DEPARTMENT OF ELECTRONIC & TELECOMMUNICATION ENGINEERING
UNIVERSITY OF MORATUWA

EN 4420 – ADVANCE SIGNAL PROCESSING

PROJECT REPORT

DESIGN OF AN M-FOLD INTERPOLATOR

NAME	INDEX NUMBER
ABEYWARDENA K.G.	160005C

THIS REPORT IS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE MODULE EN 4420 – ADVANCE SIGNAL PROCESSING.

MARCH 01, 2021

ABSTRACT

This report discusses the design and implementation of an M-fold interpolator, with the upsampling factor $M \in \mathbb{Z}^+$. The filter specification for the anti-imaging filter within the M-fold interpolator is derived based on the sampling theory for the given signal to be interpolated. Two anti-imaging filters with stopband attenuations of 30dB and 60dB are designed following the procedure used to design a Finite-Duration Impulse Response (FIR) Low-Pass filter. The truncation of the Infinite Response initially obtained, is achieved from the Kaiser Window Function. The designed anti-imaging filters are implemented using the *polyphase structure* and the interpolators are implemented using the *efficient structure* in MATLAB R2016a. The report presents the magnitude responses and impulse responses of the filters designed to confirm its characteristics with the desired specifications. Further, the performance of the two interpolators designed are evaluated based on the Root Mean Square Error (RMSE) between the sampled signal at the higher sampling rate and the outputs from each interpolator and the computational complexity difference between the original and efficient implementations of the designed interpolators.

CONTENT

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

This report describes the step-by-step procedure used to design an M-fold Interpolator with Low-Pass FIR Digital Filter as the Anti-Imaging Filter for prescribed specifications using the windowing method in conjunction with the Kaiser window. The software implementation and the evaluation of the M-fold interpolators was done by MATLAB (Version R2016a)

The ideal passband gains and the passband edge and stopband frequencies were obtained following the sampling theory for the given signal $x[n]$. When selecting the passband and stopband edge frequencies, the widest possible transition band was considered as that reduces the need for sharp transitions and hence the order of the filters to be designed. This leads to the lower computational complexity as it increases with the order of the filter.

The closed form direct approach is used by following the Fourier series method to define and design the even-order linear-phase anti-imaging FIR digital filters for the derived and given specifications, while Kaiser Window is used for windowing. The parameters of the Kaiser window were used to tune the filters to the prescribed characteristics. The designed FIR digital filters are then implemented using the polyphase structure such that the implementation of the M-fold interpolators can be done in the efficient structure derived.

The time domain and frequency domain representation of the filters are obtained throughout the different design stages to evaluate the filter characteristics. The frequency responses of the filters were obtained primarily through the Fast Fourier Transform (FFT) algorithm implementation which provides a faster implementation of the Discrete Fourier Transform (DFT). The filters were then evaluated for its capability of successful re-sampling of the original signal with negligible distortion and computations complexities and the reduction of computation complexity with the efficient implementations.

2 METHOD

The project has the following requirements.

- Deriving the Ideal Passband Gain, Passband, and Stopband Edge frequencies for the Anti-Imaging filters to be designed
- Designing Anti-Imaging filters with two different Stopband attenuations, $H_{30}(z)$ and $H_{60}(z)$
- Implementing the designed Anti-Imaging filters in Polyphase structure and the M-fold interpolators using the Efficient structure
- Evaluating the performance of the M-fold interpolators in-terms of the ability to re-sample the original signal and the computational complexities.

2.1 Deriving Filter specifications

The passband gains, passband and stopband edges with widest possible transition width are derived in this section. To derive them, I will be employing the concepts of the sampling theory.

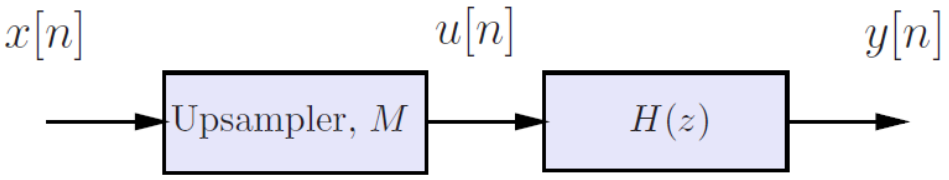

Fig. 2.1: Structure of the M-fold Interpolator

First, consider the continuous time signal $x_c(t) = 2cos(2\pi f_0 t)$ that corresponds to the given sequence $x[n] = 2\cos(2\pi f_0 n T_s)$, which is bandlimited. Using the Fourier transform, we obtain the frequency representation of the $x_c(t)$, denoted by $X_c(\Omega)$ as follows:

$$X_c(\Omega) = 2\pi\delta(\Omega - \Omega_0) + 2\pi\delta(\Omega + \Omega_0), \qquad where\ \Omega_0 = 2\pi f_0$$

By sampling the $x_c(t)$ at a sampling rate of f_s ($\geq 2f_0$), we can obtain the $x_s(t)$ with the Fourier Transform of $X_s(\Omega)$ as follows:

$$x_s(t) = \sum_{n=-\infty}^{\infty} x_c(nT_s)\delta(t - nT_s) = \sum_{n=-\infty}^{\infty} 2cos(2\pi f_0 nT_s)\delta(t - nT_s)$$

and

$$X_s(\Omega) = \frac{1}{T_s}\sum_{k=-\infty}^{\infty} X_c(\Omega - 2\pi f_s k) = \frac{2\pi}{T_s}\sum_{k=-\infty}^{\infty} [\delta(\Omega - \Omega_s k - \Omega_0) + \delta(\Omega - \Omega_s k + \Omega_0)]$$

Now the given signal $x[n]$ can be obtained as follows:

$$x[n] = \int_{-\infty}^{\infty} x_c(nT_s)\delta(t - nT_s)\ dt = x_c(nT_s) = 2cos(2\pi f_0 nT_s)$$

The Discrete-Time-Fourier-Transform (DTFT) of $x[n]$, which is denoted by $X(\omega)$, can be obtained by the evaluating $X_s(\Omega)$ at $\Omega = {}^{\omega}/_{T_s}$.

$$X(\omega) = X_s(\Omega)|_{\Omega=\frac{\omega}{T_s}} = \frac{2\pi}{T_s}\sum_{k=-\infty}^{\infty}\left[\delta\left(\frac{\omega}{T_s} - 2\pi f_s k - 2\pi f_0\right) + \delta\left(\frac{\omega}{T_s} - 2\pi f_s k + 2\pi f_0\right)\right]$$

$$X(\omega) = \frac{2\pi}{T_s}\sum_{k=-\infty}^{\infty}\left[\delta\left(\frac{1}{T_s}(\omega - \omega_s k - \omega_0)\right) + \delta\left(\frac{1}{T_s}(\omega - \omega_s k + \omega_0)\right)\right],\ where\ \omega_s = 2\pi, \omega_0 = 2\pi f_0 T_s$$

Using the relationship of $\delta\left(\frac{x}{T}\right) = T\delta(x)$

$$X(\omega) = 2\pi\sum_{k=-\infty}^{\infty}[\delta(\omega - \omega_s k - \omega_0) + \delta(\omega - \omega_s k + \omega_0)]$$

Now consider the upsampled signal, $u[n]$ by a factor $M \in \mathbb{Z}^+$ and its Fourier transform $U(\omega)$.

$$u[n] = \begin{cases} x\left[\frac{n}{M}\right], & if\ M|n \\ 0, & otherwise \end{cases}$$

$$U(\omega) = X(M\omega) = 2\pi\sum_{k=-\infty}^{\infty}[\delta(\mathrm{M}\omega - \omega_s k - \omega_0) + \delta(\mathrm{M}\omega - \omega_s k + \omega_0)]$$

$$U(\omega) = 2\pi\sum_{k=-\infty}^{\infty}\left[\delta\left(\mathrm{M}\left(\omega - \frac{\omega_s k}{M} - \frac{\omega_0}{M}\right)\right) + \delta\left(\mathrm{M}\left(\omega - \frac{\omega_s k}{M} + \frac{\omega_0}{M}\right)\right)\right]$$

$$U(\omega) = \frac{2\pi}{M}\sum_{k=-\infty}^{\infty}\left[\delta\left(\omega - \frac{\omega_s k}{M} - \frac{\omega_0}{M}\right) + \delta\left(\omega - \frac{\omega_s k}{M} + \frac{\omega_0}{M}\right)\right]$$

Now consider a sampled signal $x_u[n]$ at a sampling frequency $f_s' = Mf_s$ with a Fourier transformation $X_u(\omega)$.

$$X_u(\omega) = 2\pi\sum_{k=-\infty}^{\infty}[\delta(\omega - \omega_s' k - \omega_0) + \delta(\omega - \omega_s' k + \omega_0)],\ where\ \omega_s' = M\omega_s$$

Since $\omega_s' k = M\omega_s k = 2\pi f_s(Mk)$, this has the same impact when taking the summation from $-\infty$ to ∞ as with $2\pi f_s k$. By changing the summation variable to $k' = Mk$:

$$X_u(\omega) = 2\pi\sum_{k'=-\infty}^{\infty}[\delta(\omega - \omega_s k' - \omega_0) + \delta(\omega - \omega_s k' + \omega_0)]$$

The objective of the interpolator is to process and convert a sequence sampled at a lower sampling frequency to a higher sampling frequency by a factor $M \in \mathbb{Z}^+$. Hence by definition, $y[n]$ and $x_u[n]$ should be the same and that results $Y(\omega)$ and $X_u(\omega)$ to be same.

The spectral components of $x_u[n]$ will be located at $\frac{2\pi f_0}{Mf_s}\ rad/sample$ within the $0-\ \pi\ rad/sample$. During the upsampling, the phenomena known as "imaging" occurs where multiple spectral components get located within $0-\ \pi\ rad/sample$ including the spectral component at $\frac{2\pi f_0}{Mf_s}$ in the spectra of $u[n]$. To remove the unnecessary images present and to filter the spectral component at $\frac{2\pi f_0}{Mf_s}$ from $U(\omega)$ following Ideal Low-Pass filter can be defined.

$$H(\omega) = \begin{cases} G_p, & for\ 0\ \leq\ |\omega|\ \leq\ \dfrac{\pi}{M} \\ 0, & for\ \dfrac{\pi}{M}\ \leq\ |\omega|\ \leq\ \pi \end{cases}$$

Then we can obtain the $Y(\omega)$ as follows:

$$Y(\omega) = \frac{2\pi G_p}{M} \sum_{k=-\infty}^{\infty} \left[\delta\left(\omega - \frac{\omega_s k}{M} - \frac{\omega_0}{M}\right) + \ \delta\left(\omega - \frac{\omega_s k}{M} + \frac{\omega_0}{M}\right)\right]$$

To achieve the objective of $Y(\omega) = \ X_u(\omega)$, we can see that $G_p = M$.

Table 2.1: Parameters provided in project description

Parameter	**Symbol**	**Value**	**Units**
Upsampling factor	M	4	-
Fundamental frequency	Ω_0	60π	rad/s
Sampling frequency	Ω_s	200π	rad/s

Based on the provided data, the images are to occur at $140\pi, 260\pi$ and $340\pi\ rad/s$ (in normalized frequencies; $0.35\pi, 0.65\pi$ and $0.85\pi\ rad/sample$) with the required spectral component at $60\pi\ rad/s$. Since the practical filters designed do not have very sharp transition, we need to derive the passband and stopband edge frequencies satisfying two needs:

a. The required spectral component at $60\pi\ rad/s$ should be filtered without any distortion.
b. The images present should be attenuated to a satisfactory level.

