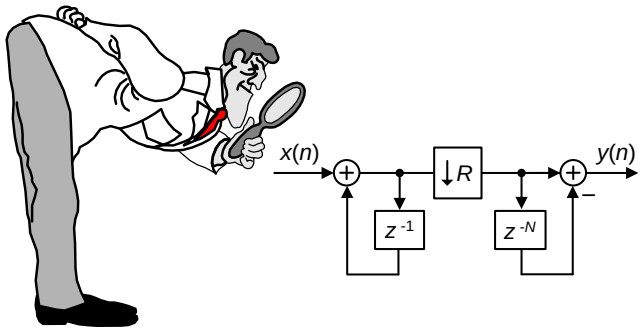
A Beginner's Guide To Cascaded Integrator-Comb (CIC) Filters

by Richard (Rick) Lyons



Cascaded integrator-comb (CIC) digital filters are computationally-efficient implementations of narrowband lowpass filters, and are often embedded in hardware implementations of decimation, interpolation, and delta-sigma converter filtering.

After describing a few applications of CIC filters, this document introduces their structure and behavior, presents the frequency-domain performance of CIC filters, and discusses several important practical issues in implementing these filters.

CIC Filter Applications

CIC filters are well-suited for anti-aliasing filtering prior to decimation (sample rate reduction), as shown in Figure 1(a); and for anti-imaging filtering for interpolated signals (sample rate increase) as in Figure 1(b). Both applications are associated with very high-data rate filtering such as hardware quadrature modulation and demodulation in modern wireless systems, and delta-sigma A/D and D/A converters.

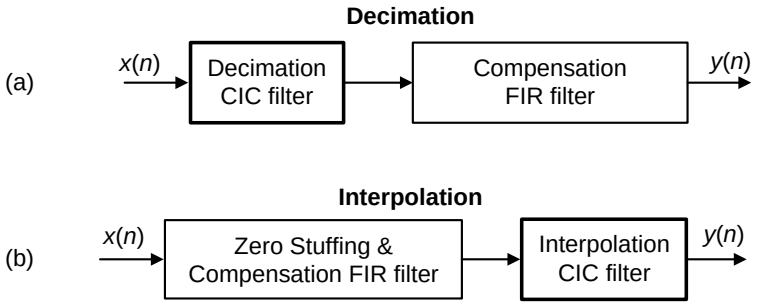


Figure 1: CIC filter applications: (a) for decimation; (b) for interpolation.

Because their frequency magnitude responses are $\sin(x)/x$-like, CIC filters are typically followed, or preceded, by linear-phase lowpass tapped-delay line finite impulse response (FIR) filters whose tasks are to compensate for the CIC filter's non-flat passband.

The Figure 1 cascaded-filter architectures have valuable benefits. For example:

- The FIR filters operate at reduced clock rates minimizing power consumption in high-speed hardware applications.
- CIC filters are popular in hardware devices; they need no filter coefficient data storage and require no multiplications. The arithmetic needed to implement CIC filters is strictly additions and subtractions only.
- narrowband lowpass filtering can be attained at a greatly reduced

computational complexity compared to using a single lowpass FIR filter. And this property is why CIC filters are so attractive in decimating and interpolating DSP systems.

With that said, let's see how CIC filters operate.

Recursive Running Sum Filter

CIC filters originate from the notion of a *recursive running sum* filter, which is itself an efficient form of a nonrecursive *moving averager*. Recall the standard D-point moving-average process in Figure 2(a). There we see that $D-1$ summations (plus one multiply by $1/D$) are necessary to compute the averager's $y(n)$ output sequence.

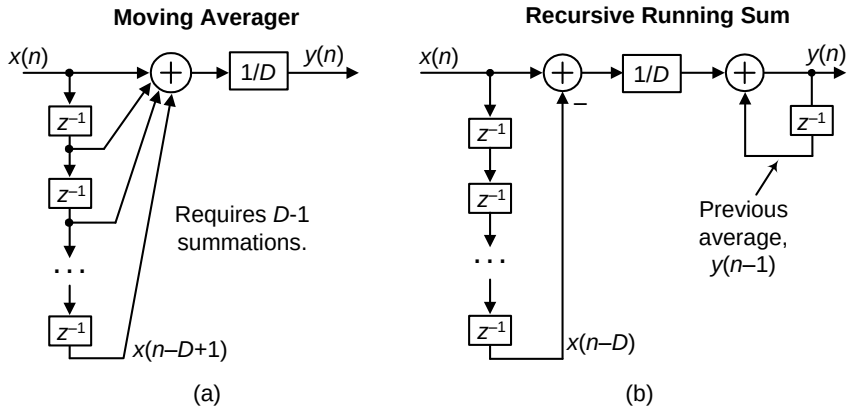


Figure 2: *D*-point moving-average and recursive running sum filters.

The D-point moving-average filter's time-domain output is expressed as

$$y(n) = \frac{1}{D}[x(n) + x(n-1) + x(n-2) + x(n-3) + \ldots + x(n-D+1)] \qquad (1)$$

where n is our time-domain index. The z-domain expression for this moving averager's output is

$$Y(z) = \frac{1}{D}[X(z) + X(z)z^{-1} + X(z)z^{-2} + \ldots + X(z)z^{-D+1}] \qquad (2)$$

while its z-domain $H_{MA}(z)$ transfer function is

$$H_{MA}(z) = \frac{Y(z)}{X(z)} = \frac{1}{D}[1 + z^{-1} + z^{-2} + \ldots + z^{-D+1}] = \frac{1}{D}\sum_{n=0}^{D-1} z^{-n}. \qquad (3)$$

We provide these equations not to make things complicated, but because they're useful. Equation (1) tells us how to build a moving averager, and Equation (3) is in the form used by commercial signal processing software to model the frequency-domain behavior of the moving averager.

The next step in our journey toward understanding CIC filters is to consider an equivalent form of the moving averager, the *recursive running sum* filter depicted in Figure 2(b). Ignoring the $1/D$ scaling, there we see that the current input sample $x(n)$ is added to, and the oldest input sample $x(n-D)$ is subtracted from, the previous output average $y(n-1)$. It's called "recursive" because it has feedback. Each filter output sample is retained and used to compute the next output value. The recursive running sum filter's difference equation is

$$y(n) = \frac{1}{D}[x(n) - x(n-D)] + y(n-1) \quad (4)$$

having a z-domain $H_{RRS}(z)$ transfer function of

$$H_{RRS}(z) = \frac{1}{D} \cdot \frac{1 - z^{-D}}{1 - z^{-1}}. \quad (5)$$

We use the same $H(z)$ variable for the transfer functions of the moving average filter and the recursive running sum filter because their transfer functions are equal to each other! It's true. Equation (3) is the nonrecursive expression, and Equation (5) is the recursive expression, for a D-point averager. The mathematical proof of this can be found in Appendix B of Reference [1], but shortly I'll demonstrate that equivalency with in example.

