Synchronous Sampling Rate Conversion using Frequency domain based techniques

Chunduri SreenivasaRao[1], A. AroakiaRaj[2] and Dhulipalla VenkataRao[3]

[1] *Software Development Engineer, AMD India Pvt Ltd, Hyderabad, India.*
[2] *Software Development Engineer, QualComm Pvt Ltd, Hyderabad, India.*
[3]*Principal, Narsaraopet Institute of Technology, Guntur (Dist), A.P., India*
chsrinivas19800305@rediffmail.com, arock0405@gmail.com,
dvenky221101@rediffmail.com

Abstract

Sampling rate conversion is, in general, implemented using time domain based poly-phase filter structures. The other way of implementing multi-rate signal processing is frequency domain based approach, which has the advantage of computational savings. This paper clearly explains how to apply frequency domain processing techniques for sampling rate conversion. The proposed approach is applied to both power of 2 and non-power of 2 sampling rate conversion factors. The properties of FFT are utilized analytically to solve the implementation problems such as non-power of 2 sampling rate factors. The theoretical computational complexity of the proposed approach is provided. The simulation results of proposed approach are compared with the quality of time domain approach and the comparison shows that differences are insignificant.

Keywords: *Complex FFT, Computational complexity, Decimation, Interpolation, Sampling Rate Conversion, Overlap save method*

1. Introduction

In many practical applications of signal processing, it is desired to convert the sampling frequency of signals due to the variation in operating bandwidth of various systems. The conversion could be either increase or decrease in sampling frequency depending on the system requirement. Sampling rate conversion (SRC) can be achieved in two ways. The signal could be converted to analog signal at input sampling frequency and resampled using analog to digital conversion at the desired sampling rate. This method has some disadvantages in terms of quantization error and signal distortion that occur during analog to digital conversion and vice versa. The other way of achieving sampling rate conversion is using digital domain [1-3].

In synchronous SRC, there exists an integer relationship between input and output clock rates, whereas in asynchronous SRC, it is the fractional relationship. In decimation, the input signal is filtered at the desired sampling rate before applying decimation factor to avoid aliasing effects. In case of interpolation, zeros are inserted between two successive samples of input signal and then filtered to avoid the image noise. The number of zeros to be inserted depends on the interpolation factor. Typically this value will be interpolation factor minus one. [1-3].

For efficient implementation of sampling rate conversion, one usually prefers time domain based poly-phase filtering structures where the redundant MAC (Multiply and ACcumulate) operations are eliminated. In decimation process, only the selected

samples based on decimation factor will be transmitted and hence filtering is required for the selected samples. In case of interpolation, the MAC operation result becomes vanish for the inserted zeros. In both methods, the time delay line filter (history buffer) is updated with appropriate contents of the input signal without losing any samples [1-3].

In [4], Ricky Lyons explained about the optimization procedure using the symmetric nature of filter coefficients. Because of symmetry, the memory to store filter coefficients becomes half. The contents of delay line filter are read from start and end simultaneously and added. The sum is multiplied with appropriate filter coefficient. With this, the number of multiplications could be reduced by half but requires more instructions to access the delay buffer contents.

The techniques explained by Candan [7], V. Valimaki [8], J. Vesma and T. Saramaki [9] are suitable for time domain based fractional delay interpolation and these are meant for asynchronous SRC.

In this paper, the frequency domain mathematical background is analyzed for synchronous SRC cases and implementation in frequency domain is explained for decimation, interpolation. For implementing these methods, overlap save method was used. A MATLAB simulator code is developed based on the proposed approach and simulation results are provided in Appendix. For the convenient of explanation, the figures in reference [1] are reused in this paper.

This paper is organized as follows. Sections 2 and 3 explain frequency domain based implementation for decimation and interpolation respectively. It contains implementation procedure and theoretical computational complexity details. Section 4 provides MATLAB code details, simulation results. Section 5 explains conclusion and future updates. MATLAB code is provided in Appendix.

2. Frequency Domain based Decimation

As shown in Figure 1, decimation process contains filtering of input signal *x(n)* with impulse response *h(n)* followed by decimation process [1]. Let *v(n)*, *y(n)* be the filtered & decimated outputs and let $V(\omega_x)$, $Y(\omega_y)$ be their continuous frequency domain equivalents respectively.

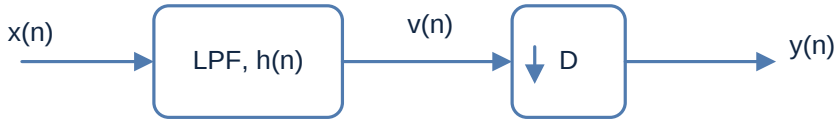

Figure 1. Decimation Process

The frequency domain outputs can be expressed as

$$V(\omega_x) = X(\omega_x)H(\omega_x) \quad (1)$$

$$Y(\omega_y) = \frac{1}{D}\sum_{i=0}^{D-1} V\left(\frac{\omega_x - 2\pi i}{D}\right) \quad (2)$$

where ω_x and ω_y are the continuous frequency variables of *x(n)* and *y(n)* respectively[1].

The above equation could be understood in better way by expanding equation (2) for D = 2 and D = 3, for example.

When D = 2,

$$Y(\omega) = \frac{1}{2}\left[V\left(\frac{\omega}{2}\right) + V\left(\frac{\omega}{2} - \pi\right)\right] \quad (3)$$

Similarly, when D = 3,

$$Y(\omega_y) = \frac{1}{3}\left[V\left(\frac{\omega_x}{3}\right) + V\left(\frac{\omega_x - 2\pi}{3}\right) + V\left(\frac{\omega_x - 4\pi}{3}\right)\right] \quad (4)$$

In equations (3) and (4), the first term in brackets *i.e.*, $V(\omega_x/2)$ or $V(\omega_x/3)$ is downsampled component of the filtered signal and the remaining terms are aliased components. The frequency domain equivalent of decimated output, is, thus, equal to the sum of the downsampled and the aliased components, divided by the decimation factor.