Further, to reduce the order of the filter, hence the computational complexity, the widest transition width should be considered when determining the passband and stopband edge frequencies. Considering these three factors, the passband edge frequency can be selected as $60\pi\ rad/s$ and stopband edge frequency can be selected as $140\pi\ rad/s$.Hence the specification of the anti-imaging filters to be designed are stated in the Table 2.2.

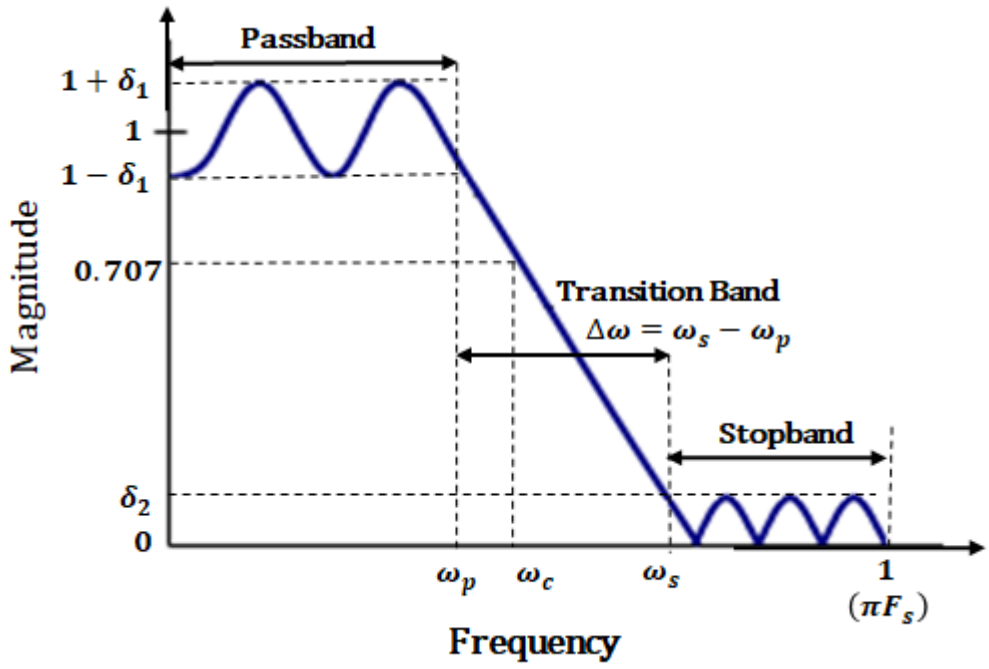

Fig. 2.2: Practical Low-Pass filter design parameters

Table 2.2: Derived and Specified Filter parameters

Specification	**Symbol**	**Value**	**Units**
Passband Gain	G_p	4	-
Stopband Gain	G_s	0	-
Passband Edge Frequency	Ω_p	188.496	rad/s
Cut-off Frequency	Ω_c	314.159	rad/s
Stopband Edge Frequency	Ω_s	439.823	rad/s
Transition width	B_T	80	rad/s
Passband Ripple	$\tilde{A}_p$	0.1	dB
Stopband Attenuation for filter 1	$\tilde{A}_{s,30}$	30	dB
Stopband Attenuation for filter 2	$\tilde{A}_{s,60}$	60	dB

2.2 Deriving the Kaiser Window parameters

The Kaiser Window function is given by,

$$w_k\,(nT_s) = \begin{cases} I_0(\beta)/I_0(\alpha) & for\ |x|\ \leq \dfrac{N-1}{2} \\ 0 & otherwise \end{cases}$$

Where α is an independent parameter and

$$\beta = \alpha\sqrt{1-\left(\frac{2n}{N-1}\right)^2} \qquad I_0(x) = 1 + \sum_{k=1}^{\infty}\left[\frac{1}{k!}\left(\frac{x}{2}\right)^k\right]^2$$

The calculation of the filter order can be done following the below equations.

$$\delta = min(\tilde{\delta}_p, \tilde{\delta}_a)$$

where $\tilde{\delta}_p, \tilde{\delta}_a$ are given as,

$$\tilde{\delta}_p = \frac{10^{0.05\tilde{A}_p} - 1}{10^{0.05\tilde{A}_p} + 1} \quad and \quad \tilde{\delta}_a = 10^{0.05\tilde{A}_a} - 1$$

Now, with the defined δ, we calculate the actual stop band loss

$$A_a = -20\log|\delta|$$

And the actual pass band ripple

$$A_p = 20\log\frac{|1+\delta|}{|1-\delta|}$$

We can choose α as,

$$\alpha = \begin{cases} 0 & for\ A_a \leq 21dB \\ 0.5842(A_a - 21)^{0.4} + 0.07886(A_a - 21) & for\ 21 < A_a \leq 50dB \\ 0.1102(A_a - 8.7) & for\ A_a > 50dB \end{cases}$$

A parameter D is chosen in order to obtain N, as

$$D = \begin{cases} 0.9222 & for\ A_a \leq 21dB \\ \dfrac{A_a - 7.95}{14.36} & for\ A_a > 21dB \end{cases}$$

Since the specifications given for the anti-image filter design requires even-order filters, N (the order of the Filter) is chosen such that it is the smallest, even integer value that satisfy the following inequality.

$$N \geq \frac{\Omega_s D}{B_t} + 1$$

Since the requirement of this project does not need to manually implement the Kaiser windowing method for filter design, the command *kaiserord* on MATLAB was used to obtain the filter orders $N_{P,30}$ and $N_{P,60}$. Due to the restrictions of the transition band obtained in the previous section, the orders of the two filters, $H_{30}(z), H_{60}(z)$, are slightly higher compared to the theoretically derived filter orders $N_{T,30}$ and $N_{P,60}$. The theoretical derivations along with the practical filter orders obtained for each filter are included in Table 2.3

Table 2.3: Derived Kaiser window parameters for 30dB filter (left) and 60dB filter (right)

Parameter	Value	Units
$\tilde{\delta}_p$	0.00576	-
$\tilde{\delta}_{a,30}$	0.03162	-
δ_{30}	0.00576	-
$A_{a,30}$	44.79	dB
$A_{p,30}$	0.1	dB
$N_{T,30}$	22	-
$N_{P,30}$	26	-

Parameter	Value	Units
$\tilde{\delta}_p$	0.00576	-
$\tilde{\delta}_{a,30}$	0.001	-
δ_{30}	0.001	-
$A_{a,30}$	60	dB
$A_{p,30}$	0.1	dB
$N_{T,30}$	30	-
$N_{P,30}$	38	-

2.3 Polyphase Filter Implementation

After the order of each filter $H_{30}(z)$ and $H_{60}(z)$ were obtained, the command *fir1()* of MATLAB was used to obtain the coefficients of the filter for the given specifications which are shown in the Fig. 2.4.

Based on the polyphase decomposition the two filters $H_{30}(z)$ and $H_{60}(z)$ were implemented following a Type-I design.

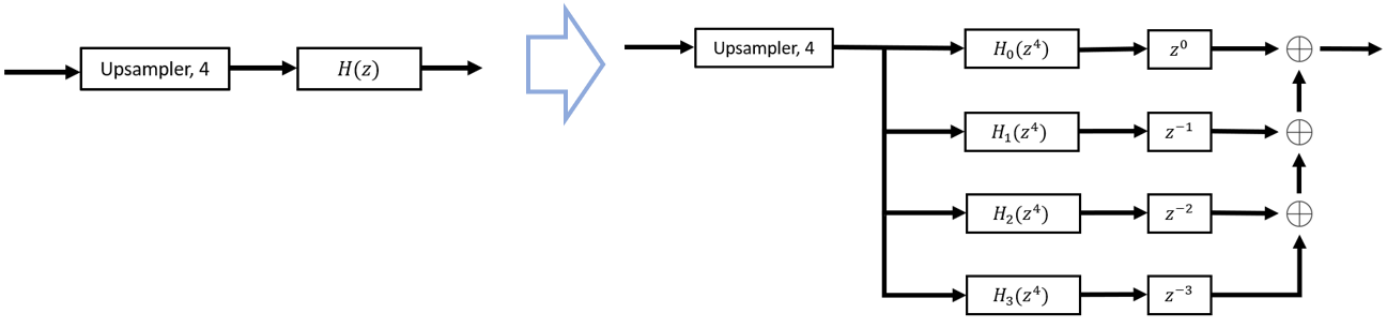

Fig. 2.3: Polyphase structure of H(z) for M = 4 and Type-I

```
Numerator:
-0.006384802067428399
 0.000000000000000005989801544802481
 0.018542928126841263
 0.039377516966182705
 0.039962425892168869
-0.000000000000000019267852173806784
-0.075918759024269145
-0.14510294775157785
-0.13883407558774635
 0.000000000000000033070025737253335
 0.27381180296402879
 0.61135163711545193
 0.89127958109887073
 1
 0.89127958109887073
 0.61135163711545193
 0.27381180296402879
 0.000000000000000033070025737253335
-0.13883407558774635
-0.14510294775157785
-0.075918759024269145
-0.000000000000000019267852173806784
 0.039962425892168869
 0.039377516966182705
 0.018542928126841263
 0.000000000000000005989801544802481
-0.006384802067428399
```

```
Numerator:
 0.000966086803646699 87
 0.002888082959978285 8
 0.003620091324138898 6
-0.000000000000000004036060643500291
-0.008803149515158833 7
-0.017981164171958258
-0.017748752006050215
 0.000000000000000001253654597150178 8
 0.032127397899646595
 0.059533889267897 62
 0.054513967607271646
-0.000000000000000002427013760049277 3
-0.089818548817701988
-0.163377505399201 96
-0.150357505970448 61
 0.000000000000000003475660734867698 2
 0.281430815799444 12
 0.618753632052790 66
 0.893943899791183 49
 1
 0.893943899791183 49
 0.618753632052790 66
 0.281430815799444 12
 0.000000000000000003475660734867698 2
-0.150357505970448 61
-0.163377505399201 96
-0.089818548817701988
-0.000000000000000002427013760049277 3
 0.054513967607271646
 0.059533889267897 62
 0.032127397899646595
 0.000000000000000001253654597150178 8
-0.017748752006050215
-0.017981164171958258
-0.008803149515158833 7
-0.000000000000000004036060643500291
 0.003620091324138898 6
 0.002888082959978285 8
 0.000966086803646699 87
```

Fig. 2.4: Obtained filter coefficients for 30dB filter (left) and 60dB filter (right)

Next, the M-fold interpolator was implemented in the efficient structure derived during the class such that the computation complexity is reduced when processing a sequence.