Here is why we care about recursive running sum filters: the standard moving averager in Figure 2(a) must perform $D-1$ additions per output sample. The recursive running sum filter has the sweet advantage that only one addition and one subtraction are required per output sample, regardless of the delay length D! This computational efficiency makes the recursive running sum filter attractive in many applications seeking noise reduction through averaging. Next we'll see how a CIC filter is, itself, a recursive running sum filter.

CIC Filter Structures

If we condense the delay line representation and ignore the $1/D$ scaling in Figure 2(b) we obtain the classic form of a 1st-order CIC filter, whose cascade structure is shown in Figure 3.

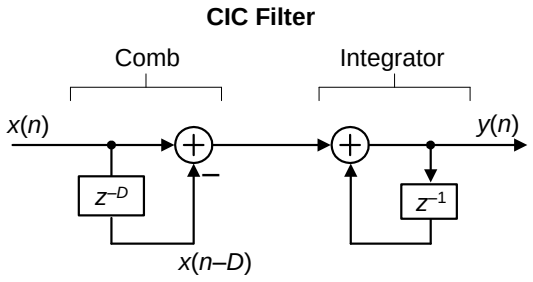


Figure 3: *D*-point recursive running sum filter in a cascaded integrator-comb implementation.

The feedforward portion of the CIC filter is called the *comb* section, whose *differential delay* is D, while the feedback section is typically called an *integrator*. The comb stage subtracts a delayed input sample from the current input sample, and the integrator is simply an accumulator. The CIC filter's time-domain difference equation is

$$y(n) = x(n) - x(n-D) + y(n-1) \quad (6)$$

and its z-domain transfer function is

$$H_{CIC}(z) = \frac{Y(z)}{X(z)} = \frac{1 - z^{-D}}{1 - z^{-1}}. \quad (7)$$

To see why the CIC filter is of interest, first we examine its time-domain behavior, for $D = 5$, shown in Figure 4. If a unit impulse sequence, a unity-valued sample followed by many zero-valued samples, was applied to the comb stage, that stage's output is as shown in Figure 4(a). Think, now, what would be the output of

the integrator if its input was the comb stage's impulse response. The initial positive impulse from the comb filter starts the integrator's all-ones output. Then, D samples later the negative impulse from the comb stage arrives at the integrator to zero all further CIC filter output samples.

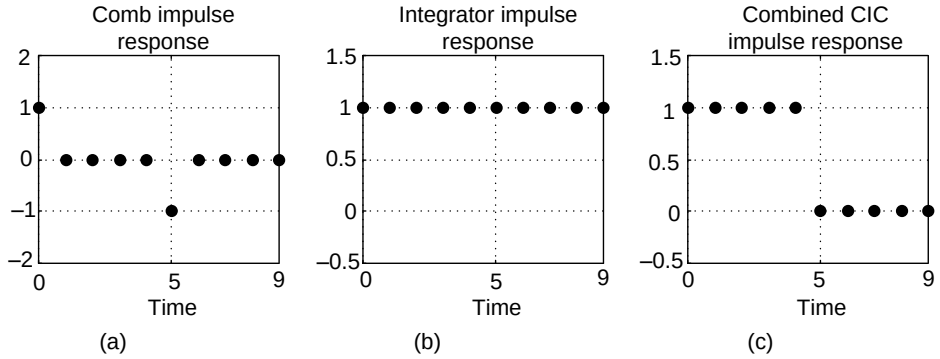


Figure 4: Single-stage CIC filter time-domain responses when D = 5.

The key issue is that the combined unit impulse response of the CIC filter, being a rectangular sequence, is identical to the unit impulse responses of a moving average filter and the recursive running sum filter. (Moving averagers, recursive running sum filters, and CIC filters are close kin. They have the same z-domain pole/zero locations, their frequency magnitude responses have identical shapes, their phase responses are identical, and their transfer functions differ only by a constant scale factor.) If you understand the time-domain behavior of a moving averager, then you now understand the time-domain behavior of the CIC filter in Figure 3.

The frequency magnitude and linear-phase response of a D = 5 CIC filter is shown in Figures 5(a) and 5(b) where the frequency f_s is the input signal sample rate measured in Hz.

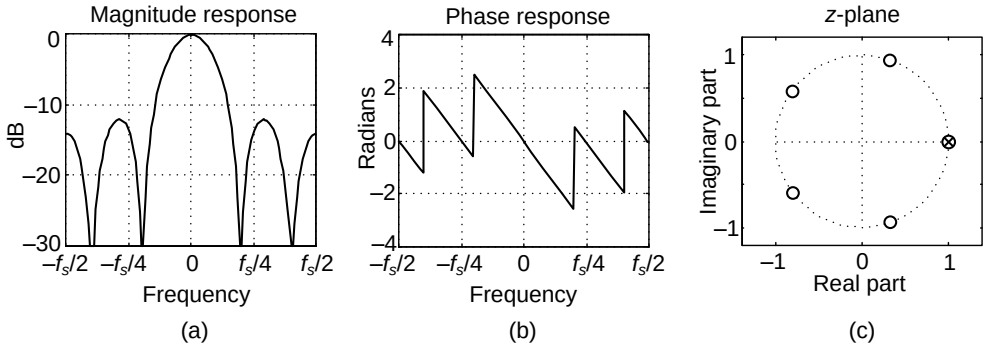


Figure 5: Characteristics of a single-stage CIC filter when D = 5.

The CIC filter's frequency response, derived in Reference [1], is:

$$H_{CIC}(e^{j2\pi f}) = e^{-j2\pi f(D-1)/2} \cdot \frac{\sin(2\pi fD/2)}{\sin(2\pi f/2)} . \quad (8)$$

If we ignore the phase factor in Eq. (8), that ratio of sin() terms can be approximated by a $\sin(x)/x$ function. This means the CIC filter's frequency magnitude response is approximately equal to a $\sin(x)/x$ function centered at 0 Hz as we see in Figure 5(a). (This is why CIC filters are sometimes called *sinc* filters.)

Digital filter designers like to see z-plane pole/zero plots, so I provide the z-plane characteristics of a $D = 5$ CIC filter in Figure 5(C), where the comb filter produces D zeros, equally spaced around the unit-circle, and the integrator produces a single pole canceling the zero at $z = 1$. Each of the comb's zeros, being a Dth root of 1, are located at $z(m) = e^{j2\pi m/D}$, where $m = 0, 1, 2, \ldots, D-1$, corresponding to a magnitude null in Figure 5(a).

The normally risky situation of having a filter pole directly on the z-plane's unit circle need not trouble us here because there can be no coefficient quantization error in our $H_{cic}(z)$ transfer function. That's because CIC filter coefficients are ones and can be represented with perfect precision with fixed-point number formats. The filter pole will *never* be outside the unit circle. Although recursive, happily CIC filters are guaranteed-stable, linear-phase, and have finite-length impulse responses.