The variation of continuous frequency variable ω_x from 0 to 2π is equivalent to the variation from 0 to $N-1$ in discrete domain. When ω_x is varying from 0 to 2π, the decimated frequency variable ω_y varies from 0 to $2\pi/D$. In other words, discrete equivalent of ω_y varies from 0 to $N/D-1$. It can be expressed, mathematically in discrete domain, as

$$V(k) = X(k)H(k), \qquad k = 0,1,...N-1 \quad (5)$$

$$Y(k) = \frac{1}{D}\sum_{i=0}^{D-1} V\left(k + \frac{Ni}{D}\right), \qquad k = 0,1,2..., N/D - 1 \quad (6)$$

As the length of discrete decimated frequency variable, *k*, is *N/D*, it is sufficient to apply FFT size of *N/D* while evaluating the IFFT of decimated frequency output.

2.1. Frequency Domain based Implementation of Decimation

The implementation procedure is explained in Figure 2 for decimation factor of 2. Usually in time –domain, the input buffer length is chosen based on the length of filter coefficients and the output frame length. The necessary condition is that input frame length must be equal to decimation factor multiplied by output frame length. For example, if output frame length is 64 and decimation factor is 2, input frame length is 128. In case of overlap save method, the input FFT length is selected as $N = L+M-1$ where L is processing frame size and M is the filter length. For this case, input buffer length becomes N = 128 + 128 -1 = 256 (as N must be a power of 2 due to FFT usage) with the assumption that filter length is 128.

Initially, all input and output buffer contents are zeros. Frame1 (128 samples) will be filled in input buffer. The FFT will be calculated for 256 length input buffer. Also FFT coefficients will be calculated with the same length of 256 at one-time and used for each frame. The two complex FFTs are multiplied. Now the first half of the resultant buffer contains the downsampled frequency contents and the 2nd half contains the aliased downsampled frequency contents. These two buffers are added (as per equation (6))and IFFT will be applied to the complex frequency sum. The IFFT length becomes

(*1/D*) times input FFT length due to addition of downsampled and aliased contents. The scaling factor *1/D* (from equation (6)) should be applied to IFFT output. From the resultant IFFT output, only *L/D* (*L* is the input time domain length) samples are transmitted as final decimation output.

When 2nd frame arrives, it will be filled in 2nd half of the input buffer without disturbing the 1st frame. Similarly, 3rd frame will be filled in 1st half of the buffer with 2nd frame retained as it is. After frequency domain process, the decimated buffer (*L/D* samples) depends on which portion of input buffer was filled for processing.

The above procedure is common for any decimation factor. But in general, the frame lengths could be selected as power of 2 due to the usage of FFT. If decimation factor is not a power of 2, this could cause in making FFT length as power of 2. For example, if decimation factor is 3 and output FFT buffer length is 128, input buffer FFT length becomes 384, which is not a power of 2. This causes issue while calculating input buffer FFT. In Appendix-A, a method was given how to find out FFT for non-power of 2 lengths.

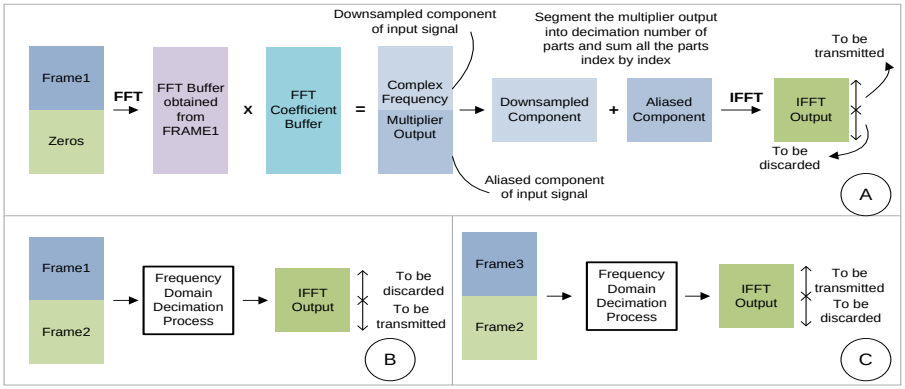

Figure 2. Frequency Domain Implementation of Decimation. The Decimation Factor here is D=2. For other Decimation Factors, the Buffer Sizes will Vary Accordingly

2.2. Computational Complexity of Frequency Domain based Decimation

Table 1. Theoretical Computational Complexity of Decimation using Overlap Save Method. Here O(N) = $N\log_2 N$

To Calculate	Complex Multiplications	Complex Additions	Remarks
X(k)	0.5 O(N)	O(N)	Complexity for calculating X(k) for each frame using FFT of length N.
V(k)	N	-	Complex Frequency Multiplication using equation (10)
Y(k)	-	N(D-1)/D	Complex frequency addition of downsampled component and aliasing components
y(n)	0.5 O(N/D)	O(N/D)	IFFT calculation of Y(k) . Here IFFT length is N/D.
Total	N+0.5[O(N)+O(N/D)]	N(D-1)/D + O(N) + O(N/D)	

The computational complexity details described in above table are valid if decimation factor is a power of 2. For non-power of 2 decimation factors, computational complexity is increased and is described in Appendix A.

3. Frequency Domain based Interpolation

As depicted in Figure 3, interpolation process contains insertion of (*I -1*) zeros between successive samples of input signal and filtering with impulse response *h(n)* for an interpolation factor of I [1].

