where the power supply noise will be filtered out – the notch filter. The filter is depicted on Figure 2.8 and the schematic used, on Figure 2.9. This filter configuration is called “Bootstrapped Twin-T notch filter”. First, while a larger number of resistors³⁴ were used, the overall resistance is the same, based on the *resistors-in-series* principle.

Figure 2.8: Bootstrapped Twin-T Notch Filter.



The transfer function that describes this circuit is³⁵:

$$H(s) = \frac{s^2 + \omega_0}{s^2 + \frac{\omega_0}{Q}s + \omega_0} \quad 2.3$$

Where:

$$\omega_0 = \frac{1}{R \cdot C} \quad 2.4$$

$$Q = \frac{1}{4(1 - k)} \quad 2.5$$

$$k = \frac{R_5}{R_4 + R_5} \quad 2.6$$

³⁴ Compared to the theoretical filter on Figure 2.8

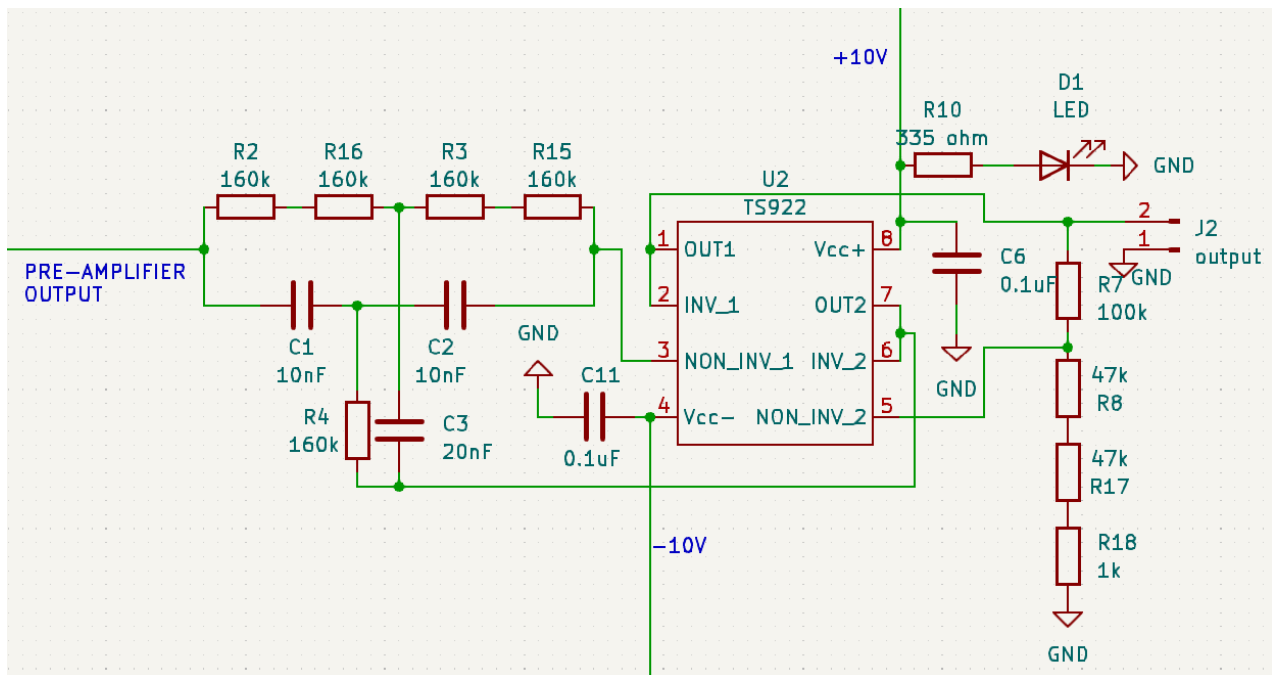
³⁵ All equations used in this subsection are derived on A.3

The first step is choosing the value of the Notch Frequency (F_{NOTCH}), and since the goal is filtering power supply noise, that value is 50Hz. To make this happen, the appropriate values for both R and C must be chosen according to:

$$\omega_{NOTCH} = \omega_0 \quad 2.7$$

so $F_{NOTCH} = 50 \Rightarrow \frac{1}{2\pi RC} = 50Hz$. Knowing what the product $R \cdot C$ must be equal to, the values shown in the schematic Figure 2.9 were opted for - $R = 320k\Omega$ and $C = 10nF$.

Figure 2.9: The notch filter schematic.

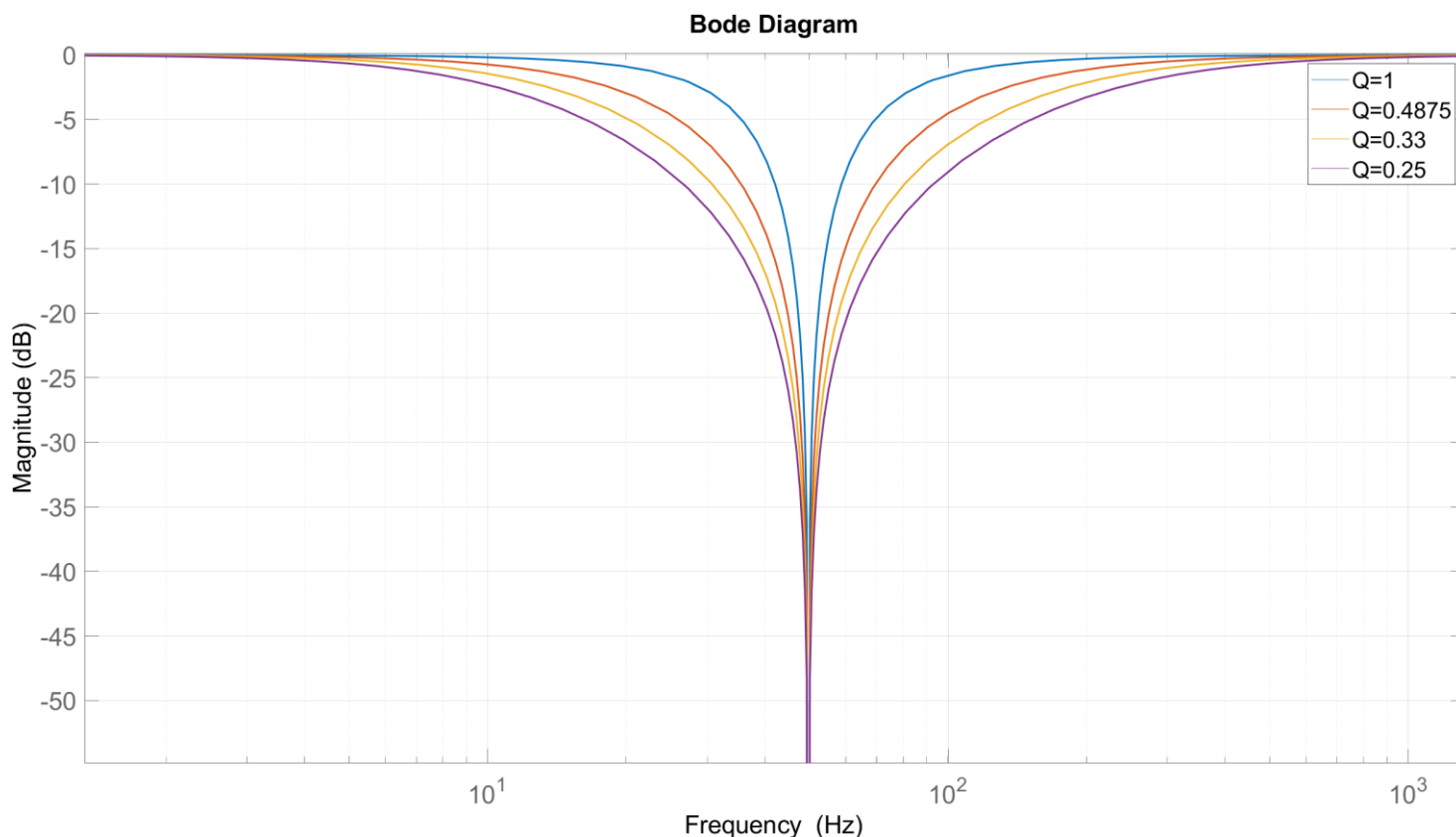


This still leaves the question of the voltage divider and the two op-amps. All three have a very significant role in the filter. Starting with the divider, already from the filter's equations – more specifically 2.5 and 2.6– it's evident that it directly affects the Quality factor of the system. The divider, and generally the whole feedback loop, is there so a higher Q factor can be achieved. Had it not been there, the value of the Q would be constant – opposed to now where it can be changed by changing the values of R_4 and R_5 ³⁶ – and lower³⁷. Lower Q means a not very precise notch, in other words, larger BW, resulting in more frequencies being affected than desired. With the voltage follower and the values chosen here, the quality factor is equal to $Q = 0.4875$.

³⁶ Or even live – while the system is on – by replacing the two resistors with a potentiometer.

³⁷ The value can be calculated by following the exact same method as in A.3, but for the same circuit – just without the feedback loop. It will result in: $Q = 0.25$.

Figure 2.10: Effect of Quality factor on the notch.

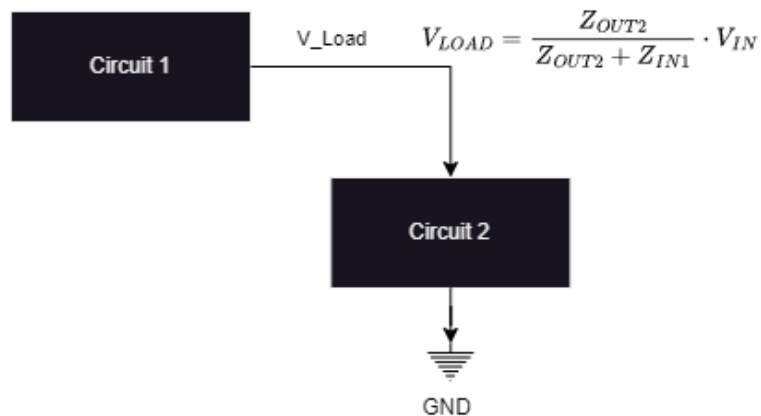


As for the op amps, both are configured in the same way – the Voltage Follower. They do not provide any amplification to their inputs, but rather isolate them from their output loop, thanks to their key principles [8, Ch. 2.1.2]; both inputs are at the same voltage, and at both the impedance is very high, meaning that they “draw” little to virtually zero current. Thirdly, the output impedance of an op amp is very low. Now, it’s very apparent that T2 (see Figure 2.8) is there so the voltage divider is functioning as expected and the well-known equation is an extremely good approximate³⁸.

T1 on the other hand, while using the same logic, is placed there for a slightly different reason. Generally, when sending a signal from one circuit to another, the input impedance of the latter (can be considered load impedance), as well as the output impedance of the former must be taken into consideration. They can form a voltage divider (see Figure 2.11) and have unwanted effects on the transmitted signal. Having a voltage follower on the output means very low output impedance and minimizes the parasitic voltage divider effect.

³⁸ If there is a significant amount of current drawn from the node between the two resistors, the equation $V_{OUT} = \frac{R_2}{R_2+R_1} \cdot V_{IN}$ is invalid. If there is a small amount of current, the equation can be still used, as it is a very good approximate.

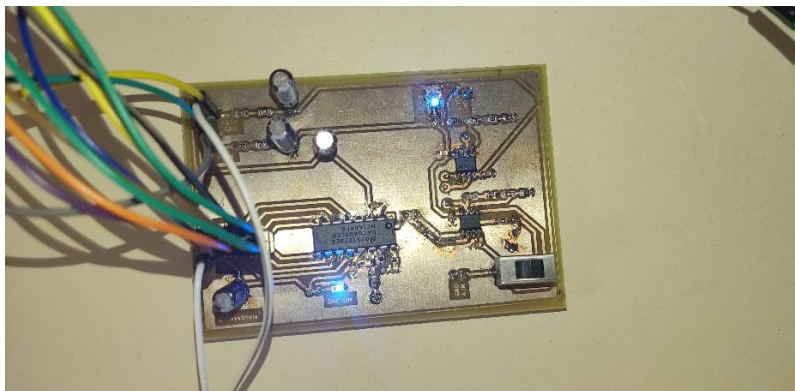
Figure 2.11: Impedances between subcircuits.



COMPLETE PCB

Now that all subcircuits have been examined, the only thing left is to combine them. The complete PCB schematic will be found on the Appendix (see here A.1), while 3D pictures of the PCB as well as layouts of both sides of the PCB are found on the next two pages. Before the conclusion of the hardware section, it is important to comment on a component shared among all subcircuits of the PCB; the decoupling capacitors. On the schematics several capacitors can be noticed either next to the PCB's power supply pins, or right before the V_{CC} pins of the ICs, all seemingly routing the traces to ground. The short answer as to why they are there, is to filter out power supply noise. The exact reason and how that happens is analyzed in detail on A.2³⁹.

Figure 2.12: Photo of the Board during testing of the DAC.



³⁹ The PCB was designed using the open source KICAD software.

Figure 2.13: Front side of the PCB on KiCad 3D Viewer.

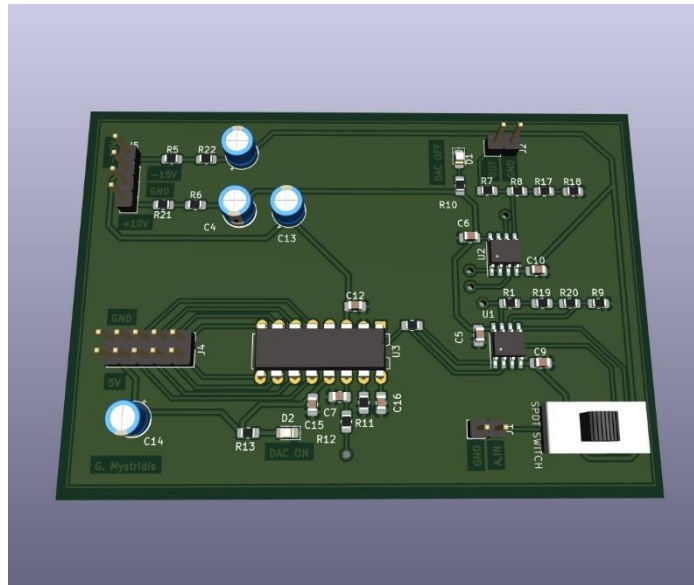


Figure 2.14: Back side of the PCB on KiCad 3D Viewer.

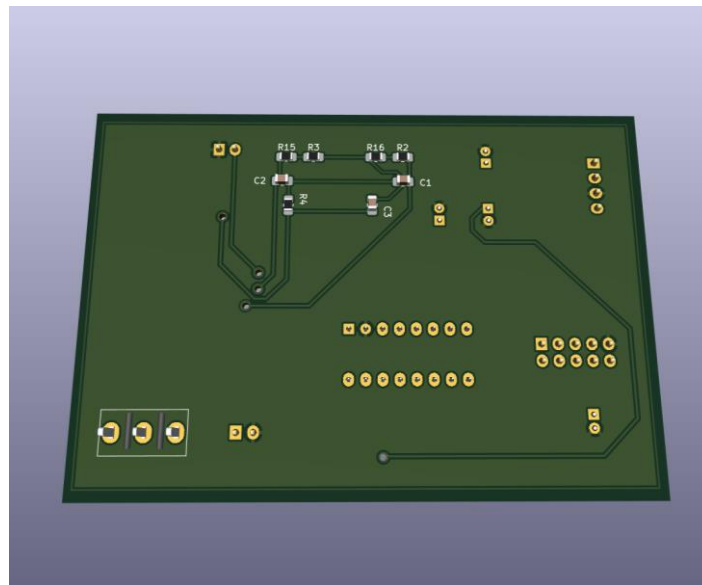


Figure 2.15: PCB Schematic of the front of the board.

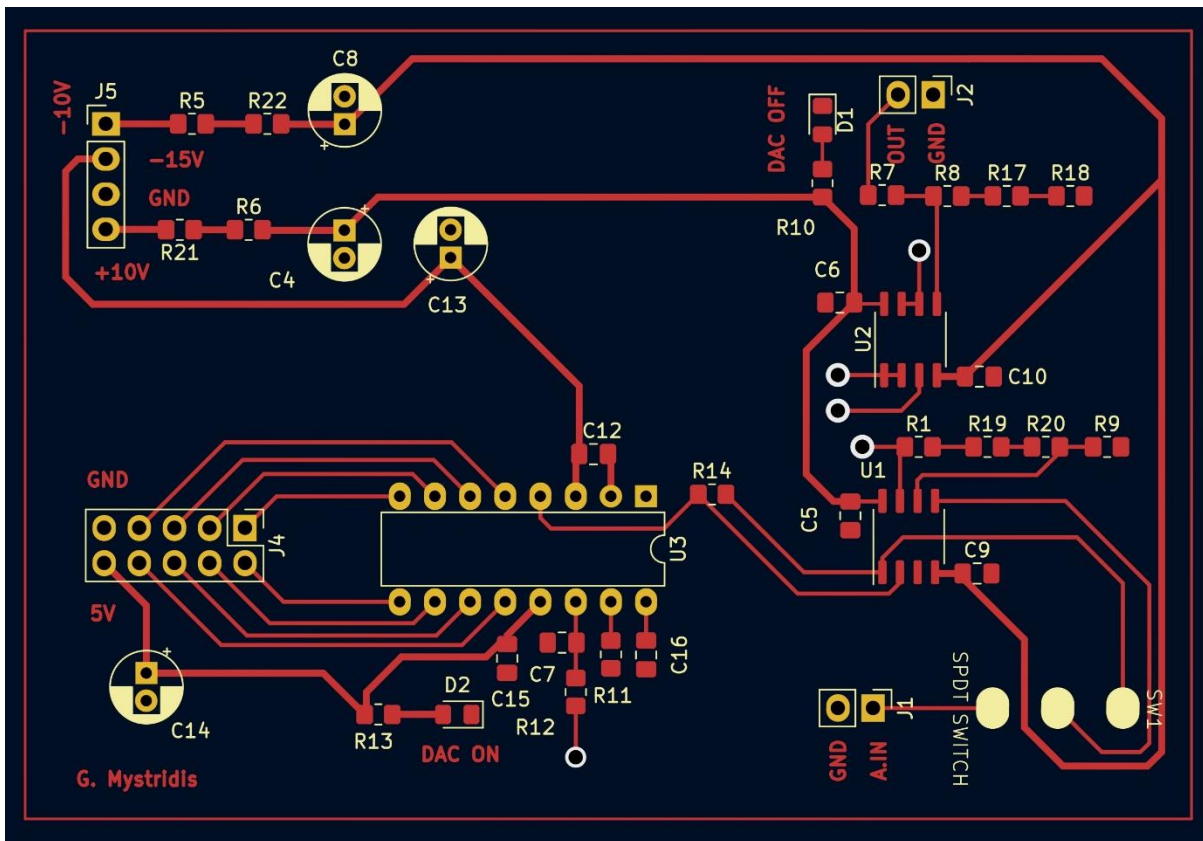
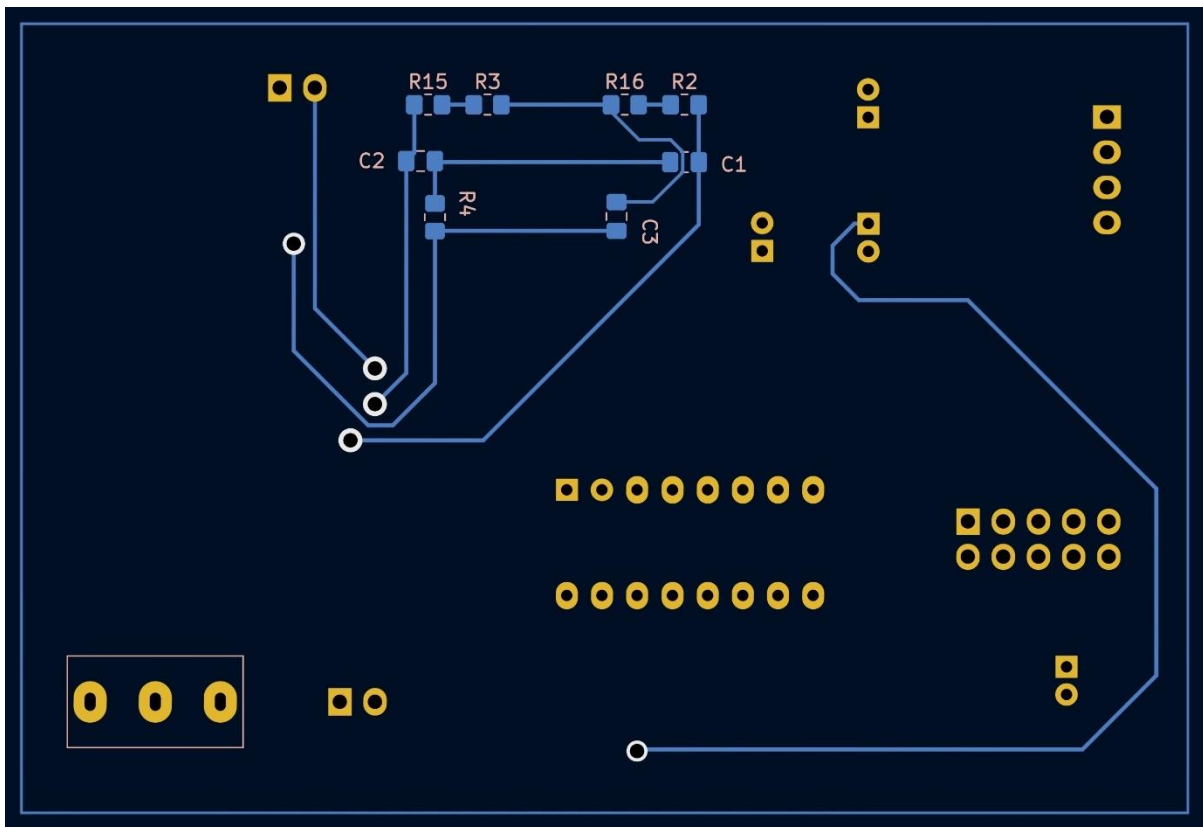


Figure 2.16: PCB Schematic of the back of the board.

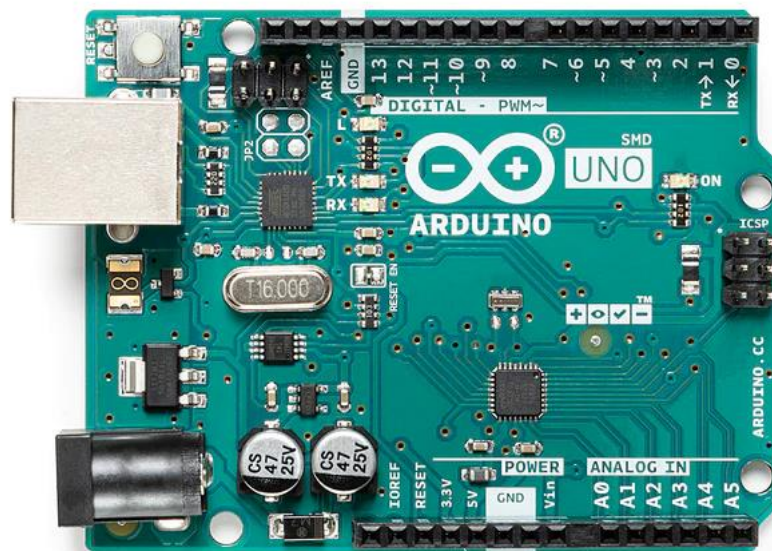


2.1.2 Arduino Firmware

The interfacing between the Computer and the Custom PCB was done using an Arduino Uno R3 board. The data from the PCB were transmitted to the Arduino board by connecting the output signal pin from the PCB to the A_0 *analog in* pin on the Arduino, while at the same time having the ground pins of both connected to each other to avoid floating voltages⁴⁰. Using the on-board ADC on the Arduino, the input signal is sampled every 2ms by reading the value of input⁴¹. That voltage immediately gets converted to digital and takes a value between 0 – 1023, because the integrated ADC is a 10-bit one. To change the input value to a range that corresponds to the voltage the pin is reading (which has to be from 0 – 5V for an Arduino Uno because that’s its limit), the following equation is used [9]:

$$Voltage = PinValue \cdot \frac{5}{1023} \quad 2.8$$

Figure 2.17: Arduino Uno R3 SMD version.



The sketch (as Arduino code files are named) starts with initializing most of the variables that will be used. Then comes the “*void setup*” function, that will be executed only one time. Within it, there is the “*Serial.begin(115200)*” command, that sets the baud rate for the communication between the computer and the Arduino board to 115200. Then follows the “*void loop()*” function. Whatever command is within it, will be executed sequentially forever until the board is turned off. This is where the sampling takes place using the “*analogRead(Insert Pin Number)*”. This command also uses the ADC and returns the value read on the pin. After the value is saved on a variable, it’s converted to the 0 – 5V range and

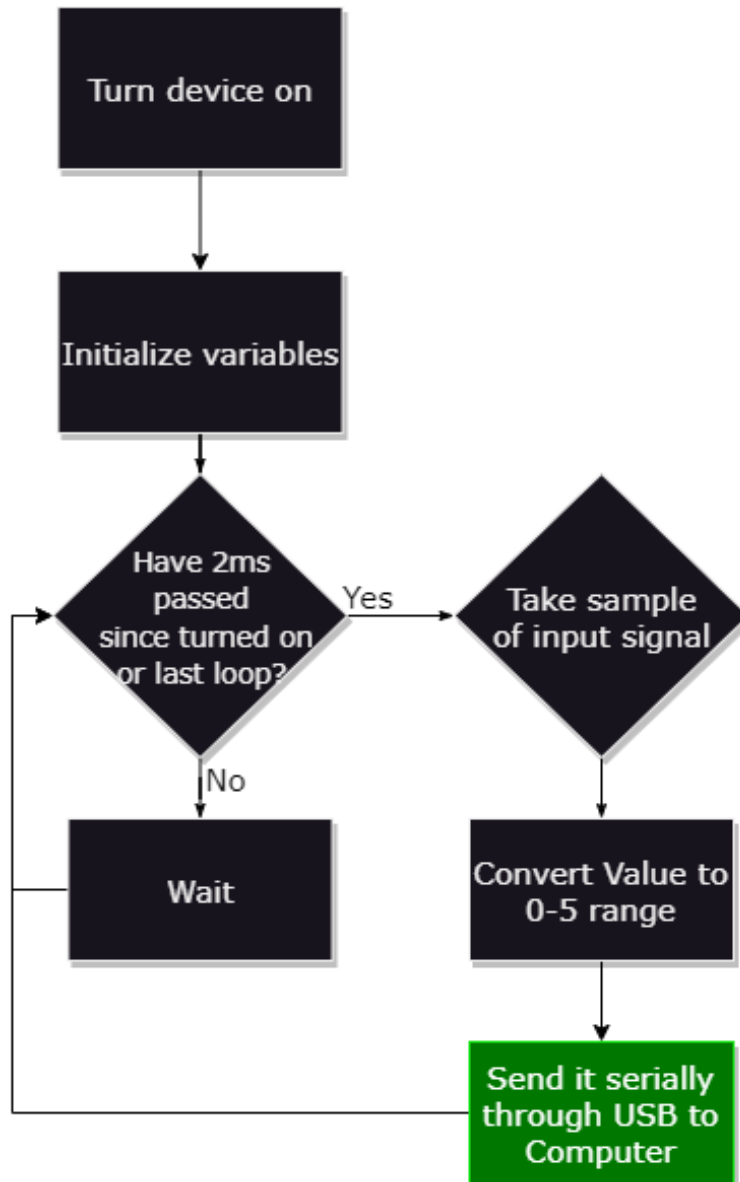
⁴⁰ The ground plane acts as a reference voltage for the input signal.

⁴¹ This interval is important, as it will be the value of the Sampling Frequency that will be used on the Software for further calculations.

sent to the computer serially through the USB port, using the Eq. 2.8 and the command “*Serial.println(Voltage_in,5)*” respectively.

The writing and loading of the firmware on the actual board was done using the Arduino Integrated Development Environment (Arduino IDE).

Figure 2.18: Flow chart of the Arduino Firmware.



2.2 Developing the Software Application

In this chapter, the software part of the system will be analyzed. Key parts of the code, as well as the Digital filter will be examined.

The entirety of the software was written MATLAB. It is a standalone app that can be installed on any computer, without the requirement of having MATLAB preinstalled, as long as the hardware can support it.

2.2.1 The interface between the Hardware and the Software

The first step in interfacing an Arduino with this app is matching the Baud rate of communication. From the Arduino side it can be done while writing the firmware, and it is covered in 2.1.2.

From the MATLAB side it is a little more complicated. First, the program needs to know to which USB port the Arduino is connected. When the user wants to start receiving data after connecting their USB-Serial Device (in this case an Arduino Uno R3), they need to press the “Start Sampling” button under the “TOOLS” tab on the top left of the main window. Then, MATLAB communicates directly with the OS (Windows) to see the existing ports on the computer [10]. It outputs the name of the available ports (for example “COM3”) and its description (for example “Communications Port”) on a matrix variable.

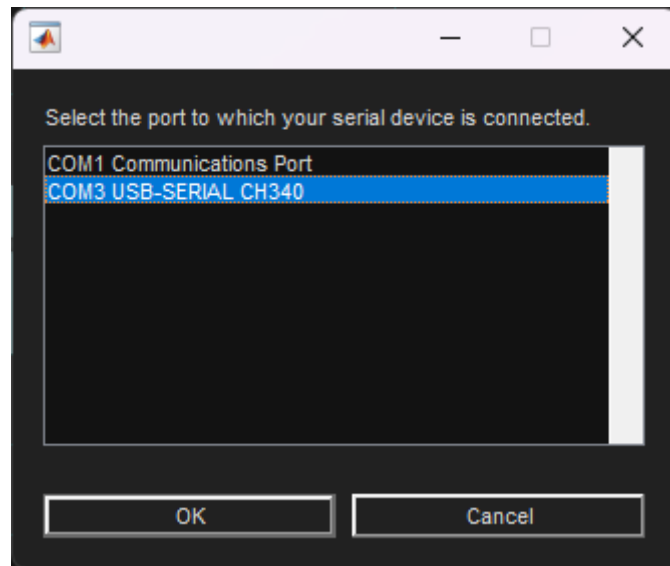
Figure 2.19: MATLAB communication with Windows for port availability.

```
1136 %% finding usb serial device connected to the computer
1137 NET.addAssembly("System.Management");
1138 mngmtQuery = System.Management.ObjectQuery();
1139 mngmtQuery.QueryString = "SELECT * FROM Win32_PnPEntity WHERE Name LIKE '%(COM%'";
1140 mngmtSearcher = System.Management.ManagementObjectSearcher(mngmtQuery);
1141 mngmtObjColl = mngmtSearcher.Get();
1142 comRep = repmat("",mngmtObjColl.Count,2);
1143 enMngmtObjColl = mngmtObjColl.GetEnumerator;
1144 p = 0;
1145 while enMngmtObjColl.MoveNext()
1146     p = p + 1;
1147     com = enMngmtObjColl.Current;
1148     caption = com.GetPropertyValue("Caption");
1149     comRep(p,1) = string(caption).extract("COM" + digitsPattern);
1150     comRep(p,2) = string(com.GetPropertyValue("Description"));
1151 end
```

The user then is met with a small window that consists of a list with two columns containing the available ports and their description respectively. They would have to pick a port with a description similar to “USB SERIAL DEVICE”, because that’s how a board like Arduino Uno would show up on Windows⁴².

⁴² The full description of the sequence of events is shown on the flow chart on Figure 2.23

Figure 2.20: Window for the user to select the port.



After picking the desired port, the user is met with another window, asking to select the values of some key variables for the sampling⁴³ that is about to take place. Those are:

- Sampling Frequency: This variable needs to be matched with the firmware in the Arduino. In this case, Arduino samples its analog input every 2ms, so the correct value for the sampling frequency would be $\frac{1}{2(ms)}$.
- Number of samples: User gets to decide how many samples they want to receive. The more samples, the better the analysis of the signal that will follow.
- Voltage Offset: Arduino can receive only positive input through the analog in pins, between 0-5 Volts. This means the custom PCB's input needs an offset – for example if the input is a sine with amplitude of 2V, the only option is to shift it, so it oscillates around 3V⁴⁴.
- Baud rate: the measure of the speed of data transmission in a communication channel/port.

⁴³ Or to be specific, the data that are about to be received.

⁴⁴ This also needs to take into account the preamplifier on the PCB. So technically, the example set, would not work.

Figure 2.21: Prompt for the user to input some key for the sampling variable values.

Now that the (serial)port⁴⁵ and Baud rate are picked by the user, they need to be coded into the program. This is done by the MATLAB function “*serialport(PORT,BAUDRATE)*”⁴⁶. After that, an additional setting needs to be made: the terminator for both read and write communications with the specified serial port will have to be defined. This is done by the MATLAB function “*configureTerminator(device,terminator)*”. As seen on the actual code (Figure 2.22), “*device*” is the serialport object⁴⁷ and “*terminator*” is “*CR/LF*”, meaning that the terminator is set for both read and write. The program as it is, will not be sending data to the port (Arduino), so it is made that way in case of future updates.

Lastly, on the top right of the window, the connected port is always displayed.

Figure 2.22: Setting the port and defining the terminator to read and write.

```

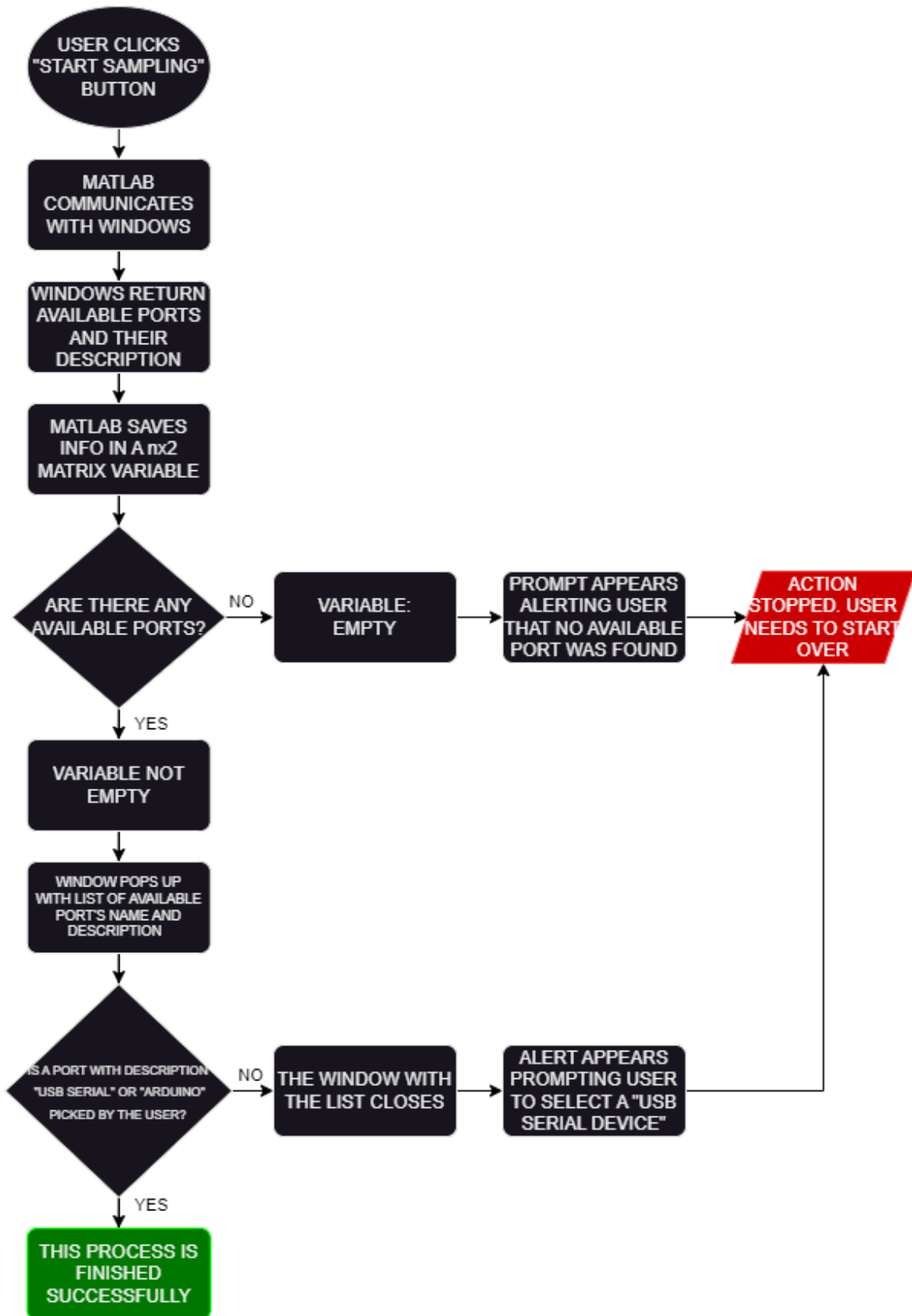
1110 try
1111     arduino = serialport(port_num,115200);
1112     catch ME
1113
1114         if (strcmp(ME.identifier,'serialport:serialport:ConnectionFailed'))
1115             msg=append('Unable to connect to the serialport device at port ',string(port_num),...
1116                 '. Verify that the port is not in use by another program, and that a device is properly connected to it.');
```

⁴⁵ “Serial”, since the data will be received one by one.

⁴⁶ In reality, after the user picks a port, this function is used with a seemingly hardcoded baud rate of 115200. But right after that, the input window shown in Figure 2.21 pops up, and the Baud rate will be immediately updated with the value chosen by the user.

⁴⁷ In a way, the output/return of the *serialport(PORT,BAUDRATE)* function.

Figure 2.23: Flow chart of the interfacing process on the MATLAB side.



2.2.2 Lowpass Butterworth Filter

This filter is responsible for eliminating signals with frequencies over 30Hz (ideally), meaning that it is a lowpass one. Having the software part of the system handle that, means that a digital one had to be designed. It has the IIR filter classification since it is made from the Butterworth approximation function – the Butterworth polynomial.

