Efficient Arbitrary Sampling Rate Conversion With Recursive Calculation of Coefficients

Andrew I. Russell, *Student Member, IEEE,* and Paul E. Beckmann, *Member, IEEE*

***Abstract*—In this paper, we present a novel algorithm for sampling rate conversion by an arbitrary factor. Theoretically, sampling rate conversion of a discrete-time (DT) sequence can be performed by converting the sequence to a series of continuous-time (CT) impulses. This series of impulses is filtered with a CT lowpass filter, and the output is then sampled at the desired rate. If the CT filter is chosen to have a rational transfer function, then this system can be simulated using a DT algorithm for which both computation and memory requirements are low. The DT implementation is comprised of a parallel structure, where each branch consists of a time-varying filter with one or two taps, followed by a fixed recursive filter operating at the output sampling rate. The coefficients of the time-varying filters are calculated recursively. This eliminates the need to store a large table of coefficients, as is commonly done.**

***Index Terms*—Irrational ratio sampling rate conversion, low complexity, low memory, rational filters, recursive algorithms.**

I. Introduction

SAMPLING rate conversion is performed whenever a discrete-time (DT) system needs to operate on data streams with differing sampling rates. It is commonly encountered in both audio and video applications. Digital audio is typically recorded at several different sampling rates. For example, compact discs (CDs) use a sampling rate of 44.1 kHz, and digital audio tapes (DATs) use a sampling rate of 48 kHz. Sometimes, audio signals in these different formats may need to be combined and processed jointly. A common method of sampling rate conversion is to use a combination of up-sampling, filtering, and down-sampling [2]. This works well if the ratio of sampling rates is a rational number. In many cases, however, the sampling rates are not related by a rational number. This occurs frequently when trying to synchronize data streams that are arriving on separate digital networks. For example, in audio applications, it is common to have a DT system operating at 44.1 kHz, whereas the input data stream is arriving at a frequency that is slightly less than 44.1 kHz. In fact, the input data rate may be slowly varying, in which case, the receiving system must accommodate these variations. In some cases, it is possible to synchronize the processing system to the incoming data rate. In other applications, there may be multiple data streams at arbitrary sampling rates. In these cases, irrational sampling rate conversion may be necessary.

The problem of performing sampling rate conversion by a rational ratio has been studied in detail, e.g., in [2]–[4] (further references are given in [5]). Methods have also been proposed that allow for conversion by irrational ratios [5]–[7]. These methods are based on the observation that a system that includes a continuous-time (CT) filter can be simulated using a time-varying DT filter. In these papers, various choices for the CT filter are proposed. In all cases, the authors conclude that there is a tradeoff between computational and memory requirements for generating the coefficients of the time-varying DT filter. In contrast, the algorithm presented in this paper is computationally efficient while requiring very little memory. This is possible because the coefficients of the time-varying DT filter are calculated recursively.

The paper is organized as follows. Section II summarizes the previously known methods and presents the theoretical framework within which our algorithm is derived. In Section III, we show that if a first-order rational CT filter is used, then the system can be efficiently implemented in discrete time. Sections IV and V extend the algorithm to allow second- and Nth-order rational filters. We then give a step-by-step summary of the overall algorithm in Section VI. In Section VII, the computational complexity of the algorithm is analyzed, and an actual design example for an audio sampling rate converter is presented and compared with traditional approaches. Section VIII discusses the advantages and disadvantages of the algorithm and summarizes the main results. Relevant proofs are given in the Appendix.

II. Background

In this paper, CT signals are generally written with parentheses, e.g., $x(t)$, whereas DT signals are written with square brackets, e.g., $x[n]$. The continuous time variable used is usually t, whereas the discrete time variable is usually n or m. The block diagram in Fig. 1(a) represents the sampling operation or continuous- to discrete-time (C/D) conversion. It is defined by the input–output relationship

$$y[n] = y(t)|_{t=nT}. \tag{1}$$

The discrete- to continuous-time (D/C) converter shown in Fig. 1(b) converts a DT sequence to a series of CT impulses and is defined by the input–output relationship

$$x_\delta(t) = \sum_{k=-\infty}^{\infty} x[k]\delta(t - kT) \tag{2}$$

Manuscript received December 20, 2000; revised January 2, 2002. This work was developed while the authors were employed by Bose Corporation [1], and its use may be restricted by United States patent law. The associate editor coordinating the review of this paper and approving it for publication was Dr. Olivier Cappe.

A. I. Russell is with the Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: air@alum.mit.edu).

P. E. Beckmann is with Enuvis Inc., South San Francisco, CA 94404 USA (e-mail: beckmann@alum.mit.edu).

Publisher Item Identifier S 1053-587X(02)02401-7.

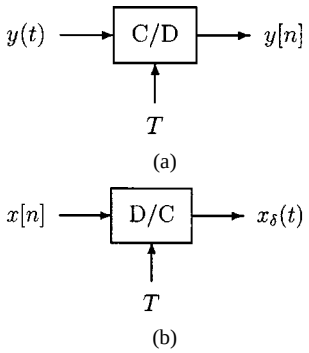

Fig. 1. (a) Continuous- to discrete-time (C/D) converter. (b) Discrete- to continuous-time (D/C) converter. The input–output relationships are given by (1) and (2), respectively.

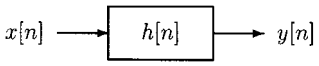

Fig. 2. LTI filtering; $y[n] = x[n] * h[n]$.

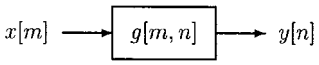

Fig. 3. DT filter with time-varying coefficients.

where $\delta(t)$ is the Dirac delta function that is otherwise referred to as a CT impulse. CT signals that consist of a series of CT impulses are generally labeled with the subscript δ, e.g., $x_\delta(t)$ in (2). Upper-case letters denote the Laplace or z-transforms of the corresponding CT or DT signals. Linear, time-invariant (LTI) filters are drawn as shown in Fig. 2, with the input signal, output signal, and impulse response written with the same time variable. The LTI filtering block is defined by

$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m]. \tag{3}$$

Time-varying filters are drawn as shown in Fig. 3, with the input and output signal using different time variables and the kernel being a function of both variables. This block is defined by

$$y[n] = \sum_{m=-\infty}^{\infty} x[m]g[m,n]. \tag{4}$$

A similar definition of a time-varying filter is given in [8], [9]. In this paper, we assume that the output sampling period $T_{\text{out}} = 1$. This assumption does not restrict us in any way since any desired change in sampling rate can still be achieved by choosing the appropriate value for the input sampling period T_{in}.

Ideal sampling rate conversion is the process of transforming an input sequence $x[m]$ into an output sequence $y[n]$. It is assumed that $x[m]$ represents a bandlimited CT signal $x(t)$ and is obtained by sampling $x(t)$ with a period of T_{in} s. Thus, $x[m] = x(mT_{\text{in}})$. The most straightforward procedure for performing sampling rate conversion is to reconstruct the CT signal $x(t)$ and then resample it using a new sampling period of $T_{\text{out}} = 1$ s. In order to prevent aliasing, $x(t)$ is filtered with an anti-aliasing lowpass filter before resampling.

Ideal sampling rate conversion can be implemented using the system shown in Fig. 4. This system converts the input sequence

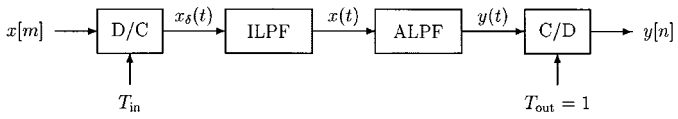

Fig. 4. Ideal sampling rate converter. ILPF is an interpolating lowpass filter with cut-off frequency π/T_{in} and gain T_{in}. ALPF is an anti-aliasing lowpass filter with cut-off frequency π and gain 1. These filters are assumed to be ideal.

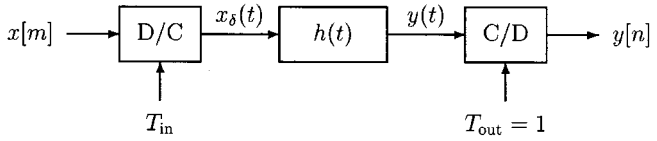

Fig. 5. Generic sampling rate converter. $h(t)$ is the impulse response of an arbitrary filter. If $h(t)$ is chosen as an ideal lowpass filter, then this system can be made to be equivalent to the system in Fig. 4.

