Digital Filter Design Toolkit

June 2008, 371988B-01

The LabVIEW Digital Filter Design Toolkit includes several filter design tools for designing, analyzing, and simulating floating-point and fixedpoint digital filters, including multirate filters. This help file discusses the general digital filter design process and introduces the tools in the Digital Filter Design Toolkit that you can use in a digital filter design.



Note All occurrences of *filters* in this book refer to <u>single-rate</u> <u>filters</u> unless these topics explicitly use <u>multirate filters</u>.

You can use the <u>Signal Processing</u> VIs in the LabVIEW Full or Professional Development System to perform waveform measurements, waveform conditioning, waveform monitoring, waveform generation, signal processing, and point-by-point analysis. The Signal Processing VIs contain some digital filter design VIs similar to VIs in the Digital Filter Design Toolkit. For example, the <u>Butterworth Coefficients</u> VI is similar to the <u>DFD Butterworth Design</u> VI and the <u>Parks-McClellan</u> VI is similar to the <u>DFD Remez Design</u> VI. However, the <u>Digital Filter Design</u> VIs provide more capabilities, such as support for arbitrary phase and magnitude specifications and fixed-point filter design.

Although the VIs have similar functionality, the results you obtain might be different because the design algorithms are different. Refer to the <u>National Instruments Web site</u> at ni.com for information about working with the Signal Processing VIs and the Digital Filter Design VIs.

This help file contains:

- <u>Concepts</u>—An overview of how to use the Digital Filter Design Toolkit.
- <u>How-To</u>—Step-by-step instructions on accomplishing tasks using the Digital Filter Design Toolkit.
- <u>Reference</u>—Detailed information about the Digital Filter Design VIs.
- <u>MathScript Functions</u>—Detailed information about the Digital Filter Design MathScript classes of functions and commands that LabVIEW MathScript supports.
- To view related topics, click the **Locate** button, shown at left, in the

toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

© 2005–2008 National Instruments Corporation. All rights reserved.

Related Documentation (Digital Filter Design Toolkit)

The following documents contain information that might be helpful as you use the LabVIEW Digital Filter Design Toolkit.

You must <u>install the PDFs</u> to access them from this help file. You must have Adobe Reader 6.0.1 or later installed to view or <u>search</u> the PDF versions of these manuals. Refer to the <u>Adobe Systems Incorporated</u> <u>Web site</u> to download Adobe Reader. Refer to the <u>National Instruments</u> <u>Product Manuals Library</u> for updated documentation resources.

- LabVIEW Digital Filter Design Toolkit Readme—Use this file to obtain introductory information about the Digital Filter Design Toolkit, such as overview, system requirements, installation instructions, and known issues with LabVIEW. Open this readme by selecting Start»All Programs»National Instruments»LabVIEW»Readme and opening readme_DFDT.html or by navigating to the labview\readme\ directory and opening readme_DFDT.html.
- LabVIEW Digital Filter Design Toolkit Example VIs—Refer to the labview\examples\Digital Filter Design directory for example VIs that demonstrate common tasks using the Digital Filter Design Toolkit. You also can access these VIs by selecting Help»Find Examples from the pull-down menu and selecting Toolkits and Modules»Digital Filter Design in the NI Example Finder window.
- Additional LabVIEW documentation

The following list of references contains more information about the theory and algorithms implemented in the Digital Filter Design Toolkit.

- Chugani, Mahesh L.; Abhay R. Samant; and Michael Cerna. *LabVIEW Signal Processing*. Upper Saddle River, NJ: Prentice Hall, 1998.
- Diniz, Paulo S. R.; Eduardo A. B. da Silva; and Sergio L Netto. *Digital Signal Processing: System Analysis and Design*. New York: Cambridge University Press, 2002.
- Hogenauer, E. B. "An economical class of digital filters for decimation and interpolation." *IEEE Transactions on Acoustics*,

Speech, and Signal Processing, ASSP-29 (2) (1981): 155–162.

- Ifeachor, E. C., and B. W. Jervis. *Digital Signal Processing: A Practical Approach*. 2nd ed. Publishing House of Electronics Industry, 2003.
- Jayasimha, S., and P. V. R. N. Rao. "An iteration scheme for the design of equiripple Mth-band FIR filters." *IEEE Transactions on Signal Processing*, vol. 43 (8) (August 1995): 1998-2002.
- Mintzer, F. "On half-band, third-band, and nth-band FIR filters and their design." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-30 (5) (October 1982): 734–738.
- Neuvo, Y; C-Y Dong; and S.K. Mitra. "Interpolated finite impulse response filters." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-32 (June 1984): 563–570.
- Oppenheim, A. V., and R. W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.
- Orfanidis, S. J. *Introduction to Signal Processing*. Upper Saddle River, NJ: Prentice Hall, 1998.
- Parks, T. W., and C. S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, Inc., 1987.
- Rabiner, L. R. "Approximate design relationships for low-pass FIR digital filters." *IEEE Transactions on Audio and Electroacoustics*, vol. 21 (5) (October 1973): 456–460.
- Selesnick, I. W., and C. S. Burrus. "Generalized digital Butterworth filter design." *IEEE Transactions on Signal Processing*, vol. 46 (6) (June 1998): 1688–1694.
- Vaidyanathan, P. P. *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice Hall, 1993.
- Vaidyanathan, P. P., and T. Q. Nguyen. "A 'trick' for the design of FIR half-band filters." *IEEE Transactions on Circuits and Systems*, vol. 34 (3) (March 1987): 297–300.

Digital Filter Design Toolkit Features

The LabVIEW Digital Filter Design Toolkit includes a variety of tools to help you design digital filters. For example, the Digital Filter Design Toolkit includes Express VIs that you can use to interact graphically with filter specifications to design appropriate digital filters.

In addition to the tools that help you quickly create digital filters, the Digital Filter Design Toolkit includes tools for single-rate and multirate digital filter design, floating-point to fixed-point conversion, filter analysis, and simulation on a desktop computer. The following sections describe the major features that the Digital Filter Design Toolkit provides.

Comprehensive Analysis Tools

You can use the <u>Filter Analysis</u> VIs to evaluate the characteristics of digital filters. You can examine the frequency response, group delay, phase delay, impulse response, step response, and pole-zero placement of a digital filter.

Large Selection of Filter Structures

When you design digital filters with the Digital Filter Design Toolkit, you can <u>select</u> from one of 23 possible filter structures, which range from the direct form and cascaded form structures to the lattice auto-regressive (AR), lattice moving average (MA), and lattice ARMA structures.

Filter structures are mathematically equivalent when you use floatingpoint computation. However, different structures can perform differently in fixed-point implementations and can lead to different computation complexity and memory usage in fixed-point or floating-point implementations. Selecting an appropriate filter structure is critical for digital filter design, especially for fixed-point digital filters in which the precision of the filter coefficients and filtering operations is more limited than for floating-point digital filters.

Multirate Digital Filter Design

The <u>Multirate Filter Design</u> VIs help you <u>design</u>, <u>analyze</u>, <u>and implement</u> single-stage multirate filters, multistage multirate filters, halfband filters, Nyquist filters, raised cosine filters, and cascaded integrator comb (CIC) filters.

Special Digital Filter Design

The <u>Special Filter Design</u> VIs help you <u>design</u> IIR notch/peak filters, IIR comb filters, maximally flat filters, narrowband filters, and group delay compensators.

Fixed-Point Filter Design and Code Generation

The Fixed-Point Tools VIs and Multirate Fixed-Point Tools VIs help you quantize, analyze, model, and simulate the fixed-point filter design, including single-rate and multirate filters. You can save the resulting fixed-point filter information as C code, which you then can implement on digital signal processing (DSP) chips. You also can save the resulting fixed-point information as LabVIEW code, which you then can implement on NI Reconfigurable I/O (RIO) targets.

Generalized Remez and Least Pth Norm Design Algorithms

You can use algorithms such as the <u>generalized Remez method</u> and the <u>least pth norm method</u> to specify an arbitrary magnitude and a phase response for a digital filter. The Digital Filter Design Toolkit includes automatic order-estimation VIs to assist you in estimating the filter order.

Supported Execution Targets (Digital Filter Design Toolkit)

You can use the LabVIEW Digital Filter Design Toolkit to execute a digital filter on many different types of targets. The following table shows the supported target types and any additional software you must install to execute a digital filter on that type of target.

Type of Target	Additional Required Software
A Windows PC	
A National Instruments field-programmable gate array (FPGA) target, such as an NI Reconfigurable I/O (NI-RIO) device. To use a digital filter on an FPGA target, you first must <u>generate FPGA code</u> for that filter.	 LabVIEW FPGA Module NI-RIO driver software
A 32-bit microprocessor. To use a digital filter on a 32-bit microprocessor, you first must generate fixed- point C code for that filter.	 LabVIEW with embedded target support The LabVIEW add-on for the embedded target
An NI Real-Time (RT) target, such as an NI PXI or CompactRIO controller, running the Ardence Phar Lap Embedded Tool Suite (ETS) or Wind River VxWorks real-time operating system (RTOS). The Digital Filter Design VIs support native execution on these operating systems.	 LabVIEW Real-Time Module Driver software for any hardware

Note Refer to the <u>National Instruments Web site</u> for information about the National Instruments products this table mentions.

 $\overline{\mathbb{N}}$

Digital Filter Design Concepts (Digital Filter Design Toolkit)

This book contains information about the digital filter design process. You can use the LabVIEW Digital Filter Design Toolkit to design floating-point and fixed-point digital filters. You need fundamental knowledge about digital signal processing to understand the content in this book. The book Discrete-Time Signal Processing provides necessary information to help you develop a firm base in the fundamentals.

You can use the Digital Filter Design Toolkit to design the following types of filters:

- **Single-rate filters**—Single-rate filters are digital filters that do not change the <u>sampling frequency</u> of a signal during the filtering process. Therefore, if you apply a single-rate filter to an input signal, the output signal has the same sampling frequency as the input signal. The <u>Designing Floating-Point Filters</u> book contains more information about designing single-rate filters.
- **Multirate filters**—Multirate filters are digital filters that convert the sampling frequency of an input signal to a new sampling frequency. Multirate filters increase or decrease the sampling frequency of the input signal while minimizing passband distortion, aliasing, and imaging in the signal. Therefore, the sampling frequency of the output signal from a multirate filter is different from that of the input signal. Multirate filters can reduce computational complexity and data volume in one system, or multirate filters can change the frequency as necessary to be compatible with other systems. In multirate signal processing, the primary consideration is the selection and modification of the proper sampling frequency. The <u>Designing Multirate Filters</u> book contains more information about designing multirate filters.
- To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

Digital Filter Design Basics (Digital Filter Design Toolkit)

Before using the LabVIEW Digital Filter Design Toolkit to design a digital filter, you need to obtain an overview of the <u>general design process</u> and <u>application areas</u> of digital filters.

When you design a digital filter, you begin by creating specifications that define the characteristics you want in the digital filter. You can design both <u>FIR and IIR filters</u> with specific <u>filter attributes</u>, and you can customize the <u>sampling frequency</u>, <u>filter specifications</u>, and <u>design method</u>. After you design a digital filter, you need to analyze the characteristics of the digital filter by evaluating the magnitude and impulse responses, <u>phase responses and group delays</u>, or <u>poles and zeroes</u>. The Digital Filter Design Toolkit provides the <u>Filter Analysis</u> VIs to help you evaluate the characteristics of a filter.

To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

Digital Filter Design Process Overview (Digital Filter Design Toolkit)

Digital filter design involves the following three steps:

- 1. Specifying the design method and target digital filter attributes, or the characteristics you want the digital filter to possess.
- 2. Analyzing the characteristics of the floating-point digital filter you designed in Step 1.
- 3. Implementing the filter on fixed-point targets with finite-precision arithmetic.

The design process is iterative. You usually experiment with different design specifications or design methods to obtain the appropriate digital filter for an application. Sometimes you might need to revise the specifications or modify the design method after you simulate the filter, especially when designing fixed-point filters.

Filter Specifications (Digital Filter Design Toolkit)

For most digital filters, you typically design the digital filter response in the frequency domain. The frequency response specification for the digital filter typically includes the target magnitude response, phase response, and the allowable deviation for each. The following figure illustrates the magnitude frequency response of a lowpass filter, which allows low frequencies to pass and attenuates high frequencies.



The frequency range from the passband edge frequency to the stopband edge frequency is the transition band, which has a frequency response that is unspecified. The filter passband and stopband can contain oscillations, which are known as ripples. A typical example of a ripple appears in the circle of the previous figure. δ_p indicates the magnitude of the passband ripple, which equals the maximum deviation from the unity. δ_s indicates the magnitude response of the stopband ripple, which equals the maximum deviation from the unity.

Notice the transition band between the passband and stopband frequencies. In an ideal design, a digital filter has a target gain in the passband and a zero gain ($-\infty$ dB) in the stopband. In a real implementation, a finite transition region between the passband and the stopband, which is known as the transition band, always exists. The gain of the filter in the transition band is unspecified. The gain usually changes gradually through the transition band from 1 (0 dB) in the passband to 0 ($-\infty$ dB) in the stopband.

You can measure the passband ripple and stopband ripple in decibels, as shown in the following equations:

passband ripple = $-20\log_{10}(1-\delta_p)$

stopband ripple = $-20\log_{10}(\delta_s)$

Based on the two equations above, you can convert the passband ripple to or from the decibel representation. For example, if passband ripple equals 0.01 dB, that is, $0.01 = -20\log_{10}(1-\delta_p)$, then $\delta_p = 0.00115$. Similarly, if stopband ripple equals 60 dB, that is $60 = -20\log_{10}(\delta_s)$, then $\delta_s = 0.001$.

The following figure illustrates the magnitude frequency responses of a highpass filter, which passes high frequencies and attenuates low frequencies.



The following figure illustrates the magnitude frequency responses of a bandpass filter, which passes a certain band of frequencies and attenuates lower and higher frequencies.



In the previous figure, stopband edge frequency 1 indicates the maximum frequency of the lower frequency range that you want to attenuate, and stopband edge frequency 2 indicates the minimum frequency of the higher frequency range that you want to attenuate. The frequency range between passband edge frequency 1 and 2 indicates the range of frequencies that can pass through the filter.

The following figure illustrates the magnitude frequency response of a bandstop filter, which attenuates a certain band of frequencies and passes all frequencies not within the band.



In the previous figure, passband edge frequency 1 indicates the maximum frequency of the lower frequency range that can pass through the filter, and passband edge frequency 2 indicates the minimum frequency of the higher frequency range that can pass through the filter. The frequency range between stopband edge frequency 1 and 2

indicates the range of frequencies that you want to attenuate.

Design Methods (Digital Filter Design Toolkit)

The LabVIEW Digital Filter Design Toolkit provides the following finite impulse response (FIR) filter design methods.

- Kaiser Window
- Dolph-Chebyshev Window
- Equi-Ripple FIR

The Kaiser Window method and the Dolph-Chebyshev Window method allows you to obtain the filter coefficients directly from the analytical equations, so these methods are easier to use than the Equi-Ripple FIR method, which also is known as the <u>Remez design method</u>, but the Equi-Ripple FIR method yields optimal filters and often produces the best results for most FIR filter design problems.

