DEFENCE R&D DÉFENSE

Digital Sampling Rate Conversion: Principles and Implementation

Sichun Wang
Intrinsix Corporation

Brian Kozminchuk
Defence Research Establishment Ottawa

Defence R&D Canada
DEFENCE RESEARCH ESTABLISHMENT OTTAWA

TECHNICAL MEMORANDUM
DREO TM 2001-032
February 2001

National Defence Défense nationale

Canada

DEFENCE R&D DÉFENSE

Digital Sampling Rate Conversion: Principles and Implementation

Sichun Wang
Intrinsix Corporation

Brian Kozminchuk
EW Signal Processing Group
Electronic Support Measures Section

DEFENCE RESEARCH ESTABLISHMENT OTTAWA

TECHNICAL MEMORANDUM
DREO TM 2001-032
February 2001

Project
02KS11

Abstract

The resampling of a signal involves the conversion from the initial sampling rate to a new and different one. This is often necessary in practical applications because the sampling rate is often fixed while the desired sampling rate may depend on the application or a signal parameter, such as the symbol rate or channel spacing. This problem often arises in software radios systems where the sampling rate is determined by hardware constraints. Although resampling algorithms have been developed and implemented in commercial software libraries, such as the Intel Signal Processing Library [2], these algorithms often have undesirable limitations. For example, a typical constraint is that the data blocks processed must be an integer multiple of the downsampling rate. This restriction simplifies the indexing in the code and reduces the complexity of the implementation. However, it decreases the flexibility and usefulness of the resulting program, since in some applications, the length of the input data blocks may not be an integer multiple of the downsampling rate. For such applications, there is a need for a resampling program that imposes no restriction on the length of the input data blocks. The MATLAB built-in function UPFIRDN.dll (or RESAMPLE.m) is designed for resampling only one block of input data. It does not provide information on the state variables of the filters and, therefore, cannot be used to resample a very large data file.

This report describes a fairly general resampling algorithm useful in many practical signal processing applications. To obtain the desired flexibility, we have implemented the periodically time varying FIR filter structure in such a way that it keeps track of the state variables and also makes provisions for input blocks of arbitrary length. In addition to its flexibility, the algorithm achieves a relatively high processing throughput when implemented in software.

Résumé

Le rééchantillonnage d'un signal implique la conversion du taux d'échantillonnage initial à un taux différent. C'est souvent une nécessité dans certaines applications pratiques dont le taux d'échantillonnage est fixe, alors qu'il faudrait que ce taux d'échantillonnage varie selon l'application ou un paramètre du signal, tels le débit des symboles ou l'espacement entre. C'est un problème qui surgit fréquemment pour les plates-formes radio logicielle pour lesquelles le taux d'échantillonnage est déterminé par des contraintes matérielles. Bien qu'on ait développé des algorithmes de rééchantillonnage dans le commerce et qu'on les ait mis en application dans des bibliothèques de logiciels, comme la Intel Signal Processing Library, ces algorithmes sont souvent à la merci de limitations indésirables. Une contrainte fréquente est l'obligation que les blocs de données soient un multiple entier du taux de sous-échantillonnage. Une telle restriction facilite l'indexation du code et réduit la complexité de la mise en application. Cependant, le programme résultant est alors moins polyvalent et moins utile, car dans certaines applications, la longueur des blocs de données d'entrée ne peut pas être un multiple entier du taux de sous-échantillonnage. Pour ces applications, il faut recourir un programme de rééchantillonnage qui n'impose aucune restriction quant à la longueur des blocs de données d'entrée. La fonction intégrée de MATLAB UPFIRDN.dll (ou RESAMPLE.m) a été écrite pour ne rééchantillonner qu'un seul bloc d'entrée de donnée. Elle ne fournit aucune information sur les variables d'état des filtres et, par conséquent, elle ne peut servir à rééchantillonner un très grand fichier de données.

Le présent rapport décrit un algorithme de rééchantillonnage suffisamment général qui pourra être utile dans de nombreuses applications pratiques de traitement des signaux. Pour obtenir la souplesse souhaitée, nous avons appliqué la structure de filtre FIR variable de telle manière qu'elle contrôle les variables d'état et permet des blocs d'entrée de longueur quelconque. En plus de cette souplesse, l'application logicielle de cet algorithme atteint un débit de traitement relativement élevé.

Executive summary

The resampling of a signal involves the conversion from the initial sampling rate to a new and different one. This is necessary in many practical applications because the sampling rate is often fixed while the desired sampling rate may depend on the application or a signal parameter, such as the symbol rate or channel spacing. It is a particular issue in digital filter bank receivers used to detect the presence of narrowband signals occupying channels over a range of frequencies. To minimize computational cost, filter bank receivers often use frequency domain techniques based on the Fast Fourier Transform (FFT) algorithm. Since it is usually desirable to select FFTs to have a length that is a power of two, the channelization frequencies may not align with the channel spacings used in a certain frequency bands. The situation is complicated further by the introduction of new channel spacing standards, e.g., 8.33 kHz. An example of this problem occurs with the Agilent Blackbird signal analysis system. For the standard sampling rate of 20.48 megasamples/s, FFT bin sizes are constrained to the set ... 1.25 kHz, 2.5 kHz, 5 kHz, 10 kHz, ... with the result that a channelization such as 8.33 kHz cannot be accommodated directly.

Although resampling algorithms have been developed and implemented in commercial software libraries, such as the Intel Signal Processing Library [2], these algorithms often have undesirable limitations. For example, a typical constraint is that the data blocks processed must be an integer multiple of the downsampling rate. This restriction simplifies the indexing in the code and reduces the complexity of the implementation. However, it decreases the flexibility and usefulness of the resulting program, since in some applications, the length of the input data blocks may not be an integer multiple of the downsampling rate. Another implementation of a resampling algorithm, the MATLAB built-in function UPFIRDN.dll (or RESAMPLE.m) is designed for resampling only single blocks of input data. It does not provide information on the state variables of the filters and, therefore, cannot be used to resample a very large data file. Consequently, there is a need for a resampling program that imposes no restriction on the length of the input data blocks. To obtain a more flexible resampling program we have implemented the periodically time varying FIR filter structure in such a way that it keeps track of the state variables and also makes provisions for input blocks of arbitrary length. The implementation is carried out in MEX in the MATLAB environment with the core of the program written in C. In addition to utilizing the very efficient periodically time-varying FIR filter structure, an attempt is made to minimize data movement in the implementation. It turns out that the implementation is not only very flexible but also works approximately twice as fast as the Matlab built-in function UPFIRDN.dll for non-trivial resampling ratios (that is, $L > 1$ and $M > 1$, with L denoting the upsampling factor and M denoting the downsampling factor).

The new algorithm is particularly useful in digital receiver systems where a continuous stream of signal data must be processed, for example, in the recovery of message content from a signal of arbitrary duration.

Sichun Wang, Brian Kozminchuk. 2001. DIGITAL SAMPLING RATE CONVERSION: PRINCIPLES AND IMPLEMENTATION. DREO TM 2001-032. Defence Research Establishment Ottawa.

Sommaire

Le rééchantillonnage d'un signal implique la conversion du taux d'échantillonnage initial à un taux différent. C'est souvent une nécessité dans les applications pratiques dont le taux d'échantillonnage est fixe, alors qu'il faudrait que ce taux d'échantillonnage varie selon l'application ou un paramètre du signal, tel le débit des symboles ou l'espacement entre. C'est un problème bien précis dans les récepteurs à banque de filtres numériques que l'on utilise pour détecter la présence de signaux à largeur de bande étroite qui occupent des dans une plage de fréquences. Pour réduire au minimum le coût des calculs. les récepteurs à banque de filtres font souvent appel des techniques du domaine de la fréquence fondées sur l'algorithme de la transformée rapide de Fourier (TRF). Comme il est généralement souhaitable de sélectionner des TRF pour avoir une longueur qui soit une puissance de 2, la resoloution en fréquence de découpage peut ne pas correspondre avec l'espacement tel des voies de certaines bandes de fréquences. La situation se complique du fait de l'introduction de nouvelles normes d'écartement de voies, tel 8,33 kHz. Un exemple de ce problème se retrouve dans le système d'analyse de signaux de Agilent Blackbird. Pour les taux d'échantillonnage standard de 20.48 mégaéchantillons/s, les longueurs des intervalles de la TRF sont restreintes à 1.25 kHz, 2.5 kHz, 5 kHz, 10 kHz, de sorte qu'un découpage de voies comme 8.33 kHz ne peut être décrit directement.

Bien qu'on ait développé des algorithmes de rééchantillonnage dans le commerce et qu'on les ait mis en application dans des bibliothèques de logiciels, comme la Intel Signal Processing Library, ces algorithmes sont souvent à la merci de limitations indésirables. Une contrainte fréquente est l'obligation que les blocs de données soit un multiple entier du taux de sous-échantillonnage. Une telle restriction facilite l'indexation du code et réduit la complexité de la mise en application. Cependant, le programme résultant est alors moins polyvalent et moins utile, car dans certaines applications, la longueur des blocs de données d'entrée ne peut pas être un multiple entier du taux de sous-échantillonnage. La fonction intégrée de MATLAB UPFIRDN.dll (ou RESAMPLE.m) a été écrite pour ne rééchantillonner qu'un seul bloc d'entrée de donnée. Elle ne fournit aucune information sur les variables d'état des filtres et, par conséquent, elle ne peut servir rééchantillonner un très grand fichier de données. Un programme de rééchantillonnage qui n'impose aucune restriction quant la longueur des blocs d'entrée est donc justifié. Pour réaliser tel programme plus polyvalent, nous avons appliqué une structure de filtre FIR temporelle à variation périodique qui contrôle les variables d'état et permet des blocs d'entrée de longueur variable. La mise en application est réalisée en MEX dans un environnement MATLAB, le noyau du programme étant rédigé en C. En plus d'exploiter la structure de filtre FIR temporelle à variation périodique très performante, nous avons tenté de réduire au minimum le mouvement des données dans cette mise en application. Cette mise en application s'est révélée moins souple que prévu, mais quand même deux fois plus rapide que la fonction intégrée MatLab UPFIRDN.dll pour des rapports de rééchantillonnage non triviaux (soit $L > 1$ et $M > 1$ o L est le facteur de sur-échantillonnage et M, le facteur de sous-échantillonnage).

Ce nouvel algorithme est particulièrement utile pour les récepteurs numériques dans lesquels le flot continu de données numérique doit être traité, par exemple, lors de la récupération du contenu d'un message dans un signal de durée quelconque.

Sichun Wang, Brian Kozminchuk. 2001. DIGITAL SAMPLING RATE CONVERSION: PRINCIPLES AND IMPLEMENTATION. DREO TM 2001-032. Centre de recherches pour la défense, Ottawa.

Table of contents

List of figures

This page intentionally left blank.

1. INTRODUCTION

The resampling of a signal involves the conversion from the initial sampling rate to a new and different one. This is often necessary in practical applications because the sampling rate is often fixed while the desired sampling rate may depend on the application or a signal parameter, such as the symbol rate or channel spacing. It is a particular issue in digital filter bank receivers used to detect the presence of narrowband signals occupying channels over a range of frequencies. To minimize computational cost, filter bank receivers often use frequency domain techniques based on the Fast Fourier Transform (FFT) algorithm. Since it is usually desirable to select FFTs to have a length that is a power of two, the channelization frequencies may not align with the channel spacings used in a certain frequency bands. The situation is complicated further by the introduction of new channel spacing standards, e.g., 8.33 kHz. An example of this problem occurs with the Agilent Blackbird signal analysis system. For the standard sampling rate of 20.48 megasamples/s, FFT bin sizes are constrained to the set ... 1.25 kHz, 2.5 kHz, 5 kHz, 10 kHz, ... with the result that a channelization such as 8.33 kHz cannot be accommodated directly.

Although resampling algorithms have been developed and implemented in commercial software libraries, such as the Intel Signal Processing Library [2], these algorithms often have undesirable limitations. For example, a typical constraint is that the data blocks processed must be an integer multiple of the downsampling rate. This restriction simplifies the indexing in the code and reduces the complexity of the implementation. However, it decreases the flexibility and usefulness of the resulting program, since in some applications, the length of the input data blocks may not be an integer multiple of the downsampling rate. Another implementation of a resampling algorithm, the MATLAB built-in function UPFIRDN.dll (or RESAMPLE.m) is designed for resampling only single blocks of input data. It does not provide information on the state variables of the filters and, therefore, cannot be used to resample a very large data file. Consequently, there is a need for a resampling program that imposes no restriction on the length of the input data blocks.

To obtain a more flexible resampling program than the ones currently available, we have implemented the periodically time varying FIR filter structure in such a way that it keeps track of the state variables and also makes provisions for input blocks of arbitrary length. The implementation is carried out in MEX in the MATLAB environment with the core of the program written in C. In addition to utilizing the very efficient periodically time-varying FIR filter structure, an attempt is made to minimize data movement in the implementation. It turns out that the implementation is not only very flexible but also works approximately twice as fast as the Matlab built-in function UPFIRDN.dll for non-trivial resampling ratios (that is, $L > 1$ and $M > 1$, with L denoting the upsampling factor and M denoting the downsampling factor).

This report is organized as follows. Section 2 summarizes the theoretical basis of digital sampling rate conversion and derives the periodically time-varying FIR filter

structure. Section 3 discusses the special implementation of the periodically time-varying FIR filter structure in MEX (named UPDNC6.c). Section 4 demonstrates the usage of two MATLAB functions UPDNM6.m and RESAM6.m based on UPDNC6.dll. Section 5 presents a comparison of the performance of our implementation UPDNC6.dll with that of the MATLAB implementation UPFIRDn.dll. Finally, section 6 presents some concluding remarks.