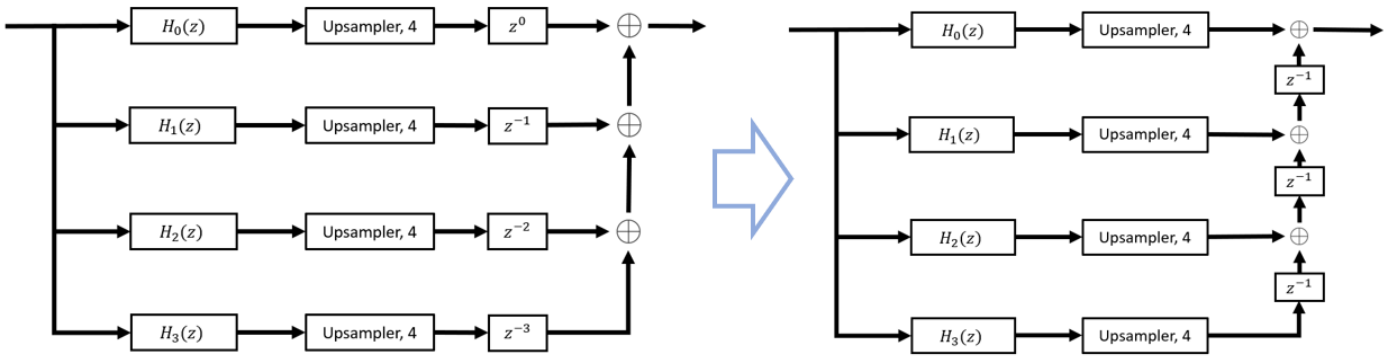

Fig. 2.5: Efficient Implementation of 4-fold Interpolator

In MATLAB, the delaying of the sequences obtained in Type-I implementation is done by shifting the row vectors by the corresponding number of samples.

2.4 Filter Evaluation

After the implementation of the filters and the M-fold interpolator in the efficient way, next is to evaluate the performance. For that, two methods were incorporated.

2.4.1 Root Mean Square Error

The two filters designed, $H_{30}(z)$ and $H_{60}(z)$, were evaluated on the ability to re-sample the original sequence $x[n]$ by interpolating it by a factor of 4 to obtain an effective sampling rate of $800\pi\ rad/s$. For this, each of the two output sequences from the two interpolators, $y_{30}[n]$ and $y_{60}[n]$, is compared with the sequence $x_u[n]$ which is obtained by sampling the continuous-time signal at a sampling frequency of $800\pi\ rad/s$ using the performance metric RMSE.

Before taking the RMSE, a correction to the output sequences should be done by removing the group delays of each filter from the corresponding output sequence. The group delays obtained were 13 and 19 samples for $H_{30}(z)$ and $H_{60}(z)$ respectively. Even though the polyphase filters may have their own group delays, the final group delay affecting the output of the LTI system is same as the overall filter implemented as it is.

After making the adjustment, samples from $1000 \leq n \leq 3000$ from both the $x_u[n]$ and $y_i[n]$ are taken to measure the RMSE using the following equation.

$$RMSE = \sqrt{\left[\frac{1}{2001}\sum_{n=1000}^{3000}(x_u[n] - y_i[n])^2\right]}$$

2.4.2 Computational Complexity

Next, the two filters were evaluated based on the computational complexity. The computational complexity was measured based on the total number of multiplications and additions performed for the input sequences they process. For this, the following evaluations were done.

i. Evaluating the computational complexity between original implementation and the efficient implementation for each of the filter
ii. Comparing the computational complexity between the two filters $H_{30}(z)$ and $H_{60}(z)$ in their efficient implementation.

3 RESULTS

In this section, the results of the design and implementation of the M-fold interpolators for 30dB and 60dB stopband attenuations are presented in three subsections. First, the time-domain and frequency-domain analysis of the two anti-imaging filters $H_{30}(z)$ and $H_{60}(z)$ are analyzed. Second, the polyphase decomposition of the two filters is exploited followed by the efficient implementation of the M-fold interpolator. Last, the filters are evaluated based on the RMSE values comparing $y_i[n]$ with $x_u[n]$ and the computational complexity based on the total number of multiplications and additions performed for the input sequences they process.

3.1 Time and Frequency domain analysis of the designed filters

In this section, the impulse responses of the designed FIR digital filters and their corresponding spectra are analyzed.

3.1.1 Filter $\mathrm{H}_{30}(\mathrm{z})$

The set of plots obtained during the design stage of the filter $H_{30}(z)$ are presented in Fig. 3.1 to Fig. 3.2. The Impulse Response of the Kaiser Window for the derived parameters is presented in Fig. 3.1. The causal ideal impulse response and the frequency response of the digital filter obtained using the Kaiser Window is depicted Fig. 3.2 and Fig. 3.3 respectively. A zoomed in plot of the pass band of the signal is obtained in order to visually identify the pass band ripples as shown in the Fig. 3.3.

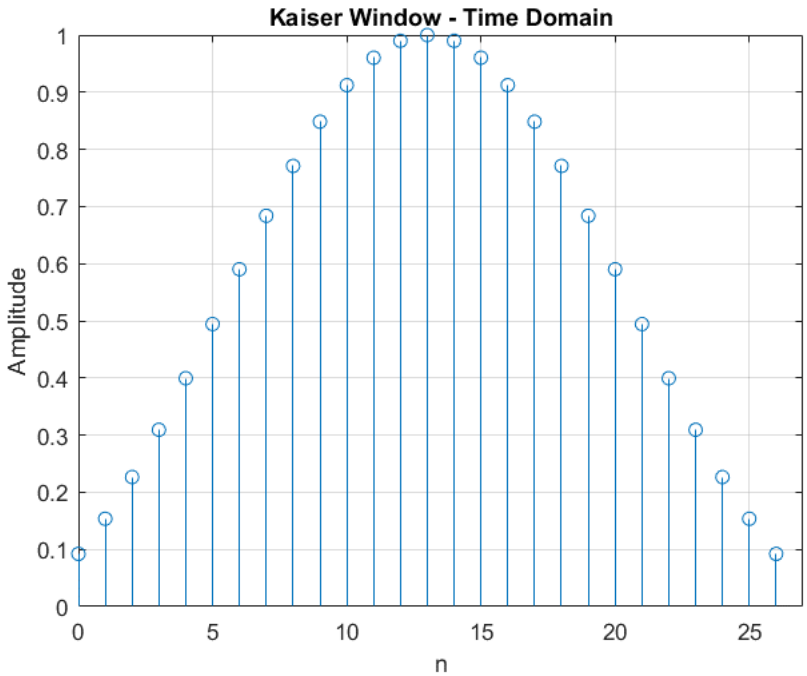

Fig. 3.1: Kaiser Window for 30dB filter - Time domain

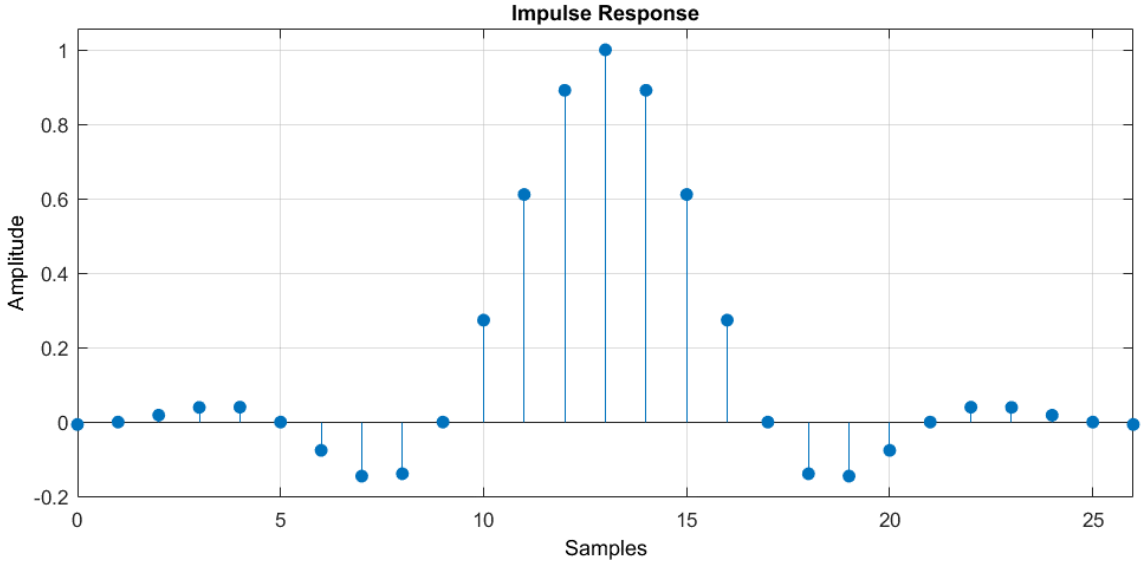

Fig. 3.2: Impulse Response of 30dB filter - Time Domain analysis

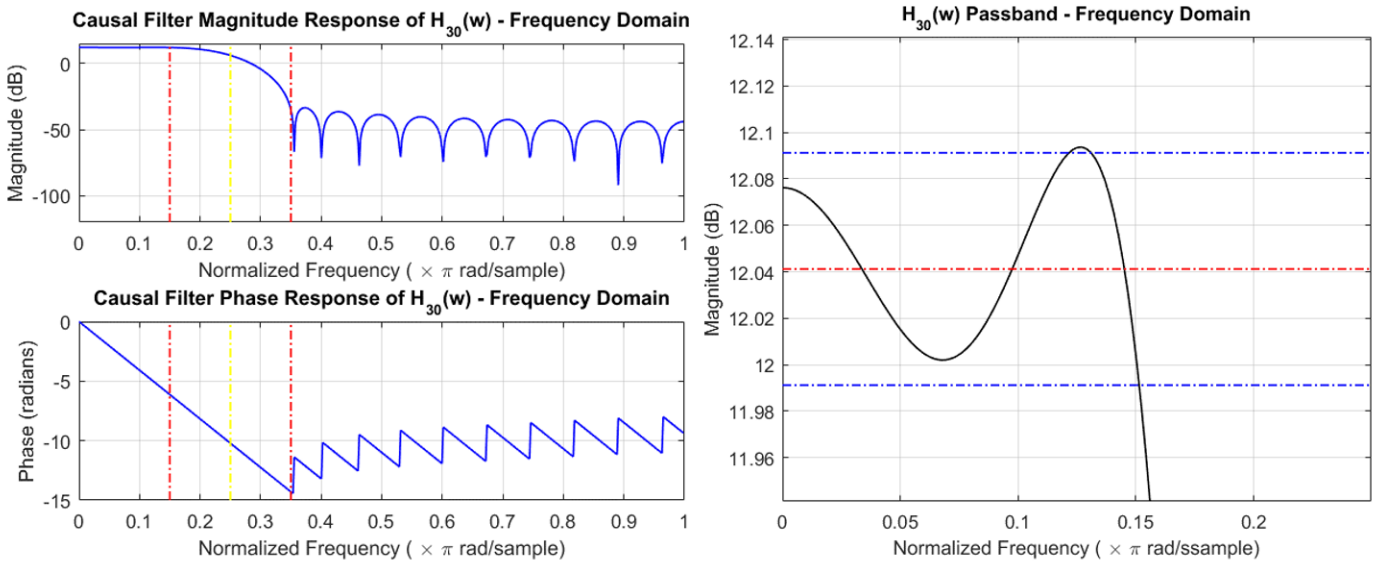

Fig. 3.3: Frequency Response (Left) and Passband of 30dB filter

By observing the figures provided, we can see that the designed filter satisfies the given specification on the minimum stopband attenuation but the passband ripple at the passband edge slightly (0.00006dB) goes beyond the maximum passband ripple specified. This is due to the low order that the Kaiser window returns for the given parameters and the transition width B_T being narrower for that order. Further the impulse response is symmetric around 13th sample and due to the odd filter length , we have a sample at the 13th time point.