Again, CIC filters are primarily used for anti-aliasing filtering prior to decimation, and for anti-imaging filtering for interpolated signals. With those notions in mind we swap the order of Figure 2(C)'s comb and integrator——we're permitted to do so because those operations are linear——and include decimation by an integer sample rate change factor R in Figure 6(a). (The reader should prove to themselves that the unit impulse response of the integrator/comb combination, prior to the sample rate change, in Figure 6(a) is equal to that in Figure 4(C).) In most CIC filter applications the rate change R is equal to the comb's differential delay D, but I'll keep them as separate design parameters for now.

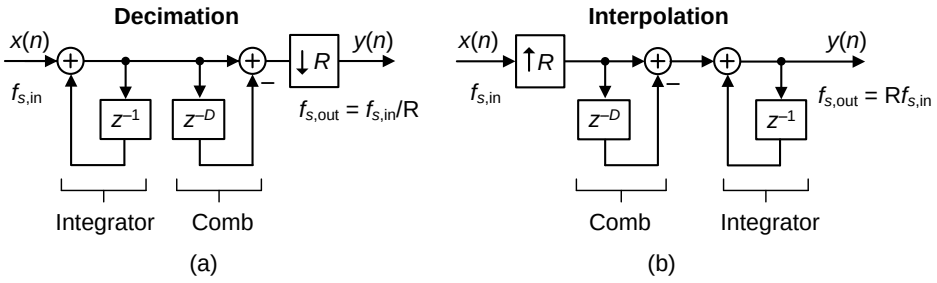


Figure 6: Single-stage CIC filters used in: (a) decimation; and (b) interpolation. (Sample rates $f_{s,in}$ and $f_{s,out}$ are the sample rates of the $x(n)$ and $y(n)$ sequences respectively.)

The decimation (also called "down-sampling") operation $\downarrow R$ means discard all but every Rth sample, resulting in an output sample rate of $f_{s,out} = f_{s,in}/R$. To investigate a CIC filter's frequency-domain behavior in more detail, Figure 7(a) shows the frequency magnitude response of a $D = 8$ CIC filter prior to decimation. The spectral band, of width B Hz, centered at 0 Hz is the desired passband of the filter. A key aspect of CIC filters is the spectral folding that takes place due to decimation.

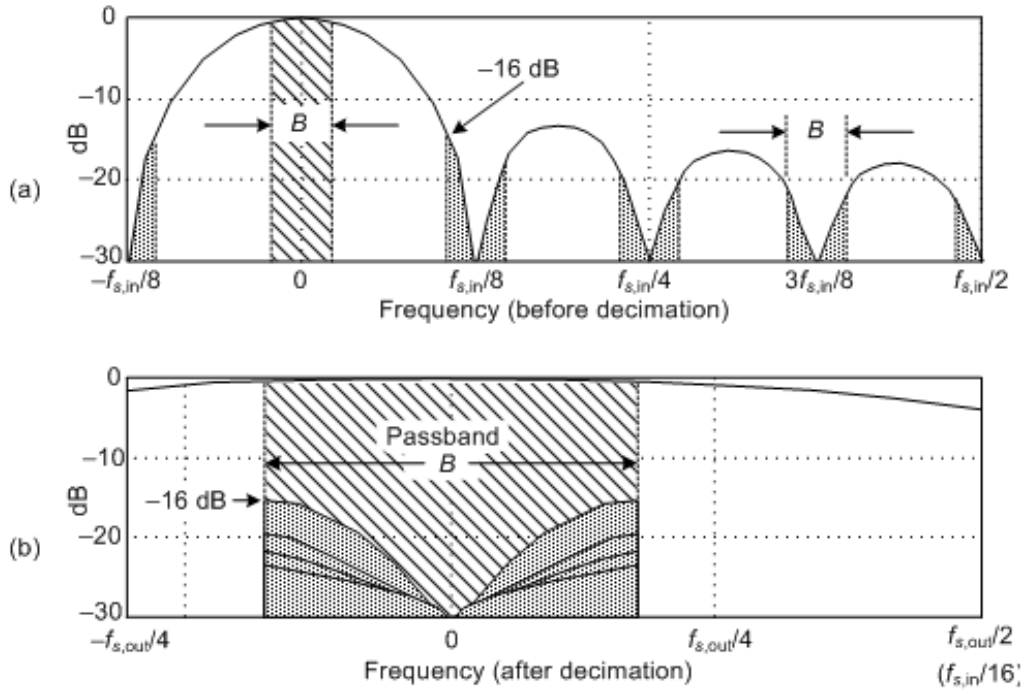


Figure 7: Magnitude response of a 1st-order, D = 8, decimating CIC filter: (a) before decimation; (b) aliasing after R = 8 decimation.

Those B-width shaded spectral bands centered about multiples of $f_{s,in}/R$ in Figure 7(a) will alias directly into our desired passband after decimation by R = 8 as shown in Figure 7(b). Notice how the largest *aliased* spectral component, in this 1st-order CIC filter example, is 16 dB below the peak of the band of interest. Of course the aliased power levels depend on the bandwidth B; the smaller B the lower the aliased energy after decimation.

Figure 6(b) shows a CIC filter used for interpolation where the $\uparrow R$ symbol means insert R−1 zeros between each $x(n)$ sample (up-sampling), yielding a $y(n)$output sample rate of $f_{s,out} = Rf_{s,in}$. (In this CIC filter discussion, *interpolation* is defined as zeros-insertion, called "zero stuffing", followed by CIC filter lowpass filtering.)

To illustrate the spectral effects of interpolation by R = 8, Figure 8(a) shows an arbitrary baseband spectrum, with its spectral replications, of a signal applied to the $D = R$ = 8 interpolating CIC filter of Figure 6(b). The interpolation filter's output spectrum in Figure 8(b) shows how imperfect CIC filter lowpass filtering gives rise to the undesired spectral images.