$x[m]$ to a series of CT impulses $x_\delta(t)$, which are spaced by T_{in} s so that

$$x_\delta(t) = \sum_{m=-\infty}^{\infty} x[m]\delta(t - mT_{\text{in}}). \tag{5}$$

A bandlimited interpolation of $x[m]$ is obtained by filtering $x_\delta(t)$ with an interpolating lowpass filter labeled ILPF. This interpolating filter is an ideal lowpass filter with cut-off frequency of π/T_{in} and a gain of T_{in}. The resulting signal $x(t)$ is the bandlimited signal that satisfies $x[m] = x(mT_{\text{in}})$. The CT signal $x(t)$ is then filtered by an anti-aliasing lowpass filter, labeled ALPF, prior to resampling. This anti-aliasing filter is an ideal lowpass filter with cut-off frequency π and a gain of 1. The output of the filter $y(t)$ is sampled with a period of 1 s to yield the output signal $y[n]$, where

$$y[n] = y(n). \tag{6}$$

More generally, any system that performs D/C conversion, filtering, and C/D conversion can be thought of as a generic sampling rate converter, as shown in Fig. 5. The impulse response of this filter is denoted by $h(t)$. Since $x_\delta(t)$ is given by (5), the output of the filter $y(t)$ can be expressed as

$$y(t) = \sum_{m=-\infty}^{\infty} x[m]h(t - mT_{\text{in}}). \tag{7}$$

If we sample $y(t)$ as indicated in Fig. 5, then

$$y[n] = \sum_{m=-\infty}^{\infty} x[m]h(n - mT_{\text{in}}). \tag{8}$$

By comparing (8) with (4), it follows that the system of Fig. 5 is equivalent to the system of Fig. 3 if

$$g[m,n] = h(n - mT_{\text{in}}). \tag{9}$$

We can therefore redraw the system in Fig. 5 as a DT system, shown in Fig. 6.

Under certain conditions, the generic sampling rate converter shown in Fig. 5 is equivalent to the ideal sampling rate converter of Fig. 4. Specifically, $h(t)$ must be chosen as an ideal lowpass filter with cut-off frequency $\min(\pi, \pi/T_{\text{in}})$ and a gain of T_{in}. This follows from combining the two filters in Fig. 4. If

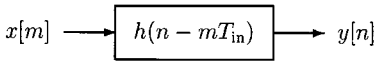

Fig. 6. Time-varying DT filter, which is equivalent to the system in Fig. 5.

$T_{\text{in}} > 1$, then the filter $h(t)$ acts as the reconstruction filter associated with converting $x[m]$ to a bandlimited CT signal. Otherwise, $h(t)$ also acts as the anti-aliasing filter associated with the sampling operation. However, if the filter is chosen to be an ideal lowpass filter, then $h(t)$ is a sinc function, which is infinitely long in both directions. This makes it very difficult to evaluate (8) explicitly. In order to be able to implement this system, we cannot choose $h(t)$ as an ideal lowpass filter. Therefore, we cannot implement an ideal sampling rate converter.

In this paper, we focus our attention on an efficient implementation of the system depicted in Fig. 5, where $h(t)$ is chosen as an approximation to an ideal lowpass filter. Several methods have been proposed for approximating an ideal filter for use in a sampling rate converter. Lagadec *et al.* [10], [11] proposed choosing $h(t)$ to have finite support. A simple method of designing such a filter is to window the impulse response of an ideal filter. This allows us to evaluate (8) explicitly since it reduces to a finite sum. A finely spaced set of samples of $h(t)$ can then be stored in memory. In order to retrieve values of $h(t)$ that fall between the stored samples, some kind of interpolation must be performed. Lagadec *et al.* [10], [11] proposed simply using the nearest coefficient (i.e., nearest-neighbor interpolation). Smith and Gossett [7] suggested a similar algorithm in which they considered using linear interpolation, and Ramstad [5] additionally considered using some higher order interpolation such as cubic or spline interpolation. Choosing the simplest form of interpolation (nearest-neighbor interpolation) is the least demanding computationally but requires a very large number of samples of the impulse response to be stored. Using higher order interpolation requires less storage but is more demanding computationally. Thus, there is a tradeoff between computation and memory requirements.

Ramstad was able to analyze these methods by showing that if the impulse response is stored as a set of samples and then interpolated as suggested above, the resulting system is still equivalent to the system in Fig. 4. In this case, the impulse response $h(t)$ can be decomposed into a finite sequence of CT impulses spaced by ϵ s convolved with a short interpolating function. This interpolating function is a rectangle of width ϵ for nearest-neighbor interpolation, a triangle of width 2ϵ for linear interpolation, and some other piecewise polynomial function for higher order interpolation. Thus, the frequency response of the filter can be calculated exactly.

Wang [12], [13] has obtained patents for a method in which $h(t)$ in Fig. 5 is chosen as a piecewise polynomial function over a finite number of regular intervals. Wang has applied his method to the synthesis of musical tones. Saramäki and Ritoniemi [6] also proposed an effecient scheme for implementing a sampling rate converter in which $h(t)$ is piecewise polynomial. An integer up-sampling stage was included before applying the piecewise polynomial filter. This integer up-sampling can be represented by a series of CT impulses. Thus, $h(t)$ can be expressed as the convolution of a series of CT impulses and a piecewise polynomial function. Here, again, there is a tradeoff between computational and memory requirements.

Instead of using these methods of approximation, we choose the transfer function $H(s)$ to be rational. For example, an elliptic, Butterworth, or Chebyshev lowpass filter can be used. Ramstad [5] showed that in this case, the CT filter in Fig. 5 can be made to have finite duration if a fixed recursive DT filter that operates at either the input or output sampling rates is introduced. Ramstad then showed that this finite-duration filter can be implemented as previously indicated by storing finely spaced samples of the impulse response. In this paper, we show that by choosing the filter to be rational, we no longer need to store a table of samples of the impulse response. The required samples at any time step can be calculated efficiently from the samples at the previous time step. Thus, the computation and memory requirements of our proposed algorithm are both low. We first consider the case where $H(s)$ is chosen as a first-order filter. We then generalize this approach to allow for higher order filters that can better approximate an ideal lowpass filter.

III. First-Order Approximation

In this section, we present a DT implementation of the system shown in Fig. 5, where $H(s)$ is chosen as a first-order rational filter. In this case, the impulse response $h(t)$ is a decaying exponential starting at $t = 0$. We begin with a simple example to illustrate the algorithm.

With reference to Fig. 5, suppose that the input $x[m]$ has exactly one nonzero value at time $m = m_0$. Then, $x_\delta(t)$, which is given by (5), is a single CT impulse occurring at time $m_0 T_{\text{in}}$, and the filter output $y(t)$ is a decaying exponential starting at time $m_0 T_{\text{in}}$. When this single decaying exponential is sampled, the DT signal $y[n]$ is also a decaying exponential. This suggests that $y[n]$ can be generated as the output of a first-order DT filter since the impulse response of a first-order DT filter is a decaying exponential that starts at $n = 0$. The input to the DT filter is a DT signal $w[n]$, which has exactly one nonzero sample. The location and amplitude of the nonzero sample depends on m_0. Filtering $w[n]$ with the DT filter results in a single decaying exponential starting at the same value of n for which $w[n]$ was nonzero. If we choose the location and amplitude of this nonzero sample and the parameters of the DT filter correctly, then we can make the output be identical to $y[n]$.

This example can be generalized to arbitrary inputs $x[m]$ consisting of many nonzero samples by using linearity. The contribution of each sample of $x[m]$ to $w[n]$ is determined, and then, $w[n]$ is passed through a first-order DT filter. We now develop this idea in more detail.

Consider the case where $h(t)$ in Fig. 5 is a first-order filter with transfer function

$$H(s) = \frac{1}{s+p} \tag{10}$$

where p is a real positive constant. The impulse response of the system is given by

$$h(t) = e^{-pt}u(t) \tag{11}$$

where $u(t)$ is the unit-step function defined by

$$u(t) = \begin{cases} 1, & t \geq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

Since p is real and positive, the impulse response is real and the system is stable.

