DESIGN AND IMPLEMENTATION OF A FINITE IMPULSE RESPONSE DIGITAL FILTER WITH ALTERNATIVE STRUCTURES

MOHAMED SYAHEER ALTAF 1917195

PROGRAMME: KOE

ELECTRONIC – COMPUTER INFORMATION ENGINEERING

FINAL YEAR PROJECT 2

SUPERVISOR: DATO' SERI DR. MASHKURI YAACOB

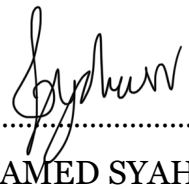
KULLIYYAH OF ENGINEERING

INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA

06/23

DECLARATION

I hereby declare that this is the result of my own investigations, except where otherwise stated. I also declare that it has not been previously or concurrently submitted for any other degrees at IIUM (International Islamic University Malaysia) or other institutions.



..... Date: 17/06/23

MOHAMED SYAHEER ALTAF

APPROVAL PAGE

I certify that I have supervised and read this study and that, it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as Final Year Project report as partial fulfilment for a degree of Bachelor of Engineering (Communication)(Honours).

.....

14/6/2023

[Dato' Seri Dr Mashkuri Yaacob]
Supervisor

Prof Emeritus Dato Seri Ir. Dr Mashkuri bin Haji Yaacob
Professor

Office Address:
Department of Electrical & Computer Engineering
Kulliyyah of Engineering
International Islamic University Malaysia
Jalan Gombak, Selangor

.....

[*Name*]
Examiner 1

.....

[*Name*]
Examiner 2

ABSTRACT

Why digital filter is important? As Manish B. Trimale & Chilveri (2017) stated, FIR filter is at the center of signal processing which is an incipient field that is active in various areas that include mobile telecommunications, biomedical applications and any implementation of digital signal processing. Therefore, it is crucial that we find the most optimized method of designing and implementing FIR filters. Suffice to say that there are several approaches when it comes to the architecture or structure of digital filters; each with their own advantages and disadvantages. The performances of the said structure can be measured by its computational complexity, filter characteristics, and quantization sensitivity. In this paper, we are going to attempt at designing and implementing some of the several known structures (i.e., direct form, cascade form, lattice structure, and symmetric FIR) and evaluate their performances in respect to the filter's characteristics and sensitivity to quantization. This paper concludes that lattice structure has overall performance and the least sensitive to quantization.

ACKNOWLEDGEMENTS

I, Mohamed Syaheer Altaf, would like to express the deepest appreciation to my supervisor, Dato' Seri Dr. Mashkuri Yaacob, for the help received, as well as the guidance and wisdom imparted to me throughout the consultation. I would also like to express my thankfulness to my colleagues, particularly Daniel bin Moktar for his relentless help and support. My gratitude is also given to the authors of the cited papers for their invaluable contributions to my report.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

For this chapter, we are going to look at the problem statement related to the design and implementation of FIR filters with alternative structures, as well as the objective and the scope of the report. Furthermore, an introductory methodology is discussed to inform in advance of the tools used in these experimental studies.

1.2 PROBLEM STATEMENT

FIR filter is an essential tool in digital signal processing as it is particularly useful in noise reduction; this allows a variety of applications ranging from communications to stock markets. There are various structures of FIR filters which aims to increase the filter's performance. This report aims to comprehensively study the performances of FIR structures such as direct form, cascade form, lattice structure, and symmetric FIR, in relation to their quantization sensitivity.

1.3 OBJECTIVE

1. To design FIR filters in MATLAB to obtain the filter coefficients based on the structure.
2. To implement FIR filters in C++ with various structures along with test signals with added Gaussian noise.
3. To comprehensively study the performance of the filter based on the filter's characteristics, and quantization sensitivity.

1.4 METHODOLOGY

The software used in this report will be of MATLAB and specifically the Signal Processing and Communication tools to obtain the filter coefficients. Furthermore, the implementation of the filter with various structure would be done in C++.

1.5 SCOPE

The scope of the report is as following:

1. A brief overview of FIR filter and its diverse structures such as direct form, cascade form, lattice form, and symmetric FIR as well as the methods of designing, and implementing the said filters.
2. A comprehensive procedure on designing and implementing FIR filter in MATLAB and C++.
3. Analysis and the evaluation of the performances of each structure and design in terms of filter's characteristics and quantization sensitivity.
4. A discussion on the advantages and disadvantages of the alternative structures.
5. The conclusion and future work on FIR filters with alternative structures.

1.6 REPORT ORGANIZATION

The layout of this report is starting with the title, that is, the design and implementation of FIR filters with alternative structures. This is followed by the abstract and the table of contents as well as the list of tables and figures. Furthermore, the report is then divided into 5 chapters: 1) the introduction, 2) literature review, 3) research methodology, 4) results and analysis and lastly, 5) the conclusion and future work. Finally, this report will include all the references used and appendices.

CHAPTER 2

LITERATURE REVIEW

2.1 OVERVIEW

Digital filter is an important aspect in Digital Signal Processing (DSP). The widespread usage of the Finite Impulse Response (FIR) filter in signal processing is noise reduction and signal recovery. This is so that the obtainable signal preserves its attributes without introducing artifacts. There are three commonly used techniques for designing FIR filters: Windowing, Frequency Sampling, and Filtering Optimization. Moreover, realizations of the filters can be done both in software and hardware.

To design an FIR filter, we have mentioned earlier that it serves as a noise reduction in our obtained signal; therefore, we are applying either 1) Lowpass filter, 2) Highpass filter, or 3) Bandpass filter. In the celebrated paper titled "Fundamentals of Digital Filter Theory" by Smith (1985), the mathematical description of finite filters is just the z- transform of difference equation. The standard description of such equation can be observed below:

$$Y(Z) = \sum_{k=0}^{N} b_k X(Z) Z^{-k} \qquad (2.1.1)$$

In this chapter, we are going to look at the current design methods of FIR filters in section 2.2.1 and the implementations of it in section 2.2.2. Moreover, section 2.2.3 will cover alternative structures of FIR filters. In addition, section 2.3 will cover the related works in studying the performance of FIR structures. Lastly, in section 2.4, we will summarize our findings of the current models and applications of FIR filters.

2.2 EXISTING WORKS

2.2.1 DESIGNING METHODS

Designing FIR filters can be of these three methods as mentioned: 1) Windowing, 2) Frequency Sampling, and 3) Filtering Optimization. We are going to briefly look at how the first two methods work as they are the most commonly used method of designing FIR filters.