2. BASIC CONCEPTS OF DIGITAL SAMPLING RATE CONVERSION

Let $L > 0$ and $M > 0$ be two arbitrary positive integers which are relatively prime, that is, the greatest common divisor of L and M is 1. A baseband digital signal $x(n)$, with sampling rate F_0, can be upsampled by a factor of L and then downsampled by a factor of M to obtain another digital signal $y(m)$ with sampling rate $F_1 = F_0\left(\frac{L}{M}\right)$. In theory, upsampling (also called interpolation) and downsampling (also called decimation) are two separate processes and upsampling must precede downsampling in order to preserve the spectrum of the signal. In practice, the upsampling and downsampling processes are combined as one and can be implemented efficiently via the periodically time-varying FIR filter structure as discussed in the classical survey paper [1]. In this section, the main concepts of digital sampling rate conversion are discussed and the periodically time-varying FIR filter structure is derived.

2.1 Downsampling by an Integer Factor of M.

Let $x(n)$ be a baseband digital signal with sampling rate F_0. The sampling rate of $x(n)$ can be decreased by a factor of M by retaining only one sample in every M samples in $x(n)$. Specifically, if the sequence $y(m)$ is defined by $y(m) = x(mM)$, we say that $y(m)$ is obtained by downsampling $x(n)$ by a factor of M (or by decimating $x(n)$ by a factor M). Direct decimation of the signal $x(n)$ without first passing it through an appropriately designed low-pass filter will in general lead to aliasing in the resultant signal $y(m)$. To demonstrate this, one can calculate the z transform of $y(m)$. Let the sequence $x'(n)$ be defined by

(1) $$x'(n) = \begin{cases} x(n), & n \text{ is an integer multiple of M,} \\ 0, & \text{otherwise} \end{cases}$$

It can be verified that

(2) $$x'(n) = x(n)\left\{\frac{1}{M}\sum_{l=0}^{M-1} e^{j2\pi ln/M}\right\}$$

and

(3) $$y(m) = x'(mM)$$

Hence, $Y(z)$, the z transform of $y(m)$, can be calculated as follows:

$$\begin{aligned} Y(z) &= \sum_{m=-\infty}^{\infty} y(m) z^{-m} \\ &= \sum_{m=-\infty}^{\infty} x'(mM) z^{-m} \\ &= \sum_{m=-\infty}^{\infty} x'(m) z^{-m/M} \\ &= \sum_{m=-\infty}^{\infty} x(m) \left\{ \frac{1}{M} \sum_{l=0}^{M-1} e^{j2\pi lm/M} \right\} z^{-m/M} \\ &= \frac{1}{M} \sum_{l=0}^{M-1} \left[\sum_{m=-\infty}^{\infty} x(m) e^{j2\pi lm/M} z^{-m/M} \right] \\ &= \frac{1}{M} \sum_{l=0}^{M-1} X(e^{-j2\pi l/M} z^{1/M}) \end{aligned} \tag{4}$$

where $X(z)$ is the z transform of $x(n)$. Evaluating $Y(z)$ on the unit circle, $z = e^{j\omega}$, $\omega \in [-\pi, \pi]$, yields the Fourier transform of $y(m)$, i.e.,

$$Y(e^{j\omega}) = \frac{1}{M} \sum_{l=0}^{M-1} X(e^{j(\omega - 2\pi l)/M}) \tag{5}$$

¿From Eq. 5 it can be seen that in general $y(m)$ is an aliased version of the original signal $x(n)$.

To avoid aliasing in $y(m)$, it is necessary to first low-pass filter $x(n)$ and then perform decimation. In fact, let $h_d(n)$ be the impulse response of a low-pass FIR filter(called the prototype downsampling filter) and $H_d(z)$ be its z transform. Let the signal $w(n)$ be the output of the filter $H_d(z)$ with input $x(n)$ and let $y(m)$ be the signal obtained by decimating $w(n)$ by a factor of M (that is, $y(m) = w(Mm)$). Let $X(z)$, $W(z)$ and $Y(z)$ be the z transforms of $x(n)$, $w(n)$ and $y(m)$ respectively. It follows that

$$W(z) = H_d(z) X(z) \tag{6}$$

and (see Eq. 4, with $W(z)$ replacing $X(z)$)

$$Y(z) = \frac{1}{M} \sum_{l=0}^{M-1} W(e^{-j2\pi l/M} z^{1/M}) \tag{7}$$

Hence

$$\begin{aligned} Y(z) &= \frac{1}{M} \sum_{l=0}^{M-1} W(e^{-j2\pi l/M} z^{1/M}) \\ &= \frac{1}{M} \sum_{l=0}^{M-1} H_d(e^{-j2\pi l/M} z^{1/M}) X(e^{-j2\pi l/M} z^{1/M}) \end{aligned} \tag{8}$$

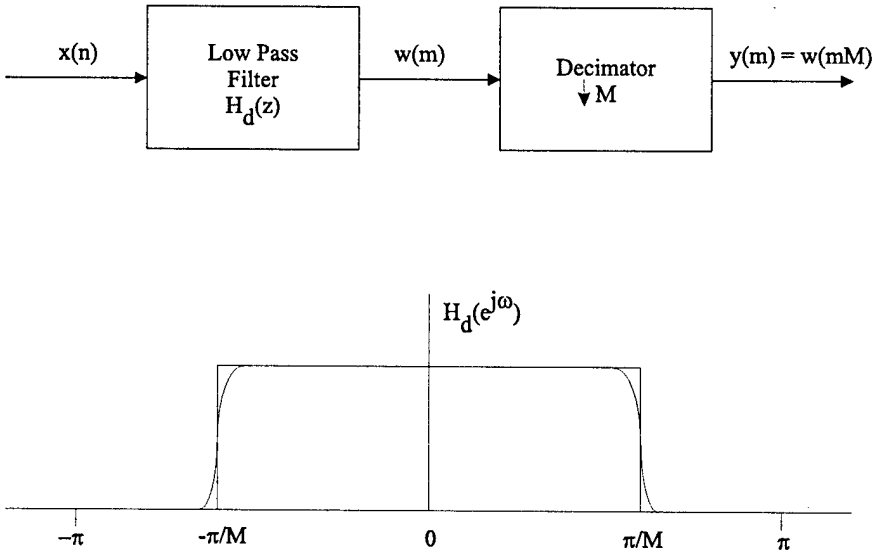


Figure 1: *Downsampler by a factor of M and prototype lowpass filter.*

Evaluating $Y(z)$ on the unit circle $z = e^{j\omega}$, $\omega \in [-\pi, \pi]$, yields the result

$$Y(e^{j\omega}) = \frac{1}{M}\sum_{l=0}^{M-1} H_d(e^{j(\omega-2\pi l)/M})X(e^{j(\omega-2\pi l)/M})$$

$$= \frac{1}{M}\left[H_d(e^{j\omega/M})X(e^{j\omega/M}) + H_d(e^{j(\omega-2\pi)/M})X(e^{j(\omega-2\pi)/M}) + \cdots\right] \quad (9)$$

If the prototype downsampling filter $H_d(z)$ is designed in such a way that

$$H_d(e^{j\omega}) \cong \begin{cases} 1, & |\omega| \le \frac{\pi}{M}, \\ 0, & \frac{\pi}{M} < |\omega| \le \pi \end{cases} \quad (10)$$

then the terms with $l \neq 0$ in Eq. 9 are removed and $Y(e^{j\omega})$ becomes

$$Y(e^{j\omega}) \cong \frac{1}{M}X(e^{j\omega/M}) \quad (11)$$

Hence by passing $x(n)$ through an appropriately designed FIR prototype downsampling filter $H_d(z)$ and then performing decimation, aliasing in $y(m)$ is eliminated. The process of downsampling a digital signal by a factor of M is illustrated in Fig. 1.

The time domain relationship among $x(n), y(m), h_d(k)$ is

$$y(m) = w(mM) = \sum_{k=-\infty}^{\infty} h_d(k)x(mM-k) \quad (12)$$

2.2 Upsampling by an Integer Factor of L.

Upsampling is the dual operation of downsampling. Suppose it is desired to increase the sampling rate of $x(n)$ by an integer factor of L. This can be accomplished by inserting $L-1$ zeros between each pair of samples in $x(n)$ and then low-pass filtering the resultant sequence. In fact, let the digital signal $w(n)$ be defined by

$$w(n) = \begin{cases} x(\frac{n}{L}), & n \text{ is an integer multiple of L,} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

The signal $w(n)$ is obtained by inserting $L-1$ zeros between each pair of samples of $x(n)$. Its z transform $W(z)$ is given by

$$\begin{aligned} W(z) &= \sum_{n=-\infty}^{\infty} w(n) z^{-n} \\ &= \sum_{n=-\infty}^{\infty} x(n) z^{-nL} \\ &= X(z^L) \end{aligned} \tag{14}$$

where $X(z)$ is the z transform of $x(n)$. Evaluating $W(z)$ on the unit circle , $z = e^{j\omega}$, $\omega \in [-\pi, \pi]$, gives the Fourier transform of $w(n)$

$$W(e^{j\omega}) = X(e^{j\omega L}) \tag{15}$$

Clearly, the spectrum of $w(n)$ preserves that of the original signal $x(n)$ on the interval $[0, \frac{\pi}{L}]$ but also contains harmonic images of the spectrum of $x(n)$ on the intervals $[\frac{\pi}{L}, \frac{3}{L}\pi], \cdots, [\frac{(L-1)}{L}\pi, \pi]$, which must be removed. This is illustrated in Figure 2 for the case $L = 4$.

To eliminate the unwanted harmonic images of the spectrum of $x(n)$, it is necessary to filter the signal $w(n)$ with a low-pass filter $H_u(z)$ (called the prototype upsampling filter) which approximates the ideal frequency response

$$H_u(e^{j\omega}) \cong \begin{cases} G, & |\omega| < \frac{\pi}{L}, \\ 0, & \frac{\pi}{L} < |\omega| \leq \pi \end{cases} \tag{16}$$

where $G = L$ is a necessary scaling factor (see Eq. 33 [1]). The process of upsampling a digital signal $x(n)$ by a factor of L is illustrated in Fig. 3.

The time domain relationship among $x(n), y(m), h_u(k)$, where $h_u(k)$ is the impulse response of the prototype upsampling filter $H_u(z)$, is given by

$$\begin{aligned} y(m) &= \sum_{k=-\infty}^{\infty} h_u(m-k) w(k) \\ &= \sum_{\frac{k}{L} \text{ is an integer}} h_u(m-k) x(k/L) \\ &= \sum_{k=-\infty}^{\infty} h_u(m-kL) x(k) \end{aligned} \tag{17}$$

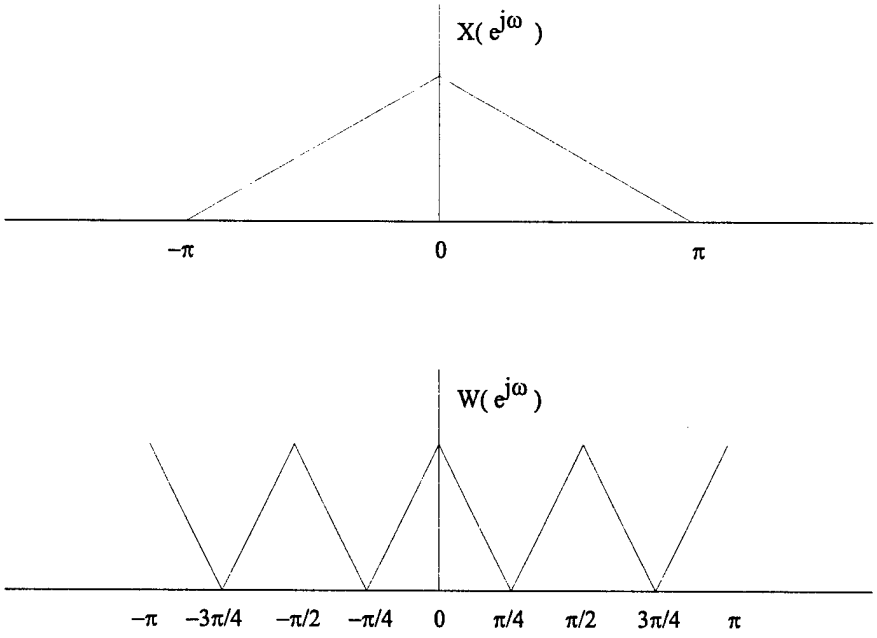


Figure 2: *Spectrum of $w(n)$ with $L-1$ zeroes inserted between samples.*

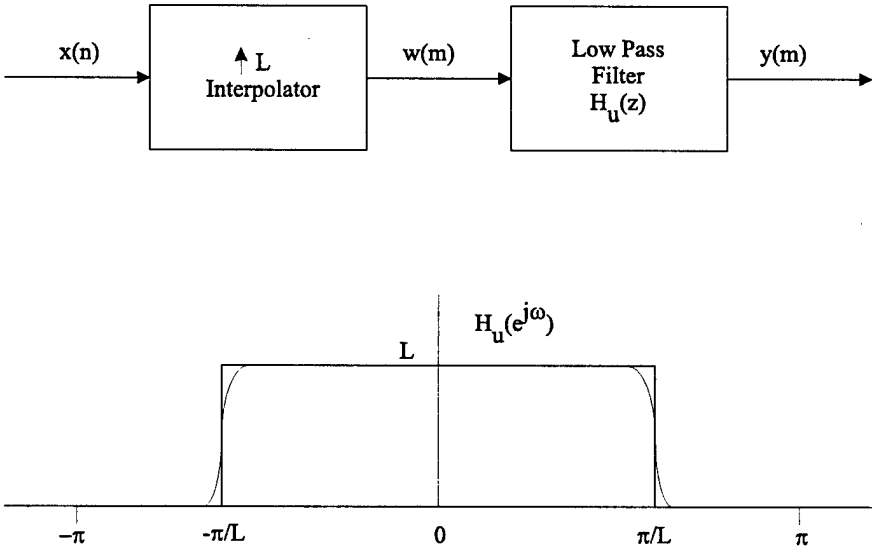


Figure 3: *Upsampler by factor of L followed by lowpass filter to remove harmonics.*