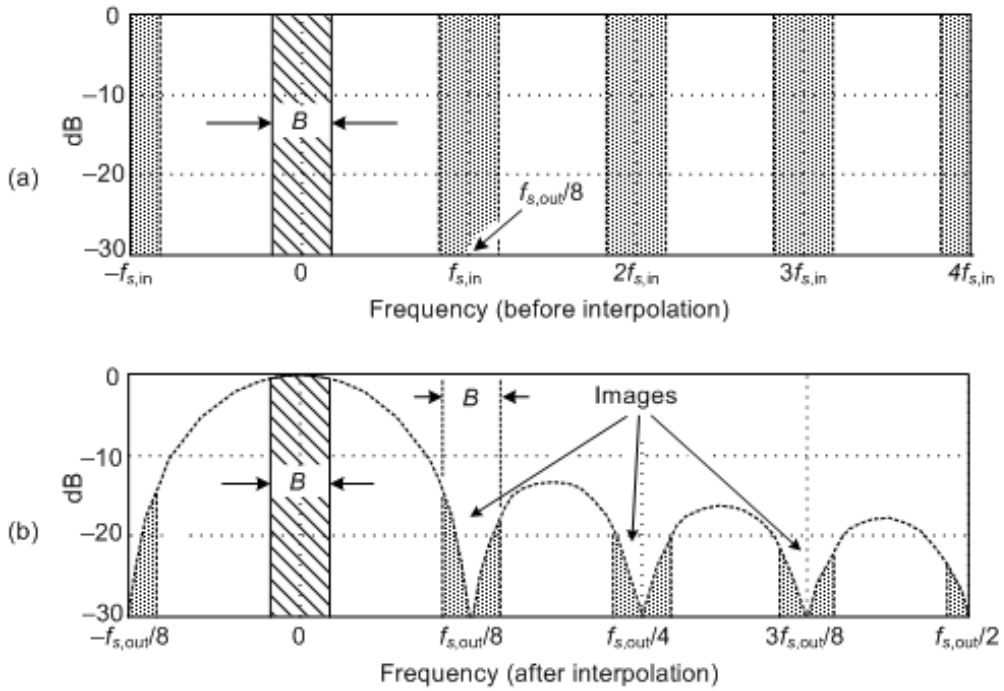


Figure 8: 1st-order, $D = R = 8$, interpolating CIC filter spectra: (a) input signal spectrum; (b) spectral images of a zero stuffed and CIC lowpass filtered output.

After interpolation, Figure 8(b) shows the unwanted images of the B-width baseband spectrum reside at the null centers, located at integer multiples of $f_{s,out}/R$. If we follow the CIC filter with a traditional lowpass tapped-delay line FIR filter, whose stopband includes the first image band, fairly high image rejection can be achieved.

Improving CIC Filter Attenuation

The most common method to improve CIC filter anti-aliasing and image-reject attenuation is by cascading multiple CIC filters. Figure 9 shows the block diagram and frequency magnitude response, up to the $w(n)$ node, of a 3rd-order ($M = 3$) CIC decimating filter.

Notice the increased attenuation at $f_{s,in}/R$ in Figure 9(b) compared to the 1st-order CIC filter in Figure 7(a). Because the $M = 3$ CIC stages are in cascade, their combined z-domain transfer function will be the product of their individual transfer functions, or

$$H_{\text{CIC},M\text{th-order}}(z) = \frac{W(z)}{X(z)} = \left(\frac{1 - z^{-D}}{1 - z^{-1}}\right)^{M}. \qquad (9a)$$

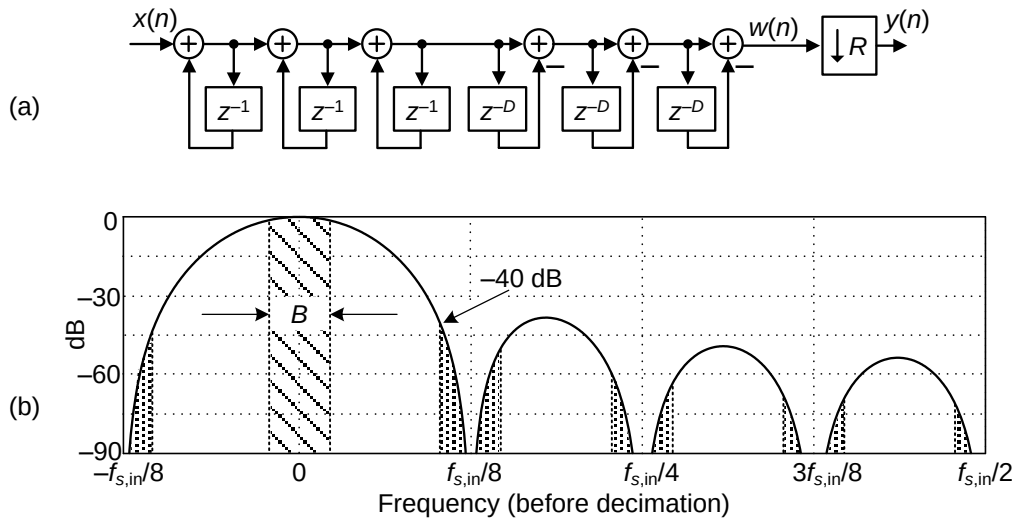


Figure 9: 3rd-order (*M* = 3) CIC decimation filter structure: (a) block diagram; (b) and frequency magnitude response before down-sampling when *D* = *R* = 8.

It's important to notice the $W(z)$ term in Eq. (9a). The transfer function in Eq. (9a) does *not* include the down-sampling by R operation of the $w(n)$ sequence in Figure 9(a). (The entire system in Figure 9(a) is a *multirate* system, and multirate systems do not have z-domain transfer functions. See Reference [2] for more information on this subject.)

The frequency magnitude response of the Figure 9(b) filter, from the $x(n)$ input to the $w(n)$ sequence, is:

$$\left|H_{\text{CIC},M\text{th-order}}(e^{j2\pi f})\right| = \left|\frac{\sin(2\pi fD/2)}{\sin(2\pi f/2)}\right|^M . \qquad (9b)$$

The price we pay for improved anti-alias attenuation is additional hardware adders and increased CIC filter passband droop (downward mainlobe curvature). An additional penalty of increased filter order comes from the gain of the filter, which is exponential with the order M. Because CIC filters generally must work with full precision to remain stable, the number of bits in the adders is $M\log_2(D)$, so there is a large data word-width penalty for higher order filters. Even so, because CIC filters drastically simplify narrowband lowpass filtering operations, multistage implementations ($M > 1$) are common in commercial integrated circuits where an Mth-order CIC filter is often called a $sinc^M$ filter.

Building a CIC Filter

In CIC filters, the comb section can precede, or follow, the integrator section. However it's sensible to put the comb section on the side of the filter operating at the lower sample rate. Swapping the Figure 6 comb filters with the down-sampling operations results in the most common implementation of CIC filters as shown in Figure 10. This was one of the key features of CIC filters introduced to the world by Hogenauer in 1981 [See Reference 3].

In Figure 10 notice the decimation filter's comb section now has a reduced delay length (differential delay) of $N = D/R$. That's because an N-sample comb delay after down-sampling by R is equivalent to a D-sample comb delay before down-sampling by R. Likewise for the interpolation filter; an N-sample comb delay before up-sampling by R is equivalent to a D-sample comb delay after up-sampling by R.

Those Figure 10 configurations yield two major benefits: first, the comb sections' new differential delay lines are decreased in length to $N = D/R$ reducing data storage requirements; second, the comb section now operates at a reduced clock rate. Both of these effects reduce hardware power consumption.