In addition to the FIR-based methods, the Digital Filter Design Toolkit supports the following infinite impulse response (IIR) filter design methods.

- Butterworth
- Chebyshev
- Inverse Chebyshev
- Elliptic

The following figure illustrates the magnitude responses of a typical lowpass filter designed by the four IIR filter design methods. Each filter has the same numerator and denominator order values.



The following table summarizes the main features of the four IIR-based design methods so you can determine the IIR filter design method to use.

IIR Filter	Ripple in Passband?	Ripple in Stopband?	Transition Bandwidth for a Fixed Order	Order for Given Filter Specifications
Butterworth	No	No	Widest	Highest
Chebyshev	Yes	No	Narrower	Lower
Inverse Chebyshev	No	Yes	Narrower	Lower
Elliptic	Yes	Yes	Narrowest	Lowest

Group Delay and Phase Delay (Digital Filter Design Toolkit)

For a filter with a frequency response of $H(e^{j\omega})$, the phase delay response $\frac{1}{2}$ is defined by the following equation:

 $\tau_p = -\frac{\arg[H(e^{j\omega})]}{\omega}$

The group delay response r_{g} is defined as the negative derivative of the phase response ω , as shown in the following equation:

 $\tau_g(\omega) = -\frac{d}{d\omega} \arg[H(e^{j\omega})]$

Both the group delay and phase delay are in samples.

For a generalized linear phase filter with $\arg[H(e^{j\omega})] = -\alpha \omega + \beta$, the group delay is represented by the following equation:

 $\tau_g(\omega) = -\frac{d}{d\omega} \arg[H(e^{j\omega})] = \alpha$

The phase delay is represented by the following equation:

 $\tau_p = -\frac{\arg[\mathcal{H}(e^{j\omega})]}{\omega} = \alpha - \frac{\beta}{\omega}$

You can represent the phase delay as the time delay in samples experienced by each frequency component of the input signal. The filter is represented by the following illustration:

 $\mathbf{x}(\mathbf{n}) \longrightarrow e^{f^{\beta}} \longrightarrow H_{new}(e^{j\alpha}) \longrightarrow \mathbf{y}(\mathbf{n})$

The filter $H(e^{j\omega})$ shifts all frequency components by a phase β and then filters the signal with a new filter $H_{\text{new}}(e^{j\omega})$ that has a phase of $-\alpha\omega$. You can interpret the group delay as the time delay in samples experienced by each frequency component through the new filter $H_{\text{new}}(e^{j\omega})$.

Linear phase filters are characterized by a constant group delay. The deviation of the group delay from a constant value within the passband indicates the degree of nonlinearity in the phase. Use the group delay to analyze the linearity of a filter.

Digital Filter Applications (Digital Filter Design Toolkit)

Filters are signal processing elements that alter the frequency spectrum of an input signal. You might use filters for the following applications:

- Attenuating noise in a signal where the noise power and signal power are concentrated at different frequencies. For example, you might use a notch filter to attenuate a 60 Hz powerline interference present in a signal.
- Extracting signal components from a signal that contains different signal components concentrated at different frequencies. For example, you might use a bandpass filter to extract a particular radio station signal from a broadband radio signal.
- Reshaping the frequency spectrum of the input signal. For example, you might use an A-weighting filter to approximate the frequency response of a human ear. As another example, you might use an equalizer filter to undo magnitude and phase distortion caused by passing a signal through a linear time-invariant communications channel.

You can use either fixed-point or floating-point arithmetic to implement digital signal processing systems. Although floating-point implementations are typically easier to design, fixed-point implementations are often less expensive and more efficient in power than floating-point implementations. Floating-point designs are typically appropriate in applications that run on desktop computers, and fixedpoint designs are often more appropriate in embedded applications, in which you need to minimize cost or power consumption.

Designing Floating-Point Filters (Digital Filter Design Toolkit)

This book explains how to use the interactive <u>Classical Filter Design</u> Express VI to design floating-point filters. This book also describes how to analyze and use a designed floating-point filter.

To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

Floating-Point Filter Design Process (Digital Filter Design Toolkit)

The following figure illustrates the floating-point filter design process. You first design the specifications of the filter. You then analyze the characteristics of the resulting filter to determine if the filter meets the requirements of the system. If the filter does not meet the requirements of the system, you can modify the specifications and repeat the process. After you design an appropriate filter, you can use the filter in the system.



For a particular design problem, you can use several different techniques and filter types to yield an acceptable result. To achieve the best results, you might need to experiment with several different approaches.

Entering Floating-Point Filter Specifications (Digital Filter Design Toolkit)

You can use the <u>Classical Filter Design</u> Express VI to <u>configure a</u> <u>classical digital filter</u> interactively. After you place the Express VI on the block diagram, the **Configure Classical Filter Design** dialog box appears, as shown in the following figure:



From the configuration dialog box, you can select the filter type and design method from the pull-down menus and then enter the filter specifications through either the numeric controls on the left side of the configuration dialog box or the graphical interface on the right side of the configuration dialog box. The results are equivalent.

Using the Numerical Controls

Classical digital filter specifications include frequency ranges and ripple constraints. You can specify the maximum allowable deviation δ_p from unity gain in the passband in the **Passband ripple** numeric control. You can specify the maximum allowable deviation δ_s from the zero gain in the stopband in the **Stopband attenuation** numeric control.

You can specify deviations in either a logarithmic or a linear scale. The Classical Filter Design Express VI uses a logarithmic scale by default. To use a linear scale, remove the checkmark from the **Magnitude in dB** checkbox in the configuration dialog box.

The following equations show the relationship between the logarithmic and linear scales.

passband ripple = $-20\log_{10}(1-\delta_p)$

stopband ripple = $-20\log_{10}(\delta_s)$

Based on the two equations above, you can convert the passband ripple to or from the decibel representation. For example, if passband ripple equals 0.01 dB, that is, $0.01 = -20\log_{10}(1-\delta_p)$, then $\delta_p = 0.00115$. Similarly, if stopband ripple equals 60 dB, that is $60 = -20\log_{10}(\delta_s)$, then $\delta_s = 0.001$.

Using the Graphical Interface

The right side of the **Configure Classical Filter Design** dialog box displays the magnitude response of the designed digital filter. The magnitude axis can be either a linear or a logarithmic scale. Remove the checkmark from the **Magnitude in dB** checkbox to use a linear scale, or keep the checkmark in the **Magnitude in dB** checkbox to use a logarithmic scale. The Frequency axis, in hertz, covers the range from 0 to half the sampling frequency, which is the Nyquist frequency.

The **Magnitude Response** graph contains a set of cursors that you can use to specify the passband and stopband. Use the passband cursor to change the passband. Under the linear scale, the distance between unity and the horizontal passband cursor specifies the maximum passband ripple. The location of the vertical passband cursor indicates the passband edge frequency. The stopband cursors work the same when defining the specifications of the stopband. The distance between the horizontal passband cursor and the horizontal stopband cursor specifies the stopband attenuation.

Guidelines for Entering Filter Specifications

As you define a filter specification, you must adhere to a set of rules to maintain valid specifications. If you do not adhere to the following rules, the **Error message** indicator of the **Configure Classical Filter Design** dialog box displays a message with suggestions for repositioning the cursors.

- Keep horizontal cursors in the range (0, 1) in a linear scale or (-inf, 0 dB) in a logarithmic scale.
- Keep the horizontal passband cursor above the horizontal stopband cursor.

Selecting the Design Method

After you enter the target digital filter specifications into the numeric controls or graphical interface, select a <u>design method</u>.

When you design a digital filter with the Classical Filter Design Express VI, the design method and the filter specifications that you specify control the shape of the frequency response. You cannot alter the phase response, even though the phase response for filters generated with the FIR methods in this VI are linear phase. If you want to specify the magnitude response and phase response, use the Advanced FIR Filter Design VIs, the Advanced IIR Filter Design VIs, or the Special Filter Design VIs.

Refer to the Lowpass_Step 1_Design Lowpass VI in the labview\examples\Digital Filter Design\Case Studies\Single-Rate Filter\ directory for an example that demonstrates how to use the Classical Filter Design Express VI to design a lowpass filter.

Open example

Analyzing Floating-Point Filters (Digital Filter Design Toolkit)

After you <u>enter the target specifications</u> for a digital filter, you can <u>analyze</u> the characteristics of the resulting filter in the **Configure Classical Filter Design** dialog box of the <u>Classical Filter Design</u> Express VI by evaluating the <u>pole-zero plot</u>, the magnitude response, and the filter order.

Magnitude Response

The frequency response of a digital filter is defined by $H(e^{j2\pi f})$, and the magnitude response is defined by $|H(e^{j2\pi f})|$. For discrete-time systems, $H(e^{j2\pi f})$ is periodic with a period of f_s . For real-valued digital filters, the magnitude response is symmetric with respect to $0, \pm f_s, \pm 2f_s, \ldots$. Therefore, you can calculate the magnitude response for only $[0, f_s/2]$, which contains the frequencies between 0 and the Nyquist frequency. The magnitude response graph in the **Configure Classical Filter Design** dialog box includes a green vertical line to indicate the location of $f_s/2$.

Filter Order Specification

The Classical Filter Design Express VI automatically computes the minimal filter order required to fulfill the given filter specification and displays the order in the **Filter order** indicator. With the same specification, you can use different algorithms to create digital filters with different filter orders. You can estimate the computational complexity and cost based on the filter order. If you have strict requirements for the system, the filter order can help you determine if the filter is acceptable.

Using Floating-Point Filters (Digital Filter Design Toolkit)

After you <u>analyze the filter design</u>, you can use the filter to <u>process</u> an input signal. Use the <u>Processing</u> VIs to process a signal with the filter you designed. The Processing VIs can process an input signal in the following two ways:

- As a sequence of data blocks
- As a sequence of data blocks with saved internal filter states

Refer to the Lowpass_Step 2_Perform Lowpass Filtering VI in the labview\examples\Digital Filter Design\Case Studies\Single-Rate Filter directory for an example that demonstrates how to apply a lowpass filter to a filtering application.

■ Open example

Designing Fixed-Point Filters (Digital Filter Design Toolkit)

This book explains how you use the <u>Fixed-Point Tools</u> VIs to implement a fixed-point digital filter from <u>a floating-point reference filter that you</u> <u>designed</u>.

To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.
Fixed-Point Filter Design Process (Digital Filter Design Toolkit)

Fixed-point signal processing platforms, such as fixed-point digital signal processors (DSPs) and field-programmable gate arrays (FPGAs), are typically more power-efficient and less expensive than floating-point alternatives. However, fixed-point systems are generally more difficult to design. For example, you must consider the effects of coarser quantizations in fixed-point systems.

To design a fixed-point filter, you first must <u>design a floating-point filter</u>, also known as a reference filter, that meets the target specifications. In some cases, for example, if you need an infinite impulse response (IIR) filter with a narrow transition band but a high stopband attenuation, you need to design a reference filter that exceeds the target specifications. The excess margin ensures a smooth conversion from a floating-point representation to a fixed-point representation. You then must modify the floating-point filter to accommodate the finite-precision constraints of the target platform while still trying to meet the target specifications. The following figure illustrates the fixed-point filter design process, the dotted lines represent optional steps, and the arrows on the left indicate to which steps you can return if the filter design fails to meet the requirements in the current step.



Designing a fixed-point filter from a reference floating-point filter involves the following steps:

- 1. <u>Selecting a filter structure</u>. In floating-point filter design, after you select a <u>design method</u>, the LabVIEW Digital Filter Design Toolkit uses a <u>default filter structure</u> according to the specified design method. However, in fixed-point implementations, different filter structures can have different memory and multiplier requirements and might cause different <u>finite word length effects</u>. To obtain the best filtering results, you must convert the default filter structure to an appropriate structure. This step is optional.
- Scaling the filter coefficients. Every filter structure contains many accumulators, each of which might use a different data range. You can scale the filter coefficients by using the <u>DFD Scale Filter</u>

VI to ensure that all of the accumulators use the same data range. Scaling the filter coefficients can help you obtain a better filtering result, especially for <u>IIR Cascaded Second-Order</u> <u>Sections Form structures</u>. This step is optional.

- 3. Quantizing the floating-point filter. Quantization is the process of approximating a fixed-point value for each reference floating-point value. You then can use the fixed-point values in fixed-point mathematical computation or a hardware implementation. By quantizing the coefficients of the reference floating-point filter, you convert a floating-point filter to a fixed-point filter.
- 4. <u>Analyzing the fixed-point filter</u>. To determine how the characteristics of the realized fixed-point filter deviate from the characteristics of the reference floating-point filter, you must analyze the fixed-point filter.
- 5. <u>Creating a fixed-point filter model</u>. To create the fixed-point filter model, you must configure the quantizers for the input and output signals and specify the settings for internal computation.
- 6. <u>Simulating the fixed-point filter</u>. Before applying the fixed-point filter model in real-world applications, you must simulate the behavior of the filter to verify if the fixed-point filter model works as you require in a simulation. If the fixed-point filter does not provide the required performance in the simulation, you can change the implementation structure, modify quantization settings, or redefine the filter specifications for the reference floating-point filter.
- 7. <u>Generating code from the fixed-point filter</u>. You can export filter coefficients and automatically generate integer LabVIEW code, LabVIEW FPGA code, and C code from the fixed-point filter for designated hardware targets.

Finite Word Length Effects

Converting a floating-point filter to fixed-point can alter the characteristics and performance of the filter significantly. You must analyze the filter and simulate the filtering process with expected input signals. Fixed-point arithmetic can have the following detrimental effects on filter performance.

- Degraded signal-to-noise ratio (SNR) due to the reduced precision of internal registers, adders, subtracters, and multipliers
- Distorted frequency response from a limited <u>word length</u> <u>representation</u> of filter coefficients
- Overflowed or clipped signal information due to insufficient headroom in the signal paths
- Zero-input limit cycles of infinite impulse response (IIR) filters due to nonlinear quantizers in the feedback loop of IIR filters or to the overflow of the summation operations

Selecting a Filter Structure (Digital Filter Design Toolkit)

A filter structure specifies how you arithmetically use a set of filter coefficients to process an input signal. For a specified digital filter, dozens of mathematically equivalent implementation structures are available. For a floating-point digital filter, the effects of different implementation structures on the filter behavior are negligible in most cases. For a fixedpoint digital filter, different implementation structures can result in different signal outputs.

In addition to FIR and IIR structures, the LabVIEW Digital Filter Design Toolkit also provides lattice structures. Lattice structures, including <u>lattice</u> <u>allpass</u>, <u>lattice AR</u>, <u>lattice ARMA</u>, and <u>lattice MA</u>, can be good alternatives for fixed-point filter implementation. For example, lattice structures can preserve the stability of fixed-point IIR filters as long as the <u>lattice reflection coefficients</u> have moduli less than one, regardless of how limited the arithmetic precision might be.

The Digital Filter Design Toolkit provides the following three categories of lattice structures.