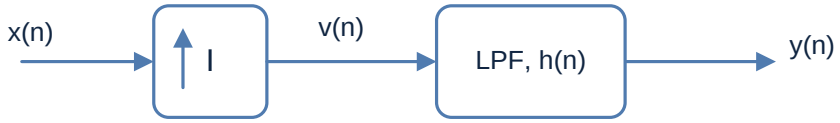

Figure 3. Interpolation Process

For signal notation of Figure 3, one could easily show that

$$V(\omega_y) = X(I\omega_y) \quad (7)$$

$$Y(\omega_y) = V(\omega_y)H(\omega_y) \quad (8)$$

where ω_x and ω_y are the continuous frequency variables of *x(n)* and *y(n)* respectively[1]. The relation $\omega_{y=}\ \omega_x/I$ holds good.

If the frequency variable ω_x varies from 0 to $2\pi/I$, then ω_y varies from 0 to 2π. The spectrum $V(\omega_y)$ contains $(I-1)$ images of spectrum $X(\omega_x)$ along with actual spectrum of *x(n)*. If *X(k)*, k = 0 to $N/I-1$ is the discrete input spectrum, discrete equivalent of $V(\omega_y)$ is obtained by copying *X(k)* in *I* times. In general, this can be expressed as

$$V\left(k + \frac{iN}{I}\right) = X(k) \quad (9)$$

where $k = 0,1,..N/I-1$ and $i = 0, 1, 2,\ldots I-1$. For $I = 4$, this could become

$$V(k) = X(k) \quad (10)$$

$$V\left(k + \frac{N}{4}\right) = X(k) \quad (11)$$

$$V\left(k + \frac{N}{2}\right) = X(k) \quad (12)$$

$$V\left(k + \frac{3N}{4}\right) = X(k) \quad (13)$$

where $k = 0,1,..N/4-1$. The discrete equivalent of filtered output spectrum is simply obtained by multiplying this spectrum with that of $H(\omega_y)$.

$$Y(k) = V(k)H(k) \quad (14)$$

where $k = 0$ to $N-1$. The FFT size should be *N* while evaluating filtered output using IFFT.

3.1. Frequency Domain Implementation of Interpolation

Figure 4 provides the details of frequency domain implementation for interpolation. The selection of input and output buffer lengths is quite opposite from decimation process. Here the output buffer length is interpolation factor times the input buffer length. For interpolation factor of 2, input FFT length becomes 128 if output FFT length is 256. The corresponding input and output time domain lengths are 64 and 128 respectively.

Initially I/O buffers contain zeros. Input Frame 1 (64 samples) will be filled in 1st half of input FFT buffer and FFT is calculated. The FFT contents are copied into separate buffer I times (as per equation (12)). The one-time coefficients FFT is calculated with size equal to output FFT length. The FFT buffer contents after forming I images is multiplied with those of coefficients FFT contents. To the resultant FFT multiplied output, IIFT is applied to get the interpolated time domain output. The 1st 128 samples are transmitted as output frame.

As described in decimation process, the interpolation factors are also non-power of 2. Appendix- A could be referred for such FFT calculations.

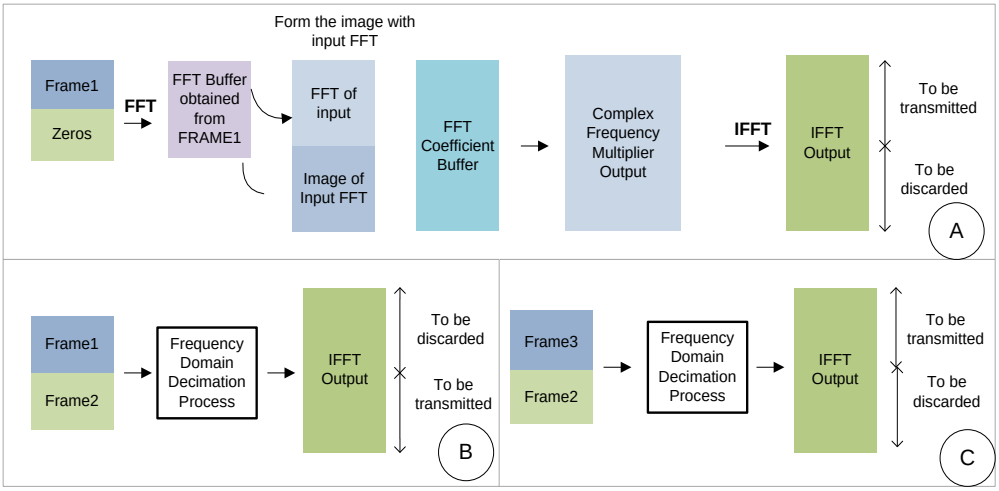

Figure 4. Frequency Domain Implementation of Interpolation. The interpolation factor here is I=2. For other Interpolation Factors, the Buffer Sizes will Vary Accordingly

3.2. Computational Complexity of Frequency Domain based Interpolation

Table 1. Theoretical Computational Complexity of Interpolation using Overlap Save Method. Here $O(N) = N\log_2 N$

To Calculate	Complex Multiplications	Complex Additions	Remarks
X(k)	0.5 O(N/I)	O(N/I)	Complexity for FFT calculation of length N/I
Y(k)	N	-	Complex frequency multiplication
y(n)	0.5 O(N)	O(N)	IFFT calculation of length N.
Total	N + 0.5 [O(N) + O(N/I)]	O(N)+O(N/I)	

The computational complexity details described in above table are valid if interpolation factor is a power of 2. If decimation factor is not a power of 2, there will be additional complexity to be considered and is described in Appendix A.

4. Simulation Results and Discussion

The proposed methods of decimation and interpolation were simulated on MATLAB [11] platform. The simulator code is capable of performing decimation and interpolation for any sampling rate factor. The input signal provided to the simulator code is 44.1kHz, 0dB, 24-bit stereo signal. This signal basically contains sine tones. The complex input is formed with two channels. The outputs of frequency domain and time domain processing are compared for various sampling rate factors. Fig. 6 shows spectrum comparison. The deviation of SNR and THD+N between input and the two outputs are clearly negligible.