2.3 Resampling by a Rational Factor of $\frac{L}{M}$.

Changing the sampling rate of a digital signal by an arbitrary rational factor of $\frac{L}{M}$, where L and M are relatively prime positive integers, is accomplished by first increasing the sampling rate by a factor of L and then decreasing the sampling rate of the resultant signal by a factor of M. As demonstrated in the preceding two subsections, for the upsampling stage, the prototype upsampling filter $H_u(z)$ should approximate the frequency response characteristic

$$H_u(z) \cong \begin{cases} L, & |\omega| \leq \frac{\pi}{L}, \\ 0, & \frac{\pi}{L} < |\omega| \leq \pi \end{cases} \tag{18}$$

and for the downsampling stage, the prototype downsampling filter $H_d(z)$ should approximate the frequency response characteristic

$$H_d(z) \cong \begin{cases} 1, & |\omega| \leq \frac{\pi}{M}, \\ 0, & \frac{\pi}{M} < |\omega| \leq \pi \end{cases} \tag{19}$$

The complete resampling process is carried out by first inserting $L-1$ zeros between each pair of samples of the input signal $x(n)$, low-pass filtering the resultant signal by the filter $H_u(z)$, then low-pass filtering the output signal of $H_u(z)$ by the filter $H_d(z)$ and finally decimating the output signal of $H_d(z)$ by a factor of M. This is demonstrated in Fig. 4. This process can be further simplified by replacing the cascade of the prototype filters $H_u(z)$ and $H_d(z)$ by one low-pass filter $H(z) = H_u(z)H_d(z)$, which has the frequency response characteristic

$$H(z) = H_u(z)H_d(z) \cong \begin{cases} L, & |\omega| \leq \frac{\pi}{\max(M,L)}, \\ 0, & \frac{\pi}{\max(M,L)} < |\omega| \leq \pi \end{cases} \tag{20}$$

The low-pass filter $H(z)$ is called the prototype resampling filter. Conceptually, resampling is now reduced to three steps: first padding $L-1$ zeros between each pair of samples of the input signal $x(n)$, then low-pass filtering the resultant signal by the resampling filter $H(z)$, and finally decimating the output signal of $H(z)$ by a factor M. This is demonstrated in Fig. 5.

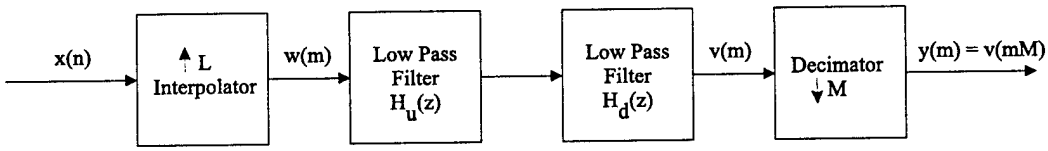


Figure 4: *The process of resampling a signal by a factor of L/M by upsampling, filtering, and downsampling.*

Let $h(k)$ be the impulse response of the prototype resampling filter $H(z)$ defined by Eq. 20. The time domain relationship among $x(n)$, $h(k)$ and $y(m)$ is

$$y(m) = v(mM) \tag{21}$$

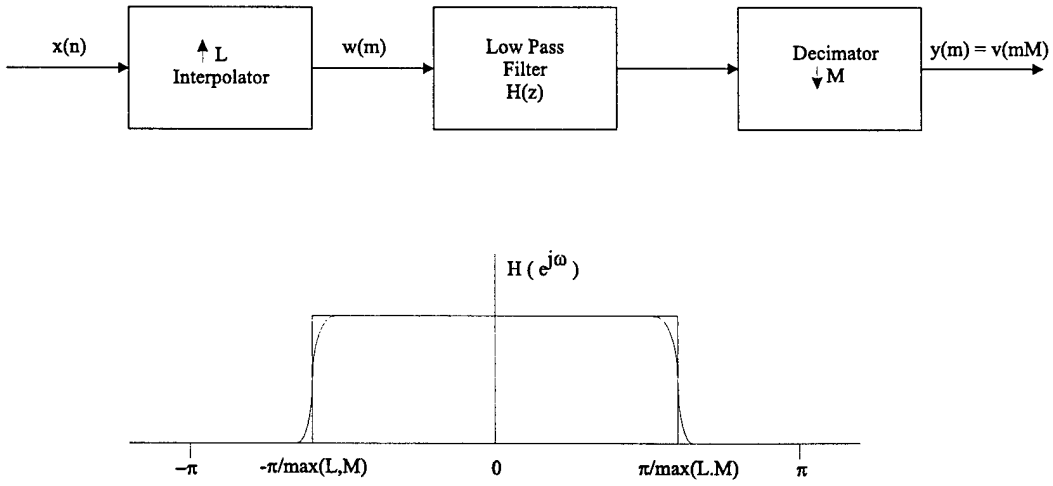


Figure 5: *The process of resampling a signal with a single prototype lowpass filter.*

where $v(n)$ is the output of the filter $H(z)$ and

$$v(m) = \sum_{k=-\infty}^{\infty} h(m - kL)x(k) \tag{22}$$

Therefore

$$y(m) = \sum_{k=-\infty}^{\infty} h(mM - kL)x(k) \tag{23}$$

Equation 23 is the mathematical basis for implementing digital sampling rate conversion.

2.4 The Periodically Time-varying FIR Filter Structure for Implementing Digital Sampling Rate Conversion.

Equation 23 can be put in a form more amenable to hardware or software implementation. Making the change of variables

$$k = \text{floor}\left(\frac{mM}{L}\right) - n \tag{24}$$

where floor $\left(\frac{mM}{L}\right)$ denotes the largest integer less than or equal to the rational number $\frac{mM}{L}$, one can write

$$\begin{aligned} mM - kL &= mM - \left(\text{floor}\left(\frac{mM}{L}\right) - n\right) L \\ &= nL + mM - \left(\text{floor}\left(\frac{mM}{L}\right)\right) L \\ &= nL + mM \oplus L \end{aligned} \tag{25}$$

where $mM \oplus L = mM - \left(\text{floor}\left(\frac{mM}{L}\right)\right) L$ denotes the remainder of mM after being divided by L (value of mM modulo L). Substituting Eqs. 24 and 25 in 23, we obtain

$$y(m) = \sum_{n=-\infty}^{\infty} h(nL + mM \oplus L)x\left(\text{floor}\left(\frac{mM}{L}\right) - n\right) \tag{26}$$

This is the actual resampling equation that is implemented in hardware or software.

In the practical implementation of Eq. 26, the prototype resampling filter $h(k)$ is assumed to be a linear phase FIR filter with length LQ , where $Q \geq 1$ is a positive integer. That is, the length of $h(k)$ is constrained to be an integer multiple of the upsampling rate L. Under this constraint, the algorithm Eq. 26 leads naturally to the periodically time-varying FIR filter structure for sampling rate conversion.

Some new notation will now be introduced. The prototype resampling filter $h(k)$, where $0 \leq k \leq LQ - 1$, can be partitioned into L polyphase filters $h_m(n)$ $(0 \leq m \leq L - 1)$, each of length Q, i.e.,

$$h_m(n) = h(nL + m), \;\; 0 \leq n \leq Q - 1 \tag{27}$$

In other words, $h_m(n)$ is obtained by decimating the sequence $h(k)$ by a factor of L. To visualize the relationship between the prototype filter $h(k)$ (with length $N = LQ$) and the polyphase filters $h_m(n)$ (each with length Q), the $Q \times L$ matrix $\mathbf{H}$ is constructed, i.e.,

$$\mathbf{H} = \begin{pmatrix} h(0) & h(1) & h(2) & \cdots & h(L-1) \\ h(L) & h(L+1) & h(L+2) & \cdots & h(2L-1) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ h(nL) & h(nL+1) & h(nL+2) & \cdots & h((n+1)L-1) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ h((Q-1)L) & h((Q-1)L+1) & h((Q-1)L+2) & \cdots & h(QL-1) \end{pmatrix} \tag{28}$$

$\mathbf{H}$ has L columns, the first column consisting of the taps of the polyphase filter $h_0(n)$, the second column consisting of the taps of the polyphase filter $h_1(n)$ and so on. On the other hand, the filter taps $h(k)$ can be recovered from $\mathbf{H}$ by concatenating the row vectors of $\mathbf{H}$ from the first to the last row. Define two integer sequences $p(m)$ and $q(m)$ by setting

$$p(m) = mM \oplus L, \;\; q(m) = \textbf{floor}\left(\frac{mM}{L}\right) \tag{29}$$

It can be verified that $p(m)$ is a periodic sequence with period L and $q(m)$ has the simple property that $q(m + kL) = kM + q(m)$, i.e.,

$$p(m + kL) = p(m) = p(m \oplus L), \;\; q(m + kL) = kM + q(m), \tag{30}$$

Using this new notation, Eq. 26 can be rewritten as

$$\begin{aligned} y(m) &= \sum_{n=0}^{Q-1} h(nL + mM \oplus L) x\left(\textbf{floor}\left(\frac{mM}{L}\right) - n\right) \\ &= \sum_{n=0}^{Q-1} h_{p(m)}(n) x(q(m) - n) \\ &= \sum_{n=0}^{Q-1} h_{p(m \oplus L)}(n) x(q(m) - n) \end{aligned} \tag{31}$$

Let

$$g_m(n) = h_{p(m \oplus L)}(n),\ 1 \le m \le L\ ,\ 0 \le n \le Q-1 \tag{32}$$

It follows from Eqs. 31 and 32 that

$$y(kL + m) = \sum_{n=0}^{Q-1} g_m(n) x(kM + q(m) - n) \tag{33}$$

where $1 \le m \le L$ and $k \ge 0$. The equation (33) can be interpreted as representing a periodically time-varying FIR filter. To illustrate this, write out Eq. 33 for $1 \le m \le L$, $k = 0, 1$, as follows:

$$\begin{aligned} y(1) &= g_1(0)x(q(1)) + g_1(1)x(q(1)-1) + \cdots + g_1(Q-1)x(q(1)-(Q-1)) \\ y(2) &= g_2(0)x(q(2)) + g_2(1)x(q(2)-1) + \cdots + g_2(Q-1)x(q(2)-(Q-1)) \\ &\cdots\cdots \\ y(m) &= g_m(0)x(q(m)) + g_m(1)x(q(m)-1) + \cdots + g_m(Q-1)x(q(m)-(Q-1)) \\ &\cdots\cdots \\ y(L) &= g_L(0)x(q(L)) + g_L(1)x(q(L)-1) + \cdots + g_L(Q-1)x(q(L)-(Q-1)) \\ &= g_L(0)x(M) + g_L(1)x(M-1) + \cdots + g_L(Q-1)x(M-(Q-1)) \\ y(L+1) &= g_1(0)x(M+q(1)) + \cdots + g_1(Q-1)x(M+q(1)-(Q-1)) \\ y(L+2) &= g_2(0)x(M+q(2)) + \cdots + g_2(Q-1)x(M+q(2)-(Q-1)) \\ &\cdots\cdots \\ y(L+m) &= g_m(0)x(M+q(m)) + \cdots + g_m(Q-1)x(M+q(m)-(Q-1)) \\ &\cdots\cdots \\ y(2L) &= g_L(0)x(2M) + g_L(1)x(2M-1) + \cdots + g_L(Q-1)x(2M-(Q-1)) \end{aligned} \tag{34}$$

The first equation in 34 shows that $y(1)$ is obtained as a weighted sum of Q sequential samples of $x(n)$ starting at the sample $x(q(1))$ and going backwards in n sequentially. The weighting coefficients are the taps of the polyphase filter $g_1(n)$, $0 \le n \le Q-1$. Similarly, $y(2)$ is obtained as a weighted sum of Q sequential samples of $x(n)$ starting

at the sample $x(q(2))$ and going backwards in n sequentially. The weighting coefficients are the taps of a different polyphase filter, namely $g_2(n)$. This pattern continues until the sample $y(L)$, with $y(L)$ computed as a weighted sum of Q sequential samples of $x(n)$ starting at the sample $x(q(L)) = x(M)$ and going backwards in n sequentially. The weighting coefficients in the computation of $y(L)$ are the taps of the polyphase filter $g_L(n)$. It is clear that different sets of filter coefficients (namely, $g_m(n)$) are used in the computation of the samples $y(m)$, $m = 1, 2, \cdots, L$. Now let us look at the computational procedure for the next L samples, $y(L+1), y(L+2), \cdots, y(2L)$. It can be observed from the equations in 34 that $y(L+1)$ is computed using the same polyphase filter as $y(1)$ (that is, $g_1(n)$). Also $y(L+2)$ is generated using the same polyphase filter as $y(2)$ (that is, $g_2(n)$), etc. Hence the sample sequence $y(m)$ is the output of a periodically time-varying FIR filter with period L.

