CHARACTERIZATION OF A DIGITAL LOWPASS FILTER'S FREQUENCY RESPONSE

Lappeenranta–Lahti University of Technology LUT

Master's Programme in Electrical Engineering, Master's thesis

2024

Anssi Ronkainen

Examiners: Professor Pertti Silventoinen

Teemu Salo, D.Sc. (Tech.))

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

LUT School of Energy Systems

Electrical Engineering

Anssi Ronkainen

Characterization of a digital lowpass filter's frequency response

Master's thesis

2024

53 pages, 50 figures and 2 tables

Examiners: Professor Pertti Silventoinen and Teemu Salo, D.Sc. (Tech.))

Keywords: lowpass filter, characterization, frequency response, ASIC

This Master's thesis focuses on characterization of a manufactured ASIC, specifically the frequency response of one of its filters. The objective of this thesis was to measure and analyze a post-processing filter's frequency response to see if it passes the internal and customer specification.

Like in many other applications, the simulated operation of an application specific integrated circuit (ASIC) might not match the manufactured ASIC's operation. The ASICs must be characterized after being manufactured to know how well they operate compared to the simulations.

The test was conducted by sending an impulse to the filter via ASIC control register and measuring the response from the output register. Impulse response was then processed in Matlab to analyze the filter's frequency response.

The results showed that the designed response matched the manufactured ASIC's response well, but the ASIC's rather low sampling frequency reduced the accuracy of the results, especially timing-wise.

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUT Energiajärjestelmät

Sähkötekniikka

Anssi Ronkainen

Digitaalisen alipäästösuodattimen taajuusvasteen karakterisointi

Sähkötekniikan diplomityö

2024

53 sivua, 50 kuvaa ja 2 taulukkoa

Tarkastaja(t): Professori Pertti Silventoinen ja TkT Teemu Salo

Avainsanat: alipäästösuodatin, karakterisointi, taajuusvaste, ASIC

Tässä diplomityössä tutkittiin valmistetun ASIC karakterisointia, tarkemmin yhden sen suodattimen taajuusvastetta. Tämän työn tavoite oli mitata ja analysoida jälkikäsittelysuodattimen taajuusvaste ja tutkia täyttääkö se sisäiset ja asiakkaiden vaatimukset.

Kuten monissa muissakin tapauksissa, sovelluskohtainen integroitu piirin (ASIC) suunniteltu toiminta ei välttämättä vastaa valmistetun ASIC:in toimintaa. ASIC:it tulee karakterisoida valmistuksen jälkeen, jotta tiedetään, kuinka ne toimivat verrattuna simulaatioihin.

Taajuusvaste mitattiin lähettämällä suodattimelle impulssi ASIC:in ohjausrekisteriin ja mittaamalla vaste ulostulorekisteristä. Tästä saatu ilmpulssivaste prosessoitiin Matlabilla, jotta voitiin analysoida suodattimen taajuusvastetta.

Tulokset osoittivat, että simulaatiot vastasivat valmistetun ASIC:in vastetta melko hyvin, vaikka ASIC:in alhainen näytteeottotaajuus laski tulosten tarkkuutta, erityisesti ajoitusten suhteen.

Abbreviations

6DOF	Six degrees of freedom, all possible movement axes
ASIC	Application Specific Integrated Circuit
BIBO	Bounded input, Bounded output
DUT	Device Under Test
FFT	Fast Fourier Transform
FIR	Finite Impulse Response, filter type
Gyro	Gyroscope, a device for measuring rotational speed
IC	Integrated Circuit
IIR	Infinite Impulse Response, filter type
LSB	Least Significant Bit
LTI	Linear Time Invariant
MSB	Most Significant Bit
PP	Post-processing
ROC	Region of Convergence
SEQ	Sequence, Labview file format for calling VIs
SPI	Serial Peripheral Interface
TDMS	Technical Data Management Streaming, Labview data file format
VI	Virtual Instrument, Labview code file format

Table of contents

Abstract

Table of Contents

Tables

1 Introduction

This thesis is about ASIC digital filter characterization. It focuses on characterization of frequency response of gyroscope post processing filter for Murata Finland's new River ASIC used in new SCH163X combo sensor series with six degrees of freedom (6DOF) inertial sensing. This study was conducted to verify and characterize the post processing filters frequency operation in real life. Previously the frequency response was only tested in simulations thus it was necessary to test the manufactured ASIC to gain knowledge of the operation, how it compares to simulation, if the operation is at a level that is acceptable for the completed component with sensing elements, and if it would be ready for customer use.

The filter was designed in Matlab and later tested in ASIC simulation software. The results of the simulation were promising, and it was decided to continue with the design. The simulation software can emulate real life situations, but of course this has limits. The software allows very precise timing for measurements and does not require any additional equipment. In simulations it is possible to set register values and voltages inside the ASIC freely and without any delay. When dealing with a real-life ASIC, test voltages need to be set externally and possibly routed inside the ASIC and register operations need communication with the ASIC to achieve the same operation. This is where the need for this study comes from.

To test the actual ASIC, first we need the manufactured ASIC, voltages set to power it on and communication between the ASIC and an electronic system via SPI protocol. When all of these are combined with real life noise and manufacturing tolerances, it is by no means certain that the simulations alone would provide enough information about the ASIC operation.

This thesis' objective was to verify that the newly designed ASIC would fulfill the customer and internal requirements for gyroscope post processing filter. For this measurement specialized equipment and software was needed. While the measurement equipment provides good results, they do not allow precise timing or similar data handling capabilities as simulation software provides, which makes testing and result handling more complicated.

The thesis is split into six different parts. We begin with signals and signal processing and then advancing into theory about digital filters. After these the test setup is discussed, which leads to results of the measurements and lastly conclusions of the test are at the end of this thesis.

2 Signals and signal processing

Signals are everywhere. Some of them can be seen and some of them are invisible. A signal can be expressed as a function of independent variables like time or temperature. For example, electrocardiogram, or ECG, is a heart's electrical signal as a function of time or a lamp's brightness can be expressed as a function of current flowing through the lamp.

Signals can be created naturally, synthetically or by computer simulation. All signals contain information, but not all information is useful. To get the information we need, a signal of interest must be chosen and then signal processing can begin. In signal processing mathematical representation of the signal is needed. This, however, contains all the information the signal holds. An algorithm can be created to remove all the unnecessary information contained by the signal thus giving us the information we want from the signal. This information may be in the form of the original signal and be in the same domain, or it can be in a transformed domain for easier processing. (Mitra, 1998)

2.1 Signals

Signals can have multiple independent variables. This means that their contained information is dependent on one or more variables. A one-dimensional signal, a signal that has one independent variable, is usually dependent on time. For each value of time, there is a signal value. If there are no delays between the values and all values of time are present, the signal is a continuous-time signal. The signal is called an analog signal if the signal value (amplitude) is also continuous. Analog signals are often created naturally. Speech, seismic activity, and ECG are all examples of analog signals. Since analog signals have all values of time, meaning there is an infinite number of time values, it cannot be measured. All we can do is measure the amplitude and the time value of the signal and take measurements often to represent the signal closely enough to resemble the original continuous-time signal. By doing this, we have created a digital signal. An example of this can be seen in figure 1. In the upper part of the figure, we can see a sine wave that is plotted over time, where $0 \leq t \leq 2$. Since all values of time are present in the signal, it is an analog signal. In the bottom of the figure, we can see a digital signal made from the above analog signal. The digital signal is not continuous and has only values at a certain interval, in this case every interval of 0.1. This

process is called discretization, and the signal is now a discrete-time signal instead of continuous-time signal. Each value pair of the digital signal is called a sample. A continuous-time signal is usually presented as $x(t)$, where x is the signal and t is the independent variable time. A discrete-time signal is presented as $x[n]$, where x is the signal and n is the number of the sample.

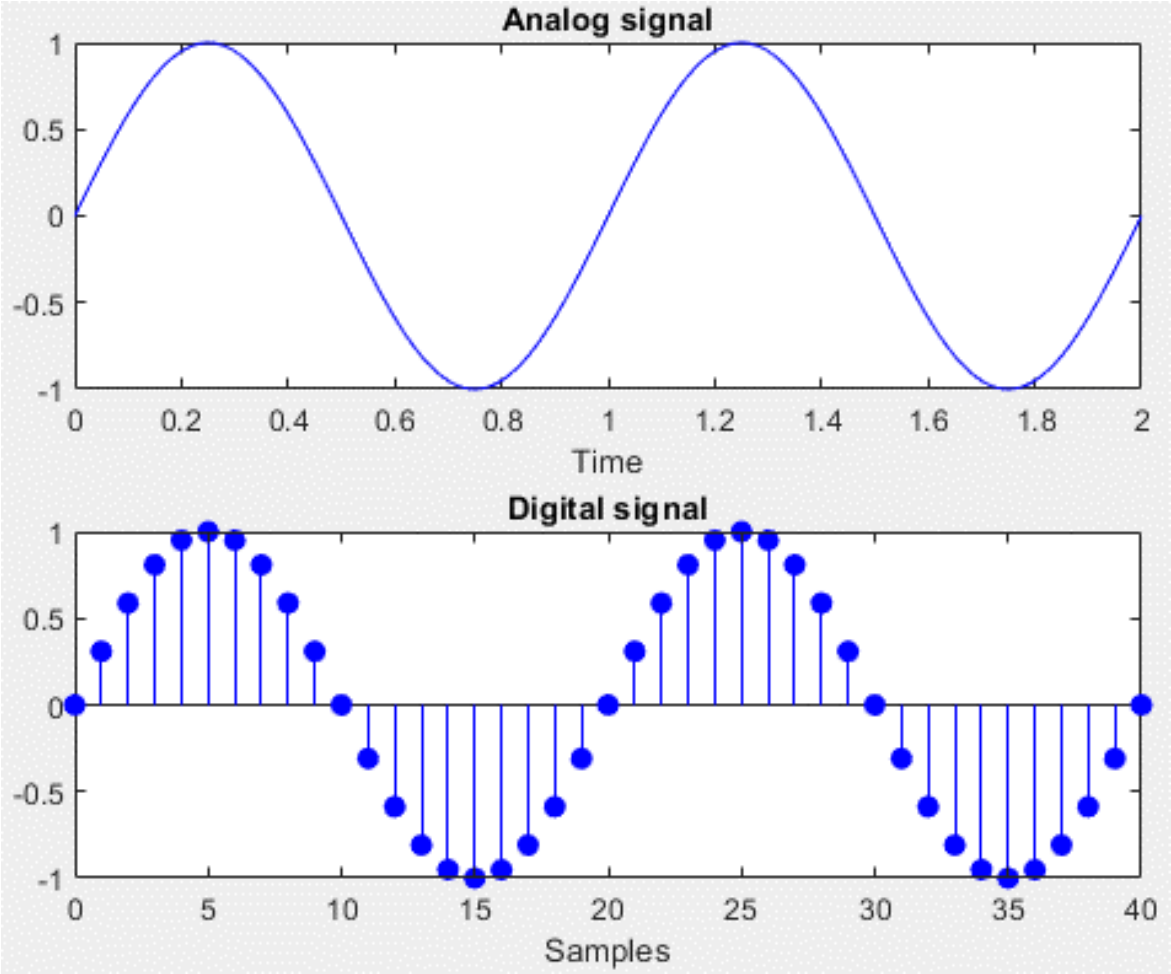


Figure 1: Above: Analog signal sin(2πt), below: above analog signal as discretized to a digital signal. Graphs made with Matlab.

(Mitra, 1998)

2.2 Discrete-time signals

Discrete-time signals can be formed by sampling a continuous-time signal $x_a(t)$ with sampling period of T and time T is usually constant.

$$x[n] = x_a(t)|_{t=nT} = x_a(nT), \qquad n = \dots\ , -2, -1, 0, 1, 2, \dots$$

When T is constant, we have a sampling frequency of F_s and the unit of F_s is samples per second or hertz (Hz).

$$F_s = \frac{1}{T}$$

As seen in Figure 1, discrete-time signals have only a limited number of values and thus discrete-time signals are called sequence of numbers.

The value of $x[n]$ can be real or imaginary for any n, which is often called sample. This means that we can write the sequence as

$$\{x[n]\} = \{x_{re}[n]\} + j\{x_{im}[n]\}$$

Where $x_{re}[n]$ is the real part of the sequence and $x_{im}[n]$ is the imaginary part of the sequence. (Mitra, 1998)

2.3 Signal processing

There are many signal processing operations that can be used. In time-domain, some of the simplest processing methods are scaling, delay and addition. With scaling we can either increase or decrease the amplitude of the signal. For example, if we want to double the amplitude, we can multiply it with two. Increasing the amplitude is called gain and decreasing the amplitude is called attenuation. Scaling operation can be expressed as

$$y(t) = ax(t)$$

Where $y(t)$ is the output signal, $x(t)$ is the input signal, and a is the scaling coefficient.

Delaying the signal is something required, for example when we want to have a synchronized operation. Delaying operation can be expressed as

$$y(t) = x(t - t_0)$$

Where t_0 is the amount of delay inserted to the output.

Signals can be added or multiplied together. This is often used in radio technology to transmit data. An example of addition operation:

$$y(t) = x_1(t) + x_2(t) - x_3(t)$$

An example of multiplication operation:

$$y(t) = x_1(t)x_2(t)$$

It is also possible to perform integration or differentiation on the signals. Differentiation operation is expressed as

$$w(t) = \frac{dx(t)}{dt}$$

And the most relevant operation for this thesis is the integration operation:

$$y(t) = \int_{-\infty}^{t} x(\tau)\, d\tau$$

(Mitra, 1998)

2.3.1 Digital signal processing

Signals can be processed digitally, and this gives us many benefits. Analog processors are physical components on a circuit, and they require precise values for electrical components to achieve their goal. Components are prone to degradation and their values vary since their production has certain tolerances. Analog processors often require adjusting after production to meet the criteria it was set to meet.

