



UPPSALA
UNIVERSITET

UPTEC E 23013

Examensarbete 30 hp

Juli 2023

Development of an embedded system platform for signal analysis and processing

Philip Lind



UPPSALA
UNIVERSITET

Development of an embedded system platform for signal analysis and processing

Philip Lind

Abstract

Information is often stored and transmitted through electrical signals. This information may need refinement, which may be done by processing and altering the electrical signals, in which it is transmitted. When refining a signal, a frequency selective filter is often used. It can be implemented through digital signal processing (DSP). DSP is a concept where signals are refined using a digital compute system. Digital systems are designed to replace their analog counterpart, mitigating their flaws in scalability, complexity and cost. A DSP system is typically implemented using software on a small computer, while analog systems are implemented through various electronic components.

The objective of this project is to design a DSP system that filters analog input data using automatically synthesised filters from user-defined input specifications. The DSP system is implemented using a microcontroller. The system designed the filters and found the filter coefficients. It then uses analog to digital converter (ADC) to sample an input signal and applies the filter. Lastly, it uses the digital to analog converter (DAC) to reconstruct a filtered, analog result. A user interface is not designed for the system, and only a limited number of filters are implemented. However, the system is successful in designing filters and finding their coefficients.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala

Handledare: Ping Wu Ämnesgranskare: Roland Hostettler

Examinator: Mikael Bergkvist

Sammanfattning

Information lagras och överförs ofta genom elektriska signaler. Denna information kan behöva förfinas, vilket kan göras genom att bearbeta och ändra de elektriska signalerna i vilka den överförs. När man förfinar en signal används ofta ett frekvensselektivt filter. Det kan implementeras genom digital signalbehandling (DSP). DSP är ett koncept där signaler förfinas med hjälp av ett digitalt beräkningssystem. Digitala system är utformade för att ersätta sina analoga motsvarigheter, och mildra deras brister i skalbarhet, komplexitet och kostnad. Ett DSP-system implementeras vanligtvis med hjälp av programvara på en liten dator medan analoga system implementeras genom olika elektroniska komponenter.

Målet med detta projekt är att utforma ett DSP-system som filtrerar analog indata med hjälp av automatiskt syntetiserade filter från användardefinierade specifikationer. DSP-systemet implementeras med hjälp av en mikrokontroller. Systemet utformar filtren och hittar filterkoefficienterna. Det använder sedan en analog till digital omvandlare (ADC) för att sampla en insignal och applicera filtret. Slutligen använder det en digital till analog omvandlare (DAC) för att återskapa ett filtrerat, analogt resultat. Ett användargränssnitt utformas inte för systemet, och endast ett begränsat antal filter implementeras. Systemet lyckas dock med att utforma filter och hitta deras koefficienter.

Acknowledgements

I would like to thank my supervisor Ping Wu who helped me define the project and provided the equipment needed to realize the project.

I would also like to thank my subject reader Roland Hostettler who has helped me throughout the project to the completion of it and its report.

Abbreviations

ADC	-	Analog to Digital Converter
BP	-	Band Pass
BS	-	Band Stop
CPU	-	Central Processing Unit
DAC	-	Digital to Analog Converter
DMA	-	Direct Memory Access
DSP	-	Digital Signal Processing
FIR	-	Finite Impulse Response
HP	-	High Pass
IIR	-	Infinite Impulse Response
ISR	-	Interrupt Service Routine
LP	-	Low Pass
USART	-	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	-	Universal Serial Bus

Contents

1	Introduction	1
1.1	Background	1
1.2	Project goals and delimitations	2
1.3	Project description	2
2	Theory	4
2.1	Hardware	4
2.1.1	Buffering	5
2.1.2	Interrupt Service Routine	5
2.1.3	USART	5
2.2	Conversion between analog and digital signals	6
2.2.1	Sampling	6
2.2.2	Analog to Digital Conversion	8
2.2.3	Digital to Analog Conversion	8
2.3	Filters	9
2.3.1	Background	9
2.3.2	Poles	10
2.3.3	Overview of filter design	12
2.4	Finite Impulse Response filters design	12
2.4.1	Background	12
2.4.2	Filter types	13
2.4.3	Windowing	14
2.5	Infinite impulse response filters design	15
2.5.1	Background	15
2.5.2	Coefficients of low pass and high pass filters	16
2.5.3	Coefficients of Band Pass and Band Stop filters	19
2.5.4	Second order sections	20
3	Implementation	22
3.1	Hardware	22
3.1.1	Microcontroller	22
3.1.2	Custom breadboard	23
3.1.3	Oscilloscope	24
3.2	Software	25
3.2.1	Application Software	26
3.2.2	Finite impulse response application	26
3.2.3	Infinite impulse response application	27

4	Results	30
4.1	Evaluation setup	30
4.2	Finite impulse response filters	31
4.3	Infinite impulse response filters	35
5	Discussion and conclusions	40
6	References	44

List of Figures

1	Block diagram of the system.	3
2	Block diagram of the used microcontroller's architecture [1].	4
3	Interface of USART connection between two devices.	6
4	Conversions between Analog and Digital data.	6
5	Aliasing illustrated from discrete data points.	7
6	Structure of a 4 bit R-2R DAC ladder.	9
7	Filter design process.	11
8	Filter design process.	12
9	Plot of coefficients of Rectangular, Hamming and Hann windows.	15
10	Example of how the characteristics of Butterworth and Chebychev filters.	16
11	Two cascaded systems.	19
12	Two parallel systems.	20
13	Increasing stability using 2nd order filter.	21
14	System overview.	22
15	AVR layout.	23
16	Custom breadboard.	24
17	PicoScope 2207A.	25
18	FIR application data flow.	27
19	A FIR filter's block diagram.	27
20	IIR application data flow.	28
21	An IIR filter's block diagram.	29
22	An IIR filter's block diagram when it is designed with second order sections.	29
23	Frequency response of low pass FIR filters.	32
24	Frequency response of high pass FIR filters.	33
25	Frequency response of band pass FIR filters.	34
26	Frequency response of band stop FIR filters.	35
27	Frequency response of low pass IIR filters.	36
28	Frequency response of high pass IIR filters.	37
29	Frequency response of band pass IIR filters.	38
30	Frequency response of band stop IIR filters.	39

31	The upper two subplots show the input and output of a high pass Butterworth filter with specifications stated in Table 2 when a signal of 1kHz is applied. The lower two subplots show the input and output of the same system when no signal is applied.	42
----	---	----

1 Introduction

1.1 Background

Information can be carried by multiple methods, such as text, sounds or electric signals. Processing these signals can be viewed as a refinement of information. The information may be optimized for a specific task, analyzed with a specific focus, or corrected for artifacts through signal processing. Certain aspects of the information may be modified, made prominent, isolated, or removed. Removing certain aspects of the presented information is often referred to as filtering. Filtering may be performed to transmit the relevant information for the right application. For example, a common application of filtering is in home theaters. The system consists of multiple speakers specified for the playback of different spectra of sound. Tweeters are solely good at reproducing high-frequency sounds, woofers reproduce mid-frequency sounds and sub-woofers are solely good at reproducing low-frequency sounds. The system needs to filter the sound so that each speaker only receives information with its specified spectra isolated for optimal performance.

Such filtering systems are traditionally constructed using analog electrical circuits containing resistors, capacitors and inductors. Such analog systems are, however, hard to tune and make modifications to, while also being expensive. Once they are designed and produced, it will be challenging to correct for flaws. Limitations of analog systems may be addressed by implementing the signal processing system digitally on computers. Such digital systems are software driven and therefore easier to apply on cheap computational devices with low power consumption. The filters may then also be modified using software updates and features to adapt settings according to individual user preferences with more flexibility may be implemented. A sophisticated digital system is also more scalable at a lower cost than an analog system, as software can be implemented in cheap hardware without alteration and without variation in quality. A high-quality digital filter with high performance is desirable to design a scalable system to address analog filters' flaws.

Filters are needed if information shall be extracted, removed, or enhanced in a signal. They can remove noise from the signal and they can specify within what frequency range a signal is to be transmitted. Filters allow us to not only find a signal that has been sent and isolate it, but also to send multiple different signals simultaneously using different frequencies for each. These characteristics are required for any digital communication, and for multiple electronic communication devices, such as phones and wireless headphones to operate in proximity to each other.

1.2 Project goals and delimitations

The goal of the project is to create a digital signal processing system that filters analog input data using automatically synthesized filters from user-defined input specifications. The system shall include windowed finite impulse response (FIR) low pass, high pass, band pass and band stop filters, as well as low pass, high pass, band pass, and band stop Butterworth and Chebychev infinite impulse response (IIR) filters. The system is also to include the ability to use an analog input signal, apply the filter digitally, and output the resulting signal using an analog output.

The focus of the project is to implement filter creation from specifications, and not to develop a consumer friendly product. There are therefore limitations on the system's user interface, its robustness, and instructions on how the system is implemented. There is no dedicated user interface for the system, which would be required for a consumer-level product. The user can make the system inoperable if text is entered as input instead of numbers, or if a line of code is deleted. Similarly, the ease of use of hardware connections and interfaces is not considered. The system also only performs a set number of pre-defined filters as stated above, although the system may be expanded with additional filters. For evaluations, the tools are limited to an oscilloscope, which is used to perform the result measurements.

1.3 Project description

The project aims at creating a system for designing and applying filters that are to be run on an easy-to-use hardware platform. The filter design process is to use user data inputs such as filter type, cut-off and sampling frequencies, as well as the number of poles. The poles affect the properties of how the system filters the information and its stability. The user shall be able to set any filter parameters and receive a means to filter input data accordingly by calculating and applying filter coefficients specific to the user preferences. These coefficients are applied to the input signal coming from the analog to digital converter (ADC) to find the digitally filtered output signal. When the output is found, it is reconstructed using the digital to analog converter (DAC). The hardware platform used for the project is Atmel's AVR UC3-A3 XPLAINED and the programming language C is used to create the application software. The filters are applied in real time using features such as direct memory access (DMA), and interrupt service routine (ISR).

An overview of the system is shown in Figure 1. The filter is designed using user inputs and outputs filter coefficients. This is solely done once for a filter, and the coefficients remain static for the user inputs. The ADC has an internal buffer that is always loading new data in an internal buffer using DMA. The buffer is loaded in an array that is used to process the data. The input data is then loaded into the filter application along with the filter coefficients when a trigger is pressed. The DAC is always reconstructing analog data using an internal buffer handled by DMA. The filter output is transferred to the DAC's internal buffer to update what signal is outputted.

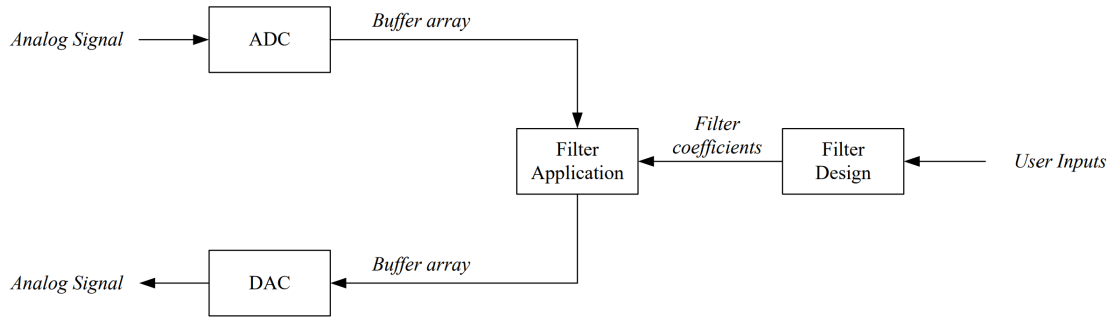


Figure 1: Block diagram of the system.

The platform has low cost, consumes little power, and meets the fundamental hardware properties required for a well-designed digital signal processing system to address the flaws of analog signal processing. The system requires little effort to reconstruct and is able to run as a low-power battery electric system. The user will be able to set the following software parameters.

- Filter type,
- window type for FIR filter,
- Butterworth or Chebyshev filter for IIR filter,
- low pass, high pass, band pass or band stop filter,
- burst or continuous buffer mode,
- sampling frequency,
- buffer size,
- cut-off frequencies,
- number of poles,
- ripple for Chebyshev filter.

2.1.1 Buffering

Buffering is the process of moving data within the system. Data is moved to and from the random-access memory (RAM) either using the central processing unit (CPU) or DMA. DMA is a hardware module that can perform data read and write operations independently from the CPU and eases its operational load. The CPU simply initiates the memory transfer and receives an interrupt once the transfer has been completed [2]. The CPU is typically involved in the whole process of moving data from one memory location to another. The DMA is used when data is required to be moved at the same time as when the CPU is to perform operations. The CPU is unavailable for any other task while performing these operations, as it solely can perform one computation at a time. This proves an inconvenience for tasks where large amounts of data need to be recorded and processed continuously. Such instances may be when inputs are processed through the ADC and when outputs are processed through the DAC. The filters may then be applied by the CPU while the DMA applies the ADC and DAC.

