FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

MASTER'S THESIS

THE EFFECT OF COEFFICIENT QUANTIZATION OPTIMIZATION ON FILTERING PERFORMANCE AND GATE COUNT

Author	Adewale Ayomikun Elijah
Supervisor	Zaheer Khan
Second examiner	Tuomo Hänninen
Technical supervisor	Seppo Liuski

FEBRUARY 2023

Ayomikun A. (2023) The effect of coefficient quantization optimization on filtering performance and gate count. Faculty of Information Technology and Electrical Engineering, Degree Programme in Electronics and Communications Engineering, 52 pages.

ABSTRACT

Digital filters are an essential component of Digital Signal Processing (DSP) applications and play a crucial role in removing unwanted signal components from a desired signal. However, digital filters are known to be resource-intensive and consume a large amount of power, making it important to optimize their design in order to minimize hardware requirements such as multipliers, adders, and registers. This trade-off between filter performance and hardware consumption can be influenced by the quantization of filter coefficients. Therefore, this thesis investigates the quantization of Finite Impulse Response (FIR) filter coefficients and analyzes its impact on filter performance and hardware resource consumption. A method called dynamic quantization is introduced and an algorithm for step-by-step dynamic quantization is provided to improve upon the results obtained with the classical fixed point quantization method. To demonstrate the effectiveness of this approach, the dynamic quantization of filter coefficients for a Low-pass Equiripple FIR filter is examined and a comparative study of the magnitude response and hardware consumption of the generated filter using both the classical and dynamic quantization methods is presented. By understanding the trade-offs and benefits of each quantization method, engineers can make informed decisions about the most appropriate approach for their specific application.

Keywords: Coefficient, Digital filters, Equiripple, FIR filter, Optimization, Quantization

TABLE OF CONTENTS

FOREWORD

This thesis was written to fulfill the requirements of the Master's in Wireless Communication Engineering program at the University of Oulu in Finland. It was completed during my time as a Digital Design Engineer at Uninord Oy and it examines the impact of quantizing filter coefficients on filter performance and hardware resources.

I am delighted to present this thesis, which represents the culmination of my academic journey. It is with great pleasure that I express my gratitude to my supervisor, Dr. Zaheer Khan, for his unwavering support and guidance throughout this research. His expertise, patience, and encouragement were invaluable in shaping my ideas and bringing my work to fruition.

I would also like to thank Seppo Liuski, my technical supervisor at Uninord Oy, for his invaluable input on the technical aspects of my work. His expertise and experience were critical in helping me to overcome the various challenges that I encountered along the way.

I cannot express enough gratitude to my parents, Professor and Mrs. Adewale, for their unending love, support, and encouragement throughout my academic journey. Their unwavering belief in my abilities has been a constant source of inspiration and motivation.

To all those who have supported me in this journey, I extend my deepest appreciation and thanks

Oulu, Decemeber 21st, 2022

Adewale Ayomikun

LIST OF SYMBOLS AND ABBREVIATIONS

1-D	1 Dimensional
2-D	2 Dimensional
ASIC	Application-Specific Integrated Circuit
CIC	Cascaded Integrator Comb
CSD	Canonical Signed Digit
CSE	Common Sub-Expressions
DSP	Digital Signal Processing
EVT	Extreme Value Theory
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPU	Graphics Processing Unit
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
IIR	Infinite Impulse Response
LPF	Low-Pass Filter
MAC	Multiply-Accumulate
MFIR	Multiplicative Finite Impulse Response
OQNP	Output Quantization Noise Power
RAM	Random Access Memory
RTL	Register Transfer Level
RTU	Remote Terminal Units
SPT	Signed integer Power of Two-term
VLSI	Very Large-Scale Integration

Hz	Hertz
kHz	Kilo-Hertz

a_m	filter denominator coefficient
$A(z)$	filter denominator
$A^{'}(z)$	quantized filter denominator
b_n	filter numerator coefficient
$b(n-k)$	filter coefficient
$B(z)$	filter numerator
$B^{'}(z)$	quantized filter numerator
$H(z)$	filter transfer function
$H^{'}(z)$	quantized filter
k	sample index
S	scaling factor
$u_n^{'}$	dynamic quantized filter coefficient
$x(k)$	filter input
$y(k)$	filter output
z	z-transform

Δa_m	quantization error of filter denominator coeffient a_m

$\Delta A(z)$	quantization error for filter denominator
Δb_n	quantization error of filter numerator coeffient b_n
$\Delta B(z)$	quantization error for filter numerator
$\Delta H(z)$	error filter
ϵ	exponential
$\approx$	approximate
$\sum()$	sum
w	Frequency

1 INTRODUCTION

This dissertation studies the quantization of digital filter coefficients. In this work, a dynamic quantization of the floating point coefficient will be introduced and a comparative study will be performed on how this choice of quantization affects the performance and behavior of a digital filter in comparison to using the classical fixed point quantization method. We will also examine the trade-offs between the filter performance and required hardware resources concerning the implementation of the Digital Signal Processing (DSP) components in Application-Specific Integrated Circuits (ASIC). Next, we provide a brief overview of Finite Impulse Response (FIR) filters, coefficient quantization, and also a summary of what will be provided in later chapters.

1.1 FIR Filter

According to [1], a digital filter is a tool that assists in eliminating unwanted signal components from the desired signal. The two primary categories of digital filters are the Finite Impulse Response (FIR) and the Infinite Impulse Response (IIR). For applications that require linear phase response, FIR filters are preferred over IIR filters due to their stability, linear phase property, and absence of limit cycles.

FIR filters are also called non-recursive. These filters are key computing units in wireless systems and signal processing systems. In FIR filters, filter coefficients represent the values of the impulse response. Constant coefficients are widely utilized in practical applications. The constant coefficients present in these filters provide flexibility in designing and optimizing the filter based on the properties of the coefficients. As a result, such filters are commonly employed in applications where low complexity and power consumption are crucial considerations [2].

Although digital filters are known to be expensive in terms of hardware and power consumption, several research studies [1], [3], [4], [5] have been carried out to optimize the design of FIR filters and reduce the required hardware resources, such as multipliers, adders, and registers, needed for their implementation. Optimizing the number of multiplications results in improvements in design parameters such as power and area consumption [6]. This section provides a brief overview of FIR filters and discusses their structures and functions.

1.1.1 Filter Structure

There are many structures for FIR filters, such as Direct Form I/II, transposed Direct Form I/II, Cascade Form, and Parallel Form. The numerator and denominator coefficients can be generated using many algorithms. However, for this work, the Equiripple FIR Filter is used because it gives us the minimum order to achieve the design specification, and the filter structure is a Direct Form.

The FIR filter structure is referred to as a tapped delay line that is weighted. This is visually depicted in Figure 1.1

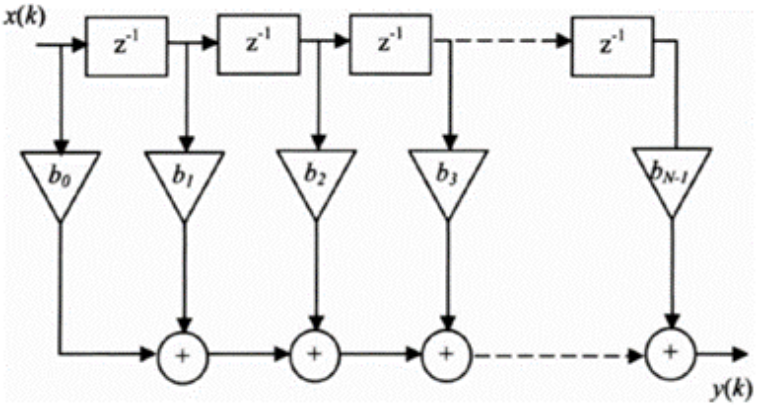

Figure 1.1. FIR filter structure (reproduced from ([7]).

To obtain the filtered signal of an FIR filter, a Fourier transform can be utilized as it has a finite impulse response. Equation 1.1 in the z-domain represents the transfer function of the filter, and Equation 1.2 represents the relationship between the input and output of the Linear Time-Invariant (LTI) FIR system with N taps [8].

$$H(z) = \sum_{n=0}^{N-1} b(n) z^{-n} \tag{1.1}$$

$$y(k) = \sum_{n=0}^{N-1} x(k) b(n-k) \tag{1.2}$$

The output of the filter is represented by $y(k)$, while $x(k)$ represents the filter's input, and $b(n-k)$ denotes the filter coefficient. From the equation, it is clear that the main operation of the filter involves multiplying the filter coefficients with the input data and accumulating the results. Furthermore, the filter's response is mostly dependent on the current and previous input samples. The filter coefficient primarily determines the filter's response [9].

Furthermore, as the number of filter coefficients increases, more computation would be required, which in turn leads to more hardware resources being used and better accuracy which is relative to the filtering results. Hence, making the number of coefficients of the filter greatly impacts the whole design. The reason behind this is that complexity of the FIR filter can be assessed by the quantity of logic utilized, the space taken up by the circuit on the chips, or the number of transistors used, all of which are directly correlated with the number of coefficients in the filter. [10].

A different method can be used to obtain this filter coefficient. The windows method is generally considered simple and robust for filter design. However, its drawback is that it does not guarantee optimal order. On the other hand, the least square method generates a filter that over-satisfies the specifications but has a large order. In this work, the Equiripple method will be utilized to design the filter, this provides a minimum filter order that satisfies the specifications [6]. This method involves the Park-McClean algorithm, which is computationally efficient and relies on linear programming.

1.1.2 Coefficients Quantization and Filter Performance

Quantization is a method of approximating data samples to a desired level of precision and representing them using a specified number of bits per sample. It is widely employed in modern electronics, and its study is crucial to software and hardware engineers. Many programmers make decisions on the type of quantization they will use and the available precision, even in the absence of a formal study of quantization.

Computers use two methods to represent numbers, namely floating point or fixed point. Each of these methods utilizes a specific number of bits to store the value, with more precision typically requiring more bits. In the case of floating-point representation, the values are discrete but not equidistant, and values that are close to one another are uniform. However, as the values increase in magnitude, the step size also increases significantly, resulting in a large dynamic range that enables us to store the values in memory using the same number of bits to represent both large and small values. On the other hand, fixed-point representation mostly involves uniformly spaced discrete values, and these values are typically stored using two-complement arithmetic.

Modern computers frequently use floating-point representation, but they come with a significant cost in terms of power consumption, chip space, and design time. To address this issue, designers can use a fixed-point implementation or a floating-point implementation with a smaller exponent size. However, reducing the filter's precision would require quantizing the double-precision coefficient. This quantization could significantly alter the filter's response, so designers must be careful to ensure that the filter still performs according to its design specification

Many researchers have conducted studies to investigate the impact of quantizing filter coefficients on the digital frequency response. [11], [12], [13], [14]. However, few studies have been conducted on how coefficient quantization affects the Equiripple FIR filter. In this dissertation, a dynamic quantization algorithm will be introduced to quantize the filter coefficient. The performance of this filtering algorithm will be measured and analyzed. A comparative analysis will be conducted to evaluate the performance of the dynamic quantization method in comparison to the classical quantization method. The basic idea behind the classical quantization method is that it is typically done by using a quantization step size and this quantization step size determines the resolution of the quantized coefficients. The quantization step size is defined as the minimum difference between two adjacent values that can be expressed using the same number of bits. In the classical quantization method, the filter coefficients are rounded to the nearest quantized level. In addition to the comparative study, we will also assess the hardware resources required for each quantization method and analyze the balance between filter performance and hardware utilization.

Given a filter design specification for an Equiripple FIR filter, the filter coefficients can be produced. These coefficients are said to be optimal in the min-max sense [15]. In other words, the coefficients generated by this filter for a designated frequency band have a lower maximum frequency when compared to other filter algorithms. To assess the performance of the Equiripple filter, the first step is to determine the maximum deviation from the desired frequency response. The resulting maximum deviation is referred to as the maximum ripple.

To understand this, it is necessary to examine the operation of a Low-pass Filter (LPF), which is designed to eliminate all frequencies within its stop band. Put simply, the LPF is expected to filter out all frequencies that exceed the cut-off frequencies. Nonetheless, it is unfeasible for a digital filter with a finite length to remove all frequencies within its stop band. As a result, the maximum magnitude response within the band is inversely proportional to the filter's performance in the stopband. [15]. The ideal response and practical response of this filter are

presented in Figure 1.2 and Figure 1.3.

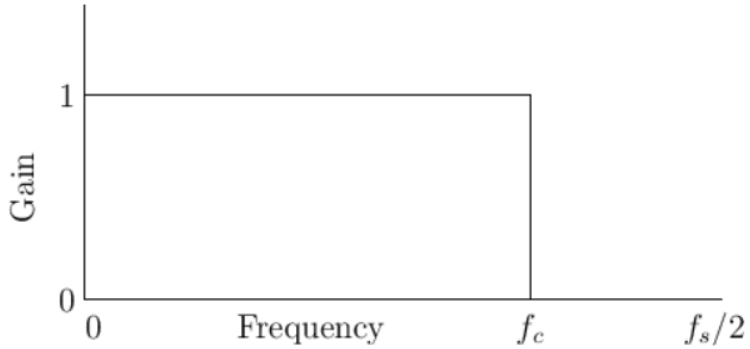

Figure 1.2. Ideal LPF Specification (reproduced from [16]).

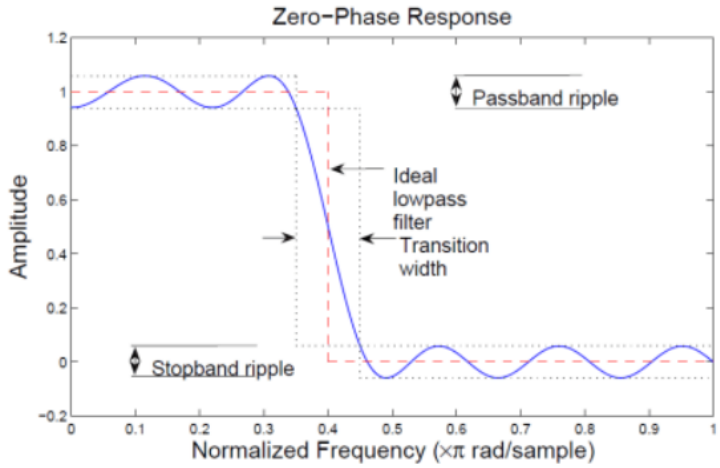

Figure 1.3. Practical LPF Specification (reproduced from [17]).

MATLAB is a software that can be used to produce the filter coefficients, although, these coefficients are represented in double-precision which has a near infinite precision. For hardware implementation purposes practical finite wordlength is required [18]. Hence, requiring the generated floating-point coefficients to be quantized [19]. This quantized coefficient will typically have less precision when compared to the double-precision coefficient. Furthermore, filter coefficient quantization will usually result in the frequency of the filter is significantly changed. In addition, because they are optimal filters, therefore any alterations to the coefficient values will lead to degradation of the filter performance.

To quantize filter coefficients, double-precision floating point values are rounded to the nearest quantized level. Adjusting the quantized coefficients can enhance filter performance. This means coefficients can be shifted down or up by one or two quantized levels, resulting in better performance. Additionally, adjusting a coefficient by increasing or decreasing it by a single quantized level can greatly affect the maximum ripple of the filter. Hardware requirement and filter performance are both influenced by the quantized coefficient. A suitable quantization technique can significantly reduce hardware complexity.

The goal of this dissertation is to quantize and optimize the filter coefficient values and compare the filter performance with a simple rounding method. The final step is to examine how much hardware resources we are saving as compared to the simple rounding method.

1.1.3 Hardware Implementation

MATLAB is used to run the Equiripple FIR filter algorithm and return a set of filter coefficients in double-precision, but as stated earlier the practical length of the coefficient is required in the hardware implementation of this filter [20]. Therefore, the generated coefficients are quantized using the proposed method and converted to fixed point coefficients. The performance of the filter is analyzed and the input, output, and coefficients of the filter are generated using MATLAB. The hardware implementation of the filter is simulated using the Vivado development software. A single-rate filter, which means the filter sampling rate is equal to the sampling frequency of the output, is used in this study. The coefficient file generated by MATLAB for Xilinx will serve as input to the FIR filter design's test bench. The coefficients will be stored in the Random Access Memory (RAM), where they will be multiplied with data samples from a parallel pipeline originating from another block in the DSP unit.

The FIR filter employs a Multiply-Accumulate (MAC) operation, where the coefficient is multiplied by the delayed data sample, and the results are accumulated. In most cases, one MAC is necessary per tap in FIR filters. The MAC structure is made of the multiplier, adder, and accumulator blocks. Every clock cycle the filter performs multiplication of the filter coefficients with the delayed sample and accumulate the result of the multiplication to produce the filter output.

In Hardware Description Language (HDL), the filter can be implemented in two forms, which are parallel and series implementation. To begin with, parallel implementation is often viewed as the most straightforward method, as the memory components can be realized as registers and the arithmetic operation can be performed using DSP slices if they are present. Nevertheless, the disadvantage of the direct form FIR filter utilized in this study is the addition process. This is because the addition of signals is often performed with one summing operation, thereby resulting in the logic path between the input and output register becoming long, which leads to timing issues. This is illustrated in Figure 1.1.