2.2.1.1 WINDOWING METHOD

In windowing method, we are applying convolution of the sinc function and a window function. The simplest function for a window is a rectangular function; however, according to Molina (n.d.), smoother functions such as Hamming, Blackman, and Hanning can reduce the ripples caused by the side-lobes of the sinc function. In the same paper, the following artifact is the result of using a rectangular function as the window:

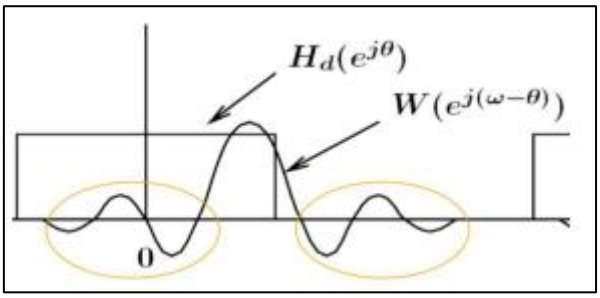


Figure 1. Ripple Effect in Rectangular Window

Although the use of smoother functions reduces this effect, they however increase the values of stop frequency (fs) and cut-off frequency (fc). Therefore, in designing an FIR filter using the windowing method, one must take into account of the said advantages and disadvantages of the window chosen. From Oppenheim & Schafer (1999), the parameters of commonly chosen windows are shown below (note that M is the number of points before truncated):

Types of Windows	Peak Side-Lobe Amplitude	Width of Main Lobe	Peak Error (dB)	Equivalent Kaiser Window, β	Transition Width
Rectangular	-13	4π/(M+1)	-21	0	1.81π/M
Bartlett	-25	8π/M	-25	1.33	2.37π/M
Hanning	-31	8π/M	-44	3.86	5.01π/M
Hamming	-41	8π/M	-53	4.86	6.27π/M
Blackman	-57	12π/M	-74	7.04	9.19π/M

Table 1. Types of Windows

2.2.1.2 FREQUENCY SAMPLING METHOD

This method evaluates the filter's frequency response at evenly spaced points within the range of 0 to 2π, resulting in an N-point Discrete Fourier Transform (DFT) of the filter. The filter coefficients can be calculated using the equation below:

$$h(n) = \frac{1}{N}\sum_{n=0}^{N-1} H(k)e^{j(2\pi n/N)k} \qquad (2.2.1)$$

2.2.2 IMPLEMENTATIONS

To put an FIR filter into practise, one must first design using the methods discussed, and then implement it with a software (such as MATLAB, or C programming language) or hardware (FPGAs or microprocessors). Typically, this would entail taking the input signal, multiplying it by the filter coefficients, and then adding the resulting products to get the filtered output. In order to produce the continuous filtered output, this process is typically repeated for each sample of the input signal. We will now look into the following implementations to get a better understanding of the material.

2.2.2.1 MATLAB

In MATLAB's Signal Processing and Communication tools, you are allowed to design an FIR filter by specifying the type of the filter (e.g., lowpass), the type of the window used (Hamming, Hanning, or Blackman), as well as the cut-off frequency, fc and the order of the filter (specified by n). Note that the higher the order of the filter, the more the complexity of the FIR filter which would be costly to materialize (Molina, n.d). For example, in the paper written by Adriana Borodzhieva (2022), they showed that you can simulate any FIR filters given the specified conditions.

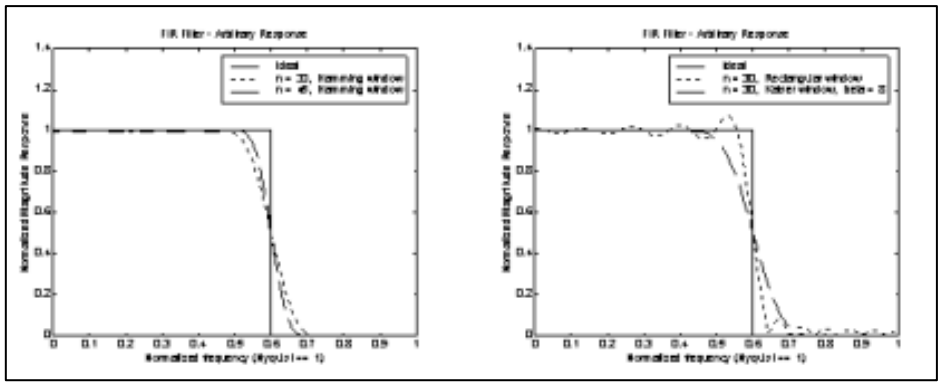


Figure 2. Implementing FIR in MATLAB

Furthermore, in official MATLAB documentation, you can also provide code to implement FIR filters by specifying their parameters. Figure below shows the example of the said implementation:

```
n = 50;
Wn = 0.4;
b = fir1(n,Wn);
```

Figure 3. Snippet of Code in MATLAB

In the example shown, the fir1 function is used to compute the filter coefficients, and then applies a smoothing window to the impulse response.

2.2.2.2 C PROGRAMMING LANGUAGE

An alternative method of implementing FIR filters is by using C programming language by utilizing the floating-point operations. As shown in the textbook *"Handbook of Digital Signal Processing"* by Douglas F. Elliot and Kuldip S. Rattan, the general steps of realizing the FIR filter are the following:

1. The coefficients of the samples are arranged in the input buffer.
2. The filter coefficients are arranged in a buffer.
3. Looping through an outer loop to compute the output buffer with the same size as the input.
4. Looping through inner loop that realized the structure of the FIR filter through means of multiplications and additions to achieve convolution.
5. Shift the previous input samples to process it again.

2.2.2.3 FPGA

To implement a FIR filter in an FPGA, you will need to use an FPGA design tool such as Xilinx Vivado or Altera Quartus. These tools allow you to create digital logic circuits that can be implemented on the FPGA. For instance, Zheng Jiayu & Wei Zhenhua (2018) showed that by using the MODELSIM tool and by invoking a Verilog HDL code, they have succeeded in simulating the desired FIR filter.

Moreover, Y.B. Liao et. al (2010) described how to apply hardware implementation to emulate the FPGA. An example of microprocessors used can be a low-end, low-spec processor, such as Intel Celeron, or you can also use microcontroller such as Arduino Uno to implement the hardware emulation of the FIR filter. The figure shows the outline of the implementation:

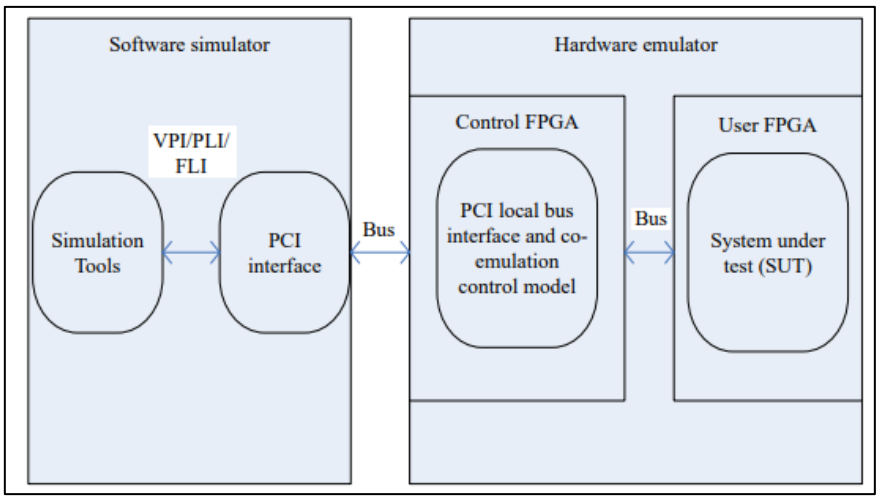


Figure 4. Hardware Emulator Logic

Implementation of an FIR filter will typically involve iterating over the input signal and utilizing the filter coefficients to the signal to produce the filtered output. The logic gate may look something similar to the figure below:

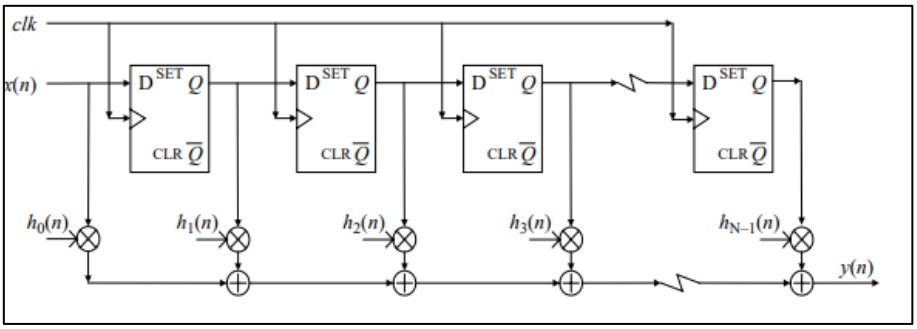


Figure 5. Implementing FIR using Processors

From the figure above, we will now describe how alternative structures are used to calculate the output of y(n) with different computational complexity as well as numerical precisions.

2.2.3 ALTERNATIVE STRUCTURES

Generally, the implementation of FIR filter is together with the structures used for the filters. The basic building blocks of FIR filter are multiplier, adders, and signal delay. Here multipliers should be fast enough so that overall throughput should not suffer. Adders are used in combination of multipliers and delays are used to store sample value in memory for one sample clock cycle (Manish B. Trimale & Chilveri, 2017). The following explanations of structures were taken from the textbook titled *Digital Filters using MATLAB* by authors Lars Wanhammar and Tapio Saramaki.

a) DIRECT FORM

The structure is called direct form as it is the direct implementation of the convolution operation of equation 2.1.1. Thus, it is the easiest structure to implement among the structures listed.

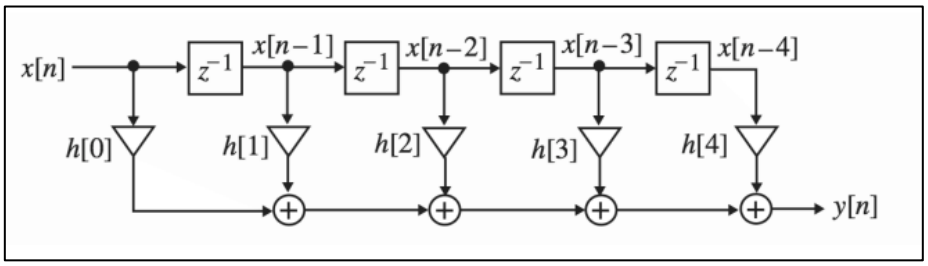


Figure 6. Direct Form

b) CASCADE FORM

The cascade form is another structure used to implement FIR filter which is obtained from the system function, i.e., the structure is a breakdown of an intended system function into a second-order FIR system cascade. The goal is to factor transfer function into product of smaller functions in a cascaded structure as seen in the figure below.

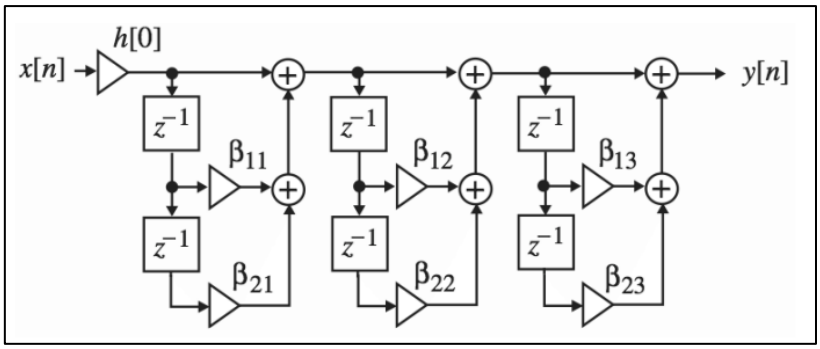


Figure 7. Cascade Form

c) LATTICE STRUCTURE

The lattice form structure is a computationally efficient way to implement Finite Impulse Response (FIR) filters. It is a feedforward filter structure that represents the filter as a series of lattice stages. Each stage consists of a reflection coefficient and a delay element. The input signal is processed through each stage, and the output of each stage is added to the output of the previous stage to produce the final output of the filter. The lattice form structure is obtained by recursively computing the reflection coefficients of the filter. It is used in a variety of adaptive filter applications and is natural for creating complementary filter pairs. The net impulse response of the lattice filter is of finite length, making it a useful structure for practical applications.

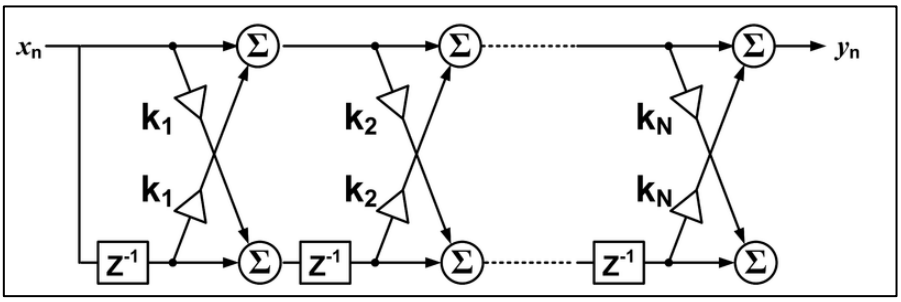


Figure 8. Lattice Structure

d) SYMMETRIC FIR

This structure uses symmetric linear phase whereby the filter coefficients reduce the complexity of the multipliers to N/2+1 for even Nth order filters and (N+1)/2 for odd Nth order filters. The goal is to reduce the number of multipliers by leveraging the symmetry around the center tap.

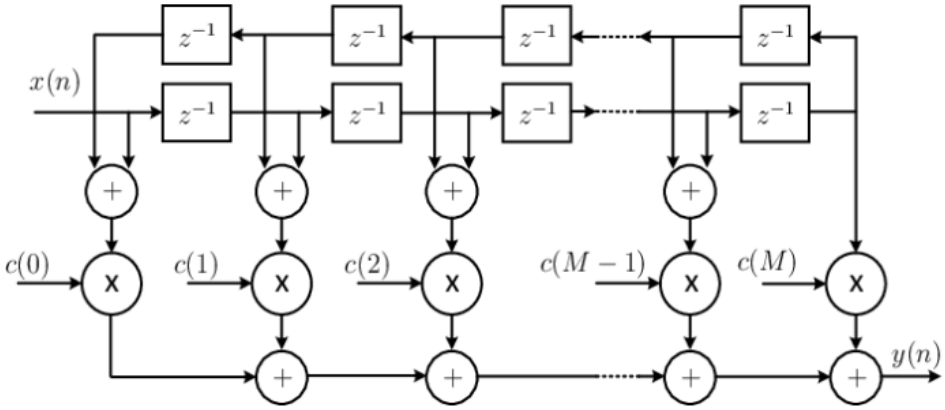


Figure 9. Symmetric FIR

2.3 RELATED WORKS

In this section, we are going to briefly look at related works that studied the comparison between the structures in terms of their performances.