Digital processors on the other hand, can be produced in mass quantities since the only thing they require is software to handle the processing. Digital processors use binary words for coefficients and signals which allows increase in accuracy by increasing wordlength. Digital circuits can be programmed for multiple purposes without needing to change any component. Programmable coefficients, adaptive coefficients or multiple different coefficients are possible with digital signals. Linear phase realization and multirate processing are something that are impossible with analog processors. Digital data can be stored almost forever, and data can be processed even after storing and not in real-time. Digital processors have made low-frequency processing easier, since analog processors require big inductors and capacitors to work in low frequencies.

There are also disadvantages in digital signal processing. The complexity of digital circuits is much greater and digital signals require pre- and/or postprocessing if used with analog signals. Digital signal processing requires sampling frequency to be two times higher than the highest frequency component in the input signal or greater. Otherwise, the frequency

components above half the sampling frequency show as lower frequency components which distort the signal and no longer represent the original signal. The integrated circuits processing the signal need to have power to run, while analog processors function relatively without power. These disadvantages, however, are minor compared to the benefits gained from using digital signal processors.

An example of a digital signal processing system, seen in Figure 2, takes an analog signal as the input, which is then converted into a digital signal with an analog to digital converter. The signal is then processed in the digital signal processor, and it produces a processed digital signal. This digital signal is then converted back to analog signal which is then used somewhere else. Depending on the input signal, processor, and output signal, it is possible to remove converters or change to analog processor. For example, this thesis focuses on digital input signal, digital signal processor and digital output signal, no converters are used in the system.

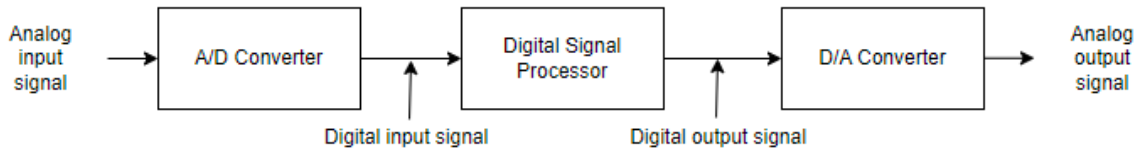


Figure 2: Example block diagram of a digital signal processing system

A/D converter takes samples at periodic intervals and the sample value is held until a new sample is taken. Example can be seen in Figure 3.

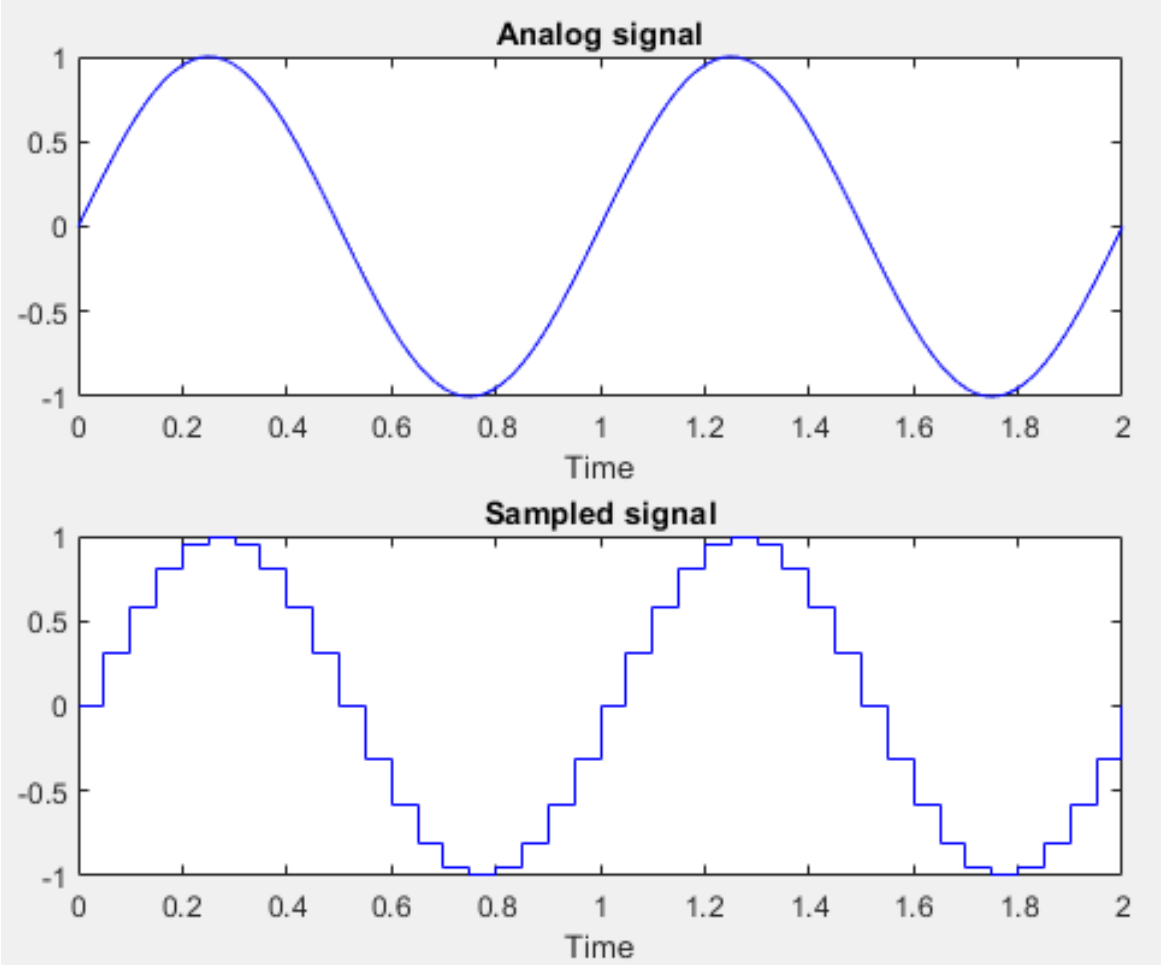


Figure 3: Above: Analog signal sin(2πt), below: above analog signal sampled at periodic intervals. Graphs made with Matlab.

This sampled signal is then converted into binary form and sent into the signal processor as a discrete-time signal seen like the one seen in Figure 1. The accuracy of the digital signal is dependent on the number of bits used for conversion and sampling frequency.

A signal processor like the one in Figure 2 is called a single input, single output system. It takes an input sequence, performs calculations on the sequence according to its programming and produces a different sequence, an output sequence. This operation might be to remove unwanted noise from the input sequence, balance audio signal levels and much more. (Mitra, 1998)

3 Digital filters

In many applications there is a need for removing signals of certain frequencies. This is called filtering. Filters come in many different forms, but in this thesis, the focus is on frequency-selective filters, more specifically low pass filters. Low pass filters let only low frequency components of the input through by attenuating or eliminating the higher frequency components. In audio application this would mean for example letting only bass sounds through and eliminating other sounds. A digital filter is a type of digital signal processor. It takes an input and provides an output, which in low pass filter case is a processor that outputs only low frequencies of the input signal. (Oppenheim, Willsky and Nawab, 1997)

A block diagram of digital processing system can be seen in Figure 2. When dealing with digital systems like the one in this thesis, there is a need to convert analog signal to digital before it can be processed. While the actual sensor does require an analog-to-digital converter to convert capacitance to digital input signal, the system in this thesis did not need an analog-to-digital converter as the input is generated inside the ASIC via register value. Digital-to-analog converter is also not part of the system as the output signal is read as a digital signal to provide user with rotational information which can be read from the ASIC register. (Proakis and Manolakis, 1992)

3.1 Transform-domain representations of digital signals

Digital filters are linear time-invariant (LTI) systems. LTI systems allow us to calculate the output of the system quite easily because of their two main properties.

Linearity means that the response of the system is simply the weighted combination of simple sequences:

$$T[x_1(n)] + T[x_2(n)] = y_1(n) + y_2(n)$$

and

$$aT[x(n)] = ay(n),$$

Where $x(n)$ is the input to the filter and $y(n)$ is the output of the filter.

The linearity property of the system allows us to examine the system with ease when using a simple sequence like the unit sample sequence mentioned earlier.

Time-invariance is a property of a system that can be defined for systems with discretized time, where n is the sample number taken between time intervals. If we have an input $x_1[n]$ and the response $y_1[n]$ to the input sequence, the input sequence can be expressed as

$$x[n] = x_1[n - n_0]$$

And the response to it will always be:

$$y[n] = y_1[n - n_0]$$

Simply put, time-invariance means that the output sequence will always be the same for the same given input sequence.

LTI digital filter is completely characterized by its impulse response in the time-domain. Impulse response $h[k]$ is the response of the system to input $x[n]$ which provides the output $y[n]$

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k]$$

(Mitra, 1998)

Unit sample sequence can be used to obtain a processing system's impulse response. Unit sample sequence is very simple: Only the value at $n = 0$ is non-zero. The sequence allows us to see the response of the system to the simplest input sequence and with this response we can calculate its response to any input sequence.

Unit sample sequence $\delta[n]$:

$$\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

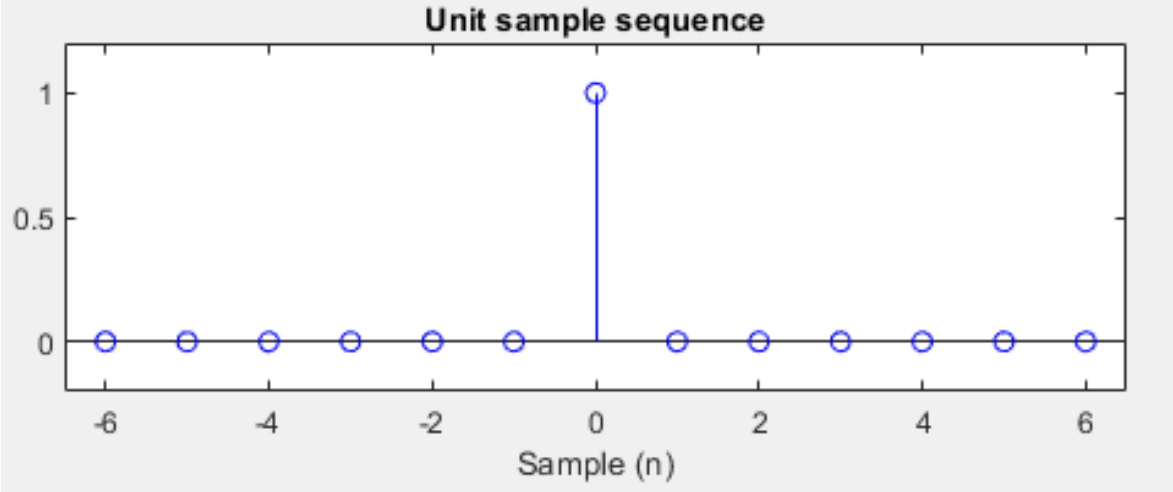


Figure 4: Unit Sample Sequence. Graph made with Matlab.

When $x[n]$ is substituted with $\delta[n]$, $y[n]$ then becomes

$$y[n] = \sum_{k=0}^{n} h[k]x[n-k], \qquad n \geq 0, \qquad x[n] = \delta[n]$$

$$y[n] = h[n]$$

Which means by knowing the output of the system to $\delta[n]$, we know it's impulse response. (Mitra, 1998; Oppenheim and Schafer, 2010)

However, the tested system used in this thesis would not have the expected impulse response by using unit sample sequence. Instead, a very similar function is used, called unit area pulse function or Dirac delta function. A Dirac delta function $\delta[n]$ is like unit sample sequence, but instead of being limited to only one sample, it can be widened to multiple samples, as was necessary in this study due to relatively slow sampling rate of the tested device. In measurements we only have samples $n \geq 0$, so the function becomes

$$\delta[n] = \begin{cases} 1, & 0 \leq n \leq \Delta \\ 0, & n > \Delta \end{cases}$$

(Mitra, 1998)

To have a filter that provides the desired output for all the inputs we have, it must be stable. A system is stable if the output $y[n]$ is bounded for all bounded inputs $x[n]$. This is called bounded-input, bounded-output (BIBO) stability. A system is BIBO stable if and only if the impulse response sequence $\{h[n]\}$ is absolutely summable

$$S = \sum_{n=-\infty}^{\infty} |h[n]| < \infty$$

The filter also needs to be realizable in actual real-life applications. Causality provides this information about the system. To define causality, we have two input sequences $x_1[n]$ and $x_2[n]$, where

$$x_1[n] = x_2[n], \qquad for\ n \leq n_0$$

We then have corresponding outputs $y_1[n]$ and $y_2[n]$ at $n = n_0$ with impulse response $\{h[n]\}$

$$y_1[n_0] = \sum_{k=-\infty}^{\infty} h[k]x_1[n_0 - k]$$

$$= \sum_{k=0}^{\infty} h[k]x_1[n_0 - k] + \sum_{k=-\infty}^{-1} h[k]x_1[n_0 - k]$$

$$y_2[n_0] = \sum_{k=-\infty}^{\infty} h[k]x_2[n_0 - k]$$

$$= \sum_{k=0}^{\infty} h[k]x_2[n_0 - k] + \sum_{k=-\infty}^{-1} h[k]x_2[n_0 - k]$$

If the system is causal, then $y_1[n_0] = y_2[n_0]$. However, it is not known if $x_1[n] = x_2[n]$, when $n > n_0$. This leads to the definition of causality of a LTI discrete-time system, which is only true when

$$h[k] = 0, \qquad for\ k < 0$$

(Mitra, 1998)

So far only time-domain representations of digital signals have been discussed. In this chapter two other domains are introduced. Frequency domain and z domain gives us more flexibility in analyzing and designing filters.