Starting with a normalized⁴⁸ Butterworth lowpass filter [11, p. 413]:

$$H_{norm}(s) = \frac{1}{a_0 + \dots + a_{n-1}s_{n-1} + a_n s_n} = \frac{1}{B_n(s)} \quad 2.9$$

Where $B_n(s)$ is the n^{th} order Butterworth polynomial, which can also be written as [12, p. 494]:

$$B_n(s) = \sum_{k=0}^n a_k s^k \quad 2.10$$

But the cutoff frequency will be at 30Hz, not at 1Hz. Because of that, the TF needs to be scaled (see 48) for a general cutoff frequency (angular in this case) ω_c :

$$H(s) = \frac{1}{\sum_{k=0}^n a_k \left(\frac{s}{\omega_c}\right)^k} \quad 2.11$$

Where the coefficients a_k can be calculated by the recursion formula [11, p. 413]:

$$a_{k+1} = \frac{\cos(k\gamma)}{\sin((k+1)\gamma)} \cdot a_k \quad 2.12$$

Or by [12, p. 494]:

$$a_k = \prod_{\mu=1}^k \frac{\cos((\mu-1)\gamma)}{\sin(\mu\gamma)} \quad 2.13$$

Where $a_0 = 1$ and:

$$\gamma = \frac{\pi}{2n} \quad 2.14$$

Now, having the continuous non-normalized TF (Eq.2.11), it's possible to derive the discrete time/digital TF. Using the Bilinear Transform Design and Eq.1.21:

$$H(z) = \frac{1}{\sum_{k=0}^n a_k \frac{1}{\omega_c^k} \left(\frac{2}{T} \cdot \frac{z-1}{z+1}\right)^k} \quad 2.15$$

⁴⁸ Normalized means it's in a form where $\omega_c = 1$. This is usually done to allow focus on the inherent characteristics of the filter design, such as order and behaviour, without being influenced by specific frequency and magnitude scales. To "un-normalize" or, in other words, scale a TF, simply substitute: $\left\{ \begin{array}{l} \text{for } H_n(\omega): \omega \rightarrow \frac{\omega}{\omega_c} \\ \text{for } H_n(s): s \rightarrow \frac{s}{\omega_c} \end{array} \right.$

Since the calculations are going to be done programmatically, the difference equation describing the, now discrete, system is needed. In general, the difference equation describing the system's output can be written as [2, p. 123]⁴⁹:

$$y(n) = \sum_{k=0}^M c_k \cdot x(n-k) - \sum_{k=1}^N d_k \cdot y(n-k) \quad 2.16$$

From the difference equation of a system its TF can also be derived according to [2, p. 125]:

$$y(n) = h(n) * x(n) \xrightarrow{\text{Z Transform}} Y(z) = X(z) \cdot H(z) \Rightarrow H(z) = \frac{Y(z)}{X(z)} \quad 2.17$$

Now, applying Z Transform on Eq.2.16:

$$Y(z) = Z \sum_{k=0}^M c_k \cdot x(n-k) - Z \sum_{k=1}^N d_k \cdot y(n-k) \quad 2.18$$

And using a property of the Z transform [2, p. 127]:

$$Z\{x(n-k)\} = X(z)z^{-k} \quad 2.19$$

So, Eq.2.18 becomes:

$$\begin{aligned} Y(z) &= X(z) \sum_{k=0}^M c_k \cdot z^{-k} - Y(z) \sum_{k=1}^N d_k \cdot z^{-k} \Rightarrow Y(z) \left(1 - \sum_{k=1}^N d_k \cdot z^{-k} \right) \\ &= X(z) \sum_{k=0}^M c_k \cdot z^{-k} \Rightarrow H(z) = \frac{\sum_{k=0}^M c_k \cdot z^{-k}}{1 - \sum_{k=1}^N d_k \cdot z^{-k}} \end{aligned} \quad 2.20$$

Now, if Eq.2.15 is expressed in the form of Eq.2.20 – that is a polynomial of z^{-k} on the denominator and the numerator – they would be directly comparable, meaning that each coefficient c_k and d_k is now calculated. Substituting them on Eq.2.16 yields the Butterworth filter's difference equations. An example of this but for a 2nd order Butterworth filter (also meaning $M, N = 2$) is on A.4.

This same exact logic is applied programmatically through MATLAB. The exact code handling this is shown on Figure 2.24.

⁴⁹ Notice that the equation is recursive as expected for an IIR filter.

Figure 2.24: Code snippet handling the difference equation.

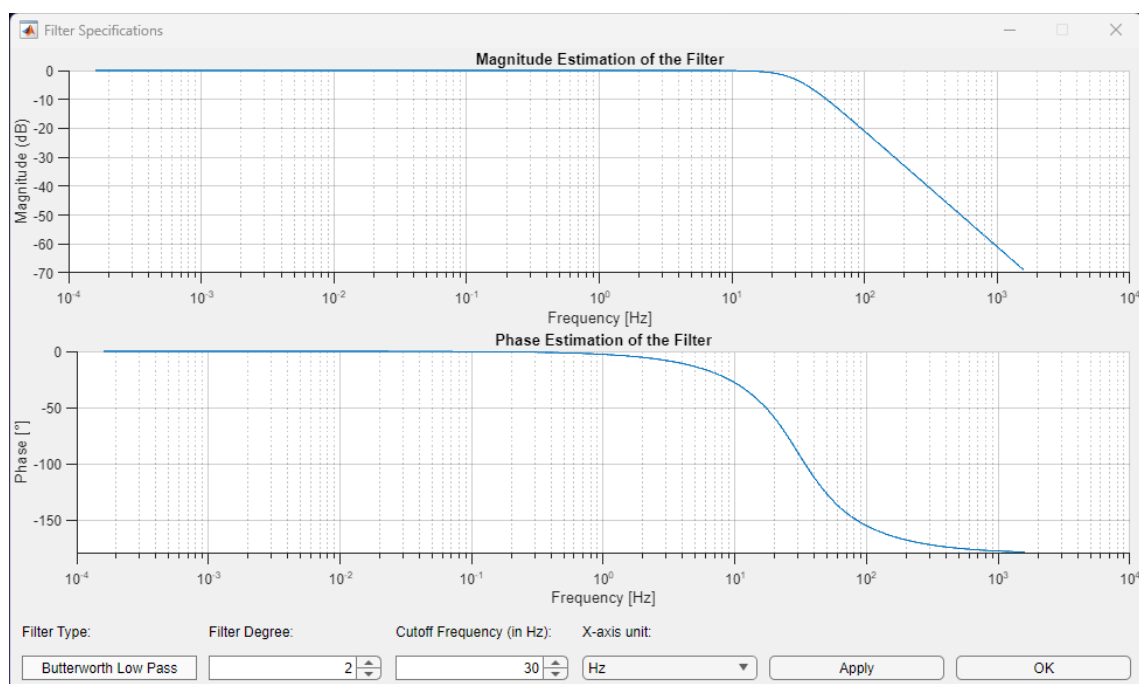
```

1520     %% all sums go from 0 to n . the 0-th will be initialized before the loop cause in matlab indexing an array starts from 1.
1521     %{ ***
1560     Ts=1/myclass.Sample_Freq;
1561     a=zeros(length(n)); % preallocating for speed
1562     coef=zeros(length(n)); % preallocating for speed
1563     a(1)=1; % this is the a_0
1564     g=pi/(2*n);
1565     coef(1)=1; % the tf im creating only has a polynomial denominator
1566     for k=0:1:n-1 % calculating the T.F. coefficients
1567         a(k+2)=a(k+1)*(cos(k*g)/sin((k+1)*g));
1568         %{ ***
1579         coef(k+2)=a(k+2)/wc^(k+1);
1580     end
1581     %creating the transfer function
1582     H_s=tf(1,flip(coef));% B = flip(A) returns array B the same size as A,
1583     % but with the order of the elements reversed. ***
1587     sysd = c2d(H_s,Ts,'tustin'); % bilinear transform
1588     [num,den] = tfdata(sysd); % taking the numerator and denominator coefficients
1589     f = cell2mat(den); % they come in cell form. Transforming in it into a matrix so i can use it later
1590     c = cell2mat(num); % they come in cell form. Transforming in it into a matrix so i can use it later
1591
1592     x=myclass.rt_data;
1593     yn=(myclass.Offset)*ones(length(x),1); % initializing;
1594     yn1=(myclass.Offset)*ones( [ length(x) (length(c)-1) ] );% preallocating for speed
1595     yn2=(myclass.Offset)*ones( [ length(x) (length(c)) ] );% preallocating for speed
1596     for nn=length(c):1:length(x)
1597         for i=1:length(c)-1
1598             yn1(nn,i)=-f(i+1)*yn(nn-i); % made nn by i so the values can be stored for troubleshooting
1599         end
1600         for i=0:length(c)-1
1601             yn2(nn,i+1)=c(i+1)*x(nn-i);
1602         end
1603         yn(nn)=sum(yn1(nn,:))+sum(yn2(nn,:));
1604     end

```

Additionally, there is a tool programmed into the app that allows the user to choose certain aspects of the lowpass filter (such as the order of the filter and the Cutoff Frequency within the range of 1 – 100Hz) and also displays, as graphs, the theoretical/estimated frequency as well as the phase response. It is accessed through a button on the top left of the main window called “Filter Specifications”. See below a photo of that tool.

Figure 2.25: Filter Specifications Window.

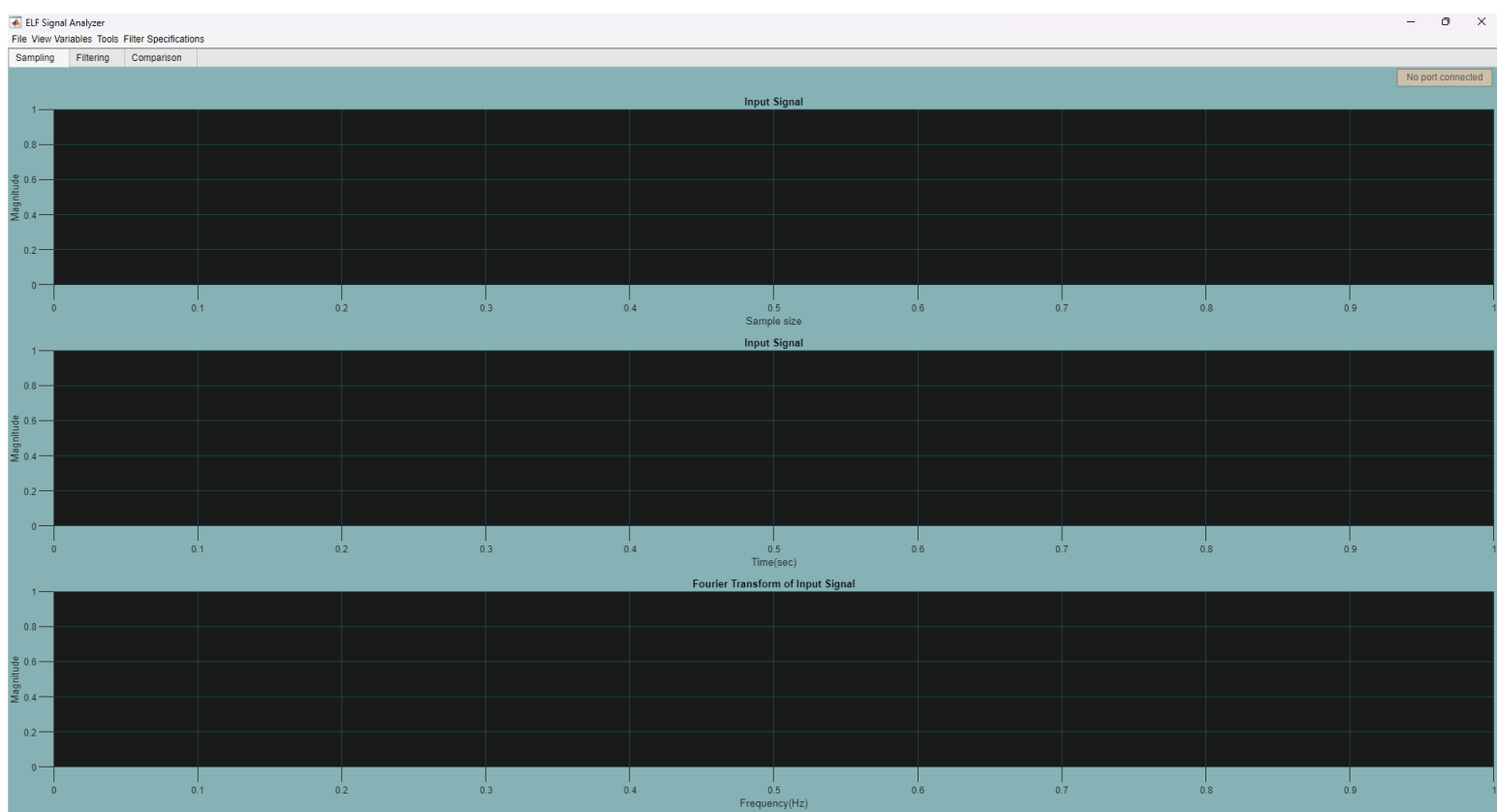


2.2.3 The software

Since the interfacing between hardware and software has already been discussed, this chapter will cover the remaining software features.

The application consists of three main tabs. Each one is a window with 3 graphs, and the user can switch between them from the top left of the window. Starting with the tab called “Sampling”, the first two graphs (counting from top to bottom) display the signal; the former with the x-axis being number of samples, and the latter with the x-axis being time (seconds). The third graph displays the Fourier transform of the input signal, showcasing its main frequencies.

Figure 2.26: Main window of the MATLAB app.



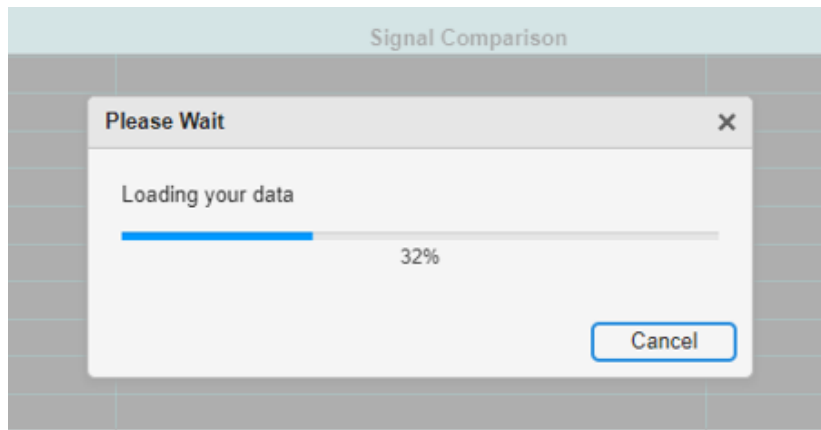
On the second tab, which is called “Filtering”, the two top graphs display the filtered signal (again with the x-axis alternating between samples and time respectively), and the last one displays its Fourier Transform.

The pattern continues on the last tab (called “Comparison”), which the three graphs display both the input and the filtered signals on top of one another, for a direct comparison⁵⁰. After

⁵⁰ Since usually the number of samples is large (the number is in the thousands) and the graphs will display the whole signal from start to finish, the outcome may look zoomed out or congested/cluttered. This is solved by a built-in MATLAB feature: When the cursor is hovering over a graph, several buttons are displayed on the top right of said graph. One of them is a zoom in button, that results in a drag box. The user can choose which part of the signal they want zoomed in and include it in the drag box. Figure 2.30 shows both before and after the zoom in.

the user follows the steps outlined while discussing the interface between hardware and software, the program will actually start receiving data via the USB port that was selected, and a progress bar will be displayed, showing the percentage of data already received.

Figure 2.27: Progress bar.



When the process is finished, the Sampling tab will look like on Figure 2.30. Now that the data have been received, the user can filter the input signal. To do that, they need to click: *Tools*→*Filter*. After filtering, all the graphs on tab 2 and tab 3 will be filled immediately.

As explained in 2.2.2, the user can change the order of the filter and the cutoff frequency, by opening the “Filter Specifications” window. If they filter the input, without opening that window first, then the app defaults the filter into a 2nd order one with $F_c = 30Hz$ ⁵¹. What was also failed to be mentioned, is that the user can filter the input multiple times (simultaneously deleting the previous filtered signal and all its data).

Figure 2.28: Data and Key Variables window.

| Input Signal | Time | Filtered Signal | Input Signal's Fourier Transform's Magnitude | Input Signal's Fourier Transform's Frequency (Hz) | Filtered Signal's Fourier Transform's Magnitude | Filtered Signal's Fourier Transform's Frequency... | Magnitude of Input Signal's Fourier Peaks | Position of Input Signal |
|--------------|--------|-----------------|--|---|---|--|---|--------------------------|
| 0 | 0 | 0 | 0.0488 | 0 | 2.7147 | 0 | 4.9978e-03 | |
| 0.3972 | 0.0010 | 0 | 0.0480 | 0.1000 | 2.7155 | 0.1000 | NaN | |
| 0.7290 | 0.0020 | 0.0118 | 0.0480 | 0.2000 | 2.7156 | 0.2000 | NaN | |
| 0.9409 | 0.0030 | 0.0423 | 0.0477 | 0.3000 | 2.7159 | 0.3000 | NaN | |
| 0.9980 | 0.0040 | 0.0924 | 0.0481 | 0.4000 | 2.7154 | 0.4000 | NaN | |
| 0.8910 | 0.0050 | 0.1577 | 0.0485 | 0.5000 | 2.7151 | 0.5000 | NaN | |
| 0.6374 | 0.0060 | 0.2294 | 0.0484 | 0.6000 | 2.7154 | 0.6000 | NaN | |
| 0.2750 | 0.0070 | 0.2983 | 0.0477 | 0.7000 | 2.7162 | 0.7000 | NaN | |
| -0.1253 | 0.0080 | 0.3487 | 0.0478 | 0.8000 | 2.7160 | 0.8000 | NaN | |
| -0.5090 | 0.0090 | 0.3708 | 0.0484 | 0.9000 | 2.7154 | 0.9000 | NaN | |
| -0.8090 | 0.0100 | 0.3625 | 0.0490 | 1.0000 | 2.7149 | 1.0000 | NaN | |
| -0.9759 | 0.0110 | 0.3208 | 0.0491 | 1.1000 | 2.7151 | 1.1000 | NaN | |
| -0.9823 | 0.0120 | 0.2487 | 0.0476 | 1.2000 | 2.7167 | 1.2000 | NaN | |
| -0.8271 | 0.0130 | 0.1581 | 0.0470 | 1.3000 | 2.7172 | 1.3000 | NaN | |
| -0.5350 | 0.0140 | 0.0583 | 0.0474 | 1.4000 | 2.7169 | 1.4000 | NaN | |
| -0.1564 | 0.0150 | -0.0309 | 0.0487 | 1.5000 | 2.7157 | 1.5000 | NaN | |
| 0.2487 | 0.0160 | -0.1118 | 0.0484 | 1.6000 | 2.7163 | 1.6000 | NaN | |
| 0.8129 | 0.0170 | -0.1590 | 0.0493 | 1.7000 | 2.7155 | 1.7000 | NaN | |
| 0.8763 | 0.0180 | -0.1720 | 0.0487 | 1.8000 | 2.7161 | 1.8000 | NaN | |
| 0.9956 | 0.0190 | -0.1505 | 0.0491 | 1.9000 | 2.7159 | 1.9000 | NaN | |
| 0.9511 | 0.0200 | -0.0999 | 0.0495 | 2.0000 | 2.7157 | 2.0000 | NaN | |
| 0.7501 | 0.0210 | -0.0291 | 0.0502 | 2.1000 | 2.7155 | 2.1000 | NaN | |
| 0.4250 | 0.0220 | 0.0482 | 0.0493 | 2.2000 | 2.7165 | 2.2000 | NaN | |
| 0.0314 | 0.0230 | 0.1187 | 0.0492 | 2.3000 | 2.7163 | 2.3000 | NaN | |
| -0.3681 | 0.0240 | 0.1699 | 0.0489 | 2.4000 | 2.7169 | 2.4000 | NaN | |
| -0.7071 | 0.0250 | 0.1930 | 0.0503 | 2.5000 | 2.7159 | 2.5000 | NaN | |
| -0.9298 | 0.0260 | 0.1835 | 0.0499 | 2.6000 | 2.7164 | 2.6000 | NaN | |
| -0.9995 | 0.0270 | 0.1429 | 0.0504 | 2.7000 | 2.7164 | 2.7000 | NaN | |
| -0.9048 | 0.0280 | 0.0774 | 0.0502 | 2.8000 | 2.7167 | 2.8000 | NaN | |
| -0.6813 | 0.0290 | -0.0022 | 0.0501 | 2.9000 | 2.7174 | 2.9000 | NaN | |
| -0.3090 | 0.0300 | -0.0028 | 0.0485 | 3.0000 | 2.7182 | 3.0000 | NaN | |
| 0.0941 | 0.0310 | -0.1513 | 0.0473 | 3.1000 | 2.7200 | 3.1000 | NaN | |
| 0.4817 | 0.0320 | -0.1983 | 0.0470 | 3.2000 | 2.7193 | 3.2000 | NaN | |
| 0.7802 | 0.0330 | -0.2103 | 0.0504 | 3.3000 | 2.7153 | 3.3000 | NaN | |
| 0.9688 | 0.0340 | -0.1909 | 0.0562 | 3.4000 | 2.7103 | 3.4000 | NaN | |
| 0.9877 | 0.0350 | -0.1412 | 0.0568 | 3.5000 | 2.7116 | 3.5000 | NaN | |
| 0.8443 | 0.0360 | -0.0693 | 0.0559 | 3.6000 | 2.7136 | 3.6000 | NaN | |

⁵¹ The ELF band.

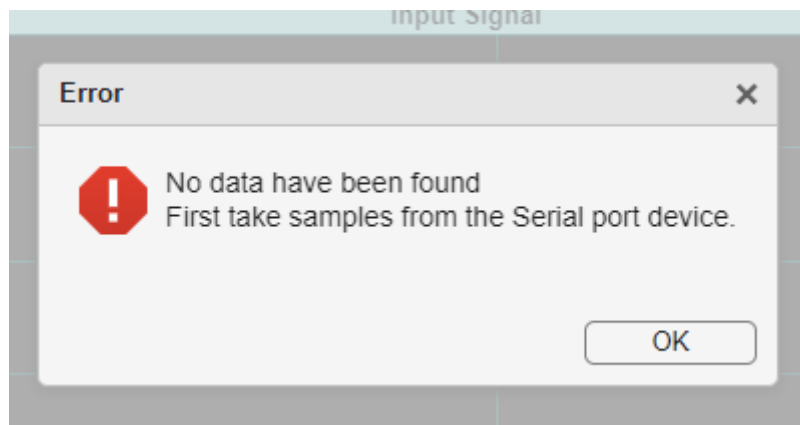
Another feature of the application is that all the data (data points, Cutoff frequency, sampling frequency etc.) are available to view on the “Data and Key Variables” window, accessed through the “View Variables” button on the top left (see Figure 2.28 above). From that window, there is the option to export all data in three ways:

1. a MATLAB “.mat” file
2. a text (.txt) file
3. an Excel file

while all of them will be formatted accordingly. The same option is given on the main application window, on the top left, through the “File” button. There, the user will be able to save all data in the three ways mentioned above, while also being given the option to load a previously saved file. It must be a MATLAB .mat file that was created previously by the same app.

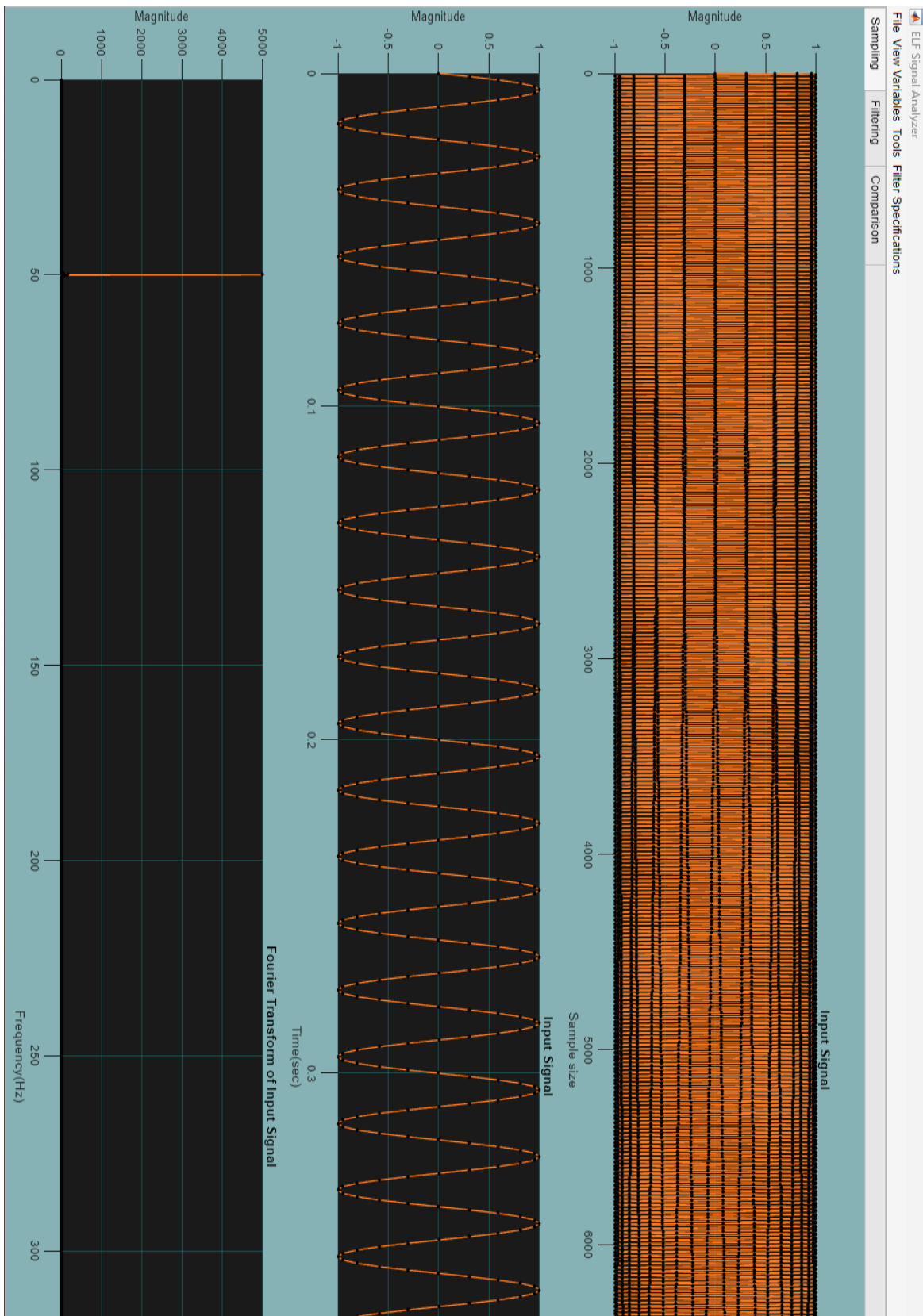
If a file made by a different MATLAB app is selected, then an error message will appear stating what went wrong and advising the user how to properly load a file. The same approach is followed for any misuse of the app, for example if the “Filter” button is pressed without having received any input data yet (input signal from the port), an error message will appear describing that there are no data/signal to filter and prompting the user to first take samples from the serial port. That message is displayed on Figure 2.29 below.

Figure 2.29: One of the error messages displayed, when the app is not used properly.



All messages and the general process is displayed on the flow chart on Figure 2.35.

Figure 2.30: Graphs with plotted data.



Coding the Fourier Transform

It is mentioned that the last graph on every tab is a plot of the Fourier Transform of either the input or the filtered signal. Specifically, it's a magnitude response of the Fourier Transform, which yields the main frequencies of the input signal. To provide a thorough explanation of the method used here, a more in-depth theoretical foundation is required that what was provided in 1.4. The Fourier analysis is a family of mathematical techniques based on decomposing signals into sinusoids. Choosing which member to utilize depends on if the signal is periodic or not and if it discrete or not. Table 1 sums up all four different cases [13, p. 144].

The Fourier Series states that every periodic continuous time signal can be represented by an infinite number of sinusoids. The mathematical expression is [6, p. 191]:

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{-jk\omega_0 t} \quad 2.21$$

Where a_k are complex coefficient measuring the portion of the signal that is at each harmonic of the fundamental component. The frequency of the initial signal is ω_0 , and by the expression above, it's apparent that the frequency of each sinusoid is $k \cdot \omega_0$. These coefficients are given by [6, p. 191]:

$$a_k = \frac{1}{T} \int_T x(t) e^{-j\omega_0 kt} dt \quad 2.22$$

Where T is the period of $x(t)$.

The Fourier Transform avoids the limitations of the periodicity of $x(t)$, and expands the same idea for non-periodic signals, building on the notion that an aperiodic signal can be viewed as a periodic signal with an infinite period. More precisely, in the Fourier series representation of a periodic signal, as the period increases, the fundamental frequency decreases and the harmonically related components (a_k) become closer in frequency. As the period becomes infinite, the frequency components form a continuum and the Fourier series sum becomes an integral [6, p. 284]. Based on this, the Fourier Transform formula can be derived (see Eq.1.11).

Table 1: The four Fourier approaches.

| Signal | Fourier approach |
|---------------------------|---------------------------------|
| Aperiodic-Continuous time | Fourier Transform |
| Periodic-Continuous time | Fourier Series |
| Aperiodic-Discrete time | Discrete Time Fourier Transform |
| Periodic-Discrete time | Discrete (Time) Fourier Series |

Additionally, the Fourier Transform can be expanded even further, to periodic signals. The resulted transform consists of a train of impulses (delta functions) in the frequency domain, with the areas of impulses proportional to the Fourier series coefficients [6, p. 297].

Both of these have their discrete time counterparts called “Discrete Time Fourier Series” and “Discrete Time Fourier Transform” respectively. It would seem that the DTFT is the approach suitable for a computer implementation of Fourier transform. However, that’s not the case, because the spectrum concept obtained from the DTFT is a continuous function of frequency as shown in the mathematical definition:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \quad 2.23$$

For the sake of conciseness, here is the definitions of the inverse DTFT:

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n} d\omega \quad 2.24$$

Additionally, the four classes of signal on Table 1 all extend to positive and negative infinity. But a computer would only have a finite number of samples stored – not infinite. The way around this problem is to make the finite data look like an infinite length signal. This is done by “imagining” that the signal has an infinite number of samples on the left and on the right of the actual data samples [13, p. 144]. If these imagined points are a duplication of the actual samples, the signal looks discrete and periodic, making it ideal for a DTFS. If they have a value of zero, the signal looks aperiodic and the DTFT applies.

But it happens that synthesizing an aperiodic signal requires an infinite number of sinusoids⁵². This fact alone makes it impossible to calculate the DTFT in a computer algorithm. However, the DTFT sum can change into a computable form. First the continuous frequency variable ω must be sampled and then the limits of the sum must become finite. The result is called DFT (Discrete Fourier Transform) and its definition is [14, p. 303]:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j\left(\frac{2\pi}{N}\right)kn} \quad 2.25$$

Where N is the signal length.

So, now there is a computable form of Fourier Transform for aperiodic signals. What about periodic signals? As it turns out, the DFT and the DTFS are mathematically the same operation⁵³ [15, p. 4]. In addition to that, the summation of inverse DFT requires the time-domain signal (input $x[n]$) must also be periodic with a period of N [14, p. 318]. To sum up, the DFT is the go-to for both periodic and non-periodic signals when done

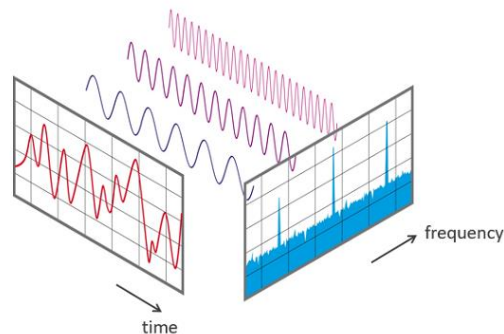
⁵² Directly related to the fact that the DTFT is a continuous function of ω .

⁵³ The difference between DTFS and DFT is a scaling factor of $\frac{1}{N}$, where N: number of samples[15, p. 4]. This is apparent when directly comparing the inverse DFT [16, p. 619] to the DTFS. Also, unlike the DTFS, it is a legitimate transform because an inverse transformation can be defined.

arithmetically on a computer, with the detail of “assuming” that the sampled signal (periodic or not) is a period of an infinite signal.

Looking at Figure 2.32, the DFT is done on line 987 using the function⁵⁴ “fft()”. Then on the next line the magnitude of the transform is taken, since the phase spectrum is of no importance for this part of the system. Next step is “rejecting” half of the values that comprise the transform.

Figure 2.31: View of a signal in the time and frequency domain.



Note: Signals - Frequency analysis. (n.d.). <https://makeabilitylab.github.io/physcomp/signals/FrequencyAnalysis/index.html>

The reason why is not obvious and very specific. The DTFT ($X(e^{j\omega})$) is always periodic with a period of 2π , and the DFT (as a sampled version of it) inherits this periodicity [14, p. 311]:

$$X[k] = X[k + N] \quad 2.26$$

When the signal $x[n]$ is real, there is also conjugate symmetry in the DTFT, meaning that, inherently, the DFT coefficients must also satisfy the following property [14, p. 312]:

$$X[-k] = X^* [k] \quad 2.27$$

Combining the two properties:

$$X[N - k] = X^* [k] \Rightarrow |X[N - k]| = |X[k]| \quad 2.28$$

Because of this, there are “duplicate” peaks on the magnitude spectrum that are of no importance in the context of this system, since the goal is knowing the main frequency of the input signal.

Figure 2.32: The function that performs Fast Fourier Transform in the app.

```

985 function [f_Hz, Y1] = fourier(myclass)
986     L=length( myclass.rt_data);
987     X_F=fft( myclass.rt_data);
988     X_mag=abs(X_F);
989     Y1=X_mag(1:round((L/2)));
990     f_bins=(0:(L-1)/2);
991     f_Hz = (myclass.Sample_Freq/L).*f_bins;
992     %% saving them as class properties so i can use them in different functions
993     myclass.Fourier_data=Y1;
994     myclass.Hertz=f_Hz;
    
```

⁵⁴ FFT (Fast Fourier Transform) is a very effective algorithm performing DFT.

Lastly, on lines 990-991 (Figure 2.32) the x-axis is converted from frequency bins into actual units of frequency [17]. These bins are essentially integers counting from 0 to $\frac{N-1}{2}$, where N is the number of samples⁵⁵. The division by 2 is there to avoid the mirroring effect that was analyzed previously. Basically, those frequencies will be ignored, as they will not be plotted. Then, the multiplication of those bins by the ratio $\frac{F_{sampling}}{Number\ of\ Samples}$ results in the true frequencies.

A thing to note is the frequency “resolution”. Each bin⁵⁶ is multiplied by the ratio mentioned above, meaning that the ratio is directly proportional to the accuracy of the true frequency values. For example, let’s say the input signal’s frequency is $f = 17.5Hz$. For the FFT to yield the exact result, along with respecting the Nyquist theorem, the ratio must be:

$$\frac{F_{sampling}}{Number\ of\ Samples} = 0.5 \Rightarrow \boxed{Number\ of\ Samples = 2 \cdot F_{sampling}} \quad 2.29$$

Communication Throughout the application

Error! Reference source not found. shows the source code of the application when it’s collapsed⁵⁷. It makes it apparent that the app is comprised of several standalone functions⁵⁸. All those functions need to communicate effectively and have equal access to all important/necessary variables for the application to work properly, and that doesn’t come inherently. To overcome that problem, a class object was created, the code of which is provided in A.8.

In MATLAB there are two types of classes [18]:

1. Value classes
2. Handle Classes

The former enable the user to create new array classes that have the same semantics as numeric classes, while the latter define objects that reference the object. Copying an object creates another reference to the same object [18]. Their basic difference is that a *value* class constructor returns an object that is associated with the variable to which it is assigned. If you reassign this variable, MATLAB® creates an independent copy of the original object [19]. If you pass this variable to a function to modify it, the function must return the modified object as an output argument.

A *handle* class constructor on the other hand, returns a handle object that is a reference to the object created. You can assign the handle object to multiple variables or pass it to functions without causing MATLAB to make a copy of the original object. A function that

⁵⁵ These bins are manually set for this app on line 990. The bins used by default from MATLAB (when someone plots the magnitude of the FFT using “*plot(magnitude)*”) range from 1 to N .

⁵⁶ From a programming point of view, the bins are elements of a matrix.

⁵⁷ For example, when a function is collapsed only its name is displayed and not its contents. It can of course be expanded to view every line of code within.

⁵⁸ And not nested ones.

modifies a handle object passed as an input argument does not need to return the object [19]. Simply put, copying a handle object does not copy the underlying data associated with the object. The copy is another handle⁵⁹ referring to the same object. Therefore, if a function modifies a handle object passed as an input argument, the modification affects the original input object in the caller's workspace [20]. Based on that last property alone, it is quite obvious why a *Handle* type of class was chosen.

In MATLAB to create a new “handle” class is a little complicated. In reality, the *handle* class is an abstract class⁶⁰, meaning that an instance of it cannot be created directly. One must use the *handle* class to derive other classes, which can be concrete ones whose instances are handle objects.

Figure 2.33: The entirety of the source code for the application in a collapsed form.

```

1 + function main_function() ...
6 + function Contents(myclass,main_fig) ...
72 + function tab1_function(tab1,myclass) ...
147 + function Resize(src,~,~) ...
156 + function View_Var(~,~,myclass,main_fig) ...
270 + function Reset(~,~,myclass,main_fig) ...
307 + function Load_func(~,~,myclass,main_fig) ...
393 + function Close(~,~,myclass,main_fig) ...
435 + function SaveAsMat(~,~,myclass,main_fig) ...
466 + function SaveAsExcel(~,~,myclass,main_fig) ...
598 + function SaveAsText(~,~,myclass,main_fig) ...
750 + function checkifavailable(~,~,myclass,main_fig) ...
803 + function [port_num] = PortCheck(main_fig) ...
849 + function Input_Box(myclass,main_fig) ...
893 + function begin(myclass,main_fig) ...
898 + function t= mytimer(myclass,main_fig) ...
912 + function pull_data(~,~,myclass,t) ...
935 + function set_prog_bar(~,~,myclass,main_fig) ...
942 + function plot_tab1(~,~,myclass) ...
985 + function [f_Hz, Y1] = fourier(myclass) ...
996 + function tab2_function(tab2,myclass) ...
1062 + function plot_tab2(~,~,main_fig,myclass) ...
1093 + function filter(myclass) ...
1146 + function Filter_Spec_Window(~,~,myclass) ...
1266 + function Plot_Spec_Window(Bode_ax1,Bode_ax2,spn1,spn2,drp,myclass) ...
1323 + function Close_Window_Request(src,~,myclass) ...
1334 + function tab3_function(tab3,myclass) ...
1400 + function plot_tab3(myclass) ...

```

To define a *handle* class, a derivation of the “new” (on Figure 2.34 it’s the “*MyHandleClass*”) class from *handle* is necessary. This is done using the syntax [20] shown on Figure 2.34.

⁵⁹ A handle is a variable that refers to an object of a handle class. There are to MATLAB what pointer are to C++.