Year	Title	Author(s)	Strength	Improvements
2016	A High-Performance FIR Filter Architecture for Fixed and Reconfigurable Applications	Basant Kumar & Pramod Kumar	The implementation was done using processors and the time complexity was used as the metric of performance among structures.	The authors did not consider noise and the quantization sensitivity of the structures.
2017	Design and implementation of an efficient FIR digital filter	Sumbal Zahoor & Shahzad Naseem	The implementation was done in FPGA. The performance is measured by filter characteristics.	The authors did not compare structures of FIR filters, but effectively compare windowing method. Noise is not considered.

2017	Comparative study of 16-order FIR filter design using different multiplication techniques	Anubhuti Mittal, Ashutosh Nandi, Disha Yadav	The implementation was done in FPGA and the number of LUTs was used as the measure of performance.	Noise and the quantization sensitivity as well as filter characteristics of the structures are not considered.

Table 2. Summary of Related Works

2.4 SUMMARY

The best implementation of FIR filters depends on the specific requirements and constraints of the application. In general, FIR filters can be implemented using hardware, such as DSPs or FPGAs or using software (e.g., MATLAB or C programming language) running on a general-purpose computer or microcontroller. The choice of implementation will depend on factors such as the required filter performance, the available resources, and the cost and complexity of the implementation which is depending on the choice of structure used. In some cases, a hybrid approach that combines hardware and software elements may be the best solution. As for related works, this study will largely benchmark against Sumbal Zahoor & Shahzad Naseem (2017) and improvements made by considering noise and quantization sensitivity of the filter structures.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 OVERVIEW

The scope of this research is to compare the performance differences among various filter structures, namely the direct form, cascade form, lattice structure, and symmetric FIR. To accomplish this, the frequency response and signal-to-noise ratio (SNR) of the quantized filter coefficients for each structure were analyzed. Moreover, the root-mean-square (RMS) value was used as a metric to evaluate the filter's performance and assess the deviation of the quantized filter coefficients from their original frequency response.

3.2 EXPERIMENTAL SETUP

3.2.1 DESIGNING FIR FILTERS

In the experimental setup, MATLAB was employed as the primary tool for designing the FIR filters with varying filter orders, ranging from N = 50 to N = 500. To ensure consistency, the specifications of the filter were held constant throughout the experiment thereby allowing for a focused assessment of its performance.

Type of Filter	**Structures**	**Window Method**	**Passband Frequency (Hz)**	**Stopband Frequency (Hz)**	**Passband Ripple (dB)**	**Stopband Attenuation (dB)**
Lowpass	Direct Form	Rectangular	0-400	500	0.1	40
	Cascade Form					
	Lattice Structure					
	Symmetric FIR					

Table 3. Filter Specifications

3.2.2 IMPLEMENTING FIR FILTER

The implementation of FIR filters with alternative structures was conducted using the C++ programming language, which offered a robust and flexible environment for the implementation and subsequent analysis of the filters. A test signal was employed in the experiment as well as adding noise to the signal thereby enhancing the reliability and significance of the results obtained. Further details and a description of the test signal are provided below:

$$x(t) = \cos(200\pi t) + \sin(1000\pi t) + \gamma(t) \qquad (3.2.1)$$

Where $\gamma(t)$ is Gaussian noise function with the mean and standard deviation of 0 and 0.25 respectively. This approach allows for the investigation of the FIR filters' performance in handling real-world scenarios where signals are subject to noise interference.

Additionally, the experiment involved uniformly quantizing the filter coefficients, which was followed by an analysis of the frequency response and signal-to-noise ratio (SNR) of the filters. By uniformly quantizing the coefficients, the experiment aimed to simulate the effects of finite precision representation in digital implementations. Both the frequency response and SNR analysis were implemented in MATLAB. The analysis of the frequency response provided insights into the filter's magnitude and phase characteristics, while the SNR assessment quantified the ability of the filter to preserve the signal fidelity in the presence of quantization noise. This comprehensive analysis enabled a thorough evaluation of the filters' performance under realistic digital signal processing conditions.

3.2.3 PSEUDOCODE FOR EACH STRUCTURE

In this section, we will examine the pseudocode for each structure implemented in the C++ code. It's important to note that several variable names are shared among the structures, representing the same data type. Specifically, **arr_x**, **arr_y**, and **arr_h** denote the arrays of input, output, and filter coefficients, respectively. The variable **SIZE_N** refers to the size of the input and output arrays, while **SIZE_M** represents the size of the filter coefficients.

a) Direct Form

```
// Define functions of Direct Form
function setArr_Y(N)
   SIZE_N = N
   arr_y = new double[N]
function getArr_Y(index)
   return arr_y[index
function delArr_Y()
   delete[] arr_y
function setDirectForm(arr_x, arr_h, SIZE_M)
   for i = 0 to SIZE_N
      for j = 0 to SIZE_M
         if i- j >= 0
            arr_y[i] += arr_h[j] * arr_x[i- j]
```

The given code snippet consists of several functions for implementing an FIR filter using the direct form method. The **setDirectForm** function calculates the values of **arr_y** based on the convolution of input arrays **arr_x** and **arr_h**. It uses nested loops to iterate over the elements of **arr_y**, multiplying the corresponding elements of **arr_h** and **arr_x**, and accumulating the results in **arr_y** according to the convolution formula. The conditions within the loops ensure that the indices are within the valid range for the convolution calculation.

b) Cascade Form:

```
// Define functions for Cascade Form
function setArr_Y(N)
   SIZE_N = N
   arr_y = new double[N]

function getArr_Y(index)
   return arr_y[index]

function delArr_Y()
   delete[] arr_y

function setCascadeForm(arr_x, arr_h, SIZE_M)
   buffer = new double[2000]

   for i = 0 to SIZE_N
      coeff = 0.0
      buffer[0] = arr_x[i]

      for j = SIZE_M - 1 to 1 step -1
         buffer[j] = buffer[j - 1]

      for j = 0 to SIZE_M
         coeff += arr_h[j] * buffer[j]

      arr_y[i] = coeff

   delete[] buffer
```

The given code snippet consists of functions for implementing an FIR filter using the cascade form method. The **setCascadeForm** function calculates the values of **arr_y** based on the convolution of input arrays **arr_x** and **arr_h**. It uses a temporary buffer array to store the intermediate values during computation. The function iterates over the elements of **arr_y** and shifts the elements of buffer to the right, keeping track of the most recent input sample. It then performs convolution by multiplying the corresponding elements of **arr_h** and buffer and accumulating the result in **coeff**. Finally, the computed **coeff** is assigned to the current element of **arr_y**.

c) Lattice Structure

```
// Define functions for Lattice Form
function setArr_Y(N)
   SIZE_N = N
   arr_y = new double[N]

function getArr_Y(index)
   return arr_y[index]

function delArr_Y()
   delete[] arr_y

function setLatticeForm(arr_x, arr_h, SIZE_M)
   u = new double[2000]
   v = new double[2000]
   for i = 0 to SIZE_N
      y = 0.0
      for j = SIZE_M downto 0
         y += arr_h[j] * u[j]
         if j == 0
            u[0] = arr_x[i] - y
         else
            u[j] = u[j - 1] - arr_h[j] * v[j - 1]
         v[j] = v[j - 1] + arr_h[j] * u[j - 1]
      arr_y[i] = y
   delete[] u
   delete[] v
```