- **Basic Section Type**—Two multipliers per lattice section. This category offers the most general lattice structure.
- **One Multiplier Section Type**—Only one multiplier per lattice section. This category saves resources on hardware targets such as field-programmable gate arrays (FPGAs).
- **Normalized Section Type**—Four multipliers per lattice section. This category automatically scales the internal signals to help minimize the quantization effects in each lattice section at the cost of increasing the implementation complexity.

When you <u>select a filter structure</u>, you must balance a number of factors, including the filter type, implementation resources, and computational complexity. For IIR filters, you also need to consider the sensitivity to coefficient quantization of each structure. The following table lists the default filter structures that the <u>Filter Design</u> VIs use.

Design Method	Default Structure
Kaiser Window	FIR Direct Form

Dolph-Chebyshev Window	FIR Direct Form
Windowed FIR	FIR Direct Form
Remez/Equi-Ripple	FIR Direct Form
Least P th Norm FIR	FIR Direct Form
IIR Notch Peak	IIR Direct Form II
IIR Comb	IIR Direct Form II
Arbitrary Group Delay	IIR Direct Form II
Least P th Norm IIR	IIR Direct Form II
Butterworth	IIR Cascaded Second-Order Sections Form II Transposed
Chebyshev	IIR Cascaded Second-Order Sections Form II Transposed
Inverse Chebyshev	IIR Cascaded Second-Order Sections Form II Transposed
Elliptic	IIR Cascaded Second-Order Sections Form II Transposed
Bessel	IIR Cascaded Second-Order Sections Form II Transposed
Maxflat	IIR Cascaded Second-Order Sections Form II Transposed
Group Delay Compensator	IIR Cascaded Second-Order Sections Form II Transposed

You can use the <u>DFD Convert Structure</u> VI to select a different filter structure, with the following caveats:

- You cannot convert an IIR structure into or from an FIR structure.
- You cannot convert a lattice allpass structure into or from a lattice AR structure.
- You can convert an FIR filter to an FIR Symmetric filter structure only if the FIR filter has symmetric coefficients.
- You can convert an FIR filter to an FIR Antisymmetric filter structure only if the FIR filter has antisymmetric coefficients.
- You can convert an FIR filter to a lattice MA (minimum phase)

filter structure only if the FIR filter is minimum phase.

- You can convert an FIR filter to a lattice MA (maximum phase) filter structure only if the FIR filter is maximum phase.
- You must use an allpass filter if you want to convert a filter structure to a lattice allpass structure.
- You must use an all-pole IIR filter if you want to convert a filter structure to a lattice AR structure.

Refer to the Change Structure of Filter VI in the labview\examples\Digital Filter Design\Getting Started\Apply Filters directory for an example that demonstrates how to change the structure of a filter.

■ Open example

IIR Cascaded Second-Order Sections Form Structures (Digital Filter Design Toolkit)

The transfer function of an infinite impulse response (IIR) filter with a Cascaded Second-Order Sections Form structure is defined as follows:

 $H(z) = Gain \bullet \prod_{n=0}^{N-1} \frac{b_n[0] + b_n[1]z^{-1} + b_n[2]z^{-2}}{1 + a_n[1]z^{-1} + a_n[2]z^{-2}}$

where z is a complex variable, N is the number of sections, a is the set of reverse coefficients, and b is the set of forward coefficients.

IIR Cascaded Second-Order Sections Form I

Comparing with the <u>IIR Direct Form structures</u>, the IIR Cascaded Second-Order Sections Form structures have more computational complexity. However, the cascaded structures help alleviate <u>finite word</u> <u>length effects</u>. The following figure represents the IIR Cascaded Second-Order Sections Form I structure. Refer to the <u>Understanding Filter</u> <u>Structure Graphs</u> topic for information that helps you read and understand a filter structure graph.



IIR Cascaded Second-Order Sections Form II

The following figure represents the IIR Cascaded Second-Order Sections Form II structure. Comparing with Form I, this structure uses the same number of mathematical operations but fewer delays.



IIR Cascaded Second-Order Sections Form I Transposed

The following figure represents the IIR Cascaded Second-Order Sections Form I Transposed structure.



IIR Cascaded Second-Order Sections Form II Transposed

The following figure represents the IIR Cascaded Second-Order Sections Form II Transposed structure.



The IIR Cascaded Second-Order Sections Form I and Form II Transposed structures implement forward coefficients first. The Form I Transposed and Form II structures implement reverse coefficients first. The IIR Cascaded Second-Order Sections Form II structure has the same computational complexity as the Form I, but the Form I requires more memory for saving internal states. The Form II Transposed is the structure that you most frequently use. Using the Form I and Form II and their transposed structures has the same advantages and disadvantages as using the <u>FIR Direct Form</u> and <u>FIR Direct Form Transposed</u> structures.

IIR Direct Form Structures (Digital Filter Design Toolkit)

The transfer function of an infinite impulse response (IIR) filter is defined as follows:

$$H(z) = Gain \cdot \frac{\sum_{n=0}^{M} b[n] z^{-n}}{1 + \sum_{n=1}^{N} a[n] z^{-n}}$$

where z is a complex variable, M is the order of the numerator, N is the order of the denominator, a is the set of reverse coefficients, and b is the set of forward coefficients.

IIR Direct Form I

The IIR Direct Form I structure is the most straightforward IIR structure from a filter transfer function perspective. The following figure represents the IIR Direct Form I structure. Refer to the <u>Understanding Filter</u> <u>Structure Graphs</u> topic for information that helps you read and understand a filter structure graph.



Note This figure and the following figures show a special case when N = M.

IIR Direct Form II

The following figure represents the IIR Direct Form II structure. You can see that this structure contains fewer mathematical operations and delays.



IIR Direct Form I Transposed

The following figure represents the IIR Direct Form I Transposed structure.



IIR Direct Form II Transposed

The following figure represents the IIR Direct Form II Transposed structure.



The IIR Direct Form I and Form II Transposed structures implement forward coefficients first. The Form I Transposed and Form II structures implement reverse coefficients first. Using Form I and Form II and their transposed structures has the same advantages and disadvantages as using the <u>FIR Direct Form</u> and <u>FIR Direct Form Transposed</u> structures. The IIR Direct Form structures usually require few mathematical operations. However, the sensitivity to finite word length effects limits the use of this form in fixed-point implementations. Use the <u>IIR Cascaded</u> <u>Second-Order Sections Form structures</u> to alleviate <u>finite word length</u> <u>effects</u>.

Lattice Allpass Structures (Digital Filter Design Toolkit)

For an allpass filter, you can implement one of the following section types of lattice allpass filter structures:

- Basic section type
- One multiplier section type
- Normalized section type

The following figure represents the basic section type of a lattice allpass filter structure. Refer to the <u>Understanding Filter Structure Graphs</u> topic for information that helps you read and understand a filter structure graph.



The total number of the lattice reflection coefficients k is M, where M is the filter order. The total number of multipliers is 2M.

The following figure represents the one multiplier section type of a lattice allpass filter structure.



The total number of the lattice reflection coefficients *k* is *M*, which is the same as in the basic section type. However, the total number of multipliers is only *M*, which is half the number that the basic section type requires. Therefore, the one multiplier section type of a lattice allpass structure involves fewer multipliers than other section types. Fewer multipliers require less hardware resources.

The following figure represents the normalized section type of a lattice

allpass filter structure.



You can derive *k*' from *k* in the normalized section type by using the formula $k'[m] = \sqrt{1 - (k[m])^2}$.

One advantage of the normalized section type structure is that this structure automatically scales the internal signals in each lattice section. Unfortunately, scaling the internal signals increases the implementation complexity.

Understanding Filter Structure Graphs (Digital Filter Design Toolkit)

The realization of a digital filter involves summations and multiplications of the output, input, and intermediate operands. You must make the values that pass along the signal path available during the realization process. Therefore, to represent the structure of a filter using a signal flow graph, you not only need adders and multipliers, but you also need delays that help you store the passed values. The following figure shows the symbol of an adder.



You can treat a consecutive sequence of adders in a filter structure as an accumulator.

The following figure shows the symbol of a multiplier.

The following figure shows the symbol of a delay.

In the previous figure, z^{-1} is a delay that stores the value of x[n]. The ztransform of x[n-1] is z^{-1} times the z-transform of x[n]. The number of adders and multipliers implies computational complexity in the realization of a filter structure, and the number of delays implies memory unit requirements in the hardware. The more adders, multipliers, and delays a filter structure contains, the more computational complexity and memory units the filter requires. You can view the signal flow graphs of the following filter structures:

- FIR structures
- IIR Cascaded Second-Order Sections Form structures
- IIR Direct Form structures
- Lattice Allpass structures
- Lattice AR structures

- Lattice ARMA structures
- Lattice MA structures

Scaling the Filter Coefficients (Digital Filter Design Toolkit)

A filter structure consists of many <u>accumulators</u>. Each accumulator might use a different data range. However, the LabVIEW Digital Filter Design Toolkit provides only one <u>sum quantizer</u> $Q_{S>}$ for all the accumulators. You can <u>scale the filter coefficients</u> before <u>quantizing</u> them to ensure that all the accumulators use the same data range. Scaling the filter coefficients can help you obtain a better filtering result, especially for <u>IIR Cascaded</u> <u>Second-Order Sections Form structures</u>. Use the <u>DFD Scale Filter</u> VI to scale the coefficients of a floating-point filter.

Refer to the Scale Filter before Targeting to FXP VI in the labview\examples\Digital Filter Design\Fixed-Point Filters\Single-Rate directory for an example that demonstrates how to scale the filter coefficients.

Open example

Quantizing Floating-Point Filters (Digital Filter Design Toolkit)

After selecting a filter structure, you must quantize the coefficients of the reference floating-point filter. Quantizing the filter coefficients is the process of approximating each floating-point value with a fixed-point value that you use in a fixed-point mathematical computation or hardware implementation. Using the DFD FXP Quantize Coef VI, you can configure the filter coefficients quantizer Q_C and convert the reference floating-point filter.

If you use the Easy instance of the DFD FXP Quantize Coef VI to quantize the filter coefficients, you must complete the following steps:

- 1. Specify appropriate <u>word length</u> values for **coefficients a/k word length** and **coefficients b/v word length**.
- 2. Specify the appropriate gain processing target.
- 3. Specify the appropriate gain word length value. This VI automatically calculates the integer word lengths for coefficients *alk*, coefficients *blv*, and gain. The VI then uses the resulting values to quantize the filter coefficients.

If you use the Advanced instance of the DFD FXP Quantize Coef VI to quantize the filter coefficients, you must complete the following steps:

- 1. <u>Configure the coefficients a/k quantizer and coefficients b/v quantizer.</u>
- 2. Specify the appropriate gain processing target.
- 3. Specify the appropriate gain word length value. This VI then automatically calculates the integer word length of gain and uses the resulting value to quantize the gain if you set the **gain processing** input to On Target.

Specifying the Gain Processing Target

A filter gain implies a multiplication operation. If you process the filtered signal on a target, for example, an NI Reconfigurable I/O (RIO) target, the filtering process requires hardware resources on the target for the multiplication operation that the filter gain introduces. However, if you want to process the filtered signal on a host machine, you can move the filter gain operation to the host machine and save resources on the target. Use the **gain processing** input of the DFD FXP Quantize Coef VI to specify an appropriate gain processing target.

Refer to the Lowpass_Step 3_Analyze Quantized Lowpass VI in the labview\examples\Digital Filter Design\Case Studies\Single-Rate Filter directory for an example that demonstrates how to quantize a floating-point lowpass filter and <u>analyze the quantized lowpass filter</u>.

Open example

Filter Structures and Filter Coefficients (Digital Filter Design Toolkit)

Different filter structures use different groups of filter coefficients. The LabVIEW Digital Filter Design Toolkit categorizes filter coefficients into three groups: zeroes, poles, and gain. For example, FIR filter coefficients, IIR forward coefficients, and lattice ladder coefficients correspond to zeroes. IIR reverse coefficients and lattice reflection coefficients, excluding lattice MA reflection coefficients, correspond to poles. Different groups of filter coefficients have different data ranges. Therefore, in addition to the gain, the Digital Filter Design Toolkit provides the following two quantizers for these groups: coefficients *a/k* and coefficients *b/v*.

When you <u>quantize the filter coefficients</u>, configure the appropriate quantizer according to the filter structure type. The following table lists the filter structures, the filter coefficients, and the corresponding quantizers.

Filter Structure	Filter Coefficients and the Corresponding Quantizer
FIR structures	All FIR filter coefficients correspond to the coefficients <i>b/v</i> quantizer.
<u>IIR Direct Form</u> <u>structures</u>	The reverse coefficients correspond to the coefficients <i>alk</i> quantizer. The forward coefficients correspond to the coefficients <i>blv</i> quantizer.
IIR Cascaded Second-Order Sections Form structures	
Lattice Allpass structures	The reflection coefficients correspond to the coefficients <i>alk</i> quantizer.
<u>Lattice AR</u> <u>structures</u>	
<u>Lattice MA</u> <u>structures</u>	
<u>Lattice ARMA</u> <u>structures</u>	The reflection coefficients correspond to the coefficients <i>a/k</i> quantizer. The ladder coefficients correspond to the coefficients <i>b/v</i> quantizer.

Filter Coefficients and Quantizers (Digital Filter Design Toolkit)

In a fixed-point filter implementation, you quantize the coefficients as well as the intermediate operands and results. The LabVIEW Digital Filter Design Toolkit uses quantizers to store different quantization settings for the coefficients and the intermediate operands and results. You must configure all quantizers correctly in a fixed-point filter implementation. The following figure shows an example of a fixed-point model.



This fixed-point model contains the following quantizers:

- Q_l is the input quantizer for the input signal of a fixed-point filter.
- Q_S is the sum quantizer for the summation of a fixed-point adder.
- Q_D is the delay quantizer for the input of a delay element.
- Q_M is the multiplicand quantizer for the multiplicand of a fixedpoint multiplier, which multiplies a quantized multiplicand by a quantized coefficient.
- Q_C is the filter coefficients quantizer for the reference floatingpoint filter. Depending on <u>the group type of the filter coefficients</u>, Q_C can be coefficients a/k or coefficients b/v.
- *Q_P* is the product quantizer for the product of a fixed-point multiplier.
- *Q*_O is the output quantizer for the output signal of a fixed-point filter.

Each quantizer has a different effect on a fixed-point filter response

depending on the filter structure. You must <u>create a fixed-point filter</u> <u>model</u> and <u>simulate the behavior of the filter model</u> through trial and error before you use the corresponding fixed-point filter. Although you can determine the effects of coefficient quantization at design time, you cannot determine other quantization effects until you filter the expected input signals. For example, the actual data might be too large or might lead to limit cycles.

When you configure the quantizers, depending on which VI you use, you might need to complete some or all of the following items:

- Specify the quantizer source
- Specify the word length and integer word length
- Handle overflows and underflows
- <u>Specify the rounding mode</u>

Specifying the Quantizer Source (Digital Filter Design Toolkit)

A fixed-point filter implementation involves many different <u>quantizers</u>, such as the coefficients quantizer, the input quantizer, and the product quantizer. Each quantizer has a different effect on a fixed-point filter response. You must specify the appropriate quantizer source when configuring the quantizers. Use the **source** input of the <u>Fixed-Point Tools</u> VIs to choose an appropriate quantizer source.