We now derive a discrete-time implementation of this continuous-time system, which is very efficient. The first step is to represent the impulse response $h(t)$ as a finite-duration or time-limited function $f(t)$, convolved with a train of decaying impulses $h_\delta(t)$. Specifically, in Appendix A, we show that we can decompose $h(t)$ as

$$h(t) = e^{-pt}u(t) = f(t) * h_\delta(t) \tag{13}$$

where

$$f(t) = \alpha^t[u(t) - u(t-1)] = \alpha^t r(t) \tag{14}$$

$\alpha = e^{-p}$, and

$$h_\delta(t) = \sum_{k=0}^{\infty} \alpha^k \delta(t-k). \tag{15}$$

$r(t)$ is a unit-height rectangular pulse over the interval $[0, 1)$ and is defined as

$$r(t) = u(t) - u(t-1). \tag{16}$$

The decomposition of $h(t)$ according to (13) is illustrated in Fig. 7. Substituting this decomposition into the system of Fig. 5 results in the system depicted in Fig. 8.

The sampling operations in the C/D block and the impulses in the filter $h_\delta(t)$ are both spaced by 1 s in time. Therefore, we can interchange the sampling and filtering operations. This results in the system shown in Fig. 9. The intermediate DT signal at the output of the C/D block is labeled $w[n]$ and serves as the input to a first-order DT filter with impulse response

$$h[n] = \alpha^n u[n] \tag{17}$$

where $u[n]$ is the DT unit step function defined by

$$u[n] = \begin{cases} 1, & n \geq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{18}$$

In order to show that the systems in Figs. 8 and 9 are equivalent, let us consider the relationship between $y[n]$ and $w(t)$. In Fig. 8, $y(t)$ is related to $w(t)$ by convolution with $h_\delta(t)$, given in (15). Thus

$$y(t) = \sum_{k=0}^{\infty} \alpha^k w(t-k). \tag{19}$$

$y[n]$ is then obtained by sampling $y(t)$ so that

$$y[n] = \sum_{k=0}^{\infty} \alpha^k w(n-k). \tag{20}$$

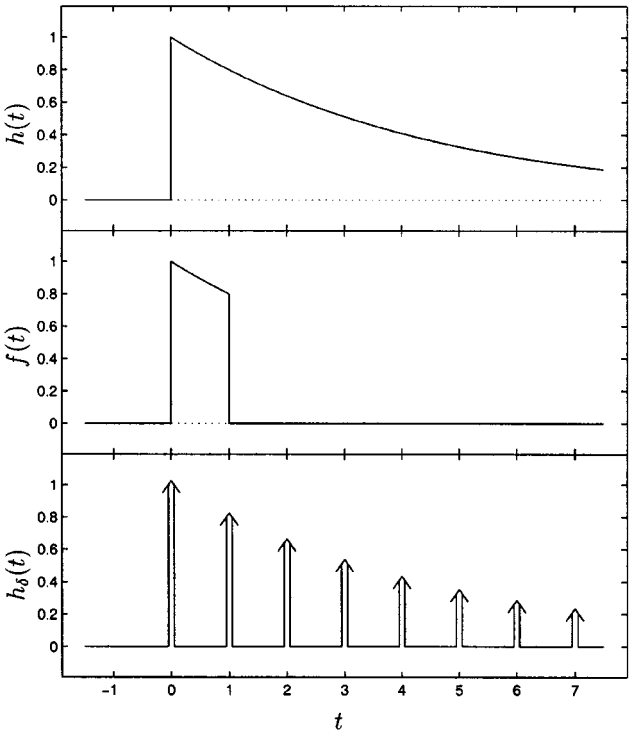

Fig. 7. Illustration of the decomposition $h(t) = f(t) * h_\delta(t)$. $h(t)$ is a decaying exponential, $f(t)$ is a finite-duration function, and $h_\delta(t)$ is a series of CT impulses.

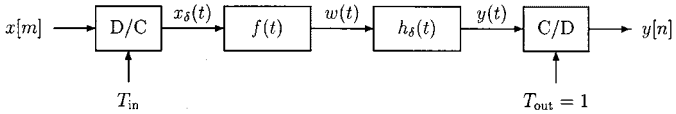

Fig. 8. System equivalent to the one shown in Fig. 5 when $h(t) = \alpha^t u(t)$. Here, $f(t) = \alpha^t r(t)$ and $h_\delta(t) = \sum_{k=0}^{\infty} \alpha^k \delta(t-k)$.

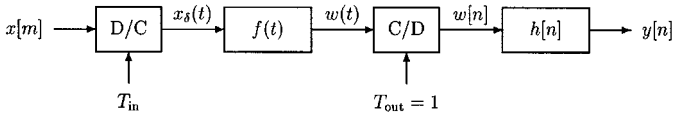

Fig. 9. System equivalent to the one shown in Fig. 8. Here, $h[n] = \alpha^n u[n]$.

In Fig. 9, $y[n]$ is related to $w[n]$ by convolution with $h[n]$, given in (17), and thus

$$\begin{aligned} y[n] &= \sum_{k=-\infty}^{\infty} h[k]w[n-k] \\ &= \sum_{k=0}^{\infty} \alpha^k w[n-k]. \end{aligned} \tag{21}$$

Substituting $w[n] = w(n)$ in (21) gives

$$y[n] = \sum_{k=0}^{\infty} \alpha^k w(n-k). \tag{22}$$

Comparing (20) and (22), we conclude that the two systems are equivalent.

Now, the first three blocks in Fig. 9 (D/C converter, CT filter $f(t)$, and C/D converter) can be replaced by a linear time-varying filter, as shown in Fig. 10. To show this, we appeal

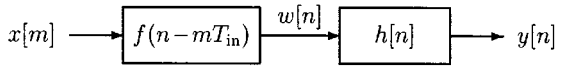

Fig. 10. Here, we have replaced the first three blocks in Fig. 9 with one block. This block represents the algorithm that simulates the other three blocks by efficiently implementing (29).

to the previously derived equivalence of the systems in Figs. 5 and 6. Thus, $w[n]$ is related to $x[m]$ by

$$w[n] = \sum_{m=-\infty}^{\infty} x[m] f(n - mT_{\text{in}}). \tag{23}$$

For any value of m, the function $f(n - mT_{\text{in}})$ is nonzero for exactly one value of n. This is because $f(\tau)$ is nonzero only where $0 \leq \tau < 1$. The value of n that makes $f(n - mT_{\text{in}})$ nonzero is $n = \lceil mT_{\text{in}} \rceil$, where $\lceil mT_{\text{in}} \rceil$ is the smallest integer greater than or equal to mT_{in}. $f(n-mT_{\text{in}})$ can then be rewritten as

$$f(n - mT_{\text{in}}) = f(\lceil mT_{\text{in}} \rceil - mT_{\text{in}}) \delta[n - \lceil mT_{\text{in}} \rceil]. \tag{24}$$

If we let

$$\tau[m] = \lceil mT_{\text{in}} \rceil - mT_{\text{in}}, \tag{25}$$

then we can rewrite (23) as

$$w[n] = \sum_{m=-\infty}^{\infty} x[m] f(\tau[m]) \delta[n - \lceil mT_{\text{in}} \rceil] \tag{26}$$

where $f(\tau[m])$ can be obtained from (14) as

$$f(\tau[m]) = \alpha^{\tau[m]} \tag{27}$$

by noting that $0 \leq \tau[n] < 1$. In order to simplify the notation, we refer to the coefficient in (26) as $c[m]$; therefore

$$c[m] = f(\tau[m]) = \alpha^{\tau[m]}. \tag{28}$$

Using this notation, (26) can be rewritten as

$$w[n] = \sum_{m=-\infty}^{\infty} x[m] c[m] \delta[n - \lceil mT_{\text{in}} \rceil]. \tag{29}$$

The summation in (29) can then be evaluated directly. For each term, we need to calculate $c[m]$ and perform a single multiplication. In order to calculate $c[m]$, we need to evaluate $f(t)$, which is an exponential function, which can be computationally demanding. Fortunately, there exists a more efficient scheme that takes advantage of the specific shape of $f(t)$ and the fact that the points at which $f(t)$ is evaluated have a regular structure.