There are two buffer modes available. The first mode is continuous where a new output is calculated to replace the previous output calculation once it is finished. There will thereby always be a new and updated output under calculation in this mode. The second mode is to calculate outputs in bursts. The DMA is still collecting data for the ADC continuously, but the data is not used. The output is also paused and the same signal remains at the output. A new set of input values are loaded from the ADC and processed through the system if a hardware indicator is triggered.

2.1.2 Interrupt Service Routine

A software is typically compiled starting from the first row of code, and ending at the last. It computes one task at a time, and this is sufficient in many applications. There is a feature used in the project called interrupt service routine (ISR) that makes it possible to execute a block of code with an external trigger. The trigger may be a timer or physical switch and once the specified event occurs, the ISR pauses the main code, while the associated method is executed [1].

The ISR needs a specified criterion for activating and performing a task. For the ISR to activate, an interrupt flag needs to be triggered. This can for example be done through an external event or on a predetermined basis using a timer. Different use cases for the ISR require different triggering methods. An ISR can for example be used for starting a memory buffer transfer through DMA or changing a flag to indicate that a setting has changed. The ISR allows for the program to run more efficiently by, for example, only initiating a new DMA transfer once its buffer has completed its task of moving data.

2.1.3 USART

Serial communication is used through the standard USART. It is used to send data between the microcontroller and the PC using low hardware complexity. USART is

based on RS-232 and can be set up differently depending on what application it is to be used for [3].

The synchronous mode is used for communication bidirectional communication. It has a clock signal and data is sent at a fixed rate. The asynchronous mode is used for unidirectional communication. It does not have a clock signal and data is sent at a bit-by-bit rate rather than sending data blocks. The asynchronous mode has a higher bandwidth than the synchronous mode given that all other aspects are kept the same. USART requires two connections that are named RxD and TxD that are used to receive and transmit data respectively. It is configured as shown in Figure 3.

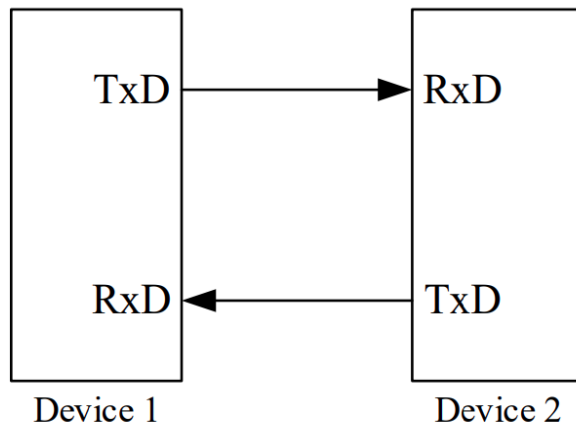


Figure 3: Interface of USART connection between two devices.

2.2 Conversion between analog and digital signals

Signal processing involves interacting with both analog and digital signals. The complete system's input and output are analog, while the signal processing is computed digitally. Analog signals are converted into digital representations, and digital signals are converted into analog representations. The conversions are illustrated in Figure 4.

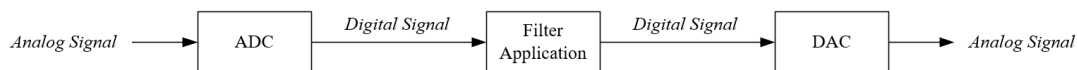


Figure 4: Conversions between Analog and Digital data.

2.2.1 Sampling

Sampling is the process of taking a measurement of a signal at instances with a constant time interval. The rate of sampling is measured through the sampling frequency f_s . A sampling frequency should be chosen carefully as it affects the discretized signal in multiple ways.

A too low sampling frequency causes the output to be an incorrect representation of the original signal. It will then not be possible to recreate the original signal with the gathered information. The phenomenon is called aliasing and is illustrated in Figure 5 [4]. First, seven data points are shown from which a signal is to be recreated. Two possible signals are then shown that have vastly different frequencies although both fit the data points.

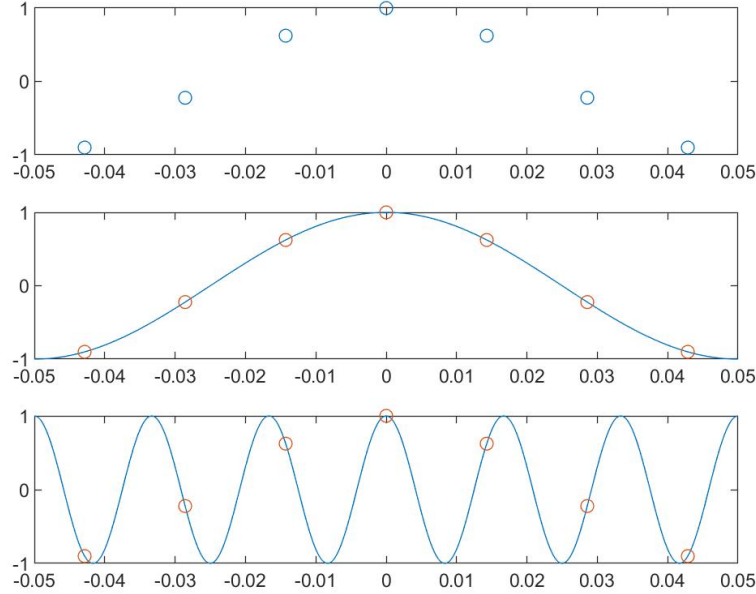


Figure 5: Aliasing illustrated from discrete data points.

Aliasing can be removed by increasing the sampling frequency, and reducing the components of the signal with frequencies that are too high. It is found that the sampling frequency needs to be set to at least double of the signal's contained highest frequency in order to remove aliasing and gain valuable information from the discretized signal. This particular frequency is named the Nyquist frequency f_N and is calculated by

$$f_N = \frac{f_s}{2}, \quad (1)$$

where f_s is the sampling frequency. The Nyquist frequency is thereby the highest possible frequency that is detectable by the system. An analog antialiasing filter is also applied to remove irrelevant high-frequency components of a signal that cannot be recreated using the ADC.

2.2.2 Analog to Digital Conversion

The ADC compares the incoming analog signal with a set of predetermined values. The predetermined values are spread between the minimum and maximum voltage of the ADC [4]. The voltage span that the analog signal is compared to can be similarly sized, or vary in size to achieve a higher resolution for voltage levels of interest. The output of each individual comparison will be 0 if the signal does not reach the baseline value, and 1 if the signal is larger than the baseline value, creating an array of ones and zeros. These digital values are fed into a converter that outputs a compressed digital interpretation of the analog signal.

The concepts of sampling and quantization are applied and adjusted to the system's required specifications. Quantization is the process of measuring the amplitude of a signal at an instance in time and converting it into a digital representation. Limitations of quantization include the range of the analog values that may be stored and the digital data's resolution [4].

Resolution is a measure of how precisely the original signal's amplitude can be determined. The resolution indicates the span of the digital value's representation in the analog domain. The resolution of an electrical signal can be calculated through

$$Q = \frac{V_{high} - V_{low}}{2^N} \quad (2)$$

if linear quantization is used, where Q is resolution, N is the number of bits, V_{high} is the ADC upper voltage limit, and V_{low} is the lower. A larger amount of bits on a fixed voltage value range results in a higher resolution. Each interval represented by a bit corresponds to a smaller data interval with a higher resolution and gives a more precise approximation of the input value. A resolution that is too low causes loss of information, and a too high resolution causes unnecessary load on the CPU. The resolution should be chosen high enough to record relevant information and maintain a high signal-to-noise ratio.

2.2.3 Digital to Analog Conversion

The DAC converts a digital signal into an analog one. The constructed analog signal is a continuous waveform with varying amplitude and frequency, but is to contain a digital representation of the information provided by the analog signal. A commonly used DAC method is the resistor ladder. The method converts digital signals to analog effectively and its structure can be seen in Figure 6.

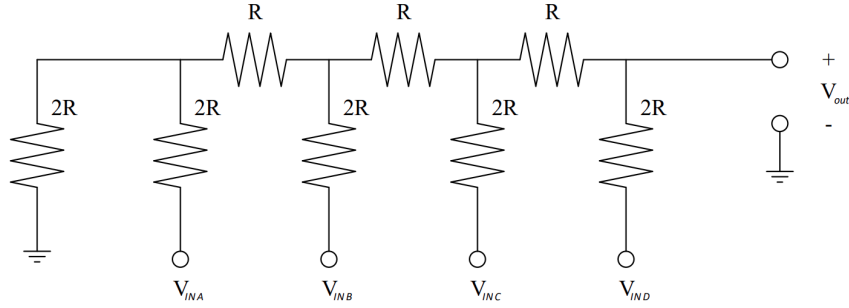


Figure 6: Structure of a 4 bit R-2R DAC ladder.

The R-2R ladder is based on the configuration of the resistors in the setup. The resistors' value need to be precise when compared to each other. Specifically, about half of the resistors need to be as close as possible to the exact same resistance R , and the remaining need to have double the resistance of R .

As the total resistance from each digital input to V_{out} is different, the inputs have an individual effect on the output as individual input voltages are added. Each V_{IN} represents a bit in the digital format and is represented by a voltage that is common between the inputs. The number of input bits is directly related to the resolution of the DAC, and how often the V_{IN} 's can be updated represents the system's sampling frequency. The R-2R ladder DAC in Figure 6 is constructed such that analog values can be constructed, varying between ground and the voltage of the logic bit. An amplifier can be used if higher voltage levels are required.

2.3 Filters

2.3.1 Background

A filter can take many forms. It may be a physical device that separates solids from liquids or a system that removes information of a certain criterion. They can generally be described as a method of sorting out objects or information from an initial set. Filters that manage information are often constructed using electrical equipment, in the analog or digital form.

The filter separates information with set parameters by adding a barrier that only the information of certain criteria can pass. A filter can have several different properties and can be constructed in different ways, but a common parameter used to filter information is its frequency. Two possible basic features of frequency selective linear filters are low pass filters and high pass filters. Those filters can then be combined to construct band pass filters and band stop filters. A filter's name corresponds to what effect the filter has on the input signal. A low pass filter lets low frequency parts of a signal through while stopping higher frequency components. The high pass does the opposite. The band pass and band stop filters create a span in the frequency spectrum where the signal is passed through, respectively stopped.

A digital filter is expressed through equations, where its output $y[n]$ is expressed as a convolution between its impulse response $h[n]$ and input $x[n]$ when in the time domain through

$$y[n] = h[n] * x[n]. \quad (3)$$

The impulse response is the concept of how a system responds when an impulse is used as an input. A unit impulse signal $\delta[n]$ reaches an amplitude of 1 at one instance in time and has an amplitude of 0 at all other instances according to

$$\delta[n] = \begin{cases} 1 & , n = 0 \\ 0 & , n \neq 0 \end{cases}. \quad (4)$$

As components of a signal are hard to identify when it is analyzed in the time domain, the filter system is often transformed into the frequency domain. A general system's function in the discrete z-domain is expressed as

$$Y[z] = H[z]X[z], \quad (5)$$

where $Y[z]$ is its output, $H[z]$ is the transfer function, and $X[z]$ is its input in the discrete z-domain. The z-domain is a representation of the frequencies contained in a discrete time signal. The transfer function can be used to understand the system's characteristics and is defined as [5]

$$H[z] = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + \dots}{1 - b_1 z^{-1} - b_2 z^{-2} - b_3 z^{-3} - \dots} \quad (6)$$

$$H[z] = \frac{\sum_{i=0}^N a_i z^{N-i}}{1 + \sum_{i=1}^N b_i z^{N-i}}.$$

N is the filter's number of poles, and a_i and b_i are the filter coefficients. The a_i and b_i coefficients define the characteristics of the filter and its features, such as stability. The filter coefficients are found through the filter design process.

Digital filters are implemented as Finite Impulse Response (FIR) or Infinite Impulse Response (IIR) filters. FIR filters have an impulse response with a finite number of nonzero samples in their impulse response, while IIR filters have an infinite number of nonzero samples [5].

The number of nonzero samples in the impulse response indicates how long time the filter requires to settle when an impulse signal is sent as input. FIR filters thereby reach an output of 0 and preserve the input after a finite number of time steps, while IIR filters require an infinite number of time steps. FIR filters only use past and present output values for their present output while IIR filters also use their past outputs.

2.3.2 Poles

A filter's poles describe its features, such as its impulse response and stability. The impulse response describes how effectively the filter archives its steady output after an

input has been entered. The stability describes if the filter archives a steady result or if oscillations will be caused by the filter [5]. The poles can be found through rewriting (6) as the pole-zero form

$$H[z] = k \frac{(z - z_1)(z - z_2) \dots (z - z_m)}{(z - p_1)(z - p_2) \dots (z - p_n)} = k \frac{\sum_{i=1}^m (z - z_i)}{\sum_{i=1}^n (z - p_i)} \quad (7)$$

where k is a gain constant, m and n are the number of zeros and poles respectively, z_i are the zeros and p_i are the poles. Filters can have poles where $z = 0$ and $z = \infty$. A system that has a pole at $z = \infty$ also has a zero at $z = 0$, which attenuates DC gain. Pole locations at $z = \infty$ also make the system non-causal, meaning that the system is not only dependent on present and past inputs but also future inputs. This can however be solved by introducing a delay in the system. These poles where $z = \infty$ are often neglected when counted as they do not affect the signal's amplitude and only do add a time delay to the system.