Pipelining is one way to solve this problem. The basic idea behind pipelining is to reduce the path between the registers without changing the design functionality. Hence, operations are put between the registers. This process breaks the addition into multiple stages by placing registers in between the process, which is more favorable for hardware implementation. By pipelining the filter can process the delayed samples as fast as the system clock can go. However, dedicated hardware is required for each element from the filter block, thereby resulting in a large consumption of the available DSP resources. One of the benefits of this implementation method is that they are useful in high-speed implementation. This process is illustrated in Figure 1.4.

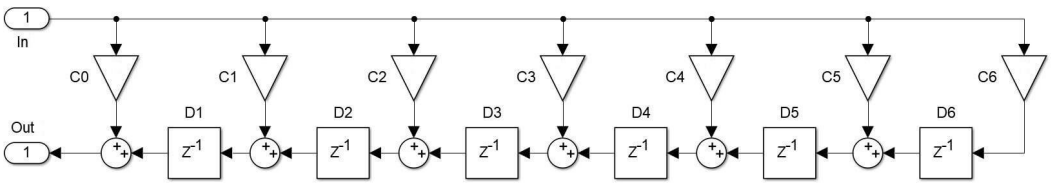

Figure 1.4. Transposed FIR filter (reproduced from ([21]).

However, the best choices are not parallel implementation because it can lead to sometimes timing issues i.e the implementation cannot process data as fast as the system clock. Therefore, serial implementation is preferred in such cases. In serial implementations, the convolution process is broken down into steps and executed one at a time. This implementation feeds the filter input and the coefficients to the same multiplier resulting in the reduction of arithmetic resource requirements[21]. The structure of the filter that is more suitable for serial implementation is the direct form which is illustrated in Figure 1.1.

One of the goals of this work is to produce an area-efficient filter realization. Since the filter coefficient is symmetry, we can exploit this to minimize the arithmetic requirements. The symmetric of the filter coefficients is presented in Figure 1.5.

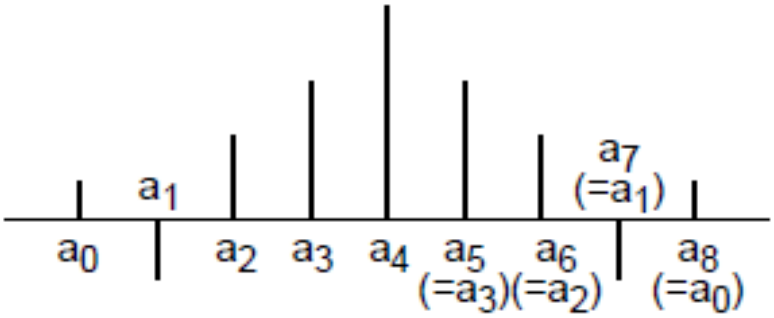

Figure 1.5. Symmetric FIR - Odd Number (reproduced from ([22]).

Filter Implementation using the structure shown in Figure 1.1 requires N multiplications and $N-1$ additions. This approach generally uses a significant amount of hardware resources. Alternatively, A more efficient approach to implementing the filter is presented in Figure 1.6. This requires $N/2$ multiplications and approximately N additions. This can be used to create hardware implementations of filters that are more efficient by taking advantage of the reduced computational workload. [22]. For this work, this approach is used to implement the FIR filter.

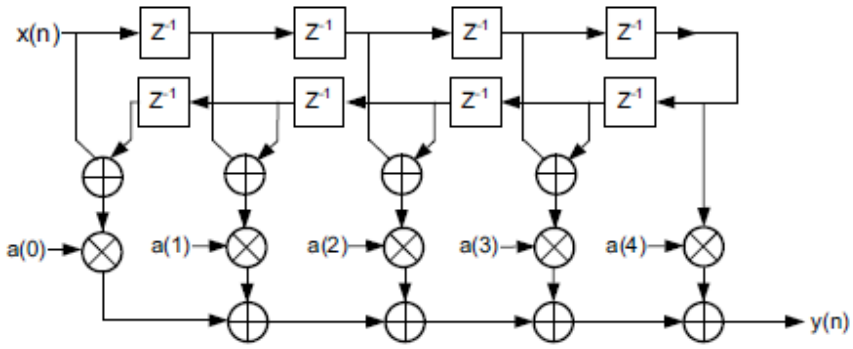

Figure 1.6. Coefficient Symmetry - Odd Number of Filter Taps (reproduced from ([22]).

In addition to that, another method in which the computation complexity of the FIR filter can be reduced is by using Canonic Signed-Digits (CSD) codes to implement the filter's coefficients

[23]. This implementation reduced the computation by replacing the multipliers that are required with a simple shift and adding a circuit [24]. Hence, leading to a reduction in area consumption and power dissipation.

1.2 Software and Tools

The Equiripple FIR filter algorithm is run using MATLAB and generates the double precision coefficients from the given filter design specification. These coefficients are quantized using the proposed method and the filter performance is measured in MATLAB. The Xilinx-coefficients(.coe) files are generated using MATLAB.

Xilinx provides a Vivado platform which is used for synthesis and analysis of the hardware design of the digital filter. The testbench of the software makes use of the coefficients and delayed data sample created in MATLAB to give us insight information on the area consumption of the filter.

1.3 Objective

The main goal of this dissertation will be to analyze the effect of the proposed dynamic quantization of coefficients on filter performance and compare this with the classical method of quantization which is a simple case of rounding the coefficients. With this comparison, we can hope to see which method gives a significantly better filtering performance. In addition to that, the effect of both quantized coefficients generated through our proposed method and the classical quantization method on Register Transfer Level (RTL) area consumption is also examined and studied.

1.4 Problem Statement

Digital filters are a key element in Digital Signal Processing systems, and when designing filters for applications that require linear phase response, the FIR filters are the most commonly used in modern digital applications due to their stability and linear phase response. FIR filters with symmetrical coefficients are commonly utilized in applications requiring linear phase response, particularly in high-frequency domains like video processing, as well as in scenarios that demand low power and high throughput, such as MIMO systems used in wireless communication. Such filters are particularly useful in data communications applications. [6].

General programmable digital signal processing cores are not sufficient for some applications, which require specific hardware design. Filtering is a fundamental operation in data processing that involves multiplying filter coefficients with delayed data samples. As a result, a significant amount of research has been conducted to enhance the design methods for FIR filters, with the goal of optimizing the hardware resources used to implement filters, including multipliers, adders, and registers. Therefore, quantizing filter coefficients can enhance several design parameters, such as area and power consumption.

Furthermore, the proposed quantization method in this work is designed to help reduce the number of hardware resources in the hardware implementation, and also the trade-off in the filter performance is also analyzed and examined.

1.5 Structure of Thesis

In Chapter 1, an introduction to the thesis, its background concepts, and objectives are provided. Chapter 2 discusses previous research on FIR filter design and coefficient quantization. Chapter 3 outlines the proposed framework, including the mathematical background of filter coefficients and algorithms, as well as the problem-solving approach. Chapter 4 presents the results and illustrations obtained from the study, along with discussions and analyses of the results and future research ideas. Finally, Chapter 5 concludes the thesis.

2 LITERATURE REVIEW

This section describes the relevant past works that have been done concerning the quantization of FIR filter coefficients and the effects they have on the FIR magnitude response. Studies on how FIR filters are implemented in FPGA are also mentioned in this section.

2.1 Related Works

Filtering make up the basis of any wireless system that is used in many applications such as cellular devices, medical equipment, and electronics. The paper by Singh T et al. presented the design of FIR filters using different techniques such as the Kaiser window, Hamming window, and Equiripple techniques [1]. A 2-D filter was designed, and the performance was compared to that of the 1-D filter. The result of this work shows that choosing one technique of designing an FIR filter over another has its advantages, which are based on the specific application they are going to be used for to achieve the desired output. The simulation results show that the FIR filter that was implemented using Equiripple techniques was more reliable than other techniques and the most promising candidate for implementing the filter if the sole purpose was to have a reduced magnitude error. Furthermore, it was discovered that filters using Hamming Window techniques, provides more accuracy to the original frequency spectrum because their ripple size gets reduced. Additionally, in filters using Kaiser Window techniques, it was discovered that changing the length of the filter does not affect the ripple size. Moreover, the comparison of the 2-D filter and the 1-D filter shows that the former is more stable and dependable, with better noise reduction capabilities. Consequently, the 2-D filter provides a more precise information signal compared to the 1-D filter.

When creating linear-phase FIR filters, a common objective is to enhance the accuracy of fixed coefficients in the filter without expanding the number of coefficients or the length of coefficients. The purpose of this goal is to achieve better performance. The article by Shen Z presents two new methods which are a parallel and serial method for coefficient quantization [14]. This approach improved the performance of the filter without requiring an increase in the number or length of the filter coefficients. The technique is compatible with both FPGA processing and DSP chips using a serial method, and with FPGA implementation using a parallel method. Another related study by Rajan A et al., titled Minimizing quantization effects in digital filtering, demonstrated the impact of utilizing integer sequences as windowing functions in FIR filtering. [13]. It was discovered in this study that while using the integer sequences window, the quantization error is much less pronounced compared to when using the Kaiser window method, but the magnitude response of both methods is comparable. Based on the results obtained from this study, the conclusion reached was that to avoid the effect of quantization significantly compromising the filter performance, the integer sequence as a windowing function is the best approach.

Kotteri K et al. presented a quantized filter design using compensating zeros techniques [25]. This method involves a cascade filter structure and involves quantizing the filter coefficient in the first cascade section. The coefficient for the second section is then computed to closely match the original filter, and this section is referred to as the compensating section. The purpose of this section is to compensate for the degradation in filter performance resulting from the quantization of the first section. The study demonstrates that the quantized filter's magnitude and phase response closely resembles those of the unquantized filter and that the area needed for implementation is found to be minimal. Izydorczyk J developed a smart and easy algorithm for FIR filters [4]. This work is focused on improving on the study from Kotteri K et al., which

was previously discussed. The proposed algorithm transforms a filter into two cascade FIR sections, in which the coefficients obtained from both sections are quantized. However, the second section coefficients are first computed to minimize Mean Squared Error (MSE) before being quantized. The algorithm is optimal in the MSE sense, and it is based on approximating the largest coefficient using one term while quantizing the difference between the approximation and the remaining coefficients. When comparing this research work to that of Kotteri K et al., in which the FIR filter was quantized using compensating zeros method [25]. It was found that the number of arithmetic operations that are required in the proposed algorithm in the Izydorczyk J work is lesser than that used in the compensating zeros method [25]. This means the number of hardware resources that would be used to implement the proposed algorithm would be less than that used in the method proposed by Kotteri K et al study.

FIR filter design methodology for hardware-optimized implementation by Rizwana et al. proposed a novel method for designing an FIR filter [26]. This method designs the filter first with a certain specification and obtains the coefficients of the filter. This first filter is redesigned by over-designing it with a strict constraint which leads to an increase in the filter order. The filter coefficient from the redesigned filter is quantized with a lower number of bits using the proposed iterative algorithm which results in the frequency response of the redesigned filter matching the original requirement. The study's results suggest that the filter's area consumption is lower than that of the original filter. Agarwal et al., put forward a hardware design approach aimed at hardware implementation optimization by reducing the filter order. [6].The Equiripple method was used to design the proposed filter, and this method helps to achieve a minimum order. The filter order is then iteratively reduced, with an increase in the stop band weight whenever the order is reduced, to ensure the attenuation remains within limits. The algorithm stops when the passband starts to increase slowly. The optimized filter is then obtained, and its coefficient is quantized. The results show that the optimized filter requires 28% less hardware resources compared to the filter in the Rizwana study [26].

DeBrunner L et al. studied the trade-off between the coefficient length and filter order in the design and implementation of high-performance filters in FPGA [20]. In this study, a non-minimum order FIR filter was designed using multiplier-less implementation techniques based on canonical signed digit (CSD). The length of the coefficients was reduced by increasing the filter order without affecting the filter performance, as the filter was already long. This reduction in bit length resulted in significant savings in the entire filter design. The total number of binary bits decreased as the order increased, leading to fewer non-zero CSD bits and reduced hardware complexity. Another investigation on coefficient quantization and internal rounding in the FIR filter was carried out by Magar M et al., to determine the minimum required area for implementation [27]. The study's findings suggest that utilizing a similar number of bits for quantizing the coefficients and internal results can lead to a decrease in area consumption while maintaining near-optimal performance, determined by the output quantization noise power (OQNP).

Yang et al. suggested a design technique for FIR filters with constant coefficients that is focused on Very-large-scale integration (VLSI) and uses algebraic integer quantization [2]. This method uses algebraic integer representation to quantize the filter coefficient. The benefit of this method is that the multipliers in the filters are replaced by a few adders, which results in a reduction in the hardware implementation resources. The synthesis results of this proposed filter show a 10-20% reduction in area consumption when compared to the traditional quantization method. In addition to that, the proposed method structure has a simpler on-chip interconnection structure with less delay. Bhalke et al. work presents the implementation of an FIR filter using a Finite State Machine (FSM) to filter out noise from biomedical signals with the aim of reducing area consumption by reusing hardware resources [9]. The use of FSM helps reduce the

logic complexity. A comparison was made between the proposed approach and the MATLAB-Simulink-based FIR filter, and the synthesis results showed that the proposed approach has lower area consumption and timing delay with high frequency compared to the MATLAB-Simulink-based FIR filter.

Implementation of FIR filter based on Xilinx IP core by Yu et al., focused on developing a new generation of high-performance and high integration Remote Terminal Units (RTU), which are used in a power supply system to collect and calculate real-time data such as voltage, current, and power [8]. However, to calculate the voltage and current signals effectively, it is important to filter out the harmonics in the signal while developing a power RTU with the Zynq-7000 series. This work considered the commonly used method for filtering out harmonics, which are the Fast Fourier Transform (FFT) and digital finite filter (FIR), however, the method proposed in the study utilized the Xilinx FIR IP core to perform filtering, and the results showed that this approach used fewer resources than the Fourier Algorithm..

Rowell A, thesis reviewed the fundamentals of coefficient quantization and explored its impact on the maximum ripple of a Parks-McClellan filter design in an FIR filter. [15]. In this work, Extreme Value Theory (EVT) was used to effectively model ripples that are caused by coefficient quantization. The performance of the filter was studied after quantization and a two-novel method for optimizing quantized filter coefficients was presented. The first method was by using gain adjustment. A single gain block was added after the filter and the purpose of this was to generate the different realization of the quantized filter, in which the one with the best performance is selected for implementation. The idea is to select a gain and then the FIR filter coefficients are divided by this gain which results in a sequence and then produces a scaled-down coefficient. These downscaled coefficients are then quantized thereby leading to different error ripple patterns. Furthermore, after the filtering, the output of this filter is multiplied by the gain to compensate for the effect of the gain. However, the system complexity will increase by adding a gain block, leading to higher hardware resource utilization. The second approach suggested is parameter perturbation. This method makes slight changes to the filter specifications and generate many realizations of the filter coefficients and selects the filter with a better performance compared to the original sets of filter coefficients. This work also analyses digital overflow using EVT. The researcher created a model of the distribution of extreme values obtained from recorded data and utilized this model to improve the estimation of overflow rates.

Bugrov V et al. introduced a new algorithm for dynamic coefficient quantization that utilizes integer nonlinear programming methods in a step-by-step process [3]. Their proposed algorithm was used to quantize the coefficients of a 4-bit cascade high-pass FIR filter with a minimum bit-depth representation of integer coefficients. Compared to the classical statistical method of quantization, the proposed algorithm resulted in a nearly ten-fold reduction in the functional error of the filter. Izydorczyk J, presented an algorithm using the signed integer power of two-term (SPT) format for the optimal quantization of FIR filter coefficients [10]. The proposed algorithm was used to quantize the Parks-McClellan filter coefficients. The idea of this algorithm is that the SPT terms are selected one at a time and they are allocated to the coefficient that is the most deserving, in other to reduce the Euclidean distance between the corresponding infinite word length and the SPT coefficients. This process is repeated iteratively until the total number of the SPT terms is the same as the prescribed number for the filter. This algorithm reduces the computation complexity of the filter. Zeinolabedin S et al. presented a new approach for quantizing FIR filter coefficients [5]. The algorithm proposed can be applied to quantize coefficients of various types of FIR filters, irrespective of the design method used. This study proposed two novel methods for quantization of the filter coefficient and the results obtained show that these two methods have a better performance in comparison to previous algorithms.

The study by Stošić B investigated the impact of coefficient quantization on novel cascaded

integrator comb (CIC) FIR filters. The work proposed generating these new filters by utilizing classical CIC FIR filters as a function [11]. To design the new CIC FIR filter, the delay in the CIC filter comb stages was spread out, and the resulting frequency response was compared with that of the conventional CIC filter. The results show that the new CIC FIR filters have a higher selectivity and insertion loss in their stopband. In addition to that, the obtained coefficients of the new CIC FIR filters have a smaller integer value when compared to the classical CIC filters, and this is important when taking the hardware realization into account due to the finite word length effects. The study found that when filter coefficients are quantized, it leads to a change in the frequency response of the filter when compared to the unquantized filter. Furthermore, reducing the word length of the coefficients resulted in undesirable effects. Stošić B presented an approach to designing filters by using identical and non-identical CIC sections [28]. This study produces a set of low-pass filter coefficients with the help of the proposed CIC sections. A comparative study between direct and cascade form realization of FIR filter was performed and the result of this analysis shows that direct form realization of filters is sensitive to quantization errors while the cascade realization produces smaller sensitivity. In addition to that, the paper suggests the use of cascade sections when implementing higher order filters due to higher order filters requiring a higher number of bits to represent its coefficients and cascade can help to significantly reduce the sensitivity of the coefficient's quantization.

