Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2018

Pedro Miguel Ribeiro Machado

Sonar Ultrassónico para cegos com Sonificação de obstáculos

Ultrasonic Sonar for the Visually Impaired with Obstacles Sonification

"Homem culto é aquele que,
de tudo a que assiste aumenta,
não os seus conhecimentos,
mas o seu estado de alma."

— Fernando Pessoa

Pedro Miguel Ribeiro Machado

Sonar Ultrassónico para cegos com Sonificação de obstáculos

Ultrasonic Sonar for the Visually Impaired with Obstacles Sonification

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Professor Doutor José Neto Vieira e coorientação do Professor Doutor Arnaldo Oliveira, Professores Auxiliares do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

To my father, thanks to whom I feel
(and know) I will never walk alone.

o júri / the jury

presidente / president	**Professor Doutor José Maria Amaral Fernandes** Professor Auxiliar da Universidade de Aveiro (por delegação da Reitora da Universidade de Aveiro)
vogais / examiners committee	**Professor Doutor José Manuel Neto Vieira** Professor Auxiliar da Universidade de Aveiro (orientador)
	Professor Doutor Diamantino Rui da Silva Freitas Professor Associado da Faculdade de Engenharia da Universidade do Porto

agradecimentos / acknowledgements

Gostaria de iniciar este pequeno trecho de texto por deixar expresso o meu agradecimento aos Professores Doutor José Vieira e Doutor Arnaldo Oliveira, orientador e coorientador, respetivamente, não só pela valiosa oportunidade que me concederam de realizar este trabalho, mas também por toda a orientação científica e prática ao longo destes meses, bem como pelo esforço empregue na transmissão de seus conhecimentos e experiência, que se revelaram imprescindíveis no decorrer do projeto.

A nível académico, devo deixar também um agradecimento a todos os meus colegas com os quais, ao longo do curso, eu trabalhei, discuti, errei, partilhei frustrações, desesperei e, como consequência, aprendi, ganhei conhecimento e experência que me ajudaram ao longo destes meses.

Gostaria de agradecer também ao Eduardo Miranda, à Catarina Santa Comba e ao Daniel Almeida, meus colegas de laboratório destes últimos tempos, não só pela ajuda disponibilizada, mas também pelas ocasionais gargalhadas que permitiram os instantâneos alívio e descompressão do estado de espírito, fundamentais para a melhoria da minha produtividade e bem-estar.

Não poderei esquecer-me de todas as pessoas que contribuíram para a formação do meu Ser, nomeadamente família e amigos mais íntimos. Todas as características do meu carácter foram moldadas pelas experiências e vivências, sejam elas boas ou más, que partilhei com as mesmas. Destas deverei destacar a minha Mãe, cuja resiliência e capacidade de luta me permitiram ter a oportunidade de elaborar este documento.

Por fim, e não menos importante (de todo), quero deixar um especial agradecimento à minha namorada Susana, não só pelo incondicional apoio que me porporcionou, mas também pela inabalável fonte de motivação que revelou ser para mim durante a elaboração desta Dissertação.

Resumo

Nesta tese de mestrado pretende-se desenvolver um dispositivo portátil que possa ser usado por pessoas com deficiência visual na ecolocalização de obstáculos. Este dispositivo deverá ser dotado de capacidade de emissão e deteção de ultrassons para funcionar como um sonar e permitir ainda o seu funcionamento como altifalante paramétrico capaz de realizar a sonificação dos obstáculos. Para tal, foi necessário desenvolver uma ADC Sigma-Delta em FPGA que permita uma alta densidade na aquisição independente de um grande número de canais num dispositivo de pequenas dimensões. Os testes realizados com a ADC Sigma-Delta revelaram uma baixa distorção e uma boa relação sinal ruído, comparáveis às ADCs do mesmo tipo existentes no mercado. Está assim preparado o caminho para a construção do dispositivo.

Abstract

In this master's thesis it is intended to develop a portable device that can be used by people with visual impairment in the echolocation of obstacles. This device must be capable of transmitting and detecting ultrasound signals to work as a sonar and still allow its operation as a parametric speaker capable of performing the sonification of obstacles. To do this, it was necessary to develop a Sigma-Delta ADC in FPGA that allows a high density in the independent acquisition of a large number of channels in a small device. Tests performed with the developed Sigma-Delta ADC revealed low distortion and good signal-to-noise ratio, comparable to same type ADCs available on the market. The path for the construction of the device is, then, open.

Contents

List of Figures

List of Tables

List of Acronyms

ΣΔM Sigma-Delta Modulator.

ADC Analog-to-Digital Converter.

BRAM Block Random-Access Memory.

CIC Cascaded Comb-Integrator.

dB Decibel.

DM Delta Modulator.

DoA Direction of Arrival.

DTFT Discrete-time Fourier Transform.

ENOB Effective Number of Bits.

FFT Fast Fourier Transform.

FIFO First in, First out.

FPGA Field-Programmable Gate Array.

GBDP Gain-Bandwidth Product.

GIS Geographic Information System.

GPS Global Positioning System.

HPF High-Pass Filter.

HW Harware.

IBSF International Blind Sports Federation.

ILA Integrated Logic Analyser.

LPF Low-Pass Filter.

LSB Least Significant Bit.

LVDS Low Voltage Differential Signalling.

MSB Most Significant Bits.

NS Noise Shaping.

NTF Noise Transfer Function.

OPAMP Operational Amplifier.

PCB Printed Circuit Board.

PDF Probability Density Function.

PMOD Peripheral Module.

PSD Power Spectral Density.

PWM Pulse-Width Modulation.

RF Radio-Frequency.

RMS Root Mean Square.

SFDR Spurious Free Dynamic Range.

SINAD Signal-to-Noise and Distortion Ratio.

SNR Signal-to-Noise Ratio.

STF Signal Transfer Function.

THD Total Harmonic Distortion.

ULA Uniform Linear Array.

UPA Uniform Planar Array.

VHDL VHSIC Hardware Description Language.

VHSIC Very High Speed Integrated Circuit.

WHO World Health Organisation.

ZOH Zero-Order Hold.

Chapter 1

Introduction

1.1 Motivation

Blindness and visual impairment are disabilities that greatly increase the hazard of accidents that might often result in injury and/or damage. The interaction of a person with the surroundings involves the use of all senses in order to avoid dangerous incidents and vision is, if not the most, one of the most important senses. Due to this fact, the independence of a blind person is greatly affected.

Blind people often adapt to the disability and end up relying on other senses, being one of them hearing. However, the sole reliance on this sense does not replace the safety that vision bestows upon sighted people.

One important characteristic of hearing is echolocation, that is the ability to detect an object at a distance and its relative position by sensing echoes. This sense is not quite developed in humans. However, some outstanding cases of blind people have been reported where they are able to safely navigate in the surroundings, successfully identifying objects by sensing the echo of sounds such as mouth-clicks, mainly in cases of people that were born blind or developed the disability in very early stages of their infant life, such as the case of Daniel Kish[1].

That said, it is apparent that a person that develops this disability at a more advanced age very seldom will develop the ability of echolocation.

Nowadays, it is relatively easy to identify electronically the direction of arrival of an echo. This means that identifying the relative position of an object by sensing an echo can be made with accuracy much superior to that of the untrained human hear. The technique that achieves this is called *beamforming* and allows to both direct a beam of ultrasonic or even Radio-Frequency (RF) waves as well as identify the Direction of Arrival (DoA) of the same carrier coming from a source with some relative position.

Using a *parametric speaker*, both can be done and, thus, achieve a system that can identify the location of an obstacle as well as direct a beam of ultrasonic waves towards it, that will reflect on the object and *sonify* it, where it creates the semblance of being the obstacle the sound source.

Such a system might be important in the adaptation of visually impaired people that developed the ailment at a more advanced stage of life, when their echolocation sense is

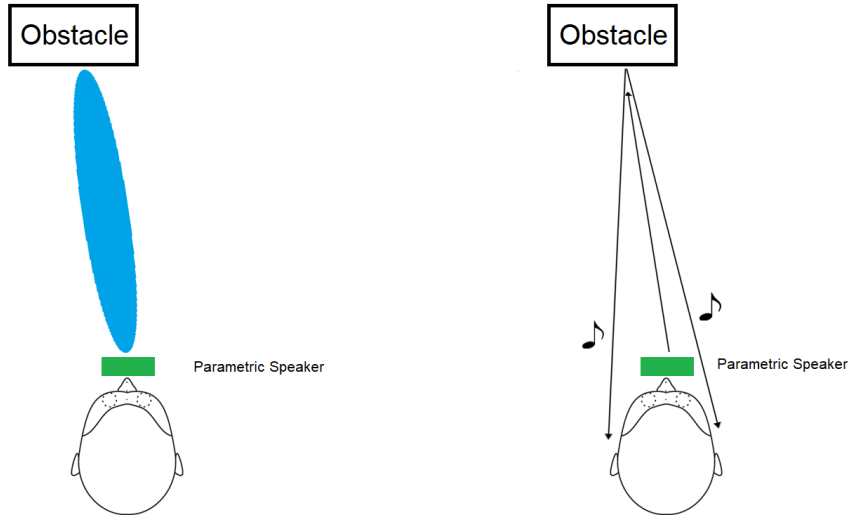
underdeveloped and, thus, are not able to distinguish an echo coming from an obstacle from the remaining echoes from the harmless surroundings.

1.2 Proposed Work and Objectives

It is intended to develop a portable sonar with *parametric speaker* that aids blind people to detect obstacles.

An ultrasound emitter should emit ultrasonic pulses that radiate in a wide field, in order to detect as much obstacles as possible with the echoes, whose DoA will be determined by the sonar. However, only in very specific situations should danger be encountered and that is when an obstacle is radially approaching the carrier of the system. In that case, due to the *Doppler Effect*, the received echoes will present a slightly different and greater frequency, because of a positive relative velocity. This frequency shift is the condition to look for in order to activate the warning mode. In this mode, a *Parametric Speaker* will direct towards the obstacle a beam of ultrasonic wave modulating an audible warning signal that will demodulate due to non-linear effects of its interaction with the air. This way, it creates the illusion that the obstacle is emitting an audible signal, easing its detection.

Figure 1.1 depicts the basic operation of the system.



(a) Sonar Detection. (b) Sonification of the obstacle.

Figure 1.1: Basic graphic description of the system's operation where both detection and warning modes are depicted.

The *Parametric Speaker* that is shown in Figure 1.2 has been previously designed and implemented to perform this function [2], as well as the software to direct the beam using the *beamforming* technique. The proposed work is to complement it with a sonar (that is a phased array of ultrasound transducers) capable of receiving the said echoes, compute the DoA and detect the risk of collision with an obstacle.

The following tasks should be completed:

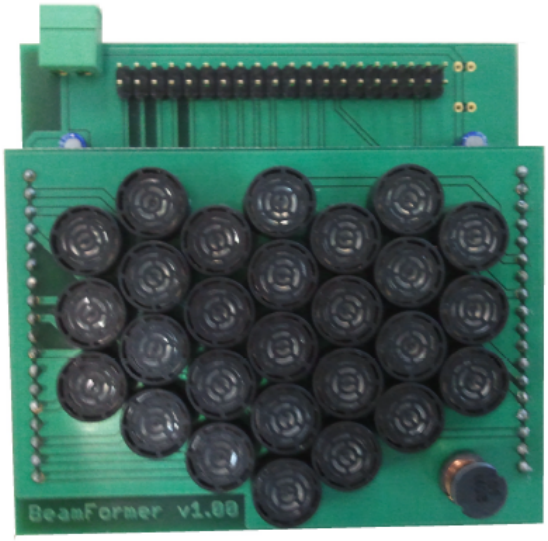


Figure 1.2: Parametric Speaker developed by [2].

Analog-to-Digital Converter Estimating the DoA involves digital signal processing techniques and, thus, the received echos must be sensed and sampled in order to perform such task. The main objective in this task is to develop an Analog-to-Digital Converter (ADC) that is as compact as possible, with the least hardware complexity that is possible to attain without compromising good performance.

Downconversion For each received echo, it is needed to evaluate the frequency, phase and amplitude of the received signal. In the downconversion process, where *In-Phase* and *Quadrature* components of the received signal are extracted, allow to compute the phase of the received signal, as well as its amplitude.

Beamformer From the data extracted from several downconverted received signals (from each transducer of the array), it is possible to determine the DoA of the echo. This computation is done by the beamformer.

1.3 Organisation

This document is divided in five chapters, which are briefly described here.

Chapter 1 A small introduction on the motives, objectives and organisation of this Thesis is done.

Chapter 2 A brief description of the state-of-the-art on the subject. What is known about blindness and echo location and some recent electronic aids.

Chapter 3 Some theory is introduced on the operation of a $\Sigma\Delta$M. The design and implementation of a $\Sigma\Delta$M is described, as well as the tests made. The results are presented and analysed.

Chapter 4 The theory of DoA is presented. A reception channel is designed, implemented, subjected to testing. The results are presented and discussed.

Chapter 5 Conclusion of the thesis and discussion of the objectives accomplished. Future work is also discussed.

Chapter 2

State of the Art

2.1 Blindness and vision impairment

Vision impairment "refers to deficits in the ability of the person to perform vision-related activities of daily living, such as: reading, orientation and mobility, and other tasks" [3].

Vision impairment, is classified having in mind the degree of difficulty the impairment brings in the near and far fields of vision [4]. At a distance, the degrees of impairment are:

- Mild
- Moderate
- Severe
- Blindness

Thus, blindness is the most severe form of vision impairment, where the sense of vision is diminished to none or close to it.

World Health Organisation (WHO) estimates that nearly 1.3 billion people in the world suffer from some kind of vision impairment, 36 million of which suffer from blindness [4].

It is also expected that the risk of more people being affected by some sort of vision impairment increases in the near future due to population increase and ageing.

Mobility and Environment Sensing

Mobility is the ability to move freely and easily between different locations in space. It is safely done with a good interaction with the environment and such requires a precise sensing of all the elements in the surrounding space.

The main challenge of people with severe vision impairment is the safe interaction with their surroundings, due to the fact that vision is the most important sense that facilitates this. Reliability on other senses is the key to a healthy interaction with the physical environment.

Nowadays, mobility for blind people is mainly possible due to the use of the traditional cane, depicted in Figure 2.1.

The cane allows a person with visual disabilities to detect obstacles that might present themselves at the ground level, like stairs or even posts. The sense that is most used with the cane is touch, because an obstacle is detected when the cane hits is and, thus, vibrates.

Figure 2.1: Traditional cane that facilitates blind people mobility [5].

Other good example on how blind people rely on other senses is Blind Football, that is currently regulated by International Blind Sports Federation (IBSF). The pitch must be uncovered to allow optimum acoustics; the ball has a sound system inside that makes a jiggling or rattling sound whenever it moves, so that players can better locate it; the field has kickboards that run along the side lines in order to be physically sensed; and there are specific areas for the presence of a guide that verbally guides the players [6]. Figure 2.2 depicts a Free Kick during a Blind Football match. Notice the kickboard parallel to the sideline on the left and the guide standing behind the goal, guiding the striking team.

Figure 2.2: Free Kick at the 2016 Paralympic Games in Rio de Janeiro [7].

Hearing and Echolocation

Hearing is the capability of perceiving sounds. Blind people, not being able to see, usually end up relying to a greater extent on their hearing capabilities when comparing to non visually impaired people. In fact, neuroimaging studies have suggested that the brains of early-blind people suffer structural changes due to this fact [8].

Echolocation is the ability to detect obstacles in the environment by sensing an echo. Bats are known to move in space through such sensing. Some dolphin species navigate in murky waters in the same way.

Humans are not so sensitive to these stimuli. However, a small number of blind people was reported to have outstandingly applied echolocation to interact with the surroundings, through the emission of sound signals such as mouth-clicks [9]. In fact, Bo N. Schenkman and Mats E. Nilsson [10] showed that both blind and sighted people can effectively locate obstacles at a near distance (less than 2 m), although blind people scored better, and that outstanding echolocators' performance was superior as well for greater distances.

Lore Thaler *et al.*, studied this sense in expert echolocators with neuroimaging scans and the findings suggest that the processing of the mouth-click echoes in the brain is similar to that of vision in sighted people [11], adding evidence that states that blind brains adapt to the lack of vision with other senses to increase capability of obstacles detection and navigation.

2.2 Assisting Devices

Besides the traditional cane shown in Figure 2.1, other kind of devices, technically more advanced, have been emerging recently. Some examples will be discussed in this section.

Cane equipped with sonar

The traditional cane most blind people use allows one to sense a surface with its tip. As a blind person moves in space and scans the surface with the tip of the cane with a circular motion, a lot of area is left unprobed. This flaw leads to the danger of not detecting small holes on the ground, and so the carrier of the cane might step on it and fall.

This cane, depicted in Figure 2.3, developed at University of Aveiro, is equipped with a sonar probe in the middle of its length and detects holes in areas not probed with the tip of the cane in an effort to complement the traditional white cane's features. It then warns the carrier of the hole existence through a vibration.

However, the major disadvantages it presents have to do with portability, given the fact that it is relatively heavy and it does not detect obstacles at levels higher than the ground.

Sunglasses with Sonar

This device, also developed at the University of Aveiro, consists on sunglasses that are equipped with sonar detection. Its main feature is to detect obstacles at head level and the system is shown in Figure 2.4.

The glasses are equipped with two ultrasonic emitters that generate ultrasound pulses and one receiver that receives the echoes. Whenever it detects an obstacle at head level approaching it will send a sound warning to the carrier.

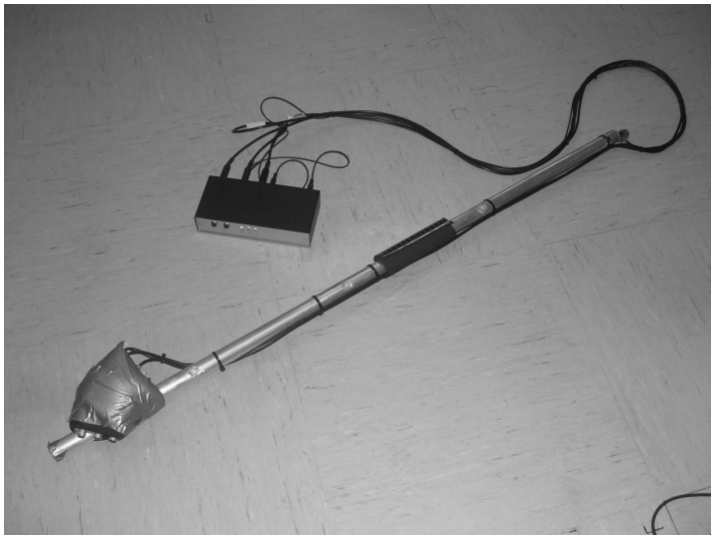


Figure 2.3: Prototype of the Cane Equipped with Sonar [12].

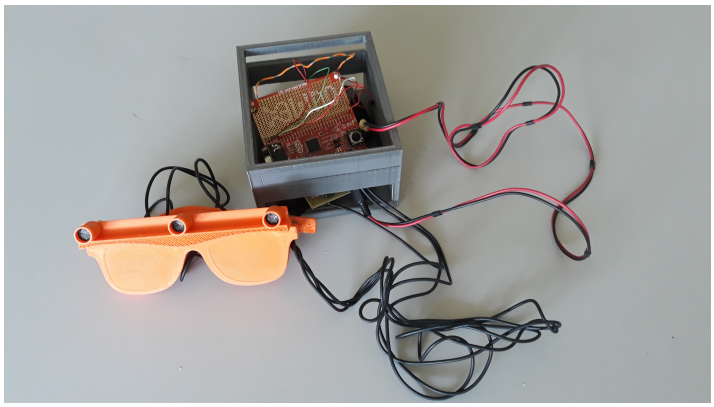


Figure 2.4: Prototype of the Sunglasses Equipped with Sonar.

The main disadvantage is that the equipment has no speaker and, thus, the sound warning is sent to headphones. This way, whenever an obstacle is detected, the blind carrier will momentarily lose his ability to listen to other ambient noises in order to focus on the warning signal. Most blind people are not comfortable with this trade-off.

NAVIG

This device is intended to help visually impaired people avoid obstacles, while mapping the carrier's location using Global Positioning System (GPS), adapted Geographic Information System (GIS) and computer vision. Its prototype is depicted in Figure 2.5.

The fusion of GPS, GIS and computer vision is mainly intended to precisely locate the users position. Besides that, the computer vision module detects relatively small and unexpected objects that might appear, as well as guide the blind person towards an intended object (mail box, for example), through a sound system that needs headphones to correctly operate.

Figure 2.5: Prototype of the NAVIG system [13].

2.3 Parametric Speaker

A Parametric Speaker is a device that enables to project sound waves in a very specific direction, as opposed to a conventional speaker, that scatters around the sound waves.

Its applications vary creatively and it is mainly intended to place the sound in a very specific location, preserving a degree of privacy. In a commercial brochure [14], Sennheiser recommends its Parametric Speaker, depicted in Figure 2.6, in situations like Museum exhibitions, some automatic machines, like ATM's and voting booths, workspaces and other situations where confidential information might be transmitted in audio signals.

Figure 2.6: Sennheiser's Audiobeam Parametric Speaker.

The same brochure states that the parametric speaker, whenever targeting the beam to a place where it is supposed to reflect, it becomes also audible. This way it creates the illusion that a sound is coming from that point of reflection.

This phenomenon is intended to be explored in this project, once it should be easier to

detected an obstacle in space by "hearing it" when it is the only one in the surrounding space emitting a certain audible signal.

The *Parametric Speaker* developed in [2] and depicted in Figure 1.2, that is intended to be used in this system, showed an overall good behaviour. It was shown to be able to correctly steer a modulated beam of ultrasound waves towards the intended direction, according to user inputs, although minor deviations were detected.

In fact, tests revealed that the angle resolutions in both *azimuth* an *elevation* angles were measured to be around 1^o, at a distance of 85cm from the transducer array.

Figure 2.7 shows the result of an experiment made in [2]. The sound pressure was measured at a distance of 85cm from the *Parametric Speaker*, when it was emitting a centered beam, *i.e.*, set with 0^o *azimuth* and *elevation* angles. It shows the higher sound pressure zones in red and the lower pressure zone in blues. It is visible that the higher pressure zones are around the origin, as it is intended.

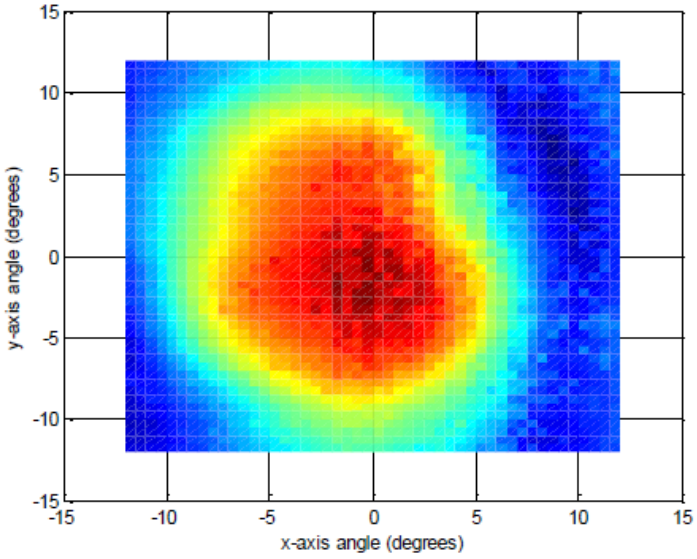


Figure 2.7: Sound pressure distribution in a plane for a centered beam [2].

Chapter 3

Sigma-Delta Converter

3.1 Introduction

The detection of objects in a sonar system involves the reception of some ultrasonic system and its posterior processing. This said, one fundamental step of this process involves Analog-to-Digital conversion.

In this chapter, a brief overview on some ADC systems is made, specially the $\Sigma\Delta$ ADC, whose main characteristics are explored and, later, it is implemented in Field-Programmable Gate Array (FPGA) and analog electronics and its performance measured and analysed.

3.1.1 Sampling Methods

Sampling is the method used to convert an analog signal into a digital one, extracting several samples from the first at a constant rate, called sampling frequency, and there are two main ways to do it.

The first way, and the most conventional, is done at a rate higher, although near, the double of the signal's highest frequency, its Nyquist rate. The Nyquist Theorem states that a signal should be sampled at a frequency greater than the double of its highest frequency, avoiding the so called *aliasing* phenomenon.

Let's consider a generic and analog signal in the time domain, $x_a(t)$, and its Fourier Transform, $X_a(f)$. Sampling the analog signal with a stream of unit impulses (delta function), separated from each other by a constant time period, T_s, meaning a constant sampling frequency, F_s, results in a discrete signal [15]:

$$x_a(n) = x(nT_s), \quad -\infty < n < \infty \tag{3.1}$$

And, in the frequency domain:

$$X_a(f) = F_s \sum_{k=-\infty}^{\infty} X(f - kF_s) \tag{3.2}$$

This means that the frequency spectrum of the original signal gets shifted of kf_s frequency units and so it is replicated an infinite number of times, each replica centred at multiples of the sampling frequency. In the case that f_s is smaller than twice the bandwidth of the baseband spectrum, every replicas overlap, creating the *aliasing* phenomenon, as illustrated in Figure

3.1, resulting in added frequency components to the baseband spectrum and, thus, undesired interferences.

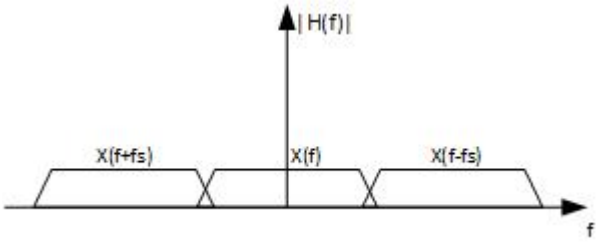


Figure 3.1: Aliasing Effect.

Therefore, considering non-idealities in practice, it is necessary to always sample a signal with a sampling frequency slightly greater than its Nyquist frequency.

This leads to the need of a Low-Pass Filter (LPF) with a narrow transition band to filter the continuous signal, protecting the output from *aliasing*. This filter will be called *aliasing* filter and its required transition band is not just difficult to implement, but also expensive.

The second way to sample a signal is less demanding on the LPF's transition band requirements. It's called *oversampling* and it consists on sampling with a frequency much higher than the signal's bandwidth. Considering a generic signal $x(t)$, with a baseband bandwidth B, sampling with a rate f_s implies that the low-pass filter's transition band will begin at B and end at frequency $f_s - B$. In *oversampling*, beeing that $f_s \gg B$, the transition band required is wide, comparing to the Nyquist rate sampling case, resulting in a need of a less complex and cheaper filter[16].

3.1.2 Quantization noise

Usually, quantization follows the sampling process, rounding the sampled value to the nearest value of previously specified quantization levels, *i.e.*, the division of the analog signal amplitude range by several equally spaced levels.

Considering Δ as the amplitude of level separation, *i.e.*, the space between quantization levels, the largest quantization error occurs when the quantized sample, before being rounded, has a value that falls exactly in between two levels so, for some quantized value, the error can be measured from $-\Delta/2$ to $\Delta/2$, and it is assumed to be uniformly distributed along this interval. Thus, the noise power is [17]:

$$P_n = \sigma^2 = \int_{-\Delta/2}^{\Delta/2} \frac{r^2}{\Delta} dr = \frac{1}{\Delta}\left[\frac{r^3}{3}\right]_{-\Delta/2}^{\Delta/2} = \frac{1}{\Delta}\frac{2\Delta^3}{24} = \frac{\Delta^2}{12} \tag{3.3}$$

Being r the value of the quantization error or, in other words, the distance of the sampled value from the nearest quantization level.

Considering that the quantization error is uniformly distributed along Δ, it adds white noise and is not related to the analog signal. As such, both *oversampling* and Nyquist rate sampling processes will be affected with the same noise power, dependent only on the quantization resolution.

However, in *oversampling*, the noise power spreads over a wider frequency band, due to its much greater sampling frequency[16]. Thus, inside the input signal's baseband, comparing

to a Nyquist rate ADC, an *oversampling* converter has a smaller fraction of noise uniformly distributed, reducing noise power inside the baseband and increasing Signal-to-Noise Ratio (SNR).

3.1.3 Performance

Signal-to-Noise Ratio

The SNR quantifies the ratio between the power of a given signal and the power of the noise it contains. It is usually expressed in Decibel (dB) and is calculated according to the following equation:

$$SNR = 10 \times log_{10}\left(\frac{P_{signal}}{P_{noise}}\right)[dB] \tag{3.4}$$

One way to compute the powers that make up the fraction in Equation 3.4 is to determine the PSD of a signal, which is what characterises the scattering of power in the frequency spectrum, find the frequency components correspondent to signal and to noise (and, eventually, to harmonics), and then sum the power present in the respective frequencies. It is usual to find in a PSD harmonic components of the input signal, as illustrated in Figure 3.2. The harmonic components can be discarded out of this computation, as they add power in the noise band to a great extent, not being noise components, but the result of non-linear effects imposed by a system.

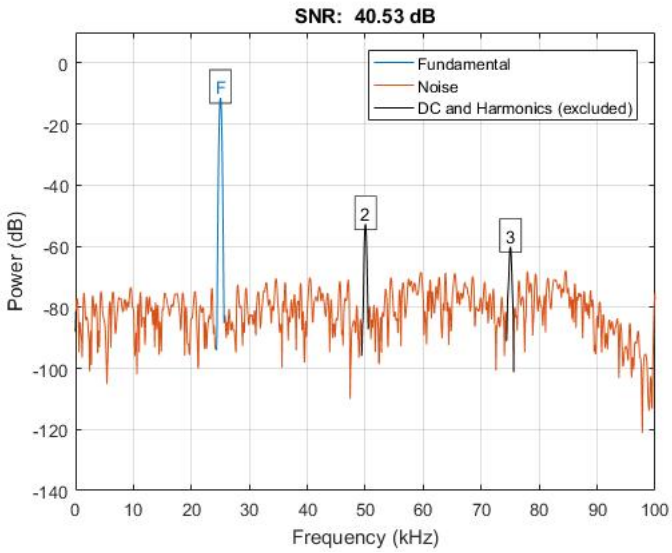


Figure 3.2: PSD of a sinusoidal signal with noise and harmonic components.