Additionally to describe a filter's features, the poles can be used effectively when designing a filter. In order to design a stable discrete time system, a plot of poles and zeros can be used. The plot is a representation of the system's transfer function in the complex plane as seen in Figure 7. The poles' placement affects the system's stability and phase response. The poles need to be placed in the shaded region inside of the dashed circle, representing the unit circle, for a discrete-time system to achieve stability. The filter type is then the base for deciding the distribution, angle and distance from the origin of the poles. The number of poles is directly related to the filter's order.

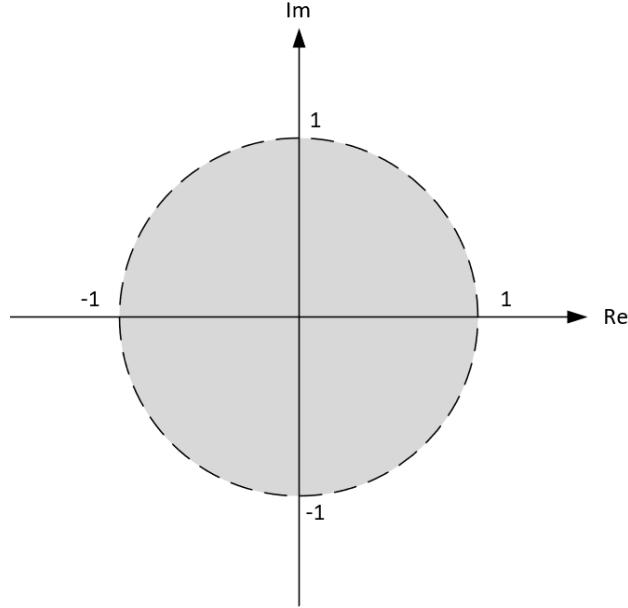


Figure 7: Filter design process.

2.3.3 Overview of filter design

The filter design process used for the project is outlined in Figure 8. The first option that needs to be selected is whether an IIR or a FIR filter is to be designed. Secondly, the characteristic of low pass, high pass, band pass or band stop needs to be decided. Lastly, the windowing method is decided for FIR filters. For the IIR filter, Butterworth or Chebychev filter type is selected. All filters need information on cut-off frequencies, sampling frequencies and their order. The Chebychev filter needs additional information on the ripple introduced, expressed in percent.

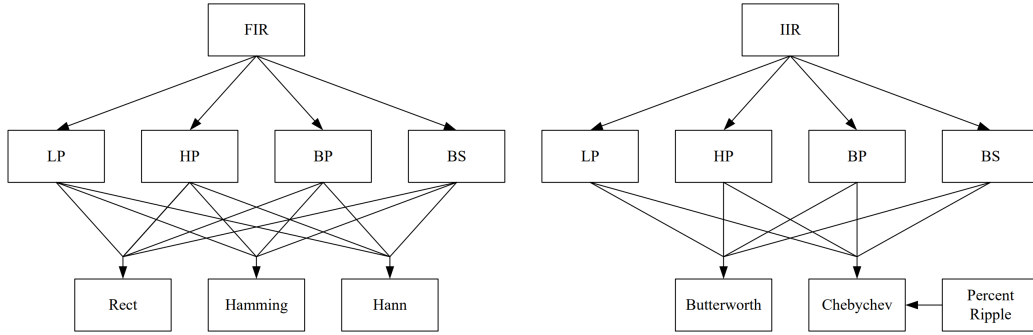


Figure 8: Filter design process.

2.4 Finite Impulse Response filters design

2.4.1 Background

FIR filters are not recursive, and the result of the filter does solely depend on its input and pre-set parameters. A general FIR filter can be described according to

$$y[n] = b_0x[n] + b_1x[n-1] + \dots + b_Nx[n-N] = \sum_{i=0}^N b_i x[n-i] \quad (8)$$

that is derived from (6) [6]. A FIR filter has a finite number of nonzero samples in its impulse response, found through

$$h[n] = \sum_{i=0}^N b_i \delta[n-i]. \quad (9)$$

It will make a linear phase possible and result in a stable output. The filter is a linear phase if and only if the system's coefficients are symmetrical around its center coefficient. A filter with a linear phase keeps the main characteristics of the signal intact where all frequencies present will be time shifted by the same amount and be a linear function with respect to frequency. The FIR filter has no analog counterpart and is computationally heavy to perform on input data compared to IIR filters. This is

due to IIR filters' utilization of previous outputs, reducing the number of computations required for each output value and reducing the system's required filter order. The filter and window type need to be decided first when designing a finite impulse response filter. The filter type decides the system's response to frequencies, and the window type decides what dampening effect is used on the filter. The filter can be applied to discrete data inputs when the filter coefficients have been calculated using the Cauchy product

$$y[n] = \sum_{i=0}^N h[i]x[n-i], \quad (10)$$

where $y[n]$ is the convoluted result of the system of the impulse response $h[i]$ and system input $x[i]$. The Cauchy product is used as an algorithm to compute the convolution of two data sets in discrete time.

2.4.2 Filter types

An impulse response corresponding to the chosen filter type is used as the base of the filter. The impulse response characterizes how the filter reacts to its input. The used impulse response for the low pass filter is

$$h_{lp}[n] = \frac{\omega_{cl}}{\pi} \text{sinc} \frac{\omega_{cl}(n-M)}{\pi}, \quad (11)$$

while the high pass filter uses the impulse response [6]

$$h_{hp}[n] = \frac{\omega_{cl}}{\pi} \text{sinc} \frac{\omega_{cl}(n-M)}{\pi} (-1)^n. \quad (12)$$

The band pass filter's impulse response is

$$h_{bp}[n] = \frac{\omega_{cu}}{\pi} \text{sinc} \frac{\omega_{cu}(n-M)}{\pi} - \frac{\omega_{cl}}{\pi} \text{sinc} \frac{\omega_{cl}(n-M)}{\pi}, \quad (13)$$

and the band stop filter's impulse response is [6]

$$h_{bs}[n] = \text{sinc}(n-M) - \frac{\omega_{cu}}{\pi} \text{sinc} \frac{\omega_{cu}(n-M)}{\pi} - \frac{\omega_{cl}}{\pi} \text{sinc} \frac{\omega_{cl}(n-M)}{\pi}. \quad (14)$$

The impulse response has N coefficients, n is an index iterated from 0 to N , ω is the angular frequency, and M is calculated through

$$M = \frac{N-1}{2}. \quad (15)$$

A longer array of filter impulse response coefficients results in a computationally heavier filter that will take a longer time to apply.

2.4.3 Windowing

Windowing is a method of modifying the sinc function to reduce undesirable effects. The sinc function's truncated ends reduce performance in the form of introduced ripple and a reduced attenuation. The windowing is thereby a means of increasing the performance. The purpose of applying a window is to mitigate the discontinuity, which reduces ripple and increases attenuation. There are different windows that have different characteristics, such as rectangular, Hamming and Hann windows that are used in the project. The rectangular window is calculated through

$$w[i] = 1, \quad 0 \leq i \leq M, \quad (16)$$

and has an amplification of one across the impulse response and does therefore not alter the filter coefficients [7]. M is the window's width and i is iterated from zero to M . Using the same variables, the Hamming window is calculated through

$$w[i] = 0.54 - 0.46 \cos \left(2\pi \frac{i}{M} \right), \quad 0 \leq i \leq M, \quad (17)$$

and the Hann window is calculated through [8] [9]

$$w[i] = \frac{1}{2} \left(1 - \cos \left(2\pi \frac{i}{M} \right) \right), \quad 0 \leq i \leq M. \quad (18)$$

The rectangular window does retain the magnitude 1 across its span and does therefore not affect the sinc function. The Hamming and Hann windows have a dampening effect as the maximum gain is 1 for each. The Hamming and Hann windows are shaped like curves with slightly different features. Both have a gain of 1 in the middle of the span. The gain then decreases towards the edges of the windows. The Hamming window has a gain close to 0.1 while the Hann window has a minimum gain close to zero. The three implemented windows are visualized in Figure 9.

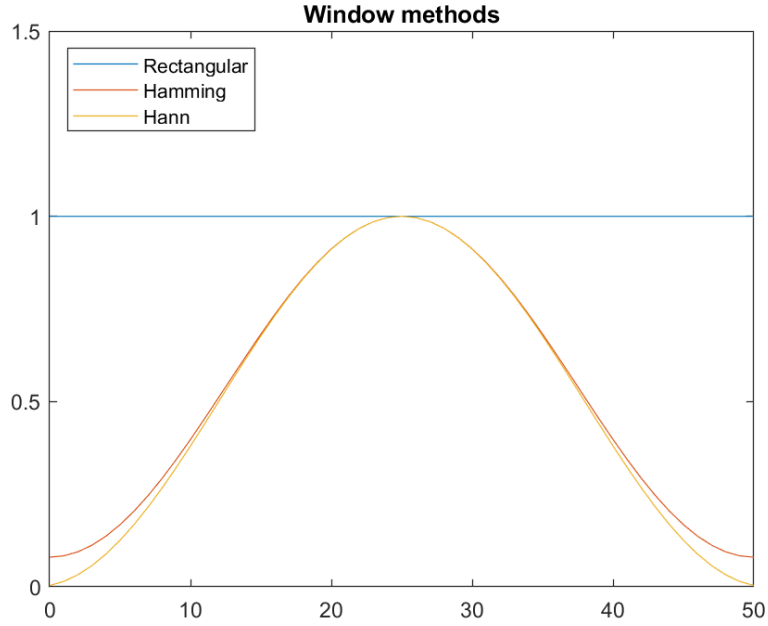


Figure 9: Plot of coefficients of Rectangular, Hamming and Hann windows.

2.5 Infinite impulse response filters design

2.5.1 Background

IIR filters are characterized as filters that are recursive. This implies that the filter is dependent on its input, pre-set variables as well as previous outputs. An IIR filter is computationally light and can therefore perform computations quicker than FIR filters, although, it is not necessarily stable. The system can start to oscillate and cause all information of the signal to be lost if the filter is not set up correctly. A general recursive filter can be described according to

$$\begin{aligned}
 y[n] &= a_0x[n] + a_1x[n-1] + b_1y[n-1] + a_2x[n-2] + b_2y[n-2] + \dots \\
 &= \sum_{i=0}^n a_i x[n-i] + \sum_{i=1}^n b_i y[n-i] \quad (19)
 \end{aligned}$$

where the filter's coefficients need to be found before applying equation [4]. The coefficients depend on filter settings such as cut-off frequency and filter type. The two implemented IIR filter types are the Butterworth and the Chebyshev filters. The two filter types have slightly different characteristics. A Butterworth filter has a near flat frequency pass band while a Chebyshev filter has ripples. The Chebyshev filter does however have a steeper roll-off compared to a Butterworth filter, given the two filters are of the same order. This relation can be visualized in Figure 10 where a general

Butterworth filter's frequency response is plotted alongside a general Chebychev filter with 0.5% ripple as well as one with 20% ripple [4].

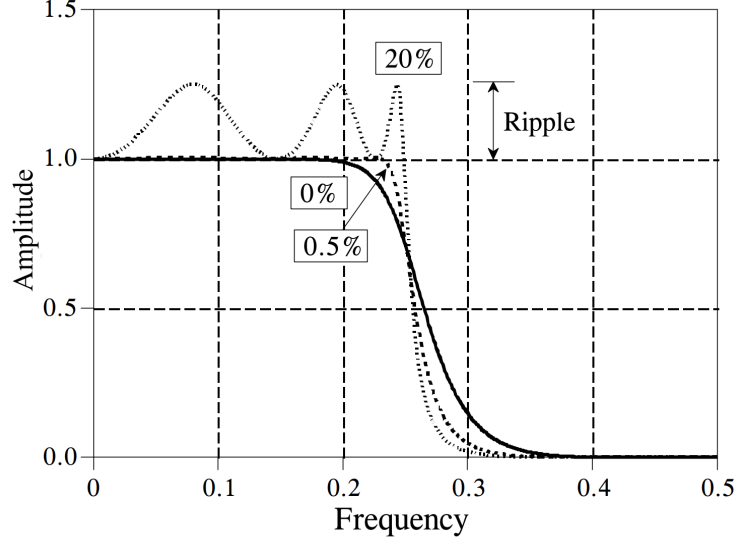


Figure 10: Example of how the characteristics of Butterworth and Chebychev filters.