This study by Vandenbussche J et al., aimed to analyze the impact of coefficient quantization on the performance of Multiplicative Finite Impulse Response (MFIR) filters, which are commonly used to model pole filters and to determine the optimal number of bits required for quantization. [12]. The analysis was performed for different coefficients length and the zero-displacement sensitivity, frequency domain, and statistical analysis were used to determine the performance of the filter. The research suggested a technique for achieving the minimum number of bits necessary for coefficient quantization, based on a predetermined maximum magnitude response deviation. The findings of this investigation indicate that the MFIR pole approximation filter does not require a large number of bits for coefficient quantization when compared to an equivalent IIR filter. Lin T et al. introduced an algorithm that quantizes the coefficients of FIR filters by distributing a preset addition budget to the quantized coefficients, with the elimination of common subexpressions (CSE) [29]. The algorithm proposes applying an optimal scale factor within the gain tolerance to collectively settle the coefficients into a quantization space. The simulation results show that the algorithm has almost the same performance as that with the CSD coefficient, and the CSD coefficients are known to have theoretically minimum non-zero terms.

2.2 Summary

This section reviewed various works and studies that have been done about the quantization of FIR filter coefficients and the FPGA implementation of these filters. According to this literature review, we can conclude that in signal processing technology the digital filter can be considered the most crucial component. They have extensive applications in many digital fields like biomedical signal processing, video processing, data transmission, image and speech processing, and digital audio. Filters are considered the basis of every communication system.

The review of the literature indicates that many studies have been conducted to examine the impact of coefficient quantization on the magnitude response of digital FIR filters, but there have been relatively few studies investigating how coefficient quantization affects the magnitude response of the Equiripple FIR filter. In this dissertation, we would examine two methods of quantizing the Equiripple filter coefficient, which is using the classical fixed-point method and

the dynamic coefficient quantization method. A comparative study between the algorithm for the dynamic coefficient quantization and the classical fixed point quantization method is done to determine which method provides us with an optimal filter performance and consumes fewer hardware resources.

3 METHODOLOGY

This section describes the proposed coefficients quantization framework. It begins with laying out the mathematical background that is required to understand this proposed quantization technique. First, we start with the basic introduction to floating point and fixed-point quantization and the concept of rounding is presented. We conclude this section with an overview of the proposed quantization technique.

3.1 Digital Value Representation

The way digital values are represented is a very important topic in DSP, and a look at how numbers are represented in a modern computer will give us more understanding of why it is considered important. Representation of numbers in a modern computer is often done in either fixed point or floating. Both fixed-point and floating-point number representations require a certain number of bits to store their value, and increasing precision typically requires a larger number of bits.

For floating point number representation, the floats are discrete but are not equidistant. In other words, floating-point number representation has uniform discrete values that are close to each other, but the step size becomes more significant as the value's magnitude increases. This feature allows the representation of both large and small values in memory using the same number of significant bits, resulting in a broad dynamic range. Despite being commonly used in modern computers, using floating-point representation for filter design can be expensive in terms of design time, area, and power consumption.

The IEEE 754 standard [30] is commonly used to implement floating-point numbers and it specifies single and double precision. Although, they can be implemented in other ways that are slightly different from the standard which is a common practice in modern Graphics Processing Unit (GPU). In addition to that, lower precision compared to the single precision that is defined in the standard can be used to produce custom Application-Specific Integrated Circuit (ASIC) chips, and simulation of floating point using fixed point processors can be done using binary shift algorithms. This type of implementation can utilize the broad range of values available in floating point numbers even when using low levels of precision. When defining the values of the floating point, these two parameters are used [31].

- The word length, which encompasses a sign bit indicating the positivity or negativity of the number, represents the total number of bits.
- The length of the mantissa refers to the number of bits used to represent the value's precision, while the remaining bits, excluding the sign bits, are used to represent the value's exponent.

In fixed-point number representation, the discrete values are uniformly spaced and these values are normally stored as 2's complement arithmetic. As explained earlier, floating point processors require more area and power consumption when compared to fixed point processors. Hence, using this processor will reduce the manufacturing price and computation complexity. When defining the values of the floating point, these three parameters are used.

- The total number of bits is also referred to as the word length.
- A signed bit is needed to indicate the positive or negative sign.

- In binary number representation, the position of the binary point is specified by the number of fractional bits. If this value is greater than the total number of bits, it means that one (1) will have a higher value than the highest representable value. This number can be negative or zero, indicating that only integers or multiples of integers can be stored.

One needs to be cautious when using a fixed point to represent the number, this is because overflow occurs when fixed point values become large for it register to store and also result in the value being wrapped around to a negative value. This overflow can cause large errors in digital systems which can be difficult to predict.

3.2 Rounding in Digital Signal Processing

Rounding proves to be necessary for DSP applications when the aim is to convert floating-point number representation to a fixed-point number. In the field of DSP, it is common to utilize floating-point representation for algorithm analysis and assessment, with fixed-point representation being employed for hardware implementation purposes at a later stage. Hence, rounding introduces quantization errors or computational noise to our algorithm. The most common techniques that are used for rounding algorithms when it comes to hardware implementation are truncation, round-half-up, and round-floor.

3.2.1 Truncation

Truncation can also be referred to as chopping, the basic principle of truncation is that unwanted digits are discarded. When working with signed and unsigned binary values, the results that are obtained from truncation are different. The outcomes of truncation differ when working with signed and unsigned binary values. In the case of unsigned binary values, truncation makes the values move toward zero. For example, a standard decimal value of 2.1 through to 2.9 would be rounded down to 2. However, for signed binary values or negative digits, truncation works differently from what is described in the example above. We would explore this in the subsequent subsection.

3.2.2 Round Half Up

Rounding-half-up is also known as arithmetic rounding. To explain how this algorithm work, ten numbers that begin with a 2 in the most significant place are considered (2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, and 2.9). In this particular instance, half of the values round up and the other half round down. In other words, the values 2.5 through 2.9 is rounded up to 3 while the remaining five values 2.0 to 2.4 are rounded down to 2. When considering negative numbers, -2.1 through to -2.4 would round to the nearest integer -2, and -2.6 through -2.9 would round to -3. However, for the -2.5 case, there is a problem with what our definition of “up” means. Since +2.5 rounds to +3, we would assume that -2.5 would round to -3. This algorithm is symmetric for both negative and positive values, which makes it appropriate for some applications. In certain scenarios, the term ”up” may be used to represent positive infinity. In this case, -2.5 rounds to -2 which would make the algorithm to be asymmetric. For this dissertation, we are using MATLAB, and this provides us with the symmetric implementation of the round algorithm.

3.2.3 Round Floor

This is commonly referred to as rounding toward negative infinity. For negative numbers, the numbers are rounded down except if the number to be discarded is zero. -2.0 would remain unchanged while -2.1 through to -2.9 would be rounded down to -3. For positive numbers, all unwanted numbers are discarded. for example, 2.0 through to 2.9 would be rounded to 2. The round floor is sometimes used in a diagnostic function to determine the algorithm's lower limit. From a hardware implementation point of view, this algorithm is considered cheap because it only involves simple truncation. However, they result in cumulative negative bias.

3.2.4 Hardware Implementation and Implications

For hardware implementation, rounding is done for unsigned and signed binary representations. Rounding is done as discussed above when working with unsigned binary representation. However, the results of rounding vary from that of unsigned binary representations when using signed binary values. In order to comprehend the mechanism of rounding in signed binary, an analysis of 8-bit signed binary is employed. The 8-bit signed binary is made up of four integer bits and fractional bits respectively. This is presented in Figure 3.1

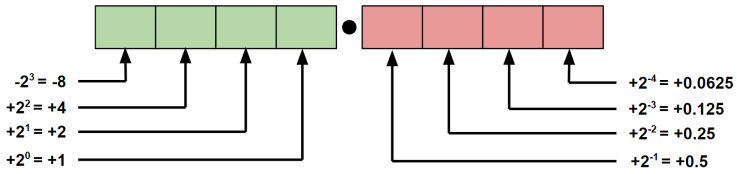

Figure 3.1. 4.4 fixed-point representation.

For example, we represent +2.5 in this format and the value equates to 0010.1000. rounding this number using the truncation method would produce a value of 0010 which equates to 2. This method as discussed earlier; discards the fractional bits. For signed bits, we represent -2.5 which equates to 1011.1000. The value represented by the integer bits and fractional bits is -2.5, where the integer bits represent -3 and the fractional bits represent +0.5. When truncated to an 8-bit signed fixed-point representation, the resulting value is 1011, which corresponds to -3. Hence, performing truncation on signed binary bits results in the round floor algorithm as discussed above.

One popular algorithm for hardware implementation is round half up, the reason for this is that we are not required to perform comparison operations. To round to the nearest integer, this algorithm involves adding 0.5 to the original value and then truncating the resulting value. However, in the case where rounding is done to the nearest half, a value of 0.25 is added and truncation is performed on the result obtained. For positive values, let us consider +2.5 which is equal to 0010.1000, the addition of 0000.1000 and will give a result in 0011.0000 and this result is truncated resulting in 0011, which equates to 3. For a negative value, let us consider -2.5 which equates to 1011.1000. Furthermore, adding 0000.1000 and truncating the results produce the value 1110, which equates to -2 in decimal. Therefore, we can conclude that the round-half-up algorithm results in an asymmetrical realization.

Rounding filter coefficients is a method that can be employed to minimize quantization

errors in filter design. When comparing the different methods of rounding on filter coefficients as presented by [32], the results show that the algorithm which provides the best results is the round-half-up algorithm, this is because rounding the coefficients to the nearest fixed-point value minimizes the absolute difference between the original floating-point value and the rounded fixed-point value. Additionally, this results in some rounding errors being negative and others being positive, which can cancel each other out. In the case of the round floor, all coefficients tend to shift toward negative infinity which results in negative biases.

3.3 Analysis of sensitivity to coefficient quantization

A digital filter H transfer function, in which each coefficient has infinite precision was presented in the first section of this dissertation. To know the effect of coefficient quantization on this filter with N zeros and M poles. We can write the filter response as

$$H(z) = \frac{\sum_{n=0}^{N} b_n z^{-n}}{1 - \sum_{n=0}^{N} a_m z^{-m}} = \frac{B(z)}{A(z)} \tag{3.1}$$

Where $B(z)$ denotes the numerator of the filter while $A(z)$ denotes the denominator. The zeros and poles of a system are respectively the roots of the numerator and denominator. In a linear time-invariant system, $H(z)$ represents the transfer function. Here, a z-transform input $X(z)$ will result in an output of $Y(z) = Y(z) \cdot H(z)$. The quantization of filter coefficients could alter some or all the filter's coefficients which in turn leads to the generation of a new system function. This new function can be written as

$$H^{'}(z) = \frac{\sum_{n=0}^{N} b_n^{'} z^{-n}}{1 - \sum_{n=0}^{N} a_m^{'} z^{-m}} = \frac{B^{'}(z)}{A^{'}(z)} \tag{3.2}$$

$$b_n^{'} = b_n + \Delta b_n \tag{3.3}$$

$$a_m^{'} = a_m + \Delta a_m \tag{3.4}$$

Where coefficient b_n and a_m quantization errors are denoted by Δb_n and Δa_m respectively. By expanding Equation 3.2 above, It is possible to represent the quantized system in the same form as the original system. Which is

$$H^{'}(z) = \frac{\sum_{n=0}^{N} b_n^{'} z^{-n}}{1 - \sum_{n=0}^{N} a_m^{'} z^{-m}} = \frac{\sum_{n=0}^{N} (b_n + \Delta b_n) z^{-n}}{1 - \sum_{n=0}^{N} (a_m + \Delta a_m) z^{-m}} \tag{3.5}$$

The equation can be defined as

$$H^{'}(z) = H(z) + \Delta H(z) \tag{3.6}$$

Where the error filter is defined to be $\Delta H(z)$, which is $H^{'}(z)$ component due to the quantization of coefficients. To analyze the deviation in the system function, let us Equation 3.5,

$$\Delta H(z) = H^{'}(z) - H(z) \tag{3.7}$$

$$\Delta H(z) = \frac{B^{'}(z)}{A^{'}(z)} - \frac{B(z)}{A(z)} \tag{3.8}$$

$$\Delta H(z) = \frac{B^{'}(z)A(z) - B(z)A^{'}(z)}{A^{'}(z)A(z)} \tag{3.9}$$

$$\Delta H(z) = \frac{(B(z) + \Delta B(z))A(z) - B(z)(A(z) + \Delta A(z))}{A^{'}(z)A(z)} \tag{3.10}$$

$$\Delta H(z) = \frac{(B(z) + \Delta B(z))A(z) - B(z)(A(z) + \Delta A(z))}{A^{'}(z)A(z)} \tag{3.11}$$

$$\Delta H(z) = \frac{A(z)B(z) + A(z)\Delta B(z) - B(z)(A(z) + \Delta A(z)B(z)}{A^{'}(z)A(z)} \tag{3.12}$$

$$\Delta H(z) = \Delta B(z)\frac{1}{A^{'}(z)} + \Delta A(z)\frac{H(z)}{A^{'}(z)} \tag{3.13}$$

Where $\Delta B(z)$ and $\Delta A(z)$ can be define as

$$\Delta B(z) = B^{'}(z) - B(z) = \sum_{n=0}^{N} \Delta b_n z^{-n} \tag{3.14}$$

$$\Delta A(z) = A(z) - A^{'}(z) = \sum_{n=0}^{N} \Delta a_m z^{-m} \tag{3.15}$$

By looking at Equation 3.13, we can see that with respect to $\Delta A(z)$ the deviation is very nearly linear in the system function. Where, $A^{'}(z) = A(z) - \Delta A(z)$ and $|A^{'}(z)| \approx |A(z)|$. This deviation is caused by the quantization error.

For the FIR filter, except for a_0 the value of all coefficients a_m is zero. In which a_0 is equal to 1. In other words, the value of $\Delta A(z)$ is zero and $A^{'}(z)$ is equal to 1. Inserting this value into Equation 3.13, the expression of our error filter can be written as

$$\Delta H(z) = \Delta B(z) = B^{'}(z) - B(z) \tag{3.16}$$

This expression shows that the quantization of FIR results in the generation of a new system function as opposed to the system function when using infinite precision coefficients. Additionally, in the case of pole filters, except for b_0 the value of all coefficients, b_m is zero. In which b_0 is equal to 1. In other words, the value of $\Delta B(z)$ is zero. Inserting this value into Equation 3.13, the expression of our error filter can be written as

$$\Delta H(z) = \Delta A(z) \frac{H(z)}{A^{'}(z)} \tag{3.17}$$

$$\Delta H(z) \approx \Delta A(z) \frac{H(z)}{A(z)} \tag{3.18}$$

From the expression above, we can see that the system function deviation is proportional to $A(z)$.

3.4 Impact of Quantized Coefficients on FIR Response

The linear nature of the frequency response of the FIR filter implies that the frequency response of $B^{'}(z)$ is the combination of the frequency responses of $B(z)$ and $\Delta B(z)$ as shown in Equation 3.14. This can be written as

$$B^{'}(\epsilon^{j\omega}) = B(\epsilon^{j\omega}) + \Delta B(\epsilon^{j\omega}) \tag{3.19}$$

From this expression, we can conclude that $\Delta B(\epsilon^{j\omega})$ perturbed the frequency of the quantized filter for any frequency ω. In addition to that, after coefficient quantization, the linear phase of the FIR filters and their error filter will remain linear phase. This is because, the equivalent coefficient sets are quantized to equal levels after quantization, thereby enabling it to maintain its linear property. A key point to keep in mind is that a filter is only considered linear phase if its filter coefficients are symmetric or anti-symmetric. In the scenario of quantization using fixed-point, the quantizer step sizes are fixed and uniform, but in floating-point quantization, the quantizer step size can be assumed to be proportional to b_n.

MATLAB is a software that can be used to produce the filter coefficients, although, these coefficients are represented in double-precision which has a near infinite precision. To demonstrate how the quantization of coefficients can affect the magnitude response of an FIR filter, we can examine a 124th-order Equiripple low-pass FIR filter design. The lowpass filter has the following specification

- Sample rate: 96000 Hz.
- Passband-edge frequency: 20000 Hz
- Stopband-edge frequency: 22000 Hz

This filter has its coefficients represented in a double-precision floating-point. To quantize these coefficients, a fixed-point quantizer with a step size of $q = 2^{-N}$ is used, where N represents the number of bits after the binary point. For this analysis, the double-precision coefficient was

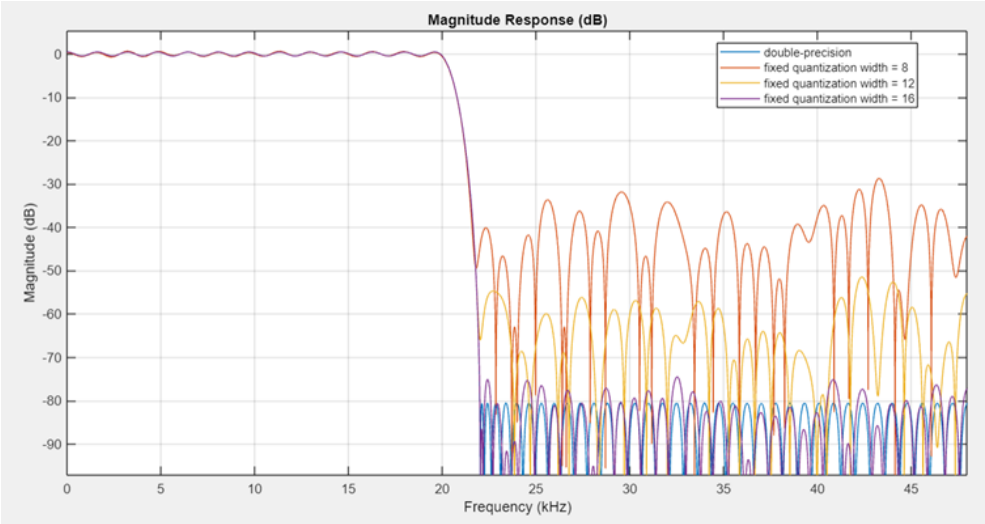

Figure 3.2. Magnitude responses of quantized FIR filters.

quantized to a coefficient width of 16, 12, and 8. The magnitude response for this different quantization width and the double precision filter is presented.

