Implementation and Analysis of Two-Stage Frequency Domain Decimator

Ina Masnikosa, Stefan Stojkov
School of electrical engineering, University of Belgrade
Mihajlo Pupin Institute, University of Belgrade
Belgrade, Serbia
ina.masnikosa@pupin.rs, stefan.stojkov@pupin.rs

Jelena Certic
School of electrical engineering, University of Belgrade
Belgrade, Serbia
certic@etf.rs

Abstract—**In this paper, frequency domain implementation of the two-stage decimator is presented. The whole filtering and decimation process is realized in frequency domain in combination with well-known fast convolution methods: *Overlap-Add* and *Overlap-Save*. The two-stage implementation makes the algorithm suitable for large decimation rates. The proposed algorithm utilizes block processing and it is compared with its one-stage equivalent and direct implementation of finite impulse response (FIR) decimator in terms of time performance. The results show that this algorithm outperforms both of these decimators.**

Keywords-fast convolution; decimation; two-stage; filtering; frequency domain; block processing

I. INTRODUCTION

The one of the most important goals in design of telecommunication devices is to perform signal processing as much as possible in digital domain. As a result, in the case of the receiver, the analog to digital conversion point is moved closer to the analog front end, i.e. digitalization of the signal is performed at relatively high frequencies. A high rate decimator is a necessary part of digital signal processing chain in that case. In this paper, the efficient realization of the two-stage decimator suitable for real time applications is presented. Both stages of decimator are realized in the frequency domain. Filtering is realized by means of fast convolution and downsampling is realized by rearranging of the spectrum of the filtered signal.

The idea for fast convolution algorithms lies in the convolution theorem: multiplication in the frequency domain is equivalent to convolution in the time domain. In digital signal processing, frequency domain representation of the signal is usually obtained by DFT (Discrete Fourier Transform). Inverse DFT of the product of two DFT sequences actually is a circular convolution of the corresponding signals in the time domain. However, in [1] and [2] was shown that linear convolution can be obtained based on the calculation of the circular convolution by DFT. Many research papers explored these DFT algorithms and showed that their results were better regarding efficiency than using direct convolution with tapped-delay line finite impulse response (FIR) filters. In order to achieve significant improvement in time efficiency, the processing signal is usually divided into blocks and the DFT of each block is calculated [1]. This ensures speeding up the process, but introduces additional delay. This is due to the fact that it is necessary to collect the number of samples equal to the block length to start processing. *Overlap-Add* and *Overlap-Save* are two most common methods for utilizing fast convolution and they can be combined with sample rate conversion techniques according to the work presented in several papers [2], [3]. Also, it is better to use multistage filtering when large decimation factors are involved [4]. The goal of this paper is to present the algorithm which combines the mentioned methods and implements two-stage frequency domain decimator. Development of the decimator, signal processing and testing is implemented in C++ programming language. Kiss FFT (Fast Fourier Transform) is used for the calculation of the Fourier transform [5]. It is a C based library, which uses fixed and floating point data types and it gave satisfactory results during testing.

After the introductory part, a brief overview of the fast convolution algorithms will be given. In Section III, the proposed algorithm is described in detail. Section IV features the experimental settings and obtained results. The concluding remarks are given in the last section.

II. FREQUENCY DOMAIN SIGNAL PROCESSING

A. Fast Convolution Methods

Fast convolution algorithms become very useful when the filter ($h(n)$) length is greater than 40 to 80 which depends on the hardware and software being used [6]. The reason for this lies in the fact that the process requires fewer computations than directly implemented convolution. The larger the lengths of the input signal and filter impulse response are, these advantages become more significant.

Fast convolution gives us output signal with length N_{CONV}. If the length of the input signal ($x(n)$) is N, and the number of filter coefficients is M, then the $N_{CONV} = N + M - 1$. In order to achieve this, we have to choose DFT size such that $N_{DFT} \geq N_{CONV}$ and then zero-pad $h(n)$ and $x(n)$ so that they have new lengths equal to N_{CONV}.

If the $x(n)$ input sequence becomes very large (i.e. endless), circular convolution cannot be performed, and some additional methods are used. The input sequence is partitioned

into smaller blocks of samples and each block is processed individually. *Overlap-Add* method computes a continuous convolution from smaller convolutions in a *buffer-by-buffer* manner. Another technique used in this case is *Overlap-Save* which breaks the rules for linear circular convolution equivalence, i.e. it uses circular convolution without zero-padding [2].

