Technical University of Crete

School of Electrical & Computer Engineering

Reconfigurable Logic Based Second Generation Digital Sound Processing and Enhancement System

Δεύτερης Γενιάς Σύστημα Επεξεργασίας και Βελτίωσης Ψηφιακού Ήχου, Βασισμένο σε Αναδιατασσόμενη Λογική

Iraklis-Taxiarchis Kokkinis

Committee

Supervisor:	Professor Apostolos Dollas
Committee Member:	Professor Michalis Zervakis
Committee Member:	Associate Professor Eftychios Koutroulis

Περίληψη

Όσον αφορά τον ψηφιακό ήχο, για μεγάλο χρονικό διάστημα η πιο κοινή μορφή αποθήκευσης και διανομής μουσικής ήταν η ποιότητα CD με ρυθμό δειγματοληψίας 44,1 kHz και βάθος πληροφορίας τα 16 bit. Για πολλούς λάτρεις της μουσικής, η ποιότητα του CD δεν μπορεί να προσφέρει την καλύτερη δυνατή ακουστική εμπειρίας, υποστηρίζοντας την ανωτερότητα των μορφών ήχου υψηλής ανάλυσης. Δεν υπάρχει προς το παρόν κοινή ετυμηγορία, σχετικά με τις ιδιότητες και τα πλεονεκτήματα των μορφών ήχου υψηλής ανάλυσης. Αυτή η διπλωματική εργασία στοχεύει στη βελτίωση των τυπικών αρχείων ποιότητας CD με την εφαρμογή της μαθηματικής μεθόδου cubic spline interpolation σε ασυμπίεστο αρχείο .wav, βασισμένη σε προηγούμενη εργασία από τον Μουρτζανό Τριαντάφυλλο [1].

Έχοντας μελετήσει τις δυνατότητες του ανθρώπινου αυτιού και τον ρόλο που έχει η ψυχοακουστική στην ακουστική εμπειρία, καθώς και τα μαθηματικά πίσω από την ηχητική μηχανική και τις μεθόδους interpolation, τα μαθηματικά μοντέλα των Linear και cubic spline interpolation αναδημιουργήθηκαν, χρησιμοποιώντας τη Matlab, όπως παρατέθηκαν από τον Μουρτζανό Τριαντάφυλλο, που χρησιμοποίησε αριθμητική κινητής υποδιαστολής. Αυτά τα μοντέλα στη συνέχεια μετατράπηκαν σε μια αποδοτικότερη από μεριάς υλικού προσέγγιση, χρησιμοποιώντας ένα παράθυρο τεσσάρων δεδομένων και αριθμητικής σταθερού σημείου. Στη συνέχεια συγκρίθηκαν τα νέα μοντέλα, με μια ποικιλία αρχείων ήχου ως εισόδους, κρίνοντας τα αποτελέσματα από τα φασματογραφήματά τους, τις κυματομορφές και την ακρόαση των παραγόμμενων αρχείων WAVE, επιβεβαιώνοντας τη καταλληλότητα της αριθμητικής σταθερού σημείου για τα μοντέλα.

Στην συνέχεια, σχεδιάστηκε μια υλοποίηση υλικού που στοχεύει ένα Zedboard FPGA, χρησιμοποιώντας τα Vivado HLS, Vivado και Vitis, που αναπτύχθηκαν από την Xilinx. Η προκύπτουσα εφαρμογή συγκρίθηκε με μια προσέγγιση λογισμικού και τα αποτελέσματα συγκρίθηκαν με το μοντέλο σταθερού σημείου που αναπτύχθηκε στο Matlab.

Abstract

When it comes to digital audio, for a long time the most common format to store and distribute music was the CD quality of 44.1 kHz sampling rate and 16 bit depth. For many music enthusiasts, the CD quality is unable offer the best possible listening experience possible, supporting the superiority of high-resolution audio formats. The jury is still out, on the qualities and advantages of high resolution audio formats. This thesis diploma aims to improve standard CD quality files with the application of the cubic spline interpolation mathematical method on uncompressed .wav file, based on previous work done by Triantafillos Mourtzanos[1].

Having studied the capabilities of the human ear and the role played by psychoacoustics on the auditory experience and the mathematics behind audio engineering as well as interpolation, the mathematical models of linear and cubic spline interpolation were recreated, using Matlab, as presented by Triantafillos Mourtzanos, which utilized floating point arithmetic. These models were then converted to a more hardware efficient approach, using a four data point window of inputs and fixed-point arithmetic. The new models were then compared, with a variety of audio files as inputs, judging the results by their spectrograms, waveforms and the audibility of the resulting WAVE files, confirming the viability of fixed point arithmetic for the models.

As a follow-up a hardware implementation was was designed targeting a Zedboard FPGA, using Vivado HLS, Vivado and Vitis, developed by Xilinx. The resulting implementation was compared to a software adaptation and its results were compared to the fixed point model developed in Matlab.

Acknowledgments

First of all I would like to thank my family, as through their toil I was able to write this Thesis in the first place. Then I would like to thank my supervising professor Apostolos Dollas, for providing the basic idea of the thesis, his numerous contributions through this work, and all in all, his eagerness to assist in the completion of the thesis any way possible. I would also like to thank Pavlos Malakonakis for providing crucial assistance with the hardware design tools used throughout this work. I would also like to thank the other members of the committee, Eftychios Koutroulis and Michalis Zervakis for their role in my graduation. Finally, I would like to thank all the my fellow students, with whom I had the honour of studying with, over the years.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

In the last few decades, digital sound has dominated the music industry, due to its low recording cost, ease of distribution in an increasingly digitalized market and increased audio processing capabilities. As a result digital sound formats became the most prevalent in our everyday lives, with a large library of great works being stored first in CDs and later on, with the advent of the internet, converted to the Mp3 format making music accessible to a level never before thought possible.

The success of the Mp3 format can be credited to its small space requirement, accomplished by the omission of sound undetectable by the human ear, compared to other compression methods. As such Mp3 is the dominant format in platforms like Spotify, and Youtube, the two most popular and accessible platforms for music.

However, when performing lossy audio encoding, such as creating an MP3 data stream, there is a trade-off between the amount of data generated and the sound quality of the results[34]. This loss of data leaves something to be desired by more demanding listeners, making CDs a better option, as far as audio quality is concerned. Unfortunately, CDA (the CD standard) is it self a lossy compression method, due to the nature of sampling[2].

Consequently, listeners well versed in music, tend to prefer LP (Long Playing) records as a medium of enjoying their favourite music. LPs are an analogue sound storage mediums and thus they maintain the sound and feel of live music to a greater extend. This statement remains the source of heated debate, leading to the emergence of multiple higher frequency formats, trying to abridge the gap between analogue and digital audio, providing more samples.

1.2 Objective

Previous, work on the subject used and compared two interpolation methods, linear interpolation and cubic spline interpolation in order to increase the quality of music tracks, extracted from CDs using an FPGA. Results were encouraging, achieving an increase in audio quality, mathematically and acoustically. Thus, in this thesis, a more hardware efficient and less computation intensive solution was explored and documented, using fixed point arithmetic and windows of four samples of input data to achieve similar results.

The experiments present in this thesis are conducted using Matlab, in order to estimate the effectiveness of the proposed solution. An argument could be made for using C/C++ for this

purpose since Matlab is optimized for floating point arithmetic calculations. However Matlab allows for easy demonstration of results, using plots, spectrograms, resampling and other functions, which are instrumental to the experiments.

First a floating point model was created to recreate the results of the aforementioned work, with the extra addition of a data window, as input. After the code for fixed point cubic spline interpolation was developed in line with the previously mentioned constraints, with the objective of four times the frequency of CDs and 16 extra bits of data per sample, bringing the final quality up to 176,4 kHz and 32 bits of depth.

This format may not be in line with the standard of the music industry of: 192kHz and 24 bits, but it does not diverge significantly from them, and is often preferred by audio engineers. Finally two FPGA designs were created, one with the purpose of data-point recovery and the other is focused on upsampling to the aforementioned specifications, using the Xilinx HLS developing kit. The algorithms tested with Matlab, was translated to HLS friendly C++ code, in order to be downloaded to a Zedboard FPGA.

1.3 Contribution of the thesis

The present thesis is proposing a different method of providing superior digital audio quality, closer to the expectations of experienced listeners, using a design tailored to the strengths of FPGA technology.

The IP developed for the purposes of this work, is a resource inexpensive and timing efficient module, that can be easily integrated in FPGA designs targeting relatively inexpensive platforms but all the while ensuring high performance and minimal power consumption.

Additionally, this thesis tries to establish the fact that there is not always need for high arithmetic precision, provided by arithmetic models such as floating-point, as far as audio engineering is concerned. The fixed point models explained on chapter 4, provide substantial results while avoiding the intricacies of implementing a floating point arithmetic model.

Finally, the thesis explores the viability of cubic spline interpolation as a standalone interpolation method for data-point recovery or upsampling applications.

1.4 Structure of the Thesis

The structure of the thesis is as follows:

In chapter 2, general information about audio engineering, along with some of its basic principles is explained, along with an explanation of the human auditory experience and

factors that affect it. Finally chapter 2 presents examples of similar work on the field of audio interpolation.

Chapter 3 presents the theoretical and mathematical framework of linear and cubic spline interpolation methods, in order to provide a basis for the algorithms explored and developed in this thesis.

Chapter 4 explains the design process of a mathematical model destined for hardware implementation. The process followed in that chapter was to recreate the original model presented in the first generation of this work, followed by its conversion to a four data point input window utilizing fixed-point arithmetic. Then the models are compared using spectrograms waveforms and auditory tests.

In chapter 5 the development of a hardware design is presented along with the interfacing methods utilized to create an efficient embedded FPGA design. This design is then compared to a C version of the mathematical model. Finally the results are compared to the ones given by the mathematical model created in Matlab.

Chapter 7 details a summary of this thesis along with commentary, conclusions, and possibilities for future research and development.

Chapter 2:
Relative Research

2.1 Introduction to Music and Sound

Music is an art form. Meaning some of its main purposes, among others, include the expression of human emotion and the exoneration of beauty. The medium of music is sound, subsequently in order to properly understand and process music one must understand sound, as well as human hearing, by virtue of being the sense by which humans perceive sound.

2.1.1 Analogue and Digital Audio

Sound is an analogue mechanical wave. It can be mathematically represented by function of one independent variable, specifically a function of sound pressure in correlation to time[3]. The diversity of sounds in an audio signal is not a result of the mere value of sound pressure, but it is dependent on the frequency of the waveform. For example, striking the string of a guitar causes it to oscillate striking the neighbouring molecules of air to oscillate at the same frequency as the string. The easiest methods of recording, processing and distributing sound waves involves using the Nyquist sampling theorem in order to produce Digital audio files, whose quality depends on the particulars of each method.

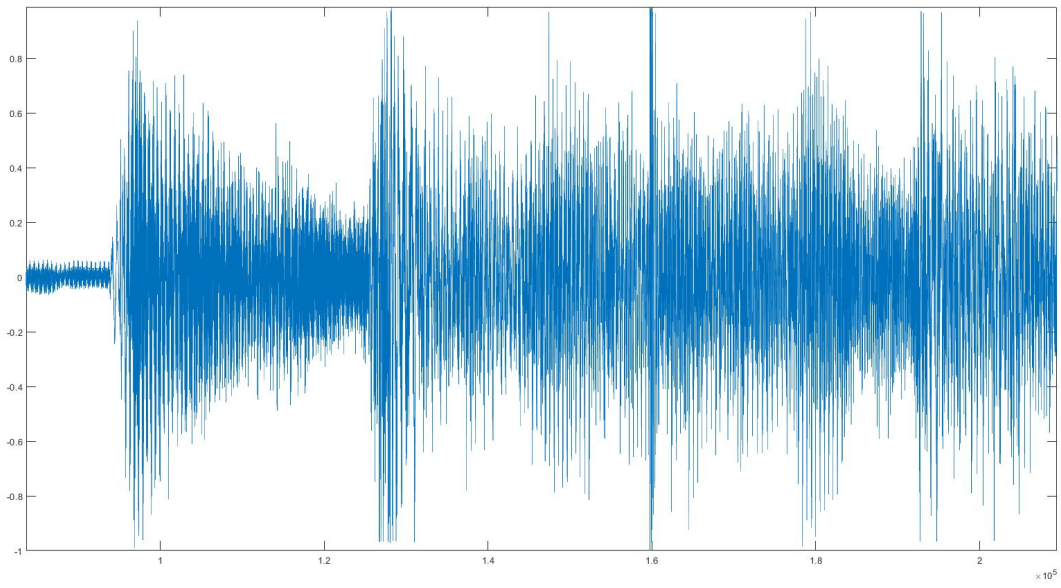

Figure 2.1: An example of an audio waveform

For analogue audio signals the variable of time is continuous. As a result there is an interrupted stream of sound data, for a listener to experience. Analogue recordings are more accurate and natural sounding due to changes in air pressure being captured exactly as recorded[22].

In contrast, for digital audio signals the variable of time is distinct, meaning that audio signals have distinct values of sound pressure, that belong to a specific and finite set represented by bits. However the gaps in the signals only have a small interval between them, undetectable to humans. Unfortunately, to trained listeners, digital might, sometimes, sound unnatural, too clean and far too perfect[22], forgoing the "warmth" of analogue sound.

2.1.2 Converting Analogue to Digital

Most commonly a digital audio signal is procured by inputting an analogue audio signal to an ADC(Analogue to Digital Converter). The ADC, most commonly, is going to apply two processes to the analogue signal: sampling and quantization [23]. applying sampling on an analogue signal, results in values evenly distributed through time. In order to procure accurate results, so that the resulting sequence of samples uniquely defines the analogue signal, the proper sampling rate must be selected[4]. The proper rate is given by the Nyquist sampling theorem.

2.1.3 Nyquist Sampling Theorem

The Nyquist sampling theorem states that: Any signal, regardless of its length, can be considered as consisting of sinusoidal components. Therefore they possess characteristics of periodic functions such as frequency(f). Being consisted of multiple such signals means that there is a frequency band(bandwidth) within a given function. The required frequency band is directly proportional to the signalling speed (f_s), and in fact needs to be equal to at least two times the sampled signal's bandwidth, in order to maintain all of the signal's information[5]. Which ultimately means that:

$$f_s \geq 2W$$

In case the sampling rate does not satisfy the above expression, there will be overlapping samples in the produced signal. This phenomenon results in the distortion of the audio signal resulting in what is called, aliasing. In many instances, before sampling, anti-aliasing, low-pass filters are used to further limit the bandwidth of the input signal[**6**]. When sampling audio signals, a low pass filter, with a cut-off frequency of 20kHz is used since the preferred

sampling rate of audio signals of 44.1 kHz, already satisfies the Nyquist sampling rate, since the absolute width of human hearing is 20Hz to 20kHz, with frequencies between 2 kHz and 5 kHz being the most sensitive to human ears.

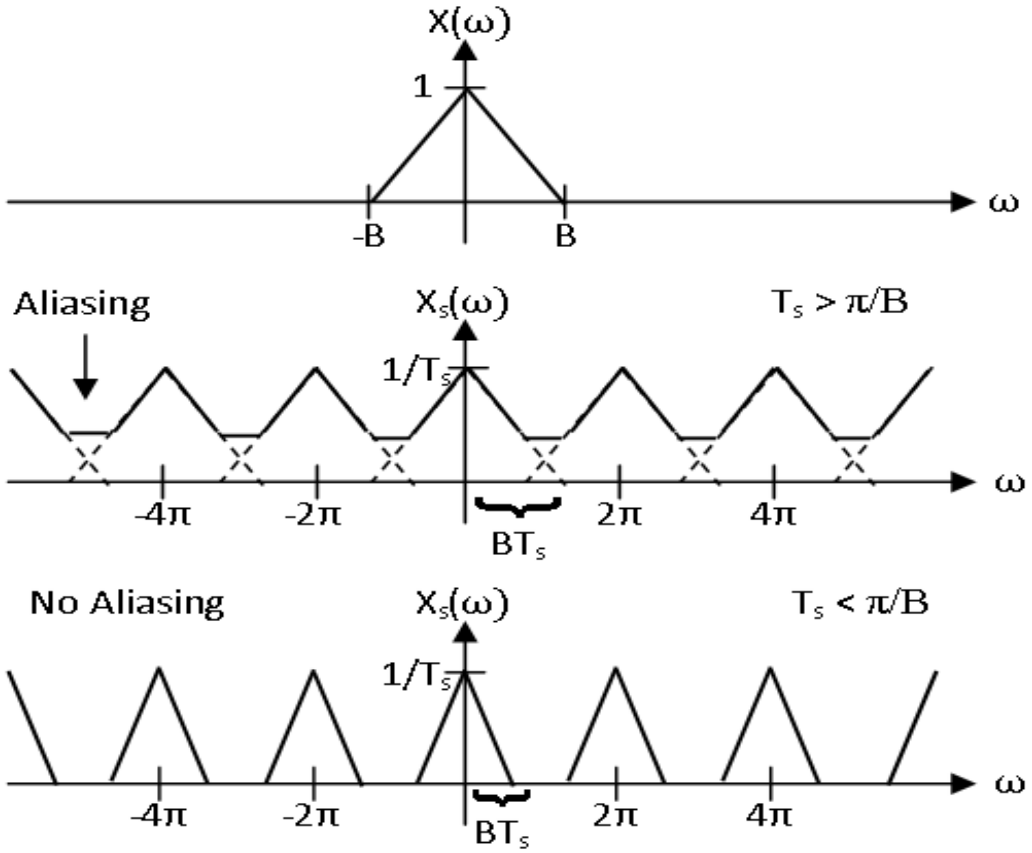

Figure 2.2: Example of aliased and non aliased output signals(image source: [35])

The typical method of digital representation of sampled analogue audio signals is Pulse-code modulation (PCM). It is the standard form of digital audio in computers, compact-discs, WAV files and other digital audio applications.

Common sample depths for PCM are 8, 16 or 24 bits per sample, with a sampling frequency depending on the desired format (e.g. CDs have a sampling frequency of 44.1 kHz, while DVDs have a frequency of 48 kHz)[33]. It should be noted here that the wikipedia article includes a small mistake as it states that 20 bits per sample are possible. However this is technically incorrect as the bit-width needs to be a multiple of 8.

In a PCM stream, the amplitude of the analogue signal is sampled regularly at uniform intervals, after which quantization must be applied.

2.1.4 Quantization

An analogue signal can, in theory, receive infinite amplitude values even if those values are within a finite range. If a signal is sampled, the potential values of each sample are also infinite. However human hearing can detect amplitude differences only if they are greater than a given value. So in order to procure a digital signal that is similar to its analogue counterpart, the difference in amplitude must remain undetected to the listener.[6]. In order to satisfy this constraint, a processes known as Quantization is applied to the samples of the analogue signal.

Quantization is the process by which the samples of a given analogue signal receive amplitude values from within a finite set of values. The set of possible amplitude values is defined by the bit depth. For example, 8-bit quantization has 2^8 = 256 possible values, 16 bit quantization has 2^{16} = 65,536 possible values, and so on[24]. The values are chosen by the quantizer based on the input voltage it receives, locating the appropriate quantization interval wherein it resides. It should be noted that the intervals at the extreme ends have no upper boundary[7].

After the quantization process is complete, the series of produced values must be represented in a format compatible with the targeted digital equipment. This processes is called encoding. During encoding, the series of bits is split into words depending on the desired bit depth.

Quantization is obviously quite a necessary, process. However unlike sampling, it is irreversible and introduces a new type of error, quantizing error, which ultimately means, that some of the information of the original signal is irrecoverable.

2.1.5 Quantizing error and Dithering

Quantization, introduces a new type of error, known simply as quantizing error, which is defined as the difference between the analogue, input signal and the quantized output signal [4]. If the input is represented by m and the output by v, the quantizing error(q) for any given time, where v is defined, is given by:

$$q = m - v$$

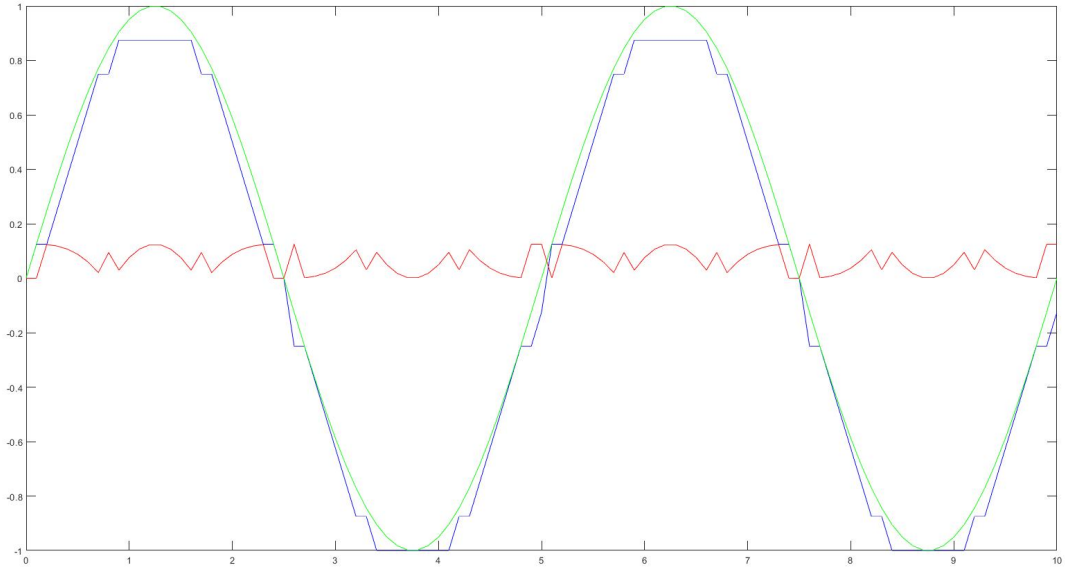

Figure 2.3: In this figure the signals m(green) and v(blue) are shown in relation to each other and the resulting error q(red).

The higher the signal level, the more the signal is distorted by the quantizing error, effectively functioning as noise, since the signal is more strongly correlated with the error. In the music industry the main method to decorrelate the error from the signal, is dither[7].

Dither makes the quantization process unpredictable and gives the system a noise floor. In digital audio systems, a dither signal is added to the signal prior to quantization and no attempt is made in the DAC to subtract it. The effect on the output signal is a slight reduction to the signal to noise ratio attainable by other dithering methods, however no non-linearities are introduced, making the noisier output an acceptable trade off. The unpredictability of quantization results from the fact that by the addition of the dithering function, the sample find their quantizing intervals in different places[7].

Obviously, dithering does not remove the quantizing error, but rather mitigates it, converting the unacceptable sounding distortion of simple quantization, into broadband noise which is more benign to the human ear[7].

2.2 Human Hearing

The aforementioned observations of sonic behaviour and the techniques derived from them raise the question of how exactly does the human ear function and how our perception of sound is affected by differences between the parameters of each sound wave. The structure of the ear is one of the prime factors that determine auditory perception.

The visible part of the ear also referenced as outer ear is named pinna. The pinna leads to the auditory canal, also called meatus, which leads to the eardrum or tympanic membrane. The area beyond the eardrum is called inner ear which contains the hidden parts of the ear. The pinna acts as as a sound collector from the outside world, reflecting the wave to the auditory canal. The auditory canal guides a sound wave, which is proportional to the intensity of the original sound wave, to the eardrum. Three little bones in the air-filled inner ear, which are attached to the eardrum, excite vibrations in the cochlea, which is a liquid-filled part of the inner ear. With the vibrations transmitted in the cochlea, the nerves behind the cochlea are excited, converting the sound wave into nerve impulses, which will be transferred to the brain[8].

The sound wave that reaches the cochlea, is affected by the very structure of the auditory system, and even the head and shoulders. For example the guiding effect of the convolutions of the pinna increases with increasing frequency. The effects of the aforementioned structures becomes significant, however, only when their size is comparable to the wavelength of the sound wave[8].

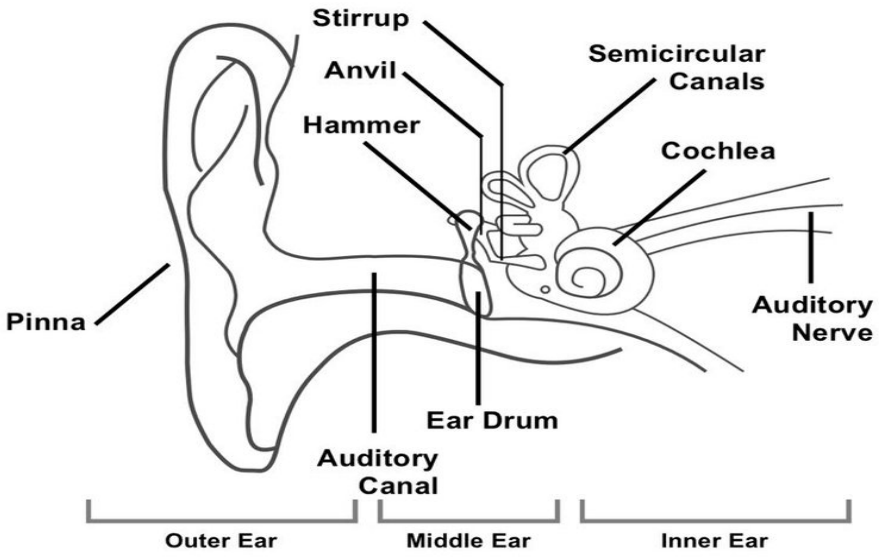

Figure 2.4: The structure of the human ear (image source:[36])

Sound can consist of multiple frequencies and the difference between these frequencies, affects how the brain interpretes it. This frequency analysis is performed by the basilar membrane of the cochlea. For example a sound consisting of two frequencies that are not widely separated, we perceive it as beats. If they differ by a few cycles we hear a single rising and falling sine wave[8].

It is even possible to mask a frequency with another frequency that is close to it by adjusting its intensity. The bandwidth within which it is possible to mask a frequency is called critical bandwidth. Outside the boundaries of the critical bandwidth it is impossible for the frequencies to mask one another or even interact, thus they are heard separately [8].

2.2.2 Human Hearing Frequency Range

Audio signals such as music, are limited in the range of 20 Hz to 20 kHz, which is the absolute border of human hearing, in a non-laboratory environment, with the ear being most sensitive in the 2 to 5 kHz range. The upper end of the range degrades with age and drops to about 16 kHz. This drop in bandwidth is attributed to the various sonic stimuli a human ear is subjected to, as they eventually damage the more fragile cells in the cochlea responsible for the perception of higher frequencies. These cells are irreplaceable. Loudness also affects the impact of sound to the ear, meaning that the effect of sound to the human ear is determined by a function of loudness and frequency as represented in the next graph (Figure 2.5). The range of loudness appears to be as low as 0 dBs and as high as 80dB. However higher values of the decibel scale are still audible but can potentially cause permanent damage and even pain to the ear.

The aforementioned values are not absolute, as there are many different attributes that must be accounted for. Such attributes include genetics, living standards and condition in tandem with the person's habitat and environment, combined with the fact that not every aspect of neither genetics nor hearing is understood perfectly to this day.

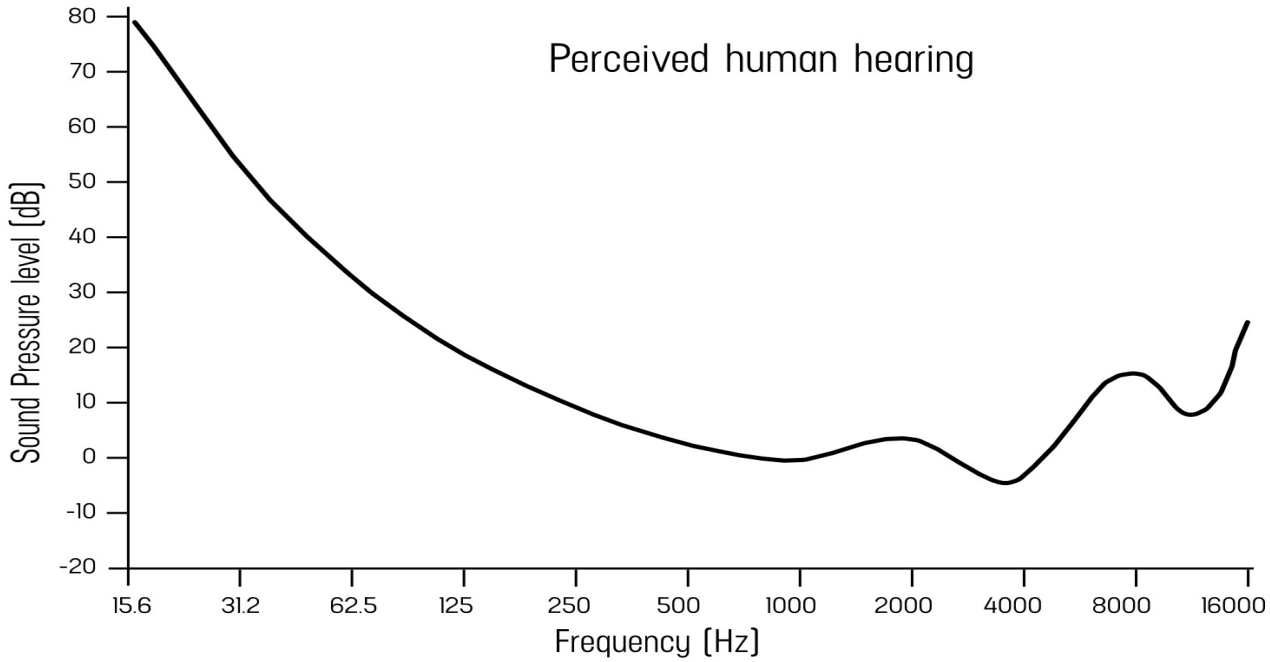

Figure 2.5 : Perceived Human Hearing (image source:[37])

2.2.3 Psychoacoustics

Psychoacoustics is the field of study which involves the scientific of sound perception, it is an interdisciplinary field which includes psychology, acoustics, electronic engineering, physics, biology, physiology, and computer science. Its contribution to this study concerns the different ways, different people, depending on their genetics, experiences and intimacy with music, impact their perception of it. As it was previously stated, sound perception is a combination of mechanical wave physics and a series of sensory and perceptual events.[25]

Psychoacoustics are often applied in computerized audio. For instance, the lossy mp3 audio compression method relies on psychoacoustics to maintain the illusion of untampered audio while drastically increasing information loss. Other applications include the masking of frequencies that audio producers want to de-emphasize, low quality audio systems etc.

2.2.4 The Mayer-Moran Experiment [9]

As it was stated in the comparison between digital and analogue, there had always been a debate about wether or not the standard 16-bit/44.1-kHz CD pulse-code modulation format is sufficient for a complete auditory experience. Furthermore, the perception of audio quality seems to be non subjective. The Mayer-Moran experiment was an attempt to answer the question: Does high resolution audio provide any augmentation to the auditory experience, or is it excessive with no significant benefits?