The poles have to be found when designing a Butterworth or Chebychev filter. They need to be within the criterion of stability for discrete-time systems. A Butterworth filter has its poles placed symmetrically in a circle within the unit circle with equal angles between each point, and the Chebychev filter has its poles placed in an ellipse. A low pass filter is then designed, which then may be converted into a high pass filter. The coefficients are found, and the gain is normalized. A low pass filter and a high pass filter are combined in cascade to form a band pass filter, or parallel to form a band stop filter.

2.5.2 Coefficients of low pass and high pass filters

Four user parameters are needed to find the coefficients of the IIR filter. The cut-off frequency, whether it is an HP or LP filter, the filter's ripple and the filter's number of poles. The number of poles needs to be even to design a stable IIR filter.

The poles of the Butterworth filter are found by distributing them in the stable region inside the unit circle in the complex frequency domain. The real part of the poles, named as p below, of the filter are placed on the unit circle in the complex plane through

$$\text{Re}\{p\} = -\cos\left(\frac{\pi}{N \cdot 2} + \frac{(i-1) \cdot \pi}{N}\right) \quad (20)$$

for the real part, and the complex part is found through [4]

$$\text{Im}\{p\} = -\sin\left(\frac{\pi}{N \cdot 2} + \frac{(i-1) \cdot \pi}{N}\right). \quad (21)$$

The variable N represents the number of poles and the integer variable i is iterated from 1 to the filter's number of pole pairs. The poles found through (20) and (21) are placed in a way that they can be used directly for designing a Butterworth filter. Further alteration of the poles is however required to design a Chebyshev filter, where the height of the circle needs to be altered, turning the circle into an ellipse. The ellipse's height is controlled with respect to the introduced ripple. The ripple is introduced as a percent of the filter's maximum gain. The real part of the Chebychev pole in the ellipse is calculated through

$$\text{Re}\{p\} = \text{Re}\{p\} \cdot \frac{\sinh(vx)}{kx} \quad (22)$$

and the complex part through

$$\text{Im}\{p\} = \text{Im}\{p\} \cdot \frac{\cosh(vx)}{kx}. \quad (23)$$

The original $\text{Re}\{p\}$ comes from (20), and the original $\text{Im}\{p\}$ comes from (21). vx and kx are calculated as

$$vx = \frac{1}{N} \cdot \sinh^{-1} \left(\frac{1}{es} \right) \quad (24)$$

and

$$kx = \cosh \left(\frac{1}{N} \cdot \cosh^{-1} \left(\frac{1}{es} \right) \right), \quad (25)$$

where es is found through

$$es = \sqrt{\left(\frac{100}{100 - pr} \right)^2 - 1}. \quad (26)$$

pr is a user specification, indicating the amount of ripple. pr is a percentage for Chebychev filters and zero for Butterworth filters.

The poles need to be converted from the continuous complex frequency domain in which they are found into the discrete complex frequency domain (z-domain) as the signal processing is digital. This conversion is done through the bilinear transformation for two poles at a time, constructing one second order filter for each iteration. This transformation is then performed for each pole pair, resulting in the coefficients for a low pass filter with a cutoff frequency of 1. The coefficients are found through

$$x_0 = \frac{t^2}{D}, \quad (27)$$

$$x_1 = \frac{2T^2}{D}, \quad (28)$$

$$x_2 = \frac{T^2}{D}, \quad (29)$$

$$y_1 = \frac{8 - 2MT^2}{D}, \quad (30)$$

and

$$y_2 = \frac{-4 - 4 \operatorname{Re}\{p\}T - MT^2}{D}. \quad (31)$$

The values of T , ω , M and D are found through

$$T = 2 \tan\left(\frac{1}{2}\right), \quad (32)$$

$$M = \operatorname{Re}\{p\}^2 + \operatorname{Im}\{p\}^2, \quad (33)$$

and

$$D = 4 - 4 \operatorname{Re}\{p\}T + MT^2. \quad (34)$$

The results found from (27) - (31) are used to find the low pass or high pass coefficients for one pole pair of a filter through

$$a_0 = \frac{x_0 - x_1k + x_2k^2}{E} \quad (35)$$

$$a_1 = \frac{-2x_0k + x_1 + x_1k^2 - 2x_2k}{E}, \quad (36)$$

$$a_2 = \frac{x_0k^2 - x_1k + x_2}{E}, \quad (37)$$

$$b_1 = \frac{2k + y_1 + y_1k^2 - 2y_2k}{E} \quad (38)$$

and

$$b_2 = \frac{-k^2 - y_1k + y_2}{E} \quad (39)$$

where E is found as

$$E = 1 + y_1k - y_2k^2. \quad (40)$$

The filter coefficients are found by transforming from an original low pass filter with a cutoff frequency of one to either a low pass filter with an altered cutoff through

$$k = \frac{\sin\left(\frac{1}{2} - \frac{\omega_c}{2}\right)}{\sin\left(\frac{1}{2} + \frac{\omega_c}{2}\right)} \quad (41)$$

or a high pass filter with an altered cutoff through

$$k = -\frac{\cos\left(\frac{\omega_c}{2} + \frac{1}{2}\right)}{\cos\left(\frac{\omega_c}{2} - \frac{1}{2}\right)}, \quad (42)$$

where

$$\omega_c = 2\pi f_c. \quad (43)$$

The method described by (20) - (43) is iterated for every pole pair, resulting in separate coefficient values for a_0 , a_1 , a_2 , b_1 and b_2 for each. Each set of coefficients

represents one second-order filter. These second-order filters are then combined in series to construct the base of the final system according to (19). Before the coefficients are complete and can be used, the variables a_1 and b_1 need to be multiplied with -1 when calculated for a high pass filter, and the filter's gain needs to be normalized for the filter output to be a good representation of the information passed through. The gain is normalized when all filter coefficients are calculated and all poles have been processed. All of the a -coefficients are divided by the gain

$$K = \frac{\sum_{i=0}^N a[i]}{1 - \sum_{i=0}^N b[i]} \quad (44)$$

for low pass filters and

$$K = \frac{\sum_{i=0}^N a[i](-1)^i}{1 - \sum_{i=0}^N b[i](-1)^i} \quad (45)$$

for high pass filters before they can be used in (19) along with b_i .

2.5.3 Coefficients of Band Pass and Band Stop filters

Band pass and band stop filters are built by combining a low pass and a high pass filter. The two filters can be combined in cascade or in parallel into a single system. The two filters form a band pass filter if they are cascaded, and they form a band stop filter if they are combined in parallel.

When a band pass or band stop filter is to be created, a low pass filter and a high pass filter is constructed with the same number of poles and the same ripple, but with individual cut-off frequencies. The both filters' coefficients, a_i and b_i are found through Section 2.5.2. The values in b_i are multiplied with -1 and the value of b_0 in the arrays is set to 0. The band pass filter is two cascaded systems and is calculated by multiplying the low pass and high pass filter coefficients as [4]

$$H_{BP}[z] = H_{LP}[z] \cdot H_{HP}[z]. \quad (46)$$

The operation of cascading two systems is illustrated in Figure 11.

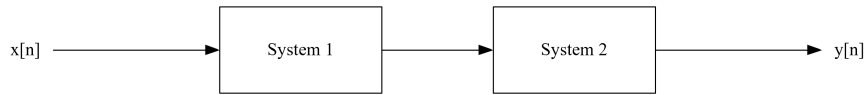


Figure 11: Two cascaded systems.

The band stop filter is calculated by having the low pass filter and high pass filter in parallel, which is done by addition between the two systems as [4]

$$H_{BP}[z] = H_{LP}[z] + H_{HP}[z]. \quad (47)$$

The operation is illustrated in Figure 12.

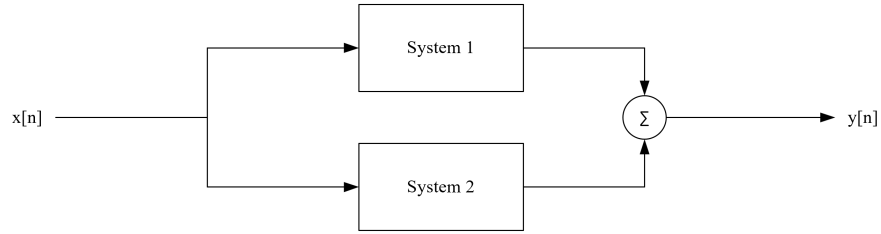


Figure 12: Two parallel systems.

where the predetermined coefficients of the low pass and high pass filters can be inserted.

The denominators are calculated in the same way for both parallel and cascaded systems. The numerators are however calculated by different algorithms, whereas the paralleled system is more complex. Lastly, all coefficients of the denominator of the resultant system are multiplied by -1 . The filter coefficients for the band pass and band stop filters will be twice as many as for low pass and high pass filters for the same number of user entered poles.

2.5.4 Second order sections

IIR filters have a tendency to become unstable when the number of poles is increased. This may cause a conflict between the desired performance of the filter, which may require more poles, and its stability. A method of increasing the performance of the system while mitigating the issue of stability is to split the higher order filter in cascaded lower-order ones.

The effect of a 10th order filter can be achieved by applying five 2nd order filters after each other, as demonstrated in Figure 13. The output of the first 2nd order filter is thereby used as input to the second filter. This method is iterated until the desired order equivalent is reached before the output becomes the finished filtered signal. This requires the filter design process to be slightly different than ordinary IIR filters. All filter coefficients will be designed for second order filters, rather than the order the user requested. The filtering properties of a higher order filter will be achieved through the application method.

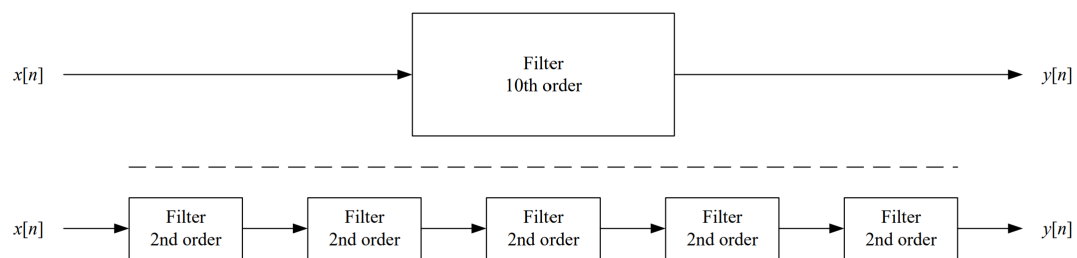


Figure 13: Increasing stability using 2nd order filter.

3 Implementation

The system is implemented through hardware and software. Software is however where the project's main focus lies. The system is designed in a way that allows the user to set the parameters of the filters freely, and the system will calculate the parameters and coefficients required to design the filter. The filter will then be run on the selected input channel using either the continuous or burst buffer mode.

The final system is composed of a microcontroller with the finished code, a custom experimental breadboard, a PicoScope that reads the system's output, and a computer to generate the filter's input signal and to view the result from the PicoScope. A computer running Microchip Studio and Flip is also used to alter the DSP filters. A visual representation of the system can be seen in Figure 14.

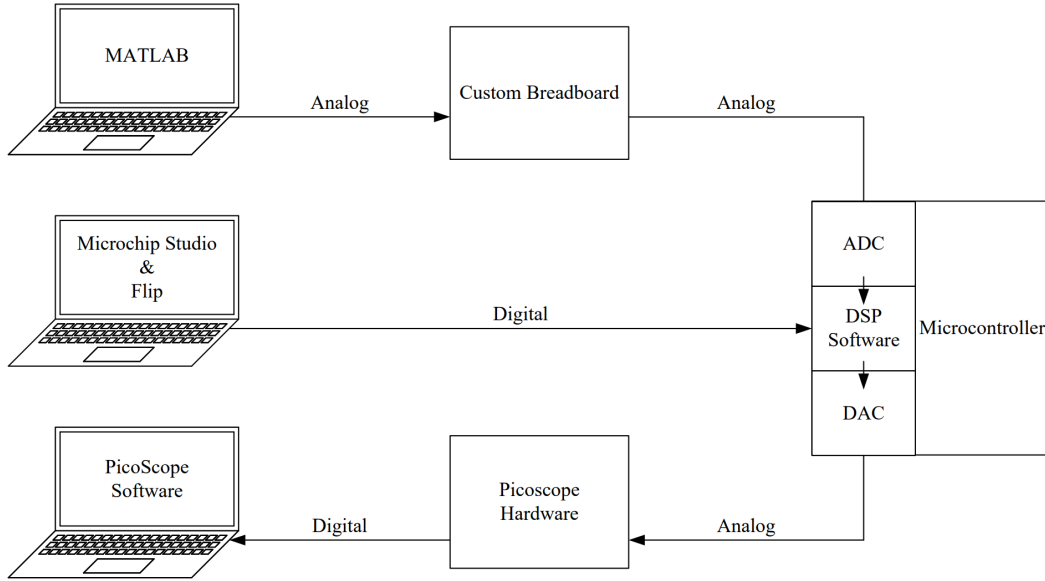


Figure 14: System overview.

