Ricardo A. Losada and Richard Lyons

Reducing CIC Filter Complexity

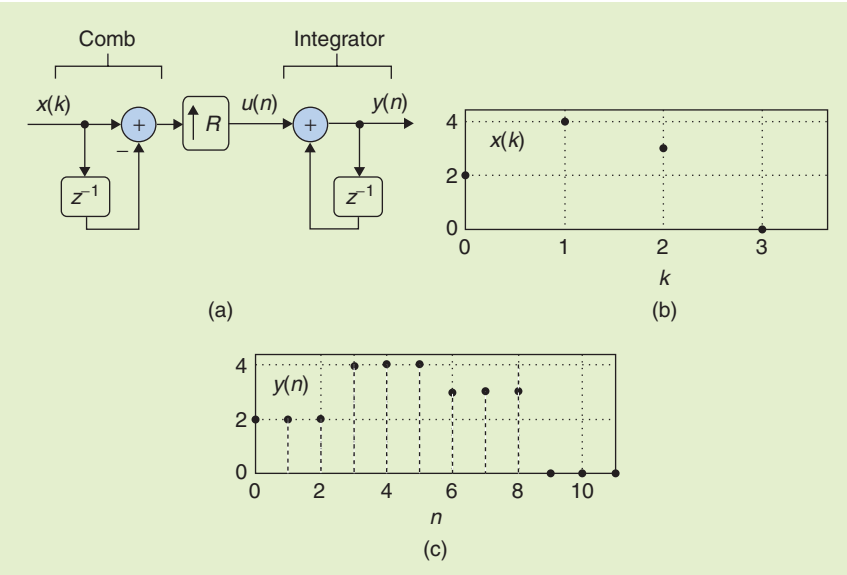
This article provides several tricks to reduce the complexity, and enhance the usefulness, of cascaded integrator-comb (CIC) filters. The first trick shows a way to reduce the number of adders and delay elements in a multistage CIC interpolation filter. The result is a multiplierless scheme that performs high-order linear interpolation using CIC filters. The second trick shows a way to eliminate the integrators from CIC decimation filters. The benefit is the elimination of unpleasant data word growth problems.

REDUCING INTERPOLATION FILTER COMPLEXITY

CIC filters are widely used for efficient multiplierless interpolation. Typically, such filters are not used stand-alone; instead, they are usually used as part of a multisection interpolation scheme, generally as the last section where the data has already been interpolated to a relatively high data rate. The fact that the CIC filters need to operate at such high rates makes their multiplierless-nature attractive for hardware implementation.

Typical CIC interpolator filters usually consist of cascaded single stages reordered in such a way that all the comb filters are grouped together as are all the integrator filters. By looking closely at a single-stage CIC interpolator, we will show a simple trick to reduce the complexity of a multistage implementation. Because multistage CIC interpolators have a single-stage CIC interpolator at its core, this trick will simplify the complexity of any CIC interpolator.

Consider the single-stage CIC interpolator in Figure 1(a). The "$\uparrow R$" symbol means insert $R-1$ zero-valued samples in between each sample of the output of the first adder comb. For illustration purposes, assume $R=3$. Now imagine an arbitrary $x(k)$ input sequence and assume the initial conditions of the delays are equal to zero. When the first $x(0)$ sample is presented at the CIC input, $u(n)=\{x(0),0,0\}$. The first $y(n)$ output will be $x(0)$, then this output is fed back and added to zero. So the second $y(n)$ output will be $x(0)$ as well; same for the third $y(n)$. Overall, the first $x(0)$ filter input sample produces the output sequence $y(n)=\{x(0),x(0),x(0)\}$. The next sample input to the comb is $x(1)$ making $u(n)=\{x(1)-x(0),0,0\}$. The integrator delay has the value $x(0)$ stored. We add it to $x(1)-x(0)$ to get the next output $y(n)=x(1)$. The value $x(1)$ is stored in the integrator delay and is then added to zero to produce the next output $y(n)=x(1)$. Continuing in this manner, the second input sample to the CIC filter, $x(1)$, produces the output sequence $y(n)=\{x(1),x(1),x(1)\}$. This behavior repeats so that for a given CIC input sequence $x(k)$, the output $y(n)$ is a sequence where each input sample is repeated R times. This is shown in Figure 1(b) and (c) for $R=3$.