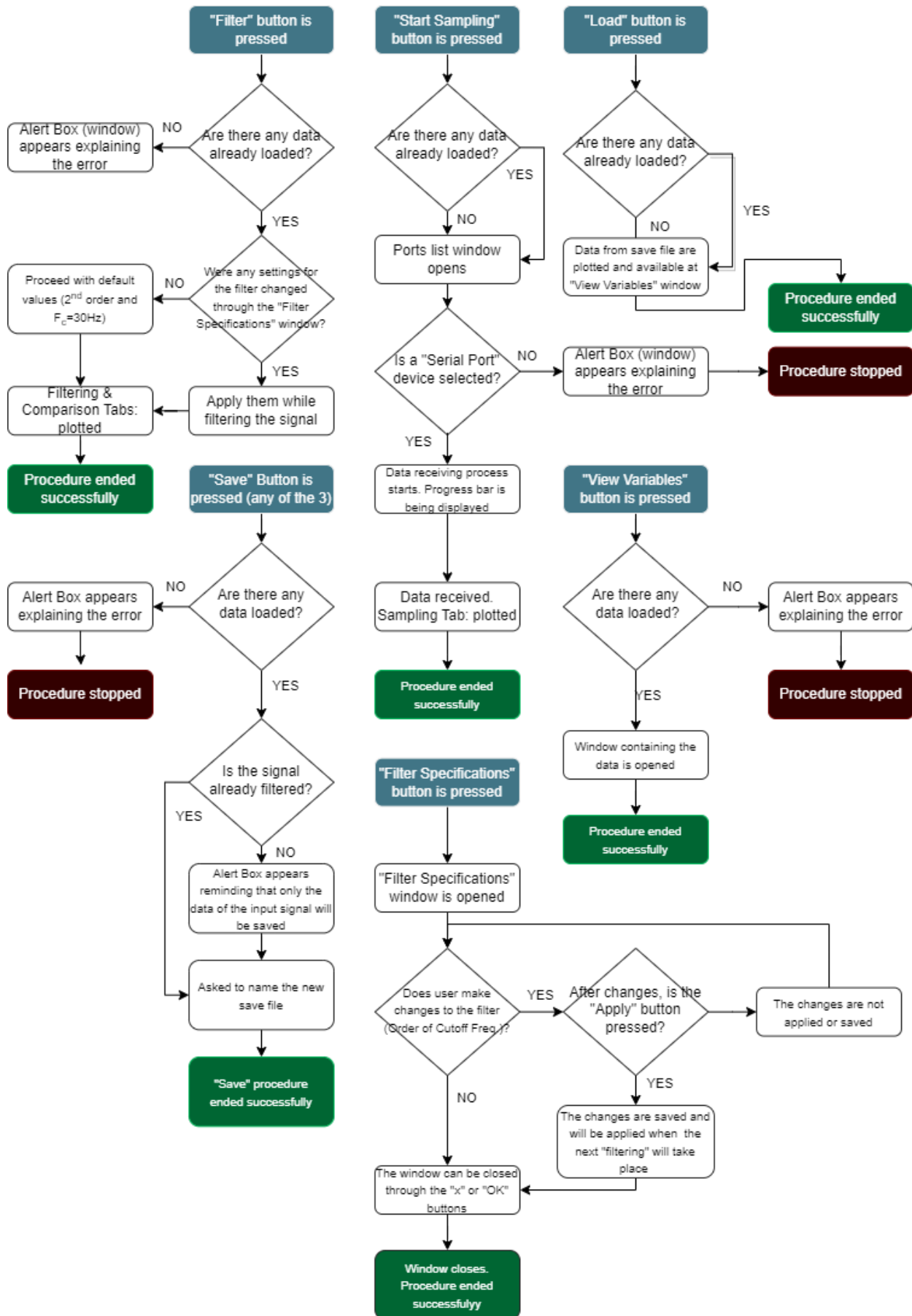
⁶⁰ The handle class is the superclass for all classes that follow handle semantics. A handle is a variable that refers to an object of a handle class. Multiple variables can refer to the same object[20].

Figure 2.34: Defining a class as a **Handle** class in MATLAB.

```
classdef MyHandleClass < handle
    ...
end
```

Using this type of class allows all the functions to access the same variable (if need be), and even change it, updating it for the other functions at the same time. All that needs to be done to use this within the app is to first create an object of the class, and then in order to give a function access to it, to use that object as an argument for that function when calling it. In this case, the object is “myclass” and the handle class is “Get_data_class2”. Creating it was done by the command “myclass = Get_data_class2;”.

Figure 2.35: Flow chart of the custom MATLAB application.



Chapter 3 Final Measurements and Conclusions

In this last chapter the final measurements and their comparison to theoretical expectations will be discussed along with final conclusions.

3.1 Transient Responses and Stability

As with every system, filters also have transient responses. Just as previously stated, to calculate it we need the transfer function.

3.1.1 Digital Filter

As a case in point, for the Butterworth filter, the transient response and stability will be calculated only the 2nd order filter with $F_c = 30\text{Hz}$ variant. The same variant was also used to plot the frequency response on a later subchapter.

The transfer function for the values stated above (see general at Eq.2.11) is⁶¹:

$$H_{2nd}(s) = \frac{1}{2.814e^{(-5)}s^2 + 0.007503s + 1} \quad 3.1$$

A quick calculation for the poles will yield these two answers:

- Pole 1: $p_1 = (-1.3329 + 1.3329j) \cdot e^2$
- Pole 2: $p_2 = (-1.3329 - 1.3329j) \cdot e^2$

Both are on the LHP, meaning that the system (filter) **is stable**.

Moving on to the expected transient response, a simple direct comparison between the TF and the general 2nd order TF's denominator (see Eq. 1.26) will provide the solution: $\zeta = 0.0038$. And since $0 < \zeta < 1$, the system is underdamped, meaning that a damped oscillation around the steady state value should be expected (of course before the steady state is reached).

⁶¹ While the actual filter is digital, the theoretical transient response and stability will be calculated using the continuous time TF. It will make virtually no difference, as both this and the discrete time TF describe two forms of the same system.

Transient Responses and Stability

Figure 3.1: Comparison between the Theoretical and the Filter's Step Response.

Note: The notable shift at the beginning is because the System data used for this graph, take into account the delay filtering causes.

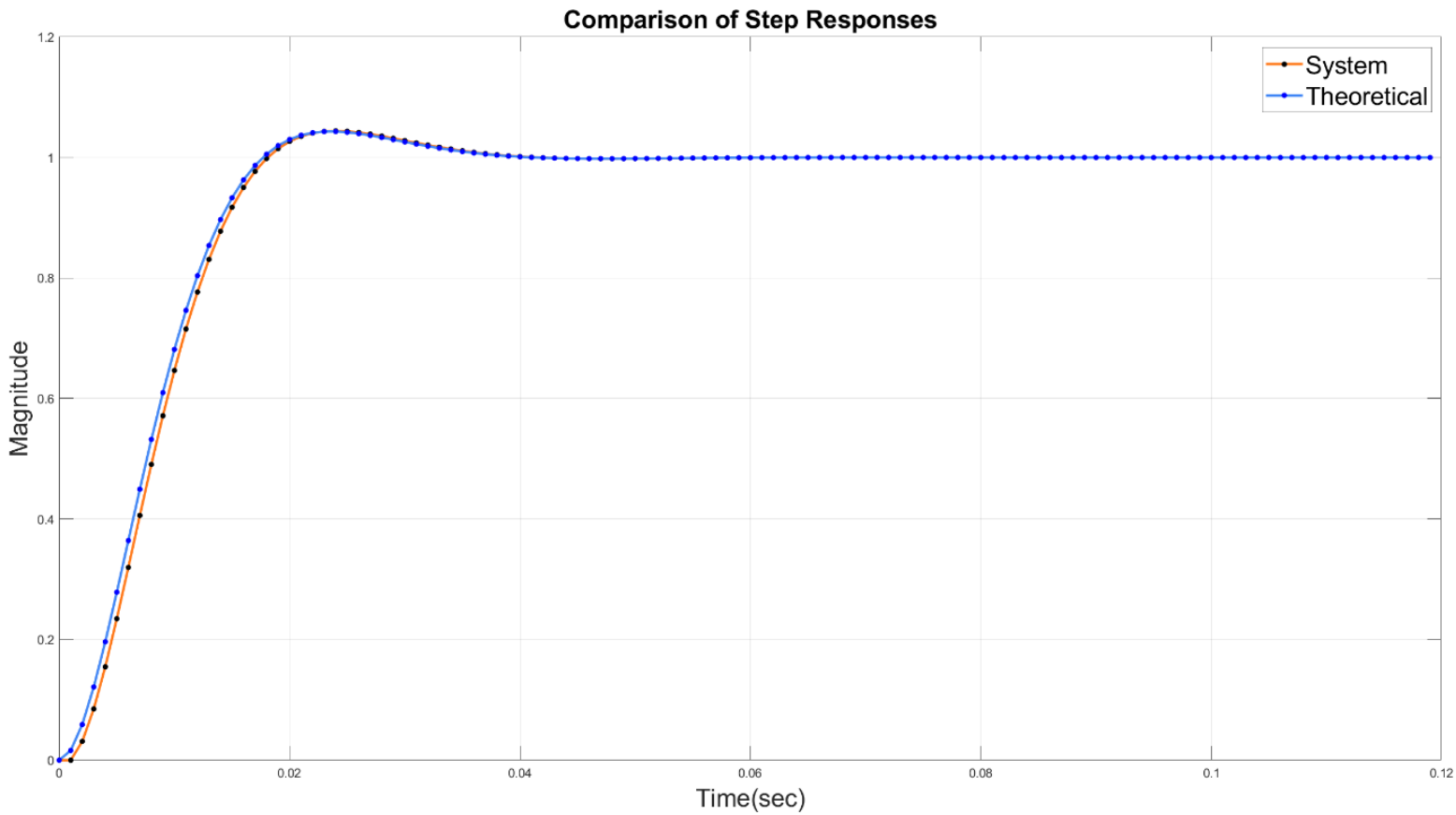
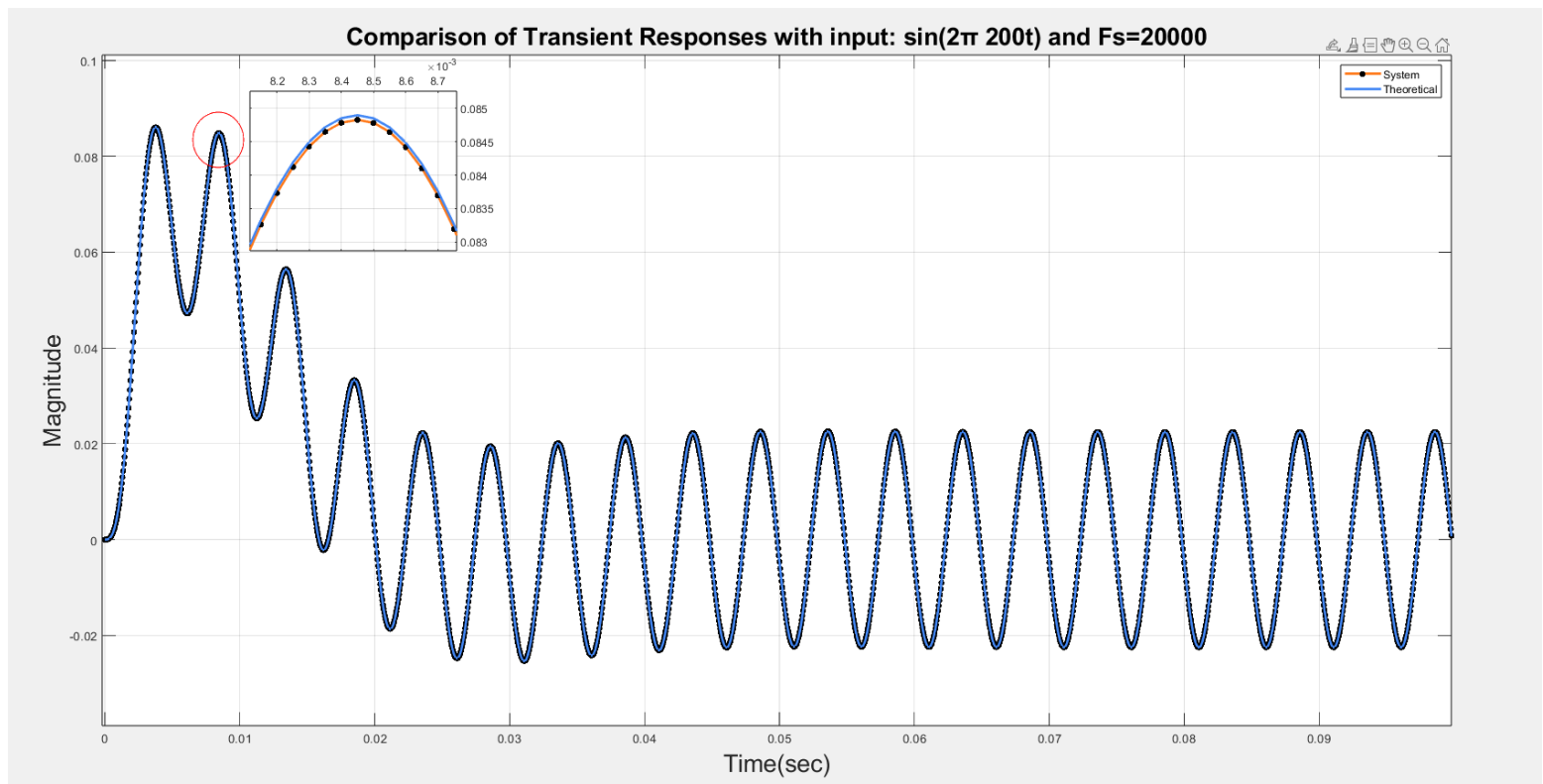


Figure 3.2: Comparison of Transient responses with a sine as an input.



3.1.2 Notch Filter

To calculate the transient response and stability for the notch filter, the same method will be followed. Comparing the denominator of the Notch filter's TF (Eq.2.3) to that of the general 2nd order TF yields $2\zeta\omega_0^2 = \frac{\omega_0}{Q} \Rightarrow \boxed{\zeta = 0.0033}$, meaning it's also an underdamped system. The reason the graph on Figure 3.3 doesn't resemble that of Figure 3.1 is because the Notch filter's TF has zeros, which, as aforementioned, affect the transient response.

Additionally, the two poles of the filter end up being:

- $p_1 = -391.7323$
- $p_2 = -249.2933$

Proving that the system is **stable** as expected.

Figure 3.3: Theoretical Step Response of the Notch Filter.

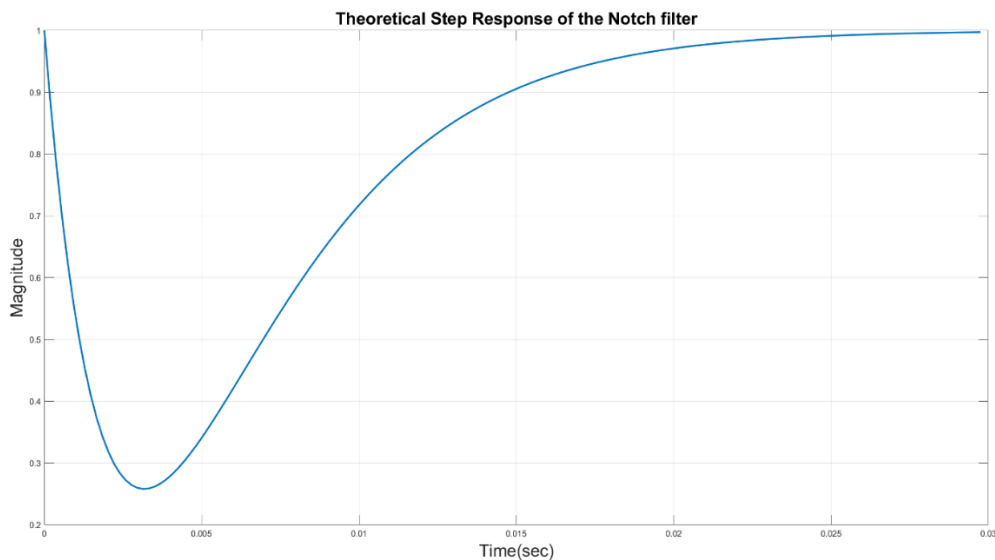
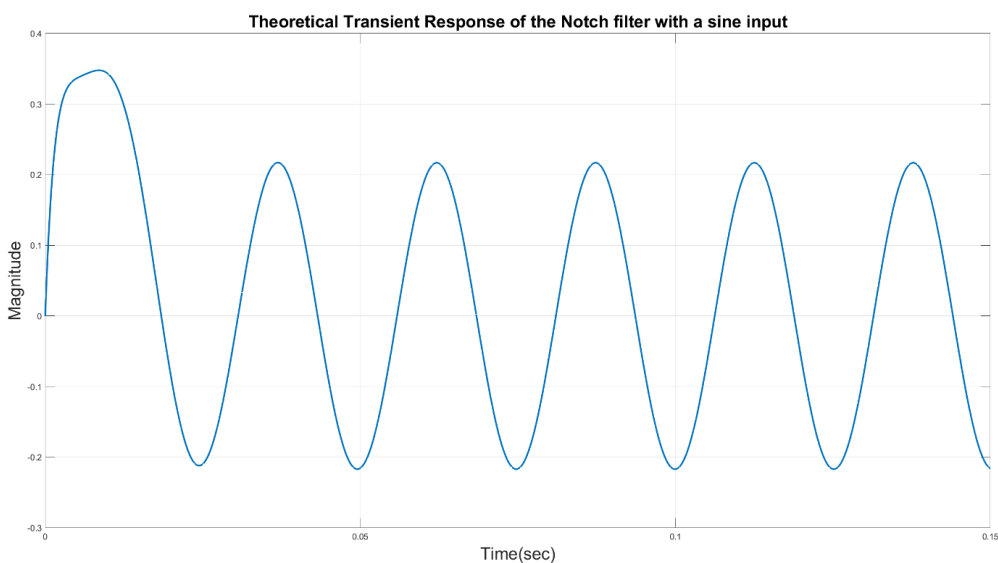


Figure 3.4: Theoretical Transient response of the Notch Filter with a sine as input.



3.2 Frequency Responses

An analysis on a filter wouldn't be complete without the magnitude/frequency⁶² responses.

3.2.1 Notch Filter's Response

Starting with the Notch Filter, measuring had to be done in real time on the PCB. To measure the Voltage (because the amplitude of the input and the output of the filter are needed to calculate the magnitude) an oscilloscope was used – specifically the PicoScope 2204 USB Oscilloscope. It's powered through a USB port by a computer, while simultaneously sending the data through it too. To use it, the PicoScope software is necessary, available for free on the company's website. Since it comes with an integrated function generator, it was used as a signal source too.

Figure 3.5: The PicoScope 2204A Oscilloscope by © 2024 Pico Technology Ltd.



Using The PicoScope as an analog signal source

Starting with the PicoScope as the (analog) signal source, the data were taken manually, by measuring the signal at the PCB's input and output pins. Using the PicoScope, the following values were measured: both the maximum and minimum values for the input and output ($V_{max_{IN}}$ and $V_{max_{OUT}}$, $V_{min_{IN}}$ and $V_{min_{OUT}}$) and the Peak-to-Peak Values⁶³ ($V_{pp_{IN}}$ and $V_{pp_{OUT}}$). Since they match (as seen in Table 2), only the peak-to-peak values will be used for the calculations.

Using the data shown in Table 2, the frequency response of the filter was plotted in Figure 3.6. On the y-axis is the ratio between input and output of the filter in dBs – calculated using $Mag = 20 * \log_{10} \frac{V_{pp_{OUT}}}{V_{pp_{IN}}}$, while the x-axis is simply the frequency in Hertz. One thing to keep in mind is that to avoid clipping due to saturation of the op amp and to keep the PCB's output signal to be within 0 – 5V (since it's sent to an Arduino Uno board), an offset had to be set,

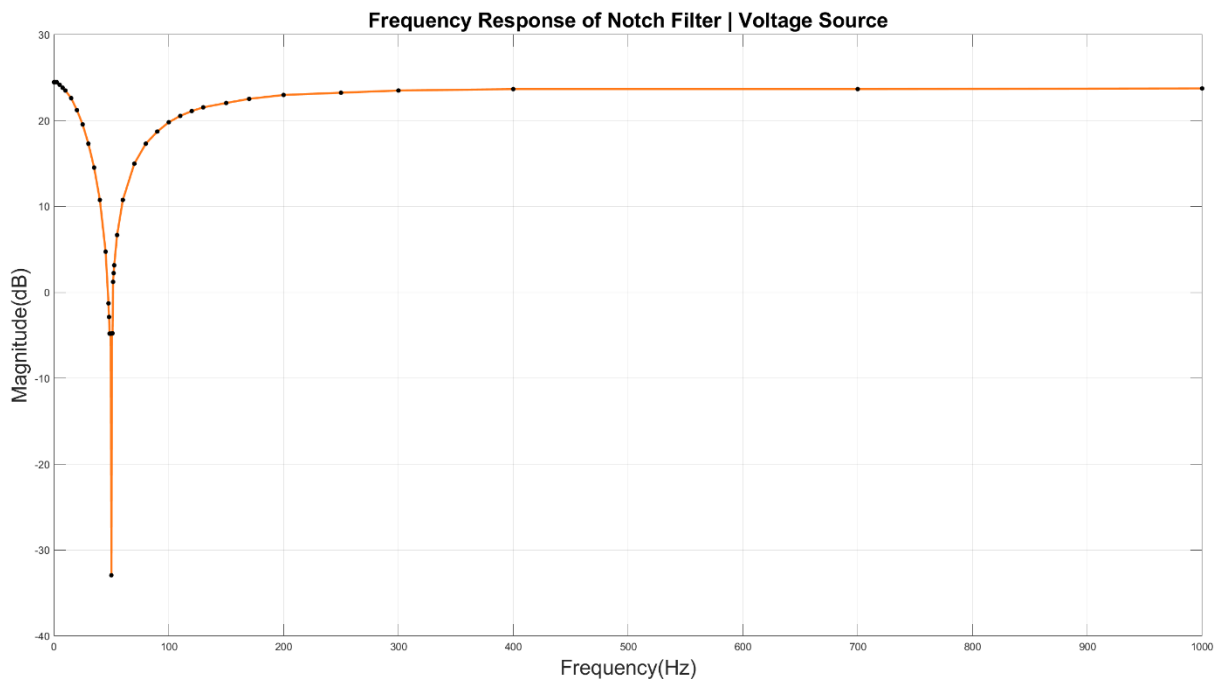
⁶² Magnitude/Frequency Response is the ratio of the output signal's amplitude to the amplitude of the input signal, both with respect to frequency.

⁶³ Both the min-max and peak-to-peak values are not needed but were taken for validation.

Frequency Responses

along with a limitation to the input signal's amplitude. Those values, in the end were: $V_{OFFSET} = 130mV$ and $Amplitude = 130mV^{64}$.

Figure 3.6: Frequency Response of the Notch filter.



On Figure 3.7 both the theoretical and the actual frequency responses are plotted. To calculate and plot the theoretical one, the filter was constructed in MATLAB using its TF. The frequency response was acquired using the built-in function “*bode(transf. function)*”. It’s clear that the maximum value for the y-axis is 0 for the theoretical plot. And that’s what it should be, given that $\log_{10}1 = 0$. The reason that the system’s plot maximum is at around 25dB, is because between the PCB’s input pin (which is where V_{ppIN} was measured) and the filter, is the preamplifier. Had the measurement taken place exactly before the filter, its max would be at 0 as well. But since it’s just “shifted” upwards compared to the theory, and on the x-axis they align, the comparison is still valid as is. However, in the interest of inclusivity, since the magnitude is calculated by the $20 \log_{10} \frac{V_{ppOUT}}{V_{ppIN}}$, and the preamplifier has a gain of 20, to roughly get an idea of what the graph would look like had the output of the preamplifier been considered as V_{ppIN} (on the above equation: $V_{ppIN} \rightarrow 20 \cdot V_{ppIN}$), all that needs to be done is subtract 20 from each magnitude point. This is thanks to the property of logarithmic functions:

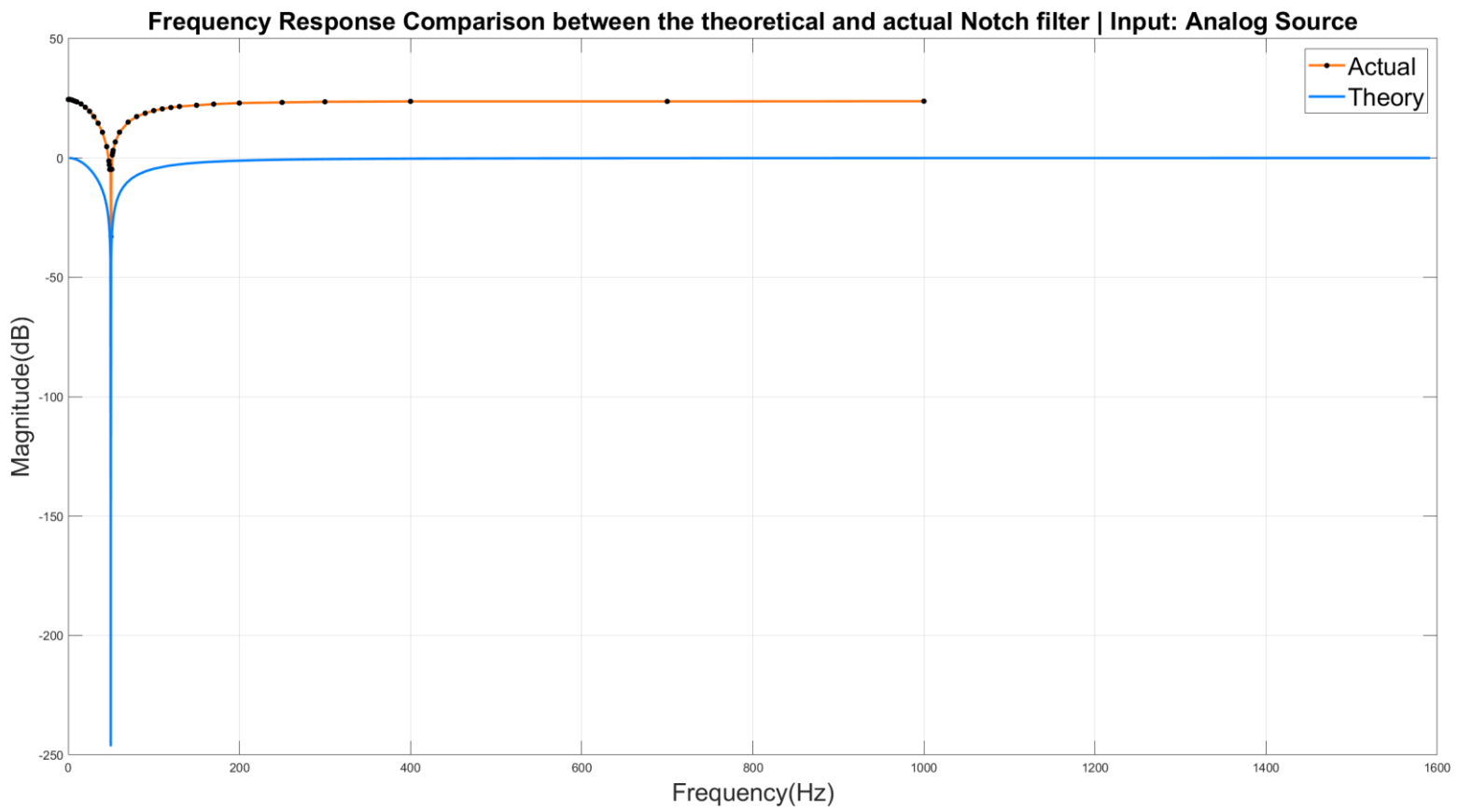
$$\log \frac{a}{b} = \log a - \log b \quad 3.2$$

Concluding this part of the testing, the notch filter is capable of a -32.928 dB drop, when the max output sits at $+24.477 \text{ dB}$. By a simple subtraction, the net attenuation is at $57,405 \text{ dB}$, making it a successful implementation.

⁶⁴ Theoretically, those values would be too high for the limitations stated above, but thanks to losses and trace resistance it results in voltages just under the limits.

Frequency Responses

Figure 3.7: Comparison of the Theoretical and actual filter's frequency responses using an analog signal source.



Frequency Responses

Table 2: Custom PCB data while measuring input directly from an analog Voltage Source.

| f(Hz) | Vpp _{out} (V) | Vpp _{in} (V) | Vmax _{out} (V) | Vmax _{in} (V) | Vmin _{out} (mV) | Vmin _{in} (V) | db(V _{out} /V _{in}) |
|-------|------------------------|-----------------------|-------------------------|------------------------|--------------------------|------------------------|--|
| 0.1 | 5.192 | 0.3101 | 5.101 | 0.2675 | -0.09003 | -0.04257 | 24.47665835 |
| 0.5 | 5.192 | 0.3101 | 5.101 | 0.2675 | -0.09003 | -0.04257 | 24.47665835 |
| 2.5 | 5.192 | 0.3101 | 5.101 | 0.2675 | -0.09003 | -0.04257 | 24.47665835 |
| 1 | 5.192 | 0.3101 | 5.101 | 0.2675 | -0.09003 | -0.04257 | 24.47665835 |
| 5 | 5.012 | 0.3101 | 5.012 | 0.2675 | -0.6104 | -0.04257 | 24.17018592 |
| 7.5 | 4.821 | 0.3101 | 4.91 | 0.2675 | 0.08911 | -0.04257 | 23.8327073 |
| 10 | 4.642 | 0.3101 | 4.82 | 0.2675 | 0.1784 | -0.04257 | 23.5040674 |
| 15 | 4.19 | 0.3101 | 4.592 | 0.2675 | 0.4015 | -0.04257 | 22.61424513 |
| 20 | 3.571 | 0.3101 | 4.285 | 0.2675 | 0.714 | -0.04257 | 21.22576168 |
| 25 | 2.946 | 0.3101 | 3.972 | 0.2675 | 1.026 | -0.04257 | 19.55461953 |
| 30 | 2.276 | 0.3101 | 3.615 | 0.2675 | 1.339 | -0.04257 | 17.31340983 |
| 35 | 1.652 | 0.3101 | 3.303 | 0.2675 | 1.651 | -0.04257 | 14.53016553 |
| 40 | 1.071 | 0.3101 | 3.035 | 0.2675 | 1.964 | -0.04257 | 10.76575409 |
| 45 | 0.5356 | 0.3101 | 2.767 | 0.2675 | 2.232 | -0.04257 | 4.746776042 |
| 47.5 | 0.2678 | 0.3101 | 2.633 | 0.2675 | 2.365 | -0.04257 | -1.273823871 |
| 48 | 0.2231 | 0.3101 | 2.633 | 0.2675 | 2.365 | -0.04257 | -2.860043919 |
| 48.5 | 0.1785 | 0.3101 | 2.589 | 0.2675 | 2.41 | -0.04257 | -4.797270916 |
| 49 | 0.1785 | 0.3101 | 2.589 | 0.2675 | 2.41 | -0.04257 | -4.797270916 |
| 49.5 | 0.1785 | 0.3101 | 2.589 | 0.2675 | 2.41 | -0.04257 | -4.797270916 |
| 50 | 0.007 | 0.3101 | 2.513 | 0.2675 | 2.443 | -0.04257 | -32.92807452 |
| 50.5 | 0.1793 | 0.3101 | 2.591 | 0.2675 | 2.412 | -0.04257 | -4.758429534 |
| 51 | 0.1793 | 0.3101 | 2.591 | 0.2675 | 2.412 | -0.04257 | -4.758429534 |
| 51.5 | 0.3571 | 0.3101 | 2.678 | 0.2675 | 2.321 | -0.04257 | 1.225761679 |
| 52 | 0.4018 | 0.3101 | 2.723 | 0.2675 | 2.321 | -0.04257 | 2.250163323 |
| 52.5 | 0.4463 | 0.3101 | 2.723 | 0.2675 | 2.276 | -0.04257 | 3.162502412 |
| 55 | 0.6694 | 0.3101 | 2.856 | 0.2675 | 2.187 | -0.04257 | 6.683678836 |
| 60 | 1.071 | 0.3101 | 3.035 | 0.2675 | 1.964 | -0.04257 | 10.76575409 |
| 70 | 1.741 | 0.3101 | 3.347 | 0.2675 | 1.607 | -0.04257 | 14.9859401 |
| 80 | 2.276 | 0.3101 | 3.615 | 0.2675 | 1.339 | -0.04257 | 17.31340983 |
| 90 | 2.678 | 0.3101 | 3.838 | 0.2675 | 1.16 | -0.04257 | 18.72617613 |
| 100 | 3.035 | 0.3101 | 4.017 | 0.2675 | 0.9818 | -0.04257 | 19.81313858 |
| 110 | 3.303 | 0.3101 | 4.151 | 0.2675 | 0.8478 | -0.04257 | 20.54813615 |
| 120 | 3.526 | 0.3101 | 4.24 | 0.2675 | 0.714 | -0.04257 | 21.11561083 |
| 130 | 3.705 | 0.3101 | 4.329 | 0.2675 | 0.6247 | -0.04257 | 21.54572892 |
| 150 | 3.928 | 0.3101 | 4.463 | 0.2675 | 0.5354 | -0.04257 | 22.05339426 |
| 170 | 4.151 | 0.3101 | 4.553 | 0.2675 | 0.4015 | -0.04257 | 22.53301934 |
| 200 | 4.374 | 0.3101 | 4.687 | 0.2675 | 0.3122 | -0.04257 | 22.98754025 |
| 250 | 4.508 | 0.3101 | 4.731 | 0.2675 | 0.2229 | -0.04257 | 23.24964282 |
| 300 | 4.642 | 0.3101 | 4.82 | 0.2675 | 0.1784 | -0.04257 | 23.5040674 |
| 400 | 4.731 | 0.3101 | 4.865 | 0.2675 | 0.1337 | -0.04257 | 23.66902364 |
| 700 | 4.731 | 0.3101 | 4.865 | 0.2675 | 0.1337 | -0.04257 | 23.66902364 |
| 1000 | 4.776 | 0.3101 | 4.865 | 0.2675 | 0.08911 | -0.04257 | 23.75125104 |

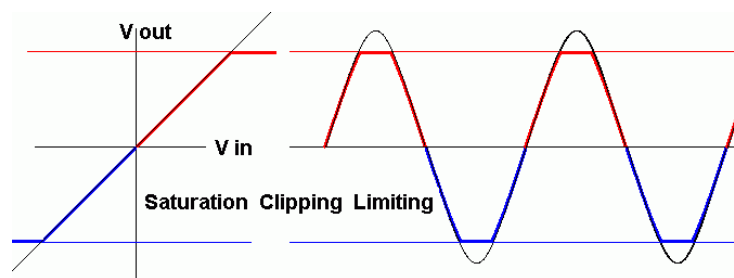
Using a Digital Signal Source

Now, all that's left for testing the PCB is measuring the Notch filter when the input signal is coming from a microcontroller. The board used is named ATmega1284P and the firmware used is analyzed in A.6.

Measuring was done using the same method as before. To measure the response ($20 \log_{10} \left(\frac{V_{ppOUT}}{V_{ppIN}} \right)$), direct probing on the PCB traces had to be done. More specifically, V_{ppIN} was measured from the signal directly before the filter, while V_{ppOUT} , naturally, is the output of the PCB (which is also the output of the notch).

But in Table 3⁶⁵, where the measuring data are displayed, there are some extra values. In particular: $V_{pp_{after\ volt\ divider}}$ and $V_{pp_{pre\ volt\ divider}}$. They refer to a voltage divider and were taken to measure its effectiveness. But there's only one voltage divider mentioned so far, and it is in the notch filter itself, so it doesn't make sense to measure there for this purpose. The schematics provided so far are the ones used to make the PCB, but while measuring using a digital signal source, it was discovered that the converted signal's (coming from the DAC subcircuit) amplitude was really high. In fact, it was too great for the preamplifier that follows, resulting in the saturation of the op amp and clipping of the signal⁶⁶.

Figure 3.8: Op amp saturation and signal clipping.



A possible solution is changing the resistance R14 at the op amp on the DAC subcircuit⁶⁷ so it wouldn't amplify the DAC IC's output as much as it does⁶⁸, but changing that resistor's value resulted to loss of the signal resolution⁶⁹.

The actual solution can be attributed to the high impedance of the op amp inputs. Since the signal is routed directly to the preamplifier (after the DAC subcircuit that is), the current on that trace would be low enough for a voltage divider to work. So, a voltage divider was implemented before the SPDT switch, resulting in approximately $V_{OUT} = 0.0383995 \cdot V_{IN}$. This change did not require a redesign and re-printing of the entire PCB, saving both time and resources. It is visible in Figure 2.12.

⁶⁵ The 30.67Hz and 145.9Hz are the minimum and maximum frequencies the firmware would allow.

⁶⁶ The supply voltage to the preamplifier's op amp is $\pm 10V$ and the gain is $\times 20$. Meaning that its input signal's amplitude must not exceed $\frac{10}{20}V$

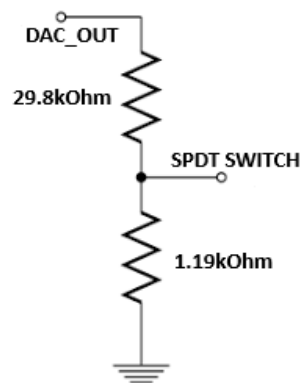
⁶⁷ R4 in Figure 2.3 or R14 in Appendix A.1.

⁶⁸ Even though it's configured as a Current-to-Voltage converter, as stated before, it still amplifies its input: $V_{OUT} = -I_{IN}R$.

⁶⁹ Discovered when that solution was attempted.

Frequency Responses

Figure 3.9: Voltage Divider Values.



Back to the notch filter, the results are displayed in Table 3. The actual notch is clearly defined, though not as prominent as the previous results. On Figure 3.11, the comparison of theoretical and experimental data is displayed.

Table 3: Measurement Data from the Notch while using ATmega as a signal source.

| f(Hz) | Vpp _{out} (V) | Vpp _{pre_Volt_div} (V) | Vpp _{after_Volt_div} (V) | Vpp _{before_filter} (V) | db(Vout/Vin_PCB) | db(Vout/Vin_FILTER) |
|-------|------------------------|---------------------------------|-----------------------------------|----------------------------------|------------------|---------------------|
| 30.67 | 2.835 | 8.056 | 0.2705 | 6.266 | 20.40771588 | -6.888746546 |
| 41 | 1.418 | 8.056 | 0.2686 | 6.266 | 14.45140445 | -12.90628319 |
| 43.42 | 1.063 | 8.056 | 0.2686 | 6.266 | 11.94854512 | -15.40914252 |
| 44.76 | 0.886 | 8.056 | 0.2686 | 6.266 | 10.36655427 | -16.99113337 |
| 47.68 | 0.7086 | 8.145 | 0.3583 | 6.087 | 5.923086797 | -18.68004308 |
| 48.73 | 0.4431 | 8.056 | 0.2686 | 6.266 | 4.347914834 | -23.00977281 |
| 50.3 | 0.5316 | 7.967 | 0.3577 | 6.176 | 3.441320642 | -21.3024463 |
| 54.16 | 0.866 | 8.145 | 0.3577 | 6.266 | 7.679979037 | -17.18944997 |
| 59.29 | 1.417 | 8.145 | 0.3577 | 6.266 | 11.9570182 | -12.91241081 |
| 60.94 | 1.506 | 8.145 | 0.3577 | 6.266 | 12.48612063 | -12.38330837 |
| 64.62 | 1.86 | 8.145 | 0.358 | 6.266 | 14.31259835 | -10.54954893 |
| 67.5 | 2.126 | 8.145 | 0.3577 | 6.266 | 15.4808864 | -9.388542607 |
| 69.6 | 2.215 | 8.145 | 0.358 | 6.266 | 15.82981408 | -9.0323332 |
| 71.1 | 2.481 | 8.145 | 0.358 | 6.265 | 16.81487475 | -8.045886221 |
| 75 | 2.747 | 8.145 | 0.3577 | 6.265 | 17.70679439 | -7.161248318 |
| 77.9 | 2.835 | 8.145 | 0.358 | 6.266 | 17.97340073 | -6.888746546 |
| 82.9 | 3.189 | 8.145 | 0.358 | 6.265 | 18.99542985 | -5.865331122 |
| 90.13 | 3.544 | 8.145 | 0.3577 | 6.265 | 19.91949546 | -4.948547242 |
| 97.45 | 3.898 | 8.145 | 0.3577 | 6.265 | 20.74645789 | -4.121584811 |
| 102.6 | 4.075 | 8.145 | 0.358 | 6.265 | 21.12489173 | -3.735869245 |
| 112.6 | 4.341 | 8.145 | 0.3577 | 6.265 | 21.68141691 | -3.18662579 |
| 118.7 | 4.518 | 8.145 | 0.3577 | 6.265 | 22.02854573 | -2.839496975 |
| 124.4 | 4.696 | 8.145 | 0.3577 | 6.265 | 22.36418296 | -2.503859742 |
| 127.2 | 4.786 | 8.145 | 0.3577 | 6.265 | 22.52907508 | -2.338967621 |
| 134 | 4.873 | 8.145 | 0.3577 | 6.265 | 22.68554942 | -2.182493279 |
| 140.8 | 5.05 | 8.145 | 0.3577 | 6.265 | 22.99544876 | -1.872593944 |
| 145.9 | 5.05 | 8.145 | 0.3577 | 6.265 | 22.99544876 | -1.872593944 |

Frequency Responses

Figure 3.10: Notch filter's frequency response when using the DAC – a digital signal source as input.