3.1.2 Filter $\mathrm{H}_{60}(\mathrm{z})$

The set of plots obtained during the design stage of the filter $H_{60}(z)$ are presented in Fig. 3.4 to Fig. 3.6. The Impulse Response of the Kaiser Window for the derived parameters is presented in Fig. 3.4. The causal ideal impulse response and the frequency response of the digital filter obtained using the Kaiser Window is depicted

in Fig. 3.5 and Fig. 3.6 respectively. A zoomed in plot of the pass band of the signal is obtained in order to visually identify the pass band ripples as shown in the Fig. 3.6.

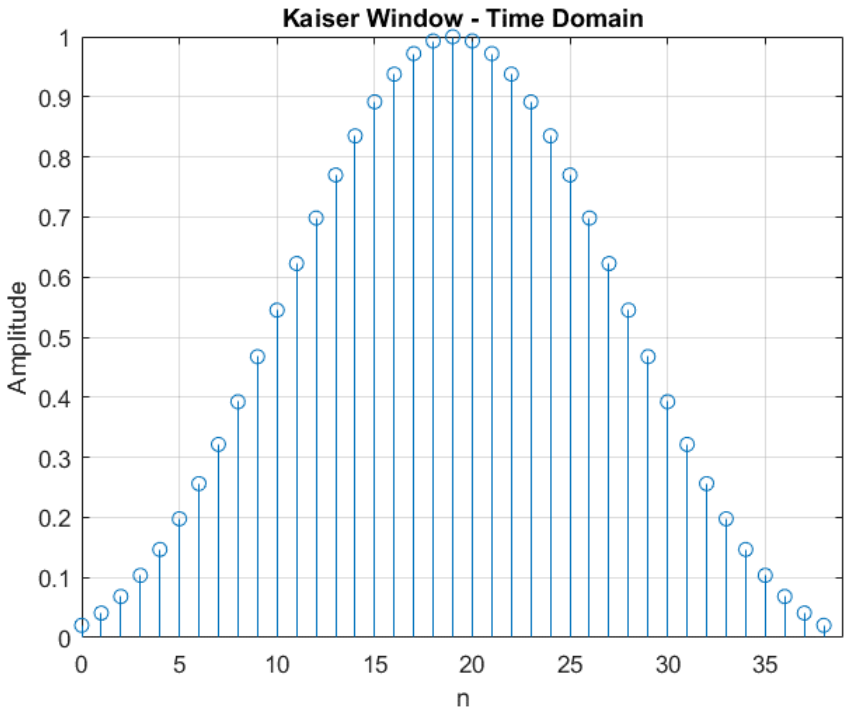

Fig. 3.4: Kaiser Window for 60dB filter - Time Domain

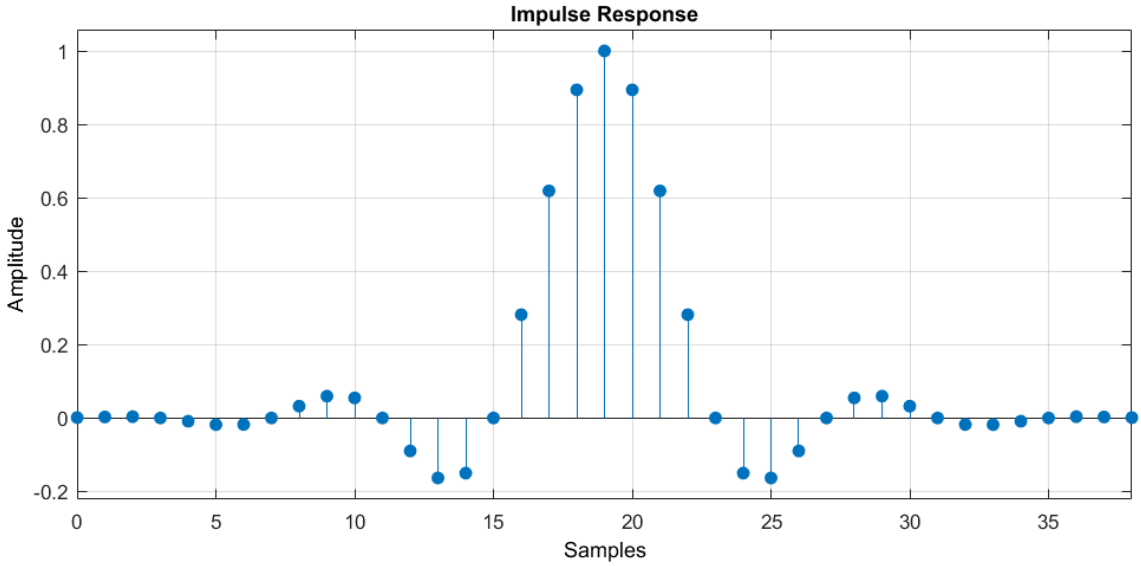

Fig. 3.5: Impulse response of 60dB filter - Time Domain analysis

By observing the figures provided, we can see that the designed filter satisfies the given specification on the maximum passband ripple as well as the minimum stopband attenuation which is 47.96dB compared to the passband gain of 12.04dB. Further the impulse response is symmetric around 19^{th} sample and due to the odd filter length , we have a sample at the 19^{th} time point.

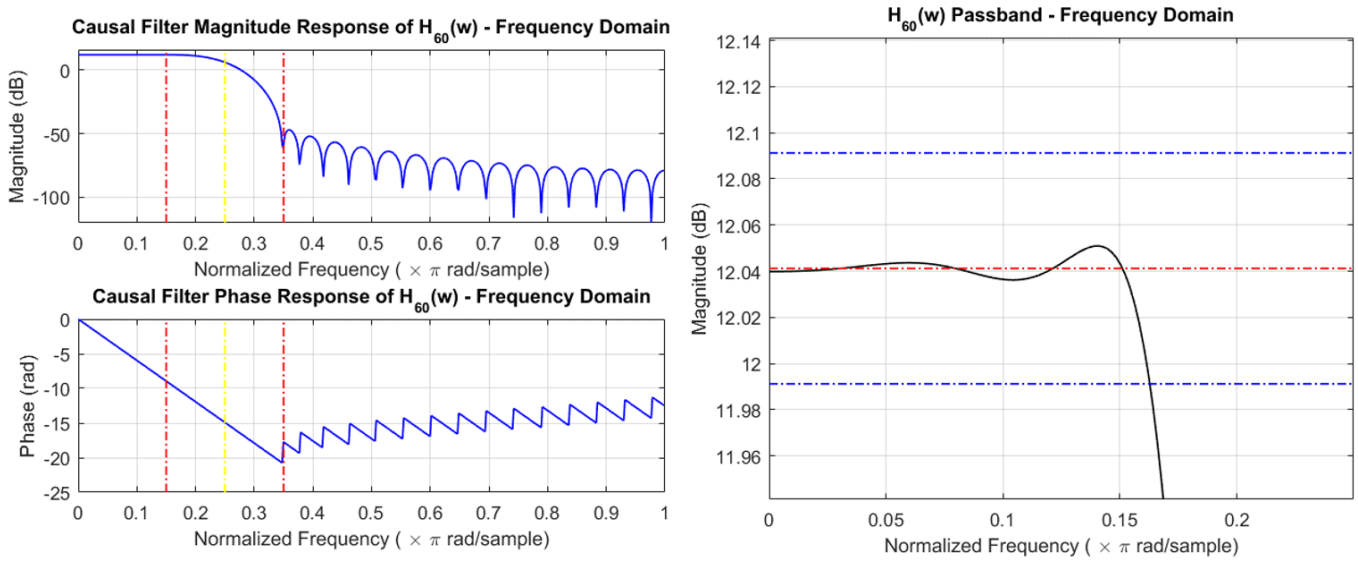

Fig. 3.6: Frequency Response (Left) and Passband of 60dB filter

3.2 Polyphase Filter Magnitude Spectra

In this section, the resultant filters using polyphase decomposition of the original filters are analyzed using their magnitude spectrums as shown.

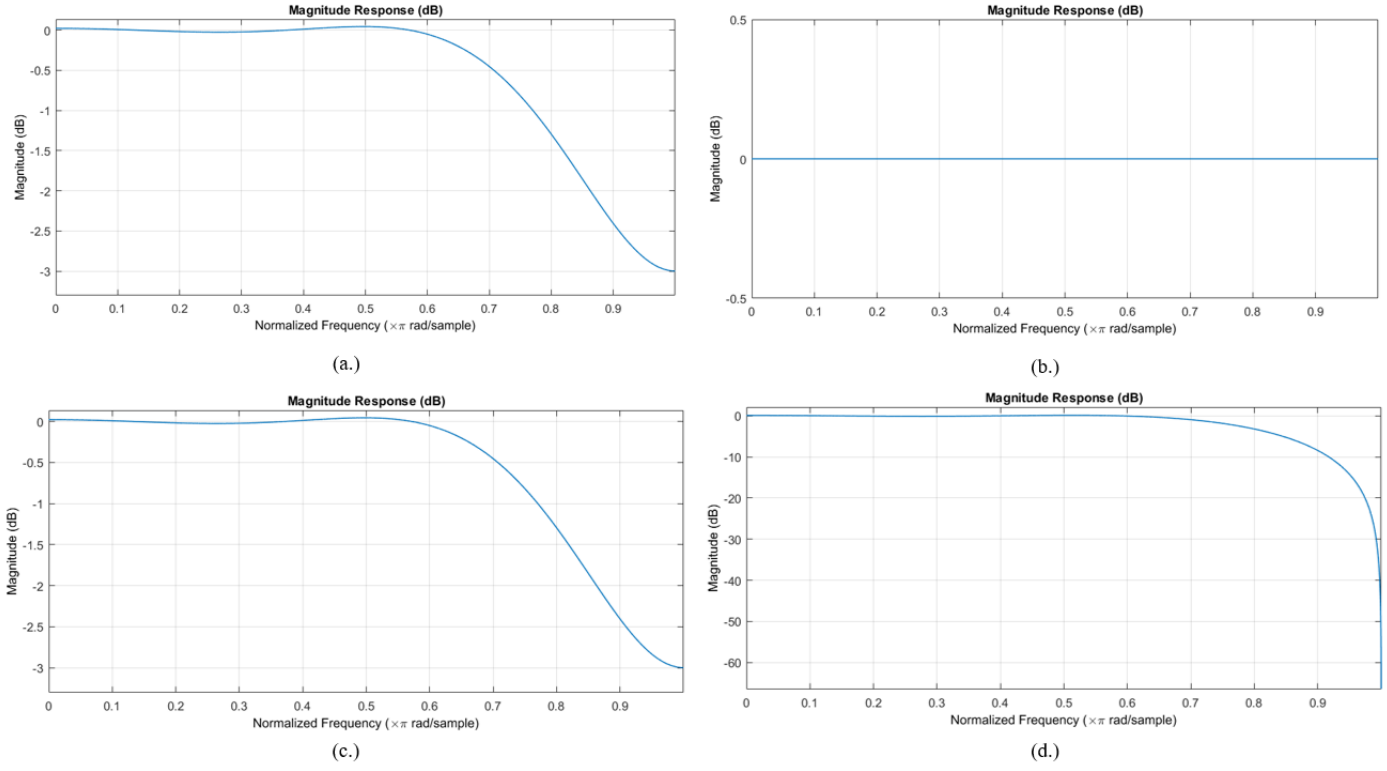

Fig. 3.7: Polyphase Filters of 30dB filter (a.) Poly-0 (b.) Poly-1 (c.) Poly-2 (d.) Poly-3