From this analysis, the stopband attenuation of the double precision filter is 80.6dB, and the performance of the filter drops to 74.6dB when the quantized coefficient width is 16. In addition, when the quantized coefficient width is 12 and 8 the stopband attenuation is 51.5dB and 28.7dB. This study shows that the performance of the filter begins to drop lower as the width of the coefficient reduces.

3.5 Proposed Framework

The previous section demonstrated that as the coefficient width decreases, the filter performance also decreases. In the framework of this dissertation, one of the goals is to increase the FIR filter quantized coefficient precision and this results in better performance of the filter. To describe this framework, let us consider one of the quantized filters that were implemented in the previous section. This filter was implemented using the serial implementation method. This method breaks the convolution process into steps and is executed one at a time. This implementation feeds the filter input and the coefficients to the same multiplier resulting in the reduction of arithmetic resource requirements. Due to the symmetry of the filter coefficient, the filter implementation requires $N/2$ multiplications, which significantly reduces the computation workload. Figure 3.3 displays the filter structure while Figure 3.4 shows the impulse response.

The filter coefficient when quantized to a 12-bit two's-complement format is presented in decimal integers and binary values in Table 1. From the table, the other coefficients are smaller when compared to the middle tap coefficient b_{62}. Due to the fixed-point quantization, the high-order bits are the same as the sign bits for the low amplitude coefficient. This is shown in the bits that are underlined in the binary format section in Table 3.1. These bits do not have an effect on the computation thereby leading to them being wasted.

Replacing this consecutive bit that is closer or adjacent to the sign bits with more significant coefficient bits is the central point of this work and the removal of this wasted bit will help us obtain improved filter coefficient precision.

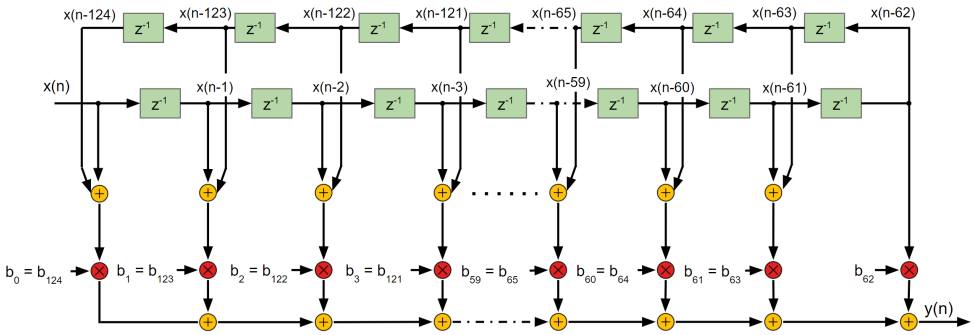

Figure 3.3. Filter structure

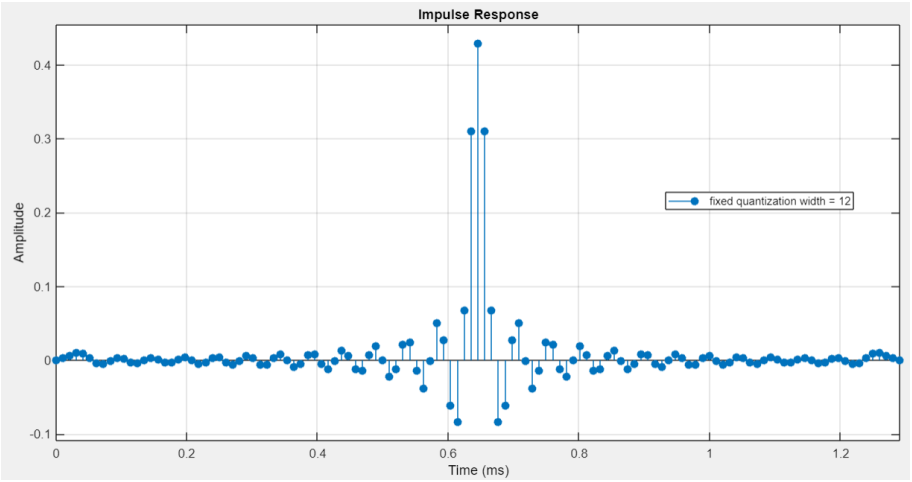

Figure 3.4. Impulse response

Table 3.1. 12-bits quantized coefficients

	Fixed Point Representation	Decimal Format	Binary Format
b_0	0.00048828125	2	000000000010
b_1	0.0029296875	12	000000001100
b_2	0.0068359375	28	000000011100
b_3	0.01025390625	42	000000101010
b_4	0.009033203125	37	000000100101
b_5	0.0029296875	12	000000001100
b_6	-0.003662109375	-15	111111110001
b_7	-0.0048828125	-20	111111101100
:	:	:	:
b_{62}	0.429443359375	1759	011011011111
:	:	:	:
b_{122}	0.0068359375	28	000000011100
b_{123}	0.0029296875	12	000000001100
b_{124}	0.00048828125	2	000000000010

3.5.1 Dynamic Quantization Method

This proposed framework is called the dynamic quantization of filter coefficients because this algorithm takes advantage of the dynamic range of floating-point filter coefficient to produce

a filter coefficient that is more accurate and allows for the use of smaller bit-width without sacrificing too much drop in performance when compared to the classical fixed point quantization method. To explain how the wasted bits are compensated, a general block diagram for step-by-step dynamic quantization of double-precision filter coefficients is shown in Figure 3.5. The process involves scaling the coefficients up in a power-of-two format until they are close to 1 for positive coefficients and -1 for negative coefficients.

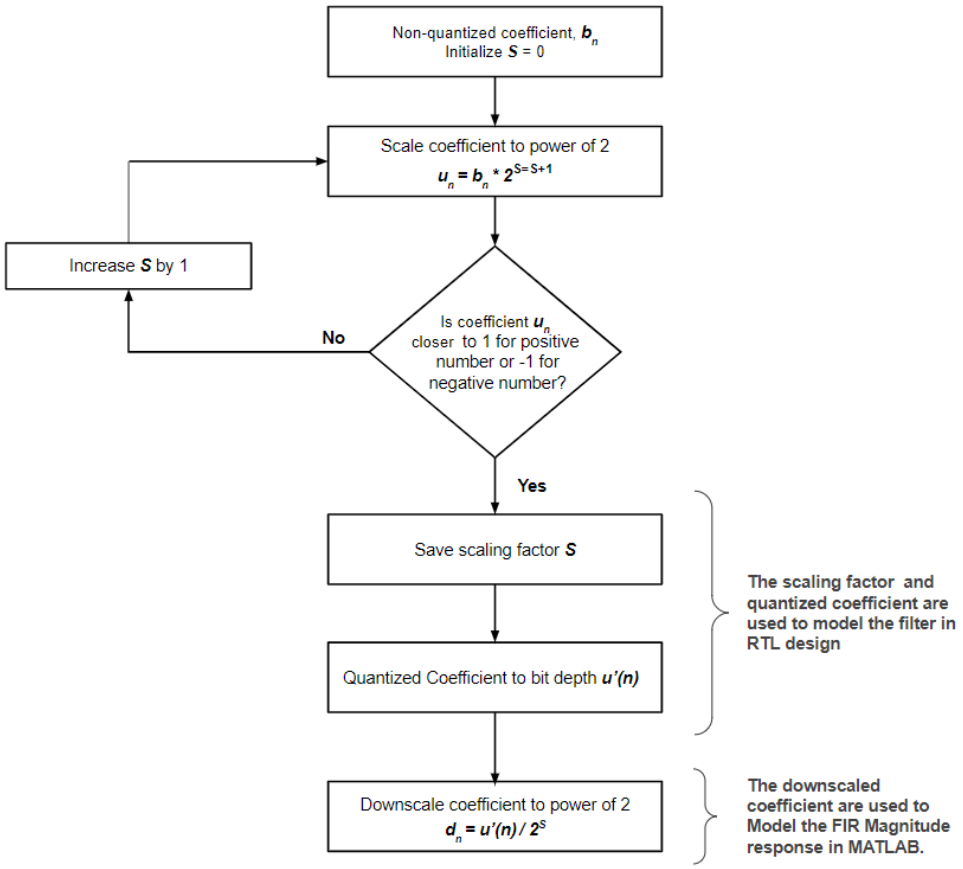

Figure 3.5. Dynamic quantization algorithm structure

To start the algorithm, we first obtain the double-precision filter coefficients b_n for a 124^{th}-order low-pass Equiripple FIR design based on the filter specifications described in the previous subsection. These coefficients are then scaled up to the nearest power of two to generate a new coefficient u_n that is approximately equal to 1 or -1. One possible scaling method involves repeatedly multiplying the coefficient by two until it approaches 1 for positive values or -1 for negative values, which can be achieved using a loop. This operation eliminates the extra signed bits. However, to ensure that the coefficient remains within a valid range, it must not exceed 1 for positive values or be less than -1 for negative values. Therefore, if the coefficient value exceeds these limits after being multiplied by two, we set the new coefficient value u_n to the previous coefficient value before the multiplication. The scaling factor S, which determines the number of times each coefficient is multiplied by 2, is obtained and used for the RTL implementation of the filter to downscale the coefficients.

The next step after scaling up the coefficient is to quantize the coefficient according to the

bit depth defined. This coefficient quantization can be referred to as a simple case of coefficient rounding. These rounding methods are generally dependent on coefficient actual values and how the nearest quantization step is far from the floating-point values. Hence, a different method of rounding will introduce a different type of quantization error. However, using the round-half-up method is the best approach for minimizing quantization error, due to it producing both positive and negative rounding errors which result in both rounding errors canceling each other out. The rounding method used for this coefficient quantization is the round-half-up method. Table 3.2 shows the floating-point coefficient b_n, Upscaled coefficient u_n, the dynamic quantized coefficient $u_n^{'}$ and scaling factor S.

Table 3.2. Dynamic quantization filter coefficient

	Float Coefficient b_n	Upscaled Coefficient u_n	Quantized $u_n^{'}$	S
b_0	0000000000 1001100011	0100110001011101 1111	010011000110	10
b_1	0000000 1011111110001	0101111110001000 0111	010111111001	8
b_2	0000001 1100010010 00	0111000100100000 1100	011100010010	7
b_3	0000010 1001100111 01	0101001100111001 1000	010100110100	6
b_4	0000010 0101001000 11	0100101000000 1011101	010010100000	6
b_5	0000000 1100000011 10	0110000001110001 0110	011000000111	8
b_6	1111111 0001000 10100	1000100010100010 1000	100010001010	8
b_7	1111110 1011110 11001	1010111101100011 1000	101011110110	7
:	:	:	:	:
b_{62}	0110110111110 11101110	0110110111110 11101110	011011011111	1
:	:	:	:	:
b_{122}	0000000 1110001001000	0111000100100000 1100	011100010010	7
b_{123}	0000000 0101111110001	0101111110001000 0111	010111111001	8
b_{124}	0000000000 1001100011	0100110001011101 1111	010011000110	10

Notice that we do not have any more wasted bits as they have been removed when the coefficient was scaled up to the power of two. However, using this quantized coefficient directly will make the filter results to be wrong and the performance of the filter to be worst when compared to the fixed-point quantization method, due to the removal of the wasted bits. Therefore, to get an accurate filter result, we must shift the filter results obtained by the filter using the dynamic quantization method to the left by the difference between the highest value of the scaling factor and the scaling factor of the coefficient. Shifting left in hardware implementation is the same as dividing the results by the power of two. This way we get an accurate filter result and much-improved filter performance.

To show how this new algorithm affects the magnitude response of the FIR filter, we down-scaled the quantized coefficient by using the scaling factor S obtained when scaling up the coefficient to a power of 2. The easiest way to downscale this coefficient is to loop and divide the coefficient by 2, with a gradual decrease in the scaling factor by 1 each loop till the scaling factor value reaches zeros. The reason for this is that MATLAB has an existing filter function that would be used to plot the magnitude response and obtain output results of the filter, thereby modeling the behavior of our hardware-implemented FIR filter design in MATLAB requires us to downscale the quantized coefficient $u_n^{'}$. The filter result obtained from this is compared with that obtained from the hardware implementation, to ensure that they correlate.

3.5.2 Hardware Implementation

The dynamic filter block implements an efficient low-pass finite response filter across a broad range of parameters described in the generic block with a sampling rate of 96 kHz. This block performs filtering on a stream of input data and produces a stream of output data with the same sampling frequencies as the input data. These output data produced are full precision fixed point type and input data and coefficients are fixed point data type. Filtering is implemented as a symmetry filter, this is because the impulse response of the filter possesses significant symmetry, and this can be taken advantage of to reduce arithmetic requirements and area consumption. Since these coefficients are constant values, these filter blocks do not allow reading and writing coefficients directly because they have predefined coefficients. However, it allows the user to choose which coefficient value that would be written as described in the generic block. The dynamic filter structure and filter block diagram are defined in Figure 1 and Figure 2.

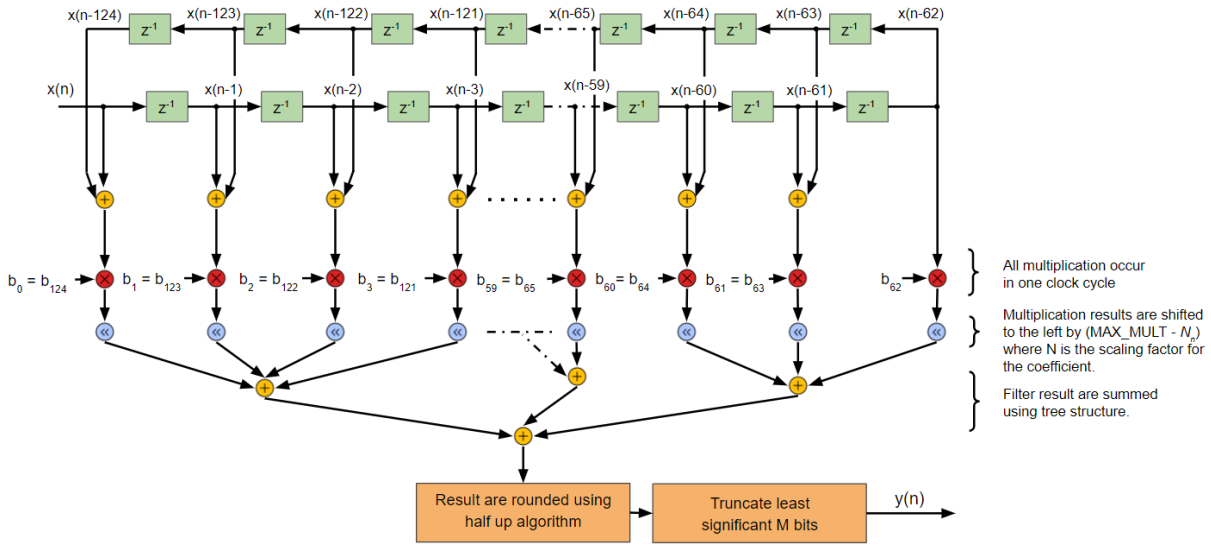

Figure 3.6. Dynamic filter structure

RTL Diagram

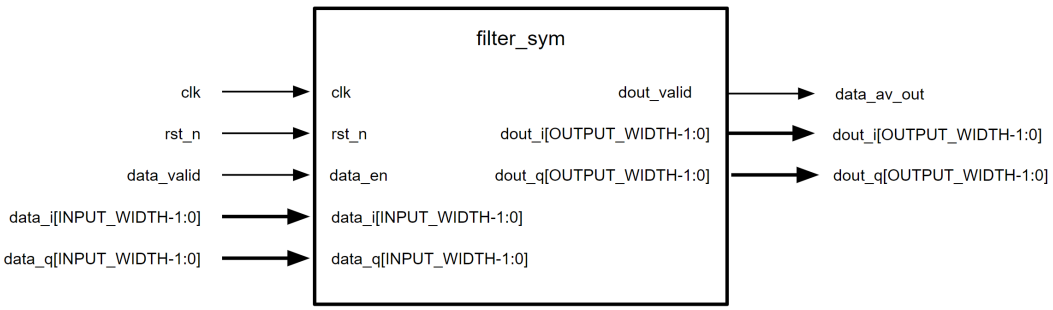

Figure 3.7. Filter block diagram

Generic

Table 3.3. Parameters for the dynamic FIR block

Generic	Type	Description	Value
FILTER_TAP	natural	The number of coefficients. Given values need to be odd because the filter is using odd symmetry	125
COEFF_TAP	string	Use to set the coefficient value for the filter	Available coefficient files: dynamic_coeffs16.txt, dynamic_coeffs12.txt, dynamic_coeffs8.txt,
INPUT_WIDTH	natural	Input data bus width in bits	16
COEFF_WIDTH	natural	Coefficient data bus width in bits	Set based on the coefficient file selected.
OUTPUT_WIDTH	natural	Output data bus width in bits	16
MAX_MULT	integer	The highest value of the scaling factor	15

Interface Ports

Table 3.4. Ports for the dynamic FIR block

Name	Direction	Width(bits)	Type	Description
clk	input	1	logic	Rising-edge sensitive clock signal.
rst_n	input	1	logic	Active-low, asynchronous reset signal.
data_i	input	(INPUT_WIDTH-1 downto 0)	logic	16-bit sample data input for the real part.
data_q	input	(INPUT_WIDTH-1 downto 0)	logic	16-bit sample data input for the imaginary part.
data_en	input	1	logic	Data load enable pulse whose 1-state indicates that *data_i* and *data_q* contain a new data sample that should be loaded into the delay line register
dout_i	output	(OUTPUT_WIDTH-1 downto 0)	logic	16-bit output data for the real part.
data_q	input	(OUTPUT_WIDTH-1 downto 0)	logic	16-bit output data for the imaginary part.
dout_valid	output	1	logic	Data out enable pulse whose 1-state indicates that *dout_i* and *dout_q* contain valid filtered result.