Specifying the Word Length and Integer Word Length (Digital Filter Design Toolkit)

The word length indicates the number of bits you want to use in representing a fixed-point number. The integer word length specifies the number of bits, including the sign bit, you use in representing the integer part of a fixed-point number. The difference in bits between the word length and the integer word length determines the digits of precision.

The finite word length of a <u>quantizer</u> can affect the frequency response of the resulting fixed-point filter. The larger word length value you specify, the less the fixed-point representation distorts the frequency response. However, a larger word length value also requires more hardware resources, so you must specify a word length that provides an acceptable tradeoff between distortion and hardware resource consumption.

Use the **wl** and **iwl** inputs of the <u>Fixed-Point Tools</u> VIs to specify the word length and integer word length, respectively, of a quantizer.

Handling Overflows and Underflows (Digital Filter Design Toolkit)

Fixed-point numbers can represent only numbers of a finite range. Overflows occur when a number is greater than the maximum representable number within the range. Underflows occur when a number is less than the minimum representable number within the range. You can handle overflows and underflows using one of the following two modes:

- **Saturation**—A <u>quantizer</u> converts the specified number to the maximum representable number in the case of an overflow or to the minimum representable number in the case of an underflow.
- Wrap—A quantizer wraps the specified value from the maximum representable number to the minimum representable number in the case of an overflow and from the minimum representable number to the maximum representable number in the case of an underflow. In the wrap mode, when an overflow or an underflow occurs, the absolute value of the error is 2^{iwl}, which is greater than the total available dynamic range.

The saturation mode of the output quantizer is preferred over the wrap mode in most real-world applications because the saturation mode helps avoid signal discontinuities, or sudden changes in the amplitudes. However, the saturation mode is more complicated than the wrap mode. For internal quantizers, such as the sum quantizer, the wrap mode is preferred because this mode allows intermediate overflows and underflows within a certain range as long as the final output does not contain overflows or underflows. Use the **overflow mode** input of the <u>Fixed-Point Tools</u> VIs to specify an appropriate setting for handling overflows and underflows.

Setting the Rounding Mode (Digital Filter Design Toolkit)

Fixed-point numbers represent discrete values with limited precision. Typically, the precision of fixed-point numbers is less than that of floatingpoint numbers. Rounding determines the most appropriate fixed-point number to represent a specified floating-point value based on the precision you specify. Use one of the following modes to specify how you want rounding to occur in a <u>quantizer</u>.

- **Nearest**—The nearest mode rounds to the closest representable number. If the two nearest representable numbers are an equal distance apart, this mode rounds to the nearest representable number whose least significant bit is 0. The rounding error of this mode is zero-mean, but this mode has higher implementation complexity than the **Truncation** mode due to the computation of choosing the closest representable number.
- **Truncation**—The truncation mode rounds to the closest representable number less than the original value. This mode is the most common rounding mode in hardware. However, the rounding error of this mode has a nonzero mean.

Use the **rounding mode** input of the <u>Fixed-Point Tools</u> VIs to set an appropriate rounding option.

Analyzing Fixed-Point Filters (Digital Filter Design Toolkit)

The <u>Filter Analysis</u> Express VI performs <u>fixed-point analysis</u> in the frequency domain. Use the calculated results to optimize the fixed-point filter. You might need to analyze and adjust the filter design iteratively until the calculated results are satisfactory.

Use the Filter Analysis Express VI to observe the response of the fixedpoint filter. Ensure that the fixed-point filter is stable by verifying that all poles are within the unit circle and that the filter maintains a satisfactory frequency response. If the fixed-point characteristics do not satisfy the requirements, try one or more of the following options:

- Return to the <u>quantization</u> step and change the quantizer settings.
- Change the implementation structure.
- <u>Change the floating-point reference filter specifications</u> to allow more headroom for finite-precision effects.
- For infinite impulse response (IIR) filters, reduce the pole radius constraint of the reference floating-point filter.

If you notice large distortions in the response, look for underflow or overflow conditions. Use the <u>DFD FXP Coef Report</u> VI to determine if overflows or underflows exist in the quantized coefficients. If you find occurrences of overflows or underflows in the coefficients report, repeat the quantization step by using an increased **iwl** value for the coefficients quantizer. Increasing the **iwl** value helps eliminate overflows and underflows and improve the frequency response of the filter. If you do not find any occurrences of overflows or underflows, try reducing the value of iwl to allow more digits of precision for the quantized coefficients. If the minimum allowable iwl value still returns large distortions in the frequency response, you need to increase the value of **wl** to minimize the distortion.

Note Underflows and overflows that happen during quantization do not always affect filter responses. Therefore, if the filter response after coefficients quantization is satisfactory, you do not need to make adjustments to avoid overflows or underflows.

Refer to the Lowpass_Step 3_Analyze Quantized Lowpass VI in the

labview\examples\Digital Filter Design\Case Studies\Single-Rate Filter directory for an example that demonstrates how to <u>quantize a floating-point lowpass filter</u> and analyze the quantized lowpass filter.

■ Open example

Creating Fixed-Point Filter Models (Digital Filter Design Toolkit)

In a fixed-point implementation, after <u>quantizing the coefficients of a</u> <u>reference floating-point filter</u>, you also must configure all other <u>quantizers</u>, such as the input, output, and multiplicand quantizers. This process <u>creates a fixed-point model</u> for the filter. You must create a fixed-point filter model before you simulate the filtering process or generate code from the filter. You can use either the <u>DFD FXP Modeling</u> VI or the <u>DFD</u> <u>FXP Set Quantizer</u> VI to create fixed-point filter models. Refer to the *Details* section of the <u>DFD FXP Set Quantizer</u> VI for information about guidelines on the quantizer settings.



Note After quantizing the coefficients of a reference floating-point filter, you automatically obtain a fixed-point filter model with the following default values: input word length = output word length = 16. The <u>Specifying the Word Length and Integer Word Length</u> topic contains more information about the effects of different word length values.

If you use the DFD FXP Modeling VI to create the fixed-point filter model, complete the following steps:

- 1. <u>Specify the word lengths</u> for the input and output signals.
- 2. <u>Set the rounding mode</u> for the output signal.

The DFD FXP Modeling VI automatically calculates all quantizer settings of the fixed-point filter model. You can use the <u>DFD FXP Get Quantizer</u> VI to retrieve the quantizer settings or use the <u>DFD FXP Set Quantizer</u> VI to modify the quantizer settings.

N

Note Output quantizers generally have different integer word lengths from the input quantizers. Before generating code, check the quantizer settings to confirm or modify the settings you want to use for the fixed-point filter. If you reduce the default output integer word length, National Instruments recommends that you change the overflow mode to **Saturate**.

If you use the DFD FXP Set Quantizer VI to create the fixed-point filter model, you must <u>configure the quantizers</u>.

Refer to the Lowpass_Step 4_Model and Simulate FXP Lowpass VI in

the labview\examples\Digital Filter Design\Case Studies\Single-Rate Filter directory for an example that demonstrates how to create a fixed-point model of a lowpass filter and <u>simulate the filtering result</u>.

■ Open example

Simulating Fixed-Point Filters (Digital Filter Design Toolkit)

After you <u>create a fixed-point filter model</u>, you must <u>simulate the filtering</u> <u>process</u> to verify that the fixed-point model works as expected in a complete implementation. You can use the <u>DFD FXP Simulation</u> VI and the <u>DFD FXP Simulation with State</u> VI to facilitate this evaluation. To verify the simulation result, you can use the following two options:

- Compare the simulation results with the filter output results that you obtain by processing the same signal with the reference floating-point filter. Ensure that the simulation results are sufficiently similar to the filtering results of the reference floating-point filter.
- Use the DFD FXP Simulation Report VI to monitor the behavior of the fixed-point filter during the simulation process by observing the filtering text report output. The report contains statistical information about all the <u>quantizers</u>—except the coefficients *a/k* and coefficients *b/v* quantizers—in the fixed-point model. Each quantizer has five data entries: max value, min value, #overflows, #underflows, and #operations. Ensure that both #overflows and #underflows equal 0 or fall below an appropriate threshold.

Like all other parts of the design process, simulation is a trial-and-error process. If you observe overflow or underflow in the filtering text report or if the simulation result does not match the actual floating-point filtering result, try making the following adjustments:

- Return to the <u>modeling</u> step. Modify the integer word lengths for the related quantizers to eliminate overflows and underflows until both **#overflows** and **#underflows** equal 0 or fall below an appropriate threshold. Use **max value** and **min value** to estimate the integer word lengths.
 - Tip The DFD FXP Modeling VI automatically calculates the integer word lengths so you cannot modify them directly. However, you can use the DFD FXP Get Quantizer VI to retrieve the integer word lengths for the related quantizers and then use the DFD FXP Set Quantizer VI to modify the integer word lengths.
- Change the implementation structure.
- Adjust the specifications and redesign the floating-point filter.

Note In the filtering text report, the **#operations** entry for the product and sum quantizers provides information about the computational requirements of the fixed-point filter. A smaller value implies a faster computational speed. If several filter structures satisfy the performance requirements of the filter, select the filter structure whose product quantizer has the smallest **#operations** value.

Refer to the Lowpass_Step 4_Model and Simulate FXP Lowpass VI in the labview\examples\Digital Filter Design\Case Studies\Single-Rate Filter directory for an example that demonstrates how to <u>create a fixed-point</u> <u>model of a lowpass filter</u> and simulate the filtering result.

Open example

Exporting Fixed-Point Integer Coefficients (Digital Filter Design Toolkit)

If you have a filter execution engine for which you need only filter coefficients, you can export the fixed-point filter coefficients to a text file using the <u>DFD Save to Text File</u> VI. You can save the coefficients to a text file and download them to the execution target. The text file contains a section that provides all information about the fixed-point integer coefficients and corresponding quantizers.

Refer to the Export FIR Coef to Xilinx COE File VI in the labview\examples\Digital Filter Design\Fixed-Point Filters\Single-Rate directory for an example that demonstrates how to export fixed-point integer coefficients.

Open example

Generating Fixed-Point C Code (Digital Filter Design Toolkit)

You usually program target digital signal processing (DSP) hardware using C code. To <u>generate C code</u> from the fixed-point filter model, use the C Code instance of the <u>DFD FXP Code Generator</u> VI. You can compile the generated code to run on a fixed-point DSP.



Note C code can yield a less compact and less efficient implementation than a hand-written, assembly-coded implementation. If you need to improve the performance of a filter, you can translate the C code to assembly code manually.

The DFD FXP Code Generator VI produces three files, where *filtername* is the string you wire to the **filter name** input:

- nidfdtyp.h contains the definitions of the data types in the C source files that the DFD FXP Code Generator VI generates.
- *filtername*.h contains type definitions, global variable declarations, and function prototypes.
- *filtername*.c contains the code that implements the filter, including the filter coefficients, information about the implementation structure and quantizer settings in the fixed-point model, and the following functions:
 - *filtername_*State *filtername_*CreateState() creates the memory space needed to store the internal states of the filter.
 - void *filtername_*DisposeState(*filtername_*State state) disposes of the memory space used to store the internal states of the filter.
 - void *filtername_*InitState(*filtername_*State state) initializes the internal states to zeroes. Call this function for the first block when processing a large data sequence that consists of multiple data blocks.
 - I16 *filtername_*Filtering(I16 sampleIn, *filtername_*State state) implements the fixed-point filter.
 - static I16 *filtername_*Coef[] contains the quantized coefficients of the fixed-point filter.

Refer to the LabVIEW C Code Generation VI in the labview\examples\Digital Filter Design\Fixed-Point Filters\Single-Rate directory for an example that demonstrates how to generate LabVIEW C code from a fixed-point filter.

■ Open example

Generating Integer LabVIEW Code (Digital Filter Design Toolkit)

Integer LabVIEW code can run on any platform or target on which you can run LabVIEW VIs. Integer LabVIEW code is useful when you simulate the behavior of a fixed-point filter on the Windows platform. You can use the DFD FXP Code Generator VI to generate integer LabVIEW code from a fixed-point filter. The LabVIEW Digital Filter Design Toolkit uses LabVIEW projects to manage the resulting integer LabVIEW code. The following figure shows an example project file that contains integer LabVIEW code.



In the previous figure, the *filtername*.lvproj file, where *filtername* denotes the name of the fixed-point filter, contains the following folders and VIs in addition to the default items.

- **filtername Block**—This folder contains all generated VIs and subVIs related to the fixed-point filter from which you generate integer LabVIEW code. You can apply the filter to another project by adding this folder into the target project file.
- Filter SubVIs—This folder contains the generated subVIs. You usually do not need to modify these subVIs.
- filtername_Filter.vi—This VI is the top-level VI of the generated

integer LabVIEW code. To use the integer LabVIEW code, place this VI on the block diagram.

Refer to the Integer LabVIEW Code Generation VI in the labview\examples\Digital Filter Design\Fixed-Point Filters\Single-Rate directory for an example that demonstrates how to generate integer LabVIEW code from a fixed-point filter.

■ Open example

Generating LabVIEW FPGA Code (Digital Filter Design Toolkit)

LabVIEW field-programmable gate array (FPGA) code is a type of code specifically optimized to run on NI Reconfigurable I/O (RIO) devices such as the NI PXI-7831R. LabVIEW FPGA code takes advantage of the specific features, such as the single-cycle Timed Loop (SCTL) and memory items, of the LabVIEW FPGA Module. Therefore, this type of code can run on an FPGA target efficiently. You can use the DFD FXP Code Generator VI to generate LabVIEW FPGA code for filters with the following filter structures:

- FIR structures
- IIR Cascaded Second-Order Sections Form structures
- Lattice MA structures
- Lattice ARMA structures

Note To generate LabVIEW FPGA code, you must install the LabVIEW FPGA Module and NI-RIO driver software with R Series support. To execute the FPGA code, you also need an FPGA target on which to run the code.

The LabVIEW Digital Filter Design Toolkit <u>uses LabVIEW projects</u> to manage the resulting LabVIEW FPGA code. The following figure shows an example project file that contains LabVIEW FPGA code.



In the previous figure, the *filtername*.lvproj file, where *filtername* denotes the name of the fixed-point filter, contains the following folders and VIs in addition to the default items.

- **filtername Block**—This folder contains all generated VIs and subVIs related to the fixed-point filter from which you generate LabVIEW FPGA code. You can apply the filter to another project by copying and pasting this folder into the target project file.
 - Note Each filter block contains some FPGA memory components to store the internal states of the fixed-point filter. Multiple filter blocks cannot share the same memory components in one FPGA project. Therefore, if you want to use the filter block multiple times in one FPGA project, you must generate the filter blocks with different filter names from the same fixed-point filter.
- **States Storage**—This folder contains information about the specific resources on an FPGA target, including FIFOs and memory items. These resources store the internal states of the fixed-point filter. You usually do not need to modify the items in this folder.
- Filter SubVIs—This folder contains the generated subVIs. You

usually do not need to modify these subVIs.