The experiment is a blind comparison between SACD(Super Audio CD)/DVD-A formatted audio, which utilizes a frequency of 96 kHz and increased word length, claimed to offer superior-sounding playback, and standard CD audio. For about a year ABX tests were conducted with the participants being of various backgrounds but also including people with significant musical experience such as audio engineers and students of musical technology.

An ABX test is simply conducted by presenting the subject with two choices of sensory stimuli to identify detectable differences between them. There are two known samples, sample A, the first reference, and sample B, the second reference, followed by one unknown sample X that is randomly selected from either A or B. The subject is then required to identify X as either A or B. If X cannot be identified reliably with a low probability value in a predetermined number of trials, then it cannot be proven that there is a perceptible difference between A and B. All in all the ABX test answers whether or not, under ideal circumstances, a perceptual difference can be found between the compared samples [26].

The subjects' upper hearing limits have been measured to spot any correlation between this parameter and the audibility of differences. The source audio is given by a SACD/DVD-A player. The CD quality signal is simulated by using a high quality CD recorder as a bottleneck for the high resolution signal. The levels of both channels were matched to within 0.1 dB. Audio switching was handled by an ABX CS-5 double-blind comparator. Just before the playback begins the signal goes through a pee-Amplifier.

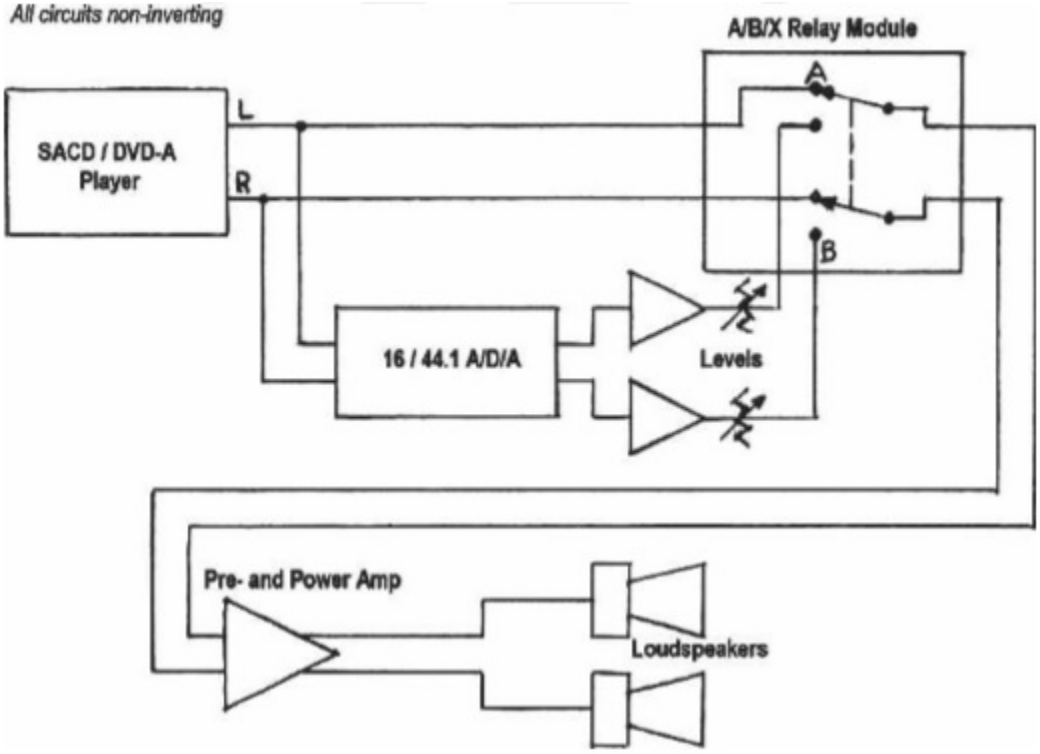

Figure 2.6: The experiment's setup(image source:[9])

2.3 Expressing Audio using Mathematics

As was already stated sound waves are sinusoidal in nature, meaning that they can be represented, mathematically, as a linear correlation of sine waves. The complexity of sound waves and music in particular, seems far removed from a linear system. This can be explained by the fact that sound propagation in air at intensities encountered in musical performances is a linear phenomenon, as are the vibrations of strings and columns of air, even the vibrations along the cochlea of the ear ca be approximated by linear systems[8].

2.3.1 Linear Systems

A linear system is a system whose behaviour is described by a linear differential equation or by a linear partial differential equation. In such an equation constant times partial derivatives with respect to time and space equals 0 or to an input driving function. For example given input signals labeled In1 and In2 which produce output signals labeled Out1 and Out2 respectively, the combined input In1+In2 should produce an output equal to Out1+Out2 [8].

2.3.2 Sinusoidal Waves

A true sine wave lasts forever, identical periods are being repeated across past and future. Its characteristics are given by the following three values: Amplitude(A), frequency(f) and phase(p), represented as:

$$Wave = A * \sin(2 * pi * f * t + p)$$

Amplitude represents the highest possible value the wave can reach, frequency represents how often the wave oscillates, measured in Hertz (cycles per second or s^{-1}) and finaly the phase describes when the wave reaches its peak amplitude.

It is possible to express sine waves in terms of relative amplitudes, this is achieved by converting the relative value to decibels (dB). Supposed two sinusoidal waves with the first have an amplitude of A1 and the second having an amplitude of A2. The relationship between the two vibrations, in dBs is given by:

$$20 * \log(A1/A2)$$

A musical sound wave can be regarded as the summation of different sinusoidal waves as as stated above the musical instruments, the propagation of their oscillations through the air and into the human ear is a linear system. Meaning that mathematical processes like interpolation can be entrusted and utilized to produce accurate results.

2.4 Oversampling

Oversampling means using a sampling rate which is substantially greater than the Nyquist rate thus as already explained above not required to obtain a given signal. On the other hand oversampling allows a given signal quality to be reached without the need for expensive recording equipment. If bandwidth is replaced by sampling rate and SNR is replaced by a function of word length, the same must be true for a digital signal. Thus raising the sampling rate potentially allows the word length of each sample to be reduced without information loss. Information theory predicts that if an audio signal is spread over a much wider bandwidth, oversampling can be used to achieve higher SNR for a given demodulated signal than that of the channel it passes through whether or not the system is digital or analogue. Neither sampling theory nor quantizing theory require oversampling to be used to obtain a given signal quality[7].

2.5 Similar work

There are multiple interpolation methods used for different purposes. The most common purpose of interpolation applications appears to be the recovery of lost data rather than utilizing it for higher quality upsampling.

2.5.1 Signal Restoration using a Gabor Regression Model

One application for signal restoration is called: Interpolation of missing data values for audio signal restoration using a Gabor regression model, which aims at the problem of missing data interpolation over repeated short gaps in audio signals, which for the purposes of their application, are assumed to be known a priori. The method is a formalization of the e overlap and add method commonly used for audio signal analysis and synthesis [10].

Garbor analysis is a branch of mathematical theory, which can be used when digital audio signals are processed. Gabor analysis is based on the idea of representing arbitrary signals of finite energy in terms of building blocks which have a well-defined “center of gravity” in a time-frequency sense. Instead of thinking of cutting a signal into pieces, Fourier analysing it and putting it back together after some processing, the Gabor approach thinks of a signal being projected onto basic functions, which are concentrated in certain regions of the time-frequency plane.[11]

The model first divides the underlining audio signal into overlapping segments via the multiplicative action of a symmetric window g whose effective size is chosen as a function of the analysis window length so that it lies in the range of 15–40 ms, depending on the time-varying nature of the audio signal class under consideration. The analysis coefficients are calculated as inner products of the original audio signal and modulated versions of some

chosen analysis window. Most audio signal processing approaches begin with the Gabor transform of the noisy data, given by $G* y$ according to the chosen Gabor system. Here, however, the synthesis coefficient representation is directly estimated. Finally, Bayesian estimation is used for the Gabor regression model, in order to calculate the noise variance appearing in the equations. [10].

Overall the researchers conclude that, the suitability of a Gabor regression model is evidenced by the fact that promising restorations, from both the objective and subjective points of view may be obtained even in cases where over 35% of the data values are missing.

2.5.2 Interpolation of Audio Signals Using Linear Prediction

This application uses linear prediction in sinusoidal models in order to achieve long interpolation. The sinusoidal model is used often for musical audio processing purposes such as musical sound processing and audio coding. Parameters of the sinusoidal model are extracted from the original sound in a frame-based manner, and a sound that is close to the original one can be synthesized from the extracted parameters. Often times, however missing information about sinusoids can occur, this application aims at recovering this information.

More specifically, sinusoidal modelling aims at representing a sound signal as a sum of sinusoids of given amplitudes, frequencies, and phases. So, after modelling a given audio signal in this manner, a common practice is for the amplitude to be interpolated linearly, and cubic interpolation is used for the phase, the frequency can be found by the differentiation of the cubic phase polynomial.

The researchers proposed a different approach, since they identify that despite the fact that the above method, which is based on a polynomial interpolation of the parameters of the partials, preserves the harmonic relation among partials, it does not take into account the modulations of the parameters of the partials. The modulations are important since the play a decisive role in audio perception.

The proposition states that parameters of partials, can be modelled by an AR model and linear prediction is used in order to predict the parameters of the partials in the missing region. In linear prediction, the current sample can be approximated by a linear combination of past samples of the input signal. The first step to interpolate corrupted sinusoidal data in the missing region is to decide which partial of one side should be linked to which partial of the other side to form a unique partial. The time interval is so long that the evolution of the partials within the missing region is taken into account to achieve a good match. The

different variables of the signal, such as amplitude frequency and phase are calculated by using the original method paired with an AR model.

In that work it is shown that, the parameter of the partials allows those partials to be interpolated reliably. Partials having simple modulations such as vibrato or tremolo allow high-quality interpolation for gap sizes up to 1 s, which is a significant data gap. The researchers had trouble, where more complex modulations were concerned, but their proposed method shows a significant improvement over a simple polynomial interpolation method[12].

Chapter 3:

Interpolation

Interpolation is a type of estimation, a method of procuring new data points within the range of a discrete set of known data points. Often enough, a number of data points, obtained by sampling or experimentation, which represent the values of a function with a set number of values of the independent variable, are sometimes insufficient. To overcome this difficulty, it is often required to estimate the value of that function for an intermediate value of the independent variable. Thus extrapolating the amount of data available. As already explained, during audio sampling, data points are lost. Interpolation is going to be utilized in order to estimate the value of the missing data points increasing audio resolution.

Interpolation and other methods of data recovery are quite necessary because filling data points in a straight line between the original points causes distortion. Distortion is the alteration of the original shape or other characteristic of a signal. In audio processing terms, it means the alteration of the waveform of an audio signal. All in all distortion is undesirable, in our case, a mathematically acute method such as interpolation is required.

In this chapter the concept and theory behind interpolation is explained, however the mathematical models developed for the purpose of this thesis have some differences, which are explained in the next chapter as they come up.

3.1 Interpolation methods

Suppose a set of real values $x_1,x_2,.....,x_n$ and each corresponds to real values $y_1,y_2,.....,y_n$ with the values of y_i being often the result of measurements, sampling or other such processes that do not require or define a proper mathematical function. Here $x_1<x_2,.....<x_n$ and y_i is some observed or mathematically defined real number. One-dimensional interpolation aims to create a function $f(x)$: so that for each i it is true that: $f(x_i)=y_i$. Using this function values of y for each arbitrary x in between the existing points can be reasonably calculated. The task of estimating $f(x)$, for an arbitrary value of x can be interpreted as a fitting a smooth curve through the x axis.[13].

Interpolation methods must model the function, in between the known points, by some plausible function form. The form should be sufficiently general so as to be able to approximate large classes of functions which might arise in practice. By far most common among the functional forms used are polynomials. Other interpolation models include: rational functions such as quotients of polynomials, trigonometric interpolation, which

utilizes Fourier related methods, as well as linear and spline interpolation. It is worth noting that interpolation is related to, but ultimately different from, function approximation, which consists of finding an approximate function to use in place of a more complicated one.[14].

Despite the fact that, there are a number of theorems about what sort of functions can be recreated to a satisfactory degree, these theorems are not applicable to real problems. If we know enough about a function to apply a theorem, typically interpolation is not really needed[14].

As already described, the heart of the interpolation problem is a definition of how a function will behave between pre- established data points. After all the data points can be interpolated by an infinite number of different functions, and we must have some criteria to select among them. The normal criteria are in terms of smoothness and simplicity of the function. Most functions that result from interpolation processes, are built out of linear combinations of elementary functions[14].

One historically important type of interpolating function is polynomial interpolation, which are functions expressed in a set of algebraic polynomials. These functions have the advantage of being simple to evaluate. Summation, multiplication and integration or even differentiation is easily applied. Any continuous function $f(x)$ can be a satisfactory approximated on a closed interval by a polynomial $p_n(x)$[14].

Conceptually, the interpolation process has two stages with them being: the fitting of an interpolated function to the provided data points and the calculation of the intermediate points. However this two stage method is considered inefficient as the result becomes more susceptible to round-off error. A better approach consists of constructing a functional estimate $f(x)$ from the N tabulated values as needed. The further away the included points from the area in the x axis, where we want to calculate intermediate values of $f(x)$, the higher the error percentage. So, given a large signal which is to be interpolated, it is considerably better to use rather small windows of points in the calculation of the interpolating function $f(x)$[14].

The number of points used to procure the interpolating function is called the order of the interpolation. As discussed, increasing the order of the interpolation does not necessarily increase its accuracy unless the added points are relatively close in regards to scale, as the true, unknown, function can behave in a considerably different manner as it progresses. Unless there is solid proof that the interpolating function is close in form to the original, caution is advised when tampering with high order interpolation. Regardless of the method interpolations with 4, 5 or 6 points are thought of performing the best[14].

3.1.1 Linear Interpolation

The simplest interpolation method is to locate the nearest data value, and assign the same value, to the arbitrary point of the x axis. However there is little merit in this approach, as linear interpolation is almost as simple and yields more accurate results. In mathematics, linear interpolation is a method of curve fitting using linear polynomials to construct new data points within the range of a discrete set of known data points. Linear interpolation takes the approximate first derivative of the data points into consideration. Consider the following example given two data points y_a and y_b which correspond to x_a and x_b respectively with $x_a < x_b$. In order to calculate the value of y in the intermediate point x using linear interpolation[27]:

$$y = y_a + \frac{(y_b - y_a)(x - x_a)}{x_b - x_a}.$$

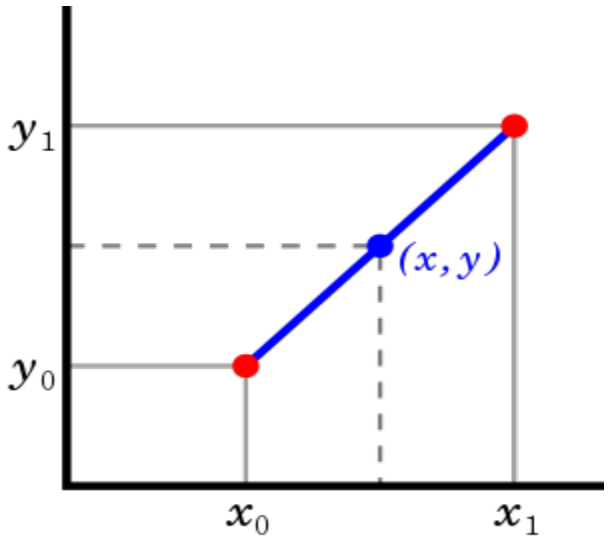

Figure 3.1: An example of linear interpolation(image source:[27])

Linear interpolation is quick and easy, but it is not very precise. Its major advantage is its simplicity and the low requirement of two required data points to calculate. Linear interpolation performs well when the difference between x_b and x_a is small with regard to the

scale, making it a competitive choice for problems where accuracy of the interpolated values is not paramount.

3.1.2 Spline Interpolation

Remember that linear interpolation uses a linear function for each of intervals $[x_k, x_{k+1}]$. Spline interpolation uses low-degree polynomials in each of the intervals, and chooses the polynomial pieces such that they fit smoothly together. The resulting function is called a spline. In the present work the spline interpolation method uses third-degree polynomials, this is called cubic spline interpolation[28].

If the spline is a function represented by $s(x)$ and if the slopes are small, the second derivative $s''(x)$ approximates the curvature and the differential arc length is approximated by dx. Thus the energy of such a linearised spline is proportional to:

$$\int \left(s''(x)\right)^2 dx .$$

When the knots $(x_1,y_1),(x_2,y_2),.....,(x_n,y_n)$ are given, the linearised interpolating spline $s(x)$ is a function such that $s(x_i)=y_i$, with $i=1,2,.....,n$ and such that the following integral is minimized:

$$\int_{x_1}^{x_n} \left(s''(x)\right)^2 dx .$$

Furthermore, the cubic spline function with $s''(x_1)=s''(x_n)=0$ is called a natural spline or in this case a natural cubic spline. This is the unique function possessing the minimum curvature property of all functions interpolating the data and having a square integrable second derivative. In this regard cubic spline is the smoothest function which can be used for data interpolation.

In $n-1$ intervals between nodes there are $n-1$ separate sections of cubic curves, each with four parameters, making $4n-4$ parameters to be determined. The fact that the function s is continuous and has continuous first and second derivatives at each of the $n-2$ interior nodes x_i amount to $3(n-2)$ conditions on s. Also, the fact that $s(x_i)=y_i$ for each of n nodes imposes n more conditions on s, making the number of required conditions equal to 4n-6. As such two more conditions are needed to completely determine the spline making the minimum required number of points equal to 4. As already explained we can set the first and last points' second derivative to 0 in order to procure the natural spline.

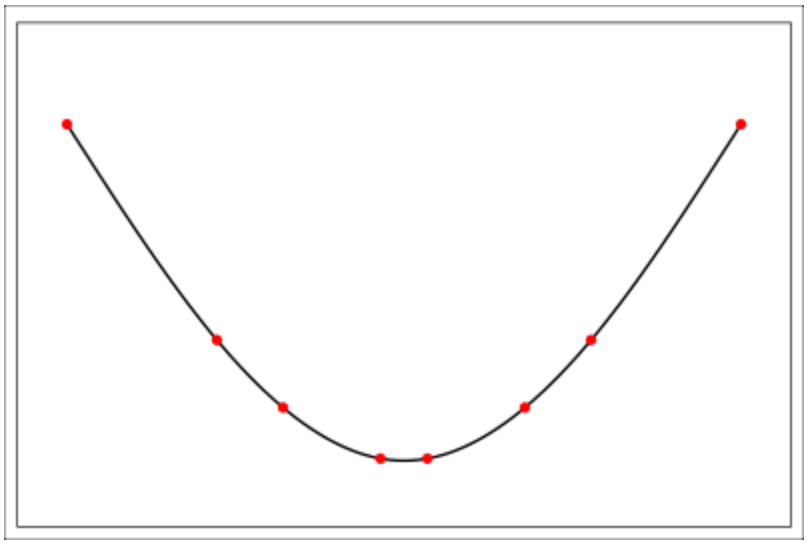

Figure 3.2: An example of cubic spline interpolation between 8 points(image source:[40])

3.2 Mathematical description of Cubic Spline Interpolation

The construction of a cubic spline is a simple and numerically stable process. Given the subinterval $[x_i, x_{i+1}]$ the following are derived:

$$h_i = x_{i+1} - x_i,$$

$$w = \frac{x - x_i}{h_i},$$

$$w' = 1 - w.$$

As x is assigned values from the aforementioned subinterval, w ranges from 0 to 1 and w' from 1 to 0. Then we can represent the spline on this particular subinterval by:

$$s(x) = wy_{i+1} + w'y_i + h_i^2[(w^3 - w)\sigma_{i+1} + (w'^3 - w')\sigma_i]$$

where σ is a certain constant that will be elaborated upon shortly. The first two terms of the expression represent standard linear interpolation as we have already seen, on the other hand the term in the brackets is, as it is called, a cubic correction term the will provide more appropriate smoothness to the interpolated data. Notice that the correction term vanishes at the end points so that:

$$s(x_i) = y_i$$

and

$$s(x_{i+1}) = y_{i+1}$$

Thus $s(x)$ interpolates the data regardless of σ. We now differentiate $s(x)$ thrice, using the chain rule and the fact that $w'=1/h_i$ and $(w')'=-1/h_i$:

$$s'(x)=\frac{y_{i+1}-y_i}{h_i}+h_i[(3w^2-1)\sigma_{i+1}-(3w'^2-1)\sigma_i],$$

$$s''(x)=6w\sigma_{i+1}+6w'\sigma_i,$$

$$s'''(x)=6\frac{\sigma_{i+1}-\sigma_i}{h_i}.$$

Note that $s''(x)$ is a linear function which interpolates the values $6\sigma_i$ and $6\sigma_{i+1}$, so as a result:

$$\sigma_i=\frac{s''(x_i)}{6}.$$

This summarizes the nature of σ but its value remains to be determined. It should be noted that $s'''(x)$ is a constant on each subinterval and as a result the fourth derivative vanishes. Evaluating $s'(x)$ at the end of the subinterval produces:

$$s'_1(x_i)=\Delta_i-h_i(\sigma_{i+1}+2\sigma_i),$$

$$s'_2(x_i)=\Delta_i+h_i(\sigma_{i+1}+2\sigma_i),$$

where

$$\Delta_i=\frac{y_{i+1}-y_i}{h_i}.$$

S_1 and S_2 are used temporarily because the formula for $s(x)$ holds only on $[x_i,x_{i+1}]$, so the derivatives at the end points are one-sided derivatives. To obtain the desired continuity in $s'(x)$ we impose the following conditions at the interior knots:

$$s_1(x_i)=s_2(x_i), i=2,.....,n-1.$$

Although the value of s_2 comes from considering the subinterval $[x_{i-1},x_i]$, a formula for it can be obtained simply by replacing I with i-1 in $s'_2(x_{i+1})$. This leads to:

$$\Delta_{i-1}+h_{i-1}(2\sigma_i+\sigma_{i-1})=\Delta_i-h_i(\sigma_{i+1}+2\sigma_i)$$

hence

$$h_{i-1}\sigma_{i-1}+2(h_{i-1}+h_i)\sigma_i+\sigma_{i+1}h_i=\Delta_i-\Delta_{i-1}, i=2,.....,n-1.$$

This is a system of $n-2$ simultaneous linear equation involving n number of unknowns, σ_i, $i=1,2,\ldots,n$. Two additional conditions must be specified to uniquely define the interpolating spline. There are several ways of picking these two conditions, one of which is the following.

Let $c_1(x)$ and $c_n(x)$, which are unique cubics which pass through the first and last four data points, respectively. The two end conditions match the third derivatives of these cubics, more precisely:

$$s'''(x_1)=c_1'''$$

and

$$s'''(x_n)=c_n'''$$

The constants c_1''' and c_n''' can be determined from the data without the need to calculate $c_1(x)$ and $c_n(x)$ as shown bellow:

Let:

$$\Delta_i^{(1)}=\frac{y_{i+1}-y\,i}{x_{i+1}-x_i},$$

$$\Delta_i^{(2)}=\frac{\Delta_{i+1}^{(1)}-\Delta_i^{(1)}}{x_{i+2}-x_i}$$

and

$$\Delta_i^{(3)}=\frac{\Delta_{i+1}^{(2)}-\Delta_i^{(2)}}{x_{i+3}-x_i}$$

The first group of the above values consists of approximations to first derivatives, while the following two are known as divided differences. $2\Delta_i^{(2)}$ and $6\Delta_i^{(3)}$ are approximations of second and third derivatives respectively. So we can deduce that:

$$c_1'''=6\,\Delta_i^{(3)}$$

and

$$c_n'''=6\,\Delta_{n-3}^{(3)}.$$

Consequently we require that:

$$\Delta_1^{(3)}=\frac{\sigma_2-\sigma_1}{h_1},$$

and

$$\Delta_{n-3}^{(3)}=\frac{\sigma_n-\sigma_{n-1}}{h_{n-1}}.$$

In order to achieve symmetry in the system of equations, these last two equations will be multiplied by h_i^2 and h_{n-1}^2, which yields:

$$h_1^2 \Delta_1^{(3)} = h_1 \sigma_2 - h_1 \sigma_1,$$

and

$$-h_{n-1}^2 \Delta_1^{(3)} = -h_{n-1} \sigma_n + h_{n-1} \sigma_{n-1}.$$

For the spline with these end conditions the σ must satisfy the following system of n linear equations in n unknowns:

$$\begin{pmatrix} -h_1 & h_1 & 0 & 0 & & & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & & & \\ 0 & h_2 & 2(h_2+h_3) & h_3 & & & \\ & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & -h_{n-2} & 2(h_{n-2}+h_{n-1}) & -h_{n-1} & \\ 0 & & & 0 & -h_{n-1} & -h_{n-1} & \end{pmatrix} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \cdot \\ \cdot \\ \cdot \\ \sigma_{n-1} \\ \sigma_n \end{pmatrix} = \begin{pmatrix} h_1^2 \Delta_1^{(3)} \\ \Delta_2 - \Delta_1 \\ \Delta_3 - \Delta_2 \\ \cdot \\ \cdot \\ \cdot \\ \Delta_{n-1} - \Delta_{n-2} \\ -h_{n-1}^2 \Delta_{n-3}^{(3)} \end{pmatrix}$$

The matrix of coefficients has several, special properties. First of all the matrix is tridiagonal, meaning that it has nonzero elements on the main diagonal, the first diagonal below, and the first diagonal above the main diagonal, with the rest of the matrix being filled with zeroes. The matrix is also symmetric and for any choice of $x_1<x_2<...<x_n$, the matrix is nonsingular and diagonally dominant.

Thus a unique solution $\sigma_1,\ldots,\sigma_n$ always exists. It can also be proved that for any reasonable choice of $x_1,x_2,\ldots,x_n$, the coefficient matrix is well conditioned. In conjunction with the fact that the matrix is diagonally dominant accurate solutions can be calculated using Gaussian elimination without scaling or pivoting.

Applying Gaussian elimination to the original system reduces it to upper triangular form:

$$\begin{pmatrix} a_1 & h_1 & & & & & 0 \\ & a_2 & h_2 & & & & \\ & & a_3 & h_3 & & & \\ & & & \cdot & \cdot & & \\ \cdot & & & & \cdot & \cdot & \\ \cdot & & & & & \cdot & \cdot \\ \cdot & & & & & \cdot & h_n \\ 0 & & & & & & a_n \end{pmatrix} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \sigma_n \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \beta_n \end{pmatrix}.$$

The elements of the a_i group are given by:

$$a_1 = -h_1,$$

$$a_i = 2(h_i + h_{i-1}) - \frac{h_{i-1}^2}{a_{i-1}}, i = 2,3,\ldots,n-1,$$

$$a_n = -h_{n-1} - \frac{h_{n-1}^2}{a_{n-1}},$$

the elements of the β_i are given by:

$$\beta_1 = h_1^2 \Delta_1^{(3)},$$

$$\beta_i = (\Delta_i^{(1)} - \Delta_{i-1}^{(1)}) - \frac{h_{i-1}\beta_{i-1}}{\alpha_{i-1}}, i = 2,3,\ldots,n-1,$$

$$\beta_n = -h_{n-1}^2 \Delta_{n-3}^{(3)} - h_{n-1}\frac{\beta_{n-1}}{a_{n-1}},$$

and finally, the coefficients σ_i can be calculated by back substitution as follows:

$$\sigma_n = \frac{\beta_n}{a_n},$$

$$\sigma_i = \frac{\beta_i - h_i\sigma_{i+1}}{a_i}, i = n-1, n-2, \ldots, 1.$$

However it may be preferable, in order to achieve easier manipulations such as derivatives or integrals, to calculate the actual cubic coefficients b_i ,c_i, and d_i , $i=1,2,\ldots,n-1$,

for each interval $[x_i,x_{i+1}]$, where:

$$s(x)=y_i+b_i(x-x_i)+c_i(x-x_i)^2+d_i(x-x_i)^3, x_i\le x\le x_{i+1}.$$

These coefficients, for the values of *i=1,2,…,n-1,* are given by:

$$b_i=\frac{y_{i+1}-y_i}{h_i}-h_i(\sigma_{i+1}+2\sigma_i),$$

$$c_i=3\sigma_i,$$

$$d_i=\frac{\sigma_{i+1}-\sigma_i}{h_i}.$$

At this point it is worth noting that nothing, in the mathematics, of cubic splines strictly forbids the use of a value of x which is greater than x_n. However this process is known as extrapolation and its results become less and less reliable the greater the value of x becomes[13].

Chapter 4:
Modelling and Simulation

The use of modelling and simulation within engineering is well recognized for its ability to aid in delivering a better solution to the problem that needs to be addressed. Modelling and simulation helps to reduce costs, increase the quality of products and systems, and document and archive lessons learned. Because the results of a simulation are only as good as the underlying model, and as a result particular attention is required to its development. To ensure that the results of the simulation are applicable to the real world, a significant amount of understanding of the implementation's assumptions, conceptualizations, and constraints. [29]

Audio models can be handled to a satisfactory degree by Matlab. The most common method of handling audio in Matlab is storing the audio signal data as a vector of samples, with each individual value being a double-precision floating point number. A sampled sound can be completely specified by the sequence of these numbers plus one other item of information: the sample rate. However the majority of digital audio systems differ from this in one significant respect, and that is they tend to store the sequence of samples as fixed-point numbers instead. The present approach will be utilizing a fixed-point data representation, as the title of this thesis suggests.[15]

On the other hand, this is second generation of a system modelled and implemented using a floating-point format. As a result a good understanding of the original model is required, in order to transition it to a fixed-point format. In order to achieve that level of understanding the first phase of the modelling process will consist of modelling an algorithm for both cubic spline and linear interpolation, using floating-point arithmetic in Matlab.

4.1 Introducing the model

Any operation that Matlab can perform on a vector can, in theory, be performed on stored audio. The audio vector can be loaded, saved, processed, and plotted in the same way as any other Matlab variable. Additionally Matlab offer functions that easily extract the data of a WAVE file, as well as functions like *sound(data, frequency)*, which immediately plays back the data. This function can provide an easy method of checking for obvious mistakes either in the importation of the original data, or the processed result. It should be kept in mind though, that this is not a reliable measure of testing the results of the model, just a useful method of probing for large scale inaccuracies. Finally, Matlab offers a large collection of functions for analysis, in both the time and frequency domains, such as plots or spectrograms which can be used to determine the quality of the model's results.

More specifically, the function *audioread* will be utilized in order to extract the desired data and the input file's sampling rate. By default *audioread* will arrange the data in a double-precision floating-point vector its dimensions are: Number of Samples by number of channels. This vector is used as is for the floating-point reproduction of the original model, but can be easily converted to fixed-point, in order to be compatible with the second generation model. Finally, the data *audioread* provides is normalized which allows for higher data resolution during the model's mathematical calculations.