Signal-to-Noise and Distortion Ratio

The Signal-to-Noise and Distortion Ratio (SINAD) measures the ratio between the signal power and the noise and harmonics powers added together. Also often expressed in dB, it is,

so, given by:

$$SINAD = 10 \times log_{10}\left(\frac{P_{signal}}{P_{noise} + P_{harmonics}}\right) \tag{3.5}$$

Spurious-free Dynamic Range

Spurious free dynamic range (SFDR) is the ratio of the Root Mean Square (RMS) value of the signal to the RMS value of the worst spurious interference, regardless of where it falls in the frequency spectrum, and it represents the smallest value of signal that can be distinguished from a large interfering (spurious) signal.[18]. It is computed by:

$$SFDR = 10 \times log_{10}\left(P_{fund}\right) - 10 \times log_{10}\left(P_{spur}\right) \tag{3.6}$$

Total Harmonic Distortion

The Total Harmonic Distortion (THD) is the ratio between the power in all the harmonic components and the fundamental power and relates to the linearity of the ADC, meaning that a lower THD value means less signal dependent distortion.[19]. Thus, its value in dB is given by:

$$THD = 10 \times log_{10}\left(\frac{\sum_{n=1}^{N} P_{harm_n}}{P_{signal}}\right) \tag{3.7}$$

Resolution

The resolution of an ADC quantifies the number of possible outputs when compared to the input range. For example, given a N-bits ADC, its resolution is $\frac{1}{2^N}$ of its input peak-to-peak value. It can also be stated that it measures the amount of voltage that a Least Significant Bit (LSB) quantizes.

Due to introduced noise and distortions, however, some LSBs might be encoding noise amplitudes that are greater than the resolution. Given this fact, only a portion of the N bits encodes the input signal and it is named Effective Number of Bits (ENOB). The ENOB tells the number of bits that encode useful information and can be computed by:

$$ENOB = \frac{SINAD - 1.76}{6.02} \tag{3.8}$$

3.1.4 Delta Modulation

A Delta Modulator (DM) is an example of an *oversampling* converter. Its block diagram is depicted in Figure 3.3.

The DM, through a feedback loop, is intended to perform periodically a two-level quantization of the difference between an input signal, $X(s)$, and its approximation, $X_q(s)$. Considering that the integrator acts as a LPF, when the mentioned difference is positive, meaning that the input is greater than its approximation signal, the output sample, $Y(s)$, is set to the logic level 1 and so the integrator will increase its output by a small amount, Δ (hence its

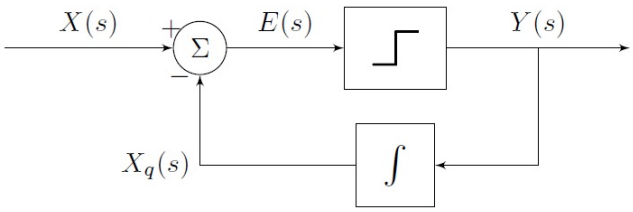


Figure 3.3: Delta Modulator block diagram.

name), getting it closer to the input sampled value. On the other way, if the difference is negative, the output of the modulator will be set to 0, decreasing the value of the approximated signal by Δ units. Such process is depicted in Figure 3.4.

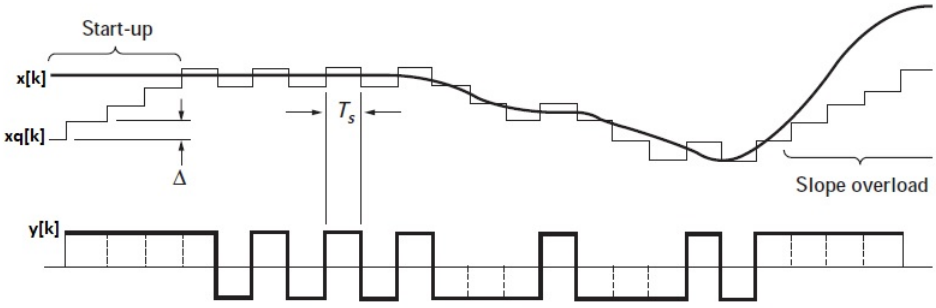


Figure 3.4: Delta Modulator's signals [20].

The demodulation of the output signal is based on the modulator's principle. An integrator converts the pulses from the modulator output to a stair shapped approximation of the input signal and a LPF converts the later into the regenerated input signal. Figure 3.5 portrays a DM immediately followed by its demodulator.

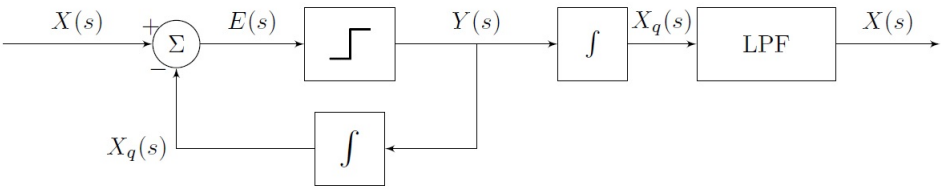


Figure 3.5: Delta Modulator followed by its demodulator.

On the right-hand side of Figure 3.4 is evident the biggest limitation of this type of modulators. The input signal must have its frequency very well delimited, otherwise it might change at a rate in which the modulator is not able to match with successive Δ additions/-subtractions, unless sampling frequency or the Δ increased/decreased by the integrator are adapted (which is highly unpractical) and thus creating undesired distortions to the modulated signal. This phenomenon is called *slope overload* and is caused by the modulator's

topology.

Taking into account Figure 3.6, and considering the respective signals in the $\mathcal{Z}$ domain, it's easy to understand that the output signal is given by the error signal, $E(z)$, added with quantization noise, $N(z)$:

$$Y(z) = E(z) + N(z) = X(z) - X_q(z) + N(z)$$

It is fair to assume that the noise introduced by quantization is negligible when compared to the other signals:

$$Y(z) \approx X(z) - X_q(z)$$

Given the fact that the integration in the $\mathcal{Z}$ domain involves a delay (one sampling period), and having in mind that $X_q(z)$ is an approximation of the input signal, the output of the DM can be considered as approximately the difference between two consecutive input samples:

$$Y(z) \approx X(z) - X(z-1) \tag{3.9}$$

It is now possible to conclude that the output of the DM is roughly a differentiation of the input signal, making it highly sensitive to it's variations and thus allowing input frequency related phenomena, such as *slope overload*, to happen.

Being that the noise added to the system during quantization is a result of stochastic process, it shows a non linear behaviour. Being so, in order to analyse its impact on the system, the modulator is approximated to is linear model, illustrated in Figure 3.6, whose signals are represented in the $\mathcal{S}$ domain.

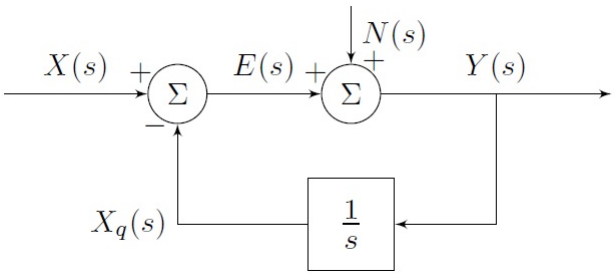


Figure 3.6: Delta Modulator's linear model.

The output of the modulator is given by:

$$Y(s) = E(s) + N(s) = X(s) - \frac{1}{s}Y(s) + N(s)$$

Solving for $Y(s)$, it is obtained:

$$Y(s) = \frac{s}{s+1}\left[X(s) + N(s)\right] \tag{3.10}$$

Being $s = j\omega$, it is easy to note that the DM acts as a High-Pass Filter (HPF) for both the input signal and the quantization noise.

3.1.5 Sigma-Delta Modulation

First Order

The ΣΔM, also refered to as Delta-Sigma, is a variation of the previous *oversampling* modulator. Considering Figure 3.5, the signal at the input of the demodulator's LPF is:

$$X_q(s) = \frac{1}{s}Y(s) \approx \frac{1}{s}\left[X(s) - X_q(s)\right] = \frac{1}{s}X(s) - \frac{1}{s}X_q(s)$$

Therefore, it is roughly equivalent to sample the difference between the input signal and the modulated output, both separately integrated, and later recover with a LPF. In other terms, the ΣΔM samples the difference between the integrated input and the approximated output. This is the ΣΔM mathematical principle, and this modulator is depicted in Figure 3.7.

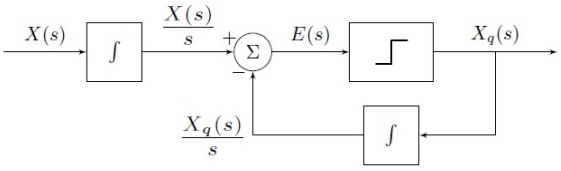


Figure 3.7: Sigma-Delta Modulator.

Given the fact that:

$$E(s) = \frac{X(s)}{s} - \frac{X_q(s)}{s} = \frac{1}{s}\left(X(s) - X_q(s)\right)$$

The ΣΔM represented in Figure 3.7 can be simplified by combining both integrators into one, placed right after the adder, as depicted in Figure 3.8, now with the output named $Y(s)$.

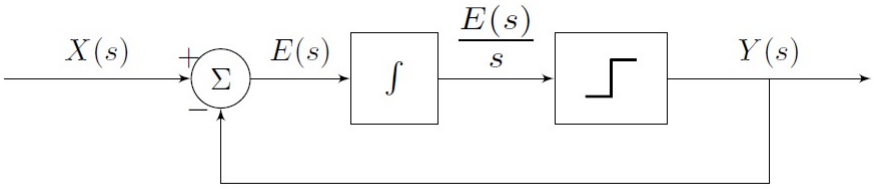


Figure 3.8: Simplified Sigma-Delta Modulator.

The quantization noise added by the comparator is a stochastic process. Being so, the noise added to the system is non-linear and can not be described through some mathematical relationship. In order to perform the noise impact analysis on the modulator, it's topology is approximated to a linear model, depicted in Figure 3.9, where the noise is referred to as $N(s)$ and so a linear behaviour is attained, which depends on the noise signal, that is non-linear.

The output of this block diagram is given by:

$$Y(s) = \frac{E(s)}{s} + N(s) = \frac{1}{s}\left[X(s) - Y(s)\right] + N(s)$$

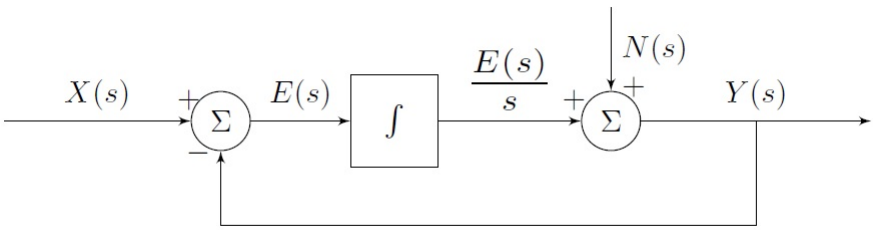


Figure 3.9: The Sigma-Delta Modulator linear model.

Solving for $Y(s)$, the following result is obtained:

$$Y(s) = \frac{1}{s+1}X(s) + \frac{s}{s+1}N(s) \tag{3.11}$$

Considering the fact that $s = j\omega$ and the absolute value of $\frac{1}{s+1}$ tends to infinity, the term (also known as Signal Transfer Function (STF)) has a transfer function similar to one of a LPF and the term $\frac{s}{s+1}$ (the Noise Transfer Function (NTF)) represents a transfer function similar to one of a HPF. Given this fact, it is possible to infer that the ΣΔM will not affect the input signal, as long as its frequency content does not exceed the LPF cut-off frequency, and the quantization noise will be pushed towards higher frequencies of the spectrum, reducing the noise in the baseband to a greater extent. This occurrence is known as *Noise Shaping* and it can be verified in Figure 3.12.

Considering Figure 3.8 and its signals in the $\mathcal{Z}$ domain, the output is:

$$Y(z) = \frac{1}{1-z^{-1}}E(z) = \frac{1}{1-z^{-1}}\left[X(z) - Y(z)\right]$$

Solving for $Y(z)$, the expression results in:

$$Y(z) = \frac{X(z) + Y(z-1)}{2} \tag{3.12}$$

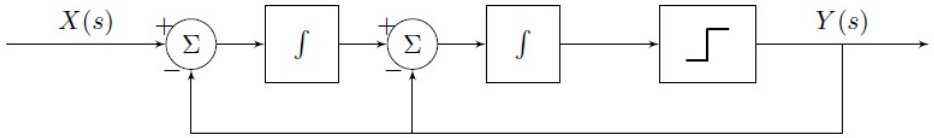
Therefore, opposing to the conclusions reached in equation 3.9, it is inferred that, contrary to the DM, the ΣΔM's output is not sensitive to its input variation and, therefore, it will not exhibit the slope overload effect.

In conclusion, the ΣΔM is to perform better than the DM for it will present under no circumstances slope overload distortion and it has the ability to significantly decrease the noise density in the bandwidth of interest, that is, the baseband, increasing even more the SNR of the recovered signal.

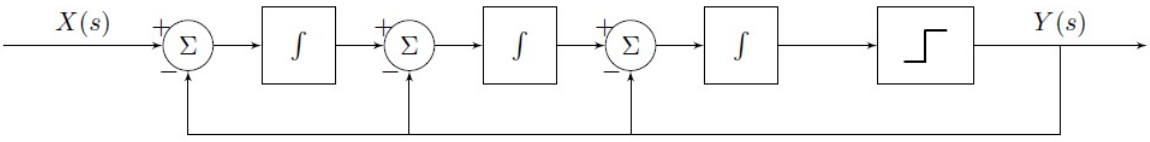
High-Order Sigma-Delta

High-order ΣΔMs are implemented by cascading N integrators fed-back with the output signal. Figure 3.10 depicts two examples of high-Order ΣΔM.

Figure 3.11 depicts a second order ΣΔM reduced to its linear model, where some non-linear quantization noise, N(s), is added in the place of the comparator, which also acts as a 1 bit quantizer.



(a) 2nd Order.



(b) 3rd Order.

Figure 3.10: Two Different High-order Sigma Delta Modulators.

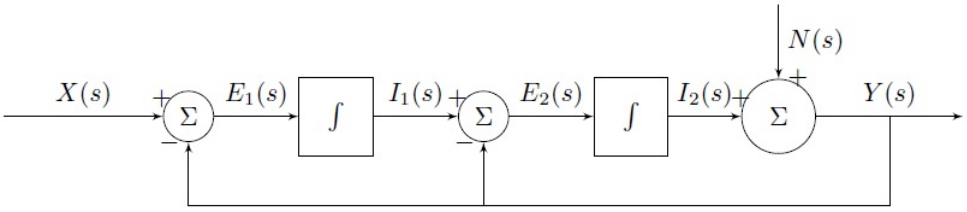


Figure 3.11: Linear model of a 2nd order ΣΔM.

Let its output be defined as the sum of $N(s)$ and $I_2(s)$:

$$Y(s) = I_2(s) + N(s) = \frac{E_2(s)}{s} + N(s)$$

Being $E_2(s) = I_1(s) - Y(s)$ and $I_1(s) = \frac{E_1(s)}{s}$, the previous expression can be stated as:

$$Y(s) = \frac{\frac{E_1(s)}{s} - Y(s)}{s} + N(s)$$

Which, considering that $E_1(s) = X(s) - Y(s)$, and solving for $Y(s)$, the transfer function of the system can be obtained:

$$Y(s) = \frac{1}{s^2 + s + 1}X(s) + \frac{s^2}{s^2 + s + 1}N(s)$$

Following the same procedure for the 3rd order ΣΔM, it yields the transfer function defined as:

$$Y(s) = \frac{1}{s^3 + s^2 + s + 1}X(s) + \frac{s^3}{s^3 + s^2 + s + 1}N(s)$$

It is, then, easily deducted that, for a N-order ΣΔM, its transfer function, as a function

of the input signal ($X(s)$) and the quantization noise ($N(s)$) is given by:

$$Y(s) = \frac{1}{\sum_{n=0}^{N} s^n} X(s) + \frac{s^N}{\sum_{n=0}^{N} s^n} N(s) \tag{3.13}$$

In terms of absolute value of the transfer function, being $s = j\omega$, for high frequencies, the STF tends to zero, acting as a LPF. On the other hand, the term with greater power on the denominator of the NTF gets much greater than the remaining terms, so $\sum_{n=0}^{N} s^n \approx s^N$, reducing the NTF to $\frac{s^N}{s^N}$, and, thus, it acts as a HPF. Since the STF has N poles and the NTF has N poles and N coincident zeros, as the order of the ΣΔM increases, the number of poles of the STF and the number of poles and zeros of the NTF also increase, increasing the attenuation outside of the passband.

In practice, higher-order ΣΔMs are more effective to filter the input signal higher frequencies and quantization noise at lower frequencies. The effect of the N^{th} order ΣΔM on the noise (Noise Shaping (NS)) is the most notable one and it is depicted in Figure 3.12, as well as the distribution of quantization noise in a Nyquist sampling ADC and a simple oversampling ADC.

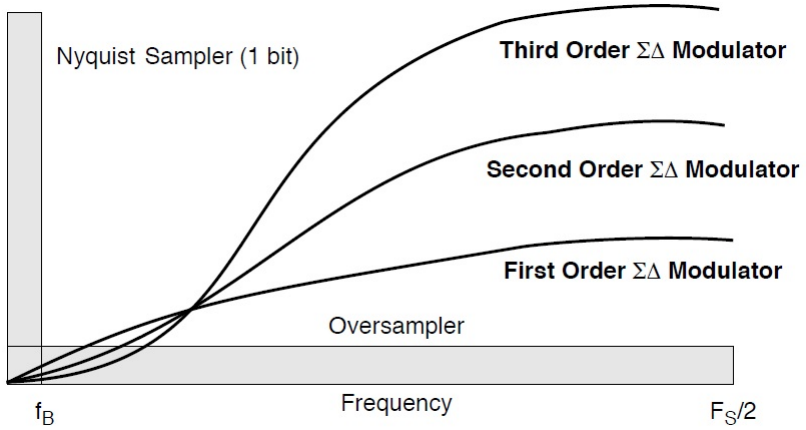


Figure 3.12: Noise Shaping for different orders ΣΔM, a Nyquist sampling ADC and a simple oversampling ADC[16].

In conclusion, the higher the order of the ΣΔM, the more effective it is at pushing the quantization noise towards high frequencies, at the expense of hardware simplicity. However, stability problems might also occur, due to the increasing number of poles of both STF and NTF.

3.2 Design of the Modulator

Given the desired portability of the overall system, hardware simplicity is a main goal of the project. For that reason a FPGA implemented ΣΔM must be as simple as possible in terms of hardware external to the FPGA.

That said, the implemented ΣΔM was a first-order one, having in mind that the benefits are not worth the added complexity brought by a higher-order ΣΔM.

Initially, it was planned to implement the topology depicted in Figure 3.8. However, given the need of an analog integrator in the feedback loop, so was needed an analog adder block, that should be implemented with an Operational Amplifier (OPAMP) in a difference configuration with unity gain. Thus, the topology depicted in Figure 3.7 was chosen, implementing the adder block with a differential input buffer, programmed in the FPGA. Although the number of integrators doubled (a total of two resistors and two capacitors), the need for a difference amplifier was eliminated and, so, the need for one OPAMP and four resistors. This way, a simplified and equally functional ΣΔM was designed, with a minimum number of electronic components outside of the FPGA.

After the selection of the topology, a simulation of the system followed in MATLAB's Simulink environment.

The integrators were simulated according to an RC transfer function:

$$H_{RC}(s) = \frac{1}{1+sRC} = \frac{1}{1+s\tau} \tag{3.14}$$

The comparator implemented on the simulation was composed of a Zero-Order Hold (ZOH), with the intention to hold an input value for a duration equal to the sampling period, and a relay that would read the output of the ZOH and round it to its nearest digital value. Therefore, these two blocks cascaded act as a 1-bit quantizer operating at a fixed sampling frequency. Figure 3.13 depicts the Simulink block diagram of the simulated system, where the red components correspond to the digital part of the modulator and the black ones to the analog part.

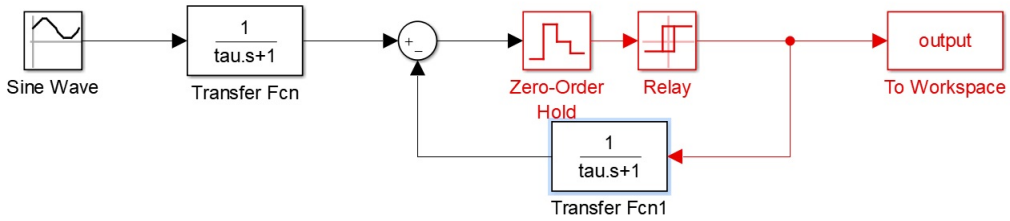


Figure 3.13: Simulink topology of the designed ΣΔM.

Due to the fact that the differential input buffer on the Artix-7 FPGA does not support 3.3V Low Voltage Differential Signalling (LVDS) I/O standard, the system was designed for a voltage range from 0V to 2.5V. As such, the sine wave at the ΣΔM's input has an amplitude of 1.25V and DC component of equal value. In the simulation, the input signal has frequency of 40kHz, given the fact that the used ultrasound transducers operate at this central frequency.

Having in mind this central frequency, the RC integrators, which also act as LPF, were designed for a cut-off frequency slightly above this frequency (approximately 48.2Hz, that is a time constant $\tau = RC = 3.3\mu s$).

The simulation was run for different sampling frequencies and the performance indicators (SNR,SINAD, Spurious Free Dynamic Range (SFDR) and THD) where computed, while a 40kHz sine wave was at the input of the ΣΔM. The results are shown on Table 3.1

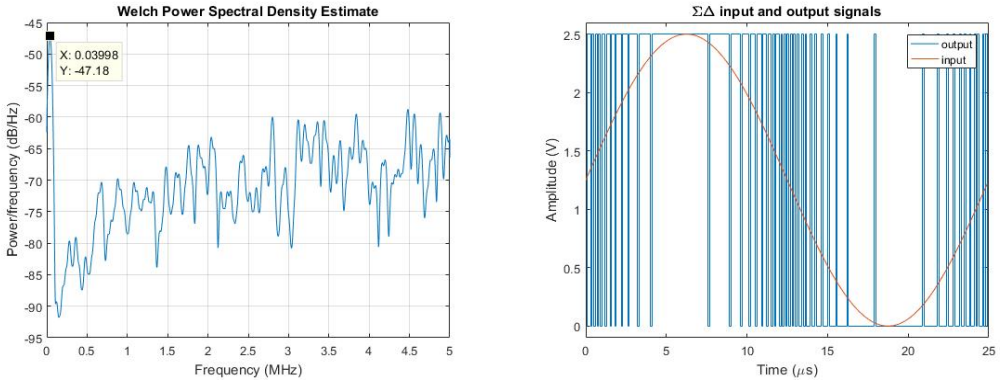
The irregularity of the SNR results is due to the fact that some sampling frequencies (like 12.5MHz) introduced a great amount of spurious in the noise band, raising the total noise power and degrading SNR.

f_s (MHz)	SNR (dB)	SINAD (dB)	SFDR (dB)	THD (dB)
10	103.4	43.0	43.1	-43.0
12.5	48.2	47.4	52.7	-55.1
20	107.0	64.2	64.2	-64.2
25	51.9	49.3	52.9	-52.7
50	90.4	68.5	69.1	-68.6
100	88.6	71.6	71.9	-71.9

Table 3.1: Performance indicators computed in the simulation for different sampling frequencies, with input frequency of 40 kHz.

Given the results, 50MHz of sampling frequency were chosen because of the greater SNR measured, in spite of the slightly lower remaining performance indicators.

Figure 3.14 shows the input and output signals, for a sampling frequency of 10MHz (chosen so a better visualisation of the output signal could be obtained) and an input frequency of 40kHz, as well as the NS characteristic.



(a) Noise shaping.

(b) Input and Output signals.

Figure 3.14: Bitstream modulation and noise shaping characteristic for a 40 kHz input sine wave and 10 MHz sampling frequency.

Finally, with the sampling frequency set, the simulation was run for different input signal frequencies. The results are shown in Figure 3.15.

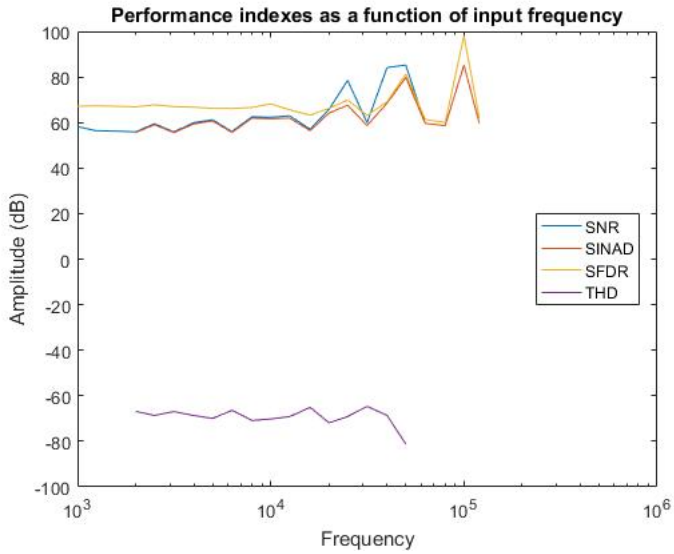


Figure 3.15: Simulated performance indicators for different input frequencies.

3.3 Implementation of the ΣΔM

The design proposal for the ΣΔM is depicted in Figure 3.16. Inside the blue box lay the digital components, implemented in VHDL and outside are the analog components that make up the integrators.

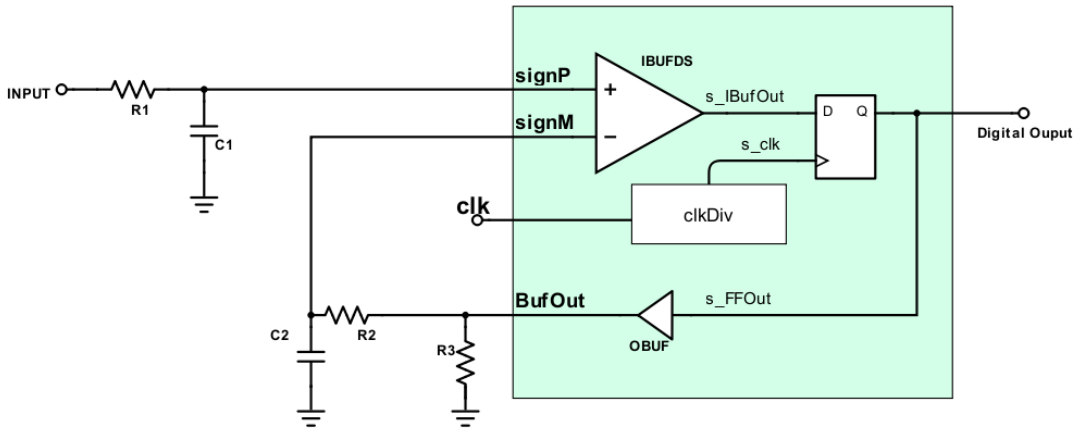


Figure 3.16: Designed ΣΔM.

3.3.1 Digital Interface

The digital interface of the ΣΔM was implemented in a VHSIC Hardware Description Language (VHDL) programmed FPGA and Figure 3.16 depicts its block diagram inside the blue box, with inputs *signP*, *signM* and *clk*, a single output *BufOut* (bold letters) and internal signals *s_IBufOut*, *s_clk* and *s_FFOut*.

The VHDL implementation of the digital interface can be found in Appendix C.

Input Differential Buffer

As stated previously, the adding block of the chosen ΣΔM topology was implemented as an input differential buffer (IBUFDS). This way, the output of the buffer would be the nearest digital value to the difference between its inputs. A VHDL primitive, part of UNISIM library, was available and used to implement this part of the circuit in a simple way, on the top level file. The differential termination was chosen to be off, once the frequencies at both inputs are expected to be relatively low (around 40kHz), not causing the reflection phenomenon expected when high-frequency signals reach the buffer inputs. The low power attribute was set to FALSE in order to improve the buffer's performance and the IOSTANDARD was defined to be LVDS_25, as $V_{CCO} = 3.3V$ powered LVDS signaling is not supported in Xilinx's 7 series FPGA families [21], as it is the case with the used Artix-7 (due to this fact, the remaining inputs and outputs were set to a LVCMOS25 IOSTANDARD).

Sampling

However, the output of the differential buffer is in no way synchronised with any clock signal. In order to perform this, a D-type flip-flop was cascaded with the differential buffer. A D-type flip flop sets its output as the input digital value at very specific time instants (in this case, at its clock signal rising edge), remaining unaltered at other instants, independently of the input value. Both the input differential buffer and the D-type flip-flop combined perform the 1-bit quantization at sampling frequency $f_s = 50MHz$. The output of the D flip-flop is to be fed-back to the negative input of the differential buffer, right after being integrated.

The clock signal controlling the sampling moments, of frequency 50MHz, corresponds to half the frequency of the internal clock of the FPGA used (division factor of 2). To implement it, a frequency divider was described. The principles behind its operation is that, dividing the input clock frequency by a division factor of N, the output signal is off during $\frac{N}{2}$ input periods and on during the remaining. Provided that N is even, the output signal is a Pulse-Width Modulation (PWM) signal with exactly 50% duty-cycle. Otherwise, it is on for $\frac{N}{2} - 1$ periods and off for $\frac{N}{2}$ input clock periods.

Output buffer

The remaining VHDL component is an output buffer that allows the output signal of the ΣΔM to be fed-back, through an analog circuit that is external to the FPGA. This output buffer is also a primitive that is part of the UNISIM library and, as with the input differential buffer, it was programmed in the top-level file. The drive attribute was set to 16, the IO/standard was set to LVCMOS25 to maintain compatibility between the pins used and the SLEW attribute, which sets the slew rate of the output buffer, was set to HIGH, so that it could get a faster response and thus improving signal quality throughout the feedback loop.