A time domain signal can be converted into frequency domain by performing Fourier transform and back to time domain by inverse Fourier transform.

When dealing with discrete time signals the Fourier transform of a sequence $x[n]$ is

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

This infinite series might not exist and hence we have a condition for its existence. The Fourier transform $X(e^{j\omega})$ exists only if it converges. The series is absolutely convergent if the Fourier transform of $x[n]$ is an absolutely summable sequence

$$\sum_{n=-\infty}^{\infty} |x[n]\,| < \infty$$

And

$$\left|X(e^{j\omega})\right| < \infty, \qquad for\ all\ \omega$$

The Fourier transform $X(e^{j\omega})$ can be divided into real and imaginary parts as follows

$$X(e^{j\omega}) = X_{re}(e^{j\omega}) + jX_{im}(e^{j\omega})$$

However, when dealing with measurements from real life, we do not have an infinite number of samples. For finite sequence $x[n]$ of length N, and $0 \le n \le N-1$, we can get distinct frequency points ω_k. When sampling at certain intervals on the ω-axis, $0 \le \omega \le 2\pi$ at $\omega_k = 2\pi k/N$ points ($k = 0, 1, \dots, N-1$), we get discrete Fourier transform $X[k]$

$$X[k] = X(e^{j\omega_k}), \qquad \omega_k = 2\pi k/N$$

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-\frac{j2\pi nk}{N}}, \qquad k = 0, 1, \dots N-1$$

By replacing $e^{j2\pi/N} = W_N$ we finally get

$$X[k] = \sum_{n=0}^{N-1} x[n]\, {W_N}^{kn}, \qquad k = 0, 1, \dots N-1$$

(Mitra, 1998)

Calculating this equation is very inefficient and fortunately can be made more efficient by algorithms called Fast Fourier Transforms (FFT).

FFT algorithms use following properties of the phase factor W_N to ease computation:

$$Symmetry\ property: \quad {W_N}^{k+N/2} = -{W_N}^{k}$$

$$Periodicity\ property: \quad {W_N}^{k+N} = {W_N}^{k}$$

(Proakis and Manolakis, 1992)

In this thesis FFT is used in Matlab to calculate the Fourier transform of the impulse response to get the filters frequency response.

The z-transform allows us to simplify the Fourier transform, making it much easier to perform calculations. Z-transform domain is used in this thesis in analyzing the designed filter.

Z-transform $G(z)$ of a sequence $g[n]$ can be written as

$$G(z) = Z\{g[n]\} = \sum_{n=-\infty}^{\infty} g[n]\, z^{-n}$$

Where z is a complex variable. If $z = re^{j\omega}$, we can reduce the equation to

$$G\left(e^{j\omega}\right) = \sum_{n=-\infty}^{\infty} g[n]\, r^{-n} e^{-j\omega n}$$

Which also can be seen as the Fourier transform of sequence $\{g[n]r^{-n}\}$. If $r = 1$, meaning $|z| = 1$, the z-transform of $g[n]$ is its Fourier transform. The circle where $|z| = 1$ in the z-plane is called the unit circle. Like the Fourier series, z-transform has a condition of convergence. $G\left(e^{j\omega}\right)$ converges, if

$$\sum_{n=-\infty}^{\infty} |g[n]\, r^{-n}| < \infty$$

For this we can define a region of convergence (ROC) with R values of z:

$$R_{g-} < |z| < R_{g+}, \qquad where\ 0 \le R_{g-} < R_{g+} \le \infty$$

The ROC of the z-transform is crucial information about the system. One sequence cannot be related to one single z-transform without a specified ROC. Like mentioned above, Fourier transform can be obtained from the ROC using the unit circle. Lastly, there is a correlation between z-transform of a causal LTI system and its BIBO stability.

For all LTI discrete time systems mentioned in this thesis the z-transform $G(z)$ can be expressed as ratio of two polynomials $P(z)$ and $D(z)$

$$G(z) = \frac{P(z)}{D(z)} = \frac{p_0 + p_1 z^{-1} + \cdots + p_{M-1} z^{-(M-1)} + p_M z^{-M)}}{d_0 + d_1 z^{-1} + \cdots + d_{N-1} z^{-(N-1)} + d_N z^{-N)}}$$

At the roots of $P(z)$, $G(z)$ will be 0. All the values of z, where $G(z) = 0$, are called the zeros of $G(z)$. At roots of $D(z)$, $G(z)$ will be ∞. All the values of z, where $G(z) = \infty$, are called the poles of $G(z)$. There are also $(N - M)$ zeros at $z = 0$, if $N > M$ or $(N - M)$ poles at $z = 0$, if $N < M$. (Mitra, 1998)

3.2 Filter characteristics

In this section the main parts of filter characteristics are discussed: impulse response, amplitude response, phase response and group delay. A low pass filter is used as an example.

3.2.1 Frequency response

All LTI systems can be defined by their impulse response, sequence $h(n)$. For all n it is possible to calculate output sequence $y(n)$ from the input sequence $x(n)$ and system impulse response $h(n)$.

$$y(n) = x(n) * h(n)$$

This can also be expressed as a difference equation seen below.

$$\sum_{k=0}^{N} a_k y(n-k) = \sum_{m=0}^{M} b_m x(n-m)$$

It is then rewritten as a recurrence formula:

$$y(n) = -\sum_{k=1}^{N} \frac{a_k}{a_0} y(n-k) + \sum_{k=0}^{M} \frac{b_k}{a_0} x(n-k)$$

Z-transform is performed on the equation to get the system function.

$$H(z) = \frac{\sum_{m=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}$$

The system function is the system's impulse response. In this thesis the impulse response to a Dirac delta function was measured and it is compared with the simulated result by using the design values of a_k and b_k.

The system is designed with coefficients a_k and b_k. Changing them modifies the system's behavior. Low pass filter is examined where the coefficients should provide attenuation for high frequency components of the input signal.

The frequency response can be calculated from the system impulse response by performing Fourier transform.

$$H(e^{j\omega}) = \frac{1}{2\pi} \int_{-\pi}^{\pi} h(n) e^{-j\omega n} \, d\omega,$$

Where $H(e^{j\omega})$ is the frequency response of the filter and $h(n)$ is the impulse response.

Frequency response $H(e^{j\omega})$ can be divided into two parts, real (H_R) and imaginary (H_i):

$$H(e^{j\omega}) = H_R(e^{j\omega}) + jH_i(e^{j\omega})$$

This can also be expressed as amplitude and phase response:

$$H(e^{j\omega}) = |H(e^{j\omega})| e^{j\angle H(e^{j\omega})}$$

where $|H(e^{j\omega})|$ is the amplitude response and $\angle H(e^{j\omega})$ is the phase response. (Oppenheim and Schafer, 2010)

3.2.2 Amplitude response

A low pass filter modifies the incoming signal so that signals with frequencies below the specified frequency point are let through and all signals with frequencies above the cut-off frequency ω_c are blocked. The amplitude response of an ideal low pass filter is:

$$|H_{lp}(e^{j\omega})| = \begin{cases} 1, & |\omega| < \omega_c, \\ 0, & \omega_c < \omega \leq \pi \end{cases}$$

Ideally this would look like the graph below.

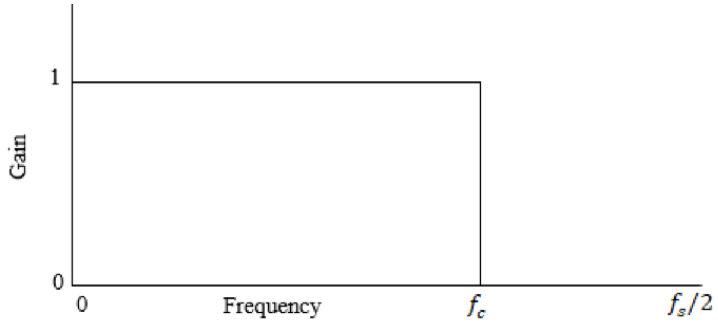


Figure 5: Amplitude response (gain versus frequency) specification for the ideal low pass filter.

Where gain is the amplification of incoming signal and f_c is the cut-off frequency for the filter and $f_s/2$ is the maximum frequency for the system, called the Nyquist frequency. The area before cut-off frequency is called the passband area of the filter and the area after it is stopband. Gain of one, up to cut-off frequency and zero after is not possible when looking at real world applications.

Real world performance can be seen in the figure below. The passband has fluctuation instead of having a constant gain as seen in the graph as δ_{p1} and δ_{p2}. Cut-off frequency is marked by w_p in this graph and there is no immediate stop of gain, but rather a sharp slope. The area between passband w_p and stopband w_s is called transition area and is marked by ΔT in the graph. In the stopband we also see some ripple of magnitude δ_s before settling to lower level of gain, but not zero. Maximum frequency of the system is π in this case.

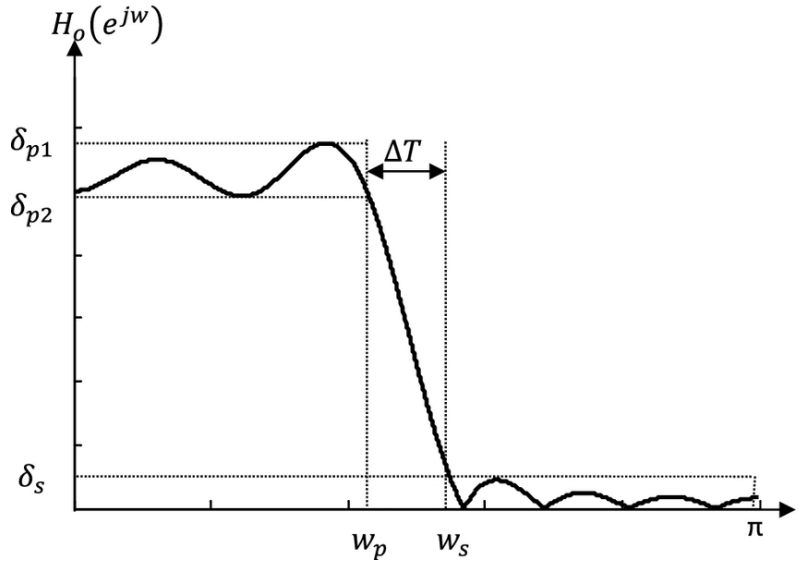


Figure 6: Frequency response of low pass filter
https://www.researchgate.net/publication/338061165/figure/fig1/AS:941480411688961@1601477830064/Frequency-response-of-low-pass-filter-specifications.png 6.4.2023

Passband, cut-off frequency, transition area and stopband can all be modified by changing the filter coefficients and the order of the filter. When designing a filter, understanding of what is needed of the filter should be present and specification should be set to implement the filter for the system.

Usually, the amplitude response is wanted in decibels, which is obtained from the gain function $G(\omega)$

$$G(\omega) = 20log_{10}\left|H^{e^{j\omega}}\right| dB$$

(Mitra, 1998)

3.2.3 Phase response

Phase response is the phase delay of the system over frequency range. Like mentioned in previously, phase response can be obtained from the frequency response:

$$\angle H\left(e^{j\omega}\right)$$

In an ideal filter phase delay is linear within its passband:

$\angle H\left(e^{j\omega}\right) = \Theta(\omega) = -\omega n_0$, where n_0 is a constant and $\omega < \omega_c$.

The output is delayed linearly, but this is the ideal case and not realizable in real life. (Proakis and Manolakis, 1992)

3.2.4 Group delay

Group delay is derivative of the phase response:

$$\tau(\omega) = -\frac{d}{d\omega}\Theta(\omega)$$

where $\Theta(\omega)$ is the unwrapped phase response of the system. If $\Theta(\omega)$ is in radians, the unit for group delay is seconds.

Which gives us the delay in the system in time instead of phase. Like amplitude and phase response, group delay is presented over frequency range, meaning that some frequency components of the input signal are seen later in the system output than others. (Mitra, 1998)

3.3 Filter types

There are two main types of digital filters: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR). As the name suggests, FIR filters have finite impulse response which means the response settles to zero in finite time. They are used in applications where phase must be linear within the passband of the filter. IIR filters have infinite impulse response. IIR filters are used when phase distortion is not an issue. They require fewer parameters, use less memory, and thus require less computation. (Proakis and Manolakis, 1992)

In this thesis IIR filters were tested. The filters with lower cutoff frequency are Butterworth-type filters and higher cutoff frequency filters are Bessel-type filters. Butterworth filters can be identified from their amplitude response curve, which is quite flat in the passband, then in transition area it turns with order of the filter defining the sharpness and in stopband is the response is relatively straight line with the angle defined by the order.