3.1 Hardware

The used hardware devices perform or measure the results of the filter computations. The main devices used in the project are the microcontroller, the custom breadboard, the oscilloscope and a computer.

3.1.1 Microcontroller

The microcontroller used throughout the project is the Atmel UC3-A3 XPLAINED and is used as the computational center of the application. Its basic layout can be seen in Figure 15. The microcontroller uses universal serial bus (USB) interface to a computer for flashing new software onto the device. USB is an industry standard used by most

computational devices. The standard describes the specifications for cables, connectors and software in order for devices to be able to communicate with each other [1]. The interface RS-232 is also used for the microcontroller to communicate with a separate computer digitally to print text, numbers or symbols on its screen using the software Putty. RS-232 is a serial communication protocol for the transmission of data. It has low data transmission capabilities and can only be used with short cables. The RS-232 standard has widely been replaced by USB.

The USB connector is used to flash the microcontroller with new software. Header J2's ADC1 connector pin is used to measure the system's input signal. A 10-bit resolution and a maximum voltage of 3.6V is used for the ADC. The approximate resolution is 3.5mV and is calculated through (2). Header J3's VCC_P5V0 and GND connector pins are used to power the custom breadboard. Header J4's TXD and RXD-connector pins are used for the controller's RS-232 connection with a PC for digital output communication. J4's SCK channel also provides the processed analog printout from the system.

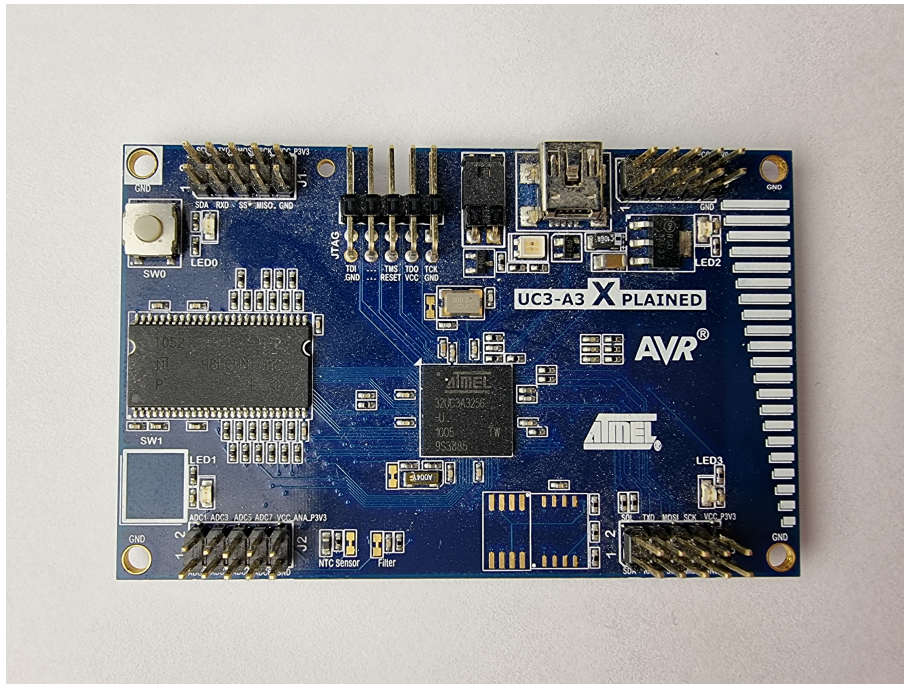


Figure 15: AVR layout.

3.1.2 Custom breadboard

The analog input signal is received through the custom breadboard through a 3.5mm headphone jack. The signal is led through an analog low pass filter built from resistors and capacitors, acting as an antialiasing filter that removes high frequencies. The signal is thereby pre-processed before it is transmitted to the microcontroller's analog input channel. The filter is driven by the operational amplifier LM358N. The board is able

to transmit one input and one output to and from the microcontroller simultaneously. The board and its layout can be viewed in Figure 16.

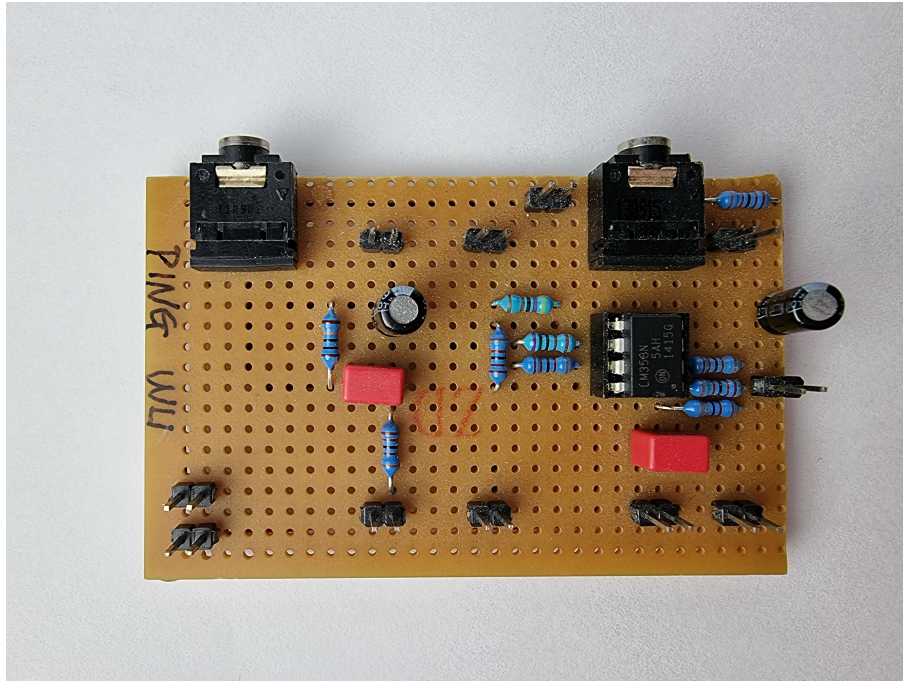


Figure 16: Custom breadboard.

3.1.3 Oscilloscope

The analog signals are measured using an oscilloscope in order to analyze them. The oscilloscope used is a PicoScope 2207A and does not have its own display. It does instead connect to a PC and uses its screen. It can simultaneously measure two independent channels. It has a resolution of 8 bits at 1 GS/s and a bandwidth of 100MHz [10]. It also has a signal generator that can be used to simulate certain signals. The oscilloscope can be viewed in Figure 17.



Figure 17: PicoScope 2207A.

3.2 Software

The software used in the project includes Microchip Studio, Matlab, Batchisp, Flip, and the project-specific C-code that was constructed in Microchip Studio.

- **Microchip Studio:** An Integrated Development Environment is a software that is used to develop applications. Microchip Studio allows for development in C, C++ or assembly code for microcontrollers. It supports most microcontrollers from Microchip. The software is a version of Microsoft's Visual Studio that has been specifically adapted for the purpose of creating code for microcontrollers. It includes basic methods and libraries to ease development and lets the developer to some extent focus more on the problem at hand. When a program, or solution, is compiled, it outputs a file of type .hex. It is used to flash the microcontroller using the software Batchisp.
- **Batchisp:** A part of the software package Flip, is used to load software onto the microcontroller using the USB interface. The class library USB Device Firmware Upgrade is used to upgrade the microcontroller's firmware without the use of specific programmer hardware. Flip requires the .hex-file that is outputted by Microchip Studio when a code is compiled as an input.

- Matlab: A software from MathWorks. It is both a program and a programming language. Matlab is based on mathematics and is often used for data collection, computation, visualization and simulation. Matlab has been used in order to create input signals for the system, which is combined sinus waves with different frequencies.
- Putty: In order to send data from the microcontroller back to the PC, the RS-232 communication protocol is used. The software Putty is used on the PC to receive the information sent by the microcontroller. Its interface is similar to the Windows command prompt where communication only is done through characters, not through a graphical interface. Putty has only been used as a means of one way communication.

3.2.1 Application Software

The filters and buffer logic are written in C-code using Microchip Studio. The code is modular and divided into several different methods. There are methods that handle the DMA, convert data types and initiate the board. The mathematical operations sinc and convolution were written as methods in the code. The convolution is processed through an operation called the Cauchy product. This is to avoid transforming the data to other domains, which would be computationally heavier. There is also a pre-processing method for the data which can be used to prepare the input data for the filters. There is one method for the design of FIR filters and one for applying them. There is also one method for the design of IIR filters as well as one for applying them. The design method is solely called once when the program initiates while the applying method is run according to the buffer settings.

The application's main method calls the required sub-methods to perform each DSP step. It also contains variables that can be set to change parameters to the filters. They can change between FIR and IIR filters, change cut-off frequencies, number of poles and decide between low pass, high pass, band pass or band stop filter.

3.2.2 Finite impulse response application

A FIR filter's coefficients need to be found first when it is applied. Through (11), (12), (13) and (14), its ideal impulse response is found in the data type *double*. Chapter 2.4.3 then explains how the window coefficients are found. The application software uses the native methods *dsp16_win_rect*, *dsp16_win_hamm*, and *dsp16_win_hann* in the design of each window of data type *uint16_t*. The windows are converted to the data type *double* and adjusted to reduce the maximum gain to 1. The ideal impulse response coefficients are then multiplied with the window, forming the windowed impulse response, $h[n]$, used when applying the filter. The length of $h[n]$ is decided from the system's number of poles. The digital input data is received from the ADC in the data type *uint16_t*. The data is converted to *double* and the DC offset is removed before (8) is applied, which is a method for computing the convolution between two discrete

data sets [6]. The method is called a Cauchy product and is used to calculate discrete convolution without the use of Fourier transformations. This calculation is done entirely in data type *double*. When the filter application is complete, the DC offset is reintroduced and the result is converted back to data type *dsp16_t*. The data is then sent to the DAC for analog reconstruction. This method is visualized in Figure 18.

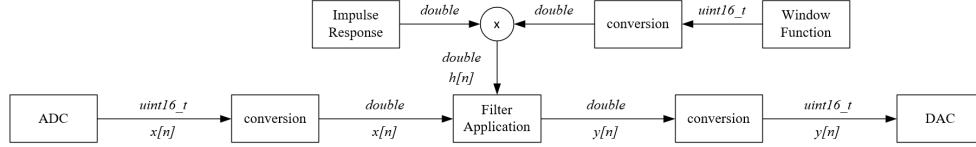


Figure 18: FIR application data flow.

The process of applying the FIR filter can be visualized through a block diagram of the system, seen in Figure 19. The $x[n]$ array is the input, and the top blocks labeled z^{-1} represent a unit delay. The middlemost triangles labeled as b_n , where $n = 1, 2, 3, \dots$, represent the filter's coefficients. Together they form the output $y[n]$ after the summation that can be seen at the bottom of the figure.

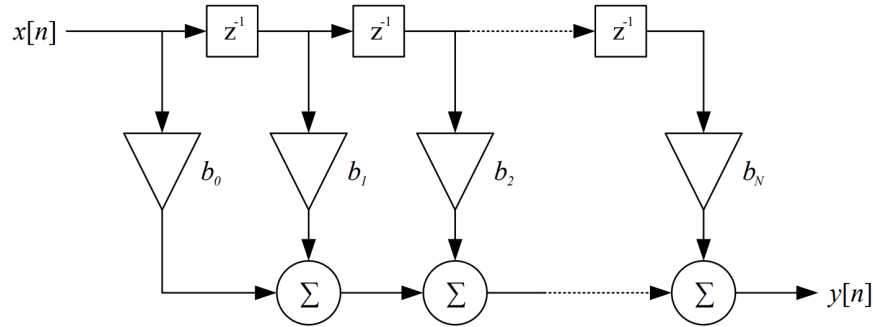


Figure 19: A FIR filter's block diagram.

3.2.3 Infinite impulse response application

A two-layered loop is used When applying the filter and finding the output through 19. The inner loop iterates from zero to the number of coefficients while the outer loop iterates from one number less than the number of coefficients, to the number of data input values. The outer loop is corresponding to the first summation in 19, where the outer loop corresponds to the second. The results from the calculations are used as the system output.

To apply the IIR filter according to (19),

$$y[n] = a_i \cdot x[n] \quad (48)$$

is calculated for the first coefficient, and

$$y[n] = y[n] + a_i \cdot x[n - i] + b_i \cdot y[n - i] \quad (49)$$

is calculated for the remaining [4]. The arrays a_i and b_i contain the numerators respectively denominators from the filter's difference equation (6). The method is performed through a two-layered loop, with iterating variables i and n . The inner loop iterates the variable i from zero to the number of coefficients while the outer loop iterates the variable n from one number less than the number of coefficients to the number of data input values. The results from the calculations are stored in the array y .