3.3.2 Analog interface

Given the fact that the input differential buffer was set to have the LVDS_25 IOSTANDARD, the output of the digital interface should have a digital high level of 2.5V. To achieve this objective, experimentally, the output impedance of the *BufOut* output was measured at

DC to be 220Ω. Once the output range would vary from 0V to 3.3V, given this measured output impedance, a resistor was connected with one terminal at the output and the other at the ground. This resistor is depicted in Figure 3.16 with the reference R3. The purpose of this resistor was to act alongside with the measured output impedance as a voltage divider and, thus, lower the output voltage when on from 3.3V to 2.5V. As such, the value of this resistor is given by the voltage divider expression:

$$2.5 = \frac{R_3}{R_3 + 220} \times 3.3$$

Solving for R_3, its value is then computed:

$$R_3 = \frac{220 \times 2.5}{3.3 - 2.5} = 687.5\Omega$$

And so the connected resistor was one of 680Ω.

The integrators were chosen to be simple RC circuits. Let the circuit in Figure 3.17 be considered.

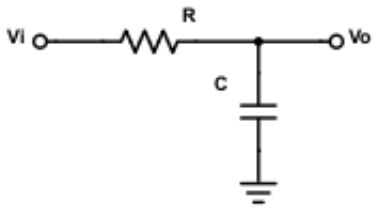


Figure 3.17: RC integrator and low-pass filter.

The output voltage is:

$$v_o(t) = \frac{1}{C}\int i(t)dt = \frac{1}{C}\int \frac{v_i(t) - v_o(t)}{R}dt$$

For high input frequencies, as the circuit acts as a low-pass filter, the output voltage is negligible when compared to the input voltage, and so:

$$v_o(t) \approx \frac{1}{C}\int \frac{v_i(t)}{R}dt$$

Thus, the circuit behaves as a voltage integrator at high frequencies, which is the required operation for the feedback loop. At low frequencies ($s = j\omega \approx 0$), the output of the circuit is approximately the same as the input signal:

$$V_o(s) = \frac{Z_C}{R + Z_C}V_i(s) = \frac{\frac{1}{sC}}{R + \frac{1}{sC}}V_i(s) = \frac{1}{1 + sRC}V_i(s) \approx V_i(s)$$

This is the desired behaviour because it is intended to subtract the input signal to its approximation and later quantize it. This is a characteristic from a DM and, as it was

previously stated, a ΣΔM is a variation of the first, so some characteristics are mutual to both.

Being so, the needed cut-off frequency of both integrators should be slightly above the intended centre frequency (40kHz). A 10kΩ resistor and a 330pF capacitor were chosen for both integrators (R_1 and C_1 and R_2 and C_2 in Figure 3.16), making the cut-off frequency approximately 48.2kHz.

It was considered that the inputs of the differential buffer had infinite input impedance and, thus, have no effect at the output of both integrators connected to its terminals.

3.4 Tests and Results

In order to measure the performance indicators of the ΣΔM, a sinusoidal signal ranging from 0V to 2.5V was connected to the modulator, and its output *bitstream* recorded in the FPGA memory and then analysed (this process is explored with more detail in Appendix A).

3.4.1 Experimental Set-up

The input sine wave is required to be as clean as possible, in order to better analyse the noise and interference introduced by the modulator, which should not be "contaminated" with the input signal imperfections. To achieve this requirement, the Audio Precision Portable One Plus, an audio measurement instrument capable of low-noise and low-distortion sine wave generation was used and it is shown in Figure 3.18.

Figure 3.18: Audio Precision Portable One Plus audio test set.

The equipment can only generate zero DC offset wave forms. Being so, a signal conditioning circuit was needed to add a DC offset so a signal with a required voltage range at the input of the modulator could be attained. The chosen configuration was of a capacitor followed by a voltage divider fed with the FPGA's 3.3V V_{ss}. This way, the output signal should be the input signal with an offset of equal value of the voltage divider DC output, and its maximum value must not exceed 3.3V. The circuit is represented in Figure 3.19.

The offset is determined in a DC analysis of the circuit. Therefore, the capacitor C acts as an open circuit or, in other words, an infinite impedance. The offset is, then, computed by:

$$V_{offset} = \frac{R_2}{R_1 + R_2} V_{ss}$$

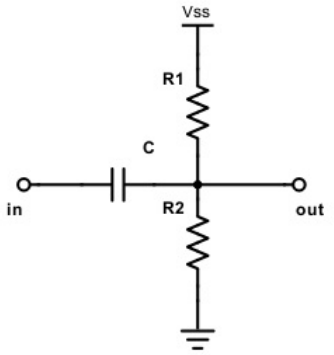


Figure 3.19: Signal conditioning circuit for offset addition.

Solving for R_1:

$$R_1 = \frac{V_{ss} - V_{offset}}{V_{offset}} R_2$$

Knowing that the required offset is halfway between the voltages for the digital values of '0' and '1':

$$V_{offset} = \frac{V^+ - V^-}{2} = \frac{2.5 - 0}{2} = 1.25V$$

And setting the value of R_2 at 10kΩ, the computed value of R_1 is, then:

$$R_1 = 16.4k\Omega$$

The effect of the capacitor C at low frequencies has to be taken into account, since it adds a HPF behaviour and depends on resistors R_1 and R_2, in parallel. Its cut-off frequency should be low and is given by the following expression:

$$f_c = \frac{1}{2\pi \left(R_1 /\!/ R_2\right) C}$$

It was chosen a 100nF capacitor, making its cut-off frequency $f_c = 256Hz$.

The resistors used on the circuit have a tolerance of 5% and a precise offset voltage is in practice unachievable. To work around the problem, the resistor R_1 was divided into one resistor of 15kΩ in series with a variable resistor of 2.2kΩ, as it is shown in Figure 3.20. This way, it is possible to calibrate the circuit to present an output offset of 1.25V, in spite of the deviations that might occur in the resistors.

Since the output of the conditioning signal represented in Figure 3.20 and designed until now is to be connected to the input of the ΣΔM depicted in Figure 3.16, one must take into account the input impedance of the modulator so that the offset does not deviate from its designed value.

At low frequencies, the input impedance of the ΣΔM is much greater than R_2 in Figure 3.19 due to the positive input of the differential buffer's impedance, so, as seen from the

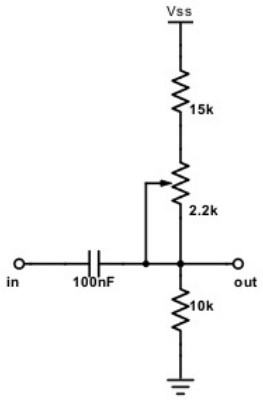


Figure 3.20: Calibratable offset signal conditioning circuit.

input of the signal conditioning circuit, there's a parallel between R_2 and an input impedance that is much greater and, thus, the equivalent is approximately R_2, so its effect on the offset voltage is negligible.

Finally, given the fact that the circuits described so far are to be integrated in a FPGA system, the voltage source of 3.3V might contain high frequency noise components. Given this fact, a RC LPF was designed with a 100Ω resistor and a 68μF capacitor, making its cut-off frequency $f_c = 23.4Hz$, to filter the DC voltage source. This way, the DC voltage source is now cleaner. It should be noted that the equivalent impedance of the circuits connected to the output of this filter is much greater than 100Ω (Figure 3.20), which is the resistor that is part of this filter, not having a damaging impact on the filtered voltage source level.

The schematics for the resulting analog circuit are depicted in Figure 3.21.

Concerning the reduction of noise added in the analog interface, it was designed in a Printed Circuit Board (PCB) with a ground plane. The PCB would be connected in a Peripheral Module (PMOD) input (as illustrated in Figure 3.22) on the Nexys-4 board for testing.

Referring to Figure 3.22, Pin 1 was programmed to be the positive terminal of the differential input buffer, Pin 7 to be the negative terminal, Pin 4 was programmed to be the output of the output buffer and the remaining Pins 2,3 and 9 to 10 where set to level '0' for minimising interferences and, thus, reduce the noise.

Taking into consideration the high frequency signal available at the output Pin 4, the PCB was designed so that the components were as close as possible and with vertically placed resistors, minimising the eventually added noise due to smaller wavelengths.

The PCB layout is depicted in Figure 3.23.

The capture of the bitstream for posterior analysis involved storing its bits in a Block Random-Access Memory (BRAM) that allowed to store 131072 words with a maximum size of 36 bits, each. To take advantage of this fact, a serial-to-parallel converter block was implemented in VHDL. The purpose of this block is to capture 36 bits of the output bitstream

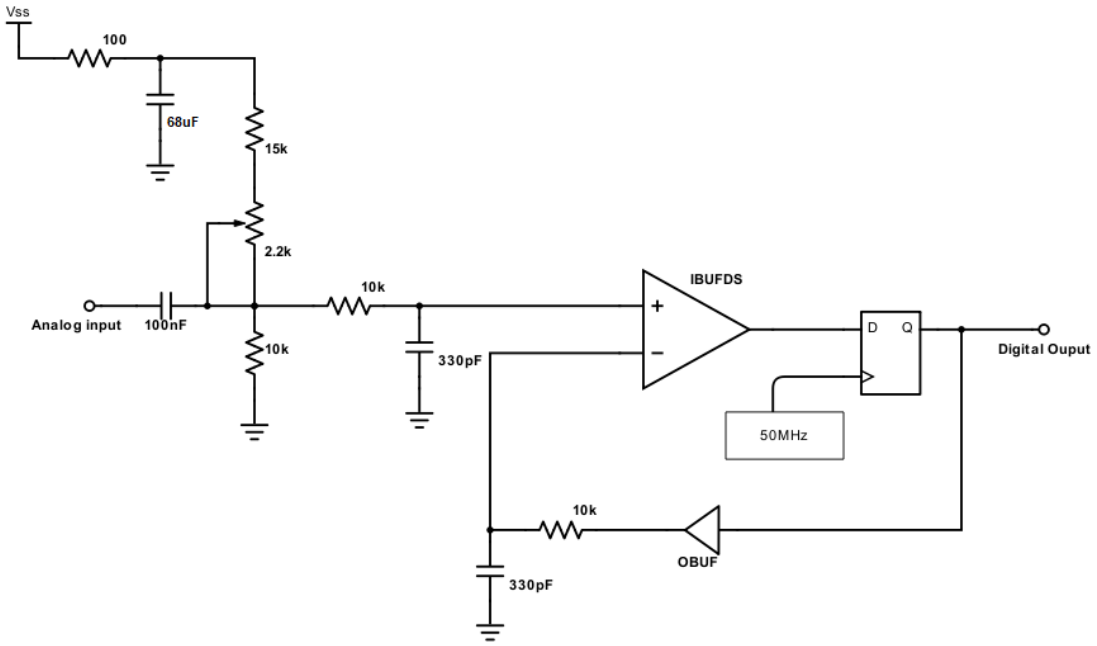


Figure 3.21: ΣΔM, with the calibratable signal conditioning circuit, DC supply LPF filter, integrators and digital interface.

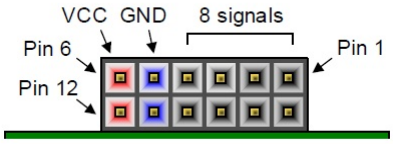


Figure 3.22: PMOD input ports front view on the Nexys 4 board [22].

and store them as a word in the BRAM. Its operation lies on the basis operation of a shift-register, where an input bit is shifted into a N-bit word. In fact, its 36 bit parallel output is changing at a rate of 50MHz, *i. e.*, an input bit is shifted and the output immediately reflects this change, concatenating the input bit with the 35 Most Significant Bits (MSB). However, a counter signal is incremented at the same frequency as the sampling one. When it reaches the number of intended bits to be stored it is reset and an additional output (a triggering signal) is put to a high level for a single cycle duration, triggering Vivado's Integrated Logic Analyser (ILA) to store a 36 bit word in the BRAM.

Refer to section C.3 for more details on the VHDL implementation of the described shift register.

Given the fact that the quatization happening in the ΣΔM is a stochastic process, the increase of the number of periods of the input signal recorded has benefits in terms of reliability in the measurement of performance indicators, such as SNR and THD, for example, because they are statistically determined.

The entire experimental set-up for the measurements of the ΣΔM performance can be

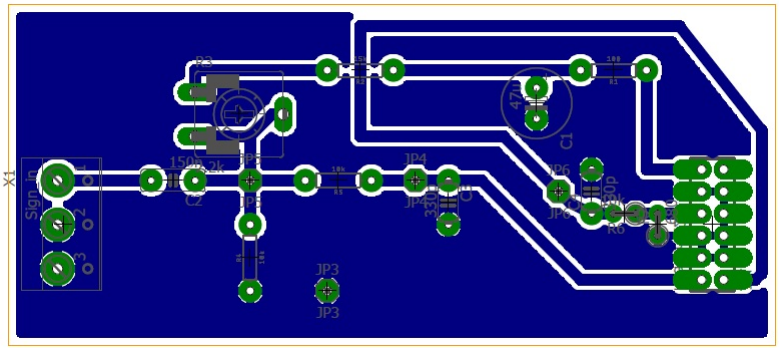


Figure 3.23: Board layout of the circuit used to take measurements to the ΣΔM.

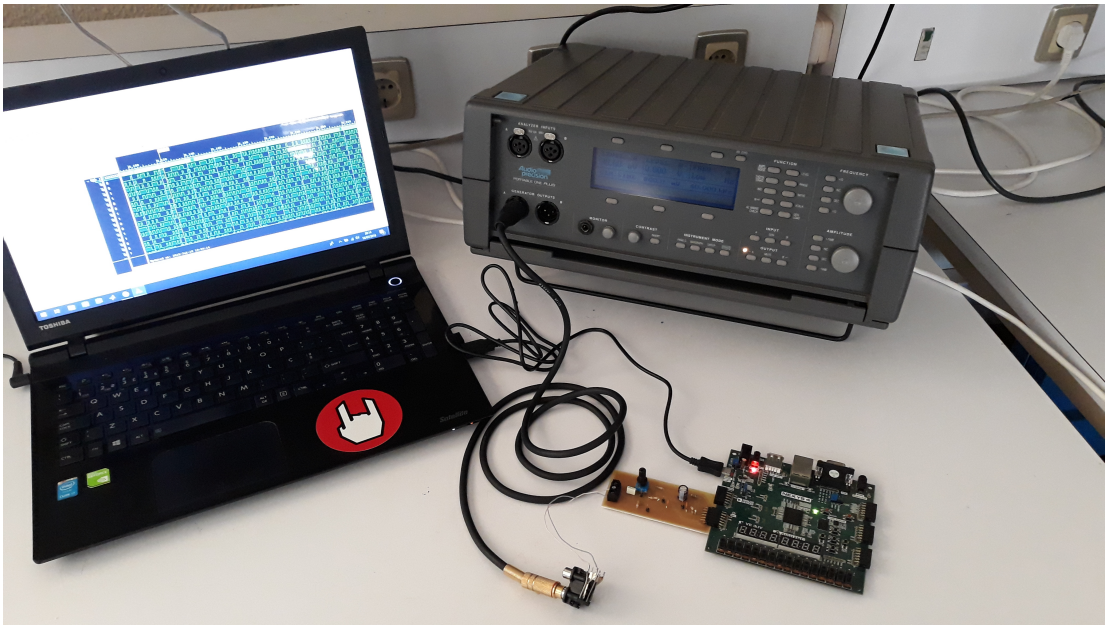


Figure 3.24: Experimental set-up for measuring the ΣΔM performance.

seen depicted in Figure 3.24.

3.4.2 Results

The Audio Precision equipment was set to generate a fixed frequency sine wave, that was to be put at the input of the ΣΔM depicted in Figure 3.21. Then, the respective generated bitstream was recorded with Vivado's ILA and later processed with MATLAB for PSD estimation and computation of the performance indicators (SNR, SINAD, SFDR and THD). This process is further explored in Appendix A.

The process was repeated for several input frequencies and the generated bitstreams were stored and processed altogether.

Output Power Spectrum Density

The estimated PSD for both the simulated and implemented ΣΔMs, when the input signal has a frequency of 40kHz is depicted in Figure 3.25.

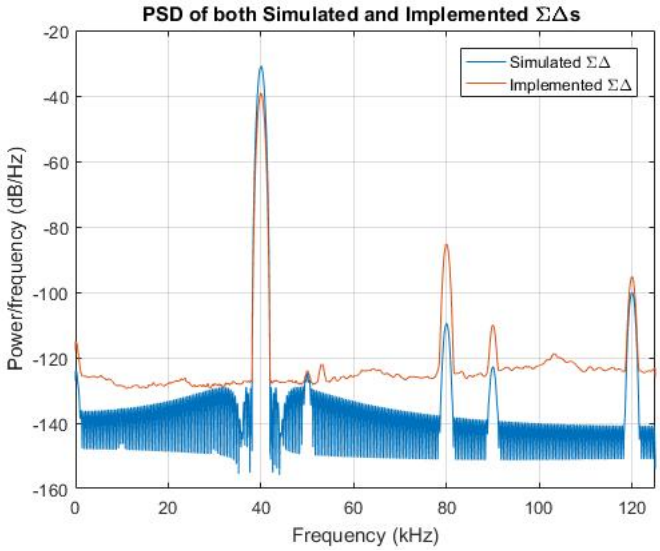


Figure 3.25: Estimated PSD for both the hardware implemented and simulated ΣΔMs, when $f_{in} = 40kHz$.

Performance Indicators

The measured performance indicators as a function of the input frequency, for both the simulated and Harware (HW) implemented ΣΔMs, are depicted in Figure 3.26.

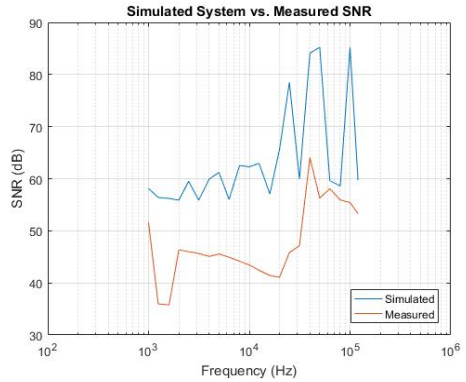
3.5 Results Analysis and Discussion

The simulated ΣΔM, pictured in Figure 3.13, turned out to exhibit an unpredictable behaviour that added vast amounts of spurious interferences outside the signal and distortion bands for some input frequencies. This phenomenon can be verified in Figure 3.27.

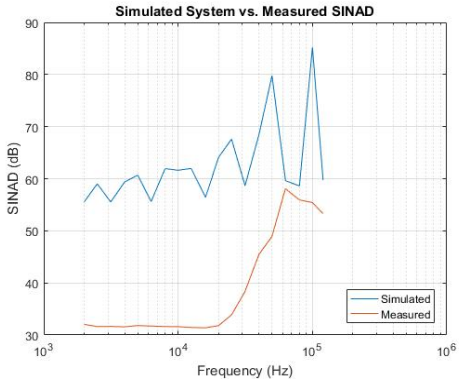
Since this phenomenon adds power to the noise band and, thus, degrades SNR, SINAD and SFDR, it justifies the somewhat unstable variation of the performance indicators for the simulated modulator. However, it can be stated that the figures of merit for both cases increase quality with input frequency.

At lower frequencies and due to the achieved frequency resolution of the PSD, the fundamental lobe was wide, encompassing within it lobe some harmonics. Due to this fact, the SNR and SFDR presented good results at low input frequencies. Thus, they should not be considered in the analysis.

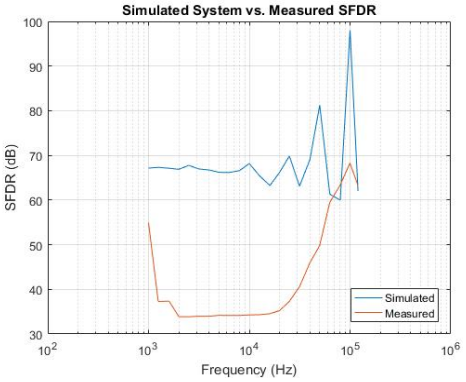
The THD decreases with frequency partially because, given the fact that the PSD is estimated in a band of 125kHz of width, as the input frequency increases, less harmonics fit inside this band, decreasing the THD. The last measured result showing in its respective bode plot is that of a 50kHz input, whose first harmonic stands at 100kHz and, so, inside the



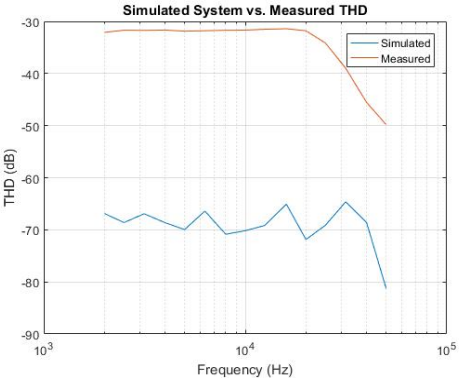
(a) SNR.



(b) SINAD.



(c) SFDR.



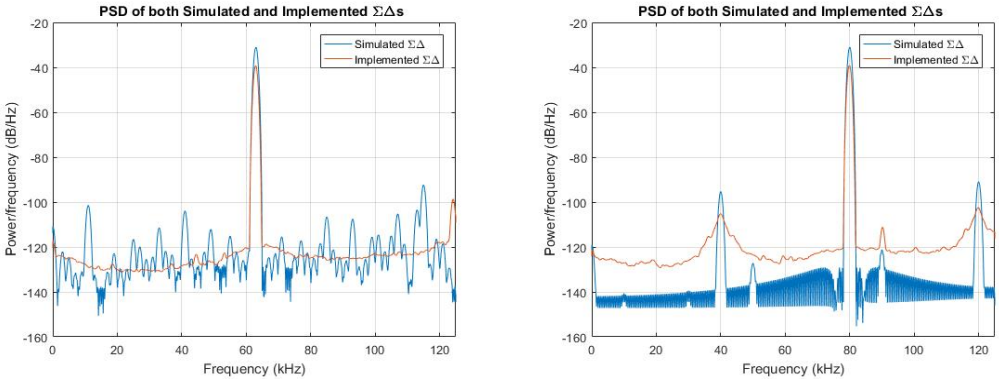
(d) THD.

Figure 3.26: Measured performance indexes for both hardware implemented and simulated ΣΔMs.

band of analysis. The next input frequency measured was of 63kHz and its first harmonic stood outside the 125kHz bandwidth. According to Equation 3.7, the THD for non existing harmonics power is then infinite.

The main reason for the difference in the values of performance indicators between the simulated and the hardware implemented ΣΔMs lies in the noise floor. In the simulated ΣΔM, in general, the recorded noise floor was lower than in the one implemented in HW. Being so, the noise power was lower and, as a consequence, both SNR and SINAD were, generally, higher in the simulated case. Also, generally speaking, the top signal power in the HW implemented ΣΔM was slightly lower and its harmonics presented greater harmonics power, as it is visible in Figure 3.25, increasing the THD when compared to the simulated modulator. Only at 63kHz did the simulated ΣΔM present a worse performance indicator and it was SFDR, as seen in Figure 3.26. Again, this took place due to the high concentration of spurious power in the noise band.

In Table 3.2 is shown the different figures of merit when the input signal is a 40 kHz sinusoid.



(a) $f_{in} = 63kHz$. (b) $f_{in} = 80kHz$.

Figure 3.27: Unpredictable behaviour of the simulated ΣΔM outside the signal and harmonics band for some input frequencies.

SNR	SINAD	SFDR	THD
64.06 dB	45.49 dB	45.97 dB	-45.55 dB

Table 3.2: Performance indicators measured with 50 MHz of sampling frequency and 40kHz of input frequency.

Finally, it must be noted that the values of SNR and SINAD, being dependent on the total noise power, depend also on the bandwidth of the analysis, *i. e.*, if the bandwidth of analysis were to be decreased, assuming frequency resolution remains unchanged, the total noise power would also decrease and so SNR and SINAD would benefit from it and come closer in value. Hereupon, the ENOB, which is computed by Equation 3.8, can be computed using SNR instead of SINAD.

The ENOB is the the resolution that can be achieved by an ideal ADC, *i. e.*, one that adds no distortion ([23]). Substituting the values of the expression, it yields:

$$ENOB = \frac{SNR - 1.76}{6.02} = \frac{64.06 - 1.76}{6.02} = 10.35\,bits$$

Which, taking into consideration the 2.5V dynamic range of the ΣΔM, its accuracy allows to measure precisely:

$$\Delta V_{in} = \frac{2.5V}{2^{ENOB}} = 1.92\,mV$$

A simple ADC was achieved with a performance similar to the one observed in some commercial 12-bit ADCs, like Microchip's MCP3201, for example, whose ENOB at an input frequency of 40kHz and 5V reference voltage was of about 10.5 bits [24].

Chapter 4

Beamforming

4.1 Introduction

Beamforming is a signal processing technique in which several sensors organised as an array collect and process several spatial samples of a propagating wave [25] in order to localise or receive a message from a source or even characterise the medium through which the wave propagates [26].

The most common forms of sensor arrays are the Uniform Linear Array (ULA) and the Uniform Planar Array (UPA), and they are to be described in the following sections.

4.1.1 Uniform Linear Array

In a ULA configuration, the sensors are found distributed along a line and equally spaced from each other, as it is depicted in Figure 4.1, where a planar wave-front that is about to interact with the array at an angle Θ, also called DoA, is also represented. The wave-front is considered linear as a mean to simplify the following calculations.

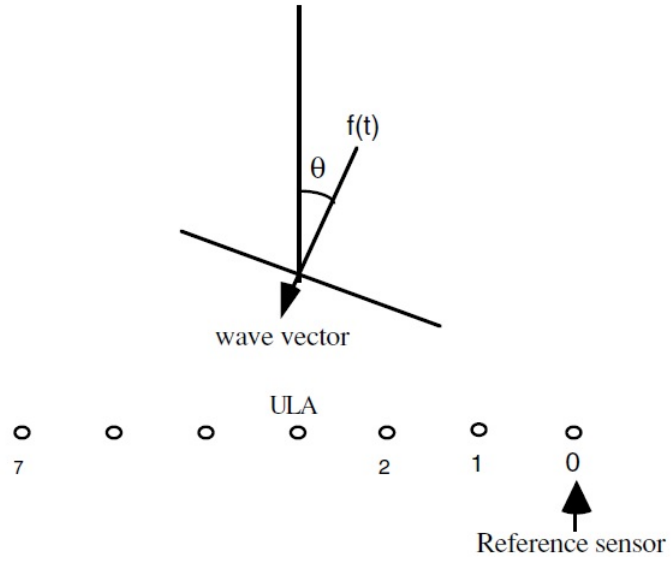


Figure 4.1: Uniform Linear Array configuration [26].

The DoA is defined as the angle that the direction of propagation of the wave that is interacting with the array makes with respect to its normal and it is depicted in Figure 4.1 with the greek letter Θ.

In this specific situation, the reference sensor is the rightmost one, given the fact that the wave-front will hit it first. The remaining sensors will interact with the wave following time delays dependent on the distance between sensors and the DoA.

In the general case where there are L sensors (elements), the wave-front will reach every l^{th} sensor with a delay τ_l, defined by:

$$\tau_l = \frac{d(l-1)}{v}\sin(\Theta), \quad 1 \leq l \leq L, \quad \Theta \in [-\pi; \pi] \tag{4.1}$$

Being v the velocity of the wave propagation.

Considering the wave-front some signal $s(t)$ modulated in frequency with a much greater carrier frequency f_c, making it a narrowband signal, the received signal at every l^{th} sensor is given by:

$$y_l(t) = s(t)e^{-j\omega_c(t-\tau_l)} \tag{4.2}$$

The derived expression states that the delay that each sensor presents induces a phase-shift since, according to Euler's formula:

$$e^{-j\omega_c(t-\tau)} = \cos(\omega_c \times t - \omega_c \times \tau_l) + j\sin(\omega_c \times t - \omega_c \times \tau_l)$$

Therefore, the analysis of the time delay in a sensor part of an ULA is, essentially, the same as the analysis of the phase-shift, when in respect to a reference sensor.

Combining all of the sensors of the array, it is obtained:

$$y(t) = \sum_{l=1}^{L} s(t)e^{-j\omega_c(t-\tau_l)} \tag{4.3}$$

In its essence, this kind of array performs spatial filtering. Being so, it is important to limit its parameters in a way that spatial aliasing can be avoided. Such condition is [27]:

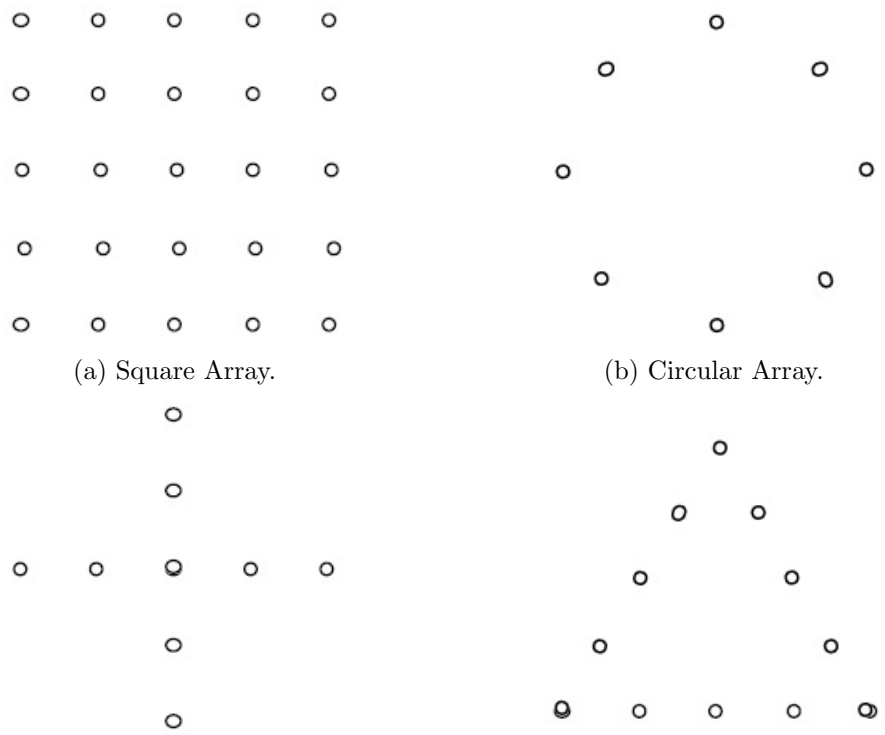
$$d < \frac{\lambda}{2} \tag{4.4}$$

4.1.2 Planar Array

A planar array has its sensors dispersed over a two-dimensional plane [26] and, thus, several geometries are possible, as depicted in Figure 4.2. Its main advantage over the ULA lies in the fact that it can measure two different incidence angles: *azimuth* and *elevation*, and it is depicted in Figure 4.3.

