Douglas W. Barker

Efficient Resampling Implementations

"DSP Tips and Tricks" introduces practical design and implementation signal processing algorithms that you may wish to incorporate into your designs. We welcome readers to submit their contributions to the Associate Editors Rick Lyons (R.Lyons@ieee.org) or Britt Rorabaugh (dspboss@aol.com).

The process of sample rate conversion (resampling) of a discrete sequence has many applications in digital signal processing, particularly in the field of digital communications. This article describes efficient implementations used to change the sample rate of a discrete signal by an arbitrary factor.

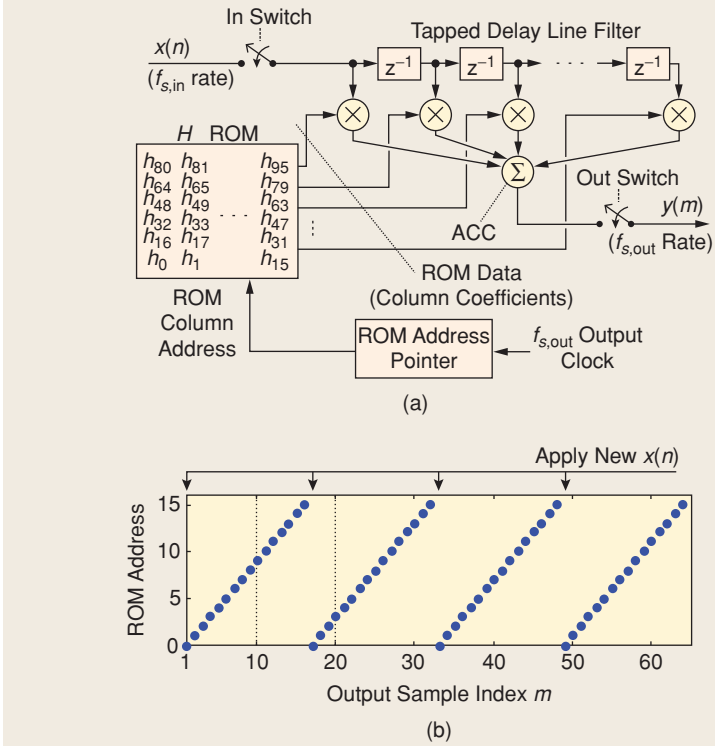
RESAMPLING USING POLYPHASE FILTERS

The fundamental process of resampling a discrete sequence by an integer factor involves either inserting a sequence of zero-valued samples between each input sample followed by low-pass filtering (interpolation) or low-pass filtering an input signal and discarding some of those filter output samples (decimation). Polyphase filters were developed to improve the computational efficiency of the low-pass filtering in interpolation by eliminating the multiply-by-a-zero-sample operations and avoiding the computation of the low-pass filter output samples to be discarded in decimation [1], [2]. With these thoughts in mind, we assume the reader is familiar with the nature of polyphase resampling filters, and here we present practical and efficient polyphase filter implementations that the reader should consider for their next resampling design.

RESAMPLING BY AN INTEGER FACTOR

To reduce input signal data storage requirements, it's smart to implement resampling filtering using a single tapped-delay line filter. Our recommended design of a single delay line, multiple coefficient set, interpolator is shown in Figure 1(a). In this figure, for example, $L = 16$ sets of coefficients (each set having length $R = 6$) are precomputed and stored as the columns of a two-dimensional matrix, H, in read-only memory (ROM). The matrix of coefficients, H, has R rows and L columns.

We implement interpolation by integer $L = 16$, with $f_{s,in}$ and $f_{s,out} = 16 f_{s,in}$ being the input and output sample rates, respectively, as follows: the "in" switch closes momentarily and applies a single $x(n)$ sample to the tapped-delay line filter. The "out" switch remains closed. At the $f_{s,out}$ output data rate, the ROM address pointer begins incrementing by one and scans through the 16 ROM column addresses, causing



[FIG1] ROM-based interpolation by an integer factor $L = 16$: (a) structure and (b) ROM column addresses.