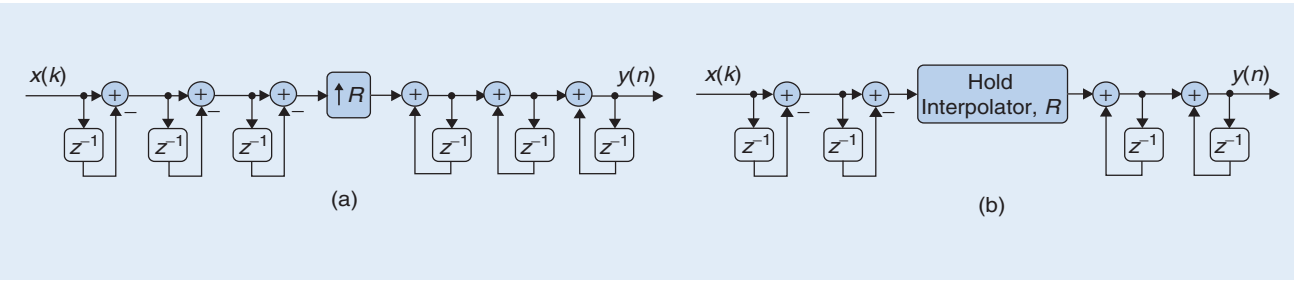


[FIG1] CIC interpolation filter: (a) structure, (b) input sequence, and (c) output sequence for $R=3$.

Naturally, when implementing a single-stage CIC filter in real hardware, it is not necessary to use the adders and delays (or the "zero-stuffer") shown in Figure 1(a). It is simply a matter of repeating each input sample $R-1$ times imposing no hardware cost.

Let us next consider a multistage CIC filter as the one shown in Figure 2(a) having three stages. At its core, there is a single-stage CIC interpolator. Our first trick, then, is to replace the innermost single-stage interpolator with a black-box, which we call a *hold interpolator* whose job is to repeat each input sample $R-1$ times as explained above. Such a reduced complexity CIC scheme is shown in Figure 2(b).

1053-5888/06/$20.00©2006IEEE



[FIG2] **Three-stage CIC interpolation filter: (a) standard structure and (b) reduced complexity structure.**

Note that in the comb sections, the number of bits required for each adder tends to increase as we move from left to right. Therefore, the adder and delay that can be removed from the comb section will typically be the ones that require the most number of bits in the entire comb section for a standard implementation of a CIC interpolator. So this trick enables us to remove the adder and delay in that section that will save us the most number of bits. However, this is not the case in the integrator section where we remove the adder and delay that would require the least number of bits of the entire section (but still as many or more than any adder or delay from the comb section).

AN EFFICIENT LINEAR INTERPOLATOR

Linear interpolators, as their name implies, interpolate samples between two adjacent samples (of the original signal to be interpolated) by placing them in an equidistant manner on the straight line that joins said two adjacent samples [1]. In general, for an interpolation factor of R, the number of multiplications required per input sample is $2R-2$. Using an example, we now present a very efficient scheme to compute those interpolated samples in a way that requires no multiplies.

The transfer function of the $R=3$ linear interpolator is given by

$$H_{\text{linear}}(z) = \frac{1}{3} + \frac{2z^{-1}}{3} + z^{-2} + \frac{2z^{-3}}{3} + \frac{z^{-4}}{3}. \quad (1)$$

Notice that this transfer function is nothing but a scaled version of a two-stage CIC interpolator. Indeed, for a two-stage CIC we have

$$\begin{aligned} H_{\text{CIC}}(z) &= \left(\frac{1-z^{-3}}{1-z^{-1}}\right)^2 \\ &= (1+z^{-1}+z^{-2})^2 \\ &= 3H_{\text{linear}}(z). \end{aligned} \quad (2)$$

Equation (2) shows that by using two-stage CIC interpolators, we can implement linear interpolation by $R=3$ without the need for the $2R-2$ multipliers. Next, we can use the hold interpolator trick, presented earlier, to simplify the linear interpolator even further.