Referring to Figure 4.3, let a Planar Array have $M \times N$ sensors, distributed uniformly on the XY plane. Let m be the index that coordinates each sensor along the x-axis ($1 \leq m \leq M$) and n along the y-axis ($1 \leq n \leq N$), so each sensor can be referred to with the ordered pair (m, n). The delay with respect to the reference sensor along the x-axis is defined as:

$$\tau_m = \frac{d_1(m-1)}{v}\sin\theta\cos\varphi$$



(a) Square Array. (b) Circular Array.

(c) Cross Array. (d) Triangular Array.

Figure 4.2: Some different geometries for the Planar Array [26].

The delay with respect to the reference sensor along the y-axis is, then:

$$\tau_n = \frac{d_2(n-1)}{v} \sin\Theta \sin\varphi$$

The delay at each m^{th},n^{th} sensor is made up of both components and so:

$$\tau_{m,n} = \tau_m + \tau_n = \frac{d_1(m-1)}{v} \sin\theta \cos\varphi + \frac{d_2(n-1)}{v} \sin\Theta \sin\varphi \tag{4.5}$$

Following the same deduction pattern as in the case of the ULA, being some baseband signal $s(t)$ modulated with some carrier frequency $f_c(t)$ much greater than its bandwidth, the signal arriving at each m^{th},n^{th} sensor is:

$$y_{m,n}(t) = s(t)e^{-j\omega_c(t-\tau_{m,n})} \tag{4.6}$$

The resulting signal is the sum of the signal captured along both the x-axis and the y-axis:

$$y(t) = \sum_{m=1}^{M} s(t)e^{-j\omega_c(t-\tau_m)} + \sum_{n=1}^{N} s(t)e^{-j\omega_c(t-\tau_n)}$$

Which can be stated as:

$$y(t) = \sum_{n=1}^{N} \left[\sum_{m=1}^{M} s(t)e^{-j\omega_c(t-\tau_m)} \right] e^{-j\omega_c(t-\tau_n)} \tag{4.7}$$

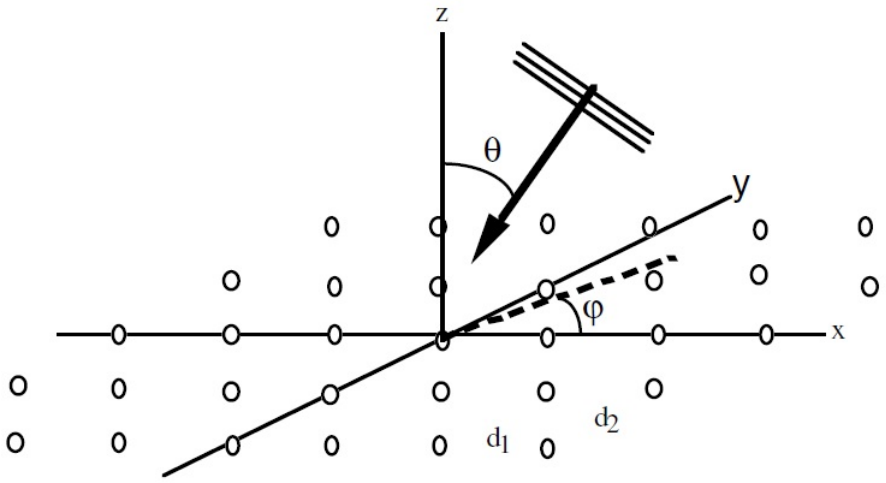


Figure 4.3: Azimuth (φ) and Elevation (Θ) Estimation in a Planar Array [26].

Referring to Equation 4.3, Equation 4.7 states that a planar array is mathematically defined as a sum of ULAs [2].

4.2 Analog Reception, Digital Down Conversion and Processing

4.2.1 Analog Reception

Depending on the emitter's signal amplitude and the distance of the echo travels, the electric signal at the receptor may have a small amplitude, at a range of few mV. Given this fact, prior to its conversion to the digital domain, it must be amplified.

Given the characteristics of the digital signal at the output of the $\Sigma\Delta$M, depicted in Figure 3.25, and given the fact that the expected input frequencies are located in the band 39-41 kHz, it is safe to say that the referred PSD represents a bandpass signal, centred at 40 kHz, which is quite narrow when compared to the analysed band of 0-125kHz. Being so, treating the signal in baseband, *i. e.*, centred at 0 Hz should facilitate its processing. Such is done through coherent detection, where the carrier phase information is extracted from the rest of the information received (in this case, phase differences).

4.2.2 Coherent Detection/Downconversion

Coherent demodulation is based on the assumption that some baseband signal carrying information is modulated with some other carrier frequency, *i.e.*, multiplied, shifting its frequency spectrum forward to be centred at the said carrier frequency, f_c. Recovering the baseband frequency components from the modulated signal is done with a mixer, as it is illustrated in Figure 4.4.

Let $x(t)$ be the input signal, a sine wave of frequency f_0, modulating a carrier with a frequency f_c (shifted by the same amount in frequency):

$$x(t) = \cos\left[2\pi(f_0 - f_c)t\right]$$

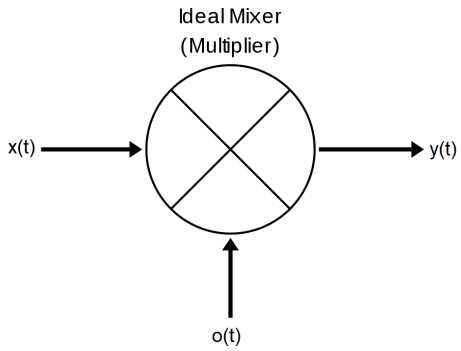


Figure 4.4: Block diagram of a mixer [28].

Let $o(t)$ be the signal generated by the local oscillator and a sine wave of frequency f_c:

$$o(t) = \cos(2\pi f_c t)$$

The mixer will multiply both signals and, thus, the output signal, $y(t)$ is, then, defined as:

$$y(t) = x(t) \times o(t) = \cos[2\pi(f_o - f_c)t]\cos(2\pi f_c t)$$

Bearing in mind the following trigonometric product identity:

$$\cos\alpha\cos\beta = \frac{\cos(\alpha-\beta)+\cos(\alpha+\beta)}{2} = \frac{\cos(\alpha-\beta)}{2} + \frac{\cos(\alpha+\beta)}{2}$$

The output of the mixer can then be stated as:

$$y(t) = \frac{\cos[2\pi(f_0 - f_c - f_c)t]}{2} + \frac{\cos[2\pi(f_0 - f_c + f_c)t]}{2} = \frac{\cos(2\pi f_0 t)}{2} + \frac{\cos[2\pi(f_0 - 2f_c t)]}{2}$$

The result indicates that the output of the ideal mixer comprises the sum of the baseband signal with itself shifted in frequency by the double of the carrier frequency, both presenting half of their original amplitude.

Low-pass filtering $y(t)$ eliminates its high-frequency components and effectively recovers the bandpass signal.

In essence, the mixer is able to shift both left and right a limited bandpass signal centred in some carrier frequency, f_c, by the same amount, originating two different bandpass signals, one centred in 0 Hz, called baseband, and the other centred in $2f_c$ Hz.

4.2.3 Quadrature Amplitude Modulated Signals

Any bandpass signal, $x(t)$ can be represented by an infinite sum of complex functions [29], as it is defined in Fourier series:

$$x(t) = \sum_{n=-\infty}^{+\infty} c_n e^{jnw_0 t}$$

Being $x(t)$ a real waveform, the negative indexes are complex conjugate of the respective positive ones, *i. e.*, $c_{-n} = c_n*$ and so the previous expression can be stated as:

$$x(t) = Re\left\{c_0 + 2\sum_{n=1}^{+\infty} c_n e^{jnw_0 t}\right\} \tag{4.8}$$

As it was previously specified, in the case of a passband signal, whose frequency components lay inside a band away from the reference, the coefficient for which $n = 0$ is null. Considering as well the fact that the band is centred in some frequency f_c, Equation 4.8 can be written as:

$$x(t) = Re\left\{\left(2\sum_{n=1}^{+\infty} c_n e^{jn(w_0 - w_c)t}\right) e^{jw_c t}\right\} \tag{4.9}$$

Equation 4.9 states that a baseband signal is the product of a bandpass signal by its carrier frequency. Let $g(t)$ be the passband signal, centered in its centre frequency, ω_c:

$$g(t) = 2\sum_{n=1}^{+\infty} c_n e^{jn(w_0 - w_c)t}$$

And so Equation 4.9 can be rewriten as:

$$x(t) = Re\left\{g(t)e^{jw_c t}\right\}$$

Which, according to Euler's formula, is:

$$x(t) = Re\left\{g(t)\cos\omega_0 t + jg(t)\sin\omega_0 t\right\} \tag{4.10}$$

Equation 4.10 allows one to understand the way one real baseband signal, $x(t)$, can be represented with an *In-phase* component, $I(t) = g(t)\cos\omega_c t$, and a *Quadrature* component, $Q(t) = g(t)\sin\omega_c t$, both components dependent on its passband component, $g(t)$. Often is $x(t)$ represented as a phasor in a complex plane (hence these kinds of signals are also named *complex signals*), where $I(t)$ defines the real, *in-phase* coordinate and $Q(t)$ the imaginary, *quadrature* coordinate, as it is illustrated in Figure 4.5. This way is easy to verify that both the phase (Φ) and amplitude of the real signal $x(t)$ depend solely on the amplitude of its complex coordinates.

In essence, modulating a complex signal in its *In-phase* and *Quadrature* forms allows one to easily represent it as a function of its phase, Φ, and amplitude, A, being:

$$\Phi(t) = \tan^{-1}\frac{Q(t)}{I(t)} \qquad A(t) = \sqrt{Q(t)^2 + I(t)^2}$$

This process is called downconversion and such representation allows to easily compute the phase difference between transducers in an array and, thus, it becomes easier to compute the *azimuth* and *elevation* angles φ and Θ, respectively.

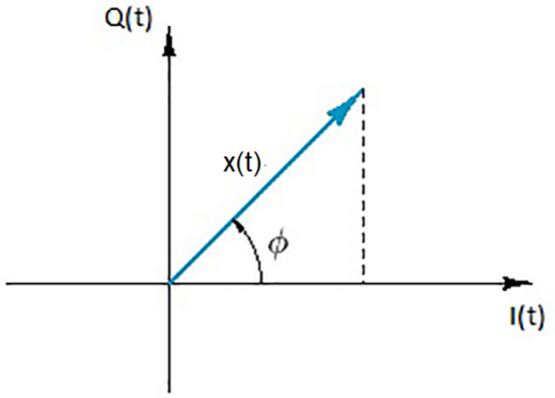


Figure 4.5: Representation of the passband in the form of a phasor dependent on its In-phase and Quadrature components.

4.2.4 Downconverter mixer

Extracting the *In-phase* and *Quadrature* components of a complex signal is based on mixing it with two sine waves of the same frequency (the carrier frequency, f_c), in which one is phase shifted of 90^o to obtain the quadrature component. The basic downconverter is depicted in Figure 4.6.

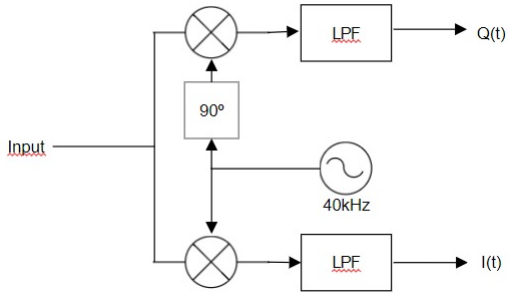


Figure 4.6: Basic downconverter.

As it was previously stated, the coherent detection shifts a bandpass signal to the left and to the right of the carrier frequency. Given that the only band of interest is situated at low frequencies, a LPF is used to eliminate the component of the signal that was shifted right.

Being it implemented in FPGA, some adjustments will have to be performed. In a first place, the output of the ΣΔM will be connected to the input of the downconverter. Its bitstream presents a bandwidth that is quite big when compared to the one of the signal of interest, and it includes the quantization noise from the ΣΔM. Therefore, downsampling followed by low-pass filtering is desirable before the downconversion process. Also, 1 bit samples are not desired at the downconverter, given the fact that they have an undesired level of resolution. A *Moving Average Filter* presents a solution to this problem, while downsampling and low-pass filtering the ΣΔM output signal.

Moving Average Filter

A moving average filter takes a fixed number of input samples (K), averages its values and outputs the result. Its output is mathematically defined as:

$$y[n] = \frac{1}{K} \sum_{k=0}^{K-1} x[n-k] \tag{4.11}$$

In the $\mathcal{Z}$ domain, it is:

$$Y(z) = \frac{1}{K} \sum_{k=0}^{K-1} X(z) \times z^{-K}$$

And so, its transfer function, defined in the $\mathcal{Z}$ domain is:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{K} \sum_{k=0}^{K-1} z^{-K} \tag{4.12}$$

The geometric series states the following:

$$\sum_{m=0}^{M-1} a \times z^m = a \frac{1 - z^M}{1 - z}$$

Equation 4.12 is a geometric series. Therefore, it can be mathematically described as:

$$H(z) = \frac{1}{K} \frac{1 - z^{-K}}{1 - z^{-1}} \tag{4.13}$$

That is:

$$Y(z) - Y(z)z^{-1} = \frac{X(z) - X(z)z^{-K}}{K} \tag{4.14}$$

The equivalent, in the discrete time domain is:

$$y[n] = \frac{x[n] - x[n-K]}{K} + y[n-1] \tag{4.15}$$

Equation 4.15 is the recursive implementation of the Moving Average Filter. When compared to Equation 4.11, it shows reduced number of additions [16]. Equation 4.13 can be factored into two separate processes (integration and differentiation):

$$Y(z) = \left[\frac{1}{1 - z^{-1}} \times \left(1 - z^{-K}\right) \right] \frac{X(z)}{K}$$

Since that it will have to be followed by a downsampling block (that will keep only every K^{th} sample), and taking into account the noble identities of decimation, it is possible to implement the filter as it is depicted in Figure 4.7 [15], reducing the delay time of the comb block and, thus, hardware complexity.

This configuration is called Cascaded Comb-Integrator (CIC) filter and it presents several advantages [16] where, among them, are:

- No filter coefficients and, thus, no memory is required to store them;

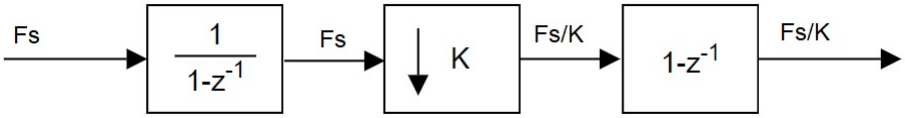


Figure 4.7: Block diagram of the One-Stage Recursive Moving Average Filtering Process.

- Hardware simplicity.

To analyse the filter's frequency response, refer to Equation 4.13. Considering that $z = e^{j\omega}$, it can be rewritten as:

$$H(e^{j\omega}) = \frac{1}{K}\frac{1-e^{-j\omega K}}{1-e^{-j\omega}} = \frac{1}{K}\frac{e^{-j\frac{\omega}{2}K}}{e^{-j\frac{\omega}{2}}}\frac{e^{j\frac{\omega}{2}K}-e^{-j\frac{\omega}{2}K}}{e^{j\frac{\omega}{2}}-e^{-j\frac{\omega}{2}}} \tag{4.16}$$

According to an Euler Identity, that states the following:

$$\frac{e^{j\Theta}-e^{-j\Theta}}{2j} = \sin\Theta \Leftrightarrow e^{j\Theta}-e^{-j\Theta} = 2j \times \sin\Theta$$

Equation 4.16 can be written as:

$$H(e^{j\omega}) = \frac{1}{K}e^{-j\frac{\omega}{2}(K-1)}\frac{\sin\left(\frac{\omega}{2}K\right)}{\sin\left(\frac{\omega}{2}\right)}, \quad 0 \leq \omega \leq 2\pi \tag{4.17}$$

Equation's 4.17 zeros depend solely on the cosine located at the numerator of the rightmost fraction. Being so:

$$\sin\left(\frac{\omega}{2}K\right) = 0 \Rightarrow \frac{\omega}{2}K = m\pi, \quad m = 0, \pm1, \pm2, \pm3, ...$$

Solving for ω results:

$$\omega = \frac{2m\pi}{K} \Leftrightarrow \frac{f}{f_s} = \frac{m}{K}$$

Solving for f, then, results in the frequencies for which the frequency response is zero:

$$f = m \times \frac{f_s}{K} \tag{4.18}$$

The derived expression allows one to understand that the zeros, besides repeating, get closer to the origin for greater amounts of K, as can be seen illustrated in Figure 4.8.

One alternative implementation is a N-stage one, where N CIC filters are cascaded. Equation 4.13, then, goes:

$$H(z)^N = \left[\frac{1}{K}\frac{1-z^{-K}}{1-z^{-1}}\right]^N = \frac{1}{K^N}\left[1-z^{-K}\right]^N\left[\frac{1}{1-z^{-1}}\right]^N$$

This is the same as cascading N integrators, downsampling as a factor of K, and then cascading N differentiators and, for greater N, the steeper the frequency response is, as illustrated in Figure 4.9.

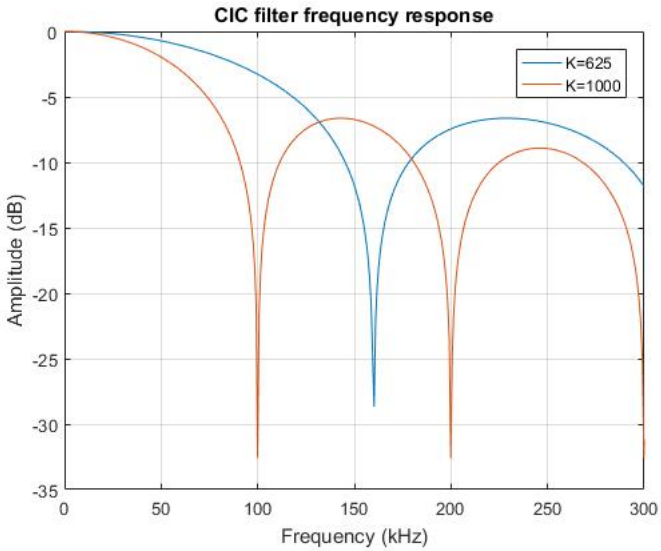


Figure 4.8: CIC Filter Frequency Response for two different values of K.

Oscillator

One problem that arises with the implementation of the downconverter in Figure 4.6 is the oscillator. Being it digital, the signal it outputs has to be stored in memory. The better the resolution of this signal, the memory required to store its samples becomes greater. One way to reduce this memory is to store only the samples equivalent to the maximum, minimum and zeros of this signal, for only one period, and repeating them over and over. Being so, its output should be:

$$osc(t) = ..., 0, 1, 0, -1, 0, 1, 0, -1, 0, 1, 0, -1, 0, ...$$

At a sampling frequency $f_s = 4 \times f_c = 160kHz$.

For this reason, the sampling frequency of the CIC filter has to be the same in order to allow synchronisation, *i. e.*, $160kHz$.

Final Downsampling

The output of the mixer should have frequency components below 1kHz. Having a sampling frequency rate of 160kHz, a decimation process is desired in order to reduce bandwidth and, thus, unnecessary higher-than-required frequency components, *i. e.*, noise.

Receiver Channel

Having into account what has been describer in the last sections, each receiver channel that is to be implemented is depicted in Figure 4.10.

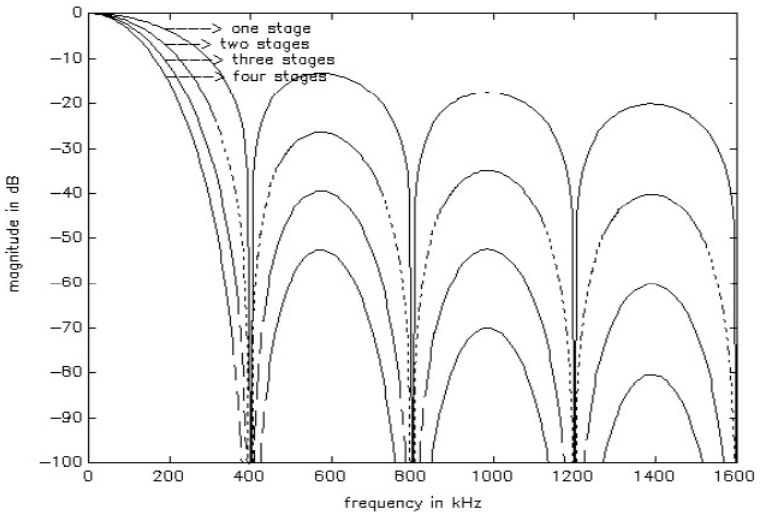


Figure 4.9: N-stage CIC Filter frequency response for different values of N [16].

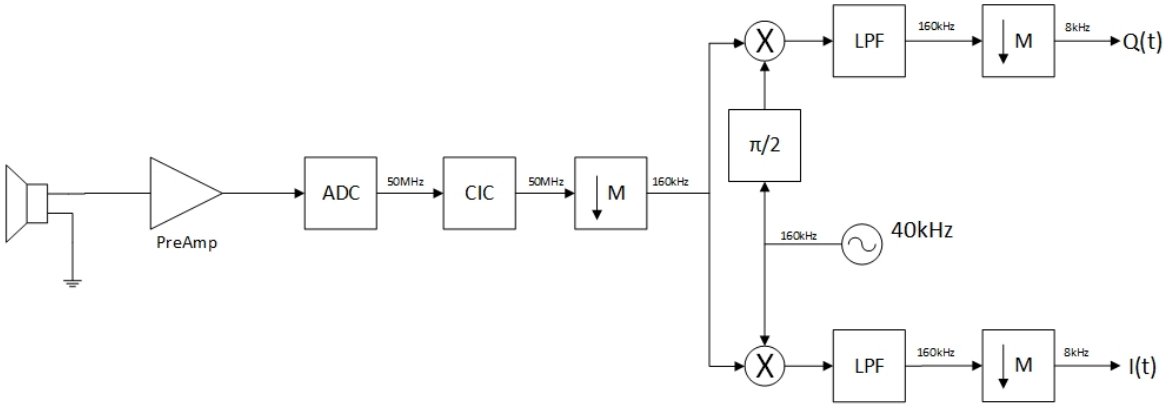


Figure 4.10: Theoretical version of each receiving channel of the Planar Array.

4.3 Implementation

4.3.1 Analog Reception

Each receiver depicted in Figure 4.10 will use a transducer MA40S4R by Murata and the designed Preamplifier that is going to amplify the voltage signal that the transducer receives is depicted in Figure 4.11.

The chosen OPAMP was the MCP6021 because, not only is it rail-to-rail, but it also presents a Gain-Bandwidth Product (GBDP) of 10Mhz. This means that, for a gain of 100, its bandwidth should be:

$$BW = \frac{GBWP}{G} = \frac{10MHz}{100} = 100kHz$$

Which, both Gain and Bandwidth, should be enough for the sought application.

The OPAMP is powered with the FPGA sources, *i. e.*, 3.3V and 0V and was chosen to implement the amplifier.

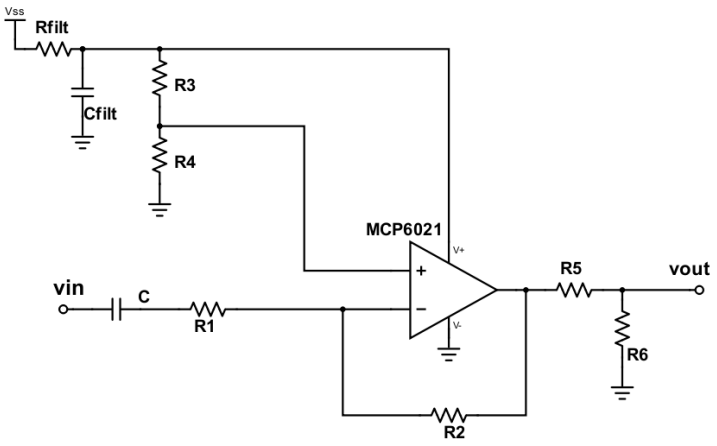


Figure 4.11: Designed Preamplifier for each reception channel.

Refering to Figure 4.11, the output of the OPAMP is computed according to the following expression:

$$V_{out} = -\frac{R_2}{R_1} + V_{REF} \tag{4.19}$$

Where V_{REF} is a DC voltage at the positive terminal of the OPAMP.

R_2 should be has high as possible, otherwise, at high frequencies, being the negative input terminal equivalent to a ground, this resistor will act as if it were in parallel with the load impedance and, thus, alter the amplifier's gain. Accordingly, and knowing that R_1 has to be 100 times lower (the selected gain is 100), the following resistors were selected:

$$R_1 = 10k\Omega \qquad R_2 = 1M\Omega$$

It is desirable to make use of the whole signal range at the output, that is, from 0 to 3.3V, and so the output DC component should be half of this range. Consequently, $V_{REF} = 1.65V$, and it is to be attained with resort to a voltage divider with two resistors of equal value (R_3 and R_4 in Figure 4.11):

$$R_3 = R_4 = 100k\Omega$$

Also, there is now a need to eliminate the DC component of the signal at the input of the amplifier, which is done by adding a capacitor in series between the transducer (v_{in}) and the amplifier. It acts, together with R_1, as a high-pass filter, and its cut-off frequency is:

$$f_c = \frac{1}{2\pi R_1 C}$$

Choosing $C = 47nF$, it results in $f_c \approx 340Hz$.

Once that, as seen on Chapter 3, the $\Sigma\Delta$M's input signal should range from 0V to 2.5V, a new voltage divider is, then, needed at the output of the OPAMP (R_5 and R_6 in Figure 4.11) and so:

$$V_o = \frac{R_6}{R_5 + R_6} V_i \Leftrightarrow R_5 = R_6 \frac{V_i - V_o}{V_o} = \frac{3.3 - 2.5}{2.5} \Leftrightarrow R_5 = 0.32 \times R_6$$

The input impedance of the ΣΔM at 40 kHz is:

$$z_{in} = \sqrt{(10k\Omega)^2 + \left(\frac{1}{2\pi \times 40kHz \times 330pF}\right)^2} = 15.7k\Omega$$

The resistors R_5 and R_6 have to be, at least, two orders of magnitude lower, so:

$$R_5 = 150\Omega \qquad R_6 = 470\Omega$$

At last, the voltage source from the FPGA that supplies the amplifier and the voltage divider might be contaminated with high-frequency noise. Therefore, it is recommended to filter it. A RC filter (with a small resistor, R_{filt}, so its voltage drop is low and a high capacitance, C_{filt}, so its cut-off frequency is as low as possible) is a simple solution and so it was chosen. The selected values were $R_{filt} = 10\Omega$ and $C_{filt} = 1\mu F$. Also, a capacitor could have been connected to the V_{REF} node of the voltage divider.

4.3.2 Digital Downconversion

The digital blocks that will be described in the following pages were implemented in VHDL and their coded implementation can be found in Appendix D.

CIC Filter

As previously stated, the main purpose of the CIC filter is to filter the output of the ΣΔM and reduce the sample rate on the downconverter.

Once it is required a sample rate of 160 kHz at the output of the moving average filter, and knowing the sample rate at its input is 50 MHz, the downsampling rate, K, is:

$$K = \frac{f_{sin}}{f_{sout}} = \frac{50MHz}{160kHz} = 312.5$$

A rational downsampling factor is unattainable and, thus, the filter will be designed in a way that it is going to sample the output of the ΣΔM with a rate twice as high. Each sample output by the modulator will be summed twice in order to compute the average and, so, the result will be the same.

The downsampling rate, then, is:

$$K = \frac{100MHz}{160kHz} = 625$$

According to Equation 4.18, the first null of the transfer function of this filter is:

$$f = \frac{m}{K} \times f_s = \frac{1}{625} \times 100MHz = 160kHz$$

Which is four times greater than the frequency of interest. The transfer function of the filter is represented in Figure 4.12, where it can be seen that a signal with a frequency of about 40 kHz suffers from an attenuation of just a little over 0.45dB.

Referring to Figure 4.7, where it is depicted that the input of the downsampler is the result of an integration, *i. e.*, at each sample of the output of the ΣΔM, the delayed output of the same integrator is added (delayed by one sampling period), and the output of the comb block that follows the downsampler is computed by subtracting to an output sample of the

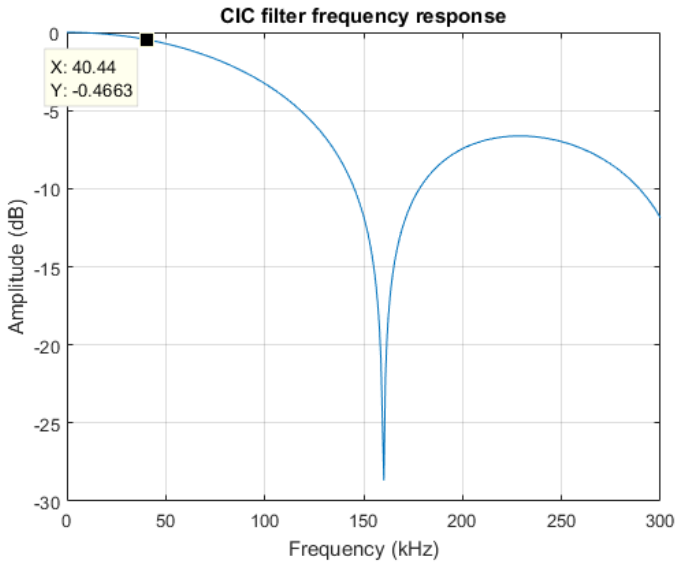


Figure 4.12: Computed Transfer Function of the Filter.

downsampler its previous output. The CIC filter can, then, be represented by the topology depicted in Figure 4.13, that is followed by a division.