In order to gain insight into how this efficient scheme works, we consider how the sequence $\tau[m]$ evolves in time. To this end, we define

$$\Delta[m] = \tau[m] - \tau[m-1]. \tag{30}$$

Substituting (25) into (30) gives

$$\begin{aligned} \Delta[m] &= \lceil mT_{\text{in}} \rceil - mT_{\text{in}} - \lceil (m-1)T_{\text{in}} \rceil + (m-1)T_{\text{in}} \\ &= \lceil mT_{\text{in}} \rceil - \lceil (mT_{\text{in}} - T_{\text{in}}) \rceil - T_{\text{in}} \\ &= \begin{cases} \lceil T_{\text{in}} \rceil - T_{\text{in}}, & \tau[m-1] < T_{\text{in}} - \lfloor T_{\text{in}} \rfloor \\ \lfloor T_{\text{in}} \rfloor - T_{\text{in}}, & \tau[m-1] \geq T_{\text{in}} - \lfloor T_{\text{in}} \rfloor \end{cases} \end{aligned} \tag{31}$$

where $\lfloor T_{\text{in}} \rfloor$ is defined as $\lceil T_{\text{in}} \rceil - 1$. From (31), it follows that $\Delta[m]$ can take on only two possible values. We denote that the positive[1] value by Δ^+,

$$\Delta^+ = \lceil T_{\text{in}} \rceil - T_{\text{in}} \tag{32}$$

and the negative value by Δ^-

$$\Delta^- = \lfloor T_{\text{in}} \rfloor - T_{\text{in}}. \tag{33}$$

We use Δ to denote either Δ^+ or Δ^-. Since

$$\tau[m] = \tau[m-1] + \Delta[m] \tag{34}$$

at each time step, $\tau[m]$ either increases by a known quantity Δ^+ or decreases by a known quantity $|\Delta^-|$. In either case, $0 \leq \tau[m] < 1$, for all m.

Comparing (28) and (34), we conclude that

$$\begin{aligned} c[m] &= \alpha^{\tau[m]} \\ &= \alpha^{(\tau[m-1]+\Delta)} \\ &= \alpha^{\tau[m-1]} \cdot \alpha^{\Delta} \\ &= c[m-1] \cdot \alpha^{\Delta}. \end{aligned} \tag{35}$$

Thus, at each step, the new coefficient $c[m]$ can be calculated from the previous one $c[m-1]$ by doing just one multiplication by α^{Δ}. This factor α^{Δ} can be calculated ahead of time and stored. Therefore, the following equation is used to update the coefficient in evaluating (29):

$$c[m] = E \cdot c[m-1] \tag{36}$$

where

$$E = \alpha^{\Delta}. \tag{37}$$

Since there are two possible values of Δ, there are two constants E^+ and E^- corresponding to $\Delta = \Delta^+$ and $\Delta = \Delta^-$, which need to be calculated and stored. Thus, each term in (29) can be calculated very efficiently with only two multiplications and one addition for each m. One of these multiplications is seen in (36), and the other is inherent in (29).

The final step in computing the output is to filter $w[n]$ by $h[n]$. $h[n]$ is given in (17) and can be implemented by the difference equation

$$y[n] = w[n] + \alpha y[n-1]. \tag{38}$$

This is a first-order recursive filter that requires only a single multiply-accumulate operation in order to generate each output sample $y[n]$.

In summary, we propose performing the sampling rate conversion process in two steps. First, the intermediate variable $w[n]$ is calculated using (29). At each time step, the new coefficient $c[m]$ is determined by (36) and then multiplied by $x[m]$. This product is then added to the correct sample of $w[n]$. In the second step, $w[n]$ is filtered using the difference equation given in (38) to produce the final output $y[n]$.

In this section, we have described a very efficient algorithm for performing sampling rate conversion by simulating the

[1] Δ^+ may in fact be equal to zero; therefore, the term "positive" is not precise. When Δ^+ is zero, we are performing integer upsampling and would therefore use a more straightforward algorithm (see [2]).

system shown in Fig. 5, where $h(t)$ is a first-order filter. The equivalent DT system is shown in Fig. 10. The implementation of this system is summarized by (29), (36), and (38). In the next two sections, we extend the algorithm to accommodate second- and Nth-order choices for the filter $h(t)$.

IV. Second-Order Approximation

In the previous section, we examined how to simulate the generic sampling rate converter, which is shown in Fig. 5, very efficiently when $h(t)$ is a first-order filter. However, in order for the filter $h(t)$ to have good image-rejection and anti-aliasing properties that are essential to high-quality sampling rate conversion, we need to use a lowpass filter of a much higher order. In this section, we extend the results of the previous section to the case where $h(t)$ is a second-order filter.

Assume that the transfer function of the filter $H(s)$ has one complex conjugate pole pair, one zero on the real axis, and one zero at infinity. In addition, assume that the poles are not on the real axis. In this case, the impulse response is a decaying sinusoid, which can be expressed in the form

$$h(t) = \gamma^t \sin(\omega t + \phi) u(t). \tag{39}$$

As in the previous section, we would like to represent this impulse response as the convolution of a finite-duration window $f(t)$ and an infinite train of impulses $h_\delta(t)$. In Appendix B, we show that

$$\begin{aligned} h(t) &= \gamma^t \sin(\omega t + \phi) u(t) \\ &= f(t) * h_\delta(t) \end{aligned} \tag{40}$$

where

$$\begin{aligned} f(t) = \gamma^t [&\sin(\omega t + \phi) r(t) \\ &- \sin(\omega(t-2) + \phi) r(t-1)] \end{aligned} \tag{41}$$

and

$$h_\delta(t) = \sum_{k=0}^{\infty} \gamma^k \frac{\sin(\omega(k+1))}{\sin \omega} \delta(t-k). \tag{42}$$

The decomposition of $h(t)$ in (40) is illustrated in Fig. 11. With this choice for $f(t)$ and $h_\delta(t)$, the system shown in Fig. 8 is still valid.

As before, the sampling and filtering operations can be interchanged, and the CT section can be replaced by a time-varying filter to give the system shown in Fig. 10, with

$$h[n] = \gamma^n \frac{\sin(\omega(n+1))}{\sin \omega} u[n]. \tag{43}$$

Here, $h[n]$ is the impulse response of a second-order, all-pole DT filter.

We see that $w[n]$ is related to $x[m]$ by

$$w[n] = \sum_{m=-\infty}^{\infty} x[m] f(n - mT_{\text{in}}). \tag{44}$$

For any value of m, the function $f(n - mT_{\text{in}})$ is nonzero for exactly two values of n. This is because $f(\tau)$ is nonzero only where $0 \le \tau < 2$. The values of n that make $f(n - mT_{\text{in}})$ nonzero are $n = \lceil mT_{\text{in}} \rceil$ and $n = \lceil mT_{\text{in}} \rceil + 1$. By using the notation in (25), we can rewrite $f(n - mT_{\text{in}})$ as

$$\begin{aligned} f(n - mT_{\text{in}}) = f(\tau[m]) &\delta[n - \lceil mT_{\text{in}} \rceil] \\ &+ f(\tau[m] + 1) \delta[n - \lceil mT_{\text{in}} \rceil - 1]. \end{aligned} \tag{45}$$

We can therefore rewrite (44) as

$$\begin{aligned} w[n] = \sum_{m=-\infty}^{\infty} x[m] \{ &a[m] \delta[n - \lceil mT_{\text{in}} \rceil] \\ &+ b[m] \delta[n - \lceil mT_{\text{in}} \rceil - 1] \} \end{aligned} \tag{46}$$

where

$$a[m] = f(\tau[m]) \tag{47}$$

and

$$b[m] = f(\tau[m] + 1). \tag{48}$$

We now consider the evaluation of the summation of (46). At each step, we multiply $x[m]$ by two different coefficients, whereas in (29), we only needed to multiply by one coefficient. These two coefficients, which can be obtained from (41), (47), and (48), are

$$a[m] = \gamma^{\tau[m]} \sin(\omega \tau[m] + \phi) \tag{49}$$

and

$$b[m] = -\gamma^{(\tau[m]+1)} \sin(\omega(\tau[m] - 1) + \phi). \tag{50}$$

For the first-order case, we saw that at each step, the new coefficients could be calculated recursively using the previous coef-