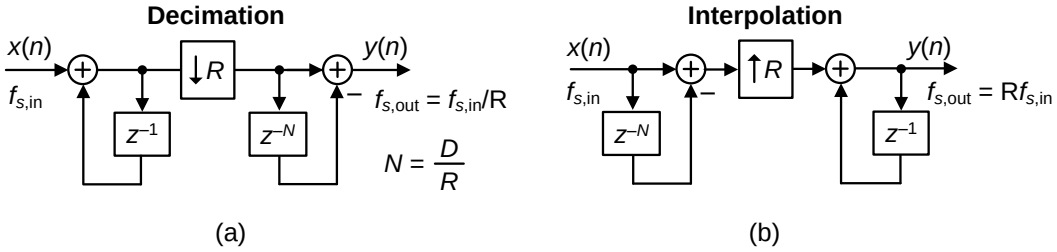


Figure 10: Reduced comb delay, single-stage, CIC filter implementations: (a) for decimation; (b) for interpolation.

The comb section's differential delay design parameter N is typically 1 or 2 for high sample rate ratios as is often used in commercial radio frequency up/down-converters. Variable N effectively sets the number of nulls in the frequency response of a decimation filter, as shown in Figure 11(a).

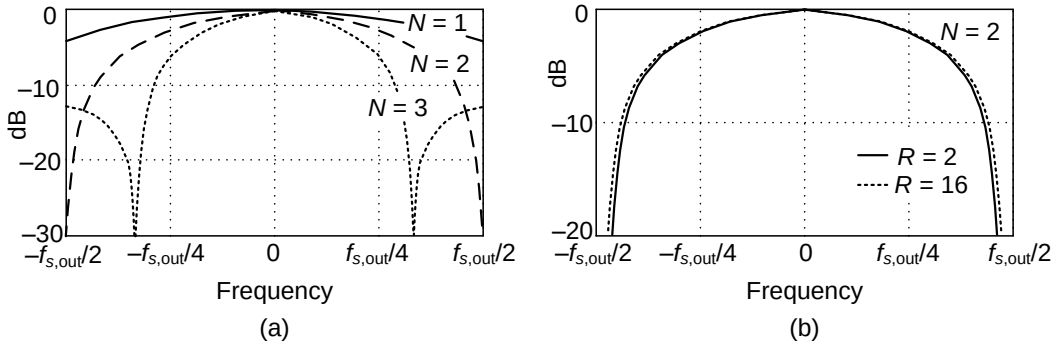


Figure 11: First-order CIC decimation filter responses: (a) for various values of differential delay N, when $R = 8$; (b) for two decimation factors when $N = 2$. (Sample rate $f_{s,out}$ is the sample rate of the decimated output sequence.)

An important characteristic of a fixed-order CIC decimator is the shape of the filter's mainlobe low frequency magnitude response changes very little, as shown in Figure 11(b), as a function of the decimation ratio. For R larger than roughly 16, the change in the filter's low frequency magnitude shape is negligible. This allows the same compensation FIR filter to be used for variable-decimation ratio systems.

CIC Filter Gain

The passband gain of a 1st-order CIC decimation filter, derived in Reference [1], is equal to the comb filter delay $D = NR$ at zero Hz (DC). Multistage Mth-order CIC decimation filters, such as in Figure 9(a), have a net gain of $(NR)^M$. The high gain of an Mth-order CIC decimation filter can conveniently be made equal to one by binary right-shifting the filter's output samples by $\log_2((NR)^M)$ bits when $(NR)^M$ is restricted to be an integer power of two.

An interpolating CIC filter has zeros inserted between its input samples reducing its gain by a factor of $1/R$ to account for the zero-valued samples, so the net gain of an interpolating CIC filter is $(NR)^M/R$.

Register Word Widths and Arithmetic Overflow

CIC filters are computationally efficient (i.e., fast). But to be mathematically accurate they must be implemented with appropriate register bit widths. Let's now consider their register bit widths in order to follow the advice of a legendary lawman. ["Fast is important, but accuracy is everything." Wyatt Earp (1848-1929)]

CIC filters suffer from accumulator (adder) arithmetic register overflow because of the unity feedback at each integrator stage. This overflow is of no consequence as long as the following two conditions are met:

- each stage is implemented with two's complement (non-saturating) arithmetic [1,3], and
- the range of a stage's number system is greater than or equal to the maximum value expected at the stage's output.

When two's complement fixed-point arithmetic is used, the number of bits in an Mth-order CIC decimation filter's integrator and comb registers must accommodate the filter's input signal's maximum amplitude times the filter's total gain of $(NR)^M$. To be specific, overflow errors are avoided if the number of integrator and comb register bit widths is at least

$$\text{register bit widths} = x(n) \text{ bits} + \lceil M \cdot \log_2(NR) \rceil \tag{10}$$

where $x(n)$ is the input to the CIC filter, and $\lceil k \rceil$ means if k is not an integer, round it up to the next larger integer.

Using Figure 12 as an example, if an $M = 3$-stage CIC decimation filter accepts 12-bit binary input words and the decimation factor is $R = D = 8$, binary overflow errors are avoided if the three integrator and three comb registers' bit widths are no less than

$$\text{register bit widths} = 12 + \lceil 3 \cdot \log_2(8) \rceil = 12 + 3 \cdot 3 = 21 \text{ bits.} \tag{11}$$

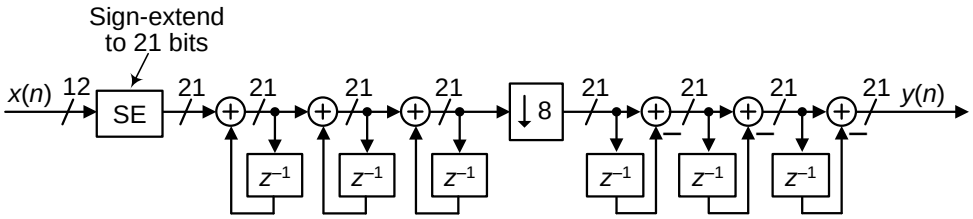


Figure 12: Filter accumulator register word widths for an $M = 3$, $D = R = 8$, decimating CIC filter when the input data word width is 12 bits.

In some CIC filtering applications there is the opportunity to discard some of the least significant bits (LSBs) within the accumulator (adder) stages of an Mth-order CIC decimation filter, at the expense of added noise at the filter's output. The specific effects of this LSB removal (called "register pruning") are, however, a complicated issue so I refer the reader to References [3-5] for more details.

Compensation/Preconditioning FIR Filters

In typical decimation/interpolation filtering applications we desire reasonably flat passband gain and narrow transition region width performance. These desirable properties are not provided by CIC filters alone, with their drooping mainlobe gains and wide transition regions. We can alleviate this problem, in decimation for example, by following the CIC filter with a compensation FIR filter, as in Figure

1(a), to sharpen the CIC filter's frequency rolloff and flatten the CIC filter's passband gain.