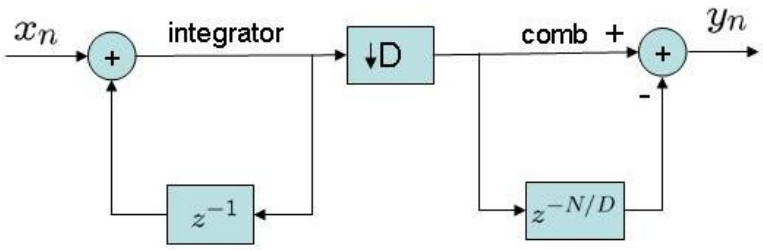


Figure 4.13: Efficient Implementation of a CIC Filter [30].

The resulting CIC filter was simulated in Simulink, where a sinusoid was sampled in a ΣΔM similar to the one simulated in Chapter 3, and its output was connected to the input of the filter. The simulated filter is depicted in Figure 4.14.

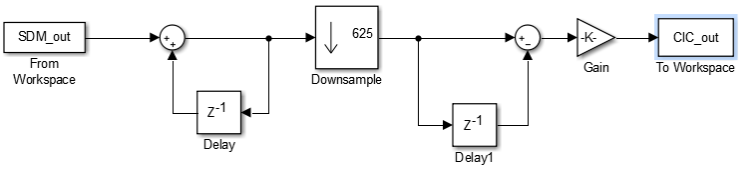
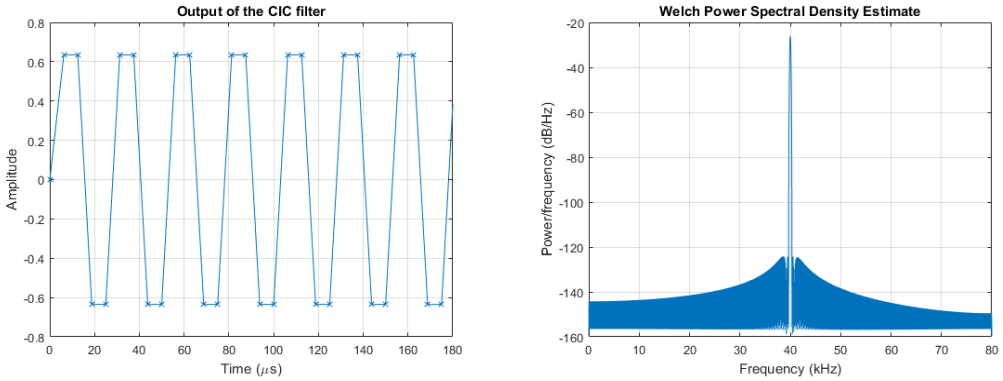


Figure 4.14: CIC Filter Block Diagram in Simulink.

The output signal of the simulated filter, in both time and frequency domains can be analysed in Figure 4.15.

In Figure 4.15 one can visualise the frequency component of 40kHz that was put through the ΣΔM as well as the number of samples per period (four), a reminiscent of the sample



(a) Time domain.

(b) Frequency Domain.

Figure 4.15: Output Signal of the CIC Filter.

rate at the output of the CIC filter - $f_s = 4 \times f_{in} = 160kHz$.

As seen in Figure 4.15a, each sample at the output of the filter is a rational number between -1 and 1. For that reason, one must take into account how such measures will be implemented in binary form. It was chosen a fixed point format represented in 16 bits for the output words, with the two leftmost bits representing the integer part and the remaining bits representing the fractional part of the sample. In others words, it is a Q2.14 format with one sign bit.

The constant to be divided in the end, 625, is represented in 10 bits. Being so, it was represented in format Q10.0 (integer). The samples must be, then, represented in format Q12.14, because when a sample of format Q12.14 is divided by a number of format Q10.0 results in a number of format Q2.14.

Given the 1 bit-length input, it was represented between -1 and 1 with the Q12.14 format, in twos complement, if the input was, respectively, bit 0 or 1.

Local Oscillator

It was previously stated that the local oscillator is to generate a cycle with the zeros, maxima and minima of a 40 kHz sine wave.

For this reason, the oscillator's operation was based in a multiplexer, whose select signal was incrementing at a rate of 160 kHz, and its output represented in Q2.14 format.

Mixer

The mixer takes two inputs in Q2.14 format and multiplies both, resulting in a Q4.28 format. However, given that one of the inputs is coming from the oscillator and is guaranteed to be 0, 1 or -1, it is safe to say that the output might be represented in format Q2.14. Therefore, the mixer discards the two MSB and the 14 LSB from the result of the multiplication and outputs the remaining.

Frequency Shift

The 90^o phase shift represented in Figure 4.10 is equivalent to a delay, once the sampling frequency is 4 times greater than the frequency of the oscillator. Being so, it is to be implemented with a D-type flip-flop whose clock runs with a frequency of 160 kHz.

Low Pass Filter

The LPF that follows the mixer will be implemented using 2 cascaded moving average filters, followed by downsampling. The benefits of cascading two of these filters lies on the possibility of selecting which frequencies can be eliminated. In order to do this, and having in mind Equation 4.18, where it says that the zeroes repeat and depend on the number of samples to average, K, one filter will be designed to have the first zero in a given, desired, frequency and the other filter will be designed to have a zero in a frequency where the first lobe of the transfer function of the first filter is maximum, as seen in Figure 4.16. This way, the cascaded moving average filters will present a sharper frequency response, with more attenuation outside of the bandpass.

The first moving average filter will be designed to present the first zero ($m = 1$) at frequency 2kHz. Being so, and knowing that the sample rate at the filter is 160 kHz, the number of samples to average is with Equation 4.18:

$$f = f_s \times \frac{m}{K} \Leftrightarrow K = f_s \times \frac{m}{f} = 160kHz \times \frac{1}{2kHz} = 80$$

Results, then, a moving average filter whose frequency response nulls are located at multiples of 2kHz. Being so, the first lobe will have a maximum at 3kHz, that is, then, the first zero of the frequency response of the next moving average filter, that must have the following number of samples to average:

$$K = f_s \times \frac{1}{3kHz} = 53.333 \approx 53$$

Figure 4.16 shows the frequency responses of both moving average filters overlapped.

The analysis of Figure 4.16 allows one to understand that the zeroes of both filters are not always interleaved, due to the fact that the filters present zeros at multiples of different frequencies (2kHz and 3kHz).Being so, the downsampler following the LPF will have a downsampling factor of 20, reducing the sampling rate from 160 kHz to 8 kHz and, thus, its output's frequency spectrum will spread through the band from 0 to 4 kHz.

The frequency response of the LPF, composed by both moving average filters cascaded, is represented in Figure 4.17.

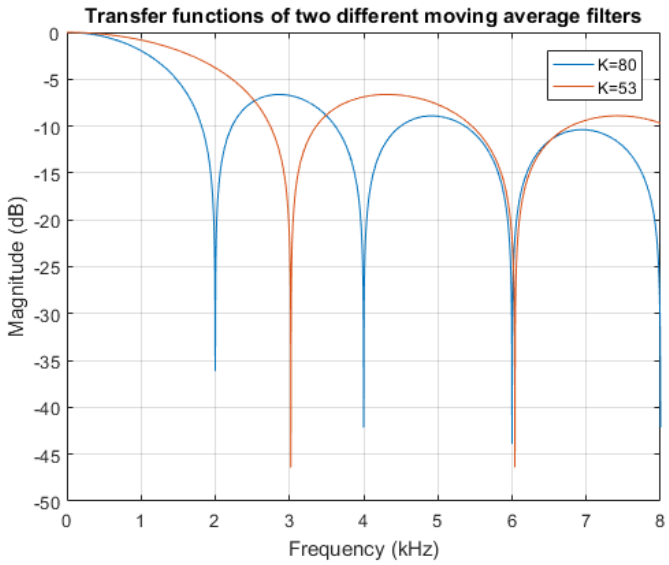


Figure 4.16: Frequency Responses of Both Moving Average Filters of the LPF.

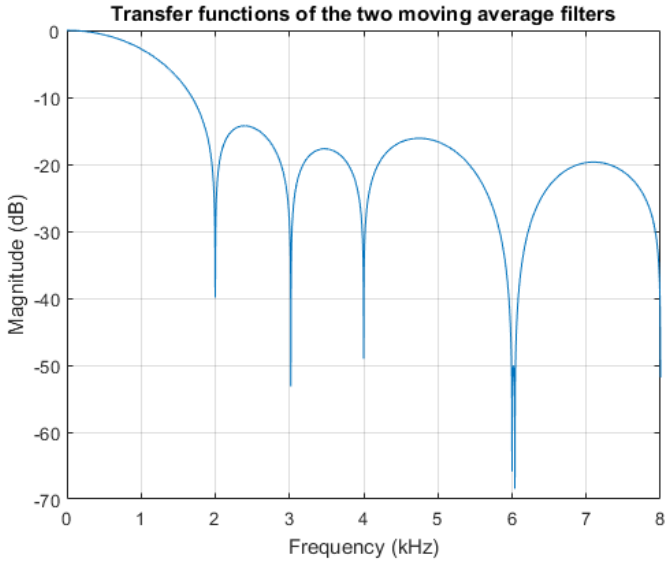


Figure 4.17: Frequency Response of LPF Composed by the Two Cascaded Moving Average Filters.

This filter was implemented using Matlab's Simulink and the diagram is shown in Figure 4.18.

Three sine waves of frequencies 1kHz, 2kHz and 2.5kHz were added up, alongside a DC component, and put at the input of the filter. Figure 4.19 depicts the Fast Fourier Transform (FFT) of both input and output signals.

As can be observed in Figure 4.19, the frequency component of 2kHz, which was standing in a local minimum of the filter's frequency response, was completely removed. The frequency component of 2.5kHz of the input signal, which stood in a local maximum of the frequency

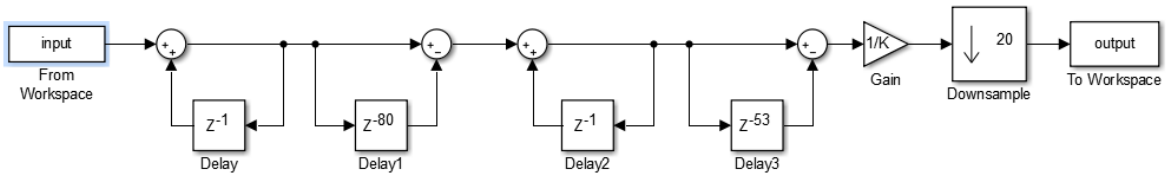
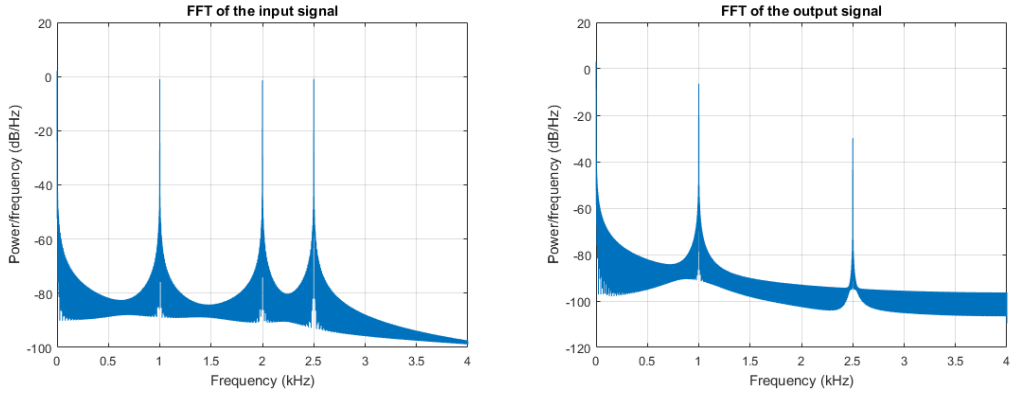


Figure 4.18: LPF implemented in Simulink.



(a) Input. (b) Output.

Figure 4.19: FFT of the input and output signals of the LPF.

response, was the one to suffer greater attenuation (without being completely removed), followed by the 1kHz frequency component, which stood inside the main lobe, although in a region prone to slight attenuation levels. The DC component appears to remain unaltered.

Although not presenting a perfect rejection band, this decimation filter seems appropriate for filtering narrow bandpass signals (like the ones depicted in Figure 3.25), after baseband conversion.

In order to implement this filter, care has to be taken in the number of bits inside the filter, because each cascaded moving average filter will be divided by a constant integer number (80 and 53, respectively). The biggest constant, 80, is represented with 7 bits, thus, assuming its representation in a fixed point format Q7.0, the samples inside the filter will have to be processed in format Q9.14 in order to output format Q2.14 samples, meaning a total of 23 bits samples.

The delay blocks can be implemented with synchronous First in, First out (FIFO) shift registers. Such block might be implemented cascading several D-type Flip-flops, as it is shown in Figure 4.20.

The delay block represented in Figure 4.20 will delay each input sample by four clock cycles. Every clock cycle the input of each D-type Flip-flop is passed to the next one. Being so, a delay block that is intended to delay a signal by an amount of K sampling cycles will require K cascaded D-type Flip-flops, as long as they run on the same clock and it has the same frequency as the sample rate.

In order to implement the delay block, D-type Flip-flops whose input and output bit length was 23 were designed and cascaded together in VHDL.

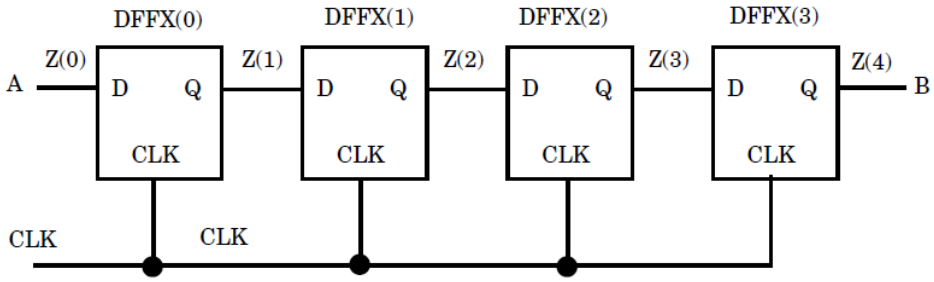


Figure 4.20: Delay Block implemented as a FIFO Shift Register [31].

The downsampler block was implemented using a D-type Flip-flop at the output of the filter, meaning a 16 bit length input and output ports, running with a clock 20 times lower than the output of the LPF sample rate. This way, the output sample rate of this Flip-flop will be 20 times lower.

The implementation of this filter was made having each integrator and comb of each moving average filter running in parallel.

Trigger Generator

In order to keep all blocks synchronised, a trigger generator block is needed.

This block is going to accept a global clock signal of frequency 100MHz, increment and, eventually, reset several counters at this rate. Parallel to these processes, other processes will output one pulse whenever the correspondent counter resets.

One trigger signal will trigger the ΣΔM to sample at a rate of 50MHz. Being so, its counter will reset every time it reaches the value 2 and start over incrementing from the value 1. This way, every time this counter resets, a pulse is output. This event occurs at a frequency of:

$$f = \frac{100MHz}{2} = 50MHz$$

Likewise, another trigger signal will trigger several events that must occur at a frequency of 160 kHz. Thus, the counter, that restarts every time from the value 1, must be reset every time it reaches the value:

$$\frac{100MHz}{160kHz} = 625$$

Whenever this dedicated counter signal reaches 625, it is reset and a single pulse is output.

At last, the trigger that will control the final downsampling, which will output a sample rate of 8kHz, and, thus, the trigger pulse will be output at the same rate, will do so every time a counter signal reaches:

$$\frac{100MHz}{8kHz} = 12500$$

This way it is ensured that all blocks in the downconverter are synchronised to a global clock signal.

4.4 Tests and Results

4.4.1 Analog Reception

The circuit schematised in Figure 4.11 was implemented on a breadboard, whose input was connected to an ultrasonic receiver MA40S4R, alongside with the analog integrators of the ΣΔM and connected to the FPGA board input ports in order to capture analog signals and convert them to the digital domain.

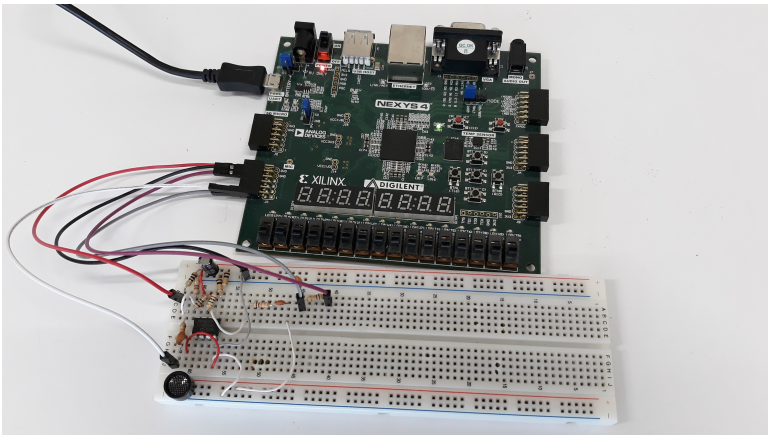


Figure 4.21: Receptor Implemented in a Breadboard.

To simulate a received wave, an ultrasonic emitter MA40S4S was connected to a function generator set to a 40kHz signal. Later, the emitter and receptor were set apart at a known distance and the tests took place. In a first place, with the emitter and receptor 50 cm apart, the saturation on the preamplifier was tested. The result is depicted in Figure 4.22.

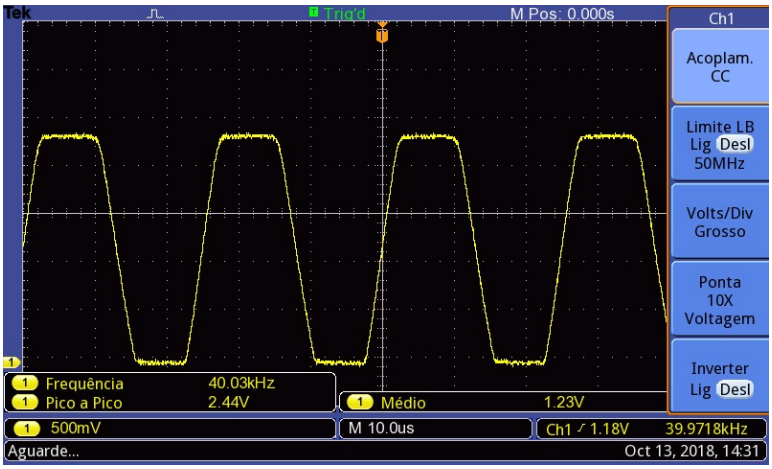


Figure 4.22: Measured saturation on the output of the Preamplifier.

A second test involved measuring the output of the preamplifier with different distances separating the emitter and the receiver, with a 5.9V (peak-to-peak) signal at the emitter's terminals. In order to do that, as it is show in Figure 4.23, both the emitter and receptor were put on top of file cabinets with wheels, to ease the mobility and, thus, better change the

distance, and were posteriorly aligned. The receptor will be still throughout the experiment and the receptor will be the one to be moved away. The floor tiles were 50cmx50cm squares and were used as reference to align the cabinets and to measure the distances.

Figure 4.23: Experimental setup for the measurement of the received signal strength as a function of distance; the emitter is on the left side, laying against the function generator, and the receptor is on the right side.

The measurements of peak-to-peak voltage at the output of the Preamplifier as a function of distance are explicit in Figure 4.24.

d (m)	V_{pp} (V)
0.5	2.32
1.0	1.14
1.5	0.86
2.0	0.30
2.5	0.20
3.0	0.40
3.5	0.14
4.0	0.07
4.5	0.14
5.0	0.09

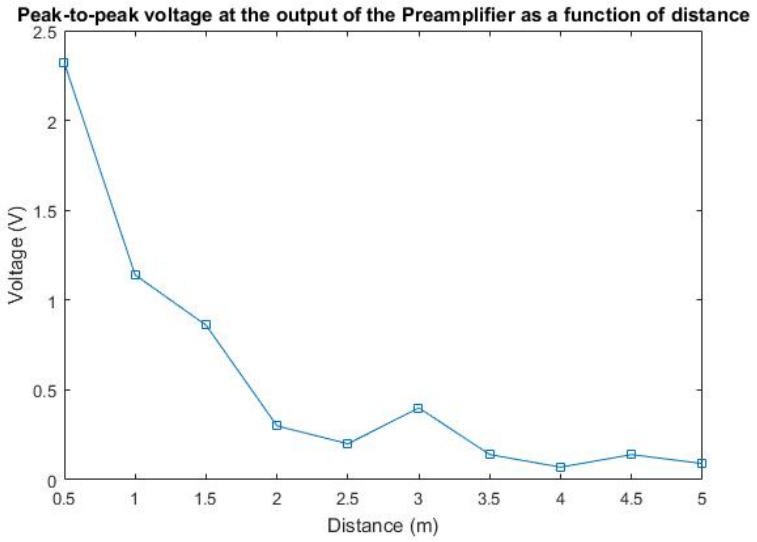


Figure 4.24: Peak-to-peak voltage at the output of the Preamplifier as a function of distance.

At last, with the emitter and receptor at a distance of 5m from each other, the received signal was captured and converted at the $\Sigma\Delta$M and its output was recorded and analysed in MATLAB in order to determine its PSD. However, this time the signal at the terminals of the emitter was measured to be 4.4V peak-to-peak. The result is depicted in Figure 4.25.

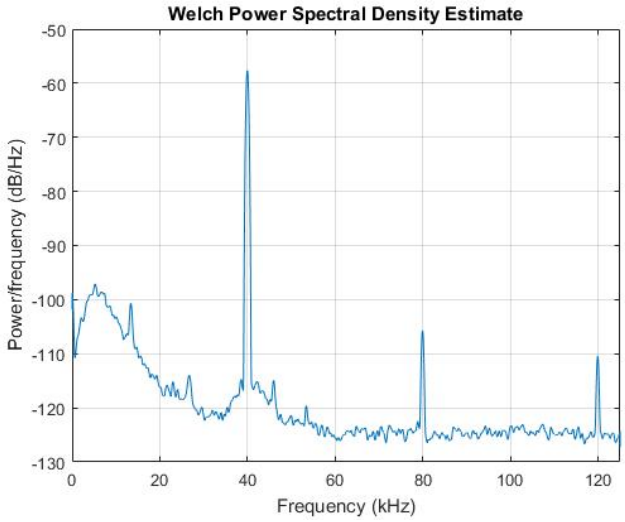


Figure 4.25: PSD of the output of the ΣΔM when its input signal is a preamplified signal with the source at a distance of 5m.

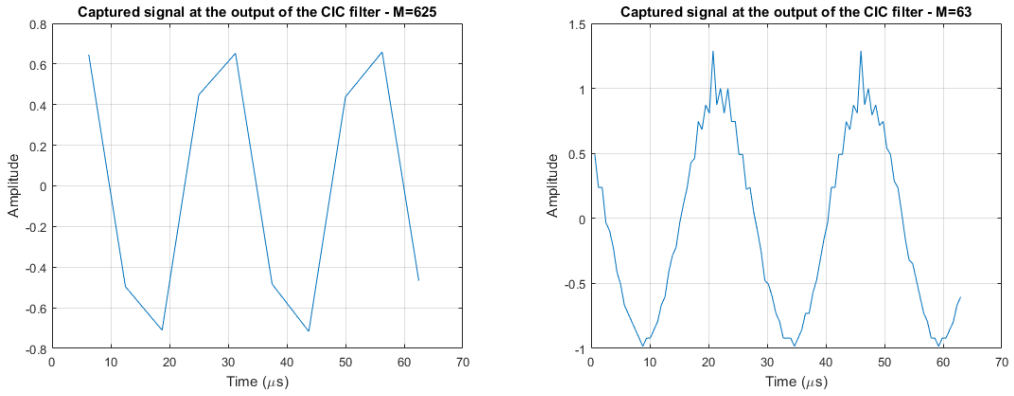
4.4.2 Digital Downconversion

CIC Filter

In order to test the CIC filter, a sine wave ranging from 0 to 2.5V and with frequency 40 kHz was sampled in the ΣΔM, that was followed by the filter, in a setup similar to the one depicted in Figure 3.24. The samples output by the CIC filter were then stored in memory and later analysed in Matlab, where there were made functions to convert the recovered samples in Q2.14 format to double format in order to compute and determine the PSD.

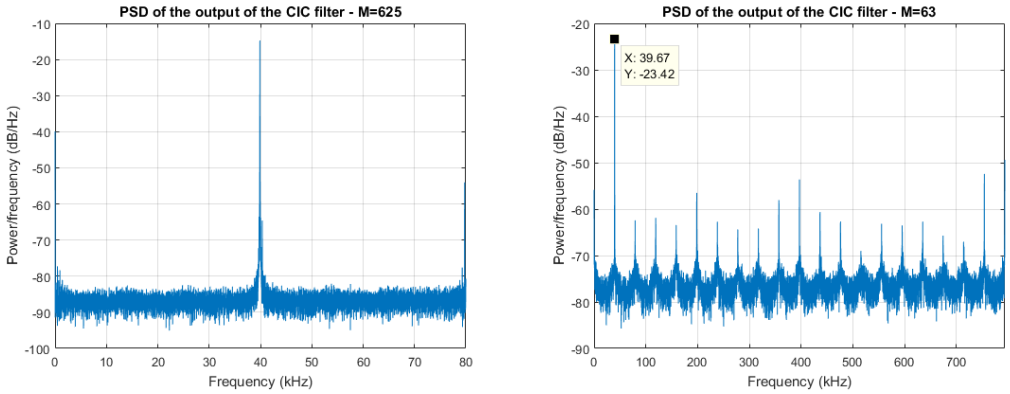
The experiment was made for the desired decimation factor $M = 625$ and for one about ten times lower, $M = 63$.

The output signals in the time domain are depicted in Figure 4.26, whereas their PSD are shown in Figure 4.27.



(a) M=625. (b) M=63.

Figure 4.26: Signals recorded at the output of the CIC filter.



(a) M=625. (b) M=63.

Figure 4.27: PSD of the signals recorded at the output of the CIC filter.

Downconversion followed by low-pass-filtering

The test of the LPF requires a downconverted digital signal at its input. Therefore, an analog sine wave must be sampled by the ΣΔM, filtered by the CIC and then mixed with the digital oscillator signal. This setup is schematised by the block diagram depicted in Figure 4.28 and it was implemented in VHDL.

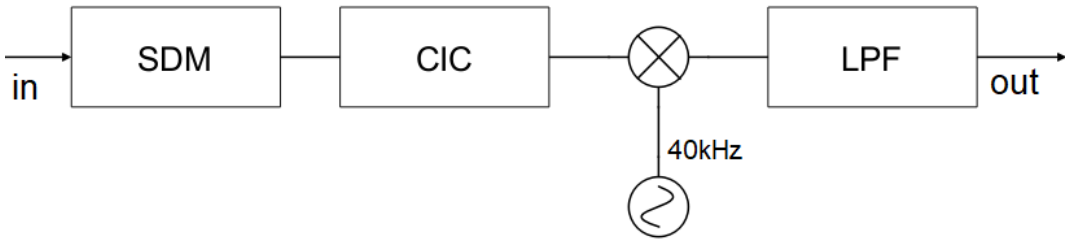
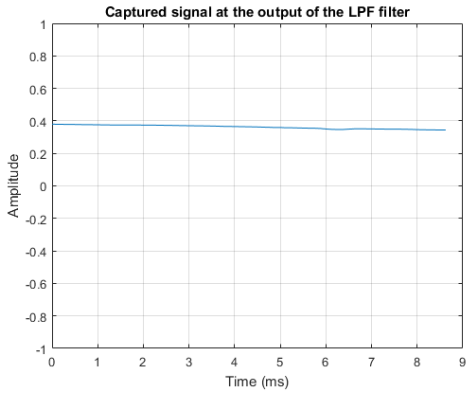
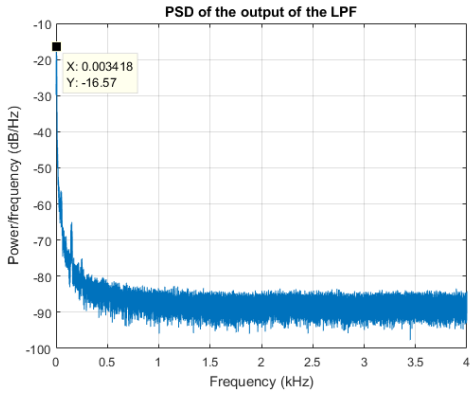


Figure 4.28: Experiment to test the LPF proper operation.

At the input of the portrayed system, several analog sine waves were input directly from a function generator, each with different frequency: 40kHz, 41kHz and 42 kHz. The results are depicted in Figures 4.29, 4.30 and 4.31, respectively.

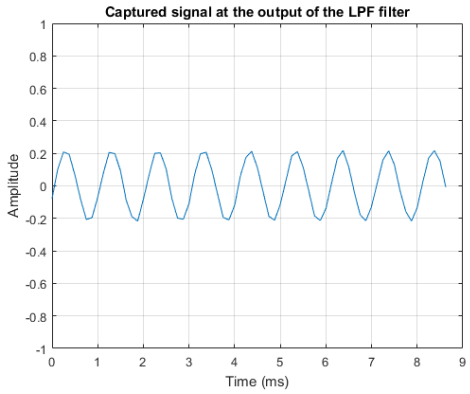


(a) Time Domain.

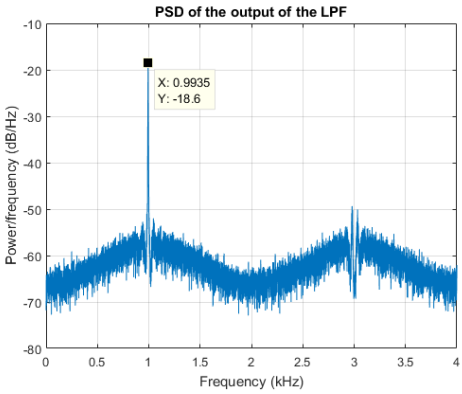


(b) Frequency Domain.