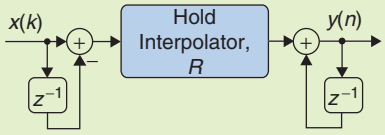
Using a hold interpolator that inserts $R-1=2$ repeated values for each input sample, we can perform efficient linear interpolation as in Figure 3. This implementation requires only two adders and two delays, no matter what the value of R. The order of this efficient linear interpolator can be increased by merely increasing the sample repetition factor R.

As with CICs, the performance of linear interpolators is not that great when used on their own. However, linear interpolators are usually not used that way. The reason is that if the interpolation factor R is high, the error introduced by assuming a straight line between two adjacent samples can be large. On the other hand, if interpolation is done in multiple sections, linear interpolation at the end (when the signal samples are already very close together) will introduce only a small error.

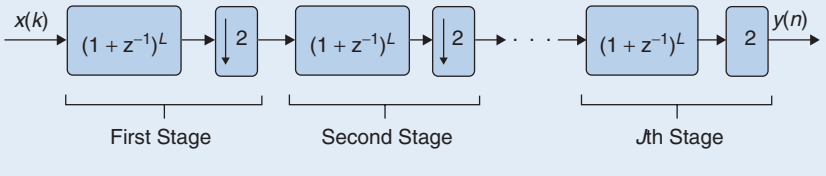
NONRECURSIVE CIC DECIMATION FILTERS

CIC filters are computationally efficient and simple to implement. However, there's trouble in paradise. One of the difficulties in using CIC filters is accommodating large data word growth, particularly when implementing integrators in multistage CIC filters. Here's a clever trick that eases the word width growth problem using nonrecursive CIC decimation filter structures, obtained by means of *polynomial factoring*. These nonrecursive structures achieve computational simplicity through *polyphase decomposition* if the sample rate reduction factor R is an integer power of two.

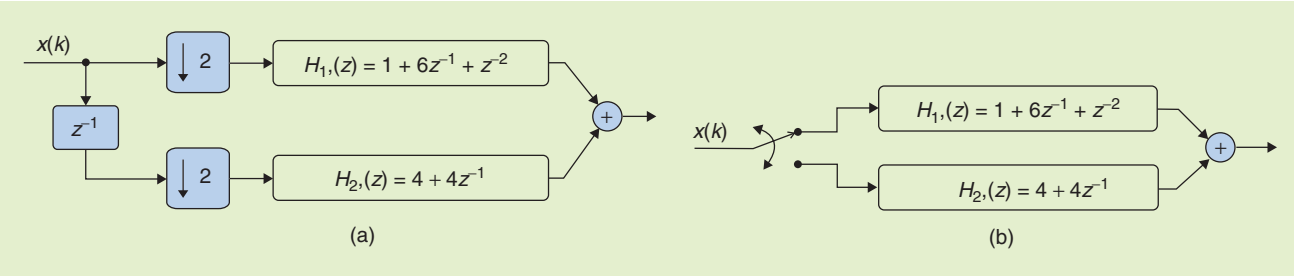
Recall that the transfer function of an Lth-order decimation CIC filter can be



[FIG3] **Multiplierless linear interpolator.**



[FIG4] **Multistage Lth-order nonrecursive CIC structure.**



[FIG5] **Polyphase structure of a single nonrecursive fifth-order CIC stage.**

expressed in either a recursive form or a nonrecursive form as given by

$$\begin{aligned} H_{\text{cic}}(z) &= \left[\frac{1-z^{-R}}{1-z^{-1}}\right]^L \\ &= \left[\sum_{n=0}^{R-1} z^{-n}\right]^L \\ &= \left(1 + z^{-1} + z^{-2} \right. \\ &\quad \left. + \cdots + z^{-R+1}\right)^L. \end{aligned} \tag{3}$$

Now if the sample rate change factor R is an integer power of two, then $R = 2^J$ where J is a positive integer, and the Lth-order nonrecursive polynomial form of $H_{\text{cic}}(z)$ in (3) can be factored as

$$\begin{aligned} H_{\text{cic}}(z) &= (1+z^{-1})^L(1+z^{-2})^L \\ &\quad \times (1+z^{-4})^L \cdots \\ &\quad \times \left(1+z^{2^{J-1}}\right)^L. \end{aligned} \tag{4}$$