The compensation FIR filter's frequency magnitude response, ideally an inverted version of the CIC filter passband magnitude response, is similar to that shown by the long-dashed curve in Figure 13(a). That curve represents a simple three-tap FIR filter whose coefficients are [-1/16, 9/8, -1/16] as suggested by Reference [6]. With the short-dash curve representing the uncompensated passband droop of a 1st-order $D = R = 8$ CIC filter, the solid curve represents the compensated magnitude response of the cascaded filters.

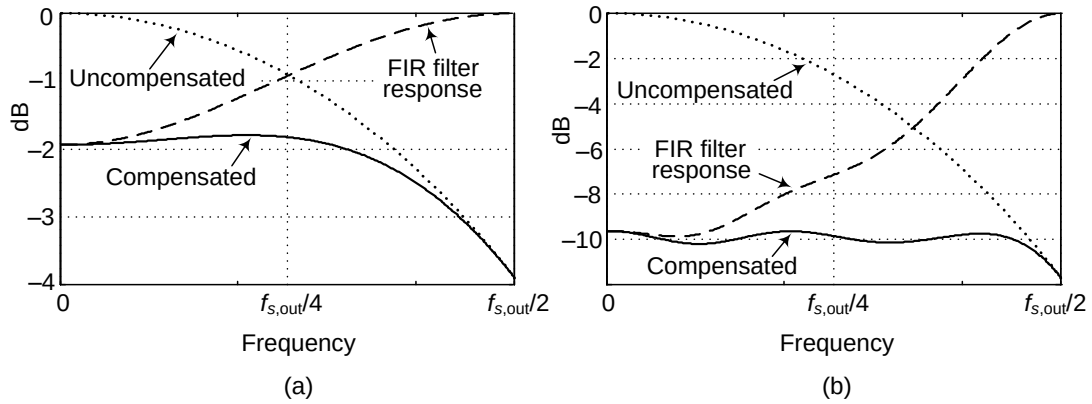


Figure 13: Compensation FIR filter responses: (a) with a 1st-order (M = 1) decimation CIC filter; (b) with a 3rd-order (M = 3) decimation CIC filter.

If either the passband bandwidth or CIC filter order increases the correction becomes more robust, requiring more compensation FIR filter taps. An example of this situation is shown in Figure 13(b) where the dotted curve represents the passband droop of a 3rd-order $D = R = 8$ CIC filter and the dashed curve, taking the form of $[x/\sin(x)]^3$, is the response of a 15-tap linear-phase compensation FIR filter having the coefficients [-1, 4, -16, 32, -64, 136, -352, 1312, -352, 136, -64, 32, -16, 4, -1].

Those dashed curves in Figure 13 represent the frequency magnitude responses of compensating FIR filters within which no sample rate change takes place. (The FIR filters' input and output sample rates are equal to the $f_{s,out}$ output rate of the decimating CIC filter.)

For completeness I'll mention that Reference [6] also presents a low-order IIR compensation filter for decimation applications where linear-phase filtering is not necessary. That simple IIR filter's z-domain transfer function is:

$$H_{IIR}(z) = \frac{9/8}{1 - (1/8)z^{-1}} . \qquad (12)$$

A Simple and Efficient FIR Compensation Filter

I recently reviewed a copy of Reference [7] that recommends the simple and computationally-efficient decimation FIR compensation filter shown in Figure 14(a). The three-tap FIR compensation filter has only one non-unity coefficient. The coefficient's value depends on the order, M, of the CIC decimation filter being compensated, as shown in the table in Figure 14(a).

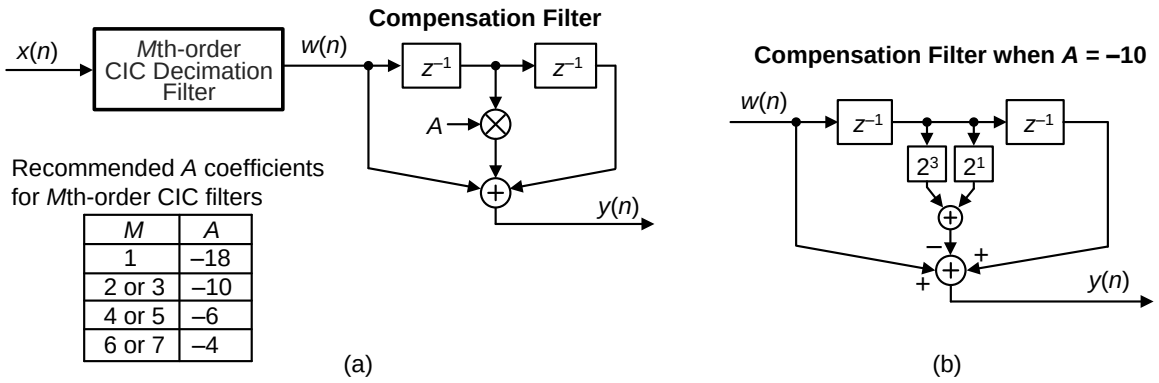


Figure 14: Decimation compensation filters: (a) three-tap FIR compensation filter; (b) multiplier-free compensation filter when *A* = -10.

This compensation filter deserves consideration because it's linear-phase and was designed to be implemented without multiplication. Figure 14(b) shows how to use binary bit left-shifting and one addition to perform a multiplication by $A = -10$.

A Decimation-By-Two FIR Compensation Filter

I've seen decimation applications where a compensating FIR filter is designed to enable an additional, and final, output decimation by two operation. That is, a CIC filter is followed by a compensation filter whose output is decimated by a factor of two.

Such a compensation filter's frequency magnitude response looks similar to that in Figure 15, where $f_{s,in}$ is the compensation filter's input sample rate. Again, the decimate-by-two compensation FIR filter's frequency magnitude response is ideally an inverted version of a CIC filter's passband magnitude response from zero Hz up to approximately $f_{s,in}/4$. In such applications the transition region width of the cascaded CIC and compensation filter is determined by the transition region width of the compensation filter.

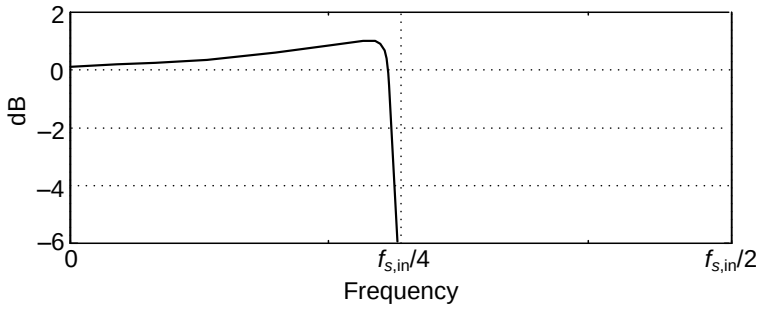


Figure 15: Frequency magnitude response of a decimate-by-2 compensation FIR filter.