Since precision in MATLAB is of double datatype, the same precision may not possible in fixed point DSP processors as bit width (typically 16) of these processors is less. To obtain high SNR and low THD+N value, it is recommended to use floating point 32-bit processors (such as SHARC 21xxx processors) for real-time implementation of these techniques.

4.1. Application to Fractional Rate Conversion

In fractional rate conversion, input and output sampling frequencies are related with arbitrary ratio. In this process, interpolation is performed first and then decimation. This is because spectral contents of input signal will be lost if decimation is performed first. Also the cutoff frequency of the filter in this method is minimum of that is needed for decimation and interpolation [1-3].

The proposed frequency domain decimation and interpolation can be directly applied to fractional rate conversion if decimation and interpolation factors are less than 5 roughly. The procedure described in Sections 2 and 3 is combined to implement fractional rate conversion. If these values are more than 5, the memory required to store FFT buffers is large. For such conversions, it is better to proceed with the existed time domain techniques based on band-limited interpolation proposed by Gosset and Smith [5]. Apart from this, Lagrange interpolation and Farrow fractional delay structures are well suited for optimized computational complexity [6-9].

4.2. Limitations

The proposed methods have some limitations. If sampling factor becomes more, either input or output FFT length becomes large, where DSP needs more memory to store the related buffers to calculate FFT. If DSP internal memory is not sufficient to store these buffers, memory management could be done with external memory storage and DMA operations.

5. Conclusion

In this paper, frequency domain based decimation and interpolation techniques are simulated on MATLAB platform. The quality of the outputs is compared against that of time domain outputs. The SNR and THD+N deviations are negligible. The theoretical computations are provided. These techniques could be optimized and could be applied to asynchronous band-limited interpolation techniques. This work could be taken as future update.

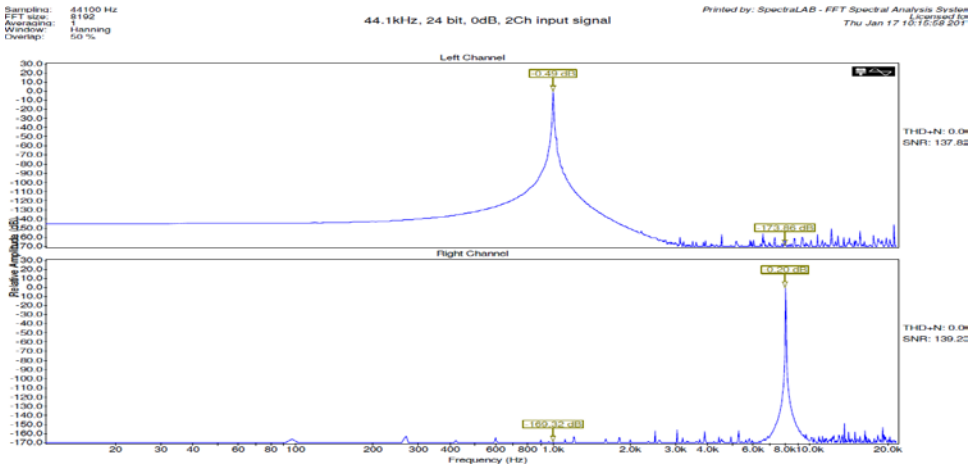

Figure 5. Input Test Signal – 44.1kHz, 2channel, 0dB, 24 Bit

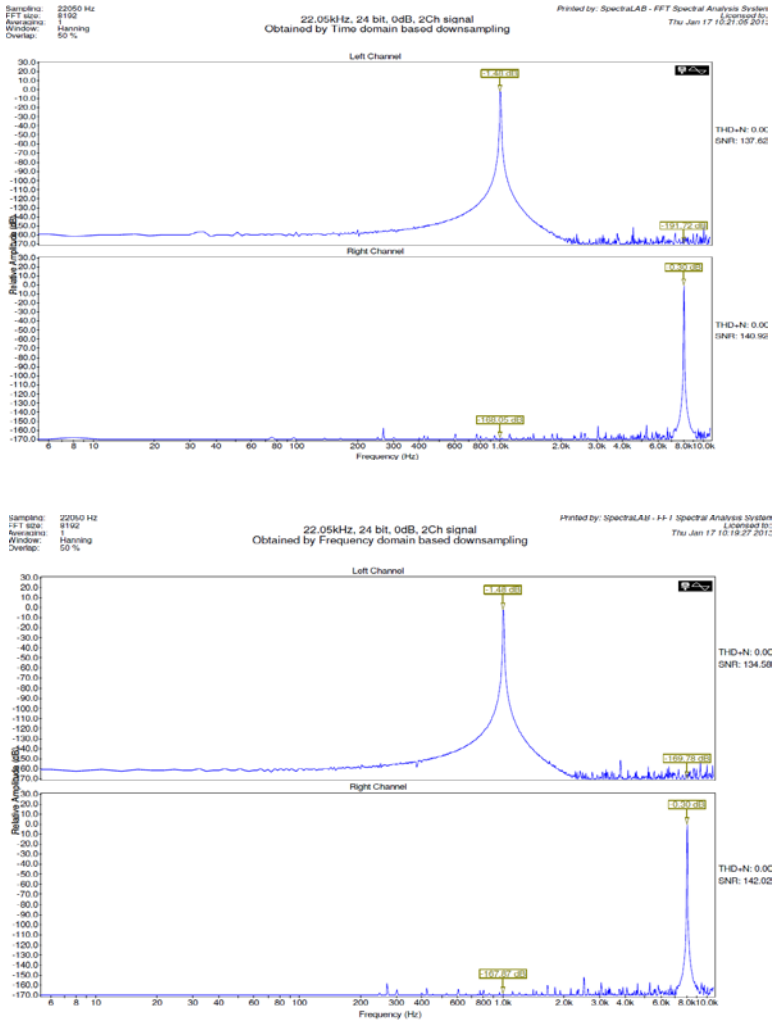

Figure 6. Time Domain (left plot) vs Frequency Domain (right plot) Downsampling comparison for D = 2