Digital Object Identifier 10.1109/MSP.2008.923508

1053-5888/08/$25.00©2008IEEE

the ROM to output R coefficients for each column address. For each set of R coefficients, we compute an interpolated $y(m)$ output sample and reset the tapped-delay line's ACC accumulator to zero after each computation.

The ROM column addresses, repeatedly incrementing from zero to $L-1=15$, are shown in Figure 1(b) for four $x(n)$ input samples. We view the ROM address pointer operation as a kind of modulo-L numerically controlled oscillator (NCO).

Again, each dot in Figure 1(b) represents the computation of an interpolated $y(m)$ output sample. Each time the ROM address pointer is reset to zero, a time period of $1/f_{s,in}$ indicated by the down arrows in the figure, the "in" switch closes, momentarily shifting a new $x(n)$ input sample into the tapped-delay line.

Note that the "in" switch in Figure 1(a) is for descriptive purposes only. In reality, upon experiencing an overflow condition, the ROM address pointer outputs what we will call an NCO overflow control signal that is applied to an external system, causing that system to supply a new $x(n)$ input sample to our interpolator.

RESAMPLING BY AN ARBITRARY FACTOR

The interpolation and decimation processes, described earlier, can be extended to produce output sample rates that are arbitrary (noninteger) multiples of the input sample rate. Such interpolation and decimation lead to two slightly different design methods and structures and are therefore discussed separately.

INTERPOLATION BY AN ARBITRARY FACTOR

For the case of interpolation by an arbitrary factor, we use the network in Figure 1(a), but we change the ROM address pointer mechanism to the process shown in Figure 2(a), where the notation $\lfloor q \rfloor$ means the integer portion of q.

In Figure 2(a), we implement an NCO that operates as a modulo-C accumulator. In Figure 1(a), the interpolation factor L was equal to the number of columns in the ROM memory, but that is not the case when interpolating by an arbitrary factor. For this discussion, we'll let L continue to represent the number of columns in the ROM memory. The value L then determines the desired interpolation accuracy, where larger L leads to more accurate interpolation. We now define F_{int} as the desired interpolation factor

$$F_{int} = \frac{f_{s,out}}{f_{s,in}}, \qquad (1)$$

where frequency $f_{s,in}$ is our input signal's sample rate and $f_{s,out}$ is our desired upsampled output sample rate in hertz.

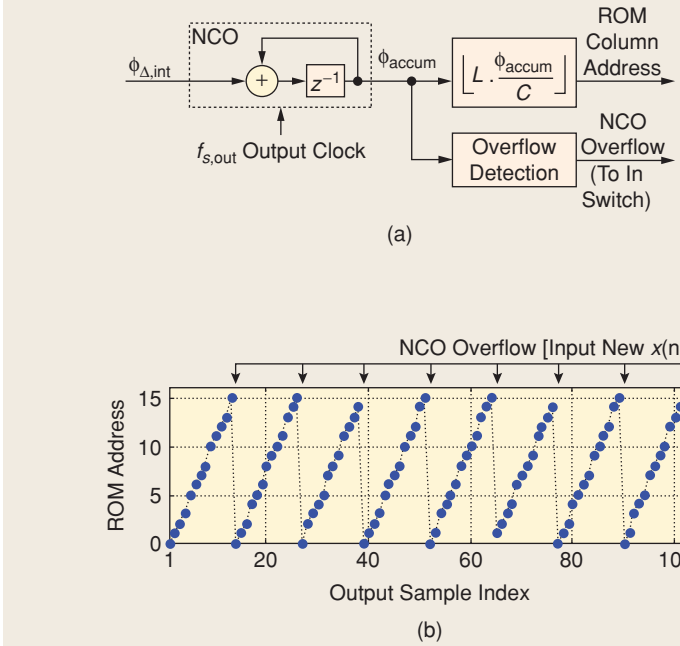
THIS ARTICLE DESCRIBES EFFICIENT IMPLEMENTATIONS USED TO CHANGE THE SAMPLE RATE OF A DISCRETE SIGNAL BY AN ARBITRARY FACTOR.