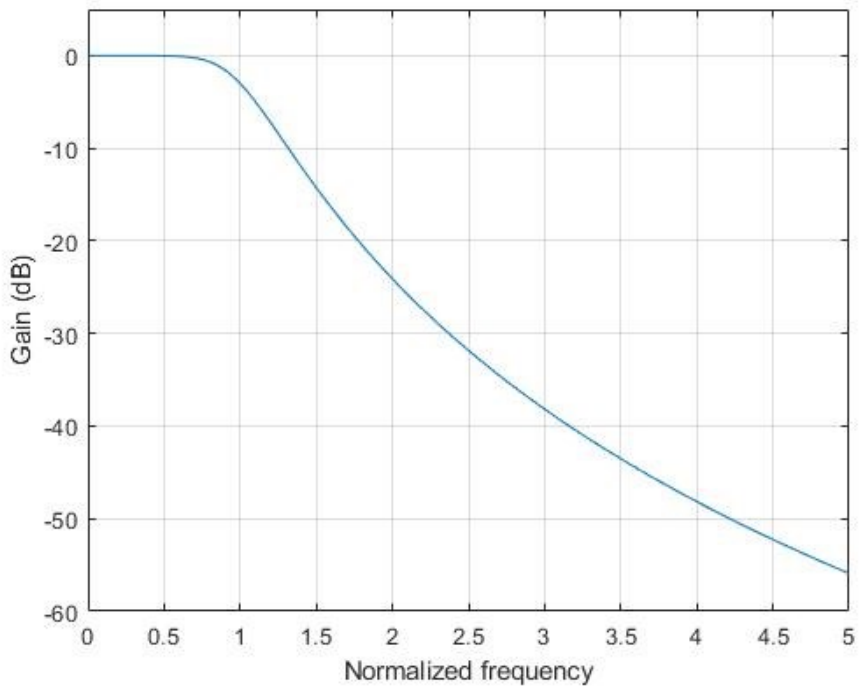


Figure 7: 3rd order Butterworth lowpass filter. Graph made with Matlab.

Bessel filters have a linear phase response in their passband, but a higher transition area.

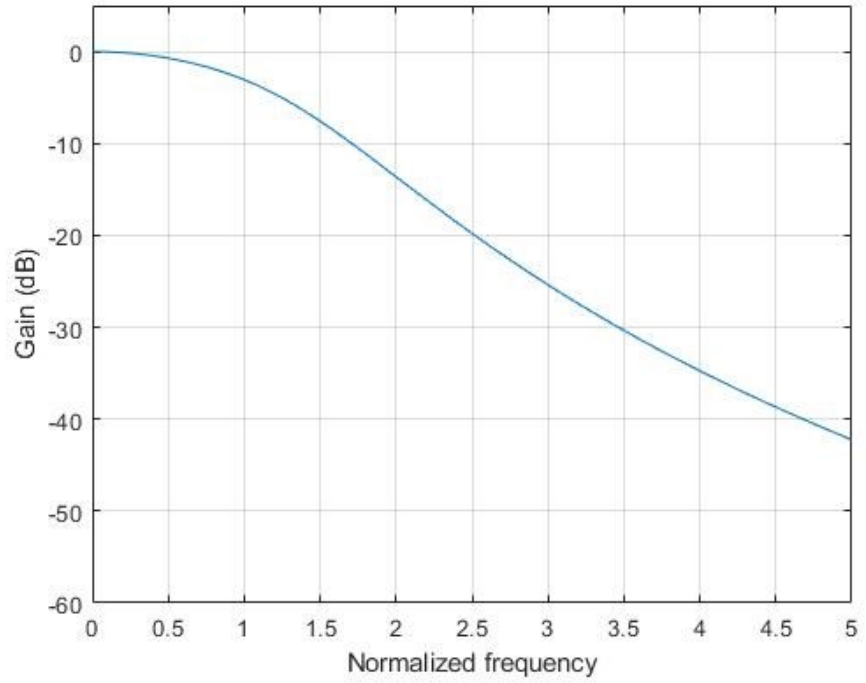


Figure 8: 3rd order Bessel lowpass filter. Graph made with Matlab.

Bessel filters allow us to introduce notches to the amplitude response as well.

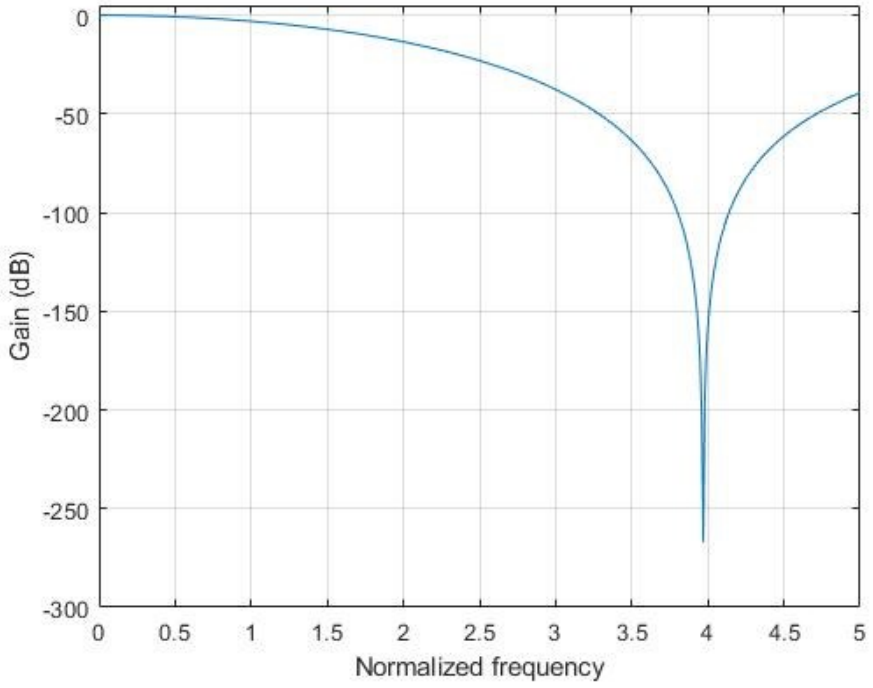


Figure 9: 3rd order Bessel lowpass filter with a notch. Graph made with Matlab.

(Mitra, 1998; Proakis and Manolakis, 1992)

3.4 Examining the filter with the design values

The ASIC and the filter tested in this thesis were designed with certain values. When it is made into the actual DUT ASIC, many changes may occur. The goal of this thesis is to characterize the post-processing filter of the gyro output, which reveals the differences in the design of the ASIC and the manufactured DUT ASIC.

The filter has multiple channels and coefficients for multiple different frequency responses. Filter setting 0 is taken into closer observation, where the designed filter is examined. The filter coefficients and design specifications can be found in Table 1 and Table 2. With the coefficients for filter setting 0 we get the following system function

$$H(z) = \frac{1.02506140131631 * 10^{-7} + 4.10024560526524 * 10^{-7}z^{-1} + 6.15036840789787 * 10^{-7}z^{-2} + 4.10024560526525 * 10^{-7}z^{-3} + 1.0250614013163 * 10^{-7}z^{-4}}{1 - 3.90538595579747z^{-1} + 5.72060809383738z^{-2} - 3.72493668798350z^{-3} + 0.9097161900418 21z^{-4}}$$

We can use the filter coefficients plot the pole-zero map and frequency response of the filter using Matlab's Filter Visualization Tool (command fvtool()).

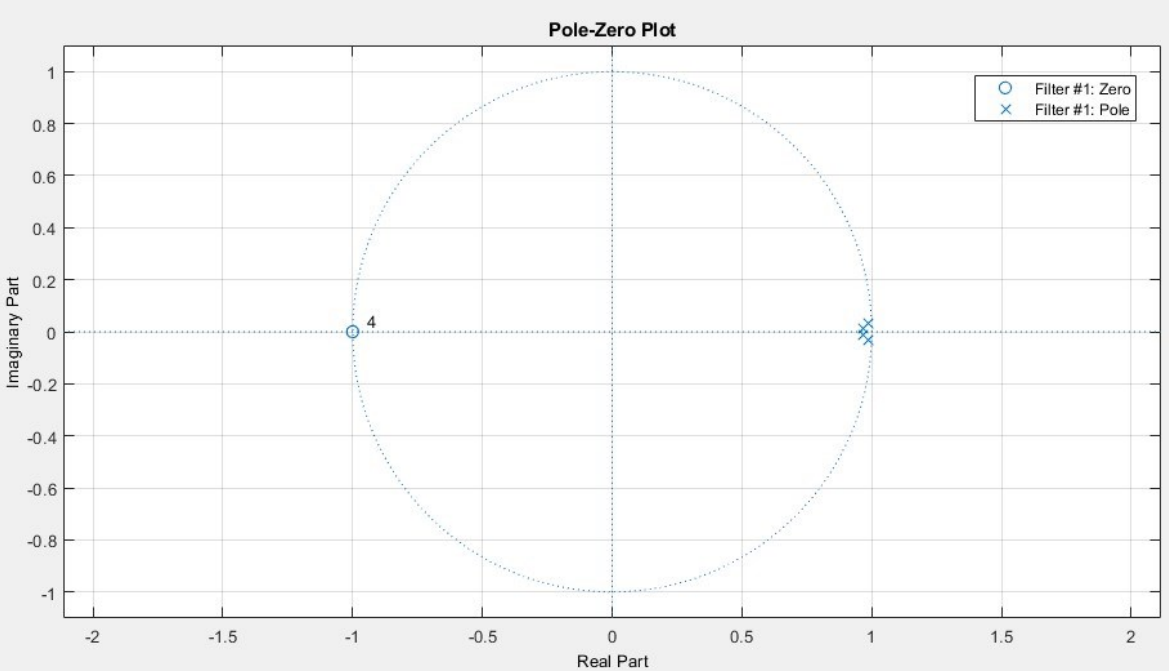


Figure 10: Pole-zero map of designed filter with coefficients of filter setting 0. Graph made with Matlab.

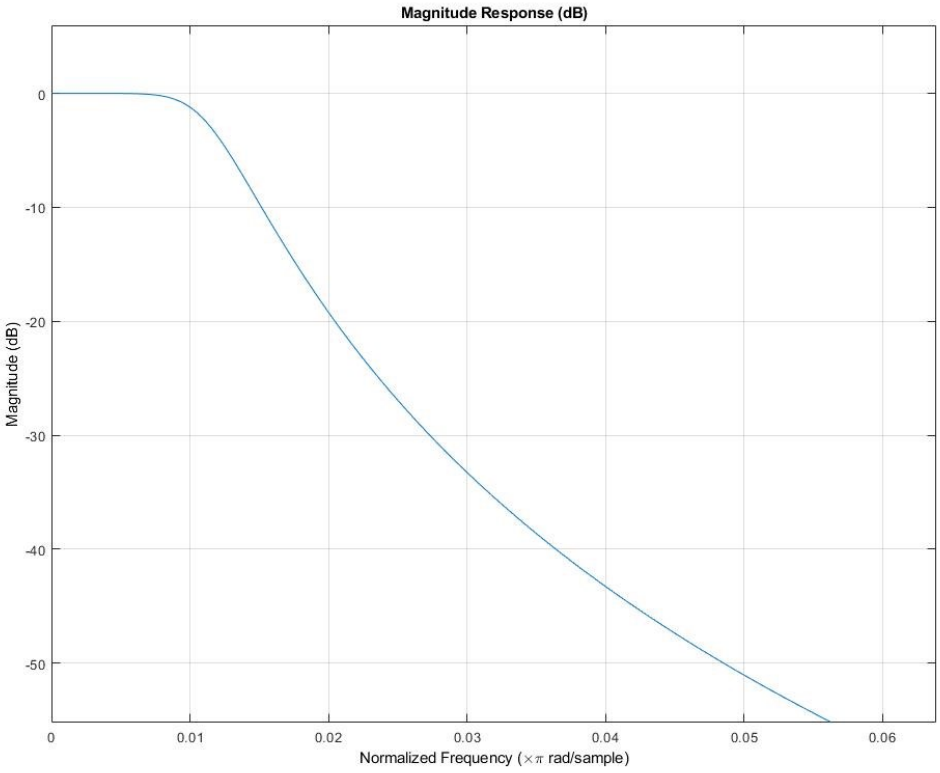


Figure 11: Magnitude response of designed filter with coefficients of filter setting 0. Graph made with Matlab.

We can see from the plots that the selected filter is a low pass filter, filter type is Butterworth like the specification in Table 2 mentioned.

4 Test setup

In this chapter the used test setup is introduced. The testing was conducted with in-house built electronics and the DUT was a packaged ASIC. Test parameters and commands were defined in Labview software.

4.1 DUT

The device under test in this thesis was an ASIC designed for 6DOF sensor with three different channels of sensing rotational speed in all three axes and three different channels for sensing acceleration in all three axes. Sensing elements were not needed in this test as this was an ASIC only test with digital input and output, so only a packaged ASIC was used.

4.2 Test electronics and setup

The test electronics used in this experiment was an in-house built electronic measurement device called “My” electronics. My utilizes FPGA and commands are sent to it via PC. On PC Labview software is used. Labview software sends commands to the FPGA which translates the code that embedded system can understand and executes them. Communication between embedded system and DUT is handled with SPI, My then translates the messages from the DUT and sends them to the Labview software.

4.2.1 SafeSPI

Connection to the DUT was established with SafeSPI. SPI was developed by Motorola in the 1980s and is now a de facto standard for communication interface in many applications of the car industry. Since no standard exists for how SPI should be used, automotive companies have tried to unify all SPI interfaces used in the industry by creating the SafeSPI standard. Many sensors used in cars are safety related and failure of these devices is potentially life-threatening, a monitor device is added to the SPI circuit to make sure the master is getting the same readings from all the slaves (sensors). Since there are multiple

slaves in the SPI bus, every device on the bus needs to be synchronous and having different SPI protocols used in this bus would make it hard.