• **filtername_Filter.vi**—This VI is the top-level VI of the generated LabVIEW FPGA code. To use the LabVIEW FPGA code, drag and drop this VI to the block diagram of the calling VI.

Refer to the LabVIEW FPGA Code Generation VI in the labview\examples\Digital Filter Design\Fixed-Point Filters\Single-Rate directory for an example that demonstrates how to generate LabVIEW FPGA code from a fixed-point filter.

Open example

You can generate both one-channel and multichannel LabVIEW FPGA code from a fixed-point filter. Refer to the Lowpass.lvproj file in the labview\examples\Digital Filter Design\Case Studies\Single-Rate Filter\Lowpass directory for an example that demonstrates how to generate one-channel LabVIEW FPGA code from a lowpass, finite impulse response (FIR) filter.

Open example

Refer to the Notch.lvproj file in the labview\examples\Digital Filter Design\Case Studies\Notch Filter\Notch directory for an example that demonstrates how to generate eight-channel LabVIEW FPGA code from an infinite impulse response (IIR) notch filter.

Open example

Resolving LabVIEW FPGA Code Compilation Failures

Sometimes the LabVIEW FPGA code might not compile successfully. One potential cause of a compilation failure is that the computation resources on the FPGA target might not meet the requirements of the fixed-point filter. For example, the NI PXI-7831R has 40 built-in 18x18 multipliers. Therefore, you cannot apply filters that require more than 40 built-in multipliers to the NI PXI-7831R. Another potential cause of a compilation failure is that the fixed-point filter is too complicated to implement in an SCTL or that the design clock rate is too high. A more complicated fixed-point filter requires more FPGA hardware resources. The compilation might fail when the FPGA hardware resources are not sufficient. For example, the compilation runs properly when the fixedpoint filter uses only 40% of the FPGA hardware resources but the compilation might fail if the fixed-point filter uses about 85% of the FPGA hardware resources. If you encounter compilation failures, try converting the filter structure to one that requires fewer resources or setting the design clock rate to a lower frequency.

The following table lists the number of multiplication units that each filter structure uses. One multiplication unit might require multiple FPGA builtin multipliers, depending on the type of multiplication unit. For example, an <u>I16xI16</u> multiplication unit requires only one FPGA built-in multiplier, but an I16xI32 multiplication unit requires two FPGA built-in multipliers. This table also lists the estimated execution time in ticks. One tick is one clock cycle, and the clock rate you specify when compiling the FPGA VI determines the length of the clock cycle. Execution time might vary because of the call overhead in loop structures.

Note Increasing the FPGA target clock rate reduces the amount of code that you can execute in the SCTL because the clock cycle is shorter.

Structure	Number of Multiplication Units ¹	Estimated Execution Time (ticks) ²
FIR Direct Form	1B	order+5
FIR Direct Form Transposed	1B	order+5
FIR Symmetric (odd order)	1B	Forder/21+5

FIR Symmetric (even order)	1B	[order/2]+6
FIR Antisymmetric (odd order)	1B	[order/2]+5
FIR Antisymmetric (even order)	1B	[order/2]+6
IIR Cascaded Second-Order Sections Form I	1A+1B	Forder/2]*4+2
IIR Cascaded Second-Order Sections Form I Transposed	1A+1B	[order/2]*3+4
IIR Cascaded Second-Order Sections Form II	1A+1B	Forder/2]*4+6
IIR Cascaded Second-Order Sections Form II Transposed	1A+1B	[order/2]*3+4
Lattice MA (minimum phase)	1A	order*2+3
Lattice MA (maximum phase)	1A	order*2+4
Lattice ARMA (basic sections)	2A+1B	order+9
Lattice ARMA (one multiplier sections)	1A+1B	order*2+6
Lattice ARMA (normalized sections)	2A+1B	order*2+6

1. Some filter structures use two groups of multiplication units because the structures contain two sets of filter coefficients. A and B in the table represent coefficients *alk* and coefficients *blv*, respectively. One multiplication unit might require different number of FPGA built-in multipliers, as shown in the following table:

Multiplicand x Coefficients	Truncated Internal Precision	Full Internal Precision	
I16xI16	One multiplier	One multiplier	
I16xI32	Two multipliers	Two multipliers	
I32xI16	Two multipliers	Two multipliers	
132x132	Three multipliers	Four multipliers	

2. The number of ticks in the Estimated Execution Time column is per-channel based. This table assumes that the filter gain processing is not on the FPGA target. Therefore, if you want to

process the filter gain on the FPGA target, add one more tick to the number of ticks.

- 3. *order* is the filter order. For IIR filters, *order* is the larger of the numerator and denominator order values.
- 4. \square is the smallest integer greater than or equal to x.

Postprocessing Filtered Signals (Digital Filter Design Toolkit)

After you deploy fixed-point filter coefficients to the target hardware, the output data uses a fixed-point representation. To convert the output signal of the fixed-point filter into floating-point representation, you must process the fixed-point signal using the DFD FXP Postprocessing VI.

Refer to the Lowpass_Step 6_Postprocessing VI in the labview\examples\Digital Filter Design\Case Studies\Single-Rate Filter directory for an example that demonstrates how to postprocess a filtered signal.

Open example

Designing Special Filters (Digital Filter Design Toolkit)

This book contains information about using the LabVIEW Digital Filter Design Toolkit to design special <u>single-rate</u> filters with the <u>Advanced FIR</u> <u>Filter Design</u> VIs, <u>Advanced IIR Filter Design</u> VIs, and the <u>Special Filter</u> <u>Design</u> VIs.

To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

Linear Phase Filters (Digital Filter Design Toolkit)

Linear phase digital filters allow all the frequency components of an input signal to pass through the filter with the same delay, which means that the group delay through the filter is a constant value independent of the frequency. Linear phase filters are useful in filtering applications in which you want to minimize signal distortion and spreading over time.

Nonlinear phase response, or dispersion, can be harmless in audio and other applications in which mild phase distortion is often imperceptible to humans. However, phase distortion can be harmful in some applications. For example, in digital communications applications, signal spreading caused by phase distortion can cause interference between time concentrated information symbols.

A minimum phase filter is a type of nonlinear phase filter that optimally minimizes the group delay at all frequencies for a given magnitude response at the expense of phase distortion. Minimum phase filters can be useful in control applications in which minimizing delay is more important than minimizing signal spreading.

Mathematical Definition

The digital filter frequency response H(f) is expressed in terms of magnitude and phase in the following equation:

 $H(f) = |H(f)|e^{j\varphi(f)}$ (A)

where |H(f)| and $\varphi(f)$ are both real-valued functions of frequency *f* and represent the magnitude and phase of the frequency response.

Only finite impulse response (FIR) filters can have exactly linear phase. You can design linear phase FIR filters using either window-based design methods or the Remez design method. The Remez design method in the <u>DFD Remez Design</u> VI is more powerful and flexible than window-based design methods.

The equation above reduces to the following equation for a linear phase FIR digital filter.

 $H(f) = (j)^m A(f) e^{-j\pi N f}$ (B)

where m = 0, 1

A(f) is the amplitude response of the filter

N is the order of the filter, which is equal to the number of filter taps minus one

f is the normalized frequency with the range [0, 0.5]

If the filter coefficients h(n) (where n = 0, 1, ..., N) are symmetric, h(n) = h(N-n), and m must be 0

If the filter coefficients h(n) are antisymmetric, h(n) = -h(N-n), and *m* must be 1

Types of Linear Phase FIR Filters

The following table lists the four types of linear phase FIR filters and the characteristics of each type. The book <u>Digital Filter Design</u> contains more information about linear phase FIR filters.

Туре	Classification	Frequency Characteristics
I	Even-order, symmetric	A(f) is symmetric about $f = 0$ and $f = 0.5A(f)$ is periodic with period 1
II	Odd-order, symmetric	A(f) is symmetric about $f = 0$ and antisymmetric about $f = 0.5$ A(f) is constrained to 0 at $f = 0.5A(f)$ is periodic with period 2
III	Even-order, antisymmetric	A(f) is antisymmetric about $f = 0$ and $f = 0.5A(f)$ is constrained to 0 at both $f = 0$ and $f = 0.5A(f)$ is periodic with period 1
IV	Odd-order, antisymmetric	A(f) is antisymmetric about $f = 0$ and symmetric about $f = 0.5$ A(f) is constrained to 0 at $f = 0A(f)$ is periodic with period 2

Use the DFD Remez Design VI to design linear phase FIR filters. Set the **order** input and the **filter type** input according to the Classification column of the table above. The DFD Remez Design VI designs the appropriate type of linear phase FIR filter based on the two inputs.

Use the following guidelines to determine the type of linear phase FIR filter you design.

- Type III and IV cannot be lowpass-like filters.
- Type II and III cannot be highpass-like filters.
- Type III and IV work well for <u>differentiators</u> or <u>Hilbert transformers</u> because they can give a constant 90° phase shift.

Experiment with different types. More than one type might produce an acceptable result for some target filter responses, but only one type can meet the target specifications. Select the filter type that has the smoothest frequency response. For example, the following figure illustrates the types of frequency response symmetry for each type of

linear phase FIR filter, assuming a sampling frequency of $f_s = 1$. Notice that in this example, a Type I or Type II filter has glitches, or rapid changes, at the frequency point of 1. A Type III or Type IV filter yields the smoothest frequency response because they do not contain glitches.



The following figure shows the magnitude response requirement of an ITU-468-weighting filter. Notice that the magnitude response is zero at DC and small but nonzero at high frequencies. Therefore, Type IV (odd order, antisymmetric) is the best choice for approximating this response.



Arbitrary Shape Filters (Digital Filter Design Toolkit)

If you want to design a linear phase finite impulse response (FIR) filter with an arbitrary magnitude response, you can use the <u>DFD Remez</u> <u>Design</u> VI. If you want to design a filter in which linear phase is not required but minimizing filter order is important, you can use the <u>DFD</u> <u>Least Pth Norm Design</u> VI to design an infinite impulse response (IIR) filter.

You can describe an arbitrary frequency response with multiple points using piecewise linear interpolation. The number of points you must supply to describe the shape depends on the target arbitrary shape magnitude response in a certain frequency band. You do not have to space the points evenly. Use more points where the magnitude response is tightly curved and fewer points where the magnitude response is more linear.

FIR Filters with Arbitrary Magnitude Responses

To design a linear phase FIR filter with an arbitrary shape magnitude response, use the DFD Remez Design VI and set the **filter type** input to Symmetric or Antisymmetric according to the <u>linear phase FIR filter type</u> table. Then describe the shape of the filter by specifying multiple points in the **band specs** input.

The following example uses a lowpass filter with a passband frequency ranging from 0 to 0.25 and a stopband frequency ranging from 0.3 to 0.5. Three points at frequencies 0, 0.1, and 0.25 with expected amplitudes of 1, 2, and 1, respectively, describe the shape of the passband range. To design this filter, enter the specifications shown in the following figure into the DFD Remez Design VI.

filterty Symr	/pe netric band spec:	or A 2 s	der 8					
÷) o		freq	amplitu	ude weight		freq	amplite	ude weight
	÷)o	e) o	$\left(\begin{array}{c} A \\ \overline{V} \end{array}\right) 1$		(÷)	ê) 0.3) o	
		0.1	(r) 2			€)0.5	e (
		0.25	$\left(\begin{array}{c} \lambda \\ \overline{v} \end{array}\right) 1$) (÷)	
		() 0	0				0	
			ripple const	raint 💮 0	1		ripple const	raint 🖞 0

The following figure shows the magnitude response of the designed filter.



The previous example describes the passband shape with only three

points. You also can describe an arbitrary passband shape with as many points as necessary. The following example uses 5,000 evenly-shaped frequency points to describe the passband shape of a sinc compensation lowpass filter. You can use this sinc compensator to correct the amplitude droop caused by a zero-order hold in a digital-to-analog (D/A) converter, as shown in the following figure:



An amplitude droop measures the amount that the signal power decreases in a specified frequency range. In the previous figure, the **Zero-Order Hold** plot shows the magnitude response of a zero-order hold in the D/A converter. You can see an amplitude droop exists in the frequency range of interest [0, 0.2]. To correct the amplitude droop, you can create a filter that whose magnitude response is similar to the **Anti-Sinc Compensator** plot in the frequency range of interest [0, 0.2]. You can create the filter by describing the passband shape with the corresponding inverse sinc function values of the droop. The following figure shows the magnitude response of the designed filter.



IIR Filters with Arbitrary Magnitude Responses

If you want to design an IIR filter with an arbitrary shape magnitude response and the phase response is not important, use the DFD Least Pth Norm Design VI and set the **filter type** input to either Minimum Phase or Maximum Phase. The DFD Least Pth Norm Design VI ignores all other phase specification inputs, including **group delay** and **phase** in the **band specs** input. Define the shape of the magnitude response by entering multiple points in the **band specs** input.

For example, you can design a minimum phase IIR filter with the same arbitrary magnitude response as the previous example. Set the **filter type** input to Minimum Phase and enter the same **band specs** into the DFD Least Pth Norm Design VI. The following figure shows the magnitude response of the designed filter.



Group Delay Compensator (Digital Filter Design Toolkit)

Infinite impulse response (IIR) filters that you design using Butterworth, Chebyshev, or Elliptic methods usually have a nonconstant group delay, which means that they have nonlinear phase or phase distortion. The greatest deviation from a constant group delay typically occurs at the edge of the passband or somewhere in the transition band.

Given a filter with phase distortion, you can cascade the filter with an allpass filter to linearize the phase response in the specified frequency ranges while keeping the magnitude response unchanged.

Let $\tau_0(f_i)$ and $\tau_{s,p}(\bar{s},f_i)$ denote the group delay of the given filter and the designed allpass filter at the *i*th frequency point, respectively. The coefficients vector of the allpass filter \bar{s} is determined by the following equation:

 $\min \sum_{i} \left| \boldsymbol{\tau}_{\partial \mathcal{P}} \left(\hat{\boldsymbol{a}}, \boldsymbol{f}_{i} \right) + \boldsymbol{\tau}_{0} \left(\boldsymbol{f}_{i} \right) - \boldsymbol{\tau} \right|^{\mathcal{P}}$

where $\ensuremath{\tau}$ is the target group delay in all user-defined frequency ranges.

A 4th order elliptic bandpass filter with a passband frequency ranging from 0.3 to 0.4 has nonconstant group delay in the specified passband. The following figure shows how to compensate the filter group delay in the specified passband to be near constant with an 8th order compensator using the <u>DFD Group Delay Compensator</u> VI.



The block diagram in the above figure uses the DFD Plot Group Delay VI to check the group delay response of the filter. The following figure shows the group delay response of the original filter and the compensated filter.



In the previous figure, you can see that the group delay of the compensated filter is fairly constant in the passband frequency ranging from 0.3 to 0.4. The constant value of the group delay indicates that the compensated filter linearly approximates the phase response in the passband. However, compared to the original filter, the compensated filter also increases the delay and filtering computation.

Narrowband FIR Filters (Digital Filter Design Toolkit)

The order of a <u>finite impulse response (FIR)</u> filter is related inversely to the transition bandwidth. Conventional FIR filters with narrow transition bands and high orders might be too complex to implement. You might consider designing narrowband filters using <u>infinite impulse response</u> (IIR) filters. However, narrowband IIR filters typically have nonlinear phase, especially near the transition band, and are numerically sensitive. You might be able to meet the target filter specifications by using narrowband FIR filters instead of using IIR filters.