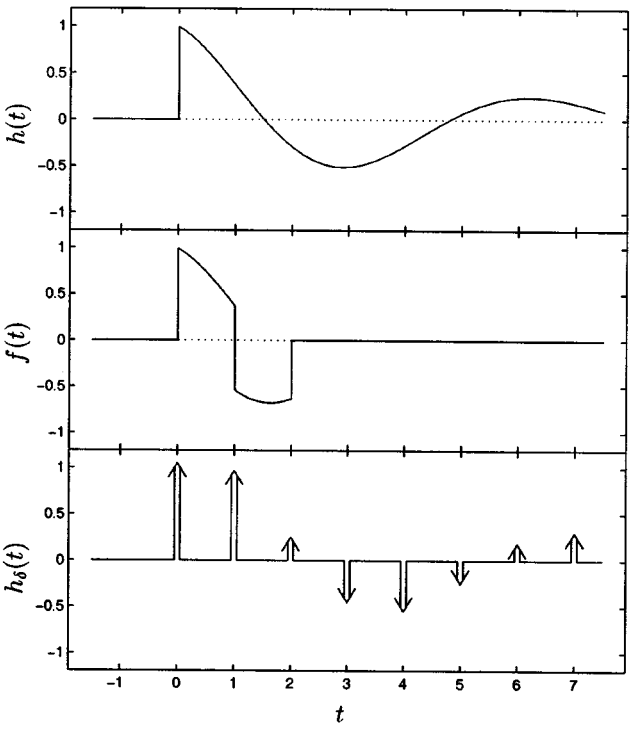

Fig. 11. Illustration of the decomposition $h(t) = f(t) * h_\delta(t)$. $h(t)$ is a decaying sinusoid, $f(t)$ is a finite-duration function, and $h_\delta(t)$ is a series of CT impulses.

ficient [see (36)]. The corresponding equations for the second-order case are

$$a[m] = A \cdot a[m-1] + B \cdot b[m-1] \tag{51}$$
$$b[m] = C \cdot a[m-1] + D \cdot b[m-1] \tag{52}$$

where

$$A = \gamma^{\Delta}(\cos\Delta\omega + \sin\Delta\omega\cot\omega) \tag{53}$$
$$B = \gamma^{(\Delta-1)}\frac{\sin\Delta\omega}{\sin\omega} \tag{54}$$
$$C = -\gamma^{(\Delta+1)}\frac{\sin\Delta\omega}{\sin\omega} \tag{55}$$
$$D = \gamma^{\Delta}(\cos\Delta\omega - \sin\Delta\omega\cot\omega) \tag{56}$$

and Δ is given by (30). The derivation of (51) through (56) is quite involved and is consequently omitted.

Since A, B, C, and D can be calculated ahead of time and stored, at each step the new coefficients can be calculated from the previous coefficients by doing only four multiplications, as seen in (51) and (52). Since there are two possible values for Δ, two different values for the constants A, B, C, and D need to be calculated and stored. We call these A^+, A^-, etc.

We see that when $h(t)$ in Fig. 5 is a second-order filter, $w[n]$ can still be generated efficiently. We do this by evaluating the summation in (46). At each step, we perform two multiplications since the input $x[m]$ must be multiplied by two different coefficients. Two additions are also required, as seen in (46). We must also determine the new coefficients from the previous coefficients. This requires four multiplications and two additions, as seen in (51) and (52).

We now derive the difference equation used to generate $y[n]$ from $w[n]$. When $h(t)$ is a second-order filter, the IIR filter in Fig. 10 $h[n]$ is also very easy to implement. By taking the z-transform of (43), we get

$$H(z) = \frac{1}{1 - (2\gamma\cos\omega)z^{-1} + \gamma^2 z^{-2}}. \tag{57}$$

Given that $Y(z) = W(z)H(z)$, we can take an inverse z-transform to get the difference equation

$$y[n] = w[n] + (2\gamma\cos\omega)y[n-1] - \gamma^2 y[n-2]. \tag{58}$$

The coefficients in this equation ($2\gamma\cos\omega$ and $-\gamma^2$) are constants that can be predetermined. Implementing this difference equation requires two multiply-accumulate operations to generate each sample of $y[n]$.

As in the first-order case, the second-order case can be implemented very efficiently by using the system shown in Fig. 10. The time-varying filter block represents our efficient evaluation of (46), which results from using (51) and (52) to determine the coefficients $a[m]$ and $b[m]$.

In this section, we have described an algorithm for performing sampling rate conversion by efficiently simulating the system shown in Fig. 5, where $H(s)$ is a second-order filter.[2] Even though a second-order filter performs better than a first-order filter, it is still not sufficient to achieve high-quality sampling rate conversion. In the next section, we explain how to use the results for first- and second-order filters in order to implement higher order filters.

[2]There are some second-order filters that we have not shown how to implement. Specifically, if ω is a multiple of π, then we must deal with this case separately in order to avoid the division by zero in (42) and (43).

V. NTH-ORDER APPROXIMATION

In Section III, we described how to efficiently simulate the system shown in Fig. 5, where $h(t)$ was chosen to be a first-order filter. In Section IV, we described how to do the same thing, where $h(t)$ is chosen to be a second-order filter. In this section, we show that these two cases are sufficient to implement any filter with a rational, proper transfer function, provided all the poles are distinct.[3] By proper, we mean that the order of the numerator polynomial is less than the order of the denominator polynomial. We do this by representing the Nth-order system function $H(s)$ as the sum of first- and second-order parts by means of a partial fraction expansion. Each of these parts can be implemented as a branch in a parallel structure. We now explain this idea in more detail.

Assume $H(s)$ is an arbitrary proper rational system function that has N distinct poles and up to $N-1$ zeros in the finite s-plane. Then, $H(s)$ can be written in the form

$$H(s) = G\frac{\prod_{k=1}^{N_Z}(s - Z_k)}{\prod_{k=1}^{N}(s - P_k)} \tag{59}$$

where G is some constant gain, Z_k and P_k are the zeros and poles of $H(s)$, respectively, and N_Z is the number of zeros in the finite s-plane, with $N_Z < N$.

A partial-fraction expansion can be performed in order to decompose this high-order filter into the sum of several first- or second-order filters. If there are N_r poles on the real axis, then there are N_p complex conjugate pole pairs, where $N_r + 2N_p = N$. In doing the partial-fraction expansion, we keep the complex conjugate pole pairs together so that we end up with N_r first-order terms and N_p second-order terms. Taking the inverse Laplace transform of this partial fraction expansion yields an impulse response of the form

$$h(t) = \sum_{i=1}^{N_r}\beta_i\alpha_i^t u(t) + \sum_{i=1}^{N_p}\theta_i\gamma_i^t\sin(\omega_i t + \phi_i)u(t). \tag{60}$$

Thus, $h(t)$ is the sum of N_r first-order terms and N_p second-order terms for a total of $M = N_r + N_p$ terms. We refer to each of these terms using in index k, as $h_k(t)$ for $k = 1, 2, \ldots, M$. Therefore

$$h(t) = \sum_{k=1}^{M} h_k(t). \tag{61}$$

This decomposition is illustrated in Fig. 12. By expressing $h(t)$ in this form, we can implement the filter $h(t)$ in Fig. 5 using the parallel structure shown in Fig. 13. The sampling operation,

[3]In this paper, we do not deal with the case in which the transfer function has repeated poles. This repeated-pole case is rare and, in fact, never occurs if the filter is an elliptic, Butterworth, or Chebyshev lowpass filter. However, in principle, there is nothing that prevents the algorithm from being extended so that it can accommodate repeated poles.

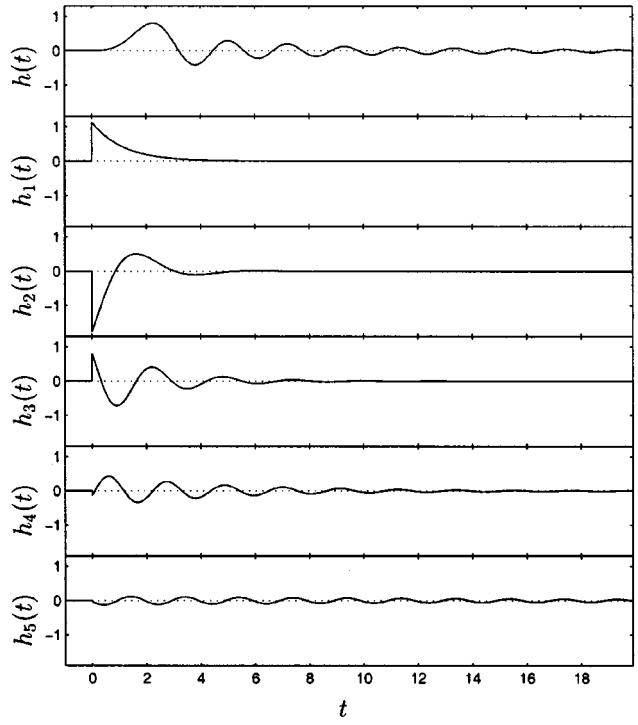

Fig. 12. $h(t)$ is the impulse response of a ninth-order elliptic filter. The $h_k(t)$s are the individual partial fraction components that add together to give $h(t)$. $h_1(t)$ is a decaying exponential and $h_2(t)$ through $h_5(t)$ are all decaying sinusoids.