The *Overlap-Add* method consists of the following steps [7]:

- decomposition of the signal into simple components,
- adequate processing (filtering) of each component,
- recombination of the processed components into the final signal.

The idea is to divide the processing signal $x(n)$ into blocks and convolve each block with the filter coefficients. If the length of the each sequence is N, and the filter length is M, then the length of the resulting signal of each processed block is $N+M-1$. The extra $M-1$ samples are added to the first $M-1$ samples of the next block. This procedure is repeated until all samples of the input signal are filtered.

The *Overlap-Save* method uses a slightly different approach. The idea is to save the part of the first input block that contributes to the second output block and use it in that calculation (input signal is overlapped). After the calculation, overlapped parts of the output are discarded.

B. Frequency Domain Decimation

Downsampling by factor D can be implemented in the frequency domain by adding the frequency components to be aliased. The speed advantage that comes with the smaller inverse DFT becomes more pronounced at higher decimation rates. The important condition is that the number of frequency bins must be a multiple of the decimation rate. Downsampling as a part of fast convolution has the following conditions [3]:

1. The filter order must be a multiple of decimation rate:

$$M - 1 = K_1 D \tag{1}$$

2. The DFT length must be a multiple of decimation rate:

$$N + M - 1 = K_2 D. \tag{2}$$

In (1) and (2), K_1 and K_2 are integer values.

III. TWO-STAGE FREQUENCY DOMAIN DECIMATOR

The algorithm is a mixture of different ideas presented in the papers that consider signal processing in the frequency domain. The advantage of this method is that the entire processing for each stage is realized in the frequency domain (filtering and decimation). As already mentioned, this advantage becomes more significant as the order of the filter increases. When the number of filter coefficients exceeds certain number, this algorithm becomes more efficient than the direct implementation in time domain. The algorithm is illustrated in Fig. 1. Since the algorithm utilizes block processing, the predefined number of samples needs to pass through the system before the processing can begin.

Signal is processed in blocks of N samples, and each block is filtered in two stages. The filtering process is carried out in two stages when the decimation factor is large ($D>20$). In that case, the decimation factor in the first stage is chosen using [4]:

$$D_1 \approx 2D \frac{1-\sqrt{DF/(2-F)}}{2-F(M+1)} \tag{3}$$

where F is the ratio of single-stage lowpass filter's transition region width to the stopband frequency of the filter (f_{stop}):

$$F = \frac{f_{stop} - f_{pass}}{f_{stop}} \tag{4}$$

The first stage decimation factor is the closest integer value to (3), which is also a divisor of the total decimation factor. The second stage decimation factor can be obtained by simply dividing the total decimation factor with the first stage value. This equation gives the optimal values for decimation factors in terms of computational savings.

Parameters of the implemented algorithm for each stage are:

- N – the number of signal samples in one block for processing;
- M – filter length;
- D – decimation factor;
- N_{DFT} – DFT size.

All the parameters were evaluated in accordance with the rules for decimation when using fast convolution, given with (1) and (2), applied for each stage separately. Also, in order to make the DFT more efficient, FFT (Fast Fourier Transform) algorithm can be used. In that case the length of the transform (N_{FFT}) can be chosen to be power of two (2^n, where n is an integer). This leads to conclusion that, in that case, the filter order also needs to be 2^n.

The first stage (using *Overlap-Add* method) consists of the following steps:

1. Zero-pad the filter $h(n)$ with N-1 zeros to make it length ($N+M-1$).
2. Calculate the FFT (length $N+M-1$) of the filter obtained in the previous step.
3. Analogously, each signal block needs to be zero-padded with $M-1$ zeros in order to make it length $N+M-1$.
4. Calculate the FFT (length $N+M-1$) of the signal block obtained in the previous step.
5. The FFTs of the filter and the signal block are then multiplied, and the resulting signal is N_{FFT} long.