The <u>DFD Narrowband Filter Design</u> VI uses interpolated FIR (IFIR) techniques and frequency response masking techniques to design narrowband FIR filters with significantly less computational complexity than conventional FIR solutions, as shown in the following figure:



- (a) Frequency response of the target narrowband FIR filter H(z)
- (b) Frequency response of a shaping filter G(z)

(c) Frequency response of an interpolated filter $G(z^N)$

(d) Frequency response of a masking filter I(z)

To better understand how these techniques work, assume that the target narrowband filter has the frequency response shown in part (a) of the previous figure. The first step is to design a shaping filter as shown in part (b). The shaping filter has a wider transition band than the target narrowband filter and determines the spectrum shape of the target narrowband filter. The next step is to design an interpolated filter with the frequency response shown in part (c). Notice that the coefficients of the interpolated filter $G(z^N)$ are constructed by inserting N-1 zeroes between every two adjacent coefficients of G(z). The magnitude response of the first image of $G(z^N)$ in part (c) is the same as that of the target filter H(z) in part (a). To remove the unwanted images of $G(z^N)$, you need to cascade the interpolated filter $G(z^N)$ with a masking filter I(z), which has the magnitude response shown in part (d).

By cascading the interpolated filter $G(z^N)$ and the masking filter I(z) as illustrated in the following figure, you can obtain the target narrowband filter H(z).



This figure shows a two-stage narrowband filter structure. Because I(z) and G(z) have a much wider transition band than the original filter H(z), the overall order of I(z) and G(z) is lower than the order of H(z), which makes the cascaded filters computationally efficient.

Similarly, if the lowpass masking filter I(z) also is a narrowband filter, you can make it more efficient by using the two-stage narrowband filter structure. The following figure illustrates the diagram of the resulting three-stage structure. This figure uses a cascaded integrator comb (CIC) filter as the first-stage lowpass masking filter in this case because of the lowpass nature and efficient implementation.



Assume a signal with a sampling frequency of 5 kHz. The signal has useful information at frequencies below 100 Hz and noise above 120 Hz. To suppress the noise, you can apply a narrowband lowpass filter with the following specifications:

Specification	Value
Passband Range	0–100 Hz
Passband Ripple	0.05 dB
Stopband Range	120-2500 Hz

Stopband Attenuation 60 dB

If you design the lowpass FIR filter using the DFD Remez Design VI, the resulting filter has 689 taps. Given the same specifications, the DFD Narrowband Filter Design VI generates a three-stage narrowband filter. The resulting CIC filter has 5 stages, the interpolated filter $F(z^M)$ has 18 nonzero coefficients and the interpolated filter $G(z^N)$ has 27 nonzero coefficients. The CIC narrowband filter is 78% less computationally complex than the lowpass FIR filter you design using the DFD Remez Design VI. The following figure shows the magnitude response of the designed narrowband filter.



To plot the frequency response of narrowband FIR filters, use the <u>DFD</u> <u>Plot Narrowband Freq Response</u> VI. To perform narrowband filtering, use the <u>DFD Narrowband Filtering</u> VI.

Designing Multirate Filters (Digital Filter Design Toolkit)

This book contains information about using the LabVIEW Digital Filter Design Toolkit to design <u>multirate filters</u>.

To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

Multirate Filter Basics (Digital Filter Design Toolkit)

Use multirate filters in digital signal processing systems when different sampling frequencies exist in different parts of a system, or when you want to reduce computational complexity in systems by using a uniform sampling frequency. You can change the sampling frequency of a filter by using decimation, interpolation, or rational resampling. This book describes the three filtering modes and the zero-phase filtering mode, which enables you to eliminate the delay between the input and output signals.

Use finite impulse response (FIR) structures to implement multirate filters. Compared to infinite impulse response (IIR) structures, FIR structures provide unconditional stability, phase linearity, and better finite-precision performance. Furthermore, FIR structures contain only feedforward signal paths that enable you to simplify the implementation of decimation and interpolation filters. In principle, you can use both lowpass and highpass FIR filters to implement multirate filters. However, the LabVIEW Digital Filter Design Toolkit provides lowpass multirate FIR filters only.

To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

Rational Resampling (Digital Filter Design Toolkit)

Rational resampling is the process of converting the sampling frequency of a signal to another sampling frequency that differs from the original frequency by a rational factor of L/M, where both L and M are integer values. Rational resampling also is known as fractional resampling.

Rational resampling is useful for interfacing with digital signal processing (DSP) systems that operate at different frequencies. By choosing *L* and *M* properly, you can approximate any desired sampling frequency change ratio. For example, you can use rational resampling with L = 147 and M = 160 to convert a 48 kHz signal from a professional audio system to a 44.1 kHz signal for an audio CD.

Note Decimation and interpolation are special cases of rational resampling. The rational factors of decimation and interpolation are 1/M and L/1, respectively.

You can implement a rational resampling system by cascading an L-fold expander with an M-fold decimator. You must place the expander before the decimator to avoid discarding useful frequency components in the decimation operation. The following figure shows a rational resampling filter with a rational factor of L/M.



This rational resampling filter first interpolates the input signal x(n) with an L-fold expander and changes the sampling frequency of the original signal f_s to a new sampling frequency Lf_s . The expander returns an output signal v(n) with this new sampling frequency. Both the interpolation filter following the expander and the decimation filter preceding the decimator are lowpass FIR filters, and the two filters operate at the same sampling frequency Lf_s . Therefore, you can integrate the two filters into one lowpass filter H(z) and place the filter between the expander and the decimator. The filter H(z) returns a new signal w(n). This rational resampling filter then decimates the signal w(n) with an M-fold decimator and changes the sampling frequency from Lf_s to $(L/M)f_s$. The decimator returns an output signal y(n) with this new sampling frequency.

When L < M, this rational resampling filter converts the original sampling frequency to a lower frequency, and H(z) acts as an anti-aliasing filter. When L > M, this rational resampling filter converts the original sampling frequency to a higher frequency, and H(z) acts as an anti-imaging filter. The cutoff frequency of H(z) is the smaller of the two values $f_s/2$ and $Lf_s/(2M)$.

Note The Multirate Processing VIs use a polyphase implementation, which is more efficient than the procedure in the previous figure, to implement rational resampling. Refer to the book Multirate Systems and Filter Banks for more information about polyphase implementations. Rational resampling filters that use a polyphase implementation compute only the final expected output samples, not the inserted zero value samples, thus reducing the computational complexity of the filters.

The following figure illustrates the rational resampling of a signal by a factor of 2/3. The different parts in the following figure correspond to the different spectra of the signal at different stages of the rational resampling process, as shown in the previous figure. Because L < M, the lowpass filter has a cutoff frequency of $Lf_s/(2M)$ and acts as an anti-aliasing filter. Part (e) of this figure shows the spectrum of the output signal if you use a lowpass filter. You can see that no aliasing occurs. From part (f) of this figure, you can see that if no lowpass filter exists, or if you choose the cutoff frequency of the lowpass filter to be $f_s/2$, aliasing occurs between each spectrum image of the decimated signal. The overlapping spectra indicate aliasing due to the decimation operation.



(a) Spectrum of the original signal x(n)

(b) Spectrum of the signal v(n) from directly interpolating the original signal by 2

(c) Magnitude response of the anti-aliasing filter H(z)

(d) Spectrum of the signal w(n) from the anti-aliasing filter H(z)

(e) Spectrum of the output signal y(n) with the anti-aliasing filter

(f) Spectrum of the output signal y(n) without the anti-aliasing filter

To design rational resampling filters, use the Rational instance of the DFD MRate Filter Design VI.

Decimation (Digital Filter Design Toolkit)

Decimation is the process of reducing the sampling frequency of a signal to a lower sampling frequency that differs from the original frequency by an integer value. Decimation also is known as down-sampling. The lowpass filtering associated with decimation removes high-frequency content from the signal to accommodate the new sampling frequency.

Decimation is useful in applications in which the Nyquist frequency of a signal is much higher than the highest frequency of the signal. Decimation filters help you remove the excess bandwidth and reduce the sampling frequency of the signal. Decimation filters also help you reduce the computational resources required for processing and storing the signal. During the analog-to-digital (A/D) conversion process, decimation filters also can reduce the variance of quantization noise in a signal and maintain the signal power, thus improving the signal-to-noise ratio (SNR).

The following figure shows a typical *M*-fold decimation filter, where *M* is the integer value by which you want to decrease the sampling frequency. This filter contains a lowpass FIR filter H(z). This lowpass FIR filter is an anti-aliasing filter followed by an *M*-fold decimator. The decimator passes every M^{th} sample and discards the other samples. After this operation, the decimation filter changes the sampling frequency f_s of the input signal x(n) to a new sampling frequency f_s/M . The decimation filter then returns an output signal y(n) with the new sampling frequency.



To prevent aliasing, this system uses the lowpass filter H(z) before the *M*-fold decimator to suppress the frequency contents above the frequency $f_s/(2M)$, which is the Nyquist frequency of the output signal. This system produces the same results as an analog anti-aliasing filter with a cutoff frequency of $f_s/(2M)$ followed by an analog-to-digital (A/D) converter with a sampling frequency of f_s/M . Because the system shown in the figure above is in the digital domain, H(z) is a digital anti-aliasing filter.

 $\overline{\mathbb{N}}$

Note The <u>Multirate Processing</u> VIs use a polyphase implementation, which is more efficient than the procedure in the

previous figure, to implement decimation. Refer to the book <u>Multirate Systems and Filter Banks</u> for more information about polyphase implementations. Decimation filters that use a polyphase implementation compute only the final expected output samples, not the samples to discard, thus reducing the computational complexity of the filters.

The following figure illustrates the potentially harmful effects of not using an anti-aliasing filter before the decimator. This figure shows the spectrum of the original signal x(n) and the spectra of the signals resulting from decimating the original signal by 2, 3, and *M*. Notice the overlapping spectra in parts (c) and (d) of the figure. The overlapping spectra indicate aliasing due to the decimation operation.



(a) Spectrum of the original signal x(n)

(b) Spectrum of the output signal y(n), decimated by a factor of 2

(c) Spectrum of the output signal y(n), decimated by a factor of 3

(d) Spectrum of the output signal y(n), decimated by a factor of M

To design decimation filters, use the Multirate Filter Design VIs with the
filtering mode input set to Decimation.

Interpolation (Digital Filter Design Toolkit)

Interpolation is the process of increasing the sampling frequency of a signal to a higher sampling frequency that differs from the original frequency by an integer value. Interpolation also is known as upsampling. The spectrum of the output signal ideally is the same as the input signal spectrum, except the output signal spectrum contains an additional high-frequency region with zero-power density.

The following figure shows a typical L-fold interpolation filter, where L is the target integer increase in the sampling frequency.



The interpolation filter contains an *L*-fold expander followed by a lowpass FIR filter H(z). The *L*-fold expander inserts L-1 zeroes between consecutive samples to the original signal x(n) and changes the sampling frequency f_s of the original signal x(n) to a new sampling frequency Lf_s . This process introduces images, as shown in the figure below, to the original signal. The interpolation filter then uses the lowpass FIR filter H(z) to remove the images. Therefore, this lowpass FIR filter is an anti-imaging filter. The interpolation filter then returns an output signal y(n) with the new sampling frequency.

Note The Multirate Processing VIs use a polyphase implementation, which is more efficient than the procedure in the previous figure, to implement interpolation. Refer to the book Multirate Systems and Filter Banks for more information about polyphase implementations. Interpolation filters that use a polyphase implementation compute only the nonzero input samples, not the inserted zero samples, thus reducing the computational complexity of the filters.

The following figure shows the spectrum of the original signal x(n) and the spectra from directly interpolating the signal by 2, 3, and *L* without using an anti-imaging filter. Notice multiple images emerge in the range from 0 to half of the resulting sampling frequency in parts (b), (c), and (d) of the figure. These images demonstrate the effect of interpolation.



The interpolation system uses the lowpass filter H(z) after the expander to attenuate the frequency components of the signal from $f_s/2$ to $Lf_s/2$. In the time domain, the effect of H(z) is to replace the inserted zero value samples that the expander introduces with the interpolated values. When replacing the inserted zeroes with interpolated values, the anti-imaging lowpass filter H(z) might alter the original values. Use a Nyquist interpolation filter for H(z) to maintain the original values.

To design interpolation filters, use the <u>Multirate Filter Design</u> VIs with the **filtering mode** input set to Interpolation.

Zero-Phase Filtering (Digital Filter Design Toolkit)

Zero-phase filtering helps you eliminate the group delay in the output signal of a filter. All multirate filters you design with the LabVIEW Digital Filter Design Toolkit, except for possible odd-order <u>cascaded integrator</u> <u>comb (CIC) filters</u>, are even-order, linear phase FIR filters. When you <u>use</u> <u>linear phase FIR filters to process signals</u>, the filters return signals with a constant group delay, as shown in the following figure:



In this figure, you can see that the **Output Signal** plot contains a constant set of zero values, which denotes the delay between the output and input signals. If you want to eliminate the delay, you can implement the filter as a zero-phase filter by setting the **zero phase?** input of the <u>Multirate Processing</u> VIs to TRUE. The following figure shows an example of an output signal that has no delay compared with the input signal.



Because zero-phase filters must be <u>noncausal</u>, you cannot achieve zerophase filtering in real-time signal processing. The <u>Multirate Processing</u> VIs achieve zero-phase by padding and trimming data. For single-block processing, the VIs pad the input data block at the beginning and end and trim the output data so the delay between the input and output is zero. For continuous processing, the VIs trim the initial transition so the delay between the input and the output is zero.



Note Zero-phase filtering works only with even-order multirate filters. All multirate filters you design using the Digital Filter Design Toolkit, except odd-order CIC filters, are even-order filters.

Designing Floating-Point Multirate Filters (Digital Filter Design Toolkit)

You can follow the <u>floating-point single-rate filter design process</u> to design a floating-point multirate filter. This topic explains how to use the <u>Multirate FIR Design</u> Express VI to design a lowpass multirate finite impulse response (FIR) filter. The following figure illustrates the typical magnitude response of a lowpass filter in multirate systems.



In this figure, f_s denotes the sampling frequency of the input signal. When designing filter specifications, you must take the filtering mode into consideration. Because <u>decimation</u> and <u>interpolation</u> are special cases of <u>rational resampling</u>, you can use the following filter specifications to design any filter with a rational factor of *L/M*.

Filter Specification	Value Range
Passband edge frequency	$0 < f_{pass} < \min(Lf_s/2M, f_s/2)$
Stopband edge frequency	$f_{pass} < f_{stop} < Lf_s/M - f_{pass}$

Typically, f_{pass} is the highest frequency of interest in the input signal. If $f_{stop} < \min(Lf_s/(2M), f_s/2)$, the transition band is free of aliases. If you change the constraints of f_{stop} to $\min(Lf_s/(2M), f_s/2) < f_{stop} < (Lf_s/2M - f_{pass})$, the filter has a wider transition band and a lower order, which can reduce the computational complexity significantly in filtering operations. However, the transition band then contains aliasing in the frequency conversions.