To the NCO in Figure 2(a) we apply a fixed phase increment value $\phi_{\Delta,int}$ defined by

$$\phi_{\Delta,int} = \frac{C}{F_{int}} = C \cdot \frac{f_{s,in}}{f_{s,out}}, \qquad (2)$$

where the subscript "int" means interpolation. At the $f_{s,out}$ rate, the value $\phi_{\Delta,int}$ is added to the NCO's accumulator, producing the value ϕ_{accum}. The ratio ϕ_{accum}/C is a value between zero and one. In fixed-point implementations, $\phi_{\Delta,int}$ is rounded to the nearest integer. This forces F_{int} in (1) to be a rational number, but we can approach any desired F_{int} value arbitrarily closely by increasing the value of C. Thus, the constant C is made as large as possible in any given implementation.

The function $\lfloor L \cdot \phi_{accum}/C \rfloor$ produces the integer part of $L \cdot \phi_{accum}/C$, selecting the appropriate one out of L ROM column addresses, causing the ROM to output R coefficients. For each set of R coefficients, as before, we compute an interpolated $y(m)$ output sample and reset the tapped-delay line's ACC



[FIG2] Interpolation by an arbitrary factor: (a) ROM address generation and (b) ROM addresses when $L = 16$ and $F_{int} = 12.6374$.

accumulator to zero after each computation. However, when interpolating by an arbitrary factor, the ROM column address no longer increments by one. For example, when $L = 16$ and $F_{\text{int}} = 12.6374$, the ROM column addresses are those shown in Figure 2(b), where we see the irregular progression of the integer ROM addresses. When the modulo-C NCO accumulator overflows, the NCO overflow control line causes the "in" switch to momentarily close and shift a new $x(n)$ input sample into the tapped-delay line.

For practicality, we let L and C be integer powers of two so that in Figure 2(a) the $L \cdot \phi_{\text{accum}} / C$ computation is performed, multiplier-free, as

$$L \cdot \frac{\phi_{\text{accum}}}{C} = \frac{\phi_{\text{accum}}}{2^{\log_2(C) - \log_2(L)}}. \quad (3)$$

This means we compute $\lfloor L \cdot \phi_{\text{accum}} / C \rfloor$ by merely shifting ϕ_{accum} to the right by $[\log_2(C) - \log_2(L)]$ bits. This process gives us an efficient way to extract the proper L bits of ϕ_{accum} to select the correct column of the ROM.

By shifting ϕ_{accum} to the right,the fractional portion of the quotient in (3) is truncated, causing a small error in the resulting interpolation. To keep the interpolation error less than the quantization error of the $x(n)$ input signal, represented by b-bit samples, the number of L columns in the coefficient matrix H should be chosen so that

$$L > \frac{2^{(b-1)} \cdot B}{f_{s,\text{in}}}, \quad (4)$$

where B is the two-sided bandwidth (centered at 0 Hz) of the low-pass $x(n)$ input signal measured in hertz [3].

COMPUTATION OF THE INTERPOLATION COEFFICIENTS

Each column of the H ROM in Figure 1(a) contains the R coefficients of a subfilter, where each subfilter is obtained by standard polyphase decomposition of a prototype filter designed using commercial filter design software. The prototype filter is a linear-phase tapped-delay line filter having $L \cdot R$ coefficients, and it is designed to eliminate all its input spectral components whose frequencies are above $f_{s,\text{in}}/2$. The prototype filter design is based entirely upon the factors L and R. This has a very practical implication; a single filter design specification (the pass-band width is always $f_{s,\text{in}}/2$ Hz) can be used to design any filter needed to interpolate any signal of lesser sample rate than $f_{s,\text{out}}$ to an $f_{s,\text{out}}$ rate with at least $1/(L \cdot f_{s,\text{in}})$ accuracy in interpolation timing.