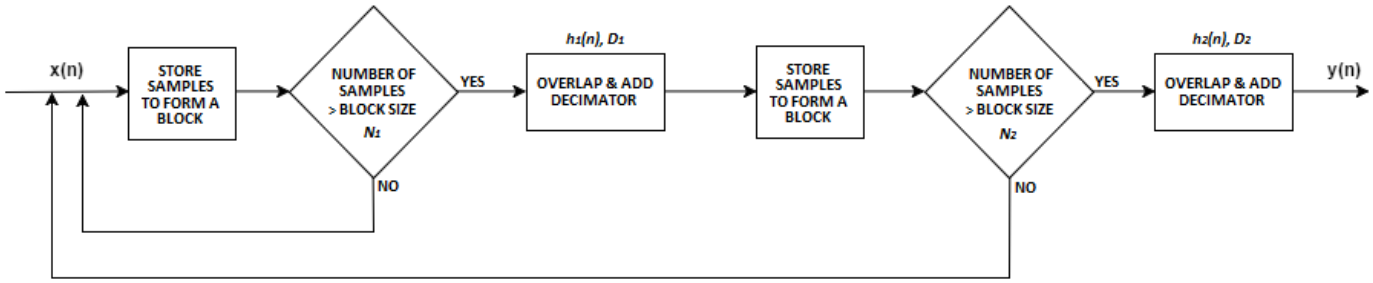

Figure 1: Two-stage frequency domain decimator

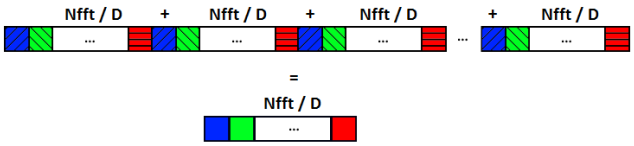

Figure 2: Frequency domain decimation

6. Coherently add the frequency components to be aliased. FFT decimation procedure is shown in Fig. 2. Basically, the signal obtained in step 5 needs to be divided into D equal intervals. Then each element of the first segment is summed with the corresponding elements in the other ones. The corresponding elements are illustrated with the same colors in Fig. 2. The resulting signal is N_{FFT}/D long.
7. Compute the inverse FFT and scale it with factor D. Note that it is important to provide that this fraction (N/D) is integer.
8. The last $(M–1)/D$ samples obtained in step 7 are added to the first $(M–1)/D$ samples of the next block processed.

In the first stage, signal sample rate was decreased D_1 times, so N input samples gave N/D_1 output samples after applying the first filter. This fraction N/D_1 must be equal to $n \times D_2$, where n is an integer (n = 1, 2, …) and D_2 is second stage decimation factor. In second stage, all steps from 1-8 are repeated, but with different filter impulse response and decimation factor. The whole process is repeated until all samples of the input signal are filtered. In practical realization, filter impulse response for both stages is pre-calculated so that there is no additional time consumed on this computation (steps 1-2) when processing each block of samples.

The algorithm with *Overlap-Save* method uses similar concept as the one with *Overlap-Add*, with the difference that the last $(M–1)/D$ samples from the previous block (previously saved) are used as the first $(M–1)/D$ samples of the current block (this creates an overlap). When filtering is applied, then these first $(M–1)/D$ samples are discarded.

IV. Experimental Results

The presented algorithm is evaluated by comparing it with the time domain two-stage decimator and frequency domain one-stage decimator. Analysis is based on the measurement of two parameters: delay and processing time.

Several filters were designed using MATLAB software package with minimum filter order and the appropriate passband and stopband attenuation. The filters were zero-padded in case they did not satisfy (1).

In real-time application, block processing is applied. For the purpose of this testing, duration of each test signal is chosen to be fixed. This value assures that the whole signal can be equally divided into n blocks (where n is an integer value). In that way, all samples are processed for every set of fast convolution parameters that was examined.

A. *Hardware Configuration*

Table I gives specification of the used hardware. Tests were performed on Ubuntu 14.04.2 operating system which provided accurate time measurements.

B. *Fast Convolution Method vs. Direct FIR Implementation (One Stage)*

As stated before, fast convolution algorithm becomes efficient when the length of the filter exceeds certain value. Since this value is greatly influenced by hardware, the first step is to determine when the usage of the fast convolution algorithm becomes justified.

The designed filter coefficients were used to process test signal while utilizing two approaches: fast convolution algorithm and direct FIR implementation. The number of coefficients used in this analysis varied from 38 to 84. It was observed that in this case fast convolution algorithm outperforms direct FIR decimator when filter length exceeds 45, for the optimal set of fast convolution parameters, which includes trade-off between time efficiency and the initial delay. In addition, when the filter length is greater than 80, fast convolution is more efficient even for the smallest possible delay. The trade-off between time efficiency and system delay will be further explained in the next section for the two-stage implementation.