Entering Floating-Point Filter Specifications

After you place the Multirate FIR Design Express VI on the block diagram, the **Configure Multirate Filter Design** dialog box appears, as shown in the following figure:



On the **Floating-Point Design** tab of the configuration dialog box, you can select the filter type and design method from the pull-down menus and specify the filter factor. You then can enter the filter specifications through either the numeric controls on the right side of the tab or the magnitude response graphical interface on the configuration dialog box. The results are equivalent.

Using the Numerical Controls

Similar to the <u>numerical controls for single-rate filter design</u>, the numerical controls for multirate filter design also contain frequency ranges and ripple constraint settings. In addition to these settings, you also need to specify the sampling frequency of the input signal in the **Input sampling frequency** numeric control. This VI then automatically calculates the sampling frequency of the output signal based on the filter specifications you entered.

Using the Graphical Interface

The **Magnitude Response** graph displays the magnitude response of the designed multirate filter. The magnitude axis can be either a linear or a logarithmic scale. Remove the checkmark from the **Magnitude in dB** checkbox to use a linear scale, or keep the checkmark in the **Magnitude in dB** checkbox to use a logarithmic scale.

The **Magnitude Response** graph contains a set of cursors that you can use to specify the <u>passband and stopband</u>. Use the passband and stopband cursors to change the passband and stopband, respectively, of the multirate filter. Under the linear scale, the distance between unity and the horizontal passband cursor specifies the maximum passband ripple. The location of the vertical passband cursor indicates the passband edge frequency. The stopband cursors work the same when defining the specifications of the stopband. Under the logarithmic scale, the distance between 0 dB and the horizontal stopband cursor specifies the stopband attenuation.

Guidelines for Entering Filter Specifications

As you define a filter specification, you must adhere to a set of rules to maintain valid specifications. If you do not adhere to the following rules, the **Configure Multirate Filter Design** dialog box displays a message in the **Tips** indicator with suggestions for repositioning the cursors.

- Keep the horizontal cursors in the range (0, 1) in a linear scale or (-inf, 0 dB) in a logarithmic scale.
- Keep the horizontal passband cursor above the horizontal stopband cursor.
- The **Passband edge frequency** value must be less than the Nyquist frequency, or you must keep the vertical passband cursor to the left of the Nyquist cursor.
- The **Stopband edge frequency** value must be greater than the **Passband edge frequency** value, or you must keep the vertical passband cursor to the left of the stopband cursor.
- If you remove the checkmark from the **Transition band aliasing allowed** checkbox to avoid aliasing in the transition band, keep the **Stopband edge frequency** value between the **Passband edge frequency** value and the Nyquist frequency, or keep the

vertical stopband cursor between the vertical passband and Nyquist cursors. If you keep the checkmark in the **Transition band aliasing allowed** checkbox to allow aliasing in the transition band, keep the vertical stopband cursor between the vertical passband cursor and the vertical stopband limit cursor.

After you finish entering the filter specifications, click the **Update Design** button to apply the new specifications.

Refer to the MRate_Step1_Design Decimation VI in the labview\examples\Digital Filter Design\Case Studies\Multirate Filter directory for an example that demonstrates how to use the Multirate FIR Design Express VI to design a decimation filter.

■ Open example

Analyzing the Floating-Point Filter Design

When you design the multirate filter using the Multirate FIR Design Express VI, you can analyze the filter design by <u>examining the magnitude</u> response and filter order in real time.

Using Floating-Point Multirate Filters

After you analyze the filter design, you can either use the multirate filter to process an input signal or <u>quantize the filter coefficients of the</u> <u>multirate filter</u>. Use the <u>Multirate Processing</u> VIs to process a signal with the multirate filter you designed. The Multirate Processing VIs can process an input signal in the following three ways:

- As a single block of data
- As a sequence of data blocks
- As a sequence of data blocks with saved internal filter states

Use the DFD MRate Filtering for Single Block VI or the DFD NStage MRate Filtering for Single Block VI to process a single block of data. When processing a single block, the VIs extend the input signal block at both ends to ensure the output signal block has the same length as the input signal block. Use the other Multirate Processing VIs to process multiple signal blocks continuously. These VIs automatically retain the internal states of the filter between blocks, and they allow you to save and restore filter states without causing artifact glitches in the processed data.

You achieve the same results when you process multiple blocks as a sequence of blocks as when you process them together as one single block. To eliminate the delay between the input and output signals, enable the <u>zero-phase filtering</u> option when using the Multirate Processing VIs.

Refer to the MRate_Step2_Perform Decimation Filtering VI in the labview\examples\Digital Filter Design\Case Studies\Multirate Filter directory for an example that demonstrates how to use a decimation filter in a filtering application.

Open example

Quantizing Multirate Filter Coefficients

After you <u>analyze the filter design</u>, you can either quantize the filter coefficients of the designed multirate filter and convert the floating-point filter into a fixed-point filter or <u>use the floating-point multirate filter to</u> <u>process an input signal</u>. To quantize the filter coefficients, place a checkmark in the **Quantize filter** checkbox on the **Fixed-Point Quantization** tab, and specify the coefficients <u>word length</u>, coefficients <u>scale type</u>, and <u>gain settings</u>.

Designing Fixed-Point Multirate Filters (Digital Filter Design Toolkit)

The fixed-point multirate filter design process is different from the <u>fixed-point single-rate filter design process</u>. The following figure illustrates a typical fixed-point multirate filter design process. The grey boxes illustrate the <u>floating-point filter design process</u> and the arrows on the left indicate to which steps you can return if the filter design fails to meet the requirements in the current step.



Designing a fixed-point multirate filter involves fewer steps than designing a fixed-point single-rate filter. The following sections describe each step in the fixed-point multirate filter design process and compare the process to the fixed-point single-rate filter design process.

Quantizing Floating-Point Multirate Filters

Use the <u>DFD FXP MRate Quantization</u> VI to quantize floating-point multirate filters. This process is different from <u>quantizing floating-point</u> <u>single-rate filters</u>.

In the fixed-point single-rate filter design process, <u>scaling the filter</u> <u>coefficients</u> is a separate step. However, the fixed-point multirate filter design process integrates this step in the quantization process. Specify an appropriate value in the **scale type** input of the DFD FXP MRate Quantization VI to scale the multirate filter coefficients.

Depending on which VI you use to quantize the coefficients of a floatingpoint single-rate filter, you might need to configure some or all of the following items: the gain settings, word length and integer word length, overflow mode, and rounding mode. However, to quantize the coefficients of a floating-point multirate filter, you only need to set the gain processing target and coefficients word length. The DFD FXP MRate Quantization VI automatically calculates the integer word length of the filter coefficients and configures the settings for the overflow and rounding modes. The VI then uses the resulting values to quantize the filter coefficients.

Refer to the MRate_Step3_Analyze Quantized Decimation VI in the labview\examples\Digital Filter Design\Case Studies\Multirate Filter directory for an example that demonstrates how to quantize a floating-point decimation filter and <u>analyze the quantized decimation filter</u>.

Open example

Analyzing Fixed-Point Multirate Filters

Quantization can cause the characteristics of a fixed-point multirate filter to deviate from the reference floating-point multirate filter. Sometimes quantization even causes the resulting fixed-point multirate filter to fail to meet the target specifications. Therefore, after quantization, you must analyze the behavior of the fixed-point multirate filter. You can use the <u>Multirate Filter Analysis</u> VIs to calculate the frequency response of the fixed-point multirate filter. Based on the frequency response results, you can determine if the fixed-point filter meets the requirements. If the fixedpoint filter does not meet the target specifications, try either or both of the following methods:

- <u>Modify the quantization settings for the word length of the coefficients</u>.
- Modify the specifications of the reference floating-point multirate filter to allow larger headroom. A larger headroom helps alleviate fixed-point effects.

To optimize the resulting multirate fixed-point filter, analyze and adjust the filter design iteratively until the frequency response meets the target specifications.

Refer to the MRate_Step3_Analyze Quantized Decimation VI in the labview\examples\Digital Filter Design\Case Studies\Multirate Filter directory for an example that demonstrates how to <u>quantize a floating-point</u> <u>decimation filter</u> and analyze the quantized decimation filter.

Open example

Creating Fixed-Point Multirate Filter Models

When you design a fixed-point multirate filter, fixed-point quantization occurs for the coefficients and for the intermediate operands and results. By using the <u>DFD FXP MRate Modeling</u> VI, you need to configure only the quantizers for the input and output signals. The DFD FXP MRate Modeling VI automatically configures the quantizers for intermediate operands.

Setting the Input and Output Word Lengths

Use the DFD FXP MRate Modeling VI to specify input and output word lengths. These word lengths determine the number of bits to use in representing the input and output signals. For a fixed-point multirate filter, the integer word length of the output signal is the same as that of the input signal.

Setting the Rounding Mode

Fixed-point numbers have limited word lengths, so the available dynamic range of fixed-point numbers is lower than the range available with double-precision and floating-point numbers. Therefore, fixed-point numbers can approximate floating-point numbers only. In the LabVIEW Digital Filter Design Toolkit, the <u>Multirate Fixed-Point Tools</u> VIs use a 32-bit computation for internal operations. Use the **output rounding mode** input of the DFD FXP MRate Modeling VI to configure the output quantizer.

Setting the Internal Precision

For a multirate filter with a finite impulse response, if both the coefficients word length and input word length are greater than 16 bits, the internal multiplication is equivalent to an <u>I32xI32</u> operation. Theoretically, you need four I16xI16 multipliers to obtain a full-precision, or I32xI32, result. However, the fixed-point hardware target usually has limited multiplier resources. To conserve the multiplier resources, you can choose the truncation option. This option enables you to use three I16xI16 multipliers to obtain an approximated output. Use the **internal precision** input of the DFD FXP MRate Modeling VI to choose an appropriate option.



Note A <u>cascaded integrator comb (CIC) filter</u> does not require multipliers, but you also can use this input to <u>set the internal</u> <u>precision for the CIC filter</u>.

Refer to the MRate_Step4_Model and Simulate FXP Decimation VI in the labview\examples\Digital Filter Design\Case Studies\Multirate Filter directory for an example that demonstrates how to create a fixed-point model of a multirate filter and simulate the filtering result.

■ Open example

Simulating Fixed-Point Multirate Filters

To determine the <u>quantization</u> and <u>modeling</u> effects on a filtering process, you not only need to <u>analyze</u> the frequency response of a fixed-point filter, but you also need to simulate the filtering process with the actual data. You can use the <u>DFD FXP MRate Simulation</u> VI to simulate the filtering process. To verify if the fixed-point multirate filter works as you expect, compare the simulation result with the actual filtering result that you obtain by processing the same signal with the reference floatingpoint multirate filter. Ensure that the simulation result matches the actual filtering result. If the simulation result does not match the actual floatingpoint filtering result, try making the following adjustments:

- Return to the quantization step. Modify the word length for the filter coefficients and the scale type to avoid overflow or underflow.
- Adjust the specifications and redesign the floating-point multirate <u>filter</u>.

Refer to the MRate_Step4_Model and Simulate FXP Decimation VI in the labview\examples\Digital Filter Design\Case Studies\Multirate Filter directory for an example that demonstrates how to <u>create a fixed-point model of a</u> <u>multirate filter</u> and simulate the filtering result.

■ Open example

Generating Code from Fixed-Point Multirate Filters

After you <u>obtain an appropriate fixed-point filter model</u>, you can implement the resulting fixed-point filter on target hardware. You can export fixed-point integer coefficients from the filter and then use the coefficients in a filter execution engine. You also can generate LabVIEW FPGA code and then use the LabVIEW FPGA Module to target and deploy the resulting FPGA code to an NI Reconfigurable I/O (RIO) target.

Exporting Fixed-Point Integer Coefficients

If you have a filter execution engine for which you need only filter coefficients, you can export the fixed-point multirate filter coefficients to a text file using the <u>DFD Save MRate to Text File</u> VI. You can save the multirate filter coefficients to a text file and load them to the execution target. The text file contains a section that provides all information about the fixed-point integer coefficients and corresponding quantizers.

Refer to the Export Multirate FIR Coef to Xilinx COE File VI in the labview\examples\Digital Filter Design\Fixed-Point Filters\Multirate directory for an example that demonstrates how to export fixed-point integer coefficients.

■ Open example

Generating LabVIEW FPGA Code

You can use the <u>DFD FXP MRate Code Generator</u> VI to generate LabVIEW field-programmable gate array (FPGA) code from a multirate filter.



Note To generate LabVIEW FPGA code, you must install the LabVIEW FPGA Module and NI-RIO driver software with R Series support. To execute the FPGA code, you also need an FPGA target on which to run the code.

The Digital Filter Design Toolkit <u>uses LabVIEW projects</u> to manage the resulting LabVIEW FPGA code. The following figure shows an example project file that contains LabVIEW FPGA code.



In the previous figure, the *filtername*.lvproj file, where *filtername* denotes the name of the fixed-point filter, contains the following folders and VIs in addition to the default items.

- **filtername Block**—This folder contains all generated VIs and subVIs related to the fixed-point filter from which you generate LabVIEW FPGA code. You can apply the filter to another project by copying and pasting this folder into the target project file.
 - Note Each filter block contains some FPGA memory components to store the internal states of the fixed-point multirate filter. Multiple filter blocks cannot share the same memory components in one FPGA project. Therefore, if you want to use the filter block multiple times in one FPGA project, you must generate the filter blocks with different filter names from the same fixed-point multirate filter.
- **States Storage**—This folder contains information about the specific resources on an FPGA target, including FIFOs and memory items. These resources store the internal states of the fixed-point filter. You usually do not need to modify the items in

this folder.

- **Filter SubVIs**—This folder contains the generated subVIs. You usually do not need to modify these subVIs.
- **filtername_DataIn**—This item defines the input FIFO channel to the filter block of a fixed-point multirate filter. The fixed-point multirate filter uses the FIFO to communicate with other sections of the FPGA code, such as the FPGA I/O Node.
- **filtername_DataOut**—This item defines the output FIFO channel from the filter block of a fixed-point multirate filter. If you want to return the filtered signal directly to a host machine, you can modify the property of this file by completing the following steps:
 - 1. Right-click **filtername_DataOut**.
 - 2. Choose **Properties** from the shortcut menu.
 - 3. Choose **Target to Host-DMA** from the **Type** menu.
 - 4. Click OK.
- **filtername_FIR.vi**—This VI is the top-level VI of the generated LabVIEW FPGA code. To use the LabVIEW FPGA code, drag and drop this VI to the block diagram of the calling VI.

The way you use LabVIEW FPGA code is different from the way you use a general LabVIEW VI. The following figure shows an example of a block diagram that uses LabVIEW FPGA code generated from the *filtername_*FIR VI.



In the previous figure, the *filtername_*FIR VI is not connected to any other items on the block diagram. However, this VI actually communicates with the items in the loop structures by processing the input signal from the *filtername_*DataIn FIFO and returning the output signal to the *filtername_*DataOut FIFO.