It is also important to understand how the input data samples $x(n)$ enter into the computation in Eq. 33. Writing out Eq. 33 for $y(m)$ and $y(m+1)$, where $1 \leq m \leq L-1$, yields

$$y(m) = g_m(0)x(q(m)) + g_m(1)x(q(m)-1) + \cdots + g_m(Q-1)x(q(m)-(Q-1)) \tag{35}$$

and

$$y(m+1) = g_{(m+1)}(0)x(q(m+1)) + g_{(m+1)}(1)x(q(m+1)-1) + \cdots + g_{(m+1)}(Q-1)x(q(m+1)-(Q-1)) \tag{36}$$

One can see that $y(m)$ is the weighted sum of Q sequential samples of $x(n)$ starting at the sample $x(q(m))$ and going backwards sequentially. The samples from the input sequence $x(n)$ which are involved in the weighted sum are

$$\mathbf{S} = [x(q(m)), x(q(m)-1), \cdots, x(q(m)-(Q-2)), x(q(m)-(Q-1))] \tag{37}$$

The samples $x(q(m)), x(q(m)-1), \cdots, x(q(m)-(Q-2)), x(q(m)-(Q-1))$ are called the state variables of the time-varying FIR filter in Eq. 33 and the vector $\mathbf{S}$ is called the state variable buffer corresponding to the output sample $y(m)$. In the computation of $y(m+1)$, the state variable buffer has changed to

$$\mathbf{S} = [x(q(m+1)), x(q(m+1)-1), \cdots, x(q(m+1)-(Q-1))] \tag{38}$$

If $q(m+1) = \mathbf{floor}\left(\frac{(m+1)M}{L}\right) > q(m) = \mathbf{floor}\left(\frac{mM}{L}\right)$, there are $q(m+1) - q(m)$ new samples shifted into and $q(m+1) - q(m)$ old samples shifted out of the state variable buffer $\mathbf{S}$. On the other hand, if $q(m+1) = q(m)$, then the state variable buffer $\mathbf{S}$ remains unchanged. It should be noted that the sequence $q(m)$ is non-decreasing and it is $q(m)$ that determines the input samples $x(n)$ entering into the computation of the output sample $y(m)$. Recall that $q(m+kL) = kM + q(m)$ (see Eq. 30), hence for $1 \leq m \leq L-1$, $k \geq 0$,

$$\begin{aligned} q(m+1+kL) - q(m+kL) &= kM + q(m+1) - (kM + q(m)) \\ &= q(m+1) - q(m) \end{aligned} \tag{39}$$

Equation 39 implies that the pattern in which the data samples $x(n)$ are shifted into and out of the state variable buffer $\mathbf{S}$ also repeats itself periodically. Let us examine in some detail how the state variable buffer $\mathbf{S}$ is updated for the first L samples $y(1), y(2), \cdots, y(L)$. Denoting the contents of the state variable buffer $\mathbf{S}$ corresponding to the samples $y(1), y(2), \cdots, y(L)$ by $\mathbf{S}_1, \cdots, \mathbf{S}_L$, respectively, we have

$$\begin{aligned}
\mathbf{S}_1 &= [x(q(1)), x(q(1)-1), \cdots, x(q(1)-(Q-1))] \\
\mathbf{S}_2 &= [x(q(2)), x(q(2)-1), \cdots, x(q(2)-(Q-1))] \\
\mathbf{S}_3 &= [x(q(3)), x(q(3)-1), \cdots, x(q(3)-(Q-1))] \\
&\cdots\cdots \\
\mathbf{S}_L &= [x(q(L)), x(q(L)-1), \cdots, x(q(L)-(Q-1))] \\
&= [x(M), x(M-1), \cdots, x(M-(Q-1))]
\end{aligned} \tag{40}$$

There are three cases to be considered: (1) $q(1) \geq Q$, (2) $1 \leq q(1) < Q$, and (3) $q(1) < 1$. It is assumed that $1 \leq q(1) < Q$, i.e., $1 \leq M/L < Q$. The other two cases are different but very similar. The input samples $x(M), x(M-1), x(M-2), \cdots, x(2), x(1), x(0), x(-1), \cdots, x(q(1)-(Q-1))$ are involved in the computation of the first L output samples $y(1), y(2), \cdots, y(L)$. If it is assumed that the input samples $x(n)$ start at the sample $x(1)$, then the samples $x(0), \cdots, x(q(1)-(Q-1))$ in the first state variable buffer $\mathbf{S}_1$ are not available and have to assume certain given values (usually zeros). Thus to compute $y(1)$, the state variable buffer $\mathbf{S}$ is first initialized to be an all-zero vector and then $q(1)$ samples $x(q(1), x(q(1)-1), x(1)$ are shifted into it. The samples in the buffer $\mathbf{S}$ are then weighted with the coefficients of the polyphase filter $g_1(n)$. The most recent sample in the state variable buffer is $x(q(1))$. To compute $y(2)$, $q(2)-q(1)$ new samples from $x(n)$ are shifted into $\mathbf{S}$ and $q(2)-q(1)$ old samples are shifted out of $\mathbf{S}$. The samples in the updated buffer are weighted with the coefficients of the polyphase filter $g_2(n)$. The most recent sample in the state variable buffer has become $x(q(2))$. In general, to compute $y(m)$, $3 \leq m \leq L-1$, $q(m)-q(m-1)$ new samples are shifted into the buffer $\mathbf{S}$ and $q(m)-q(m-1)$ old samples are shifted out of the buffer $\mathbf{S}$. The samples in the buffer are weighted with the coefficients of the polyphase filter $g_m(n)$. To compute the last sample $y(L)$ in the first block of L output samples, $q(L)-q(L-1) = M-q(L-1)$ samples are shifted into and out of the buffer $\mathbf{S}$ and the samples in the state variable buffer are weighted with the coefficients of the polyphase filter $g_L(n)$. The most recent sample in the buffer has become $x(M)$. Note that the first M input samples $x(M), x(M-1), \cdots, x(1)$ have passed into the buffer $\mathbf{S}$ during the computation of the first L output samples $y(L), y(L-1), \cdots, y(1)$.

The pattern in which the data samples $x(n)$ are shifted into the state variable buffer $\mathbf{S}$ during the computation of the next block of L output samples $y(L+1), y(L+2), \cdots, y(2L)$ will now be examined. Denoting the contents of the state variable buffer $\mathbf{S}$ corresponding to the samples $y(L+1), y(L+2), \cdots, y(2L)$ by $\mathbf{S}_{L+1}, \cdots, \mathbf{S}_{2L}$, respectively , we have

$$\mathbf{S}_{L+1} = [x(M+q(1)), x(M+q(1)-1), \cdots, x(M+q(1)-(Q-1))]$$

$$
\begin{aligned}
\mathbf{S}_{L+2} &= [x(M+q(2)), x(M+q(2)-1), \cdots, x(M+q(2)-(Q-1))] \\
\mathbf{S}_{L+3} &= [x(M+q(3)), x(M+q(3)-1), \cdots, x(M+q(3)-(Q-1))] \\
&\cdots\cdots \\
\mathbf{S}_{2L} &= [x(M+q(L)), x(M+q(L)-1), \cdots, x(M+q(L)-(Q-1))] \\
&= [x(2M), x(2M-1), \cdots, x(2M-(Q-1))]
\end{aligned}
\tag{41}
$$

After the computation of $y(L)$ and before the computation of $y(L+1)$, the state variable buffer $\mathbf{S}$ is

$$
\begin{aligned}
\mathbf{S}_L &= [x(q(L)), x(q(L)-1), \cdots, x(q(L)-(Q-1))] \\
&= [x(M), x(M-1), \cdots, x(M-(Q-1))]
\end{aligned}
\tag{42}
$$

To compute $y(L+1)$, $q(1)$ new samples are shifted into $\mathbf{S}$ and $q(1)$ old samples are shifted out of $\mathbf{S}$. The samples in the buffer are weighted with the same set of coefficients as in the computation of $y(1)$, namely, the polyphase filter $g_1(n)$. To compute $y(L+2)$, $q(2)-q(1)$ new samples are shifted into $\mathbf{S}$ and out of $\mathbf{S}$. The samples in the buffer are weighted with the same set of coefficients as in the computation of $y(2)$, namely, the polyphase filter $g_2(n)$. It is now apparent that just as the polyphase filters enter into the computation of the output samples $y(m)$ periodically, the input data samples $x(n)$ are shifted into the state variable buffer in a manner that also repeats itself periodically. Thus the computation of the output samples $y(m)$ can be done most conveniently on a block by block basis. For each block of input samples of length M, a block of output samples of length L can be computed. This is illustrated in the diagram of Fig. 6.

3. IMPLEMENTATION OF THE PERIODICALLY TIME-VARYING FIR FILTER STRUCTURE IN MEX

Implementation of the periodically time varying FIR filter structure (Eq. 33) in some software packages, including the Intel Signal Processing Library [2], requires that the length of each input data block from the sequence $x(n)$ be an integer multiple of the downsampling factor M. On the one hand, this restriction considerably simplifies the indexing in the code and reduces the complexity of implementation. On the other hand, it also, to a certain degree, unnecessarily limits the flexibility and usefulness of the resulting program, since in certain applications the length of the input data blocks may not be an integer multiple of the downsampling factor M. For such applications, there is a need for an implementation of the resampling algorithm (Eq. 33) that imposes no restriction on the length of the input data blocks. The MATLAB built-in function UPFIRDN.dll (or RESAMPLE.m) is designed for resampling only one block of input data. It does not provide information on the state variables and cannot be used to resample a huge data file. To obtain a more flexible resampling program than the ones currently available, we have implemented the periodically time varying FIR filter

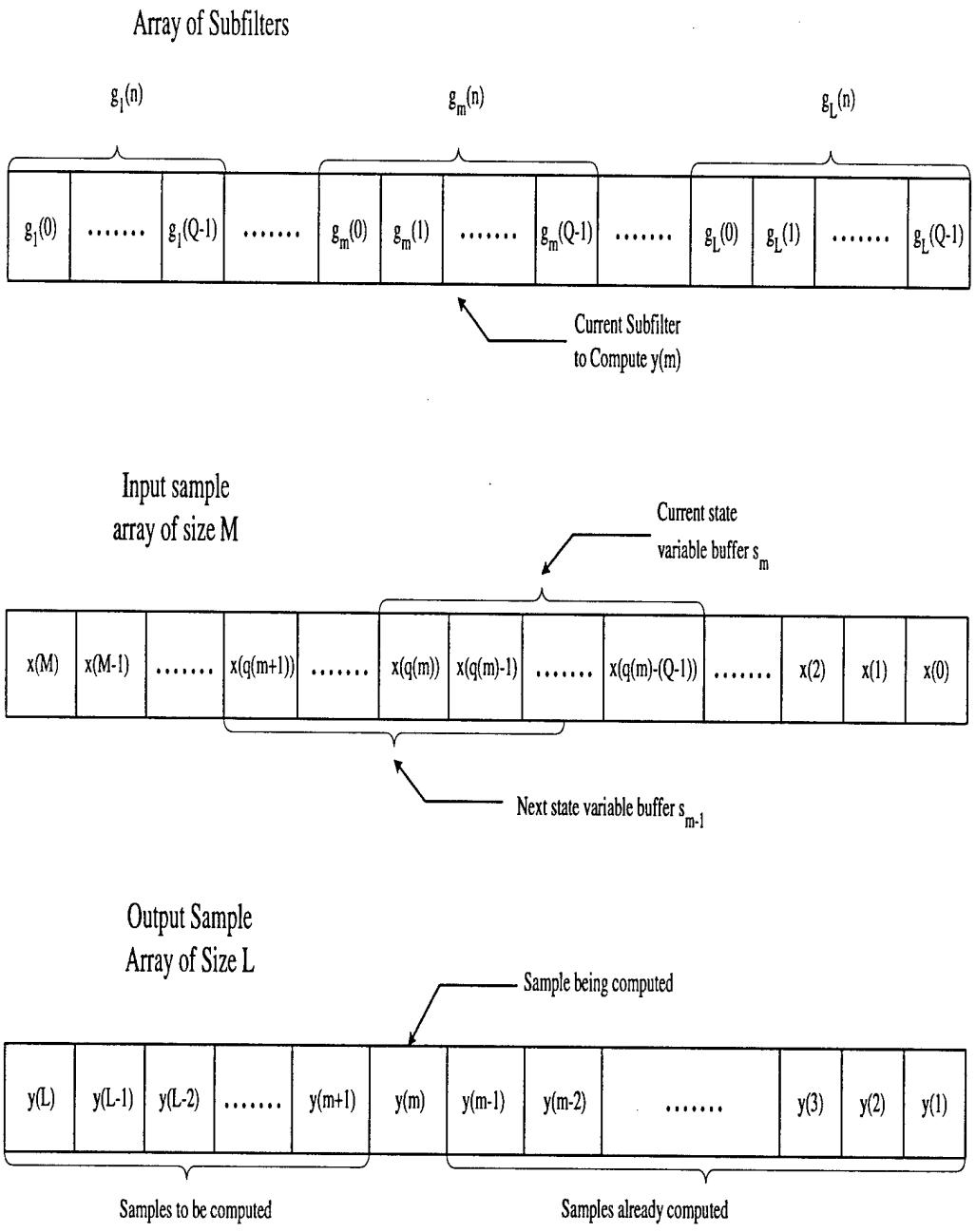


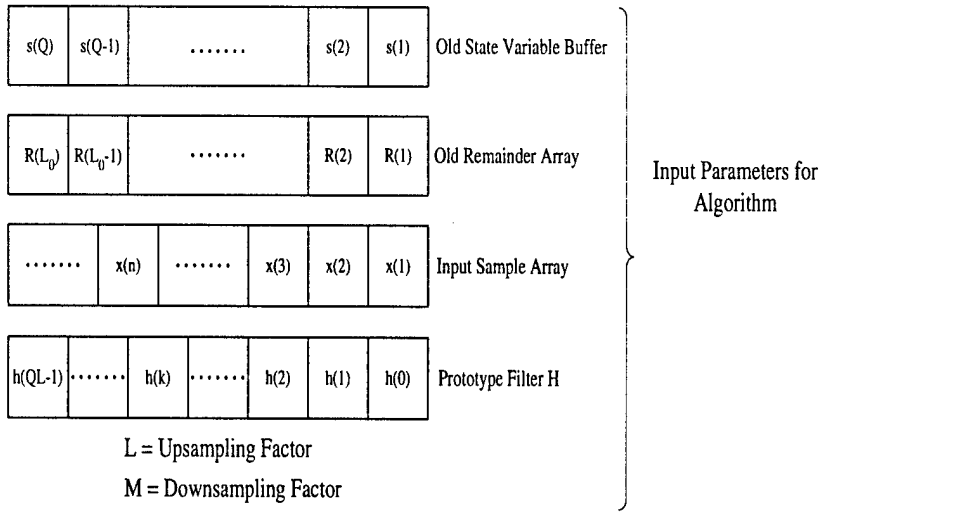
***Figure 6:** Array of subfilters.*

structure of Eq. 33 in such a way that it keeps track of the state variables and also makes provisions for input blocks of arbitrary length. The implementation is carried out in MEX in the MATLAB environment with the core of the program written in C. In addition to utilizing the very efficient periodically time-varying FIR filter structure, we also strive to minimize data movement in our implementation. It turns out that our implementation is not only very flexible but also works almost twice as fast as the Matlab built-in function UPFIRDN.dll if the resampling ratio $\frac{L}{M}$ is non-trivial (that is, $L > 1$ and $M > 1$).