Despite its many advantages Matlab has a minor drawback. That being the fact that it is geared and optimized for floating-point data and calculations. However, it does support fixed-point mathematics albeit with a relative lack of functions and operators which accept fixed-point inputs. Another drawback is the fact that fixed-point arithmetic operations are simulated by functions provided Matlab, resulting in very slow execution times.

4.2 First Generation model reproduction

Typically the input files of the model are of CD quality, meaning uncompressed files following the WAVE format, with a sampling rate of 44100 Hz, and 16-bit signed integers for the data. The data will be normalized in order to make higher resolution results, such as 24-bit and 32-bit resolution, possible.

CD quality WAVE files can be either mono or stereo. As a result, the number of channels is either one or two, which results in the vector, returned by *audioread*, having either one or two columns. It is arguably a better approach, for stereo data, to split the vector in two, process them separately and merge them back together to procure the final audio signal.

4.2.1 Resampling

One of the methods that will be used to sufficiently test the cubic spline model is resampling. Resampling or sometimes called sample-rate conversion is the process of changing the sampling rate of a discrete signal to obtain a new discrete representation of the underlying continuous signal[30]. It essentially serves the same purpose as interpolation, in the scence that it attempts to estimate the original signal's missing points. The method by which it tries to achieve that goal is to first insert zeros to upsample the signal by the desired amount *(p)*, followed by the appliance of an FIR anti-aliasing filter to the upsampled signal, and finally discarding samples to downsample the filtered signal by another amount (q). The ratio of p/q multiplied by the original sampling rate, which will commonly be 44.1 kHz, together with the vector of the original data-points, serves as the input for Matlab's resample function which will be compared to the primary algorithm for cubic spline and linear interpolations[31].

4.2.2 Linear Interpolation

The first step in reproducing the linear interpolation model is to determine its inputs. The model's input should be the vector in which the data is stored *(y)*, the file's sampling rate *(F_s)*, and the desired frequency for the new signal. Most tests presented in this chapter will be aiming for a frequency of 176.4 kHz, but it should be noted that any frequency higher that 44.1 kHz is possible. The next step is to determine the number of data points in the vector *(n)*, which will be used to determine the time vector *t*, as follows:

$$t_i = iT_s, i = 0,1,\ldots,n,$$

where

$$T_s = \frac{1}{F_s}.$$

The same process is used for the result's time axis, but first the duration of the input is required, which is given by:

$$duration = \frac{n}{F_s},$$

with knowledge of the duration the number of samples can be calculated as follows:

$$n_{new} = duration \times F,$$

and then:

$$t_j = jT,\ j = 0,1,\ldots,n_{new}.$$

Then, for every t_j in the interval of $[t_0, t_n]$ with each t corresponding to exactly one y, the linear interpolation equation is used:

$$y_j = y_a + \frac{(y_b - y_a)(t_j - t_a)}{t_b - t_a},\ t_a < t_j < t_b.$$

More specifically, linear interpolation only requires two data points, as explained in the previous chapter. These two data points are represented here by y_a and y_b whith each corresponding to t_a and t_b respectively. So in order to calculate the value of the audio signal at a point t_j the data window must be moved by one data point every time t_j becomes greater or equal to t_b.

The only data that needs to be returned by the model is the resulting data vector as the time axis does not serve any further function. A simple sine pulse, with a frequency of F_s=100 Hz is used as testing signal. Linear interpolation is then used on that signal to procure additional data points as shown in the images bellow.

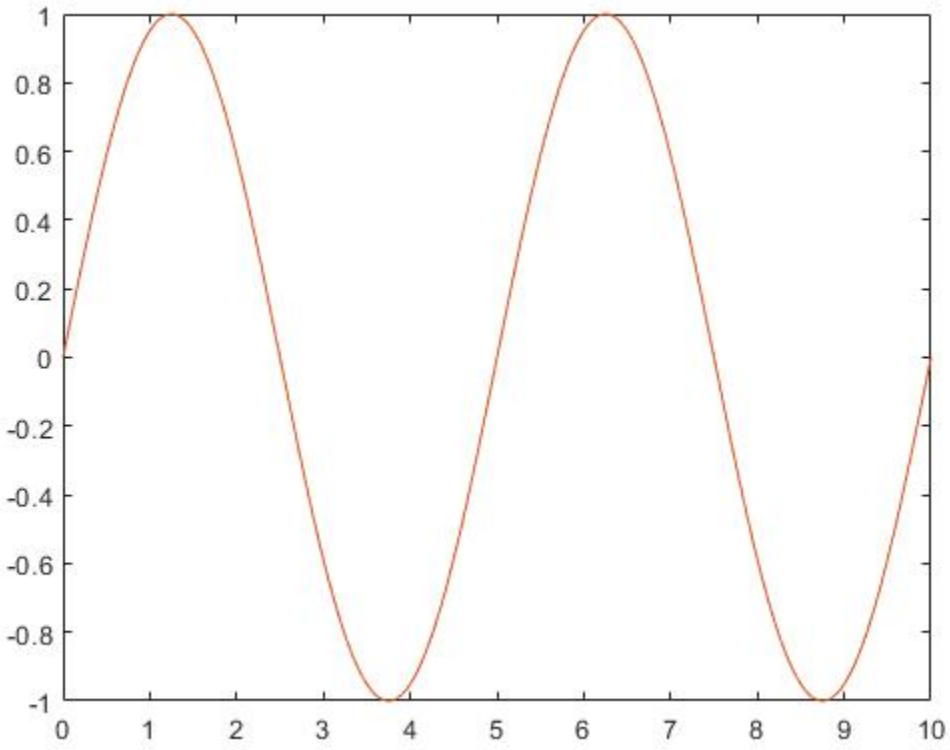

Figure 4.1:The tested sine wave

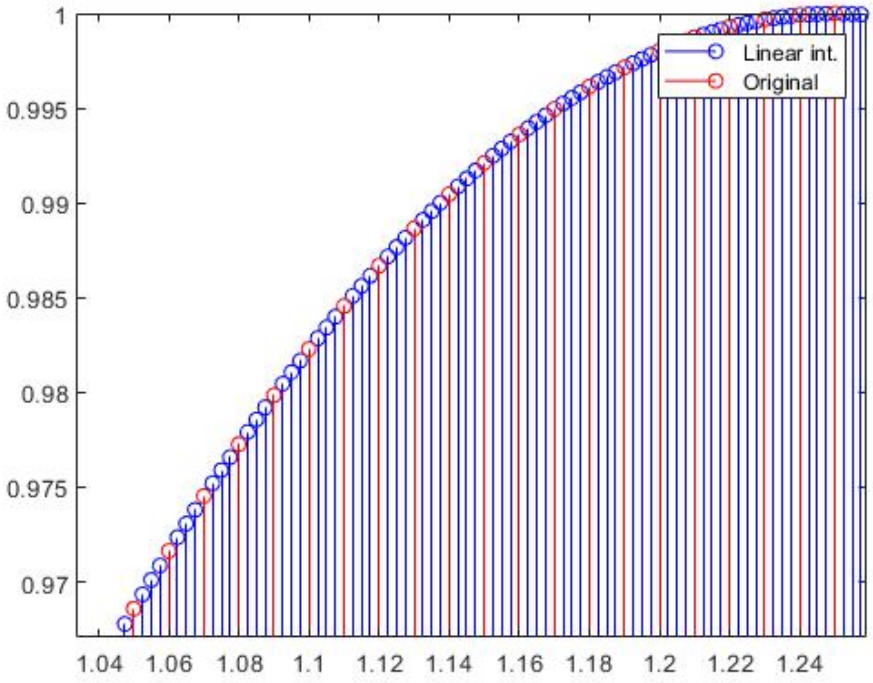

Figure 4.2 Linear interpolation applied on a part of a sine wave

4.2.3 Cubic Spline Interpolation

Over the development of the model three cubic spline algorithms were implemented and tested with varying degrees of success. Each algorithm's inputs are exactly the same as with linear interpolation. More specifically the inputs are: the data vector *(y)*, the original sampling rate(F_s), and the model's target frequency *(F)*. The preferred target frequency for the tests remains at 176.4 kHz. The values of $t_i, t_j, n, n_{new}, T_s$, are calculated in the same way as linear interpolation as well, since these values are derived from signal processing and general interpolation theory and are not bound to any specific interpolation method.

4.2.3.1 The first Algorithm

The first algorithm, which was unable to recreate the desired results, tries to follow a less direct approach to cubic splines by attempting to calculate the coefficients of the spline directly. But first some standard values need to be calculated as they are integral to the final coefficients, the values are:

$$h_i = t_{i+1} - t_i, i = 1,2,\dots,n$$

$$\Delta_i^{(1)} = \frac{y_{i+1} - y\,i}{h_i},$$

$$\Delta_i^{(2)} = \frac{\Delta_{i+1}^{(1)} - \Delta_i^{(1)}}{t_{i+2} - t_i},$$

which can be written as

$$\Delta_i^{(2)} = \frac{\Delta_{i+1}^{(1)} - \Delta_i^{(1)}}{t_{i+2} - t_i + t_{i+1} - t_{i+1}},$$

and finally as:

$$\Delta_i^{(2)} = \frac{\Delta_{i+1}^{(1)} - \Delta_i^{(1)}}{h_{i+1} + h_i},$$

and

$$\Delta_i^{(3)} = \frac{\Delta_{i+1}^{(2)} - \Delta_i^{(2)}}{t_{i+3} - t_i},$$

which becomes:

$$\Delta_i^{(3)} = \frac{\Delta_{i+1}^{(2)} - \Delta_i^{(2)}}{t_{i+3} - t_i + t_{i+1} - t_{i+1} + t_{i+2} - t_{i+2}},$$

and then:

$$\Delta_i^{(3)} = \frac{\Delta_{i+1}^{(2)} - \Delta_i^{(2)}}{h_{i+2} + h_{i+1} + h_i}.$$

These coefficients are already explained in the previous chapter but for further elaboration are repeated bellow:

$$a_1 = -h_1,$$

$$a_i = 2(h_i + h_{i-1}) - \frac{h_{i-1}^2}{a_{i-1}}, i = 2,3,\ldots,n-1,$$

$$a_n = -h_{n-1} - \frac{h_{n-1}^2}{a_{n-1}},$$

$$\beta_1 = h_1^2 \Delta_1^{(3)},$$

$$\beta_i = (\Delta_i^{(1)} - \Delta_{i-1}^{(1)}) - \frac{h_{i-1}\beta_{i-1}}{\alpha_{i-1}}, i = 2,3,\ldots,n-1,$$

$$\beta_n = -h_{n-1}^2 \Delta_{n-3}^{(3)} - h_{n-1}\frac{\beta_{n-1}}{a_{n-1}}.$$

All the values are know so a simply substitution produces the values of a_i and β_i. These coefficients are, in turn, used to calculate the values of σ_i :

$$\sigma_n = \frac{\beta_n}{a_n}$$

and

$$\sigma_i = \frac{\beta_i - h_i\sigma_{i+1}}{a_i}, i = n-1, n-2, \ldots, 1.$$

Afterwards the final coefficients can be calculated:

$$b_i = \frac{y_{i+1} - y_i}{h_i} - h_i(\sigma_{i+1} + 2\sigma_i),$$

$$c_i = 3\sigma_i,$$

and

$$d_i = \frac{\sigma_{i+1} - \sigma_i}{h_i}.$$

Which are then substituted on the following equation in order to produce the final result for each interval of $[t_i, t_{i+1}]$:

$$s_j = y_i + b_i(t_j - t_i) + c_i(t_j - t_i)^2 + d_i(t_j - t_i)^3, i = 1,2,\ldots n, j = 1,2,\ldots n_{new}.$$

This algorithm, however was producing non optimal results compared to other algorithms and compared to the first generation's model. As a result it was discontinued in favour of less computation intensive algorithms that calculate a natural spline, meaning the boundary of s values for t_1 and t_n are set to zero, reducing workload and increasing the quality of the results.

4.2.3.2 The second Algorithm

The second algorithm, which was ultimately unsuccessful in providing reliable results, is derived directly from the bibliography and attempts to calculate a natural spline and attempts

a more direct approach by solving the tridiagonal matrix instead of calculating the coefficients as directly as the previous algorithm. This is achieved by using two vectors named *u* and *y2,* which serve as temporary storage of the decomposed factors, which are calculated in a decomposition loop for the tridiagonal matrix So the first step in the algorithm is:

$$u_1 = y2_1 = u_n = y2_n = 0.$$

Of course the result's time vector (t_j) needs to be calculated again, together with the other basic values required for signal processing, referenced already in the first algorithm. Additionally, *h* as defined by the previous algorithm will be needed for the calculation of the spline:

$$h_i = t_{i+1} - t_i, i = 1,2,\ldots,n-1$$

The next step is the decomposition loop, which is:

$$sig_i = \frac{x_i - x_{i-1}}{x_{i+1} - x_{1-1}},$$

$$p_i = sig_i \times y2_{i-1} + 2,$$

$$y2_i = \frac{sig_i - 1}{p_i},$$

$$ut_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}},$$

$$u_i = 6\frac{ut_i}{x_{i+1} - x_{i-1}} - sig_i \frac{ut_{i-1}}{p_i},$$

where:

$$i = 1,2,\ldots,n.$$

Now that the decomposed factors have been calculated, the back-substitution for *y2,* can be executed:

$$y2_j = y2_j \times y2_{j+1} + u_j, j = n-1, n-2, \ldots, 1$$

The follow up is to calculate the spline's coefficients and the spline itself, for each t_j in the interval of $[t_i, t_{i+1}]$:

$$a_j = \frac{t_{i+1} - t_j}{h_i}$$

and

$$b_j = \frac{x_j - x_i}{h_i},$$

with these coefficients, the spline can now be calculated:

$$y_j = a_j y_i + b_j y_{i+1} + \frac{h_i^2}{6}\left(\left(a_j^3 - a\right) y2_i + \left(b^3 - b\right) y2_{i+1}\right),$$

where:

$$i = 1,2,\dots,n$$

and

$$j = 1,2,\dots,n_{new}.$$

Using the same sine wave as the one used for the linear interpolation test the output is close enough to a proper result.

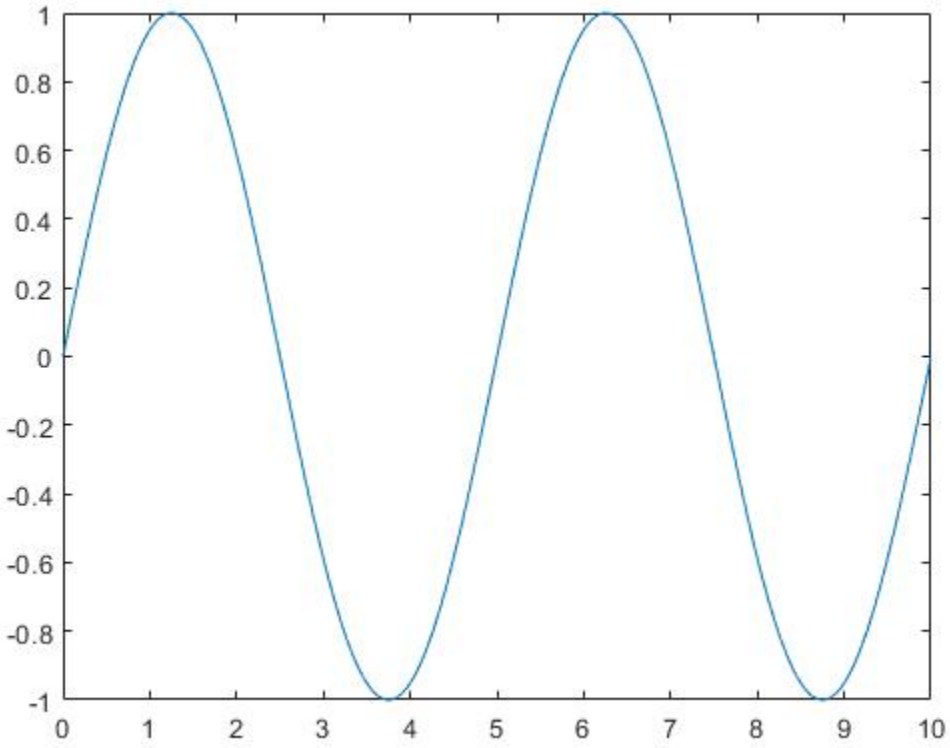

Figure 4.3: The results of the second algorithm when applied on a sine wave

On further inspection however, the results of the algorithm are almost identical to the results of linear interpolation, which should not be the case, and as such the algorithm is directly compared to linear interpolation with the same sine wave as an input as well as a 10 s, audio signal.

As shown in figure 4.4, cubic spline's second iteration yields perplexingly similar results to those of linear interpolation. The value of y at $t=2.5s$ is their common value, derived from the original signal. In figure 4.5, the results are zoomed in to a point where the two values finally become distinct to the eye. In order fully test the algorithm's results, an actual audio signal is required, thus ushering into another round of tests.

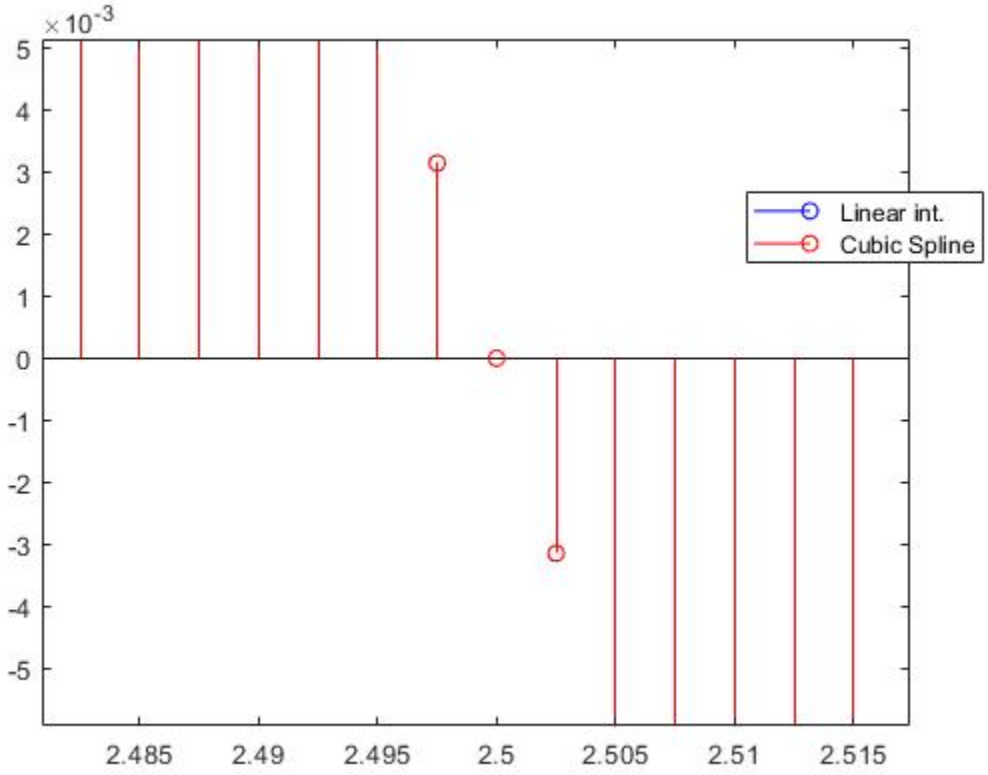

Figure 4.4 A comparison between cubic spline's second iteration and linear interpolation.

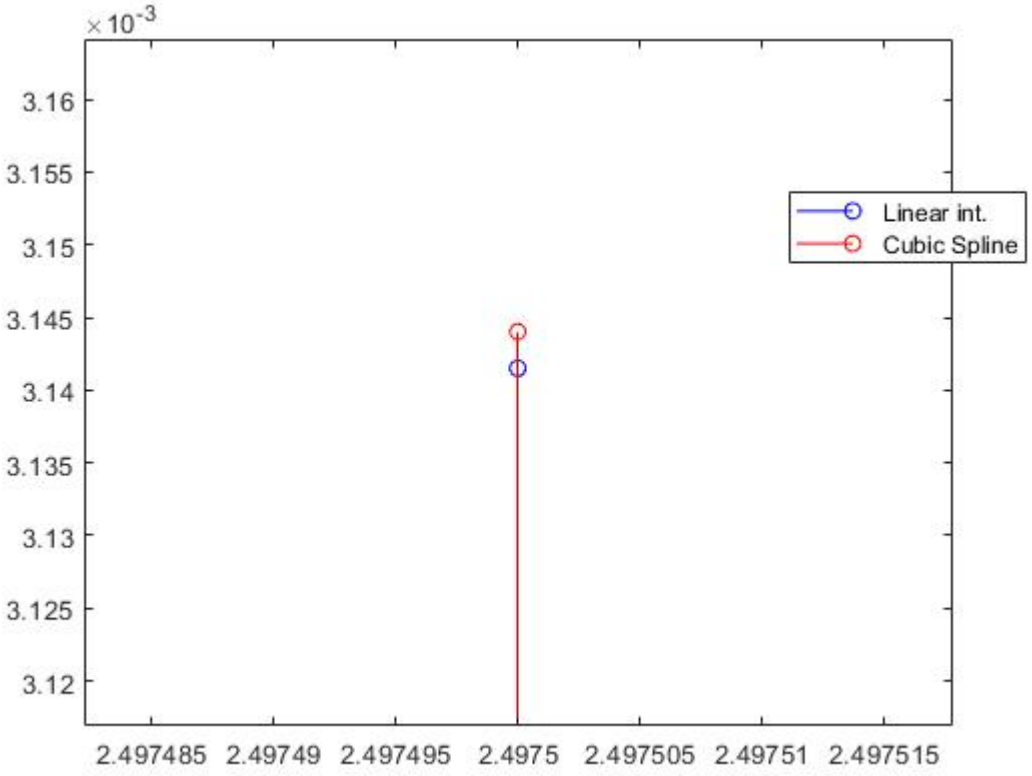

Figure 4.5: The difference between the two results

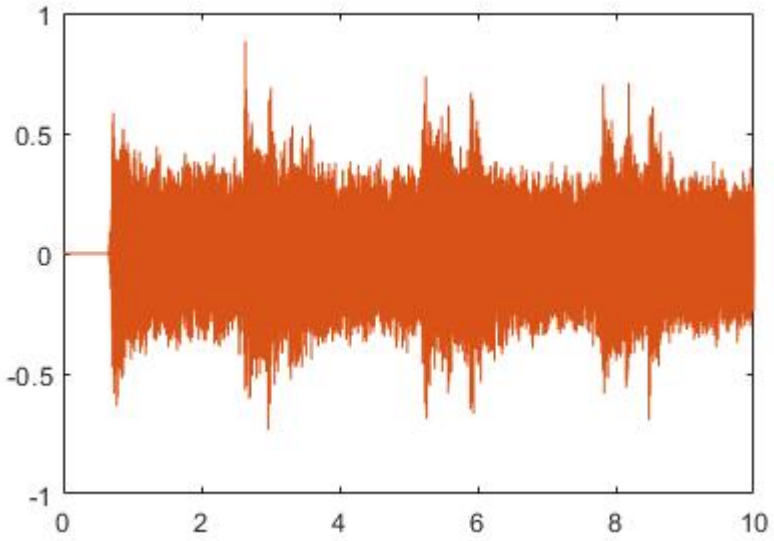

Figure 4.6: A ten second audio signal, resulting from linear interpolation and cubic spline, with practically identical results

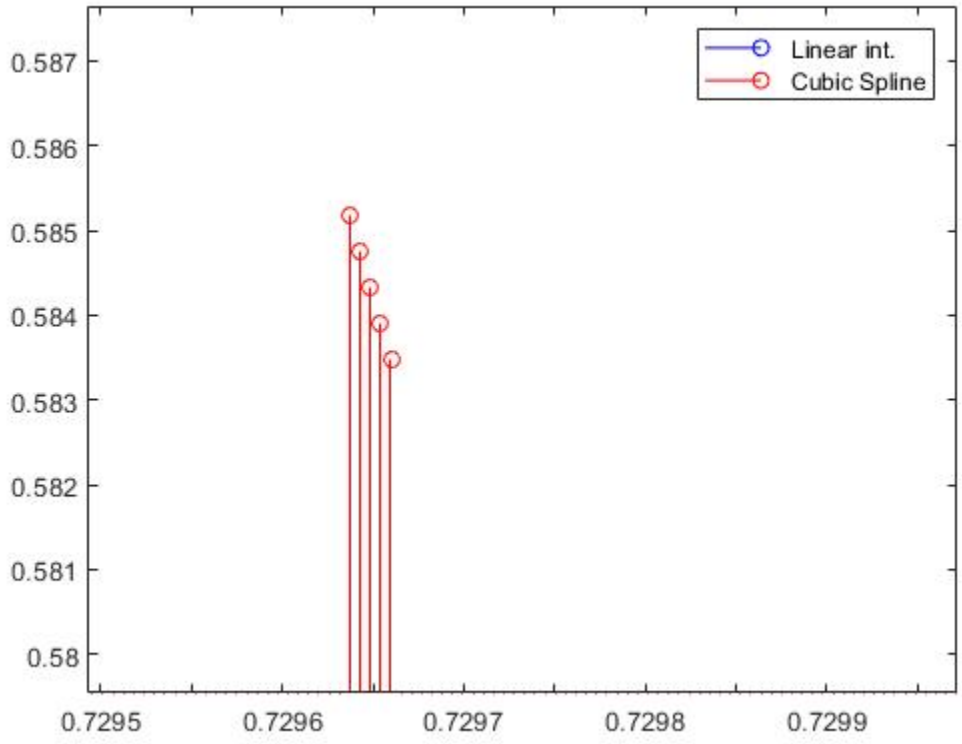

Figure 4.7: Zooming in on the interpolated audio signal, even now the results appear identical

Inspecting figure 4.6, which depicts an audio signal with a duration of ten seconds, the two interpolated signals are indistinguishable. This observation is not considered abnormal as the sampling rate of the signal is 44.1 kHz, meaning there is a high density of data points. Zooming in on the signal the points of the signals are again extremely close in value, and they seem to ignore the curvature that cubic splines are supposed to provide.

In conclusion this algorithm provides incorrect results and is more of an equivalent of linear interpolation than a cubic spline one.

4.2.3.3 The third Algorithm

The third algorithm will be calculating the spline following all the steps underlined in chapter 3, by taking advantage of the Matlab's backslash(\) operator to solve the tridiagonal matrix. So once again the values of $h, t_i, t_j, n, n_{new}, T_s$, need to be calculated as was the case with the other algorithms, the calculation process is the same.

The next step is to set up the group of linear equations, which involve the tridiagonal matrix. As such Δ_i is the next value to be calculated:

$$\Delta_i = \frac{y_{i+1} - y_i}{h_i}, i = 1,2,\dots,n-1.$$

Which is then used to calculate the right part of the linear system as follows:

$$R_i = 6(\Delta_{i+1} - \Delta_i), i = 1,2,\dots,n-2.$$

Another step is, of course, to set up the tridiagonal matrix, which is named mT, but without the first and the last point of the matrix as the spline that the algorithm calculates is a natural spline. As such the matrix is written as:

$$mT = \begin{pmatrix} 2(h_1+h_2) & h_2 & 0 & & & & 0 \\ h_1 & 2(h_2+h_3) & h_3 & & & & \\ & h_2 & 2(h_3+h_4) & h_4 & & & \\ & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-1} & \\ 0 & & & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & \end{pmatrix}$$

Now the liner system common in cubic spline interpolation is written as:

$$\begin{pmatrix} 2(h_1+h_2) & h_2 & 0 & & & 0 \\ h_1 & 2(h_2+h_3) & h_3 & & & \\ & h_2 & 2(h_3+h_4) & h_4 & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-1} \\ 0 & & & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ \cdot \\ \cdot \\ \cdot \\ m_{n-3} \\ m_{n-2} \end{pmatrix} = \begin{pmatrix} R_1 \\ R_2 \\ R_3 \\ \cdot \\ \cdot \\ \cdot \\ R_{n-3} \\ R_{n-2} \end{pmatrix},$$

This system can be solved in Matlab quite easily using the backslash (\) operator, by writing the system as: m=mT\R. Additionally one zero is added at the beginning and another one at the end of the *m* vector, as they represent the boundary conditions for a natural spline.

All that now remains is to calculate the coefficients of the spline, as well as the new data points:

$$s1 = \Delta_i - h_i(2m_i + m_{i+1}), i = 0,1,2,\ldots,n-1,$$

$$s2_i = \frac{m_i}{2},$$

$$s3_i = \frac{m_{i+1} - m_i}{6h_i}, i = 0,1,\ldots,n-1,$$

$$s_j = y_i + s1_i(t_j - t_i) + s2_i(t_j - t_i)^2 + s3_i(t_j - t_i)^3,$$

where t_j, is every point, which needs to be associated with a value s_j, in the interval $[t_i, t_{i+1}]$.

Using the same sine wave to compare the third spline algorithm with the linear algorithm, plotting them together, and zooming in enough a small difference between the two algorithms is finally visible in figure 4.8. In this example the cubic spline algorithm provides a steeper curve to the final result in comparison to the linear algorithm, serving as a good indication that the algorithm is on the right track, and it is in line with the result of the first generation's model.

As a second test, the input is changed from the sine wave to the audio signal used in figure 4.6. As seen, in areas were the curve of the audio signal is supposed to be steeper, cubic spline provides such a result, while linear provides an angle instead of the desired curve, granting further confirmation of the algorithm's validity.

Now that a sufficiently accurate algorithm for cubic spline interpolation has been established, measures are needed, in order to adapt it for hardware implementation.