TABLE I. Hardware Specification

Hardware	Specification
RAM	8 GB
CPU Model	Intel(R) Core(TM) i5-4690 CPU @ 3.50GHz
Number of CPU Cores	4
Magnetic Hard Disk Drive	Seagate Barracuda 1TB, 3.5"
Operating System	Ubuntu 14.04.2 LTS Desktop 64-bit

C. *Analysis of the Two-Stage Frequency Domain Decimator*

In cases where decimation factor is high, it is better to use multistage decimators. The proposed algorithm is tested in case of two stages in accordance to (3). The advantage of the algorithm is that during single stage complete processing (filtering plus decimation) is done in frequency domain, which reduces processing interval. The processing is carried out in blocks, which size depends on the FFT length. With the increase in block size, the processing time decreases. Unfortunately, as the block size increases, so does the delay in the system, if it is used in real-time applications. Larger FFT length means that more samples are needed for the calculation. In two-stage approach, FFT length of the second stage affects the delay more than the one in the first stage. For large FFT size, the delay can become unacceptable if the algorithm is used for real-time applications. Specification of the filters used in this analysis is shown in Table II (sampling frequency – f_s, passband frequency – f_{pass}, stopband frequency – f_{stop}, passband attenuation – A_{pass}, stopband attenuation – A_{stop}, filter length – M).

Table III illustrates the previous statements (FCII - Frequency Domain Two-Stage Decimator, FCI - Frequency Domain One-Stage Decimator, FIRII - Time Domain Two-Stage Decimator). It was filled with results for the signal with initial sampling frequency of 256 kHz. First stage filter consists of 94 coefficients, while the second stage filter has 110 coefficients (table II). Both filters were zero-padded to the length of 128. The total decimation factor is 32 (D_1 = 16, D_2 = 2), i.e. the resulting sampling frequency is 8 kHz. Time efficiency measurement refers to the processing interval per one second of the signal, and it is given in milliseconds. The initial delay refers to the delay in the fast convolution algorithm. For example, in case where $N_1 = N_2 = 128$, the algorithm needs 2048 samples in order to start processing. After this first block is processed, 64 output samples will be produced.

It can be clearly seen that time efficiency of two-stage frequency domain decimator increases to certain extent when using larger blocks of samples. On the other hand, the delay increases as well, especially with the increase of the block size in second stage. The best result regarding time efficiency was obtained for $N_1 = N_2 = 896$, but because of the delay, it is suitable for offline processing. Since this solution is impractical for real-time applications, in that case the best choice would be to use blocks of $N_1 = 896$ and $N_2 = 128$.

TABLE II. FILTERS SPECIFICATION

	f_s [kHz]	f_{pass} [Hz]	f_{stop} [Hz]	A_{pass} [dB]	A_{stop} [dB]	M
I Stage	256	3400	12600	0.1	80	94
II Stage	16	3400	4000	0.01	80	110
One-Stage Equivalent	256	3400	4000	0.11	80	1406
I Stage	216	3400	20600	0.1	85	44
II Stage	24	3400	4000	0.1	80	115
One-Stage Equivalent	216	3400	4000	0.2	80	1114

TABLE III. DECIMATORS – TIME EFFICIENCY COMPARISON (FFT)

I Stage		**II Stage**		**Initial Delay [ms]**	**Time Measurement [ms]**		
N_{FFT1}	N_1	N_{FFT2}	N_2		***FCII***[a]	***FCI***[b]	***Direct FIRII***[c]
256	128	256	128	8	33.84	34.85 (N_{FFT} = 4096 N = 2048 Delay = 8 ms)	51.8
512	384			9	25		
1024	896			10.5	**21.45**		
2048	1920			15	22.02		
256	128	512	384	24	33.17	24.57 (N_{FFT} = 8192 N = 6144 Delay = 24 ms)	
512	384			24	23.85		
1024	896			24.5	20.46		
2048	1920			30	21.27		
256	128	1024	896	56	33.02		
512	384			57	23.7		
1024	896			56	**19.95**		
2048	1920			60	20.88		

a. Frequency Domain Two-Stage Decimator

b. Frequency Domain One-Stage Decimator

c. Time Domain Two-Stage Decimator

It can also be observed that the increase of the second stage signal length greatly increases the delay, which is consistent with the previous claims. The optimal fast convolution parameters can be determined for different filters and decimation rates by performing this type of analysis.