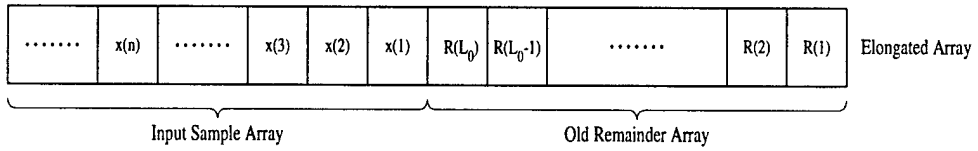
How the program works will now be explained. It is assumed that the input signal sequence is fed into the computer memory on a block by block basis. To make the program general enough, the individual input data blocks are not assumed to have equal length. Hence each individual input data block can be of any length. Once a data block is fed into the computer memory, it is resampled and the resampled data is then stored in the computer memory for further processing or output to a disk for storage. Since the algorithm in Eq. 33 is much easier to implement for an input data block whose size is an integer multiple of the downsampling factor M, we divide each input data block into two segments. The length of the first segment is an integer multiple of M(in some extreme cases it may be empty) and the second segment consists of less than M samples (it may also be empty). The samples in the first segment are processed in blocks of M samples each according to the scheme discussed in the preceding section (see Fig. 6). The remaining samples at the end of the input data block, which number less than M, are not processed. They are returned in an array called REMAINDER. The program will add the samples in the REMAINDER array to the beginning of the next input data block to form a new elongated array. This new input data block is again divided into two segments. The length of the first segment is an integer multiple of M and the second segment consists of less than M samples. Again the first segment is processed and the second segment is returned in the REMAINDER array.

More specifically, the processing in the program is carried out in three steps which are depicted in the diagram in Fig. 7. The first step in the program is to append the old remainder array of length L_0 (if it is not empty) to the beginning of the input data array. This step results in an elongated input data array. It is emphasized that this is accomplished via pointer manipulation and no data are really moved in the process. After this is done, the elongated input data array is partitioned into two segments. The length of the first segment is an integer multiple of M and there are t such blocks each of size M (t may be zero). The second segment consists of less than M samples (L_1 samples). The first segment is processed on a block by block basis as explained earlier and the results are returned, which make up an output array of length tL consisting of t blocks each of size L. The second segment is returned as the new remainder array. The contents of the state variable buffer right after processing the t input blocks of size M are returned as the new state variable buffer samples.

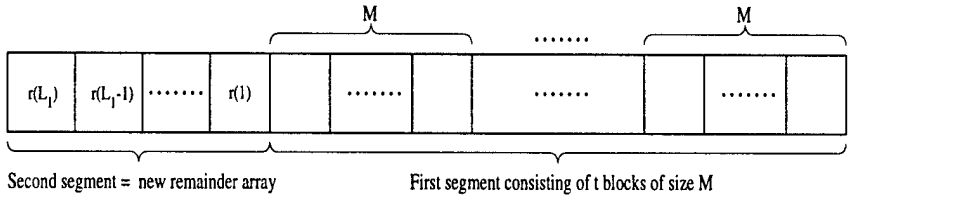
In the program, there is an interface component (named the MexFunction) that connects the MATLAB environment with the actual data processing performed in C. A large



Step 1: Append old remainder array **R** to input array **x** to produce an elongated array.



Step 2: Partition the elongated array into two segments. The length of the first segment is a multiple of M, and the length of the second segment is less than M.



Step 3: Process first segment from Step 2, resulting in an output array of length tL consisting of t blocks of length L.

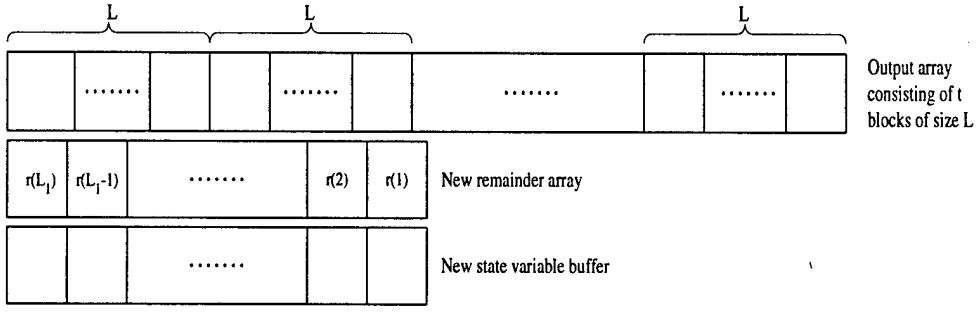


***Figure 7:** The three steps of processing in the program.*

portion of indexing management is done in this interface component. As pointed out in the preceding section, for each of the three separate cases, $q(1) > Q$, or $1 \leq q(1) < Q$ or $q(1) < 1$, the manner in which data are shifted into and out of the state variable buffer is different and the subscripting of the data arrays has to be managed in the program accordingly. To simplify the indexing and reduce the complexity of the code, three C functions, RESAMP1, RESAMP2 and RESAMP3, which, respectively, correspond to the three separate cases, $q(1) \geq Q$, or $1 \leq q(m) < Q$, or $q(m) < 1$, are written in the program. Since the data processing procedure is exactly the same for both the real and imaginary parts of the data sample blocks, the input data sample blocks are separated into the real and imaginary parts and the C functions RESAMP1, RESAMP2 and RESAMP3 are written to accept double precision real data arrays and to return double precision real data arrays. The C function RESAMP2 is discussed in some detail here. The other two functions are very similar. The prototype for the function RESAMP2 is shown below, and the parameters for this function are defined in Table 3.

```
RESAMP2( double Input[ ],  long InputLen,
         double Output[ ], long OutputLen,
         double State[ ], int StateLen,
         double Filter[ ], int FilterLen,
         double Remainder[ ], int RemainderLen,
         double Newremainder[ ], int NewremainderLen,
         int L, int M );
```

In the C function RESAMP2, the prototype filter array Filter[ ] is rearranged to result in a new filter array named firfilter[ ], which corresponds to the array of polyphase filters in the upper row in Fig. 6. Two arrays, named yindxmodL[ ] and newyindxmodL[ ] respectively, are defined to hold the control sequences $q(m)$ and $p(m)$, $1 \leq m \leq L$. The program completes the processing of data blocks of size M in the **for** loops, returns the remainder samples and updates the state variable samples. The complete MEX file is named UPDNC6.c. After compilation, UPDNC6.dll can be invoked directly in the MATLAB environment. The usage of UPDNC6.dll and two related MATLAB functions is discussed in the next section. For details of the C functions, RESAMP1, RESAMP2 and RESAMP3, the reader is referred to the file UPDNC6.c.

4. USAGE OF THE FUNCTIONS UPDNM6.m AND RESAM6.m

Although the program UPDNC6.dll can be used directly in the MATLAB environment, we have written another two MATLAB functions named UPDNM6.m and RESAM6.m respectively, which are more convenient to apply in certain situations. The function UPDNM6.m calls UPDNC6.dll and the function RESAM6.m in turn calls

Table 1: *Parameters for Resampler Program*

Parameter	Description
Input[ ]	Input data array
InputLen	Input data array length, integer multiple of M
Output[ ]	Output array
OutputLen	Output array length, integer multiple of L
State[ ]	Old state variable buffer from earlier input data block
StateLen	State variable buffer length Q =length(Filter[])/L
Filter[ ]	Taps of prototype filter, $h(k), 0 \leq k \leq LQ-1$
FilterLen	Length of prototype resampling filter
Remainder[ ]	Old remainder array
RemainderLen	Length of old remainder
Newremainder[ ]	New remainder array
NewremainderLen	Length of new remainder array
L	Upsampling factor
M	Downsampling factor

UPDNM6.m. Their usage is explained here.

4.1 Usage of UPDNM6.m.

The MATLAB function UPDNM6.m is called with the syntax:

```
                [Y,NEWSTATE,NEWREMAINDER,NEWYINDEX] =
(43)            updnm6(X,H,L,M,STATE,REMAINDER,YINDEX)
```

where the inputs are:

1. X: input signal with arbitrary length, where X can be real or complex;
2. H: prototype resampling filter designed by the user;
3. L: upsampling factor, where L must be a positive integer;
4. M: downsampling factor, M a positive integer; L and M relatively prime;
5. STATE: old state variables, length(STATE)=length(H)/L;
6. REMAINDER: samples left over from the preceding block of input data;
7. YINDEX: length of the resampled data obtained by resampling the input data before the current input block X;

The outputs are:

1. Y: the resampled data from processing all the data blocks of size M contained in X;
2. NEWSTATE: updated state variables;
3. NEWREMAINDER: samples left over from the current block X;
4. NEWYINDEX: length of the resampled data obtained by resampling input data up to and including the current block X;

When using UPDNM6.m, it is necessary for the user to supply the prototype resampling filter H. It should be emphasized that there is one restriction on the length of H, namely, the length of H must be an integer multiple of the upsampling factor L. If the user does not want to design the prototype resampling filter H, we have written another MATLAB function which designs a prototype resampling filter for the user. This program is named RESAM6.m. Its usage is explained next.

4.2 Usage of RESAM6.m.

The MATLAB function UPDNM6.m is called with the syntax

$$\begin{aligned} &[\text{Y,NEWSTATE,NEWREMAINDER,NEWYINDEX,H}] \\ (44) \qquad &= \text{resam6(X,L,M, STATE, REMAINDER,YINDEX)} \end{aligned}$$

where the inputs are:

1. X: Current input signal with arbitrary length;
2. L: Upsapling rate, L must be a positive integer;
3. M: Downsampling rate, M must be a positive integer; L and M must be relatively prime;
4. STATE: Old state variables after processing the input data block which precedes the current input block X;
5. REMAINDER: Input samples left over from the input data block which precedes the current input block X;

The outputs are:

1. Y: The resampled data from processing all the input data blocks of size M contained in X;
2. NEWSTATE: updated state variables;
3. NEWREMAINDER: input samples left over from the input block X;
4. NEWYINDEX: length of the resampled data Y.
5. H: FIR filter designed and used in the resampling operation;

The function RESAM6.m does the following:

1. Creates a prototype resampling filter H, where H is designed using the following MATLAB parameters:

$$\begin{aligned} r_p &= 1\ (\text{dB});\ \text{—passband ripple} \\ r_s &= 60\ (\text{dB});\ \text{—stopband ripple} \\ F_s &= 2\ (\text{Hz});\ \text{—normalized sampling frequency} \\ f &= [\,1/\max(L,M),\ \ 1.5/\max(L,M)\,];\ \text{—normalized cutoff frequencies} \\ a &= [1,\ 0];\ \text{—desired amplitudes} \\ dev &= [\tfrac{(10^{(\frac{r_p}{20})}-1)}{(10^{(\frac{r_p}{20})}+1)},\ 10^{(-\frac{r_s}{20})}];\ \text{—deviations from the ideal response} \\ [n,\ f_0,\ a_0,\ w] &= \text{remezord}(f,\ a,\ dev,\ F_s);\ \text{—filter order estimation} \\ n &= \text{ceil}(n/L) * L - 1;\ \text{—choosing filter length} \\ \text{H} &= L * \text{remez}(n,\ f_0,\ a_0,\ w);\ \text{—design filter with the REMEZ function} \\ \text{H} &= \text{H(:)}; \end{aligned}$$

2. Resample the data block X using the filter H by calling the function UPDNM6.m.

5. PERFORMANCE COMPARISON

To obtain an idea of how our function UPDNC6.dll compares with the MATLAB built-in function UPFIRDN.dll in performance, we tested them for various resampling ratios $\frac{L}{M}$ and various input block lengths. The personal computer used in the simulations has a clock rate of 200 MHz. The computer processing time (in seconds) is plotted as a function of the length of the input block in Figs. 8 to 30. It can be observed from Fig. 8 to 13 that if the downsampling rate $M = 1$, our function UPDNC6.dll outperforms the MATLAB implementation UPFIRDN.dll for shorter filter lengths and has roughly the same performance as UPDFIRDN.dll for longer filter lengths. If the upsampling rate $L = 1$, the MATLAB implementation UPFIRDN.dll outperforms our implementation UPDNC6.dll for shorter filter lengths but has roughly the same performance as ours for longer filter lengths. This is demonstrated in Figs. 14 to 19. If the resampling ratio $\frac{L}{M}$ is non-trivial, that is, if $L > 1$ and $M > 1$, then our implementation UPDNC6.dll outperforms the MATLAB implementation UPFIRDN.dll by a factor of about 2 for all filter lengths. This is demonstrated in Figs. 20 to 30.