Appendix A of this document presents MATLAB code for designing decimating FIR compensation filters similar to that shown in Figure 15.

Conclusions

Here's the bottom line of our CIC filter presentation: a decimating CIC filter is merely a very efficient recursive implementation of a moving average filter, with NR taps, whose output is decimated by R. Likewise, an interpolating CIC filter is insertion of $R-1$ zero samples between each input sample followed by an NR tap moving average filter running at the output sample rate $f_{s,out}$. The cascade implementations in Figure 1 result in total computational workloads far less than using a single FIR filter alone for high sample rate change decimation and

interpolation. CIC filters are designed to maximize the amount of low sample rate processing to minimize power consumption in high-speed hardware applications.

Again, CIC filters require no multiplications, their arithmetic is strictly additions and subtractions. Their performance allows us to state that, technically speaking, CIC filters are lean mean fat-free filtering machines. In closing, be informed that there are ways to build nonrecursive CIC filters, and this eases the word width growth problem of the above traditional recursive CIC filters. Those advanced CIC filter architectures are discussed in Reference [1].

In case you're interested, the history of CIC filters is presented in Reference [8].

References

[1] Lyons, R., *Understanding Digital Signal Processing, 2nd Ed.*, Prentice Hall, Upper Saddle River, New Jersey, 2004, pp. 550-566.

[2] Lyons, R., "Do Multirate Systems Have Transfer Functions?". Available online at: **https://www.dsprelated.com/showarticle/143.php**

[3] Hogenauer, E. "An Economical Class of Digital Filters For Decimation and Interpolation," *IEEE Trans. Acoust. Speech and Signal Proc.*, Vol. ASSP-29, pp. 155-162, April 1981. Available online at: http://read.pudn.com/downloads163/ebook/744947/123.pdf

[4] harris, f., *Multirate Signal Processing for Communication Systems*, Prentice-Hall, Upper Saddle River, New Jersey, 2004, pp. 341-358.

[5] Lyons, R., "Computing CIC Filter Register Pruning Using Matlab". Available online at: https://www.dsprelated.com/showcode/269.php

[6] Mitra, S., *Digital Signal Processing, A computer-Based Approach*, McGraw-Hill, New York, New York, 2011, pp. 885.

[7] Jovanovic Dolecek, G., and harris, f., "Design of Wideband CIC Compensation Filter for a Digital IF Receiver" *Digital Signal Processing*, (Elsevier) Vol. 19, No. 5, pp. 827-837, Sept, 2009.

[8] Lyons, R., "The History of CIC Filters: The Untold Story". Available online at: https://www.dsprelated.com/showarticle/160.php