Functional Description

The filter operation starts when the $data\_en$ pulse is raised, the filter takes in data samples from $data\_i$ and $data\_q$ every clock cycle and stores them in a delay line register $delay\_line\_i(A)(INPUT\_WIDTH-1\ downto\ 0)$ and $delay\_line\_q(A)(INPUT\_WIDTH-1\ downto\ 0)$ respectively, where A ranges from 1 to $FILTER\_TAP$. These delay line registers are a two-dimension array that has a length of $FILTER\_TAP$. Additionally, the memory element of the delay line structure is shifted up ($delay\_line\_i(A) <= delay\_line\_i(A-1)$) every clock cycle, and the new data is stored at $delay\_line\_i(0)$. The coefficient file selected is mapped into the $coeff\_s$ constant array, and it is the same length as the delay line register. The scaling factor S is also mapped into a constant array num_mult.

The filter performs a pre-addition operation on the data stored in the delay structure register, in which the first memory element stored in the delay structure $delay\_line\_i(0)$ is added to the last memory element stored in the delay register $delay\_line\_i(124)$, and the second memory element stored in the delay structure register $delay\_line\_i(1)$ is added to the second to the last memory element stored in the delay register $delay\_line\_i(123)$. This addition cycle continues till the middle element is reached ($delay\_line\_i(62)$) in which no pre-addition is required. This operation is illustrated in Figure 3.7. The result of this addition is stored in the pre-addition register $add\_i(B)(INPUT\_WIDTH\ downto\ 0)$ and $add\_q(B)(INPUT\_WIDTH\ downto\ 0)$, where B ranges from 0 to $(FILTER\_TAP-1)/2$. These pre-addition registers are a two-dimension array that has a length of 63. All addition is done in one clock cycle. Notice that the width of the pre-addition register is 1 bit larger than that of the delay line structure, the reason for this is to avoid overflow.

The filter performs the multiplication operation, in which the pre-addition register element is multiplied with their corresponding coefficients $add\_i(C)*coeff\_s(C)$, and the result are stored in the multiplication register $p\_mult\_i(C)(INPUT\_WIDTH+COEFF\_WIDTH\ downto\ 0)$. Where C ranges from 0 to $(FILTER\_TAP-1)/2$. These multiplication registers are a two-dimension array that has a length of 63. Notice that the width of the multiplication registers is larger than that of the pre-addition register, the reason for this is that when two values of N-bits and M-bit are multiplied, the output result will be (N+M)-bits, this way we avoid overflow. It is important to note that all multiplication is done in one clock cycle.

The filter multiplication operation results are shifted left by the difference between the maximum number of multiplication MAX_MULT and the number of multiplications corresponding to the coefficient $num\_mult(D)$, where D ranges from 0 to $(FILTER\_TAP-1)/2$. The result of this operation is stored in the shift register $s\_mult\_i(D)(INPUT\_WIDTH+COEFF\_WIDTH+MAX\_MULT\ downto\ 0)$. These shift registers are a two-dimension array that has a length of 63. Note that all shifting is done in one clock cycle.

The next operation the filter perform is the addition of the shifted result stored in the shifted register s_mult_i. To avoid timing violations, all results can not be added at the same clock cycle, therefore a tree structure for accumulating this result was deployed. The first process is summing four shifted registered elements at a time (i.e $s\_mult\_i(0)$ through to $s\_mult\_i(3)$) and stored in the result of this summing in the first result register $res\_add\_1\_i(E)$ register, where E ranges from 0 to 15. The next four shifted registered element is added and stored in $res\_add\_1\_i(E+1)$ and this cycle continues till all the shifted results are added. It is important to note that four results are added at the same time for all samples in the shifted register except for the last three results, in which only three are added. These first result addition registers ($res\_add\_1\_i$) are a two-dimension array that has a length of 16. The next operation in the tree structure is summing the element in the $res\_add\_1\_i$ register in the same way as the first operation and storing this result in the second results register $res\_add\_2\_i$. The second result registers are a two-dimension

array that has a length of 4.

The final operation in the tree structure is summing the four elements in the second result register and storing it in the final result register $res\_add\_3\_i$. The result obtained is then rounded using the round half-up algorithm and the least significant bits are truncated to produce output data of the filter which are placed on the $dout\_i$ and $dout\_q$ port. $dout_valid$ pulse is raised to notify that valid results are available.

4 RESULTS AND DISCUSSION

This section provides the results that are obtained in the different steps of this research work. It begins with presenting the results and is followed by an explanation of the results. It is concluded with a discussion of the thesis work.

4.1 Results

The first step in this research study was to compare the performance of a 124th-order low-pass Equiripple FIR design, in which its coefficient was quantized to three different bit-widths using the classical quantization method to that of the double precision coefficient. The performance of an FIR filter can be evaluated using its impulse response and magnitude response. The impulse response indicates the output of the filter when it is given a delta function or impulse as input. The response is a sequence of values that represent the filter's output signal over time. The behavior of the filter is determined by the filter coefficients, which also influence the magnitude response of the filter. The magnitude response, on the other hand, measures the amplitude of the filter's output signal as a function of frequency.

The first task was to quantize this double precision coefficient to 16 bits using the classical fixed point quantization method, which is just a case of rounding the coefficient to the desired bit-width. The magnitude response of the filter that has undergone quantization is then evaluated by comparing it to the magnitude response of the filter with double precision coefficients. Figure 4.1 shows the impulse response of the 16-bit quantized filter.

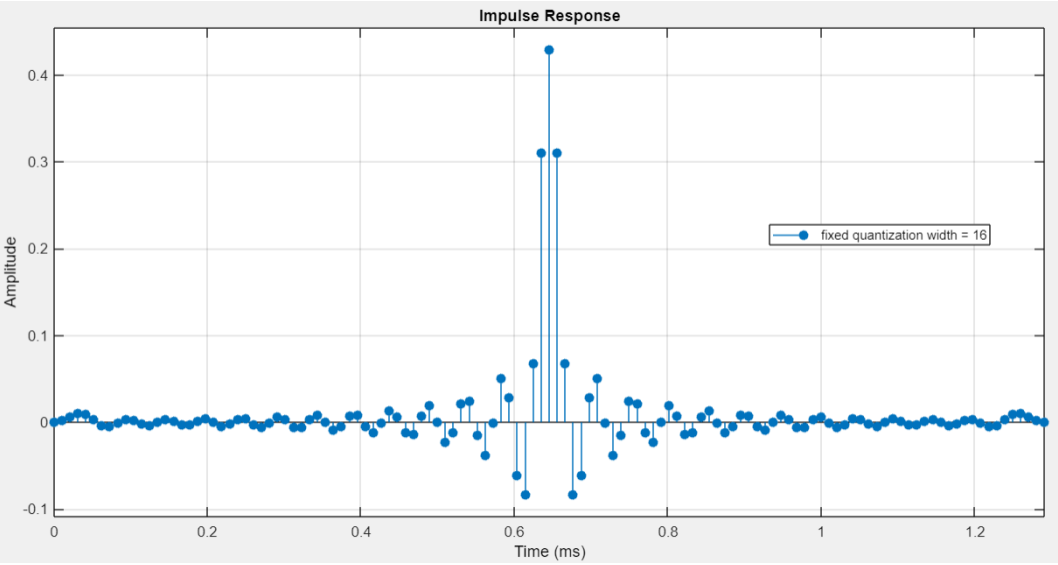

Figure 4.1. Impulse response 16-bits quantization

Figure 4.2 illustrates the magnitude responses of the filter after being quantized to a 16-bit representation.

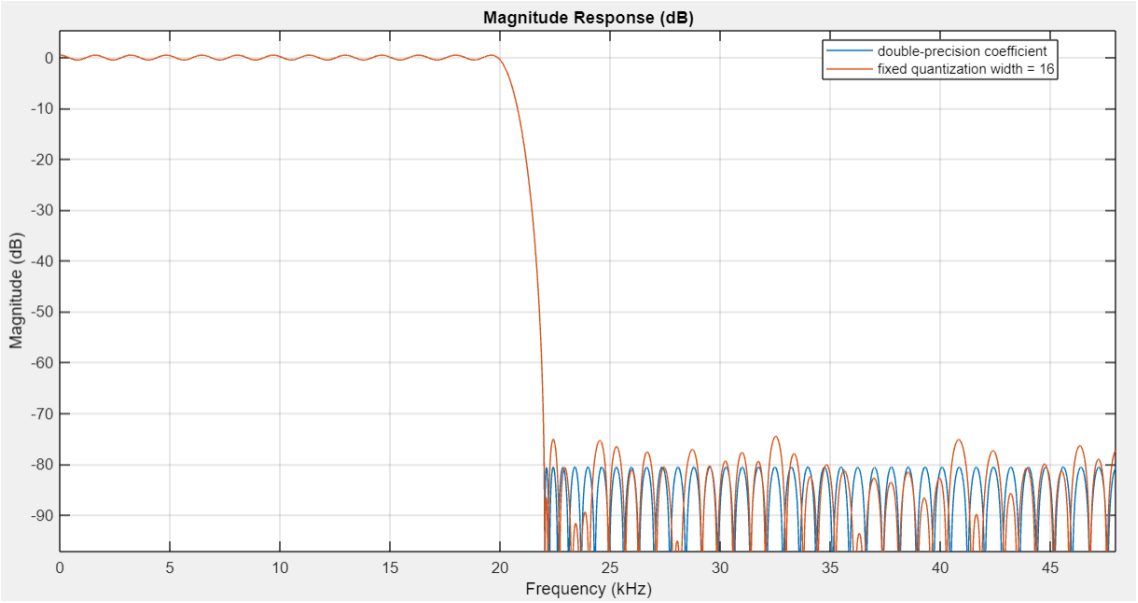

Figure 4.2. Magnitude response 16-bits quantization

From the magnitude response graph, the stop band attenuation for double precision is measured to be 80.6dB and that of the 16-bit quantized coefficient is 74.6 dB. This shows that the filter performance deteriorates due to quantization of the coefficient, which reduces the filter coefficient precision, hence resulting in a loss in performance. The filter coefficient was further quantized to a bit-width of 12 bits, and the impulse response of the quantized 12-bit filter is presented in Figure 4.3

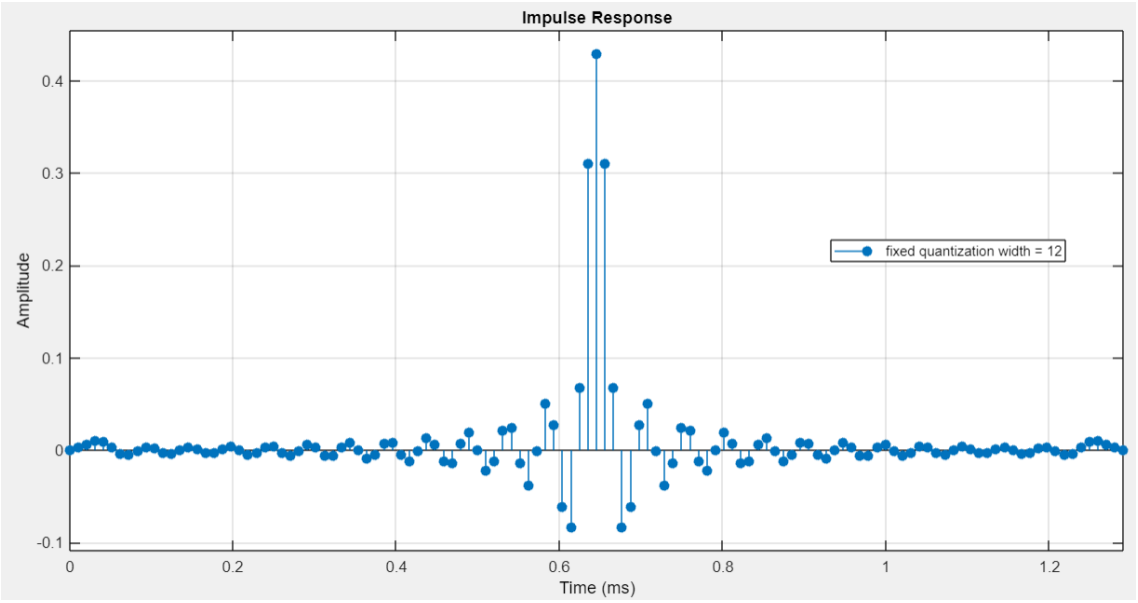

Figure 4.3. Impulse response 12-bits quantization

Figure 4.4 illustrates the magnitude responses of the filter after being quantized to a 12-bit representation.

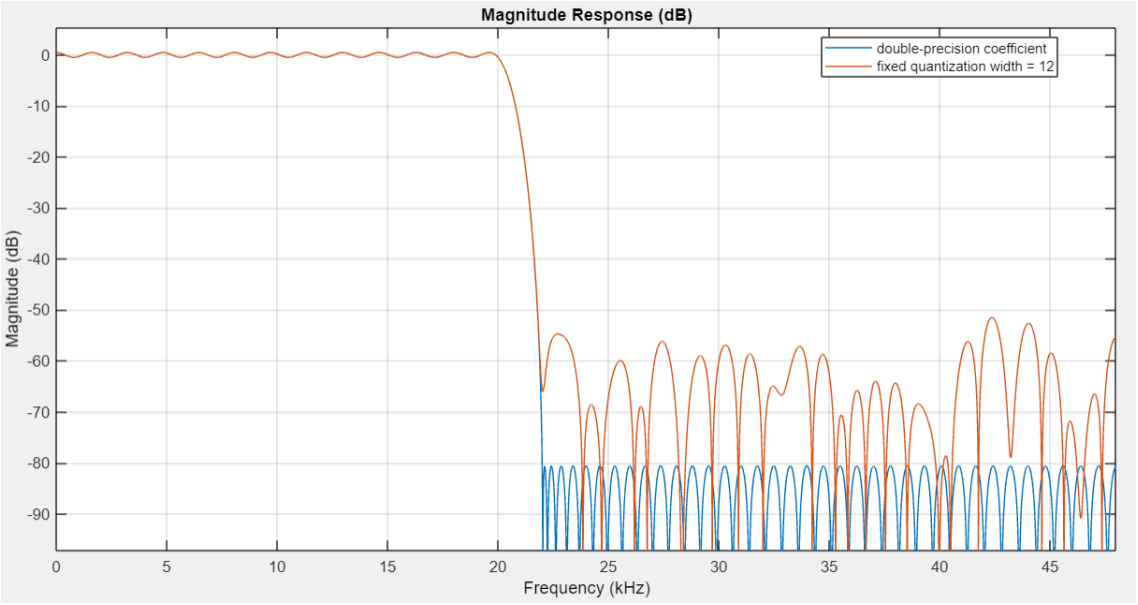

Figure 4.4. Magnitude response 12-bits quantization

The magnitude response graph shows that the stop band attenuation for the 12-bit quantized coefficient is 51.5dB. This filter produces a loss of about 29.1dB when compared to the double precision coefficient, which shows that the filter performance is dropping lower as the number of coefficient bits is decreasing. The filter coefficient was further quantized to a bit-width of 8 bits, and the impulse response and the impulse response of the quantized 12-bit filter are presented in Figure 4.5

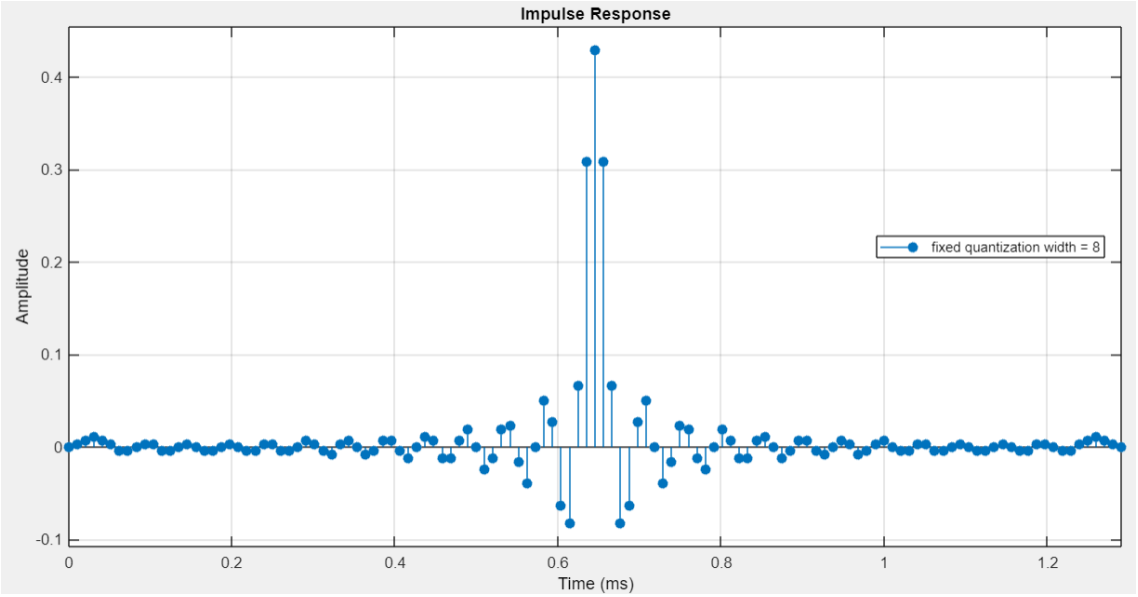

Figure 4.5. Impulse response 8-bits quantization

Figure 4.6 illustrates the magnitude responses of the filter after being quantized to a 16-bit representation.