When an IIR filter is applied, its coefficients first need to be found. The operations described in Sections 2.5.2 and 2.5.3 are applied to find the filter's coefficients in data type *double*. The length of $h[n]$ is decided from the system's number of poles, although a band pass or band stop filter will have twice the number of coefficients compared to low pass and high pass filters. The digital input data that is received from the ADC is of the data type *uint16_t*, but is converted to *double* before the DC offset is removed. The filter is applied to the input data according to the IIR's differential equation that can be seen in (19). The filter output is of data type *double*, but is converted to *uint16_t* after the DC offset has been reintroduced. The data is then sent to the DAC for analog reconstruction. This method can be visualized in Figure 21.

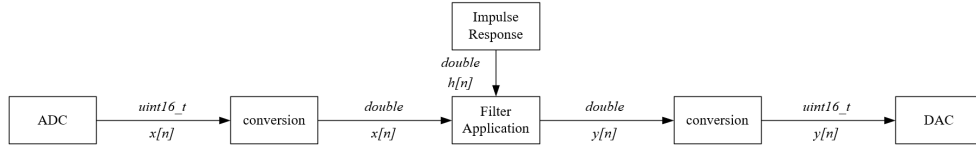


Figure 20: IIR application data flow.

A general n th order IIR filter's structure can be illustrated using a block diagram seen in Figure 21. The $x[n]$ array represent the data digital input data and the blocks labeled z_{-1} represent a unit delay. The triangles labeled as a_n and b_n , where $n = 1, 2, 3, \dots$, represent the coefficients from the filter design. Together they form the output after the summation that can be seen at the top and middle of the figure.

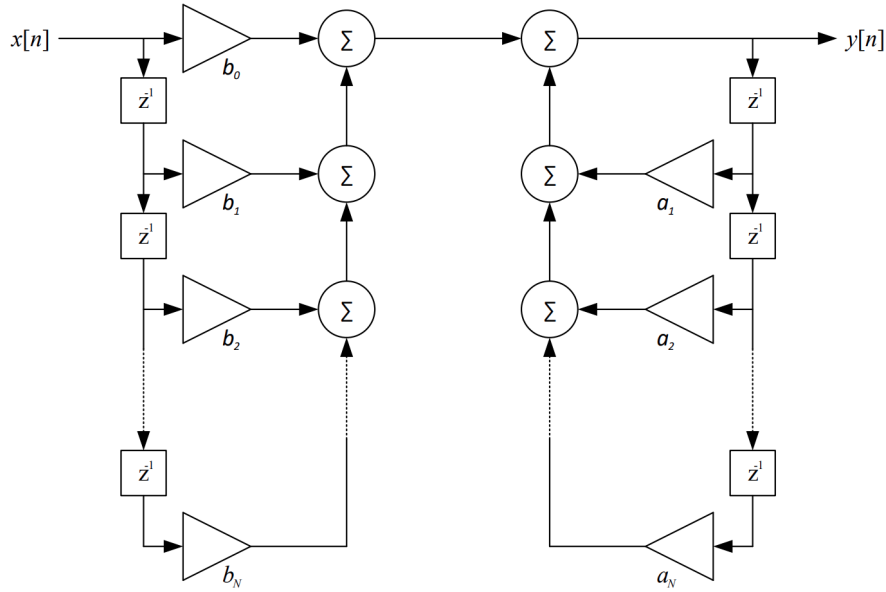


Figure 21: An IIR filter's block diagram.

However, when using multiple second order sections to construct a more stable filter according to Section 2.5.4, the block diagram would instead look like Figure 22. One dashed encasement represents one second order filter. The number of encasements that are put in series is selected in accordance with how many second order sections are to be used, discussed in Section 2.5.4.

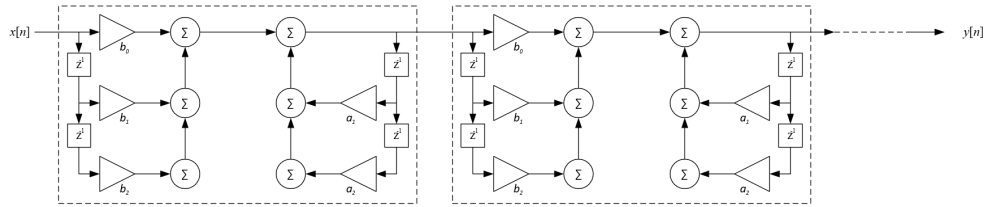


Figure 22: An IIR filter's block diagram when it is designed with second order sections.

4 Results

4.1 Evaluation setup

The setup used for evaluating the system includes of a PC running

- Batchisp: Used to flash the AVR with new application software, containing the filter parameters that are to be executed.
- Matlab: Running on a PC, outputting an audio signal that is used as an input for the filter.
- Custom breadboard: The signal is run through an antialiasing filter before the AVR.
- AVR microcontroller: Running the application software, using the signal from the custom breadboard as input and outputs a filtered signal.
- PicoScope: Using software running on a PC, the device is the oscilloscope used to measure the analog filter output.

The results are shown through graphs plotting each filter's frequency response. The data was collected by simultaneously measuring the filter's analog input and output. Sine signals of different frequencies were sent through the filter using Matlab, where the input and output amplitudes were measured using the Picoscope and a PC. The filter input and output signals were measured simultaneously for different frequencies and their amplitudes were recorded. The frequency response plots were then plotted in dB in Sections 4.2 and 4.3, calculated through [5]

$$dB = 20 \log_{10} \frac{V_{output}}{V_{input}}. \quad (50)$$

The frequencies used for the input signal range from $500Hz$ to $20kHz$. Signals are then used with frequencies increasing with $1kHz$, starting at $1kHz$, and ending at $17kHz$. Used input frequencies are also denser within the transition band of the filters, with a minimum difference of $250Hz$ between each.

The plots show how the filter passes the signal through at different frequencies. A dB value of 0 lets the signal pass through without dampening, while the attenuation increases with a lower value. All results are found using the sampling frequency of $46875Hz$ and the Nyquist frequency is determined by (1), and found to be

$$\frac{46875Hz}{2} = 23437.5Hz. \quad (51)$$

The buffer was in burst mode, and the buffer size was 1024 samples. The order and cut-off frequencies used for evaluating the FIR filters are as presented in Table 1. The

poles, cut-off frequencies, and the ripple used to design the IIR filters are presented in Table 2. Only one order of each filter is presented as the focus of the result is to compare the different methods' performance rather than finding the optimal settings for each filter.

Table 1: The pole and cut-off frequencies for FIR filters.

	Order	$f_{cLP}[Hz]$	$f_{cHP}[Hz]$
LP	10	7000	
HP	10		7000
BP	10	6000	10000
BS	10	6000	10000

Table 2: The pole and cut-off frequencies for IIR filters.

		Poles	$f_{cLP}[Hz]$	$f_{cHP}[Hz]$	Ripple [percent]
Butterworth	LP	10	7000		0
	HP	10		7000	0
	BP	8	6000	10000	0
	BS	8	6000	10000	0
Chebychev	LP	10	7000		15
	HP	10		7000	15
	BP	8	6000	10000	15
	BS	8	6000	10000	15

The parameters of the filters are intentionally kept similar in order to ease comparisons between the different filter methods, as well as somewhat close to the middle of the frequency band usable by the system. For FIR filters, the number of poles is kept identical among the filters, as well as cut-off frequencies between the windowing methods. The number of poles is selected such that the filter's performance does not increase significantly with an increased number of poles. This is to keep a balance between performance and a light computing load to run the filter.

For IIR filters, most variables are kept identical between the Butterworth and Chebychev filters, except for the ripple. It is selected to 15% for the Chebychev filter in order to add a substantial amount. Ten poles are selected for the low pass and high pass filters, similar to the FIR filters. The band pass and band stop filters are designed with eight poles to give each of the two underlying low pass and high pass filters in the design process an even number of poles.

4.2 Finite impulse response filters

The frequency responses of the FIR filters are shown below. The low pass, high pass, band pass, and band stop filters of each window type are shown in the same graph using

different colours. The settings stated in Table 1 are used. The measured values from the filter with the different windows are compared to simulated filters with Hamming windows, using the same parameters as the measured ones.

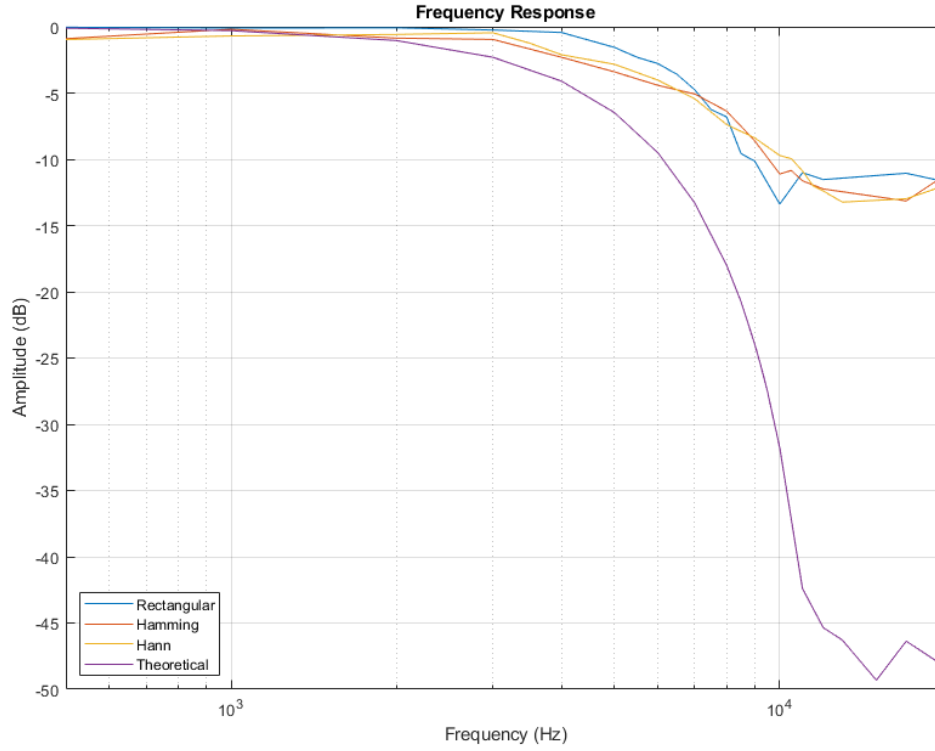


Figure 23: Frequency response of low pass FIR filters.

The frequency responses of the low pass FIR filters are shown in Figure 23. The measured filters do have somewhat similar characteristics. The rectangular window does give a steeper transition band than the Hamming and Hann windows, although it also contains more ripple in the stop band. The simulated filter does have more attenuation in the stop band.

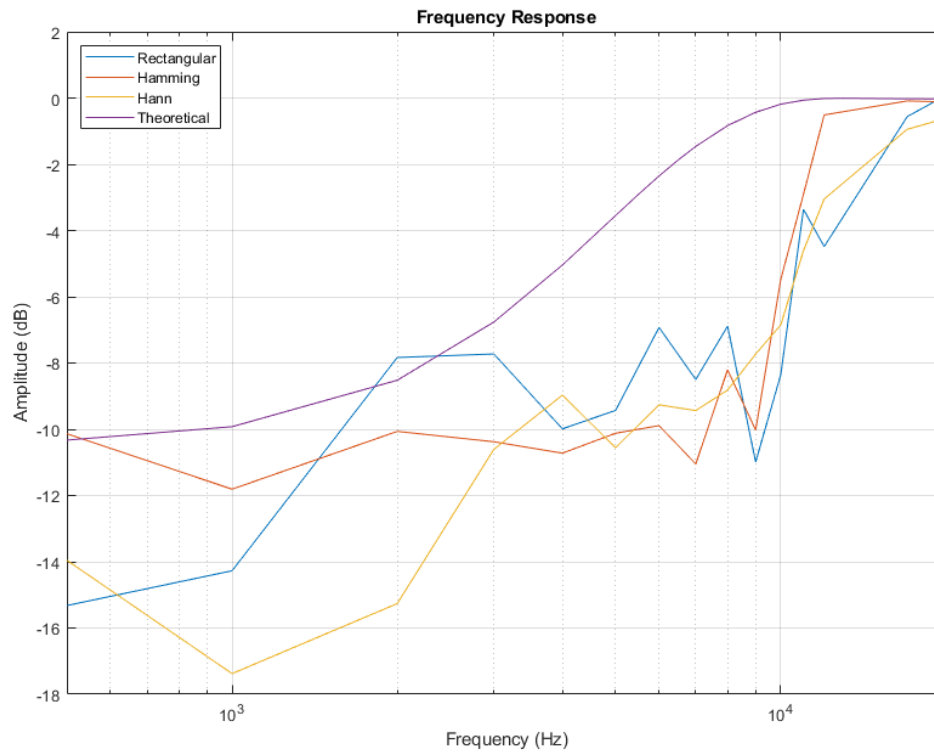


Figure 24: Frequency response of high pass FIR filters.

The frequency responses of the high pass FIR filters are shown in Figure 24. The measured filters do follow the trend of the simulated filter, with the hamming window being closest to the theoretical one. A substantial amount of ripple can however be observed in each measured filter, although the rectangular filter has the most.