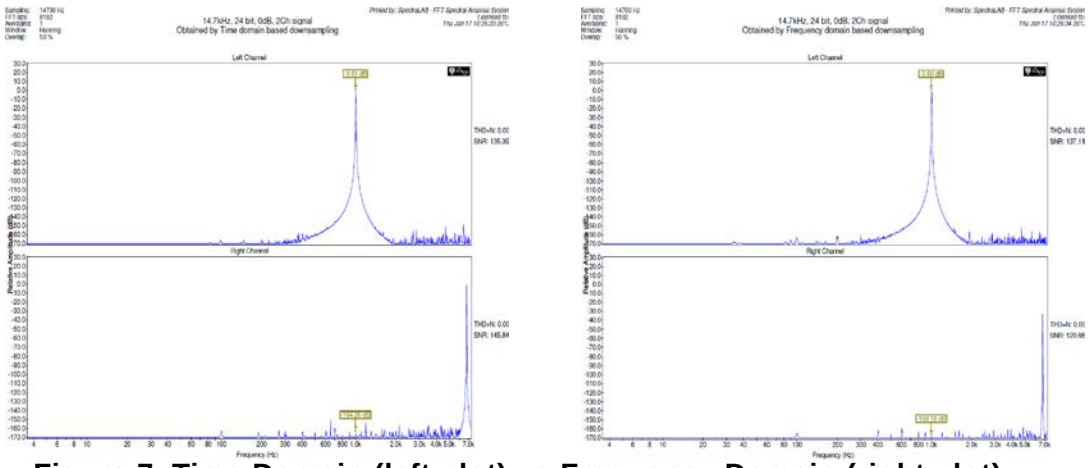

Figure 7. Time Domain (left plot) vs Frequency Domain (right plot) Downsampling Comparison for D = 3

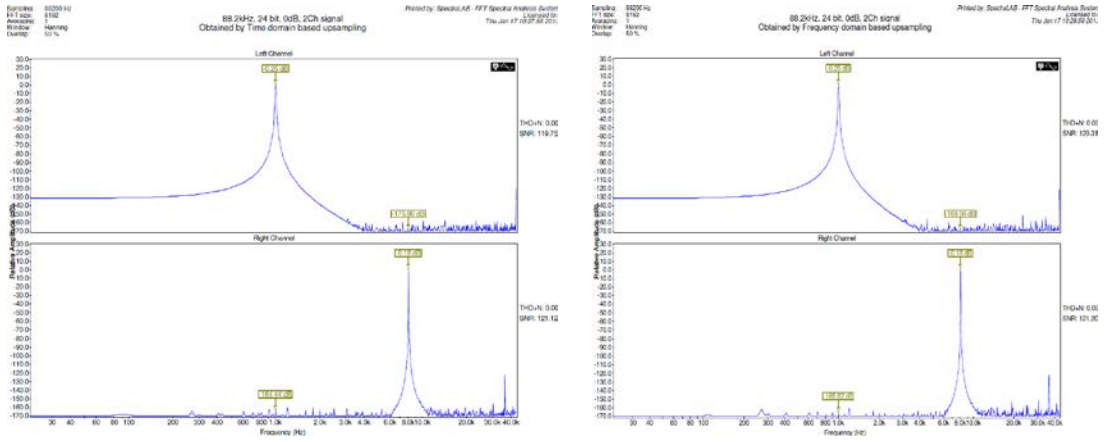

Figure 8. Time Domain (left plot) vs Frequency Domain (right plot) Upsampling Comparison for I = 2

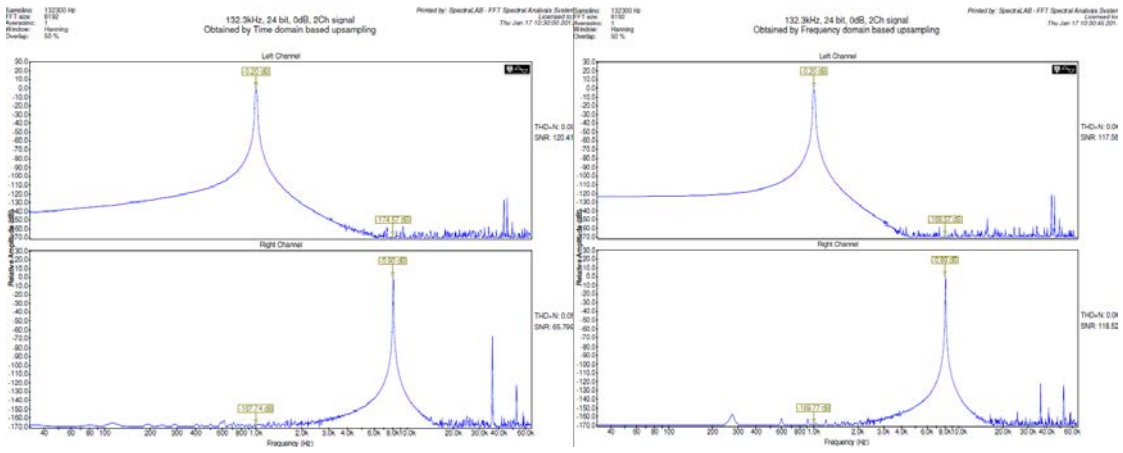

Figure 9. Time Domain (left plot) vs Frequency Domain (right plot) Upsampling Comparison for I = 3

Appendix - A. Frequency Domain Process for Non-power of 2 Sampling Rate Factors

When applying FFT, the buffer length should be a power of 2, in general, as FFT needs this requirement. Usually input FFT length is chosen as sampling rate factor times the output buffer FFT length. If this factor is not a power of 2, input FFT size obviously is not a power of 2. This issue could be solved with the following approach. The necessary condition for this is that FFT size must be an integer multiple of power of 2, even though it is not an exact power of 2. Here sampling factor is considered as 3 to explain this technique.