You can generate LabVIEW FPGA code from a fixed-point single-stage multirate filter by using the DFD FXP MRate Code Generator VI. Refer to the MRateDecimation.lvproj file in the labview\examples\Digital Filter Design\Case Studies\Multirate Filter directory for an example that demonstrates how to generate LabVIEW FPGA code from a fixed-point single-stage decimation filter.

Open example

You also can generate LabVIEW FPGA code from a fixed-point multistage multirate filter by using the DFD FXP NStage MRate Code Generator VI. Refer to the NStageMRateDecimation.lvproj file in the labview\examples\Digital Filter Design\Case Studies\Multistage Multirate Filter\NStageMRate directory for an example that demonstrates how to generate LabVIEW FPGA code from a fixed-point multistage multirate filter.

Open example

Postprocessing Filtered Signals

After you deploy fixed-point filter coefficients to the target hardware, you can <u>postprocess the filtered signal</u> using the <u>DFD FXP MRate</u> <u>Postprocessing</u> VI. Refer to the MRate_Step6_Postprocessing VI in the labview\examples\Digital Filter Design\Case Studies\Multirate Filter directory for an example that demonstrates how to postprocessing a filtered signal.

Open example

Designing Special Multirate Filters (Digital Filter Design Toolkit)

This book discusses how to design <u>cascaded integrator comb</u>, <u>Nyquist</u>, and <u>multistage</u> multirate filters.

To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

Cascaded Integrator Comb (CIC) Filters (Digital Filter Design Toolkit)

A CIC filter is a special class of linear phase, finite impulse response (FIR) filter. CIC filters do not require multipliers and use a limited amount of storage. Therefore, CIC filters are more efficient than conventional FIR filters, especially in fixed-point applications. You usually use CIC filters in multirate systems with large sampling frequency conversion factors, such as digital down converters (DDC) and digital up converters (DUC) in communication systems.

CIC Filter Basics

CIC filters do not have multipliers and consist of only adders, subtracters, and registers. Therefore, you can implement multirate filters efficiently using the CIC filter structure. CIC filters are defined by the following transfer function:

$$H(z) = H_I^N(z)H_C^N(z) = \frac{\left(1 - z^{-RM}\right)^N}{\left(1 - z^{-1}\right)^N} = \left(\sum_{k=0}^{RM-1} z^{-k}\right)^N$$

where z is a complex variable

I is a basic integrator section

C is a basic comb section

M is the sampling frequency conversion factor

R is the differential delay

N is the number of stages

Theoretically, *R* and *N* can be any positive integer value, but the LabVIEW Digital Filter Design Toolkit constrains *R* to be either 1 or 2 because you do not need to use other values in most cases. *N* is in the range [1, 8]. The equation above shows that a CIC filter is equivalent to *N* stages of cascaded FIR filters with unit coefficients. Each FIR filter has a rectangular impulse response. All coefficients of the FIR filters are 1 and therefore symmetric, so the CIC filter has a linear phase response and constant group delay.

Use the <u>Multirate CIC Design</u> Express VI to design a CIC filter. Refer to the CIC Filter Design VI in the labview\examples\Digital Filter Design\Floating-Point Filters\Multirate directory for an example that demonstrates how to use the Multirate CIC Design Express VI to design a CIC filter.

Open example

Implementing Fixed-Point CIC Filters

The Digital Filter Design Toolkit supports fixed-point implementation of only lowpass CIC filters. To implement fixed-point CIC filters, cascade *N* basic integrator sections (the *I* block) and *N* basic comb sections (the *C* block) together with a sampling frequency conversion factor. The following figure shows an example of a fixed-point implementation of an *N*-stage decimation CIC filter, where *M* is the sampling frequency conversion factor.



The following figure shows an example of a fixed-point implementation of an *N*-stage interpolation CIC filter.



The following figure shows a basic integrator section in detail.



The following figure shows a basic comb section in detail.



Setting the Internal Precision

In a fixed-point implementation, the maximum bit width required for a CIC decimation filter is the sum of the input bits and the bits that the filter uses in accommodating the maximum filter gain. Using the maximum bit width for each integrator or comb section guarantees that no overflow occurs at

the output of the filter. The maximum bit width also ensures that you obtain a full-precision result. However, obtaining the full-precision result requires the maximum field-programmable gate array (FPGA) hardware resources.

In most real-world applications, the required output bit width is smaller than the maximum bit width. Therefore, you can discard the least significant bits (LSBs) from the maximum bit width to obtain a smaller output bit width. Using the <u>DFD FXP MRate Modeling</u> VI, you can prune the LSBs in each successive integrator or comb section. This operation is known as bit pruning. Bit pruning enables you to obtain a precision that approximates the full precision and to spare the FPGA hardware resources. However, bit pruning introduces additional noise to each processing section, and the amount of LSBs that you discard determines the noise level.

When using the DFD FXP MRate Modeling VI to model a fixed-point CIC filter, you can set the **internal precision** input to Truncated to prune the intermediate bit widths. This option is valid for only multirate FIR filters and fixed-point CIC decimation filters. If you set **internal precision** to Full, this VI applies the maximum bit width to each processing section.

The following figure shows an example of filtering results by using both the **Truncated** and **Full** options.



In the **Output Signal** graph of the previous figure, you can see that the **Truncated** plot renders nearly the same filtering result as the **Full** plot does. The **Comparing the Full and Truncated Options** graph shows the detailed difference between the two filtering results. The few nonzero values indicate the slight precision difference between the two internal precision options.

Multistage Multirate Filters (Digital Filter Design Toolkit)

The filters you design in the <u>Designing Floating-Point Multirate Filters</u> and the <u>Designing Fixed-Point Multirate Filters</u> books are single-stage multirate filters. In single-stage multirate filters, the normalized transition bandwidth of the lowpass FIR filter H(z) is inversely related to the filter order. The narrower the normalized transition bandwidth, the higher the filter order. A lowpass FIR filter with a narrow normalized transition bandwidth therefore requires more resources to implement.

In decimation and interpolation multirate filters, the normalized transition bandwidth inversely relates to the decimation factor M and the interpolation factor L. The order of a decimation or interpolation filter increases as M or L increases, and the resulting multirate filter uses more resources to implement. You can use multistage multirate filters to simplify multirate filters that have large sampling frequency conversion factors.

A multistage filter gradually increases or decreases the sampling frequency by passing the signal through two or more resampling stages. Each stage has a lower decimation or interpolation factor than the corresponding single-stage multirate filter and contains fewer operations. Except when the sampling frequency conversion factor is a prime number, multistage filtering is more efficient than single-stage filtering because you can change the sampling frequency in multiple stages rather than in a single stage. Using multiple stages reduces the computation operations and memory usage. Refer to the book <u>Multirate</u> <u>Systems and Filter Banks</u> for more information about multistage multirate filter design.

In a multistage decimation system, the overall decimation factor M is equal to $M_1M_2...M_N$, where M_i is the decimation factor of stage i. The following figure illustrates this N-stage decimation process.



In a multistage interpolation system, the overall interpolation factor L is

equal to $L_1L_2...L_N$, where L_i is the interpolation factor of stage *i*. The following figure illustrates the *N*-stage interpolation process.



You can use the <u>DFD NStage MRate Filter Design</u> VI to design multistage multirate filters with either of the following approaches:

- Specify the overall sampling frequency change factor and the factors for every stage.
- Specify only the overall sampling frequency change factor and use the DFD NStage MRate Filter Design VI to determine the factors for every stage.

Use the following guidelines when you manually specify factorizations.

- Use two or three stages for optimal or near optimal results.
- Use the largest factor at the highest sampling frequency. Decimate in order from the largest to the smallest factor and interpolate in order from the smallest to the largest factor.

When you implement a fixed-point multistage multirate filter, the output signal word length of the previous filter stage must be the same as the input signal word length of the next filter stage. Refer to the Multistage Multirate Filter Design VI in the labview/examples/Digital Filter Design/Floating-Point Filters/Multirate directory for an example that demonstrates how to use the DFD NStage MRate Filter Design VI to design a multistage multirate filter.

Open example

Advanced Techniques for Designing Filters (Digital Filter Design Toolkit)

This book contains an overview of advanced techniques for developing digital filters with the <u>DFD Remez Design</u> VI and the <u>DFD Least Pth</u> <u>Norm Design</u> VI. The book <u>Digital Filter Design</u> contains more information about these advanced techniques.

To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

Minimum and Maximum Phase FIR Design (Digital Filter Design Toolkit)

The <u>Designing Special Filters</u> book discusses linear phase finite impulse response (FIR) filter designs, which are actually amplitude approximations that use the following complex approximation criterion:

$$\min\left(\max\left(\sum_{i=0}^{L-1} \left(W(i) \cdot \left|H\left(\omega_{i}\right) - \mathcal{O}\left(\omega_{i}\right)\right|\right)\right)\right)$$

where $D(\omega_i)$ is the ideal frequency response, $H(\omega_i)$ is the frequency response of the designed filter, and W(i) is the positive weight at the *i*th frequency point.

Many applications require linear phase to ensure that the frequency components of an input signal pass through the filter with the same delay. If you have an application that does not require linear phase, you can control the magnitude response of the filter and allow the filtering process to change the delay relationship between different frequency components arbitrarily. You can use the following expression for this type of magnitude approximation problem.

$$\min\left(\max\left(\sum_{j=0}^{L-1} \left(W(j) \bullet \left\| H(\omega_j) \right\| - \left| \mathcal{O}(\omega_j) \right\| \right)\right)\right)$$

Without the phase constraint, you can achieve the same approximation error magnitude with a lower filter order or a smaller approximation error magnitude with the same filter order. Both options reduce the implementation cost.

More than one set of filter coefficients can have the same magnitude response. In the *z*-plane you can create a new set of coefficients with the same magnitude response, unless all zeroes are on the unit circle, through the allpass transformation of flipping zeroes to their conjugate reciprocal locations relative to the unit circle. You can specify minimum phase or maximum phase to eliminate the ambiguity about which set of FIR filter coefficients you are using.

All zeroes in a minimum phase FIR filter are inside or on the unit circle. Minimum phase filters sometimes are called minimum energy delay filters because the energy of the impulse response is maximally concentrated toward the beginning of the impulse response. All zeroes in a maximum phase FIR filter are outside or on the unit circle. The energy of the impulse response is maximally concentrated toward the end of the impulse response. Given a certain magnitude response, the impulse responses of the minimum and the maximum phase FIR filters are time-reversed.

The following figure shows the magnitude response of a 16th order FIR filter.



You can use two different sets of filter coefficients to match the magnitude response of the FIR filter. The following figure shows the impulse response and zeroes of a minimum phase filter with the same magnitude response of the 16th order FIR filter.



The following figure shows the impulse response and zeroes of a maximum phase filter with the same magnitude response of the 16^{th} order FIR filter.



Minimum phase filters are especially useful in control applications. A filtering process in a control loop typically requires the response to an input stimulus to be as quick as possible. A large delay in the filtering process can cause a negative feedback control loop to become unstable.

One disadvantage of linear phase FIR filters is that the system delay, also called the group delay, is fixed to half the filter order, which might be unacceptably long when the filter order is large. Use minimum phase filters in situations where minimizing delay is critical unless the system requires a linear phase filter or the linear phase filter order is relatively small.

You can design minimum or maximum phase FIR filters using the <u>DFD</u> <u>Remez Design</u> VI. Set the **filter type** input to Minimum Phase or Maximum Phase and the remaining specifications just as you <u>design a linear phase</u> <u>filter</u>. Refer to the <u>Exact Gain Control Design</u> and <u>Ripple Constraint</u> <u>Design</u> topics for information about designing minimum and maximum phase filters with exact gain control or ripple constraints.

You can create a minimum phase filter by designing a linear phase filter and converting it to a minimum phase filter by flipping the zeroes that are outside the unit circle to their conjugate reciprocal position inside the unit circle. Although the new minimum phase filter possesses the same magnitude response as the original linear phase filter, the new filter is not optimal relative to the filter magnitude specification. You can achieve a closer match to the magnitude specification or a lower filter order by specifying a minimum phase filter directly.

 $\overline{\mathbb{N}}$

Note Minimum phase filters can achieve better results in terms of a lower ripple or lower order for a given magnitude specification than equivalent linear phase filters. Therefore, minimum phase can be a better choice than linear phase in applications where the

phase response is not constrained.
Least Pth Norm Design Method (Digital Filter Design Toolkit)

You can use the <u>DFD Least Pth Norm Design</u> VI to design the following infinite impulse response (IIR) and finite impulse response (FIR) filters.

- Linear phase FIR design
- <u>Approximated linear phase IIR design</u>
- <u>Minimum and maximum phase IIR design</u>

Although you can design linear phase filters using the DFD Remez Design VI, you can design FIR and IIR filters with arbitrary magnitude and phase constraints using the DFD Least Pth Norm Design VI.

Least Pth Norm Linear Phase FIR Design (Digital Filter Design Toolkit)

You can design a linear phase FIR filter using the <u>DFD Least Pth Norm</u> <u>Design</u> VI by setting the following specifications. Set the denominator order to 0, the **filter type** to Symmetric or Antisymmetric, all phases in the band specifications to 0, and the group delay to half of the numerator order.

For example, suppose you want to design a linear phase FIR lowpass filter with a passband frequency range of [0, 0.2] and a stopband frequency range of [0.3, 0.5]. Set the specifications as shown in the following figure:



The following figure shows the magnitude response of the designed filter. Because \mathbf{p} is 128, the designed filter is almost identical to the result using the Remez equi-ripple design with the same filter specification.



Approximated Linear Phase IIR Design (Digital Filter Design Toolkit)

You can design IIR filters with approximately linear phase using the DFD Least Pth Norm Design VI. You must set filter type to Symmetric or Antisymmetric.

Although it is theoretically impossible to design causal IIR digital filters with exactly linear phase, you can design IIR filters with approximately linear phase. For example, suppose you want to design an approximately linear phase IIR lowpass filter with a passband frequency range of [0, 0.2] and a stopband frequency range of [0.3, 0.5]. You can set the specifications as shown in the following figure:



The following figure shows the magnitude response of the designed filter.



The following figure shows the phase response of the designed filter.



Notice that this filter has greater stopband attenuation than the <u>linear</u> <u>phase FIR filter designed</u>, and this filter keeps the passband phase response roughly linear.

Minimum and Maximum Phase IIR Design (Digital Filter Design Toolkit)

You can use the **Minimum Phase** or **Maximum Phase** option of the **filter type** input if you want a minimum or maximum phase response or if the phase response is not important. When you use the **Minimum Phase** or **Maximum Phase** option, the <u>DFD Least Pth Norm Design</u> VI ignores the **phase** and **group delay** inputs.

For example, suppose you want to design a minimum phase IIR lowpass filter with a passband frequency range of [0, 0.2] and a stopband frequency range of [0.3, 0.5]. You can set the specifications as shown in the following figure:



The following figure shows the magnitude response of the designed filter.



The following figure shows the phase response of the designed filter.



Notice that the designed filter has greater stopband attenuation than the <u>approximately linear phase IIR filter designed</u>, but the passband phase response is now highly nonlinear.