The given code snippet consists of functions for implementing an FIR filter using the lattice form method. The **setLatticeForm** function calculates the values of **arr_y** based on the lattice form implementation of the FIR filter. It uses two temporary arrays, **u** and **v**, to store intermediate values during computation. The function iterates over the elements of **arr_y** and performs the lattice filtering operations. It calculates the output y by multiplying the corresponding elements of **arr_h** and **u** and adding them to the previous value of **y**. At each stage of the lattice filter, it updates the values of **u** and **v** based on the previous values and the filter coefficients. Finally, the computed y is assigned to the current element of **arr_y**.

d) Symmetric FIR

```
// Define functions of Symmetric Even
function setArr_Y(N)
   SIZE_N = N
   arr_y = new double[N]

function getArr_Y(index)
   return arr_y[index]

function delArr_Y()
   delete[] arr_y

function setSymmetricFIR(arr_x, arr_h, SIZE_M)
   delay = SIZE_M / 2 - 1
   for i = delay to SIZE_N - delay
      for j = 0 to SIZE_M
         if i - j >= 0 and i + j < SIZE_N
            arr_y[i] += arr_h[j] * arr_x[i + j - delay]
```

The given code snippet consists of functions for implementing an FIR filter using the symmetric structure. The **setSymmetricFIR** function calculates the values of **arr_y** based on the symmetric even form implementation of the FIR filter. It uses a variable delay to represent the delay introduced by the filter. The function iterates over the elements of **arr_y** within the valid range and performs the filter calculations. It multiplies the corresponding elements of **arr_h** and **arr_x** and adds them to the output **arr_y**. The indices are adjusted by subtracting the delay to ensure proper alignment.

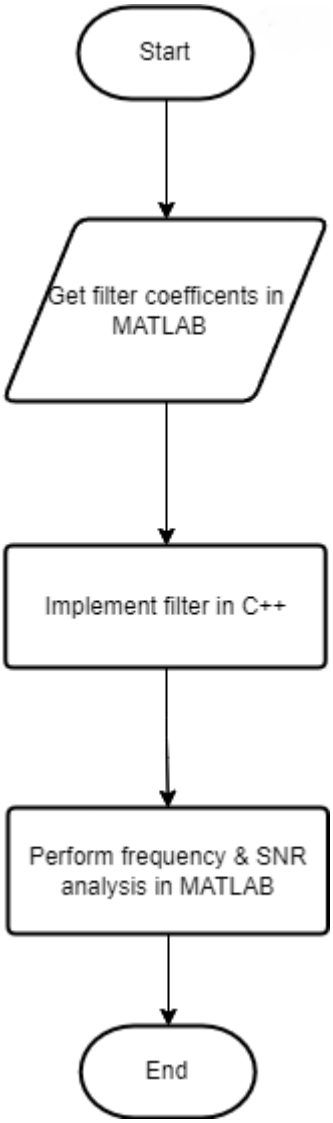
3.2.4 PSEUDOCODE FOR QUANTIZATION METHOD

Quantizing the filter coefficients is done using a uniform quantization scheme with the number of bits set to 8. These values are then stored in **arr_h** to see the differences in the filter behavior under this condition during the FIR implementation.

```
// Define steps of Quantization
n_bits = 8
coeff_range = max(arr_h) – min(arr_h)
step_size = coeff_range / (2**n_bits –1)
for i = 0 to SIZE_M
        arr_h[i] = round(arr_h[i] / step_size ) * step_size
```

The pseudocode provided quantizes an array **arr_h** using a specified number of bits, **n_bits**. It calculates the range of values in the array and determines the step size for quantization. Then, it iterates through each element of **arr_h**, divides it by the step size, rounds the result to the nearest integer, and multiplies it back by the step size. This process ensures that each value is quantized to a discrete level determined by the number of bits, resulting in a quantized version of the array.

3.3 FLOWCHART



CHAPTER 4

RESULT AND ANALYSIS

4.1 OVERVIEW

In this section, we will examine the noteworthy and meaningful outcomes of the experiment to evaluate the performance of different FIR filter structures using two methods, focusing on their sensitivity to quantization: 1) frequency response analysis, and 2) SNR analysis. Additionally, we will provide a detailed explanation of the observed results and objectively determine the best-performing filter structure based on the defined criteria.