Figure 4.29: Output signal of the LPF when at input is a sine wave of frequency 40kHz.

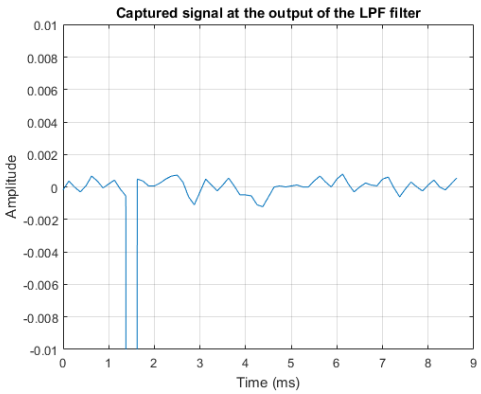


(a) Time Domain.

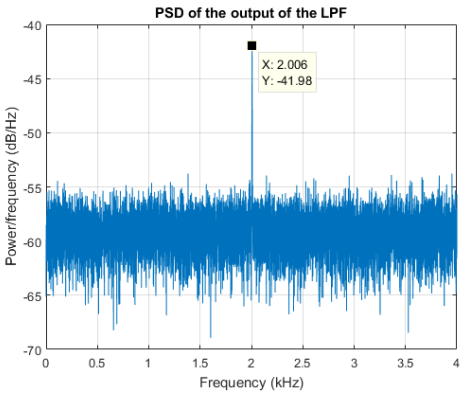


(b) Frequency Domain.

Figure 4.30: Output signal of the LPF when at input is a sine wave of frequency 41kHz.



(a) Time Domain.



(b) Frequency Domain.

Figure 4.31: Output signal of the LPF when at input is a sine wave of frequency 42kHz.

The power peaks of each PSD is stated in Table 4.1.

f_{in} (kHz)	40	41	42
Peak (dB)	-16.6	-18.6	-42.0

Table 4.1: PSD peaks for every input sine wave.

Downconverter

All the described blocks were assembled together in VHDL following their described operation and synchronisation in order to create the downconverter.

After that, several sine waves of different frequencies (39kHz, 40kHz and 41kHz) were input and both the *In-phase* and *Quadrature* signals were recorded and processed for further analysis.

Figures 4.32 and 4.33 show fractions of both *In-phase* and *Quadrature* signals recorded for an input sine wave of 41 kHz and 39kHz, respectively.

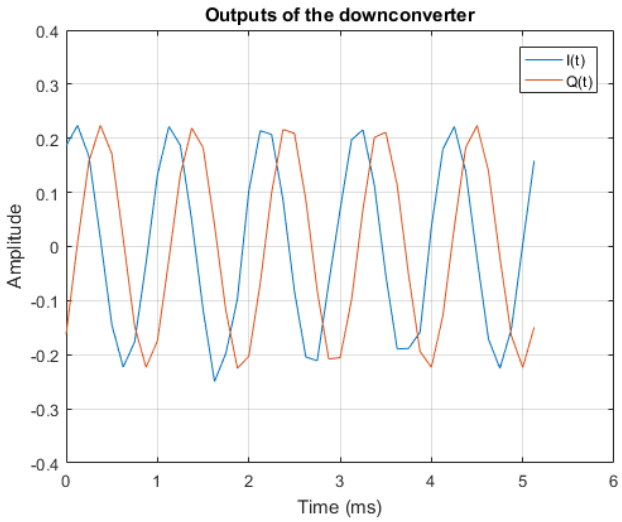


Figure 4.32: Recorded *In-Phase* - I(t) - and *Quadrature* - Q(t) - signals for an input sine wave of 41kHz.

The downconverted signal is given by:

$$x(t) = I(t) + jQ(t)$$

In MATLAB, $x(t)$ was computed with the recovered *In-phase* and *Quadrature* components that were recorded and its PSD was determined through Welch's method. Figures 4.34, 4.35 and 4.36 depict the result for, respectively, input frequencies of 39kHz, 40kHz and 41kHz.

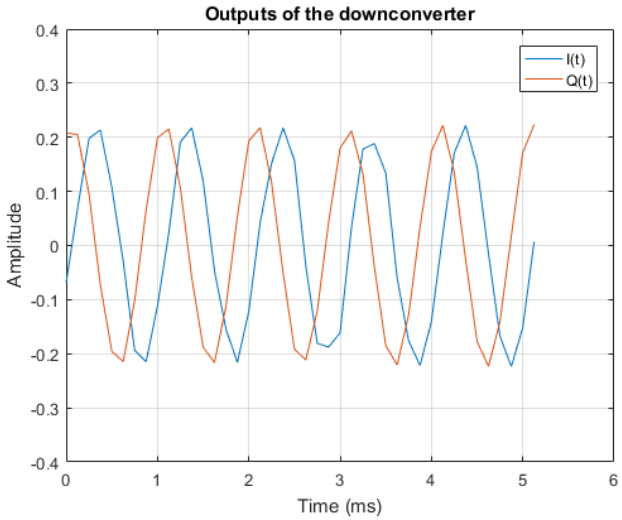


Figure 4.33: Recorded *In-Phase* - I(t) - and *Quadrature* - Q(t) - signals for an input sine wave of 39kHz.

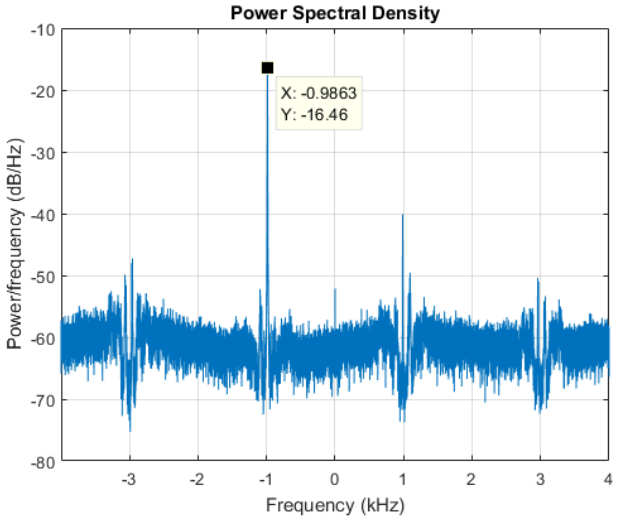


Figure 4.34: PSD of the downconverted signal when the input is a 39kHz sine wave.

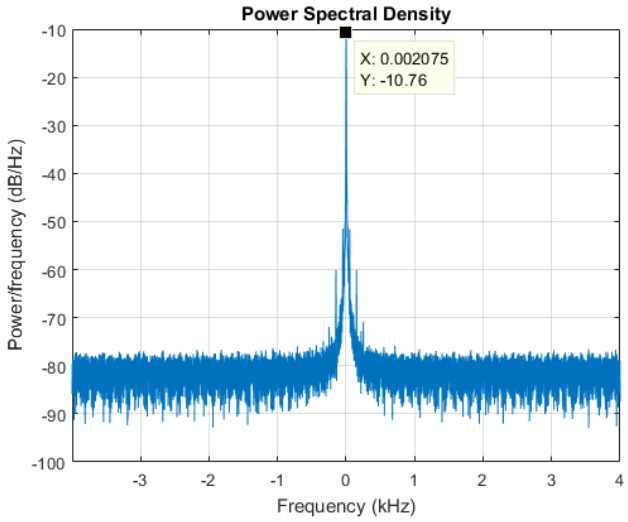


Figure 4.35: PSD of the downconverted signal when the input is a 40kHz sine wave.

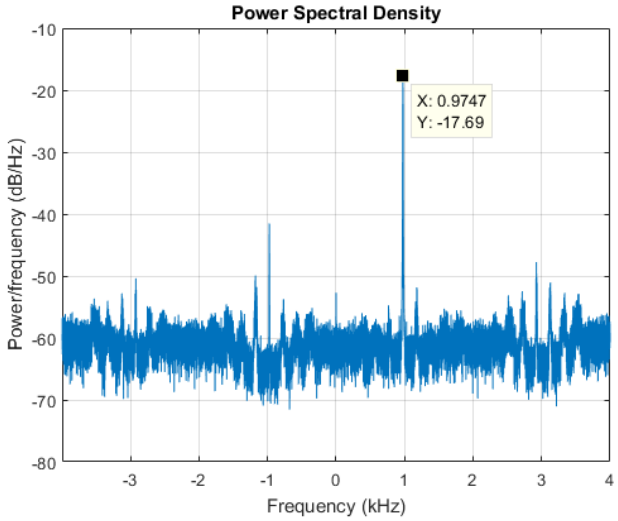


Figure 4.36: PSD of the downconverted signal when the input is a 41kHz sine wave.

4.5 Results Analysis and Discussion

4.5.1 Analog Reception

In Figure 4.22 is stated that the signal range at the output of the preamplifier (0V to 2.44V) is nearly that of the ΣΔM input and the DC offset (1.23V) is also near the theoretical one valued 1.25V. This rather small deviation is mainly due to the use of resistors with 5% tolerance that induce a small error on both the DC offset on the inverter's amplifier positive terminal, as well as in its gain.

In Figure 4.24, it is quite visible an attenuation effect of a propagating ultrasonic wave

with distance. However, it is also observable, at some points in space (namely at 3m and 4.5m) an increase in signal strength in respect to the previous point in space. This observed phenomenon was even more evident in closer points in space, it was studied, and the reason behind this turned out to be the effect of Lloyd's mirror, that, due to reflections on the ground, created constructive and destructive interferences that were dependent on the distance between emitter and receiver, as well as their height from the reflection plane (more on this topic on Appendix B). Hence, the signal increases its strength at some points, where there is constructive interference, and decreases it where there is destructive interference.

Figure 4.25 allows to conclude that, at a 5 meters range of the emitter from the receptor, the received signal of 40kHz is heavily protruding from the rest of the frequency spectrum. It is visible some power distributed along the low frequencies region, thought to be provoked by the fact that the circuit was mounted on breadboard and, thus, noise-prone. However, the SFDR is about 40dB, which means that the most powerful frequency component outside the band of interest (39-41kHz) is close to 10000 times lower than the power of the fundamental signal.

This helps lead to the conclusion that, considering that a reflection would not attenuate an ultrasonic wave or, at least, to a great extent, an object at a distance of 2.5m is easily detected, which is a reasonable distance, given the requirements of the sonar. Due to the fact that the voltage measured at the emitter's terminals was 4.4V peak-to-peak and the maximum voltage is 20 V peak-to-peak [32], increasing this range is perfectly feasible.

4.5.2 Digital Downconversion

CIC Filter

As deduced previously, for a CIC filter with a downsampling factor of 625 and an input sampling rate of 100 MHz, its output would have a sample rate of 160 kHz. For a 40kHz signal, this means four samples per period at the output of the CIC filter. Such can be observed in Figure 4.26a. Its PSD, that is represented in Figure 4.27a states clearly that the signal has a frequency of 40kHz.

On the other hand, when an input sine wave is sampled in the $\Sigma\Delta$M and put through a CIC filter with downsampling factor of 63, the output in the time domain resembles more like a sine wave, as shown in Figure 4.26b, although contaminated with noise, due to a larger bandwidth that contains harmonics, as depicted in Figure 4.27b.

Low Pass Filter

Inspecting Figures 4.29b, 4.30b and 4.31b, it is possible to understand that the mixer is effectively converting a bandpass signal to the baseband.

Table 4.1 shows that the operation of the filter is close to the expected one. The input sine wave of frequency 41kHz suffer a slightly greater attenuation when compared with the 40kHz input. The 42 kHz input, although not being completely eliminated as expected and shown in Figure 4.19 , it suffered an attenuation of about 20dB more. In linear terms, it means that its output amplitude is about 100 times smaller. Such can be observed in Figures 4.30a and 4.31a, where the later shows an amplitude of about 100 times lower.

The peak observed in Figure 4.31a shows a phenomenon where a sample is output with a value much lower than the remaining samples shown. This apparently random phenomenon

also happens for positive samples and it might be due to noise components, mainly the one at 2kHz that is ineffectively eliminated. Besides that the signal is quite noisy due to its low amplitude. However, fundamental components of 2kHz are not expected at reception, and so this phenomenon should not cause concern.

Figure 4.30a shows an almost perfect sine wave with a period of about 1ms, which is equivalent to a frequency of 1kHz, which is an expected behaviour.

Figure 4.29a shows an almost DC signal, which was the expected behaviour. According to Figure 4.29b, the most powerful frequency component is 3Hz. This deviation is most certainly due to the uncertainty of the function generator.

Downconverter

Figures 4.34, 4.35 and 4.36 show the PSD of the downconverted signals for inputs of, respectively, 39kHz, 40kHz and 41kHz. They all reflect the expected behaviour, because, knowing they were mixed with a 40kHz signal, the baseband components should be, respectively, $-1kHz$, $0kHz$ and $+1kHz$.

Figures 4.32 and 4.33 show the result of the downconversion, *i. e.*, the *In-phase* and *Quadrature* components of two different modulated signals.

Figure 4.32 results from an input of 41kHz and, thus, it is a baseband signal of 1kHz. Because of this positive frequency, the phase shift of 90^o or $pi/2$ rad delays the *In-phase* component in time to form the *Quadrature* component and, thus, it is shifted right in relation to the *In-phase* component. On the other hand, Figure 4.33 results from an input of 39kHz, representing a baseband signal of -1kHz. Being the baseband signal's frequency negative, a positive phase shift of 90^o advances the *Quadrature* component in time and, thus, it is shifted left when compared to the *In-phase* component.

Chapter 5

Conclusions

5.1 Conclusions

The first main objective of this project is considered successfully accomplished. A simple ADC was designed, with few passive components and, at the carrier frequency, a performance that was close to that of some commercial options available.

This way, it is possible to implement the system with a simple ADC that is integrated within the implemented hardware, without the need of many components, eliminating the need to buy a commercially available one, and obtaining similar performances. It helps reduce costs as well as improve the portability of the system, which is a major requirement.

The second task was also completed. A reception channel was designed and shown to be able to detect echoes coming from reasonable distances. Also, the downconverter would output signals whose PSD would clearly indicate small frequency shifts due to obstacles velocity. The *In-Phase* and *Quadrature* components that where recovered enabled to extract instantaneous amplitude and phase of the receiving signal, that are the main requirements to be able to implement the beamformer.

It was shown that a reception channel of ultrasound carrier can be implemented with most of its complexity implemented in FPGA, which is also a great contribution for portability increase and cost reduction.

Besides these objectives, there were implemented functions in MATLAB that enabled the computation of performance indicators with good precision. The indicators are SNR, SINAD, THD and SFDR and the methods used are described in detail in Appendix A.

5.2 Future Work

The future work of this project, in a first phase, is to finish the third proposed task, *i. e.*, design and implement the beamformer. This is done after multiplying the reception channel designed in Chapter 4 and estimate the DoA from relating all extracted phases from each channel. This step might as well include the implementation of an array of ultrasound receivers, ideally in PCB.

An alternative implementation is to make an array of ultrasound transceivers and switch

between two modes: reception and emission. This way, a single array might be involved in the estimation of the DoA as well as in operating as a parametric speaker in order to *sonify* the detected object, integrating the work developed and presented on this thesis with the one presented in [2].

Having done this, the system will have to be submitted to tests in order to determine the optimum modulation of the emitting signal as well as the best audio signal possible to be detectable in spite of other ambient noises. The main objective is to try to make the audible signal stand out of the remaining noises, easing the echolocation of the obstacle.

A last, tests should be performed with blind people in order to detect flaws and improve the system.

Appendix A

Performance Indicators

A.1 Introduction

It is described in this section the mathematical analysis performed on the ΣΔM output signal in order to estimate the distribution of power throughout a specified frequency spectrum and, thus, compute the performance indicators of the modulator.

The main intention is to evaluate the PSD of the output signal of the ΣΔM, determine the power inside the input signal band (signal power), the equivalent to its harmonics (harmonics power) and the power distributed in the remaining frequency components (noise power). The several performance indicators are ratios of these computed values, as it is to be discussed.

A.1.1 Discrete-time Random Processes

A *random* or *stochastic process* is a physical phenomenon whose future output or result cannot be predicted within a reasonable experimental error and, thus, cannot be described by an explicit mathematical relationship[33]. In order to fully understand the data, several experiments will have to be performed, *i. e.*, different time observations resulting in several recorded time histories. These time histories records, collectively, are called the *ensemble* and it defines the *random process*, through some averaging properties.

The *expected value* (also said *average value*) of a *random process* at discrete time n_1 is computed by averaging over the *ensemble* at the said moment:

$$\mu(n_1) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} x_i[n_1] \tag{A.1}$$

On the other hand, its *mean square value* is computed by averaging, at moment n_1, over the squared *ensemble*:

$$\psi^2(n_1) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} x_i^2[n_1] \tag{A.2}$$

Furthermore, the average product of the data values at times n_1 and $n_1 + k$, is said to be

the *autocorrelation function* at time n_1 and delay k, and is given by:

$$R_{xx}(n_1, k) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} x_i[n_1] x_i[n_1 + k] \tag{A.3}$$

The *autocorrelation function* provides information about de degree of linear dependence between two random variables. For example, if:

$$R_{xx}(n, k) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} x[n] x[n + k] = 0$$

The random variables x[n] and x[n+k] are uncorrelated and one can not be estimated from the knowledge of the other's value [34].

One other way to interpret the autocorrelation function of random data is that it measures how well future values of data can be predicted from past observations[33]. Given a null result, as stated previously, it is possible to infer that there is no correlation between random variables and, so, it is not possible to predict the future value. Otherwise, if $R_{xx}(n, k) = 1$, there is a perfect direct linear correlation and, if $R_{xx}(n, k) = -1$, there is a perfect inverse linear correlation.

Stationarity

A *stochastic process* is said to be *stationary* when its *expected value*, *mean square value* and *autocorrelation function* remain constant at a given time n_x. In other words, the averaged value over the *ensemble* remains constant with changes in the time domain.

Otherwise, it is considered a sufficient condition if any of the averaging constants just described is not constant in time for the process to be considered *non-stationary.*

Ergodicity

A *stochastic stationary process* is said to be *ergodic* when, with probability 1, all its statistical averages can be predicted from a single waveform of the process *ensemble* via time averaging[35].

This way, only one time history is sufficient, *i.e.*, one realization, to be able to estimate the statistical averages of the process. Given only one time realization, $x[n]$, with N samples, its *mean value*, provided that the number of samples is large, is given by:

$$\mu_x(N) = \frac{1}{N} \sum_{n=0}^{N} x[n] \tag{A.4}$$

Its *mean squared value* is defined as:

$$\psi^2(N) = \frac{1}{N} \sum_{n=0}^{N} x^2[n] \tag{A.5}$$

And, finally, its *autocorrelation function* is:

$$R_{xx}(k) = \frac{1}{N} \sum_{n=0}^{N} x[n] x[n + k] \tag{A.6}$$

A.1.2 The Power Spectrum

The *Power Spectrum*, also called *Power Spectral Density*, is defined as the Discrete-time Fourier Transform (DTFT) of the *autocorrelation function*[35]:

$$P_{xx}(f) = T \sum_{m=-\infty}^{\infty} R_{xx}[m] e^{j\omega mT} \tag{A.7}$$

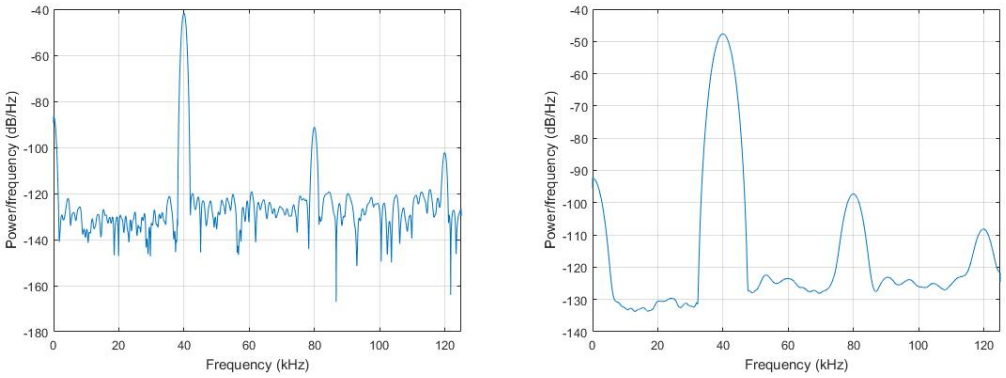
Thus, the PSD provides a frequency domain description of the process[34].

A.1.3 Welch's Method of PSD Estimation

As stated in Equation A.7, the PSD can only be precisely computed for an infinite number of samples. In practice, such is not possible. On account of that, several methods of PSD estimation arose, being *Welch's Method* the one explored here.

The *Welch's Method* was based on previous methods, namely *Bartlett's Method*, in wich a discrete signal with a finite sample number would be divided in smaller, adjacent segments, each of them used to compute a Power Spectrum, and then all the resulting Power Spectrums would be averaged together, resulting in the estimated PSD. In order to do this, the signals would have to be ergodic in order to compute precise approximations of power distribution in the frequency spectrum.

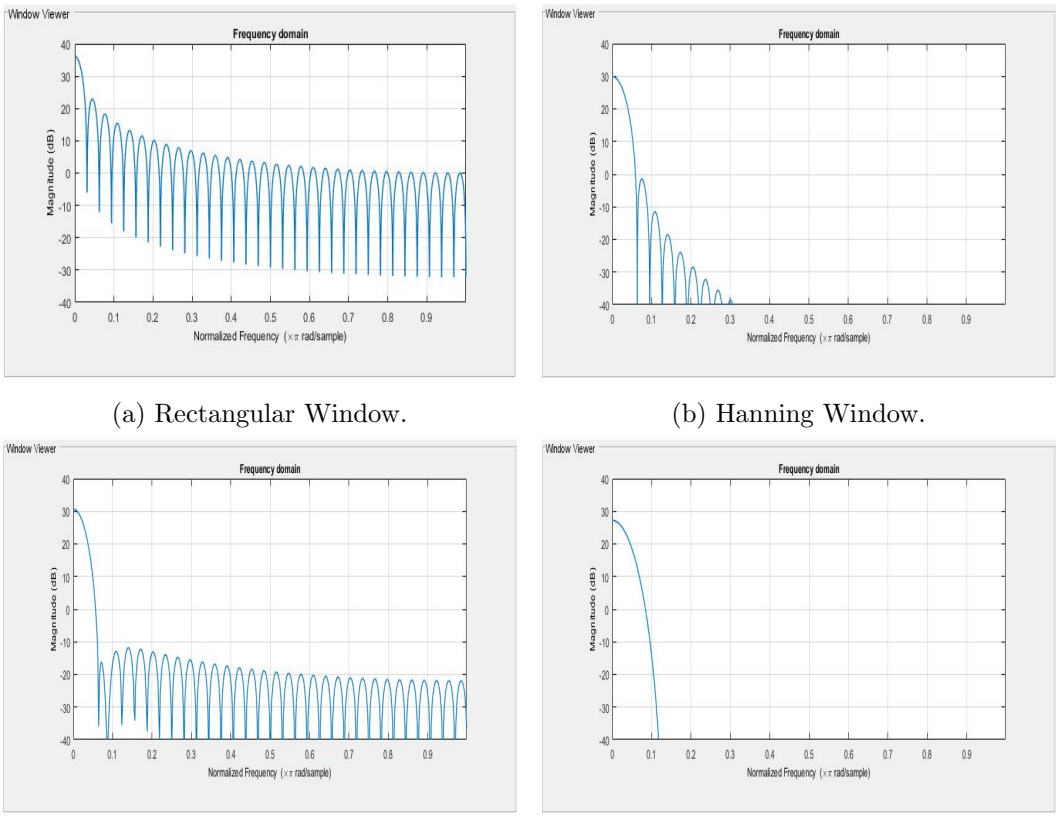
This process results in a statistically stable spectral estimate[35], involving a trade off between the required smoothness in the spectral estimate versus the frequency resolution, controlled by the length of each segment (number of samples), *i.e.*, for an increasing length of each segment, there are less PSD's averaged together and, thus, the frequency resolution increases, decreasing the smoothing degree of the resulting estimated PSD, as it is illustrated in Figure A.1.



(a) Segment Size: 512 samples. (b) Segment Size: 128 samples.

Figure A.1: Estimated PSD of the same signal, with different segment sizes.

Welch's Method modified the described segment-and-average methods by windowing the segments and overlapping them[35]. Overlapping increases the number of segments whose spectra is averaged, decreasing the PSD estimate variance. On the other hand, windowing reduces spectral leakage, by selecting a window that presents attenuated sidelobes, as depicted



(a) Rectangular Window.

(b) Hanning Window.

(c) Hamming Window.

(d) Nuttall Window.

Figure A.2: Different 64 sample length Windows Frequency Response.

in Figure A.2, where it is obvious that the rectangular window (used in the other methods) has the worst sidelobe attenuation and the Nuttall Window has the best.

Spectral leakage, in practice, adds power to the noise band, increasing the noise floor level and, thus, degrading the performance indicators dependent on the noise power, such as SNR.

A.2 Spectral Analysis of the Output of the ΣΔM

To measure the performance indicators, a low distortion sine wave is put at the input of the ΣΔM. Being the input periodic, considering that comparator on the ΣΔM adds random, non-linear noise and that the output depends to a great extent on the input, it is fair to conclude that the output is ergodic, *i. e.*, it's *expected value* (mean), *mean squared value* and *auto-correlation function* are constant in time, for all time histories taken. As such, the first step in the spectral analysis is data acquisition, being that one time history is sufficient to achieve fulfilling results.

The maximum memory capacity of the FPGA device used (Nexys 4 with XC7A100T Artix-7) would only allow to capture 131072 words of 36 bits each, meaning a total of 4718592 bits, captured at 50 MHz of sampling frequency. At this sampling frequency, a 1 kHz input sine wave would have about 94 periods sampled, which should be considered enough to achieve a good level of precision in the averaged PSD.

Followed a reduction of the sampling frequency with lowpass filtering, enabling a better analysis of the power distribution at lower frequencies. It was chosen a nem sampling frequency of 250 kHz:

```
fs=50e6;                            %Initial sampling frequency
M=fs/250e3;                         %Decimation factor
output = decimate(output,M);        %Downsampling + LPF
output = output-mean(output);       %DC component removal
fs = fs/M;                          %New sampling frequency
L = length(output);                 %New bitstream length
```

Next step was to apply the *Welch's Method* in order to estimate the PSD of the resulting decimated bitstream. A Nuttal window of 512 samples in length was chosen in order to reduce spectral leakage and, thus, avoid a noise floor above its effective level. The chosen overlapping factor was 0.5 and the number of FFT points used to estimate de PSD was 2^{16}:

```
wind_size = 512;            %Window size
[P,F] = pwelch(output,nuttallwin(wind_size),wind_size/2,2^16,fs);   %Welch's Method
```

Figure A.3 depicts the resulting PSD of the generated bitstream for a 40kHz sine wave at the $\Sigma\Delta$M's input, sampled at a frequency of 50 MHz, where the fundamental frequency and respective harmonic (80 and 120kHz) components protrude from the noise floor.

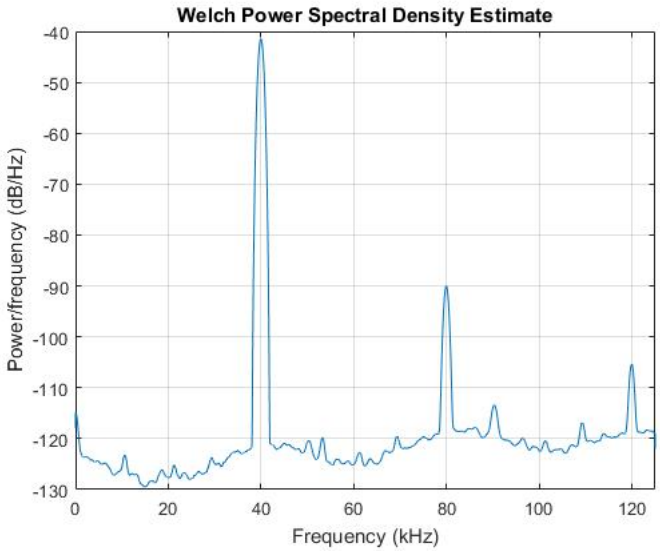


Figure A.3: Estimated PSD for a 40 kHz input sine wave.

A.3 Performance Indicators Computation

The basis for the computation of the performance indicators lies in identifiyng which frequency relates to the several signal components - signal, harmonics and noise.

As such, the method applied involves the computation of local minima, dividing the spectrum in different lobes, recognize the fundamental frequency, its harmonics and the noise components and, at last, compute the power in each frequency component.

A.3.1 Signal-to-Noise-Ratio

In MATLAB, it is pretended to separate a signal whose frequency band is centered in f and rests between frequencies f_{c1} and f_{c2} ($f_{c1} < f < f_{c2}$), discarding the fist n harmonic components, assuming that the following components may be negligible when comparing to noise levels.