Appendix A

Below is MATLAB code used to design decimating compensation filters similar to that shown in Figure 15.

```
%  Filename: CIC_compensation_filter.m
%
% Uses MATLAB's 'fir2()' command to
% design an FIR CIC-compensation filter by applying the
% inverse of a "final passband width" portion of a pre-defined
% CIC filter's mainlobe magnitude response as an input vector of
% magnitude values to Matlab's 'fir2()' command.
%
% Assumes the CIC filter is followed by a binary
```

```
% right-shift operation to reduce the CIC filter's
% DC (zero Hz) gain from R*N^M to one (unity gain).
% The values 'R', 'N', and 'M' are defined below.
%
% Note: The FIR Compensation filter's passband cutoff
% frequency (Fp), passed to Matlab's fir2() command, is
% the frequency where the FIR compensation filter's
% magnitude response is -6 dB.
%
% The final FIR Compensation Filter's coefficients are 'FIR_Coeffs'.
%
% The final cascaded CIC/Comp. Filter combination's
% frequency magnitude response is shown in Figure 4.
%
%    [Richard (Rick) Lyons, March, 2020]

clear, clc

% Set the order of the FIR compensation filter
FIR_Order = 31;  % One less than number of FIR coefficients

% Number of positive-freq points used for plotting spectra.
Num_Freq_Points = 512; % Also try, 256 or 1024

% Set minimum  mag. level (in dB) in spectral mag. plots
Threshold = -80;    % Minimum dB plot level

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define parameters of your desired CIC filter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
M = 3;         % Number of cascaded CIC stages.
R = 10;        % Decimation sample rate-change factor.
N = 1;         % Differential delay after sample rate change.
Fs = 1000;     % CIC input sample rate before decimation (Hz).
Fp = 25;       % Final cascaded-filters' passband cutoff freq
                % (-6 dB mag. point) after decimation (Hz).
disp(' ')
disp(['CIC filter order = ', num2str(M), ' stages'])
disp(['CIC filter input sample rate = ', num2str(Fs), ' Hz'])
disp(['CIC decimation factor R = ', num2str(R)])
disp(['CIC filter output sample rate = ', num2str(Fs/R), ' Hz'])
disp(['Final cascaded-filter -6 dB passband width = ', num2str(Fp), '
Hz'])
disp(['FIR filter number of taps = ', num2str(FIR_Order+1), ' taps'])

    % Just for fun, estimate number of taps for a traditional
    %  tapped-delay line (non-CIC) "Parks-McClellan-designed"
    %  lowpass FIR filter.
    [Taps, W, beta, ftype] = firpmord...
                    ([Fp, 1.3*Fp], [1,0], [0.05, 0.01], Fs);
    disp(['[Equivalent tapped-delay line FIR lowpass filter' ...
        ' requires approximately ', num2str(Taps), ' taps]']), disp(' ')

    % Check to see that passband width, Fp, is not too large
    if Fp >= 0.5*Fs/R % Half the final output sample rate
        beep,  pause(0.5), beep, disp(' ')
        disp('WARNING !'), disp('WARNING !')
        disp(['Fp = ',num2str(Fp),' Hz passband width is too large!'])
        disp(['Fp must be less than one half the final Fs/R = ',...
                                num2str(Fs/R) ,' Hz sample rate!'])
```

```
        return % Stop all processing
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plot target CIC filter positive-freq. mag. response
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define frequency vector (Freq) in the range of 0 -to- 0.5
Freq = 0:1/(2*Num_Freq_Points):0.5 -1/(2*Num_Freq_Points);

% CIC filter frequency response equation
Spec_CIC = (sin(pi*Freq*R*N)./sin(pi*Freq)).^M;
Spec_CIC(1) = (R*N)^M; % Set correct DC (zero Hz) gain

Spec_Mag_CIC = abs(Spec_CIC); %Magnitude
Spec_Mag_CIC = Spec_Mag_CIC/max(Spec_Mag_CIC); % Set max mag. to one
Spec_Mag_dB_CIC = 20*log10(Spec_Mag_CIC); % Decibels

    figure(1), clf
    Freq_Plot_Axis_1 = (0:Num_Freq_Points-1)*Fs/(2*Num_Freq_Points);
    plot(Freq_Plot_Axis_1, Spec_Mag_dB_CIC,  '-k')
    axis([0, Fs/2, Threshold, 5])
    xlabel('Hz'), ylabel('dB'),
    title('CIC mag resp. before decimation'), grid on, zoom on

% Determine & plot CIC filter aliasing after decimation
Mainlobe_Indices = 1:round(2*Num_Freq_Points/(R));
Mainlobe = Spec_Mag_dB_CIC(Mainlobe_Indices);
Mainlobe_Flipped = fliplr(Mainlobe);

% Find "aliased-mainlobe < mainlobe" freq indices
Indices = [1];  % Intialize
for Loop = 2:round(2*Num_Freq_Points/(R));
    if Mainlobe(Loop) >= Mainlobe_Flipped(Loop)
    Indices = [Indices, Loop];
    else, end
end
    hold on
    plot(Freq_Plot_Axis_1(Indices), Mainlobe_Flipped(Indices), '-r')
    hold off
    text(Fs/R, 0, 'Red shows the most significant')
    text(Fs/R, -7, ' (but not the total) CIC filter')
    text(Fs/R, -14, '          aliasing after decimation')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of CIC filter design
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Design the FIR compensation filter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Num_FIR_Design_Freq_Points = 4*FIR_Order;
FIR_Freq_Vector = linspace(0, 1, Num_FIR_Design_Freq_Points);

% Create freq vector for defining inverse CIC mag. response
% over the FIR compensation filter's 'Fp' passband.
CIC_Freq_vector = 0:round(Num_FIR_Design_Freq_Points*Fp/(Fs/(2*R)));
CIC_Freq_vector = (1/R)*CIC_Freq_vector/Num_FIR_Design_Freq_Points;

% Define an inverse CIC magnitude vector
Inverse_CIC = 1./((1/(R*N)^M)*(sin(pi*CIC_Freq_vector*R*N/2)...
                               ./sin(pi*CIC_Freq_vector/2)).^M);
```

```
Inverse_CIC(1) = 1; % Eliminate the NaN first sample
Inverse_CIC(Num_FIR_Design_Freq_Points) = 0;
Freq_Plot_Axis_2 = (0:Num_FIR_Design_Freq_Points-1)*(Fs/(2*R))...
                                        /(Num_FIR_Design_Freq_Points);
    figure(2), clf
    plot(Freq_Plot_Axis_2, Inverse_CIC, '-bs', 'markersize', 2)
    title('Desired FIR resp.= Blue, Actual FIR resp.= Red')
    ylabel('Linear'), xlabel('Hz'), grid on, zoom on

% Compute FIR compensation filter coeffs
FIR_Coeffs = fir2(FIR_Order, FIR_Freq_Vector, ...
                           Inverse_CIC, chebwin(FIR_Order+1, 50));

disp(['FIR coefficients = ', num2str(FIR_Coeffs)])

% Compute freq response of a unity-gain FIR Comp. filter
Num_Freq_Points = 512;
Spec_Comp_Filter = fft(FIR_Coeffs, 2*Num_Freq_Points);
Spec_Comp_Filter = Spec_Comp_Filter(1:Num_Freq_Points); % Pos. freqs only

Mag_Comp_Filter = abs(Spec_Comp_Filter);
Mag_dB_Comp_Filter = 20*log10(Mag_Comp_Filter);

%Freq_Plot_Axis_3 = (0:Num_Freq_Points-1)*(Fs/R)/(Num_Freq_Points);
Freq_Plot_Axis_3 = (0:Num_Freq_Points-1)*(Fs/(2*R))/(Num_Freq_Points);

    figure(2)
    hold on
    plot(Freq_Plot_Axis_3, Mag_Comp_Filter, '-r', 'markersize', 2)
    hold off
    axis([0, Fs/(2*R), 0, 1.2*max(Inverse_CIC)])

    figure(3), clf
    plot(Freq_Plot_Axis_3, Mag_dB_Comp_Filter)
    axis([0, Fs/(2*R), -80, max(Mag_dB_Comp_Filter)+5])
    xlabel('Hz'), ylabel('dB')
    title('FIR Comp. Filter mag. resp.'), grid on, zoom on

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of FIR compensation filter design
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Multiply the unity-gain FIR comp. filter complex freq response times
%   the CIC filter's complex **mainlobe-only** freq response
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Compute CIC filter's mainlobe-only response (using Num_Freq_Points)
CIC_Mainlobe_Freq_vector = 0:Num_Freq_Points -1;
CIC_Mainlobe_Freq_vector = 0.5*(2/R)*...
                  CIC_Mainlobe_Freq_vector/Num_Freq_Points;

CIC_Mainlobe = ((1/(R*N))*(sin(pi*CIC_Mainlobe_Freq_vector*R*N/2)...
                          ./sin(pi*CIC_Mainlobe_Freq_vector/2))).^M;
CIC_Mainlobe(1) = 1; % Zero Hz

% Compute the frequency magnitude response of the cascaded filters
Cascaded_Filter_Spec_Mag = abs(CIC_Mainlobe...
                                       .*Spec_Comp_Filter);
```

```
% Compute cascaded response in dB for plotting
Cascaded_Filter_Spec_Mag_dB = 20*log10(Cascaded_Filter_Spec_Mag);
Freq_Plot_Axis_3 = (0:Num_Freq_Points-1)*(Fs/(2*R))/(Num_Freq_Points);

    % Plot cascaded filters' combined freq. magnitude response
    figure(4), clf
    subplot(2,1,1)
    plot(Freq_Plot_Axis_3, Cascaded_Filter_Spec_Mag_dB, '-k')
    hold on
    plot(Freq_Plot_Axis_1(1:51), Spec_Mag_dB_CIC(1:51), '-r')
    plot(Freq_Plot_Axis_3, Mag_dB_Comp_Filter, '-b')
    hold off
    axis([0, Fs/(2*R), Threshold, max(Mag_dB_Comp_Filter)+10])
    xlabel('Hz'), ylabel('dB')
    title('Black = cascaded filter,  Blue = comp. filter, Red = CIC')
    grid on, zoom on

    subplot(2,1,2)
    plot(Freq_Plot_Axis_3, Cascaded_Filter_Spec_Mag_dB, '-k')
    hold on
    plot(Freq_Plot_Axis_1(1:51), Spec_Mag_dB_CIC(1:51),  '-r')
    plot(Freq_Plot_Axis_3, Mag_dB_Comp_Filter, '-b')
    hold off
    title('Black = cascaded filter,  Blue = comp. filter, Red = CIC')
    axis([0, 1.2*Fp, -6-N, max(Mag_dB_Comp_Filter)+2])
    xlabel('Hz'), ylabel('dB')
    grid on, zoom on
```