Let *x(n)* and *X(k)* are Fourier Transform pair of length *N*, which is not a power of 2. *X(k)* can be expressed as

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk}, \qquad W_N=\exp\left(-\frac{j2\pi}{N}\right) \quad \& \quad k=0,1,..N-1$$

$$=x(0)+x(1)W_N^{k}+x(2)W_N^{2k}+....+x(N-1)W_N^{(N-1)k}$$

$$=\left[x(0)+x(3)W_N^{3k}+..\right]+\left[x(1)W_N^{k}+x(4)W_N^{4k}+..\right]+\left[x(2)W_N^{2k}+x(5)W_N^{5k}+..\right]$$

$$=\left[x(0)+x(3)W_{N/3}^{k}+..\right]+W_N^{k}\left[x(1)+x(4)W_{N/3}^{k}+..\right]+W_N^{2k}\left[x(2)+x(5)W_{N/3}^{k}+..\right]$$

$$=X_0(k)+W_N^{k}X_1(k)+W_N^{2k}X_2(k)$$

The FFT Size of $X_0(k)$, $X_1(k)$, $X_2(k)$ is N/3 – 1. The periodic property of FFT is utilized to derive the values from k = N/3 to N-1.

$$X\left(k+\frac{N}{3}\right)=X_0\left(k+\frac{N}{3}\right)+W_N^{k+N/3}X_1\left(k+\frac{N}{3}\right)+W_N^{2(k+N/3)}X_2\left(k+\frac{N}{3}\right)$$

$$=X_0(k)+W_N^{k+N/3}X_1(k)+W_N^{2k+2N/3}X_2(k)$$

$$X\left(k+\frac{2N}{3}\right)=X_0\left(k+\frac{2N}{3}\right)+W_N^{k+2N/3}X_1\left(k+\frac{N}{3}\right)+W_N^{2(k+2N/3)}X_2\left(k+\frac{N}{3}\right)$$

$$=X_0(k)+W_N^{k+2N/3}X_1(k)+W_N^{2k+N/3}X_2(k)$$

For example, FFT for length, N = 6 is obtained as follows. Since N = 2*3, as per the above procedure $X_0(k)$, $X_1(k)$, $X_2(k)$ are calculated with k=0,1,2,3.

The time domain sequences are

$$x_0(n)=[x(0),x(3)], \qquad x_1(n)=[x(1),x(4)] \qquad x_2(n)=[x(2),x(5)]$$

The FFT of x(n) is obtained from $X_0(k)$, $X_1(k)$, $X_2(k)$ as follows.

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \end{bmatrix} = \begin{bmatrix} X_0(0) \\ X_0(1) \\ X_0(0) \\ X_0(1) \\ X_0(0) \\ X_0(1) \end{bmatrix} + \begin{bmatrix} \exp(-j2\pi.0/6) \\ \exp(-j2\pi.1/6) \\ \exp(-j2\pi.2/6) \\ \exp(-j2\pi.3/6) \\ \exp(-j2\pi.4/6) \\ \exp(-j2\pi.5/6) \end{bmatrix} \begin{bmatrix} X_1(0) \\ X_1(1) \\ X_1(0) \\ X_1(1) \\ X_1(0) \\ X_1(1) \end{bmatrix} + \begin{bmatrix} \exp(-j2\pi.0/6) \\ \exp(-j2\pi.2/6) \\ \exp(-j2\pi.4/6) \\ \exp(-j2\pi.6/6) \\ \exp(-j2\pi.8/6) \\ \exp(-j2\pi.10/6) \end{bmatrix} \begin{bmatrix} X_2(0) \\ X_2(1) \\ X_2(0) \\ X_2(1) \\ X_2(0) \\ X_2(1) \end{bmatrix}$$

The same technique is applicable for IFFT calculation also except the –ve sign in twiddle factor.

By comparing this with normal FFT/IFFT calculation, the additional complexity is due to twiddle factor multiplication and addition of resultant complex values due to complex multiplication. For a sampling rate factor of R, this additional complexity contains N(R-1) complex additions and N(R-1) complex additions.

Appendix - B. MATLAB Simulation Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Sampleing Rate conversion-  Main function
%   1. Downsampling
%   2. Upsampling
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;

[InStream,SRC.InputFs, SRC.Nbits] =wavread('Tones_2p0_0dB_44.1kFs_24b.wav');

disp('1. Downsampling');
disp('2. Upsampling');

SRC.Method = input('Enter SRC Method');


if SRC.Method == eSRCMethod.Downsampling

    SRC.DecFactor = input('Enter Decimation Factor');

    SRC.IntrpFactor     = 1;
    SRC.Time.OutBufL    = 64;                              % Output frame size is fixed @ 64
    SRC.Time.InBufL     = SRC.Time.OutBufL*SRC.DecFactor; % Required input frame size
    SRC.Freq.OutFFT_L   = 128;                             % Output FFT length
    SRC.Freq.InFFT_L    = SRC.Freq.OutFFT_L*SRC.DecFactor;% Required input FFT length
    SRC.Freq.CoeffFFT_L = SRC.Freq.InFFT_L;                % Set coefficients FFT length equal
                                                           % to input FFT length

    SRC.Coeffs = DeriveSRCCoeffs(SRC.DecFactor);           % Designing of SRC Coeffs for given
                                                           % decimation factor
    SRC.Freq.Coeffs = fft(SRC.Coeffs',SRC.Freq.CoeffFFT_L); %   Caclulate FFT of coefficients
    %   Downsampling calling function