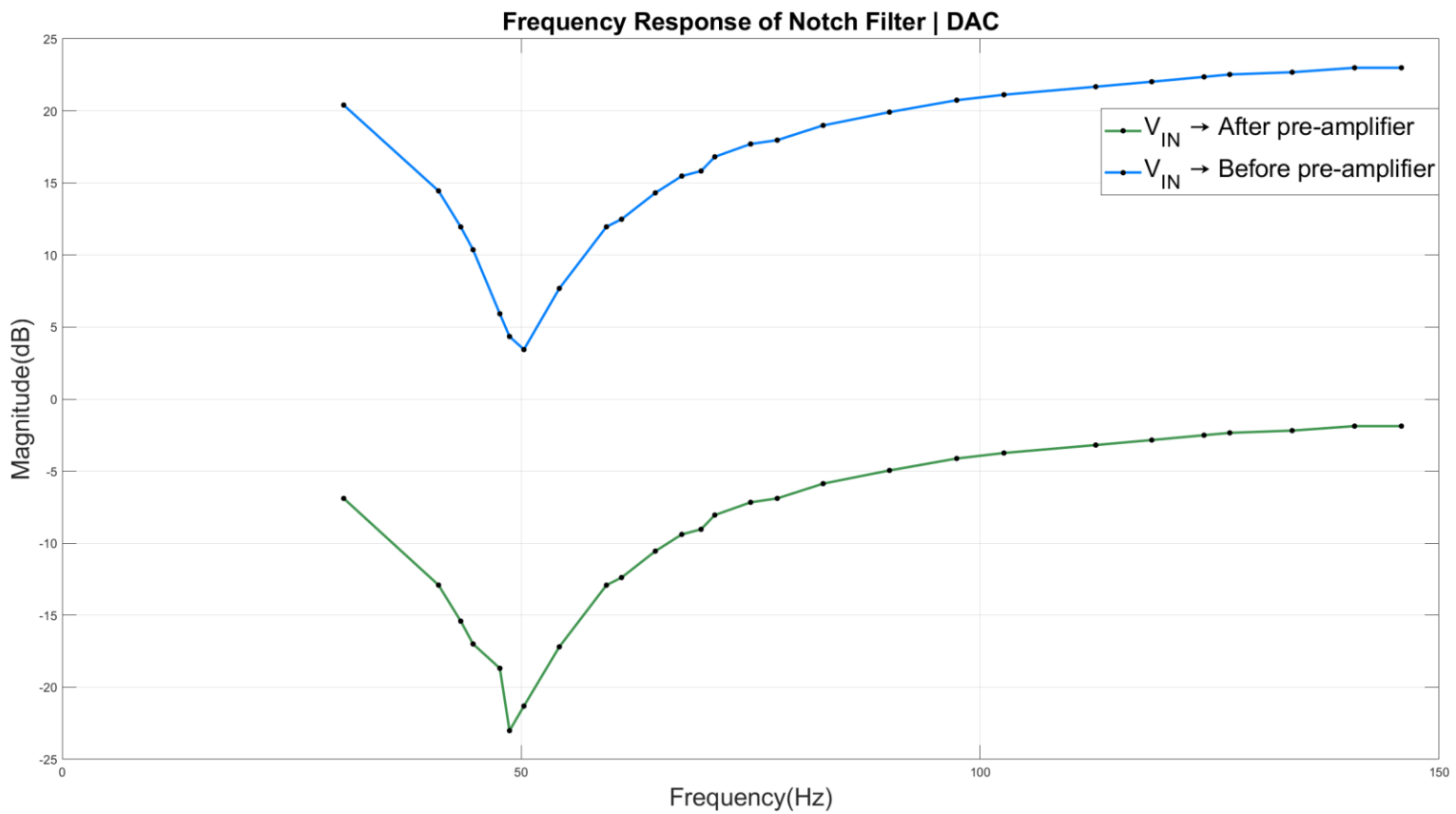
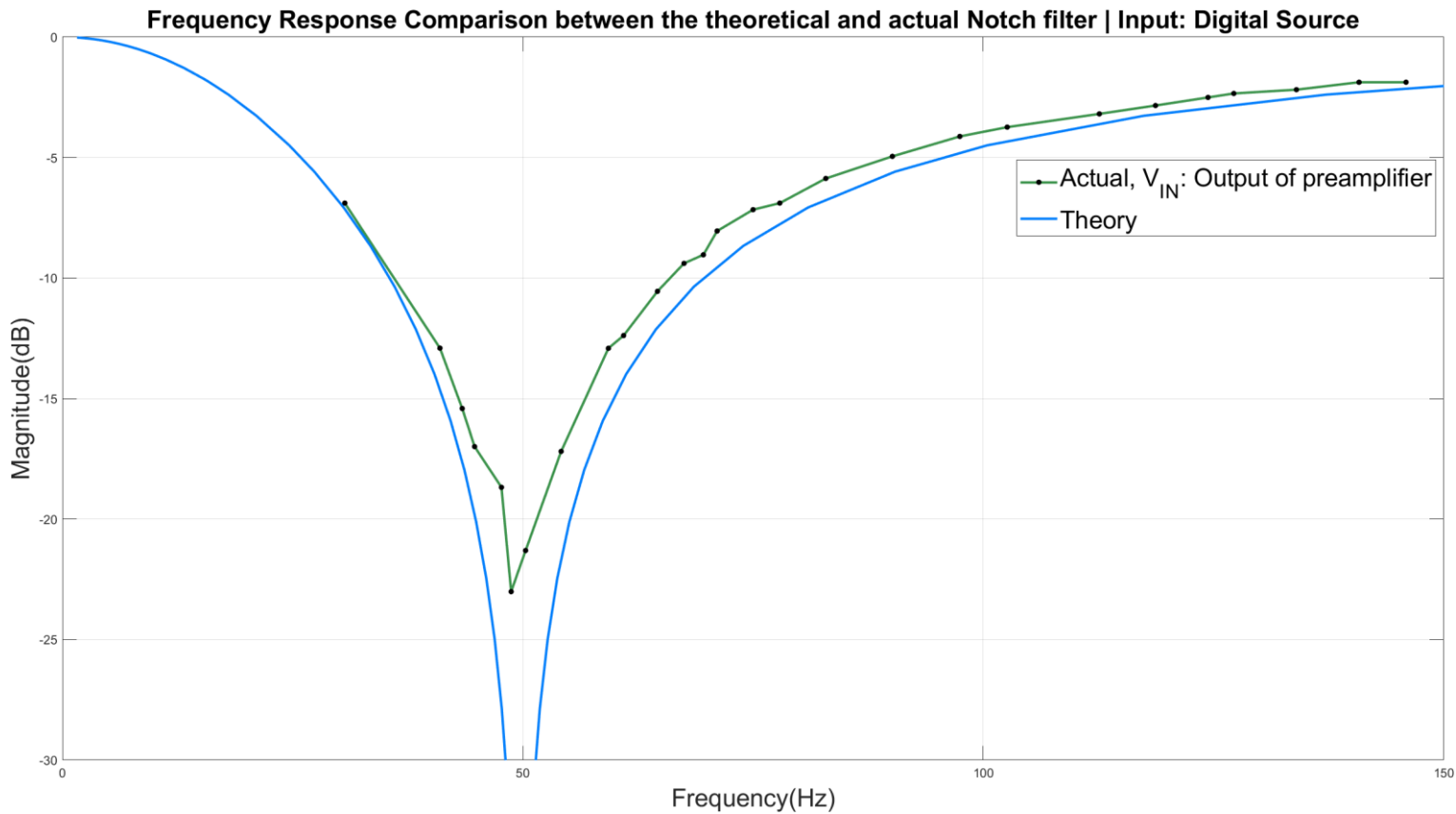


Figure 3.11: Comparison of theoretical and actual data, when the input is from a digital signal source.



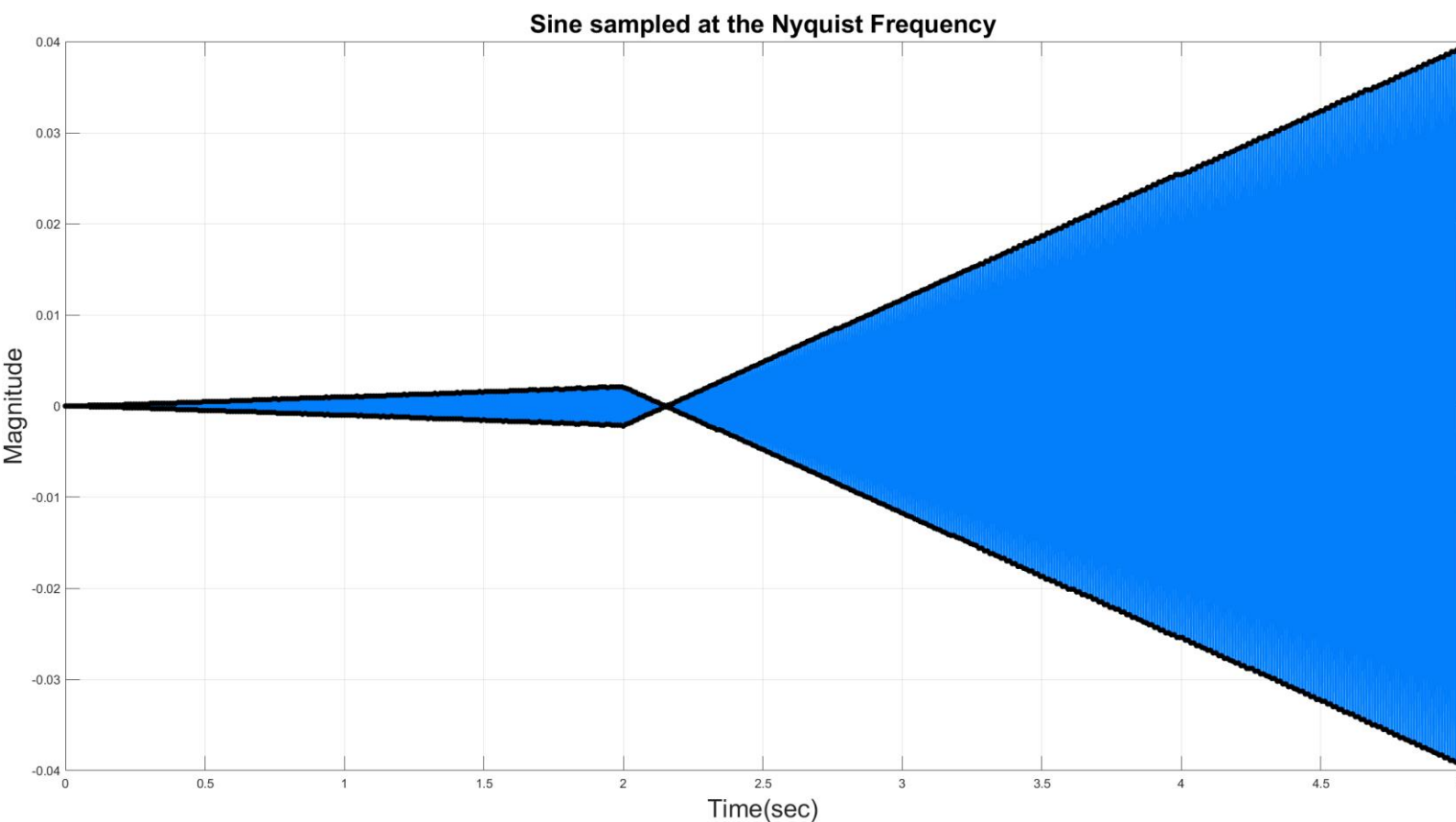
3.2.2 Butterworth Digital Filter's Response

Moving on to testing the Digital Filter, the measuring process went as follows: First, an Arduino Uno was programmed to output a sine signal at a specific frequency. Since it's a digital signal, it is discrete, with a specific time interval. This interval essentially is the sampling frequency of the signal, and it was chosen to be much greater than the signal's innate frequency and by extension than the Nyquist frequency, valued at $F_s = 20kHz$ ⁷⁰. The reason being not to lose any information that is in the input signal.

Technically, according to the Nyquist theorem, a sampling frequency at twice the Nyquist frequency should suffice for an accurate reconstruction of the signal. Indeed, when measured at close to that value⁷¹, the Fourier Transform would yield the correct results, but the sampled signal would not resemble a sine. That would lead to a very peculiar looking signal after undergoing filtering at the Butterworth filter, making it difficult to accurately measure the peak-to-peak values in order to evaluate the digital filter.

Figure 3.12: Sine sampled at Nyquist Frequency.

Note: The amplitude of the input signal is 1, while the sampled is around 0,04. That is because the samples align at the zeros of the sine. The reason because they are not exactly zero is the accuracy of the calculations.



⁷⁰ Additionally the number of samples was 10000 for each frequency.

⁷¹ If a sine is sampled at exactly the Nyquist frequency, it so happens that the samples align with the zeros of the signal, making reconstruction impossible.

Frequency Responses

Figure 3.13: A zoom in of the above graph.

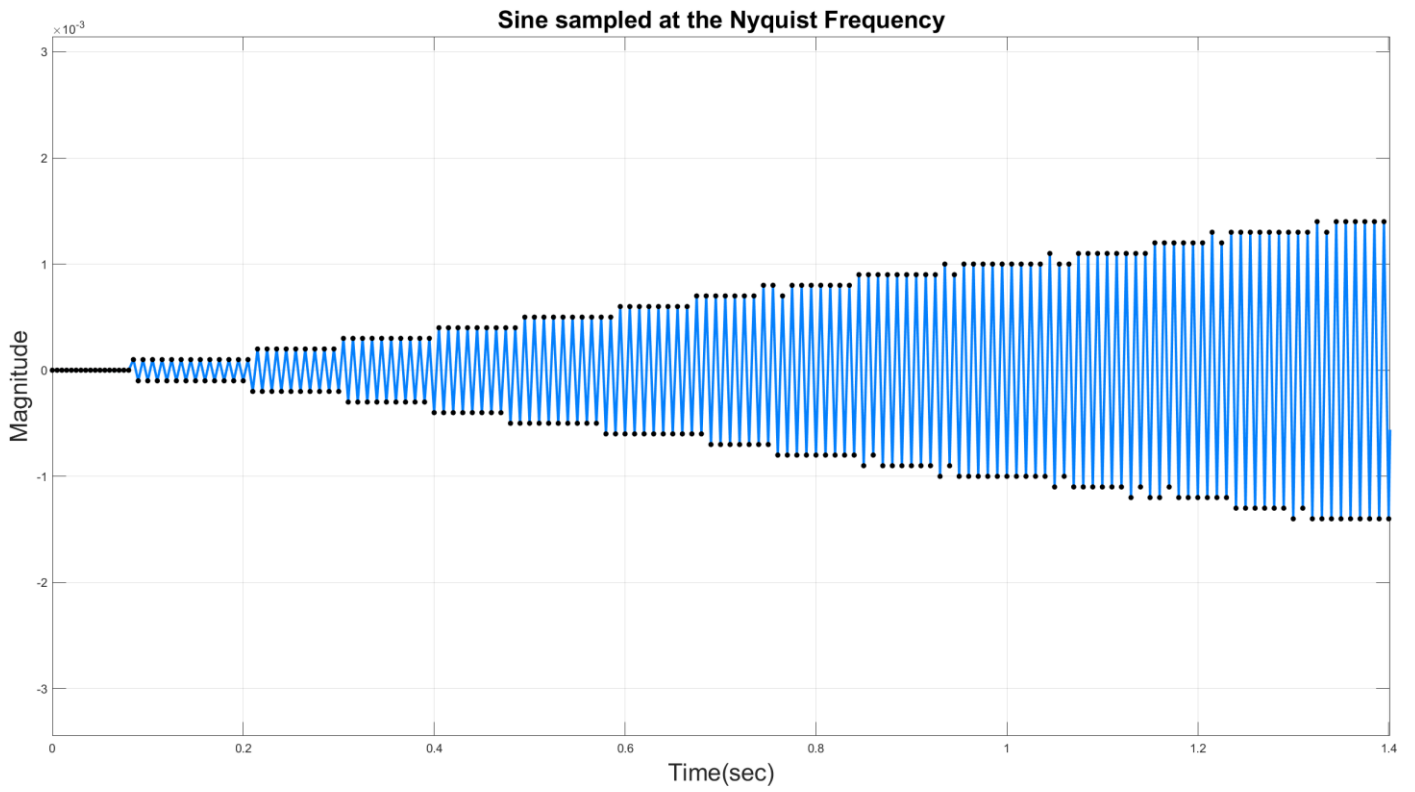
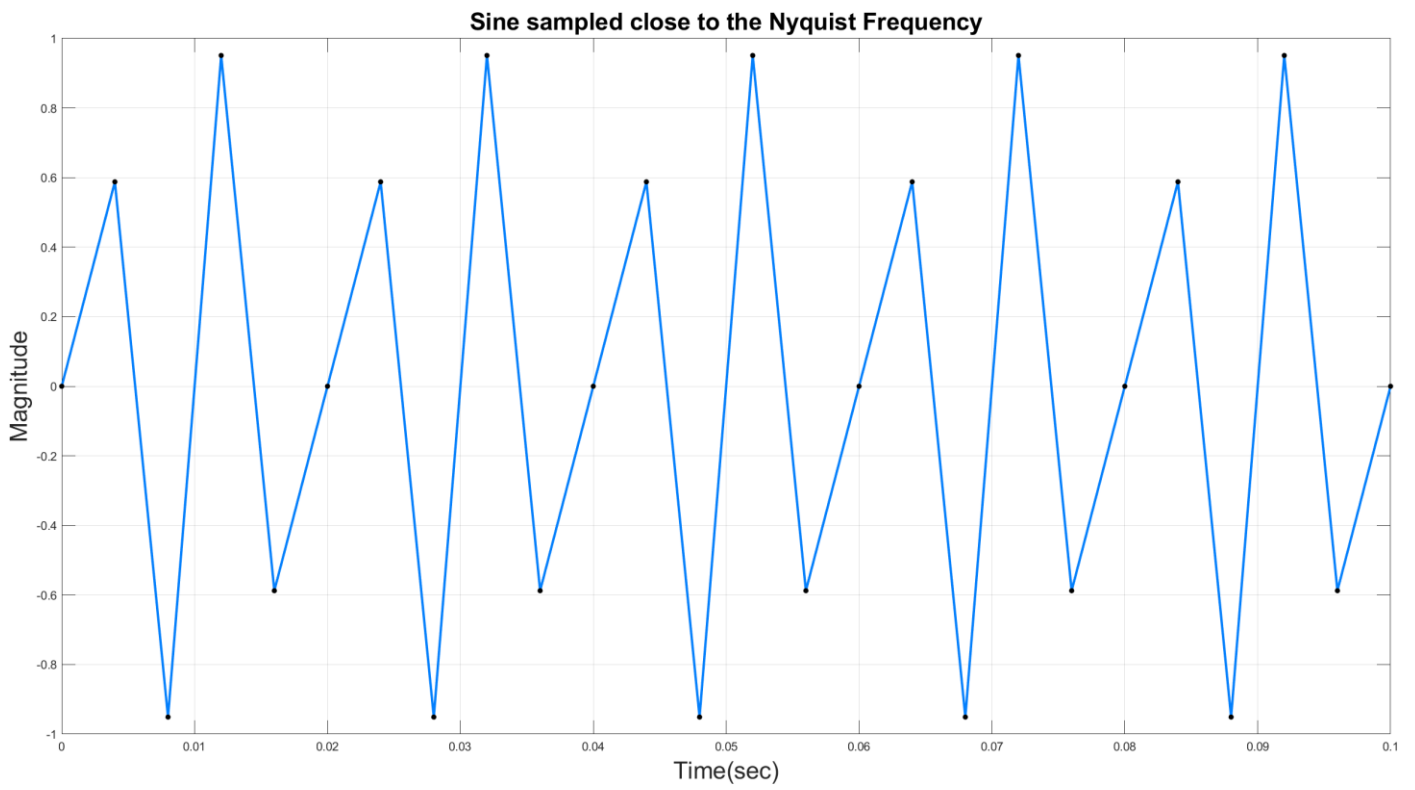


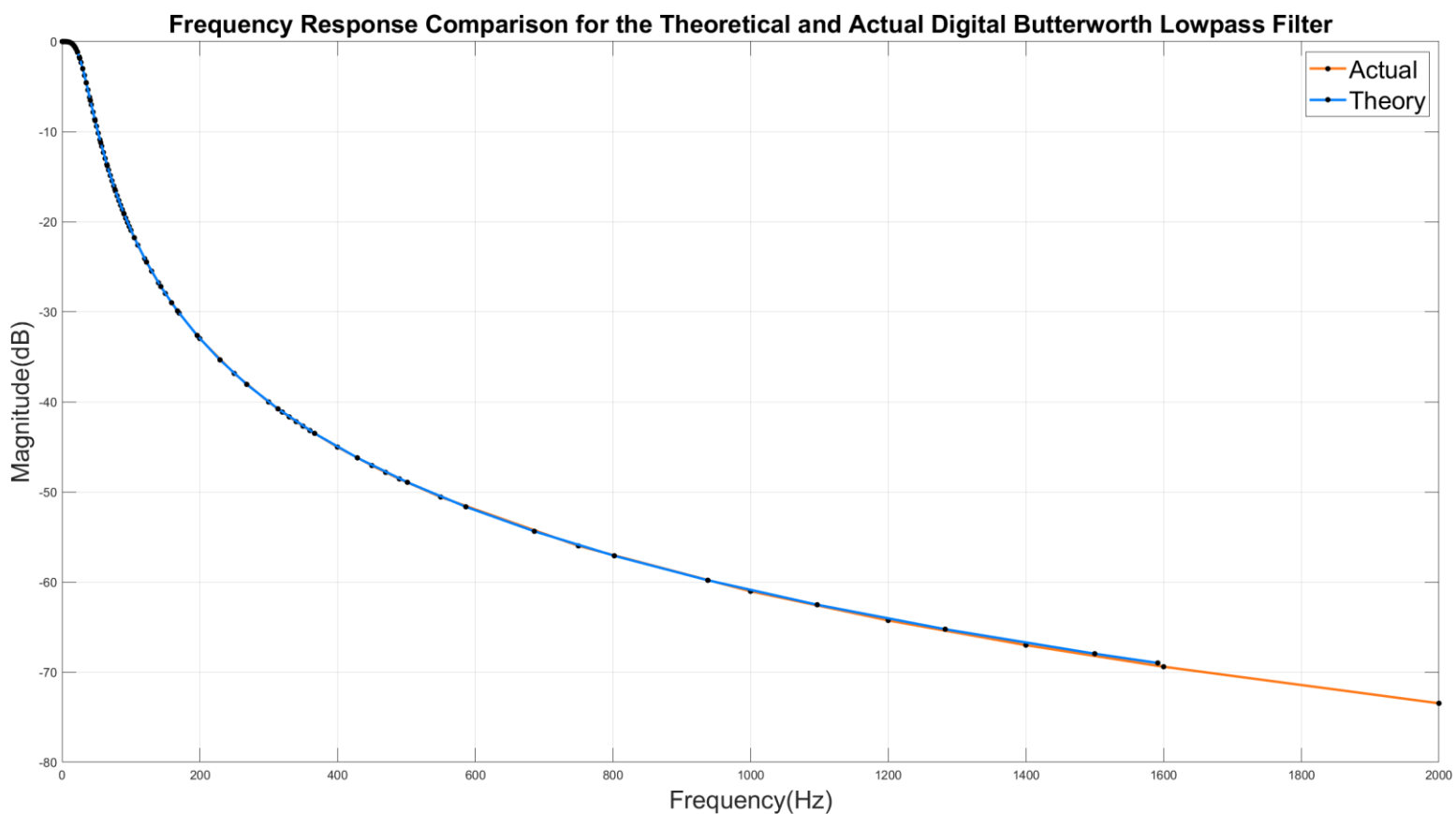
Figure 3.14: Sine sampled close to the Nyquist frequency.



Frequency Responses

Next step was to connect the custom MATLAB app to the same port and receive the data. Then, after filtering, all the data were exported to excel files using the app's built-in feature. Finally, from there, the peak-to-peak values for both the input and the filtered signal were measured, and then the magnitude response was plotted (Figure 3.15). To measure the peak-to-peak value for the filtered signal, the transient response had to be taken into account. The values were calculated using the last half of the samples, where the steady state was undoubtedly reached. The testing was indicatively done only for the 2nd order and for $F_C = 30\text{Hz}$ to match the ELF band. The data used for the plot are presented on Table 4. Finally, the theoretical value was calculated using the filter's TF for these values, and plotted using the built-in MATLAB function "bode()".

Figure 3.15: Magnitude Response Comparison.



Frequency Responses

Table 4: Data taken while testing the digital filter.

| f (Hz) | A(dB) |
|--------|--------------|
| 2.5 | -0.000207252 |
| 5 | -0.003335147 |
| 7.5 | -0.016860944 |
| 10 | -0.053322686 |
| 12.5 | -0.129053701 |
| 15 | -0.263409222 |
| 17.5 | -0.475870621 |
| 20 | -0.783351116 |
| 22.5 | -1.194479981 |
| 25 | -1.709615592 |
| 27.5 | -2.320115799 |
| 30 | -3.010355066 |
| 32.5 | -3.762229994 |
| 35 | -4.553442218 |
| 37.5 | -5.367914145 |
| 40 | -6.191867484 |
| 42.5 | -7.015947735 |
| 45 | -7.828085636 |
| 47.5 | -8.62515304 |
| 50 | -9.403909244 |
| 52.5 | -10.16441557 |
| 55 | -10.90014653 |
| 57.5 | -11.61362178 |
| 60 | -12.3055007 |
| 62.5 | -12.97597113 |
| 65 | -13.62711583 |
| 67.5 | -14.25518869 |
| 70 | -14.86430029 |
| 72.5 | -15.45520629 |
| 75 | -16.03068003 |
| 77.5 | -16.58589605 |
| 80 | -17.12519459 |

| | |
|------|--------------|
| 82.5 | -17.64980009 |
| 85 | -18.16226343 |
| 87.5 | -18.65749536 |
| 90 | -19.13995458 |
| 92.5 | -19.61010963 |
| 95 | -20.07104942 |
| 97.5 | -20.51715924 |
| 100 | -20.95244856 |
| 110 | -22.59735421 |
| 120 | -24.10229064 |
| 130 | -25.48852818 |
| 140 | -26.7724855 |
| 150 | -27.96899165 |
| 170 | -30.14169209 |
| 200 | -32.96319396 |
| 250 | -36.84045766 |
| 300 | -40.01038702 |
| 320 | -41.13302616 |
| 330 | -41.66911309 |
| 340 | -42.18983736 |
| 350 | -42.69277732 |
| 360 | -43.18305929 |
| 400 | -45.01564768 |
| 450 | -47.06644367 |
| 470 | -47.823312 |
| 490 | -48.55291692 |
| 550 | -50.56470225 |
| 750 | -55.98569231 |
| 1000 | -61.02615965 |
| 1200 | -64.26065987 |
| 1400 | -67.00538998 |
| 1600 | -69.40381382 |
| 2000 | -73.45988464 |

3.3 Conclusions

Within the scope of this postgraduate dissertation, a PCB was designed and printed, capable of amplifying both digital and analog signals as well as filtering out their power supply noise. Interfacing directly with an Arduino Uno board, the analog signal, is sampled, converted to digital and sent to a computer for further processing. There, an app was developed using MATLAB, able to communicate with the Arduino via the USB port and receive the sampled signal. Then it can be analyzed to its frequencies using FFT, and be filtered by a digital Lowpass Butterworth filter, attenuating higher frequencies, in order to match the ELF band. All the while, the order and cutoff frequency are not hardcoded but chosen by the user. Finally, the data can be saved in 3 different ways (.mat, .xlsx ,.txt) for future analysis.

Looking at the results above, the implementation of the system was generally successful, as it can carry out all its assigned tasks mentioned above.

The digital filter is almost an exact match of the theoretical one as shown in their comparison graph. The FFT provides the correct results, as long as the right values are selected by the user, who needs to match the sampling frequencies on the app and the Arduino, while simultaneously respecting the Nyquist sampling theorem.

The notch filter when using a digital signal source, was seemingly the least successful implementation. While a notch is clearly visible, the dB drop is not at the ideal level, sitting at only $-23.01dB$ maximum attenuation at $48.73Hz$ ⁷².

However, the notch filtering results when an analog signal source was used, imply that the filter itself is well designed and assembled (as theoretical and measured data align almost perfectly), and that there is a flaw elsewhere.

A possible cause for this outcome is that noise was detected from the 5V pin, as it close to the digital data pins. Even if it's not the sole cause, it certainly is a change that needs to be made in future revisions of the PCB.

3.4 Future Improvements

There are several basic improvements that can be done, mainly on the PCB side of the system. First and foremost, the power supply pins could be replaced for example by a DC power jack (common barrel-type power connector). It could supply a higher Voltage and then using Voltage regulators, such as buck or step-down converters, that Voltage could be reduced to supply each IC with the voltage level they need. This would minimize the number of power supply connectors to just one, making the interfacing with the board easier (less cluttered). It also gives the side bonus of making the board more durable, as those pins can be broken fairly easily, while a barrel connector is more robust. Doing this, would also remove the need for a 5V, which, as already mentioned, was producing measurable noise during the DAC testing.

⁷² A measurement at exactly $50Hz$ could not be taken due to limitations of the hardware while measuring.

Future Improvements

Additionally, a low-pass filter could be added immediately after the DAC IC to filter out the “steps” of the converted signal and possible noise created by IC itself⁷³.

On the software side, some more features could be added. For instance, using the data that are already available (after receiving from the Arduino), there could be a window plotting the magnitude and the phase response of the Butterworth filter. Another one is more types of filters. For now, only the Butterworth Lowpass is available, but an option like a Chebyshev could be added. Or even different filters in terms of frequency response, like bandpass, stopband, etc. Although this would go beyond the goal set for this system; Attenuating signals that are out of the ELF band.

Lastly, the user is required to fill in the offset value of the input signal. This can change on future iterations of the app, because this value can actually be calculated programmatically using the well-known formula [21, p. 63]:

$$x_{dc} = \frac{1}{T} \int_0^T x(t) dt$$

T is the frequency of the signal and its value is already known by the app thanks to the Fourier analysis.

⁷³ These filters are known as *reconstructive filters*.

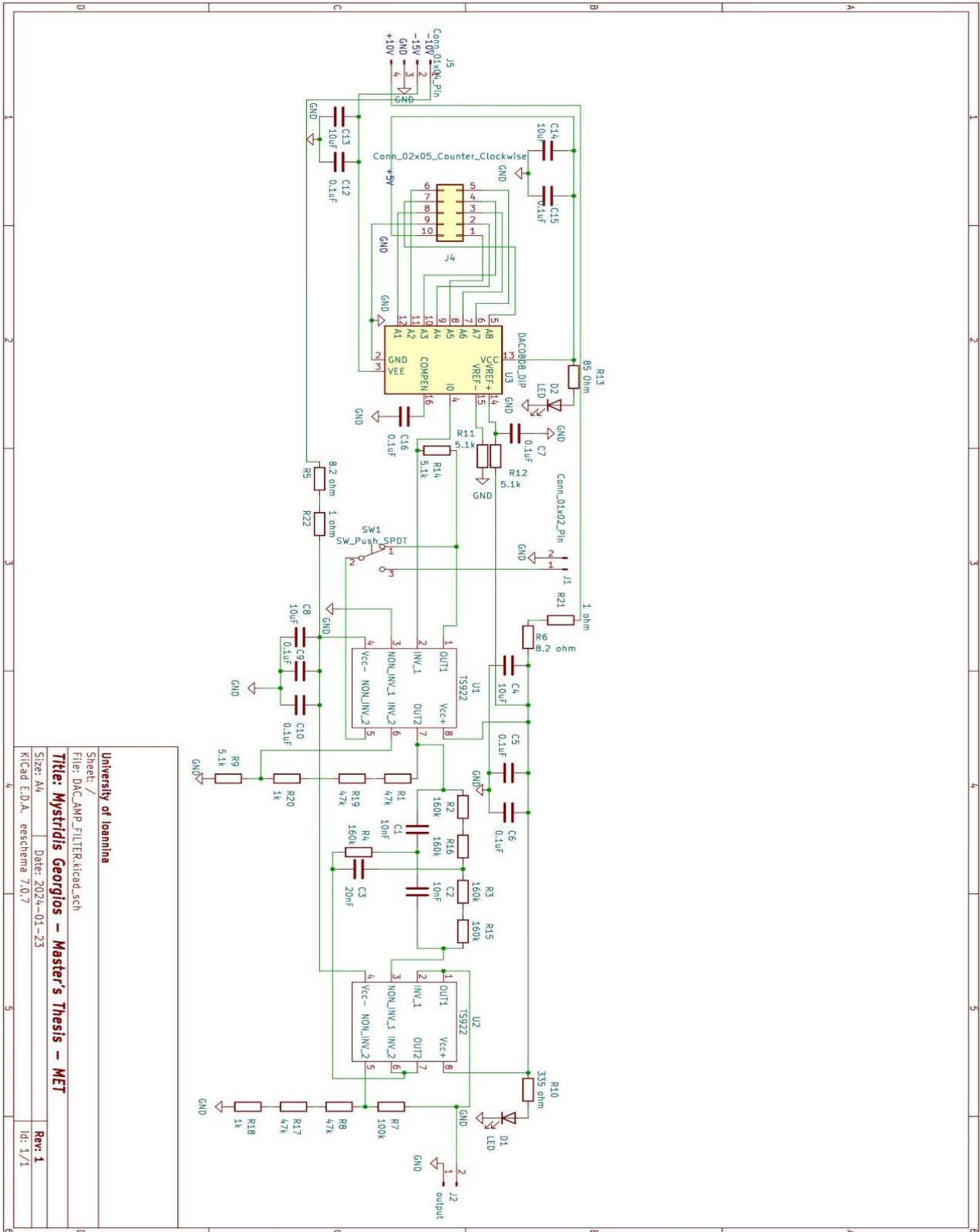
Bibliography

- [1] P. Horowitz, *The art of electronics*, Third edition. New York, NY: Cambridge University Press, 2015.
- [2] L. Thede, *Practical analog and digital filter design*. in Artech House microwave library. Boston, Mass. London: Artech House, 2005.
- [3] A. R. Hambley, *Electrical engineering: principles and applications*, Seventh edition, Global edition. NY, NY: Pearson, 2019.
- [4] H. Zumbahlen, *Basic linear design*. Norwood, Mass.: Analog Devices, 2007.
- [5] N. S. Nise, *Control systems engineering*. Place of publication not identified: John Wiley, 2014.
- [6] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals & systems*, 2nd ed. in Prentice-Hall signal processing series. Upper Saddle River, N.J: Prentice Hall, 1997.
- [7] "dac0808.pdf." Accessed: Jan. 24, 2024. [Online]. Available: https://www.ti.com/lit/ds/symlink/dac0808.pdf?ts=1706036605362&ref_url=https%253A%252F%252Fwww.google.com%252F
- [8] A. S. Sedra and K. C. Smith, *Microelectronic circuits*, Seventh edition. in The Oxford series in electrical and computer engineering. New York ; Oxford: Oxford University Press, 2015.
- [9] "Read Analog Voltage." Accessed: Jan. 26, 2024. [Online]. Available: <https://www.arduino.cc/en/Tutorial/BuiltInExamples/ReadAnalogVoltage>
- [10] "How to read COM port caption." Accessed: Jan. 28, 2024. [Online]. Available: https://www.mathworks.com/matlabcentral/answers/447648-how-to-read-com-port-caption#comment_1970290
- [11] S. Winder, *Analog and digital filter design*, 2. ed. Amsterdam: Newnes, 2002.
- [12] L. Weinberg, *Network analysis and synthesis*, Repr. with corr. Huntington: Krieger, 1975.
- [13] S. W. Smith, *The scientist and engineer's guide to digital signal processing*, 1. ed. San Diego, Calif: California Technical Publ, 1997.
- [14] J. H. McClellan, R. W. Schafer, and M. A. Yoder, *DSP First*, Second edition, Global edition. in Always Learning. Harlow: Pearson, 2017.
- [15] "ch_DFT.pdf." Accessed: Feb. 06, 2024. [Online]. Available: https://web.mit.edu/~gari/teaching/6.555/lectures/ch_DFT.pdf
- [16] C. L. Phillips, J. M. Parr, and E. A. Riskin, *Signals, systems, and transforms*, 4. ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2008.