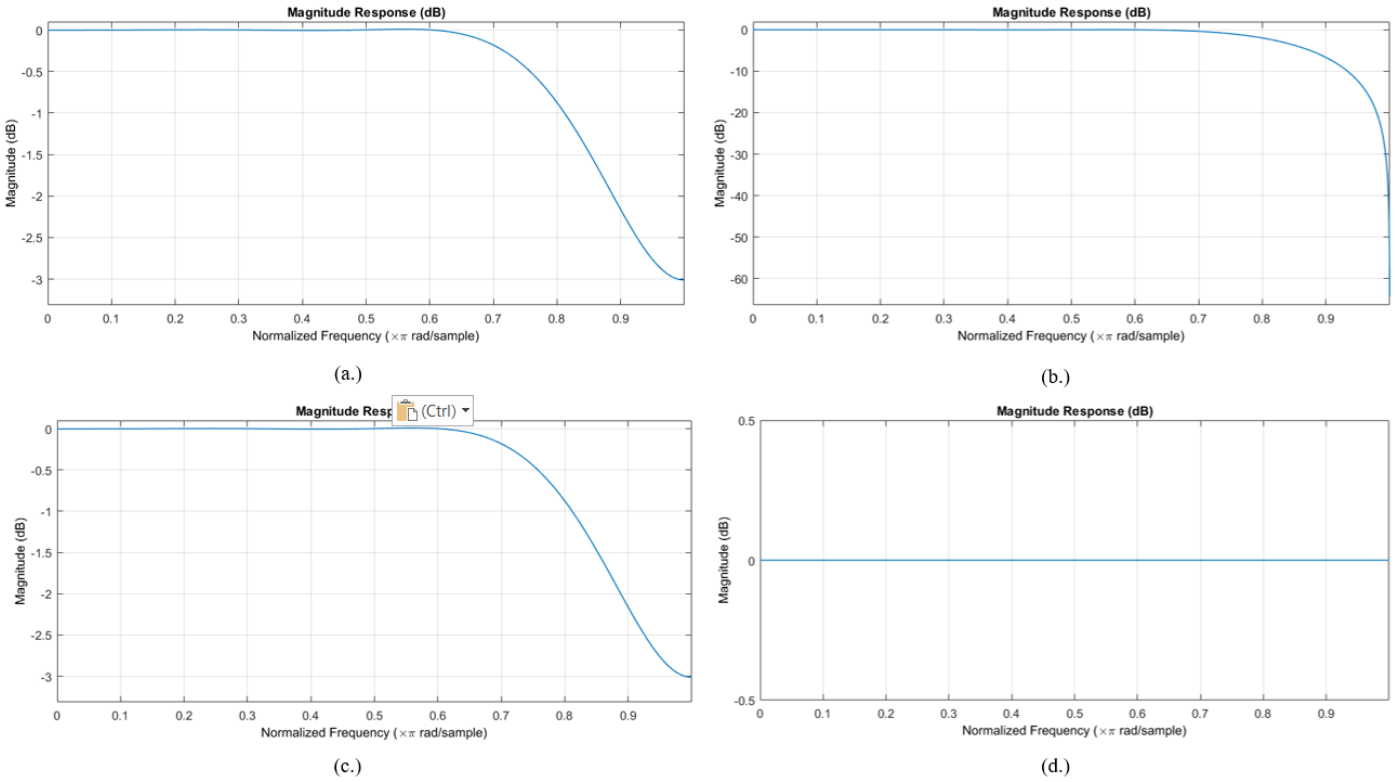

Fig. 3.8: Polyphase Filters of 60dB filter (a.) Poly-0 (b.) Poly-1 (c.) Poly-2 (d.) Poly-3

3.3 Filter Evaluation

In this section, the filters are evaluated in two ways:

a. Their performance to re-sample the sequence given ($x[n]$) by a factor of 4 so that it will be equivalent to sampling the continuous-time signal $x_c(t)$ at a sampling frequency of $800\pi\ rad/s$ $(400Hz)$ using the RMSE metric.
b. The computational complexity both in terms of original implementation vs. efficient implementation and 30dB filter vs. 60dB filter.

3.3.1 RMSE between the output sequences and reference sequence

As described in the Section 2 , the RMSE metric will be used to evaluate how well the implemented M-fold interpolator re-samples the given input signal by comparing it with the sequence $x_u[n]$.

First, the sequences $x[n]$, $u[n]$ and $y_i[n]$ are analyzed both in the time-domain and frequency-domain.

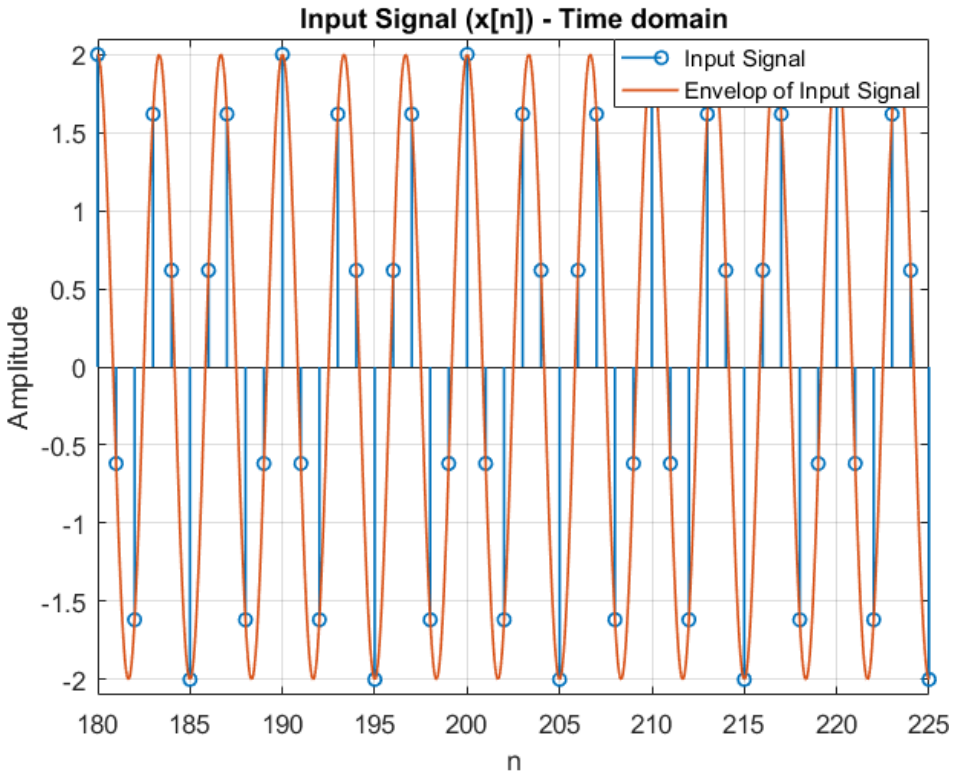

Fig. 3.9: Input Signal sampled at 100 Hz (x[n]) - Time Domain

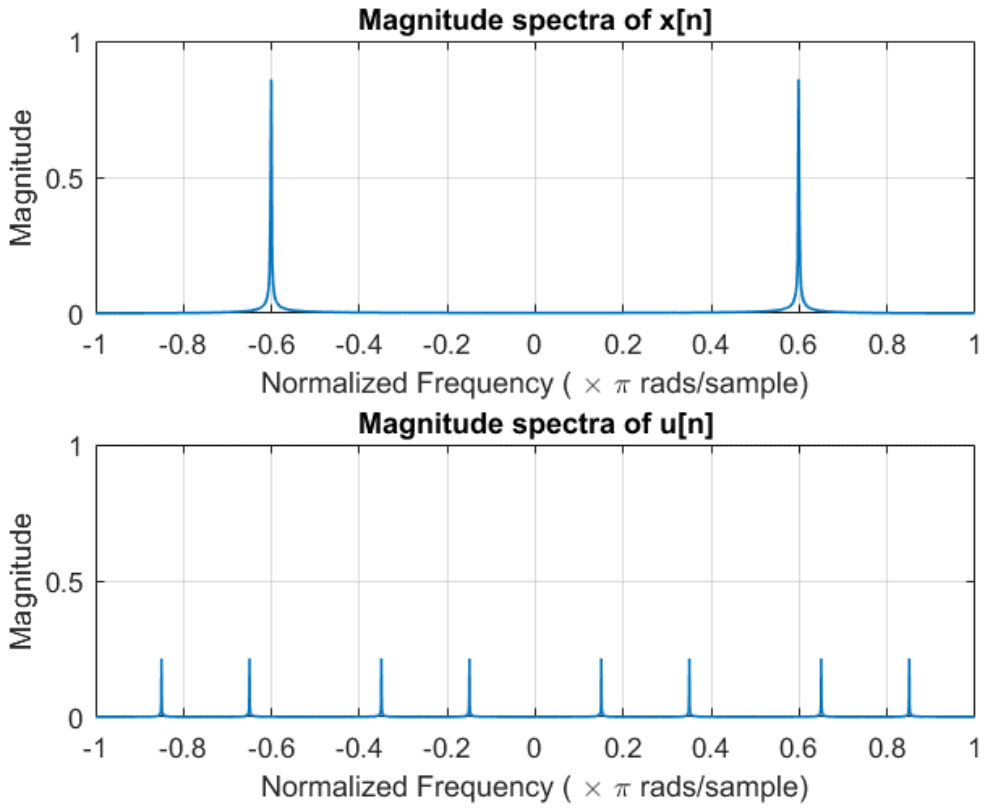

Fig. 3.10: Magnitude Spectrums of x[n] and u[n] (with 4004 samples)

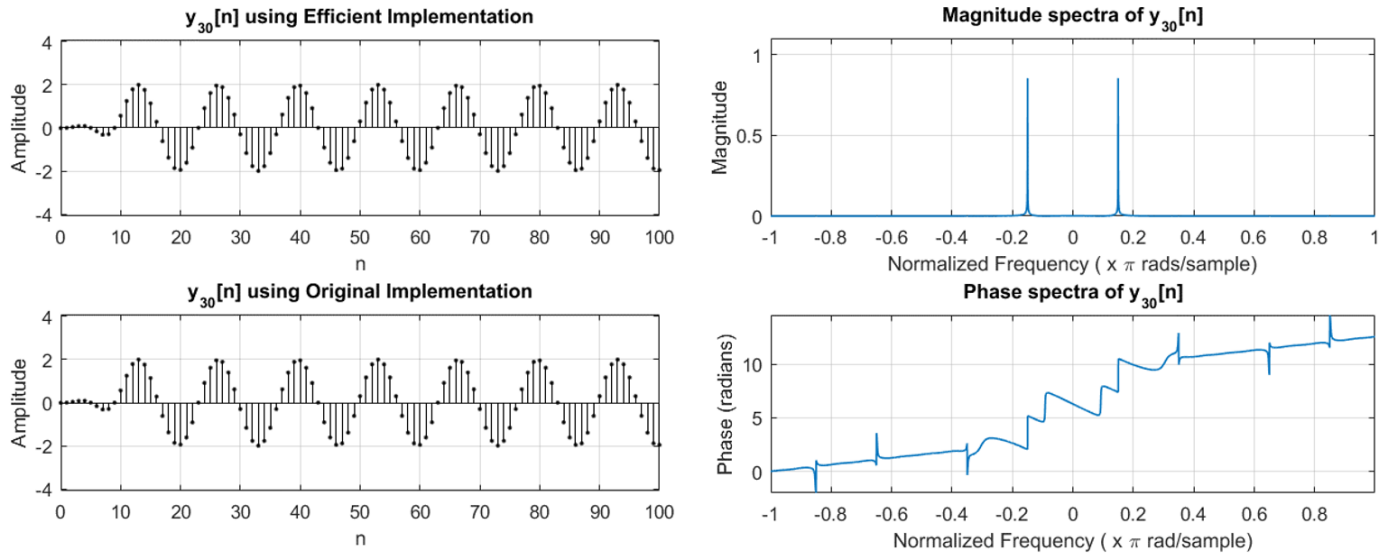

Fig. 3.11: Output sequence from 30dB Filter - Time Domain (left) and Frequency Domain (right)