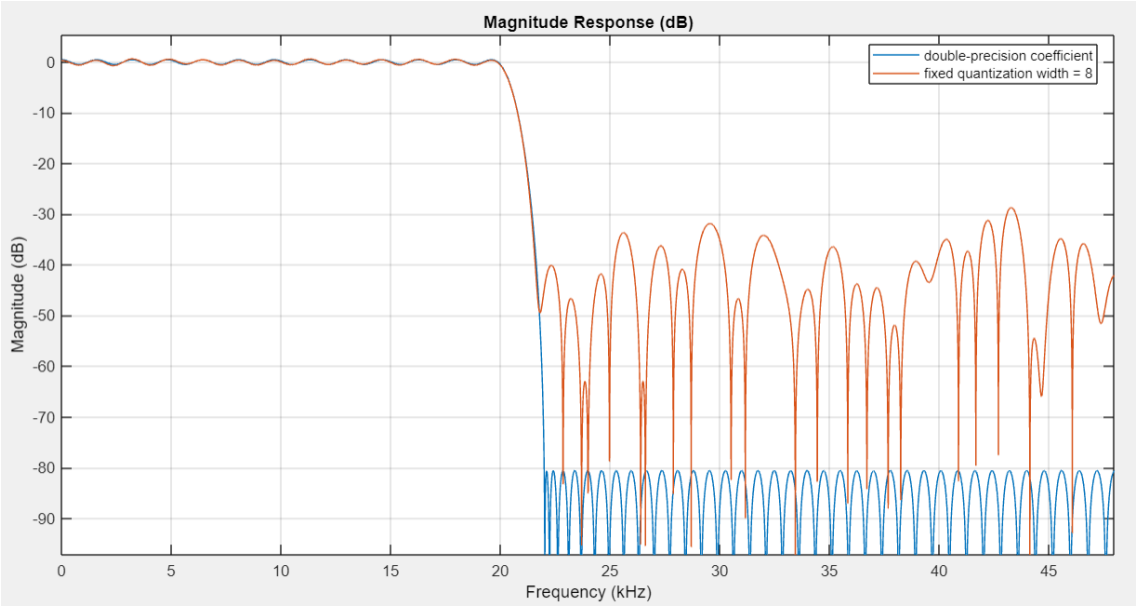

Figure 4.6. Magnitude response 8-bits quantization

From the magnitude response result, the stopband attenuation of the 8-bit quantized coefficient is 28.7dB. This indicates that more than half of the filter has been lost when quantized to 8 bits. A conclusion can be reached that as the filter coefficient bits-width increases the filter performance will further increase, however increasing the filter coefficient would result in more computation complexity and hardware resources that would be used to implement the filter design.

Another measure that can be used to evaluate the quality of the output generated by the different quantized filter coefficients is the quantization error. Quantization error refers to the difference between the filter's output when it is implemented using infinite-precision arithmetic, and its output when it is implemented using finite-precision arithmetic. Hence, making it an important characteristic in determining the performance of our filter. Table 4.1, presents the quantization error for both real and imaginary output data for the filter implemented using the classical quantization method.

Table 4.1. Classical quantization error

	Coefficient bit-widths		
	16	**12**	**8**
ΔH Fixed Coefficient (Real)	0.0001	0.0026	0.0380
ΔH Fixed Coefficient (Imag)	0.0002	0.0025	0.0399

Figure 4.7 displays a visual representation of the quantization error that arises from the use of the classical quantization technique for quantizing filter coefficients.

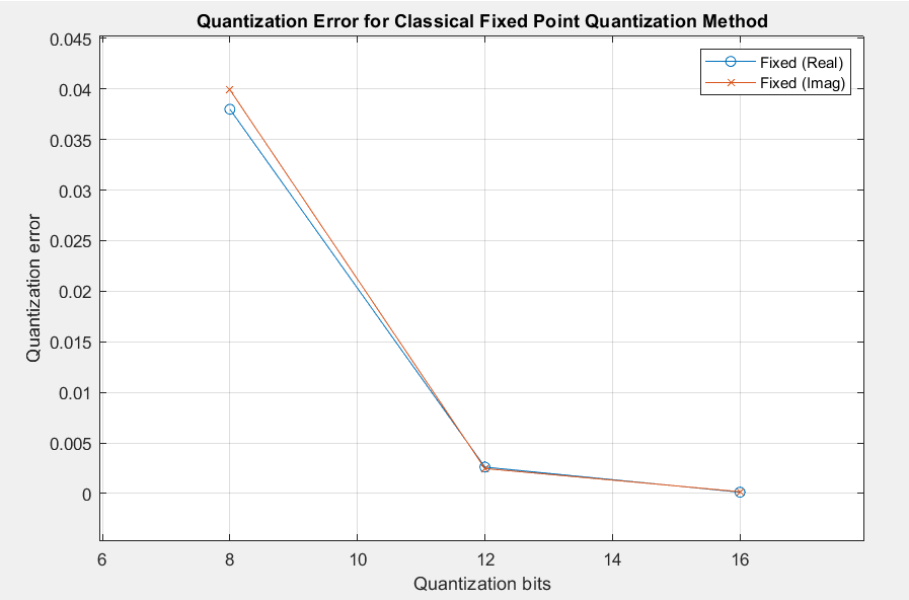

Figure 4.7. Classical quantization error

The quantization error result accounts for the difference between the double precision coefficient filter result and the fixed point design. Hence, the result presented further emphasis that using a greater number of bits for coefficient quantization reduces the functional error and enables us to match the double-precision performance. In other words, the quantization error can be reduced by using higher-precision arithmetic. However, this comes at the cost of increased computational complexity and reduced speed.

Now let us implement the proposed dynamic filter coefficient technique, and perform a comparative study on how the performance is compared to the double precision and the classical method of quantization. This comparative study quantized the filter coefficients using the same bit-width as the classical method. It begins with the quantization of the double-precision coefficient to a 16-bit dynamic quantization coefficient using the proposed algorithm. The filter's impulse response, after quantization to 16-bit, is illustrated in Figure 4.8

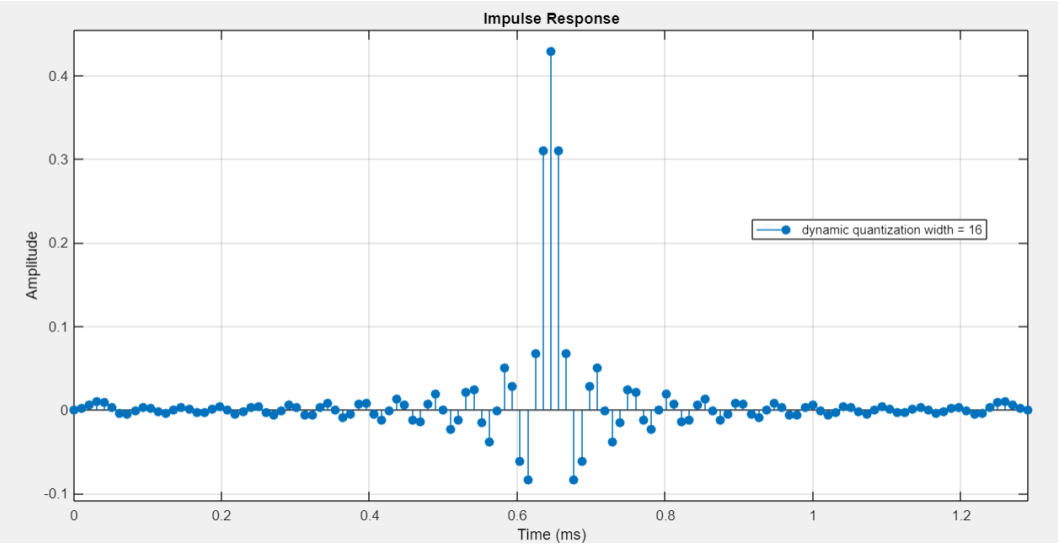

Figure 4.8. Impulse response 16-bit dynamic quantization

The magnitude responses of both double precision filter designs, 16-bit quantized filter design using the classical quantization method, and 16-bits dynamic filter coefficients are presented in Figure 4.9

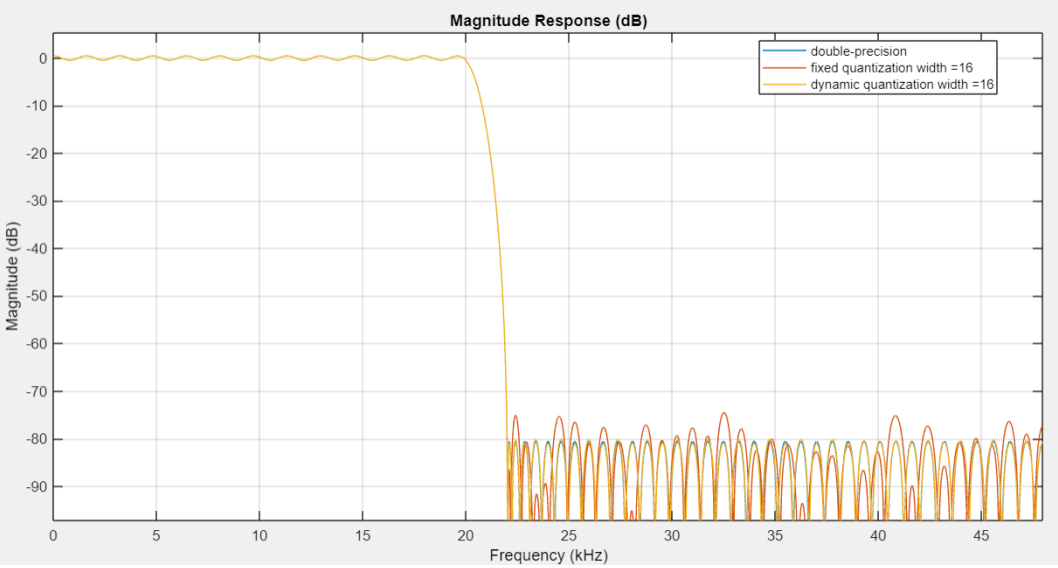

Figure 4.9. Magnitude response 16-bit dynamic quantization

The magnitude response result shows that the stop band attenuation for the 16-bit dynamic quantized coefficient is 80.1dB. Hence, there is an improvement in filter performance by 5.5dB when compared to the classical fixed point quantization method. However, there is an 0.5dB drop in performance when it is compared to the double-precision filter design, and this is expected due to the quantization of the coefficient, which generally results in the reduction of coefficient precision thereby dropping the performance of the filter. The double precision coefficient was further quantized to a bit-width of 12-bit using the dynamic quantization method, and the impulse response of the 12-bit quantized filter is shown in Figure 4.10

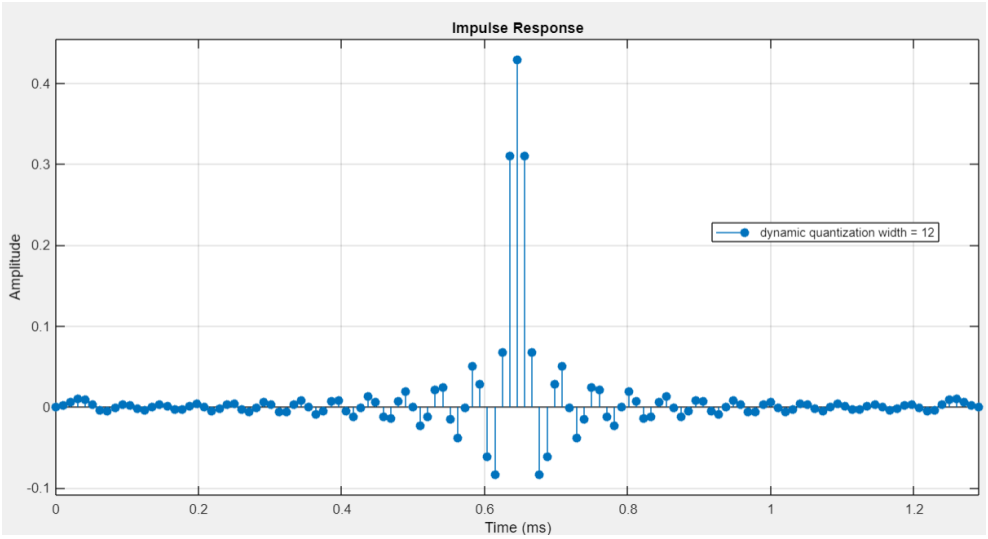

Figure 4.10. Impulse response 12-bit dynamic quantization

The magnitude responses of both double precision filter designs, 12-bit quantized filter design using the classical quantization method, and 12-bits dynamic filter coefficients are presented in Figure 4.11

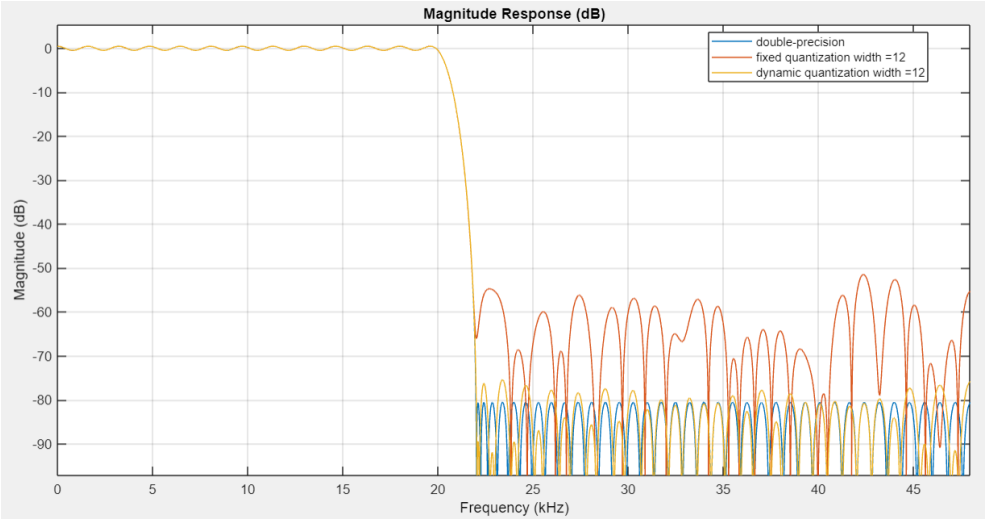

Figure 4.11. Magnitude response 12-bit dynamic quantization

According to the magnitude response graph, the stopband attenuation for the 12-bit dynamic quantized coefficient is measured at 75.8 dB. When compared to the 12-bit classical quantization method, this result represents a significant improvement in filter performance by 24.3 dB and a slight improvement by 1.2 dB compared to the 16-bit classical fixed point quantization method. However, there is a decrease in performance compared to the double precision coefficient, with a drop of 4.8 dB. To further analyze the filter performance, the double precision coefficient was quantized using the dynamic method to a bit-width of 8 bits, and the resulting impulse response is shown in Figure 4.12.