- [17] “Fast Fourier transform - MATLAB fft.” Accessed: Feb. 12, 2024. [Online]. Available: <https://www.mathworks.com/help/matlab/ref/fft.html#d126e476156>
- [18] “Handle Classes - MATLAB & Simulink.” Accessed: Feb. 12, 2024. [Online]. Available: <https://www.mathworks.com/help/matlab/handle-classes.html>
- [19] “Comparison of Handle and Value Classes - MATLAB & Simulink.” Accessed: Feb. 12, 2024. [Online]. Available: https://www.mathworks.com/help/matlab/matlab_oop/comparing-handle-and-value-classes.html
- [20] “Superclass of all handle classes - MATLAB.” Accessed: Feb. 12, 2024. [Online]. Available: <https://www.mathworks.com/help/matlab/ref/handle-class.html>
- [21] A. IZADIAN, *FUNDAMENTALS OF MODERN ELECTRIC CIRCUIT ANALYSIS AND FILTER SYNTHESIS: a transfer function approach*. S.I.: SPRINGER INTERNATIONAL PU, 2024.
- [22] E. Bogatin, *Bogatin’s Practical Guide to Prototype Breadboard and PCB Design*. Boston London: Artech House, 2021.
- [23] E. Bogatin, L. Smith, and S. Sandler, “The Myth of Three Capacitor Values | 2020-03-03 | Signal Integrity Journal.” Accessed: Feb. 05, 2024. [Online]. Available: <https://www.signalintegrityjournal.com/articles/1589-the-myth-of-three-capacitor-values>
- [24] “ATmega164A_PA-324A_PA-644A_PA-1284_P_Data-Sheet-40002070A.pdf.” Accessed: Feb. 05, 2024. [Online]. Available: https://alpha.physics.uoi.gr/foudas_public/Micro2019Uoi/ATmega164A_PA-324A_PA-644A_PA-1284_P_Data-Sheet-40002070A.pdf
- [25] “AVR® Interrupts - Developer Help.” Accessed: Feb. 21, 2024. [Online]. Available: <https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/structure/interrupts/>
- [26] “Atmel.Schematics_1.pdf.” Accessed: Feb. 05, 2024. [Online]. Available: https://alpha.physics.uoi.gr/foudas_public/Micro2019Uoi/Atmel.Schematics_1.pdf
- [27] “AVR364: MEGA-1284P Xplained Hardware User’s Guide”.
- [28] “AVR Instruction Set Manual”.
- [29] “ATmega1284P-Datasheet-Summary.pdf.” Accessed: Feb. 05, 2024. [Online]. Available: https://alpha.physics.uoi.gr/foudas_public/Micro2019Uoi/ATmega1284P-Datasheet-Summary.pdf
- [30] The Mathworks Inc., “MATLAB.” in MATLAB Version: 9.14.0.2206163 (R2023a). The MathWorks Inc., 2023. [Online]. Available: <https://www.mathworks.com>
- [31] “DSP System Toolbox.” Accessed: Jan. 29, 2024. [Online]. Available: <https://www.mathworks.com/products/dsp-system.html>

- [32] “Διακόπτης Συρόμενος Mini SPDT - 1A/250VAC,” grobotronics.com. Accessed: Jan. 24, 2024. [Online]. Available: <https://grobotronics.com/mini-spdt-1a-250vac.html>
- [33] “LTST-C171TBKT.pdf.” Accessed: Jan. 24, 2024. [Online]. Available: <https://grobotronics.com/images/companies/1/datasheets/LTST-C171TBKT.pdf?1521198076155>
- [34] “Rail-to-rail high output current dual operational .pdf.” Accessed: Jan. 24, 2024. [Online]. Available: <https://grobotronics.com/images/companies/1/ts922.pdf>

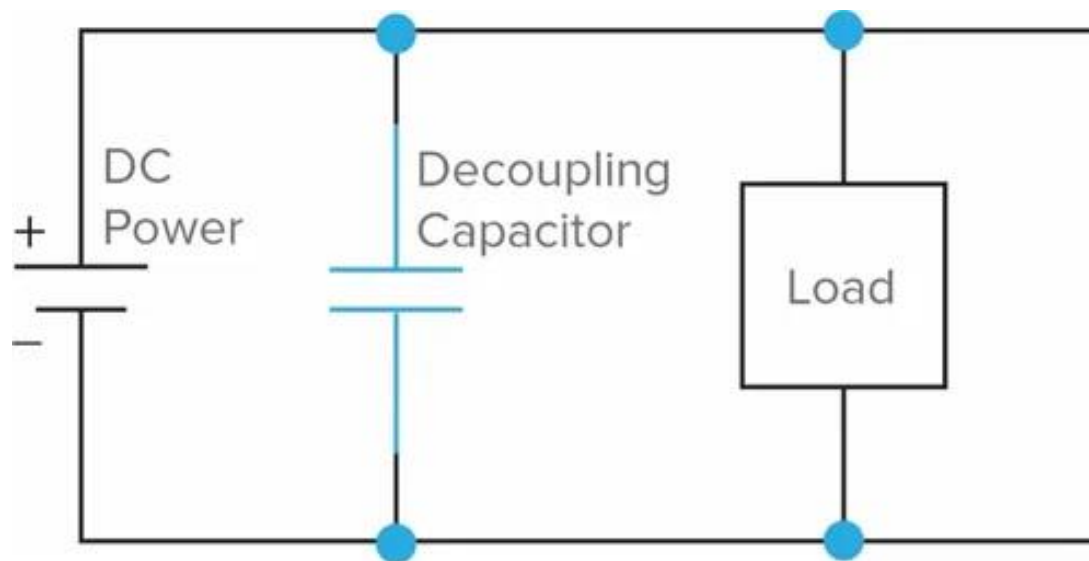
A.1 PCB SCHEMATIC



A.2 Decoupling Capacitors

When designing a circuit many novice engineers take a stable and well-regulated power supply for granted, only to find that their circuits don't perform as expected during testing. Analog circuits, such as audio amplifiers or radios may produce a strange hum or a crackling noise audible in the background, and digital circuits such as microcontrollers may become unstable and unpredictable. If there is a voltage spike, the program loaded in the processor could skip instructions, which is clearly not wanted. The reason for this underperformance often lies in the fact that the input voltage (from the power supply) is rarely stable in practice. However, when viewed through an oscilloscope, a DC power supply shows many glitches, voltage spikes and AC voltage components. For this reason, **Decoupling Capacitors** are added to the circuit to smooth out the power supply voltage.

Appendix Figure 1: Decoupling Capacitor.



Note: Yates, J. (2023, August 31). What Are Decoupling Capacitors? Knowles Precision Devices Blog. <https://blog.knowlescapacitors.com/blog/decoupling-capacitors>

Additionally, in digital circuits the power source may be contaminated from the logic board or other devices. More specifically, logic circuits are made of millions of logic gates, which constantly change their output states between ON and OFF. These logic gates, in turn, consist of transistors, that generate what is called **Transient Load** during the switching. As a result, the current drawn by the devices fluctuates, generating noise, which propagates back to the power source. Therefore, capacitors used for decoupling essentially serve three roles:

- a) Protecting the power source from electrical noise generated within the circuit.
- b) Protecting the circuit from electrical noise generated by other devices connected to the same power source.
- c) Filter out voltage spikes and dips while allowing only the DC component to pass through.

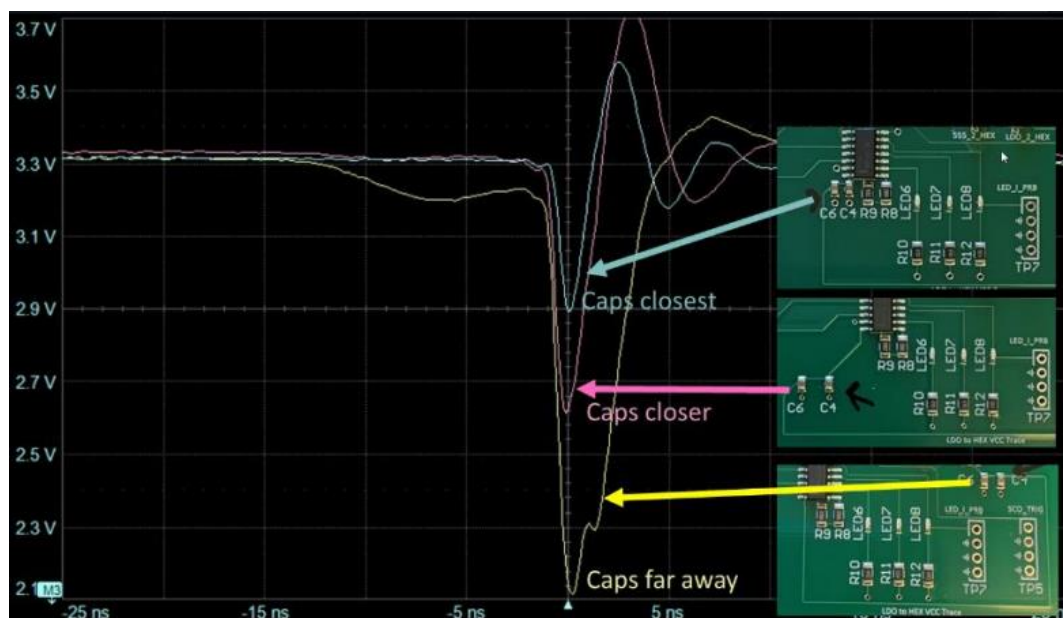
But how do they work? A decoupling capacitor acts as a local electrical energy reservoir. Capacitors, like batteries, need time to charge and discharge. When used as decoupling capacitors, they oppose quick changes of voltage. If the input voltage suddenly drops, the

Appendix-Decoupling Capacitors

capacitor provides the energy to keep the voltage stable. Similarly, if there is a voltage spike, the capacitor “absorbs” the excess energy. So, they essentially bypass the power source when needed, earning themselves their second name **Bypass Capacitors**. It’s not uncommon to have a single capacitor for each integrated circuit used. As a matter of fact, in digital systems almost all capacitors on the board may be used for decoupling.

Now, it’s already established that power signal reaching the ICs is not inherently stable and that it can be contaminated from the fast switching of the states of the transistors that make up the ICs⁷⁴. Another reason for the unstable state is the intrinsic inductance and impedance of the power trail/trace/wire. Specifically, between the VRM and the pads of the IC is the inductance of the VRM⁷⁵. The transient current⁷⁶, flowing through this inductance, generates a voltage drop, otherwise called “switching noise”. This appears on the power rails of the IC. The way to reduce this noise is to reduce the loop inductance from the IC pads to the VRM [22, p. 323].

Appendix Figure 2: The effect of the distance between the decoupling capacitors and the IC on the transient response.



Note: Image taken from "Bogatin's Practical Guide to Prototype Breadboard and PCB Design, p.325"

Since, an intrinsic inductance is present on all traces on the PCB, then also present on the trace between the decoupling capacitor and the IC⁷⁷. Hence, the most important quality of

⁷⁴ Additionally, this switching to its outputs of the ICs, needs power that will ultimately come from the power supply. This switching can be very quick from a few nanoseconds to a few microseconds [22, p. 323]. This requires different power levels from the supply, very quickly, making the power signal unstable.

⁷⁵ VRM stands for Voltage Regulator Module. It’s an IC responsible for the power distribution on the PCB. If a PCB doesn’t have one integrated, does not matter in the context of this chapter, as everything said is true for VRMs or simple Power pins, power jacks, etc.

⁷⁶ The term “transient current” refers to the current during the switching of the output states, while the power source is still “unstable” – before the steady state.

⁷⁷ The inductance of the trace between the power supply and the capacitor is already decoupled by it. But the inductance of the trace between the capacitor and the IC is yet to be addressed.

Appendix-Decoupling Capacitors

the decoupling capacitor is the loop⁷⁸ inductance between the capacitor itself and the device to which it provides the charge. The closer the capacitor is to the device, the lower the loop inductance [22, p. 305]. The longer the path length between the decoupling capacitor and the pads of the IC, the higher the loop inductance and the higher the switching noise [22, p. 323]. Because of that, the preferred placement of decoupling capacitors is as close as possible to the IC they are decoupling.

So, the main idea, how they work and where to place them are now explained. But how is the capacitance of each one chosen? To answer that question accurately⁷⁹, the value of the current draw, the current's transient time and the acceptable voltage change are necessary [22, p. 303]:

$$C = \frac{\Delta Q}{\Delta V} = \frac{\Delta t \cdot I}{\Delta V} \quad 1$$

Where C is the amount of decoupling capacitance needed, ΔQ is the charge depletion of the capacitor to supply the current requirements of the chip during the transient current time, ΔV is the acceptable voltage drop (noise) on the capacitor, which is the voltage droop on the capacitor as the current bleeds off charge, Δt is the time during which the transient current occurs and lastly I the current draw of the die (IC) [22, p. 303]. However, knowing all these values for each IC can be challenging. Fortunately, the precise value of capacitance is not important as long as it is larger than the minimum required to supply the local charge during the switching event [22, p. 304]. Unless someone has a specific requirement for their device, a good starting place is to use at least $10\mu F$ of decoupling capacitance per IC [22, p. 305].

A common recommendation is using three decoupling capacitors for each IC, separated in value by a decade. This is completely wrong. To understand why, one must look at a capacitor as its real equivalent circuit and not as an ideal component. An ideal capacitor's impedance is analogous to $\frac{1}{f}$ forever. But a real one has some self-inductance and an internal equivalent series resistance. This means that a real capacitor behaves like an ideal RLC circuit [23].

Appendix Figure 3: Equivalent circuit model of a real capacitor.



Note: Image taken from "Bogatin's Practical Guide to Prototype Breadboard and PCB Design, p.327"

On the figure above is shown an equivalent circuit model of a real capacitor. The L is the loop self-inductance (Equivalent Self Inductance or ESL) between the capacitor and the pads on the

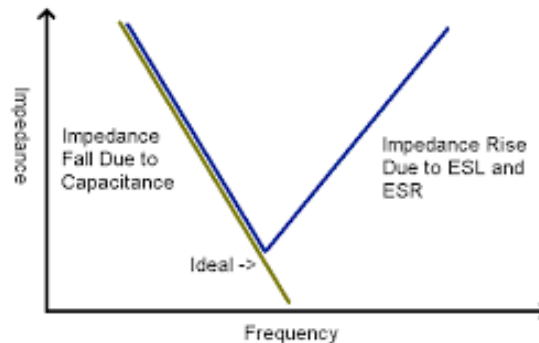
⁷⁸The word "loop" refers to the loop from power to ground trail.

⁷⁹ The capacitor is selected so that during the time in which we expect to see a transient current, the rise or fall time, the voltage drop from the current draw is an acceptably small drop.

Appendix-Decoupling Capacitors

IC, while the C is its intrinsic capacitance, and the R is the equivalent series resistance (ESR) due to the conductor plates in series with the capacitor.

Appendix Figure 4: Impedance of real vs ideal capacitor.



The impedance profile of a real capacitor has a dip in the middle at the frequency referred to as self-resonant frequency (SRF). It is the point where the reactance of the L and ideal C are equal but of opposite sign [22, p. 327]. And it can be calculated by:

$$f_{SRF} = \frac{1}{2\pi\sqrt{LC}} \quad 2$$

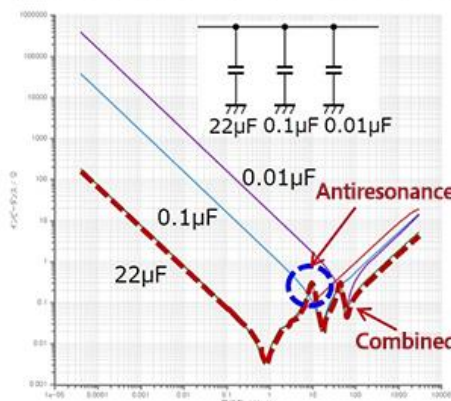
And its impedance by:

$$Z_{CAP} = ESR + X_C + C_L \Rightarrow Z_{CAP} = ESR + 2\pi fL + \frac{1}{2\pi fC} \quad 3$$

Where L is the loop inductance between the capacitor and the pads of the IC. When three capacitors are added in parallel, while there will be three dips, there will also be two peaks corresponding to the parallel resonances of adjacent L and C values (example given on Appendix Figure 5). And what generates the noise on the power rail is a high impedance. It is not about how low the impedance goes, it is about how high the impedance goes [22, p. 329].

Appendix Figure 5: The impedance profile of three different valued decoupling capacitors in parallel.

Addition of Capacitors with Different Values

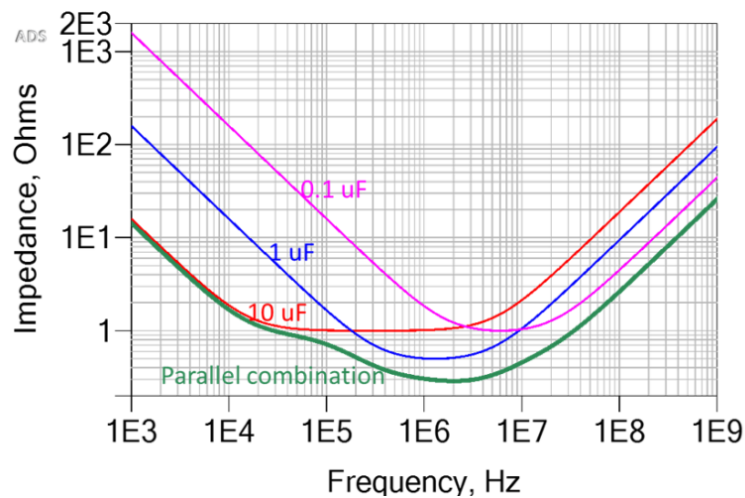


Note: Effective Use of Decoupling (Bypass) Capacitors Point 1 | Dealing with Noise Using Capacitors | TechWeb. (n.d.). <https://techweb.rohm.com/know-how/nowisee/7669/>.

Appendix-Decoupling Capacitors

Back to the recommendation to use three different valued decoupling capacitors for each IC. It assumes that the capacitors are through-hole. Generally, in through-hole capacitors the smaller the capacitance, the smaller its body. This matters because different body sizes will have different mounting inductance (the bigger the body, the higher the inductance). Indeed, if through-hole capacitors are used, choosing three different valued ones in parallel makes sense as portrayed below (the parallel combination yields lower loop inductance).

Appendix Figure 6: Impedance profile of three leaded capacitors and their parallel

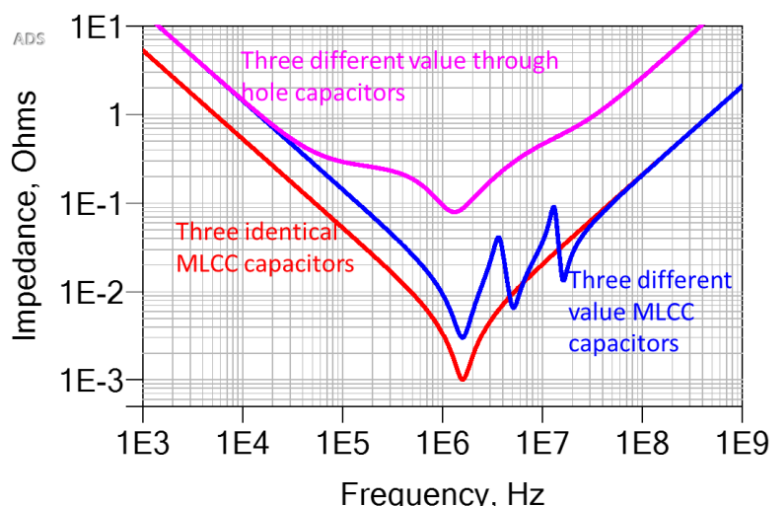


Note: Image taken from "Bogatin's Practical Guide to Prototype Breadboard and PCB Design, p.332"

However, nowadays with SMD or MLCC (Multi-Layered Ceramic Capacitors) the capacitor's body size is the same (at least for the values of interest for decoupling), meaning that they all have the same loop inductance (much lower than the through-hole counterpart). A consequence of that is: MLCC capacitors, all capacitors in the same body size are high-frequency capacitors [22, p. 333].

Concluding this discussion, below is a comparison of the impedance profiles for the cases discussed.

Appendix Figure 7: Comparing three different combinations of three capacitors.



Note: Image taken from "Bogatin's Practical Guide to Prototype Breadboard and PCB Design, p.334".

What was used

It's apparent from the last four figures that at high frequencies, the loop inductance dominates over capacitance when it comes to their impedance contribution. So, a correct conclusion is that when selecting capacitors as decoupling capacitors, a large capacitance for low-frequency performance and a low inductance for high-frequency performance should be the intention.

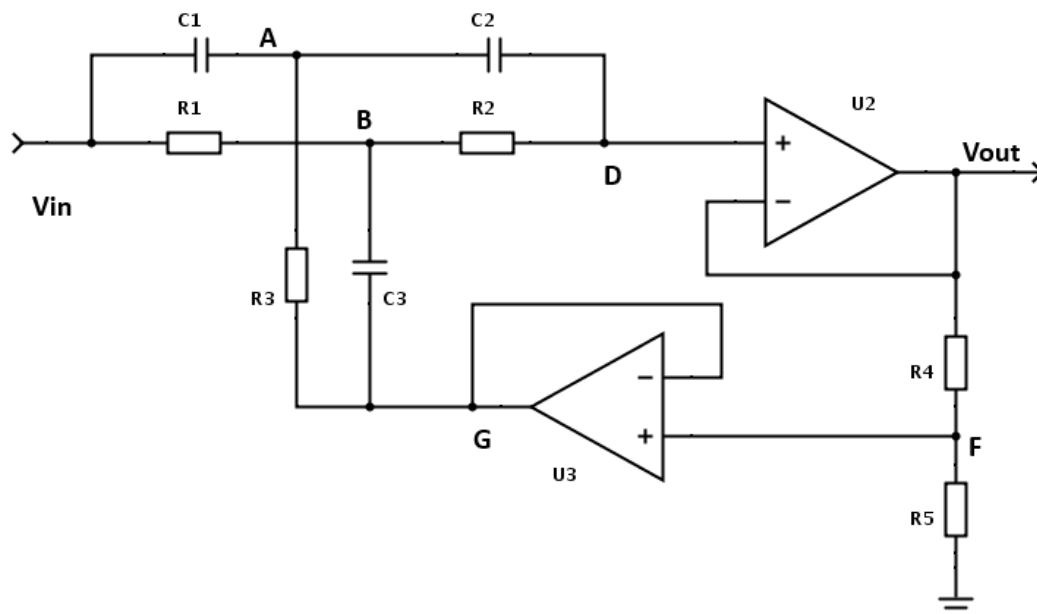
While it was just explained what the ideal combination of capacitors is, generally, a single, low-inductance capacitor may be perfectly adequate [22, p. 334].

That why on this system a $0.1\mu F$ SMD capacitor for each power IC pin was used, eliminating the switching and high frequency noise. But in the PCB schematic there can also be seen $10\mu F$ electrolytic capacitors on each power-in pin of the PCB⁸⁰ (they are parallel with the $0.1\mu F$ ones for each power trail respectively). This higher capacity, bigger volume capacitor takes care of the low carrying-low frequency noise⁸¹.

A.3 DERIVATION OF THE TRANSFER FUNCTION OF THE NOTCH FILTER

On this chapter, the derivation of the notch filter – otherwise called “Bootstrapped Twin-T Notch filter” – will take place.

Appendix Figure 8: The Bootstrapped Twin-T Notch Filter.



⁸⁰ That is 4 pins: +10V, -10V, -15V and + 5V

⁸¹ Notice that the placement of the electrolytic and the SMD capacitors are in line with everything analyzed on this chapter. The electrolytic is responsible for lower frequencies but has a large loop inductance. That loop inductance (that produces high-frequency noise) is taken care of by the SMD capacitor, placed AFTER the electrolytic one (along the power trace) and as close as possible to the IC. Of course, this is true for all ICs and power traces on the PCB.

Appendix-DERIVATION OF THE TRANSFER FUNCTION OF THE NOTCH FILTER

This derivation assumes that the op amps are ideal (their properties are on an earlier chapter - *Digital to Analog Converter*27). That means:

$$V_D = V_{out} \tag{4}$$

And:

$$V_F = V_G \tag{5}$$

Where because of the voltage divider (see node F):

$$V_F = \frac{R_5}{R_4 + R_5} V_{out} \Rightarrow V_F = k V_{out} \tag{6}$$

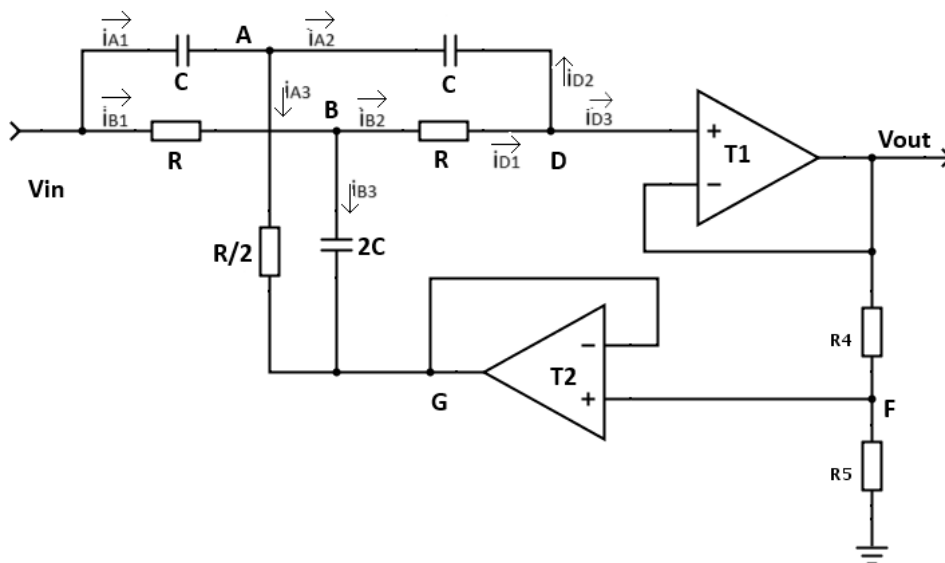
The constant k is equal to:

$$k = \frac{R_5}{R_4 + R_5} \tag{7}$$

Now that some basic properties of the circuit have been laid out, there's one more thing that needs to be pointed out. If the values of the resistors and capacitors (on the filter part – not the voltage divider part) form a “symmetry”, not only yields the frequency response needed, but also makes the derivations easier. Specifically:

$$\begin{cases} R_1 = R_2 = R \\ C_1 = C_2 = C \\ R_3 = \frac{R}{2} \\ C_3 = 2C \end{cases} \tag{8}$$

Appendix Figure 9: The Symmetrical Bootstrapped Twin-T Notch Filter.



Now let's start the derivation by performing nodal analysis on nodes A, B, D. Using Kirchhoff's Current Law (KCL) on node A:

$$i_{A1} = i_{A2} + i_{A3} \quad 9$$

By Ohm's Law:

$$i = \frac{V_{HIGH} - V_{LOW}}{R} \quad 10$$

The Laplace Transform of a capacitor is:

$$Z_C = \frac{1}{sC} \quad 11$$

Where:

$$s = j\omega \quad 12$$

Using all that:

$$\begin{aligned} (V_{IN} - V_A)sC &= (V_A - V_D)sC + \frac{V_A - V_G}{R_3} \Rightarrow \\ (V_{IN} - V_A)sC &= (V_A - V_{OUT})sC + (V_A - kV_{OUT})\frac{2}{R} \Rightarrow \\ V_{IN}sC - V_AsC &= V_AsC - V_{OUT}sC + V_A\frac{2}{R} - \frac{2k}{R}V_{OUT} \Rightarrow \\ V_{IN}sC &= 2V_AsC - V_{OUT}sC + V_A\frac{2}{R} - V_{OUT}\frac{2k}{R} \Rightarrow \\ V_{IN}sC + \left(sC + \frac{2k}{R}\right)V_{OUT} &= \left(2sC + \frac{2}{R}\right)V_A \Rightarrow \\ \boxed{V_A = \frac{V_{IN}sC + \left(sC + \frac{2k}{R}\right)V_{OUT}}{2sC + \frac{2}{R}}} & \quad 13 \end{aligned}$$

Applying the same logic on node B yields:

$$\begin{aligned} i_{B1} &= i_{B2} + i_{B3} \Rightarrow \\ \frac{(V_{IN} - V_B)}{R1} &= \frac{V_B - V_{OUT}}{R} + \frac{V_B - kV_{OUT}}{\frac{1}{2Cs}} \Rightarrow \\ \frac{1}{R}V_{IN} - \frac{1}{R}V_B &= \frac{1}{R}V_B - \frac{1}{R}V_{OUT} + 2sCkV_{OUT} \Rightarrow \\ \frac{1}{R}V_{IN} + \left(\frac{1}{R} + 2skC\right)V_{OUT} &= \left(2sC + \frac{2}{R}\right)V_B \Rightarrow \\ \boxed{V_B = \frac{\frac{1}{R}V_{IN} + \left(\frac{1}{R} + 2skC\right)V_{OUT}}{2sC + \frac{2}{R}}} & \quad 14 \end{aligned}$$

And lastly for node D:

$$i_{D1} = i_{D2} + i_{D3} \quad 15$$

from ideal op amp and from the schematic we get:

$$\begin{cases} i_{D3} = 0 \\ i_{D1} = i_{B2} \\ i_{D3} = -i_{A2} \end{cases} \quad 16$$

Substituting the above equations we get:

$$\begin{aligned} i_{B2} + i_{A2} &= 0 \Rightarrow \\ \frac{V_B}{R} - \frac{V_D}{R} + V_A sC - V_D sC &= 0 \Rightarrow \\ \frac{1}{R} V_B - \frac{1}{R} V_{OUT} + sC V_A - sC V_{OUT} &= 0 \Rightarrow \\ sC V_A + \frac{1}{R} V_B &= \left(sC + \frac{1}{R} \right) V_{OUT} \end{aligned} \quad 17$$

By substituting V_A and V_B on the equation above yields:

$$\begin{aligned} sC \frac{V_{IN} sC + \left(sC + \frac{2k}{R} V_{OUT} \right)}{2sC + \frac{2}{R}} + \frac{1}{R} \frac{\frac{1}{R} V_{IN} + \left(\frac{1}{R} + 2sCk \right) V_{OUT}}{2sC + \frac{1}{R}} &= \left(sC + \frac{1}{R} \right) V_{OUT} \Rightarrow \\ \frac{V_{IN} (sC)^2 + \left[(sC)^2 + \frac{2sCk}{R} \right] V_{OUT} + \frac{V_{IN}}{R^2} + \left(\frac{1}{R^2} + \frac{2sCk}{R} \right) V_{OUT}}{2sC + \frac{2}{R}} &= \left(sC + \frac{1}{R} \right) V_{OUT} \Rightarrow \\ \left[(sC)^2 + \frac{1}{R^2} \right] V_{IN} + \left[(sC)^2 + \frac{2ksC}{R} + \frac{1}{R^2} \right] V_{OUT} &= \left(sC + \frac{1}{R} \right) \left(2sC + \frac{2}{R} \right) V_{OUT} \Rightarrow \\ \left[(sC)^2 + \frac{1}{R^2} \right] V_{IN} = - \left[(sC)^2 + \frac{2ksC}{R} + \frac{1}{R^2} \right] V_{OUT} + \left[2(sC)^2 + \frac{2sC}{R} + \frac{2sC}{R} + \frac{2}{R^2} \right] V_{OUT} &\Rightarrow \\ \left[(sC)^2 + \frac{1}{R^2} \right] V_{IN} = \left[(sC)^2 + \frac{4sC}{R} - \frac{4sCk}{R} + \frac{1}{R^2} \right] V_{OUT} \Rightarrow \\ \frac{V_{OUT}}{V_{IN}} = \frac{(sC)^2 + \frac{1}{R^2}}{(sC)^2 + \frac{1}{R^2} + \frac{4sC(1-k)}{R}} \Rightarrow \\ \boxed{H(s) = \frac{s^2 + \frac{1}{(RC)^2}}{s^2 + \frac{4(1-k)s}{RC} + \frac{1}{(RC)^2}}} \end{aligned} \quad 18$$

Now setting:

$$\omega_0 = \frac{1}{R \cdot C}, s = j\omega \quad 19$$

Resulting in:

$$H(\omega) = \frac{\omega^2 - \omega_0^2}{\omega^2 - 4j\omega_0\omega(1-k) - \omega_0^2} \quad 20$$

Now let's write the denominator in quadratic form:

$$\omega^2 - \omega_0^2 - 4j\omega_0\omega(1-k) \Rightarrow \frac{\omega^2}{\omega_0^2} - 1 = 4j \frac{\omega}{\omega_0} (1-k) \quad 21$$

Setting:

$$g = \frac{\omega}{\omega_0} \quad 22$$

And taking the magnitude on both sides:

$$\begin{aligned} |g^2 - 1| &= \sqrt{(4g(1-k))^2} \Rightarrow g^2 - 1 = \pm 4g(1-k) \\ &\Rightarrow g^2 \pm 4g(1-k) - 1 \end{aligned} \quad 23$$

Using the quadratic formula to solve, we get:

$$g_{1,2} = \pm 2(1-k) \pm \sqrt{4(1-k)^2 + 1} \quad 24$$

Two of those answers are not valid. The variable g is a fraction of two frequency units, that is why only a positive value for it is of interest. It's also true that:

$$k = \frac{R_5}{R_4 + R_5} \rightarrow k < 1 \Rightarrow 1 - k > 0 \quad 25$$

Now let's assume that:

$$2(1-k) < \sqrt{4(1-k)^2 + 1} \Rightarrow 4(1-k)^2 < |4(1-k)^2 + 1| \Rightarrow$$

$$4(1-k)^2 < 4(1-k)^2 + 1 \Rightarrow 0 < 1, \text{ true} \quad 26$$

So, the initial assumption is true as well. All the above leave us with:

$$g_1, g_2 = \sqrt{4(1-k)^2 + 1} \pm 2(1-k) \quad 27$$

The values g_1 and g_2 , are also the cutoff 'frequencies' for the H(g). That is easily proven by just calculating:

$$20 \log|H(g_1)| \text{ and } 20 \log|H(g_2)| \quad 28$$

That means, it's very easy to calculate the Quality factor of this Transfer Function, and by extend, the system it is representing. First, we know that the center frequency (notch frequency) is the geometric mean of the two cutoff frequencies. So:

$$g_0 = \sqrt{g_1 g_2} = \dots = 1 \quad 29$$

Now using equation 22 to go back to actual frequency units:

$$\omega_{NOTCH} = \omega_0 \quad 30$$

Taking the equation for the Bandwidth:

$$BW = |\omega_1 - \omega_2| \Rightarrow BW = (g_1 - g_2) \cdot \omega_0 = \omega_0 \cdot 4(1 - k) \quad 31$$

Now, all information needed for calculating the quality factor are known:

$$Q = \frac{\omega_{NOTCH}}{BW} \Rightarrow Q = \frac{\omega_0}{\omega_0 \cdot 4(1 - k)} \Rightarrow Q = \frac{1}{4(1 - k)} \quad 32$$

Finally, based on the result, we can substitute Q on the initial Transfer Function of the system:

$$\boxed{H(s) = \frac{s^2 + \frac{1}{R^2 C^2}}{s^2 + \frac{1}{QRC} s + \frac{1}{R^2 C^2}}} \text{ or } \boxed{H(s) = \frac{s^2 + \omega_0}{s^2 + \frac{\omega_0}{Q} s + \omega_0}} \quad 33$$

A.4 Deriving the difference equation for a 2nd order Butterworth Lowpass Filter

Starting with the Butterworth Lowpass transfer function:

$$H(s) = \frac{1}{\sum_{k=0}^n a_k \left(\frac{s}{\omega_c}\right)^k} \quad 34$$

Taking the Bilinear transform:

$$s = \frac{2}{T} \cdot \frac{1 - z^{-1}}{1 + z^{-1}} \quad 35$$

We get:

$$H(z) = \frac{1}{\sum_{k=0}^n a_k \frac{1}{\omega_c^k} \left(\frac{2}{T} \cdot \frac{1 - z^{-1}}{1 + z^{-1}}\right)^k} \quad 36$$

For n=2 (second order filter):

$$H(z) = \frac{1}{a_0 + \frac{a_1}{\omega_c} \cdot \frac{2}{T} \cdot \frac{1 - z^{-1}}{1 + z^{-1}} + \frac{a_2}{\omega_c^2} \cdot \left(\frac{2}{T} \cdot \frac{1 - z^{-1}}{1 + z^{-1}}\right)^2} \quad 37$$

After some tedious algebra we get:

$$H(z) = \frac{\frac{A^2}{D} + \left(\frac{2A^2}{D}\right) z^{-1} + \frac{A^2}{D} z^{-2}}{1 + \left[\frac{2A^2 a_0 - 8a_2}{D}\right] z^{-1} + \left[\frac{a_0 A^2 - 2a_1 A + 4a_2}{D}\right] z^{-2}} \quad 38$$

Where:

$$\alpha = T \cdot \omega_c \text{ and } D = A^2 a_0 + 2a_1 A + 4a_2 \quad 39$$

Now the form of the transfer function on equation 38 is directly comparable with the one from equation 2.20 (repeated below for the reader's convenience).

$$H(z) = \frac{\sum_{k=0}^M c_k \cdot z^{-k}}{1 - \sum_{k=1}^N d_k \cdot z^{-k}} \quad 40$$

By direct comparison:

$$\boxed{c_0 = \frac{A^2}{D}}, \quad \boxed{c_1 = \frac{2A^2}{D}}, \quad \boxed{c_2 = \frac{A^1}{D}} \text{ and } \boxed{d_1 = \frac{2A^2 a_0 - 8a_2}{D}}, \quad \boxed{d_2 = \frac{a_0 A^2 - 2a_1 A + 4a_2}{D}}$$

And given that the (general) difference equation is (see equation 2.16):

$$y(n) = \sum_{k=0}^M c_k \cdot x(n-k) - \sum_{k=1}^N d_k \cdot y(n-k) \quad 41$$

we now have everything we need to construct it ($M = N = 2$ since this is a 2nd order example):

$$y(n) = -d_1 y(n-1) - d_2 y(n-2) + b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) \quad 42$$

Which is straight forward to transform into code.

A.5 ARDUINO CODE

FOR OPERATIONAL SYSTEM

```
const int BaudRate = 115200.00;
const unsigned long eventInterval = 2;
unsigned long previousTime = 0;

void setup() {
  Serial.begin(115200);
}

void loop() {
  unsigned long currentTime = millis();
  if (currentTime - previousTime >= eventInterval) {
    int SensorValue= analogRead(A0);
    float Voltage_in = SensorValue * (5.0 / 1023.0);
    Serial.println(Voltage_in, 5);
    previousTime = currentTime;
  }
}
```

FOR TESTING THE DIGITAL FILTER ON MATLAB

```
const float f = 10;
const float pi = 3.14159;
```


Appendix-ARDUINO CODE

```
float mysine = 0.00000;  
float fs=20000;  
void setup() {  
  Serial.begin(115200);  
}  
float t=0.000;  
void loop() {  
  mysine = sin(2*pi*f*t);  
  Serial.println(mysine, 5);  
  t = t + 1/(fs);  
}
```

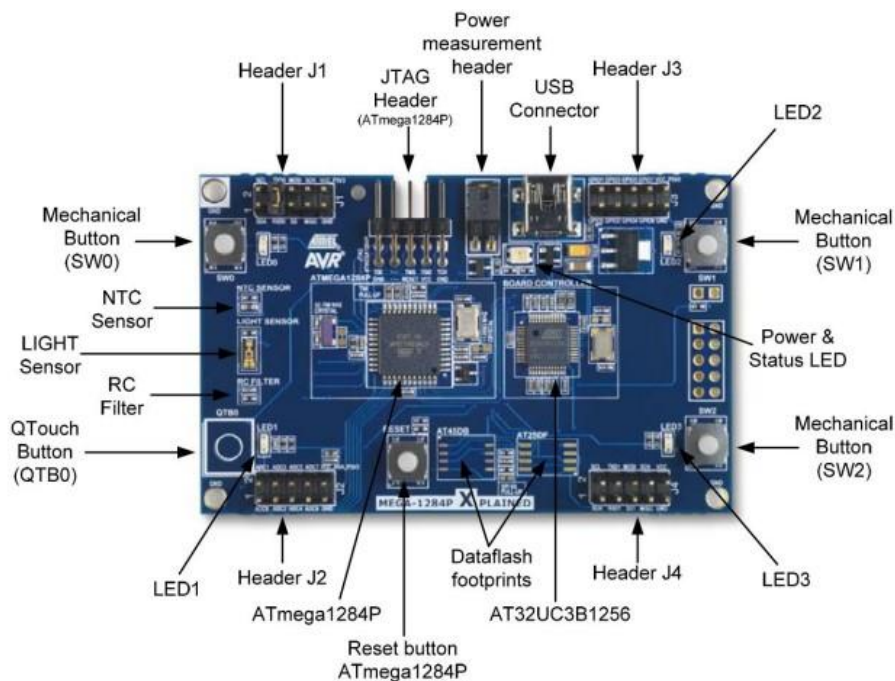
A.6 ATmega Firmware

To test the system with a digital signal source, a microcontroller board can be used. So, for that purpose a sine generator with adjustable frequency was designed using the ATMEL ATmega1284P programming board, which is a development board for the ATmega1284P microcontroller. On Appendix Figure 10 all the capabilities, I/O, buttons and sensors on the board are shown.