48-bit SPI frame was used in the measurements. The SPI frame consists of multiple different bits, the most important ones being TA, DATA and C. Target Address (TA)/ Source Address (SA), which determines the register address in the ASIC that is wanted to be written or read. DATA is the actual data that is read or to be written. Cyclic Redundancy Check (C) checks for data integrity. Other bits control frame length (FrTyp), capture mode (CAP) used for freezing sensor data, so all registers values have same time stamp, and a read or write bit for informing the slave which operation is wanted.

REQ_135 The following table shows the command frame format for the out-of-frame protocol

Bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28
MOSI	TA9:0										RW	CAP	FrTyp	*						
value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Bit	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8
MOSI	DATAI19:0																			
value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Bit	7	6	5	4	3	2	1	0
MOSI	C7:0							
value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Figure 12: 48-bit SPI frame. SafeSPI Organization (2021, p. 27)

The system used in the tests requires the user to only input the address and data to communicate with the ASIC via SPI. All other bits are calculated and set automatically by Labview software.

4.2.2 Timing

When measuring frequency response of a system, timing is critical. The timings were measured using an oscilloscope. Duration of SPI words and delay between SPI words were measured to gain understanding of the duration of SPI commands before a new one is started.

4.3 Test software

This test was conducted using National Instruments Labview software. Labview is a graphical programming language which is often used in automated research, validation, and production test systems. Labview developers create VIs to execute certain tasks. At Murata Finland there is an in-house built sequencer where multiple VIs can be called using sequence files, which provides flexibility in using the software. (National Instruments, 2022)

4.3.1 Labview

Labview sequences were used to conduct the testing. Labview saved all the measurement results in a TDMS file. The main VI used in this characterization was SPISequencer.vi, which allows the user to store multiple SPI commands in advance for the test electronics to execute. It is used when timing is critical, since using VIs separately the commands are sent one by one with no control over timing. A text file is given to SPISequencer.vi, commands are sent to My, My executes the commands and then sends the results back to Labview.

Delay between frames can also be defined in SPISequencer.vi. This was used to have less data with the slower filter settings and get proper results with the faster filter settings. The text file contains lines which all have defined read or write command, the address used, and data to be sent.

5 Test conditions and results

The goal of this thesis was to measure the frequency response of digital filters. The tests were conducted on low pass IIR filter, but the tests are identical no matter what filter type is used.

The DUT ASIC has eight different filter settings for gyro output post processing. Filter settings from 0 to 5 are defined by their -3dB point: 68Hz, 30Hz, 13Hz, 280Hz, 370Hz, 235Hz. Filter setting 6 is a programmable filter and filter setting 7 is a bypass filter.

All gyro channels use the same filter, and all these channels were measured to gain understanding of the filters and verify that all channels function in the same way.

Tests were conducted in ASIC test mode, which is not accessible for customers.

Calculations and plotting were made in Matlab.

5.1 Conditions

Frequency response was tested by impulse method instead of testing different frequency points. This was due to simulations having been done by impulse method and functionality needed to be the same for this test.

Frequency response was tested by sending an impulse to the ASIC via ASIC registers. The initial value of the input register was first set to 0 then a delay was introduced to make sure the output is zero before starting the impulse. After the delay an impulse was sent and then a delay was needed for the ASIC to update values. When ASIC updated the value, input was set to 0 and measurements were started.

Before starting the test, a command was sent to change filter setting and then measurements could be started.

5.1.1 Test method

A series of SPI commands were sent to My for execution. These SPI commands were read and write commands to the registers related to gyro post processing: Filter output registers:

G*_PP_STAT_FILT_DATA_LSB and G*_PP_STAT_FILT_DATA_MSB, 24 bits in total, filter control input registers: G_CAL_CH*_LSB and G_CAL_CH*_MSB, 24 bits in total. The star * indicates the gyro channel used (1-3).

First G_CAL_CH*_LSB and G_CAL_CH*_MSB were set to zero. After these commands, dummy read commands were executed to introduce delay. Each read or write command takes 4.855 ns, and another 1 ns delay was added to SPISequencer.vi to prevent any signal distortion, making total delay of 5.855 ns for shortest delay.

When the registers G*_PP_STAT_FILT_DATA_LSB and G*_PP_STAT_FILT_DATA_MSB were known to be zero, an impulse is created. Impulse was generated by changing the values of registers G_CAL_CH*_LSB and G_CAL_CH*_MSB to their maximum, G_CAL_CH*_MSB 20 bits (1111 1111 1111 1111 1111 = 1 048 575) and G_CAL_CH*_LSB 4 bits (1111 = 15).

After this a delay was needed for the ASIC to update the register values. ASIC sampling frequency is ASIC operating frequency F_{nom} divided by two. F_{nom} target is 24 kHz, so with sampling frequency F_s = 12 kHz the shortest delay should be

$$\frac{\frac{1}{12000}}{5.855\ ns} = 14.233 \approx 14$$

SPI commands to get the desired delay. However, there are many factors that affect this delay and simply calculating the needed number of commands was not sufficient but was instead found out through trial and error.

When the delay was sufficient G_CAL_CH* registers were set to zero, ending the impulse and immediately G*_PP_STAT registers were started to be read. SPI command file was around 5000 commands long for each filter setting, where most of the commands were read commands for reading G*_PP_STAT registers after the impulse to get enough samples for frequency response.

For bypass filter the test was different. To get information about the received impulse, another register pair was measured: G_CAL_CH*_DATA_MSB and G_CAL_CH*_DATA_LSB. These registers measure what the filter control input register values are and in this test the filter output should match these two new registers. To see the

impulse, both register pairs read multiple samples before the impulse, then impulse was set and both registers were read again multiple samples to see if the register values match.

5.2 Matlab

The data was processed with Matlab to get the results. Matlab is a computing platform that is for example used for data analysis and simulation. In this case it was used to perform calculations, generate ideal response, and plot data. (Mathworks, 2022a)

Data was read from TDMS to Matlab using xlsread() command. The file contained measured data for all filters. Data is read for one filter setting at a time. After data is read, MSB and LSB register values are combined to get the full 24-bit data.

Since the MSB and LSB were read with a higher sampling frequency than the ASIC sampling frequency, there are multiple readings with the same value before the register value changes in the ASIC. Next step is to remove these extra values from the data vector by taking the first occurrence of each value and removing the rest. This is done by using the diff() function to see the value change and remove all points where diff() result is 0.

5.2.1 Amplitude response

Amplitude response is calculated from the impulse response by performing Fourier transform.

$$H\left(e^{j\omega}\right) = \frac{1}{2\pi}\int_{-\pi}^{\pi} h(n)e^{-j\omega n}\ d\omega$$

and then taking the absolute value of the frequency response.

$$H\left(e^{j\omega}\right) = |H\left(e^{j\omega}\right)|e^{j\angle H\left(e^{j\omega}\right)}$$

where $|H\left(e^{j\omega}\right)|$ is the amplitude response of the system.

fft() function of Matlab was used to perform the Fourier transform. Matlab's documentation page for fft() function was used as a reference. (Mathworks, 2022b)

First the length of the filter data vector is stored in variable L and frequency vector f is generated with it making a vector from 0 to L, dividing it with L and then multiplying it with ASIC sampling frequency F_s:

$$f = F_s \cdot \frac{0.5a}{L}, where\ a = 0, 1, 2, \dots, L$$

First, the frequency response vector Y is calculated from the filter data.

The amplitude response $|H(e^{j\omega})|$ is generated from the frequency response vector Y by taking its absolute value. Since this is a two-sided spectrum and only positive side of the spectrum is needed, it is converted into one-sided spectrum by dividing Y with vector L, halving the vector length, and multiplying the result vector P1 by two. To match the vector P1 with the frequency vector f, f must also be halved.

Once we have the amplitude response vector, we convert it to decibels and subtract the maximum value of the vector to start the amplitude response at zero decibels.

Matlab code for amplitude response of channel one can be found below.

```
filt_data1 = d_FILT0_b1; % filter data for channel 1
L1 = length(filt_data1); % length of signal
f1 = Fs*(0:(L1/2))/L1; % generate the frequency vector
Y1 = fft(filt_data1); % perform Fourier transform on the filter data
P21 = abs(Y1/L1); % generate the amplitude response vector
P11 = P21(1:L1/2+1); % change amplitude response vector from two sided to
P11(2:end-1) = 2*P11(2:end-1); % one sided
P11 = 20*log10(P11); % convert results into decibels
P11 = P11-max(P11); % change offset level to 0 dB
```

5.2.2 Phase response

Phase response is the imaginary component of the frequency response.

$$H(e^{j\omega}) = |H(e^{j\omega})|e^{j\angle H(e^{j\omega})}$$

where $e^{j\angle H(e^{j\omega})}$ is the phase response of the system.

Phase response was calculated with the angle() function of Matlab. Phase angle is taken from the frequency response vector Y calculated previously. To properly examine the phase response, it is unwrapped by the function unwrap() so the phase will continue to decrease

instead of reverting back to 0° after -359°. After the phase response vector has been created, a frequency vector is created with the same length.

Matlab code for phase response of channel one can be found below.

```
phase1=unwrap(angle(Y1)); % generating the phase response from frequency
response vector
fp1 = (0:length(phase1)-1)/length(phase1)*Fs; % frequency vector for phase
response
```

5.2.3 Group delay

Group delay is calculated from phase response by derivation.

$$\tau(\omega) = -\frac{d}{d\omega}\angle H\left(e^{j\omega}\right)$$

$$\tau(\omega) = -\frac{\Delta\varphi}{2\pi \cdot \Delta f} = -\frac{(\varphi_{i+1} - \varphi_i)}{2\pi \cdot (f_{i+1} - f_i)}$$

Since group delay is calculated between measured points, frequency vector must be adjusted accordingly. This is done by taking the average of the current and next frequency point.

$$f_{GD} = \frac{f_{i+1} + f_i}{2}$$

Since we're using F_s for the frequency vector we have frequency points at equal intervals, the new created frequency vector f_{GD} is the same vector as previous frequency vector with $\frac{\Delta f}{2}$ offset.

Matlab code for group delay of channel one can be found below.

```
gd1 = -diff(phase1)./(2*pi*diff(fp1)); % generating the group delay vector from
phase response vector
for i = 1:(length(fp1)-1)
     f_gd1(i)=(fp1(i+1)+fp1(i))/2; % frequency vector adjusted by taking
average of current and the next frequency point

end
```

5.2.4 Frequency response with design values

Frequency responses for designed values were created using MATLAB's functions. All filters' coefficients are listed in Table 1. These coefficients were used to plot the designed

responses for each filter setting. freqz() was used to get amplitude response and phasez() was used to get phase response. Both functions calculate their respective response using a and b coefficients, sample number and sampling frequency. To get all filter setting designed responses, coefficients for each setting were input to the function, sample number was chosen to be 4000 to have enough data points but still a number not too heavy for calculations. ASIC sampling frequency is measured during calibration and can be verified during the measurements. Matlab code is found below.

```
[h,f_sim] = freqz(b,a,4000,Fs); % Designed results for amplitude response
[phi,f_sim] = phasez(b,a,4000,Fs); % Designed results for phase response
```

Designed group delay was calculated the same way as with measurement data. Phase response data vector is used for calculation and new frequency vector is created.

```
gd_sim = -diff(phi)./(2*pi*diff(f_sim)); % generating the group delay vector
from phase response vector

for i = 1:(length(f_sim)-1)
      fgd_sim(i) = (f_sim(i+1)+f_sim(i))/2; % frequency vector adjusted by
taking average of current and the next frequency point
 end
```

5.3 Results

The results are shown in this chapter. Data for all three channels are channels are seen in all graphs, except for filter setting 7, where all channels are plotted in their own graphs for better visualization. Graphs for all filter settings are presented and specification limits are also drawn in the graph if defined. All graphs are made with Matlab.

Lower bandwidth filters were relatively easy to measure, and results were repeatable. Filter setting 2 (13Hz filter) measurement ends before the output has settled completely, which might affect response results. With higher bandwidth filter settings (filters 3, 4 and 5) the measurement was not as repeatable. The test setup timings could not be aligned with the ASIC sampling times, which in many cases caused the impulse not to be seen by the ASIC or the measurements started only after the impulse had already started. It was also discovered that the ASIC doesn't update all its registers at the same time. G_CAL_CH*_DATA is updated before G*_PP_STAT_FILT_DATA, which resulted in only a small time window

to have impulse registered by the input register and the impulse seen at the output from the beginning of the impulse and not during it.

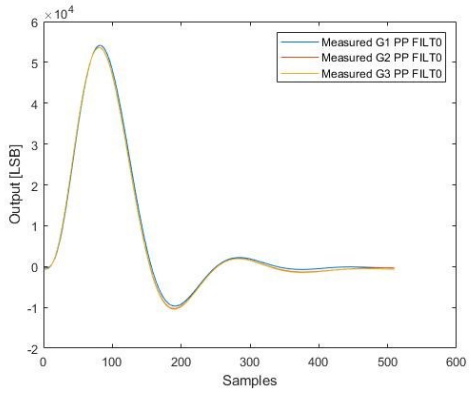
ASIC operating frequency F_{nom} was measured during DUT ASIC calibration and its value is $23939.207\ Hz$. Sampling frequency used in all calculations was calculated from this value:

$$F_s = \frac{F_{nom}}{2} = 11\,969.6035\ Hz$$

, both with measured results and results with designed values.

5.3.1 Impulse response

This chapter shows the combined register data from G*_PP_STAT_FILT_DATA after data removal. Length of the data vector is shortened; new vector length depends on the bandwidth of the filter setting. Amplitude remains the same. A small overshot after impulse can be seen with all filter settings.