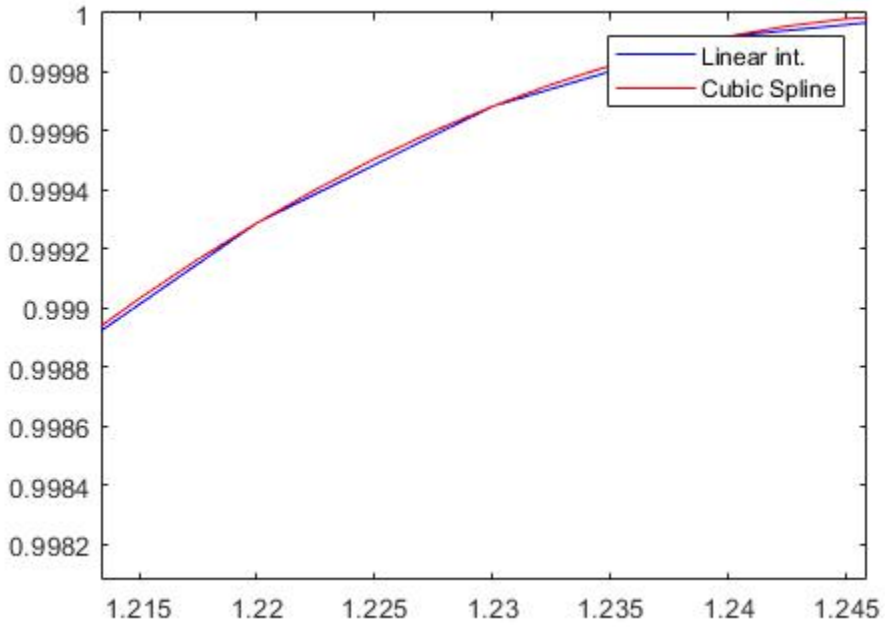

Figure 4.8: Plot comparison of Linear and cubic spline interpolation using the third algorithm

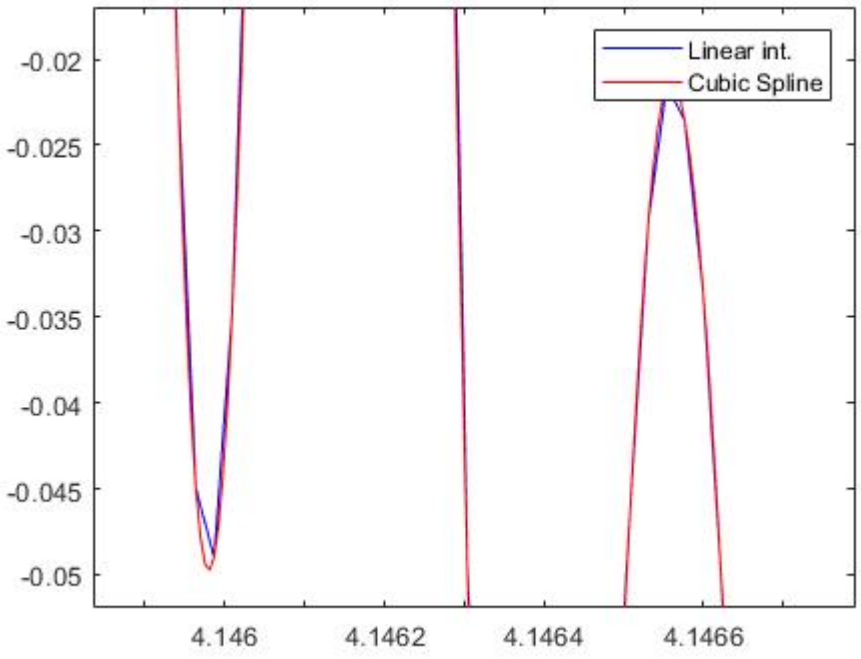

Figure 4.9:Comparison of Linear and cubic spline interpolations on an audio file

4.3 Adapting the Model for Hardware efficiency

The algorithm described at this stage is using a large matrix and two large vectors before even calculating the coefficients. For example, suppose a stereo audio signal with a duration of ten minutes, a sampling frequency of 44.1 kHz and a resolution of 16 bits per sample. This translates into 26,460,000 data-points per channel. So it becomes obvious that a way to reduce resources is needed. As already explained there is no need to insert every data point into the system. Instead a window of data-points which traverses the whole input vector can be utilized to dramatically decrease hardware resources. The minimum number of data points needed for cubic spline interpolation is four. A four data point window is a fairly reliable option, since it is not realistically possible to determine the optimal width of the window since it varies between audio signals. Again, it is useful to note that adding extra data-points in the calculations can either increase or reduce the accuracy of the results, depending on the signal.

Now that the resource requirements have been significantly reduced, and the results yielded are superior to the linear interpolation's results, the resources can be reduced even further by replacing floating-point with fixed-point arithmetic. Matlab, however, struggles with fixed point arithmetic operations as they are simulated using specific functions such as the *fi* function. This function receives three arguments as inputs: the vector to be converted, an object type returned by the *numerictype* function, and another object returned by the *fimath* function.

Numerictype is used to define the length, in bits, of the fixed point value as well as how many of these bits are used for the fraction or for the integer part of the number. *Fimath* is used to set-up the settings for each value, in this case for example saturation will be used in dealing with overflow, additionally it provides options concerning the precision of summation and multiplication. As for the precision of divisions, unfortunately it must be set arbitrarily as there is no method of determining the optimal bit-depth for every possible pair of numbers. The way Matlab determines the final precision of a division is by adapting it to the size of the variable it will be stored in. As a side note, the size of the value divided must be significantly higher that the divider's size as Matlab, in an attempt to set a cap to the result's depth, it will produce a result with a width equal to the difference of the two numbers' widths. Consequently often the algorithm increases the size of values before a division, there is no carry over of this practice in the hardware implementation discussed in the next chapter.

Adapting to the fact that the input is now a streamed window of four data-points and the fact that the backslash(\) operator, utilized to solve the linear system, is not supported for fixed

point arithmetic calculations, the algorithm needs a small adaptation, in that particular part of the code. More specifically n, which denotes the number of inputs is always set to four. This change affects the algorithm at large and simplifies it on specific key points as follows:

$$h_i = t_{i+1} - t_i, i = 1,2,3$$

and then

$$\Delta_i = \frac{y_{i+1} - y_i}{h_i}, i = 1,2,3,$$

$$R_i = 6(\Delta_{i+1} - \Delta_i), i = 1,2.$$

This change also affect the mT matrix changing it to a 2x2 matrix and the whole linear system is depicted as:

$$\begin{pmatrix} 2(h_1+h_2) & h_2 \\ h_1 & 2(h_2+h_3) \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \end{pmatrix} = \begin{pmatrix} R_1 \\ R_2 \end{pmatrix}.$$

This means the system can be solved by calculating the inverse of mT and multiplying it by the R matrix, as a first step in calculating the inverse the determinant of the matrix is calculated:

$$det = \frac{1}{4(h_1+h_2)(h_2+h_3) - h_1 h_2},$$

so in turn the inverse is:

$$mT^T = det \begin{pmatrix} 2(h_2+h_3) & -h_1 \\ -h_2 & 2(h_1+h_2) \end{pmatrix},$$

and multiplying it by the R matrix, thus providing a solution to the system:

$$m = mT^T \times R.$$

After this step the calculations proceed as expected for both the coefficients of the spline as well as the new data points.

Finally, the four point input window does bring some changes additional changes to the algorithm that need to be further explained. The new data-points produced by increasing the sampling rate of the original audio signal, are always placed between the second and the third point. As a result in order to achieve proper results, the window needs to progress only by one data-point thus creating what is called an overlapping window. Although it is possible to place new data-points in the intervals between all of the points being processed each time, the results diverge widely, from what is expected by theory, because the ultimate number of points that are taken into account is reduced, by the nature of the algorithm.

4.4 The Final Model

Having successfully reproduced the algorithms necessary to attain a grasp of the first generation's model and implementing necessary changes to make the cubic spline's algorithm more hardware friendly, a new model is needed to compare each method with each other. This model is depicted in figure 4.10.

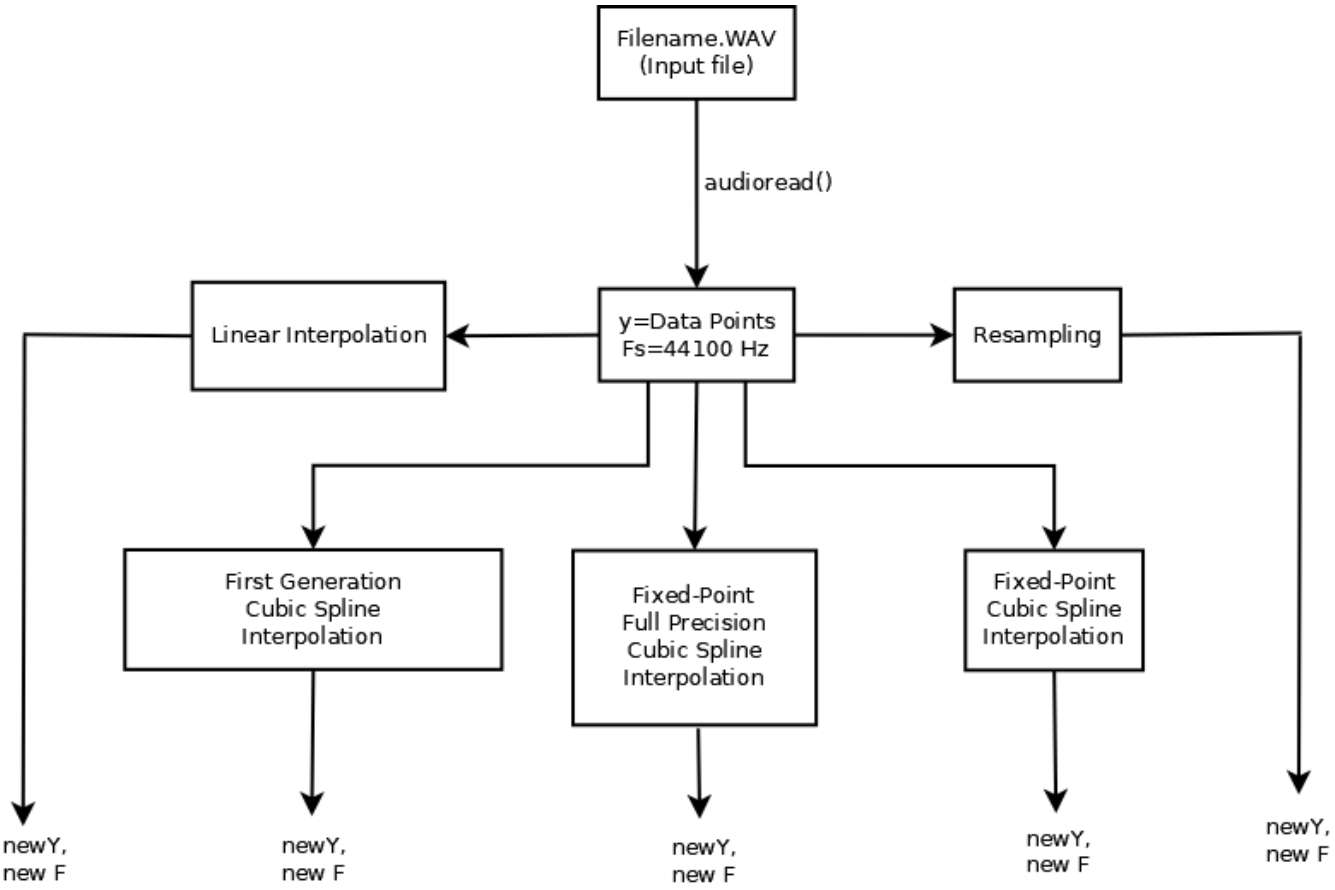

Figure 4.10: The final Matlab model

To elaborate further on the diagram, the value depicted as *new F* is the new sampling rate and it must be equal to, or greater than the original sampling rate of F_s. Both fixed point functions use a 4 data-point window as their input, while linear and floating-point cubic spline, which is labelled as the first generation's cubic spline, process all the data points in a large matrix keeping in line with the first generation. As a reminder this difference does not impact the accuracy of the new data points, for reasons highlighted earlier. The reason for the existence

of two different fixed-point cubic spline algorithms is to determine how much accuracy will impact the results in other to further reduce the resources of the hardware implementation.

The model's methods will be compared to one another by utilizing tests in both the time and frequency domains, so that a broad picture of the actual results can be distilled from the results. The tests that will be conducted involve the recovery of missing data points while maintaining the original sampling rate as well as the recovery of data-points and bit-depth by increasing the sampling rate.

4.5 Frequency domain analysis

For the frequency domain analysis two different audio signals will be used as inputs. The first signal is a 10 second audio signal cut from the song called Book of Souls by Iron Maiden. As seen in figure 4.11, the audio file is in stereo format, meaning there are two channels to process separately. The first two seconds of the signal are low a low amplitude, guitar string vibration, then it transitions into all the organs, such as bass, drums as well as guitars, producing sound waves all at once. Vocals are missing in this particular input file. All in all this input covers a significant area of the musical spectrum and will make for relatively reliable tests.

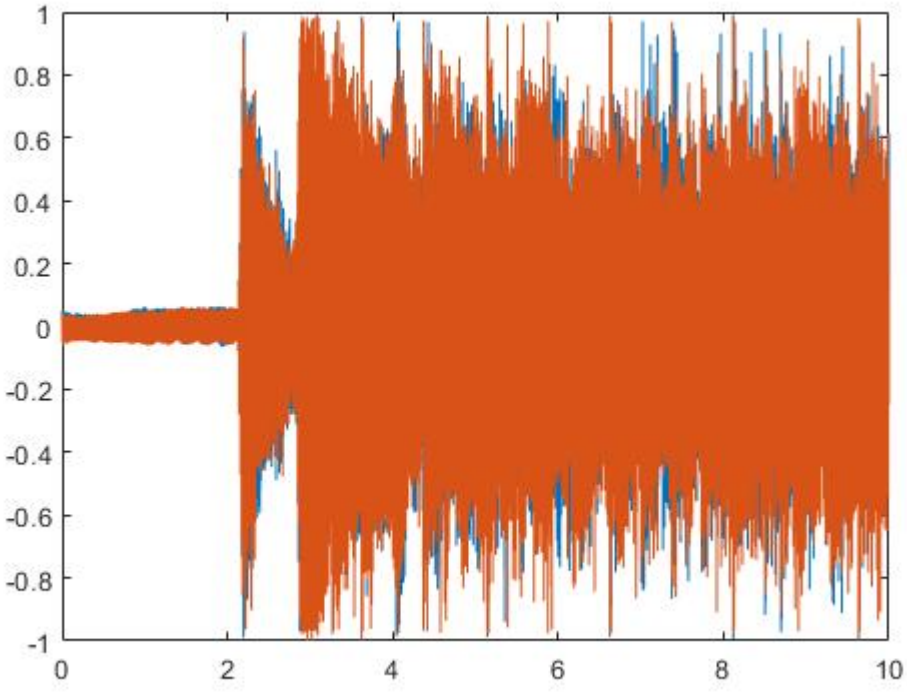

Figure 4.11: The input file for the first round of tests, channel one in blue, channel two in orange

Spectrograms will be the main method of comparing the different results of each algorithm. Spectrograms offer the ability to study the frequency spectrum of each signal by describing its amplitude for each individual frequency present in the audio signals.

The first step in creating a spectrogram for a given audio signal is by applying discreet Fourier transformation on a Hamming window of data points, in order to achieve clear results on the spectrograms. The parameters of the set-up are: a Hamming window with a width of 1024 data-points, 50% overlap between contiguous sections and the number of DFT(Discreet Fourier Transformation) points is set at 2048[32].

The ratio of Power(dB) to Frequency(Hz) represents the Power Spectral Density, or PSD for sort, of the signal during the DFT in relation to the frequency. The smoother the transitions of the PSD, the smoother the audio will appear to the human ear. The sampling rates that will be tested in this procedure are: 44.1 kHz, which is the standard CD quality, 48 kHz, which is the standard DVD quality, 96 kHz, which is the sampling rate of what is called super audio quality and finally 176,4 kHz a favourite sampling rate among audio engineers because of the fact that it is a multiple of 44.1 which is known to provide higher quality results.

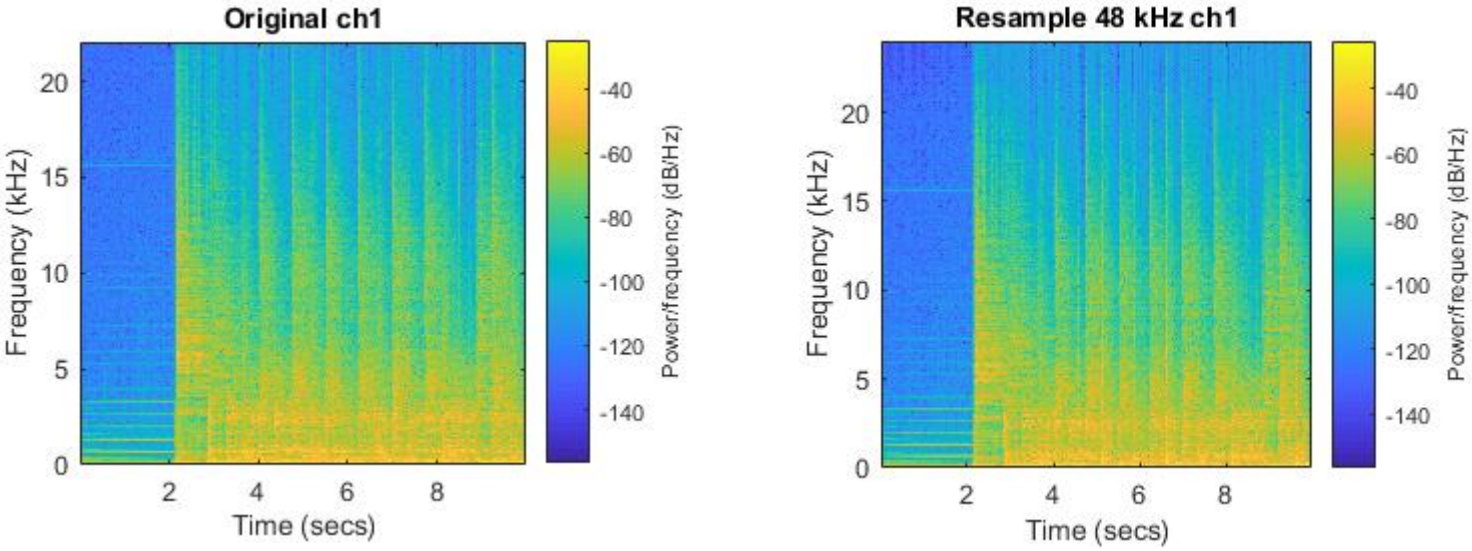

Figure 4.12: Spectrogram of 44.1 kHz of the original and 48 kHz after resampling for the first channel

Looking at figure 4.12 and figure 4.13 resampling appears to not provide significant changes to the spectrum, apart from a relative small increase in the smoothness between transitions. Over all though the improvement is so minor that significant increase in quality by using the other methods is not to be expected for a frequency of 48 kHz. Additionally on the lower frequencies that are placed in the first two seconds of the test file the difference is not visible whatsoever.

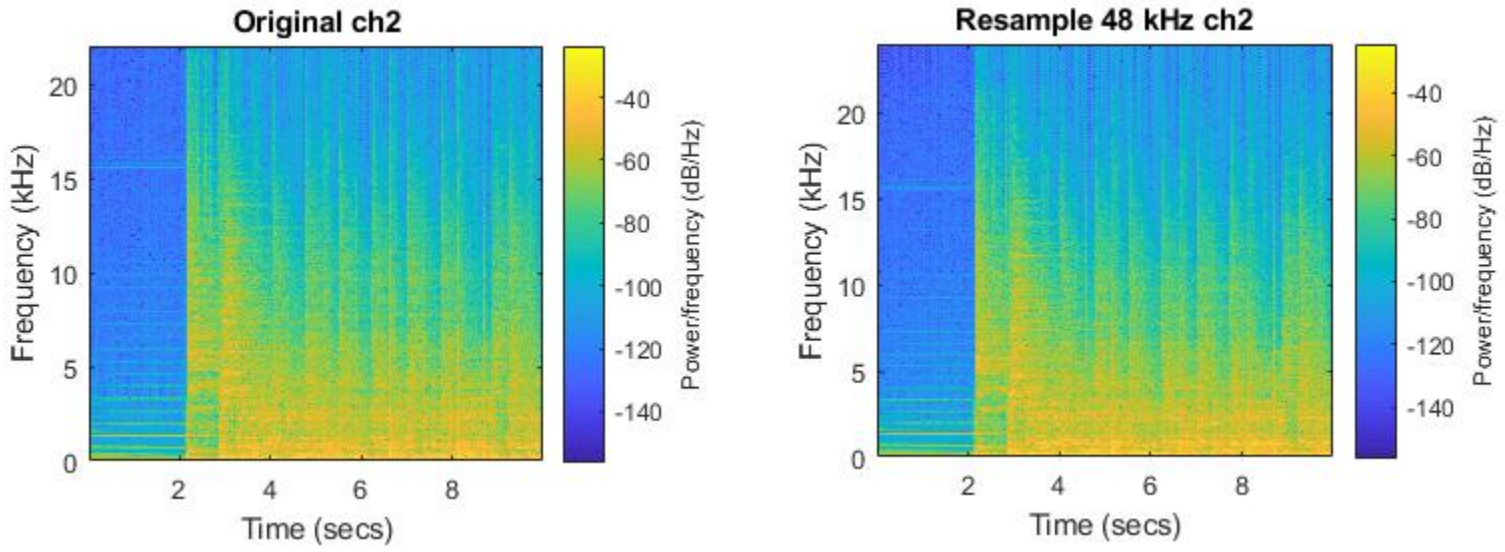

Figure 4.13: Spectrogram of 44.1 kHz of the original and 48 kHz after resampling for the second channel

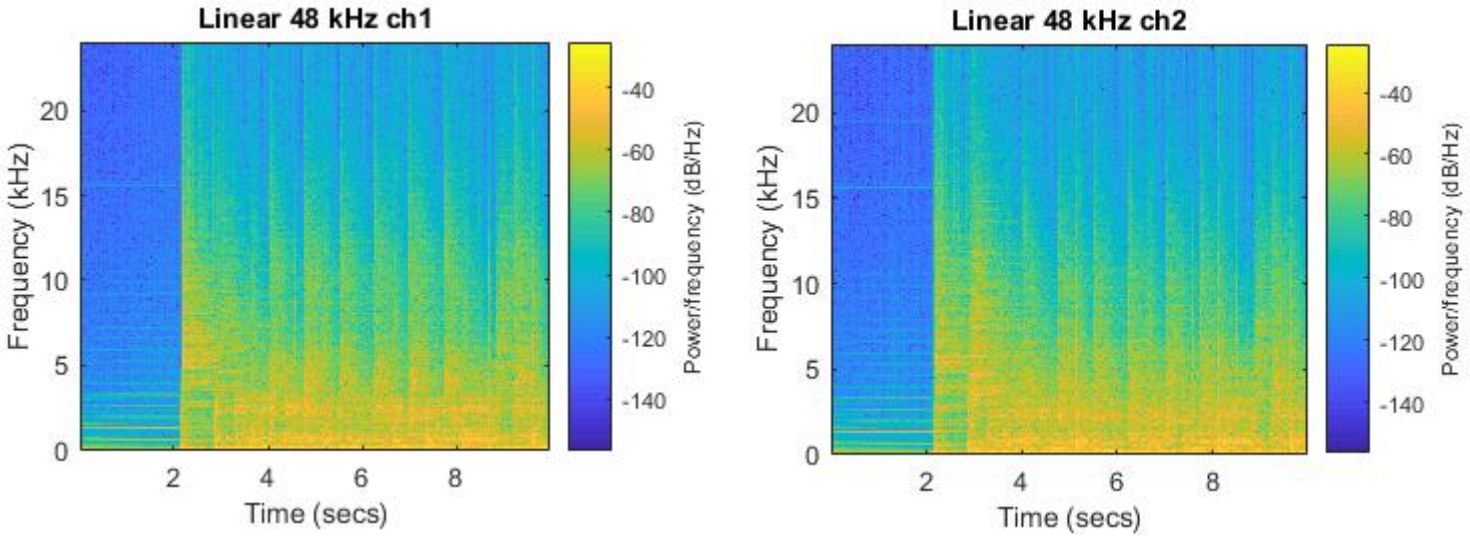

Figure 4.14: Spectrogram of 48 kHz for Linear Interpolation for both channels

Comparing the results for linear interpolation (figure 4.14) the improvements over the original signal are even lesser, but slightly inferior to the results provided by resampling. After observing figure 4.15, where floating point cubic spline interpolation was applied, the results are on par with resampling, and superior to those of linear interpolation. Especially on

higher frequencies, the transition between edges seems to cleared out to a small degree resulting in a slightly smoother transition.

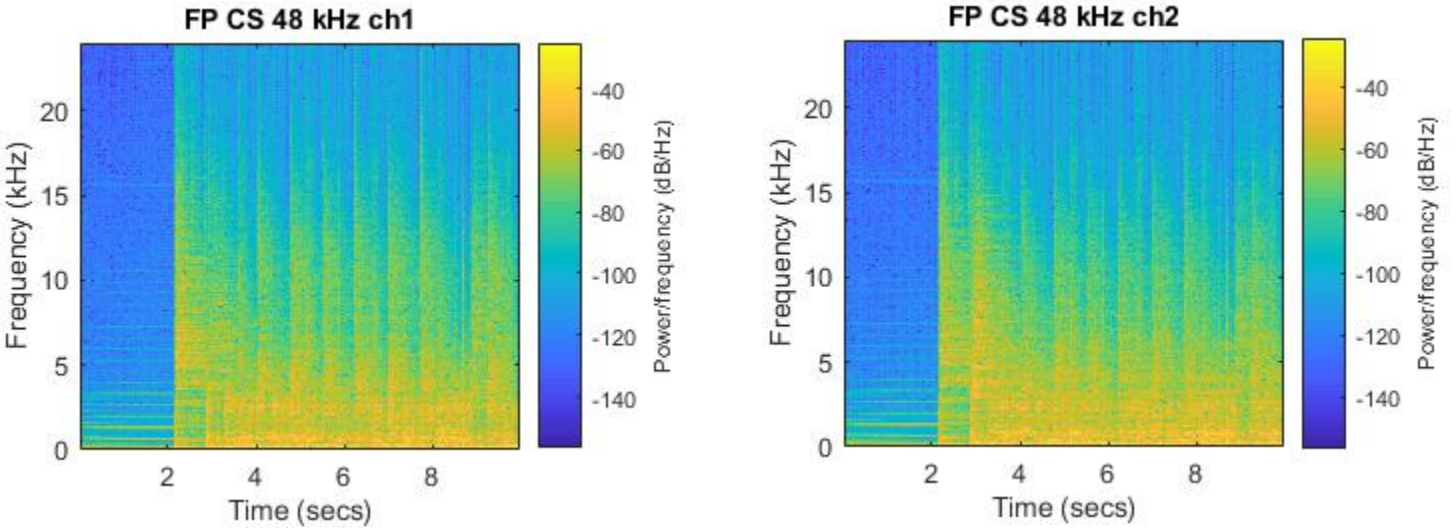

Figure 4.15: Spectrogram of 48 kHz for Floating Point Cubic Spline Interpolation for both channels

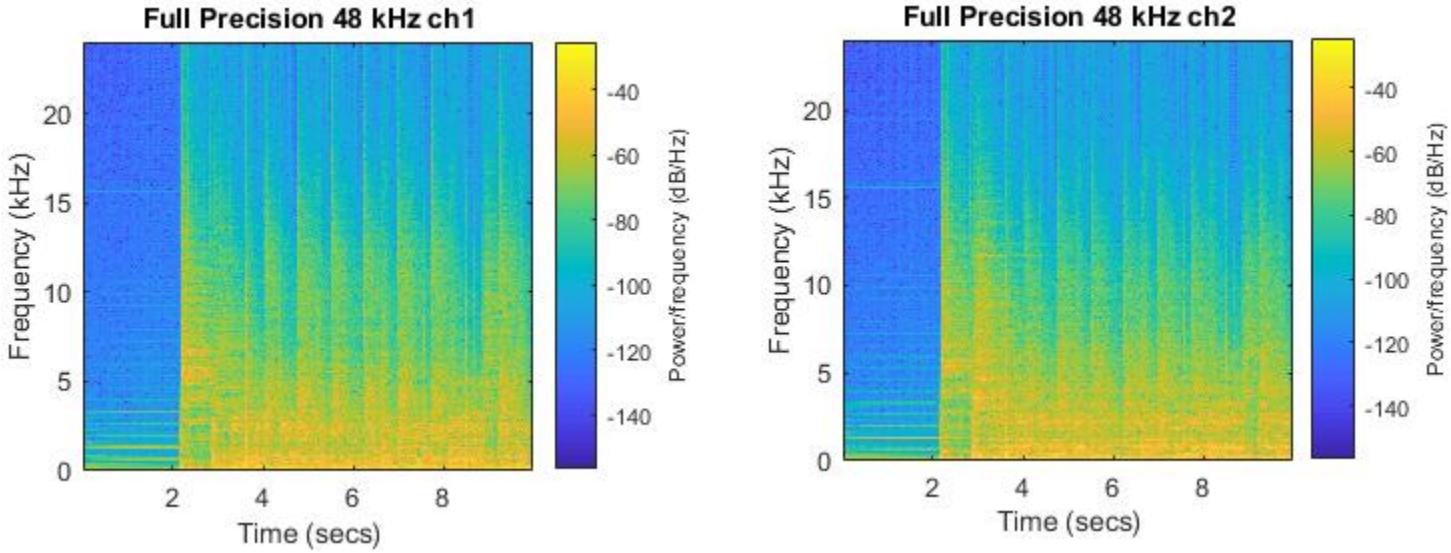

Figure 4.16: Spectrogram of 48 kHz for Full Precision Fixed Point Cubic Spline Interpolation for both channels

In figure 4.16, the full precision version of the fixed point algorithm is shown to have almost the same results as its floating point counterpart, with the only difference being that shortly after two seconds, where the input signal transitions to a more complex and lively tune, the fixed point algorithm provides a slightly smoother transition. This improvement in the transitional area can be attributed to the fixed point algorithm's for point window input due

to the fact that the results of that area are not impacted by points further away, as is the case with the other algorithms being compared here. Adding the results of the limited precision cubic spline algorithm to the comparison, as seen in figure 4.17, the results are equivalent to the full precision algorithm, with the improvement on the transitional area present, although the rest of the spectrogram is not as clean as the floating point version or the full precision version of the algorithm.