To achieve this, the main idea was the so-called **look-up table**. The microcontroller would read the values of a table which are sampled values of a sine wave, while the adjusting of the frequency would happen by changing the rate at which the look-up table values are read.

This firmware was coded in AVR-Assembly using the IDE **Microchip Studio**⁸² through the JTAG header (using a debugger⁸³). The actual code can be seen on A.7.

Appendix Figure 10: The ATmega 1284p board.



Note: Picture taken from "AVR364: MEGA-1284P Xplained Hardware User's Guide".

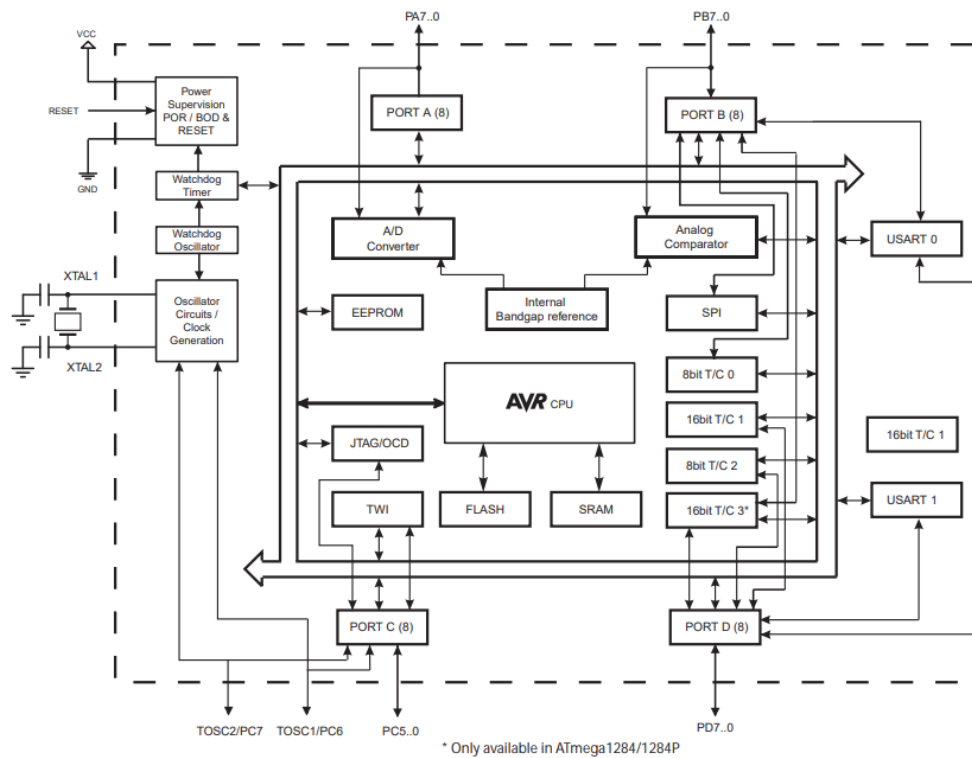
MCU's Architecture

Before explaining the details of the firmware, it's necessary to dive into the microcontroller's architecture. The ATmega1284P is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega1284P achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed [24, p. 13].

⁸² It's the IDE used for developing and debugging AVR® and SAM microcontroller applications.

⁸³ More specifically, Microchip Atmel-ICE, Programming Kit for SAM and Atmel AVR Microcontrollers.

Appendix Figure 11: ATmega1284P board's block diagram.



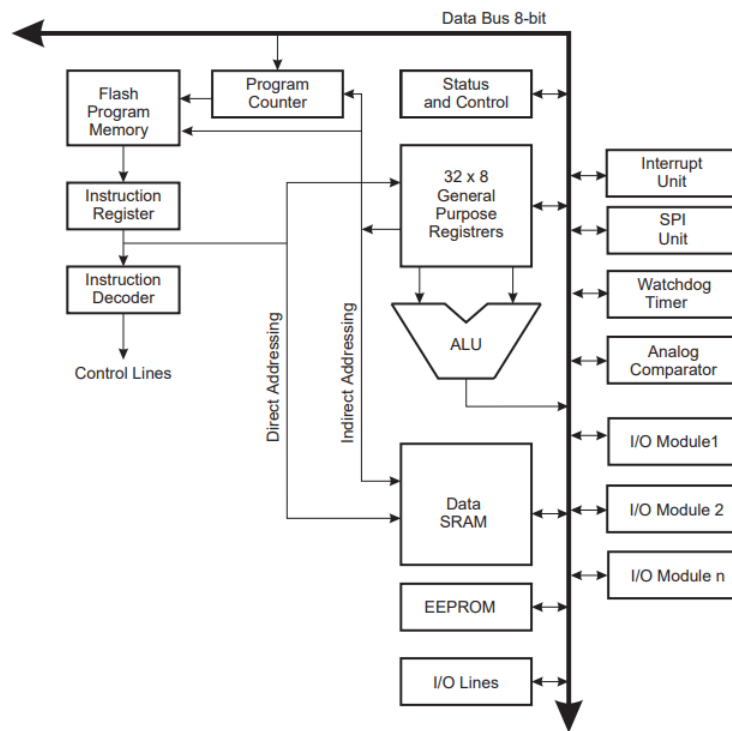
Its features include but not limited to 128KB of In-System Programmable Flash with Read-While-Write capabilities, 4KB EEPROM, 16KB SRAM, 32 general purpose I/O lines, 32 general working registers⁸⁴, three flexible Timer/Counters with compare modes and PWM, a 10-bit ADC with optional differential input stage with programmable gain.

Those 32 I/O lines are divided into 4 PORTS: PORTA, PORTB, PORTC, PORTD. All four of them are 8-bit bi-directional I/O ports with internal pull-up resistors (selected for each bit) and all of them also serve the functions of various other features of the microcontroller. But, only one of them – PORTA – serves as analog input for the board, because it's connected to the board's ADC.

Moving on to the AVR core architecture, its block diagram is shown in Appendix Figure 12. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

⁸⁴ The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle [24, p. 14].

Appendix Figure 12: Block diagram of the AVR architecture.



The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle [24, pp. 18–19].

Six of the 32 registers⁸⁵ can be used as three 16-bit indirect address register pointers for Data Space addressing – enabling efficient address calculations (specifically R26-R31 as shown below). One of these address pointers can also be used as an address pointer for look-up tables in Flash program memory. These added function registers are the 16-bit X-, Y-, and Z-register (which will be used in the code).

Appendix Figure 13: AVR CPU General Purpose Working Registers (Register File).

| | 7 | 0 | Addr. | |
|-----------------------------------|-----|---|-------|----------------------|
| General Purpose Working Registers | R0 | | 0x00 | |
| | R1 | | 0x01 | |
| | R2 | | 0x02 | |
| | ... | | | |
| | R13 | | 0x0D | |
| | R14 | | 0x0E | |
| | R15 | | 0x0F | |
| | R16 | | 0x10 | |
| | R17 | | 0x11 | |
| | ... | | | |
| | R26 | | 0x1A | X-register Low Byte |
| | R27 | | 0x1B | X-register High Byte |
| | R28 | | 0x1C | Y-register Low Byte |
| | R29 | | 0x1D | Y-register High Byte |
| | R30 | | 0x1E | Z-register Low Byte |
| | R31 | | 0x1F | Z-register High Byte |

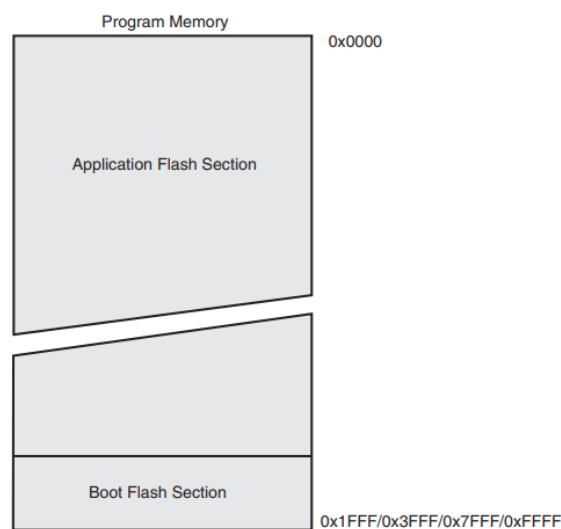
⁸⁵ The registers store data elements.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the **Status Register** is updated to reflect information about the result of the operation [24, p. 19].

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction [24, p. 19].

Program Flash memory space is divided into two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection [24, p. 19].

Appendix Figure 14: Program memory map.



During interrupts⁸⁶ and subroutine calls, the return address Program Counter⁸⁷ (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the Stack Pointer (SP) in the Reset routine (before subroutines or interrupts are executed). The Stack Pointer⁸⁸ is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

But what is the role of this **Status Register** that is mentioned in earlier paragraphs? The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the AVR Instruction Set Manual. This will in many cases remove the need for using

⁸⁶ Interrupts are used on this firmware and will be explained further in the next pages.

⁸⁷ The Program Counter also known as Instruction Pointer is a special register that keeps track of the memory address of the next instruction to be executed in a program.

⁸⁸ The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space [24, p. 22].

the dedicated compare instructions, resulting in faster and more compact code. The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software [24, p. 20]. It's an 8-bit register and it's vital for interrupts since its most significant bit is the **Global Interrupt Enable** bit. It must be set (to high) for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings [24, p. 20]. This fact is important as this Bit will be set multiple times throughout the code.

Going over the code

This firmware uses what is called an interrupt. Programming using interrupts is a technique based on an automatic mechanism in the hardware of the microcontroller, which allows a device to provide service to internal or external devices, only at the moment it is required. The interrupt is asynchronous and can occur at any time during the execution of the main program. Then they can stop the main program from executing to perform a separate interrupt service routine (ISR). When the ISR is completed, program control is returned to the main program at the instruction that was interrupted [25].

The main program (main loop) of this system's firmware is shown on the below. This loop runs continuously, until the board is disconnected from power, or an interrupt is enabled. The instruction "***nop***" basically means "no operation". It essentially keeps the microprocessor idle until further notice. This means that the goal of this firmware will be handled by the interrupt technique alone.

Appendix Figure 15: Main Program/Loop.

```
main:
  nop
  nop
  nop
  nop
  nop
  nop
  nop
  nop
  rjmp main
```

There are two interrupts used here. One is utilizing a timer (see ***TIMER1_OVF***" - Vector 16 on Table 5) and the other an onboard ADC ("***ADC***" - Vector 25). The idea is that each time the timer/counter interrupt goes off, a value of the look-up table will be read. How it works, is it starts counting from an initial value at a certain frequency. If somehow that initial value was changed live-while the board is on, let's say increased, then the look-up table's values would be read faster. That is where the ADC comes in. By connecting a potentiometer between the V_{CC} and GND pins with its output connected to a PORTA pin (since it's connected to the

onboard ADC), then by turning the potentiometer the counter's initial value could be changed, altering the sine's frequency live.

Table 5: Reset and Interrupt Vectors.

| Vector Number | Program Address | Source | Interrupt Definition |
|---------------|-----------------|---------------|---|
| 1 | \$0000 | RESET | External Pin. Power-on Reset, Brown-out Reset, Watchdog Reset. and JTAG AVR Reset |
| 2 | \$0002 | INT0 | External Interrupt Request 0 |
| 3 | \$0004 | INT1 | External Interrupt Request 1 |
| 4 | \$0006 | INT2 | External Interrupt Request 2 |
| 5 | \$0008 | PCINT0 | Pin Change Interrupt Request 0 |
| 6 | \$000A | PCINT1 | Pin Change Interrupt Request 1 |
| 7 | \$000C | PCINT2 | Pin Change Interrupt Request 2 |
| 8 | \$000E | PCINT3 | Pin Change Interrupt Request 3 |
| 9 | \$0010 | WDT | Watchdog Time-out Interrupt |
| 10 | \$0012 | TIMER2_COMPA | Timer/Counter2 Compare Match A |
| 11 | \$0014 | TIMER2_COMPB | Timer/Counter2 Compare Match B |
| 12 | \$0016 | TIMER2_OVF | Timer/Counter2 Overflow |
| 13 | \$0018 | TIMER1_CAPT | Timer/Counter1 Capture Event |
| 14 | \$001A | TIMER1_COMPA | Timer/Counter1 Compare Match A |
| 15 | \$001C | TIMER1_COMPB | Timer/Counter1 Compare Match B |
| 16 | \$001E | TIMER1_OVF | Timer/Counter1 Overflow |
| 17 | \$0020 | TIMER0_COMPA | Timer/Counter0 Compare Match A |
| 18 | \$0022 | TIMER0_COMPB | Timer/Counter0 Compare match B |
| 19 | \$0024 | TIMER0_OVF | Timer/Counter0 Overflow |
| 20 | \$0026 | SPI_STC | SPI Serial Transfer Complete |
| 21 | \$0028 | USART0_RX | USART0 Rx Complete |
| 22 | \$002A | USART0_UDRE | USART0 Data Register Empty |
| 23 | \$002C | USART0_TX | USART0 Tx Complete |
| 24 | \$002E | ANALOG_COMP | Analog Comparator |
| 25 | \$0030 | ADC | ADC Conversion Complete |
| 26 | \$0032 | EE_READY | EEPROM Ready |
| 27 | \$0034 | TWI | two-wire Serial Interface |
| 28 | \$0036 | SPM_READY | Store Program Memory Ready |
| 29 | \$0038 | USART1_RX | USART1 Rx Complete |
| 30 | \$003A | USART1_UDRE | USART1 Data Register Empty |
| 31 | \$003C | USART1_TX | USART1 Tx Complete |
| 32 | \$003E | TIMER3_CAPT | Timer/Counter3 Capture Event |
| 33 | \$0040 | TIMER3_COMPA | Timer/Counter3 Compare Match A |
| 34 | \$0042 | TIMER3_CONAPB | Timer/Counter3 Compare Match B |
| 35 | \$0044 | TIMER3_OVF | Timer/Counter3 Overflow |

A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register⁸⁹. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower the Interrupt Vector address, the higher the priority [24, p. 19].

The code starts by tackling the interrupts. This is because the interrupts are placed at the start of the Flash memory⁹⁰ [24, p. 73]. The first command is “*jmp INIT*”, which makes sure the initialization routine for the interrupts is run first (see Appendix Figure 17 and Appendix Figure 18). Immediately after there is a series of pairs of “*nop*” and “*reti*” commands – 35 pairs to be exact – the same number as the interrupts. Each pair of these commands disables an interrupt (if all interrupts are enabled, the main loop may be interrupted at seemingly random times, which is obviously not wanted). They are disabled one by one in the order shown in Table 5, and that’s because they are identified based on their Flash memory address. Notice that the **TIMER1_OVF** and **ADC_INTER** which will be used, are number 15 and 24 respectively instead of 16 and 25 like in Table 5. This is because the first command of the code “*jmp INIT*” takes up 2 flash addresses⁹¹.

Appendix Figure 16: Enabling the two interrupts that will be used.

```
31  jmp TIMER1_OVF ;15
32  ;reti
33  nop;16
34  reti
35  nop;17
36  reti
37  nop;18
38  reti
39  nop;19
40  reti
41  nop;20
42  reti
43  nop;21
44  reti
45  nop;22
46  reti
47  NOP;23
48  RETI
49  jmp ADC_INTER;24 ;flash memory address -> 0030 + 0031
50  reti
```

Also notice that the “*reti*” command after the “*jmp TIMER1_OVF*” is commented out while the one after “*jmp ADC_INTER*” is not. That is because the “*jmp TIMER1_OVF*” takes up address on the flash memory. Had the “*reti*” not been commented out, then the addresses would be misaligned and the enabling of the ADC_INTER interrupt would have failed⁹².

Now to explain what’s in the initialization (initial settings for their first run) for the interrupts. Both of their behaviour (along with all the interrupts) are controlled by the values of some

⁸⁹ Will be explained in the next pages.

⁹⁰ This can be changed by changing a bit in the MCUCR (MCU control Register) – specifically the IVSEL bit. When it’s set to high, the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash [24, p. 73]. But this is not the case here.

⁹¹ While “*nop*” and “*reti*” take up 1 flash memory address (together-not each).

⁹² This of course happens because the ADC interrupt has lower priority than the TIMER1_OVF since its vector is larger – comes at a later address.

registers (each interrupt has its own of course). First of all, while configuring these settings⁹³, all interrupts must be disabled – and reenabled when done. This is achieved through the “cli” and “sei” commands.

Starting with the ADC, it is enabled by setting the ADC Enable bit (ADEN) in the ADCSRA register. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX. If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the Data Registers belongs to the same conversion. Once ADCL is read, ADC access to Data Registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled [24, p. 245]. The ADC has its own interrupt which can be triggered when a conversion is completed. Additionally, with ADMUX’s last five LSBs one can choose which pin to use as an analog input. In the end the value 00000 was given to them, making the ADC0 pin the input pin, to which the output of the potentiometer will be connected.

A conversion can be triggered automatically by various sources. Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit, ADSC in ADCSRA. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB [24, p. 245]. Finally, the DIDR0 register’s bits should all be set to 1 in order to disable digital input (since the input in this case is analog). More specifically, when this bit is written logic one, the digital input buffer on the corresponding ADC pin is disabled. When an analog signal is applied to the ADC7:0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer [24, p. 261].

Appendix Figure 17: Initialization settings for the ADC interrupt.

```
76 INIT:
77
78     cli ; disables interrupts
79
80 ;-----setting the adc interrupt-----
81     ldi r16,$E0
82     STS ADMUX,r16
83
84     ldi r16,$E8
85     STS ADCSRA,r16
86
87     ldi r16,$00
88     STS ADCSRB,r16
89
90     ldi r16,$FF
91     STS DIDR0,r16
92
93 ;ADCL and ADCH - The ADC Data Register
94     lds r20,ADCL
95     lds r21,ADCH
96
97 ;ADMUX->11 0 00000
98 ;     ; ↓
99 ;     ; right adjust
```

⁹³ This also goes for later configuring – not just the initialization.

Concluding this part, the appropriate values were given to the ADC's Register's bit in order to have auto triggering, the trigger source to be on Free Running mode (ADCSRB register), to choose an input pin and various other details in order for it to work properly. More in depth analysis can be found on the board's datasheet.

Appendix Figure 18: Initialization settings for the Timer interrupt.

```

101 ;-----setting the timer1 interrupt initial settings-----
102
103 ; σεταρω καταλληλα τα registers
104 ldi r16,$00
105 STS TCCR1A,r16 ;-> 00 00 00 00
106
107 ldi r16,$01
108 STS TCCR1B,r16 ;-> 00 00 01 01
109 ;SO bySETTING TCCR1A AND TCCR1B LIKE THAT, WGM13 WGM12 WGM11 WGM10-> 0000 WHICH MEANS NORMAL OPERATION (NO PWM etc)
110 ;(2 of them are in TCCR1A and the other 2 in TCCR1B)
111 ;the last 3 bits of TCCR1B are choosing the clock prescaler
112
113 ldi r16,$00
114 STS TCCR1C,r16
115
116 ldi r16,$01
117 STS TIMSK1, r16
118
119 ldi r16,$01
120 STS TIFR1,r16
121 ; δινω στο counter αρχικη τιμη
122 ldi r16,$01
123 ldi r17,$01
124 STS TCNT1H,r16
125 STS TCNT1L,r17
126
127 sei ;enables interrupts

```

Moving on to the initialization of the TIMER1_OVF. The Timer/Counter (TCNT1) is a 16-bit register. The Timer/Counter Control Registers (TCCR1A/B/C), on the other hand, are an 8-bit registers. Interrupt requests signals are all visible in the Timer Interrupt Flag Register (TIFR1). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK1) [24, p. 116].

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the Tn pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock (clkTn) [24, p. 116].

The simplest mode of operation is the Normal mode (WGMn3:0 = 0⁹⁴), which is the mode used in this system. In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value (MAX = 0xFFFF) and then restarts from the BOTTOM (0x0000). In normal operation the Timer/Counter Overflow Flag (TOV1) will be set in the same timer clock cycle as the TCNT1 becomes zero. The TOV1 Flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV1 Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime [24, p. 127].

⁹⁴ Controlled by the TCCR1A and TCCR1B registers.

Appendix-ATmega Firmware

Lastly, the TCNT1H and TCNT1L registers give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter [24, p. 138]. And by changing their value, the initial value (the one from which the Timer/Counter starts counting)⁹⁵.

So, all that needs to be done, is equalize the values of these 2 registers to those of the ADC's result (ADCL and ADCH) and then the goal is achieved.

Finally, at the end of the *INIT* part, there are some necessary settings to be addressed that have nothing to do with the interrupts. On lines 130 and 131 (on the figure below), the PORTB is configured as an output (it's through there that the look-up table values are sent to the PCB). On 133 and 134 PORTA is configured as an input (since it's going to be to where the potentiometer is connected) and on 136 its internal pull-up resistors are enabled. Line 137 resets the general-purpose register r17 to all zeros. Lastly, on lines 139 and 140, the address of look-up table is acquired. This can only be done (as previously stated on page - 17 -) by using the general-purpose register Z (comprised of r30 and r31)⁹⁶. More specifically, the two registers (r30 and r31) now have the address on the Flash Memory of the look-up table's first value. So to get to the rest of the values and eventually output them, all that needs to be done is "go" to the next address. This is done by the command "*ADIW ZL,\$01*", which will be used later on. The reason the table's address was first acquired here (on the *INIT* where the initial setting for the interrupts were configured) is because when the time comes for the first look-up table value to be read and sent to the PCB, the program knows where to look for it.

Appendix Figure 19: Configuring the ports and getting the flash address of the look up table.

```
130 LDI R17,$FF
131 OUT DDRB,R17; output
132
133 LDI R17,$00
134 OUT DDRA,R17;INPUT (IN PINA WE WILL CONNECT THE POTENTIOMETER, CAUSE PORTA IS CONNECTED TO AN ONBOARD ADC)
135 ldi r17, $FF ; Init value
136 out PORTA, r17 ; Enable pull-up resistors
137 LDI R17,$00
138
139 LDI ZH, HIGH(sinetable*2)
140 LDI ZL, LOW(sinetable*2) ; τα settings εδώ, γιατί την πρώτη φορά που θα κάνει ο timer interrupt, θα πρέπει να ξέρει που θα διαβάσει.
141
```

The main loop is covered, as is the initialization and enabling of the interrupts. That only leaves explaining the part of the code that's executed when each interrupt happens.

⁹⁵ For more information about each bit of each register and their functions see the board's datasheet.

⁹⁶ Two register because the Flash memory address is 16bits.

Appendix Figure 20: What 's executed when the ADC interrupt happens.

```

164 ADC_INTER:
165
166 cli ; disables interrupts
167 in r19,sreg
168
169 ldi r16,$E8
170 STS ADCSRA,r16
171
172 ;ADCL and ADCH – The ADC Data Register
173 ;ADCL and ADCH – The ADC Data Register
174 lds r20,ADCL; πρέπει να διαβαστούν και τα δυο για να κάνει update
175 lds r21,ADCH; πρέπει να διαβαστούν και τα δυο για να κάνει update
176
177 LSR r21
178 LSR r21
179 LSR r21
180 LSR r21
181
182 LDI R16,$F0
183 add r21,r16; 1111 xxxx
184 LDI R16,$00; xx00 0000
185 add r20,r16
186 out sreg,r19
187 sei ;enables interrupts
188 reti
    
```

Starting with the ADC: after temporarily disabling the interrupts, the next thing is saving the Status Register’s information to general purpose register 19 for “safekeeping”. It’s a safety measure because depending on what’s in the main program, maybe while this part of the code is run, the Status Register’s data get changed which could lead to abnormal and unpredictable behaviour. The necessity of this was already stressed earlier (see page - 19 -). Next, the ADCRA register is re-configured the same way (as in the initialization), to reenale the ADC and auto-triggering among other minor settings. This is necessary – without it there would be an interrupt each time the potentiometer was used – but only the first time.

Then, the result of the ADC – stored in ADCL and ADCH registers, is moved to the general-purpose registers r20 and r21 respectively (see line 174-175 on Appendix Figure 20). Through the value given to the ADMUX register, the ADCL and ADCH are left adjusted (visualized on Appendix Figure 21).

Appendix Figure 21: Left adjusted ADCL and ADCH.

| | | | | | | | | | |
|--------|------|------|------|------|------|------|------|------|------|
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| (0x79) | ADC9 | ADC8 | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADCH |
| (0x78) | ADC1 | ADC0 | - | - | - | - | - | - | ADCL |

This means that the MSB (ADC9) is located on the ADCH on the left of the “matrix”. By the command “*LSR r21*” the MSB (which was saved on r21 earlier) is shifted right four times (as the command is repeated four times). This leaves the four “most significant” spaces of the r21 vacant. The next two commands are “*LDI r16,\$F0*” and “*add r21,r16*”. The former gives the value of *F0* (HEX) to the r16 register, and the latter adds the value of r16 to that of the r21 and saves it on the r21. This results in the r21 having the value (from MSB: left to LSB: right): 1 – 1 – 1 – 1 – ADC9 – ADC8 – ADC7 – ADC6. All this was done so the counter will still

count fast enough (at the appropriate range) for measuring on the custom PCB, while at the same time the potentiometer's value has a noticeable effect on the frequency⁹⁷.

Then, on lines 184 and 185, the r20 gets its "vacant" places filled with 0's, so later when this value is transferred to the counter's starting number, it would lead to a bug.

Lastly, the saved state of the Status Register is transferred back to the current Status Register, and the interrupts are reenabled.

Appendix Figure 22: The code executed when the Timer interrupt happens.

```
190  TIMER1_OVF:
191
192      cli
193      in r19,SREG
194
195      lpm
196      adiw z1,$01
197      mov r23,r0
198      cpi r23,$18 ; με αυτη την τιμη τελειωνει το table
199      breq finito_sine
200      OUT portb, R23
201
202      ;;ldi r20,$00
203      ;ldi r21,$00
204      STS TCNT1H,r21; οι τιμες απο το ADC - ετσι ελεγχω την συχνότητα
205      STS TCNT1L,r20; οι τιμες απο το ADC - ετσι ελεγχω την συχνότητα
206      out SREG,r19
207      sei
208  reti
```

Continuing with the code that's executed only when the Timer interrupts occurs, after the two known commands (lines 192, 193), the command "*lpm*" is executed. This command moves the contents of the ZL (==address) to r0. This is done cause the Z register is not a "normal" one like the other 32. This command along with r0 act as mediators. Indeed, after going to the next address of the table (line 196), the register r0 is used on line 197. All its data are transferred to the register r23. Now r23 "points" to the value the address on r0 was leading.

On line 198 the value that the address on r23 points, is compared to \$18, which is the last value of the look-up table (that's how it's detected if the end of the table is reached). If they are equal, the code jumps to the loop called "finito_sine". If not, then the value is outputted to PORTB. Then, on lines 204-205, the values of the ADC's current result are transferred to the TCNT1H and TCNT1L respectively, so when the counter starts counting again, it starts from the value given by the potentiometer. The rest three commands are already explained.

Finally, on the "finito_sine" loop, the look-up table's first value's address is acquired again (lines 212-213), so we can start over and the output can be continuous and periodic as it should be. Again, the ADC's current values are transferred to the initial number of the counter (lines 216 217).

This conclude the ATmega1284P's firmware analysis.

⁹⁷ Had this not been done, the frequency would be too low (without the four MSBs being constantly 1111)

Appendix-ATmega Firmware

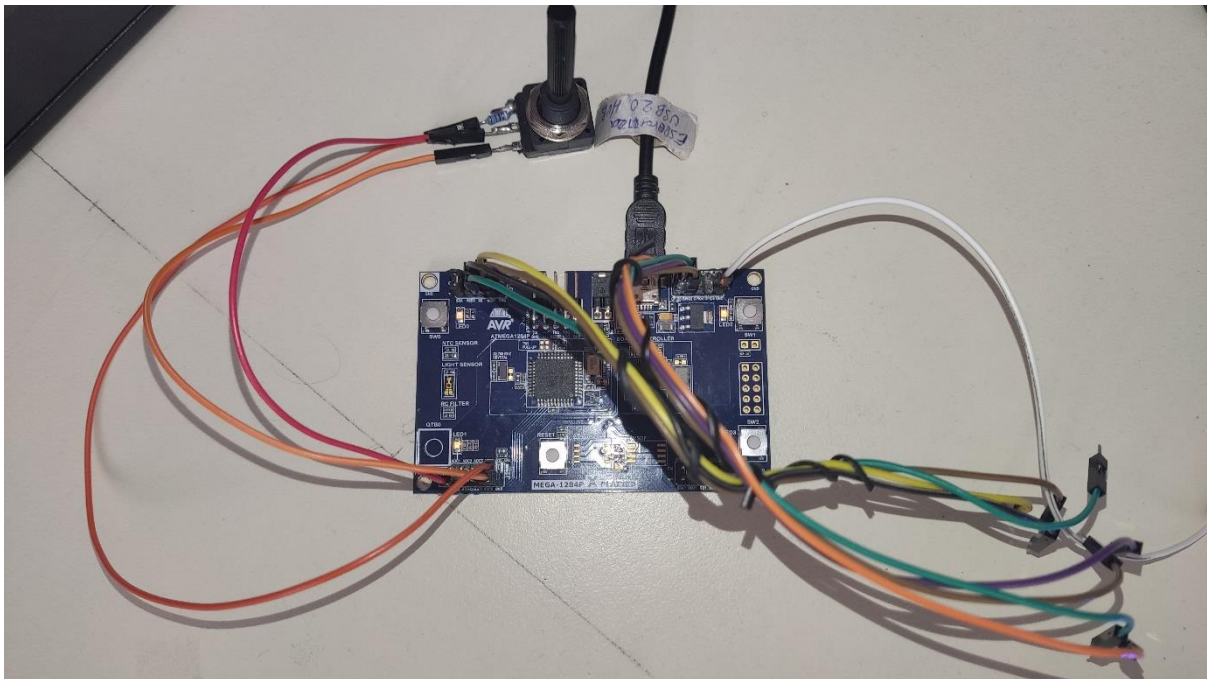
Appendix Figure 23: *finito_sine* loop's code.

```
210  finito_sine:
211
212      LDI ZH, HIGH(sinetable*2)
213      LDI ZL, LOW(sinetable*2)
214      ;ldi r20,$00
215      ;ldi r21,$00
216      STS TCNT1H,r21; οι τιμες απο το ADC - ετσι ελεγχω την συχνότητα
217      STS TCNT1L,r20; οι τιμες απο το ADC - ετσι ελεγχω την συχνότητα
218
219      out SREG,r19
220      sei
221      reti
```

Appendix Figure 24: The look-up table.

```
sinetable:
.DW $887F,$9A91,$ACA3,$BDB4,$CCC5,$DAD3,$E6E0,$EFEB,$F7F3,$FBF9,$FDFD,$FDFD,$F9FB,$F3F7,$EBEF,$E0E6,$D3DA,$C5CC
.DW $B4BD,$A3AC,$919A,$7F88,$6C75,$5A63,$4951,$3840,$2A31,$1D23,$1217,$0A0E,$0406,$0002,$0000,$0200,$0604,$0E0A
.DW $1712,$231D,$312A,$4038,$5149,$635A,$756C,$0018
```

Appendix Figure 25: The ATmega1284P board ready to be connected to the rest of the system.



A.7 ATmega CODE

```
jmp INIT ; 0000 + 0001 -> flash memory address

nop;1
reti
nop;2
reti
nop;3
reti
nop;4
reti
nop;5
reti
nop;6
reti
nop;7
reti
nop;8
reti
nop;9
reti
nop;10
reti
nop;11
reti
nop;12
reti
nop;13
reti
nop;14
reti
jmp TIMER1_OVF ;15
;reti
nop;16
reti
nop;17
reti
nop;18
reti
nop;19
reti
nop;20
reti
nop;21
reti
nop;22
reti
NOP;22
RETI
jmp ADC_INTER;23 ;flash memory address -> 0030 + 0031
reti
nop;24
reti
nop;25
reti
nop;26
reti
nop;27
```

Appendix-ATmega CODE

```
reti
nop;28
reti
nop;29
reti
nop;30
reti
nop;31
reti
nop;32
reti
nop;33
reti
nop;34
reti
nop;35
reti

INIT:

    cli ; disables interrupts

;-----setting the adc interrupt-----
    ldi r16,$E0
    STS ADMUX,r16

    ldi r16,$E8
    STS ADCSRA,r16

    ldi r16,$00
    STS ADCSRB,r16

    ldi r16,$FF
    STS DIDR0,r16

;ADCL and ADCH - The ADC Data Register
lds r20,ADCL
lds r21,ADCH

;ADMUX->11 0 00000
;           ; ↓
;           ;right adjust

;-----setting the timer1 interrupt initial settings-----

; σετάρω κατάλληλα τα registers
ldi r16,$00
STS TCCR1A,r16 ;->00 00 00 00

ldi r16,$01
STS TCCR1B,r16 ;-> 00 00 01 01    SO bySETTING TCCR1A AND TCCR1B LIKE
THAT, WGM13 WGM12 WGM11 WGM10→ 0000 WHICH MEANS NORMAL OPERATION (NO PWM
etc) (2 of them are in TCCR1A and the other 2 in TCCR1B)
;the last 3 bits of TCCR1B are choosing the clock prescaler

ldi r16,$00
STS TCCR1C,r16

ldi r16,$01
STS TIMSK1, r16
```


Appendix-ATmega CODE

```
ldi r16,$01
STS TIFR1,r16
; Im giving to the counter an initial value
ldi r16,$01
ldi r17,$01
STS TCNT1H,r16
STS TCNT1L,r17

sei ;enables interrupts
;-----

LDI R17,$FF
OUT DDRB,R17; output

LDI R17,$00
OUT DDRA,R17;INPUT (IN PINA WE WILL CONNECT THE POTENTIOMETER, CAUSE
PORTA IS CONNECTED TO AN ONBOARD ADC)
ldi r17, $FF ; Init value
out PORTA, r17 ; Enable pull-up resistors
LDI R17,$00

LDI ZH, HIGH(sinetable*2)
LDI ZL, LOW(sinetable*2); ta settarw edw, γιατι την πρώτη φορά που θα
κάνει ο timer interrupt, θα πρέπει να ξέρει που θα διαβάσει.

main:
nop
nop
nop
nop
nop
nop
nop
nop
rjmp main

sinetable:
.DW
$887F,$9A91,$ACA3,$BDB4,$CCC5,$DAD3,$E6E0,$EFEB,$F7F3,$FBF9,$FDFD,$FDFD,$F9
FB,$F3F7,$EBEF,$E0E6,$D3DA,$C5CC,$B4BD,$A3AC,$919A,$7F88,$6C75,$5A63,$4951,
$3840
.DW
$2A31,$1D23,$1217,$0A0E,$0406,$0002,$0000,$0200,$0604,$0E0A,$1712,$231D,$31
2A,$4038,$5149,$635A,$756C,$0018

ADC_INTER:

cli ; disables interrupts
in r19,sreg

ldi r16,$E8
STS ADCSRA,r16

;ADCL and ADCH - The ADC Data Register
;ADCL and ADCH - The ADC Data Register
lds r20,ADCL; Both must be read so the register gets updated
lds r21,ADCH; Both must be read so the register gets updated

LSR r21
```

Appendix-ATmega CODE

```
LSR r21
LSR r21
LSR r21

LDI R16,$F0
add r21,r16; 1111 xxxx
LDI R16,$00; xx00 0000
add r20,r16
out sreg,r19
sei ;enables interrupts
reti

TIMER1_OVF:

cli
in r19,SREG

lpm
adiw z1,$01
mov r23,r0
cpi r23,$18 ; that's the last value of the table
breq finito_sine
OUT portb, R23

;;ldi r20,$00
;;ldi r21,$00
STS TCNT1H,r21; Taking the values from the ADC - that's how I can
change the frequency

STS TCNT1L,r20; Taking the values from the ADC - that's how I can
change the frequency

out SREG,r19
sei
reti

finito_sine:

LDI ZH, HIGH(sinetable*2)
LDI ZL, LOW(sinetable*2)
;;ldi r20,$00
;;ldi r21,$00
STS TCNT1H,r21; Taking the values from the ADC - that's how I can
change the frequency

STS TCNT1L,r20; Taking the values from the ADC - that's how I can
change the frequency

out SREG,r19
sei
reti
```

A.8 MATLAB CODE

CODE CREATING THE CLASS

```

classdef Get_data_class2 < handle %This is a Handle Class
    properties (SetAccess = public)
        rt_data=[];
        time_variable=[];
        board=[];
        Sample_Freq=[];
        Samples=[];
        Fourier_data=[];
        Hertz=[];
        Filtered_data=[]; %
        Filter_Fourier_data=[];
        Filter_Hertz=[];
        tab1_plots=zeros(3,1);
        tab2_plots=zeros(3,1);
        tab3_plots=zeros(3,1);
        Filter_Spec_plots=zeros(2,1);
        prog_bar;
        Fourier_Peaks_Input=[];
        Fourier_Peak_Pos_Input=[];
        Fourier_Peaks_Filtered=[];
        Fourier_Peak_Pos_Filtered=[];
        tab1_edt_field;
        tab2_edt_field;
        tab3_edt_field;
        Filter_Cutoff_Ang_Freq;
        Filter_order;
        Var_window=zeros(2,1);
        Spec_window=zeros(2,1);
        Baudrate;
        Offset;
    end
    methods
        function S=saveobj(obj) % defining a save object method
            %S.board=obj.board; no need to be saved
            S.rt_data=obj.rt_data;
            S.time_variable=obj.time_variable;
            S.Sample_Freq=obj.Sample_Freq;
            S.Samples=obj.Samples;
            S.Filtered_data=obj.Filtered_data;
            S.Fourier_data=obj.Fourier_data;
            S.Hertz=obj.Hertz;
            S.Filter_Fourier_data=obj.Filter_Fourier_data;
            S.Filter_Hertz=obj.Filter_Hertz;
            S.Fourier_Peaks_Input=obj.Fourier_Peaks_Input;
            S.Fourier_Peak_Pos_Input=obj.Fourier_Peak_Pos_Input;
            S.Fourier_Peaks_Filtered=obj.Fourier_Peaks_Filtered;
            S.Fourier_Peak_Pos_Filtered=obj.Fourier_Peak_Pos_Filtered;
            %s.tab1_edt_field=obj.tab1_edt_field;→ no need to be saved
            %s.tab2_edt_field=obj.tab2_edt_field;→ no need to be saved
            %s.tab3_edt_field=obj.tab3_edt_field;→ no need to be saved
            S.Filter_Cutoff_Ang_Freq=obj.Filter_Cutoff_Ang_Freq;
            S.Filter_order=obj.Filter_order;
            S.Baudrate=obj.Baudrate;
            S.Offset=obj.Offset;
        end
    end
end