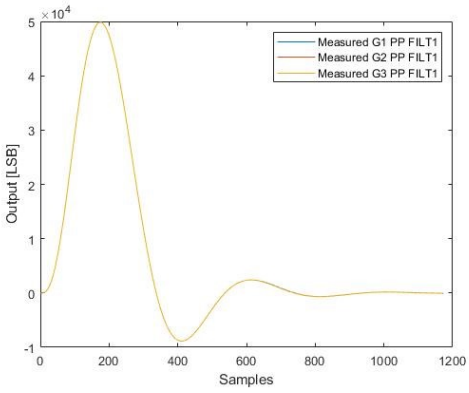


Figure 13: Impulse response for filter setting 0 after data removal (left)

Figure 14: Impulse response for filter setting 1 after data removal (right)

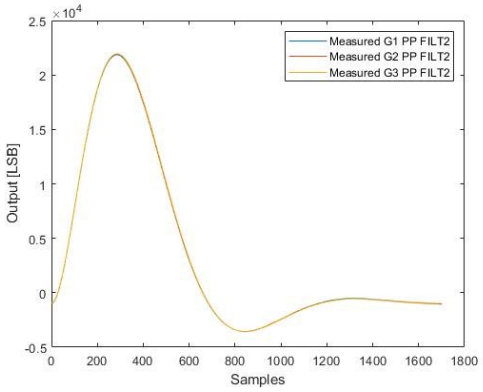


Figure 15: Impulse response for filter setting 2 after data removal (left)

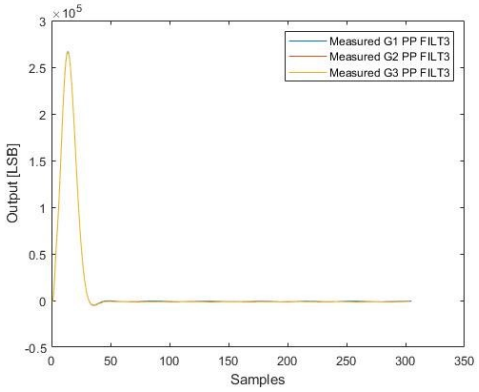


Figure 16: Impulse response for filter setting 3 after data removal (right)

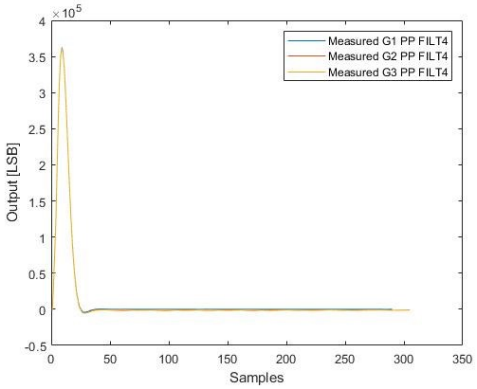


Figure 17: Impulse response for filter setting 4 after data removal (left)

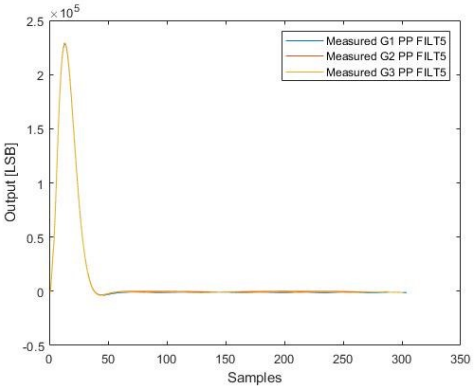


Figure 18: Impulse response for filter setting 5 after data removal (right)

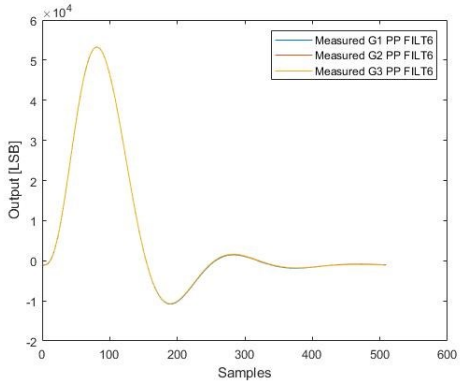


Figure 19: Impulse response for filter setting 6 after data removal (filter setting 0 coefficients were used)

5.3.2 Amplitude response

Amplitude response generated from measured data using Matlab. Amplitude response for all three channels is drawn along with designed response using the coefficients used to build the filter. All filters were within specification. With filters 0, 1 and 2 typical Butterworth

type graph can be seen and filters 3, 4 and 4 can be identified as Bessel type filters. With filter setting 0 channel 2 attenuation is less than channels 1 and 3 in the higher frequencies and results were repeatable. The result is still within specification, but this behavior was not seen in filter setting 6 measurements, which should give the same results.

Butterworth type filters start to drift from the designed response in higher frequencies, but Bessel type filter follow the designed curve closely.

The first channel is drawn with blue color, second channel with red color and third channel with green color. The designed response is plotted with green line with circles and all specification is plotted with dashed red line.

Specification for each filter setting can be found in Table 2. There are six columns related to amplitude response: Columns f3dB (Hz), minimum, typical, and maximum, passband (Hz), passband tolerance (dB) and stopband attenuation.

For the f3dB columns, minimum is the minimum frequency where -3dB is allowed to be, maximum tells the maximum frequency and typical is the designed value.

The passband specification is up to certain frequency in the filter's passband, only specified for filter settings 0-2. The filter output must stay within passband tolerance in the defined passband area.

The specification for stopband attenuation does not include full frequency range but is instead checked for each filter setting at only a defined frequency. The attenuation must be greater than the specified value in dB at the given frequency. Filter settings 2, 4 and 5 have two frequency points where this is checked.

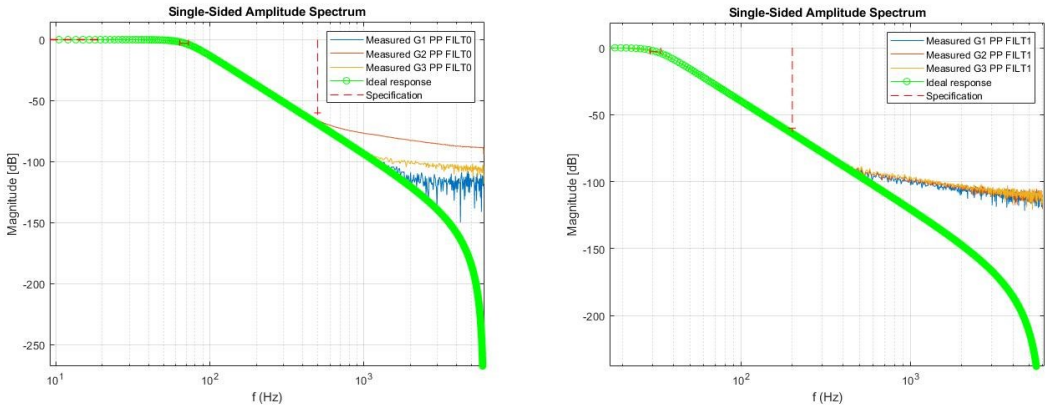
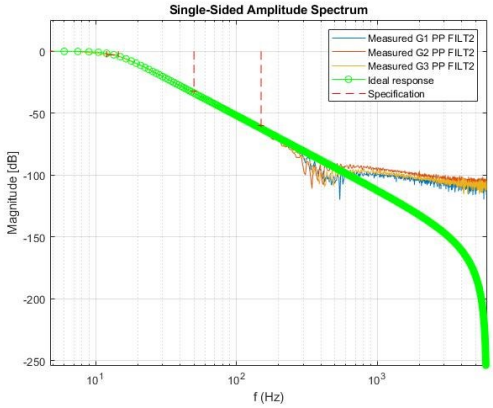


Figure 20: Amplitude response of filter setting 0 (left)

Figure 21: Amplitude response of filter setting 1 (right)



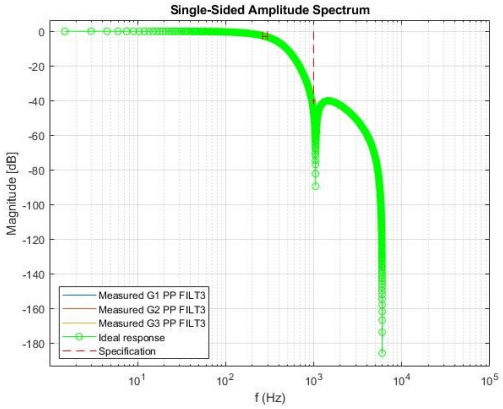
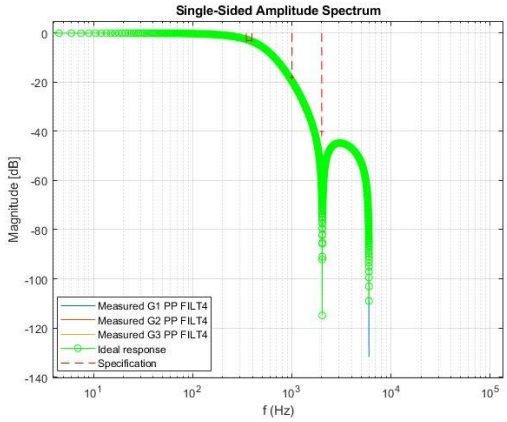


Figure 22: Amplitude response of filter setting 2 (left)

Figure 23: Amplitude response of filter setting 3 (right)



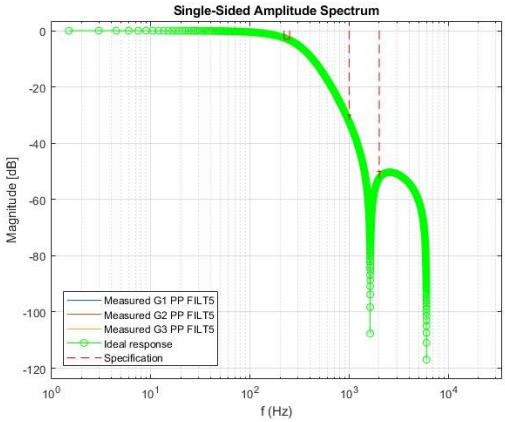


Figure 24: Amplitude response of filter setting 4 (left)

Figure 25: Amplitude response of filter setting 5 (right)

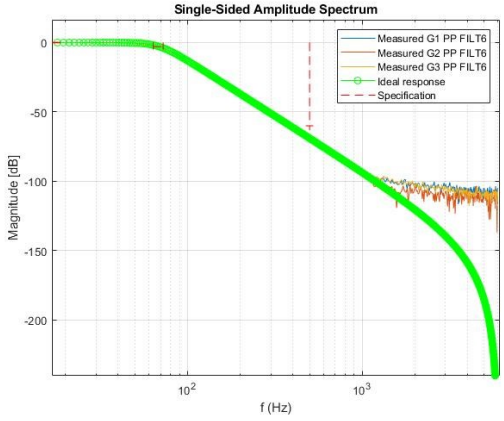


Figure 26: Amplitude response of filter setting 6 (filter setting 0 coefficients were used)

All the filter settings matched their specification for -3dB point as can been seen in the next graphs.

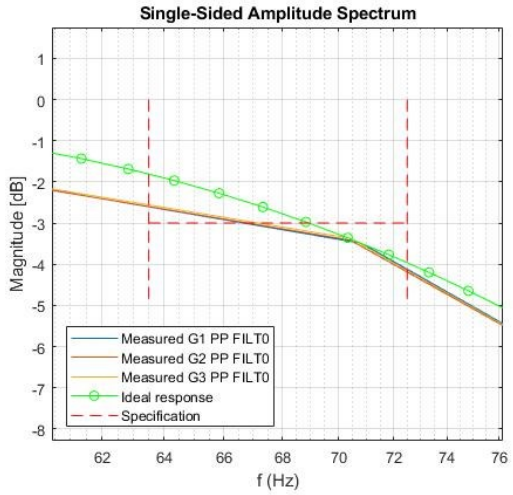


Figure 27: -3dB point for filter setting 0 (left)

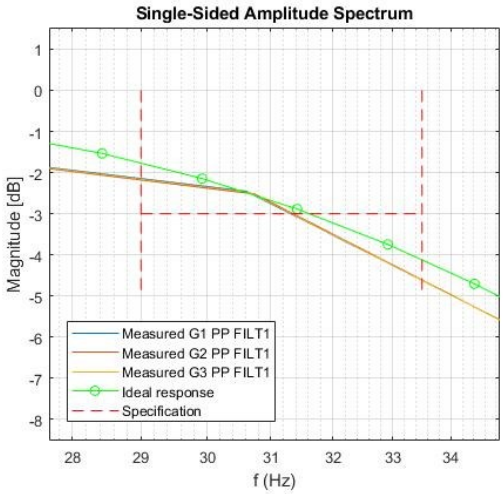


Figure 28: -3dB point for filter setting 1 (right)

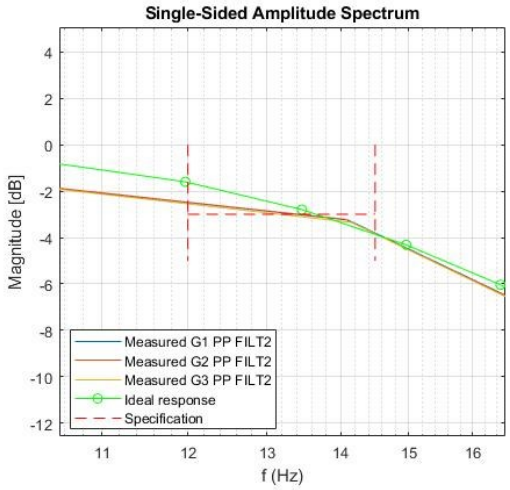


Figure 29: -3dB point for filter setting 2 (left)