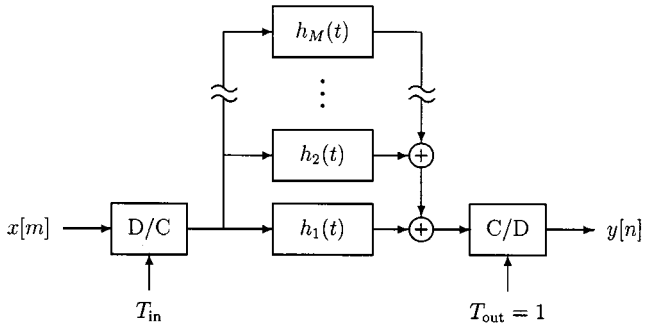

Fig. 13. System that is equivalent to the one shown in Fig. 5, when $h(t)$ is chosen as a Nth-order rational filter. Each of the branches in the parallel structure contains either a first- or second-order filter. $M = N_r + N_p$, where N_r is the number of real poles of $H(s)$, N_p is the number of complex conjugate pole pairs, and $N = N_r + 2N_p$.

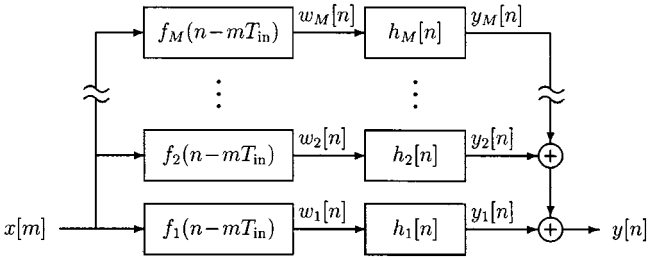

Fig. 14. Final system for high-quality sampling rate conversion. Each branch implements the filter whose impulse response is one of the terms in (60), and is either a first- or second-order filter.

represented by the C/D block, can be done before the addition, with one C/D block appearing on each branch. The D/C block can also be moved onto the individual branches producing a parallel structure in which each branch has the same overall structure as the original system shown in Fig. 5. Since each of the filters is now either a first- or second-order filter, each branch can be implemented using the methods described in the previous two sections.[4] This results in the final system shown in Fig. 14.

In the next section, we give an overview of the algorithm and summerize the implementation steps. We then analyze the overall computational complexity of the algorithm.

VI. Summary of Algorithm

This summary includes design as well as implementation steps. With reference to Fig. 5, we assume that $T_{\text{out}} = 1$ and that we are given T_{in}. We also assume that a specification on the frequency response of the filter has been determined.

A. Design and Preparation

Here, we describe the setup stage of the algorithm, which must be performed before we start processing the input signal. The following is a list of the setup steps.

1) **Design the filter:** The filter $h(t)$ must be designed so that its frequency response meets the given specification. The transfer function $H(s)$ should be rational with distinct poles and must be proper. That is, the order of the numerator must be less than the order of the denominator. Standard elliptic, Butterworth, or Chebyshev filters may be used. If the filter is elliptic or Chebyshev type II, then the order should be chosen to be odd so that there is a zero at infinity, forcing the system function to be proper.
2) **Decompose the filter:** The filter's impulse response $h(t)$ should be decomposed into a partial fraction expansion as explained in Section V so that it can be expressed in the form given by (60). For each first-order term, the parameters α and β must be found. For each second-order term, the parameters γ, ω, ϕ, and θ must be found.
3) **Determine the constants:** From the filter parameters, the values of the constants can be precomputed and stored. For each first-order term, E^+ and E^- must be determined using (37). For each second-order term, A^+, A^-, B^+, B^-, C^+, C^-, D^+, and D^- must be determined using (53)–(56). These constants must be stored and will be used when the algorithm is running. The plus and minus versions are found by using $\Delta = \Delta^+$ and $\Delta = \Delta^-$, respectively, given by (32) and (33). Finally, the difference equation coefficients should be determined and stored. There is one coefficient α for each first-order term, as seen in (38), and two coefficients $2\gamma\cos\omega$ and $-\gamma^2$ for each second-order term, as seen in (58). Only the constants A^+, A^-, etc., and the difference equation coefficients need to be stored. The other parameters may be discarded.
4) **Initialize the coefficients:** We need to initialize the values of the coefficient $c[n]$ for each first-order term and the two coefficients $a[n]$ and $b[n]$ for each second-order term. If $n = 0$, then $\tau[n] = \tau[0] = 0$. Therefore, for each

[4]As mentioned before, if ω is a multiple of π for any of the second-order branches, then this must be dealt with separately. We ignore this added complication because it is not commonly encountered in practice.

first-order term, $c[0] = \beta$, and for each second-order term, $a[0] = \theta \sin \phi$ and $b[0] = \theta \sin(\phi - \omega)$. We obtain these values by first letting $\tau[0] = 0$ in (28), (49), and (50) and then performing the appropriate scaling.

B. Running the Algorithm

We now describe the runtime stage of algorithm, which is performed while we are processing the input signal. Note that this part of the algorithm can be executed in real time. Since the filter has been decomposed into first- and second-order components, we implement our algorithm in a parallel structure, as shown in Fig. 14. There are two sets of tasks that need to be accomplished. The first set of tasks occur at the input sampling rate. These are performed whenever the next sample of our input $x[n]$ becomes available. The second set of tasks occur at the output sampling rate and are performed whenever the next sample of the output $y[n]$ is required. We now describe these tasks in detail.

1) Input Becomes Available: Whenever the next input sample becomes available, we perform the following steps for each of the branches. The steps are described for the ith branch and the kth input sample.

1) **Determine the sign of Δ:** We must determine whether $\Delta[k] = \Delta^+$ or $\Delta[k] = \Delta^-$ will cause $\tau[k]$ in (34) to satisfy $0 \leq \tau[k] < 1$. If $\Delta[k] = \Delta^+$, then we use the plus constants $A = A^+$, $B = B^+$, etc. If $\Delta[k] = \Delta^-$, then we use the minus constants $A = A^-$, $B = B^-$, etc.
2) **Update the coefficients:** If the ith branch is first order, there is only one coefficient $c[k]$, which is updated using (36). If the branch is second order, then there are two coefficients $a[k]$ and $b[k]$, which are updated using (51) and (52).
3) **Multiply by the input:** We then multiply the input $x[k]$ by the coefficient(s) and add the result(s) to the correct location(s) in $w_i[n]$. For the first-order case, the result is added to $w_i[\lceil kT_{\text{in}} \rceil]$, in accordance with (29). For the second-order case, there are two results that are added to $w_i[\lceil kT_{\text{in}} \rceil]$ and $w_i[\lceil kT_{\text{in}} \rceil + 1]$, respectively, in accordance with (46).

2) Output Is Required: Whenever it is time to output the next sample of $y[n]$, the following steps are performed. Assume we need the kth output sample.

1) **Apply the difference equation:** We generate the next sample of the output for the ith branch $y_i[k]$ from the intermediate variable $w_i[k]$ using the appropriate difference equation. This must be done for all M branches. If the branch is first order, then the difference equation used is (38). If the branch is second order, then the difference equation used is (58).
2) **Sum over the branches:** We then add all of the outputs from each branch to produce the final output. $y[k] = \sum_{i=1}^{M} y_i[k]$, where M is the number of branches. We then output this final value $y[k]$.

The two sets of tasks that run at two different rates can be implemented on a digital signal processing (DSP) chip, with each set of tasks programmed as an interrupt service routine (ISR). The first set of tasks can be performed by the input ISR, whereas the second set can be performed by the output ISR.