TO REDUCE INPUT SIGNAL DATA STORAGE REQUIREMENTS, IT'S SMART TO IMPLEMENT RESAMPLING FILTERING USING A SINGLE TAPPED-DELAY LINE FILTER.

DECIMATION BY AN ARBITRARY FACTOR

The complementary operation to interpolation is decimation, the ROM address generation of which is shown in Figure 3.

Our resampling network effectively interpolates $x(n)$ by L' and decimates by M, where $M > L'$ and L' is defined as

$$L' = \frac{(M \cdot f_{s,\text{out}})}{f_{s,\text{in}}}. \quad (5)$$

Decimation is implemented as follows: the fixed integer phase increment value $\phi_{\Delta,\text{dec}}$, given by

$$\phi_{\Delta,\text{dec}} = \left\lfloor C \cdot \frac{f_{s,\text{out}}}{f_{s,\text{in}}} \right\rfloor \quad (6)$$

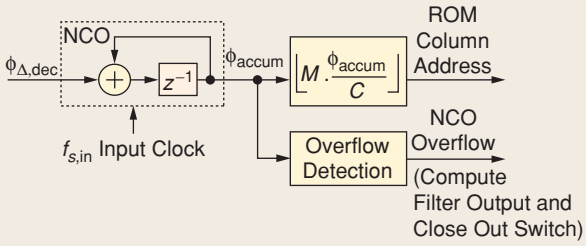
in Figure 3, is added to the NCO's accumulator at the $f_{s,\text{in}}$ rate and, again, the ratio ϕ_{accum} / C is a value between zero and one. As before, the NCO operates as a modulo-C accumulator and L represents the number of columns in the ROM memory. For decimation, the integer value M determines the resampling accuracy, and M is chosen to be an integer power of two (for the same reason that we chose L to be an integer power of two in the above interpolation case). As such, M is determined using (4) with the L replaced by M.

The value L' in (5) then is determined by M and our desired overall decimation resampling rate, and we use this value L' to compute the coefficients of the H ROM in Figure 1(a).

At the $f_{s,\text{in}}$ rate, we apply a new $x(n)$ input sample to the tapped-delay line and increment the NCO by $\phi_{\Delta,\text{dec}}$. When the NCO overflows, we compute the function $\lfloor M \cdot \phi_{\text{accum}} / C \rfloor$ defining the appropriate one out of L ROM column addresses, causing the ROM to output R coefficients. [The integer value $\lfloor M \cdot \phi_{\text{accum}} / C \rfloor$ is computed, multiplier free, as before in (3).] The R coefficients are then used to compute a $y(m)$ output sample, the "out" switch in Figure 1(a) is momentarily closed, and the tapped-delay line's ACC accumulator is reset to zero. With this operation in mind, we see that the NCO is driven with a phase increment that causes an overflow at the desired $f_{s,\text{out}}$ output sample rate.

Once the value of M is chosen based on the interpolation accuracy requirements for the output signal, we then compute the value L' using (5). Next we compute the number of columns in the H coefficient ROM matrix, L, using

$$L = \lceil L' \rceil, \quad (7)$$



[FIG3] Decimation ROM address generation.

where $\lceil L' \rceil$ means take the next integer larger than L', if L' is not an integer.