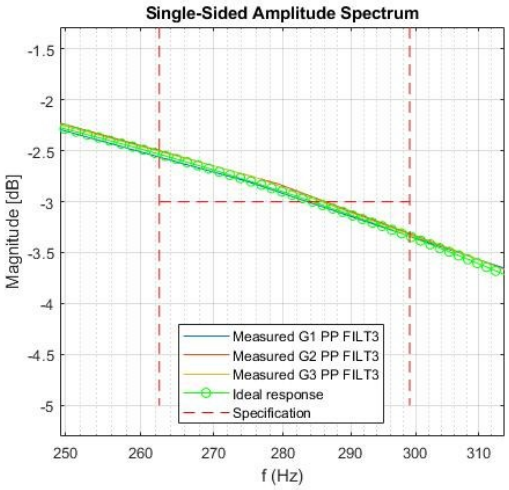


Figure 30: -3dB point for filter setting 3 (right)

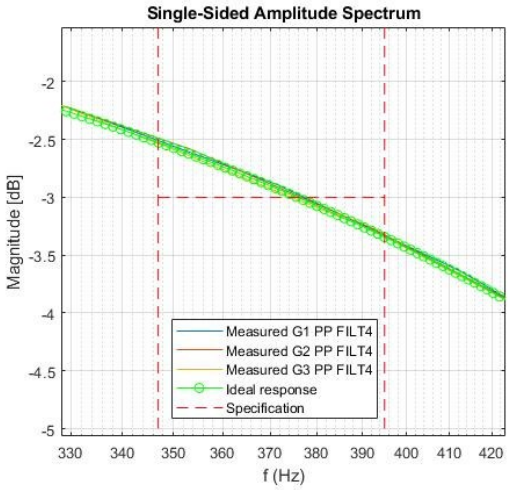


Figure 31: -3dB point for filter setting 4 (left)

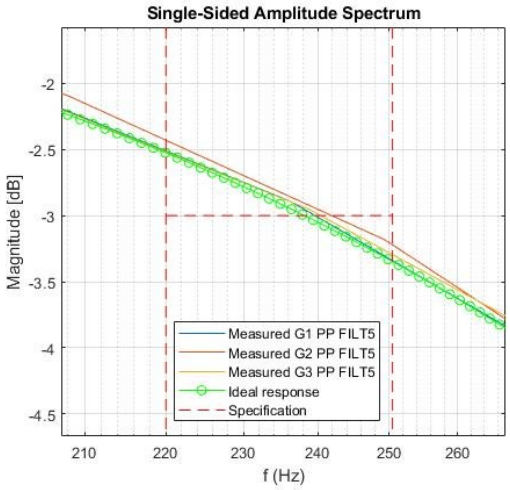


Figure 32: -3dB point for filter setting 5 (right)

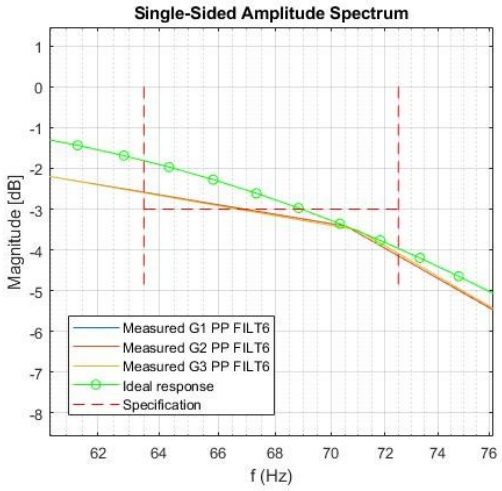


Figure 33: -3dB point for filter setting 6 (filter setting 0 coefficients)

5.3.3 Phase response

Phase response was also generated using Matlab. All the filters follow the designed curve at first, but all of them start to drift away from it after a certain frequency. Lower bandwidth setting filters response becomes unstable after 100 Hz with setting 2 and after 200 Hz with setting 1. Higher bandwidth filter responses are more stable and with the highest bandwidth settings the response is near the designed response even in higher frequencies.

Table 2 also includes the specification for phase response. For phase response there's only one column containing relevant information, labelled phase response. Only filter setting 2

has specification for phase response and that is the -45° point which with setting 2 should be at 5 Hz.

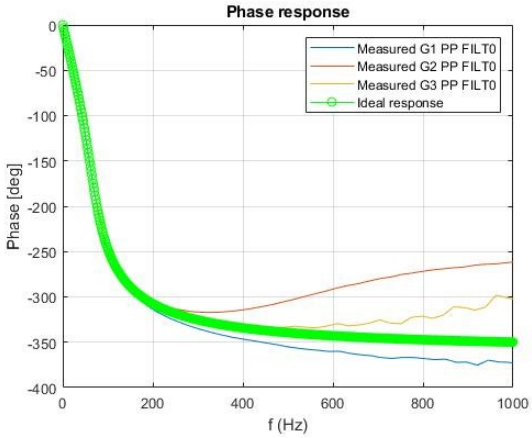


Figure 34: Phase response for filter setting 0 (left)

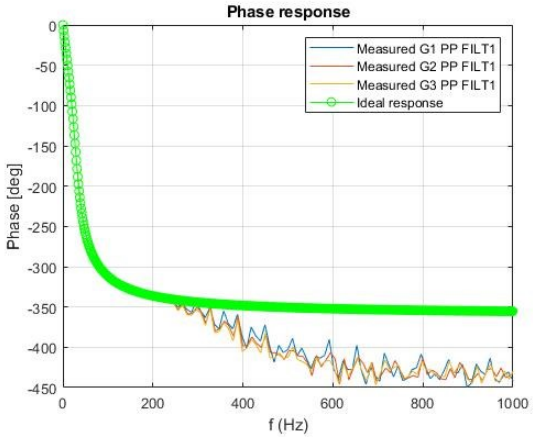


Figure 35: Phase response for filter setting 1 (right)

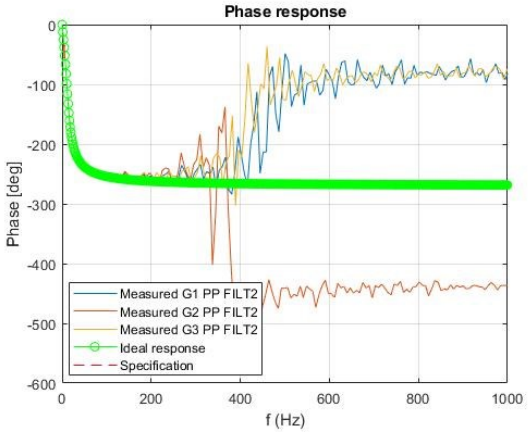


Figure 36: Phase response filter setting 2 (left)

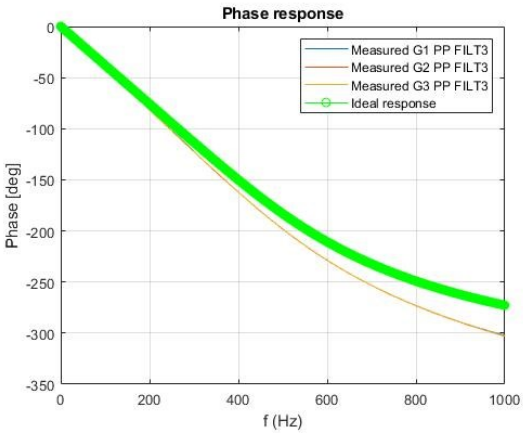
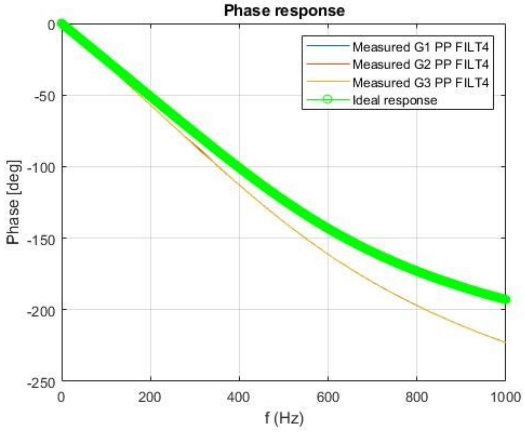


Figure 37: Phase response for filter setting 3 (right)



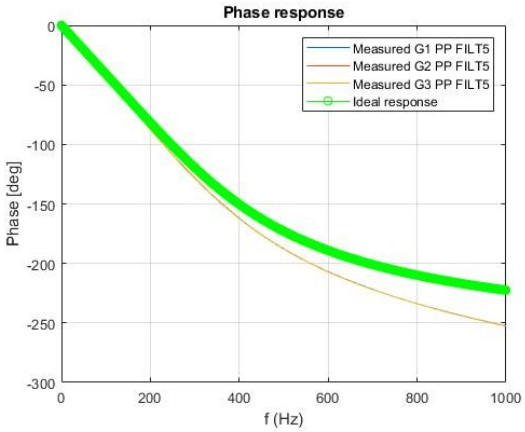


Figure 38: Phase response for filter setting 4 (left)

Figure 39: Phase response for filter setting 5 (right)

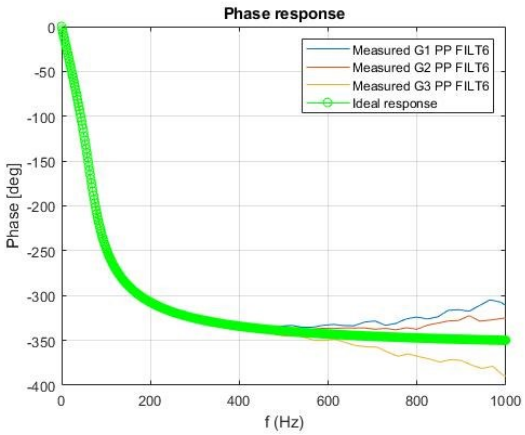
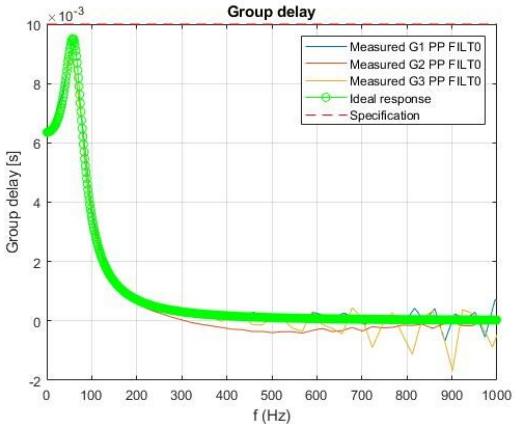


Figure 40: Phase response for filter setting 6 (filter setting 0 coefficients)

5.3.4 Group delay

Group delay was calculated from phase response in Matlab. With lower filter bandwidth settings, the filter group delay is within specification and follows the designed group delay until 100 Hz and then starts to drift off. With higher filter bandwidth settings, the filter group delay matches with the designed group delay, but has an additional offset in it, causing it to be out of specification.

Specification for group delay can be found in the last two columns of Table 2. The first column is group delay in milliseconds at 10 Hz and the second one is group delay in milliseconds after 10 Hz. However, only filter setting 1 has different value in column two, with all other filter settings the value is the same as in the first column.



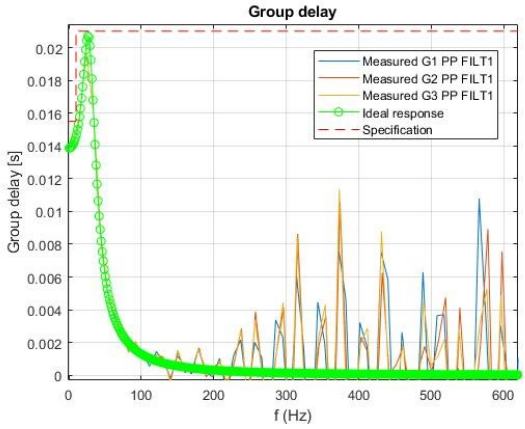


Figure 41: Group delay for filter setting 0 (left)

Figure 42: Group delay for filter setting 1 (right)

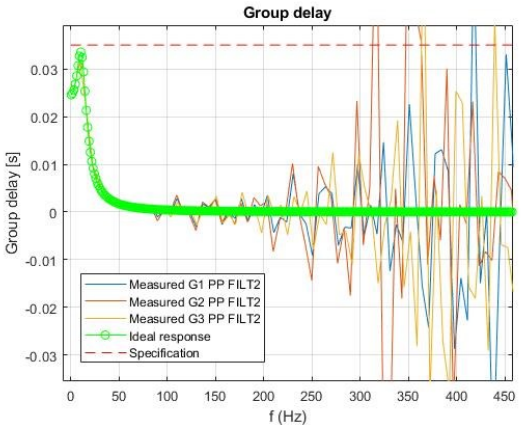


Figure 43: Group delay for filter setting 2 (left)

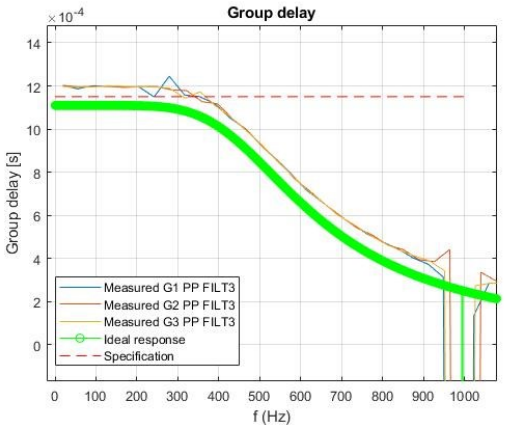


Figure 44: Group delay for filter setting 3 (right)