In a first stage, the PSD of the noisy signal is estimated through Welch's method, as previously described, and the power contained in each frequency bin is then determined:

```
[Pxx,F] = pwelch(x,nuttallwin(wind_size),wind_size/2,NFFT,fs);
df = mean(diff(F));                 %Frequency resolution
Pxx = df.*Pxx;                      %Total power in each frequency bin
```

Follows the identification of the values of the fundamental frequency and its harmonics in the spectrum:

```
fund = F(find(Pxx==max(Pxx)));      %Fundamental Frequency

%Computation of a vector with the fundamental frequency and harmonics to be
%excluded from noise power computation
k=1;                                %Harmonic index (1st, 2nd, ...)
harms = fund;                       %Harmonics vector (fundamental included)
while k≤n                           %n is the number of harmonics to exclude
    if fund*(k+1) ≤ fs/2            %Inside the PSD band
        harms = [harms fund*(k+1)]; %Consider as harmonic
    else break;
    end
    k = k+1;
end
```

And then the respective lobes:

```
%Local minima in PSD function to determine lobe widths
[Min, Locs] = findpeaks(-10*log10(Pxx));

f_lims = 0; %Vector that contains beginning and end frequencies of fundamental and ...
    harmonic lobes
k = 1;
i=1;
f1 = 0;         %Beginning of a lobe
f2 = 0;         %End of a lobe

%Creation of a vector, f_lims, with the lobe limits of each harmonic and
%the fundamental
while k≤length(harms)
    while(F(Locs(i))≤harms(k)) %Get the beginning of the lobe
        f1 = F(Locs(i));
        i = i+1;
        if i==length(Locs)      %Stop in the last minimum
           break;
        end
    end
    f2 = F(Locs(i));            %End of the lobe (minimum right next to f1)
    f_lims = [f_lims f1 f2];    %Concatenate
    k = k+1;                    %Next Harmonic
end

f_lims = f_lims(2:end);         %Discard first index (equals zero)
```

Follows the computation of the powers inside each lobe, that are computed by summing up the powers of the frequency bins inside the respective lobe:

```
sp = 0;          %Signal power
np = 0;          %noise power
i=1;             %Index for frequency array
k=1;             %Index for harmonic component

%Signal and Noise powers computation
for i=1:length(Pxx)
    if F(i)≥f_lims(2*k-1) & F(i)≤f_lims(2*k)     %Harmonics/Fundamental band
        if k==1                                  %Fundamental lobe
            sp = sp + Pxx(i);                    %Signal Power
        else                                     %Harmonic lobe (do nothing)
        end
    elseif F(i) > f_lims(end)                    %Noise band w/o harmonics
        np = np + Pxx(i);                        %Noise power
    else                                         %Noise band w/ Harmonics
        if i>1 && F(i-1) == f_lims(2*k)          %End of Harmonic lobe
            k = k+1;                             %Next Harmonic
            if (k==length(f_lims)/2+1)           %Out of band (f>fs/2)
                break;                           %Break cycle
            end
        end
        if F(i) ≤ F(Locs(1))                       %Discard DC component
        else
            np = np + Pxx(i);                    %Noise Power
        end
    end
end
```

At last, the PSD is computed, according to Equation 3.4:

```
out = 10*log10(sp/np);
```

The code presented was implemented in a MATLAB function, given as input arguments the sampling frequency (`fs`), the number of harmonics to be discarded (`n`) and, of course, the signal that is intended to be analysed (`x`).

A.3.2 Signal-to-Noise and Distortion Ratio

As for the SINAD estimation, it is intended to obtain an approximation of the PSD of a noisy signal through Welch's method in order to determine the power available in the signal and noise bands, including the frequency bins corresponding to distortion and compute the SINAD. In order to accomplish that, in a first moment the fundamental frequency lobe should be identified:

```
fund = F(find(Pxx==max(Pxx)));      %Fundamental Frequency

%Local minima in PSD function to determine Fundamental Ferquency lobe width
[Min, Locs] = findpeaks(-10*log10(Pxx));

i=1;
f1 = 0;          %Fundamental frequency lobe minimum frequency
f2 = 0;          %Fundamental frequency lobe maximum frequency
while F(Locs(i))<fund
    f1 = F(Locs(i));
    i = i+1;
end
```

```
f2 = F(Locs(i));
```

Then, the fundamental frequency, as well as the noise with distortion powers are to be computed:

```
sp = 0;          %Signal Power
np = 0;          %Noise+Distortion Power

for i=1:length(F)
    if F(i) ≤ F(Locs(1))                %DC component (discarded)
    elseif F(i) ≥ f1 & F(i) ≤ f2        %Inside lobe (signal band)
       sp = sp + Pxx(i);
    else                                 %Outside lobe (noise band)
       np = np + Pxx(i);
    end
end
```

Finally, the SINAD is computed, accordingly with Equation 3.5:

```
out = 10*log10(sp/np);
```

The code presented was implemented in a function, given as input parameters the sampling frequency, `fs` and the noisy signal that is to be analysed, `x`.

A.3.3 Spurious Free Dynamic Range

In order to computed SFDR in MATLAB, it is necessary to compute the power of the fundamental frequency and the biggest power present outside this band.

On the algorithm, the first step is to identify the spectral area containing the fundamental frequency, *i. e.*, the lobe limits:

```
fund = F(fund_idx);                 %Fundamental Frequency

%Local minima in PSD function to determine Fundamental Frequency lobe width
[Min, Locs] = findpeaks(-10*log10(Pxx));

i=1;
f1 = 0;          %Fundamental Frequency lobe minimum frequency
f2 = 0;          %Fundamental Frequency lobe maximum frequency
while F(Locs(i))<fund
    f1 = F(Locs(i));
    i = i+1;
end
f2 = F(Locs(i));
```

It follows the identification of the most powerful noise and distortion component , *i. e.*, the spurious:

```
max_spur = 1;       %Index of maximum spurious location
Pow = 0;            %Maximum spurious power
for i=1:length(Pxx)
    if F(i) ≤ F(Locs(1))           %Ignore DC component
    elseif F(i) < f1 | F(i) > f2    %Noise band
        if Pxx(i) > Pow
            max_spur = i;
            Pow = Pxx(max_spur);    %Greatest power so far
        end
    end
end
```

The algorithm ends with the SFDR computation:

```
out = 10*log10(Pxx(fund_idx))-10*log10(Pow);
```

The code presented was implemented in a MATLAB function, given as input parameters the sampling frequency, `fs` and the noisy signal that is to be analysed, `x`.

A.3.4 Total Harmonic Distortion

The algorithm starts by identifying the harmonics location in frequency:

```
k=1;                    %Harmonic index
harms = fund;           %Start in fundamental frequency
aux = 0;

while k≤n               %n is the number of harmonics taken into consideration
    aux = fund*(k+1);   %k-th harmonic
    if aux ≤ fs/2       %k-th harmonic inside interest band
        harms = [harms fund*(k+1)];
    else break;
    end
    k = k+1;            %Next harmonic
end
```

The next step of the algorith is to find each harmonic lobe extremes and putting them together on a vector:

```
[Min, Locs] = findpeaks(-10*log10(Pxx));        %Local minima

f_lims = 0;         %Lobes extremes vector
k = 1;              %Harmonic index
i=1;                %Minimum index
f1=0;               %Lobe minimum
f2=0;               %Lobe maximum

while k≤length(harms)
    while(F(Locs(i))≤harms(k))      %frequencies under the given harmonic
        f1 = F(Locs(i));            %Lobe minimum frequency
        i = i+1;
        if i==length(Locs)          %end of loop on last local minimum
           break;
        end
    end
    f2 = F(Locs(i));                %Lobe maximum frequency
    f_lims = [f_lims f1 f2];        %lobe extremes are concatenated into the vector
    k = k+1;                        %Next harmonic
end
f_lims = f_lims(2:end);             %First vector element is discarded (f_lims(1)=0)
```

It follows the computation of the signal and the combined harmonics power:

```
sp = 0;     %Signal Power
hp = 0;     %Harmonics Power
i=1;        %Frequency index
k=1;        %Harmonic index

for i=1:length(Pxx)
    if F(i) ≤ F(Locs(1))    %Discard DC component
    elseif F(i)≥f_lims(2*k-1) & F(i)≤f_lims(2*k) %Inside lobes
        if k==1     %Inside signal power lobe
            sp = sp + Pxx(i);
```

```
        else        %Inside harmonic lobe
            hp = hp + Pxx(i);
        end
    else    %Outside lobes
        if i>1 && F(i-1) == f_lims(2*k) %End of harmonic lobe
            k = k+1;        %Next harmonic
            if k==n+1       %Last of considered harmonics (break loop)
                break;
            end
            if (k==length(f_lims)/2+1) %Last harmonic inside interest band (break loop)
                break;
            end
        end
    end
end
```

The algorithm ends with the computation of the THD in dB:

```
out = 10*log10(hp/sp);
```

The code presented was implemented in a function, given as input parameters the sampling frequency, `fs`, the noisy signal that is to be analyzed, `x` and the number of harmonics that are to be taken into consideration on the THD computation, `n`.

A.4 Validation

In order to verify the correct operation of the described functions for the estimation of the performance indicators, in MATLAB, it was generated a signal comprised of a fundamental frequency with two harmonics and zero-mean random noise. Theoretically, the powers of each component (fundamental frequency, harmonics and random noise) were calculated, following the determination of such indicators (computed), later compared to the output results of the functions (estimated).

Considering the discrete nature of the signals, the power of a finite and periodic signal with fundamental period N is given by[15]:

$$P = \frac{1}{N}\sum_{n=0}^{N}|x(n)|^2 \tag{A.8}$$

Equation A.8 will be used to compute the powers (fundamental, harmonics and noise) needed to calculate the theoretical values of the indicator performances.

In MATLAB, firstly, a signal with fundamental frequency of 400 Hz, a total duration of a hundred periods and sampling frequency of 4kHz is implemented, alongside with its first two harmonics, of amplitude 1/4 and 1/16 of its total amplitude:

```
fs = 4000;          %Sampling frequency
Ts = 1/fs;          %Sampling period
f = 400;            %Fundamental frequency
N = 10*100;         %Number of points for the sine wave (100 periods)
t = [0:N-1]*Ts;     %Time vector
T = 1/f;            %Fundamental period
f2 = 2*f;           %First Harmonic
f3 = 3*f;           %Second Harmonic

A =1;                       %Amplitude of the fundamental frequency
```

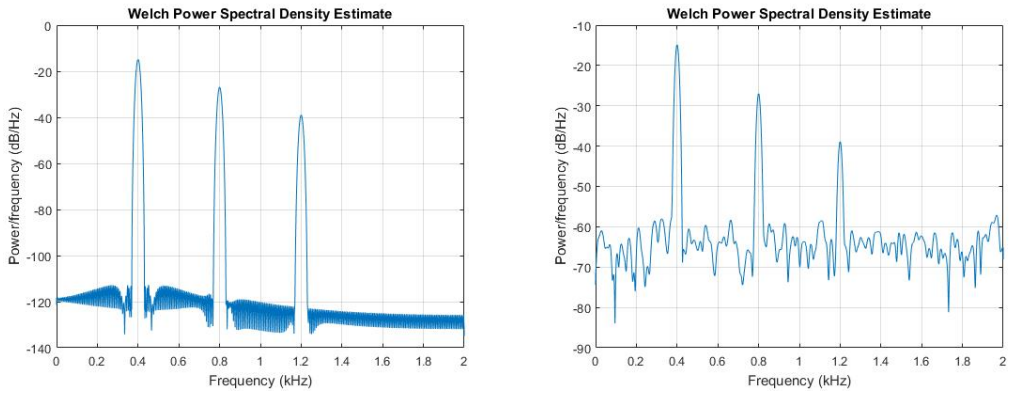
```
A2 = A/4;                    %Amplitude of the first harmonic
A3 = A2/4;                   %Amplitude of the second harmonic

%Definition of the sine wave with harmonics
fund = A*cos(2*pi*f*t);      %Fundamental
h1 = A2*cos(2*pi*f2*t);      %1st Harmonic
h2 = A3*cos(2*pi*f3*t);      %2nd Harmonic
harms = h1 + h2;
x = fund + harms;            %Resulting signal
```

The zero-mean random noise is implemented with 1/10 of the fundamental frequency amplitude and then added to the signal:

```
n = A/10*rand(1,length(x)); %10% of the fundamental signal amplitude
n = n-mean(n);              %Zero mean
x_n = x+n;                  %Noisy signal
```

Figure A.4 depicts the Power spectra of the generated signal, with and without noise, obtained using the described Welch method with a *Nuttall* window. The harmonic components, as well as the fundamental frequency are quite clear in both cases.



(a) Clean signal PSD.

(b) Noisy signal PSD.

Figure A.4: Power spectra of the generated signals.

The following step involved the theoretical computation of the powers of the fundamental signal, harmonic distortion and noise, accordingly with Equation A.8:

```
fund_power = mean(fund.^2);
harm_power = mean(harms.^2);
noise_power = mean(n.^2);
```

Finally, the algorithm computes the performance indicators according to Equations 3.4, 3.5, 3.6 and 3.7:

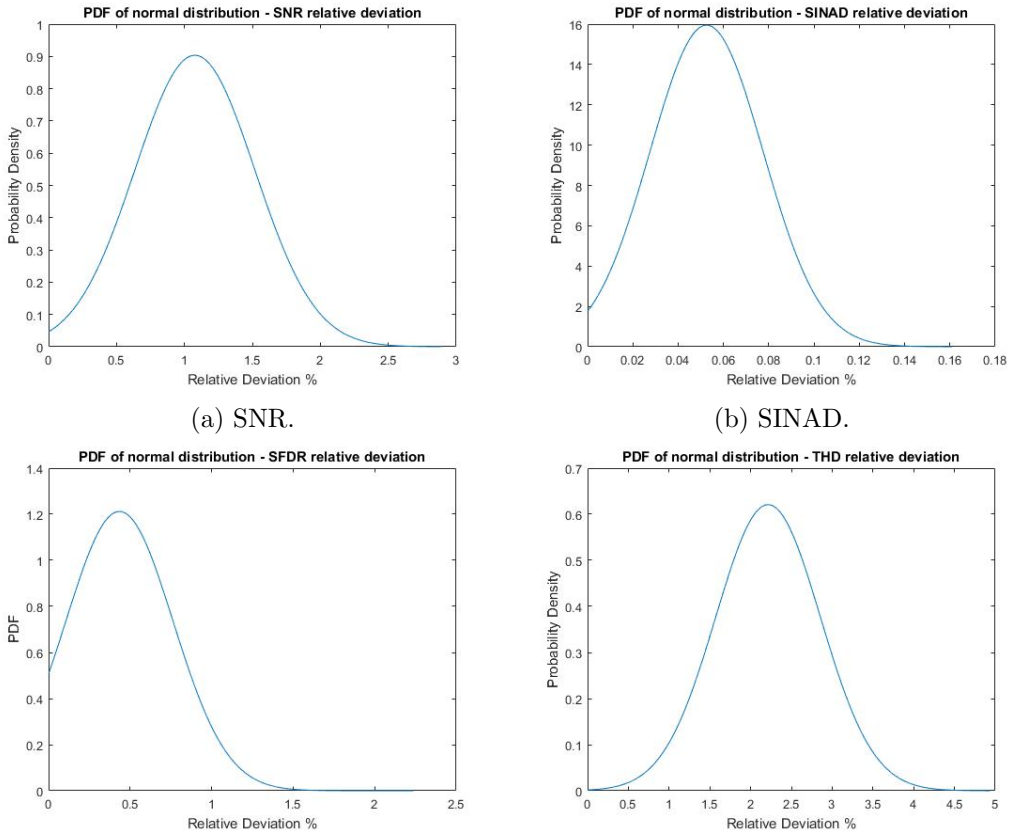
```
snr_calc = 10*log10(fund_power/harm_power);                    %Theoretical SNR value
sinad_calc = 10*log10(fund_power/ (harm_power + noise_power)); %Theoretical SINAD value
sfdr_calc = 10*log10(mean(fund.^2)) - 10*log10(mean(h1.^2));   %Theoretical SFDR value
thd_calc = 10*log10(harm_power / fund_power);                  %Theoretical THD value
```

Running the simulation, the results on Table A.1 have been achieved:

Indicator	Computed (dB)	Estimated (dB)	Relative Deviation (%)
SNR	27.76	28.22	1.66
SINAD	11.67	11.69	0.171
SFDR	12.04	12.03	0.0831
THD	-11.78	-12.02	2.03

Table A.1: Computed and estimated through Welch's Method Performance Indicators of the MATLAB generated signal.

Being the added noise a random variable, the indicators, which depend on the noise power, end up being random as well, such as the relative deviations. Being so, to better understand how different the estimated and computed values might be, the same simulation was run 100000 times, the relative deviations obtained registered and a normal distribution curve was drawn. The curves are depicted in Figure A.5 and the parameters are represented in Table A.2.



(a) SNR.

(b) SINAD.

(c) SFDR.

(d) THD.

Figure A.5: Normal Distribution PDF's.

The results in Table A.2 show that the estimated results will most likely differ at a low degree from the computed values. The SNR and THD represent greater averages, although not quite high, as well as maximum values. However, due to its low dispersion (given by the standard deviation), it is highly unlikely that such maximum relative deviations occur.

	μ	σ	Maximum	Minimum
SNR	1.71%	0.72%	5.53%	7.29e-4%
SINAD	0.49%	0.37%	2.69%	9.41e-6%
SFDR	0.43%	0.33%	2.24%	4.27e-5%
THD	2.21%	0.64%	4.94%	8.31e-4%

Table A.2: Parameters of the normal distributions (Mean - μ - and Standard Deviation - σ) and extremes of the measured relative deviations.

Given the results, it is safe to conclude that the method used to estimate the performance indicators will most likely be accurate.

Appendix B

Interference of Ultrasonic Waves

Wave propagation, although having different natures, such as *electromagnetic* and *mechanical*, behaves with a huge amount of similarities. Simply put, *electromagnetic waves* differ from *mechanical waves* in a way in which it does not need a medium to be propagated [36].

The following pages will describe phenomena that were initially verified with light waves. Nonetheless, the same phenomena are verified in sound wave propagation, although much slower.

B.1 Introduction

There are two conditions that must be met in order to attain spacial interference of waves and they are [36]:

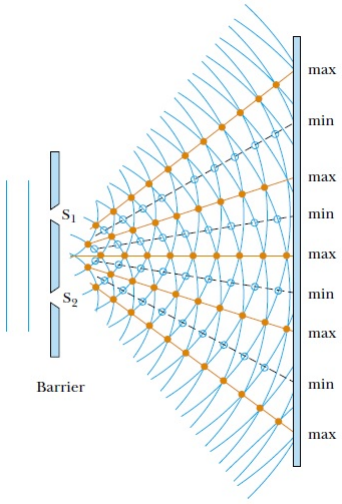
- They must be **coherent**, *i. e.*, they must maintain a constant phase difference;
- Their sources must be monochromatic, *i. e.*, of only a single wavelength.

Interference happens when 2 different waves that follow the described conditions overlap in space, creating what is called a constructive interference when they are in phase (amplifying each other) and destructive interference when they are in quadrature (cancelling each other out).

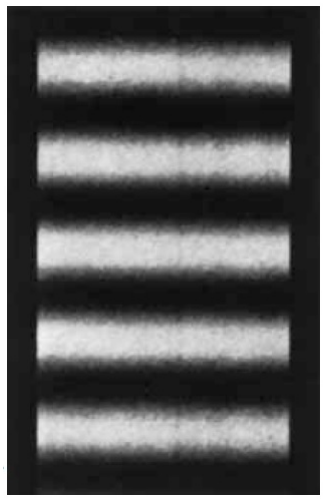
B.1.1 Double-Slit Experiment

Thomas Young's double-slit experiment demonstrates this phenomenon. It uses a monochromatic light source that is aimed at two parallel slits standing close together. The light passing through each slit starts propagating as if it were a new source. Being so, both waves are coherent and monochromatic. A screen standing parallel to and apart from the slits' plane can capture the interferences pattern , as illustrated in Figure B.1.

Let two slits, s_1 and s_2, hit by a monochromatic wave, be separated by a distance d and standing from a screen at a distance L, as it is depicted if Figure B.2. Let a point P be at a distance r_1 from one slit and r_2 from the other and at a height y from the horizontal plane standing exactly in between the slits, where it intersects in point Q. *Theta* is the angle between the said plane and the line connecting points Q and P.



(a) Constructive and Destructive Interferences in Space.



(b) Resulting Pattern on the Screen.

Figure B.1: Thomas Young's Double-slit Experiment[36].

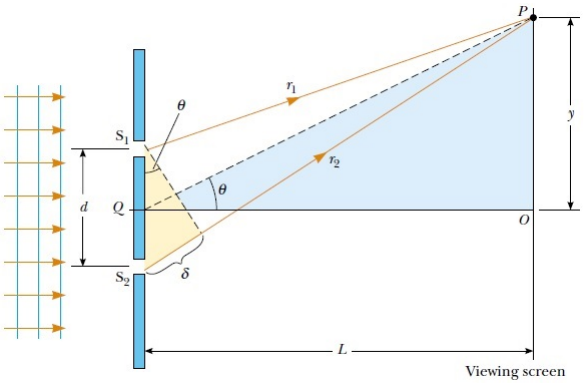


Figure B.2: Geometric reference for the Double-Slit Experiment Analysis[36].

The interference at point P depends on the phase difference between the waves coming from sources s_1 and s_2, which is related to the difference in travelled space, *i. e.*, the **path difference**:

$$\delta = d \sin \Theta \tag{B.1}$$

In its turn:

$$\sin \Theta = \frac{y}{\sqrt{y^2 + L^2}}$$

And, thus, the path difference becomes:

$$\delta = d \times \frac{y}{\sqrt{y^2 + L^2}} \tag{B.2}$$

Whenever there is a constructive interference, both waves in point P are in phase, which means that the path difference is a multiple of the wavelength:

$$\delta = n\lambda \Leftrightarrow n\lambda = d\frac{y}{\sqrt{y^2 + L^2}} \tag{B.3}$$

Solving for L gives:

$$L = \sqrt{\left(\frac{dy}{n\lambda}\right)^2 - y^2} \tag{B.4}$$

Which are the points in space where constructive interference occurs.

Whenever destructive interference occurs, at point P, both waves are in quadrature, meaning that the path difference is an odd multiple of half of the wavelength:

$$\delta = (2n+1)\frac{\lambda}{2} \Leftrightarrow (2n+1)\frac{\lambda}{2} = d\frac{y}{\sqrt{y^2 + L^2}} \tag{B.5}$$

Solving for L gives:

$$L = \sqrt{\left[\frac{2dy}{(2n+1)\lambda}\right]^2 - y^2} \tag{B.6}$$

Which are the points in space where destructive interference occurs.

B.1.2 Lloyd's Mirror

Lloyd's mirror is a phenomenon that allows to generate the same pattern as the Double-Slit Experiment with just one slit, that acts as a single source. When a reflective plane (mirror) is placed beneath this source, reflections occur and end up simulating the existence of a virtual source under the reflective plane, at the same distance as the real source. This can be visualised in Figure B.3

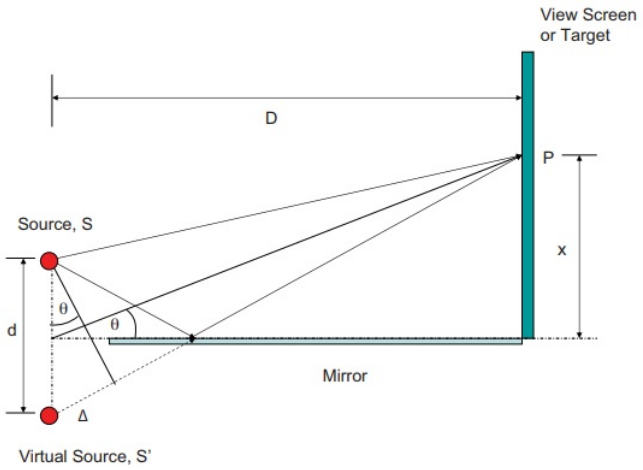


Figure B.3: Visualisation of the Lloyd's Mirror phenomenon[37].

Referring to Figure B.3, the path difference between the covered distances from the real and virtual sources is:

$$\Delta = d \times \sin\Theta = d\frac{x}{\sqrt{x^2 + D^2}}$$

Whenever a wave hits a reflective medium and gets reflected, it undergoes a phase shift of 180°[36], which is equivalent to an increase of half a wavelength in the path difference. Being so, the path difference for the Lloyd's Mirror phenomenon is:

$$\Delta = d\frac{x}{\sqrt{x^2 + D^2}} + \frac{\lambda}{2}$$

Just like in the Double-Slit Experiment case, also for the Lloyd's Mirror situation happens constructive interference whenever the path difference is a multiple of the wavelength:

$$\Delta = m\lambda \Leftrightarrow d\frac{x}{\sqrt{x^2 + D^2}} + \frac{\lambda}{2} = m\lambda$$

Solving for D gives:

$$D = \sqrt{\left(\frac{2dx}{\lambda(2m-1)}\right) - x^2} \tag{B.7}$$

And whenever there's destructive interference, the path difference is an odd multiple of half of the wavelength:

$$\Delta = (2m+1)\frac{\lambda}{2} \Leftrightarrow m\lambda \Leftrightarrow d\frac{x}{\sqrt{x^2 + D^2}} + \frac{\lambda}{2} = (2m+1)\frac{\lambda}{2}$$

Solving for D results in:

$$D = \sqrt{\left(\frac{dx}{\lambda m}\right)^2 - x^2} \tag{B.8}$$

B.2 Experimental Verification

In order to verify this phenomenon, an experimental setup similar to the one depicted in Figure 4.23 and described in its Section was set.

Firstly, knowing at which height the transducers were (0.59m), the frequency of the carrier - 40kHz - (and, thus, its wavelength - 8.6mm - assuming ambient temperature - v=343m/s), and for several different values of m on Equations B.7 and B.8, the several points in space where constructive and destructive interferences occur were computed, through MATLAB. Some results are shown in Table B.1, rounded toward the *millimetre*.

m	Constructive (m)	Destructive (m)
35	2.278	2.243
36	2.210	2.177
37	2.145	2.114
38	2.083	2.054
39	2.025	1.996
40	1.969	1.942
41	1.916	1.890
42	1.865	1.841
43	1.817	1.794
44	1.771	1.748
45	1.727	1.705
46	1.684	1.664
47	1.643	1.624
48	1.604	1.585
49	1.567	1.548
50	1.530	1.513
51	1.496	1.479
52	1.462	1.446
53	1.430	1.414
54	1.398	1.383

Table B.1: Some Constructive and Destructive points in space computed in MATLAB.

Knowing these points, both emitter and receptor transducers are to be put at said distances and the signal strength (peak-to-peak voltage) is to be measured at the output of the preamplifier.

The distances between transducers from 1.5m to 2m to measure signal strength where chosen because there were 24 interference points computed in this space and the closest reflective plane (a wall) was standing at a greater distance (around 2.3m), as it is desired to avoid interferences coming from another mirror.

The results follow in Figure B.4.

The peak-to-peak voltage results showed a constant oscillation around the measured value with an amplitude of about 0.02V and the distance between transducers was rounded to the *centimetre* due to the difficulty to align the transducers with *millimetric* precision.

Nonetheless, although some fluctuations are visually observable in the plot, the peak-to-peak voltages at the output of the preamplifier were always higher at computed constructive interference distances than the ones at destructive interference points of space.

d (m)	Interf.	V_{pp} (V)
1.50	C	1.0
1.51	D	0.4
1.53	C	0.9
1.55	D	0.6
1.57	C	1.0
1.59	D	0.6
1.60	C	0.8
1.62	D	0.6
1.64	C	1.0
1.66	D	0.6
1.68	C	1.0
1.71	D	0.6
1.73	C	1.0
1.75	D	0.3
1.77	C	1.0
1.79	D	0.4
1.82	C	1.1
1.84	D	0.4
1.87	C	0.8
1.89	D	0.4
1.92	C	0.7
1.94	D	0.2
1.97	C	0.7
2.00	D	0.3

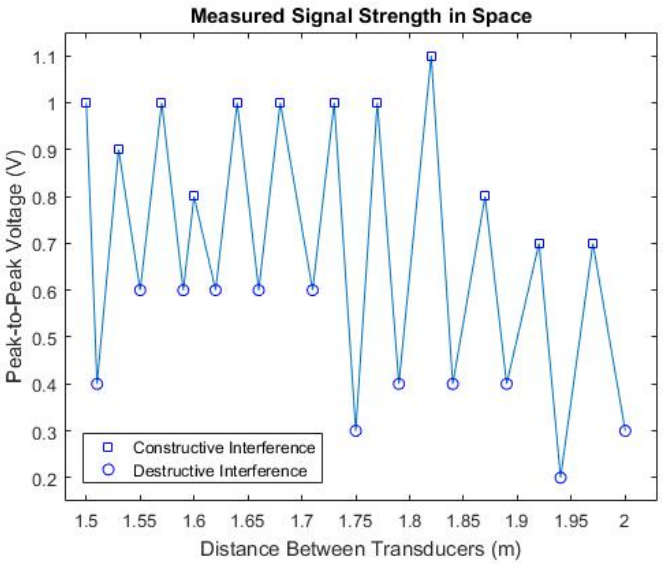


Figure B.4: Results from the Interference Measurement Experience.