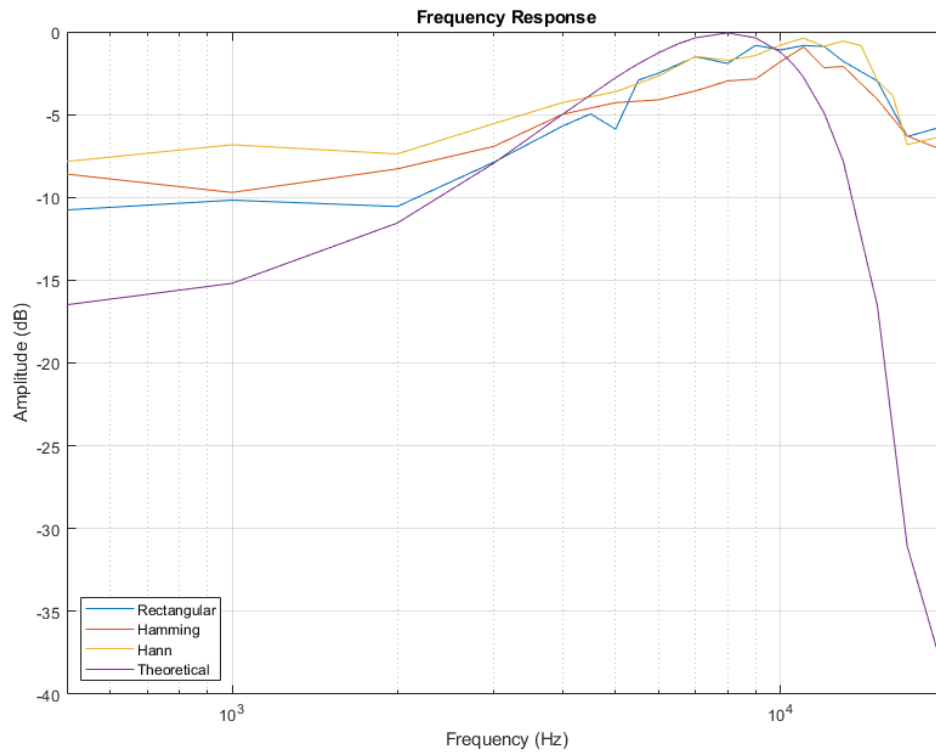


Figure 25: Frequency response of band pass FIR filters.

The frequency responses of the band pass FIR filters are shown in Figure 25. The measured filters follow the characteristic of the theoretical filter. The pass band is slightly shifted towards a higher frequency than the theoretical filter, which also has a higher attenuation.

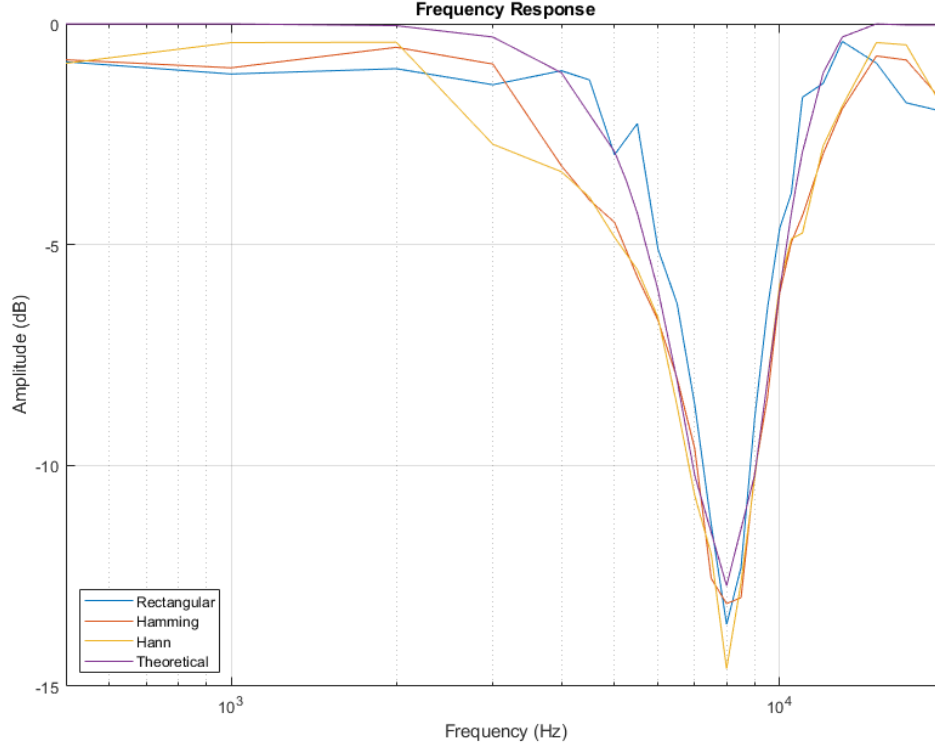


Figure 26: Frequency response of band stop FIR filters.

The frequency responses of the band stop FIR filters are shown in Figure 26. The measured filters follow the theoretical filter well. The Hamming and Hann windowed filters have a flatter transition band than the rectangular windowed and theoretical filters. The rectangular windowed filter does however introduce more ripple than the others.

4.3 Infinite impulse response filters

The frequency responses of the IIR filters are shown below. The low pass, high pass, band pass and band stop filters of each window type are shown in the same graph using different colours. The settings stated in Table 2 are used. The measured values from the filters are compared to simulated Butterworth filters using the same parameters as the measured ones.

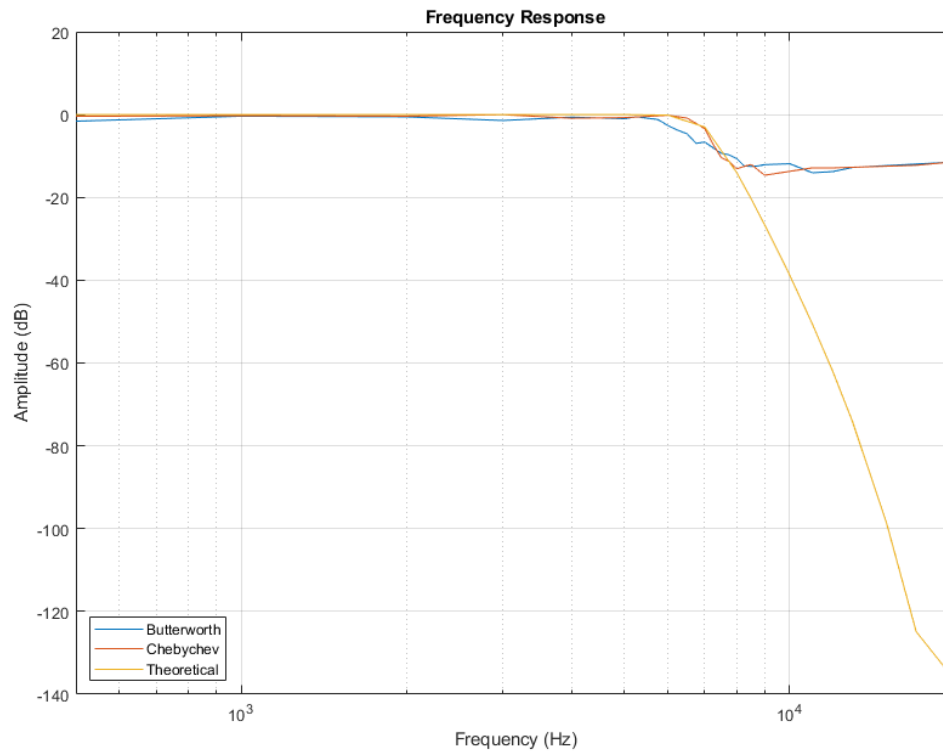


Figure 27: Frequency response of low pass IIR filters.

The frequency responses of low pass IIR filters are shown in Figure 27. The Butterworth filter shows a less steep transition band that also starts to attenuate signals at a lower frequency than the Chebyshev filter and the simulated one. The Chebyshev filter does however follow the theoretical filter well until its maximum attenuation is reached.

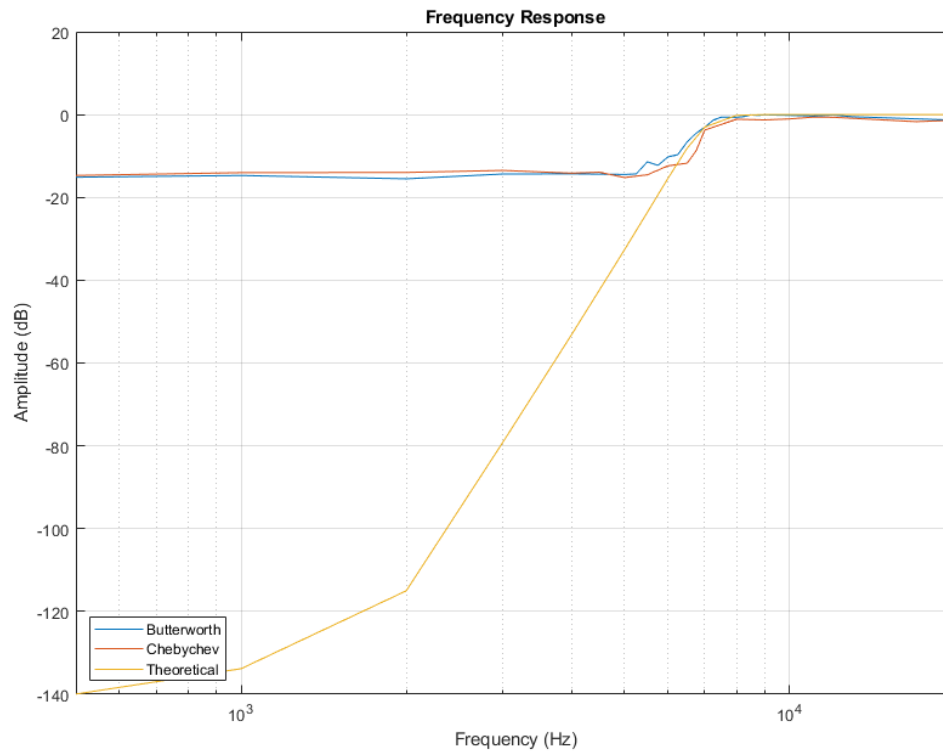


Figure 28: Frequency response of high pass IIR filters.

The frequency responses of high pass IIR filters are shown in Figure 28. The Butterworth filter does have a slightly flatter transition band and introduces more ripple than the Chebyshev and the theoretical filter. Both measured filters follow the theoretical filter well before they reach their maximum attenuation.

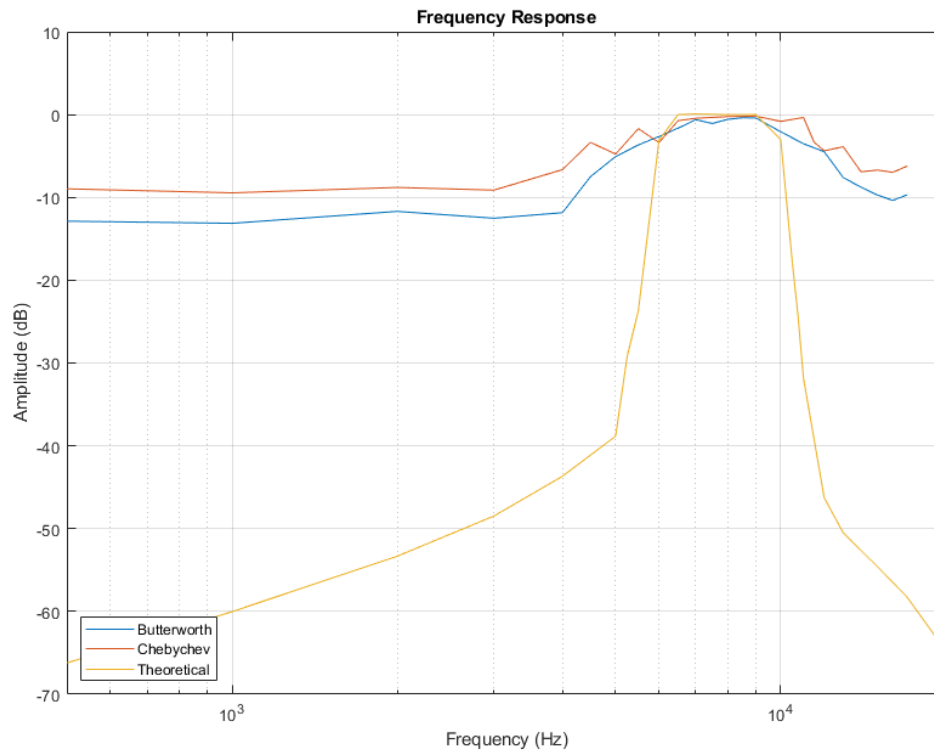


Figure 29: Frequency response of band pass IIR filters.