4.2 FREQUENCY RESPONSE ANALYSIS

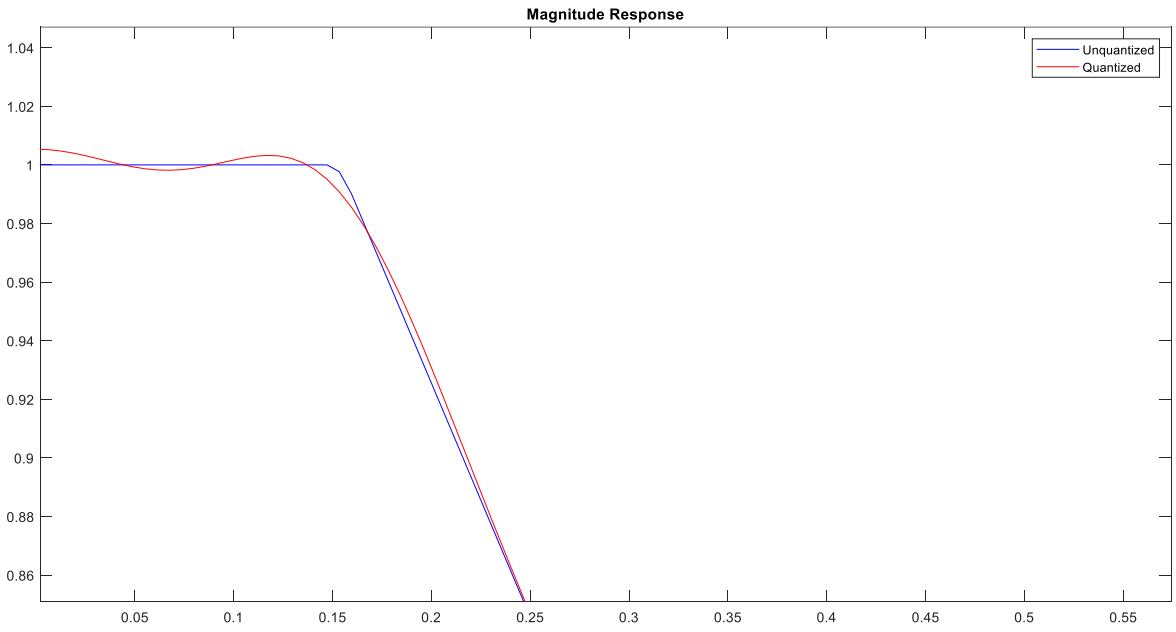


Figure 10. Direct Form Magnitude Response

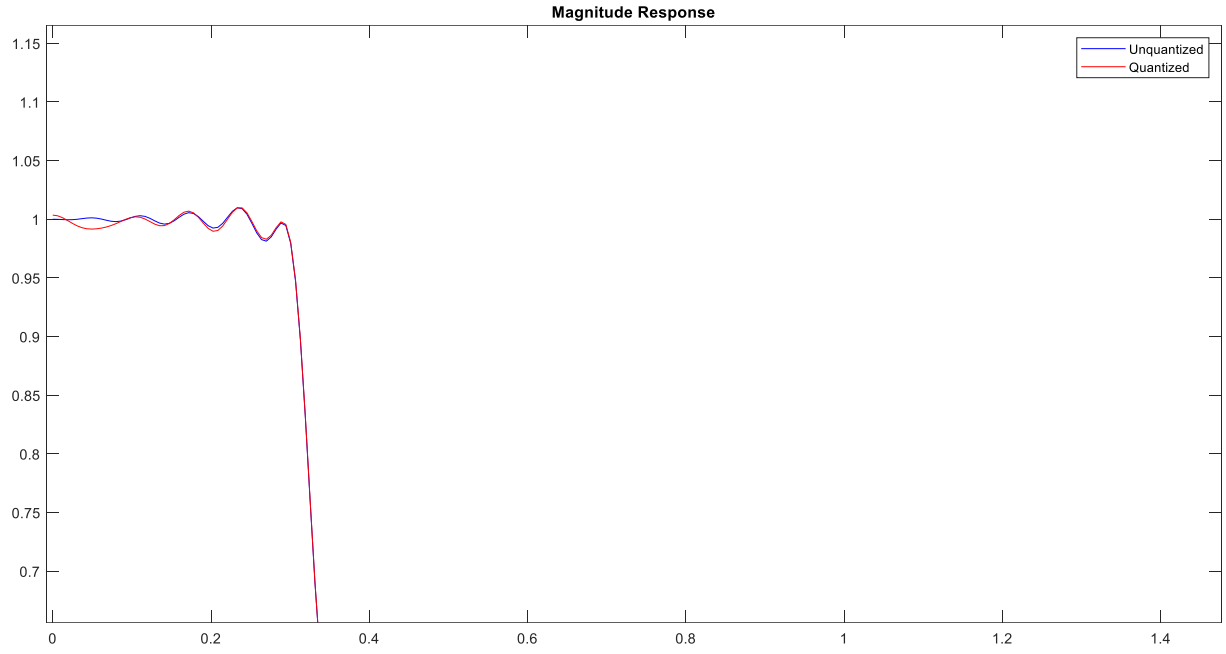


Figure 11. Symmetric FIR Magnitude Response

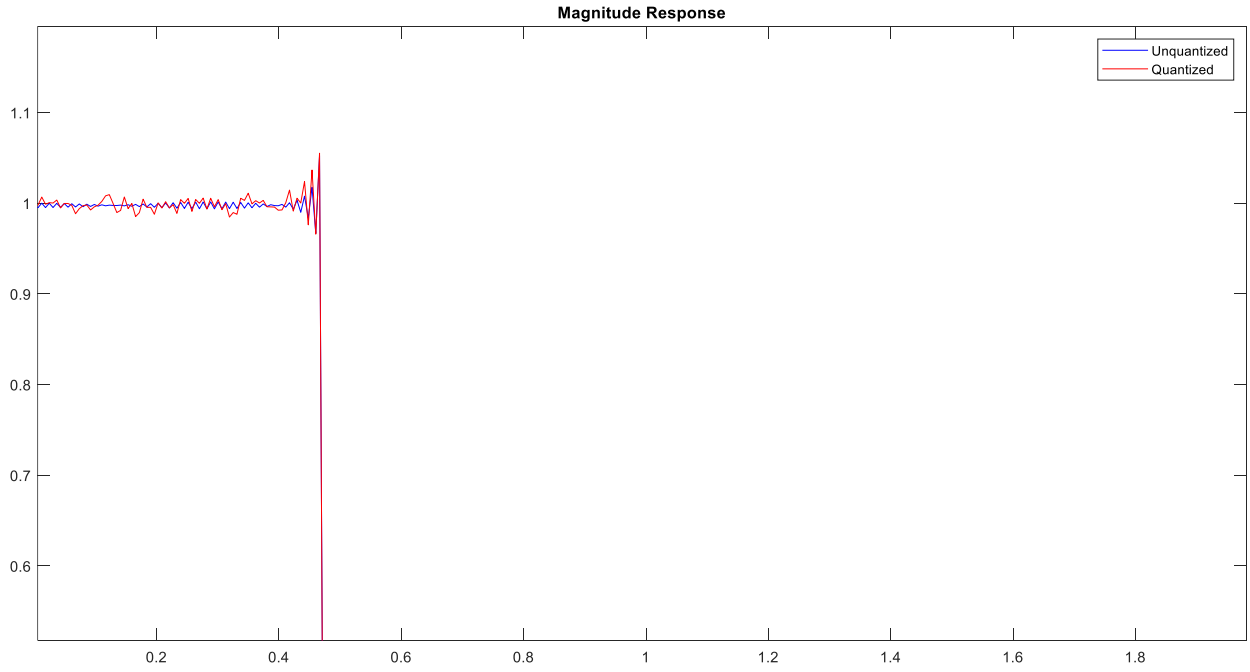


Figure 12. Cascade Form Magnitude Response

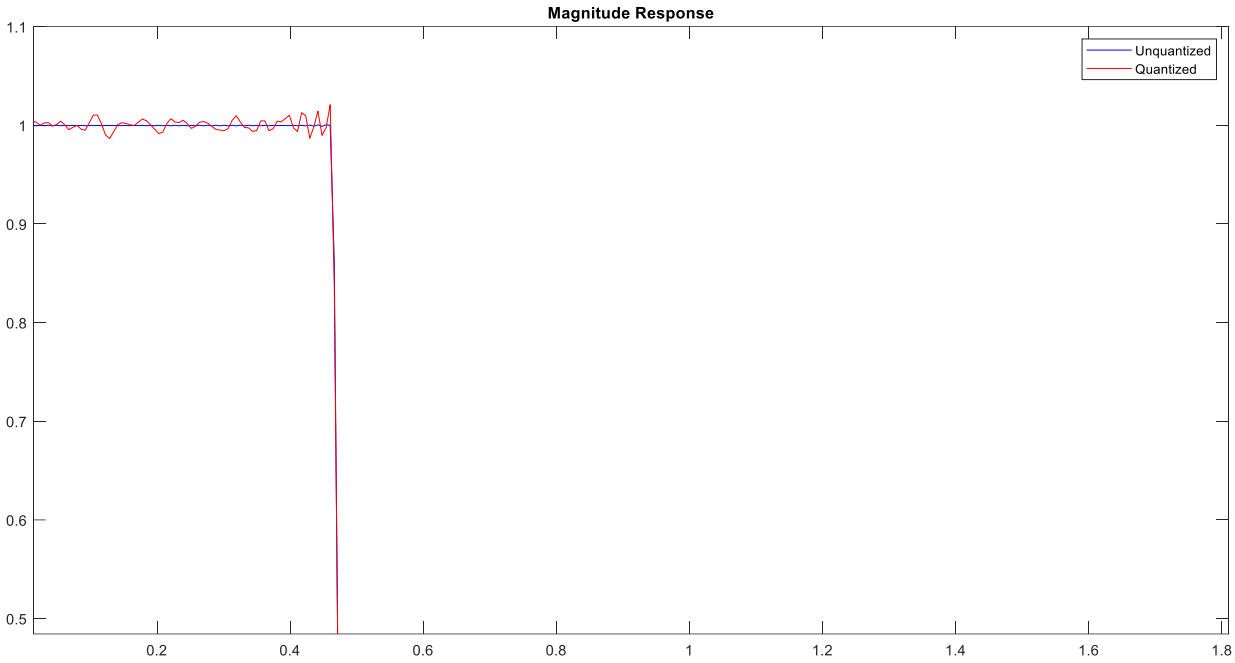


Figure 13. Lattice Form Magnitude Response

Structures	RMS
Direct Form	0.000183
Symmetric FIR	0.000161
Cascade Form	0.000199
Lattice Structure	0.000062