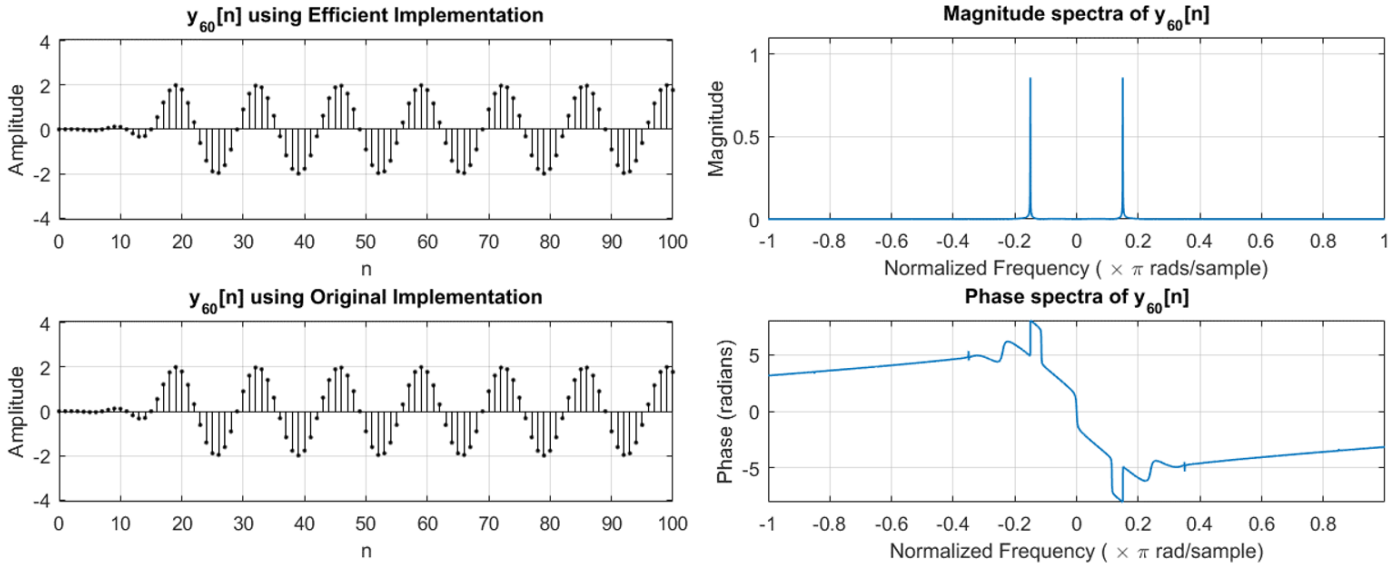

Fig. 3.12: Output sequence from 60dB Filter - Time Domain (left) and Frequency Domain (right)

We can note that in the time-domain analysis of the output signals, $y_i[n]$, distortion at the beginning is present due to the effects from the group delay by the two filters that are designed which corresponds to 13 and 19 samples, respectively. When comparing with the reference signal $x_u[n]$, first the effect from the group delays should be removed from $y_i[n]$s.

Next, the comparison with the reference signal $x_u[n]$ is presented. Fig. 3.13 shows the time-domain representation of the reference signal (only the samples from 180 to 225 was plotted for the visualization purpose). Fig. 3.14 and Fig. 3.15 shows the comparison of $y_i[n]$ with the reference signal by plotting them on top of each other. The figures show that the outputs obtained from the interpolators are much closer to the reference signal in the time-domain representation. Note that now the effect from the group delay is removed at the beginning of the output sequences. Hence, we can conclude that both the designed interpolators are re-sampling the input fairly good using the qualitative analysis that we can perform based on the figures.

Further to quantify the qualitatively obtained conclusion , the RMSE values were calculated as said earlier. Table 3.1 shows the results. The results re-establish the conclusion made qualitatively having a very low RMSE value between the output sequences and the reference sequences.

Table 3.1: RMSE values calculated

Filter	RMSE
$H_{30}(z)$	0.009256
$H_{60}(z)$	0.000542

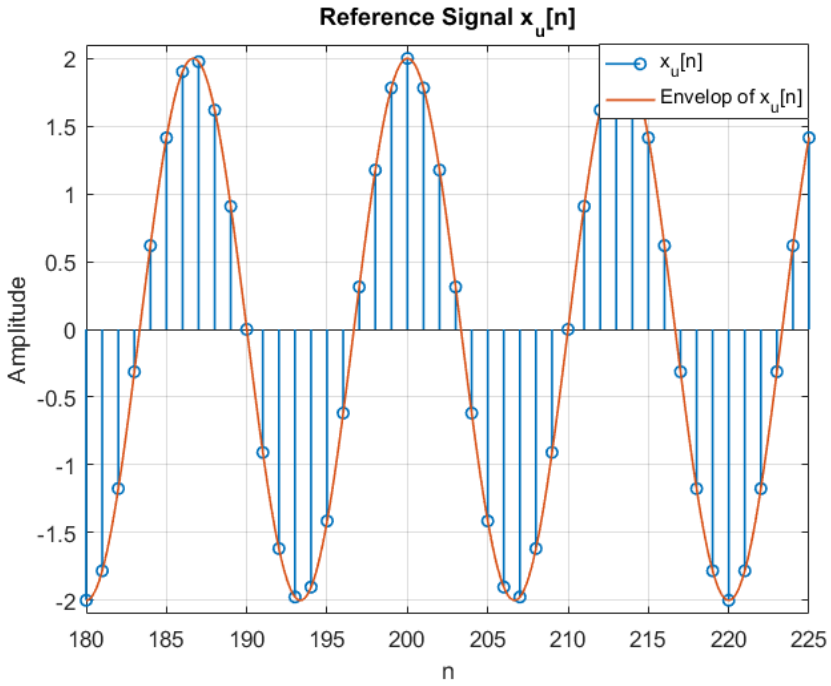

Fig. 3.13: Time Domain representation of the Reference Signal

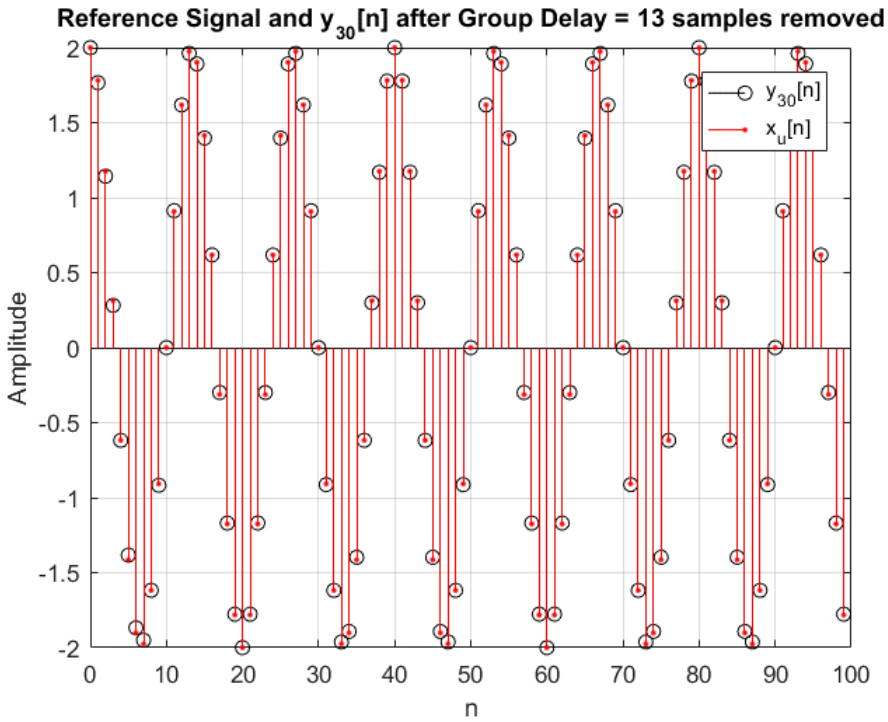

Fig. 3.14: Comparison of Output from 30dB filter and Reference Signal

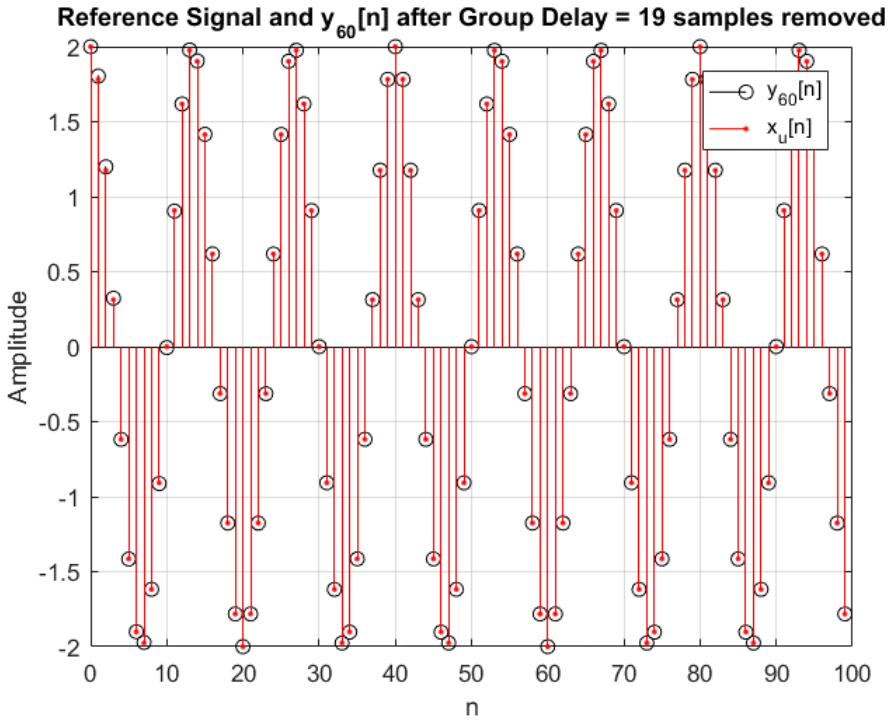

Fig. 3.15: Comparison of Output from 60dB filter and Reference Signal

3.3.2 Computational Complexity

Next the designed filters are evaluated based on the computation complexity they are having. It is done in terms of original implementation vs. efficient implementation of each of the filter and the $H_{30}(z)$ vs. $H_{60}(z)$ using their efficient implementations.