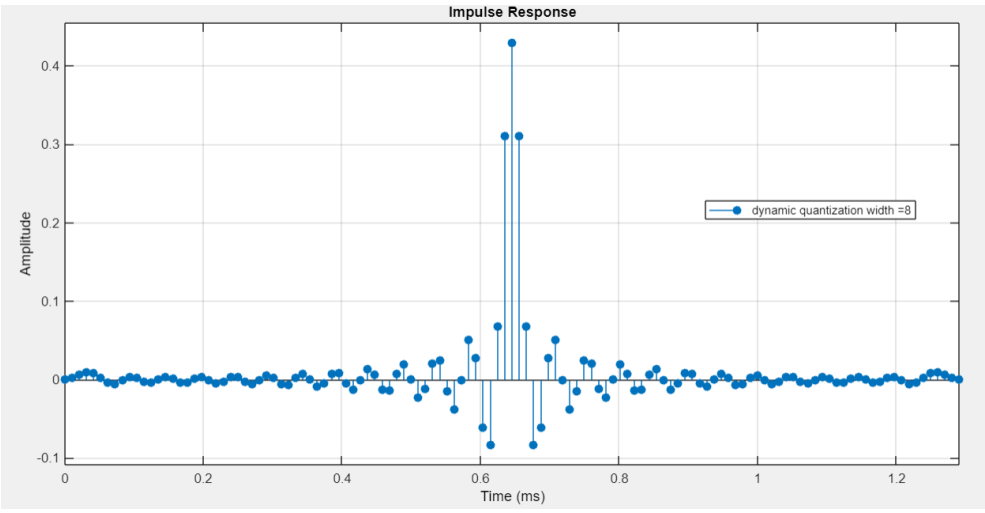

Figure 4.12. Impulse response 8-bit dynamic quantization

The magnitude responses of both double precision filter designs, 8-bit quantized filter design using the classical quantization method, and 8-bit dynamic filter coefficients are presented in Figure 4.13

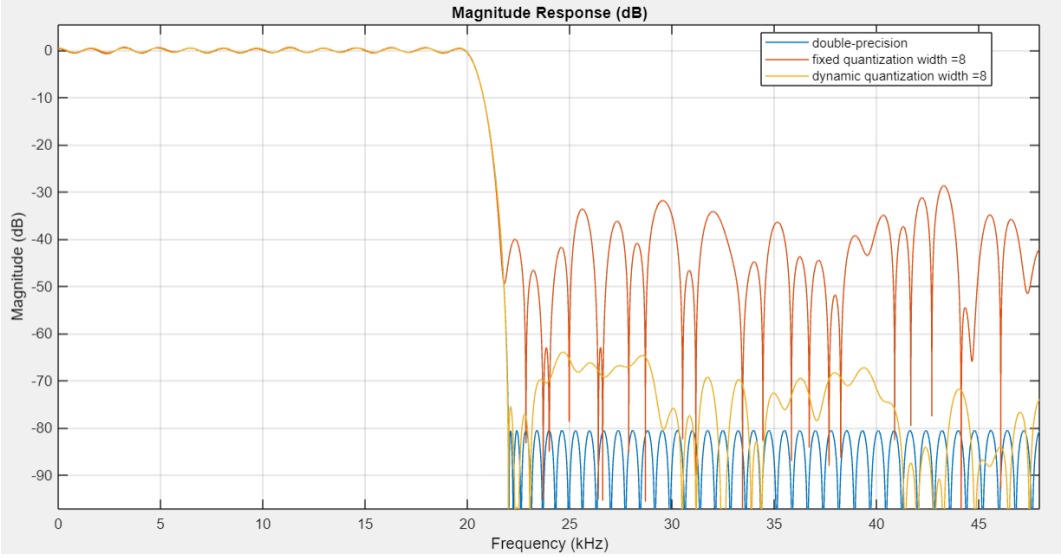

Figure 4.13. Magnitude response 8-bit dynamic quantization

As depicted in Figure 4.13, the magnitude response for the 8-bit dynamic quantized coefficient exhibits a stopband attenuation of 64 dB, representing a 35.3 dB and 12.5 dB improvement in filter performance compared to the 8-bit and 12-bit classical fixed point quantization, respectively. However, It should be noted that the 8-bit dynamic quantized coefficient shows a slight decline in performance compared to the 16-bit fixed point coefficient, with a drop of about 10.6 dB. In order to gain a better understanding of the advantages and disadvantages of each quantization method, it is important to compare the quality of the filter output generated by the various dynamic quantization bit-widths with the classical quantization method. This comparison is provided in Table 4.2. This information can help inform the decision-making process when choosing the appropriate quantization method for a specific application. Overall, the results suggest that the dynamic quantization method may be a more effective approach for improving filter performance, particularly at lower bit-widths, compared to the classical fixed point method.

Table 4.2. Classical and dynamic quantization error

	Coefficient bit-widths		
	16	**12**	**8**
ΔH Fixed Coefficient (Real)	0.0001	0.0026	0.0380
ΔH Fixed Coefficient (Imag)	0.0002	0.0025	0.0399
ΔH Dynamic Coefficient (Real)	0.0001	0.0002	0.0023
ΔH Dynamic Coefficient (Imag)	0.0002	0.0002	0.0021

The graphical representation of the real and imaginary output data quantization error for both dynamic and classical fixed point quantization is shown in Figures 4.14 and 4.15 respectively.

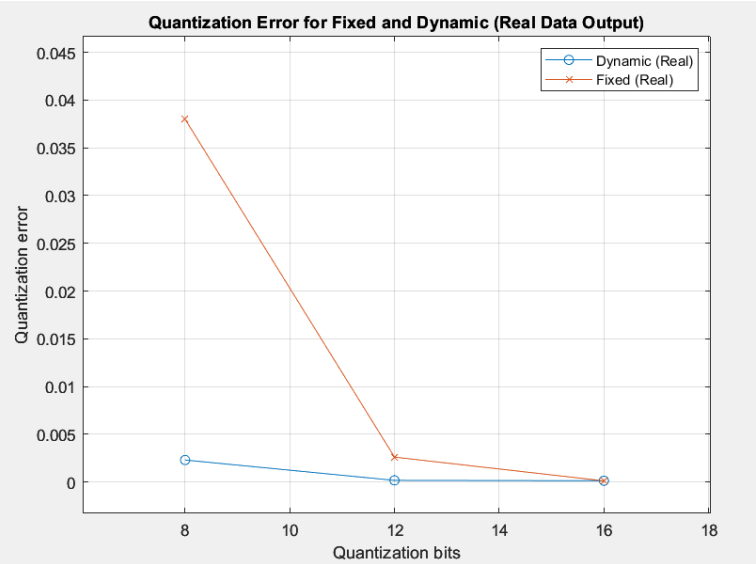

Figure 4.14. Classical and dynamic quantization error (real)

According to the data in Table 4.2 and the graphical representations in Figures 4.14 and 4.15, the classical quantization method leads to a sharp increase in quantization error when the FIR filter coefficient is quantized to 12 bits, while the proposed dynamic quantization method only exhibits a slight increase in error from 16 bits to 8 bits. At the 16-bit level, both methods produce similar quantization error, but at 12 bits the dynamic method greatly reduces the error by 92% compared to the classical method. Additionally, the dynamic method significantly reduces the quantization error when the filter coefficient is quantized to 8 bits. These results suggest that using the dynamic quantization method improves filter performance compared to the classical fixed point method.

The next step in the analysis was to compare the hardware area consumption of the dynamic and classical quantization filter designs. To do this, the functional requirements of the design were implemented using the Xilinx Vivado platform. The implementaion place and route simulation results provided information about the hardware resources required for each filter implementation, which are presented in 4.3.

Table 4.3. Hardware resources for fixed and dynamic filter design

Slice	Fixed coefficient widths			Dynamic coefficient widths		
	16	**12**	**8**	**16**	**12**	**8**
LUT	14979	9441	5459	21382	17587	10601
Register	10837	10083	8777	11833	11175	10491

The look-up table (LUT) shown in this implementation result can be described as a memory bank that stores a set of pre-computed values. These values are typically output by the LUT in response to a set of inputs. A slice LUT is a specific type of LUT that is implemented using a slice, which is a small building block of digital logic that typically contains several logic gates and flip-flops. On the other hand, a slice register is a type of register that is implemented using one or more slices of digital logic. It can be described as a device that stores a set of binary values and allows for the retrieval, modification, and transfer of these values.

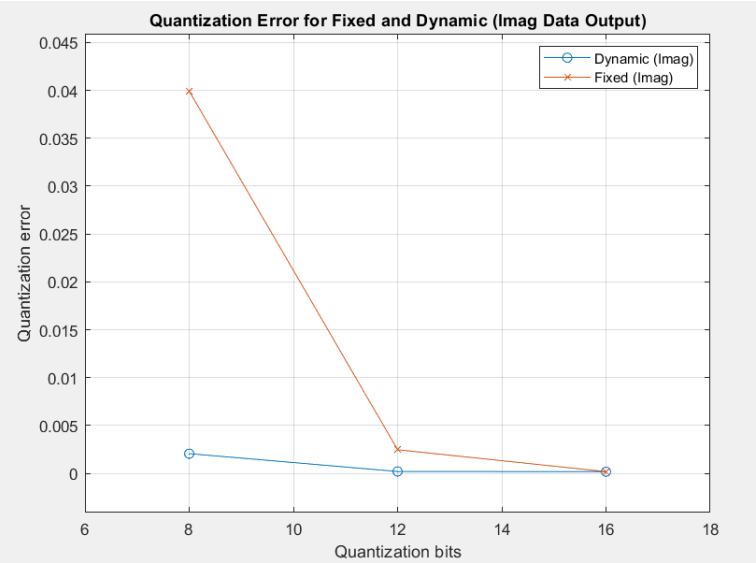

Figure 4.15. Classical and dynamic quantization error (imaginary)

A slice register is a register that is implemented using slices of digital logic. This allows for the efficient implementation of registers in digital circuits, as the use of slices allows for the modular construction of larger digital circuits. Hence, hardware area consumption is determined by the number of logic and other components that are used in its implementation. Slice LUTs and slice registered are designed to be implemented using a small number of logic gates and other components which makes them relatively efficient in terms of hardware area consumption. Therefore, the area consumption of the hardware is determined by the number of slice LUT and registered that the hardware is used for its implementation.

It can be seen from Table 4.3 that the dynamic coefficient consumes more hardware when compared to the classical fixed point quantization method. The reason for this is because the process of shifting the filter output left to recover the original precision of the coefficients which is done in the dynamic quantization filter design requires additional hardware resources, such as additional registers to store the intermediate values and additional logic to perform the shift operation. Hence, leading to this process consuming more hardware resources than the classical quantization method of simply rounding the coefficients, due to the additional operations and intermediate values involved. However, the dynamic quantization method which can be described as shifting the coefficients right and then shifting the filter output results left to recover the precision of the coefficients introduces a more improved numerical accuracy which leads to the reduction in the error introduced by the quantization process compared to classical method. This is augment in the quantization error results provided in Figure 4.14, 4.15 and 4.2. In which we can see that the dynamic quantization method provide a significant improvement in the quantization error.

From the implementation results presented in Table 4.3, the filter design that uses the least amount of area resource is the 8-bit classical fixed point quantization but this filter also produces the worst filter magnitude response and quantization error, thereby not making it an ideal choice for implementation purposes. Let us take a look at the classical 12-bit classical quantization, this uses a few hardware resources when compared to the rest of the filter design. However, the magnitude response produced by this filter and the quantization error does not make this a good trade-off, as we would be sacrificing a lot of performance for a reduction in hardware resources.

In the case of the 16 bits classical quantization, the filter performance of the 16-bit classical fixed point quantization method gives an acceptable performance but this comes at a cost of

using a lot of hardware resources. From this table, we can implement our filter using the 8-bit dynamic quantization method, with this method we get a reduced hardware resource when compared to the 16-bit classical fixed point quantization method, but at the cost of our filter performing slightly worst filter performance which we can consider as an acceptable trade-off because the dynamic quantization gives us the opportunity to use smaller bits-width without compromising too much of the filter performance.

4.2 Discussion

In this research, we studied how the quantization of the coefficients in an FIR filter impacts its performance and hardware requirements. These coefficients, which determine the behavior of the filter, were found to have a direct effect on both the filter's magnitude response and the hardware resources needed for implementation. A crucial element in evaluating the performance of a filter is the magnitude response, which indicates the amplitude of the output signal at various frequencies. Meanwhile, the precision of the coefficients influences the amount of storage and computational resources required for the filter.

We can examine the coefficient of the double-precision FIR filter, which is represented by floating point numbers with double precision. This means that the coefficient is represented using a binary number with a fractional part and an exponent, and it has a high degree of precision. This results in achieving high accuracy in the magnitude response, as the high precision of the coefficients allows for more accurate calculations. Using double-precision coefficients can significantly increase the storage and computational requirements of a filter as they are usually represented with a large number of bits. This, in turn, can result in a larger hardware area required for the filter. Therefore, there is a need to quantize this coefficient, to reduce the hardware resources we use, which saves us computation complexity and cost.

A quantized coefficient, on the other hand, is a coefficient that has been discretized into a set of specific values, rather than being represented as floating point numbers. This means that they are represented using a limited number of bits, which reduces their precision compared to double precision coefficients. This quantization can affect the filter's magnitude response, as it can introduce errors and cause the response to deviate from the desired behavior, as demonstrated in the results section where we show a drop in performance when the coefficient is quantized to different bit widths. On the other hand, quantized coefficients can also help to reduce the hardware resources needed for implementation, as they can be represented using fewer bits and thus require less storage and computational power.

Achieving higher performance in a filter often comes at the expense of increased hardware resources. More complex coefficients and structures are typically required for higher performance, which can lead to an increase in the hardware area necessary for the filter. Thus, there is often a tradeoff between hardware resources for a filter and performance. On the other hand, simpler coefficients and structures can help to reduce the hardware area consumption, but may also reduce the filter's performance. To manage this tradeoff, it is important to carefully select the coefficients and structures used in the filter's implementation. For example, using higher precision coefficients may improve performance but also increase hardware area consumption, while using simpler structures can reduce hardware requirements but also lower performance. The optimal equilibrium between performance and hardware utilization will vary based on the particular needs of the application. This tradeoff is illustrated in the implementation results shown in Table 4.3, where we see that reducing the coefficient bit width leads to a reduction in hardware resources required.

The way we quantize this coefficient is the central point of this thesis work. From the result

section, we show the classical quantization method. This method is typically done by using a quantization step size, which is the difference between the adjacent quantized levels. This quantization step size determines the resolution of the quantized coefficients and therefore has a direct impact on the performance. To elaborate, the classical quantization method rounds the coefficients to the nearest quantized level based on the number of bits allocated to represent them. Analyzing the classical quantization method coefficient, we can see that they contain a lot of wasted bits that have no weight in the calculation, therefore getting rid of these wasted bits and replacing them with more significant coefficients bits was the priority. From the result obtained from this method of quantization, we see that as we reduced the number of bits representing the coefficients, the performance of the filter begins to reduce significantly. Hence, there is a need to find a way to improve this performance. Hence, a different method for quantizing this coefficient was presented to achieve improved performance across different bit widths and also without wasted bits.

In this proposed method, we use dynamic quantization to optimize the filter coefficients. By taking advantage of the dynamic range of the floating point coefficients, we are able to eliminate wasted bits, resulting in improved performance compared to the classical quantization method. Additionally, we analyzed the hardware area consumption for both methods. However, before we can determine the hardware resources required for each filter, we must first design the filter according to its functional requirements, as outlined in the previous chapter. To understand how data flows through the filter and ensure its functional correctness, we conduct a behavioral simulation of the dynamic quantization method. The simulation results are presented in Figure. 4.16

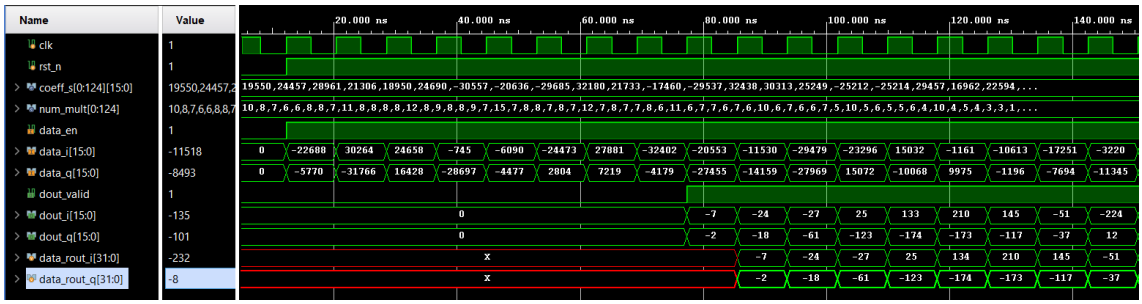

Figure 4.16. Behavioural simulation of dynamic quantization method

From the behavior simulation, it can be seen that the filter works as described in the functional requirement in the previous section. The signal *data_rout_i* and *data_rout_q* are the output results generated from MATLAB compared to the results obtained from the behavioral simulation. When the *dout_valid* pulse is raised, the output results are read from MATLAB on the next clock cycle, and then compared to the previous result of the filter output which is the first result. From the behavior simulation, we can see that the result correlates with the result generated from MATLAB. However, in some cases for the result obtained, we have a little difference due to rounding in the hardware implementation. The internal signal of the filter is presented in Figure 4.17 and Figure 4.18.