Table III also gives time measurements for one-stage frequency domain decimator (with the equivalent filter from table II) and time domain two-stage decimator (which uses the same filters as FCII). The usage of time domain one-stage decimator was not taken into consideration, since it would be highly impractical. The results justify the usage of the proposed algorithm.

The analysis can also be performed when the size of Fourier transform is not 2^n (table IV). In this case, there are more possible values, which can satisfy (1) and (2), which can reduce the delay in the system. The problem actually comes down to the DFT efficiency. The parameters must be chosen in a manner which would avoid dealing with the known problems in this field (like calculating inverse Fourier Transform with the size which is a prime number, etc.). As an example, test signal with the initial sampling frequency of 216 kHz and total decimation factor of 27 (D_1 = 9, D_2 = 3) was used. The designed filter for the first stage had 44, and for the second stage 115 coefficients (table II). The second filter is zero-padded to the 117.

TABLE IV. DECIMATORS – TIME EFFICIENCY COMPARISON (DFT)

I Stage		**II Stage**		**Initial Delay [ms]**	**Time Measurement [ms]**		
N_{DFT1}	N_1	N_{DFT2}	N_2		***FCII***[a]	***FCI***[b]	***Direct FIRII***[c]
90	45	234	117	5	33.96	38.43 (N_{DFT} = 2268 N = 1134 Delay = 5.25 ms)	25.85
180	135			5	25.55		
225	180			5	**22.51**		
270	225			5.21	22.40	26.03 (N_{DFT} = 4536 N = 1134 Delay = 15 ms)	
360	315			5.83	22.68		
720	675			6.25	21.62		

a. Frequency Domain Two-Stage Decimator

b. Frequency Domain One-Stage Decimator

c. Time Domain Two-Stage Decimator

The results are consistent with the previous analysis. The difference is that decreased DFT efficiency and filter length reduce the advantage of the frequency domain decimators.

V. CONCLUSION

The purpose of this paper is to present an efficient algorithm for combining filtering and decimation of the signal in the frequency domain for large decimation rates. It uses fast convolution methods and it is realized in two stages in order to improve time performance of the signal processing. The algorithm implements either *Overlap-Add* or *Overlap-Save* method and can be used for real-time applications. Development of the system, including signal processing and testing of the various signals and filters, was implemented in C++ programming language.

As expected, the first part of the analysis showed that the fast convolution algorithm outperforms direct FIR implementation when the number of filter coefficients exceeds 45 with the usage of optimal parameters. One-stage implementation becomes impractical for time domain decimator and less efficient for frequency domain decimator in case of large decimation factors. Therefore, the proposed algorithm utilizes the advantages of two-stage processing which leads to further improvement in time efficiency at the expense of the initial delay. The second part of the analysis addresses to this trade-off.

Time efficiency of the proposed algorithm can be further improved with the improvement of the FFT efficiency. Depending on the usage, the filters can be chosen in a manner which would minimize the initial delay.

REFERENCES

[1] I. Selesnick, C. Burrus, "Fast Convolution and Filtering," in The Digital Signal Processing Handbook, V. Madisetti and D. Williams, Eds. CRC Press, 1999, pp. 8.1-8.21.

[2] M. Borgerding, "Fast Convolution (FFT) Filtering: From Basics to Filter Banks", Comp.DSP Conference, July, 2004.

[3] M. Borgerding, "Turning Overlap-Save into a Multiband Mixing, Downsampling Filter Bank", IEEE Signal Processing Magazine, March, 2006.

[4] R. G. Lyons, “Sample Rate Conversion,” in Understanding Digital Signal Processing, New Jersey, USA: Prentice Hall, 2011, ch. 10, pp. 304–306.

[5] http://sourceforge.net/projects/kissfft/

[6] R. G. Lyons, “Fast FIR Filtering Using The FFT,” in Understanding Digital Signal Processing, New Jersey, USA: Prentice Hall, 2011, ch. 13, pp. 416–419.

[7] S. W. Smith, “FFT Convolution” in The Scientist and Engineer’s Guide to Digital Signal Processing, Second Edition, San Diego, California Technical Publishing, 1999, ch. 18, pp. 311–316.