    OutStream = Downsampling(InStream,...
                             SRC.Time.InBufL,...
                             SRC.Time.OutBufL,...
                             SRC.Freq.InFFT_L,...
                             SRC.Freq.OutFFT_L,...
                             SRC.DecFactor,...
                             SRC.Freq.Coeffs);

    SRC.OutputFs = SRC.InputFs/SRC.DecFactor;

    wavwrite( OutStream,SRC.OutputFs,SRC.Nbits,'Decimated_output.wav');
```

```
elseif SRC.Method == eSRCMethod.Upsampling
    SRC.DecFactor   = 1;
    SRC.IntrpFactor = input('Enter Interpolation Factor');

    SRC.Time.InBufL     =  64;                              %  Input frame length is fixed @ 64
    SRC.Time.OutBufL    =  SRC.Time.InBufL*SRC.IntrpFactor; %  Required output frame length
    SRC.Freq.InFFT_L    =  128;                             %  Input FFT length
    SRC.Freq.OutFFT_L   =  SRC.Freq.InFFT_L*SRC.IntrpFactor;%  Output FFT length
    SRC.Freq.CoeffFFT_L =  SRC.Freq.OutFFT_L;               %  Set coefficients FFT length equal
                                                            %  to output FFT length

    SRC.Coeffs = DeriveSRCCoeffs(SRC.IntrpFactor);          % Designing of SRC Coeffs for given
                                                            %  interpolation factor
    SRC.Freq.Coeffs = fft(SRC.Coeffs',SRC.Freq.CoeffFFT_L); % Caclulate FFT of coefficients

    OutStream = Upsampling(InStream,...
                           SRC.Time.InBufL,...
                           SRC.Time.OutBufL,...
                           SRC.Freq.InFFT_L,...
                           SRC.Freq.OutFFT_L,...
                           SRC.IntrpFactor,...
                           SRC.Freq.Coeffs);

    SRC.OutputFs = SRC.InputFs*SRC.IntrpFactor;

    wavwrite( OutStream,SRC.OutputFs,SRC.Nbits,'Interpolator_Output.wav');

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Filter Frequency Response and filter coefficinets generation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Coeffs = DeriveSRCCoeffs(SamplingFactor)

Filter_Length  = 127;               %   Coefficients Length

InFs     =   96000;
max_FreqContent = InFs/2/SamplingFactor;

d1 = fdesign.lowpass('N,Fc,Ap,Ast',Filter_Length,max_FreqContent,0.001,120,InFs);
Hd1 = design(d1,'equiripple');

Coeffs = Hd1.Numerator;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Frequency Domain based Downsampling Process
%   ---------------------------------------------------------------------
%   1. Applicable for any decimation factor
%   2. Time domain filter coefficients length is 128
%   3. Two channels of input signal are processed directly by forming
%       complex signal with two channels. After downsampling, two channels
%       are separated with real and imaginary outputs of downsampling.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Out     =   Downsampling(Input,...
                                  InFrmL,...
                                  OutFrmL,...
                                  InFFT_L,...
                                  OutFFT_L,...
                                  DecFactor,...
                                  Freq_Coeffs)

% Determine the total number of frames in input stream
if rem(length(Input),InFrmL) ~= 0
    TotalFrames         =   ceil(length(Input)/InFrmL);
else
    TotalFrames         =   length(Input)/InFrmL;
end

Input(length(Input):TotalFrames*InFrmL,:) = 0;
```

```
%Form complex input with stereo channels of input
Instream    = Input(:,1)+ j *Input(:,2);
InputBuf    = zeros(InFFT_L,1);
FFT_Buf     = zeros(InFFT_L,1);  % FFT process buffer
InWrPtr     = 1;% used to copy input frames into process buffer
OutRdPtr    = 1;        % used to copy output frames into transmit buffer
Output      = [];

for i = 1:TotalFrames

        % Copy current input frame to be processed
        InputBuf(InWrPtr:InWrPtr+InFrmL-1,1) = Instream((i-1)*InFrmL + 1 : i*InFrmL);

        % Update the input write pointer by frame length
        InWrPtr = InWrPtr + InFrmL;

        if(InWrPtr >= length(FFT_Buf))
            InWrPtr = 1;
        end
        % Check decimation factor is power of 2 or not
        if dspCheckPow2(DecFactor) == 1
            FFT_Buf = fft(InputBuf,InFFT_L);
            FFT_Buf = FFT_Buf.*Freq_Coeffs;
        else
            %Calculation of small FFTs
            InFFTSize = length(FFT_Buf)/DecFactor;
          for m=1:DecFactor
              smallFFT(1:InFFTSize,m)=fft(InputBuf(m:DecFactor:length(FFT_Buf)),InFFTSize);
          end

          FFT_Buf(1:length(FFT_Buf),1) = 0;
          %Calculation of large FFTs using small FFTs
          for n = 1:DecFactor

              for m = 1:DecFactor

                  tw = exp(-j*2*pi*[(n-1)*InFFTSize:n*InFFTSize-1]'*(m-1)/length(FFT_Buf));

                  FFT_Buf((n-1)*InFFTSize+1:n*InFFTSize,1)=FFT_Buf((n-1)*InFFTSize+1:n*InFFTSize,1)+tw.*smallFFT(:,m);
              end
              clear tw;
          end
          FFT_Buf(:,1) = FFT_Buf(:,1).*Freq_Coeffs;
      end

      SRC_OutBuf = zeros(1,OutFFT_L);
      %   Add downsampled content and aliased contents
      for k = 1:DecFactor

          alias_FFTBuf(1:OutFFT_L) = FFT_Buf((k-1)*OutFFT_L+1:k*OutFFT_L,1);

          SRC_OutBuf = SRC_OutBuf + alias_FFTBuf;
      end
      % Apply IFFT to the sum and scale with decimation factor.
      SRC_OutBuf = (1/DecFactor)*ifft(SRC_OutBuf,OutFFT_L);
      % Transmit the OutFrmL samples
      Output = [Output,SRC_OutBuf(OutRdPtr:OutRdPtr+OutFrmL-1)];