The frequency responses of band pass IIR filters are shown in Figure 29. The Butterworth filter and Chebychev filter have a wider transition band than the theoretical filter. The Chebychev filter also introduces ripple to its pass band and reaches a lower attenuation than the Butterworth filter. The theoretical filter reaches a substantially higher damping on its stop bands.

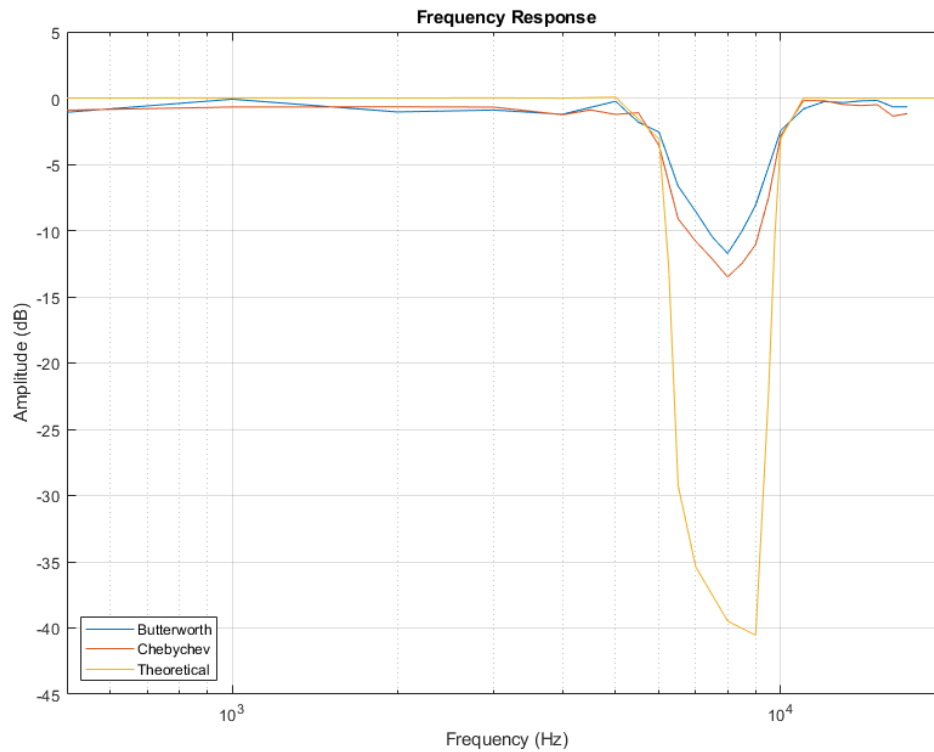


Figure 30: Frequency response of band stop IIR filters.

The frequency responses of band stop IIR filters are shown in Figure 30. The transition band is steeper in the theoretical filter than the Butterworth or the Chebyshev filter, which have similar properties. The Chebyshev has a steeper transition band, and reaches a higher attenuation.

5 Discussion and conclusions

The project's goal was to design an easy to use digital signal processing system. The system was implemented on a microcontroller with digital filters, buffering logic, and conversions between analog and digital data. The system's analog outputs and inputs were measured using an oscilloscope. The results shown in Section 4 are given as frequency response plots. All subsystems are required simultaneously for the system to output a filtered signal, including the ADC, buffering logic, filters, and DAC. The filters are applied individually using samples with different frequencies, covering the characteristics of the filters from $500Hz$ to $20kHz$.

The results for the FIR filters are of varying quality. The low pass filters perform well in removing the higher frequencies, although the rectangular window introduces ripple. The high pass filters do, however, generally perform worse. All windows introduce a substantial amount of noise, and the pass band is shifted too high in frequency. At the cut-off frequency of $7kHz$, the attenuation of each measured filter is too high for a satisfactory design. The filters do not let signals through before they reach a frequency of $10kHz$. Although the Hamming window performs the best, there is too much noise for its origin to be filter ripple. This is seemingly caused by a poor filter definition in (12) as all window types are affected similarly, and this particular issue only is present in the FIR high pass filter. The band pass filters are also shifted up in frequency for all window types, but not as much as the high pass filters. There is ripple introduced in the measurements regardless of filter type. The band stop filters perform well for all window methods where the middle frequency has been removed, preserving the lower and higher frequencies. The rectangular window does however introduce more noise and ripple than the Hamming and Hann windows. The FIR filters are generally easy to design, both with respect to the complexity of the filters' logic and the computational demand of the processor. They are however demanding in their application in the same aspects.

The results for the IIR filters are also of varying quality. The low pass and high pass filters have similar characteristics, where the Butterworth filters' attenuation initially is close to the simulated filter's. Butterworth has a wider transition band but introduces less ripple. The measured band pass filters have a wider pass band than what is seen in the simulated filter, and their transition band is much wider. The Chebychev filter introduces much ripple, while it never reaches the attenuation of the Butterworth filter. The measured band stop filters have well-located transition bands, but they are sharper in the Chebychev filter. All of the filters successfully pass the part of the signal that is inside of the filter's pass band through. The outputs of the Butterworth and Chebyshev filters do look similar in many cases, showing only minor differences. A Chebychev filter often demonstrates a sharper cut-off and more introduced ripple. The measured filters have a maximum attenuation of around $-10dB$, while the simulated filter's maximum attenuation is around $140dB$. The high attenuation of the simulated filter is mainly caused by the absence of noise in the model. This causes the system output signal to have an amplitude so small that it is not realistic to recreate outside of the simulation.

Secondly, the low attenuation of around $-10dB$ of the measured filters is caused by noise introduced in the system. The IIR filters are generally complex to design, both with respect to the complexity of the filters' logic and the computational demand of the processor. They are light in their application in the same aspects.

Most of the filters are behaving favorably with respect to frequencies, but their attenuation is generally low compared to the theoretical filters. The low noise of the measured systems is caused by noise introduced. When an input signal that is far within a system's stop band is used as an input, it is not possible to distinguish the output from one where the filter has no signal as input at all, shown in Figure 31. The output in both cases appears to be unrelated to the input signal. This causes the output signal to never reach a voltage level close enough to 0 for the attenuation to increase further in the measurements, even though the higher voltage level is caused by other factors than the filtered input data. The signal gets less noisy when outputting the signal digitally using USART, suggesting that most noise is introduced by the DAC. As the theoretical filters do not introduce any artificial noise, its theoretical attenuation continues to increase for the span in which it is plotted. Its signal-to-noise ratio does not increase within the stop band in the way that it does for the filters running on the AVR. The gain in the pass band is not always zero. A variable gain could be implemented as a scaling factor that is multiplied by the output before it is sent to the DAC. The mathematical operation would be simple, but one more variable that might not be intuitive would also have to be introduced for the user to interact with. This would decrease the ease of use of the system since it does not have a graphical user interface.

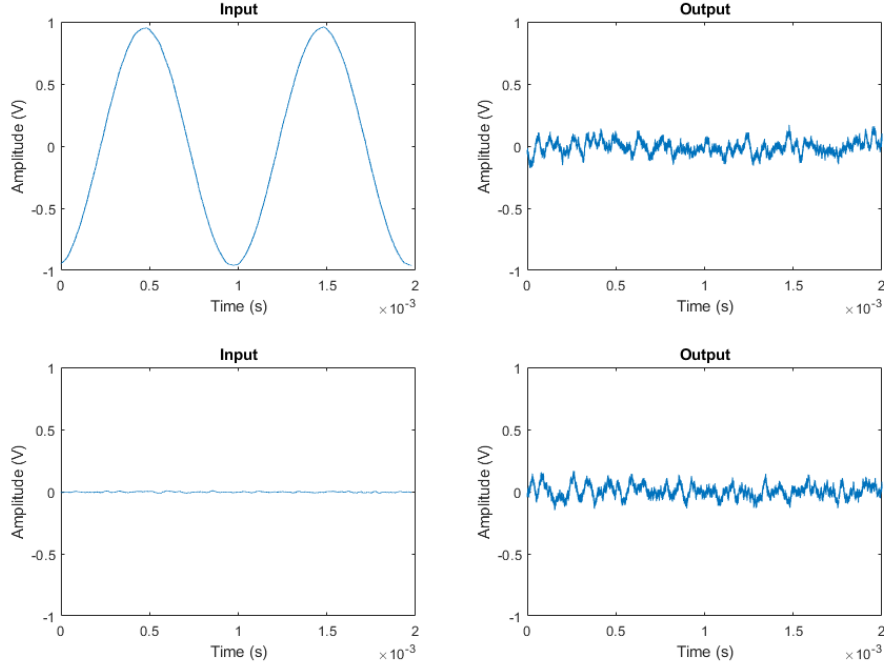


Figure 31: The upper two subplots show the input and output of a high pass Butterworth filter with specifications stated in Table 2 when a signal of 1kHz is applied. The lower two subplots show the input and output of the same system when no signal is applied.

Although the filter settings are easy to change in the system's code, the code itself may not be considered easy to change. The variables are collected at the beginning of the system's main method and are well documented, and thus not hard to find or alter. A graphical user interface should, however, be implemented in order to create a truly easy-to-use system for someone that is not involved in the project. When a filter is altered, the relevant variable needs to be found in the C-code and changed, and a valid value must be entered. The project code then needs to be recompiled before it is flashed onto the microcontroller. The filter with altered characteristics may then be applied to a signal. There are two major drawbacks to the current solution, apart from the inconvenience of the many steps. Firstly, the source code needs to be distributed for the system to be usable. This would be a big barrier of entry for the system as installation of multiple PC softwares would be required to compile the application software and flash it onto the AVR. The process of installing these PC softwares require guides and depends on them to remain available and compatible with new computers in the future. Secondly, the user may unintentionally change values in the code which causes the system to malfunction. Such changes are easy to make by mistake and may be hard to troubleshoot, as well as to revert. It would be possible to show user guides within the program if a proper graphical user interface would be implemented. For

example, the user may be informed of the Nyquist frequency when selecting a sampling frequency, or information may be given on how to select the number of poles for a filter.

Some of the limitations of the system are related to the hardware used. The hardware mainly has limited precision in the modules for ADC and DAC, which alters the quality of the result. The limitations cause the noise gradually to increase and the signal to get weaker. It can be seen in the results that the output signals consistently have lower attenuation than the theoretical filters. This difference is caused by noise in the output signal that is being introduced throughout the system. The system's sampling frequency and resolution play a role in distorting the input signal in the ADC and noise will inherently be introduced. This phenomenon can be illustrated by digitally printing the input data array from the system and comparing it to an analog measurement of the input. The DAC faces similar dilemmas as the ADC, introducing artifacts due to sampling frequency, resolution and noise. This distortion can be visualized in the same way as for the ADC. The quality of the ADC and DAC may be improved using different hardware. External hardware modules for ADC and DAC may be used with the current hardware platform for higher quality measurements. A different microcontroller with higher quality ADC and DAC could also be used to improve the system's accuracy. Though the noise introduced can be lowered, it cannot be eliminated completely.

Further work that could be made to increase the quality of the system would be to revise the design method of the filters further. Mainly the FIR high pass filter and IIR band pass filter would need to be revised and optimized. The noise introduced by the system should also be further investigated, potentially by using hardware containing higher quality ADC and DAC, a general user interface should be implemented, and the system may be expanded with more filter design methods.

6 References

- [1] Atmel. *Datasheet 32-bit AVR Microcontroller*. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/doc32072.pdf>. (Accessed: 2023.04.09).
- [2] Robert Oshana. “DSP Software Development Techniques for Embedded and Real-Time Systems”. In: (2006). URL: <https://www.sciencedirect.com/science/article/pii/B9780750677592500247>. (Accessed: 2023.04.09).
- [3] Microchip. *Getting started - USART Asynchronous*. URL: <http://ww1.microchip.com/downloads/en/devicedoc/usart.pdf>. (Accessed: 2023.04.09).
- [4] Steven W. Smith. “The Scientist and Engineer’s Guide to Digital Signal Processing”. In: (Mars 1, 1998). URL: <http://www.dspguide.com/>. (Accessed: 2023.04.09).
- [5] Torkel Glad and Lennart Ljung. *Reglerteknik*. 2016.
- [6] Douglas F. Elliott. “Handbook of Digital Signal Processing”. In: (1987). URL: <https://www.sciencedirect.com/science/article/pii/B9780080507804500023>. (Accessed: 2023.04.09).
- [7] Mathworks. *Rectangular window*. URL: https://se.mathworks.com/help/signal/ref/rectwin.html#mw_9a267489-7763-4e7f-bee4-4392392f48b4. (Accessed: 2023.04.09).
- [8] Mathworks. *Hamming window*. URL: <https://se.mathworks.com/help/signal/ref/hamming.html>. (Accessed: 2023.04.09).
- [9] Mathworks. *Hann (Hanning) window*. URL: <https://se.mathworks.com/help/signal/ref/hann.html>. (Accessed: 2023.04.09).
- [10] Pico Technology. *Data Sheet PicoScope 2000 Series*. URL: <https://www.picotech.com/download/datasheets/picoscope-2000-series-data-sheet-en.pdf>. (Accessed: 2023.04.09).