6. CONCLUDING REMARKS

We have successfully implemented in software the periodically time-varying FIR filter structure in digital sampling rate conversion and obtained a very flexible resampling function UPDNC6.dll. This program outperforms the MATLAB built-in function UPFIRDN.dll by a factor of about 2 for non-trivial resampling ratios $\frac{L}{M}$ (that is, L and M are relatively prime and $L > 1$ and $M > 1$).

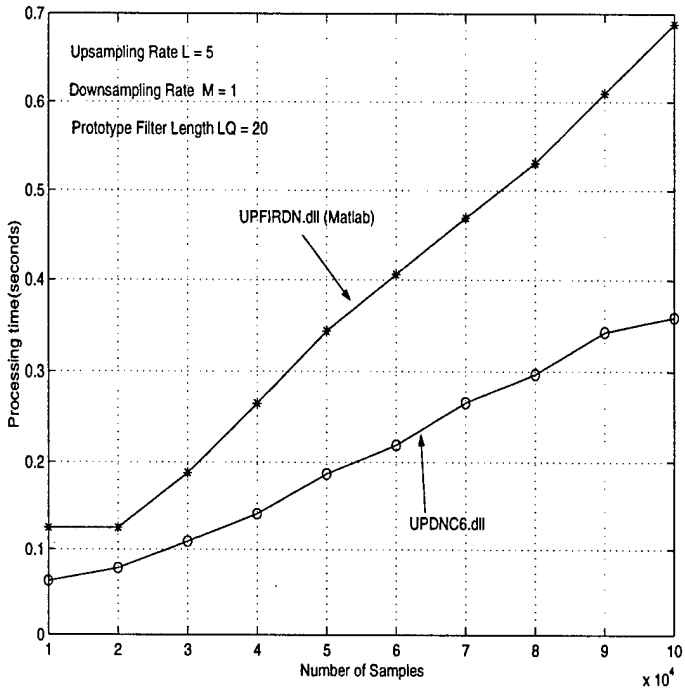


Figure 8: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor* $L = 5$*, downsampling factor* $M = 1$*, and filter length = 20.*

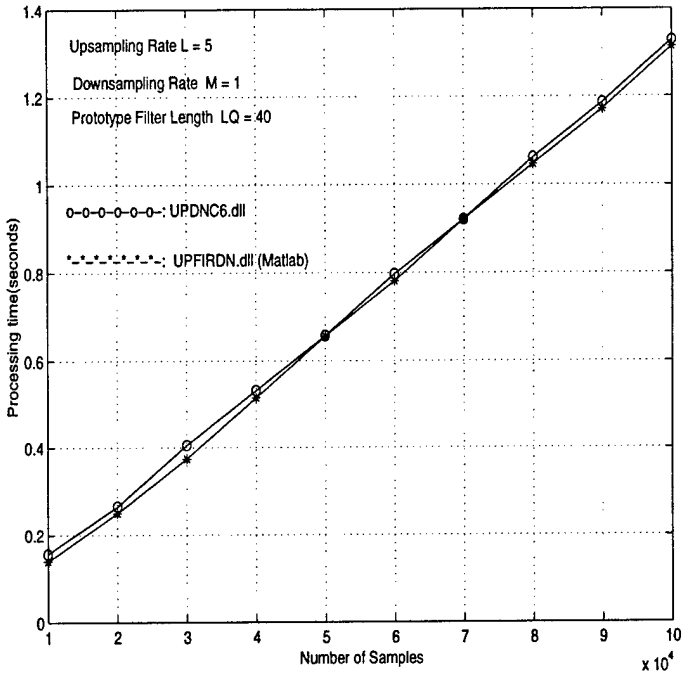


Figure 9: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor* $L = 5$*, downsampling factor* $M = 1$*, and filter length = 40.*

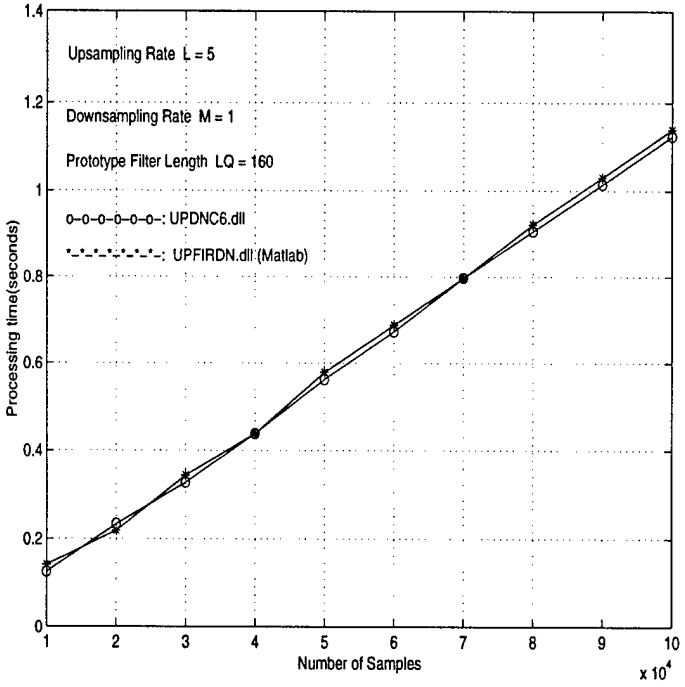


Figure 10: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor* $L = 25$*, downsampling factor* $M = 1$*, and filter length = 160.*

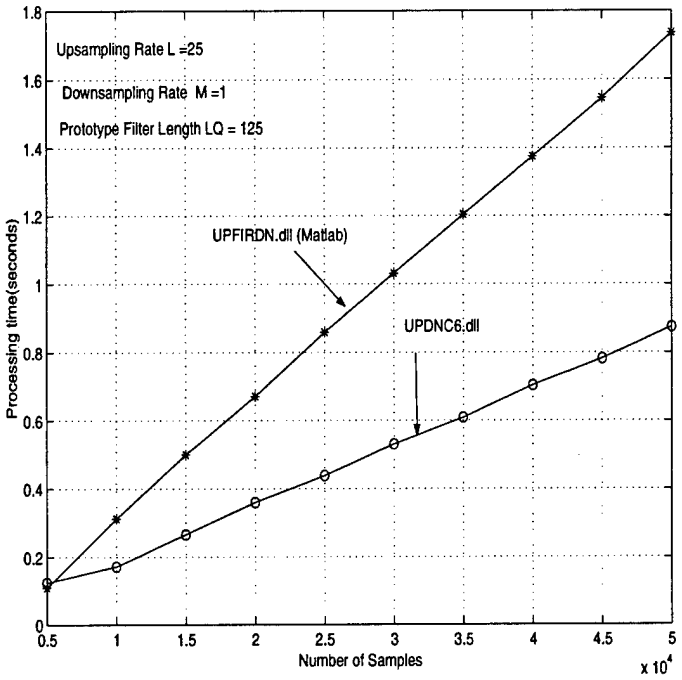


Figure 11: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor* $L = 25$*, downsampling factor* $M = 1$*, and filter length = 125.*

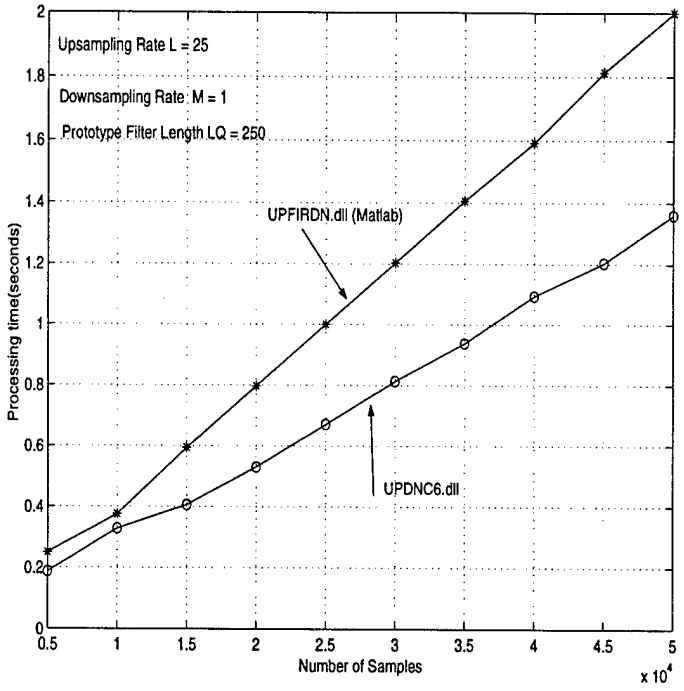


Figure 12: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor* $L = 25$*, downsampling factor* $M = 1$*, and filter length = 250.*

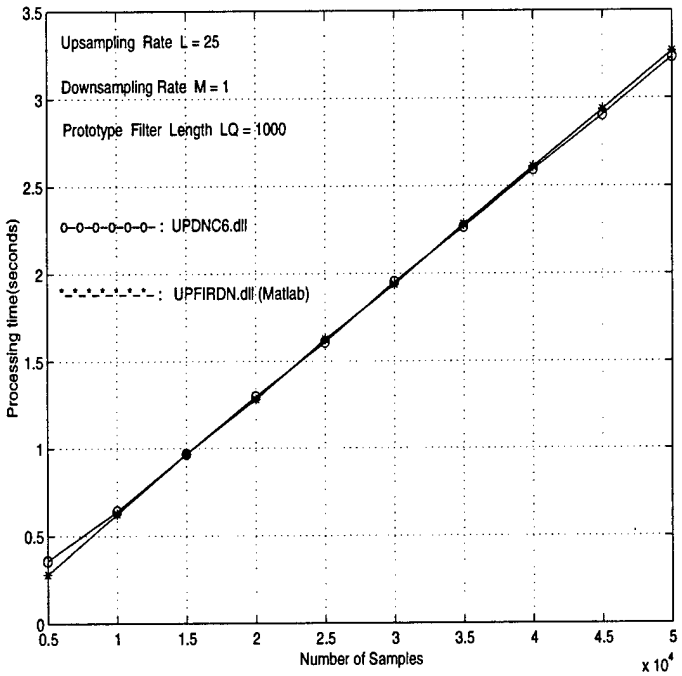


Figure 13: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor* $L = 25$*, downsampling factor* $M = 1$*, and filter length = 1000.*

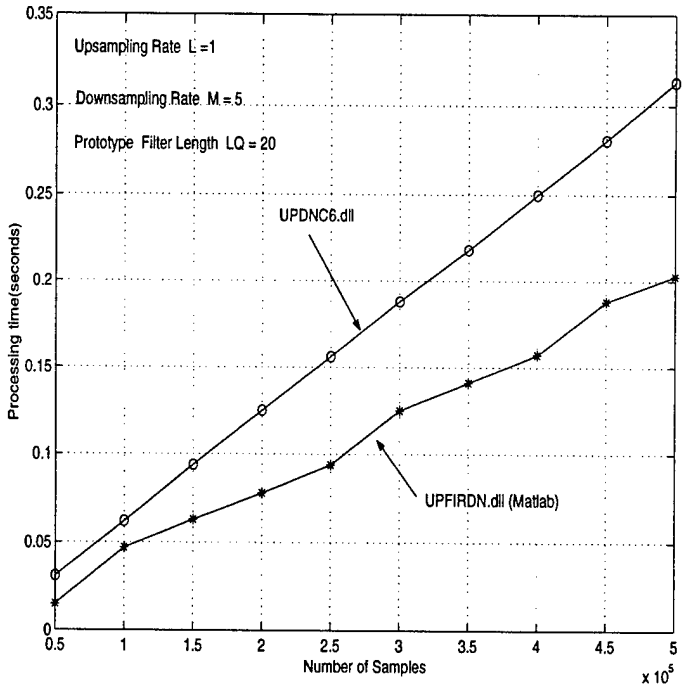


Figure 14: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 1$, downsampling factor $M = 5$, and filter length = 20.*

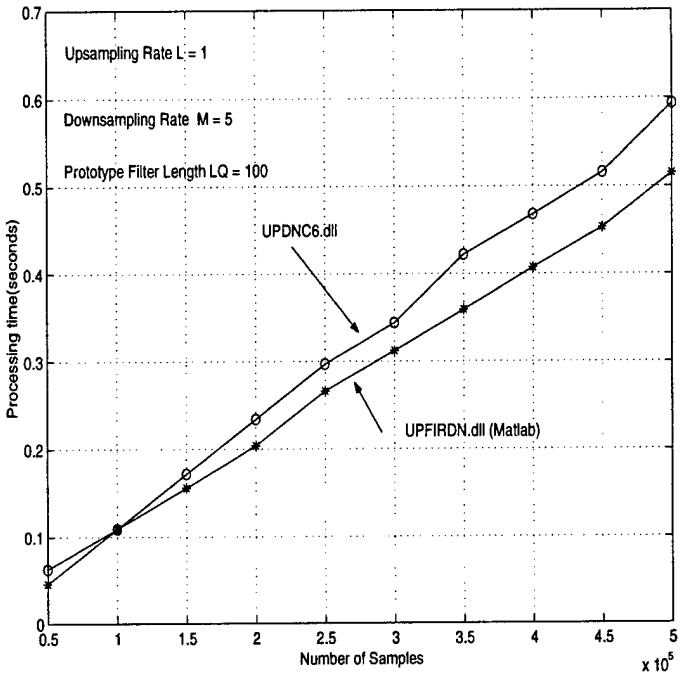


Figure 15: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 1$, downsampling factor $M = 5$, and filter length = 100.*