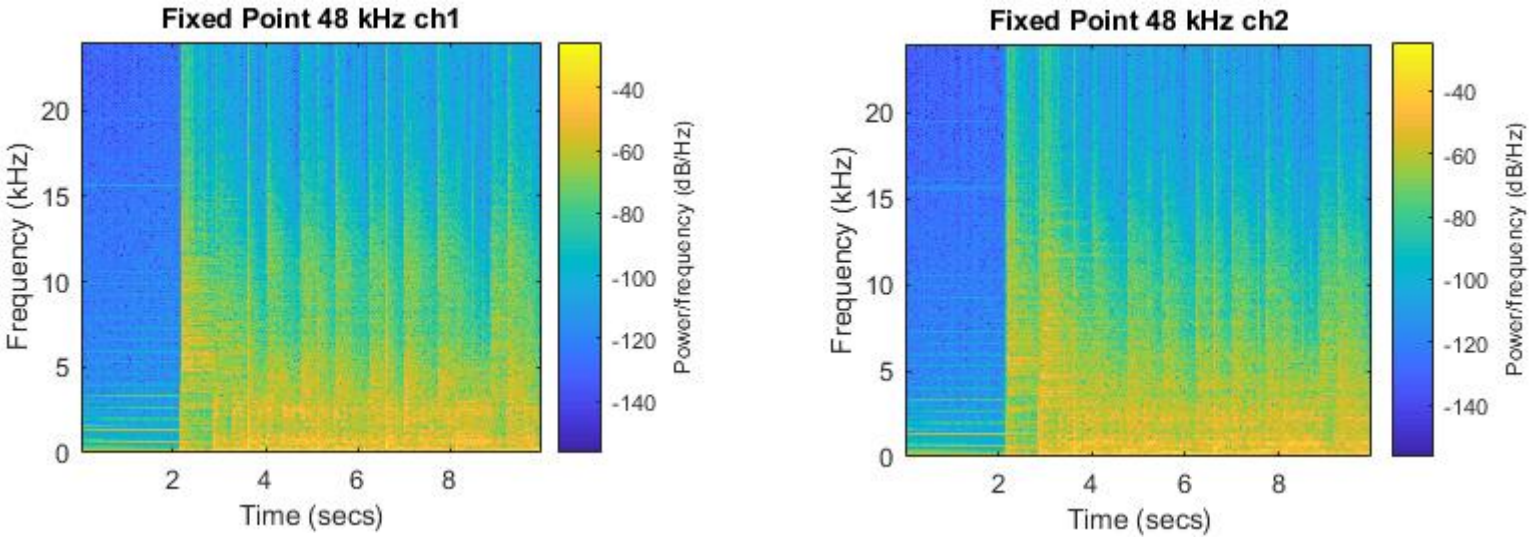

Figure 4.17: Spectrogram of 48 kHz for limited precision Fixed Point Cubic Spline Interpolation for both channels

Increasing the sampling rate to 96 kHz provides more distinguishable improvements over the original signal for all methods, with the exception of the linear interpolation method. The improvements are especially visible when it comes to frequencies above 20 kHz, as the higher the frequencies the more distinguishable the signal's amplitude.

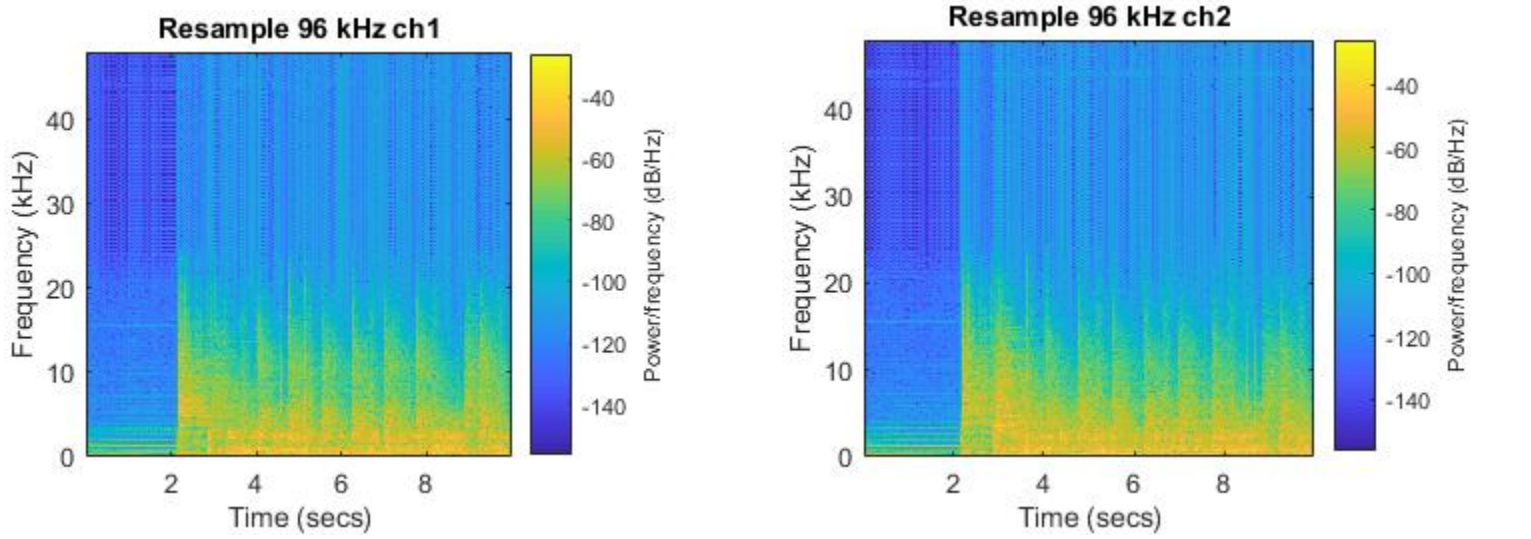

Figure 4.18: Spectrogram of 96 kHz after Resampling

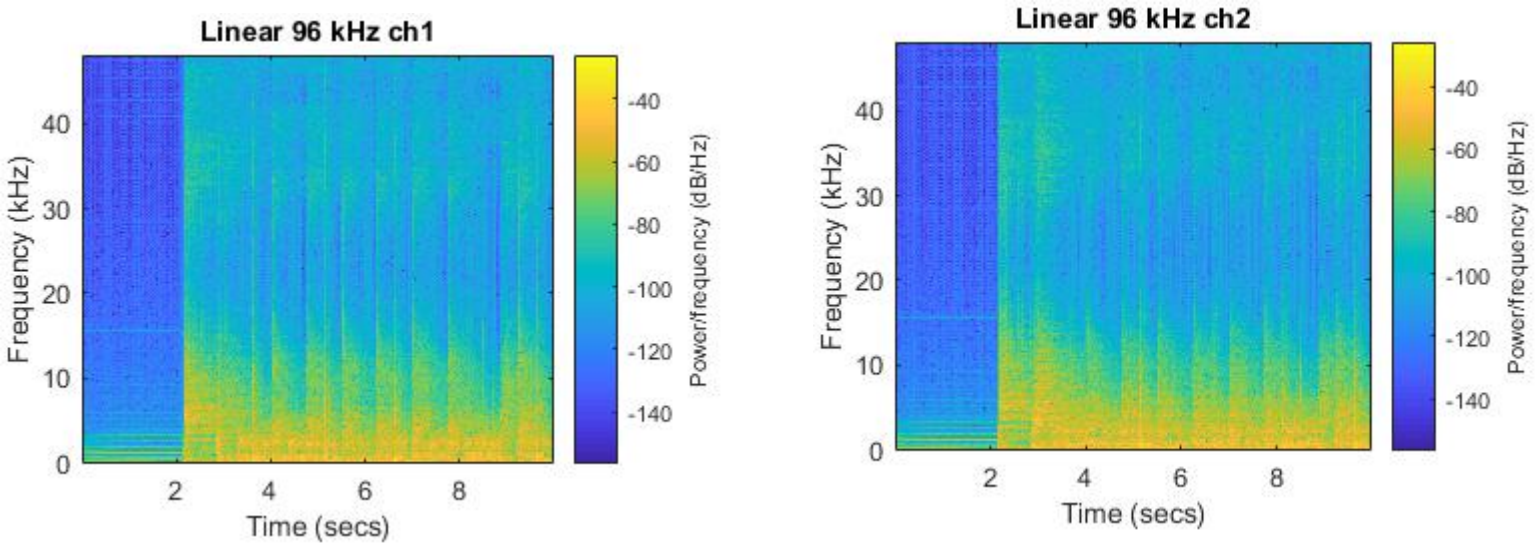

Figure 4.19: Spectrogram of 96 kHz for Linear Interpolation for both channels

Looking at the spectrogram resulting from the linear interpolation algorithm it becomes quite obvious that no significant improvements over the original signal have been made. As for the new frequencies which are over 20 kHz the power over frequency performance is rather poor. In contrast, the cubic spline methods provide less blur between power spikes.

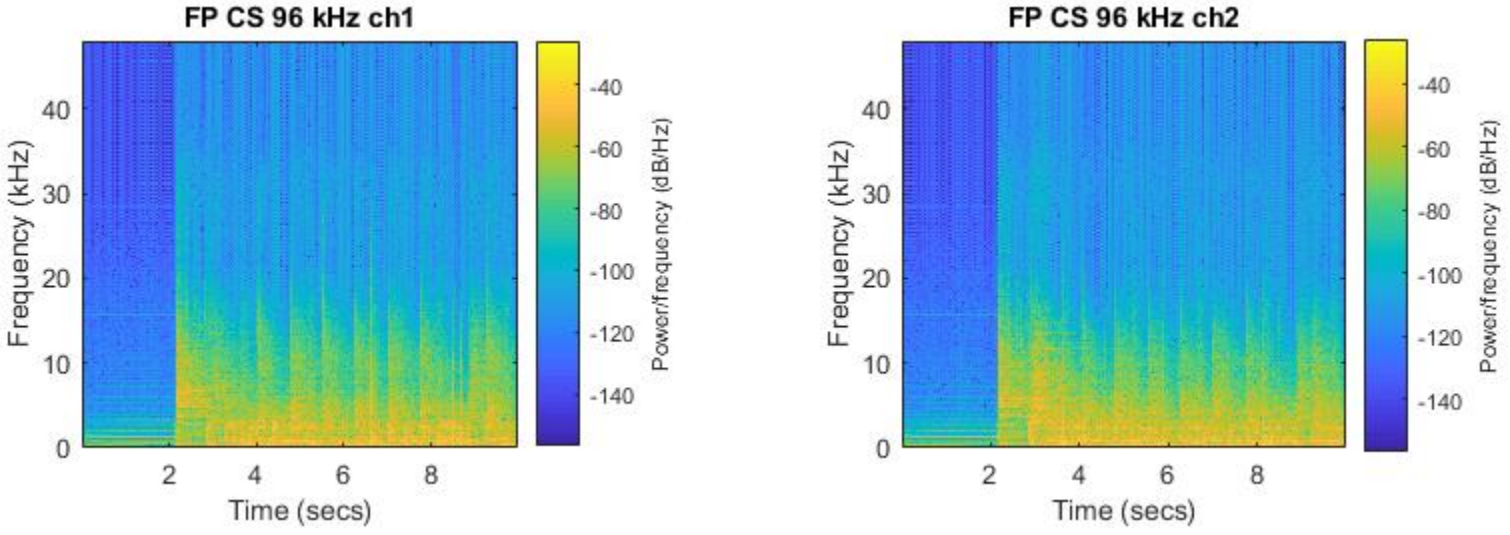

Figure 4.20: Spectrogram of 96 kHz for Floating Point Cubic Spline Interpolation for both channels

Comparing the cubic spline methods with linear interpolation the power spikes have been normalised significantly and are easily distinguishable with the naked eye while at the same time providing more power on higher frequencies than resampling. When it comes to comparing the methods between themselves, the same pattern as with 48 kHz can be

observed. More specifically, the floating point algorithm seems to perform very slightly better than the full precision fixed point algorithm but nothing significant, the same can be said when comparing the limited precision algorithm with the full precision version.

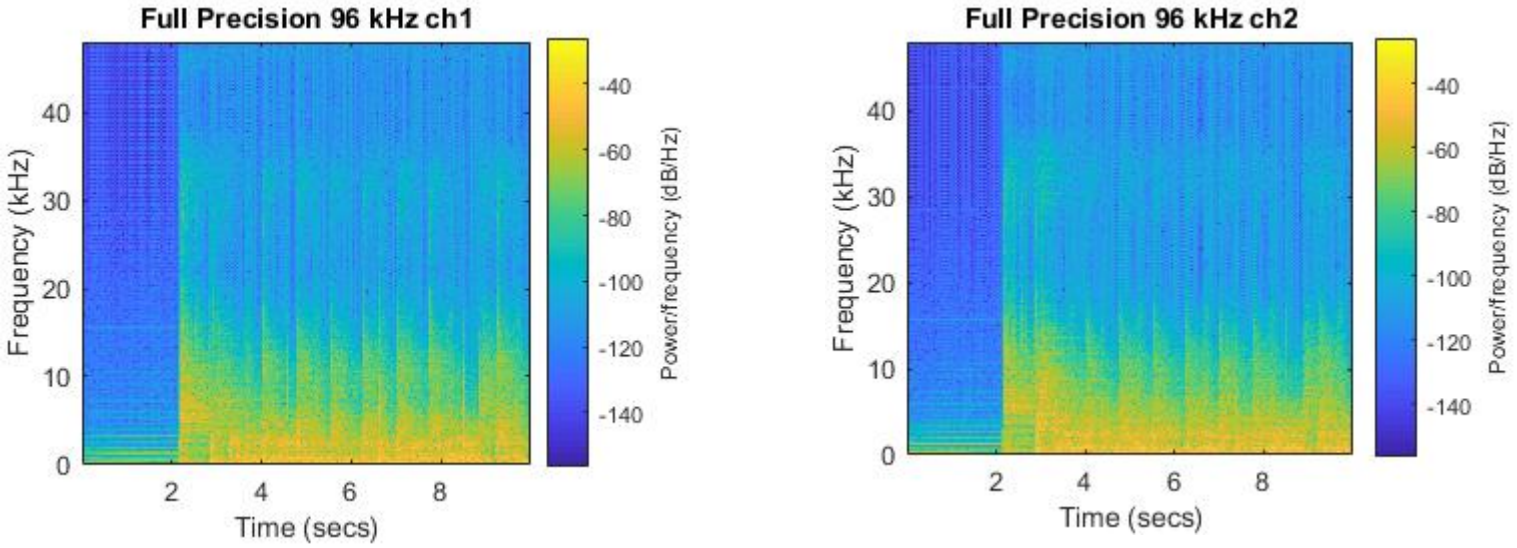

Figure 4.21: Spectrogram of 96 kHz for Full Precision Fixed Point Cubic Spline Interpolation for both channels

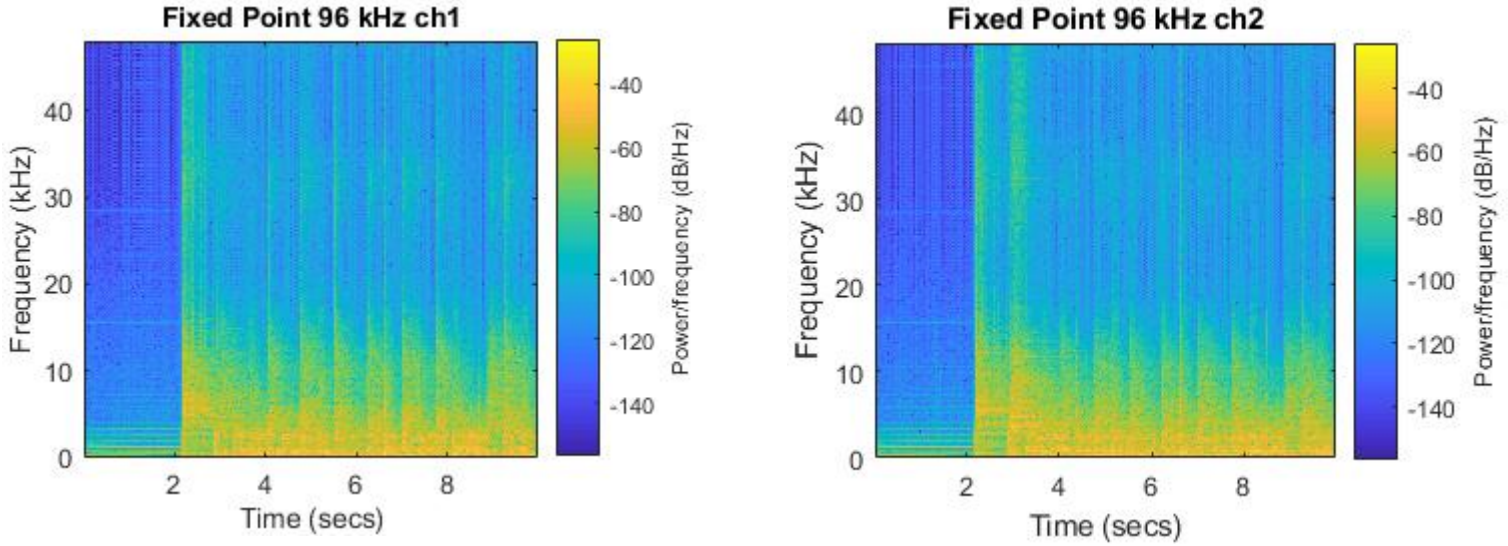

Figure 4.22: Spectrogram of 96 kHz for limited precision Fixed Point Cubic Spline Interpolation for both channels

Moving on to the final sampling rate being tested, 176.4 kHz, the changes are again in line with what was previously observed. Resampling once again provides clear, improvements compared to the original, and linear interpolation seems to be less and less effective as sampling rates increase. Again the cubic spline algorithms have improved, yet similar

results, and seem to retain the characteristics of previous results, albeit the difference between them seems more pronounced.

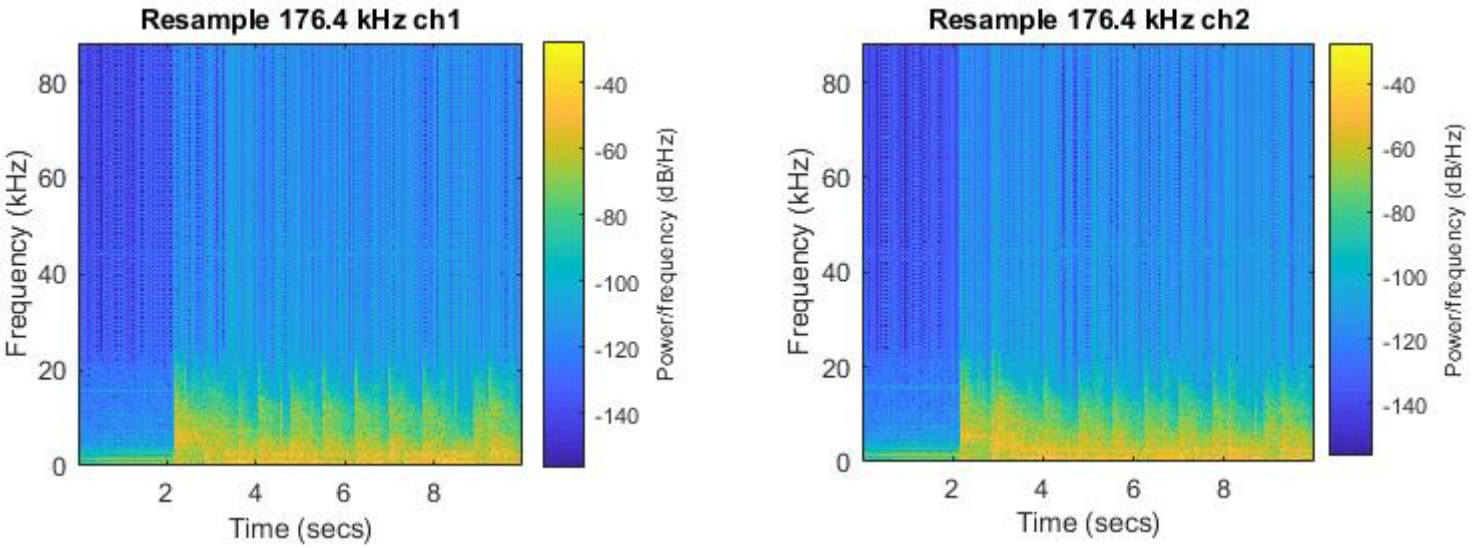

Figure 4.23: Spectrogram of 176.4 kHz after Resampling

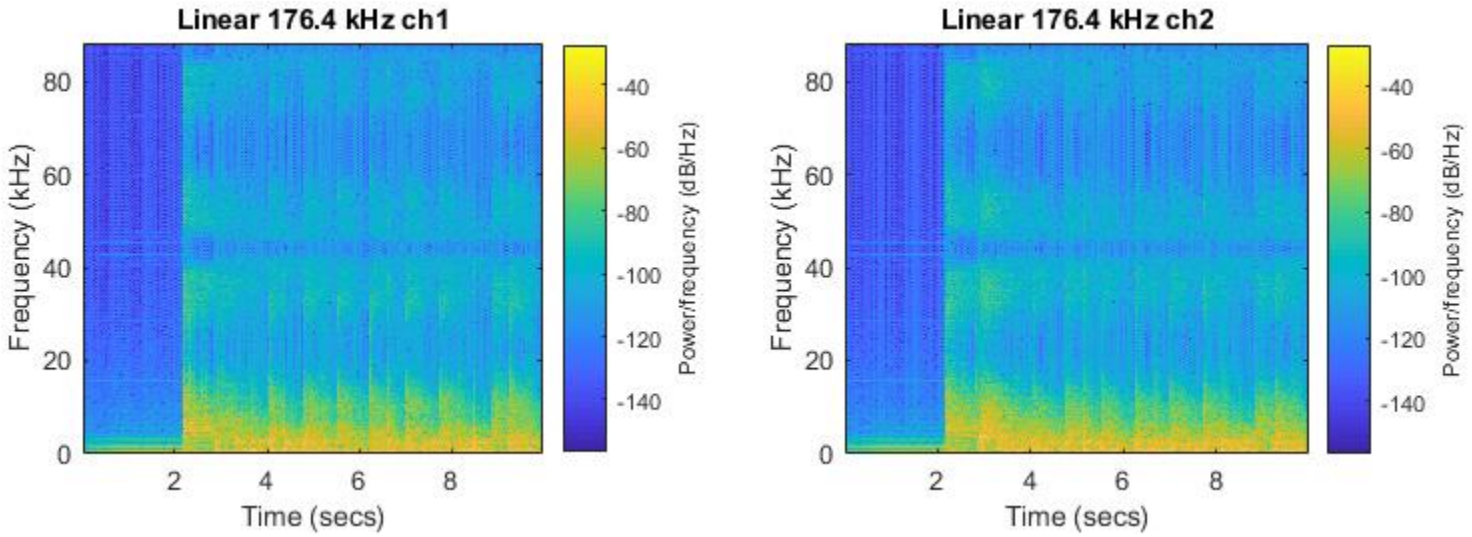

Figure 4.24: Spectrogram of 176.4 kHz for Linear Interpolation for both channels

Another bi-product of the increase in sampling rate is that the differences between the almost identical channels are becoming more pronounced and are more easily distinguishable with one another, while also retaining their basic structure. Additionally, increasing the sampling rate had an effect on the bandwidth of the signal proportional to the *new Fs*/*Fs* ratio. As previously stated, despite the fact that the human ear can perceive up to 20 kHz, if ultrasonic

frequencies, actually impact audio perception then there should be an improvement, however small, on the acoustic experience.

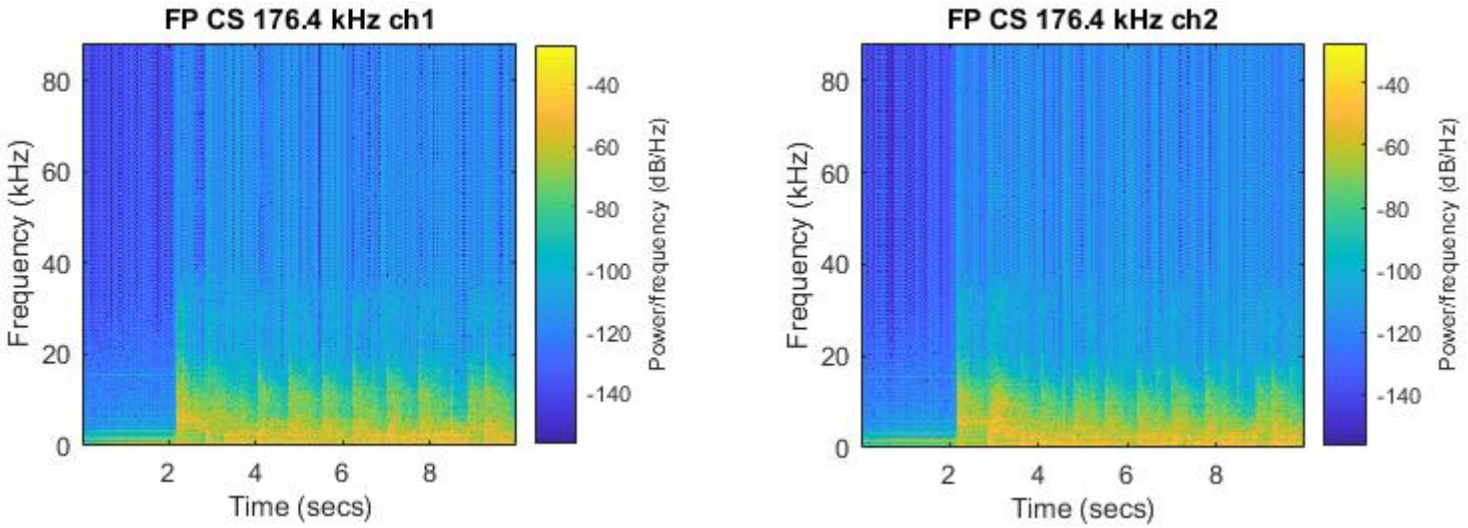

Figure 4.25: Spectrogram of 96 kHz for Floating Point Cubic Spline Interpolation for both channels

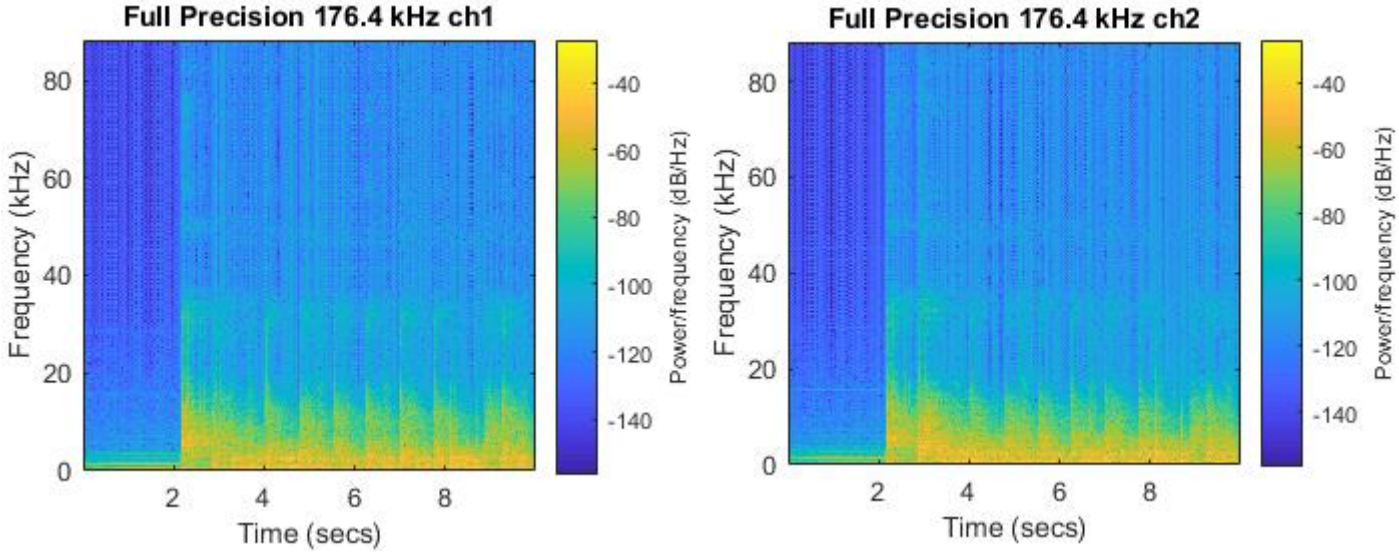

Figure 4.26: Spectrogram of 176.4 kHz for Full Precision Fixed Point Cubic Spline Interpolation for both channels

All in all, as far as spectral analysis is concerned, cubic spline interpolation seems to produce encouraging prospects, especially when compared to linear interpolation. Linear interpolation appears to perform best when the sampling rate is relatively low compared to the original and it is possibly a viable prospect for low cost designs.

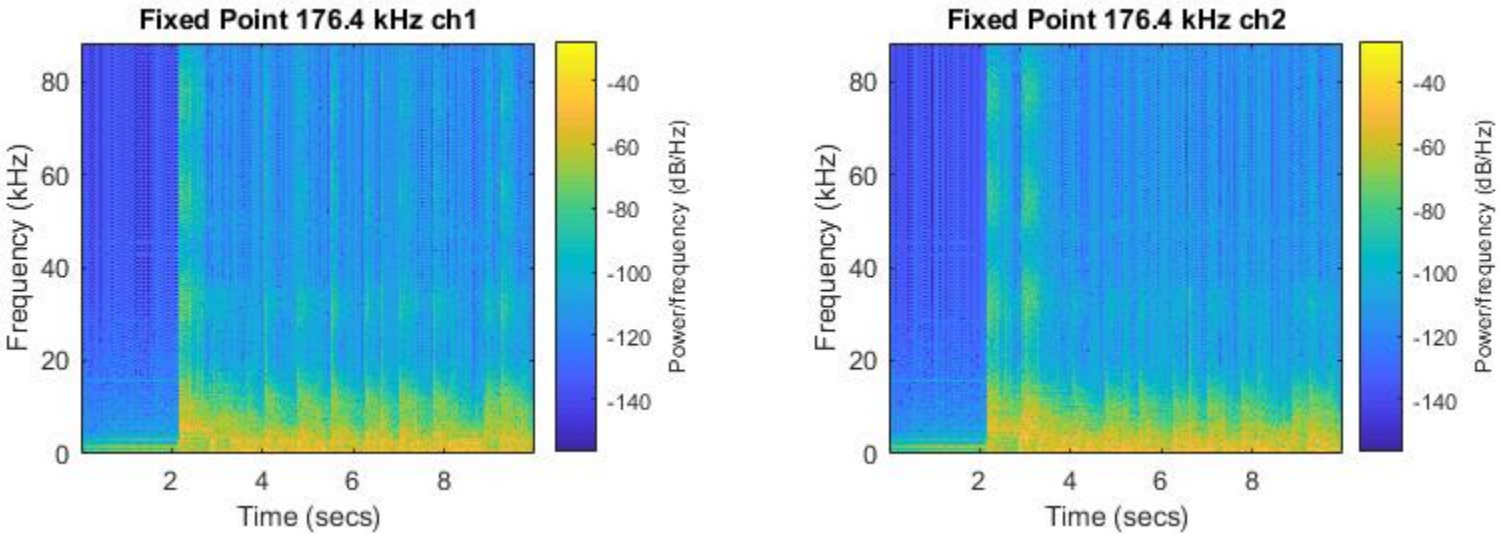

Figure 4.27: Spectrogram of 176.4 kHz for limited precision Fixed Point Cubic Spline Interpolation for both channels

Floating point cubic spline performed slightly better than its fixed point counterparts. This can be attributed to two facts: Firstly, the short length of the input which did not allow the four-point window input of the fixed point algorithms to perform its best by not including points that would negatively affect the results. Secondly divisions in fixed point arithmetic do not have a defined full precision depth, as such the precision of divisions is inferior to floating point divisions. In order to better support this statement a test with a longer audio signal follows.

The next input signal is called if eternity should fail, by iron maiden. This audio signal is ten minutes in duration and will allow for more accurate observations. It is difficult to draw conclusions out of a ten minute spectrogram, as such after the results for the whole signal are processed only two seconds are displayed.