Table 4. RMS of Quantized Magnitude Response

The figures above show two overlapping magnitude responses of unquantized and quantized filter coefficients for each structure. Both direct form and symmetric FIR exhibit lesser ripples though the values fluctuate with higher amplitude than cascade form and lattice structure; the analysis can be observed from the given figures and table. These observations suggest that the direct form and symmetric FIR filters have better performance in terms of

ripple reduction. However, it's important to note that the higher amplitude fluctuations in these filters may introduce additional quantization noise or errors in the output.

Additionally, **Figure 12** clearly indicates the presence of the Gibbs phenomenon in the magnitude response of the cascade form filter for both unquantized and quantized filter coefficients, which can be attributed to the inherent characteristics of this filter structure. The Gibbs phenomenon refers to the occurrence of overshoots or ripples at the stopband attenuation in the magnitude response. However, it is important to note that the effect of the Gibbs phenomenon is more pronounced in the case of quantized filter coefficients in this structure. The quantization process amplifies the overshot, leading to more prominent artifacts in the magnitude response. As a result, this can potentially introduce unintended consequences or artifacts that affect the overall performance of the filter. This observation was supported by **Table 4** which proves that the RMS of quantized filter coefficients of this structure is the highest among the structures listed.

Lastly, **Figure 13** reveals that the quantized filter coefficients of the lattice structure introduce ripples in the magnitude response. Nonetheless, despite the presence of these ripples, **Table 4** provides valuable insight by indicating that the RMS value associated with the lattice structure is the least among the listed structures. This observation suggests that although the quantized filter coefficients in the lattice structure exhibit ripples in the magnitude response, the overall energy or power of these ripples is relatively lower compared to the other filter structures under consideration. The lower RMS value indicates a potentially improved performance in terms of reducing the overall impact of these ripples on the filtered output.

4.3 SNR ANALYSIS

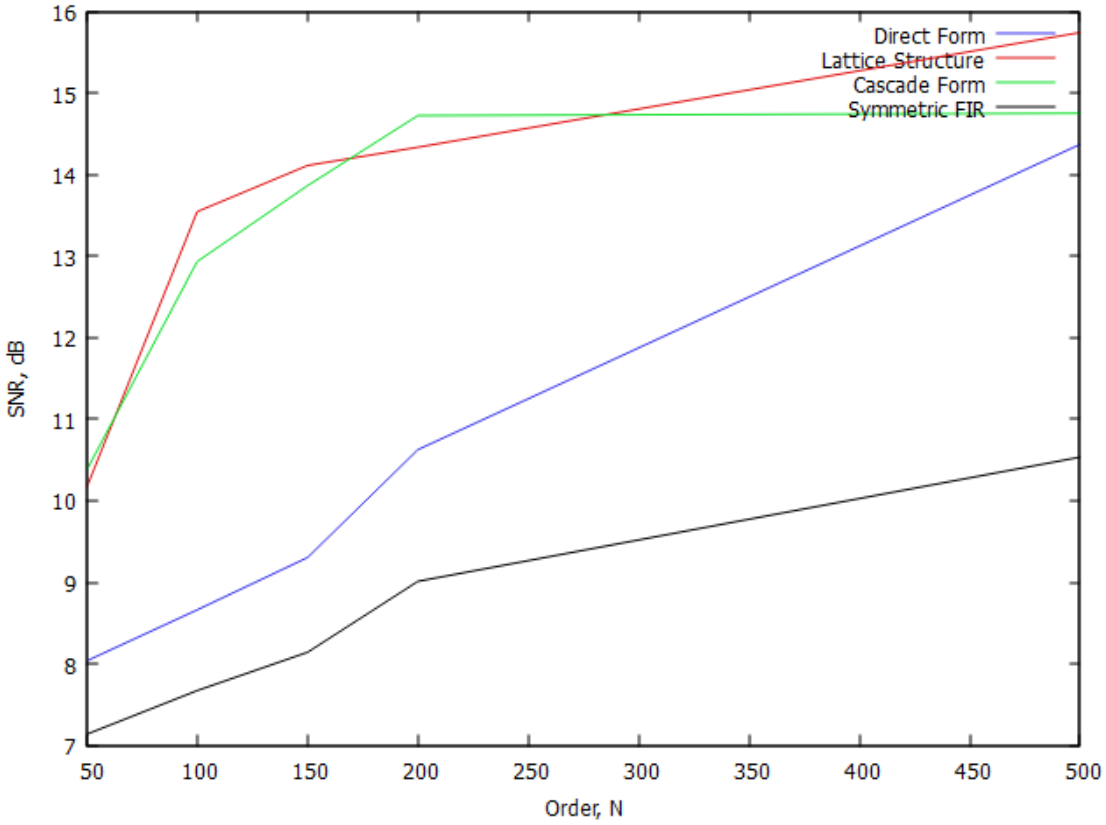


Figure 14. SNR for Quantized Filter Coefficients

Figure 14 illustrates the overlapping graphs of SNR of quantized filter coefficients of alternative structures of FIR filter in respect to the order of the filter. As shown in the figure, the lattice structure outperforms other structures listed with a margin of approximately +1.0 dB more than cascade form, +2.0 dB more than direct form, and +5.0 dB more than symmetric FIR when order N = 500. By achieving higher SNR values, the lattice structure demonstrates a more effective suppression of the quantization noise compared to the other considered FIR filter structures. This suggests that the lattice structure, in its design and characteristics, has properties that enable it to mitigate the impact of quantization noise more efficiently. The superior performance of the lattice structure in preserving the SNR is particularly valuable in applications where maintaining signal quality and minimizing noise artifacts are crucial.

4.4 COMPARISON OF ALTERNATIVE STRUCTURES

Structures	Filter Characteristics
Direct Form	• Easy to implement. • Less passband ripples. • High RMS value. • Low SNR.
Cascade Form	• Difficult to implement. • More passband ripples. • Introduce Gibbs phenomenon. • High RMS value. • High SNR.
Lattice Structure	• Difficult to implement. • More passband ripples. • Low RMS value. • High SNR.
Symmetric FIR	• Easy to implement. • More passband ripples. • High RMS value. • Low SNR.