Appendix C

$\Sigma\Delta$ Modulator – VHDL implementation

C.1 Top-level file

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
Library UNISIM;
use UNISIM.vcomponents.all;

entity SigmaDelta is
    port(SignP  :   in  std_logic;
         SignM   :   in  std_logic;
         BufOut :   out std_logic;
         clk    :   in  std_logic;
         );
end SigmaDelta;

architecture Behavioral of SigmaDelta is
    signal  s_clk, s_IBufOut, s_FFOut   :   std_logic;


begin
    --INPUT DIFFERENTIAL BUFFER
    DifBuf : IBUFDS
            generic map (
                -- Differential Termination
                DIFF_TERM => FALSE,
                -- Low power (TRUE) vs. performance (FALSE)
                IBUF_LOW_PWR => FALSE,
                IOSTANDARD => "DEFAULT")
            port map (O => s_IBufOut, -- Buffer output
                    -- Diff buffer positive terminal
```

```
                    I => SignP,
                     -- Diff buffer negative terminal
                    IB => SignM
                    );

    --FREQUENCY/CLOCK DIVIDER
    clkDiv: entity work.ClkDividerN(Behavioral)
        --Division Factor
        generic map(divFactor => 2
        )

        port map(clkIn => clk,      -- input is FPGA's internal clock
                 clkOut => s_clk);  -- output is an internal signal

    --D-TYPE FLIP-FLOP
    DFF:    entity work.dflipflop(Behavioral)
        port map(dataIn => s_IBufOut,   --input is IBUFDS's output
                 dataOut => s_FFout,    --output is OBUF's input
                 enable => '1',         --always enabled
                 sysclk => s_clk        --clock is output of clock divider
                 );

    --OUTPUT BUFFER
    OBUF1:  OBUF
         generic map (DRIVE => 16,      -- drive strength
                      IOSTANDARD => "DEFAULT",
                      SLEW => "Fast")   --Fast slew rate
         port map (O => BufOut, -- Buffer output
                   I => s_FFOut -- Buffer input is Flip-Flops's output
                   );
end Behavioral;
```

C.2 Sampling

C.2.1 D-type Flip-flop

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Dflipflop is
    port(dataIn :   in  std_logic;  --input port (1 bit)
         dataOut :  out std_logic;  --output port (1 bit)
         sysclk  :  in  std_logic;  --external clock input
         enable  :  in  std_logic   --enable input
         );
end Dflipflop;
```

```
architecture Behavioral of Dflipflop is

begin

    process(sysclk)
    begin
        if(rising_edge(sysclk)) then
            --only if "enable" signal is on
            if(enable = '1') then
                --input equals output at clock rising edge
                dataOut <= dataIn;
            end if;
            -- otherwise, maintains value
        end if;
    end process;

end Behavioral;
```

C.2.2 Frequency Division

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity ClkDividerN is
    generic map(divFactor : natural := 2);   --division factor
    port map(clkIn : in std_logic;           --input clock signal
             clkOut : out std_logic          --output clock signal
             );
end ClkDividerN;

architecture Behavioral of ClkDividerN is
     --Signal to count the number of input clock cycles
        signal s_divCounter : natural;

begin
        assert(divFactor >= 2);

        process(clkIn)
        begin
                if (rising_edge(clkIn)) then
                    --when N periods have passed
                        if (s_divCounter = divFactor - 1) then
                            --output turns off
                                clkOut <= '0';
                                --counter resets
                                s_divCounter <= 0;
```

```
                        else
                            --N/2 clock cycles have passed
                                if (s_divCounter = (divFactor / 2 - 1)) then
                                    --output turns on
                                        clkOut <= '1';
                                end if;
                                --counter signal increments
                                s_divCounter <= s_divCounter + 1;
                        end if;
                end if;
        end process;
end Behavioral;
```

C.3 Shift-Register

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity shift_register is
    generic (N  :   integer);       --Number of output bits
     port(Data_in    :   in  std_logic;
          Data_out   :   out std_logic_vector(N-1 downto 0);
          sysclk     :   in  std_logic; --Clock input
          en         :   in  std_logic; --Enable
          trig       :   out std_logic  --Trigger signal
         );
end shift_register;

architecture Behavioral of shift_register is
    signal  count   :   integer;        --counter signal
    signal  tmp     :   std_logic_vector(N-1 downto 0); --output
begin

process(sysclk)     --this process controls the counter signal
begin
if(rising_edge(sysclk)) then
        if(count = N-1) then    --N bits shifted at this point
            count <= 0;         --Counter is reset
            trig <= '1';        --trigger signal set to high level
        else                    --Otherwise,
            count <= count + 1; --counter is incremented
            trig <= '0';        --trigger is forced to low level
        end if;
end if;
end process;
```

```
--this process shifts the output signal, while inserting the input bit
process(sysclk)
begin
    if(rising_edge(sysclk)) then
            --(N-1) least significant bits concatenated with the input bit
            tmp <= tmp(N-2 downto 0) & Data_In;
    end if;
end process;

Data_out <= tmp;

end Behavioral;
```

C.4 Constraint File

```
##CLOCK
set_property PACKAGE_PIN E3 [get_ports clk]
set_property IOSTANDARD LVCMOS25 [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000}
    -add [get_ports clk]

## IBUFDS INPUTS
##Positive Terminal
set_property PACKAGE_PIN K2 [get_ports SignP]
set_property IOSTANDARD LVDS_25 [get_ports SignP]
##Negative Terminal
set_property PACKAGE_PIN K1 [get_ports SignM]
set_property IOSTANDARD LVDS_25 [get_ports SignM]

## OBUF
set_property PACKAGE_PIN J4 [get_ports BufOut]
set_property IOSTANDARD LVCMOS25 [get_ports BufOut]
```

Appendix D

Downconverter - VHDL Implementation

To ease the comprehension of some implemented blocks (namely, Top-level file, CIC filter and LPF), in Section D.12 can be found the respective block diagrams with the signals used, as well as input and output ports also represented.

D.1 Top-Level File

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_STD.all;

Library UNISIM;
use UNISIM.vcomponents.all;

entity downconverter is
    port(clk    :   in  std_logic; --Clock input(internal - 100MHz)
        signP   :   in  std_logic; --Positive terminal of IBUFDS
        signM   :   in  std_logic; --Negative terminal of IBUFDS
        bufOut  :   out std_logic  --Output for SDM's feedback loop
        );

end downconverter;

architecture Behavioral of downconverter is
    --Trigerring signals
    signal  s_50MHz, s_160kHz, s_8kHz       :   std_logic;
    --Output of input differencial buffer
    signal  ibufds_out                      :   std_logic;
     --Sigma-Delta Modulator output
    signal  sdm_out                         :   std_logic;
    --Output of the CIC filter
    signal  cic_out                         :   signed(15 downto 0);
```

```
    --Output of the Oscillator
    signal  osc_out                         :   signed(15 downto 0);
    --In-phase branch mixer output
    signal  mix_out_i                       :   signed(15 downto 0);
    --Quadrature branch mixer output
    signal  mix_out_q                       :   signed(15 downto 0);
    --90 Phase Shift Block output signal
    signal  shift_ph_out                    :   signed(15 downto 0);
    --In-Phase branch LPF output
    signal  lpf_out_i                       :   signed(15 downto 0);
    --Quadrature branch LPF output
    signal  lpf_out_q                       :   signed(15 downto 0);

    --Enable debug of the output signals (disable optimisation)
    attribute dont_touch : string;
    attribute dont_touch of lpfI    : label is "true";
    attribute dont_touch of lpfQ    : label is "true";

begin

    -- Trigger Signals Generator
ccu:    entity work.CCU(Behavioral)
        port map(clk_in =>  clk,
                 clk_50MHz   =>  s_50MHz,
                 clk_160kHz  =>  s_160kHz,
                 clk_8kHz    =>  s_8kHz);

    --Input Differential Buffer
DifBuf : IBUFDS
        generic map (
            -- Differential Termination
            DIFF_TERM => FALSE,
            -- Low power (TRUE) vs. performance (FALSE)
            IBUF_LOW_PWR => FALSE,
            IOSTANDARD => "DEFAULT")
        port map (O => ibufds_out,      -- Buffer output
                -- Diff_p buffer input
                 I => SignP,
                 -- Diff_n buffer input
                 IB => SignM
                 );
        --SDM's Flip-Flop
DFF1:   entity work.DFlipFlop(Behavioral)
        port map(dataIn =>  ibufds_out,
                 enable  =>  '1',
                 sysclk  =>  s_50MHz,
                 dataOut =>  sdm_out);
```

```
        --Output Buffer
OBUF1:  OBUF
        generic map (DRIVE => 16,
                     IOSTANDARD => "DEFAULT",
                     SLEW => "Fast")
        port map (O => BufOut,      -- Buffer output
                  I => sdm_out      -- Buffer input
                  );

        -- CIC filter instantiation
cic:    entity work.CIC_filter(Behavioral)
        generic map(M => 625)
        port map(x => sdm_out,
                 clkIn => clk,
                 clkOut => s_160kHz,
                 y => cic_out);

        -- Oscillator instantiation
osc:    entity work.oscillator(Behavioral)
        port map(clkIn  =>  s_160kHz,
                 output  => osc_out
                 );

----------------------------------
--        In-phase branch
----------------------------------
        --Mixer
mix1:   entity work.mixer(Behavioral)
        port map(signalIn   =>  cic_out,
                 oscIn      =>  osc_out,
                 trig       =>  s_160kHz,
                 output     =>  mix_out_i);
         --LPF
lpfI:   entity work.lpf(Behavioral)
        port map(inPort =>  mix_out_i,
                 outPort =>  lpf_out_i,
                 clkIn   =>  s_160kHz,
                 clkOut  =>  s_8kHz
                 );

----------------------------------
--        Quadrature branch
----------------------------------
        --Phase-shift (90)
dff2:   entity work.DFlipFlop_16bits(Behavioral)
        port map(dataIn => osc_out,
```

```
                dataOut => shift_ph_out,
                sysclk => s_160kHz
                );

        --Mixer
mix2:   entity work.mixer(Behavioral)
        port map(signalIn   =>  cic_out,
                oscIn       =>  shift_ph_out,
                trig        =>  s_160kHz,
                output      =>  mix_out_q);

        --LPF
lpfQ:   entity work.lpf(Behavioral)
        port map(inPort =>  mix_out_q,
                outPort =>  lpf_out_q,
                clkIn   =>  s_160kHz,
                clkOut  =>  s_8kHz
                );

end Behavioral;
```

D.2 Trigger Generator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;

entity CCU is
    port(clk_in     :   in  std_logic;              --input clock
        clk_50MHz   :   out std_logic;              --50MHz output
        clk_160kHz  :   out std_logic;              --160kHz output
        clk_8kHz    :   out std_logic)              --8kHz Output
end CCU;

architecture Behavioral of CCU is
--output signals
    signal  s_clk_50MHz, s_clk_160kHz, s_clk_8kHz       :   std_logic   := '0';
--counters
    signal  s_count_50MHz, s_count_160kHz, s_count_8kHz :   integer := 1;
begin
--Counter Increment process
cnt8k:      process(clk_in)
            begin
                if(rising_edge(clk_in)) then
                    if(s_count_8kHz = 12500) then --100M/8k = 12500
                        s_count_8kHz <= 1;        --restart counter
```

```
                    else
                        s_count_8kHz <= s_count_8kHz + 1; --increment
                    end if;
                end if;
            end process cnt8k;

 --Output update Process
clk8k:      process(s_count_8kHz)
            begin
                if (s_count_8kHz = 12500) then --if restarting
                    s_clk_8kHz <= '1';          --output pulse
                else
                    s_clk_8kHz <= '0';
                end if;
            end process clk8k;


--Counter Increment process
cnt160k:    process(clk_in)
            begin
                if(rising_edge(clk_in)) then        --100M/160k = 625
                    if(s_count_160kHz = 625) then   --restart counter
                        s_count_160kHz <= 1;
                    else
                        s_count_160kHz <= s_count_160kHz + 1; --increment
                    end if;
                end if;
            end process cnt160k;

--Output update process
clk160k:    process(s_count_8kHz)
            begin
                if (s_count_160kHz = 625) then  --if restarting
                    s_clk_160kHz <= '1';        --output pulse
                else
                    s_clk_160kHz <= '0';
                end if;
            end process clk160k;


--Counter Increment Process
cnt50M:    process(clk_in)
            begin
                if(rising_edge(clk_in)) then
                    if(s_count_50MHz = 2) then          --100M/50M = 2
                        s_count_50MHz <= 1;             --restart counter
                    else
```

```
                    s_count_50MHz <= s_count_50MHz + 1; --increment
                end if;
            end if;
        end process cnt50M;

--Output update process
clk50M:     process(s_count_50MHz)
            begin
                if (s_count_50MHz = 2) then     --if restarting
                    s_clk_50MHz <= '1';         --output pulse
                else
                    s_clk_50MHz <= '0';
                end if;
            end process clk50M;

clk_50MHz <= s_clk_50MHz;
clk_160kHz <= s_clk_160kHz;
clk_8kHz <= s_clk_8kHz;

end Behavioral;
```

D.3 1 bit D Flip-Flop

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Dflipflop is
    port(dataIn :   in  std_logic;
        dataOut :   out std_logic;
        sysclk  :   in  std_logic;
        enable  :   in  std_logic
        );
end Dflipflop;

architecture Behavioral of Dflipflop is

begin

    process(sysclk)
    begin
        if(rising_edge(sysclk)) then
            if(enable = '1') then
                dataOut <= dataIn;
            end if;
        end if;
    end process;
```

```
end Behavioral;
```

D.4 CIC Filter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CIC_filter is
    generic(M    :   integer :=  625);
    port(x       :   in  std_logic;
        clkOut   :   in  std_logic;
        clkIn    :   in  std_logic;
        y        :   out signed(15 downto 0)
        );
end CIC_filter;

architecture Behavioral of CIC_filter is
    --sign extended version of the input
    signal  x_sxt    :   signed(24 downto 0)   :=    (others => '0');
    --x1[n] = x[n] + x1[n-1]
    signal  x1       :   signed(24 downto 0)   :=    (others => '0');
    --output of the downsampler
    signal  y1       :   signed(24 downto 0)   :=    (others => '0');
    -- output of the integrator part (input of the comb)
    signal  cmb_out  :   signed(24 downto 0)   :=    (others => '0');
    --output of the integrator and, thus, of the CIC filter
    signal  int_out  :   signed(24 downto 0)   :=    (others => '0');

begin

--Integrator Process
int:    process(clkIn)
        begin
            if rising_edge(clkIn) then
                if (x='0') then
                    x_sxt <= "111111111100000000000000";    -- -1
                elsif (x='1') then
                    x_sxt <= "000000000010000000000000";    -- 1
                end if;
                x1 <= x_sxt + x1;         --integration
            end if;
        end process;

        cmb_out <= x1;
```

```
--Comb Process
comb:   process(clkOut)
        begin
            if rising_edge(clkOut) then
                int_out <= (cmb_out - y1)/M;        --Comb with division
                y1 <= cmb_out;
            end if;
        end process;

        y <= int_out(15 downto 0);
end Behavioral;
```

D.5 Oscillator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity oscillator is
    port(clkIn  :   in  std_logic;
        output  :   out signed(15 downto 0)
    );
end oscillator;

architecture Behavioral of oscillator is
    --Counter (select signal)
    signal  s_count :   std_logic_vector(1 downto 0) := (others => '0');
begin

--Counter Process
process(clkIn)
begin
    if(rising_edge(clkIn)) then         --Reset select signal
        if (s_count = "11") then
            s_count <= "00";
        else
            s_count <= std_logic_vector(unsigned(s_count) + 1); --increment
        end if;
    end if;
end process;

output <= "0100000000000000" when (s_count = "00") else -- 1 in Q2.14
          "1100000000000000" when (s_count = "10") else -- -1 in Q2.14
          "0000000000000000";                           -- 0 in Q2.14
```

```
end Behavioral;
```

D.6 Mixer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity mixer is
    port(signalIn   :   in  signed(15 downto 0);
         oscIn      :   in  signed(15 downto 0);
         trig       :   in  std_logic;
         output     :   out signed(15 downto 0)
         );
end mixer;

architecture Behavioral of mixer is
    --Result of the multiplication of both inputs (Q4.28)
    signal  s_mult  :   signed(31 downto 0);
begin

process(trig)
begin
    if(rising_edge(trig)) then
        s_mult <= signalIn * oscIn;
    end if;
end process;

output <= s_mult(29 downto 14); --Q2.14

end Behavioral;
```

D.7 Low-Pass Filter with Downsampling

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;

entity lpf is
    port(inPort :   in  signed(15 downto 0);
        outPort :   out signed(15downto 0);
        clkIn   :   in  std_logic;
        clkOut  :   in  std_logic
        );
end lpf;
```

```
architecture Behavioral of lpf is
    --First Integrator output
    signal  s1                      :   signed(22 downto 0);
    --First Delay Output
    signal  s2                      :   signed(22 downto 0) :=  (others => '0');
    --First Comb output
    signal  s3                      :   signed(22 downto 0) :=  (others => '0');
    --Second Integrator output
    signal  s4                      :   signed(22 downto 0) :=  (others => '0');
    --Second Delay Output
    signal  s5                      :   signed(22 downto 0) :=  (others => '0');
    --Second Comb output
    signal  s6                      :   signed(22 downto 0) :=  (others => '0');
    --Integrators output
    signal int_1_out, int_2_out     :   signed(22 downto 0) :=  (others => '0');
    --Combs output
    signal comb_1_out, comb_2_out   :   signed(22 downto 0) :=  (others => '0');
    --Output of the Downsampler
    signal  s_out                   :   signed(22 downto 0) :=  (others => '0');
    --First Delay Constant
    constant    K1                  :   integer :=  80;
    --Second Delay Constant
    constant    K2                  :   integer :=  53;
begin

    --  First delay block instantion
d1: entity work.delay(Behavioral)
    generic map(M => K1)
    port map(inPort =>  s1,
             outPort =>  s2,
             clk     =>  clkIn
             );

    --Second delay block instantiation
d2: entity work.delay(Behavioral)
    generic map(M => K2)
    port map(inPort =>  s4,
             outPort =>  s5,
             clk     =>  clkIn
             );

    --First Integrator
p1: process(clkIn)
    begin
        if(rising_edge(clkIn)) then
            s1 <= s1 + inPort;
        end if;
```

```
    end process p1;

    int_1_out <= s1;

    -- First comb
p2: process(clkIn)
    begin
        if(rising_edge(clkIn)) then
            s3 <= (int_1_out - s2)/K1;
        end if;
    end process p2;

    comb_1_out <= s3;

    --Second Integrator
p3: process(clkIn)
    begin
        if (rising_edge(clkIn)) then
            s4 <= s4 + comb_1_out;
        end if;
    end process p3;

    int_2_out <= s4;

    --Second comb
p4: process(clkIn)
    begin
        if (rising_edge(clkIn)) then
            s6 <= (int_2_out - s5)/(K2);
        end if;
    end process p4;

    comb_2_out <= s6;

    --Downsampling
dff:    entity work.DFlipFlop_23bits(Behavioral)
        port map(dataIn =>  comb_2_out,
                 dataOut =>  s_out,
                 sysclk  =>  clkOut
                 );
    --Output port of the filter
    outPort <= s_out(15 downto 0); --Q2.14

end Behavioral;
```

D.8 Delay Block

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity delay is
    --Delay M samples
    generic(M   :   integer :=  1);
    port(inPort :   in  signed(22 downto 0);
        outPort :   out signed(22 downto 0);
        clk     :   in  std_logic   := '0'
        );
end delay;

architecture Behavioral of delay is
    --Array of M D Flip-Flops
    type delay_block is array (M downto 0) of signed(22 downto 0);
    signal del      : delay_block;

begin
    --First Flip-Flop input is connected to input port
    del(0) <= inPort;
    -- Cascading the M D-type Flip-Flops
G: for I in 0 to M-1 generate
    dffx:   entity work.DFlipFlop_23bits(Behavioral)
            port map(dataIn =>  del(i),
                    dataOut =>  del(i+1),
                    sysclk  =>  clk
                    );
    end generate;

    outPort <= del(M);

end Behavioral;
```

D.9 23 bits D Flip-Flop

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DFlipFlop_23bits is
    port(dataIn :   in  signed(22 downto 0);
        dataOut :   out signed(22 downto 0);
        sysclk  :   in  std_logic
```

```
        );
end DFlipFlop_23bits;

architecture Behavioral of DFlipFlop_23bits is

begin
process(sysclk)
begin
    if(rising_edge(sysclk)) then
            dataOut <= dataIn;
    end if;
end process;

end Behavioral;
```

D.10 16 bits D Flip-Flop

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DFlipFlop_16bits is
    port(dataIn :   in  signed(15 downto 0);
        dataOut :   out signed(15 downto 0);
        sysclk  :   in  std_logic
        );
end DFlipFlop_16bits;

architecture Behavioral of DFlipFlop_16bits is

begin
process(sysclk)
begin
    if(rising_edge(sysclk)) then
            dataOut <= dataIn;
    end if;
end process;

end Behavioral;
```

D.11 Constraint File

```
## Clock signal
##Bank = 35, Pin name = IO_L12P_T1_MRCC_35, Sch name = CLK100MHZ
set_property PACKAGE_PIN E3 [get_ports clk]
set_property IOSTANDARD LVCMOS25 [get_ports clk]
```

```
create_clock -period 10.000 -name sys_clk_pin -waveform
            {0.000 5.000} -add [get_ports clk]


##Pmod Header JC
##Bank = 35, Pin name = IO_L23P_T3_35, Sch name = JC1
set_property PACKAGE_PIN K2 [get_ports signP]
set_property IOSTANDARD LVDS_25 [get_ports signP]
##Bank = 35, Pin name = IO_L21P_T3_DQS_35, Sch name = JC4
set_property PACKAGE_PIN J4 [get_ports bufOut]
set_property IOSTANDARD LVCMOS25 [get_ports bufOut]
##Bank = 35, Pin name = IO_L23N_T3_35, Sch name = JC7
set_property PACKAGE_PIN K1 [get_ports signM]
set_property IOSTANDARD LVDS_25 [get_ports signM]
```

D.12 Block Diagrams

D.12.1 Downconverter - Top-level file

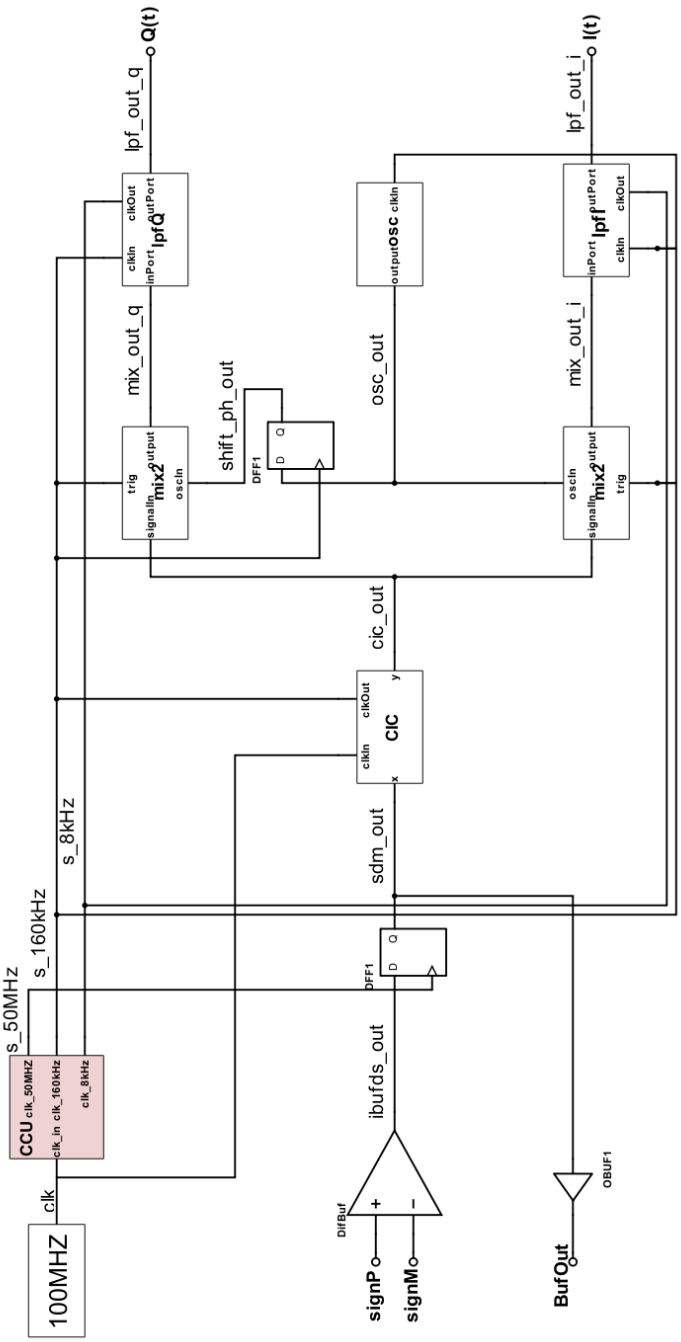


Figure D.1: Block diagram of the downconverter implemented in VHDL.

D.12.2 CIC filter

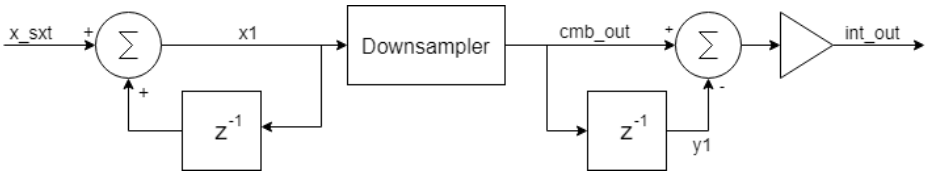


Figure D.2: Block diagram of the CIC filter implemented in VHDL.

D.12.3 Low-pass Filter with Downsampler

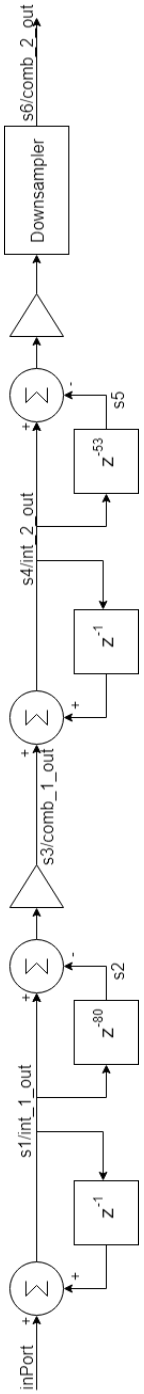


Figure D.3: Block diagram of the LPF filter implemented in VHDL.

Bibliography

[1] https://www.youtube.com/watch?v=A8lztr1tu4o.

[2] Rui F. Cordeiro. Digital Beam-steering in a Parametric Array. Master's thesis, University of Aveiro, 2012.

[3] World health organisation's international classification of diseases, 11$^{\text{th}}$ edition. 2018.

[4] http://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment.

[5] https://brailleworks.com/free-white-cane/.

[6] *Football Five-A-Side Laws - B1 Category.* International Blind Sports Federation, 2017.

[7] http://www.ibsasport.org/photos/36/football-5-a-side-blind-football-at-the-rio-2016-paralympic-games.

[8] Corinna M Bauer, Gabriella V Hirsch, Lauren Zajac, Bang-Bon Koo, Olivier Collignon, and Lotfi B Merabet. Multimodal mr-imaging reveals large-scale structural and functional connectivity changes in profound early blindness. *PloS one*, 12(3):e0173064, 2017.

[9] Lore Thaler, Galen M Reich, Xinyu Zhang, Dinghe Wang, Graeme E Smith, Zeng Tao, Raja Syamsul Azmir Bin Raja Abdullah, Mikhail Cherniakov, Christopher J Baker, Daniel Kish, et al. Mouth-clicks used by blind expert human echolocators–signal description and model based signal synthesis. *PLoS computational biology*, 13(8):e1005670, 2017.

[10] Bo N Schenkman and Mats E Nilsson. Human echolocation: Blind and sighted persons' ability to detect sounds recorded in the presence of a reflecting object. *Perception*, 39(4):483–501, 2010.

[11] Lore Thaler, Stephen R Arnott, and Melvyn A Goodale. Neural correlates of natural human echolocation in early and late blind echolocation experts. *PLoS one*, 6(5):e20162, 2011.

[12] Pedro R. M. Rosa. Bengala de apoio a cegos com deteção de buracos. Master's thesis, University of Aveiro, 2009.

[13] Adrien Brilhault, Slim Kammoun, Olivier Gutierrez, Philippe Truillet, and Christophe Jouffrais. Fusion of artificial vision and gps to improve blind pedestrian positioning. In *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, pages 1–5. IEEE, 2011.

[14] http://av.loyola.com/products/audio/pdf/audiobeam.pdf.

[15] J.G. Proakis and D.G. Manolakis. *Digital Signal Processing*. Pearson Prentice Hall, 2007.

[16] S. Park and Motorola. *Motorola Digital Signal Processors: Principles of Sigma-delta Modulation for Analog-to-digital Converters*. Motorola, 1993.

[17] S. Haykin. *Digital Communication Systems*. Wiley, 2013.

[18] Walt Kester. *Understand SINAD, ENOB, SNR, THD, THD+ N, and SFDR so you don't get lost in the noise floor*. Citeseer, 2009.

[19] E. Janssen and A. van Roermund. *Look-Ahead Based Sigma-Delta Modulation*. Analog Circuits and Signal Processing. Springer Netherlands, 2011.

[20] A.B. Carlson and P.B. Crilly. *Communication System*. Tata McGraw-Hill Education, 2010.

[21] *7 Series FPGAs SelectIO Resources - User Guide*. Xilinx, Inc., 2011.

[22] *Nexys 4 FPGA Board Reference Manual*. Digilent, Inc., 2016.

[23] The effective number of bits (ENOB) of my R&S digital oscilloscope, 2018.

[24] *MCP3201 Datasheet - 2.7V 12-Bit A/D Converter with SPI Serial Interface*. Microchip Technology, Inc., 2007.

[25] Barry D Van Veen and Kevin M Buckley. Beamforming: A versatile approach to spatial filtering. *IEEE assp magazine*, 5(2):4–24, 1988.

[26] P.S. Naidu. *Sensor Array Signal Processing*. Taylor & Francis, 2000.

[27] S. Haykin and K.J.R. Liu. *Handbook on Array Processing and Sensor Networks*. Adaptive and Cognitive Dynamic Systems: Signal Processing, Learning, Communications and Control. Wiley, 2010.

[28] https://en.wikipedia.org/wiki/Frequency_mixer.

[29] L.W. Couch. *Digital and Analog Communication Systems*. Prentice-Hall international editions. Pearson, 2013.

[30] http://www.dsplog.com/2007/07/01/example-of-cascaded-integrator-comb-filter-in-matlab/.

[31] D.L. Perry. *VHDL: Programming by Example*. McGraw-Hill Education, 2002.

[32] *MA40S4S/MA40S4R Datasheet, year=2017, publisher=Murata Manufacturing Co.*

[33] J.S. Bendat and A.G. Piersol. *Engineering applications of correlation and spectral analysis*. J. Wiley, 1993.

[34] M.H. Hayes. *Statistical digital signal processing and modeling*. John Wiley & Sons, 1996.

[35] S.L. Marple. *Digital Spectral Analysis with Applications*. Prentice-Hall, 1987.

[36] R.A. Serway and J.W. Jewett. *Physics for Scientists and Engineers*. Brooks/Cole, 2003.

[37] *Application Note 49 - Theory of Lloyd's Mirror Interferometer*. Newport Corporation, 2012.