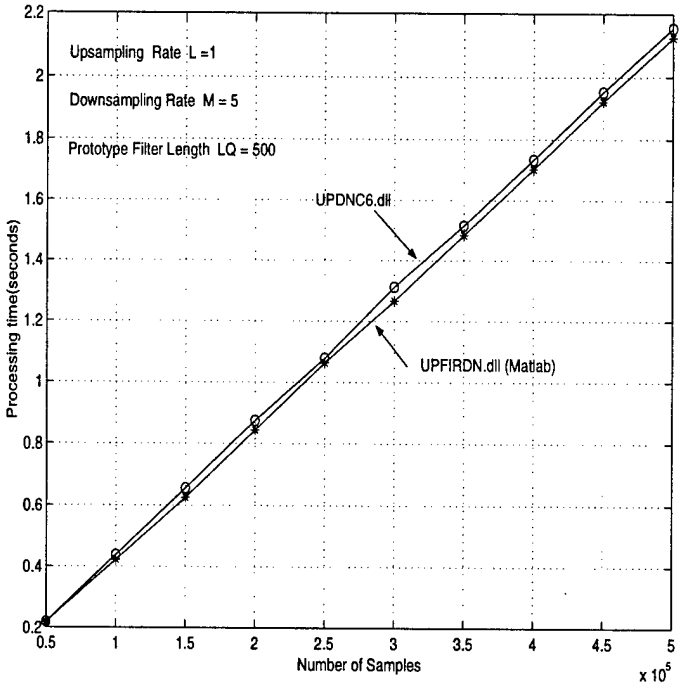


Figure 16: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor* $L = 1$*, downsampling factor* $M = 5$*, and filter length = 500.*

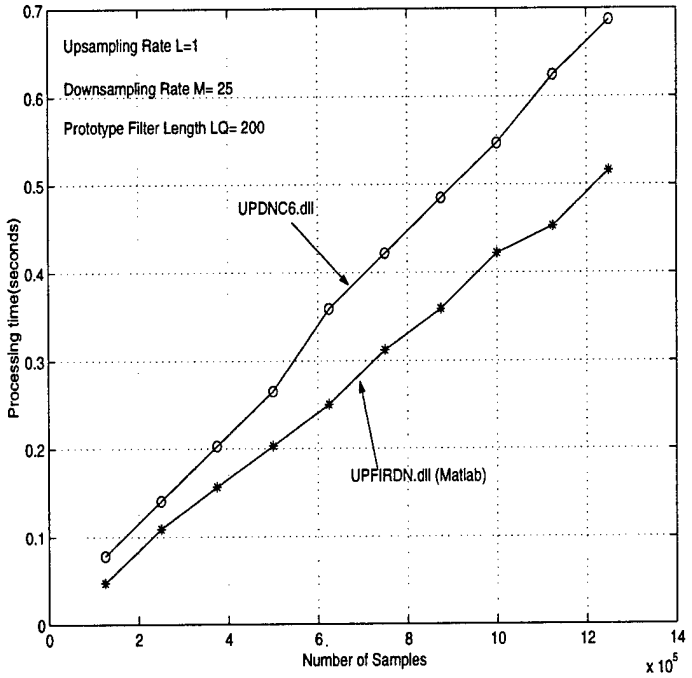


Figure 17: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 1$, downsampling factor $M = 25$, and filter length = 200.*

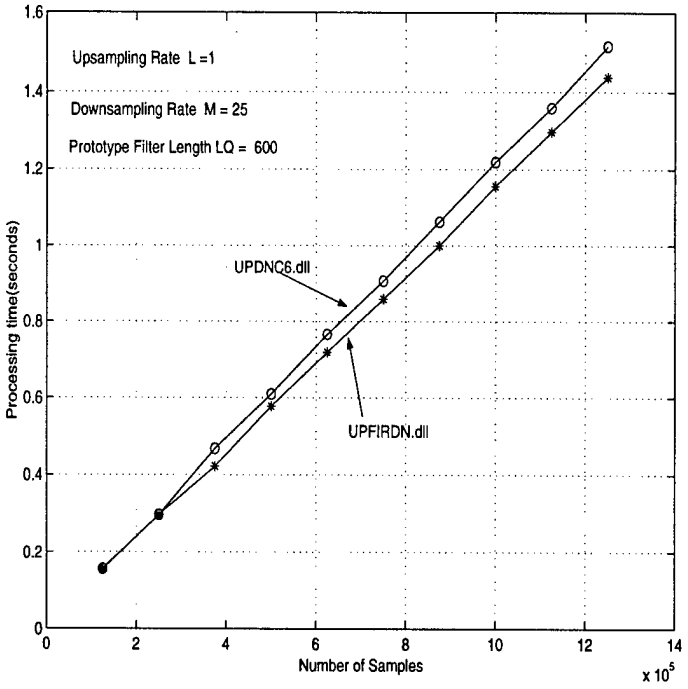


Figure 18: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 1$, downsampling factor $M = 25$, and filter length = 600.*

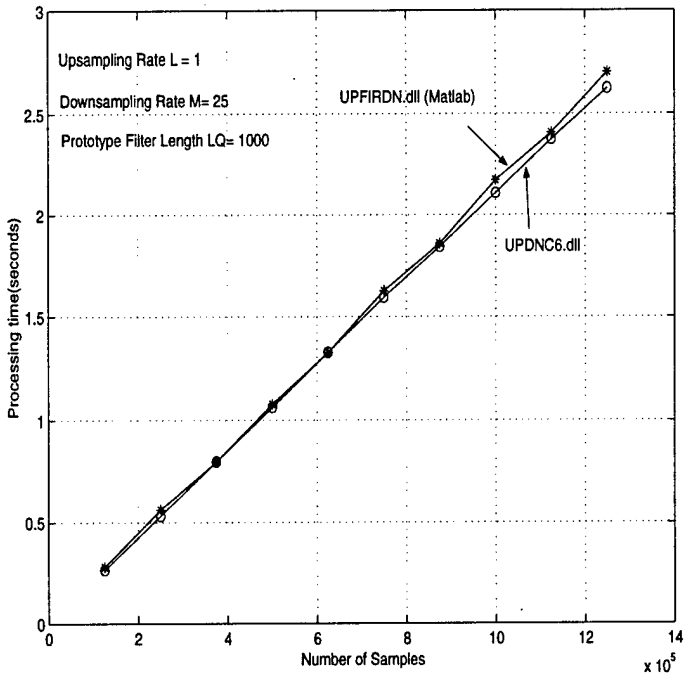


Figure 19: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 1$, downsampling factor $M = 25$, and filter length = 1000.*

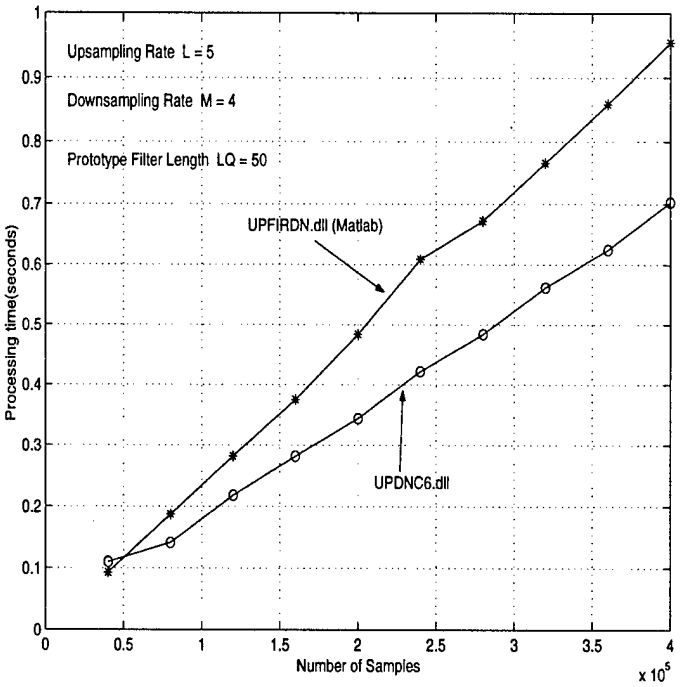


Figure 20: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 5$, downsampling factor $M = 4$, and filter length = 50.*

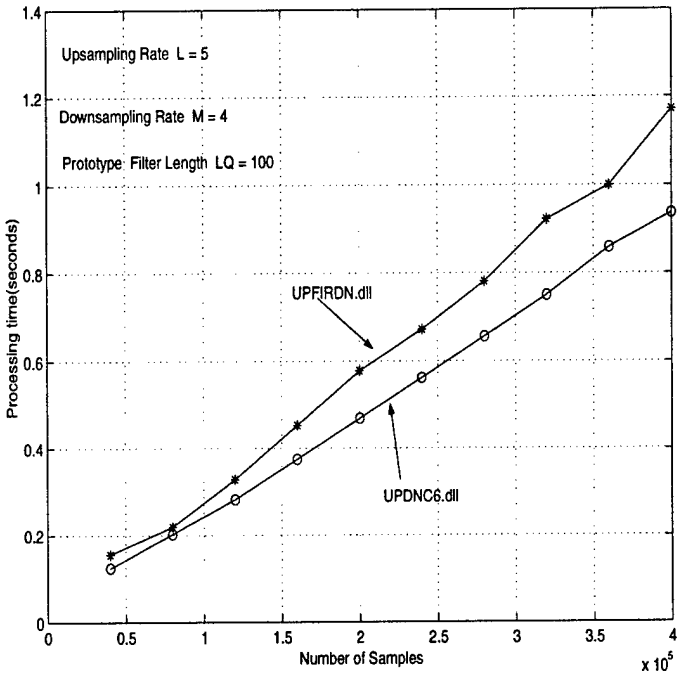


Figure 21: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 5$, downsampling factor $M = 4$, and filter length = 100.*

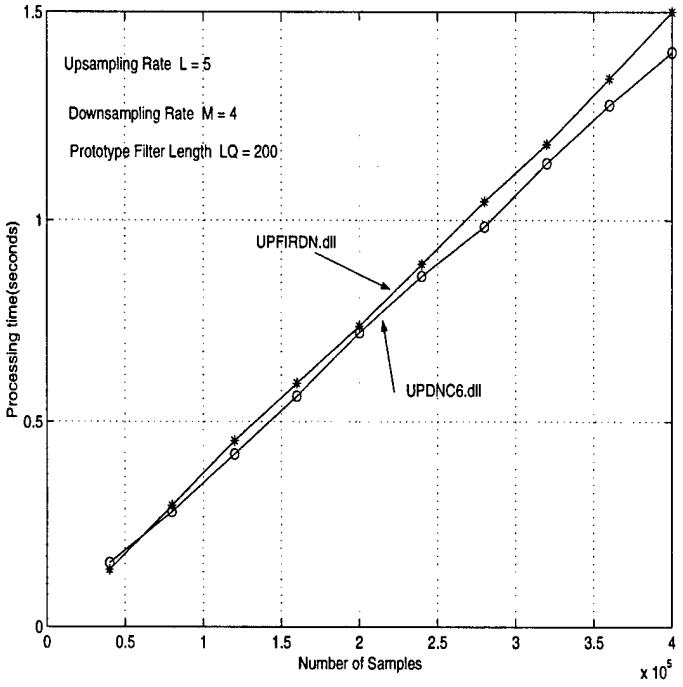


Figure 22: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor* $L = 5$*, downsampling factor* $M = 4$*, and filter length = 200.*

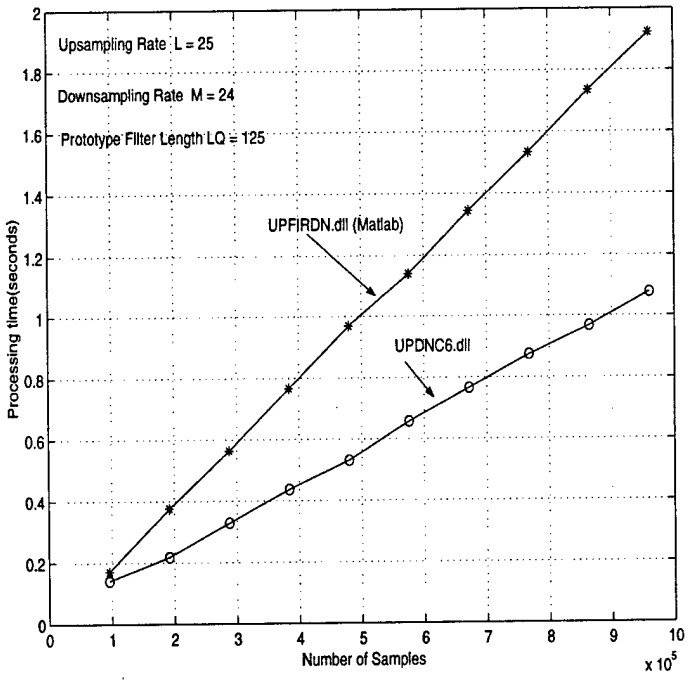


Figure 23: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 25$, downsampling factor $M = 24$, and filter length = 125.*