Table 3.2: Computation Details of the 30dB Filter

Specification	**Original Implementation**	**Efficient Implementation**				
		Poly-0	**Poly-1**	**Poly-2**	**Poly-3**	
Filter length	27	7	7	7	6	
Sequence Length per filter	4004	1001	1001	1001	1001	
No. of Multipliers	26	7	6	7	6	
No. of Adders	26	6	6	6	5	
No. of delay units	0	1	2	1	0	
Multiplications per Input sample	26	7	6	7	6	
Additions per Input sample	26	6	6	6	5	
Total no. of multiplications for input sequence	104104	7007	6006	7007	6006	26026
Total no. of additions for input sequence	104104	6006	6006	6006	5005	23023

Table 3.3: Computation Details of 60dB Filter

Specification	**Original Implementation**	**Efficient Implementation**				
		Poly-0	**Poly-1**	**Poly-2**	**Poly-3**	
Filter length	39	10	10	10	9	
Sequence Length per filter	4004	1001	1001	1001	1001	
No. of Multipliers	38	10	10	10	8	
No. of Adders	38	9	9	9	8	
No. of delay units	0	0	1	2	3	
Multiplications per Input sample	38	10	10	10	8	
Additions per Input sample	38	9	9	9	8	
Total no. of multiplications for input sequence	152152	10010	10010	10010	8008	38038
Total no. of additions for input sequence	152152	9009	9009	9009	8008	35035

Considering the original implementation and the efficient implementations for the filter $H_{30}(z)$, it can be noted that 104104 multiplications need to be performed to process the upsampled sequences while the polyphase structure in total has to perform only 26026 multiplications which is a 75% reduction. Further, the original implementation requires 104104 additions to be performed to process the upsampled sequence whereas 23023 which is a 77.88% reduction. Similarly, for the filter $H_{60}(z)$, 152152 multiplications need to be performed in original implementation while only 38038 multiplications need to be performed in the efficient structure that has led to 75% reduction. Comparing the number of additions to be performed, it follows a similar pattern where the computation complexity is reduced by 76.97% in the efficient implementation.

This leads to the conclusion that the efficient structure that was derived during the class has vastly reduced the computational complexity compared to the original implementation provided in the diagram.

When comparing the efficient implementations of $H_{60}(z)$ to the $H_{30}(z)$, the computational complexity has risen from 26026 to 38038 multiplications (46% increase for $H_{60}(z)$ compared to $H_{30}(z)$) and from 23023 to 35035 (52.2% increase for $H_{60}(z)$ compared to $H_{30}(z)$) during which the RMSE has reduced from 0.009256 to 0.000542 which is a 94% reduction for $H_{60}(z)$ compared to $H_{30}(z)$ This suggests, for an increase in one multiplication/addition in the system, only $(0.009256 - 0.000542)/(38038 - 26026) = 7.25 \times 10^{-7}$ of RMSE is reduced.

Thus the $H_{30}(z)$ provides acceptable RMSE with lower computation complexity compared to the $H_{60}(z)$.

4 CONCLUSION

The computational efficiency of implementing an M-fold Interpolator using the *polyphase structure* and *efficient implementation* is evident with approximately 75% reduction of the multiplications and the additions required to process and upsample the input sequence successfully. Also the two implementations (*original* and *efficient*) provide the similar outputs without any additional distortions apart from the group delay which is inherent to the designed digital filter and is not depending on the structure used to implement them. It can be seen , to obtain a higher stopband attenuation, higher order filters are required that increases the computational complexity with a very minimum reduction in the RMSE.

Further, the results depicts that the M-fold Interpolator is not only computationally efficient but also provides a very close representation of a sequence sampled at a sampling frequency M times higher than the sampling frequency used to obtain input sequence.

When designing the Anti-Imaging filters, the flexibility of the Kaiser Window has been incorporated. Since the ideal filters cannot be practically implemented, it is advantageous to be able to make a flexible filter of which the limitations can be controlled. This is a practical approach since small imperfections such as pass band ripple will not cause an observable difference in the filtered output. This means that the parameters of the filter can be adjusted until the differences between the filter output and an ideal output become indistinguishable for all the practical purposes.

5 REFERENCE

[1] Alan V. Oppenheim and Ronald W. Schafer. 2009. Discrete-Time Signal Processing (3rd. ed.). Prentice Hall Press, USA .

[2] Adams, M. D. (2013). Multiresolution signal and geometry processing: filter banks, wavelets, and subdivision (version: 2013-09-26). Michael Adams.

6 APPENDIX – MATLAB CODE

Initial Parameters

```
clc;
clear all;
close all;
set(groot,'defaultAxesXGrid','on')
set(groot,'defaultAxesYGrid','on')
set(0, 'DefaultLineLineWidth', 1.0);

OsI = 2*pi*100;                         % rad/s
TsI = 2*pi/OsI;                         % s
Omega0 = 2*pi*30;                       % rad/s

N = 1001;                               % samples
M = 4;                                  % upsample factor

% after upsampled parameters
Ts = TsI/M;
Os = OsI*M;
```

Input Signal

```
close all;
n = 0:N-1;                              % discrete time vector
t = 0:TsI:N-1;
xn = 2*cos(Omega0*TsI*n);
xt = 2*cos(Omega0*TsI*t);

figure
stem(n, xn, 'MarkerSize',5, 'LineWidth', 1.0); axis tight;
xlabel('n'); ylabel('Amplitude');

hold on;
plot(t,xt); axis tight;
axis([180 225 -2.1 2.1]);
legend('Input Signal','Envelop of Input Signal');

title('Input Signal (x[n]) - Time domain');

Xw = fftshift(fft(xn)/N);
fxn = [-N/2:N/2-1]*(2/N);

figure
subplot(2,1,1)
plot(fxn, abs(Xw)); axis tight;
xlabel('Normalized Frequency ( \times \pi rad/sample)'); ylabel('Magnitude');
axis([-1 1 0 1]);
title(strcat(['Magnitude Spectrum of Input Signal x[n]']));

subplot(2,1,2)
plot(fxn, unwrap(angle(Xw)));
xlabel('Normalized Frequency ( \times \pi rad/sample)'); ylabel('Phase (radians)');
title(strcat(['Phase Spectrum of Input Signal x[n]']));
```

FIR Filter Design

```
clc;
close all;

% Question 01 - Derived Specifications

PassGain = M; StopGain = 0;
CutOff = (pi/M)/Ts;                     % rad/s
BT = 0.2*pi/Ts;                         % rad/s
PassEdge = CutOff - BT/2;               % rad/s
StopEdge = CutOff + BT/2;               % rad/s
Norm = 2*pi;
Ap = 0.1;                               % Passband Ripple (dB)
Aa30 = 30;                              % Stopband attenuation 1 (dB)
Aa60 = 60;                              % Stopband attenuation 2 (dB)

cutoffs = [PassEdge/Norm StopEdge/Norm];    % Edge freqs (Hz)
gains = [1 StopGain];                       % Passband and Stopband gain

% Question 02 - Designing Even-Order Linear Anti-Imaging Filters
```

```
% Question 03 - Visualization of the filters

% H_30(z)
deltaP = (10^(0.05*Ap)-1)/(10^(0.05*Ap)+1);
deltaA = 10^(-0.05*Aa30);
dev30 = [deltaP deltaA];                    % Ripple (%)

[o30, w30, beta30, typ30] = kaiserord(cutoffs, gains, dev30, Os/Norm);
o30 = o30 + rem(o30, 2);

K30 = kaiser(o30 + 1, beta30);

figure
stem(0:o30,K30)
xlabel('n'); ylabel('Amplitude');
xlim([0 o30+1]);
title('Kaiser Window - Time Domain');

hnb30 = PassGain*fir1(o30, w30, typ30, K30, 'noscale'); % adjusting the gain
[H30, W30] = freqz(hnb30,1);

% Visulaizing the Magnitude and Phase responses of the filter H_30(W)
figure;

subplot(2,1,1);
Om = W30/pi;
logH30 = 20*log10(abs(H30));                % Magnitude of Filter in dB
plot(Om,logH30, '-b'); hold on;
axis([0 1 -120 15]);

vertLine = get(gca, 'YLim');
line( [PassEdge*(Ts/pi) PassEdge*(Ts/pi)], vertLine, 'Color','red','LineStyle','-.'); hold on;
line( [StopEdge*(Ts/pi) StopEdge*(Ts/pi)], vertLine, 'Color','red','LineStyle','-.'); hold on;
line( [CutOff*(Ts/pi) CutOff*(Ts/pi)], vertLine, 'Color','yellow','LineStyle','-.')

xlabel('Normalized Frequency ( \times \pi rad/sample)');
ylabel('Magnitude (dB)');
title(strcat(('Causal Filter Magnitude Response of H_{30}(w) - Frequency Domain')));

subplot(2,1,2);

PhaseH30 = unwrap(angle(H30));               % Phase of Filter
plot(Om,PhaseH30, '-b'); hold on;
axis([0 1 -15 0]);

vertLine = get(gca, 'YLim');
line( [PassEdge*(Ts/pi) PassEdge*(Ts/pi)], vertLine, 'Color','red','LineStyle','-.'); hold on;
line( [StopEdge*(Ts/pi) StopEdge*(Ts/pi)], vertLine, 'Color','red','LineStyle','-.'); hold on;
line( [CutOff*(Ts/pi) CutOff*(Ts/pi)], vertLine, 'Color','yellow','LineStyle','-.')

xlabel('Normalized Frequency ( \times \pi rad/sample)');
ylabel('Phase (radians)');
title(strcat(('Causal Filter Phase Response of H_{30}(w) - Frequency Domain')));

figure;
SF = 1; EF = round(CutOff);
w_pass = Om(SF:EF);
h_pass = logH30(SF:EF);
plot(w_pass,h_pass, '-k'); hold on;
plot(w_pass, ones(length(w_pass)) * (20*log10(M) + Ap/2),'-.b'); hold on;
plot(w_pass, ones(length(w_pass)) * (20*log10(M) - Ap/2),'-.b'); hold on;
plot(w_pass, ones(length(w_pass)) * 20*log10(M),'-.r');
axis([0,EF*(Ts/pi),20*log10(M)-0.1,20*log10(M)+0.1]);
xlabel('Normalized Frequency ( \times \pi rad/ssample)');
ylabel('Magnitude (dB)');
title('H_{30}(w) Passband - Frequency Domain')

fvtool(hnb30);

% H_60(z)

deltaP = (10^(0.05*Ap)-1)/(10^(0.05*Ap)+1);
deltaA = 10^(-0.05*Aa60);
dev60 = [deltaP deltaA];                    % Ripple (%)

[o60, w60, beta60, typ60] = kaiserord(cutoffs, gains, dev60, Os/Norm);
o60 = o60 + rem(o60, 2);

K60 = kaiser(o60 + 1, beta60);

figure
stem(0:o60,K60)
xlabel('n'); ylabel('Amplitude');
xlim([0 o60+1]);
title('Kaiser Window - Time Domain');
```

```
hnb60 = PassGain*fir1(o60, w60, typ60, K60, 'noscale');
[H60, W60] = freqz(hnb60,1);

% Visulaizing the Magnitude and Phase responses of the filter H_30(w)
figure;

subplot(2,1,1);
Om = W60/pi;
logH60 = 20*log10(abs(H60));                % Magnitude of Filter in dB
plot(Om,logH60, '-b'); hold on;
axis([0 1 -120 16]);

vertLine = get(gca, 'YLim');
line( [PassEdge*(Ts/pi) PassEdge*(Ts/pi)], vertLine, 'Color','red','LineStyle','-.'); hold on;
line( [StopEdge*(Ts/pi) StopEdge*(Ts/pi)], vertLine, 'Color','red','LineStyle','-.'); hold on;
line( [CutOff*(Ts/pi) CutOff*(Ts/pi)], vertLine, 'Color','yellow','LineStyle','-.')

xlabel('Normalized Frequency ( \times \pi rad/sample)');
ylabel('Magnitude (dB)');
title(strcat(('Causal Filter Magnitude Response of H_{60}(w) - Frequency Domain')));
```

```
subplot(2,1,2);

PhaseH30 = unwrap(angle(H60));                % Phase of Filter
plot(Om,PhaseH30, '-b'); hold on;
axis([0 1 -25 0]);

vertLine = get(gca, 'YLim');
line( [PassEdge*(Ts/pi) PassEdge*(Ts/pi)], vertLine, 'Color','red','LineStyle','-.'); hold on;
line( [StopEdge*(Ts/pi) StopEdge*(Ts/pi)], vertLine, 'Color','red','LineStyle','-.'); hold on;
line( [CutOff*(Ts/pi) CutOff*(Ts/pi)], vertLine, 'Color','yellow','LineStyle','-.')


xlabel('Normalized Frequency ( \times \pi rad/sample)');
ylabel('Phase (rad)');
title(strcat(('Causal Filter Phase Response of H_{60}(w) - Frequency Domain')));

figure;
SF = 1; EF = round(CutOff);
w_pass = Om(SF:EF);
h_pass = logH60(SF:EF);
plot(w_pass,h_pass, '-k'); hold on;
plot(w_pass, ones(length(w_pass)) * (20*log10(M) + Ap/2),'-.b'); hold on;
plot(w_pass, ones(length(w_pass)) * (20*log10(M) - Ap/2),'-.b'); hold on;
plot(w_pass, ones(length(w_pass)) * 20*log10(M),'-.r');
axis([0,EF*(Ts/pi),20*log10(M)-0.1,20*log10(M)+0.1]);
xlabel('Normalized Frequency ( \times \pi rad/sample)');
ylabel('Magnitude (dB)');
title('H_{60}(w) Passband - Frequency Domain')

fvtool(hnb60);
```