Here we can see a more detailed data flow. From the results, when the *data_en* pulse is raised, the input data is fed into the filter, the data flow through the register as specified in the functional description, and the *dout_valid* pulse is raised immediately we have a filtered output data is available at which point the testbench starts the comparison analysis between the output generated on MATLAB and RTL simulation output. The behavior simulation also shows that

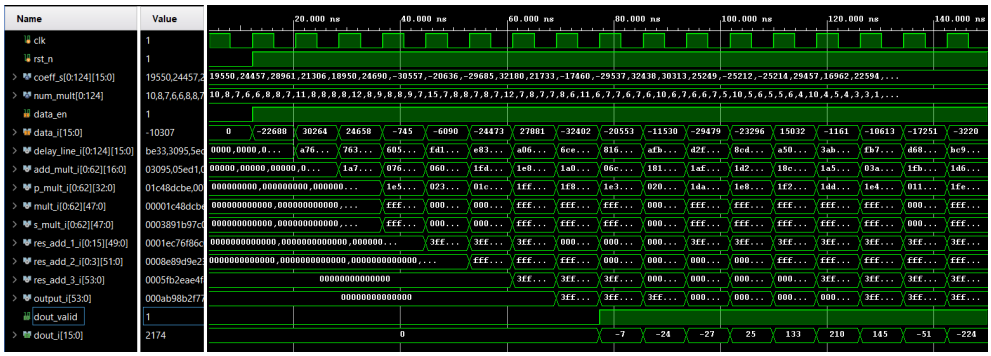

Figure 4.17. Behavioural simulation with internal signal (real)

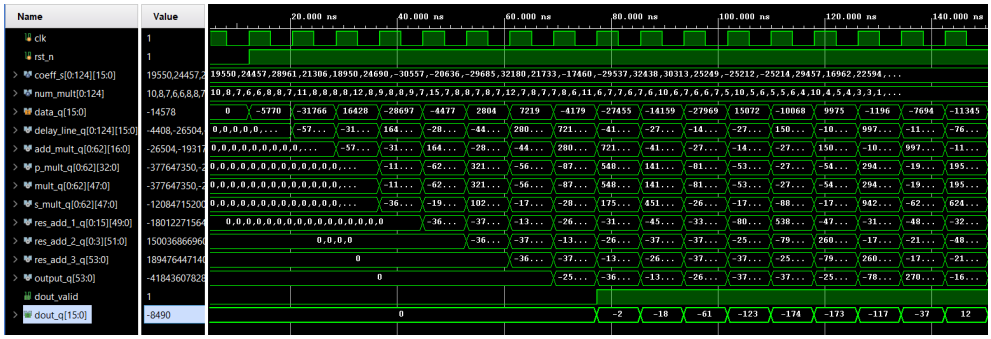

Figure 4.18. Behavioural Simulation of Dynamic Quantization Method (imaginary)

the design is a single rate filter in which the input and output sample rates are the same. This means that it operates on the input signal at the same rate at which it produces the output signal. In the case of the classical quantization method RTL implementation, the idea is the same as the dynamic quantization RTL implementation except there is no shift register in the classical approach. The internal signal of this RTL implementation is shown in Figure 4.19 and Figure 4.20.

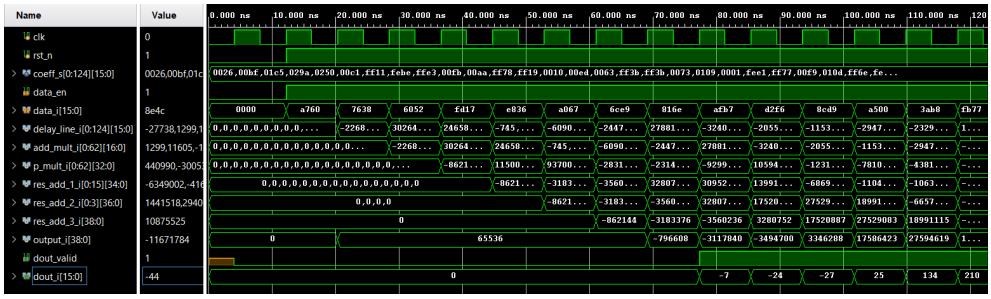

Figure 4.19. Behavioural simulation of classical quantization filter design (real)

From the result, it can be seen that the shift register *s_mult_i* and *s_mult_q* are no longer used in the classical method and the scaling factor is not mapped into the filter design. When comparing the result generated from MATLAB and the result obtained from the RTL simulation, all results match and correlate. This shows that our implementation of the design was accurate.

The next analysis that was performed, was to examine the hardware area consumption of

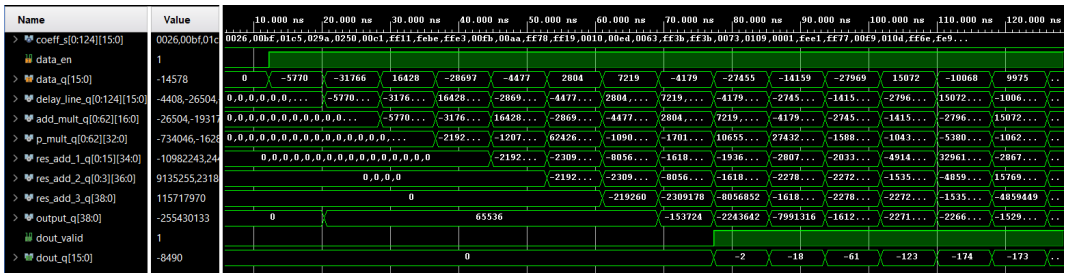

Figure 4.20. Behavioural simulation of classical quantization filter design (imaginary)

the dynamic quantization and classical quantization design. This was done by synthesizing the design that was implemented and the result of this implementation is presented in Table 4.3. From the result obtained above, it can be seen that dynamic coefficient quantization requires more hardware resources than classical fixed point quantization because it involves shifting the filter output to recover the original precision of the coefficients, which requires additional registers to store intermediate values and logic to perform the shifting operations. This additional processing consumes more hardware resources than simply rounding the coefficients, as used in the classical quantization method.

However, it is important to carefully consider the trade-off between performance and hardware resources. Because implementing a filter with high performance typically requires more hardware resources, such as a larger number of logic gates or flip-flops, or a more complex and power-hungry digital circuit. This can be a significant limitation in many applications, such as mobile devices or embedded systems, where power consumption and area constraints are important considerations. On the other hand, implementing with fewer hardware resources can result in lower performance, such as reduced accuracy or slower speed. In the case of FIR filters, achieving high performance requires a large number of arithmetic operations and memory accesses, making the tradeoff between performance and hardware resources is particularly significant. Hence, it is crucial to carefully consider this tradeoff when designing a filter and choose a design that fulfills the application's requirements while minimizing the hardware resources necessary for implementation.

Hence, when analyzing the results obtained from this work, it was shown that the 8-bit classical fixed point quantization method uses the least amount of hardware resources, but produces the worst magnitude response and quantization error. The 12-bit classical quantization method uses fewer hardware resources than the other methods, but its magnitude response and quantization error make it a poor choice. The 16-bit classical quantization method has acceptable performance, but at the cost of using a large number of hardware resources. The 8-bit dynamic quantization method uses fewer hardware resources than the 16-bit classical method but has slightly worse filter performance. However, its performance is significantly better when compared to the 8 and 12-bit classical methods, thereby this dynamic quantization method gives us an acceptable trade-off between performance and hardware resources used because it gives us the opportunity to use smaller bits-widths without compromising the filter performance too much when compared to the classical method of quantization.

5 SUMMARY

This thesis focuses on the Equiripple FIR filter, in which the performance of this filter was studied after quantization of the filter coefficient, and a method for improving this performance was introduced. Additionally, this study investigates the impact of quantization on the hardware resources necessary for implementing the filter. The research began by analyzing the conventional method of quantizing filter coefficients. This method is typically done by using a quantization step size and this quantization step size determines the resolution of the quantized coefficients. In other words, for the classical quantization method, the filter coefficients are rounded to the nearest quantized level. The study evaluated the filter's performance by examining its magnitude response and found that increasing the number of bits used to represent the coefficient is necessary to improve filter performance when using the classical quantization method. However, this means that the computation complexity would increase and more hardware would be required to achieve a good performance. This theory is augmented with the implementation place and route result obtained from synthesizing the classical quantization filter design, which shows an increase in hardware resources when the coefficient bit-width was gradually increased from 8 bits to 16 bits.

Therefore, there is a need to improve on the classical method of quantization. The dynamic quantization method of quantization was built on the idea of improving the classical method. In the classical method, from the analysis of the filter coefficients, it can be seen that the high-order bits are the same as the sign bits for the low amplitude coefficient. These bits that are the same have no weight in the computation of the filter results, thus replacing these bits with more significant bits helped us in improving the precision of the filter coefficients. A study was carried out to compare the performance of the proposed method to that of the classical method of quantization. The results indicate that the proposed method performed considerably better with a reduced number of bits, while its performance is comparable to that of the classical method with a higher number of bits.

Furthermore, when we compared the implementation result we see that the dynamic method uses more hardware resources when compared to the classical quantization method. This is due to the process of shifting the filter output left to recover the original precision of the coefficients which is done in the dynamic quantization filter design which requires this method to need additional hardware resources, such as additional registers to store the intermediate values and additional logic to perform the shift operation. However, dynamic quantization gives us the opportunity to use smaller bits-width without compromising too much of the filter performance. Hence, we would save a significant amount of hardware resources by using smaller bit widths and not lose too much filter performance thereby making it an acceptable trade-off. Therefore, based on the results and analysis done on the dynamic quantization of filter coefficients, we conclude that dynamic quantization provides us with an improved filter performance.

Future work can be done to improve this dynamic quantization method by exploring the shifting operation of the dynamic quantization algorithm. One way to implement the shift operation efficiently is to use a pipeline design, in which the operation is divided into stages that can be executed in parallel. This can allow the filter to be implemented more efficiently by overlapping the execution of different stages, potentially reducing the overall hardware resource consumption.

One potential way for further improvement in this area of research is to examine the effects of limiting the maximum shift value when multiplying the filter coefficients by 2 and quantizing the result to the desired bit-width in Matlab. This may have implications for the performance and hardware requirements of the filter design. For example, limiting the maximum shift too aggressively, that is the shift value is limited to a relatively small value, the filter coefficients may be less precise, which could potentially degrade the performance of the filter. Reducing

the number of bits needed to represent filter coefficients may lead to a less hardware-intensive implementation of the filter, as it reduces the amount of additional logic required for shift operations and the number of registers needed to store intermediate values. This could result in a more efficient implementation of the filter.

In contrast, if the shift value is not limited aggressively (i.e., it is allowed to be larger), resulting in a more precise representation of the filter coefficients. This can potentially improve the performance of the filter, as a more precise representation of the coefficients can result in a more accurate filter. However, this can also increase the hardware resource consumption of the filter, as more bits may be needed to represent the filter coefficients, the number of additional logic needed to perform shift operations would increase, and the needed register to store intermediate values would be reduced thereby potentially resulting in a less efficient implementation of the filter. Hence, future studies on the optimization of this shift value can explore a way to improve this dynamic quantization method. The main idea is to experiment with different shift values and evaluate the resulting performance and resource consumption of the filter with respect to design goals.

In addition to that, for hardware area consumption we could also explore the idea of reducing the filter coefficient's most significant bits and see how it affects the number of resources consumed and the filter performance. The use of a reduced number of bits to represent filter coefficients may potentially decrease the hardware resources necessary for filter design. This is because using fewer bits generally results in a simpler and more efficient implementation, as there are fewer bits to manipulate and store. On the other hand, decreasing the number of significant bits in filter coefficients may have an adverse effect on the filter's performance. The coefficients reflect the contribution of each tap in the filter to its overall response, and a more accurate representation of these coefficients can lead to better filter accuracy. The reduction in the number of bits used to represent the filter coefficients may result in a loss of precision, which in turn may cause a decline in the filter's performance. This trade-off between hardware resource consumption and filter performance is something that needs to be carefully studied when designing a filter.

6 BIBLIOGRAPHY

[1] Singh T.C. & Kumar M. (2021) Digital FIR Filter Designs. In: *Asian Conference on Innovation in Technology, ASIANCON*, Institute of Electrical and Electronics Engineers Inc.

[2] Yang Z., Ma S., Hu S., Zheng K., Cao X. & Chen J. (2018) Design of FIR Filter Based on Algebraic Integer Quantization. Technical report.

[3] Bugrov V., Fitasov E., Sataev V. & Kudryashova O. (2022) Dynamic Coefficient Quantization Digital FIR Filter. In: *24th International Conference on Digital Signal Processing and its Applications, DSPA*, Institute of Electrical and Electronics Engineers Inc.

[4] Izydorczyk J. (2006) An algorithm for optimal terms allocation for fixed point coefficients of FIR filter. In: *Proceedings - IEEE International Symposium on Circuits and Systems*, pp. 609–612.

[5] Zeinolabedin S.M.A. & Karimi N. (2012) A new quantization algorithm for FIR filters coefficients. In: *ICEE - 20th Iranian Conference on Electrical Engineering*, pp. 1120–1124.

[6] Agarwal D., Reddy K.S. & Sahoo S.K. (2013) FIR filter design approach for reduced hardware with order optimization and coefficient quantization. In: *International Conference on Intelligent Systems and Signal Processing, ISSP*, pp. 293–296.

[7] Najim M. (2006), Digital Filters Design for Signal and Image Processing.

[8] Yu J., Zhong H. & Yan P. (2020) Implementation of FIR filter based on Xilinx IP core. In: *Proceedings - 2nd International Conference on Information Technology and Computer Application, ITCA*, Institute of Electrical and Electronics Engineers Inc., pp. 79–85.

[9] Bhalke S., Manjula B.M. & Sharma C. (2013) FPGA implementation of efficient FIR Filter with quantized fixedpoint coefficients. In: *Proceedings - International Conference on Emerging Trends in Communication, Control, Signal Processing and Computing Applications, IEEE-C2SPCA*, IEEE Computer Society.

[10] Izydorczyk J. (2007) Coding FIR Filter Coefficients in Power-Of-Two Format. Technical report.

[11] Stosic B.P. (2020) The Effects of Coefficient Quantization on New CIC FIR Filters. In: *55th International Scientific Conference on Information, Communication and Energy Systems and Technologies, ICEST - Proceedings*, Institute of Electrical and Electronics Engineers Inc., pp. 151–154.

[12] Vandenbussche J.J., Lee P. & Peuteman J. (2013) On the coefficient quantization of Multiplicative FIR filters. Digital Signal Processing: A Review Journal 23, pp. 689–700.

[13] Rajan A., Jamadagni H.S. & Rao A. (2009) Minimizing quantization effects in digital filtering. In: *IEEE 13th Digital Signal Processing Workshop and 5th IEEE Signal Processing Education Workshop, DSP/SPE, Proceedings*, pp. 501–506.

[14] Shen Z. (2010) Improving FIR filter coefficient precision. In: *IEEE Signal Processing Magazine*, Institute of Electrical and Electronics Engineers Inc., volume 27, pp. 120–124.

[15] Rowell A.S. (2012) Digital Filters with Quantized Coefficients: Optimization and Overflow and Using Extreme Value Theory. Technical report.

[16] The Ideal Lowpass Filter — Spectral Audio Signal Processing.

[17] Practical Introduction to Digital Filter Design - MATLAB & Simulink Example.

[18] Çiloglu T. & Ünver Z. (1994) Design of FIR filters with powers of two coefficients based on a new quantization quality criterion. In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, Institute of Electrical and Electronics Engineers Inc., volume 3, pp. III561–III564.

[19] Ciloglu T. & Unver Z. (1994) New gradient approximation method for the design of minmax FIR filters with powers of two coefficients. In: *Mediterranean Electrotechnical Conference - MELECON*, IEEE, volume 1.

[20] DeBrunner L.S. & Yunhua W. (2006) Optimizing filter order and coefficient length in the design of high performance FIR filters for high throughput FPGA implementations. In: *2006 IEEE 12th Digital Signal Processing Workshop and 4th IEEE Signal Processing Education Workshop*, pp. 608–612.

[21] Dimitar M. (2022), Part 2: Finite impulse response (FIR) filters - VHDLwhiz.

[22] Xilinx & Inc FIR Compiler v7.2 LogiCORE IP Product Guide. Technical report.

[23] Chen C & Alan N. Willson J. (1997) Higher Order Sigma-Delta Modulation Encoding for The Design of Efficient Multiplierless FIR Filter with Powers-of-Two Coefficients. IEEE.

[24] Dam H.H., Nordebo S., Teo K.L. & Cantoni A. (1998) Design of linear phase FIR filters with recursive structure and discrete coefficients. In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, Institute of Electrical and Electronics Engineers Inc., volume 3, pp. 1269–1272.

[25] Kotteri K.A., Bell A.E. & Carletta J.E. (2003) Quantized FIR Filter Design Using Compensating Zeros. IEEE Signal Processing Magazine 20, pp. 60–67.

[26] Mehboob R., Khan S.A. & Qamar R. (2009) FIR filter design methodology for hardware optimized implementation. IEEE Transactions on Consumer Electronics 55, pp. 1669–1673.

[27] Magar M. & DeBrunner L.S. (2004) Balancing the tradeoffs between coefficient quantization and internal quantization in FIR digital filters. In: *Conference Record - Asilomar Conference on Signals, Systems and Computers*, volume 1, pp. 493–497.

[28] Stosic B.P. (2021) Improved Classes of CIC Filter Functions: Design and Analysis of the Quantized-Coefficient Errors. In: *56th International Scientific Conference on Information, Communication and Energy Systems and Technologies, ICEST - Proceedings*, Institute of Electrical and Electronics Engineers Inc., pp. 65–68.

[29] Lin T.J., Yang T.H. & Jen C.W. (2003) Coefficient optimization for area-effective multiplierless FIR filters. In: *Proceedings - IEEE International Conference on Multimedia and Expo*, IEEE Computer Society, volume 1, pp. I125–I128.

[30] (2019) 754-2019 - IEEE Standard for Floating-Point Arithmetic .

[31] Goldberg D (1992) What Every Computer Scientist Should Know About Floating-Point Arithmetic. Technical report.

[32] Maxfield C (2006), Rounding Algorithms Compared - EE Times.