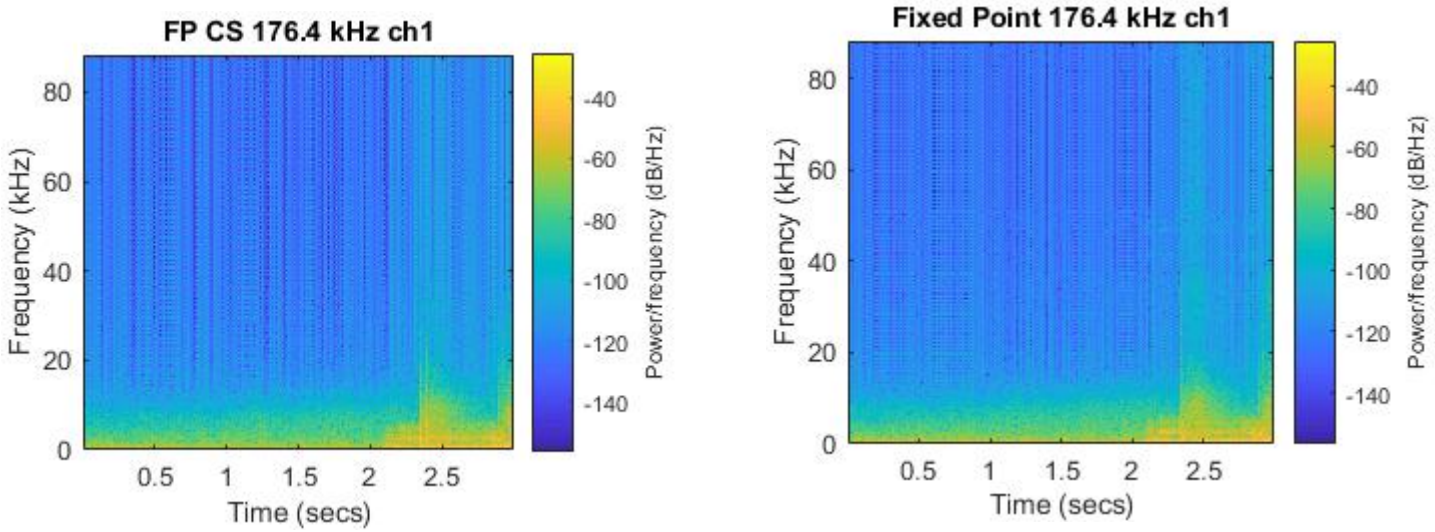

Figure 4.28: Spectrogram of 176.4 kHz for Floating Point Cubic Spline Interpolation and limited precision Fixed Point Cubic Spline Interpolation

On figure 4.28 it can be observed that indeed the two signals, which were produced by floating point cubic spline interpolation and limited precision cubic spline interpolation, with a four point input window, are seen as more similar than the signals previously tested which confirms the effectiveness of the four point window, not only as a necessary adaptation for hardware integration but as a measure that improves numerical accuracy.

4.6 Time Domain analysis

When it comes to analysis of the produced waveforms in the time domain, it will be focused around the curvature of the signals produced by the model. More specifically an interval of 64 randomly chosen data-points will be plotted each time for the same sampling rates as the spectrograms, which were discussed previously, namely: the original frequency of 44.1 kHz and the frequency of 176.4 kHz, since 176.4 kHz stresses out the algorithms the most. As there is no difference between using different audio channels only one of them will be displayed. For the purpose of this demonstration a full length version of Transylvania by iron maiden will serve as input.

In figure 4.29, the results of linear interpolation for a sampling rate of 176.4 kHz are plotted together with the original signal. Linear interpolation produces results that match the original perfectly. This is because Matlab just draws straight lines to connect the points of each table being plotted, and that's exactly how linear interpolation operates by definition hence the identical graph.

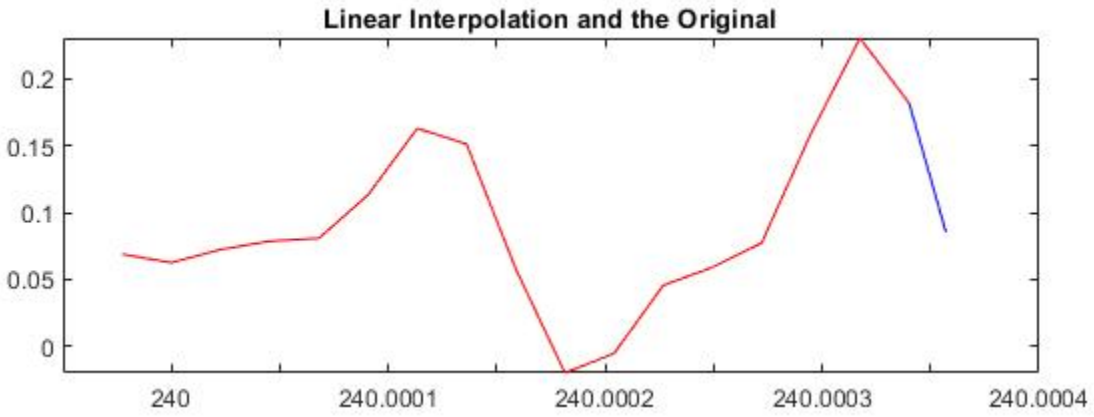

Figure 4.29: Comparison of 176.4 kHz Linear interpolation in blue and the original in red

Moving on to cubic spline interpolation, the floating point results presented in figure 4.30,4.31 and 4.32, are the kind of results expected by cubic spline interpolation. More specifically there is obvious curve fitting on the part of all three versions of the algorithm, creating the curves expected of an analogue signal. As with the spectrograms, it appears that as expected the floating point version provides slightly better results, with the full precision algorithm as a

close second and the more limited iteration third. It is confirmed that once again the difference in the results is not significant enough to warrant neither a floating-point with a full tridiagonal matrix nor a full precision fixed point iteration, since the resources required for either of these two algorithms will be significantly higher than the more modest limited precision and four data point input version.

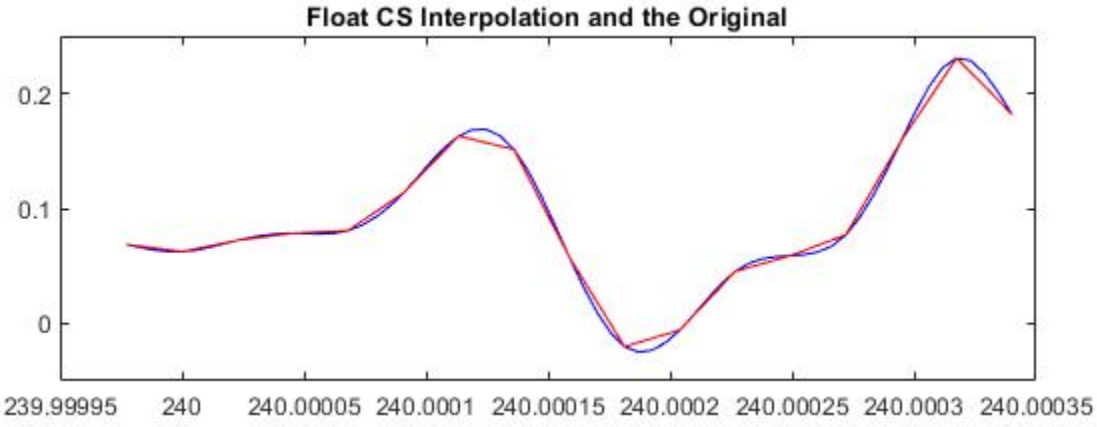

Figure 4.30: Comparison of 176.4 kHz floating point cubic spline interpolation in blue and the original in red

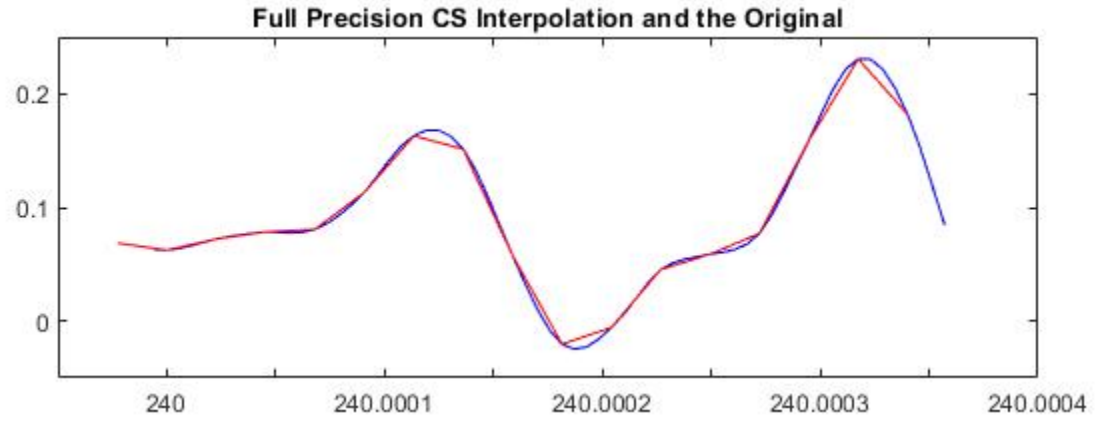

Figure 4.31: Comparison of 176.4 kHz full precision fixed-point cubic spline interpolation in blue and the original in red

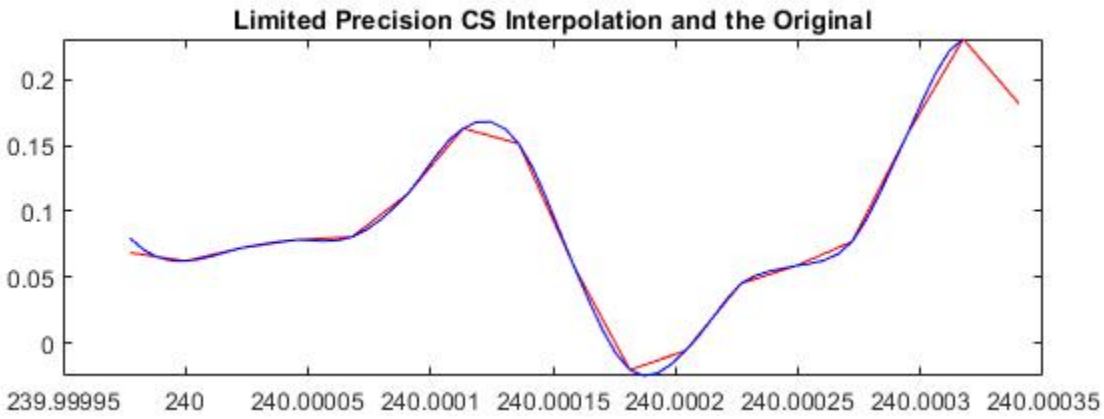

Figure 4.32: Comparison of 176.4 kHz limited precision fixed-point cubic spline interpolation in blue and the original in red

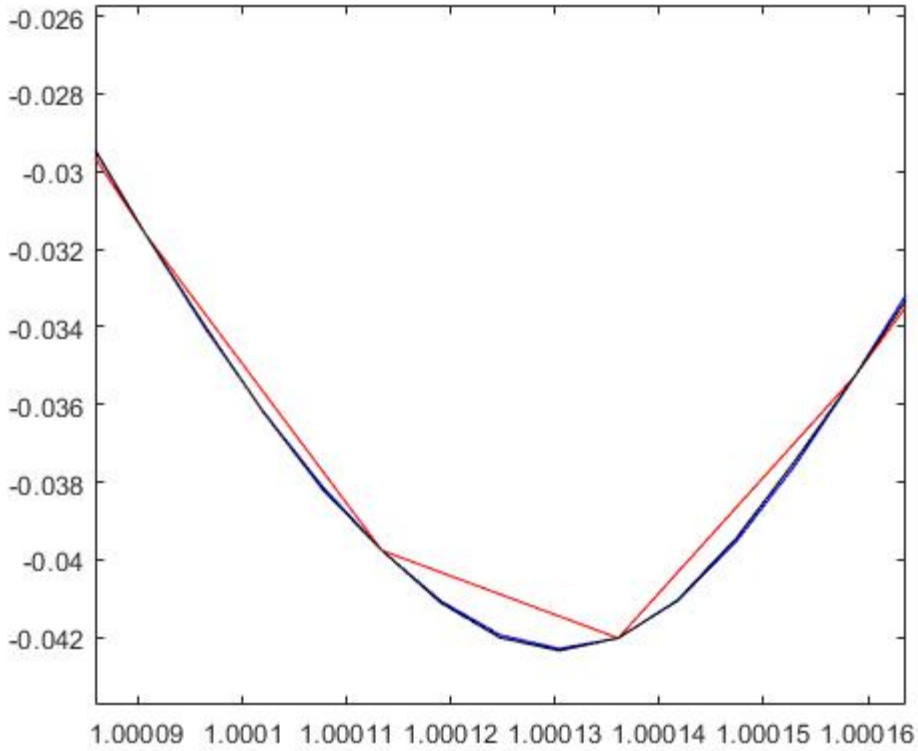

Figure 4.33: All the Cubic Spline algorithms and the original signal

Finally, all three versions of the cubic spline algorithm are plotted together on a random part, and compared with the input, which is on red. The results again are similar.

4.7 Acoustic Tests

After inspecting the behaviour of the algorithms in both time and frequency domains, the next step is to perform acoustic tests, since the subject of this work primarily concerns music and audio at large. Normally for this kind of testing requires setting up a proper ABX trial system. ABX trials require a digital to analogue converter connected with speakers or headphones, paired with an ABX comparator, to secure unbiased responses by test subjects. None of these resources were procured and as a result, the data presented here are this student's interpretations, using common PC speakers to playback the .wav files. The .wav files, for resampling linear interpolation and floating point cubic spline interpolation were created by utilizing Matlab's writewav() function. The model's fixed point .wav files were created by exporting the data to a .txt file, and then importing them on a c++ program to create .wav files. The original audio files used as inputs are the same files used in previous tests meaning: Book of Souls, Transylvania and If Eternity should Fail, all three by iron maiden.

There is not much of a point in listening to only one channel at a time since the songs were recorded to be played in stereo mode, as such both channels will be listened to as it is expected. The sampling rates being tested are 48 kHz, 96 kHz and 176.4 kHz. The first track to be tested is Book of Souls. As with the spectrograms the re was no particular difference between any algorithm, in fact it was difficult to distinguish it from the original. There are no noticeable differences with the other two audio signals either.

Increasing the sampling rate to 96 kHz had overall, a larger impact than 48 kHz. Linear interpolation was indistinguishable from the original, but the cubic spline algorithms and resampling appear to result in in "deeper" and "fuller" sound, seemingly slightly increasing the quality, although, it should be noted that such an increase could be the result of psychoacoustic factors. Finally, there was a slight difference when using headphones, in the way the lower frequency sounds, like the bass and the bass-drum, sounded like, but it might not have been consistent because as the song progressed it became less obvious.

Setting the sampling rate at 176.4 kHz there were some small changes as far as linear interpolation is concerned, since it seemed like it produced a "warmer" sound. In addition, the other methods managed once again to seemingly produce a "deeper", "warmer" and "fuller" sound. In some cases they also seemed to improve the output a bit making it even "clearer" at times, especially on the song Transylvania which was famously recorded with lower quality equipment.

To elaborate further on the results, all the algorithms appeared to slightly improve the acoustic experience. This improvement could very easily be attributed to psychoacoustic effects that result from increasing the sampling rate of the audio signals, since the higher the

sampling rate is set the more noticeable and improved the results. Resampling and cubic spline interpolation appeared to provide equivalent acoustic experiences, and where on par on previous tests. Finally, the results of the acoustic tests are, obviously, not final, since normally a proper ABX trial with multiple subjects, preferably well versed in music should be conducted, in order for the results to be entirely legitimate.

4.8 Regaining Lost Signal Information

In order to determine wether or not any information can be recovered by a heavily distorted audio file, a harsh test was devised. This test includes removing whole data-points from an original signal. Using cubic spline interpolation the objective is to determine whether or not any data can be recovered, by comparing the original, the distorted and the interpolated signal. The solution tested here is a modification of the limited precision cubic spline interpolation algorithm.

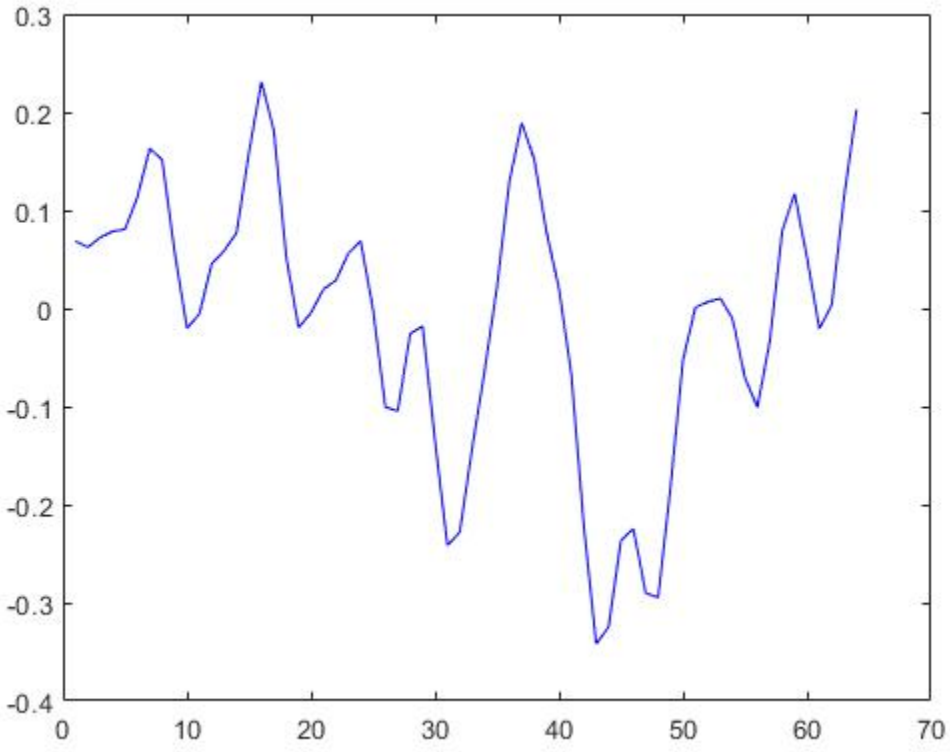

Figure 4.34: The original signal

The results of the test are seen in figures 4.33,4.34 and 4.35. Observing figure 4.34, which depicts the distorted signal, and comparing it to figure 4.33, which displays the original signal, it becomes clear that the signal was distorted significantly, and reproducing the signal through speakers yields an unrecognizable sound wave.

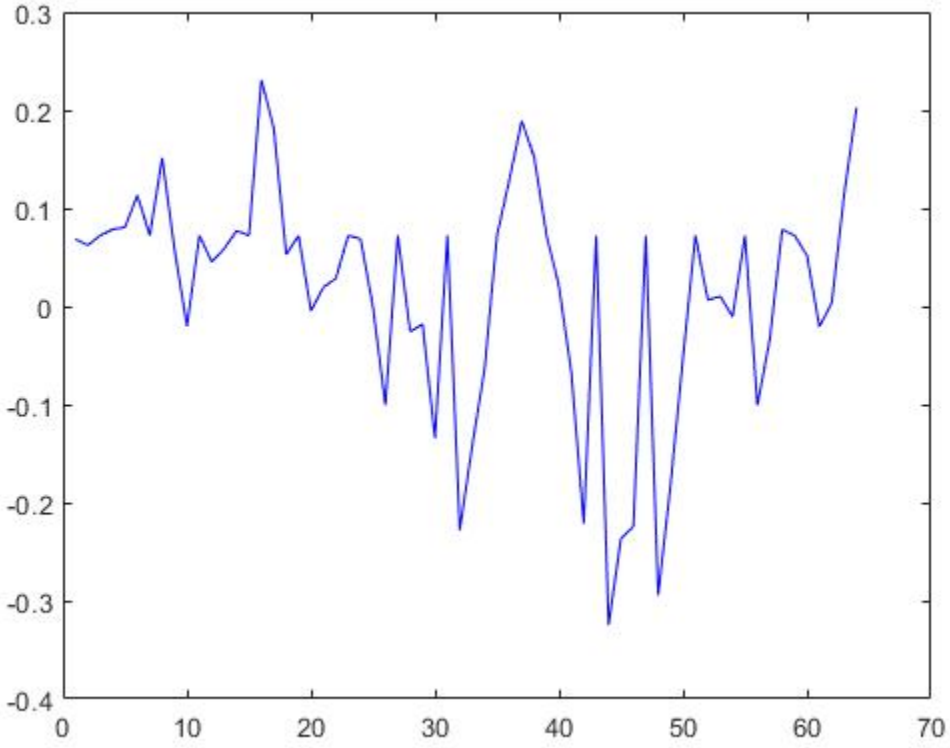

Figure 4.35: The distorted signal

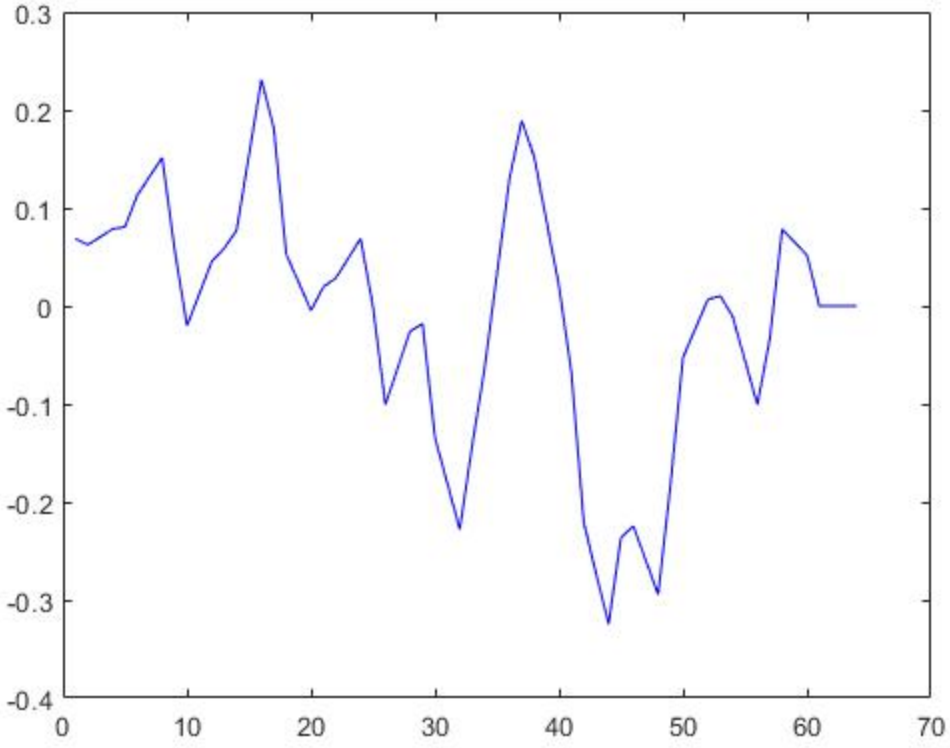

Figure 4.36: The interpolated signal

Comparing figure 4.35, which displays the distorted signal after cubic spline interpolation has been applied, the results of the signal resemble the original closely albeit with small small differences, which is to be expected, considering the awful state of the distorted signal. By this comparison, it can be confirmed that a small amount of the original data can be recovered by using cubic spline interpolation and perhaps other interpolation methods. However when the signal is played back is only slightly recognizable, which means that the whole of 16 bits of missing data every for data-points is out of reach for the algorithm, which is not surprising.

Calculation the nominal SNR of the signals, and considering the original signal as the clean and pure signal the results are reinforced. The distorted signal, in this case, yields an SNR of 3.5309 while the interpolated signal yields a 17.9001, which represents a considerable improvement as a result of the appliance of cubic spline interpolation. The SNR was calculated using Matlab's snr(x,y) function, where x represents the signal whose SNR needs to be calculated and y represents the signal's noise.

Chapter 5:
Embedded System Design

With the development of a fixed-point and hardware friendly, mathematical model for cubic spline interpolation, the next obvious step is to design an embedded system on an FPGA. Tools to be used in this thesis are made by Xilinx and are part of their webpack licence. The design of the module responsible for cubic spline interpolation will be attempted using Vivado High Level Synthesis (HLS), for no other reason than this student's lack of prior experience with the tool.

5.1 Hardware and Design Tools

Embedded systems are complex. Hardware and software portions of an embedded design are projects in themselves. Merging the two design components so that they function as one system creates additional challenges. In tandem with an FPGA design project, the overall complexity of the project increases drastically. For the purposes of this design a Zedboard kit will be used, which is is a complete development kit utilizing the Xilinx Zynq-7000 all programmable SoC. Using an SoC in combination with an FPG reduces the overall complexity of the design process by offering an Arm Cortex-A9 dual core, along with programmable logic. Thus, the tools necessary for the simplification of the design process are the Vivado design suite, which includes both regular Vivado and Vivado HLS, and the Vitis software platform. This combination of tools offers hardware and software application design, debugging capability, code execution, and transfer of the design onto actual boards for verification and validation[17].

The first thing needed for a good understanding of the functionality of the tools is a proper understanding of their workflow, which is shown on figure 5.1. To elaborate further, the first step is designing the necessary modules in Vivado HLS. Vivado HLS serves as a compiler providing a programming environment similar to those available for application development on processors. The main difference is in the execution target of the application, which in this case is an FPGA. Vivado HLS enables for optimizations for throughput, power, and latency without the need to address the performance bottleneck of a single memory space and limited computational resources[17].

The programming language used in the tool is C\C++, with application code targeting the Vivado HLS compiler using the same categories as any processor compiler. Vivado HLS analyses all programs in terms of operations, conditional statements, loops and functions[18].

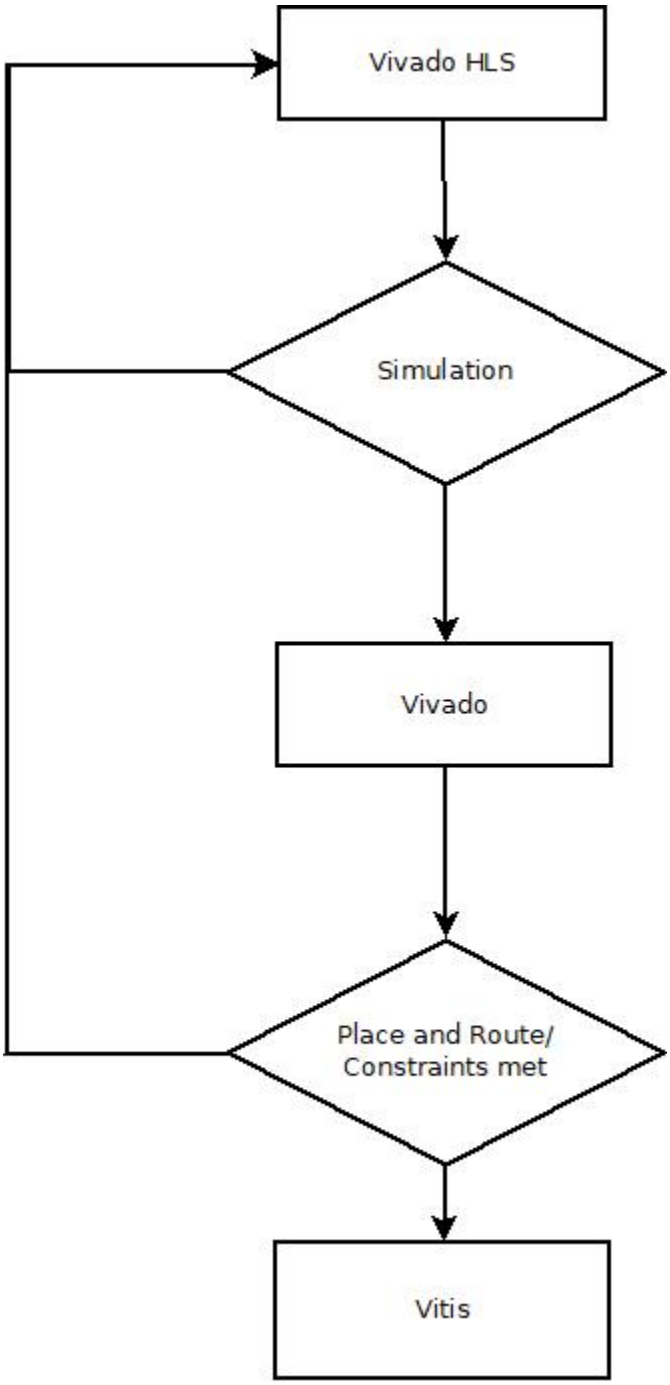

Figure 5.1: The workflow of the design process

As far as operations are concerned the main difference between normal C\C++ and its version targeting the Vivado HLS compiler can be seen in a comparison between figures 5.2 and 5.3, where the same snippet of code is executed on a processor and on an FPGA respectively, with the snippet being:

```
A[i]=B[i]*C[i];
D[i]=B[i]*E[i];
F[i]=A[i]*D[i];
```

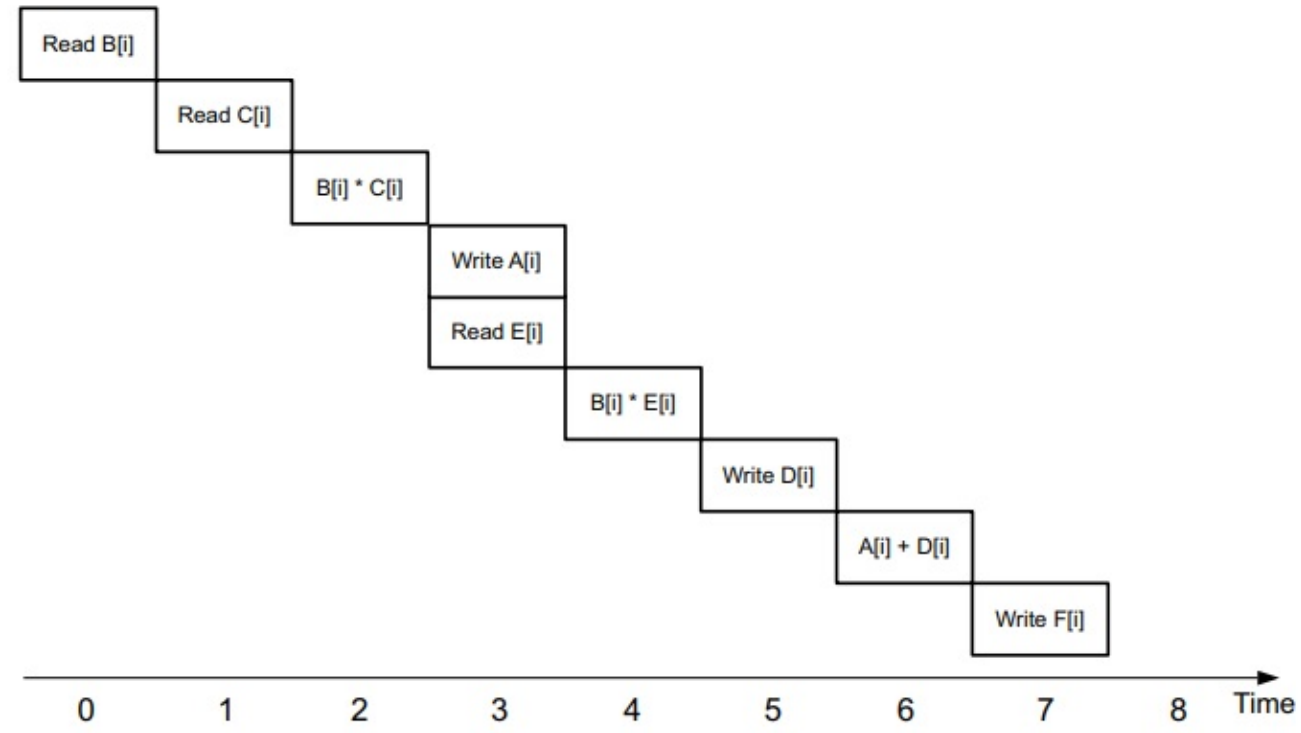

Figure 5.2: Processor execution of example code(image source:[18])