Computation of Interpolated Signals

```
clc;
close all;

InterL = 4*N;                   % NOT SURE Currently its at 4004
un = upsample(xn, M);
un = un(1:InterL);
n = 0:InterL-1;

% H_30 Efficient Implementation

% Polyphase filters (type-I)
h30 = hnb30(1:M:end);           % 0 - Polyphase
h31 = hnb30(2:M:end);           % 1 - Polyphase
h32 = hnb30(3:M:end);           % 2 - Polyphase
h33 = hnb30(4:M:end);           % 3 - Polyphase

fvtool(h30,1);
fvtool(h31,1);
fvtool(h32,1);
fvtool(h33,1);

% Filtered Polyphase components
f30 = filter(h30, 1, xn);           % Filtered input with 0 - Polyphase
```

```
f31 = filter(h31, 1, xn);           % Filtered input with 1 - Polyphase
f32 = filter(h32, 1, xn);           % Filtered input with 2 - Polyphase
f33 = filter(h33, 1, xn);           % Filtered input with 3 - Polyphase

% Upsampled sub-bands
y30 = upsample(f30, M);             % Upsampled output (samples 4004 with 3 0s added at the end)
y31 = upsample(f31, M);
y32 = upsample(f32, M);
y33 = upsample(f33, M);

% Delayed sub-bands
y30 = [zeros(1,0) y30];
y31 = [zeros(1,1) y31];
y32 = [zeros(1,2) y32];
y33 = [zeros(1,3) y33];

% Final Output from the efficient structure
y30n = zeros(1, InterL);

y30n = y30n + y30(1:InterL);
y30n = y30n + y31(1:InterL);
y30n = y30n + y32(1:InterL);
y30n = y30n + y33(1:InterL);

y30ref = filter(hnb30, 1, un);

figure
subplot(2,1,1)
stem(n, y30n, 'k.'); axis tight;
xlabel('n'); ylabel('Amplitude');
axis([0 100 -4.1 4.1]);
title('y_{30}[n] using Efficient Implementation');

subplot(2,1,2)
stem(n, y30ref, 'k.'); axis tight;
xlabel('n'); ylabel('Amplitude');
axis([0 100 -4.1 4.1]);
title('y_{30}[n] using Original Implementation');

% H_60 Efficient Implementation

% Polyphase filters (type-I)
h60 = hnb60(1:M:end);           % 0 - Polyphase
h61 = hnb60(2:M:end);           % 1 - Polyphase
h62 = hnb60(3:M:end);           % 2 - Polyphase
h63 = hnb60(4:M:end);           % 3 - Polyphase

fvtool(h60,1);
fvtool(h61,1);
fvtool(h62,1);
fvtool(h63,1);

% Filtered Polyphase components
f60 = filter(h60, 1, xn);           % Filtered input with 0 - Polyphase
f61 = filter(h61, 1, xn);           % Filtered input with 1 - Polyphase
f62 = filter(h62, 1, xn);           % Filtered input with 2 - Polyphase
f63 = filter(h63, 1, xn);           % Filtered input with 3 - Polyphase

% Upsampled sub-bands
y60 = upsample(f60, M);             % Upsampled output (samples 4004 with 3 0s added at the end)
y61 = upsample(f61, M);
y62 = upsample(f62, M);
y63 = upsample(f63, M);

% Delayed sub-bands
y60 = [zeros(1,0) y60];
y61 = [zeros(1,1) y61];
y62 = [zeros(1,2) y62];
y63 = [zeros(1,3) y63];

% Final Output from the efficient structure
y60n = zeros(1, InterL);

y60n = y60n + y60(1:InterL);
y60n = y60n + y61(1:InterL);
y60n = y60n + y62(1:InterL);
y60n = y60n + y63(1:InterL);

y60ref = filter(hnb60, 1, un);

figure
subplot(2,1,1)
stem(n, y60n, 'k.'); axis tight;
xlabel('n'); ylabel('Amplitude');
axis([0 100 -4.1 4.1]);
```

```
title('y_{60}[n] using Efficient Implementation');

subplot(2,1,2)
stem(n, y60ref, 'k.'); axis tight;
xlabel('n'); ylabel('Amplitude');
axis([0 100 -4.1 4.1]);
title('y_{60}[n] using Original Implementation');
```

Question 5 - Visualizing the Spectra

```
clc;
close all;

ff = [-N/2:N/2-1]*(2/N);
fff = [-InterL/2:InterL/2-1]*(2/InterL);

Uw = fftshift(fft(un)/InterL);
Y3w = fftshift(fft(y30n)/InterL);
Y6w = fftshift(fft(y60n)/InterL);

% Visualizing the magnitude spectra of x[n] and u[n]
figure;
subplot(2,1,1);
plot(ff, abs(Xw)); axis tight;
xlabel('Normalized Frequency ( \times \pi rads/sample)'); ylabel('Magnitude');
axis([-1 1 0 1]);
title(strcat(['Magnitude spectra of x[n]']));

subplot(2,1,2);
plot(fff, abs(Uw)); axis tight;
axis([-1 1 0 1]);
xlabel('Normalized Frequency ( \times \pi rads/sample)'); ylabel('Magnitude');
title(strcat(['Magnitude spectra of u[n]']));

% Visualizing the y_30[n] - magnitude and phase spectra
figure
subplot(2,1,1)
plot(fff, abs(Y3w)); axis tight;
axis([-1 1 0 1.1]);
xlabel('Normalized Frequency ( x \pi rads/sample)'); ylabel('Magnitude');
title(strcat(['Magnitude spectra of y_{30}[n]']));

subplot(2,1,2)
plot(fff, unwrap(angle(Y3w))); axis tight;
xlabel('Normalized Frequency ( x \pi rad/sample)'); ylabel('Phase (radians)');
title(strcat(['Phase spectra of y_{30}[n]']));

% Visualizing the y_60[n] - magnitude and phase spectra
figure
subplot(2,1,1)
plot(fff, abs(Y6w)); axis tight;
axis([-1 1 0 1.1]);
xlabel('Normalized Frequency ( \times \pi rads/sample)'); ylabel('Magnitude');
title(strcat(['Magnitude spectra of y_{60}[n]']));

subplot(2,1,2)
plot(fff, unwrap(angle(Y6w))); axis tight;
xlabel('Normalized Frequency ( \times \pi rad/sample)'); ylabel('Phase (radians)');
title(strcat(['Phase spectra of y_{60}[n]']));
```

```
clc;
close all;

n = 0:1:InterL;
t = 0:Ts:InterL;

xun = 2*cos(Omega0*Ts*n);
xut = 2*cos(Omega0*Ts*t);

figure
stem(n, xun, 'LineWidth',1, 'MarkerSize',5); axis tight; hold on;
plot(t, xut);
xlabel('n'); ylabel('Amplitude');
axis([180 225 -2.1 2.1]);
legend('x_u[n]', 'Envelop of x_u[n]');
title('Reference Signal x_u[n]');

grpD3 = mean(grpdelay(hnb30));
grpD6 = mean(grpdelay(hnb60));

% Comparison of x_u[n] and y_30[n]
xun30 = xun(1:end-grpD3);                       % Removing the group delay
```

```
y3D = y30n;
y3D(1:grpD3) = [];
n3 = n(1:end-grpD3);

figure;
stem(n3(1:100), y3D(1:100), 'k'); hold on;
stem(n3(1:100), xun30(1:100), 'r.');
xlabel('n'); ylabel('Amplitude');
legend('y_{30}[n]','x_u[n]');
title(sprintf('Reference Signal and y_{30}[n] after Group Delay = %d samples removed',grpD3));

% Comparison of x_u[n] and y_60[n]
xun60 = xun(1:end-grpD6);                       % Removing the group delay
y6D = y60n;
y6D(1:grpD6) = [];
n6 = n(1:end-grpD6);

figure;
stem(n6(1:100), y6D(1:100), 'k'); hold on;
stem(n6(1:100), xun60(1:100), 'r.');
xlabel('n'); ylabel('Amplitude');
legend('y_{60}[n]','x_u[n]');
title(sprintf('Reference Signal and y_{60}[n] after Group Delay = %d samples removed',grpD6));

% RMSE comparison

% for y_30[n]
comp_xun30 = xun30(1000:3000);
comp_y30n = y3D(1000:3000);
RMSE30 = sqrt(mean((comp_xun30 - comp_y30n).^2));
fprintf('RMSE comparing x_u[n] and y_{30}[n] after adjusting group delay = %f\n', RMSE30);

% for y_60[n]
comp_xun60 = xun60(1000:3000);
comp_y60n = y6D(1000:3000);
RMSE60 = sqrt(mean((comp_xun60 - comp_y60n).^2));
fprintf('RMSE comparing x_u[n] and y_{60}[n] after adjusting group delay = %f', RMSE60);
```

Published with MATLAB® R2016a