COMPUTATION OF THE DECIMATION COEFFICIENTS

Calculating the filter coefficients for the decimation case is somewhat tricky because the rows and columns of the H coefficient matrix are not spaced regularly in time. We cannot simply use a commercial filter design routine because such software only produces prototype filter impulse response samples at regularly spaced time points. The method we chose to compute the coefficients is to evaluate the continuous-time impulse response of the desired prototype FIR filter at the time points of H using a time-point matrix T. Matrix T has the same rows and columns of H and is computed first. The time points in T are specified in units based on the interpolated sample rate of $L' \cdot f_{s,in} = M \cdot f_{s,out}$. The elements of T are then given by

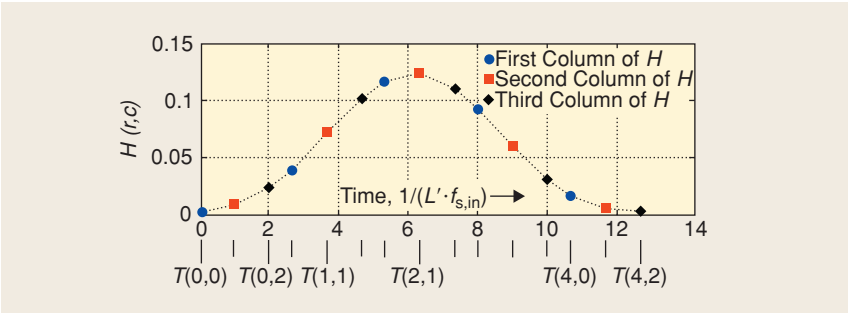
$$T(r, c) = rL' + c, \qquad (8)$$

where row index r ranges from zero to $R-1$ and column index c ranges from zero to $L-1$. The filter coefficients are calculated by evaluating the prototype filter's impulse response at the time points in T. Notice that the spacing between the columns (phases) is equal to unity, while the spacing between the rows is L', which is equal to the time spacing of the input data.

For example, if $L' = 2.666$, $L = 3$, and $R = 5$, the T matrix would be

$$T(r, c) = \begin{matrix} 0 & 1.000 & 2.000 \\ 2.666 & 3.666 & 4.666 \\ 5.333 & 6.333 & 7.333 \\ 8.000 & 9.000 & 10.000 \\ 10.666 & 11.666 & 12.666. \end{matrix}$$

If the prototype filter's impulse response is the dashed curve in Figure 4, the H decimation coefficients in this example are the dots. The tick marks at the bottom of Figure 4 show the values of time matrix T. The irregular spacing of the coefficients in time is due to the fractional nature of L'.



[FIG4] H matrix coefficients when $L' = 2.666$, $L = 3$, and $R = 5$.

It is important to note that the value of the NCO that occurs after rollover, $M \cdot \phi_{accum} / C$, represents the interval of time between the current state of the NCO and the point in time that must be interpolated. The larger the rollover value, the further back in time we must interpolate. For this reason, the columns of H must be reversed so that larger NCO overflow values produce interpolations that are further back in time.

CONCLUSIONS

This article has presented methods for efficiently resampling a discrete time signal. We described techniques to compute the coefficients for both interpolation and decimation filters, and provided information with regard to minimizing the resamplers' timing jitter errors.

MATLAB code illustrating the operation of our efficient resampling techniques, and design guidance on using field-programmable gate array (FPGA) implementations, are available on the *IEEE Signal Processing Magazine* Web site at www.ee.columbia.edu.spm.MATLAB code, and FPGA implementation guidance, available at http://apollo.ee.columbia.edu/spm/?i=external/tipsandtricks.

ACKNOWLEDGMENTS

Many thanks to Rick Lyons for his efforts in analyzing the details of these resampling designs and his assistance with the text of this article.

AUTHOR

Douglas W. Barker (doug.barker@itt.com) is a senior principal engineer with ITT Corporation, Advanced Engineering and Sciences. His interests are in communications systems design and digital signal processing. He is a Member of the IEEE.

REFERENCES

[1] A. Oppenheim, R. Schafer, and J. Buck, *Discrete-Time Signal Processing, 2/E*. Upper Saddle River, NJ: Prentice-Hall, 1989, pp. 179–184.

[2] R. Crochiere and L.R. Rabiner, *Multirate Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1983, ch. 2–3.

[3] F. Harris, *Multirate Signal Processing for Communications Systems*. Upper Saddle River, NJ: Prentice Hall, 2004, pp. 172–175.

SP