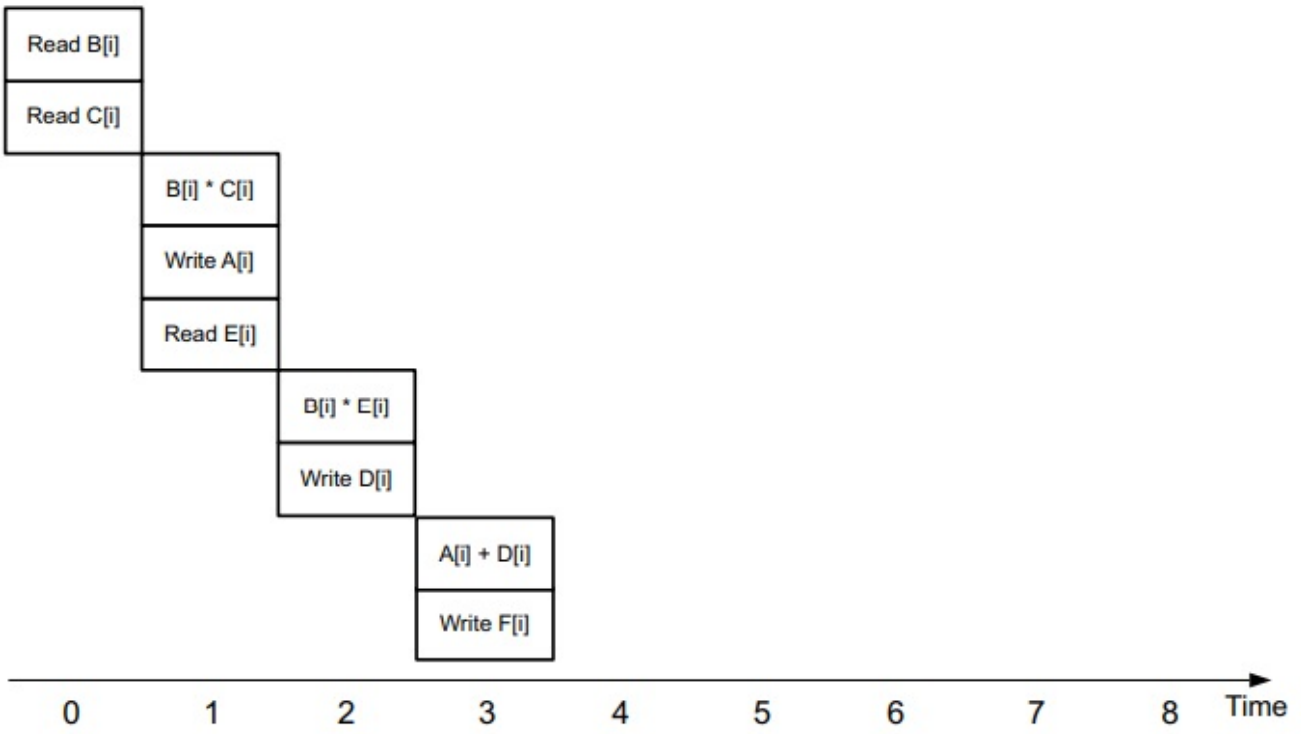

Figure 5.3: FPGA execution of example code(image source:[18])

The application profile, depicted in figure 5.2, focuses only on the EXE stage of instruction processing in a central processing unit (CPU). This is the only stage in instruction processing that is shared between processors and FPGAs. In this example, the execution trace is sequential due to the execution platform, not the algorithm. Based on the algorithm, the values of A[i] and D[i] can be computed in any order or at the same time. The only algorithmic restriction is that both of these values must be computed before F[i].When using the default settings in Vivado HLS, the resulting execution profile is similar to that of the processor in that the multiplications and addition occur in sequential order. The reason for this default behaviour is to minimize the number of building blocks required to implement the user application. Although an FPGA does not have a fixed processing architecture, each device has a maximum number of building blocks it can sustain. Even with the default behaviour, the implementation outperforms the processor execution due to the custom memory architecture created for the algorithm[18].

When it comes to conditional statements, such as C's typical if statement, a processor will have to execute a branch operation, which may or may not result in a context switch, as such resulting in algorithmic dependencies and impacting performance. In an FPGA, a conditional statement does not have the same potential impact on performance as in a processor. Vivado HLS creates all the circuits described by each branch of the conditional statement. Therefore, the runtime execution of a conditional software statement involves the selection between two possible results rather than a context switch[18].

Concerning loops, a processor is forced to schedule loop iterations sequentially. As shown in figure 5.4, a loop that requires four clock cycles per iteration, will approximately need forty cycles to complete a loop of ten iterations. In contrast, HLS does not have that limitation. Because HLS creates the hardware for the algorithm, it can alter the execution profile of a loop by pipelining iterations. Loop iteration pipelining extends the concept of operation parallelization from within loop iterations to across iterations, by basically performing loop unrolling, as seen in figure 5.5. With the purpose of reducing iteration latency, the first automatic optimization applied by Vivado HLS is operator parallelization to the loop iteration body. The second optimization is loop iteration pipelining. This optimization requires user input, because it affects the resource consumption and input data rates of the FPGA implementation. HLS can parallelize or pipeline the iterations of a loop to reduce computation latency and increase the input data rate. The user controls the level of iteration pipelining by setting the loop initialization interval (II). The II of a loop specifies the number of clock cycles between the start times of consecutive loop iterations. However the desired II cannot always be forced through by HLS, and the user is often called upon to implement optimizations manually[18].

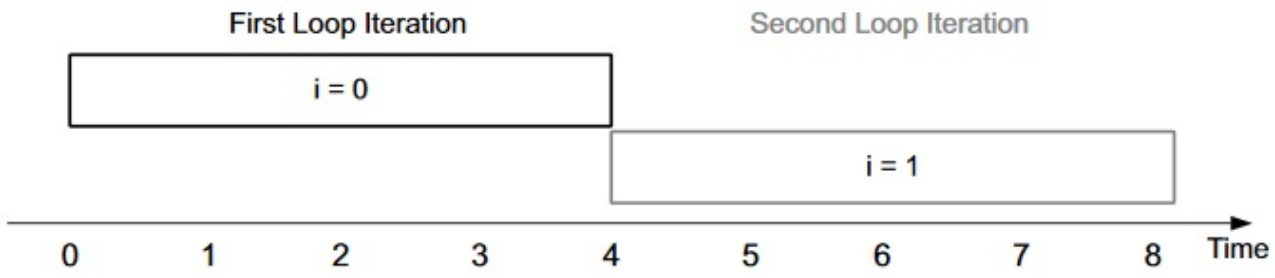

Figure 5.4: Loop execution on a processor(image source:[18])

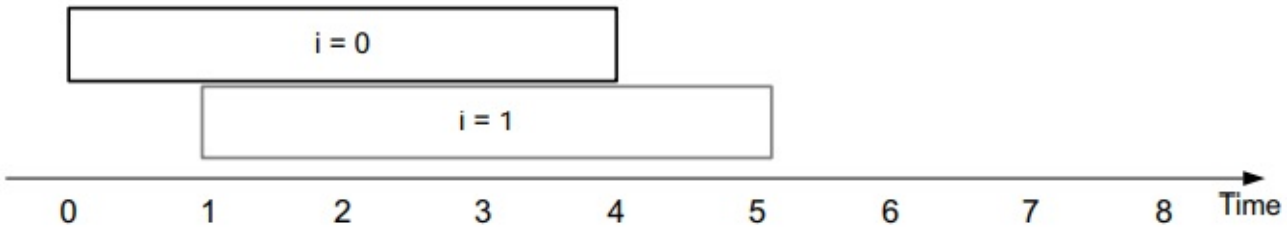

Figure 5.5: Loop execution on an FPGA(image source:[18])

Of course this is an FPGA design and these principles explain the function of the tool which is used to develop independent modules. In order for these modules to function properly, they need to be connected to a more expansive design, and in order to achieve that connection and communication, an interface between this module and the larger design must be implemented. In C based design, all input and output operations are performed, in zero time, through formal function arguments. In an RTL design these same input and output operations must be performed through a port in the design interface and typically operates using a specific I/O protocol. There are multiple I/O protocol types, the type chosen for the present work is provided through SystemC designs, where the I/O control signals are specified in the interface declaration and their behaviour specified in the code. When the top-level function is synthesized, the arguments and/or parameters of the function are synthesized into RTL ports [19].

Most of the interfacing is done using what is called pragmas. Pragmas are directives directly embedded on the source file. Of course when a directive is applied to an interface, Vivado HLS applies the directive to the top-level function, because the top-level function is the scope that contains the interface.

After the design of the module is complete it can be simulated as a normal C\C++ by using a C\C++ test bench, which serves as the main of the program and the module as a function. Vivado HLS offers the ability to execute what is called a hardware co-simulation where the results of the software simulation are compared to the results of Vivado's simulation, complete with waveforms available. Finally the resulting module is exported as a Vivado IP (intellectual property).

Moving to Vivado the, the first step is to create a block design where the IP generated by Vivado HLS is imported. Next the IP's ports should be suitably connected using Xilin's or other custom IPs as required in order connect the design with the FPGA's SoC, which is responsible for the interfacing between the RTL design and the FPGA's DDR and other peripherals. There will be a short example where this procedure is explored further. Finally a debug module can be added so that when the design is downloaded on an FPGA the waveforms can be monitored on Vivado's hardware manager. Obviously this module must be removed from the final design.

After the block design is complete, the next step is synthesis and implementation of the design. During implementation Vivado determines how the design will be placed and routed on the specified FPGA, and it is during this step that the final resource utilizations are calculated and the constraints of the design are enforced. If there are not enough resources on the FPGA to accommodate the design or the timing is not met, then the modules designed by Vivado HLS need to be adjusted in order satisfy the requirements of the platform. When dealing with resource overutilization a common method is to try and make the algorithm of the module more efficient or, when it can't be improved further on that regard, to reduce arithmetic precision where it is affordable to do so. When dealing with timing issues the most common solution is to try and make the pipeline, if present, more efficient by reducing its critical path.

The next step is exporting the Vivado design onto Vitis, in the form of a ,XSA file. The Vitis IDE is designed to be used for the development of embedded software applications targeted towards Xilinx embedded processors and aids in pairing them with hardware designs developed in Vivado. Vitis overall serves as an upgrade to the older Xilinx SDK, in this software platform, two new concepts are introduced in the workspace: the platform project and the system project. In the SDK workspace, the hardware specification, software board support package (BSP), and application all live at the top level. The SDK BSP concept is upgraded to a domain in Vitis. A domain can refer to the settings and files of a standalone BSP, a Linux OS, a third party OS/BSP like FreeRTOS, the choice of a domain is instrumental and defining int overall design process. A platform project groups hardware and domains together. Boot components like FSBL are automatically generated in platform projects. A system project groups together applications that run simultaneously on the device. Figure 5.6 shows the tool's workspace structure as described above[20].

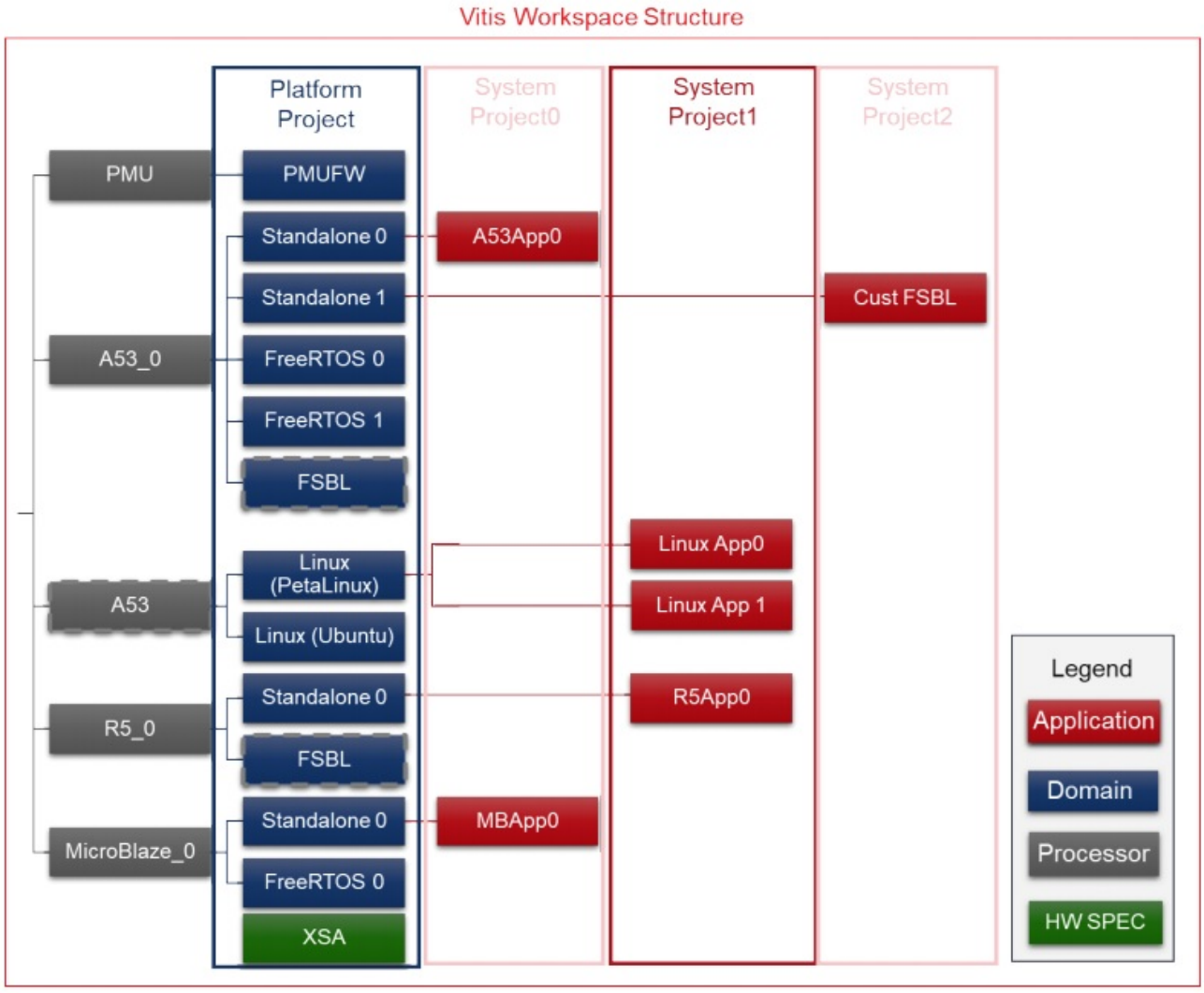

Figure 5.6: The Vitis workspace structure(image source:[20])

Now that the XSA file from Vivado is available a platform project can be created in Vitis. A platform project is the container for the hardware platform, runtime library, the settings for each processor, and the bootloader for the device. It can be as simple as a standalone board support package, or a combination of different kinds of runtime configurations. With the platform project, containing the hardware established it needs to be pair with an application project, with one or more contained in a system project, depending on the number of on-board processors/cores. As such a software application project needs to be created first. Software application projects are the final application containers. The project directory that is created contains, or links to, C/C++ source files, executable output file, and associated utility files, such as the Makefiles used to build the project[20].

With this steps done, the host program can now be developed and the design can finally be used to program the targeted FPGA. In this work, the method of programming the Zedboard is using two USB cables and setting it to its JTAG configuration, whose jumper configuration can be seen on figure 5.7.

However, depending on the needs of the application, there might arise the need for extra heap or stack memory, these sizes can be adjusted by the linker script. To elaborate further, the application executable building process can be divided into compiling and linking. Linking is performed by a linker that accepts linker command language files called linker scripts. The primary purpose of a linker script is to describe the memory layout of the target machine, and specify where each section of the program should be placed in memory and the size of other memory regions such as the aforementioned stack and heap[20].

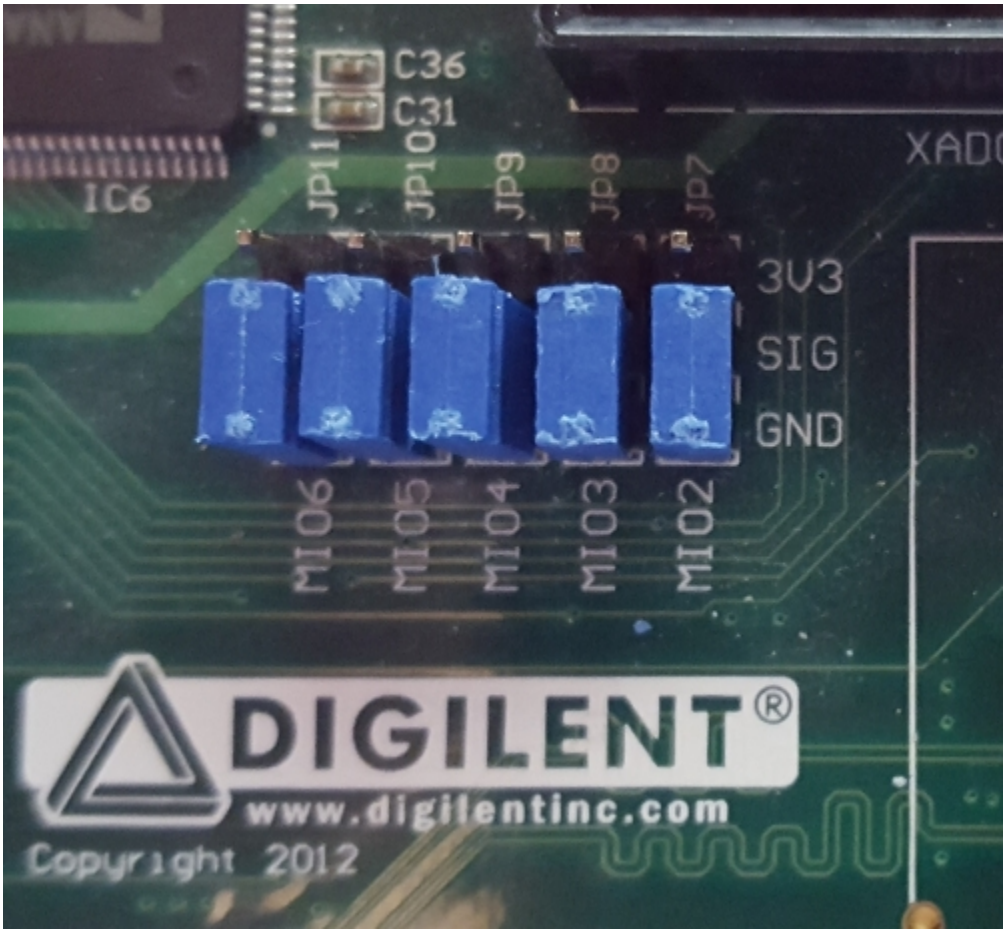

Figure 5.7: Zedboard JTAG jumper configuration(image source:[40])

5.2 Additional Familiarization

As first step in putting these design principles in practice a small project that would serve as a tutorial was designed. The module developed in Vivado HLS is simple, receiving an array of

integers as input add 100 to that number and have another array as the output. The idea for this tutorial was to help with familiarization with Vivado HLS and its interfacing practices, as such a protocol had to be selected to fetch data from the processor and the DDR RAM and write the results on a different area of the DDR. The protocol chosen was AXI.

5.2.1 AXI Protocol

AXI, or Advanced eXtensible Interface, is part of ARM AMBA, a family of micro controller buses, with its latest version being AXI4. There are three types of AXI4 interfaces: AXI4, meant for high-performance memory-mapped requirements, AXI4-Lite, intended for simple, low-throughput memory-mapped communication, commonly to and from control and status registers, and AXI4-Stream, used for high-speed streaming data. For the purposes of this work AXI 4 and AXI-Lite will be the ones used, because the burst of up to 256 data transfer for AXI4 is enough for the I/O needs of the application and since AXI-Lite, is designed primarily with control signals and scalar inputs in mind, and thus allows only 1 data transfer per transaction, which is often enough. [21].

The typical AXI interface consists of a single AXI master and a single AXI slave, representing IP cores that exchange information with each other. Memory mapped AXI masters and slaves can be connected together using a structure called an Interconnect block. The AXI Interconnect IP is sufficient as it already contains AXI-compliant master and slave interfaces with the option to add more per IP block, and can be used to route transactions between one or more AXI masters and slaves[21]. This set-up is commonly called AXI-master-slave protocol or AXI-master, and is the type of interface protocol that will be used in this work.

Both AXI4 and AXI4-Lite interfaces consist of five different channels: Read Address Channel, Write Address Channel, Read Data Channel, Write Data Channel, Write Response Channel Data can move in both directions between the master and slave simultaneously, and data transfer sizes can vary. AXI4 provides separate data and address connections for reads and writes, which allows simultaneous, bidirectional data transfer. AXI4 requires a single address and then bursts, as already mentioned, up to 256 words of data. AXI-master is a memory mapped protocol, which means all transactions involve the concept of a target address within a system memory space and data to be transferred. Examples of read and write transactions are show on figures 5.7 and 5.8 respectively[21].

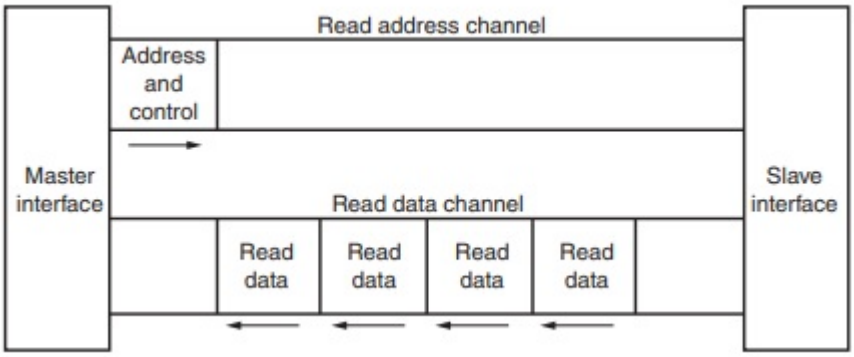

Figure 5.8: Channel Architecture for Data Reads(image source:[21])

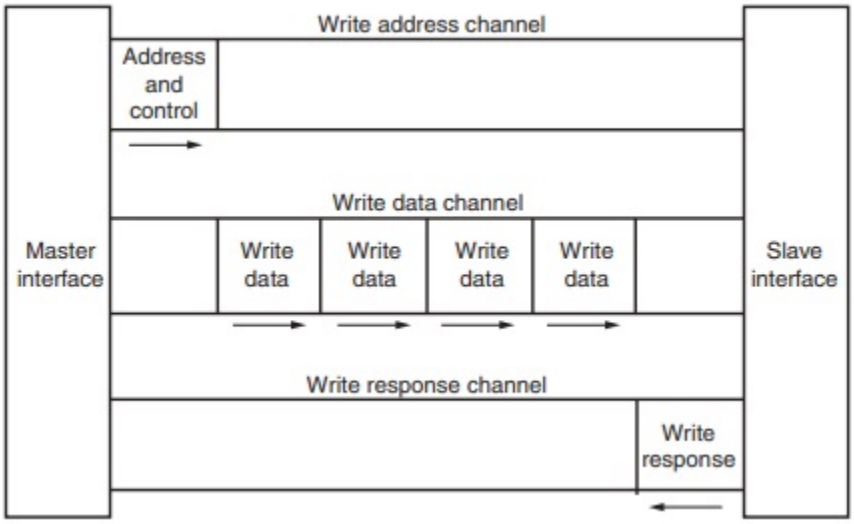

Figure 5.9: Channel Architecture of Data Writes(image source:[21])

To clarify further, despite the fact that AXI-master supports up to 256 words of data burst, the high performance AXI slave interface (HP), which is the best available interface on the Zynq-7000 SoC, used in this work has a bus length of 64 bits, as such only 64 bits are transferred every cycle, however the fact that only one address is needed for 256 words is still a considerable advantage. The AXI protocol does not specify or enforce the interpretation of data, therefore, the data contents must be interpreted by the destination module.

The AXI Interconnect core IP, provided by Xilinx with Vivado, connects one or more AXI memory mapped master devices to one or more memory-mapped slave devices. The AXI interfaces conform to the AMBA AXI4 specification from ARM, including the AXI4-Lite control register interface subset. The AXI Interconnect core consists of the SI, the MI, and the functional units that comprise the AXI channel pathways between them. The SI accepts Write and read transaction requests from connected master devices. The MI issues transactions to slave devices. At the center is the crossbar that routes traffic on all the AXI channels between

the various devices connected to the SI and MI. The AXI Interconnect core also includes other functional units located between the crossbar and each of the interfaces that perform various conversion and storage functions. The crossbar effectively splits the AXI Interconnect core down the middle between the SI-related functional units and the MI-related units. These units are depicted in figure 5.9[21].

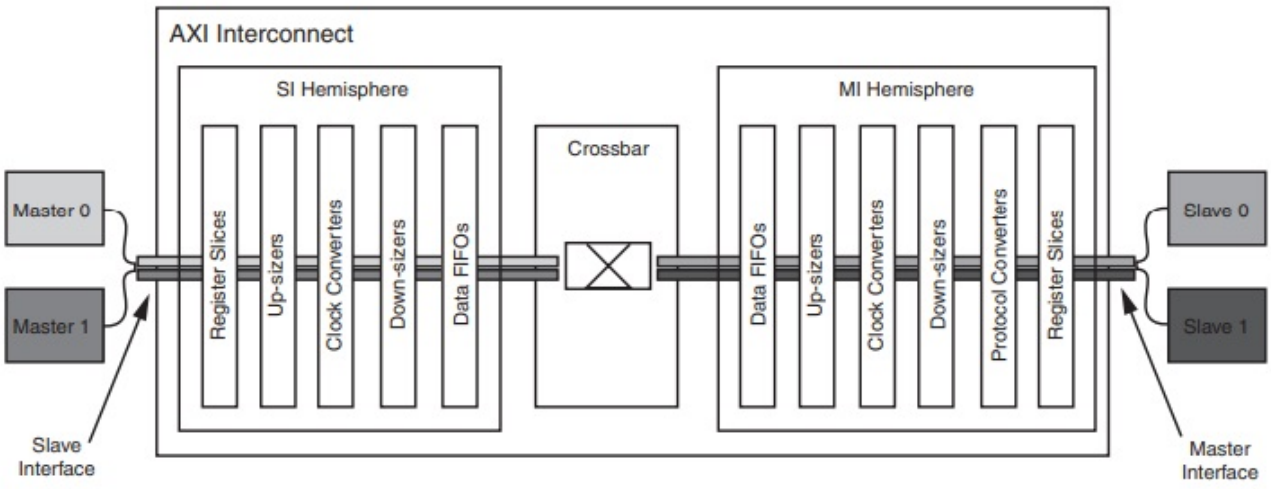

Figure 5.10: AXI Interconnect Top-Level(image source:[21])

There are multiple configurations provided by Vivado, the ones that are relevant for this work are: the pass-through configuration and the N to one configuration. Pass-through configuration is utilized when there is only one master device and only one slave device connected to the AXI Interconnect module, and the module is not performing any optional conversion functions or pipelining, all pathways between the slave and master interfaces degenerate into direct wire connections with no latency and consuming no logic resources. However, the interconnect module is still required as it continues to resynchronize the INTERCONNECT_ARESETN input to each of the slave and master interface clock domains for any master or slave devices that connect to the ARESET_OUT_N outputs, which consumes a small number of flip-flops. N to one configurations are used when there are more than one master interface that need to be connected to exactly one slave interface. In that case, address decoding logic might be unnecessary and omitted from the AXI Interconnect module, unless address range validation is needed. Conversion functions, such as data width and clock rate conversion, can also be performed in this configuration[21].

5.2.2 Integrating AXI to the design

After the small algorithm described previously is implemented in Vivado HLS, the interface ports need to be added, and this is done by directives, such as pragmas. There only two data ports in the top function and are declared as pointers *a* and *b*, which point to the areas of the memory which the data is read from or written at. In order to make these ports into AXI4

interfaces and be able to read and write from them, in that effect the pragmas are written as follows:

```
#pragma HLS INTERFACE m_axi port=a offset=slave bundle=indata
#pragma HLS INTERFACE m_axi port=b offset=slave bundle=outdata
#pragma HLS INTERFACE s_axilite port=return bundle=S_AXI_CONTROL
```

Note that there is an AXI-Lite slave interface which designates as its port, the function's return command. This is done in order merge the control signals in one bundle, that can be guided by the SoC, additionally it generates an interrupt port, however interrupts are not utilized neither on this tutorial nor on the two main designs.

Figure 5.10 shows the tutorial's design. The module created in Vivado HLS is example_axi_m_3, whose two AXI interfaces need to be connected to the Zynq processing system through its slave AXI HP interface. To achieve this a two to one AXI Interconnect is used. As already explained there are control signals that the processing unit must deliver to the main module's slave AXI-Lite interface, and as such the processing system's master AXI general purpose(GP) interface connects to a pass-through AXI interconnect in order to connect with the module's slave interface. The remaining two IPs shown in the figure arer not of great importance to the overall function, as processing system reset is responsible for handling the design's reset function and system ILA is a debug core, whose purpose is to enable Vivado's hardware manager to monitor the signals it is attached to through the hardware manager.

Despite the fact that the module designed in Vivado HLS is very basic, the rest of the design serves as a framework for the development process of the two main modules discussed in this chapter, and was instrumental in the learning process required to understand the inner workings of the AXI4 protocol and its master-slave model, as well as the overall understanding of the tools mentioned here, their supposed workflow and overall function.