Table 5. Comparison of Alternative Structures

4.5 SUMMARY

To conclude, the experiments conducted provide evidence supporting the superior performance of the lattice structure in comparison to the direct form, cascade form, and symmetric FIR filter. The key contributing factor to this performance advantage is the lattice structure's low sensitivity to quantization. This conclusion is supported by two significant observations: the low root mean square (RMS) value in the magnitude response and the high signal-to-noise ratio (SNR) in relation to quantized filter coefficients. The low RMS value observed in the magnitude response of the lattice structure signifies its ability to minimize the amplitude of ripples introduced by quantization. This indicates improved accuracy and reduced distortion in the filter's frequency response, resulting in a more precise representation of the desired signal. Additionally, the high SNR values observed for the lattice structure highlight its capability to effectively suppress quantization noise while preserving the quality of the desired signal.

CHAPTER 5

CONCLUSION

5.1 CONCLUSION

To summarize, this paper shows that in terms of overall performance, lattice structure has the lowest sensitivity to quantization as it has the highest SNR as well as small ripple amplitude in the passband. Both direct form and symmetric FIR produce the least passband ripple due to quantization; however, their SNRs are the lowest. Moreover, cascade form introduces Gibbs phenomenon in the frequency response, but it has relatively high SNR when submitted to quantization.

5.2 FUTURE WORK

In future works, it will be crucial to conduct tests on these structures using various parameters to evaluate their performance, including factors such as time complexity and space requirements for implementing the FIR filters. These tests will yield more comprehensive results, aiding in the determination of the most optimal FIR structures for implementation purposes.

REFERENCES

Wanhammar, L., & Saramäki, T. (n.d.). *Digital Filters Using MATLAB*.

Digital Filters and Signal Processing. (2013). In *Digital Filters and Signal Processing*. InTech. https://doi.org/10.5772/45654

(Prentice-Hall Signal Processing Series) Alan V. Oppenheim, Ronald W. Schafer, John R. Buck - Discrete-Time Signal Processing -Prentice Hall (1999). (n.d.).

Zahoor, S., & Naseem, S. (2017). Design and implementation of an efficient FIR digital filter. *Cogent Engineering*, *4*(1). https://doi.org/10.1080/23311916.2017.1323373

Mohanty, B. K., & Meher, P. K. (2016). A High-Performance FIR Filter Architecture for Fixed and Reconfigurable Applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *24*(2), 444–452. https://doi.org/10.1109/TVLSI.2015.2412556

Julius O. (1985). Fundamentals of Digital Filter Theory. In *Autumn* (Vol. 9, Issue 3). https://www.jstor.org/stable/3679573

Mittal, A., Nandi, A., & Yadav, D. (2017). Comparative study of 16-order FIR filter design using different multiplication techniques. *IET Circuits, Devices and Systems*, *11*(3), 196–200. https://doi.org/10.1049/iet-cds.2016.0146

Zheng, J., & Wei, Z. (2018). FIR filter design based on FPGA. *Proceedings - 10th International Conference on Measuring Technology and Mechatronics Automation, ICMTMA 2018*, *2018-January*, 36–40. https://doi.org/10.1109/ICMTMA.2018.00016

Molina, B., Abdullah, M. W., & Calatayud, B. M. (n.d.). *FIR Filters: Design and Implementation using FPGAs Related papers A Pract ical Approach DIGITAL DESIGN OF SIGNAL PROCESSING SYST EMS DIGITAL DESIGN OF … FIR Filters: Design and Implementation using FPGAs*.

Roychowdhury, A. (n.d.). *FIR Filter Design Techniques*.

Sri Venkateshwara College of Engineering. Department of Electronics and Communication Engineering, Institute of Electrical and Electronics Engineers. Bangalore Section, IEEE Computer Society, & Institute of Electrical and Electronics Engineers. (n.d.). *RTEICT-2017 : 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology : proceedings : 19-20 May 2017*.

Liao, Y. B., Li, P., Ruan, A. W., & Li, W. C. (2010). Design and verification of an ALU-based universal FIR filter. *COMPEL - The International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, *29*(2), 317–326. https://doi.org/10.1108/03321641011014788

Podder, P., Zaman Khan, T., Haque Khan, M., & Muktadir Rahman, M. (2014). Comparative Performance Analysis of Hamming, Hanning and Blackman Window. In *International Journal of Computer Applications* (Vol. 96, Issue 18).

Borodzhieva, A. (2022). Active Learning Used for Teaching the Topic "Design of Finite Impulse Response Filters in MATLAB" in the Course "Digital Signal Processing." *International Conference on Systems, Signals, and Image Processing*, *2022-June*. https://doi.org/10.1109/IWSSIP55020.2022.9854458

APPENDIX

This appendix consists of the MATLAB codes that generate filter coefficients for each structure.

a) Direct Form

```
Fpass = 100;    % Passband frequency (Hz)
Fstop = 500;    % Stopband frequency (Hz)
Fs = 4000;      % Sampling rate (Hz)
Rp = 0.5;       % Passband ripple (dB)
Rs = 40;        % Stopband attenuation (dB)

% Set the filter order to 200
N = 200;

% Design the filter using the fir2 function
f_cutoff = [0 Fpass Fstop Fs/2];
magnitudes = [1 1 0 0];
b = fir2(N, f_cutoff/(Fs/2), magnitudes);

% Create a cascade filter structure using the dfilt.cascade function
Hd = dfilt.cascade(dfilt.df1(b));

% Write the filter coefficients to a text file
fileID = fopen('coefficient_direct_form.txt','w');
fprintf(fileID,'%.16f\n', b);
fclose(fileID);
```

b) Cascade Form

```
Fpass = 100;     % Passband frequency (Hz)
Fstop = 500;     % Stopband frequency (Hz)
Fs = 4000;       % Sampling rate (Hz)
Rp = 0.5;        % Passband ripple (dB)
Rs = 40;         % Stopband attenuation (dB)

% Set the filter order to 200
N = 200;

% Design the filter using the fir1 function
f_cutoff = (Fstop + Fpass) / 2;
b = fir1(N, f_cutoff/(Fs/2), 'low');

% Create a cascade filter structure using the dfilt.cascade function
Hd = dfilt.cascade(dfilt.df1(b));

% Write the filter coefficients to a text file
fileID = fopen('coefficient_cascade.txt','w');
fprintf(fileID,'%.16f\n', b);
fclose(fileID);
```

c) Lattice Structure

```
Fpass = 100; % Passband frequency (Hz)
Fstop = 500; % Stopband frequency (Hz)
Fs = 4000; % Sampling rate (Hz)
Rp = 0.5; % Passband ripple (dB)
Rs = 40; % Stopband attenuation (dB)

% Set the filter order to 200
N = 200;

% Design the filter using the fir1 function
f_cutoff = (Fstop + Fpass) / 2;
f_norm = f_cutoff / (Fs/2);
b = fir1(N, f_norm, 'bandpass', kaiser(N+1, Rp));

% Generate the lattice filter coefficients using the tf2latc function
[a, k] = tf2latc(b, 1);

% Write the filter coefficients to a text file
fileID = fopen('coefficient_lattice.txt','w');
fprintf(fileID,'%.16f\n', k);
fclose(fileID);
```