VII. Computational Complexity

In this section, we analyze the computational complexity of our algorithm. As a measure of complexity, we use the number of multiplications per output sample (MPOS), that is, the average number of multiplications needed to produce one sample of $y[n]$. The number of additions is not counted.

For simplicity, we assume that $h(t)$ in Fig. 5 is chosen as a Nth-order elliptic, Butterworth, or Chebyshev lowpass filter, where N is odd. This implies that $H(s)$ has one real pole and $N_p = (N-1)/2$ complex conjugate pole pairs. Consequently, our implementation requires $N_p + 1$ branches. We also assume that our input consists of Q channels that use the same time axis and that all need to be converted. For example, a stereo audio signal has two channels: the left channel and the right channel. This allows us to perform the parts of the algorithm that do not need to be repeated for each channel only once. We first analyze the part of the algorithm that runs at the input sampling rate and then analyze the part that runs at the output sampling rate.

With reference to Section VI-B1, step 1 requires no multiplications. Step 2 requires one multiplication for the first-order branch and four multiplications for each second-order branch. All together, we have $2N-1$ multiplications for step 2, independent of the number of channels since the coefficients only need to be updated once. Step 3 requires one multiplication for the first-order branch and two multiplications for each second-order branch for each channel. This gives NQ multiplications for step 3. Since these multiplications are performed at the input sampling rate, the total number of MPOS required for this part of the algorithm is $(NQ + 2N - 1)/T_{\text{in}}$ MPOS.

Now, we analyze the computational complexity of the part of the algorithm that runs at the output sampling rate, with reference to Section VI-B2. Step 1 requires one multiplication for the first-order branch and two multiplications for each of the second-order branches. These multiplications need to be done for each of the Q channels, and therefore, this gives a total of NQ MPOS. Step 2 does not require any multiplications.

Our conclusion is that for an Nth-order filter with one real pole, our algorithm requires $NQ + [(NQ + 2N - 1)/T_{\text{in}}]$ MPOS to convert Q channels of data. Here, N is taken to be odd. Similarly, we can show that if N were even such that $H(s)$ had no poles on the real axis, the computation required is $NQ + [N(Q + 2)/T_{\text{in}}]$ MPOS.

We now look at a specific example in order to compare the performance of our algorithm with conventional methods. Suppose that we have two digital audio streams that we would like to add together and output through the same digital to analog converter. Each input is two-channel (stereo) audio. However, they are being generated by two different sources. As a result, one of them is arriving at exactly 44.1 kHz, and the other is arriving at, say, 44.097 94 kHz because the clock from that CD player is slow. We would like to increase the sampling rate of the second signal by a small fraction. For this case, $T_{\text{in}} = 0.999\,532\,8\ldots$

We choose $h(t)$ in Fig. 5 to be an elliptic filter with a passband edge frequency of 20 kHz and a stopband edge frequency of 24.1 kHz. These band edge frequencies are scaled up by a factor of 44.1 kHz in order to compensate for the fact that $T_{\text{out}} = 1$. We also allow ± 0.2 dB of ripple in the passband and require at

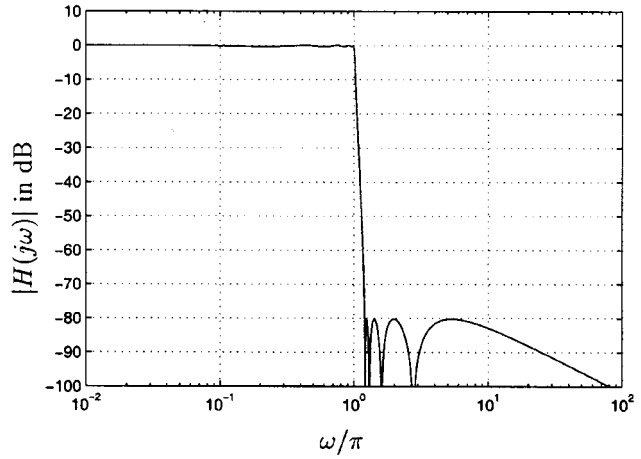

Fig. 15. Magnitude of the frequency response of the ninth-order elliptic filter used in the design example.

least 80 dB of attenuation in the stopband. To match this specification, we use a ninth-order elliptic filter, whose magnitude response is shown in Fig. 15. The impulse response $h(t)$ and its individual partial fraction components are shown in Fig. 12.

By using the complexity formula given earlier in this section, we find that $18 + (35/T_{\text{in}})$ MPOS or 2.34 million multiplications/s are required.

In order to match the same specification using conventional FIR filter techniques, a time-varying impulse response of length 27 is required (31 if the filter is linear phase). Therefore, 27 multiplications per channel are needed for each output sample. That is, 2.38 million multiplications/s—which is only slightly more than the proposed method. The most significant difference between the two methods is the amount of memory required.

If the taps of the time-varying filter are generated using nearest-neighbor interpolation of a stored impulse response, then no computation is needed to generate them. This is why only 2.38 million multiplications/s are required. Based on Ramstad's analysis described in Section II, the largest value of ϵ that would allow a given specification to be met can be obtained. In this example, ϵ would have to be about 0.0002 s. Consequently, the table of stored coefficients would need to be quite large; it would need to contain about 140 000 points.

More typically, linear interpolation is used to generate the taps of the time-varying filter. In that case, one multiplication per tap would also be needed in order to perform the linear interpolation. This would increase the total amount of computation required to 3.57 million multiplications/s. Because linear interpolation is used, ϵ can be about 100 times larger than for nearest-neighbor interpolation. Therefore, only about 1400 points need to be stored.

The memory requirements for our algorithm are very small. Only about 75 constants need to be stored for the ninth-order elliptic filter used in this example.

In order to emphasize the computational efficiency of our algorithm, suppose that five channels need to be converted, e.g., in a digital surround-sound audio system. In this case, our algorithm only requires about 66% of the MPOS required by the conventional FIR technique with linear interpolation of the impulse response. In addition, our algorithm only uses about 5% of the memory used by the FIR method.

If we use the method presented by Ramstad [5] for implementing a rational filter, then the savings are not as dramatic. Assume that we wish to implement the same ninth-order elliptic filter we have used in this section. Then, a time-varying filter with nine taps must be implemented. Let us also assume that the computation allowed for calculating the coefficients is the same as for the proposed method, with two multiplications used for updating each coefficient. This corresponds to a quadratic interpolation of the stored impulse response. In this case, ϵ is approximately 0.03 s. Then, about 300 points need to be stored, which is four times that required by the proposed method.

VIII. Conclusion

In this paper, we presented a novel algorithm for arbitrary ratio sampling rate conversion. The algorithm is based on discrete-time simulation of a continuous-time system. The algorithm has advantages in computational and memory requirements over traditional FIR techniques because the coefficients that are used can be calculated recursively. We now highlight some of the salient features of our algorithm and present its known drawbacks.

Under some circumstances, this algorithm is computationally more efficient than traditional approaches. This is especially true for multichannel systems, where the computation can approach that of straightforward digital filtering, i.e., one multiplication per output sample (MPOS) per pole or zero when the input and output sampling rates are approximately equal.

The memory requirements of the algorithm are very small. Because the algorithm generates the coefficients recursively, the amount of memory needed is only a fraction of that needed for conventional techniques.

Another advantage is that filters can be designed easily using conventional analog methods such as elliptic, Butterworth, or Chebyshev design techniques. If one of these methods is used, the computational requirement is $NQ + [(NQ + 2N - 1)/T_{\text{in}}]$ MPOS if N is odd and $NQ + [N(Q + 2)/T_{\text{in}}]$ MPOS if N is even. Q is the number of channels being converted, and N is the order of the filter used.

One of the limitations of this algorithm is that the filters used are not linear phase and, thus, introduce group-delay distortion. This is because causal realizable continuous-time filters are used. For audio applications, this is usually not a problem since the ear is somewhat insensitive to mild phase distortions. For video applications where linear-phase filters must be used, the causality requirement may be relaxed since the entire image may be available. In this case, linear-phase, two-sided, recursive filters may be used. Any signal that may be recorded and processed off-line can be processed using linear-phase filters because causality is no longer a requirement.