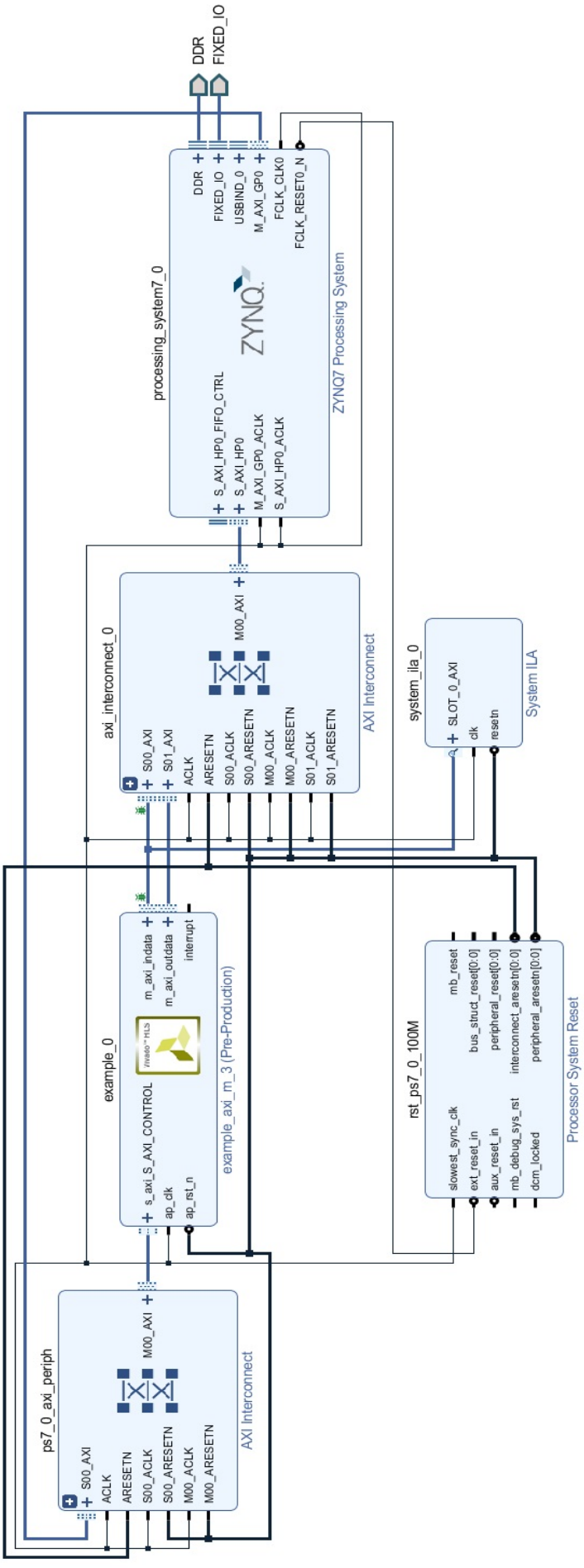

Figure 5.11: The tutorial's Block Design in Vivado

5.3 Transitioning the mathematical models to Vivado HLS

There are two algorithms that will be making the transition from Matlab to hardware, the first one is the data-point recovery algorithm and the second one is the limited precision fixed point algorithm. The mathematical models built in Matlab for these two algorithms during the model phase of this work, were designed with hardware in mind already, in fact the limited precision fixed point algorithm was developed in parallel with its HLS module in order to ensure relatively low resource utilization, without sacrificing too much in the way of high quality results.

Generally both algorithms, follow similar design practices when it comes to their Vivado HLS adaptations. There is a single main loop which is responsible for retrieving the data from the DDR memory using AXI-master bursts, calculating all values dependent on the input data and transferring the results back to the memory with another AXI burst. The "#pragma HLS pipeline" directive is used in order to create a pipelined design, the iteration interval achieved is II=4 cycles, as four inputs are read and four outputs are written per loop iteration.

However, not every calculation needs to be incorporated into the pipeline. Some calculations are independent of the input signal and need to be calculated once. This is true for values dependent on frequency or period as a result they can be calculated using a simple combinational circuit and incorporating them into the pipeline would be a waste of resources. On the other hand, that combinational circuit needs to complete its calculations in a time-frame of a single cycle to avoid timing issues such as negative slack. To that end, a more specific design was chosen for the upsampling version of the algorithm, setting its output sampling rate at 176.4 kHz and simplification to the mathematics of the formula were done whenever the designed allowed. The sampling rate of 176.4 kHz was chosen as during modelling and simulation it appeared to provide the best quality as well as being the most computation intensive input for the model, further justifying its hardware implementation. The design of the data-point recovery algorithm was already specific about its input and output sampling rate and no simplifications needed to be made compared to Matlab's version.

The most expensive calculations, both resource wise and time wise, are by far divisions, followed by multiplications. Whenever possible simple multiplications with integers are replaced with bitwise shifts, although they are not common in the algorithm in the first place. For example simple multiplications by two are replaced with a single shift left and multiplications by six can be replaced with shifting the number by one to the left, on a different register shifting it left by two and adding the result. As for divisions, only a few division by two are implicated, and replaced by one shift to the right.

With these coding practices implemented the designs can now be synthesized and exported to Vivado.

5.4 Finalizing the Hardware Designs on Vivado

Both designs have an interface similar to the one used for the tutorial with the extra addition of the original sampling rate as a scalar input, which is handled by the AXI-Lite slave interface which is also responsible for the control signals associated with the AXI-master protocol. As a result, the same framework built around the tutorial design seen, previously on figure 5.10, can be utilized with no additional changes, the only difference per design being the central module generated by Vivado HLS.

Name	Slice LUTs	Slice Register	F7 MUXes	Slice	LUT as Logic	LUT as Mem	Block RAM	DSPs	Bonded IOPADs	BUFGCTRL
Available	53200	106400	26600	13300	53200	17400	140	220	130	32
Upsampling	4994	6311	32	2071	4758	236	1	115	130	1
Point Recovery	4475	4297	2	1722	4271	204	1	79	130	1

Table 5.1: Utilization Report of Hardware Designs(Post Implementation)

The final utilization report can be seen on table 5.1. During the design process, after the implementation of each design was complete, if the resource or timing constraints were not met then the HLS module would need further optimization, as suggested by figure 5.1. This is how these two final designs would be created. More specifically the first implementation was a full precision model presented in chapter 4. That model was utilizing more resources than were available on a Zedboard, consequently the limited precision model was co-developed in Matlab and Vivado HLS. Parallel development was necessary to ensure that the reduction of resources resulting from the reduction of arithmetic precision, would not be to the detriment of quality results.

5.5 Creating the Host Program on Vitis

Having finalized both hardware designs it is time to create a host program for each hardware design. Vitis creates hardware platforms out of the .XSA files, exported by Vivado, host programs can be written for the Cortex-A9 on-board the Zedboard.

Before developing the host program, there is one more important design choice to be made, what operating system (OS) will be used. Among the plethora of OS choices that Vtiis supports two stand out, a Linux OS and a standalone-baremetal application. A Linux OS would allow for better file management and manipulation, by offer Linux libraries for the host program, on the other hand it requires some additional development time due to the fact

that boot components would have to be designed using PetaLinux, as well as additional hardware drivers for the hardware platforms. Standalone offers automatically generated boot components and hardware drivers, but file management is restricted to FAT file systems(FFS), done with the use of the xilffs library provided by Xilinx. Since the advantages provided by a Linux OS, are not essential while FFS is serviceable for the purposes of both designs. As such a standalone-baremetal approach is chosen.

Both designs follow a similar approach. The host program reads a WAVE file from an SD card formatted in FAT32 file system, the channels are separated from each other and stored in matrices A and A2, which are dynamically allocated using the information about the number of samples found in a WAVE file's header. Since the sampling rate targeted by the designs is known, and preset at 176.4 kHz for the upsampling version and at 44.1 kHz for the data-point recovery version, the result buffers can be dynamically allocated. Next, the inputs can be set on the correct variable specified by the automatically generated driver and the execution signal can be sent to the FPGA. After the FPGA has completed its processing the results are written in a WAVE file with the appropriate modifications to the each header struct. More specifically the header file required for each design's result is different, while both use PCM, data-point recovery has a bit-width of 16 bits and a sampling rate of 44.1 kHz, just like the supposed input files. The upsampling version has 32 bits of depth and a sampling rate 176.4 kHz, which affects most values of the header file.

On a slightly different note, there is a lot of misinformation about WAVE files, probably as a result of their under-utilization. For example a lot of sources such as forums and sites as stackoverflow have discussion threads where it is stated that WAVE files of more than 16 bits of data per sample need to use the WAVEFORMATEXTENSIBLE header. This is not true as the normal header is perfectly serviceable and changing values to over 16-bits is recognized by media players and correctly recognise the encoding. Additionally in questions regarding 24 bit WAVE files no one ever references that a 24 bit file must be written as 32 bit integer just with zeroes in the place of the integer's most significant bits. Wave all in all is built with standard programming languages in mind and does not need custom data types to be written properly.

5.6 Verification and Results

The design's results were compared to the results of Vivado HLS' C simulation results and the mathematical model's results. The simulated results and the hardware's results were identical due to the obvious fact that Vivado HLS' C debugger and simulator tries to simulate the design as closely as is possible. Matlab's results were identical in most cases, with the results produced by the other two methods, with the only divergence happening in one or two least significant bits. This divergence can be attributed fact that, in order for the bit length

of the variables to be accurately portrayed some bits needed to be removed manually, by reapplying the fi() function, as described in chapter 4.

The fact that the results, produced by all three methods, are matching should not be a surprise considering the fact that the Vivado HLS module and the Matlab model, where designed simultaneously, in order to achieve this level of validity for their results.

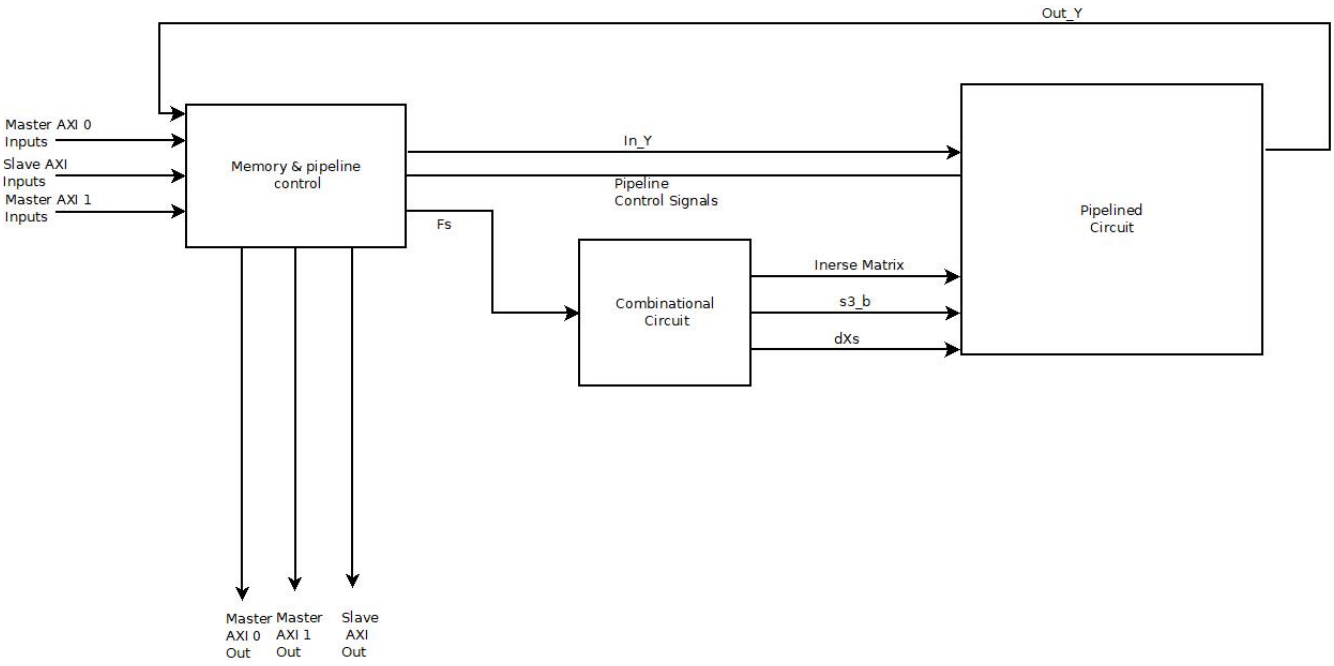

Figure 5.12: The design produced by Vivado HLS

The design generated, by Vivado HLS was also validated and examined in regard to efficiency. The designs presented in figures 5.12, 5.13, and 5.14 are derived from the RTL design given by Vivado and simplified, in order to be easier to understand the way the design operates.

On figure 5.12, a high level block diagram of the module is depicted, the memory and pipeline control module is responsible for I/O operations as well as the control for the pipeline. The combinational part, referenced earlier and depicted in figure 5.14, is receiving the scalar input Fs which is the sampling rate, and calculates all the static values of the algorithm.

Figure 5.13 depicts the pipelined portion of the algorithm, whose results depend on the data-points read from the DDR-RAM. The module rhs, represents the value of R as referenced in theory, s1, s2 and s3 represent the different coefficients needed for the final calculations.

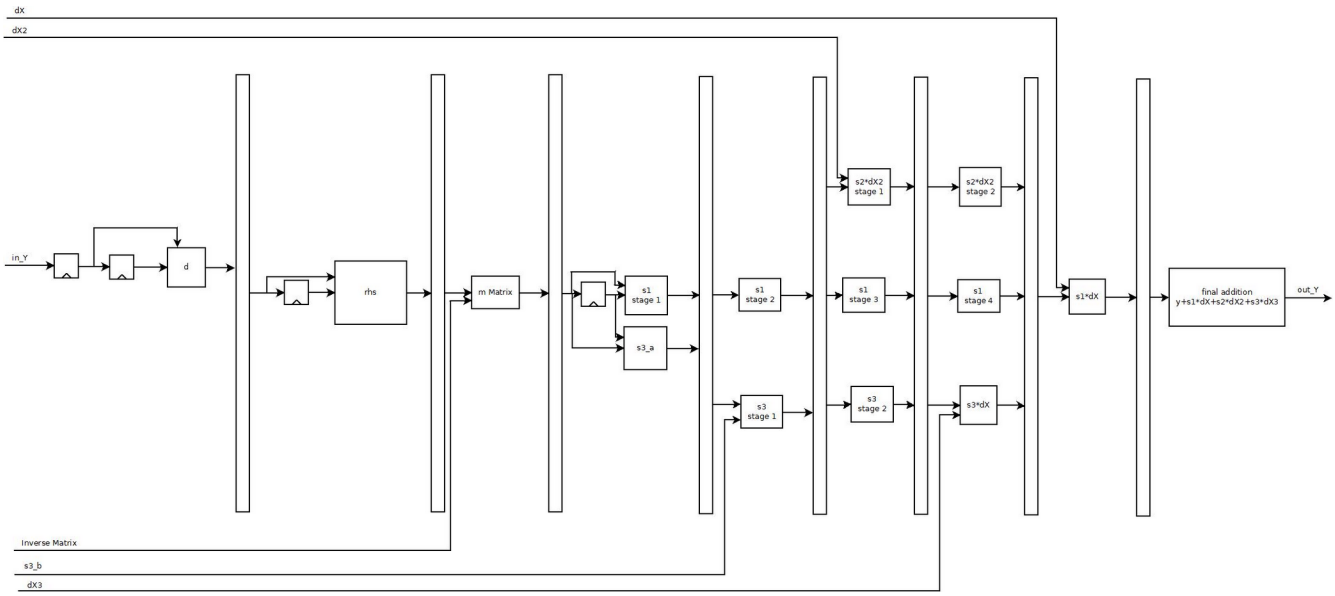

Figure 5.13: The pipelined portion of the design

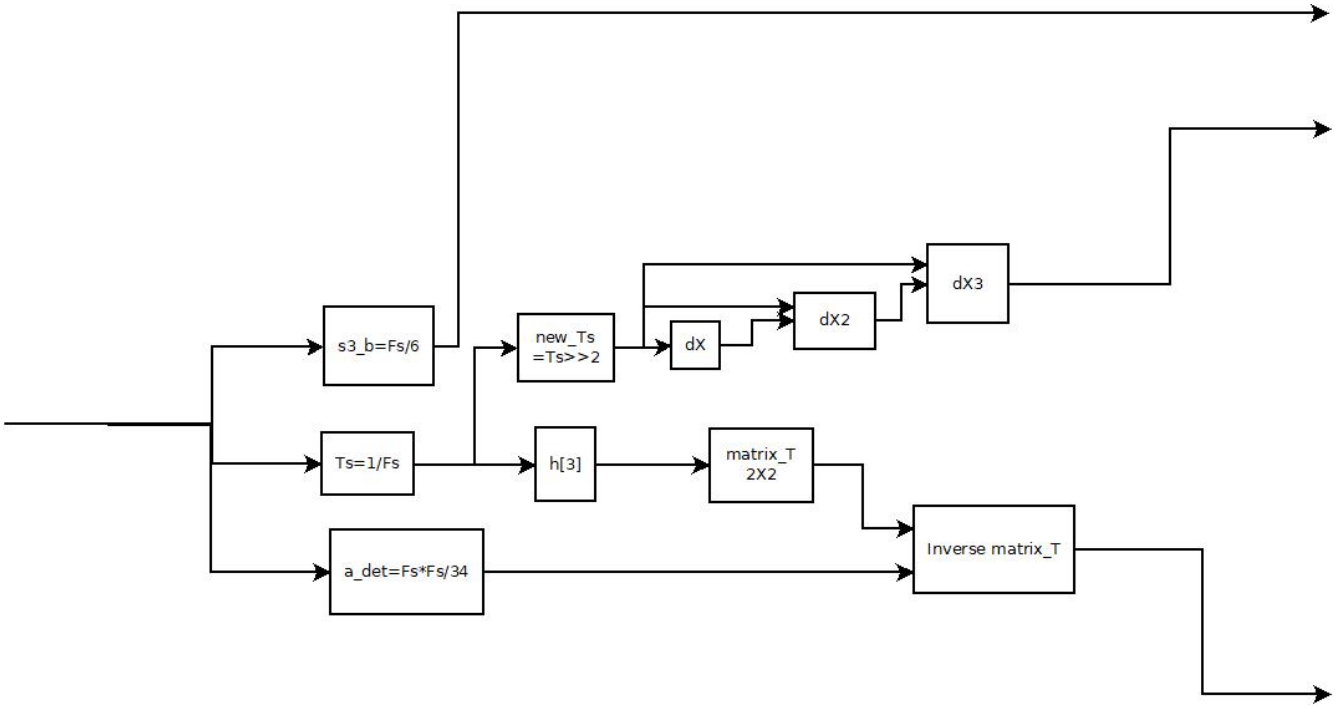

Figure 5.14: The combinational portion of the design

5.7 Hardware and Software timing comparison

In order to measure the performance of the designs, two algorithms designed in c, and meant to run on a common processor were designed, utilizing floating point arithmetic. The libraries time.h and xtime_l.h were used, on the C program and the Vitis host program respectively, in order to measure execution times. The file chosen as input had a size of 30.186 Megabytes. The execution time on hardware for the upsampling algorithm was 1.5 seconds and 0.5 seconds for the data-point recovery algorithm, in contrast their software counterparts had average execution times of 0.7 and 0.14 seconds, when ran on an Intel i3-2370 processor operating at 2.4 GHz. For reference the FPGA's clock is set at 10 ns or 100 MHz.

Judging by the results it would appear that that there is not much of an incentive to design a hardware implementation for the cubic spline algorithm since it appears that common PC processors can execute the algorithm faster. Even if the II of the hardware design was dropped to two, down from four, the results would have a similar performance but the advantage would still lie with the software versions. The addition of a DMA module and the implementation of an AXI-stream interface instead of an AXI-master interface would also help improve the design's timing but not to a significant enough degree.

However, when factoring in the cost of a processor, the FPGA solution becomes more appealing, since the module developed in this thesis could be used for an FPGA design with significantly lower cost, especially when energy consumption costs are factored in as well.

These results are not surprising, due to the fact that multiplications and especially divisions are expensive operations and cubic spline requires a lot of them to perform its calculations. It should also be noted that a lot of effort has gone into reducing the impact of the aforementioned operations and their impact is minimized as much as possible. On the other hand, it would appear that cost-wise the hardware implementation has the upper hand, significantly reducing cost with only a minimal trade-off performance wise.

Chapter 6:
Summary, Conclusions and Future Work

6.1 Summary

In the present thesis, the idea of a more hardware efficient cubic spline interpolation model, using fixed-point arithmetic, was explored and developed, based on previous work done by Triantafillos Mourtzanos[1]. As a result the first step in the creation of this endeavour was studying his thesis and most of his cited sources, in an effort to build up understanding for the mathematics used and a general knowledge of audio-engineering. More specifically, the first step was understanding the basic principles of digital audio processing, signal theory and their mathematical concepts, as well as psychoacoustics and human sound cognition, followed by the mathematics of interpolation methods and the approach chosen in the Mourtzanos' thesis. Other applications of interpolation methods were also researched along with other methods of data-recovery by other researchers.

When it came to development, the first step was recreating the models of the previous work, using Matlab, and especially the model for cubic spline interpolation, to that effect three different models were developed with only the third presenting results in line with what was expected. The criteria were mainly two: the recovery of some lost information from damaged audio signals and the capacity of the algorithm to increase the quality of an existing audio signal by increasing the sampling rate and the bit-depth form 16 bits to 32 bits. The models presented had the capacity to produce a signal with any sampling rate above 44.1 kHz, which is the standard for CD quality, however the sampling rates that were primarily tested where: 48 kHz, 96 kHz and 176.4 kHz. After ensuring the competence of the algorithm it was converted to a full-precision fixed point algorithm which received four data-points of input per iteration, which is the minimum amount of data-points required for cubic spline interpolation.

The full precision version of the algorithm had an unreasonable demand on hardware resources. As a result, a more limited precision model was developed by systematically reducing arithmetic precision without impacting the results in any significant degree. This was achieved by developing the hardware implementation in parallel with the Matlab model. Spectrograms, waveforms and audio resulting from each Matlab model were tested and compared, resulting in the limited precision algorithm proving its competence by providing results in line with the other more resource demanding algorithms.

Finally two hardware designs were implemented one aiming at recovering lost data-points and the other increasing the sampling-rate and bit-depth. A lot of familiarization was needed with the design tools, namely: Vivado HLS, Vivado, and Vitis. The AXI protocol was studied extensively and was used to connect the board's SoC to the main module. A software version

of the two algorithms was designed and proved superior to the hardware designs, albeit not by a large degree.

6.2 Conclusions and Commentary

In this thesis the cubic spline interpolation algorithm was greatly improved when it comes to hardware integration and was tested as both an upsampling, bit-depth increasing method and as data recovery method with regard to audio signals. Its results were in line with what was described in Mourtzanos' thesis[1], all the while drastically improving resource consumption and timing using a relatively common and cheap FPGA(Zedboard), bringing it on relative parity with software implementations. Although it should be noted that the performance of C versions of the algorithms, which were developed in an afternoon's time, was still better, making processors the better platform for cubic spline interpolation.

The huge discrepancy in timing between these two designs and the original floating point design, reinforces the position that significant improvements have to be implemented in hardware implementations of floating point arithmetic, for them to compete with fixed-point designs, which generally appear to be more resource efficient, for less than 64 bits of precision, and often enough faster than floating point designs.

A minor critique of Mourtzanos' thesis, is necessary here. The thesis promises and proclaims encouraging results for the cubic spline interpolation algorithm, when it comes to its application in audio engineering. It is this student's opinion that despite the fact that improvements were seen in signals during tests, the point of reference was another interpolation method, namely linear interpolation. Linear interpolation is an overall cheaper method than cubic spline interpolation, both resource wise and time wise, and thus the comparison is unfair. Linear interpolation can be easily developed on an even cheaper FPGA, or for that mater a relatively old microcontroller. Furthermore, there are other mathematical models that provide better results than cubic spline interpolation, such as the tried and true method of sinusoidal modelling, to separate the audio signal in basic sinusoids and use different interpolation methods for a sinusoid's different attributes, managing to recover data of several micro-seconds more reliably than cubic spline interpolation. Overall, the results were mediocre compared to more complex techniques already developed.

In hindsight, the mediocrity of the results, when it came to data-point recovery, was to be expected. Interpolation and extrapolation methods were primarily designed and developed for curve fitting and prediction of measurements and statistics. As such, cubic spline interpolation was developed to encompass a large variety of different data, resulting in a jack of all trades, master of none, model. This is incredibly evident in the bibliography, by the fact that there were no publications about the use of a standalone cubic spline interpolation model on audio engineering, it had to always be supplemented by other models, to bring it more in

line with the challenges of audio engineering. The only instances where cubic spline interpolation was used as a standalone model were related to scientific measurements and not audio.

All in all cubic spline interpolation performed its best when used to increase sampling rates and bit-depth, but this was easily done in C as well, performing slightly better than the hardware design.

As a final remark it should be noted that, Matlab while serviceable and accurate, was extremely slow when it came to fixed-point arithmetic calculations. This was because Matlab is optimized for floating point arithmetic and fixed point was an afterthought. Matlab simulates fixed point numbers and operations by a series of function calls, slowing down its calculations by a considerable degree. In hindsight, a better way to compare the models presented in this work would have been to have Vivado HLS, handle the fixed point calculations through its C\C++ simulator, have them exported to a file and imported to Matlab in order to compare waveforms and spectrograms.

6.3 Future Work

As already mentioned the most probable application for a cubic spline interpolation design, is upsampling and increasing bit-depth simultaneously, as such there might be merit in the idea of integrating the design as a cloud service. Additionally the design could be moved to more modern FPGA and have it process both channels of an audio file simultaneously, or used on a smaller FPGA without an SoC, dramatically reducing costs at the cost of having to implement an Ethernet connection and slight changes to the module's interface.

The most exciting idea that came to mind during the final stages of writing this thesis was to have a distributed system of microcontrollers that would utilize sinusoidal modelling and various interpolation methods as described in chapter 2. The objective of the system would be to bridge gaps in audio files. Sinusoidal attributes would have to be calculated separately, amplitude would have to be interpolated linearly, while cubic interpolation is used for the phase, the frequency can be found by the differentiation of the cubic phase polynomial.

Alternatively the same objective can be achieved with another method referenced in chapter 2, namely: interpolation of missing data values using a Garbor regression model[10]. However this model might be better suited for software platforms rather than hardware.

Bibliography

Books and Papers

[1] Mourtzanos Triantafillos, Embedded Processing System for Digital Sound, Techical University of Crete, 2016

[2] Lynn Blair. Data Interpolation and Its Effects on Digital Sound Quality, McMurry University, 2008.

[3] Alan V. Oppenheim, Alan S. Willsky, S. Hamid Nawab, Signals and Systems, Second Edition

[4] Simon S. Haykin, Digital Communication Systems

[5] H. Nyquist, Certain Topics in Telegraph Transmission Theory, 1928

[6] G.K. Karagiannidis and K. N. Pappi, Telecommunication Systems, Third Edition

[7] John Watkinson , The Art of Digital Audio, Third Edition

[8] Perry R. Cook, Music, Cognition, and Computerized Sound. An introduction to Psychoacoustics

[9] E.Brad, Meyer,David, R.Moran, Audibility of a CD-Standard A/DA/A Loop Inserted into High-Resolution Audio Playback. Boston Audio Society, Lincoln, MA, USA, 2007

[10] P. J. Wolfe and S. J. Godsill, "Interpolation of missing data values for audio signal restoration using a Gabor regression model," Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005., Philadelphia, PA, USA, 2005, pp. v/517-v/520 Vol. 5, doi: 10.1109/ICASSP.2005.1416354

[11] Doerfler, Monika. (2001). Time-Frequency Analysis for Music Signals: A Mathematical Approach. Journal of New Music Research. 30. 3-12. 10.1076/jnmr.30.1.3.7124.

[12] A. Lukin, and J. Todd, "Parametric Interpolation of Gaps in Audio Signals," Paper 7512, (2008 October.).

[13] George E.Forsythe, Michael A. Malcomlm, Cleve B. Moler, Computer Methods for Mathematical Computations, Prentice-Hall series in Automatic Computation

[14] William H. Press, Brian R. Flannery, Saul A. Teukolsky, William T. Vetterling, Numerical Recipes in C, the art of Scientific Computing, Cambridge University Press, 1988

[15] Ian McLoughlin, Applied Speech and Audio Processing: With Matlab Examples, Cambridge University Press, 2009

Manuals

[17] Xilinx,Zynq-7000 SoC: Embedded Design TutorialA Hands-On Guide to Effective Embedded System Design, ug1165

[18] Xilinx, Introduction to FPGA Design with Vivado High-Level Synthesis, ug998

[19] Xilinx, Vivado Design Suite User Guide: High Level Synthesis, ug902

[20] Xilinx, Vitis Unified Software Platform Documentation, Embedded Software Development, ug1400 (v2019.2)

[21] Xilinx, AXI reference guide, ug761

Links

[22] https://charmain2010.wordpress.com/2014/05/01/analogue-vs-digitaladvantages-vs-disadvantages/

[23] **http://musicweb.ucsd.edu/** (**http://musicweb.ucsd.edu/~trsmyth/digitalAudio171/Analog_Digital_Conversion.html**)

[24] https://www.mediacollege.com/glossary/q/quantization.html

[25] **https://en.wikipedia.org/wiki/Psychoacoustics**

[26] **https://en.wikipedia.org/wiki/ABX_test**

[27] **https://en.wikipedia.org/wiki/Linear_interpolation**

[28] **https://en.wikipedia.org/wiki/Interpolation**

[29] **https://en.wikipedia.org/wiki/Modeling_and_simulation**

[30] **https://en.wikipedia.org/wiki/Sample-rate_conversion**

[31] **https://www.mathworks.com/help/signal/ref/resample.html#mw_7adbf990-9b5e-4677-ac50-8997f886114c**

[32] **https://www.mathworks.com/help/signal/ref/spectrogram.html**

[33] https://en.wikipedia.org/wiki/Pulse-code_modulation

[34] **https://en.wikipedia.org/wiki/MP3**

[35] **http://pilot.cnxproject.org/content/collection/col10064/latest/module/m34847/latest**

[36] **https://www.researchgate.net/figure/Diagram-showing-the-structure-of-the-human-ear-detailing-the-parts-of-the-outer_fig1_324547019**

[37] **https://chromatone.center/apps/sound-and-tone-perception/**

[38] **https://en.wikipedia.org/wiki/Interpolation**

[39] **https://en.wikipedia.org/wiki/Spline_interpolation**

[40] **https://reference.digilentinc.com/learn/programmable-logic/tutorials/zedboard-programming-guide/start**