      OutRdPtr = OutRdPtr + OutFrmL;

      if(OutRdPtr >= length(SRC_OutBuf))
          OutRdPtr = 1;
      end
end

Out(:,1) = real(Output);
Out(:,2) = imag(Output);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Frequency Domain based Downsampling Process
%   ---------------------------------------------------------------------------
%   1. Applicable for any decimation factor
%   2. Time domain filter coefficients length is 128
%   3. Two channels of input signal are processed directly by forming
%       complex signal with two channels. After downsampling, two channels
%       are separated with real and imaginary outputs of downsampling.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Out     =   Upsampling(Input,...
                                InFrmL,...
                                OutFrmL,...
                                InFFT_L,...
                                OutFFT_L,...
                                IntrpFactor,...
                                FFT_Coeffs)

if rem(length(Input),InFrmL) ~= 0
    TotalFrms         =   ceil(length(Input)/InFrmL);
else
    TotalFrms         =   length(Input)/InFrmL;
end

Input(length(Input):TotalFrms*InFrmL,:) = 0;
%Form complex input signal
Instream    = Input(:,1)+ j *Input(:,2);
Input       =   zeros(1,InFFT_L);
WrPtr       =   1;
FFT_ProcBuf  =   zeros(InFFT_L,1);
OutRdPtr = 1;
Output   = [];

for i = 1:TotalFrms

        % Copy current input frame to be processed
        Input(WrPtr:WrPtr+InFrmL-1) = Instream((i-1)*InFrmL + 1 : i*InFrmL);

        WrPtr = WrPtr + InFrmL;

        if(WrPtr >= length(FFT_ProcBuf))
            WrPtr = 1;
        end
        % FFT calculation for input buffer
        ProcBuf(1:InFFT_L,1) = fft(Input,InFFT_L);

        % Images FFT copying
        for l=2:IntrpFactor
            ProcBuf((l-1)*InFFT_L+1:l*InFFT_L,1) = ProcBuf(1:InFFT_L,1);
        end
        %Frequency Multiplication
        ProcBuf = ProcBuf.*FFT_Coeffs;

        if dspCheckPow2(IntrpFactor) == 1
            largeFFT_Buf = IntrpFactor*ifft(ProcBuf,OutFFT_L);
        else

            InFFTSize = length(ProcBuf)/IntrpFactor;

            for m=1:IntrpFactor
                smallFFT(1:InFFTSize,m)=ifft(ProcBuf(m:IntrpFactor:length(ProcBuf)),InFFTSize);
            end
             largeFFT_Buf(1:length(ProcBuf),1) = 0;

             for n = 1:IntrpFactor

                 for m = 1:IntrpFactor

                     tw = exp(j*2*pi*[(n-1)*InFFTSize:n*InFFTSize-1]'*(m-1)/length(largeFFT_Buf));

                     freq_index = (n-1)*InFFTSize+1:n*InFFTSize;

                     largeFFT_Buf(freq_index,1)=largeFFT_Buf(freq_index,1)+tw.*smallFFT(:,m);
                 end
                 clear tw;
             end
        end
        %Transmit IFFT output
        Output = [Output;largeFFT_Buf(OutRdPtr:OutRdPtr+OutFrmL-1)];

        OutRdPtr = OutRdPtr + OutFrmL;
        if OutRdPtr >= length(largeFFT_Buf)
            OutRdPtr = 1;
        end
end

Out(:,1) = real(Output);
Out(:,2) = imag(Output);
```

References

[1] J. G. Proakis and D. G. Manolakis, Prentice-Hall International, "Digital Signal Processing Principles", Algorithms and Applications, 3rd Edition.
[2] A. Oppenham and R. Schafer, Prentice-Hall International, "Discrete-Time Signal Processing", **(1989)**.
[3] L. Rabiner and B. Gold, Prentice-Hall International, "Theory and Application of Digital Signal Processing", 1975.
[4] R. G. Lyons, Prentice-Hall International, "Understanding Digital Signal Processing", **(2004)**.
[5] J. O. Smith and P. Gossett, "A Flexible Sampling-Rate Conversion Method", Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, San Diego, (ICASSP-84), New York: IEEE Press, vol. 2, **(1984)** March, pp. 19.4.1–19.4.2.
[6] H. Johansson and O. Gustafsson, "Linear-phase FIR interpolation, decimation, and mth-band filters utilizing the Farrow structure", IEEE Trans. Circuits Syst. I, vol. 52, **(2005)**, pp. 2197-2207.
[7] Candan, "An Efficient Filtering Structure for Lagrange Interpolation", IEEE Trans. Circuits Syst. I, vol. 52, **(2005)**, pp. 2197-2207.
[8] V. Valimaki, "A new filter implementation strategy for Lagrange interpolation", Proc. IEEE Int. Symp. Circuits and Systems, **(1995)**, pp. 361-364.
[9] L. Erup, F. M. Gardner and R. A. Harris, "Interpolation in digital modems Part II: Implementation and performance", IEEE Trans. Commun, vol. 41, **(1993)**, pp. 998-1008.
[10] J. Vesma and T. Saramaki, "Optimization and efficient implementation of FIR filters with adjustable fractional delay", Proc. IEEE Int. Conf. Acoust. Speech Signal Process., **(1997)**, pp. 2256-2259.
[11] MATLAB, "The Language of Technical Computing", Version 7.9.0.529 (R2009b) 32-bit (win32), **(2009)**, August 12.

Authors

Chunduri SreenivasaRao, MTS Software Development Engineer in AMD, Hyderabad, India.

A. Arockiaraj, Software Development Engineer in Qualcomm, Hyderabad, India

Dr. D. VenkataRao, Prinicipal, Narsaraopet Institure of Technology, A.P., India