Another problem to which our algorithm is susceptible is that of coefficient drift. Since the coefficients are calculated recursively using the previous value to determine the current one, small errors due to roundoff noise can tend to accumulate over time. The expected magnitude of our error actually grows with

time. This can be dealt with by using a low-rate background process running on our DSP chip in order to periodically recalculate the correct values of the coefficients. Since the coefficients are being updated periodically, it may be possible to extend the algorithm to allow for slowly varying conversion ratios, provided that the practical issues are dealt with.

As a final note, we have presented an implementation in which the fixed recursive part of the filter operates at the output sampling rate, whereas the time-varying FIR part operates at the input sampling rate. A different implementation can be derived in which the fixed recursive part operates at the input sampling rate, whereas the time-varying FIR part operates at the output sampling rate. In either case, the coefficients of the time-varying filter can be calculated recursively. Depending on whether the sampling rate is being increased or decreased, one of these implementations may offer some computational advantage over the other.

APPENDIX

Here, we derive two of the results used in the paper.

A. First-Order Filter Decomposition

We show that

$$\alpha^t u(t) \equiv \alpha^t r(t) * \sum_{k=0}^{\infty} \alpha^k \delta(t-k). \tag{62}$$

This can be seen graphically in Fig. 7. The convolution in the right-hand side of the equation can be done by inspection, giving the following.

$$\begin{aligned}
&\sum_{k=0}^{\infty} \alpha^k \alpha^{(t-k)} r(t-k) \\
&\quad = \sum_{k=0}^{\infty} \alpha^t r(t-k) \\
&\quad = \alpha^t \sum_{k=0}^{\infty} r(t-k) \\
&\quad = \alpha^t \sum_{k=0}^{\infty} u(t-k) - u(t-(k+1)) \\
&\quad = \alpha^t u(t).
\end{aligned}$$

Therefore, the equivalence holds.

B. Second-Order Filter Decomposition

Here, we show that

$$\begin{aligned}
&\gamma^t \sin(\omega t + \phi) u(t) \\
&\quad \equiv \gamma^t [\sin(\omega t + \phi) r(t) \\
&\qquad - \sin(\omega(t-2) + \phi) r(t-1)] \\
&\qquad * \sum_{k=0}^{\infty} \gamma^k \frac{\sin(\omega(k+1))}{\sin \omega} \delta(t-k)
\end{aligned} \tag{63}$$

provided that ω is not a multiple of π.

This is more difficult to show than the first-order decomposition. A graphical representation is presented in Fig. 11. This figure does not provide much intuition; therefore, we resort to a mathematical proof.

Again, the convolution in the right-hand side of the equation can be done by inspection, giving

$$\begin{aligned}
\sum_{k=0}^{\infty} \gamma^k \frac{\sin(\omega(k+1))}{\sin \omega} \gamma^{(t-k)} &[\sin(\omega(t-k) + \phi) r(t-k) \\
&- \sin(\omega(t-k-2) + \phi) r(t-k-1)].
\end{aligned} \tag{64}$$

Now, substitute $t = m + \tau$, where m is an integer, and $0 \le \tau < 1$. Because of the rectangular windows $r(t-k)$ and $r(t-k-1)$, this sum reduces to only two terms when $t \ge 0$. This gives

$$\begin{aligned}
&\gamma^m \frac{\sin(\omega(m+1))}{\sin \omega} \gamma^{\tau} \sin(\omega\tau + \phi) \\
&\quad - \gamma^{m-1} \frac{\sin \omega m}{\sin \omega} \gamma^{\tau+1} \sin(\omega(\tau-1) + \phi) \\
&\qquad = \frac{\gamma^{(m+\tau)}}{\sin \omega} [\sin(\omega(m+1)) \sin(\omega\tau + \phi) \\
&\qquad\quad - \sin \omega m \sin(\omega(\tau-1) + \phi)] \\
&\qquad = \frac{\gamma^{(m+\tau)}}{2 \sin \omega} [\cos(\omega(m+1-\tau) - \phi) \\
&\qquad\quad - \cos(\omega(m+1+\tau) + \phi) \\
&\qquad\quad - \cos(\omega(m-\tau+1) - \phi) \\
&\qquad\quad + \cos(\omega(m+\tau-1) + \phi)] \\
&\qquad = \frac{\gamma^{(m+\tau)}}{2 \sin \omega} [\cos(\omega(m+\tau) + \phi - \omega) \\
&\qquad\quad - \cos(\omega(m+\tau) + \phi + \omega)] \\
&\qquad = \frac{\gamma^{(m+\tau)}}{\sin \omega} \sin(\omega(m+\tau) + \phi) \sin \omega \\
&\qquad = \gamma^t \sin(\omega t + \phi)
\end{aligned}$$

where we use the identity

$$\sin A \sin B \equiv \frac{\cos(A-B) - \cos(A+B)}{2}. \tag{65}$$

This concludes the proof of the decomposition.

ACKNOWLEDGMENT

The authors would like to thank Y. C. Eldar for her many suggestions on improvements to the manuscript. The authors also thank the anonymous reviewers for many helpful comments and for pointing out additional references.

REFERENCES

[1] A. I. Russell and P. E. Beckmann, Sampling rate conversion, Bose Corp., Framingham, MA, U.S. patent (pending).
[2] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Englewood Cliffs, Nj: Prentice-Hall, 1993, sec. 4.3.
[3] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1983.
[4] A. I. Russell, "Efficient rational sampling rate alteration using IIR filters," *IEEE Signal Processing Lett.*, vol. 7, pp. 6–7, Jan. 2000.
[5] T. A. Ramstad, "Digital methods for conversion between arbitrary sampling frequencies," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 577–591, June 1984.
[6] T. Saramäki and T. Ritoniemi, "An efficient approach for conversion between arbitrary sampling frequencies," *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, pp. 285–288, 1996.

[7] J. O. Smith and P. Gossett, "A flexible sampling-rate conversion method," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 2, pp. 19.4.1–19.4.2, Mar. 1984.
[8] R. E. Crochiere and L. R. Rabiner, "Interpolation and decimation of digital signals—A tutorial review," *Proc. IEEE*, vol. 69, pp. 300–331, Mar. 1981.
[9] B. Liu and P. A. Franaszek, "A class of time-varying digital filters," *IEEE Trans. Circuit Theory*, vol. CT-16, pp. 467–471, Nov. 1969.
[10] R. Lagadec and H. O. Kunz, "A universal, digital sampling frequency converter for digital audio," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, pp. 595–598, 1981.
[11] R. Lagadec, D. Pelloni, and D. Weiss, "A 2-channel, 16-bit digital sampling frequency converter for professional digital audio," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, pp. 93–96, 1982.
[12] A. L. Wang and B. S. Read, Arbitrary-ratio sampling rate converter using approximation by segmented polynomial functions, Chromatic Res., Inc., Mountain View, CA, U.S. Patent 5 859 787, Jan. 1999.
[13] A. L. Wang, Wavefunction sound sampling synthesis, ATI Int. SRL, Barbados, St. Kitts/Nevis, U.S. Patent 6 124 542, Sept. 2000.

Andrew I. Russell (S'01) was born in Kingston, Jamaica, in 1975. He received the S.B. and M.Eng. degrees from the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (MIT), in 1998 and 1999, respectively. He is pursuing the Ph.D. degree at MIT.

Since 1998, he has been with the Digital Signal Processing Group, Research Laboratory of Electronics, MIT. His research interests include filter design, sampling rate conversion, nonuniform sampling, and nonlinear signal processing. He has served as a Teaching Assistant for classes in linear systems, digital signal processing, and biomedical signal processing.

Mr. Russell received the 1996 Organization of American States Fellowship for Jamaica.

Paul E. Beckmann (S'90–M'92) received the S.B. and S.M. degrees in electrical engineering in 1989 and the Ph.D. degree in 1992, all from the Massachusetts Institute of Technology, Cambridge.

He held the Rockwell Doctoral Fellowship from 1989 to 1992 while working in the MIT Digital Signal Processing Group. He is currently Director of Research at Enuvis, Inc., South San Francisco, CA, and is involved in the research and productization of GPS algorithms. From 1992 to 2001, he was with Bose Corporation, Framingham, MA. He held a number of positions in research and product development and eventually lead the home audio advanced development group. At Bose, he specialized in the theory and implementation of efficient audio algorithms.