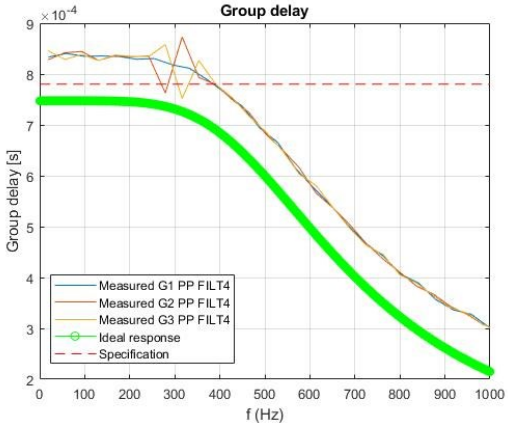


Figure 45: Group delay for filter setting 4 (left)

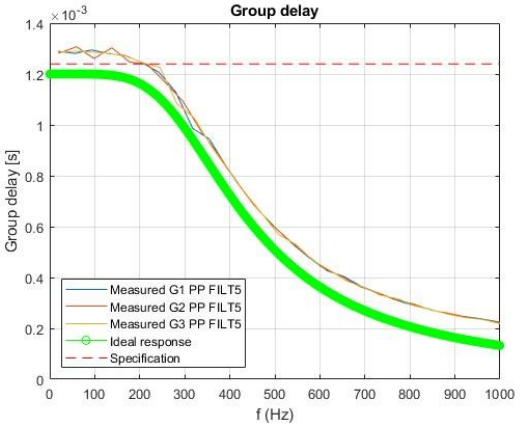


Figure 46: Group delay for filter setting 5 (right)

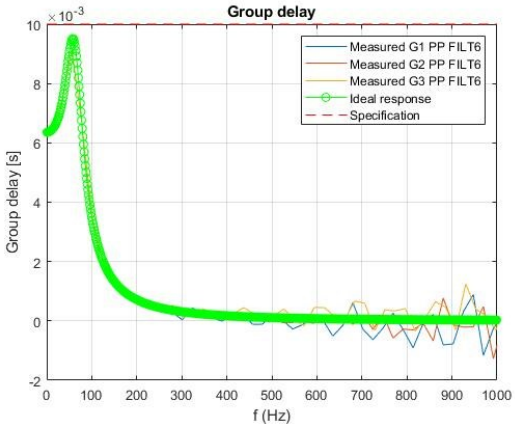


Figure 47: Group delay for filter setting 6 (filter 0 coefficients)

5.3.5 Bypass filter

Results for bypass filter mode include two registers: G* PP FILT7 and G* CAL FILT7. G* PP FILT7 is the output of the filter and G* CAL FILT7 is the register containing the filter control input data is. In this measurement the delay caused by F_s slower than test sampling frequency is clearly seen. Two measurements happen before register value is updated. G* CAL FILT7 updates twice as fast as G* PP FILT7, which can be seen in sample points 127 to 130. G* CAL FILT7 has only two values indicating the impulse whereas G* PP FILT7 impulse length is four values before output is back to zero.

In channel 3 results the impulse happens slightly later during the ASIC cycle, which causes the impulse to be seen later in G* PP FILT7 than in the other channels.

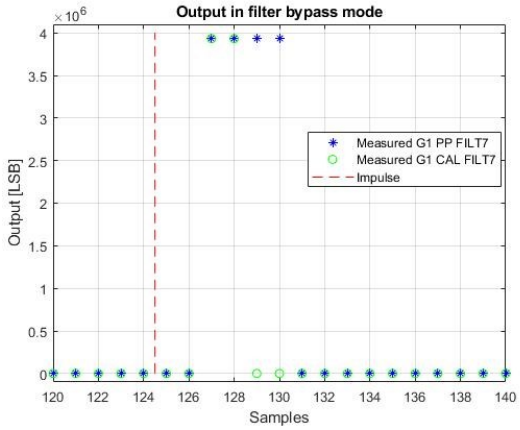


Figure 48: Output in filter bypass mode, channel 1 (left)

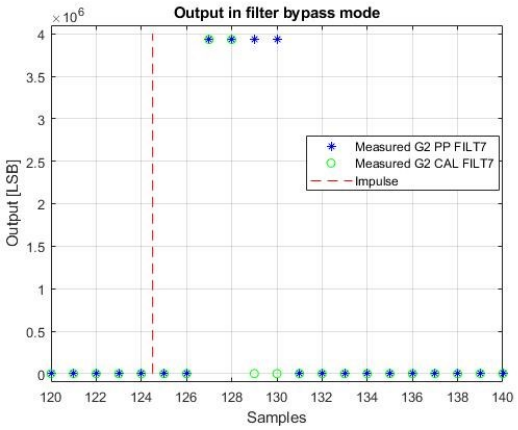


Figure 49: Output in filter bypass mode, channel 2 (right)

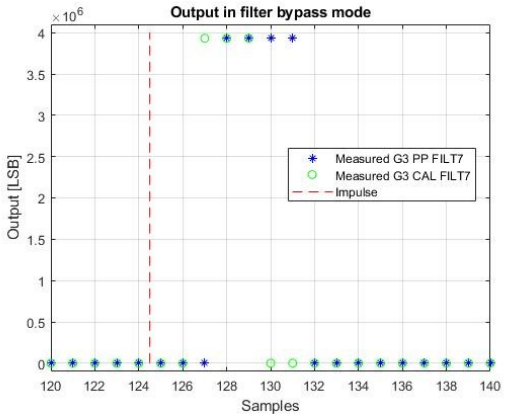


Figure 50: Output in filter bypass mode, channel 3

6 Conclusions

The measured results matched the designed responses well. The amplitude response matched the designed curves almost perfectly with the Bessel filters, but Butterworth filters started to drift away from the design value response after stopband. This is something that was seen in simulations and is an expected result. All the filters were within specification and very close to their designed -3dB points. Specification for amplitude response had multiple criteria.

The phase response of the filters started to drift away from the designed response after 100 - 200 Hz. There was no specification for most of the filters other than the low frequency results should follow the designed response, the point where response is -45° being the most critical point. All filters fulfilled this criterion, and the results were, if not good, at least acceptable. Measuring phase response is extremely sensitive about timing of the measurement and since the test setup was not optimal, we can only say the results are directive and not definite.

Group delay was calculated from phase difference and any errors in there directly reflect to group delay results. Group delay main specification was *x milliseconds delay at 10 Hz,* where x would change by the filter used. However, some filters had higher peaks after 10 Hz, so the highest peak in the graph was used to determine if the result was acceptable. As mentioned with phase response, results started to drift away from the designed response after 100-200 Hz, so results after that were mostly ignored, even though there was also specification: *x milliseconds delay for full frequency range.* This was interpreted to simply mean that no limit violations are allowed where the data is reliable, instead of the full frequency range.

Bypass filter results were almost as expected. When impulse was set to input register, value of register monitoring the input signal and the value of DUT output register saw the change after two samples when the ASIC had updated the register values. The input monitoring register returned from the impulse after two samples because the register is updated faster than DUT output register and the DUT output register returned to zero level after four samples, which is the expected duration of the ASIC cycle. The only oddity was a small delay in channel 3 filter output, which was caused by impulse happening later during the

ASIC cycle. Timing was not as repeatable as hoped for, but results are satisfactory and clearly show the correct operation of the bypass mode of the filter.

All in all, results were satisfactory and measured filter performance is almost as expected. This confirms the design of the post-processing filter works also when manufactured to a real ASIC. The characterization of the filter allows the use of these filters with this ASIC in Murata Finland's components.

During testing it became clear that some of the measurements were not repeatable, and measurements had to be rerun multiple times to get the expected result. This is probably due to ASIC timing and impulse timing mismatch. Especially the larger bandwidth filters were extremely elusive when trying to get the impulse response right; the filter usually didn't see any impulse, or the measurement started during the impulse and not at the beginning. It is related to ASIC updating its registers at a different time compared to other registers. It would be possible to make this better by using the DataReady-SYNC pin to synchronize the measurement with ASIC cycle. DataReady-SYNC allows the user to synchronize the readings from the component to other devices. This would, however, require a lot of work from multiple engineers and the results can be obtained without it thus making it more efficient to simply rerun the tests.

To make the measurement more accurate, exact timings should be used. Murata has a VI for measuring with accurate timing, but it was not available while these measurements were made. It is possible to define an initial command, in this case the impulse, with accurate timings and the rest is only reading of the correct registers where even MSB and LSB are automatically combined for user when the register is defined with correct parameters. It also allows the user to control the sampling frequency which makes it ideal for this test. This test should be repeated using this VI when it is available to see if the test would be more repeatable. Another approach would be to use a timing sensitive digital pattern editor to have full control over the timing of the SPI commands sent. While Murata now has this capability, it was not available during these measurements and would have required a huge amount of work to get full results as all commands need to be specified individually bit by bit.

The measurement could have also been conducted by sending the impulse only to the MSB input register to reduce the time between sending impulse and reading the output register.

This would allow more flexibility in the measurement, especially in the higher frequency filter measurements, but would still likely yield the same results.

The results would have been more precise with more measurement points during the impulse. This was limited by ASIC sampling frequency of 12 kHz, which severely reduces the accuracy of calculations. This can be seen in the graphs as long straight lines which sharply turn when it reaches its next data point. When conducting a time critical measurement like this it would be beneficial to have better sampling frequency.

References

Mathworks (2022a) *Math. Graphics. Programming.* Available at: https://www.mathworks.com/products/matlab.html (Accessed 20 November 2022).

Mathworks (2022b) *Fast Fourier transform.* Available at: https://www.mathworks.com/help/matlab/ref/fft.html (Accessed 25 November 2022)

Mitra, S. K. (1998) *Digital Signal Processing: A Computer-Based Approach.* McGraw-Hill.

National Instruments (2022) *What is Labview?*. Available at: https://www.ni.com/fi-fi/shop/labview.html (Accessed 20 November 2022).

Oppenheim, A. V. and Schafer, R. W. (2010) *Discrete-time signal processing.* 3rd edn. Pearson Education.

Oppenheim, A. V., Willsky, A. S., and Nawab, S. H. (1997) *Signals and Systems.* 2nd edn. New Jersey: Prentice-Hall, Inc.

Proakis, J. G. and Manolakis D. G. (1992) *Digital signal processing: Principles, Algorithms, and Applications.* 2nd edn. New York: MacMillan Publishing Company.

SafeSPI Organization (2021) *SafeSPI Specification v2.0*. Available at: https://safespi.org/specification (Accessed 20 November 2022).

Table 1: Filter coefficients

Filter setting	a	b
FILT0	1,-3.90538595579747,,-3.72493668798350,0.909716190041821	1.02506140131631e-07,4.10024560526524e-07,6.15036840789787e-07,4.10024560526525e-07,1.02506140131631e-07
FILT1	1,-3.95658794234727,5.87070364910068,-3.87163159685665,0.957515964648974	4.65910830749206e-09,1.86364332299683e-08,2.79546498449524e-08,1.86364332299683e-08,4.65910830749209e-09
FILT2	1,-2.98562325101211,2.97134966176443,-0.985726041962762	4.60986945699515e-08,1.38296083709855e-07,1.38296083709855e-07,4.60986945699515e-08
FILT3	1,-3.23612148075360,3.96128453443040,-2.17232360942431,0.450030888203082	0.00242334816819439,0.000717583113894660,-0.00341153010859945,0.000717583113894670,0.00242334816819439
FILT4	1,-2.33189382606763,1.83996542551964,-0.490419427971269	0.00856154878568047,0.00026453695468662 3,0.00026453695468664 1,0.00856154878568047
FILT5	1,-2.57504952516995,2.22276856591576,-0.642912623545695	0.00351872427473437,-0.00111551567467555,-0.00111551567467555,0.00351872427473437
FILT6	Programmable	Programmable
FILT7	Bypass	Bypass

Table 2: Specification for all filter settings

Mode	f3dB (Hz), minimum	f3dB (Hz), typical	f3dB (Hz), maximum	Order	Type	Passband (Hz)	Passband tolerance (dB)	Stopband Attenuation	Phase Response	Group Delay @ 10 Hz (ms)	Group Delay @ all Hz (ms)
FILT0	63,5	68	72,5	4	Butter	20	0,2	-60 dB @ 500 Hz	na	<10	<10
FILT1	29	30	33,5	4	Butter	7	0,2	-60 dB @ 200 Hz	na	<15.5	<21
FILT2	12	13	14,5	3	Butter	5	0,2	-32dB @ 50Hz -60 dB @ 150 Hz	-35..-55 deg @ 5 Hz	<35	<35
FILT3	262,5	281	299	4	Bessel		0,04	-40 dB @ 1000 Hz	na	<1.15	<1.15
FILT4	347	371	395	3	Bessel		0,04	-18 dB @ 1000 Hz -40 dB @ 2000 Hz	na	<0.78	<0.78
FILT5	220	235	250,5	3	Bessel		0,04	-30 dB @ 1000 Hz -50 dB @ 2000 Hz	na	<1.24	<1.24
FILT6	Prog. (10-400 Hz)			Prog. (1-4)	Prog						
FILT7	Bypass										