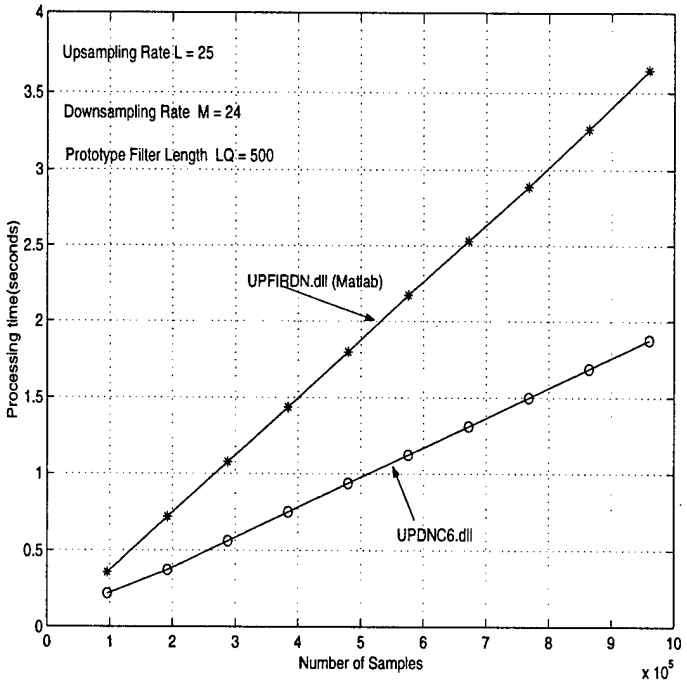


Figure 24: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 25$, downsampling factor $M = 24$, and filter length = 500.*

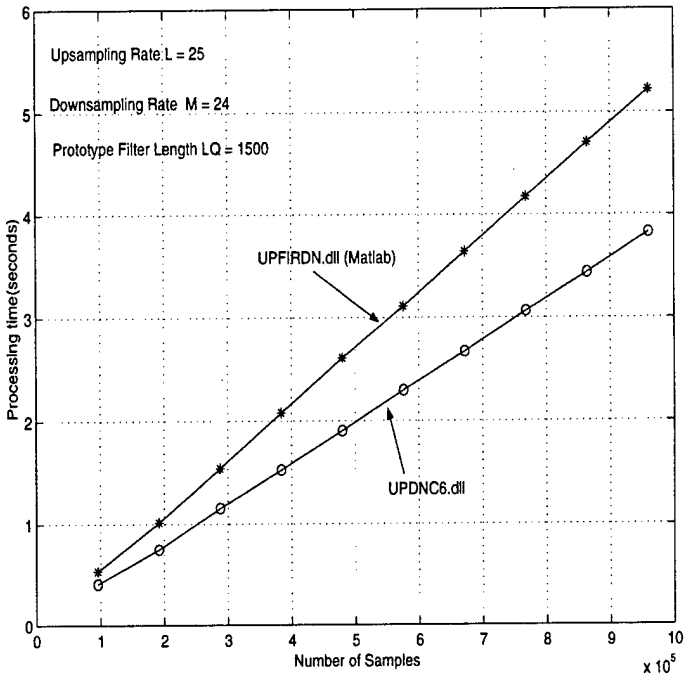


Figure 25: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 25$, downsampling factor $M = 24$, and filter length = 1500.*

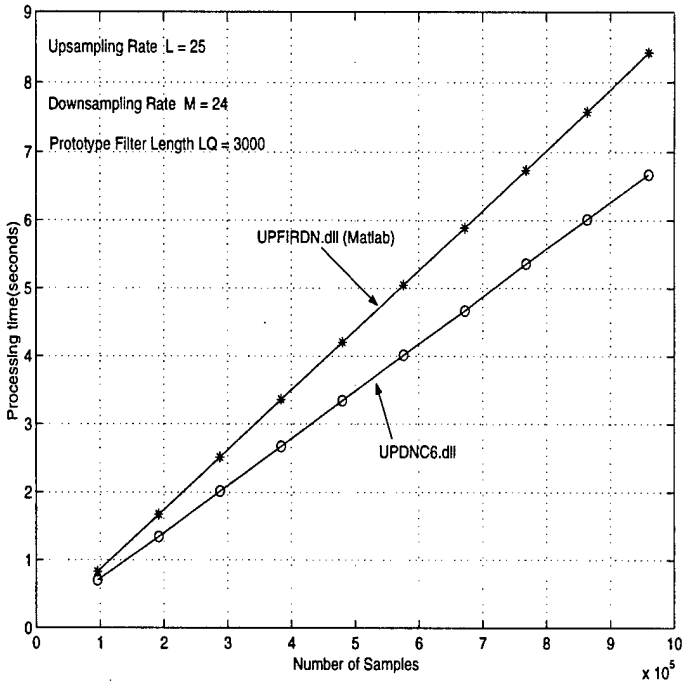


Figure 26: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 25$, downsampling factor $M = 24$, and filter length = 3000.*

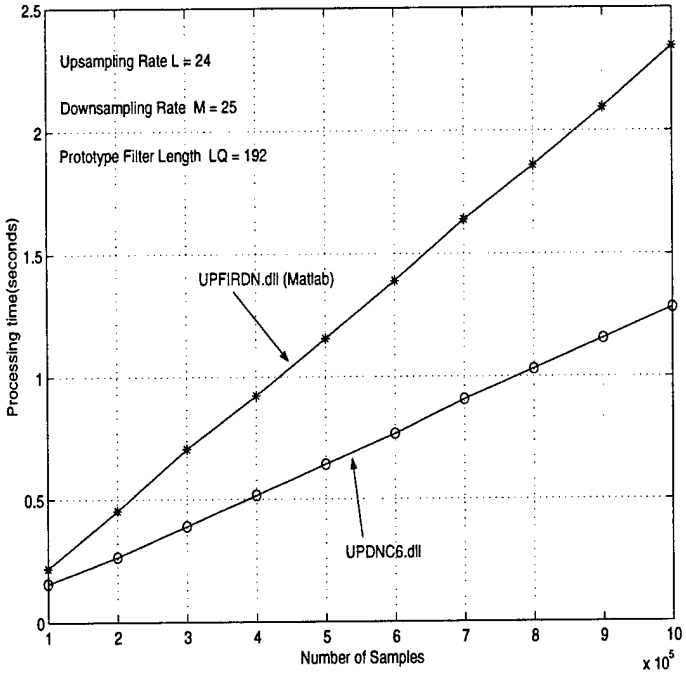


Figure 27: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 24$, downsampling factor $M = 25$, and filter length = 192.*

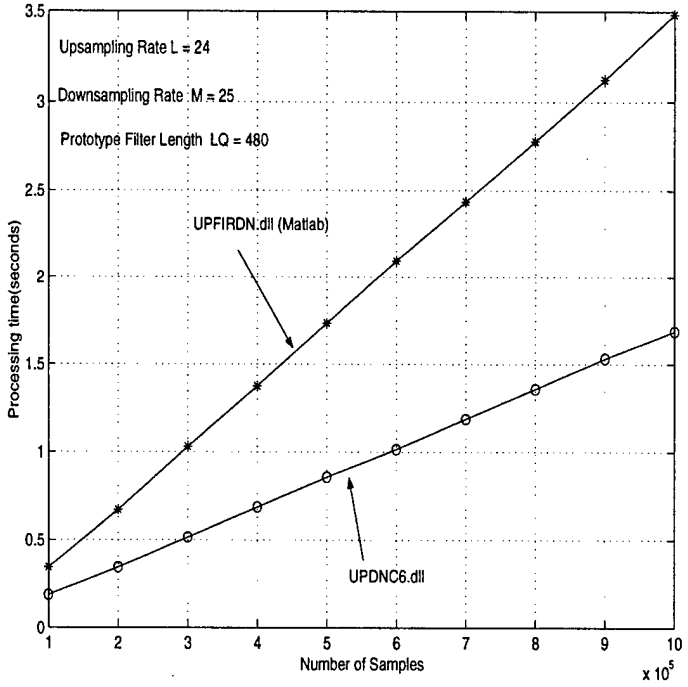


Figure 28: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 24$, downsampling factor $M = 25$, and filter length = 480.*

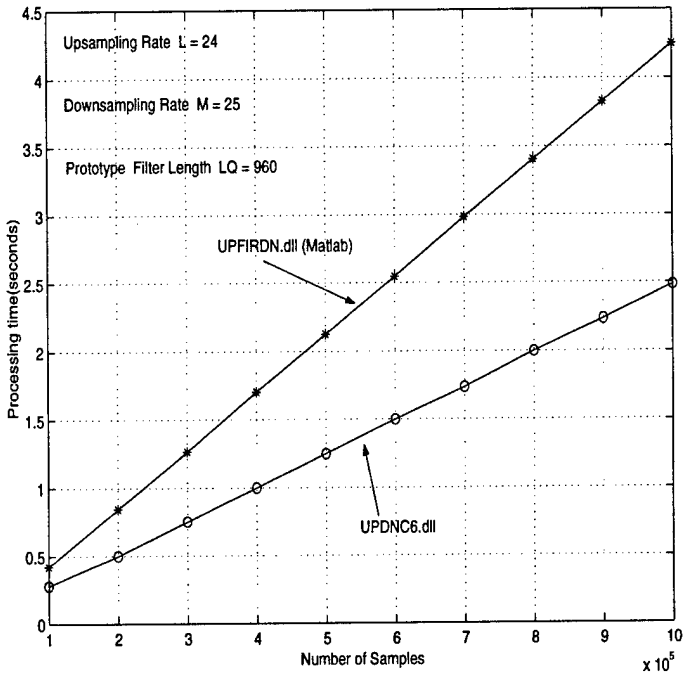


Figure 29: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 24$, downsampling factor $M = 25$, and filter length = 960.*

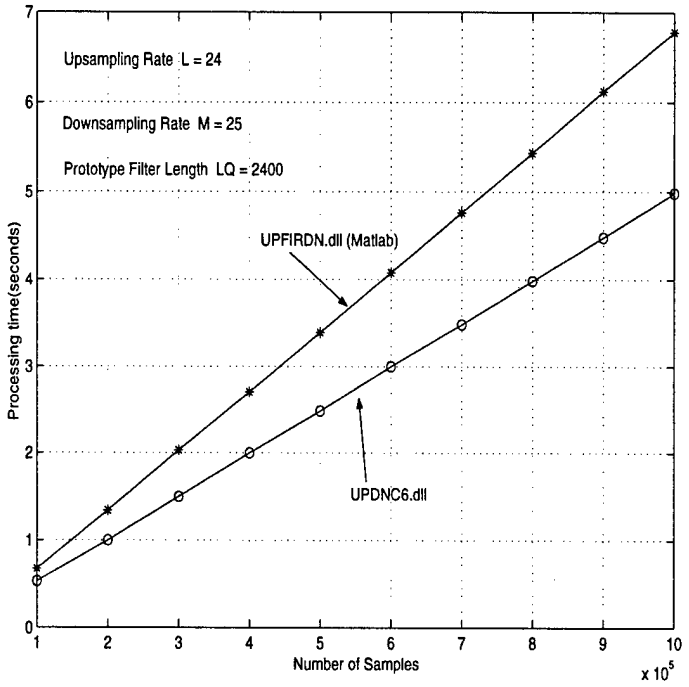


Figure 30: *Comparison of MATLAB function UPFIRDN and UPDNC6, for upsampling factor $L = 24$, downsampling factor $M = 25$, and filter length = 2400.*

References

1. Crochiere, R. E., Rabiner, L. R., "Interpolation and Decimation of Digital Signals: A Tutorial Review," Proceedings of IEEE, vol. 69, No. 3, March 1981, pp.300-331.

2. Intel Signal Processing Library, version 4.2, Intel Corporation, 2000.

3. Vaidyanathan, P. P., "Multirate Systems and Filter Banks," Prentice Hall Signal Processing Series, Prentice-Hall Inc., 1993.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research Establishment Ottawa
Department of National Defence
Ottawa, Ontario Canada K1A 0Z4

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

DIGITAL SAMPLING RATE CONVERSION: PRINCIPLES AND IMPLEMENTATION (U)

4. AUTHORS (Last name, first name, middle initial)

KOZMINCHUK, BRIAN W., AND WANG, SICHUN

5. DATE OF PUBLICATION (month and year of publication of document)

FEBRUARY 2001

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

45

6b. NO. OF REFS (total cited in document)

3

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.

DREO TECHNICAL MEMORANDUM

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.

Defence Research Establishment Ottawa
Department of National Defence
Ottawa, Ontario Canada K1A 0Z4

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

2KS11

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

TM 2001-032

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). however, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.) UNLIMITED

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

(U) The resampling of a signal involves the conversion from the initial sampling rate to a new and different one. This is often necessary in practical applications because the sampling rate is often fixed while the desired sampling rate may depend on the application or a signal parameter, such as the symbol rate or channel spacing. This problem often arises in software radios systems where the sampling rate is determined by hardware constraints. Although resampling algorithms have been developed and implemented in commercial software libraries, such as the Intel Signal Processing Library \cite{Intel}, these algorithms often have undesirable limitations. For example, a typical constraint is that the data blocks processed must be an integer multiple of the downsampling rate. This restriction simplifies the indexing in the code and reduces the complexity of the implementation. However, it decreases the flexibility and usefulness of the resulting program, since in some applications, the length of the input data blocks may not be an integer multiple of the downsampling rate. For such applications, there is a need for a resampling program that imposes no restriction on the length of the input data blocks. The MATLAB built-in function UPFIRDN.dll (or RESAMPLE.m) is designed for resampling only one block of input data. It does not provide information on the state variables of the filters and, therefore, cannot be used to resample a very large data file.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

RESAMPLE
INTERPOLATION
DECIMATION
SIGNAL PROCESSING

Defence R&D Canada
is the national authority for providing
Science and Technology (S&T) leadership
in the advancement and maintenance
of Canada's defence capabilities.

R et D pour la défense Canada
est responsable, au niveau national, pour
les sciences et la technologie (S et T)
au service de l'avancement et du maintien des
capacités de défense du Canada.

www.drdc-rddc.dnd.ca