```

```

function main_function()
    %% creating an object by calling a predefined class
    myclass = Get_data_class2;
    Contents(myclass)
end
function Contents(myclass,main_fig)
    %% creating the main uifigure(window), in which the whole app will be
    displayed
    if (~exist('main_fig','var'))
        main_fig=uifigure();
        main_fig.WindowState='maximize';
    end
    main_fig.MenuBar='none';
    main_fig.NumberTitle='off';
    main_fig.Name='ELF Signal Analyzer';
    main_fig.Resize='on';
    main_fig.Color = "#86b2b5";
    main_fig.AutoResizeChildren='off';
    main_fig.CloseRequestFcn=@Close,myclass,main_fig};
    %% creating a grid just so I can make the Tabs take on the full window
    pause(2); %% Delay so everything loads before opening the window
    grid=uigridlayout(main_fig,[1 1]);
    grid.BackgroundColor=[0.8 0.8 0.8];
    grid.RowHeight={'1x'};
    grid.ColumnWidth={'1x'};
    grid.RowSpacing=0;
    grid.Padding=[0 0 0 0];
    %% setting the autoresize callback function ↓
    main_fig.SizeChangedFcn=@Resize,myclass};
    %% creating a tab group and assigning some of its attributes
    tg = uitabgroup(grid);
    tg.TabLocation='top';
    tg.Units='normalized';
    %% setting the 3 tabs and their attributes
    tab1 = uitab(tg,"Title","Sampling");
    tab1.BackgroundColor= '#86b2b5';
    tab1.ForegroundColor=[0 0 0];
    tab1.AutoResizeChildren='off';

    tab2 = uitab(tg,"Title","Filtering");
    tab2.BackgroundColor= "#86b2b5";
    tab2.ForegroundColor=[0 0 0];
    tab2.Scrollable='on';

    tab3 = uitab(tg,"Title","Comparison");
    tab3.BackgroundColor= "#86b2b5";
    tab3.ForegroundColor=[0 0 0];
    tab3.Scrollable='on';
    %% creating a menu to view different key variables in this confined space
    m1 = uimenu('Parent',main_fig,'Text','File');
    m11= uimenu(m1,'Text','Save');
    uimenu(m11,'Text','Matlab File','Tooltip','As a .mat
file','Accelerator','M','MenuSelectedFcn',{@SaveAsMat,myclass,main_fig});
    uimenu(m11,'Text','Excel
File','Separator','on','Accelerator','E','MenuSelectedFcn',{@SaveAsExcel,my
class,main_fig})
    uimenu(m11,'Text','Text
File','Separator','on','Accelerator','T','MenuSelectedFcn',{@SaveAsText,myc
lass,main_fig})

```

Appendix-MATLAB CODE

```
    uimenu(m1,'Text','Load','Separator','on','Tooltip',"Load previous
files",'Accelerator','L','MenuSelectedFcn',{@Load_func,myclass,main_fig})

    m2 = uimenu('Parent',main_fig,'Text','&View Variables');
    m2.MenuSelectedFcn=@View_Var,myclass,main_fig};
    m2.Accelerator='K';

    m4= uimenu('Parent',main_fig,'Text','Tools');
    uimenu(m4,'Text','Start
Sampling','Accelerator','B','MenuSelectedFcn',{@checkifavailable,myclass,ma
in_fig});
    uimenu(m4,'Text','Reset','Tooltip',"Reset all
data",'Separator','on','Accelerator','R','MenuSelectedFcn',{@Reset,myclass,
main_fig})
    uimenu(m4,'Text','Filter','Tooltip',"Filter the input
signal",'Separator','on','Accelerator','G','MenuSelectedFcn',{@plot_tab2,ma
in_fig,myclass})

    uimenu('Parent',main_fig,'Text','&Filter
Specifications','Tooltip',"View and change filter
specifications",'MenuSelectedFcn',{@Filter_Spec_Window,myclass});

    %% now im calling the tab1_function, tab2functions and tab3functions, so
Everything is loaded as soon as the user opens the app
    tab1_function(tab1,myclass);
    tab2_function(tab2,myclass);
    tab3_function(tab3,myclass);
end
function tab1_function(tab1,myclass)
    %% main window properties
    mw1=tab1; %mw1=main window
    %% creating a grid inside the tab1 window
    gr1=uigridlayout(mw1,[4 1]); % 2x1 grid (rows x columns)
    gr1.Padding=[0 0 0 0]; % [left bottom right top]
    gr1.BackgroundColor='#86b2b5';
    gr1.RowHeight={'fit','fit','fit','fit'};
    gr1.Scrollable='on';
    gr12=uigridlayout(gr1,[1 5]);
    gr12.BackgroundColor='#86b2b5';
    gr12.RowHeight={'0.5x'};
    gr12.ColumnWidth={'1x','1x','1x','1x','fit'};
    gr12.ColumnSpacing=15;
    gr12.Layout.Row=1;
    gr12.Padding=[10 0 10 2];% [left bottom right top]
    %% creating an edifield so the port is Displayed
    edt_tab1=uieditfield(gr12);
    edt_tab1.Layout.Row=1;
    edt_tab1.Layout.Column=5;
    edt_tab1.HorizontalAlignment='center';
    edt_tab1.Placeholder='No port connected';
    edt_tab1.Editable='off';
    edt_tab1.BackgroundColor="#f0c7a3";
    edt_tab1.Tooltip='This field displays the current port to which the app
is connected.';
    myclass.tab1_edt_field=edt_tab1;
    %% setting my plot attributes
    ax1=uiaxes('Parent',gr1);
    ax2=uiaxes('Parent',gr1);
    ax3=uiaxes('Parent',gr1);

    ax1.Layout.Row=2;
```

Appendix-MATLAB CODE

```
ax2.Layout.Row=3;
ax3.Layout.Row=4;

ax1.Title.String='Input Signal';
ax1.XLabel.String='Sample size';
ax1.YLabel.String='Magnitude';
ax1.XGrid='on';
ax1.YGrid='on';
ax1.Color=[0.1 0.1 0.1];
ax1.GridColor='#00FFFF';
ax1.GridAlpha=0.22;
ax1.TickDir='out';

ax2.Title.String='Input Signal';
ax2.XLabel.String='Time(sec)';
ax2.YLabel.String='Magnitude';
ax2.XGrid='on';
ax2.YGrid='on';
ax2.Color=[0.1 0.1 0.1];
ax2.GridColor='#00FFFF';
ax2.GridAlpha=0.22;
ax2.TickDir='out';

ax3.Title.String= 'Fourier Transform of Input Signal';
ax3.XLabel.String='Frequency(Hz)';
ax3.YLabel.String='Magnitude';
ax3.XGrid='on';
ax3.YGrid='on';
ax3.Color=[ 0.1 0.1 0.1];
ax3.GridColor='#00FFFF';
ax3.GridAlpha=0.22;
ax3.TickDir='out';
% plotting in the axis if data exist (in case user presses "refresh"
and has already sampled data)
plot(ax1,myclass.rt_data,'- .','Color',[1.0000 0.4706
0.0902],"MarkerEdgeColor","k","MarkerSize",8);

plot(ax2,myclass.time_variable, myclass.rt_data,'- .','Color',[1.0000
0.4706 0.0902],"MarkerEdgeColor","k","MarkerSize",8); % plot marker
properties : https://www.mathworks.com/help/matlab/creating\_plots/specify-
line-and-marker-appearance-in-plots.html

plot(ax3,myclass.Hertz,myclass.Fourier_data,'- .','Color',[1.0000
0.4706 0.0902],"MarkerEdgeColor",[0 0 0],"MarkerSize",8)
% giving the axis a "global" handle
myclass.tab1_plots(1)=ax1;
myclass.tab1_plots(2)=ax2;
myclass.tab1_plots(3)=ax3;
end
function Resize(src,~,~)
% getting the full size of tab1
width=src.Position(3);
height=src.Position(4);
% resize grid to give it the full window space
children=get(src,'Children');
children(5).RowHeight= height;
children(5).ColumnWidth=width;
end
function View_Var(~,~,myclass,main_fig)
```

Appendix-MATLAB CODE

```
    if
        (isempty(myclass.rt_data)) && (isempty(myclass.time_variable)) && (isempty(myclass.Samples)) && (isempty(myclass.Fourier_data)) && (isempty(myclass.Hertz))
            beep;
            uialert(main_fig,{'No data found to view','First take samples from the Serial port device.'},'Error','Icon','error');
            return
        end
        if myclass.Var_window(1)==true
            return
        end
        myclass.Var_window(1)=true;
        %% creating a Table with the data and Variables
        extra_samples=zeros(1,length(myclass.rt_data));
        extra_samples(1)=myclass.Samples;
        extra_samples(2:end)=0/0;

        extra_Sample_Freq=zeros(1,length(myclass.rt_data));
        extra_Sample_Freq(1)=myclass.Sample_Freq;
        extra_Sample_Freq(2:end)=0/0;

        extra_Baudrate=zeros(1,length(myclass.rt_data));
        extra_Baudrate(1)=myclass.Baudrate;
        extra_Baudrate(2:end)=0/0;

        extra_Offset=zeros(1,length(myclass.rt_data));
        extra_Offset(1)=myclass.Offset;
        extra_Offset(2:end)=0/0;

        extra_Peaks_Input=zeros(1,length(myclass.rt_data));

        extra_Peaks_Input(1:length(myclass.Fourier_Peaks_Input))=myclass.Fourier_Peaks_Input;
        extra_Peaks_Input(length(myclass.Fourier_Peaks_Input)+1:end)=0/0;

        extra_Peaks_Pos_Input=zeros(1,length(myclass.rt_data));

        extra_Peaks_Pos_Input(1:length(myclass.Fourier_Peak_Pos_Input))=myclass.Fourier_Peak_Pos_Input;
        extra_Peaks_Pos_Input(length(myclass.Fourier_Peak_Pos_Input)+1:end)=0/0;

        extra_Peaks_Filtered=zeros(1,length(myclass.rt_data));

        extra_Peaks_Filtered(1:length(myclass.Fourier_Peaks_Filtered))=myclass.Fourier_Peaks_Filtered;
        extra_Peaks_Filtered(length(myclass.Fourier_Peaks_Filtered)+1:end)=0/0;

        extra_Peaks_Pos_Filtered=zeros(1,length(myclass.rt_data));

        extra_Peaks_Pos_Filtered(1:length(myclass.Fourier_Peak_Pos_Filtered))=myclass.Fourier_Peak_Pos_Filtered;
        extra_Peaks_Pos_Filtered(length(myclass.Fourier_Peak_Pos_Filtered)+1:end)=0/0;

        extra_Input_Fourier_data=zeros(1,length(myclass.rt_data));

        extra_Input_Fourier_data(1:length(myclass.Fourier_data))=myclass.Fourier_data(1:end);
        extra_Input_Fourier_data(length(myclass.Fourier_data)+1:end)=0/0;
```

Appendix-MATLAB CODE

```
extra_Input_Hertz=zeros(1,length(myclass.rt_data));
extra_Input_Hertz(1:length(myclass.Hertz))=myclass.Hertz(1:end);
extra_Input_Hertz(length(myclass.Hertz)+1:end)=0/0;

extra_Filtered_Fourier_data=zeros(1,length(myclass.rt_data));

extra_Filtered_Fourier_data(1:length(myclass.Filter_Fourier_data))=myclass.
Filter_Fourier_data(1:end);

extra_Filtered_Fourier_data(length(myclass.Filter_Fourier_data)+1:end)=0/0;

extra_Filtered_Hertz_data=zeros(1,length(myclass.rt_data));

extra_Filtered_Hertz_data(1:length(myclass.Filter_Hertz))=myclass.Filter_He
rtz(1:end);
extra_Filtered_Hertz_data(length(myclass.Filter_Hertz)+1:end)=0/0;

Filtered_data=zeros(1,length(myclass.rt_data));

extra_Filter_Cutoff_Ang_Freq=zeros(1,length(myclass.rt_data));

extra_Filter_order=zeros(1,length(myclass.rt_data));

if ~isempty(myclass.Filter_Cutoff_Ang_Freq) &&
~isempty(myclass.Filter_order) && ~isempty(myclass.Filtered_data)
    Filtered_data(1:end)=myclass.Filtered_data;

extra_Filter_Cutoff_Ang_Freq(1)=myclass.Filter_Cutoff_Ang_Freq/(2*pi);
extra_Filter_Cutoff_Ang_Freq(2:end)=0/0;

extra_Filter_order(1)=myclass.Filter_order;
extra_Filter_order(2:end)=0/0;
else
    Filtered_data(1:end)=0/0;
    extra_Filter_Cutoff_Ang_Freq(1:end)=0/0;
    extra_Filter_order(1:end)=0/0;
end
% saving all ↓
Vars=table(myclass.rt_data',myclass.time_variable',Filtered_data',...

extra_Input_Fourier_data',extra_Input_Hertz',extra_Filtered_Fourier_data',e
xtra_Filtered_Hertz_data',...

extra_Peaks_Input',extra_Peaks_Pos_Input',extra_Peaks_Filtered',extra_Peaks
_Pos_Filtered',...

extra_Sample_Freq',extra_samples',extra_Filter_Cutoff_Ang_Freq',extra_Filte
r_order',extra_Baudrate',extra_Offset');
% creating a uifigure and grid layout to show everything
Var_fig=uifigure('Name','Data and Key Variables');
myclass.Var_window(2)=Var_fig;
Var_fig.DeleteFcn=@Close_Window_Request,myclass};
Var_grid1=uigridlayout(Var_fig,[2 1]);
Var_grid1.RowHeight={'1x',50};
Var_grid2=uigridlayout(Var_grid1,[1 4]);
Var_grid2.Layout.Row=2;
Var_grid2.ColumnWidth={'1x','1x','1x','0.5x'};
% creating a uitable to show the data in a specific format
uit=uitable(Var_grid1);
```


Appendix-MATLAB CODE

```
    uit.ColumnName={'Input Signal','Time','Filtered Signal',...
        "Input Signal's Fourier Transform's Magnitude","Input Signal's
Fourier Transform's Frequency (Hz)",...
        "Filtered Signal's Fourier Transform's Magnitude","Filtered Signal's
Fourier Transform's Frequency (Hz)",...
        "Magnitude of Input Signal's Fourier Peaks","Position of Input
Signal's Fourier Peaks (Hz)",...
        "Magnitude of Filtered Signal's Peaks","Position of Filtered
Signal's Peaks (Hz)",...
        'Sampling Frequency','Number of Samples','Cutoff Frequency
(Hz)','Filter Order','Baudrate used','Signal's Voltage Offset (V)'};
    uit.Data=Vars;
    uit.Layout.Row=1;
    uit.RowName = 'numbered';
    %% creating a menu bar(so the user can save data from here too)
    export = uimenu('Parent',Var_fig,'Text','Exp&ort Data');
    uimenu(export,'Text','Matlab File','Tooltip','As a .mat
file','Accelerator','M','MenuSelectedFcn',{@SaveAsMat,myclass,main_fig});
    uimenu(export,'Text','Excel
File','Separator','on','Accelerator','E','MenuSelectedFcn',{@SaveAsExcel,my
class,main_fig})
    uimenu(export,'Text','Text
File','Separator','on','Accelerator','T','MenuSelectedFcn',{@SaveAsText,myc
lass,main_fig})
    %% creating a close button
    Cls_btn=uibutton(Var_grid2);
    Cls_btn.Layout.Column=4;
    Cls_btn.Text='Close';
    Cls_btn.ButtonPushedFcn=@(x,y)delete(Var_fig));
end
function Reset(~,~,myclass,main_fig)
    main_fig.Name='ELF Signal Analyzer';
    for k1=1:1:3
        c1=myclass.tab1_plots(k1);
        cla(c1);
    end
    % resetting them as empty variables
    myclass.rt_data=[];
    myclass.time_variable=[];
    myclass.board=[];
    myclass.Sample_Freq=[];
    myclass.Samples=[];
    myclass.Fourier_data=[];
    myclass.Hertz=[];
    myclass.Filtered_data=[];
    myclass.Filter_Fourier_data=[];
    myclass.Filter_Hertz=[];
    myclass.Fourier_Peaks_Input=[];
    myclass.Fourier_Peak_Pos_Input=[];
    myclass.Fourier_Peaks_Filtered=[];
    myclass.Fourier_Peak_Pos_Filtered=[];
    myclass.prog_bar=[];
    myclass.tab1_edt_field.Value='';
    myclass.tab2_edt_field.Value='';
    myclass.tab3_edt_field.Value='';
    myclass.Filter_Cutoff_Ang_Freq=[];
    myclass.Filter_order=[];
    myclass.Baudrate=[];
    myclass.Offset=[];

    for k2=1:1:3
```

Appendix-MATLAB CODE

```
        c2=myclass.tab2_plots(k2);
        cla(c2);
        c3=myclass.tab3_plots(k2);
        cla(c3);
    end
end
function Load_func(~,~,myclass,main_fig)
    [file,path] = uigetfile('.mat');

    if (isequal(file,0)) && (isequal(path,0))
        return
    end

    file_type=extractAfter(file,".");
    if strcmp(file_type,"mat")==0
        beep
        uialert(main_fig,{'You tried to load an incompatible save file.',...
            'Please ensure that the save files have a ".mat"
extension.'},'Error while loading data','CloseFcn',@(h,e) close(gcf));
        return
    end
    full_path=strcat(path,file);
    load(full_path,'S');
    if (isempty(S.rt_data))|| (isempty(S.time_variable))...
        || (isempty(S.Sample_Freq))|| (isempty(S.Samples))...
        || (isempty(S.Fourier_data))|| (isempty(S.Hertz))...

|| (isempty(S.Fourier_Peaks_Input))|| (isempty(S.Fourier_Peak_Pos_Input))

        beep
        uialert(main_fig,{'You tried to load an incompatible save file.',...
            'Please ensure that the save files came from this app.'},'Error
while loading data','CloseFcn',@(h,e) close(gcf));
        clear S;
        return
    end
    %% importing (in the current class instance) the data that are definitely
saved
    myclass.rt_data=S.rt_data;
    myclass.time_variable=S.time_variable;
    myclass.Samples=S.Samples;
    myclass.Fourier_data=S.Fourier_data;
    myclass.Hertz=S.Hertz;
    myclass.Sample_Freq=S.Sample_Freq;
    myclass.Fourier_Peaks_Input=S.Fourier_Peaks_Input;
    myclass.Fourier_Peak_Pos_Input=S.Fourier_Peak_Pos_Input;
    myclass.Baudrate=S.Baudrate;
    myclass.Offset=S.Offset;
    %% Changing the name of the window so the user can see the name of the
file they loaded while working on the app
    main_fig.Name=append('ELF Signal Analyzer',' - ',file);
    %% plots of tab1
    ax1=myclass.tab1_plots(1);
    plot(ax1,myclass.rt_data,'- .','Color',[1.0000 0.4706
0.0902],"MarkerEdgeColor","k","MarkerSize",8)
    ax2=myclass.tab1_plots(2);
    plot(ax2,myclass.time_variable,myclass.rt_data,'- .','Color',[1.0000
0.4706 0.0902],"MarkerEdgeColor","k","MarkerSize",8);
    ax3=myclass.tab1_plots(3);
    plot(ax3,myclass.Hertz,myclass.Fourier_data,'- .','Color',[1.0000
0.4706 0.0902],"MarkerEdgeColor",[0 0 0],"MarkerSize",8)
```

Appendix-MATLAB CODE

```
% checking if the rest of the class data were saved in the file we are
trying to load and if yes, importing them as well
if
(isempty(S.Filtered_data)) || (isempty(S.Filter_Fourier_data)) || (isempty(S.Fi
lter_Hertz))...

|| (isempty(S.Filter_Cutoff_Ang_Freq)) || (isempty(S.Filter_order))
return
end
%% importing (in the current class instance) the rest of the saved data
myclass.Filtered_data=S.Filtered_data;
myclass.Filter_Fourier_data=S.Filter_Fourier_data;
myclass.Filter_Hertz=S.Filter_Hertz;
myclass.Fourier_Peaks_Filtered=S.Fourier_Peaks_Filtered;
myclass.Fourier_Peak_Pos_Filtered=S.Fourier_Peak_Pos_Filtered;
myclass.Filter_Cutoff_Ang_Freq=S.Filter_Cutoff_Ang_Freq;
myclass.Filter_order=S.Filter_order;

%% plots of tab2
plot(myclass.tab2_plots(1),myclass.Filtered_data,'- .','Color',[1.0000
0.4706 0.0902],"MarkerEdgeColor","k","MarkerSize",8);

plot(myclass.tab2_plots(2),myclass.time_variable,myclass.Filtered_data,'-
.','Color',[1.0000 0.4706 0.0902],"MarkerEdgeColor","k","MarkerSize",8);

plot(myclass.tab2_plots(3),myclass.Filter_Hertz,myclass.Filter_Fourier_data
,'- .','Color',[1.0000 0.4706 0.0902],"MarkerEdgeColor","k","MarkerSize",8);
%% plots of tab3
plot(myclass.tab3_plots(1),myclass.rt_data,'Color',[1.0000 0.4706
0.0902],"MarkerEdgeColor","k","MarkerSize",8)
hold(myclass.tab3_plots(1),'on');% 'hold on' but for uiaxes
plot(myclass.tab3_plots(1),myclass.Filtered_data,'Color',[0 0.4470
0.7410],"MarkerEdgeColor","k","MarkerSize",8)
legend(myclass.tab3_plots(1),{'Input Signal','Filtered
Signal'},'EdgeColor','white','TextColor','white');
hold(myclass.tab3_plots(1),'off');

plot(myclass.tab3_plots(2),myclass.time_variable,myclass.rt_data,'Color',[1
.0000 0.4706 0.0902],"MarkerEdgeColor","k","MarkerSize",8)
hold(myclass.tab3_plots(2),'on');% 'hold on' but for uiaxes

plot(myclass.tab3_plots(2),myclass.time_variable,myclass.Filtered_data,'Col
or',[0 0.4470 0.7410],"MarkerEdgeColor","k","MarkerSize",8)
legend(myclass.tab3_plots(2),{'Input Signal','Filtered
Signal'},'EdgeColor','white','TextColor','white');
hold(myclass.tab3_plots(2),'off');

plot(myclass.tab3_plots(3),myclass.Hertz,myclass.Fourier_data,'Color',[1.00
00 0.4706 0.0902],"MarkerEdgeColor","k","MarkerSize",8)
hold(myclass.tab3_plots(3),'on');% 'hold on' but for uiaxes

plot(myclass.tab3_plots(3),myclass.Filter_Hertz,myclass.Filter_Fourier_data
,'Color',[0 0.4470 0.7410],"MarkerEdgeColor","k","MarkerSize",8)
legend(myclass.tab3_plots(3),{'Input Signal','Filtered
Signal'},'EdgeColor','white','TextColor','white');
hold(myclass.tab3_plots(3),'off');
```

Appendix-MATLAB CODE

```
end
function Close(~,~,myclass,main_fig)
    msg='You are about to close the program';
    title='Confirm Exit';
    selection=uiconfirm(main_fig,msg,title,'Icon','warning',...
        'Options',{'Save and Exit','Don't Save','Cancel'},...
        'DefaultOption',1,'CancelOption',3);
    switch selection
        case 'Save and Exit'
            if
                (~isempty(myclass.rt_data)) && (~isempty(myclass.time_variable))...
                && (~isempty(myclass.Samples)) && (~isempty(myclass.Sample_Freq))...
                && (~isempty(myclass.Fourier_data)) && (~isempty(myclass.Hertz))...
                && (~isempty(myclass.Fourier_Peaks_Input)) && (~isempty(myclass.Fourier_Peak_Pos_Input))...
                && (~isempty(myclass.tab1_edt_field)) && (~isempty(myclass.tab2_edt_field)) && (~isempty(myclass.tab3_edt_field))
                    if (isempty(myclass.Filtered_data))...
                        && (isempty(myclass.Filter_Fourier_data)) && (isempty(myclass.Filter_Hertz))...
                        .
                        && (isempty(myclass.Fourier_Peaks_Filtered)) && (isempty(myclass.Fourier_Peak_Pos_Filtered))...
                        && (isempty(myclass.Filter_Cutoff_Ang_Freq)) && (isempty(myclass.Filter_order)
                    )
                        beep;
                        uialert(main_fig,{'You are only saving the data of the
Input signal.','If you filter first, you will save additional data about the
filter and the filtered signal.'},...
                            'Warning','Icon','info','CloseFcn',{@(x,y)
uiresume(main_fig)})
                        uiwait(main_fig);
                        end
                        %% creating a dialog box for the user
                        defaultFileName=fullfile(pwd,'*.mat');
                        [baseFileName, folder] = uiputfile(defaultFileName, 'Specify
a file');
                        if (baseFileName == 0)
                            return;
                        end
                        [~, baseFileNameNoExt, ~] = fileparts(baseFileName);
                        fullFileName = fullfile(folder, [baseFileNameNoExt,
'.mat']);
                        S=myclass.saveobj();
                        save(fullFileName,'S');
                        delete(findall(0));
                    else
                        beep;
                        uialert(main_fig,{'No data found to save','First take
samples from the Serial port device.'},'Error','Icon','error');
                        return
                    end
                case "Don't Save"
                    delete(findall(0));
            end
    end
```

Appendix-MATLAB CODE

```
end
function SaveAsMat(~,~,myclass,main_fig)
    if (~isempty(myclass.rt_data)) && (~isempty(myclass.time_variable)) ...
        && (~isempty(myclass.Baudrate)) && (~isempty(myclass.Offset)) ...

&& (~isempty(myclass.Fourier_data)) && (~isempty(myclass.Hertz)) ...

&& (~isempty(myclass.Fourier_Peaks_Input)) && (~isempty(myclass.Fourier_Peak_Pos_Input)) ...

&& (~isempty(myclass.tab1_edt_field)) && (~isempty(myclass.tab2_edt_field)) && (~isempty(myclass.tab3_edt_field))
    if (isempty(myclass.Filtered_data)) ...

&& (isempty(myclass.Filter_Fourier_data)) && (isempty(myclass.Filter_Hertz)) ..
.

&& (isempty(myclass.Filter_Cutoff_Ang_Freq)) && (isempty(myclass.Filter_order)
)
    beep;
    uialert(main_fig,{'You are only saving the data of the Input signal.', 'If you filter first, you will save additional data about the filter and the filtered signal.'},...
        'Warning','Icon','info','CloseFcn',{@(x,y)
uiresume(main_fig)}})
    uiwait(main_fig);
end
%% creating a dialog box for the user
defaultFileName=fullfile(pwd,'*.mat');
[baseFileName, folder] = uinputfile(defaultFileName, 'Specify a file');
if (baseFileName == 0)
    return;
end
[~, baseFileNameNoExt, ~] = fileparts(baseFileName);
fullFileName = fullfile(folder, [baseFileNameNoExt, '.mat']);
S=myclass.saveobj();
save(fullFileName,'S');
return
else
    beep;
    uialert(main_fig,{'No data found to save','First take samples from the Serial port device.'},'Error','Icon','error');
    return
end
end
function SaveAsExcel(~,~,myclass,main_fig)
    if (~isempty(myclass.rt_data)) && (~isempty(myclass.time_variable)) ...

&& (~isempty(myclass.Samples)) && (~isempty(myclass.Sample_Freq)) ...
        && (~isempty(myclass.Baudrate)) && (~isempty(myclass.Offset)) ...

&& (~isempty(myclass.Fourier_data)) && (~isempty(myclass.Hertz)) ...

&& (~isempty(myclass.Fourier_Peaks_Input)) && (~isempty(myclass.Fourier_Peak_Pos_Input)) ...

&& (~isempty(myclass.tab1_edt_field)) && (~isempty(myclass.tab2_edt_field)) && (~isempty(myclass.tab3_edt_field))

    old_rt_data = myclass.rt_data;
```

Appendix-MATLAB CODE

```
old_time=myclass.time_variable;

old_samples=zeros(1,length(myclass.rt_data));
old_samples(1)=myclass.Samples;
old_samples(2:end)=0/0;

old_Sample_Freq=zeros(1,length(myclass.rt_data));
old_Sample_Freq(1)=myclass.Sample_Freq;
old_Sample_Freq(2:end)=0/0;

old_Baudrate=zeros(1,length(myclass.rt_data));
old_Baudrate(1)=myclass.Baudrate;
old_Baudrate(2:end)=0/0;

old_Offset=zeros(1,length(myclass.rt_data));
old_Offset(1)=myclass.Offset;
old_Offset(2:end)=0/0;

old_Peaks_Input=zeros(1,length(myclass.rt_data));

old_Peaks_Input(1:length(myclass.Fourier_Peaks_Input))=myclass.Fourier_Peaks_Input;
old_Peaks_Input(length(myclass.Fourier_Peaks_Input)+1:end)=0/0;

old_Peaks_Pos_Input=zeros(1,length(myclass.rt_data));

old_Peaks_Pos_Input(1:length(myclass.Fourier_Peak_Pos_Input))=myclass.Fourier_Peak_Pos_Input;

old_Peaks_Pos_Input(length(myclass.Fourier_Peak_Pos_Input)+1:end)=0/0;

old_Input_Fourier_data=zeros(1,length(myclass.rt_data));

old_Input_Fourier_data(1:length(myclass.Fourier_data))=myclass.Fourier_data(1:end);
old_Input_Fourier_data(length(myclass.Fourier_data)+1:end)=0/0;

old_Input_Hertz=zeros(1,length(myclass.rt_data));
old_Input_Hertz(1:length(myclass.Hertz))=myclass.Hertz(1:end);
old_Input_Hertz(length(myclass.Hertz)+1:end)=0/0;

% creating a Tables with the data and Variable Names

T1=table(old_rt_data,'VariableNames',{'Input Signal'});
T2=table(old_time,'VariableNames',{'Time Variable (sec)'});
T3=table(old_Input_Fourier_data,'VariableNames',{'Magnitude of the Fourier Transform of Input Signal'});
T4=table(old_Input_Hertz,'VariableNames',{'Hertz of the Fourier Transform of Input Signal'});
T5=table(old_samples,'VariableNames',{'Number of samples'});
T6=table(old_Sample_Freq,'VariableNames',{'Sampling Frequency (Hz)'});
T7=table(old_Baudrate,'VariableNames',{'Baudrate used'});
T8=table(old_Offset,'VariableNames',{'Voltage Offset of Input Signal (V)'});
T9=table(old_Peaks_Input,'VariableNames',"Input Signal's Fourier Transform's Peaks' Magnitude");
T10=table(old_Peaks_Pos_Input,'VariableNames',"Input Signal's Fourier Transform's Peaks' Position (Hz)");

if (isempty(myclass.Filtered_data))...
```

Appendix-MATLAB CODE

```
&&(isempty(myclass.Filter_Fourier_data)) &&(isempty(myclass.Filter_Hertz)) ..
.

&&(isempty(myclass.Fourier_Peaks_Filtered)) &&(isempty(myclass.Fourier_Peak_
Pos_Filtered)) ...

&&(isempty(myclass.Filter_Cutoff_Ang_Freq)) &&(isempty(myclass.Filter_order
)
    beep;
    uialert(main_fig,{'You are only saving the data of the Input
signal.','If you filter first, you will save additional data about the filter
and the filtered signal.'},...
    'Warning','Icon','info','CloseFcn',{@(x,y)
uiresume(main_fig)})
    uiwait(main_fig);

    T_all=[T1,T2,T3,T4,T5,T6,T7,T8,T9,T10];
    elseif (~isempty(myclass.Filtered_data)) ...

&&(~isempty(myclass.Filter_Fourier_data)) &&(~isempty(myclass.Filter_Hertz))
...

&&(~isempty(myclass.Filter_Cutoff_Ang_Freq)) &&(~isempty(myclass.Filter_orde
r))

    old_Filtered_data=myclass.Filtered_data;

    old_Peaks_Filtered=zeros(1,length(myclass.rt_data));

old_Peaks_Filtered(1:length(myclass.Fourier_Peaks_Filtered))=myclass.Fourie
r_Peaks_Filtered;

old_Peaks_Filtered(length(myclass.Fourier_Peaks_Filtered)+1:end)=0/0;

    old_Peaks_Pos_Filtered=zeros(1,length(myclass.rt_data));

old_Peaks_Pos_Filtered(1:length(myclass.Fourier_Peak_Pos_Filtered))=myclass
.Fourier_Peak_Pos_Filtered;

old_Peaks_Pos_Filtered(length(myclass.Fourier_Peak_Pos_Filtered)+1:end)=0/0
;

    old_Filtered_Fourier_data=zeros(1,length(myclass.rt_data));

old_Filtered_Fourier_data(1:length(myclass.Filter_Fourier_data))=myclass.Fi
lter_Fourier_data(1:end);

old_Filtered_Fourier_data(length(myclass.Filter_Fourier_data)+1:end)=0/0;

    old_Filtered_Hertz_data=zeros(1,length(myclass.rt_data));

old_Filtered_Hertz_data(1:length(myclass.Filter_Hertz))=myclass.Filter_Hert
z(1:end);

old_Filtered_Hertz_data(length(myclass.Filter_Hertz)+1:end)=0/0;

    old_Filter_Cutoff_Ang_Freq=zeros(1,length(myclass.rt_data));

old_Filter_Cutoff_Ang_Freq(1)=myclass.Filter_Cutoff_Ang_Freq/(2*pi);
    old_Filter_Cutoff_Ang_Freq(2:end)=0/0;
```

Appendix-MATLAB CODE

```
old_Filter_order=zeros(1,length(myclass.rt_data));
old_Filter_order(1)=myclass.Filter_order;
old_Filter_order(2:end)=0/0;

T1 =table(old_rt_data','VariableNames',{'Input Signal'});
T2 =table(old_time','VariableNames',{'Time Variable (sec)'});
T3      =table(old_Filtered_data','VariableNames',{'Filtered
Signal'});
T4      =table(old_Input_Fourier_data','VariableNames',{'Magnitude
of the Fourier Transform of Input Signal'});
T5      =table(old_Input_Hertz','VariableNames',{'Frequency of the
Fourier Transform of Input Signal (Hz)'});
T6      =table(old_Peaks_Input','VariableNames',"Input Signal's
Fourier Transform's Peaks' Magnitude");
T7      =table(old_Peaks_Pos_Input','VariableNames',"Input Signal's
Fourier Transform's Peaks' Position (Hz)");
T8=table(old_Filtered_Fourier_data','VariableNames',{'Magnitude
of the Fourier Transform of Filtered Signal'});
T9=table(old_Filtered_Hertz_data','VariableNames',{'Frequency
of the Fourier Transform of Filtered Signal (Hz)'});
T10=table(old_Peaks_Filtered','VariableNames',"Filtered
Signal's Fourier Transform's Peaks' Magnitude");
T11=table(old_Peaks_Pos_Filtered','VariableNames',"Filtered
Signal's Fourier Transform's Peaks' Position (Hz)");
T12=table(old_Filter_Cutoff_Ang_Freq','VariableNames',{'Cutoff
Frequency of the Filter (Hz)'});
T13=table(old_Filter_order','VariableNames',{'Filter Order'});
T14=table(old_samples','VariableNames',{'Number of samples'});
T15=table(old_Sample_Freq','VariableNames',{'Sampling
Frequency'});
T16=table(old_Baudrate','VariableNames',{'Baudrate used'});
T17=table(old_Offset','VariableNames',{'Voltage Offset of Input
Signal (V)'});

T_all=[T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17];
end
%-----creating a dialog box-----
-----
defaultFileName=fullfile(pwd,'*.xlsx');
[baseFileName, folder] = uiputfile(defaultFileName, 'Specify a
file');
if (baseFileName == 0)
    return;
end
[~,baseFileNameNoExt,~] = fileparts(baseFileName);
fullFileName = fullfile(folder, [baseFileNameNoExt, '.xlsx']);
writetable(T_all,fullFileName);
return
else
    beep;
    uialert(main_fig,{'No data found to save','First take samples from
the Serial port device.'},'Error','Icon','error');
    return
end
end
function SaveAsText(~,~,myclass,main_fig)
    if(~isempty(myclass.rt_data))&&(~isempty(myclass.time_variable))...
&&(~isempty(myclass.Samples))&&(~isempty(myclass.Sample_Freq))...
```


Appendix-MATLAB CODE

```

    &&(~isempty(myclass.Baudrate)) &&(~isempty(myclass.Offset)) ...

&&(~isempty(myclass.Fourier_data)) &&(~isempty(myclass.Hertz)) ...

&&(~isempty(myclass.Fourier_Peaks_Input)) &&(~isempty(myclass.Fourier_Peak_Pos_Input)) ...

&&(~isempty(myclass.tab1_edt_field)) &&(~isempty(myclass.tab2_edt_field)) &&(~isempty(myclass.tab3_edt_field))

    Fourier_Peaks_Input=zeros(1,length(myclass.rt_data));

Fourier_Peaks_Input(1:(length(myclass.Fourier_Peaks_Input)))=myclass.Fourier_Peaks_Input(1:end);
    Fourier_Peaks_Input(length(myclass.Fourier_Peaks_Input)+1:end)=' ';

    Fourier_Peak_Pos_Input=zeros(1,length(myclass.rt_data));

Fourier_Peak_Pos_Input(1:(length(myclass.Fourier_Peak_Pos_Input)))=myclass.Fourier_Peak_Pos_Input;
Fourier_Peak_Pos_Input(length(myclass.Fourier_Peak_Pos_Input)+1:end)=' ';

    Samples=zeros(1,length(myclass.rt_data));
    Samples(1)=myclass.Samples;
    Samples(2:end)=' ';

    Sample_Freq=zeros(1,length(myclass.rt_data));
    Sample_Freq(1)=myclass.Sample_Freq;
    Sample_Freq(2:end)=' ';

    Baudrate=zeros(1,length(myclass.rt_data));
    Baudrate(1)=myclass.Baudrate;
    Baudrate(2:end)=' ';

    Offset=zeros(1,length(myclass.rt_data));
    Offset(1)=myclass.Offset;
    Offset(2:end)=' ';

    Fourier_data=zeros(1,length(myclass.rt_data));
    Fourier_data(1:length(myclass.Fourier_data))=myclass.Fourier_data;
    Fourier_data(length(myclass.Fourier_data)+1:end)=' ';

    Hertz=zeros(1,length(myclass.rt_data));
    Hertz(1:length(myclass.Hertz))=myclass.Hertz;
    Hertz(length(myclass.Hertz)+1:end)=' ';

    txt_data=compose('%.5f',myclass.rt_data);

    txt_time=compose('%.3f',myclass.time_variable);

    txt_samples=compose('%c',Samples);
    txt_samples(1)=compose('%d',Samples(1));

    txt_Fourier_data=compose('%.14f',Fourier_data);

txt_Fourier_data(length(myclass.Fourier_data)+1:end)=compose('%c',Fourier_data(length(myclass.Fourier_data)+1:end));

    txt_Hertz=compose('%f',Hertz);
```