The benefit of the factoring given in (4) is that the CIC decimation filter can then be implemented with J nonrecursive stages as shown for the multistage CIC filter in Figure 4. This implementation trick eliminates the integrators with their unpleasant binary word width growth. The data word widths increase by only L bits per stage, while the sampling rate is reduced by a factor of two for each stage. By the way, the cascade of nonrecursive subfilters in Figure 4 are still called CIC filters even though they have no integrators!

Lucky for us, further improvements are possible with each stage of this nonrecursive structure [2]–[4]. For example, assume $L = 4$ for the first stage in Figure 4. In that case the first stage transfer function is

$$\begin{aligned} H(z) &= (1+z^{-1})^4 \\ &= 1 + 4z^{-1} + 6z^{-2} \\ &\quad + 4z^{-3} + z^{-4} \\ &= 1 + 6z^{-2} + z^{-4} \\ &\quad + (4 + 4z^{-2})z^{-1} \\ &= H_1(z) + H_2(z)z^{-1}. \end{aligned} \tag{5}$$

The last step in (5), known as polyphase decomposition, enables a polyphase implementation having two parallel paths as shown in Figure 5(a). Because we implement decimation by two before the filtering, the new polyphase components are $H_{1'}(z) = 1 + 6z^{-1} + z^{-2}$, and $H_{2'}(z) = 4 + 4z^{-1}$ implemented at half the data rate into the stage. (Reducing data rates as early as possible is a key design goal in the implementation of CIC decimation filters.) The initial delay element and the dual decimation by two operations can implemented by routing the odd-index input samples to $H_{1'}(z)$ and the even-index samples to $H_{2'}(z)$, as shown in Figure 5(b). Of course the $H_{1'}(z)$ and $H_{2'}(z)$ polyphase components are implemented in a tapped-delay line fashion. Methods exist to eliminate the multipliers in the Figure 5 tapped-delay line implementation [2]–[4].

The nonrecursive CIC decimation filters described above have the restriction that the R decimation factor must be an integer power of two. That constraint is loosened due to a clever scheme assuming R can be factored into the product of prime numbers. Details of that process, called prime factorization, are available in [2] and [5].

AUTHORS

Ricardo A. Losada (Ric.Losada@mathworks.com) is the product lead for the Filter Design Toolbox at The MathWorks Inc. where he has worked for over eight years. He received an electronics engineering degree from the Universidad Distrital Francisco José de Caldas in Bogotá, Colombia, and an M.Sc. degree from Rutgers University, New Jersey. He is a Member of the IEEE.

Richard Lyons (r.lyons@ieee.org) is a consulting systems engineer and lecturer with Besser Associates in Mountain View, California. He is the author of *Understanding Digital Signal Processing 2/E* (Prentice-Hall, 2004) and an associate editor for *IEEE Signal Processing Magazine*.

REFERENCES

[1] S.J. Orfanidis, *Introduction to Signal Processing*. Upper Saddle River, NJ: Prentice Hall, 1996.

[2] R. Lyons, *Understanding Digital Signal Processing*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2004.

[3] L. Ascari et al., "Low power implementation of a sigma delta decimation filter for cardiac applications," in *Proc. IEEE Instrumentation and Measurement Technology Conf.*, Budapest Hungary, May, 21–23, 2001, pp. 750–755.

[4] Y. Gao et al, "Low-power implementation of a fifth-order comb decimation filter for multi-standard transceiver applications," in *Proc. Int. Conf. Signal Proc. Applications and Technology (ICSPAT)*, Orlando, FL, Oct. 1999. [Online]. Available: http://www.pcc.lth.se/events/workshops/1999/posters/Gao.pdf

[5] Y. Jang and S. Yang, "Non-recursive cascaded integrator-comb decimation filters with integer multiple factors," in *Proc. 44th IEEE Midwest Symp. Circuits and Systems (MWSCAS)*, Dayton, OH, Aug. 2001, pp. 130–133. SP