Appendix-MATLAB CODE

```
txt_Hertz(length(myclass.Hertz)+1:end)=compose('%c',Hertz(length(myclass.He
rtz)+1:end));

    txt_Sample_Freq=compose('%c',Sample_Freq);
    txt_Sample_Freq(1)=compose('%d',Sample_Freq(1));

    txt_Baudrate=compose('%c',Baudrate);
    txt_Baudrate(1)=compose('%d',Baudrate(1));

    txt_Offset=compose('%c',Offset);
    txt_Offset(1)=compose('%d',Offset(1));

    txt_Fourier_Peaks_Input=compose('%c',Fourier_Peaks_Input);

txt_Fourier_Peaks_Input(1:(length(myclass.Fourier_Peaks_Input)))=compose('%
d',Fourier_Peaks_Input(1:(length(myclass.Fourier_Peaks_Input))));

    txt_Fourier_Peak_Pos_Input=compose('%c',Fourier_Peak_Pos_Input);

txt_Fourier_Peak_Pos_Input(1:(length(myclass.Fourier_Peak_Pos_Input)))=comp
ose('%d',Fourier_Peak_Pos_Input(1:(length(myclass.Fourier_Peak_Pos_Input)
)));

    T1=table(txt_data,'VariableNames','|Input Signal|');
    T2=table(txt_time,'VariableNames','|Time(sec)|');
    T3=table(txt_Fourier_data,'VariableNames','|Fourier          Magnitude
(Input Signal)|');
    T4=table(txt_Hertz,'VariableNames','|Fourier Frequency - Hz (Input
Signal)|');
    T5=table(txt_samples,'VariableNames','|Number of Samples|');
    T6=table(txt_Sample_Freq,'VariableNames','|Sampling Frequency|');
    T7=table(txt_Baudrate,'VariableNames','|Baudrate|');
    T8=table(txt_Offset,'VariableNames','|Offset Value(V)|');
    T9=table(txt_Fourier_Peaks_Input,'VariableNames','|Magnitude
Fourier Transform's Peaks (Input Signal)|');
    T10=table(txt_Fourier_Peak_Pos_Input,'VariableNames','|Position
Fourier Transform's Peaks (Hz) (Input Signal)|');
    if (isempty(myclass.Filtered_data))...

    &&(isempty(myclass.Filter_Fourier_data))&&(isempty(myclass.Filter_Hertz))..
.
    &&(isempty(myclass.Filter_Cutoff_Ang_Freq))&&(isempty(myclass.Filter_order)
)
        T_all=[T1 T2 T3 T4 T5 T6 T7 T8 T9 T10];
    else % if they saved the filtered data:
        Fourier_Peaks_Filtered=zeros(1,length(myclass.rt_data));

Fourier_Peaks_Filtered(1:(length(myclass.Fourier_Peaks_Filtered)))=myclass.
Fourier_Peaks_Filtered(1:end);

Fourier_Peaks_Filtered(length(myclass.Fourier_Peaks_Filtered)+1:end)=' ';

        Fourier_Peak_Pos_Filtered=zeros(1,length(myclass.rt_data));

Fourier_Peak_Pos_Filtered(1:(length(myclass.Fourier_Peak_Pos_Filtered)))=my
class.Fourier_Peak_Pos_Filtered;

Fourier_Peak_Pos_Filtered(length(myclass.Fourier_Peak_Pos_Filtered)+1:end)=
' ';
```

Appendix-MATLAB CODE

```
Filter_Cutoff_Ang_Freq=zeros(1,length(myclass.rt_data));
Filter_Cutoff_Ang_Freq(1)=myclass.Filter_Cutoff_Ang_Freq;
Filter_Cutoff_Ang_Freq(2:end)=' ';

Filter_order=zeros(1,length(myclass.rt_data));
Filter_order(1)=myclass.Filter_order;
Filter_order(2:end)=' ';

Filter_Fourier_data=zeros(1,length(myclass.rt_data));

Filter_Fourier_data(1:length(myclass.Filter_Fourier_data))=myclass.Filter_F
ourier_data;

Filter_Fourier_data(length(myclass.Filter_Fourier_data)+1:end)=' ';

Filter_Hertz=zeros(1,length(myclass.rt_data));

Filter_Hertz(1:length(myclass.Filter_Hertz))=myclass.Filter_Hertz;
Filter_Hertz(length(myclass.Filter_Hertz)+1:end)=' ';

txt_Filtered_data=compose('%.5f',myclass.Filtered_data);

txt_Filter_Cutoff_Ang_Freq=compose('%c',Filter_Cutoff_Ang_Freq);
txt_Filter_Cutoff_Ang_Freq(1)=compose('%d',Filter_Cutoff_Ang_Freq(1));

txt_Filter_Fourier_data=compose('%.14f',Filter_Fourier_data);

txt_Filter_Fourier_data(length(myclass.Filter_Fourier_data)+1:end)=compose(
'%c',Filter_Fourier_data(length(myclass.Filter_Fourier_data)+1:end));

txt_Filter_Hertz=compose('%f',Filter_Hertz);

txt_Filter_Hertz(length(myclass.Filter_Hertz)+1:end)=compose('%c',Filter_He
rtz(length(myclass.Filter_Hertz)+1:end));

txt_Filter_order=compose('%c',Filter_order);
txt_Filter_order(1)=compose('%d',Filter_order(1)); ;

txt_Fourier_Peaks_Filtered=compose('%c',Fourier_Peaks_Filtered);

txt_Fourier_Peaks_Filtered(1:(length(myclass.Fourier_Peaks_Filtered)))=comp
ose('%d',Fourier_Peaks_Filtered(1:(length(myclass.Fourier_Peaks_Filtered))
));

txt_Fourier_Peak_Pos_Filtered=compose('%c',Fourier_Peak_Pos_Filtered);

txt_Fourier_Peak_Pos_Filtered(1:(length(myclass.Fourier_Peak_Pos_Filtered))
)=compose('%d',Fourier_Peak_Pos_Filtered(1:(length(myclass.Fourier_Peak_Pos
_Filtered))));

T11=table(txt_Filtered_data,'VariableNames','|Filtered
Signal|");
T12=table(txt_Filter_Fourier_data,'VariableNames','|Fourier
Magnitude (Filtered Signal)|");
T13=table(txt_Filter_Hertz,'VariableNames','|Fourier Frequency
- Hz (Filtered Signal)|");
```

Appendix-MATLAB CODE

```
T14=table(txt_Filter_Cutoff_Ang_Freq','VariableNames','|Filter's Cutoff  
Frequency (Hz)|");  
T15=table(txt_Filter_order','VariableNames','|Filter Order|");  
  
T16=table(txt_Fourier_Peaks_Filtered','VariableNames','|Magnitude Fourier  
Transform's Peaks (Filtered Signal)|");  
  
T17=table(txt_Fourier_Peak_Pos_Filtered','VariableNames','|Position Fourier  
Transform's Peaks (Hz) (Filtered Signal)|");  
T_all=[T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15 T16  
T17];  
end  
%-----creating a dialog box-----  
defaultFileName=fullfile(pwd,'*.txt');  
[baseFileName, folder] = uiputfile(defaultFileName, 'Specify a  
file');  
if (baseFileName == 0)  
return;  
end  
[~, baseFileNameNoExt, ~] = fileparts(baseFileName);  
fullFileName = fullfile(folder, [baseFileNameNoExt, '.txt']);  
writetable(T_all,fullFileName,'Delimiter','tab');  
return  
else  
beep;  
uialert(main_fig,{'No data found to save','First take samples from  
the Serial port device.'},'Error','Icon','error');  
return  
end  
end  
function checkifavailable(~,~,myclass,main_fig)  
% setting and configuring the port  
[port_num] = PortCheck(main_fig);  
if port_num=="None"  
return  
end  
  
myclass.rt_data=[];  
myclass.time_variable=[];  
myclass.board=[];  
myclass.Sample_Freq=[];  
myclass.Samples=[];  
myclass.Fourier_data=[];  
myclass.Hertz=[];  
myclass.Filtered_data=[];  
myclass.Filter_Fourier_data=[];  
myclass.Filter_Hertz=[];  
myclass.Fourier_Peaks_Input=[];  
myclass.Fourier_Peak_Pos_Input=[];  
myclass.Fourier_Peaks_Filtered=[];  
myclass.Fourier_Peak_Pos_Filtered=[];  
myclass.prog_bar=[];  
myclass.Filter_Cutoff_Ang_Freq=[];  
myclass.Filter_order=[];  
myclass.Baudrate=[];  
myclass.Offset=[];  
for k2=1:1:3  
c2=myclass.tab2_plots(k2);  
cla(c2);  
c3=myclass.tab3_plots(k2);
```

Appendix-MATLAB CODE

```
        cla(c3);
    end

    try
        arduino = serialport(port_num,115200);
    catch ME
        if (strcmp(ME.identifier,'serialport:serialport:ConnectionFailed'))
            msg=append('Unable to connect to the serialport device at port
',string(port_num),' . Verify that the port is not in use by another program,
and that a device is properly connected to it. ');
        end
        beep;
        uialert(main_fig,msg,'Error','Icon','error');
        return
    end
    arduino.StopBits=1; % not necessary
    configureTerminator(arduino,"CR/LF");
    flush(arduino);
    myclass.board=arduino;
    myclass.tab1_edt_field.Value=arduino.Port;
    myclass.tab2_edt_field.Value=arduino.Port;
    myclass.tab3_edt_field.Value=arduino.Port;
    %% calling the dialog box for Sampling Freq + # of samples
    Input_Box(myclass,main_fig);
end
function [port_num] = PortCheck(main_fig)
    %% finding usb serial device connected to the computer
    NET.addAssembly("System.Management");
    mngmtQuery = System.Management.ObjectQuery();
    mngmtQuery.QueryString = "SELECT * FROM Win32_PnPEntity WHERE Name LIKE
'%'(COM%')";
    mngmtSearcher = System.Management.ManagementObjectSearcher(mngmtQuery);
    mngmtObjColl = mngmtSearcher.Get();
    comRep = repmat("",mngmtObjColl.Count,2);
    enMngmtObjColl = mngmtObjColl.GetEnumerator();
    p = 0;
    while enMngmtObjColl.MoveNext()
        p = p + 1;
        com = enMngmtObjColl.Current;
        caption = com.GetPropertyValue("Caption");
        comRep(p,1) = string(caption).extract("COM" + digitsPattern);
        comRep(p,2) = string(com.GetPropertyValue("Description"));
    end
    if isempty(comRep) % in case there are NO PORTS AVAILABLE at the time
        beep
        uialert(main_fig,"No available port found on your computer. First
connect your USB serial device.",'No device is found','CloseFcn',@(h,e)
close(gcf));
        port_num="None";
        return
    end
    [numRows,~] = size(comRep);
    for k=1:1:numRows
        lists(k)=join(comRep(k,:));
    end
    %% creating a list of com ports so the user can choose
    list=num2cell(lists(:));
    [indx,tf] =
listdlg('ListSize',[300,150],'ListString',list,'PromptString',"Select the
port to which your serial device is connected.",'SelectionMode','single');
    if tf==0 %if user clicked cancel or the x button
```

Appendix-MATLAB CODE

```

    port_num="None";
elseif tf==1
    Str=comRep(indx,2);
    newStr = erase(Str," ");
    newStr = erase(newStr,"-");
    newStr = lower( newStr );
    if ( contains(newStr,"usbserial")==1 ) || (
contains(newStr,"arduino")==1 )
        port_num=comRep(indx);
    elseif comRep(indx,2)~="USB Serial Device"
        beep
        uialert(main_fig,'Please choose a USB Serial Device.','No
compatible device is selected','CloseFcn',@(h,e) close(gcf));
        port_num="None";
    end
end
end
function Input_Box(myclass,main_fig)
    %% %-----Dialog Box-----
    prompt = {'Enter the appropriate Sampling Frequency (Hz):','Enter the
number of samples:','Enter the Voltage Offset Value (V):','Enter Baudrate:'};
    dlgtitle = 'Input';
    dims = [1 35];
    definput = {'1000','2500','2.5','115200'};
    opts.WindowStyle = 'modal';
    answers=inputdlg(prompt,dlgtitle,dims,definput,opts);
    %-----interpreting user's inputs-----
    if (isempty(answers)) %checks if the cancel button was pressed
        return
    elseif
(isempty(answers{1,1}))||(isempty(answers{2,1}))||(isempty(answers{3,1}))||
(isempty(answers{4,1})) %checks if one or more inputs were left empty
        beep
        uialert(main_fig,'You left one or more inputs blank. Please fill in
both before continuing.','Empty Inputs','CloseFcn',@(h,e) close(gcf));
        return
    elseif
(~isempty(answers{1,1}))&&(~isempty(answers{2,1}))&&(~isempty(answers{3,1}))
&&(~isempty(answers{4,1})) %checks if both inputs are filled
        inputs1=str2num(answers{1,1});
        inputs2=str2num(answers{2,1});
        inputs3=str2num(answers{3,1});
        inputs4=str2num(answers{4,1});
        if ( ~isempty(inputs1) ) && ( ~isempty(inputs2) ) && (
~isempty(inputs3) ) && ( ~isempty(inputs4) ) % checks if only numbers are
inserted by the user
            if ( class(inputs1)~="double" ) || ( class(inputs2)~="double" )
|| ( class(inputs3)~="double" ) || ( class(inputs4)~="double" ) % for example
if someone inserts a HEX number
                beep
                uialert(main_fig,'One or both inputs were not a correct type.
Please make sure to only insert real (R) numbers.','Incorrect
Inputs','CloseFcn',@(h,e) close(gcf));
                return
            end
            if (mod(inputs2,1)~=0)|| (mod(inputs4,1)~=0) % making sure the #
of samples input is an integer
                beep
                uialert(main_fig,'The Number of Samples and the Baudrate
should be integers.','Incorrect value of Number of Samples or
Baudrate','CloseFcn',@(h,e) close(gcf));
            end
        end
    end
end

```

Appendix-MATLAB CODE

```
        return
    end
    myclass.Sample_Freq=inputs1;
    myclass.Samples=inputs2;
    myclass.Offset=inputs3;
    myclass.Baudrate=inputs4;
    myclass.board.BaudRate=myclass.Baudrate;
    configureCallback(myclass.board,"terminator",@(obj,evt)
begin(myclass,main_fig))
    else
        beep
        uialert(main_fig,'One or both inputs were not numbers. Please
make sure to only insert numbers.','Incorrect Inputs','CloseFcn',@(h,e)
close(gcf));
    return
end
end
end
function begin(myclass,main_fig)
    configureCallback(myclass.board,"off"); % ↓
    t = mytimer(myclass,main_fig);%initialize the t variable cause it is used
on the next command
    start(t);
end
function t= mytimer(myclass,main_fig)
    t = timer;
    %timer's timing properties↓
    t.StartDelay=0;
    t.Period=0.001;
    t.TasksToExecute=myclass.Samples;
    t.BusyMode='queue';
    t.ExecutionMode='fixedSpacing';
    % callback function properties↓
    t.StartFcn = {@set_prog_bar,myclass,main_fig};
    t.TimerFcn = {@pull_data,myclass,t} ;
    t.StopFcn = {@plot_tab1,myclass};
    t.ErrorFcn = {@(x,y)disp('unspecified error in timer')};
end
function pull_data(~,~,myclass,t)
    progbar=myclass.prog_bar;
    myclass.rt_data(timerfind(t).TasksExecuted) =
str2double(readline(myclass.board)); %Read line of ASCII string data from
serial port
    progress=(timerfind(t).TasksExecuted/timerfind(t).TasksToExecute);
    switch progbar.CancelRequested
        case 0
            if(progress<0.8)
                progbar.Message='Loading your data';
                progbar.Value=progress;
            elseif (progress>0.8)
                progbar.Message='Finishing up';
                progbar.Value=progress;
            end
        case 1 % if user presses cancel: ↓
            myclass.rt_data=[];
            myclass.time_variable=[];
            myclass.Samples=[];
            myclass.Fourier_data=[];

            stop(t);
            return
    end
```

Appendix-MATLAB CODE

```
end
end
function set_prog_bar(~,~,myclass,main_fig)
    wait_window = uiprogresdlg(main_fig,'Title','Please Wait','Value',0);
    wait_window.ShowPercentage='on';
    wait_window.Cancelable='on';
    wait_window.CancelText='Cancel';
    myclass.prog_bar=wait_window;
end
function plot_tab1(~,~,myclass)

    delete(timerfindall); % deleting all timers

    close(myclass.prog_bar);
    myclass.prog_bar=[];

    if (~isempty(myclass.rt_data))&&(~isempty(myclass.Samples))
        ax1=myclass.tab1_plots(1);
        plot(ax1,myclass.rt_data,'- .','Color',[1.0000 0.4706 0.0902], "MarkerEdgeColor","k", "MarkerSize",8)
        ax2=myclass.tab1_plots(2);
        %-----evaluating time axis-----
        time=0:(1/myclass.Sample_Freq):myclass.Samples;
        myclass.time_variable=time(1:myclass.Samples);% taking as many
points as there are samples

        plot(ax2,myclass.time_variable, myclass.rt_data,'-
.', 'Color',[1.0000 0.4706 0.0902], "MarkerEdgeColor","k", "MarkerSize",8); %
plot marker properties :
https://www.mathworks.com/help/matlab/creating\_plots/specify-line-and-marker-appearance-in-plots.html

        [f_Hz, Y1]=fourier(myclass);

        ax3=myclass.tab1_plots(3);
        plot(ax3,f_Hz,Y1,'- .','Color',[1.0000 0.4706 0.0902], "MarkerEdgeColor",[0 0 0], "MarkerSize",8)
    end
    %% finding the maximum values of the fourier transform (and their pos
in Hz)

    if isempty(myclass.Fourier_data) || isempty(myclass.Hertz) % this is
incase user pressed cancel while sampling
        return
    end

    [pks,locs]=findpeaks(myclass.Fourier_data,myclass.Hertz,'MinPeakProminence'
,10);

    if myclass.Fourier_data(1)>=5*myclass.Fourier_data(2)
        pks=[myclass.Fourier_data(1) ,pks];
        [pks_sorted,index]=sort(pks,'descend');
        position = find(pks==myclass.Fourier_data(1));
        locs=[locs(1:position-1) myclass.Hertz(1) locs(position:end)];
        locs_sorted=locs(index);
    else
        [pks_sorted,index]=sort(pks,'descend');
        locs_sorted=locs(index);
    end
    myclass.Fourier_Peaks_Input=pks_sorted;
end
```


Appendix-MATLAB CODE

```
myclass.Fourier_Peak_Pos_Input=locs_sorted;
end
function [f_Hz, Y1] = fourier(myclass)
L=length( myclass.rt_data);
X_F=fft( myclass.rt_data);
X_mag=abs(X_F);
Y1=X_mag(1:round((L/2)));
f_bins=(0:(L-1)/2);
f_Hz = (myclass.Sample_Freq/L).*f_bins;
%% saving them as class properties so i can use them in different
functions
myclass.Fourier_data=Y1;
myclass.Hertz=f_Hz;
end
function tab2_function(tab2,myclass)
grid1_tab2=uigridlayout(tab2,[4 1]);
grid1_tab2.RowHeight={'fit','fit','fit','fit'};
grid1_tab2.BackgroundColor='#86b2b5';
grid1_tab2.Scrollable='on';
grid1_tab2.Padding= [0 0 0 0] ; % [left bottom right top]

grid2_tab2=uigridlayout(grid1_tab2,[1 5]);
grid2_tab2.Layout.Row=1;
grid2_tab2.BackgroundColor='#86b2b5';
grid2_tab2.ColumnWidth={'1x' '1x' '1x' '1x' 'fit'};
grid2_tab2.RowHeight={'0.5x'};
grid2_tab2.ColumnSpacing=15;
grid2_tab2.Padding=[10 0 10 2];% [left bottom right top]

%% creating an edifield so the port is Displayed
edt_tab2=uieditfield(grid2_tab2);
edt_tab2.Layout.Row=1;
edt_tab2.Layout.Column=5;
edt_tab2.HorizontalAlignment='center';
edt_tab2.Placeholder='No port connected';
edt_tab2.Editable='off';
edt_tab2.BackgroundColor="#f0c7a3";
edt_tab2.Tooltip='This field displays the current port to which the app
is connected.';
myclass.tab2_edt_field=edt_tab2; % so i can update the value as as a
port is connected
%% creating the axis where the plots are gonna be displayed in.
ax21=uiaxes('Parent',grid1_tab2);
ax21.Layout.Row=2; % επιλέγω την θέση στο grid (2x1)
ax21.Title.String='Filtered Signal';
ax21.XLabel.String='Sample size';
ax21.YLabel.String='Magnitude';
ax21.XGrid='on';
ax21.YGrid='on';
ax21.Color=[0.1 0.1 0.1];
ax21.GridColor='#00FFFF';
ax21.GridAlpha=0.22;
ax21.TickDir='out'; %direction of the Tick marks

ax22=uiaxes('Parent',grid1_tab2);
ax22.Layout.Row=3; % επιλέγω την θέση στο grid (2x1)
ax22.Title.String='Filtered Signal';
ax22.XLabel.String='Time (sec)';
ax22.YLabel.String='Magnitude';
ax22.XGrid='on';
ax22.YGrid='on';
```

Appendix-MATLAB CODE

```
ax22.Color=[0.1 0.1 0.1];
ax22.GridColor='#00FFFF';
ax22.GridAlpha=0.22;
ax22.TickDir='out'; %direction of the Tick marks

ax23=uiaxes('Parent',grid1_tab2);
ax23.Layout.Row=4; % επιλέγω την θέση στο grid (2x1)
ax23.Title.String='Fourier of the filtered signal';
ax23.XLabel.String='Frequency (Hz)';
ax23.YLabel.String='Magnitude';
ax23.XGrid='on';
ax23.YGrid='on';
ax23.Color=[0.1 0.1 0.1];
ax23.GridColor='#00FFFF';
ax23.GridAlpha=0.22;
ax23.TickDir='out'; %direction of the Tick marks
%% giving the axis a "global" handle
myclass.tab2_plots(1)=ax21;
myclass.tab2_plots(2)=ax22;
myclass.tab2_plots(3)=ax23;
end
function plot_tab2(~,~,main_fig,myclass)
    if
        (isempty(myclass.rt_data)) && (isempty(myclass.time_variable)) && (isempty(myclass.Samples)) && (isempty(myclass.Fourier_data)) && (isempty(myclass.Hertz))
            beep;
            uialert(main_fig,{'No data have been found','First take samples from the Serial port device.'},'Error','Icon','error');
            return %if the user hasnt sampled their signal, then nothing will be shown
        end

        filter(myclass);

        plot(myclass.tab2_plots(1),myclass.Filtered_data,'- .','Color',[1.0000 0.4706 0.0902],"MarkerEdgeColor","k","MarkerSize",8);

        plot(myclass.tab2_plots(2),myclass.time_variable,myclass.Filtered_data,'- .','Color',[1.0000 0.4706 0.0902],"MarkerEdgeColor","k","MarkerSize",8);

        plot(myclass.tab2_plots(3),myclass.Filter_Hertz,myclass.Filter_Fourier_data,'- .','Color',[1.0000 0.4706 0.0902],"MarkerEdgeColor","k","MarkerSize",8);

        plot_tab3(myclass); % so it plots the comparison graphs
        %% finding the maximum values of the fourier transform (and their position in Hz)

        [pks,locs]=findpeaks(myclass.Filter_Fourier_data,myclass.Filter_Hertz,'MinPeakProminence',10);
        if myclass.Filter_Fourier_data(1)>=10*myclass.Filter_Fourier_data(2)
            pks=[ (myclass.Filter_Fourier_data(1) ) ,pks'];
            [pks_sorted,index]=sort(pks,'descend');
            position = find(pks==myclass.Filter_Fourier_data(1));
            locs=[locs(1:position-1) myclass.Filter_Hertz(1) locs(position:end)];
            locs_sorted=locs(index);
        else
            [pks_sorted,index]=sort(pks,'descend');
```

Appendix-MATLAB CODE

```

        locs_sorted=locs(index);
    end
    myclass.Fourier_Peaks_Filtered=pks_sorted;
    myclass.Fourier_Peak_Pos_Filtered=locs_sorted;
end
function filter(myclass)
    if isempty(myclass.Filter_order) %&
        isempty(myclass.Filter_Cutoff_Ang_Freq)
            n=2;
            wc=2*pi*30;
            myclass.Filter_order=n;
            myclass.Filter_Cutoff_Ang_Freq=wc;
        else
            n=myclass.Filter_order;
            wc=myclass.Filter_Cutoff_Ang_Freq;
        end
        %% calculating the Transfer Function's a_k coefficients.

        Ts=1/myclass.Sample_Freq;
        a=zeros(length(n));
        coef=zeros(length(n));
        a(1)=1;
        g=pi/(2*n);
        coef(1)=1;
        for k=0:1:n-1
            a(k+2)=a(k+1)*(cos(k*g)/sin((k+1)*g));
            coef(k+2)=a(k+2)/wc^(k+1);
        end
        H_s=tf(1,flip(coef));
        sysd = c2d(H_s,Ts,'tustin'); % bilinear transform
        [num,den] = tfdata(sysd);
        f = cell2mat(den);
        c = cell2mat(num);

        x=myclass.rt_data;
        yn=(myclass.Offset)*ones(length(x),1);
        yn1=(myclass.Offset)*ones( [ length(x) (length(c)-1) ] );
        yn2=(myclass.Offset)*ones( [ length(x) (length(c)) ] );
        for nn=length(c):1:length(x)
            for i=1:1:length(c)-1
                yn1(nn,i)=-f(i+1)*yn(nn-i);
            end
            for i=0:1:length(c)-1
                yn2(nn,i+1)=c(i+1)*x(nn-i);
            end
            yn(nn)=sum(yn1(nn,:))+sum(yn2(nn,:));
        end
        %% performing fourier transform on the filtered signal
        L=length(yn);
        X_F_yn=fft(yn);
        X_mag_yn=abs(X_F_yn);
        Y1_yn=X_mag_yn(1:round((L/2)));
        f_bins=(0:(L-1)/2);
        f_Hz_yn = (myclass.Sample_Freq/L).*f_bins;
        %% saving them as class properties so i can use them in different
        functions
        myclass.Filtered_data=yn;
        myclass.Filter_Fourier_data=Y1_yn;
        myclass.Filter_Hertz=f_Hz_yn;
    end
    function Filter_Spec_Window(~,~,myclass)

```

Appendix-MATLAB CODE

```
if myclass.Spec_window(1)==true % checking if the window is already open
    return
end
Spec_Fig=uifigure();
Spec_Fig.Visible='off';
Spec_Fig.MenuBar='none';
Spec_Fig.NumberTitle='off';
Spec_Fig.Name='Filter Specifications';
Spec_Fig.Resize='off';
Spec_Fig.Units='pixels';
Spec_Fig.Position=[300 300 970 550]; %[left bottom width height]
movegui(Spec_Fig,'center');
Spec_Fig.Visible='on';
Spec_Fig.DeleteFcn=@Close_Window_Request,myclass};

myclass.Spec_window(1)=true;
myclass.Spec_window(2)=Spec_Fig;

%% setting up a grid layout

Spec_grid=uigridlayout(Spec_Fig,[3 1]);% 3x1 grid (rows x columns)
Spec_grid.RowHeight={'1x' '1x' '0.3x'};
Spec_grid.ColumnWidth={'1x'};
Spec_grid.RowSpacing=0;
Spec_grid.Padding=[0 0 0 0];
%% creating some axes to display my bode plots
Bode_ax1=uiaxes("Parent",Spec_grid);
Bode_ax2=uiaxes("Parent",Spec_grid);

Bode_ax1.Layout.Row=1;
Bode_ax2.Layout.Row=2;

Bode_ax1.Title.String='Magnitude Estimation of the Filter';
Bode_ax1.YLabel.String='Magnitude (dB)';
Bode_ax1.XGrid='on';
Bode_ax1.YGrid='on';
Bode_ax1.GridAlpha=0.22;
Bode_ax1.TickDir='out';

Bode_ax2.Title.String='Phase Estimation of the Filter';
Bode_ax2.YLabel.String='Phase [°]';
Bode_ax2.XGrid='on';
Bode_ax2.YGrid='on';
Bode_ax2.GridAlpha=0.22;
Bode_ax2.TickDir='out';
%% creating options to change the filter to your liking

Spec_grid2=uigridlayout(Spec_grid,[2 6]); % 1x3 grid (rows x columns)
Spec_grid2.Layout.Row=3;
Spec_grid2.RowHeight={'1x' '1x'};
Spec_grid2.ColumnWidth={150 150 150 150 150 150};

lbl1=uilabel('Parent',Spec_grid2,"Text",'Filter Type:');
lbl2=uilabel('Parent',Spec_grid2,"Text",'Filter Degree:');
lbl3=uilabel('Parent',Spec_grid2,"Text",'Cutoff Frequency (in Hz):');
lbl4=uilabel('Parent',Spec_grid2,"Text",'X-axis unit:');

lbl1.Layout.Row=1;
lbl2.Layout.Row=1;
lbl3.Layout.Row=1;
lbl4.Layout.Row=1;
```

Appendix-MATLAB CODE

```
    lbl1.Layout.Column=1;
    lbl2.Layout.Column=2;
    lbl3.Layout.Column=3;
    lbl4.Layout.Column=4;

    edt_field=uieditfield("Parent",Spec_grid2);
    edt_field.Editable='off';
    edt_field.Value='Butterworth Low Pass';
    edt_field.HorizontalAlignment='center';
    edt_field.Layout.Column=1;
    edt_field.Layout.Row=2;

    spn1=uispinner("Parent",Spec_grid2);
    spn1.Layout.Column=2;
    spn1.Layout.Row=2;
    spn1.Limits=[1 10];
    spn1.LowerLimitInclusive='on';
    spn1.UpperLimitInclusive='on';
    if isempty(myclass.Filter_order)
        spn1.Value=2;
    else
        spn1.Value=myclass.Filter_order;
    end
    spn2=uispinner("Parent",Spec_grid2);
    spn2.Layout.Column=3;
    spn2.Layout.Row=2;
    spn2.Value=30;% Default Value
    spn2.Limits=[1 100];
    spn2.LowerLimitInclusive='on';
    spn2.UpperLimitInclusive='on';
    if isempty(myclass.Filter_Cutoff_Ang_Freq)
        spn2.Value=30; % Default Value
    else
        spn2.Value=myclass.Filter_Cutoff_Ang_Freq/(2*pi);
    end
    %% creating dropdown for x-axis unit choice
    drp=uidropdown(Spec_grid2);
    drp.Layout.Column=4;
    drp.Layout.Row=2;
    drp.Items={'rad/s','Hz'};
    drp.Value='Hz'; % Default value
    drp.Placeholder='Options';
    %% couple of control buttons
    apply_btn=uibutton(Spec_grid2);
    apply_btn.Text='Apply';
    apply_btn.Layout.Row=2;
    apply_btn.Layout.Column=5;
    apply_btn.ButtonPushedFcn=
Plot_Spec_Window(Bode_ax1,Bode_ax2,spn1,spn2,drp,myclass) };

    cncl_btn=uibutton(Spec_grid2);
    cncl_btn.Text='OK';
    cncl_btn.Layout.Row=2;
    cncl_btn.Layout.Column=6;
    cncl_btn.ButtonPushedFcn= {@(x,y)delete(Spec_Fig)};

    Plot_Spec_Window(Bode_ax1,Bode_ax2,spn1,spn2,drp)
end
function Plot_Spec_Window(Bode_ax1,Bode_ax2,spn1,spn2,drp,myclass)
    %% Creating the actual filter specs
```

Appendix-MATLAB CODE

```

wc=2*pi*spn2.Value;
g=pi/(2*spn1.Value);
a=zeros(length(spnl.Value));
coef=zeros(length(spnl.Value));
a(1)=1;
coef(1)=1;
for k=0:1:(spn1.Value)-1
    a(k+2)=a(k+1)*(cos(k*g)/sin((k+1)*g));
    coef(k+2)=a(k+2)/wc^(k+1);
end
if spn1.Value==1 % unfortunately this is the only way to do this
    H_s=@(s)1./(coef(1)+coef(2)*s.^1);
elseif spn1.Value==2
    H_s=@(s)1./(coef(1)+coef(2)*s.^1+coef(3)*s.^2);
elseif spn1.Value==3
    H_s=@(s)1./(coef(1)+coef(2)*s.^1+coef(3)*s.^2+coef(4)*s.^3);
elseif spn1.Value==4
    H_s=@(s)1./(coef(1)+coef(2)*s.^1+coef(3)*s.^2+coef(4)*s.^3+coef(5)*s.^4);
elseif spn1.Value==5
    H_s=@(s)1./(coef(1)+coef(2)*s.^1+coef(3)*s.^2+coef(4)*s.^3+coef(5)*s.^4+coef(6)*s.^5);
elseif spn1.Value==6
    H_s=@(s)1./(coef(1)+coef(2)*s.^1+coef(3)*s.^2+coef(4)*s.^3+coef(5)*s.^4+coef(6)*s.^5+coef(7)*s.^6);
elseif spn1.Value==7
    H_s=@(s)1./(coef(1)+coef(2)*s.^1+coef(3)*s.^2+coef(4)*s.^3+coef(5)*s.^4+coef(6)*s.^5+coef(7)*s.^6+coef(8)*s.^7);
elseif spn1.Value==8
    H_s=@(s)1./(coef(1)+coef(2)*s.^1+coef(3)*s.^2+coef(4)*s.^3+coef(5)*s.^4+coef(6)*s.^5+coef(7)*s.^6+coef(8)*s.^7+coef(9)*s.^8);
elseif spn1.Value==9
    H_s=@(s)1./(coef(1)+coef(2)*s.^1+coef(3)*s.^2+coef(4)*s.^3+coef(5)*s.^4+coef(6)*s.^5+coef(7)*s.^6+coef(8)*s.^7+coef(9)*s.^8+coef(10)*s.^9);
elseif spn1.Value==10
    H_s=@(s)1./(coef(1)+coef(2)*s.^1+coef(3)*s.^2+coef(4)*s.^3+coef(5)*s.^4+coef(6)*s.^5+coef(7)*s.^6+coef(8)*s.^7+coef(9)*s.^8+coef(10)*s.^9+coef(11)*s.^10);
end
%% getting the x axis right
omega =2*pi*logspace(-3,4,2000);
Hz=logspace(-3,4,2000);
if drp.Value=="rad/s"
    X_axis_choice=omega;
    Bode_ax1.XLabel.String='Angular Frequency [rad/s]';
    Bode_ax2.XLabel.String='Angular Frequency [rad/s]';
elseif drp.Value=="Hz"
    X_axis_choice=Hz;
    Bode_ax1.XLabel.String='Frequency [Hz]';
    Bode_ax2.XLabel.String='Frequency [Hz]';
end
%% setting up the change of phase y axis
phase_Deg = rad2deg( angle(H_s(1i*X_axis_choice)) );
%% and now the magnitude y axis
magn = abs(H_s(1i*X_axis_choice));

```

Appendix-MATLAB CODE

```
magn_dB = 20 * log10(magn);
%% now actually creating the plots
semilogx(Bode_ax1,X_axis_choice/2/pi,magn_dB);
semilogx(Bode_ax2,X_axis_choice/2/pi,phase_Deg);
%% saving the user selecteed cutoff freq and filter order so they can
be used to actually filter the rest of the signal
myclass.Filter_Cutoff_Ang_Freq=wc;
myclass.Filter_order=spn1.Value;
end
function Close_Window_Request(src,~,myclass)
if src.Name=="Filter Specifications"
    myclass.Spec_window(1)=false;
    return
end

if src.Name=="Data and Key Variables"
    myclass.Var_window(1)=false;
    return
end
end
function tab3_function(tab3,myclass)
grid1_tab3=uigridlayout(tab3,[4 1]);
grid1_tab3.RowHeight={'fit','fit','fit','fit'};
grid1_tab3.BackgroundColor='#86b2b5';
grid1_tab3.Scrollable='on';
grid1_tab3.Padding= [0 0 0 0] ; % [left bottom right top]

grid2_tab3=uigridlayout(grid1_tab3,[1 5]);
grid2_tab3.Layout.Row=1;
grid2_tab3.BackgroundColor='#86b2b5';
grid2_tab3.ColumnWidth={'1x' '1x' '1x' '1x' 'fit'};
grid2_tab3.RowHeight={'0.5x'};
grid2_tab3.ColumnSpacing=15;
grid2_tab3.Padding=[10 0 10 2];% [left bottom right top]
%% creating an edifield so the port is Displayed
edt_tab3=uieditfield(grid2_tab3);
edt_tab3.Layout.Row=1;
edt_tab3.Layout.Column=5;
edt_tab3.HorizontalAlignment='center';
edt_tab3.Placeholder='No port connected';
edt_tab3.Editable='off';
edt_tab3.BackgroundColor="#f0c7a3";
edt_tab3.Tooltip='This field displays the current port to ehich the app
is connected.';
myclass.tab3_edt_field=edt_tab3;

%% creating the axis where the plots are gonna be displayed in.
ax31=uiaxes('Parent',grid1_tab3);
ax31.Layout.Row=2;
ax31.Title.String='Signal Comparison';
ax31.XLabel.String='Sample size';
ax31.YLabel.String='Magnitude';
ax31.XGrid='on';
ax31.YGrid='on';
ax31.Color=[0.1 0.1 0.1];
ax31.GridColor='#00FFFF';
ax31.GridAlpha=0.22;
ax31.TickDir='out';

ax32=uiaxes('Parent',grid1_tab3);
ax32.Layout.Row=3;
```

Appendix-MATLAB CODE

```
ax32.Title.String='Signal Comparison';
ax32.XLabel.String='Time (sec)';
ax32.YLabel.String='Magnitude';
ax32.XGrid='on';
ax32.YGrid='on';
ax32.Color=[0.1 0.1 0.1];
ax32.GridColor='#00FFFF';
ax32.GridAlpha=0.22;
ax32.TickDir='out';

ax33=uiaxes('Parent',grid1_tab3);
ax33.Layout.Row=4;
ax33.Title.String='Comparison of each Fourier Transform';
ax33.XLabel.String='Frequency (Hz)';
ax33.YLabel.String='Magnitude';
ax33.XGrid='on';
ax33.YGrid='on';
ax33.Color=[0.1 0.1 0.1];
ax33.GridColor='#00FFFF';
ax33.GridAlpha=0.22;
ax33.TickDir='out';
%% giving the axis a "global" handle
myclass.tab3_plots(1)=ax31;
myclass.tab3_plots(2)=ax32;
myclass.tab3_plots(3)=ax33;
end
function plot_tab3(myclass)
    plot(myclass.tab3_plots(1),myclass.rt_data,'Color',[1.0000 0.4706
0.0902],"MarkerEdgeColor","k","MarkerSize",8)
    hold(myclass.tab3_plots(1),'on');% 'hold on' but for uiaxes
    plot(myclass.tab3_plots(1),myclass.Filtered_data,'Color',[0 0.4470
0.7410],"MarkerEdgeColor","k","MarkerSize",8)
    legend(myclass.tab3_plots(1),{'Input Signal','Filtered
Signal'},'EdgeColor','white','TextColor','white');
    hold(myclass.tab3_plots(1),'off');

    plot(myclass.tab3_plots(2),myclass.time_variable,myclass.rt_data,'Color',[1
.0000 0.4706 0.0902],"MarkerEdgeColor","k","MarkerSize",8)
    hold(myclass.tab3_plots(2),'on');% 'hold on' but for uiaxes

    plot(myclass.tab3_plots(2),myclass.time_variable,myclass.Filtered_data,'Col
or',[0 0.4470 0.7410],"MarkerEdgeColor","k","MarkerSize",8)
    legend(myclass.tab3_plots(2),{'Input Signal','Filtered
Signal'},'EdgeColor','white','TextColor','white');
    hold(myclass.tab3_plots(2),'off');

    plot(myclass.tab3_plots(3),myclass.Hertz,myclass.Fourier_data,'Color',[1.00
00 0.4706 0.0902],"MarkerEdgeColor","k","MarkerSize",8)
    hold(myclass.tab3_plots(3),'on');% 'hold on' but for uiaxes

    plot(myclass.tab3_plots(3),myclass.Filter_Hertz,myclass.Filter_Fourier_data
,'Color',[0 0.4470 0.7410],"MarkerEdgeColor","k","MarkerSize",8)
    legend(myclass.tab3_plots(3),{'Input Signal','Filtered
Signal'},'EdgeColor','white','TextColor','white');
    hold(myclass.tab3_plots(3),'off');
end
```